博士論文

Structure-Aware Latent-Variable Models for Neural Machine Translation

(ニューラル機械翻訳のための文構造を考慮した潜在変数モデル)

朱　中元

(Raphael Shu)

Supervisor: Hideki Nakayama

Graduate School of Information Science and Technology
The University of Tokyo

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Raphael Shu
December 2019

Abstract

Recent advance for training neural networks has allowed end-to-end neural-based machine translation models. Though achieving considerable high translation quality, a regular neural machine translation model still faces multiple folds of challenges. These challenges include the exponentially increasing model size, high translation latency and difficulties of obtaining translation results with diversity. In this thesis, we explore a novel approach that leverages latent-variable models to improve neural machine translation.

Our first contribution directly learns continuous latent variables to capture the information about target tokens, which enables non-autoregressive neural machine translation. By updating the posterior on the latent variables, we found that the evidence lower bound of the log-likelihood can be improved significantly. We also show that the proposed latent-variable approach coverages rapidly during updating. The resultant model translates at a speed around 8x $\sim$ 12x faster than the baseline autoregressive models. The proposed model outperforms baselines on the ASPEC Japanese-English translation task in terms of translation quality. On the WMT'14 English-German translation task, it narrows the performance gap between baselines down to 1.0 BLEU point.

As our second contribution, we apply the discretization bottleneck in an auto-encoder to learning discrete representation of words. The motivation is to compress neural models by replacing the giant word embedding matrix with discrete codes. Natural language processing (NLP) models often require a massive number of parameters for word embeddings, resulting in a large storage or memory footprint. In our approach, we assign each word a discrete code. To recover the word embeddings, we learn code vectors and composing them according to the codes. To maximize the compression rate, we adopt the multi-codebook quantization approach instead of a binary coding scheme. Each code has multiple discrete numbers, such as $(3, 2, 1, 8)$. We directly learn the discrete codes in an end-to-end neural network by applying the Gumbel-softmax trick. Experiments show that the compression rate achieves 98%

in a sentiment analysis task and $94\% \sim 99\%$ in machine translation tasks without performance loss.

The third contribution is to learn a sentence-level discrete representation. In this work, we add a planning phase in neural machine translation to control the sentence structure. Our approach learns discrete structural representations to encode syntactic information of target sentences. During translation, we can either let beam search choose the structural codes automatically or specify the codes manually. The word generation is then conditioned on the selected discrete codes. Experiments show that the translation performance remains intact by learning the codes to capture pure structural variations. Through structural planning, we are able to control the global sentence structure by manipulating the codes. By evaluating with a proposed structural diversity metric, we found that the sentences sampled using different codes have much higher diversity scores.

To summarize, we show in this thesis that a well-trained latent-variable model can capture structured linguistic features. This claim is mainly supported by the first and third contribution where we show latent-variable models can capture intra-word dependencies and syntactic structures in sentence level. For application, both studies utilize the learned latent variable for controlling the text generation. By sharpening the prediction of latent variables, we observe significant quality improvement of translations. By manipulating the discrete latent codes learned with syntactic structures, the model can produce sentences with great syntactic diversity. As a sideline topic, we also show that discrete latent-variable learning models can be good alternatives for quantizing long matrices such as the word embedding matrix.

Table of contents

List of figures xi

List of tables xv

Nomenclature xvii

1 Introduction and Background 1

1.1 Deep Learning for Natural Language Processing . . . 1
1.2 Learning Representation for Natural Language . . . 3
1.3 Machine Translation based on Neural Networks . . . 5
1.3.1 Disruptive Innovation against Statistical Machine Translation 6
1.3.2 Building Blocks of Neural Models . . . 8
1.4 Challenges in Neural Machine Translation . . . 11
1.5 Importance of Structure in Natural Language Generation . . . 13
1.6 Learning Latent Variables for Neural Machine Translation . . . 14
1.6.1 Enabling Non-autoregressive Translation by Encoding Word Dependencies . . . 14
1.6.2 From Continuous to Discrete Representation Learning . . . 18
1.6.3 Learning Compact Representation with Discrete Coding . . . 19
1.6.4 Syntactic Coding for Diverse Translation . . . 21
1.7 Contributions of This Thesis . . . 22

2 Prerequisite Knowledge for Neural Machine Translation 25

2.1 Recurrent Neural Networks . . . 26
2.2 Neural Machine Translation . . . 31
2.3 Attention Mechanism . . . 34
2.4 Transformer . . . 36
2.5 Training . . . 41

2.6 Decoding and Evaluation 41

3 Prerequisite Knowledge for Latent-Variable Likelihood Models 45
3.1 Approximate Inference 45
3.2 Variational Auto-Encoders 50
3.3 Reparameterization Trick 51
3.4 Learning Discrete Latent Variables 52
3.4.1 Straight-Through Estimator 53
3.4.2 Semantic Hashing 54
3.4.3 Gumbel-Softmax Reparameterization Trick 55
3.4.4 Vector-Quantization VAE 57

4 Latent-Variable Non-autoregressive Neural Machine Translation 59
4.1 Motivation 59
4.2 Non-Autoregressive NMT 60
4.3 Latent-Variable Non-Autoregressive NMT 61
4.3.1 A Modified Objective Function with Length Prediction 62
4.3.2 Model Architecture 63
4.3.3 Training 64
4.4 Inference with a Delta Posterior 65
4.5 Related Work 67
4.6 Experimental Settings 68
4.7 Result and Analysis 70
4.7.1 Quantitative Analysis 70
4.7.2 Non-autoregressive NMT Models 71
4.7.3 Analysis of Deterministic Inference 72
4.7.4 Decoding Speed Related to Sentence Length 74
4.7.5 Qualitative Analysis 75
4.8 Conclusion 76

5 Discrete Representation Learning for Model Compression 79
5.1 Motivation 79
5.2 Representing Words with Discrete Codes 80
5.3 Related Work 82
5.4 Advantage of Compositional Codes 84
5.5 Code Learning with Gumbel-Softmax 85
5.6 Experiments 87

5.6.1 Code Learning . . . 87
5.6.2 Sentiment Analysis . . . 88
5.6.3 Machine Translation . . . 90
5.7 Qualitative Analysis . . . 91
5.7.1 Examples of Learned Codes . . . 91
5.7.2 Analysis of Code Efficiency . . . 92
5.7.3 Shared Codes . . . 93
5.7.4 Semantics of Codes . . . 94
5.8 Conclusion . . . 94

6 Learning Syntactic Latent Variables for Diverse Translation 97
6.1 Motivation . . . 97
6.2 Proposed Approach . . . 99
6.2.1 Extracting Sentence Codes . . . 99
6.2.2 Diverse Generation with Code Assignment . . . 101
6.3 Related Work . . . 101
6.4 Experiments . . . 103
6.4.1 Experimental Settings . . . 103
6.4.2 Obtaining Sentence Codes . . . 104
6.4.3 Quantitive Evaluation of Diversity . . . 105
6.4.4 Experiment Results . . . 106
6.5 Analysis and Conclusion . . . 106

7 Summary 109
7.1 Contributions and Insights . . . 109
7.1.1 A Latent-variable Framework for Neural Machine Translation 111
7.2 On Interpretability of Latent Variables . . . 112
7.2.1 Latent Variables for Sentence Generation . . . 113
7.2.2 Qualitative Analysis of Discrete Word Codes . . . 115
7.2.3 Discrete Coding for Syntactic Structures . . . 115
7.3 Discussions . . . 117
7.3.1 Modeling Strategies of Latent-Variable Models . . . 117
7.3.2 Evaluating Performance of Discrete Bottleneck . . . 119
7.4 Future Outlook: Learning Emergent Language . . . 122
7.4.1 Compositionality and Interpretability of Emergent Language . 124
7.4.2 Thoughts on Future Works for Natural Language Generation . 126

References **129**

List of figures

1.1 Number of ACL accepted papers and neural papers from 2012 to 2019 2
1.2 Selected word pairs plotted in the vector space. The word vectors are taken from Glove 300D (Pennington et al., 2014a), visualized with t-SNE. 4
1.3 High-level comparison between Statistical Machine Translation and Neural Machine Translation 7
1.4 Five types of neural models for creating sentence representations . . . 10
1.5 A Japanese-to-English example that illustrates reordering and word-level ambiguity problems in machine translation task. 11
1.6 Increasing number of parameters of neural machine translation models. 12
1.7 Three different latent-variable models: (a) only one variable (b) the number of latent variables is determined by the number of target words (c) the number of latent variables is determined by the number of source words. 17
1.8 Derived tree structures with context-free grammar (top) and dependency grammar (bottom). 21

2.1 Illustration of the computational graph of recurrent neural networks . 28
2.2 Encoder-decoder framework of neural machine translation 32
2.3 Illustration of the model architecture of Transformer 37
2.4 Comparing batch normalization and layer normalization 40
2.5 Illustration of beam search algorithm in the setting of Japanese-to-English translation task. The beam size here is 2. The numbers show the step when running the algorithm. 42

4.1 Architecture of the proposed non-autogressive model. The model is composed of four components: prior $p(z|x)$, approximate posterior $q(z|x, y)$, length predictor $p(l_y|z)$ and decoder $p(y|x, z)$. These components are trained end-to-end to maximize the evidence lowerbound. 62
4.2 Illustration of the length transformation mechanism. 64
4.3 ELBO and BLEU scores measured with the target predictions obtained at each inference step for ASPEC Ja-En and WMT'14 En-De datasets. 71
4.4 Trade-off between BLEU scores and speedup on WMT'14 En-De task by varying the number of candidates computed in parallel from 10 to 100. 73
4.5 Relation between decoding speed and input sentence length of autoregressive baseline and proposed model, running on Nvidia V100 GPU. 74
4.6 Relation between decoding speed and input sentence length of proposed model, running on multiple devices. 74

5.1 Comparison of embedding computations between the conventional approach (a) and compositional coding approach (b) for constructing embedding vectors 81
5.2 The network architecture for learning compositional compact codes. The Gumbel-softmax computation is marked with dashed lines. . . . 86
5.3 Visualization of code balance for different coding scheme. Each cell in the heat map shows the count of words containing a specific subcode. The results show that any codeword is assigned to more than 1000 words without wasting. 93

6.1 Architecture of the TreeLSTM-based auto-encoder with a discretization bottleneck for learning the sentence codes. 100

7.1 BLEU scores on ASPEC Ja-En translation task produced by interpolated latent variables. The interpolation is created by computing weighted average of the prior mean and variational density mean. . . 111
7.2 Learned vector space of syntactic structures projected onto 2 dimensions by t-SNE. The labels for 80 most frequent trees are shown in the figure. 117

7.3 Cramér's V correlation matrix produced by analyzing a set of 8x8 discrete codes for all words in IMDB vocabulary. The codes are produced by encoding using Gumbel-Softmax discretization bottleneck. 121
7.4 Illustration of compositionality in emergent language during multi-agent communication. Here, /C B L S P/ renders the meaning of "A blue circle in the left of a purple square". /S P L C B/ means "A purple square in the left of a blue circle". 124

List of tables

3.1 Examples of how latent variables interpret the data distribution . . . 46
3.2 Input and output examples of a sampling operation of a multivariate Bernoulli distribution. 53
3.3 Input and output examples of a sampling operation of a categorical distribution. 55

4.1 Comparison of the proposed non-autoregressive (NAR) models with the autoregressive baselines. Our implementation of the Base Transformer is 1.0 BLEU point lower than the original paper (Vaswani et al., 2017) on WMT'14 dataset. 68
4.2 A comparison of non-autoregressive NMT models on WMT'14 En-De dataset in BLEU(%) and decoding speed-up. $\star$ measured on IWSLT'14 DE-EN dataset. 72
4.3 Ja-En sample translation with the proposed iterative inference algorithm. In the first example, the initial guess is refined without a change in length. In the last two examples, the iterative inference algorithm changes the target length along with its content. This is more pronounced in the last example, where a whole clause is inserted during refinement. 75

5.1 Comparison of different coding approaches. To support N basis vectors, a binary code will have $N/2$ bits and the embedding computation is a summation over $N/2$ vectors. For the compositional approach with M codebooks and K codewords in each codebook, each code has $M \log_2 K$ bits, and the computation is a summation over M vectors. . 85
5.2 Reconstruction loss and the size of embedding layer (MB) of difference settings 89

5.3 Trade-off between the model performance and the size of embedding layer on IMDB sentiment analysis task 89
5.4 Trade-off between the model performance and the size of embedding layer in machine translation tasks 92
5.5 Examples of learned compositional codes based on GloVe embedding vectors 92
5.6 Examples of words sharing same codes when using a 8×8 code decomposition 93
5.7 Some code examples using a 3×256 coding scheme. 94

6.1 A comparison of different code settings on IWSLT14 De-En and ASPEC Ja-En datset. For each code setting, we report the accuracy of recovering the syntactic tag sequence using the codes (tag accuracy), and the accuracy of predicting the codes using the source sentence (code accuracy). 104
6.2 Results for different approaches. The BLEU(%) are reported for the first sampled candidate. Oracle BLEU scores are produced with reference codes. Diversity scores (DP) are evaluated with Eq. (6.3). . 105
6.3 A comparison of candidates produced by beam search (A), semantic coding model based on BERT (B) and syntactic coding model (C) in Ja-En task 107

7.1 Most frequent parse trees in each clusters as a result of training the syntactic coding model with 256 clusters. We show the linearized form of top two levels of each tree. 116
7.2 A high-level comparison of continuous and discrete latent-variable models. 118

Nomenclature

Acronyms / Abbreviations

ACL Association for Computational Linguistics

CFG Context-free Grammar

CNN Convolutional Neural Networks

DG Dependency Grammar

ELBO Evidence LOwerBound

LM Language Model

LSTM Long Short-term Memory

MCMC Markov Chain Monte Carlo

NLP Natural Language Processing

NMT Neural Machine Translation

RNN Recurrent Neural Network

SMT Statistical Machine Translation

STE Straight-Through Estimator

VAE Variational Auto-Encoder

Chapter 1

Introduction and Background

1.1 Deep Learning for Natural Language Processing

Back in 2012, deep learning has proven its great potential in computer vision with the name of Convolutional Neural Networks (CNN). In ILSVRC 2012, a computer vision competition, the team "SuperVision" won with a superior prediction accuracy ; an absolute 10% higher than the second team. The winner team is led by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton from University of Toronto (Krizhevsky et al., 2012). Their model is a five-layer CNN, with max-pooling layers and dropout. The training takes about a week on two GPUs.

In the mean time, the research community in Natural Language Processing (NLP) are spending time on improving statistical models for various tasks[1]. From statistical language model (LM) to statistical machine translation (SMT), the models are generally count-based. As machine translation was regarded as one of the most difficult task of artificial intelligence, many researchers were dedicated to this field. In SMT community, people were using non-parameteric phrase-based translation models (Zens et al., 2002, Chiang, 2007) or its extensions. The state-of-the-art SMT models were those incorporating syntactic parse trees (Yamada and Knight, 2001, Liu et al., 2006). For English-Japanese language pair, some state-of-the-art models were supported by strong pre-processing methods, such as pre-reordering (Isozaki et al., 2010b). To further improve the translation quality, people were looking at more complicated grammar formalism such as combinatory categorial grammar (Auli,

[1]From a machine translation researcher point of view.

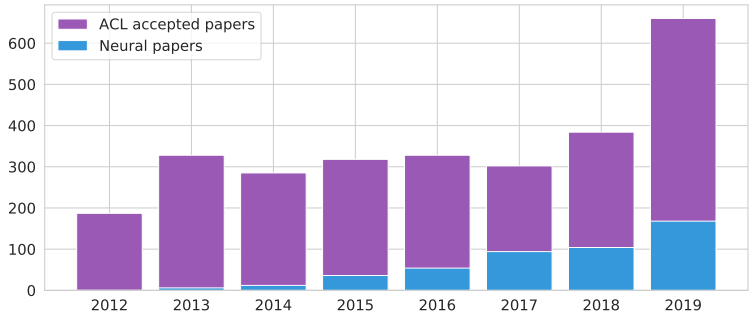


Fig. 1.1 Number of ACL accepted papers and neural papers from 2012 to 2019

2009). Few people attempted to adopt the cutting-edge deep learning models to NLP tasks.

Nevertheless, some researchers noticed that the trend of disruptive evolution in computer vision will inevitably affect the research field of computational linguistics. In ACL 2012, Richard Socher, Yoshua Bengio and Christoper Manning presented a talk titled "Deep Learning for NLP without magic", which turns out to be the first deep learning talk in the history of ACL. In the following years, NLP community started to recognize neural networks and conduct experiments on their tasks. I can recall in JNLP 2014 which is a domestic NLP conference in Japan, some open-mind researchers had a discussion on whether deep learning is going to revolutionize mainstream NLP tasks.

Surprisingly, the question received a definite answer not until the end of the year. In 2014, a neural-based sequence-to-sequence model (Sutskever et al., 2014) is published in NIPS. The model is powered by long short-term memory (LSTM), which is an extension of recurrent neural networks. Most importantly, the model is trained in an end-to-end fashion. The trained model can be directly utilized to translate a sentence from one language to another language. Although skepticism exists as the performance of the LSTM-based model was not on par with state-of-the-art SMT models, researchers were astonished by the ability of neural networks to encode the information of a full sentence into a single continuous vector. Furthermore, the model can unroll the vector to generate a sequence of translation. The exciting results motivated many NLP researchers to reproduce and explore the potentiality of neural networks, including the author of this thesis.

In the infancy of applying neural networks to NLP tasks, many models were borrowed from computer vision, such as attention mechanism (Gregor et al., 2015), residual connection (He et al., 2016) and batch normalization (Ioffe and Szegedy, 2015). In 2016, combining these new methods and other tricks, Google proposed a deep neural machine translation system (Wu et al., 2016) that achieves human-level translation performance. Fig. 1.1 shows the numbers of accepted papers and the papers applied neural networks in ACL from year 2012 to 2019. In six years, the ratio of deep learning papers grew from zero to 27% [2]. The success of deep learning in multiple core NLP tasks established it as an important machine learning model for language processing.

1.2 Learning Representation for Natural Language

The fusion of natural language processing and neural networks began with learning a vectorized representation of basic linguistic units. Along with the emerging of Word2Vec (Mikolov et al., 2013) and Glove (Pennington et al., 2014a), it became a norm in NLP community to represent words with countinuos vectors. Indeed, the adoption of feature vectors is not a new thing in the field, as old NLP pipelines are using all sorts of hand-crafted features. For instance, in part-of-speech tagging, we may have a feature to detect whether a word ends with a "-tion", which is an obvious sign of noun. However, the game changer is that, we do not need hand-crafted features anymore.

In Mikolov's 2010 paper (Mikolov et al., 2010), the first neural-based language model was introduced. In the paper, the feature vectors of words are obtained in an unsupervised way along with the training of language model. The language model is trained on large-scale corpora such as all written documents on Wikipedia. In such a model, each input word is considered as a contextual information to predict future words. Therefore, the word vectors in similar context are forced to carry similar information.

Then Word2Vec (Mikolov et al., 2013) was introduced 3 years later, which is also referred to as SkipGram model. Word2Vec is a predictive model that directly predicts the surrounding context of a specific word. Specifically, given a word

[2]The numbers are counted based on the data provided in ACL anthology. Only papers with "deep" or "neural" are examined. After 2016, many papers without a neural title actually adopt deep learning as a tool. Thus, the real number can be larger than reported.

w_t at position t, the model attempt to predict the following neighboring words: $w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}$. Thus, we can say that the SkipGram model predicts a *context window*, or a *co-occurance window*. Glove, another impactful word representation model works similarly as Word2Vec. The difference is that Glove constructs a giant co-occurence matrix and then factorize it into a low-dimension representation. Glove achieves high impact in recent NLP researches as the pre-trained vectors are publicly available on the Stanford NLP page[3]. In practice, those vectors corresponding to words with similar semantic meaning tend to be close to each other.

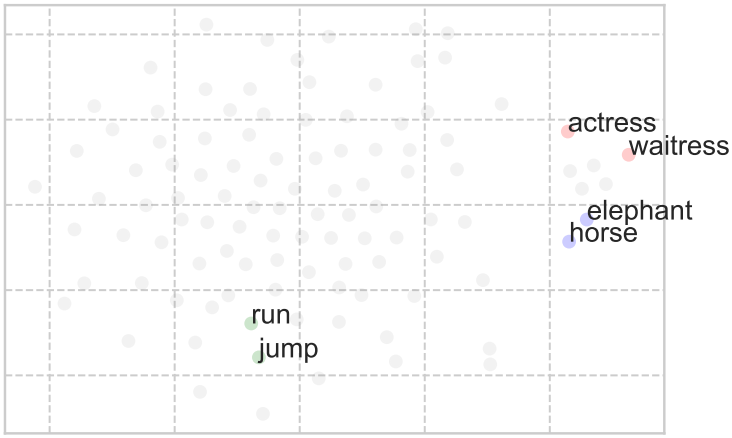


Fig. 1.2 Selected word pairs plotted in the vector space. The word vectors are taken from Glove 300D (Pennington et al., 2014a), visualized with t-SNE.

In Fig.1.2, we plot three pairs of words and their word vectors obtained from the public Glove vectors (Pennington et al., 2014b). The word vectors are trained on a dataset combines Wikipedia 2014 and Gigaword 5 texts. Each vector has 300 dimensions. The visualization is produced by t-SNE. The grey dots show the vectors of 100 random words. We can see that the chosen words with high semantic similarity are placed in close positions. Such a property of word vectors benefits classification tasks such as sentiment analysis, where the semantic meaning of a language is much more important that morphological traces.

After conquering the problem of representing a single word, the community started to think about compositionality, that is, how to represent more complicated linguistic units: phrases, sentences, and finally documents. The mainstream approaches now

[3] https://nlp.stanford.edu/projects/glove/

builds a sentence-level representation with unsupervised learning, which are also essentially language models. The current state-of-the-art approach, BERT (Devlin et al., 2018), is built as a *masked language model*.

One major division of opinions, however, still exists in the community and is not expected to be resolved in near future. Heavily influenced by recent advance in computer vision, most of the best NLP models are trained in end-to-end fashion and does not rely on any linguistic feature, such as a linguistic parser. Meanwhile, researchers that believe the benefits of adding linguistic bias are trying to mix the conventional linguistic analysis with neural-based models. Some report positive results (Socher et al., 2011b, Eriguchi et al., 2016, Nadejde et al., 2017, Aharoni and Goldberg, 2017) in various NLP tasks. For machine translation, the majority of the community holds an engineering point of view, at least until recently. The researchers are open to take in any advance that pushes the translation quality higher.

1.3 Machine Translation based on Neural Networks

Machine Translation is a task that convert a sequence written in one language into another language. We can think a machine translation model as a sequence transducer, produces a new sequence with unconstrained length by observing an existing sequence, following hidden rules. The machine translation model itself does not observe the sequence in its utterance form as humans do. It reads the sequence as a series of discrete symbols without even knowing which language is the sequence corresponding to. In machine translation community, We typically creates the symbols by assigning each word a unique ID in a vocabulary. Thus, the model works more like the ribosomes in the cytoplasm that synthesize proteins by processing DNA series or a crpytanalyst deciphering unknown numeric codes. The following snippet gives a picture on how a machine translation model observes paired translation data.

```
Source Language:  f4 e8 ad e8 ad f4 b9 f4 ad b1
Target Language:  ff f4 b6 f2 65 30 e9 b1
```

As the picture conveys, a model with no clues on how human language works have to do an incredible job to extract patterns of translation from massive bilingual

translation data. Then it has to further leverage the acquired knowledge to perform translation, or more often we say "decoding".

1.3.1 Disruptive Innovation against Statistical Machine Translation

The conventional data-driven approach for machine translation is known as Statistical Machine Translation (Brown and Piet, 2002, Koehn et al., 2003, 2007, Chiang, 2007), or SMT. It takes a straight-forward way to tackle the translation problem. In Statistical Machine Translation, we explicitly extract all possible patterns of translation. When the model is asked to translate, it tries to match portions in the given sentence with the extracted patterns. As we know the translation of these patterns in the target language, we combine all partial translations or fragments in the target language to form a fluent translation. More formally, the goal of translation is to obtain an English sentence candidate e that maximizes the probability of $p(e|f)$ given a foreign sentence f[4]. Statistical Machine Translation rewrites this target into $\text{argmax}_e\, p(f|e)p(e)$ by applying Bayes' rule. There are two models in the equation: a translation model $p(f|e)$ and a language model $p(e)$. The translation model proposes plausible candidate translations by pattern matching. The language model then finds the candidate that mostly looks like an English sentence.

In fig. 1.3, we give an illustration on the pipeline of Statistical Machine Translation in the lower half of the figure. Firstly, noisy word alignments are learned based on corpus statistics using IBM models (Brown and Piet, 2002). Then, all possible bi-lingual phrase pairs are extracted by recognizing aligned blocks in word alignments. These phrase pairs are stored in a list named *phrase table* (Koehn et al., 2003). At this stage, the phrase table can be tremendously noisy as it tries to capture all translation-alike patterns. Then, based on the statistics of the phrase pairs, each pair is assigned a set of weights. They are used to determine the probability $p(f|e)$ of a candidate translation.

The second major component of Statistical Machine Translation is the language model $p(e)$, it evaluates whether the language of a candidate translation is fluent and understandable. At its core, it computes a count-based statistical model for all n-grams in a sentence. For instance, the probability of "`I found a cat`" is

[4] f also means French in the context of SMT, as the most investigated language pair is French-to-English.

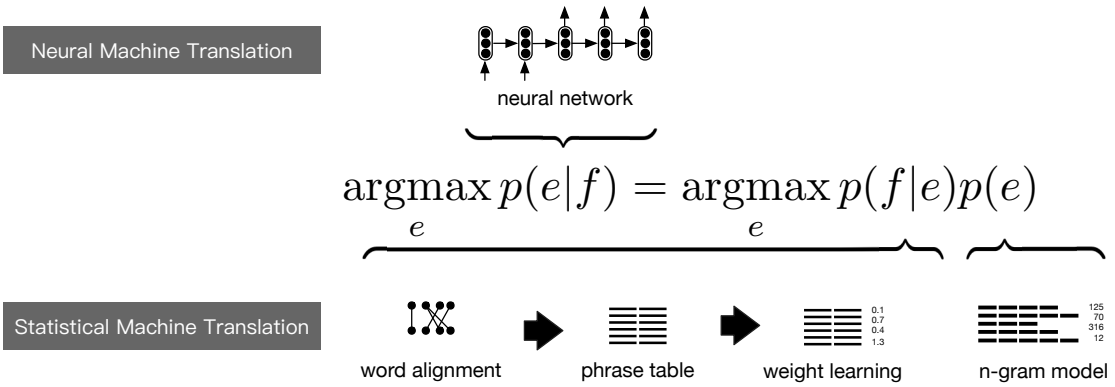


Fig. 1.3 High-level comparison between Statistical Machine Translation and Neural Machine Translation

decomposed into:

$$p(\texttt{I found a cat}) = p(\texttt{cat}|\texttt{I found a})p(\texttt{a}|\texttt{I found})p(\texttt{found}|\texttt{I})p(\texttt{I}). \tag{1.1}$$

A good language model assigns more probability to an English-alike sentence. Combining the two models together, a SMT system selects good candidates when the strengths of translation model and language model are well balanced. If the translation model is too strong, the result may be a precise translation, however, not readable. If the language model is too strong, the result can be a fluent sentence but fail to preserve the semantics of the input. In practice, the weights of translation model and language model are tuned on a held-out development dataset.

Statistical Machine Translation is a non-parametric approach, which means it directly utilize the training data as its "parameters". In the decoding time, an large phrase table has to be loaded into the memory. In some cases, it can consume over 100 Gigabytes of the on-board memory. Unfortunately, the language model is even more memory hungry. As a language model can be trained on massive monolingual corpus. When the n-gram table is loaded into memory, it can consume more than 1 Terabytes without compression. Multiple language model compression methods (Heafield et al., 2013) are thus proposed to lower the memory consumption with modest accuracy loss. Finally, to find a good translation in a reasonable time budget, carefully designed algorithms are employed in the decoder (Koehn et al., 2007).

Combing all techniques in such a framework, Statistical Machine Translation is a highly complicated pipeline. All models and decoding algorithms have to be carefully tuned to achieve a high translation quality and decoding speed. However,

as a common phenomenon in machine learning, the errors generated in every phases of the pipeline will be propagated and negatively affect the final translation quality.

In 2014, the first workable neural-based sequence-to-sequence model (Sutskever et al., 2014) is proposed. The model is built with solely neural networks. It is trained by directly maximizing the log-likelihood $\log p(e|f)$ with stochastic gradient descent. The only essential component other than the neural networks is the beam search algorithm, which is used to reduce the search space during decoding.

It took Neural Machine Translation (NMT) approximately three years to surpass Statistical Machine Translation in terms of performance. Notice that the latter approach was developed for about 20 years. Most importantly, the neural approach does not have a "pipeline". It trains in an end-to-end fashion, where we feed the source sentence as input and obtain the target sentence as the output. In contrast to a SMT decoder, which is usually a large project that contains over 5,000 lines of codes, an NMT model can be written in a single script with modern deep learning frameworks.

Other than the contrasting simplicity, neural machine translation has a controllable memory footprint. The model size only depends on the number of parameters which is pre-determined when designing the model. In practice, neural models typically take less than 1 gigabyte of GPU memory during testing time. Therefore, even at its infancy, the neural approach is not merely an academic prototype, but a strong competitor.

1.3.2 Building Blocks of Neural Models

To facilitate the understanding of discussion on neural machine translation, we give a brief overview of the model architecture in this section. Note that from this section, we change the notation of $p(e|f)$ to $p(Y|X)$ when describing the translation problem, which is more commonly used nowadays in deep learning community.

Despite recent advance in creating more powerful translation models, the high-level framework remains the same. Almost all NMT models follow the encoder-decoder framework (Sutskever et al., 2014). The framework can be simply described as

follows:

$$h_x = f_{enc}(\mathrm{E}(X)) \tag{1.2}$$

$$p(Y|X) = f_{dec}(h_x), \tag{1.3}$$

where, $E(\cdot)$ return a seqeuence of word vectors of the input sentence X. Then the encoder $f_{enc}(\cdot)$ creates an internal representation of the input sequence. The representation is further passed to the decoder $f_{dec}(\cdot)$. The decoder finally predicts the translation probability $p(e|f)$. Usually, both the encoder and the decoder are parameterized by same kind of models. There, there are only two major building blocks in encoder-decoder framework: the word embedding function $E(\cdot)$, and the neural functions operating on the hidden states used for $f_{enc}(\cdot)$ and $f_{dec}(\cdot)$. The latter one is implemented with *recurrent neural networks* in naive cases.

Word Embedding In natural language, words are considered as discrete symbols. However, as neural networks are commonly trained with gradient descent, the intermediate representations have to be continuous vectors in order to obtain valid gradients. Therefore, the very first step is to convert words to their continuous representations, particularly, word embeddings.

Word embeddings[5] are fixed-size continuous vectors, which usually contain 256 $\sim$ 1024 float numbers depending on the task. Each word in the vocabulary is assigned a vector. Stacking all vectors together, we usually maintain a giant embedding matrix in a neural network. The information contained in each embedding vector is expected to represent the word. In other words, the neural model shall be able to distinguish a word by looking at its vector. When two words share similar meaning, their embeddings are expected to placed closely in a vector space. However, in classification tasks, we often want the embeddings to capture task-dependent information rather than its meaning, such as sentiment and syntactic roles. Generally, the information captured by word embeddings depends on the training strategy we use. The details of training word embeddings will be described in chapter 2.

Recurrent Neural Networks As word embedding provides a way to represent words in continuous vectors, neural networks further provide ways to represent a sentence. As a sentence is a sequence of words, the problem now becomes how to

[5]In some papers, word embeddings are referred to as word vectors.

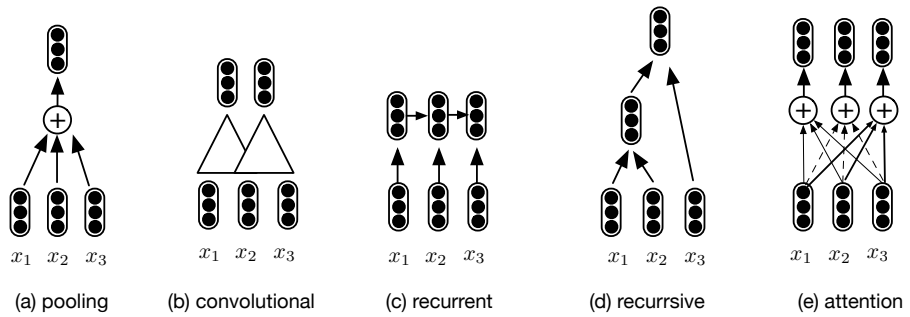


Fig. 1.4 Five types of neural models for creating sentence representations

combine word embedding. Actually, exploiting the compositionality of a language with neural networks is still currently an active research area. Multiple models are proposed to create sentence-level representations. These models can be generally classified into five categories: pooling models, convolutional models, recurrent models, recursive models and attentional models. Fig. 1.4 illustrates their architectures. For pooling models, the most commonly used ones are averaging models such as Deep Average Network (Iyyer et al., 2015). The convolutional models apply 1-D convolution on embedding vectors to create less or same numbers of vectors. Then, the recursive mode ls are proposed by Socher et al. (2013), which combine multiple word embeddings following a given tree structure. Usually, the structure is extracted from the parse result of a syntactic parser (e.g., CFG parser). The attentional models in turn do not create a single vector representation, but multiple vectors in a sequence.

The most impactful model is the recurrent neural network (RNN), which process a sequence of vectors with an elegant recurrent function. Let the input sequence be $x_1, \ldots, x_T$, then a RNN has the computation in the following form:

$$h_t = f_{\text{RNN}}(E(x_t), h_{t-1}; \theta) \,, \tag{1.4}$$

where $E(x_t)$ returns the word embedding vector in t time step. $f_{\text{RNN}}(\cdot)$ is a recurrent function updates the hidden state h_t in time step t when receiving the input vectors and the previous hidden state. The recurrent function is parameterized by θ. Eq. (1.4) gives a high-level abstraction of recurrent models, which does not reveal its detailed implementation. The notations in Eq. (1.4) will be used repetitively in this thesis when encountering RNN computations.

After processing the word embeddings with RNN, we can treat the last hidden state h_T as the sentence representation. Empirically, with a carefully designed recurrent function, the RNN can "memorize" a fairly long sequence into a single continuous vector. In classification tasks, the representation is handed over to a neural classifier to predict the targets of interest.

1.4 Challenges in Neural Machine Translation

As Jason Brownlee stated in his article[6], automatic machine translation is perhaps one of the most challenging artificial intelligence tasks on human language. This statement is still valid today. Even recent advance in training deeper and bigger models raises the translation quality to a new level, and multiple research institutes claimed that neural machine translation systems reached super-human performance, we are still facing problems when translating real documents. The model still makes catastrophic errors when translating sentences that human can easily understand.

The challenges are imposed from two sources. The first difficulty comes from the machine translation task, and the second comes from the neural model itself. In machine translation, the task becomes significantly difficult when handling two conundrums: flexibility and ambiguity of the language. Natural languages are flexible because the sentences can have arbitrary lengths and orders.

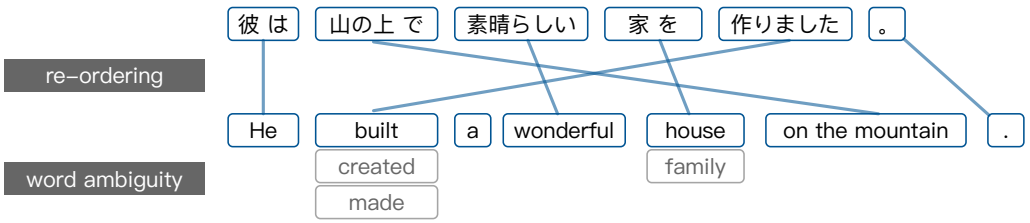


Fig. 1.5 A Japanese-to-English example that illustrates reordering and word-level ambiguity problems in machine translation task.

Challenges in MT Tasks In Fig. 1.5, we illustrate the aforementioned issues in translation tasks. The figure shows an example translation from Japanese to English. Generally speaking, translation is more challenging between languages with drastically different typologies. In the case of Japanese-English translation,

[6] https://machinelearningmastery.com/introduction-neural-machine-translation/

we can see long-range reordering is a common phenomenon as Japanese language usually has the main verb in the end of a sentence. In order to correctly handle long-range reordering, a machine translation model is required to precisely recognize the underlying structure of the sentence. Another fold of difficulty is ambiguity. In Fig. 1.5, we demonstrate the word-level ambiguity, where a single word can have multiple possible target translations. In this case, the main verb of the Japanese sentence is translated into "built" because it acts on the noun "house". Therefore, understanding the context is crucial to choose correct words from candidates. Beyond word-level ambiguity, we also have the ambiguity in grammar, syntax and semantics. For instance, grammar-level ambiguity pervasively exists in informal writing such as conversation, mails and tweets. As the sentences are deliberately shortened, a machine translation model has to uncover its broken grammatical structure in order to translate well.

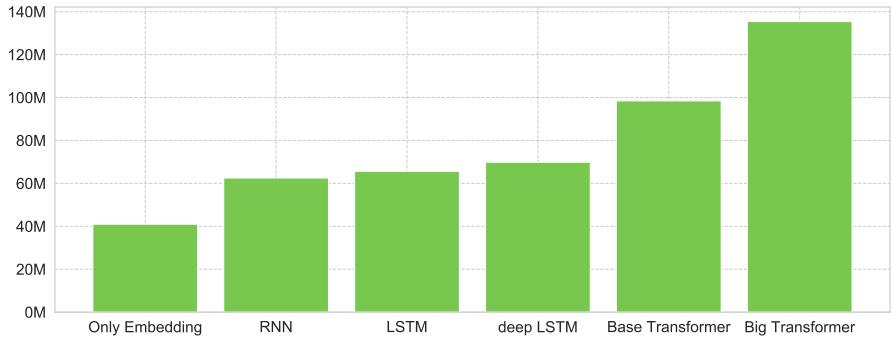


Fig. 1.6 Increasing number of parameters of neural machine translation models.

Challenges in MT Models The second source of challenge lies in the model design. As the neural models become deeper and more complicated, the number of parameters in a single NMT model explodes. We have the parameters in a model way more than the number of training data points. In Fig. 1.6, we show the number of parameters in standard NMT models. As the bar plot demonstrates, the word embedding layers alone already have 40M parameters. The LSTM-based models add around 20M parameters. With the recent state-of-the-art Transformer model (Vaswani et al., 2017), a single neural network reaches around 100M parameters. Then the Big Transformer doubles the number of parameters in LSTM models, reaching 130M parameters. Although the over-parameterization is proved to significantly

improve the resultant quality, it makes the model almost infeasible to be applied on low-end devices.

In addition to the oversizing issue, the translation latency has a linear relationship to the number of words it processes. This problem is caused by the autoregressive modeling, which factorize the conditional probability $p(Y|X)$ by:

$$p(Y|X) = \prod_{t=1}^{T} p(y_t|X, y_{<t}). \tag{1.5}$$

Here, $y_{<t}$ denotes the history $y_1, \ldots, y_{t-1}$. As the equation suggests, the target word y_t cannot be predicted unless its prefixing context is already generated. In practice, the decoding algorithm has to execute the computational graph of the decoder neural network in each time step. Therefore, the advantage of parallel computation of modern processing units such as GPU cannot be exploited in this case. In the following sections, we show that by learning latent variables of the language, one can alleviate these issues.

1.5 Importance of Structure in Natural Language Generation

Natural language is complex as the meaning of language is composed by the structure of sentences. There are two perspective related to the sentence structure: dependency and syntax. Conventional natural language generation models such as language models and machine translation models do not separate the generation process of the structure and the words of a sentence. They are determined simultaneously during generation. If we are able to develop a mechanism to capture these structures, we can plan the structure before the generation in utterance level. This motivates us to investigate the latent-variable approach for this purpose.

Dependency An output sentence can be treated as an interconnected graph that all words in the sentence depends on each other. A good language generation model has to correctly capture word dependency in a sentence in order to produce consistent results.

Syntax In the syntax level, we care about whether a sentence is a well-formed hierarchical structure. To express the same meaning, a language may allow multiple syntactic structures to be used. A model has to correctly choose one valid syntax for generation.

1.6 Learning Latent Variables for Neural Machine Translation

In this thesis, we introduce multiple latent-variable models with a purpose of tackling the challenges in machine translation. Through empirical analysis, we show that learned latent variables are capable of capturing semantics in words, intra-word dependencies and even syntactical features. In this work, we not only propose and learn latent-variable models, but further examine their application on large machine translation datasets. We first propose a generative model that that learns low-dimensional continuous latent variables to capture the dependencies among words. We demonstrate that such a model can enable non-autoregressive translation with significantly improved latency. Then we shift to discrete representation learning. We propose a model that captures the semantics in word vectors with discrete variables. In our experiments, we show that such a model can greatly compress the word embeddings by over 90%, while still preserving the performance in down-stream tasks. Finally, we propose a syntactic encoding model that encodes parse-tree level features into discrete variables. For application, we show that learned discrete latent variables can be used to generate drastically diverse translation candidates.

1.6.1 Enabling Non-autoregressive Translation by Encoding Word Dependencies

In computer vision, we see both adversarial neural networks (Goodfellow et al., 2014) and flow-based likelihood models (Kingma and Dhariwal, 2018) are now capable of generating realistic images. In general, these models do not generate images pixel by pixel. However, we can observe that the dependencies between pixels and local features are well captured. If the model paints a blue eye in the left, then it will do the same thing in the right side. Therefore, for neural language generation models, we rethink the necessity of the autoregressive modeling, which force the model to

generate only one word in a step. As a result, researchers began to think about the possibility of non-autoregressive machine translation models (Gu et al., 2018a).

As mentioned in the previous section, neural machine translation models are usually trained as autoregressive models, where the objective function is

$$\log p(y|x) = \sum_{t=1}^{T} \log p(y_t|x, y_{<t}). \tag{1.6}$$

Such a way of modeling forces us to sample from the neural network recursively, which impose a constraint that the decoding phase has to rely on searching algorithms. Beam search, which is a depth-first algorithm is commonly used in neural machine translation to find a translation with high log probability. This algorithm considers K possible hypotheses of partial translation in each time step. After the decoding ends, we pick the completed translation with the highest score in K candidates. Unfortunately, beam search increases the computational burden in each decoding step, negatively contributing to the translation speed.

Other than the speed issue, autoregressive modeling also forces researchers to implement two different computational graphs for one NMT model. The training graph predicts the target words in parallel, while the testing graph has to act like a recurrent function that processes only one word at a time.

Non-autoregressive neural machine translation has an objective similar to autoregressive models, except without the context in the conditional probability (i.e. assumes all words are independent given their positions):

$$\log p(y|x) = \sum_{t=1}^{T} \log p(y_t|x, t). \tag{1.7}$$

Each target word y_t is generated only based on the source sentence and its position t. In reality, such a model cannot be successfully trained to provide good translations, as the position alone does not encode any information about the word dependencies. Without capturing the word dependencies, the model is not capable of resolving the ambiguity of language. Consider the following example translation:

```
Source:  自動車事故に遭いました。(Japanese)
Candidate 1:  I ___ my car.
Candidate 2:  I ___ an accident.
```

Here, the source Japanese sentence has a meaning of "I was involved in a car accident.". Given two partially filled candidates, our guesses on the missing word are different. For the first candidate, the missing word shall be "crashed", whereas "encountered" for the second one. Therefore, the correct choice of the word not only depends on the source sentence, but also the target-side context. In this example, the second word (verb) depends on the last word (noun) in the sentence. As Eq. (1.7) is insufficient for capturing the word dependencies, a trained model will produce a sub-optimal translation "I crashed an accident.".

One simple solution to effectively remove the ambiguity is to include a latent variable z, by doing which we have the following objective (assuming z is continuous):

$$\log p(y|x) = \log \int p(y, z|x) dz \tag{1.8}$$

$$= \sum_{t=1}^{T} \log \int p(y_t, z|x) dz \tag{1.9}$$

$$= \sum_{t=1}^{T} \log \int p(y_t|x, z) p(z|x) dz. \tag{1.10}$$

A simple way to do that is to use the main verb in the sentence as a latent variable. Note that in this scenario, z becomes discrete. Thus, when performing translation, we first select the main verb, then generate the whole sentence as follows:

```
Source:  車事故に遭いました。(Japanese)
Latent Variable:  z = ''crashed''
Candidate 1:  I crashed my car.
Candidate 2:  I crashed my vehicle.
```

Now we can see that when we fixed the main verb, all candidate translations are valid sentences. Unfortunately, such a hand-crafted latent variable does not work for all situations in reality. A complex sentences may have multiple verb clauses, and the

ambiguity can also happen in a noun clause. Therefore, we let the neural network to automatically figure out the best latent variables by training with Eq. (1.8).

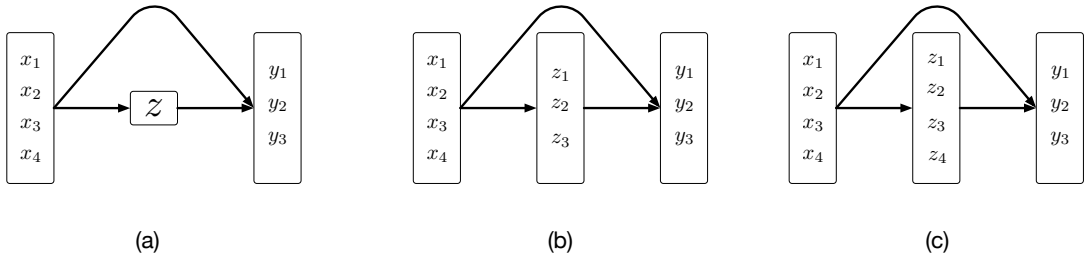


Fig. 1.7 Three different latent-variable models: (a) only one variable (b) the number of latent variables is determined by the number of target words (c) the number of latent variables is determined by the number of source words.

Now we consider how many latent variables are required to eliminate the ambiguity. In Fig. 1.7, we show three different modeling strategies. The first approach is in line with the example above, where we try to use only one latent variable to capture the dependencies. The single latent variable z can be a continuous vector, and we can model $p(z|x)$ with spherical Gaussian as $p(z|x) = N(z|\mu_x, \sigma_x)$. So the capacity of z is determined by the number of dimension in z. In our experiments, however, such a model results in disappointing performance. We hypothesize that as the amount of information to capture grows linearly (or exponentially) with the number of target words, it is tremendously difficult for the model to realize and efficiently encode the information with a fixed capacity. A perhaps better strategy can be the one in Fig. 1.7 (b), where the number of latent variables is equivalent to the number of target words. The only problem is that we have to model the target length l as another latent variable as

$$\log p(y|x) = \log \int \int p(y, z, l|x)\, dz\, dl \tag{1.11}$$

$$= \log \int \int p(y|x, l, z)p(z|x, l)p(l|x)\, dz\, dl. \tag{1.12}$$

As the probability of generating y with a length variable not matching it, that is $p(y|x, l \neq l_y, z)$, is always zero, we can remove the integral on l as

$$\log p(y|x) = \log \int p(y|x, l_y, z)p(z|x, l_y)p(l_y|x)\, dz. \tag{1.13}$$

This approach is adopted by multiple existing works on non-autoregressive neural machine translation (Gu et al., 2018a, Lee et al., 2018). In order to preform correct inference, we have to search over both the length variable l_y and the latent variable z, which lacks elegancy and may leads to sub-optimal results. The third strategy in Fig. 1.7 (c) forces the number of latent variables to be determined by the number of source tokens, which is formulated as follows:

$$\log p(y|x) = \log \int \int p(y, z, l|x)\, dz\, dl \tag{1.14}$$

$$= \log \int p(y|x, l_y, z) p(l_y|x, z) p(z|x)\, dz. \tag{1.15}$$

Contrary to previous two approaches, the third approach is essentially encoding the length information into the latent variable z. Such a modeling strategy paves the way for us to refine the latent variable after obtaining from the prior distribution $p(z|x)$, where the length prediction will be automatically updated when we update the latent variables.

As a main contribution, we derive an iterative algorithm to deterministically update the posterior on the latent variables based on an initial guess drawn from the prior. The algorithm converges rapidly and pushes higher a lowerbound on $\log p(y|x)$. In our experiments, our non-autoregressive NMT model closes the performance gap on ASPEC Japanese-to-English task with a 8.6x speed improvement. On WMT'14 English-to-German task, the model achieves only 1.0 BLEU point gap with a 6.8x faster decoding speed.

1.6.2 From Continuous to Discrete Representation Learning

The continuous representation dominates almost all recent neural NLP models as it can be efficiently trained in an end-to-end neural network. The continuous representation has the advantage of high capacity, which means it can encode the input sequence with almost no information loss. As the capacity is controlled by the number of dimensions in the vectors, one can easily enlarge the information capacity by increasing the vector sizes.

Despite the inarguable success of neural networks which learn layers of continuous representations, we can still enumerate some shortcomings of continuous representa-

tions. The main disadvantages exist in three aspect: memory efficiency, computation speed, interpretability.

Memory Efficiency A trained neural NLP model usually consumes significant amount of space when it is loaded into memory. Commonly, the bottleneck of the problem happens in the word embedding, the most basic building block of NLP models. Suppose a neural model has a vocabulary of 200K words, each word is assigned a 300-dimensional vector, then the embedding matrix will contain 60M parameters. In our experiments, about 98% of the parameters of a baseline sentiment analysis model are belong to the word embeddings. When the whole embedding matrix is loaded onto memory, the uncompressed memory consumption will be about 240 megabytes, which is far above the memory budget especially for mobile platforms.

Computation Speed The computation with continuous vectors also consumes huge amount of resource. Although using GPU acceleration can largely solve the problem, it is not widely available on low-end devices. Even the GPU acceleration is available, the intensive power consumption still hinders the application of neural models.

Interpretability In computational linguistics, the interpretability of the model is treated importantly. Learning interpretable representation can greatly help us to understand the underlying factors of a language unit which is useful in a specific task. For example, when a sentence representation is learned for sentiment analysis task, we want the representation to directly reflect the most influential aspects related to the sentiment. Let's consider the sentence "The hotel has a great service, however the room was dark and cold.". An interpretable representation shall provide insights on three aspects: "great service", "dark room" and "cold room". When a continuous representation is learned, the only way for us to reveal the information encapsulated in the vector is to plot it onto the vector space. Then, we can compare the distance among multiple different representations. However, such analysis does not provide direct insights.

1.6.3 Learning Compact Representation with Discrete Coding

Here, we are motivated to show the benefits brought by learning discrete representations for languages. In particular, we demonstrate that discrete representations have

great potential to counter both the resource-intensive problem and the difficulty of interpretation caused by the continuous counterpart. A discrete representation can be seen as a set of discrete symbols. Inside a neural model, it can be formulated either as binary vectors or one-hot vectors. Suppose we represent the word "dog" with a discrete code $(1, 3)$, then in the binary case, the code will be $\left([0, 1]^\top, [1, 1]^\top\right)$. In the one-hot case, the code will be $\left([0, 1, 0, 0]^\top, [0, 0, 0, 1]^\top\right)$. Please note that a code can mean either a continuous or a discrete variable in the literature of generative modeling. However in this thesis, it is mainly used when we mean a discrete variable.

For discrete representations, the information capacity is determined by the number of dimensions. A binary vector with K dimensions can represent 2^K symbols, whereas a one-hot vector with K dimensions can represent exactly K symbols. Thus, the one-hot representation often has a low capacity even with a high dimension.

Limited Capacity However, even for the binary vector, the information capacity is much lower than a continuous vector with the same size. Such property of discrete vectors is important for us to consider when choosing between continuous and discrete representations. The low capacity is a double-edge sword, applying it in a discriminative neural network will almost certainly cause damage to the model performance. When it is used in generative models, the limited capacity can force the model to learn an efficient latent that explains the data. Therefore, discrete representation learning is mainly being considered in the context of generative models, specifically, variational auto-encoders (Kingma and Welling, 2014).

Methodology of Training The major difficulty of training a stochastic neural net with discrete variables comes at the training phase. As we introduce a non-differentiable layer inside the neural network, the gradients for the discrete variables can not be efficiently estimated. In an early paper, Bengio (2013) evaluates four ways to train Bernoulli variables (i.e., binary codes) in the neural networks, and empirically found that the *straight-through estimator* (STE) proposed by Hinton (Hinton, 2012) in his coursera video lecture has the best performance. The idea of STE is fairly simple, which copies the gradients of a discretized layer to the continuous neurons right before the threshold function. However, the problem of STE is that such an estimator is biased and does not provide a way to learn categorical variables (i.e., one-hot codes).

In 2017, two research teams from DeepMind and Google Brain independently discovered an effective continuous relaxation for the categorical variables, and simultaneously published in ICLR (Jang et al., 2016, Maddison et al., 2016). The new

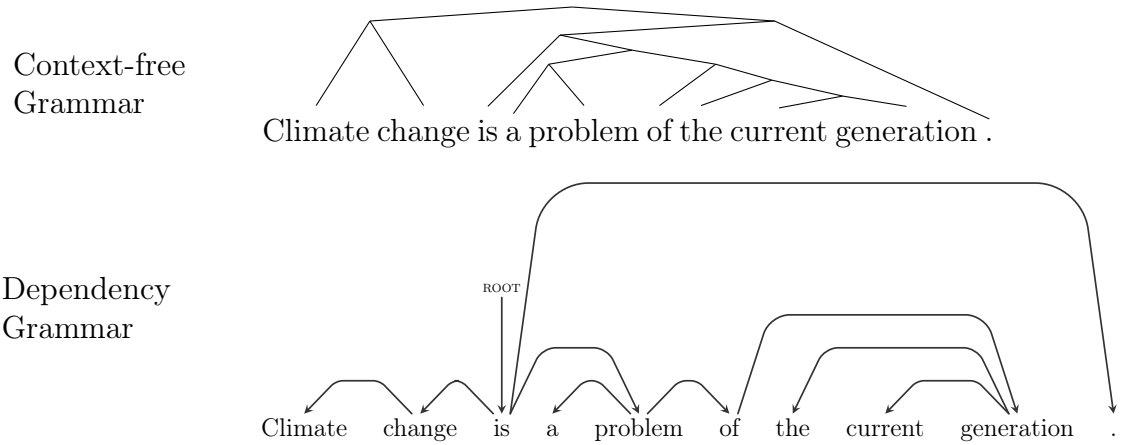


Fig. 1.8 Derived tree structures with context-free grammar (top) and dependency grammar (bottom).

method is a reparameterization trick that uses Gumbel-Softmax random variables to approximate the categorical variables. The technique can be seen as a break-through as it allows the gradients to directly backpropagate through discrete categorical samples. Experiments show that the new trick drastically outperforms other estimators such as STE with Bernoulli latents when evaluating with the variational lower bound. The details of recently proposed learning methods will be given later in chapter 3.

1.6.4 Syntactic Coding for Diverse Translation

In this thesis, we are also interested in learning latent variables to encode higher-level linguistic features rather than just words. In any language, the meaning of a sentence is created with words and grammar. Words are units that convey semantic meaning, whereas the grammar creates a structure to combine individual words together, giving birth to complex meaning. Therefore, one obvious direction worth probing is how can we learn latent variables to capture sentence structures.

In computational linguistics, researchers often adopt the formalism of content-free grammars (CFG) developed by Noam Chomsky in 1950s to describe the block structures of sentences, which is also referred to as constituency grammar. Context-free grammar creates recursive block structures. On the other hand, we also have dependency grammar (DG) that is also commonly used, which describes the relations between linguistic units in a sentence.

Both grammars can be expressed by tree structures. In Fig. 1.8, we demonstrate an example sentence and the trees induced from its grammars. For simplicity, here we omit the annotation for the non-terminal nodes and dependency relation. They are distinct in two aspects. First, for the dependency tree, each node is a word, while in the case of CFG tree, each non-terminal node covers a span. Second, a CFG tree can be and usually is a binary tree. In dependency trees, each node can have arbitrary number of child nodes.

It is challenging to learn latent variables to encode such tree structures as it is unclear on how to correctly design a generative model to let the latent variables capture similarity among different trees. In this thesis, we propose a latent-variable model that encodes the tree structures with a TreeLSTM (Tai et al., 2015) auto-encoder and produce syntactic codes with a discretization bottleneck. By using the artificially created codes to condition the generation of neural machine translation, we find that we can obtain translations with drastically different syntactic structures. In the experiments, we perform quantitative and qualitative evaluation on the diversity of translations and show the model is capable of significantly improve the diversity of candidate translations.

1.7 Contributions of This Thesis

In this thesis, we introduce latent-variable models attempting to capture different levels of linguistic features. We extensively evaluated our proposed models on neural machine translation, a typical natural language generation task. With the recently advanced neural-based machine translation models, the sentence-level translation quality is already considerably high. As the outcome of this thesis, we further make the state-of-the-art machine translation models more efficient by non-autoregressive modeling and quantization. Moreover, we show a latent-variable model for learning syntactic structures that can help machine translation models to produce highly diverse translations.

We first show the strong capability for a latent-variable generative model with continuous latent variables to direct model the machine translation problem. The results demonstrate a 8x $\sim$ 12x speed-up over the conventional neural machine translation baseline using transformers on two large translation datasets.

We also introduce the readers to discrete representation learning and its application in text generation. We provide a comparison and analysis for variously discretization methods. As a major contribution, we propose an auto-encoder to learn compositional discrete codes of words. We demonstrate the compactness of the learned codes by compressing the word embeddings over 94% without empirical performance loss. In our experiments, we also compare our methods with previous approaches including weight pruning and extension of produce quantization. The results show that our approach has a higher compression rate as the code learning model directly optimize the codes in a neural network to minimize the target loss. Moreover, we show that the learned codes have strong interpretability. Comparing the codes of multiple words that share same concepts, we are able to identify the codes for specific concepts.

The third contribution is to propose a syntactic code learning model. Here, we use the model to learn a set of codes to represent the structure of a sentence. By augmenting the sentences with the learned syntactic codes, we are able to control the output structure before generating the text. In our experiments, we show that the quality of generated sentences is not negatively affected under the constraint of the codes. By manipulating the codes, we are able to sample sentences with drastically different structures. With a proposed quantitive diversity metric, we show the sampled sentences have much higher structural diversity comparing to sampling using beam search.

This thesis is organized as follows. In chapter 2, we introduce basic network architectures and building blocks of neural NLP models including neural language model and neural machine translation. In chapter 3, we introduce the training methods of latent-variable models and some recently proposed discretization bottlenecks and their training methods. We also provide an empirical comparison of different methods.

As main contributions, in chapter 4, we introduce a latent-variable neural machine translation model trained using variational methods. In chapter 5, we describe our approach for learning word-level discrete representation, and the effectiveness on compressing word embeddings. In chapter 6, we propose a model to learn sentence-level discrete representation and its application in generating diverse translations.

The content is arranged according to the training framework of the latent-variables. The model described in chapter 4 is trained using the variational auto-encoder framework. The models in chapter 5 and chapter 6 are trained with discretization

bottlenecks. In Finally, in chapter 7, we summarize the insights we obtained from our experiments, and provide some future directions of research.

Chapter 2

Prerequisite Knowledge for Neural Machine Translation

In this chapter, we explain the core mechanisms of neural machine translation models, the neural-based components, training algorithms and methods to search for translation results.

In 2014, Devlin et al. (2014) won the best paper award in ACL'14. Their proposal is to implant a neural network based feature estimator in to a statistical machine translation decoder. By tuning on a validation corpus, the weight of this neural estimator can be automatically adjusted to maximize the translation performance. The movement of merging neural networks and the SMT models did not draw researchers to further improve the conventional approach of machine translation. Rather, it opened a new direction of research. Researchers started to think an exodus from the 20-years-old well established statistical approach to create a new family of translation models. From then on, multiple papers emerges to prob the possibility of solving machine translation problem with neural networks alone, such as some early works done by a group in Oxford (Kalchbrenner and Blunsom, 2013, Kalchbrenner et al., 2014).

The first workable end-to-end neural machine translation model is proposed by Sutskever et al. (2014), coined sequence-to-sequence model. For the first time, we are able to train the model using back-propagation of the gradients. The model is not positioned as a feature estimator to be used with statistical machine translation. Rather, the model itself is a full translation model. With simple search strategies, you can sample translations from a trained model. The work was developed on the

ideas of recurrent neural networks (RNNs), that we use one RNN to recursively encode the input sequence into one continuous vector. The vector can be consider as a highly compressed form that holds all information in the source sequence. Then we use another RNN to unroll the vector by T time steps until we reach the end of target-side sentence.

However, there was a significant performance gap between the neural sequence-to-sequence model, around 7 BLEU points away from the state-of-the-art SMT systems without ensembling. The results caused some speculations among the research community. Some negative thinking researchers were refusing to believe that the "just neural" machine translation can outperform statistical machine translation in near term. Some doubts whether deep learning is going to finally take over NLP just like what it did in computer vision.

The next break-through comes from Bahdanau et al. (2015) to significantly push up the performance, which is a follow-up work of the sequence-to-sequence model. Notably, all these three milestone works are published in the same year, implying a fast pace of the changing research field. Bahdanau et al. (2015) brings the attention mechanism from the vision literature (Mnih et al., 2014) to neural machine translation. Based on this proposal, Google built its large-scale neural machine translation system (Wu et al., 2016), that finally brings the translation quality very close to human level.

The proposed attention mechanism finally leads to a recently massive evolution of neural machine translation, the Transformer (Vaswani et al., 2017). Transformer stacks layers of attention-based computations. The model results higher translation quality and is much faster to train comparing to previous recurrent-based models. Recently, transformer becomes the de-facto model in neural machine translation. In the following sections, we introduce the details of the major model families for neural machine translation. We put focus on understanding how different models process high flexible sequences of natural languages.

2.1 Recurrent Neural Networks

Recurrent neural networks are basic components in the neural networks for NLP. In computer vision tasks, the model only deals with inputs that have fixed shapes. In contrast, the most distinguishable difference in NLP tasks is that we have to deal

with a sequence consisting arbitrary number of tokens. In machine translation, we have to further generate a sequence with arbitrary number of tokens in the target side. The first step is to build a vector representation of a given sentence.

Computing Softmax To begin with, we first consider a simple neural model that predicts the next word. To represent words as continuous vectors to be used in neural networks, we creates "embeddings" vectors for all words in our vocabulary. In practice, the number of words in a vocabulary ($|V|$) can range from 20K to 200K. Therefore, if we stack the word embeddings together, it will be a matrix that have a shape $|V| \times D$, where D is the dimensionality of each vector. We use $\mathrm{E}(w)$ to denote an operation that returns the embedding vector for word w.

To compute the probability for w_i to be the next word, we have the following equation:

$$p(w_{t+1} = w_i | w_t) = \frac{\exp(s_i)}{\sum_j \exp(s_j)}, \tag{2.1}$$

where s is a vector containing the logits before probability normalization. The loigits can be computed by a matrix-vector multiplication as

$$s = \mathbf{W_o}^\top \mathrm{E}(\mathrm{w_t}) + b_o, \tag{2.2}$$

where $\mathbf{W_o}$ is a $D \times |V|$ weight matrix, and b_o is a bias vector. For convenience, we denote the probability prediction of all words with a softmax operation:

$$p(w_{t+1} | w_t) = \mathrm{softmax}(\mathbf{W_o}^\top \mathrm{E}(\mathrm{w_t}) + b_o). \tag{2.3}$$

Elman Recurrent Neural Network Note that in Eq. (2.3), the neural network for next-word prediction is a simple linear model. To increase the expressiveness of this model, we add a hidden layer with an activation function as

$$h_t = \tanh(\mathbf{W_x}^\top \mathrm{E}(\mathrm{w_t}) + b_h), \tag{2.4}$$

$$p(w_{t+1} | w_t) = \mathrm{softmax}(\mathbf{W_o}^\top h_t + b_o), \tag{2.5}$$

where the first line of the equations computes the hidden state with two new parameters: $\mathrm{W_x}$ and b_h. The *element-wise* activation function $\tanh(\cdot)$ is defined as

$$\tanh(\mathrm{x}) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}. \tag{2.6}$$

Next, we move on to consider including more context in improve the prediction of next words. To do this, we add all preceding words in a sentence to the condition side of the probability, which gives $p(w_{t+1}|w_1, ..., w_t)$. In this paper, we denote a sequence of words $w_1, ..., w_t$ using $w_{1:t}$ for simplicity. As we are already have a model to compute the probability $p(w_2|w_1)$, the next step is to consider how can we compute $p(w_3|w_1, w_2)$. The difference here is that we have to build a vector representation of multiple tokens instead of only one token. In other words, we have to expand the hidden state computed in Eq. (2.4) to enable it capture arbitrary numbers of tokens. The obvious way to achieve this is to include the hidden state in a previous time step into the equation as

$$h_t = \tanh(\mathbf{W_h}^\top h_{t-1} + \mathbf{W_x}^\top \mathrm{E(w_t)} + b_h), \tag{2.7}$$

$$p(w_{t+1}|w_{1:t}) = \mathrm{softmax}(\mathbf{W_o}^\top h_t + b_o). \tag{2.8}$$

Now we add a new parameter $\mathbf{W_h}$ to the model for processing the hidden states in different time steps. We notice that Eq. (2.7) becomes a recurrent function, which can be simply denoted with

$$h_t = f_{\mathrm{RNN}}(h_{t-1}, w_t). \tag{2.9}$$

The parameterization of f_{RNN} described in Eq. (2.7) is firstly proposed by Elman (1990), and thus is also referred to as Elman networks. We call any neural network that implements this framework a *recurrent neural network*.

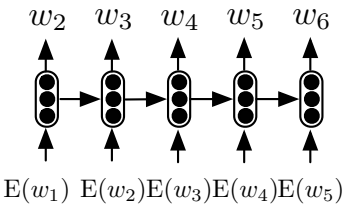


Fig. 2.1 Illustration of the computational graph of recurrent neural networks

Long Short-term Memory Units In Fig. 2.1, we illustrate the computational graph of recurrent neural networks. We can see that the hidden states have similar function as the memory cells. A hidden state h_t holds the information of all preceding words $w_{1:t}$ until step t. The state vectors have to be arranged in a way that the softmax operation can utilize the information contained in them to predict the probability of the next word. In the same time, the vectors also have to preserve useful information for future steps although they are blind to future inputs.

However, it is easy to notice that Eq. (2.7) will completely rewrites all dimensions of the hidden states in each time step. Therefore, information can be easily erased from the vectors, and such implementation of the recurrent function also causes *gradient diminishing problem*. The error signal from a future word will diminish quickly when propagated in the backward direction. For this reason, we have to design a mechanism having stable memory cells. The information in the memory cells should remain intact if there is no need to update them. The most broadly adopted variation of recurrent neural network is Long Short-term Memory (LSTM) (Hochreiter and Schmidhuber, 1997), which is now the de-facto implementation of RNN in NLP tasks.

In LSTM, two separate units are updated in each time step. The memory cells keep useful information, which can only be updated when an input gate function is activated. The gate is controlled based on the values in the hidden states. We denote the hidden state in time step t as h_t, the memory cell as c_t. First, the input gate produces a sigmoid vector with

$$\begin{aligned} i_t =& f^i_{\text{gate}}(w_t, h_{t-1}, c_{t-1}) && (2.10)\\ =& \sigma(\mathbf{W_{xi}}^\top \mathrm{E}(w_t) + \mathbf{W_{hi}}^\top h_{t-1} + \mathbf{W_{ci}}^\top c_{t-1} + b_i), && (2.11) \end{aligned}$$

where $\sigma(\cdot)$ is a sigmoid activation function defined as $\sigma(x) = \exp(x)/(\exp(x)+1)$. The purpose of applying sigmoid activation function is to ensure that the values in the gating vectors range from 0 to 1. We can control the flow of information with an element-wise multiplication with another vector. When the value of a specific dimension is close to zero, it blocks the information in the corresponding dimension in the other vector. Similarly, we also have forget gate and output gate in LSTM as

$$\begin{aligned} i_t =& f^i_{\text{gate}}(w_t, h_{t-1}, c_{t-1}), && (2.12)\\ o_t =& f^o_{\text{gate}}(w_t, h_{t-1}, c_t). && (2.13) \end{aligned}$$

Both gates are computed in the same way as the input gate. One exception is that the output gate is computed based on the memory cell in the current time step c_t, as it is designed to apply after updating the cells. For updating the memory cells, LSTM first propose a new cell content $\tilde{c}_t$ with

$$\tilde{c}_t = \tanh(\mathbf{W_{xc}}^\top \mathrm{E}(w_t) + \mathbf{W_{hc}}^\top h_{t-1} + b_c). \tag{2.14}$$

Then we use the input gate and forget gate to selectively write and remove contents from the cells, and use the output gate to selectively write information back to the hidden states with

$$c_t = f_t c_{t-1} + i_t \tilde{c}_t, \tag{2.15}$$

$$h_t = o_t \tanh(c_t). \tag{2.16}$$

The computational graph of LSTM is summarized in these two lines of equations [1]. When the values in the forget gate f_t is 1, and the values in the input gate i_t is 0, the memory cell c_t be an exact copy of the cell in the previous step. The recurrent model is thus capable of preserving the information for multiple time steps to capture long-distance dependencies among the tokens. When the memory cell is not changed, the gradient will also avoid being diluted during back-propagation, thus prevent the magnitude of the gradients to vanish. For comparison, Elman network can be written as $h_t = \tilde{c}_t$.

Training of Recurrent Neural Networks To train the recurrent neural networks with back-propagation, we have to correctly estimate the gradients for all parameters in the recurrent function f_{RNN}. Fortunately, it is still straight-forward. As we can see that the computation over time can thought as stacked feed-forward neural networks with shared parameters. For instance, the hidden state in the second step can be written as

$$h_2 = f_{\mathrm{RNN}}(f_{\mathrm{RNN}}(h_0, w_1), w_2), \tag{2.17}$$

here h_0 is the initial hidden state, usually a zero vector. This suggests a backward path to pass gradient from the end of sequence back to the beginning of the sequence. Such way of gradient computation is named back-propagation through time (BPTT).

[1] There are multiple variations of LSTM computation. We follow the equations in Graves (2013).

However, the downside of BPTT is that the high backward cost when we have a long path. In language model tasks, a single sequence may contain all words in a paragraph. In this scenario, it will be extremely time-consuming to compute the full BPTT. As a compromise, *truncated* BPTT is often used in practice, which back-propagate the gradients only for a fixed number of time steps. Truncated BPTT can limit the backward cost in a fixed budget, it however may damage the model performance for capturing long-term dependencies.

2.2 Neural Machine Translation

Autoregressive Modeling In machine translation, our goal is to translate a source sequence $X = \{x_1, ..., x_{|X|}\}$ into a target sequence $Y = \{y_1, ..., y_{|Y|}\}$. Until recently, the majority of neural machine translation works adopt autoregressive modeling to tackle the problem, where we model the conditional model as

$$p(Y|X) = \prod_{t=1}^{|Y|} p(y_t|y_{1:t-1}, X). \tag{2.18}$$

The equation suggests to generate one target token in each time step. The generation is conditioned on the source sequence X and the a history of tokens $y_{1:t-1}$ in target side. Therefore, with this modeling strategy, the NMT model is effectively a conditional language model. To train the NMT model, it is a common practice to minimize the negative log likelihood as a loss function:

$$\mathcal{L} = -\sum_{t=1}^{|Y|} \log p(y_t|y_{1:t-1}, X). \tag{2.19}$$

Encoder-decoder Framework With the basis of recurrent neural networks, we move on to introduce the architecture of NMT model. In 2014, Sutskever et al. (2014) proposed *encoder-decoder* framework for translating one sequence into another sequence, which is illustrated in Fig. 2.2.

We can see from the figure that there are two main components in the encoder-decoder framework. The encoder creates a representation of the source sequence, which is fed into the decoder. The decoder produces hidden vectors that contain

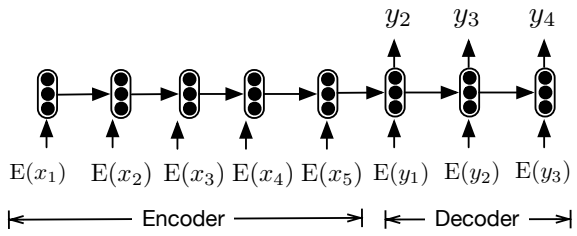


Fig. 2.2 Encoder-decoder framework of neural machine translation

information for predicting the target tokens. The framework can abstracted as

$$h^{\text{src}} = \text{Encode}(\text{E}(x_1), ..., \text{E}(x_{|X|})), \tag{2.20}$$

$$h_t^{\text{tgt}} = \text{Decode}(h^{\text{src}}, h_{1:t-1}^{\text{tgt}}, \text{E}(y_t)), \tag{2.21}$$

$$p(y_t|y_{1:t-1}, X) = \text{TokenPredict}(h_t^{\text{tgt}}), \tag{2.22}$$

where the "TokenPredict" function in the last line is implemented with a softmax applied after computing the logits. Surprisingly, even after 5 years, almost all newly proposed translation models still follow the encoder-decoder framework. In the original work of Sutskever et al. (2014), both the encoder and decoder are implemented with LSTMs. The encoder state h^{src} is the final hidden state produced by the encoder LSTM. As the encoder-decoder framework does not estimate the probability of emitting the first target token y_1, we usually use a begin-of-sentence token "BOS" to prefix the target sentence. We also append a "EOS" token to the tail of the target sentence as a sign to stop generation.

Teacher Forcing and Exposure Bias As a result of autoregressive modeling, given a trained model, we recurrently apply argmax on the probability to obtain the prediction of translation as

$$\hat{y}_t = \underset{y_t}{\text{argmax}}\, p(y_t|y_{1:t-1}, X). \tag{2.23}$$

However, the issue is that if we apply argmax to sample tokens in the training time, the model cannot be well trained. As argmax is a discrete node in the computational graph, the gradients cannot be correctly estimated for it. To bypass the issue, although we sample from model distribution during inference, in the training time, we force to model to take tokens from training data (data distribution) as input. The consequence of *teacher forcing* is that the model experiences *exposure bias*, which

means the inputs to the model are from a different distribution during inference. This issue remains as an open problem.

Batching and Sharding To speedup the training phase, we combine sentences from different datapoints into mini-batches. As different sentences have different number of tokens, suppose a specific sentence is shorter than the longest one in the batch, we fill the remaining positions with zeros. A source or target batch will appear similar to the following matrix

```
x = [
[1, 32, 54, 75, 71, 53, 63, 2,  0, 0, 0],
[1, 25, 58, 42, 68, 30, 57, 18, 40, 42, 2],
[1, 53, 34, 21, 2, 0, 0, 0, 0, 0, 0]
]
```

To reduce the number of padded zeros in the batches, we may sort the training sentences according to their lengths. The sorting step force each batch to contain sentences with similar numbers of tokens. As GPU are good at parallel computing, the training is more efficient when batching more sentences together. When we attempt to train models with large batches, we however start to encounter the insufficient GPU memory problem. The bottleneck typically occurs in the softmax computation. Recall that weight parameter for computing the logits is a $D \times |V|$ matrix. To make the problem worse, the hidden states now have a shape of $B \times T \times D$, where B is the batch size and T is length of the longest sentence in the batch. The result of this matrix-tensor multiplication will have a shape of $B \times T \times |V|$, which takes tremendously huge memory space on GPUs.

To alleviate the problem, we slice the a single batch further into multiple shards before computing softmax. For each shard, we continue to compute the final cross-entropy loss and obtain the gradients on the decoder states $h_1^{\text{tgt}}, ..., h_{|Y|}^{\text{tgt}}$. Similarly, we obtain gradients for the decoder states iteratively for all shards, and accumulate gradients on the decoder states. By doing this, the shape of the logit tensor will not exceed $S \times T \times |V|$, where S is the number of batches in a shard. After the gradients are accumulated on the decoder states for all shards, we further back-propagate the gradients to other nodes in the computational graph. The pseudo-code of sharded training is demonstrated in the following snippet.

```
B ← batch size
S ← shard size
h^src ← Encode(E(x_1), ..., E(x_|X|));
h^tgt ← Decode(h^src, E(y_1), ..., E(y_|Y|));
For i ← 1 to floor(B/S):
    s_i^tgt ← h^tgt[i × S : (i + 1) × S] ;
    loss ← CrossEntropy(TokenPredict(s^tgt));
    BackPropagateToNode(loss, h^tgt);
Loop;
BackPropagate(h^tgt);
```

2.3 Attention Mechanism

In our discussion on training recurrent neural networks, we articulate that one issue of truncated BPTT is that the resultant model may fail to capture dependencies between two distant tokens. One the other hand, if we consider this problem more thoroughly, we may realize the root cause is not the training algorithm. In a classical RNN-based language model or neural machine translation model, we rely on an assumption that the memory cells in LSTM are capable of encoding all information in a sequence, even when the given sequence is extremely long. This assumption however does not hold a the memory cells only have fixed number of dimension, implying a limited capacity. To release the pressure on the memory capacity, one clever way is to introduce directly links between source and target tokens, proposed by Bahdanau et al. (2015).

The core idea is to feed a specific source-side hidden state at position i to the recurrent computation in the decoder LSTM as

$$h_t^{\text{att}} = h_i^{\text{src}}, \tag{2.24}$$

$$\tilde{c}_t = \tanh(\mathbf{W_{xc}}^\top \mathrm{E}(w_t) + \mathbf{W_{hc}}^\top h_{t-1} + \mathbf{W_{ac}}^\top h_t^{\text{att}} + b_c), \tag{2.25}$$

where the source position i is determined by a separate model:

$$i = \text{HardAttention}(h_{t-1}, h_1^{\text{src}}, ..., h_{|X|}^{\text{src}}). \tag{2.26}$$

We refer to such a mechanism as *hard attention*, as the the model choose only one source state to look at. As the hard attention mechanism is making a discrete decision when selecting source states, the model has to be trained with reinforcement learning or other tricks but not simple end-to-end back-propagation. With the high variance generated by reinforcement learning, the hard attention mechanism makes the model training more challenging, and potentially cancels out its positive impact.

To maintain the end-to-end trainable nature of the neural network, the authors (Bahdanau et al., 2015) propose to apply a soft version of the attention mechanism, we call it *soft attention*. Instead of picking only one source states, soft attention considers all source states by assigning weights on different states as

$$h_{\text{att}} = \sum_{i=1}^{|X|} a_i^t h_i^{\text{src}}, \tag{2.27}$$

$$a_i^t = \frac{\exp(\tilde{a}_i^t)}{\sum_{j=1}^{|X|} \exp(\tilde{a}_j^t)}, \tag{2.28}$$

$$\tilde{a}_i^t = v_a^\top \tanh(\mathbf{W_{ax}}^\top h_i^{\text{src}} + \mathbf{W_{ay}}^\top h_{t-1}^{\text{tgt}} + b_a). \tag{2.29}$$

where the attention vector is produced by a weight summarization over the source states. The weights are normalized to be a categorical distribution. The logits of this distribution are computed by a one-layer neural network, which introduces three new parameters. Other than this neural-network approach, a simpler way of computing the attentional weights is proposed by Luong et al. (2015) that simply computes the dot product as

$$\tilde{a}_i^t = (h_i^{\text{src}})^\top h_{t-1}^{\text{tgt}}. \tag{2.30}$$

the dot-product approach is much simpler and faster. However, it shall be used cautiously. Firstly, the approach assumes both vectors have same dimensions. Secondly, the dot-production produces higher weight values when two vectors are similar. Sometimes, we want the model to attend on distinct vectors but not similar vectors.

To wrap up, attention mechanism, especially soft attention is the break-through that significantly boost the translation quality. It solve three major problems of the recurrent neural networks within one model: (1) truncated BPTT problem (2) memory capacity problem (3) it learns the alignments among tokens.

2.4 Transformer

Although LSTM-based neural machine translation models are widely adopted due to the simplicity and significant performance gain, one critical drawback still exists when using recurrent neural networks. As each hidden state h_t in recurrent nets is computed based on previous states, the computation in each time step has to "wait" for previous computations to finish. When training a large model, this constraint on training speed makes the process enormously time-consuming. Researchers in machine translations community started to realize that this issue has to be solved.

Fully Convolutional Neural Machine Translation Around 2017, multiple works have been proposed aiming to tackle this problem. Gehring et al. (2017) proposed to replace LSTM-based encoder and decoder with fully convolutional neural networks. Because 1-d convolution does not force the temporal dependency, the computation during training can be greatly speedup. When the filter size is 3, the t-th hidden state in l-th layer is computed with

$$h_t^l = \phi(\text{conv}(h_{t-1}^{l-1}, h_t^{l-1}, h_{t+1}^{l-1})), \tag{2.31}$$

where $\phi(\cdot)$ is an non-linear activation function. The convolution-based models suffer from two critical problems. First, when as we can observe from Eq. (2.31), if our model has only one convolutional layer then it considers only the neighboring tokens when building the hidden representation for a specific token. When the model stacks two convolutional layer, it is capable of considering 5 surrounding tokens in the encoder side. For the decoder side, however, as the convolution is masked to consider only the left-hand tokens, it considers only 2 preceding tokens in the case of two convolutional layers. Generally, when N convolutional layers are stacked, the encoder is capable of considering $2N + 1$ tokens as context, the decoder is capable of considering N contextual tokens. Therefore, the convolution-based NMT model has to be considerably deep to capture long-range dependencies. Second, even the decoder has 20 convolutional layers, which means that each target token is predicted based on 20 tokens in the left side, the model can still fail to get useful information for the prediction. As in some languages such as Japanese, the last word may depends on the first word in the sentence, and a sentence may have more than 100 words. In this case, even a 20-layers deep model has no chance to make a correct prediction for the

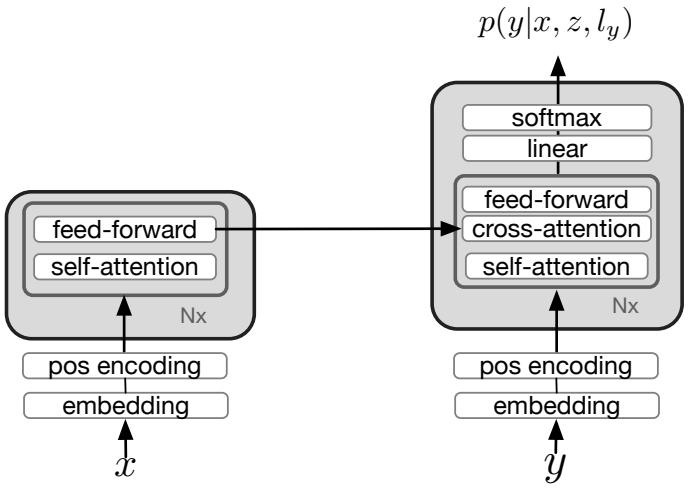


Fig. 2.3 Illustration of the model architecture of Transformer

last word. These drawbacks significantly limits the application of convolution-based NMT models.

Key-Value Attention To fundamentally improve the problematic recurrent-based NMT models. (Vaswani et al., 2017) proposed to only use attention mechanism to build a translation model, which finally gained huge success. The core idea of Transformer is based on key-value attention (Miller et al., 2016), which is important for understanding the mechanisms in Transformer. Recall the equation for computing soft attention with weighted summarization:

$$h_{\text{att}} = \sum_{i=1}^{|X|} \text{align}(h_{t-1}, h_i^{\text{src}}) h_i^{\text{src}}. \tag{2.32}$$

Here, we use align$(\cdot)$ to abstract a function that determines the weight for each source states. Examining the equation, we can see that the equation is similar to the process of finding a item in a key-value store, that has the following form when one query only matches one key:

$$\text{result} = \sum_{i=1}^{|\text{key}|} \text{match}(\text{query}, \text{key}_i)\text{value}_i, \tag{2.33}$$

where in the equation, the match$(\cdot)$ function returns 1 when the query matches a key. Otherwise, it returns 0. The soft attention in Eq. (2.32) can be thought as a

soft version of this searching process. Therefore, we can generalize Eq. (2.32) to

$$h_{\text{att}} = \text{Attention}(q_t, k, v) \tag{2.34}$$

$$= \sum_{i=1}^{|X|} \text{match}(q_t, k_i) v_i, \tag{2.35}$$

$$\text{match}(q_t, k_i) = \frac{\exp({q_t}^\top k_i)}{\sum_{j=1}^{|X|} \exp({q_t}^\top k_j)}. \tag{2.36}$$

Here, we compute the soft matching function with dot product. The arguments of this equation, q_t, k_i, v_i, are then computed with linear transformations as

$$q_t = {\mathbf{W_q}}^\top h_{t-1} + b_q, \tag{2.37}$$

$$k_i = {\mathbf{W_k}}^\top h_i^{\text{src}} + b_k, \tag{2.38}$$

$$v_i = {\mathbf{W_v}}^\top h_i^{\text{src}} + b_v. \tag{2.39}$$

Comparing with Eq. (2.32), we found that the major difference is that the key k_i and value v_i now can be different vectors. This allows the value v_i to have more capacity because it is no longer being used in the matching function.

Multi-head Attention We continue to extend the power of attention mechanism. Examining Eq. (2.35), we can see that when match$(\cdot)$ is an indicator that returns either 1 or 0. The attention mechanism can only gather information from one source position. In order to gather information from multiple sources, we have to stack multiple layers of such attention models. Although match$(\cdot)$ draws a categorical distribution on source states but not a indicator in reality, source information can be diluted if we mix too many vectors together. One clever workaround is to compute multiple attentions in parallel. Here, we call each attention vector a head, which can be thought as a glimpse to the source sequence. If we compute K attention heads, then the decoder can gather information from K different source positions with just one attention layer. We formally define multi-head attention as

$$\text{MultiHeadAttention}(q_t, k, v) = [h_1^{\text{head}}; ...; h_K^{\text{head}}], \tag{2.40}$$

where [...] is a concatenation of all attention heads. However, if we compute each head with the normal key-value attention model, the output of multi-head attention will be have K times more dimensions comparing to other hidden vectors in the

neural network. To avoid this problem, we reduce the dimensionality of each head by a factor of $1/K$. Suppose the hidden states have N dimensions, then each head is computed with

$$h_k^{\text{head}} = \text{Attention}(q_t[kN/K:(k+1)N/K], k_i[kN/K:(k+1)N/K], v_i[kN/K:(k+1)N/K]). \quad (2.41)$$

As suggested by the equation, we actually computing separated attentions on different portions of the vectors.

Self-Attention and Cross-Attention Next, we describe the most important components in a Transformer model. Self-attention builds complex sentence-level representation by summarizing information from multiple tokens in the same sentence. When applying on the source sequence, self-attention computes attention vector for a source state h_i^{src} with

$$\text{SelfAttention}(h_i^{\text{src}}) = \text{MultiHeadAttention}(h_i^{\text{src}}, h^{\text{src}}, h^{\text{src}}). \quad (2.42)$$

We can see from the equation that both query, keys and values are from the same sequence. The same computation can also applied on the target sequence. Very similarly, we can use query vectors from one sentence, in machine translation the target sentence, to match vectors in the source sentence. We define cross-attention as

$$\text{CrossAttention}(h_t^{\text{tgt}}) = \text{MultiHeadAttention}(h_t^{\text{tgt}}, h^{\text{src}}, h^{\text{src}}). \quad (2.43)$$

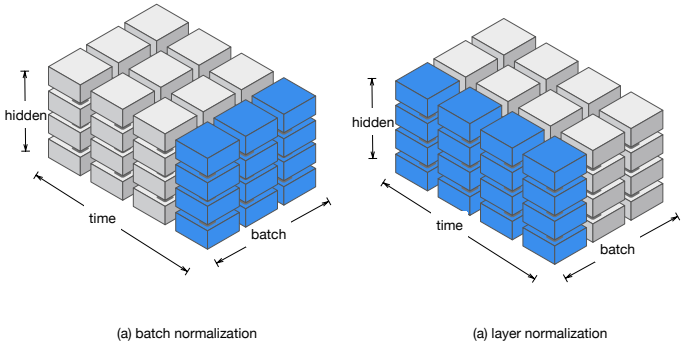
Cross-attention modules only exist in the decoder. Transformer uses cross-attention to selectively gather source-side information as evidence to improve the prediction of target tokens.

Feed-forward and Layer Normalization The attention mechanism alone does not provide the biggest benefits of neural networks: non-linearity. Transformer adds non-linear activations using feed-forward layers, which are composed by two linear transformations and a ReLU(Nair and Hinton, 2010) activation as

$$\text{FF}(h) = \mathbf{W_2}h' + b_2, \quad (2.44)$$

$$h' = \text{ReLU}(\mathbf{W_1}h + b_1). \quad (2.45)$$

Note that the first linear transformation parameterized by $\mathbf{W_1}$ and b_1 increases the dimensions by a factor of 8 according to the paper. Then the second linear transformation parameterized by $\mathbf{W_2}$ and b_2 reduce the dimensions and recover the original vector sizes. The non-linear activation is applied on the enlarged intermediate vector. The feed-forward computation is applied after attentional computations.



(a) batch normalization (a) layer normalization

Fig. 2.4 Comparing batch normalization and layer normalization

Because the Transformer is a 12-layer deep model, we want the training to be as stable as possible. As a quick trick, layer normalization is applied in the output end of each Transformer layer. layer normalization is similar to batch normalization (Ioffe and Szegedy, 2015). The only difference is that batch normalization is applied on the batch dimension, but layer normalization is applied on the hidden dimension. The comparison is illustrated in Fig. 2.4. For layer normalization the resultant vector x'_b in batch b and position i is computed with

$$x'_{bi} = \gamma \frac{x_{bi} - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}} + \beta, \tag{2.46}$$

$$\mu_b = \frac{1}{N} \sum_{i=1}^{N} x_{bi}, \tag{2.47}$$

$$\sigma_b^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{bi} - \mu_b)^2, \tag{2.48}$$

where, γ and β are trainable parameters of layer normalization, and ϵ is a small number to ensure the denominator does not drop to zero.

2.5 Training

For training NMT models, we normally apply stochastic gradient descending methods to minimize the log-likelihood over training data D:

$$\hat{\theta} = \operatorname*{argmin}_{\theta} \mathbb{E}_{(X,Y)\sim D}\big[\log p(Y|X;\theta)\big]. \tag{2.49}$$

We can monitor the log-probability averaged for each token with

$$J(\theta) = \mathbb{E}_{(X,Y)\sim D}\big[\frac{1}{T}\sum_{t=1}^{T}\log p(y_t|y_{<t},X;\theta)\big]. \tag{2.50}$$

Although the average log-probability is high correlated with the model performance (i.e. translation quality), the parameters saved when the model reaches a highest log-probability on a valid dataset may not yield highest translation quality. The reason has at least two folds. The first cause is teacher forcing, In Eq. (2.50), the preceding tokens in the conditional probability are directly obtained from the data samples. During real decoding, they are sampled from the model, which has a different distribution. The second cause comes from the evaluation metric of translation quality. The average log-probability evaluates the performance for each token independently. However, most of time, our evaluation metric for translation quality considers phrases (n-grams) rather than just individual words. Therefore, the token-level log-probability cannot reflect the translation quality globally.

A simple work-around is to run the real decoding algorithm on the valid dataset several times in each epoch, and report the BLEU scores. This approach, however, is not easy to implement due the huge difference in the training computational graph and decoding computation graph. In the current state-of-the-art transformer models, the model is trained for a fixed number of iterations. Then the final parameters are selected by averaging 5 to 10 different checkpoints.

2.6 Decoding and Evaluation

To generate translation results, NMT models rely on search algorithms, particularly beam search, to find a good translation candidate. Beam search finds a hypothesis Y that maximizes the log probability $\log p(Y|X)$, given an input sentence X. In

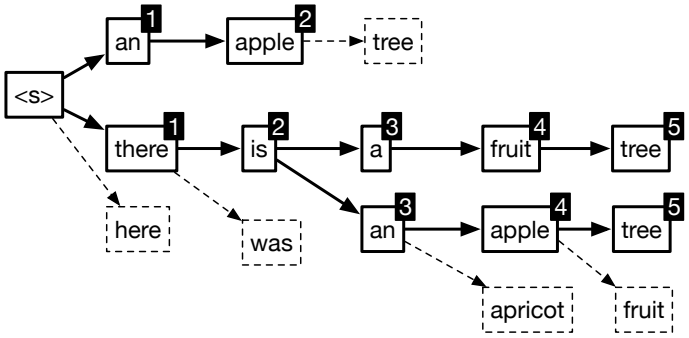


Fig. 2.5 Illustration of beam search algorithm in the setting of Japanese-to-English translation task. The beam size here is 2. The numbers show the step when running the algorithm.

each step, a fixed number of hypotheses are considered by the algorithm. Then the NMT model predicts the probabilities of the next output token for each hypothesis. Suppose that the fixed number (beam size) is B, and the vocabulary size is V. Then, theoretically, we can obtain a maximum of $B \times V$ new hypotheses. Beam search then keeps top-zB hypotheses with highest log probabilities. Thus, the hypotheses considered in each step have the same length (i.e., number of tokens). The algorithm ends when B finished translations are collected.

Deficiency of Beam Search Since the beam size is fixed, when the algorithm attempts to explore multiple new decoding paths for a hypothesis, it has to discard some existing decoding paths. However, the new decoding paths may lead to bad hypotheses in the near future. As past hypotheses can not be revisited again, the beam search has to decode the hypotheses with degraded qualities continually. This phenomenon is illustrated in Fig. 2.5 (a), where the graph depicts the decoding process of a sentence. The correct output is supposed to be "*an apple tree is there*" or "*there is an apple tree*". In step 3, as the algorithm explores two new hypotheses in the bottom branch, the hypothesis "*an apple tree*" is discarded. In the next step, it realized that the hypothesis "*there is a*" leads to a wrong path. However, as the algorithm can not return to a discarded hypothesis, the beam search has to keep searching in the hopeless path. In this case, the candidate "*an apple tree is there*" can never be reached.

Automatic Evaluation Evaluating machine translation results automatically is a challenging task itself. To accurately evaluate a small amount of translation results,

hiring expert translators is still the best choice nowadays. The main purpose of automatic evaluation is not to reveal the quality of a specific set of translations, but to compare two systems. Commonly used evaluation metrics are used to seek the statistical significance when applied on a large dataset. Therefore, comparing the automatic scores on one data sample is statistically meaningless.

The most commonly used metric is BLEU score (Papineni et al., 2001). The statistical significance test for BLEU scores is proposed in Koehn (2004). Given a translation hypothesis h and a reference r, BLEU measures the overlap of n-grams in h and r. Typically, it considers the precision from uni-grams to 4-grams. A brevity penalty is introduced to punish short translations.

The correlation between BLEU scores and human evaluation can be problematic depending on the languages. Recently, increasing research papers adopt newly proposed metrics such as RIBES (Isozaki et al., 2010a) and AF-MF (Banchs and Li, 2011). However, the majority of the neural machine translation community still uses BLEU as the main evaluation metric due to complicated reasons.

Chapter 3

Prerequisite Knowledge for Latent-Variable Likelihood Models

In regular neural machine translation models, translation is modeled as a classification problem. For each generation step, a discriminative model is used to produce the probability distribution of emitting a certain word $p(y_t|y_{<t}, X)$[1]. Discriminative models are considered to be more powerful and easier to train because the model can choose to omit all information unrelated to the final prediction. In language modeling, a typical case is that if a natural or artificial language has only short-term dependency among tokens, then the discriminative model can just consider few preceding tokens and exploit the full model capacity to predict the next token.

3.1 Approximate Inference

In contrast to discriminative models, a generative model attempts to capture the generation process of observed data. The distribution of interest is therefore $p(x)$. In order to correctly estimate this probability, all useful information in the given data has to be considered. If the given data is an image of cat, then the model has to know all subtle features of a real cat image, so the model can decide whether to assign a high probability to any input datapoint. For natural language, a model has

[1]Note that in natural language generation, the boundary between discriminative model and generative model is difficult to define clearly. The word-prediction model $p(y_t|y_{<t}, X)$ can also be viewed as a generative model where y_t is not treated as label, but the data we want to model.

to know the correct relations among multiple words or phrases, so it can correctly identify whether a sentence is natural.

To learn the probability distribution $p(x)$, it is common in generative models to introduce one latent variable z or several latent variables $\mathbf{z} = z_1, \ldots, z_K$. We assume the relation between the observed variable x and the (unobserved) latent variable z follows the directed probabilistic graph $z \to x$, which tells that the joint probability of x and z can be computed by

$$p(x, z) = p(x|z)p(z). \tag{3.1}$$

The latent variable helps to capture the hidden mechanism that produces the data distribution. Latent variables in successfully learned generative models explain the data distribution in an efficient way. In Table 3.1, we give two examples showing how latent variables can interpret the observed data.

x	z_1	z_2	z_3
dog images	skin color	angle of camera	size of the dog
movie review	sentiment	object of complaint	number of sentences

Table 3.1 Examples of how latent variables interpret the data distribution

It is important to note that the latent variables can also be the parameters of the distributions, which are usually denoted by θ rather than z. Inference in such a Bayesian model requires to compute the following posterior:

$$p(z|x) = \frac{p(x|z)p(z)}{\int p(x|z)p(z)dz}. \tag{3.2}$$

The integral in the denominator is intractable when the likelihood $p(x|z)$ is a complicated model or the amount of data is too large. In machine learning community, researchers often use *approximate inference* as a solution to this problem.

Classical approaches for approximate inference rely on sampling. The most well-explored approach is Markov Chain Monte Carlo (MCMC) (Hastings, 1970, Gelfand and Smith, 1990). In MCMC, we construct a Markov chain so that the stationary distribution equals to the posterior $p(z|x)$. Based on the stationary distribution, we can collect samples and form approximation of the true posterior. Popular methods in MCMC includes Metropolis-Hastings algorithm (Metropolis et al., 1953, Hastings, 1970) and Gibbs sampling (Geman and Geman, 1984).

The drawback of MCMC methods is the speed. Sampling is required every time we want to compute the posterior. This problem becomes critical when we use deep neural networks to model the distribution. With expensive models, the time cost for computing each sample increases significantly.

To solve the same problem, contrasting to the sampling approach, *variational inference* developed by (Jordan et al., 1999, Wainwright and Jordan, 2008) formulates the inference as an optimization problem. The core idea of variational inference is to restrict the family of an approximate posterior $q(z)$ of the latent variables. Then we minimize the Kullback-Leibler (KL) divergence between the approximate posterior and the true posterior to find the best approximate posterior $q(z)$ inside the family:

$$\operatorname*{argmin}_{q(z)} \mathrm{KL}(q(z)||p(z|x)). \tag{3.3}$$

Note that in order to make a more accurate approximate posterior, we may use $q(z|x)$ rather than $q(z)$.

Evidence Lower Bound However, the KL divergence in Eq. (3.3) is not directly computatable because $p(z|x)$ is intractable in our problem setting, which is also the motivation of approximate inference. The core problem is the intractability of $\log p(x)$, which is also called *evidence*. By marginalizing out the latent variables, we know it can be computed by

$$\log p(x) = \log \int p(x, z) dz. \tag{3.4}$$

Although this density cannot be directly computated, we can derive a lower bound by using Jensen's inequaliy. Such a derivation was described in (Jordan et al., 1999), which has three main steps. First, we write the marginal density in the form of an expectation under $q(z)$:

$$\log p(x) = \int p(x, z) dz \tag{3.5}$$

$$= \log \int q(z) \frac{p(x, z)}{q(z)} dz \tag{3.6}$$

$$= \log \mathbb{E}_{z \sim q(z)} \Big[\frac{p(x, z)}{q(z)} \Big]. \tag{3.7}$$

The second step applies Jensen's inequality to obtain a lower bound of the evidence:

$$\log p(x) >= \mathbb{E}_{z\sim q(z)}\big[\log \frac{p(x,z)}{q(z)}\big] \tag{3.8}$$
$$= \mathbb{E}_{z\sim q(z)}\big[\log p(x|z) + \log p(z) - \log q(z)\big] \tag{3.9}$$
$$= \mathbb{E}_{z\sim q(z)}\big[\log p(x|z)\big] - \mathbb{E}_{z\sim q(z)}\big[\log q(z) - \log p(z)\big] \tag{3.10}$$

The second term in Eq. (3.10) is the KL divergence between $q(z)$ and $p(z)$. Therefore, we write the lower bound in the following form as the final step:

$$\log p(x) >= \mathbb{E}_{z\sim q(z)}\big[\log p(x|z)\big] - \mathrm{KL}(q(z)||p(z)) \tag{3.11}$$
$$= \mathrm{ELBO}(x,q) \tag{3.12}$$

We call this function as the *evidence lower bound*, which is namely a lower bound to the evidence of observation. Please note that the ELBO is a function of the observation x and the approximate posterior $q(\cdot)$ we choose. The value of ELBO does not depend on the actual value of a specific latent variable. Therefore, many papers evaluates the model performance by comparing the ELBO of different choices of $q(\cdot)$.

Derivation from KL Divergence with True Posterior There are multiple ways for deriving the same evidence lower bound. We show another derivation that directly starts from $\mathrm{KL}(q(z)||p(z|x))$, which is the quantity we want to minimize. Details of such a derivation can be found in Blei et al. (2017). By the definition of KL divergence, we know the aforementioned quantity can be expanded as

$$\mathrm{KL}(q(z)||p(z|x)) = \mathbb{E}_{z\sim q(z)}\big[\log q(z) - \log p(z|x)\big] \tag{3.13}$$
$$= \mathbb{E}_{z\sim q(z)}\big[\log q(z) - \log p(x,z)\big] + \log p(x) \tag{3.14}$$

As the log evidence $\log p(x)$ can be viewed as a constant. We can instead minimize the first part of Eq. (3.14), which also means maximizing the following term:

$$\mathbb{E}_{z\sim q(z)}\big[\log p(x,z) - \log q(z)\big] \tag{3.15}$$
$$= \mathbb{E}_{z\sim q(z)}\big[\log p(x|z) + \log p(z) - \log q(z)\big] \tag{3.16}$$
$$= \mathbb{E}_{z\sim q(z)}\big[\log p(x|z)\big] - \mathrm{KL}(q(z)||p(z)). \tag{3.17}$$

Here, we obtained the same function of the evidence lower bound we derived using Jensen's inequality. This derivation helps to realize that by optimizing the ELBO, we are indirectly minimizing the KL divergence between the variational density and the true posterior $\mathrm{KL}(q(z)||p(z|x))$.

Connection with EM algorithm Expectation-Maximization algorithm (Dempster et al., 1977) can be applied to compute the first term in the evidence lower bound (Eq. (3.11)). The difference between EM algorithm and variational inference is that EM usually compute point estimates of the parameters of the distributions. Variational inference estimates the probabilistic distributions of parameters.

Mean-field Variational Family Mean-field variational family is the most common family of the variational density $q(\mathbf{z})$, where $\mathbf{z} = z_1, \ldots, z_K$ and we assume the K latent variables are independent from each other. Such a variational density can be expressed as

$$q(\mathbf{z}) = \prod_{k=1}^{K} q(z_k). \tag{3.18}$$

The mean-field family encourages the latent variables to capture distinct factors of the observation. However, due to the strong assumption, the variational density has a low expressiveness. Some advanced variational inference methods use more complex families for modeling $q(z)$.

Exact Inference In variational inference, we can only compute a lower bound, an approximate value of the true posterior. By carefully design our model, there are methods allow us to perform exact inference. One recent actively investigated approach is normalizing flow (Rezende and Mohamed, 2015). A flow model transforms base densities into data distributions. Normalizing flow rely on invertible functions for density transformation. The capacity of the transformation is strictly limited to make the function invertible. Therefore, in real applications, flow-based models are usually implemented with fairly deep neural networks.

In this thesis, we introduce and develop our proposed models based on the approximate inference approach. In the rest of this chapter, we introduce methods

for learning generative models with neural networks. We start from continuous latent variables, then we cover approaches for learning discrete latent variables.

3.2 Variational Auto-Encoders

In this section, we consider the scenario that we use a more precise variational density $q(z|x)$. The ELBO of our model is then

$$\text{ELBO}(x, q) = \mathbb{E}_{z\sim q(z|x)}\big[p(x|z)\big] - \text{KL}(q(z|x)||p(z)). \tag{3.19}$$

When we maximize this function, we notice that the model is acting similar to a regularized auto-encoder. The first term $\mathbb{E}_{z\sim q(z|x)}\big[p(x|z)\big]$ in the equation is exactly a reconstruction objective, which encodes the information of data x into the latent variables and then reconstruct the original data. However, without any regularization, the hidden states inside a conventional auto-encoder will be a pure compression of the input, thus highly entangled. With the regularization imposed by the KL divergence, the objective forces that latent variables to be somehow predictable and less entangled.

Armotized Inference In classical approaches, we may use EM algorithm to iteratively update the variational densities for each datapoint. The updated estimation is saved in a table. Therefore, the table grows in size when the dataset is large, which is inefficient. With the advance of neural networks, we use deep learning models (it can also be traditional machine learning models) to estimate both the approximate posterior $q(z|x)$ and the likelihood $p(x|z)$. Therefore, we only need to maintain a fixed number of parameters in our models. Such approach is called *armotized inference*, and is more efficient than conventional approaches. In practice, we may call the neural network that computes $q(z|x)$ a *encoder*, and the neural network that computes $p(x|z)$ a *decoder*. The input vectors of the decoder are sampled from the distribution produced by the encoder.

Interpretability As approximate posterior is constrained by the KL divergence. When we use a multivariate normal distribution with diagonal matrix as the variance matrix as the prior, the KL term will discourage the entanglement in the latent variables.

To understand this property, suppose the prior has a zero mean and unit variance for all latent variables, that is $p(z_k) = N(0, 1)$. Then, the approximate posterior has to set $q(z_k|x)$ closing to $N(0, 1)$ in order to reduce the penalty. For example, when modeling face images, $q(z_k|x)$ can produce $N(-2, 1)$ for males, and $N(2, 1)$ for females. The latent variable z_k sampled from $N(2, 1)$ (distribution for female) has a 95% chance falling into the range $[0, 4]$, and it is unlikely to be mistreated as a male variable in the decoder. However, if the variational density attempts to mix other meanings in the same latent variable, then it needs to produce $N(6, 1)$ to avoid being confused as a female variable. Such a variational density will be heavily published by the KL divergence.

3.3 Reparameterization Trick

In this section, we focus on one problem: how to obtain the gradient of the variational density $q(z|x)$. Let $q(z|x)$ and $p(x|z)$ be parameterized by ϕ and θ. As the KL divergence can be computed by an analytical solution when the $q(z|x)$ is an isotropic Gaussian (each variable is independent), the gradient for the KL term can be easily computed. Here, we focus on computing the gradient of the reconstruction objective:

$$\nabla_\phi \mathbb{E}_{z\sim q_\phi(z|x)}\big[\log p_\theta(x|z)\big]. \tag{3.20}$$

The problem that hinders the gradient estimation is the expectation in the equation. Consider the following pseudo code for implementing this equation:

```
1.  sample z from q_φ(z|x)
2.  logp = log(p_θ(x|z))
3.  logp.backward()
```

To compute the expectation, we use Monte Carlo sampling methods. If we run this code, the parameter ϕ cannot obtain a correct gradient as the sampling operation in Line 1 is non-differentiable. Intuitively, to solve this problem, we have to move the computation involving ϕ to somewhere after the sampling operation. In Kingma and Welling (2014), the authors proposed an unbiased and differentiable solution to this problem, which is commonly referred to as the *reparameterization trick* in the paper.

The core idea is simple. Instead of directly sample z from $q_\phi(z|x)$, we compute z with a proxy function as

$$z = g_\phi(\epsilon, x), \tag{3.21}$$

$$\epsilon \sim p(\epsilon). \tag{3.22}$$

Here, ϵ is a standard noise, which is often obtained from a standard Gaussian $N(0, 1)$. Then the function $g_\phi(\cdot)$ shifts and scales this noise according to the parameter. By moving the randomness ϵ into the input, the function $g_\phi(\cdot)$ is fully differentiable. When we model z with isotropic Gaussian, $g_\phi(\cdot)$ has the following implementation:

$$\epsilon \sim N(0, 1), \tag{3.23}$$

$$\mu_x, \sigma_x = f_\phi(x), \tag{3.24}$$

$$z = \mu_x + \sigma_x \epsilon, \tag{3.25}$$

where, $f_\phi(x)$ is a neural network that outputs a mean vector and a variance vector. If we want the variance vector to contain only positive numbers, we can add a *softplus* activation before computing z.

3.4 Learning Discrete Latent Variables

In this section, we discuss the methods of learning the variational auto-encoder when the latent variable z is discrete. In particular, the variational density is either a multivariate Bernoulli distribution or a multivariate categorical distribution. Please notice the difference of them comparing with binomial and multinomial distributions, which can be easily confused.

We first discuss the simpler case that q is a multivariate Bernoulli distribution with K variables. Such a distribution is parameterized by K probabilities $(p_1, \ldots, p_K)$. Usually, the prior $p(z)$ for each variable is a uniform distribution with a probability

of 0.5. In this case, the KL divergence is computed as

$$\begin{aligned}
\mathrm{KL}(q(z|x)||p(z)) &= \sum_{k=1}^{K} \mathrm{KL}(q(z_k|x)||p(z_k)) && (3.26)\\
&= \sum_{k=1}^{K} q(z_k = 1|x) \log \frac{q(z_k = 1|x)}{p(z_k = 1)} + q(z_k = 0|x) \log \frac{q(z_k = 0|x)}{p(z_k = 0)} && (3.27)\\
&= \sum_{k=1}^{K} H(q(z_k|x)) + \log 2 && (3.28)\\
&= H(q) + K. && (3.29)
\end{aligned}$$

In the categorical case, the KL divergence also has a similar form, which is the sum of entropy and a constant number. Therefore, in the rest of this section, we still focus on computing the gradient of the reconstruction objective $\nabla_\phi \mathbb{E}_{z\sim q_\phi(z|x)}\big[\log p_\theta(x|z)\big]$.

3.4.1 Straight-Through Estimator

If we treat the stochastic sampling operation as a function sample, then the gradient of interest can be written in the following form:

$$\nabla_\phi \log p_\theta(x|\mathrm{sample}(q_\phi(z|x))). \tag{3.30}$$

Thus, the problem is indeed how to design such a sampling function and make it differentiable. To be concrete, in Table 3.2, we show one example input and three independent outputs of the sampling function.

Type	Input	Output 1	Output 2	Output 3
stochastic	$[0.3, 0.9, 0.6]^\top$	$[0, 1, 0]^\top$	$[0, 1, 1]^\top$	$[1, 1, 1]^\top$
deterministic	$[0.3, 0.9, 0.6]^\top$	$[0, 1, 1]^\top$	$[0, 1, 1]^\top$	$[0, 1, 1]^\top$

Table 3.2 Input and output examples of a sampling operation of a multivariate Bernoulli distribution.

We can see that the outputs are binary vectors in the multivariate Bernoulli case. When computing the reconstruction objective, we perform stochastic sampling. A close function is deterministic sampling, which always outputs the binary vector with

maximum probability. The deterministic sampling function can be implemented with $sign(\cdot)$. In Bengio (2013), the author proposed to make the deterministic sampling differentiable by simply copy the gradients in the backward phase. This method is referred to as *straight-through estimator* (STE). It can be described using the following equation:

$$f_{\text{STE}}(x) = \begin{cases} \text{sign}(x) & \text{forward} \\ x & \text{backward} \end{cases} \tag{3.31}$$

As shown in the equation, STE pretend to be an identity function in the backward phase. In PyTorch, STE can be easily implemented by the following snippet:

```
def STE(x):
␣␣␣␣return x + x.sign().detach() - x.detach()
```

Following our discussion on the sampling operator, STE indicates that we can just use the gradient of sampled discrete variables to compute the gradients of the distribution parameters. Recent studies adopted STE on various applications including binarizing neural networks (Courbariaux et al., 2016). However, the gradient estimated by this method by no means is the true gradient of the loss function. More recently, Yin et al. (2019) provide a theoretical jusitification and show that the expected coarse gradient of STE correlates with the population gradient.

3.4.2 Semantic Hashing

In Salakhutdinov and Hinton (2009), the authors described a technique to push the output of a sigmoid neuron towards binary. Their method is to compute $\sigma(x + \epsilon)$, where $\sigma(\cdot)$ is the sigmoid activation and ϵ is a Gaussian noise in the original paper. The authors use this technique to compute the hash of documents, attempting to group semantically similar documents together. Kaiser and Bengio (2018) improves the semantic hashing by using saturating sigmoid, which is defined as

$$\sigma'(x) = \max(0, \min(1, 1.2\sigma(x) - 0.1)). \tag{3.32}$$

Here, $\sigma(x)$ computes the original sigmoid function. Comparing to the original version of sigmoid, the saturating counterpart is able to output edge cases (0 or 1) while

the original version cannot. In the paper, the discretization method of the proposed *improved semantic hashing* can be summarized as

$$f_{\text{ISH}}(x) = \begin{cases} 1(x + \epsilon > 0) & 50\% \text{ forward passes} \\ \sigma'(x + \epsilon) & 50\% \text{ forward passes and all backward passes} \end{cases} \tag{3.33}$$

Here, $1(\cdot)$ is an indicator function, and ϵ is a standard Gaussian noise. We can see that the mechanism of improved semantic hashing is similar to STE. In the forward pass, it computes both saturating sigmoid and binarization. The result is randomly chosen between the two. However, in the backward pass, the gradient always flow back through the saturating sigmoid operation.

Similar to the straight-through estimator, the improved semantic hashing method also has the advantage of simplicity. We adopt this method to compute the discretization bottleneck in chapter 6.

3.4.3 Gumbel-Softmax Reparameterization Trick

Let us then consider the scenario of dealing with categorical variables. Similar to the sampling for Bernoulli variables. We show three example outputs sampled from one categorical distribution. Please note that for simplicity, the example does not show multivariate categorical distributions.

Type	Input	Output 1	Output 2	Output 3
stochastic	$[0.3, 0.2, 0.5]^\top$	$[1, 0, 0]^\top$	$[0, 0, 1]^\top$	$[0, 0, 1]^\top$
deterministic	$[0.3, 0.2, 0.5]^\top$	$[0, 0, 1]^\top$	$[0, 0, 1]^\top$	$[0, 0, 1]^\top$

Table 3.3 Input and output examples of a sampling operation of a categorical distribution.

Comparing to the Bernoulli variables, the difference is that each sampled variable is a one-hot vector but a binary vector. In this case, we still can consider to adopt STE for backpropagation. However, when the number of categories increases, as the gradient of the sample only tells the story of one specific category, the training is likely to become unstable.

Following the success of reparameterization trick for continuous variables. Can we derive a reparameterization rule for discrete variables? Denote $P(z)$ as the categorical distribution of interest with K categories. To sample a categorical variable, Gumbel

and Lieblein (1954) described a simple sampling method as

$$z = \text{one\_hot}\Big(\operatorname*{argmax}_{i\in\{1,\dots,K\}}[\log P(z)_i + g_i]\Big), \tag{3.34}$$

$$g_i \sim \text{Gumbel}(0,1). \tag{3.35}$$

where each g_i is an independent Gumbel noise, which can be computed by

$$g_i = -\log(-\log(\epsilon)), \epsilon \sim \text{Uniform}(0,1). \tag{3.36}$$

Here, the discrete variable is randomly sampled by applying `argmax` to the log-probabilities injected Gumbel noises. The `one_hot` function simply convert the category id into a one-hot vector.

Suppose we use the Gumbel rule to implement the sampling function, there is only one problem: the `argmax` operation is non-differentiable. However, it has a differentiable counterpart: softmax function. This suggests us to use softmax to replace the `argmax` function in Gumbel rule. In 2016, Maddison et al. (2016) and Jang et al. (2016) discovered the Gumbel-Softmax distribution independently and published the work in ICLR. It can be considered as a reparameterization of the discrete sampling. The Gumbel-Softmax rule can be described as

$$z = \text{softmax}_\tau\big(\log P(z) + \mathbf{g}\big) \tag{3.37}$$

$$g_i \sim \text{Gumbel}(0,1). \tag{3.38}$$

Note that here $\mathbf{g}$ is a vector containing independently sampled Gumbel noises. The temperature of the softmax is controlled by τ. When the temperature is 0, Gumbel-softmax collapses to the Gumbel-argmax rule. When $\tau > 0$, Gumbel-softmax is differentiable, however, the results are not strictly correct samples of the categorical distribution. In the paper, the authors mentioned a way to start the training with a high temperature, and anneal τ to a small value.

Straight-through Gumbel-Softmax One problem with the Gumbel-softmax samples is that they are not strict one-hot vectors, which means a sampled vector can have positive values in one or more categories. The authors (Jang et al., 2016) propose to solve this problem by applying STE, which forces the discretization in the forward pass, but not in the backward pass. The STE-enhanced Gumbel-Softmax

rule can be described as

$$f_{\text{GumbelST}}(P(z)) = \begin{cases} \text{one\_hot}\big(\underset{i\in\{1,\dots,K\}}{\text{argmax}}(\text{GumbelSoftmax}(P(z))_i)\big) & \text{forward} \\ \text{GumbelSoftmax}(P(z)) & \text{backward} \end{cases} \tag{3.39}$$

Gumbel-Softmax is a theoretical-sound and reliable technique for implementing a discrete VAE. We adopt this method in chapter 5.

3.4.4 Vector-Quantization VAE

Despite the success of various applications applying Gumbel-Softmax, it has a significant practical drawback. Sampled Gumbel-Softmax variables are softmax vectors, each vector K dimensions, identical to the number of categories. This means that we have to scale our neural network according to the number of categories. Suppose the discrete distribution has 65535 categories, then our neural net has to handle a 65535-dimensional giant vector, which is almost impractical.

van den Oord et al. (2017) proposed a different approach to make the discrete sampling operation differentiable, which is coined *Vector-Quatization VAE* (VQ-VAE). Here, we still focus on computing the reconstruction objective:

$$L = \log p(x|\text{sample}(q(z|x))). \tag{3.40}$$

In VQ-VAE, we first prepare K vectors randomly distributed in the vector space, each vector is treated as the embedding of one category. We denote these code vector as $\{e_1, \dots, e_K\}$. Suppose we use an encoder function $z_e(x)$ to map the input x into a vector with the same dimensionality as the code vectors, then the sampling function is computed by

$$\text{sample}(q(z|x)) = e_k, \tag{3.41}$$

$$k = \underset{j\in\{1,\dots,K\}}{\text{argmin}} ||z_e(x) - e_j||_2. \tag{3.42}$$

Intuitive, VQ-VAE is a quantization that finds the code vector closest to the encoder output. In this setting, we then need to solve two problems: how can we optimize the encoder $z_e(x)$ and the code vectors $\{e_1, \dots, e_K\}$? For the first problem, similar

to other discretization methods, VQ-VAE uses STE to create a pseudo gradient for the encoder. It means that regardless of that we compute the decoder based on a sampled vector, the model copies the gradient from the decoder directly to the encoder output. The mechanism can be described as

$$f_{\text{VQVAE}}(q(z|x)) = \begin{cases} \text{sample}(q(z|x)) & \text{forward} \\ z_e(x) & \text{backward} \end{cases} \tag{3.43}$$

The second problem is to optimize the code vectors. If the code vectors are not optimize to fit the encoder outputs, the pseudo gradient will be highly inaccurate. In the paper, the authors propose to modify the loss function to push the encoder output and code vectors closer in the vector space.

$$L = \log p(x|f_{\text{VQVAE}}(q(z|x))) + ||\text{sg}[z_e(x)] - e_k||_2^2 + \beta||z_e(x) - \text{sg}[e_k]||_2^2. \tag{3.44}$$

Here, sg$[\cdot]$ is a stop gradient function, whereas β is a coefficient for the loss term of updating the encoder. In this loss function, the three terms individually update the decoder, code vectors and encoder. As the hidden size in VQ-VAE is not affected by the number of categories. In practice, we can train the model with fairly large discrete latent variables. Please note that the variational density $q(z|x)$ is actually deterministic by computing `argmin` in the sampling function. Although we can easily modify the VQ-VAE equations to make it stochastic, it is not covered in the original paper.

Chapter 4

Latent-Variable Non-autoregressive Neural Machine Translation

4.1 Motivation

Following the story presented by chapter 2, we can see see significant improvements in neural machine translation over years (Bahdanau et al., 2015, Wu et al., 2016, Gehring et al., 2017, Vaswani et al., 2017). Despite impressive improvements in translation accuracy, the autoregressive nature of NMT models have made it difficult to speed up decoding by utilizing parallel model architecture and hardware accelerators. This has sparked interest in *non-autoregressive* NMT models, which predict every target tokens in parallel. In addition to the obvious decoding efficiency, non-autoregressive text generation is appealing as it does not suffer from exposure bias and suboptimal inference.

Inspired by recent work in non-autoregressive NMT using discrete latent variables (Kaiser et al., 2018a) and iterative refinement (Lee et al., 2018), we introduce a sequence of continuous latent variables to capture the uncertainty in the target sentence. We motivate such a latent variable model by conjecturing that it is easier to refine lower-dimensional continuous variables[1] than to refine high-dimensional discrete variables, as done in Lee et al. (2018). Unlike Kaiser et al. (2018a), the posterior and the prior can be jointly trained to maximize the evidence lowerbound of the log-likelihood $\log p(y|x)$.

[1] We use 8-dimensional latent variables in our experiments.

In this work, we propose a deterministic iterative algorithm to refine the approximate posterior over the latent variables and obtain better target predictions. During inference, we first obtain the initial posterior from a prior distribution $p(z|x)$ and the initial guess of the target sentence from the conditional distribution $p(y|x,z)$. We then alternate between updating the approximate posterior and target tokens with the help of an approximate posterior $q(z|x,y)$. We avoid stochasticity at inference time by introducing a *delta posterior* over the latent variables. We empirically find that this iterative algorithm significantly improves the lowerbound and results in better BLEU scores. By refining the latent variables instead of tokens, the length of translation can dynamically adapt throughout this procedure, unlike previous approaches where the target length was fixed throughout the refinement process. In other words, even if the initial length prediction is incorrect, it can be corrected simultaneously with the target tokens.

Our models outperform the autoregressive baseline on ASPEC Ja-En dataset with 8.6x decoding speedup and bring the performance gap down to 2.0 BLEU points on WMT'14 En-De with 12.5x decoding speedup. By decoding multiple latent variables sampled from the prior and rescore using a autoregressive teacher model, the proposed model is able to further narrow the performance gap on WMT'14 En-De task down to 1.0 BLEU point with 6.8x speedup. The contributions of this work can be summarize as follows:

1. We propose a continuous latent-variable non-autoregressive NMT model for faster inference. The model learns identical number of latent vectors as the input tokens. A length transformation mechanism is designed to adapt the number of latent vectors to match the target length.

2. We demonstrate a principle inference method for this kind of model by introducing a deterministic inference algorithm. We show the algorithm converges rapidly in practice and is capable of improving the translation quality by around 2.0 BLEU points.

4.2 Non-Autoregressive NMT

Although autoregressive models achieve high translation quality through recent advances in NMT, the main drawback is that autoregressive modeling forbids the

decoding algorithm to select tokens in multiple positions simultaneously. This results in inefficient use of computational resource and increased translation latency.

In contrast, non-autoregressive NMT models predict target tokens without depending on preceding tokens, depicted by the following objective:

$$\log p(y|x) = \sum_{i=i}^{|y|} \log p(y_i|x). \tag{4.1}$$

As the prediction of each target token y_i now depends only on the source sentence x and its location i in the sequence, the translation process can be easily parallelized. We obtain a target sequence by applying *argmax* to all token probabilities.

The main challenge of non-autoregressive NMT is on capturing dependencies among target tokens. As the probability of each target token does not depend on the surrounding tokens, applying *argmax* at each position i may easily result in an inconsistent sequence, that includes duplicated or missing words. It is thus important for non-autoregressive models to apply techniques to ensure the consistency of generated words.

4.3 Latent-Variable Non-Autoregressive NMT

In this work, we propose a latent-variable non-autoregressive NMT model by introducing a sequence of continuous latent variables to model the uncertainty about the target sentence. These latent variables z are constrained to have the same length as the source sequence, that is, $|z| = |x|$. Instead of directly maximizing the objective function in Eq. (4.1), we maximize a lowerbound to the marginal log-probability $\log p(y|x) = \log \int p(y|z,x)p(z|x)dz$:

$$\begin{aligned}\mathcal{L}(\omega, \phi, \theta) = \mathbb{E}_{z\sim q_\phi}\big[\log p_\theta(y|x,z)\big] \\ - \mathrm{KL}\big[q_\phi(z|x,y)||p_\omega(z|x)\big],\end{aligned} \tag{4.2}$$

where $p_\omega(z|x)$ is the prior, $q_\phi(z|x,y)$ is an approximate posterior and $p_\theta(y|x,z)$ is the decoder. The objective function in Eq. (4.2) is referred to as the evidence lowerbound (ELBO). As shown in the equation, the lowerbound is parameterized by three sets of parameters: ω, ϕ and θ.

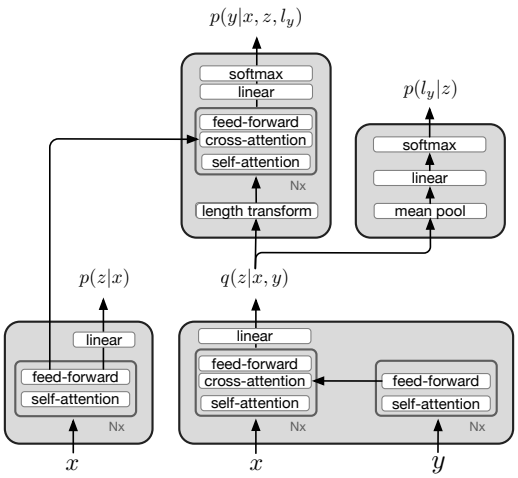


Fig. 4.1 Architecture of the proposed non-autogressive model. The model is composed of four components: prior $p(z|x)$, approximate posterior $q(z|x,y)$, length predictor $p(l_y|z)$ and decoder $p(y|x,z)$. These components are trained end-to-end to maximize the evidence lowerbound.

Both the prior p_ω and the approximate posterior q_ϕ are modeled as spherical Gaussian distributions. The model can be trained end-to-end with the reparameterization trick (Kingma and Welling, 2014).

4.3.1 A Modified Objective Function with Length Prediction

During training, we want the model to maximize the lowerbound in Eq. (4.2). However, to generate a translation, the target length l_y has to be predicted first. We let the latent variables model the target length by parameterizing the decoder as:

$$\begin{aligned} p_\theta(y|x,z) &= \sum_l p_\theta(y,l|x,z) \\ &= p_\theta(y,l_y|x,z) \\ &= p_\theta(y|x,z,l_y)p_\theta(l_y|z). \end{aligned} \tag{4.3}$$

Here l_y denotes the length of y. The second step is valid as the probability $p_\theta(y, l \neq l_y|x,z)$ is always zero. Plugging in Eq. (4.3), with the independent assumption on

both latent variables and target tokens, the objective has the following form:

$$\mathbb{E}_{z\sim q_\phi}\Big[\sum_{i=1}^{|y|}\log p_\theta(y_i|x,z,l_y)+\log p_\theta(l_y|z)\Big] \\ -\sum_{k=1}^{|x|}\mathrm{KL}\big[q_\phi(z_k|x,y)||p_\omega(z_k|x)\big]. \tag{4.4}$$

4.3.2 Model Architecture

As evident from in Eq. (4.4), there are four parameterized components in our model: the prior $p_\omega(z|x)$, approximate posterior $q_\phi(z|x,y)$, decoder $p_\theta(y|x,z,l_y)$ and length predictor $p_\theta(l_y|z)$. The architecture of the proposed non-autoregressive model is depicted in Fig. 5.2, which reuses modules in Transformer (Vaswani et al., 2017) to compute the aforementioned distributions.

Main Components To compute the prior $p_\omega(z|x)$, we use a multi-layer self-attention encoder which has the same structure as the Transformer encoder. In each layer, a feed-forward computation is applied after the self-attention. To obtain the probability, we apply a linear transformation to reduce the dimensionality and compute the mean and variance vectors.

For the approximate posterior $q_\phi(z|x,y)$, as it is a function of the source x and the target y, we first encode y with a self-attention encoder. Then, the resulting vectors are fed into an attention-based decoder initialized by x embeddings. Its architecture is similar to the Transformer decoder except that no causal mask is used. Similar to the prior, we apply a linear layer to obtain the mean and variance vectors.

To backpropagate the loss signal of the decoder to q_ϕ, we apply the reparameterization trick to sample z from q_ϕ with $g(\epsilon,q)=\mu_q+\sigma_q*\epsilon$. Here, $\epsilon\sim\mathcal{N}(0,1)$ is Gaussian noise.

The decoder computes the probability $p_\theta(y|x,z,l_y)$ of outputting target tokens y given the latent variables sampled from $q_\phi(z|x,y)$. The computational graph of the decoder is also similar to the Transformer decoder without using causal mask. To combine the information from the source tokens, we reuse the encoder vector representation created when computing the prior.

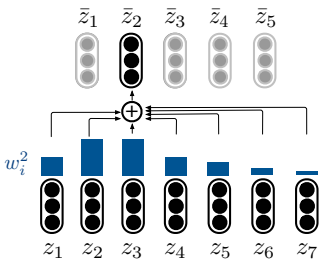


Fig. 4.2 Illustration of the length transformation mechanism.

Length Prediction and Transformation Given a latent variable z sampled from the approximate posterior q_ϕ, we train a length prediction model $p_\theta(l_y|z)$. We train the model to predict the length difference between $|y|$ and $|x|$. In our implementation, $p_\theta(l_y|z)$ is modeled as a categorical distribution that covers the length difference in the range $[-50, 50]$. The prediction is produced by applying softmax after a linear transformation.

As the latent variable $z \sim q_\phi(z|x, y)$ has the length $|x|$, we need to transform the latent variables into l_y vectors for the decoder to predict target tokens. We use a monotonic location-based attention for this purpose, which is illustrated in Fig. 4.2. Let the resulting vectors of length transformation be $\bar{z}_1, ..., \bar{z}_{l_y}$. we produce each vector with

$$\bar{z}_j = \sum_{k=1}^{|x|} w_k^j z_k, \tag{4.5}$$

$$w_k^j = \frac{\exp(a_k^j)}{\sum_{k'=1}^{|x|} \exp(a_{k'}^j)}, \tag{4.6}$$

$$a_k^j = -\frac{1}{2\sigma^2}(k - \frac{|x|}{l_y} j)^2, \tag{4.7}$$

where each transformed vector is a weighted sum of the latent variables. The weight is computed with a softmax over distance-based logits. We give higher weights to the latent variables close to the location $\frac{|x|}{l_y} j$. The scale σ is the only trainable parameter in this monotonic attention mechanism.

4.3.3 Training

If we train a model with the objective function in Eq. (4.4), the KL divergence often drops to zero from the beginning. This yields a degenerate model that does not use

the latent variables at all. This is a well-known issue in variational inference called posterior collapse (Bowman et al., 2015, Dieng et al., 2018, Razavi et al., 2019). We use two techniques to address this issue. Similarly to Kingma et al. (2016), we give a budget to the KL term as

$$\sum_{k=1}^{|x|} \max(b, \mathrm{KL}\big[q_\phi(z_k|x,y)||p_\omega(z_k|x)\big]), \tag{4.8}$$

where b is the budget of KL divergence for each latent variable. Once the KL value drops below b, it will not be minimized anymore, thereby letting the optimizer focus on the reconstruction term in the original objective function. As b is a critical hyperparameter, it is time-consuming to search for a good budget value. Here, we use the following annealing schedule to gradually lower the budget:

$$b = \begin{cases} 1, & \text{if } s < M/2 \\ \frac{(M-s)}{M/2}, & \text{otherwise} \end{cases} \tag{4.9}$$

s is the current step in training, and M is the maximum step. In the first half of the training, the budget b remains 1. In the second half of the training, we anneal b until it reaches 0.

Similarly to previous work on non-autoregressive NMT, we apply sequence-level knowledge distillation (Kim and Rush, 2016) where we use the output from an autoregressive model as target for our non-autoregressive model.

4.4 Inference with a Delta Posterior

Once the training has converged, we use an inference algorithm to find a translation y that maximizes the lowerbound in Eq. (4.2):

$$\begin{aligned} \operatorname*{argmax}_y \mathbb{E}_{z\sim q_\phi}\big[\log p_\theta(y|x,z)\big] \\ - \mathrm{KL}\big[q_\phi(z|x,y)||p_\omega(z|x)\big] \end{aligned}$$

It is intractable to solve this problem exactly due to the intractability of computing the first expectation. We avoid this issue in the training time by reparametrization-based Monte Carlo approximation. However, it is desirable to avoid stochasticity at

inference time where our goal is to present a single most likely target sentence given a source sentence.

We tackle this problem by introducing a proxy distribution $r(z)$ defined as

$$r(z) = \begin{cases} 1, & \text{if } z = \mu \\ 0, & \text{otherwise} \end{cases}$$

This is a Dirac measure, and we call it a *delta posterior* in our work. We set this delta posterior to minimize the KL divergence against the approximate posterior q_ϕ, which is equivalent to

$$\nabla_\mu \log q_\phi(\mu|x, y) = 0 \Leftrightarrow \mu = \mathbb{E}_{q_\phi}[z]. \tag{4.10}$$

We then use this proxy instead of the original approximate posterior to obtain a *deterministic lowerbound*:

$$\hat{\mathcal{L}}(\omega, \theta, \mu) = \log p_\theta(y|x, z = \mu) - \log p_\omega(\mu|x).$$

As the second term is constant with respect to y, maximizing this lowerbound with respect to y reduces to

$$\operatorname*{argmax}_y \log p_\theta(y|x, z = \mu), \tag{4.11}$$

which can be approximately solved by beam search when p_θ is an autoregressive sequence model. If p_θ factorizes over the sequence y, as in our non-autoregressive model, we can solve it exactly by

$$\hat{y}_i = \operatorname*{argmax}_{y_i} \log p_\theta(y_i|x, z = \mu).$$

With every estimation of y, the approximate posterior q changes. We thus alternate between fitting the delta posterior in Eq. (4.10) and finding the most likely sequence y in Eq. (4.11).

We initialize the delta posterior r using the prior distribution:

$$\mu = \mathbb{E}_{p_\omega(z|x)}[z].$$

Algorithm 1 Deterministic Iterative Inference

Inputs:
x : source sentence
T : maximum step
$\mu_0 = \mathbb{E}_{p_\omega(z|x)}[z]$
$y_0 = \text{argmax}_y \log p_\theta(y|x, z = \mu_0)$
for $t \leftarrow 1$ to T **do**
$\mu_t = \mathbb{E}_{q_\phi(z|x,y_{t-1})}[z]$
$y_t = \text{argmax}_y \log p_\theta(y|x, z = \mu_t)$
if $y_t = y_{t-1}$ **then**
break
output y_t

With this initialization, the proposed inference algorithm is fully deterministic. The complete inference algorithm for obtaining the final translation is shown in Algorithm 1.

4.5 Related Work

This work is inspired by a recent line of work in non-autoregressive NMT. Gu et al. (2018b) first proposed a non-autoregressive framework by modeling word alignment as a latent variable, which has since then been improved by Wang et al. (2019). Lee et al. (2018) proposed a deterministic iterative refinement algorithm where a decoder is trained to refine the hypotheses. Our approach is most related to Kaiser et al. (2018a), Roy et al. (2018). In both works, a discrete autoencoder is first trained on the target sentence, then an autoregressive prior is trained to predict the discrete latent variables given the source sentence. Our work is different from them in three ways: (1) we use continuous latent variables and train the approximate posterior $q(z|x, y)$ and the prior $p(z|x)$ jointly; (2) we use a non-autoregressive prior; and (3) we propose a novel iterative inference procedure in the latent space.

Concurrently to our work, Ghazvininejad et al. (2019) proposed to translate with a masked-prediction language model by iterative replacing tokens with low confidence. Gu et al. (2019), Stern et al. (2019), Welleck et al. (2019) proposed insertion-based NMT models that insert words to the translations with a specific strategy. Unlike these works, our approach performs refinements in the low-dimensional latent space, rather than in the high-dimensional discrete space.

	ASPEC Ja-En		WMT'14 En-De	
	BLEU(%)	speedup	BLEU(%)	speedup
Base Transformer, beam size=3	27.1	1x	26.1	1x
Base Transformer, beam size=1	24.6	1.1x	25.6	1.3x
Latent-Variable NAR Model	13.3	17.0x	11.8	22.2x
+ knowledge distillation	25.2	17.0x	22.2	22.2x
+ deterministic inference	27.5	8.6x	24.1	12.5x
+ latent search	28.3	4.8x	25.1	6.8x

Table 4.1 Comparison of the proposed non-autoregressive (NAR) models with the autoregressive baselines. Our implementation of the Base Transformer is 1.0 BLEU point lower than the original paper (Vaswani et al., 2017) on WMT'14 dataset.

Similarly to our latent-variable model, Zhang et al. (2016) proposed a variational NMT, and Shah and Barber (2018) models the joint distribution of source and target. Both of them use autoregressive models. Shah and Barber (2018) designed an EM-like algorithm similar to Markov sampling (Arulkumaran et al., 2017). In contrast, we propose a deterministic algorithm to remove any non-determinism during inference.

4.6 Experimental Settings

Data and preprocessing We evaluate our model on two machine translation datasets: ASPEC Ja-En (Nakazawa et al., 2016) and WMT'14 En-De (Bojar et al., 2014). The ASPEC dataset contains 3M sentence pairs, and the WMT'14 dataset contains 4.5M senence pairs.

To preprocess the ASPEC dataset, we use Moses toolkit (Koehn et al., 2007) to tokenize the English sentences, and Kytea (Neubig et al., 2011) for Japanese sentences. We further apply byte-pair encoding (Sennrich et al., 2016a) to segment the training sentences into subwords. The resulting vocabulary has 40K unique tokens on each side of the language pair. To preprocess the WMT'14 dataset, we apply sentencepiece (Kudo and Richardson, 2018) to both languages to segment the corpus into subwords and build a joint vocabulary. The final vocabulary size is 32K for each language.

Learning To train the proposed non-autoregressive models, we adapt the same learning rate annealing schedule as the Base Transformer. Final model parameters is selected based on the validation ELBO value.

The only new hyperparameter in the proposed model is the dimension of each latent variable. If each latent is a high-dimension vector, although it has a higher capacity, the KL divergence in Eq. (4.2) becomes difficult to minimize. In practice, we found that latent dimensionality values between 4 and 32 result in similar performance. However, when the dimensionality is significantly higher or lower, we see a performance drop. In all experiments, we set the latent dimensionionality to 8. We use a hidden size of 512 and feedforward filter size of 2048 for all models in our experiments. We use 6 transformer layers for the prior and the decoder, and 3 transformer layers for the approximate posterior.

Evaluation We evaluate the *tokenized BLEU* for ASPEC Ja-En datset. For WMT'14 En-De datset, we use SacreBLEU (Post, 2018a) to evaluate the translation results. We follow Lee et al. (2018) to remove repetitions from the translation results before evaluating BLEU scores. To measure the decoding speed, the main computational device we use is Nvidia V100 GPU. It has the 16GB on-board memory with 5120 CUDA cores. The GPU is operating at a frequency of 1246 MHz.

Latent Search To further exploit the parallel computation ability of GPUs, we sample multiple initial latent variables from the prior $p_\omega(z|x)$. Then we perform the deterministic inference on each latent variable to obtain a list of candidate translations. However, we can not afford to evalaute each candidate using Eq. (4.4), which requires importance sampling on q_ϕ. Instead, we use the autoregressive baseline model to score all the candidates, and pick the candidate with the highest log probability. Following Parmar et al. (2018), we reduce the temperature by a factor of 0.5 when sampling latent variables, resulting in better translation quality. To avoid the stochasticity, we fix the random seed during sampling.

4.7 Result and Analysis

4.7.1 Quantitative Analysis

Our quantitative results on both datasets are presented in Table 4.1. The baseline model in our experiments is a base Transformer. Our implementation of the autoregressive baseline is 1.0 BLEU points lower than the original paper (Vaswani et al., 2017) on WMT'14 En-De datase. We measure the latency of decoding each sentence on a single NVIDIA V100 GPU for all models, which is averaged over all test samples.

As shown in Table 4.1, without knowledge distillation, we observe a significant gap in translation quality compared to the autoregressive baseline. This observation is in line with previous ones on non-autoregressive NMT (Gu et al., 2018b, Lee et al., 2018, Wang et al., 2019). The gap is significantly reduced by using knowledge distillation, as translation targets provided by the autoregressive model are easier to predict.

With the proposed deterministic inference algorithm, we significantly improve translation quality by 2.3 BLEU points on ASPEC Ja-En dataset and 1.9 BLEU points on WMT'14 En-De dataset. Here, we only run the algorithm for one step. We observe gain on ELBO by running more iterative steps, which is however not reflected by the BLEU scores. As a result, we outperform the autoregressive baseline on ASPEC dataset with a speedup of 8.6x. For WMT'14 dataset, although the proposed model reaches a speedup of 12.5x, the gap with the autoregressive baseline still remains, at 2.0 BLEU points. We conjecture that WMT'14 En-De is more difficult for our non-autoregressive model as it contains a high degree of noise (Ott et al., 2018).

By searching over multiple initial latent variables and rescoring with the teacher Transformer model, we observe an increase in performance by $0.7 \sim 1.0$ BLEU scores at the cost of slower translation speed. In our experiments, we sample 50 candidate latent varaibles and decode them in parallel. The slowdown is mainly caused by rescoring. With the help of rescoring, our final model further narrows the performance gap with the autoregressive baseline to 1.0 BLEU with 6.8x speedup on WMT'14 En-De task.

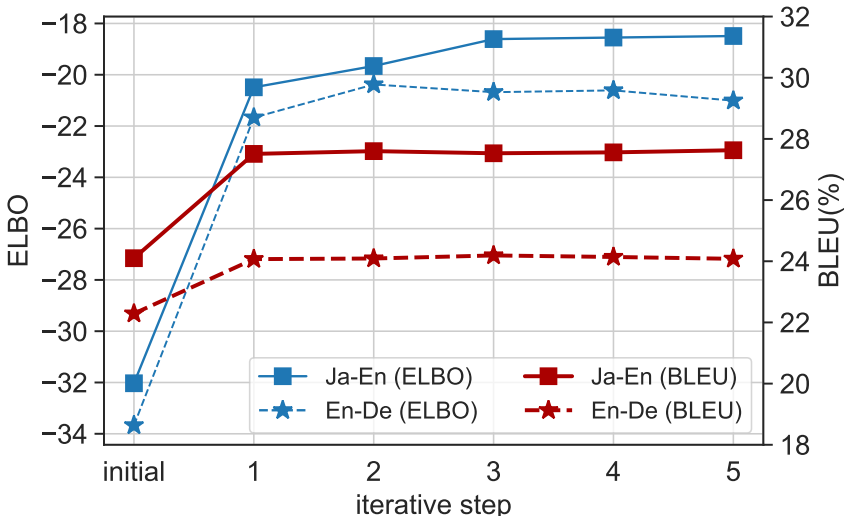


Fig. 4.3 ELBO and BLEU scores measured with the target predictions obtained at each inference step for ASPEC Ja-En and WMT'14 En-De datasets.

4.7.2 Non-autoregressive NMT Models

In Table 4.2, we list the results on WMT'14 En-De by existing non-autoregressive NMT approaches. All the models use Transformer as their autoregressive baselines. In comparison, our proposed model suffers a drop of 1.0 BLEU points over the baseline, which is a relatively small gap among the existing models. Thanks of the rapid convergence of the proposed deterministic inference algorithm, our model achieves a higher speed-up compared to other refinement-based models and provides a better speed-accuracy tradeoff.

Concurrently to our work, the mask-prediction language model (Ghazvininejad et al., 2019) was found to reduce the performance gap down to 0.9 BLEU on WMT'14 En-De while still maintaining a reasonable speed-up. The main difference is that we update a delta posterior over latent variables instead of target tokens. Both Ghazvininejad et al. (2019) and Wang et al. (2019) with autoregressive rescoring decode multiple candidates in batch and pick one final translation from them. As our proposal is orthogonal to using BERT-style training (Devlin et al., 2018), it is an interesting future direction to investigate their combination.

	BLEU(%)	SPD
Transformer (Vaswani et al., 2017)	27.1	-
Baseline (Gu et al., 2018b)	23.4	1x
NAT (+FT +NPD S=100)	19.1 (-4.3)	2.3x
Baseline (Lee et al., 2018)	24.5	1x
Adaptive NAR Model	21.5 (-3.0)	1.9x
Baseline (Kaiser et al., 2018a)	23.5	1x
LT, Improved Semhash	19.8 (-3.7)	3.8x
Baseline (Wang et al., 2019)	27.3	1x
NAT-REG, no rescoring	20.6 (-6.7)	27.6x*
NAT-REG, autoregressive rescoring	24.6 (-2.7)	15.1x*
BL (Ghazvininejad et al., 2019)	27.8	1x
CMLM with 4 iterations	26.0 (-1.8)	-
CMLM with 10 iterations	26.9 (-0.9)	2∼3x
Baseline (Ours)	26.1	1x
NAR with deterministic Inference	24.1 (-2.0)	12.5x
+ latent search	25.1 (-1.0)	6.8x

Table 4.2 A comparison of non-autoregressive NMT models on WMT'14 En-De dataset in BLEU(%) and decoding speed-up. $\star$ measured on IWSLT'14 DE-EN dataset.

4.7.3 Analysis of Deterministic Inference

Convergences of ELBO and BLEU In this section, we empirically show that the proposed deterministic iterative inference improves the ELBO in Eq. (4.2). As the ELBO is a function of x and y, we measure the ELBO value with the new target prediction after each iteration during inference. For each instance, we sample 20 latent variables to compute the expectation in Eq. (4.2). The ELBO value is further averaged over data samples.

In Fig. 4.3, we show the ELBO value and the resulting BLEU scores for both datasets. In the initial step, the delta posterior is initialized with the prior distribution $p_\omega(z|x)$. We see that the ELBO value increases rapidly by performing the iterative inference, which means a higher lowerbound to $\log p(y|x)$. The improvement is highly correlated with increasing BLEU scores. For around 80% of the data samples, the

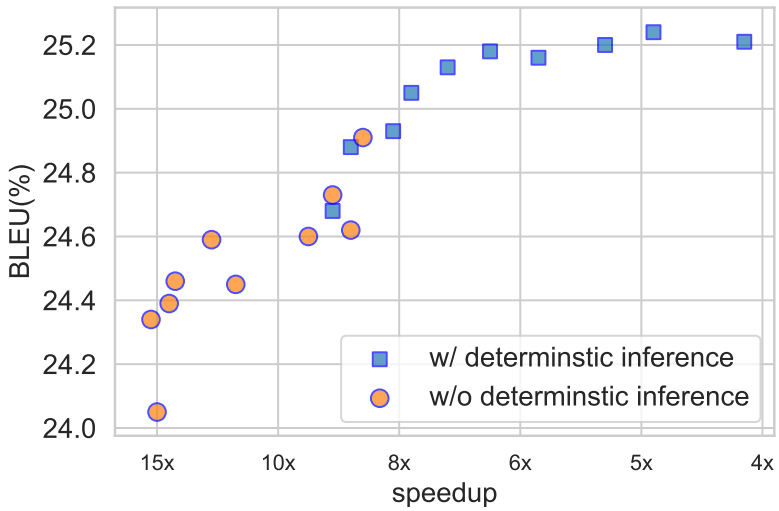


Fig. 4.4 Trade-off between BLEU scores and speedup on WMT'14 En-De task by varying the number of candidates computed in parallel from 10 to 100.

algorithm converges within three steps. We observe the BLEU scores peaked after only one iterative step.

Trade-off between Quality and Speed In Fig. 4.4, we show the trade-off between translation quality and the speed gain on WMT'14 En-De task when considering multiple candidates latent variables in parallel. We vary the number of candidates from 10 to 100, and report BLEU scores and relative speed gains in the scatter plot. The results are divided into two groups. The first group of experiments search over multiple latent variables and rescore with the teacher Transformer. The second group applies the proposed deterministic inference before rescoring.

We observe that the proposed deterministic inference constantly improves the translation quality in all settings. The BLEU score peaks at 25.2 after increasing the number of candidates to a large value. As GPUs are good at processing massive computations in parallel, we can see that the translation speed only degrades by a small magnitude when decoding less than 50 candidates.

4.7.4 Decoding Speed Related to Sentence Length

In Fig. 4.5 and Fig. 4.6, we the relation between the decoding speed and the input sentence length. Fig. 4.5 compares the proposed model and autoregressive baseline on Nvidia V100 GPU, whereas Fig. 4.6 compares the decoding speed of proposed model on different devices.

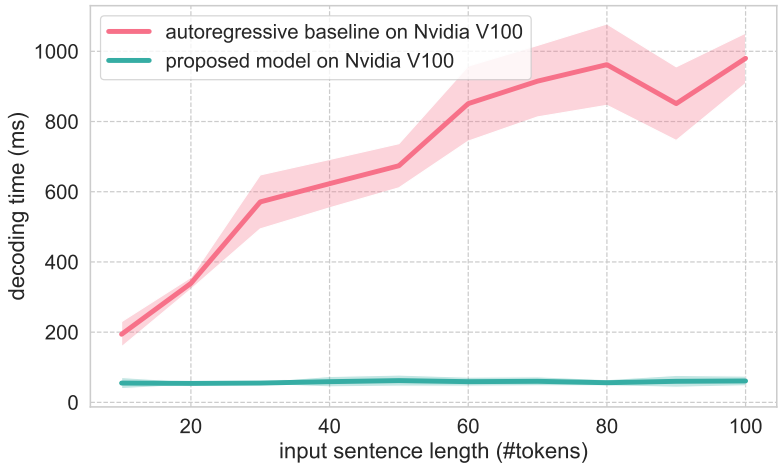


Fig. 4.5 Relation between decoding speed and input sentence length of autoregressive baseline and proposed model, running on Nvidia V100 GPU.

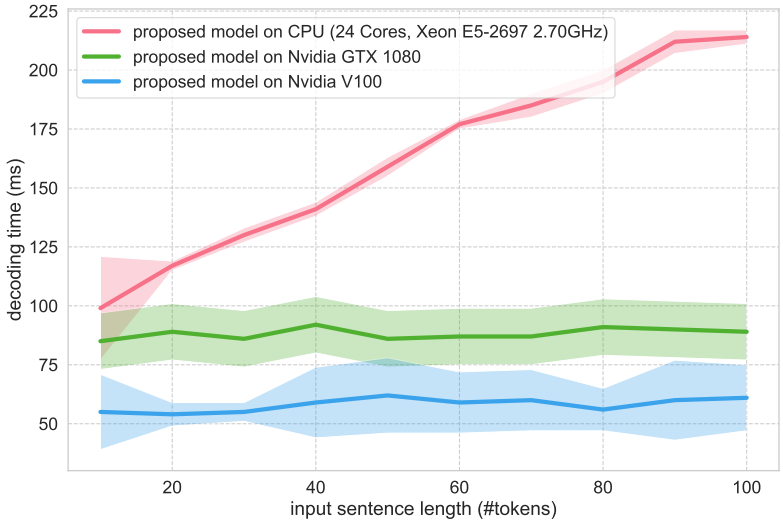


Fig. 4.6 Relation between decoding speed and input sentence length of proposed model, running on multiple devices.

We observe that the decoding time is constant for our proposed model when running on GPU. In contrast, autoregressive model slows down linearly when translating

longer sequences. Interestingly, the decoding speed of proposed model on CPU is even faster than the autoregressive model on GPU. The results demonstrate that the non-autoregressive models can be well parallelized given enough computational capacity.

Example 1: Sequence modified without changing length

Source	hyouki gensuiryou hyoujun no kakuritsu wo kokoromita.
Reference	the establishment of an optical fiber attenuation ...
Initial Guess	an attempt was made establish establish damping ...
After Inference	an attempt was to establish the damping attenuation ...

Example 2: One word removed from the sequence

Source	...“sen bouchou keisu no toriatsukai” nitsuite nobeta.
Reference	... handling of linear expansion coefficient .
Initial Guess	... “ handling of of linear expansion coefficient ” ...
After Inference	... “ handling of linear expansion coefficient ” are ...

Example 3: Four words added to the sequence

Source	... maikuro manipyureshon heto hatten shite kite ori ...
Reference	... with wide application fields so that it has been ...
Initial Guess	... micro micro manipulation and ...
After Inference	... and micro manipulation , and it has been developed ...

Table 4.3 Ja-En sample translation with the proposed iterative inference algorithm. In the first example, the initial guess is refined without a change in length. In the last two examples, the iterative inference algorithm changes the target length along with its content. This is more pronounced in the last example, where a whole clause is inserted during refinement.

4.7.5 Qualitative Analysis

We present some example translations to demonstrate the effect of the proposed iterative inference in Table 5.5. In the first example, the length of the target sequence does not change but only the tokens are replaced over the refinement iterations. The second and third examples show that the algorithm removes or inserts words to the sequence during the iterative inference by adaptively changing the target

length. Such a significant modification to the predicted sequence mostly happens when translating long sentences.

For some test examples, however, we still find duplicated words in the final translation after applying the proposed deterministic inference. For them, we notice that the quality of the initial guess of translation is considerably worse than average, which typically contains multiple duplicated words. As the decoder $p_\theta(y|x,z)$ is trained to reconstruct the y sequence given to the approximator q_ϕ, it is not expected to drastically modify the target prediction. Thus, a high-quality initial guess is crucial for obtaining good translations.

4.8 Conclusion

Our work presents the first approach to use a continuous latent-variable model for non-autoregressive Neural Machine Translation. The key idea is to introduce a sequence of latent variables to capture the uncertainly in the target sentence. The number of latent vectors is always identical to the number of input tokens. A length transformation mechanism is then applied to adapt the latent vectors to match the target length. We train the proposed model by maximizing the lowerbound of the log-probability $\log p(y|x)$.

We then introduce a deterministic inference algorithm that uses a *delta posterior* over the latent variables. The algorithm alternates between updating the delta posterior and the target tokens. Our experiments show that the algorithm is able to improve the evidence lowerbound of predicted target sequence rapidly. In our experiments, the BLEU scores converge in only one iteration. Despite its effectiveness, the algorithm can be easily implemented.

Our non-autoregressive NMT model closes the performance gap with autoregressive baseline on ASPEC Ja-En task with a 8.6x speedup, and reduces the gap on WMT'14 En-De task down to 2.0 BLEU point with a 12.5x speedup. By decoding multiple latent variables sampled from the prior, our model outperforms the baseline by 1.2 BLEU points on En-Ja task with 4.8x speedup, brings down the gap on En-De task down to 1.0 BLEU with a speedup of 6.8x.

When decoding multiple latent variables, a teacher model is essential as the latent-variable model framework does not provide a way to correctly evaluate candidate translations. The teacher model typically takes 15ms to compute. Future work that

enables rescoring without the help of a external model may further improve the decoding speed.

Chapter 5

Discrete Representation Learning for Model Compression

5.1 Motivation

Word embeddings play an important role in neural-based natural language processing (NLP) models. Neural word embeddings encapsulate the linguistic information of words in continuous vectors. However, as each word is assigned an independent embedding vector, the number of parameters in the embedding matrix can be huge. For example, when each embedding has 500 dimensions, the network has to hold 100M embedding parameters to represent 200K words. In practice, for a simple sentiment analysis model, the word embedding parameters account for 98.8% of the total parameters.

As only a small portion of the word embeddings is selected in the forward pass, the giant embedding matrix usually does not cause a speed issue. However, the massive number of parameters in the neural network results in a large storage or memory footprint. When other components of the neural network are also large, the model may fail to fit into GPU memory during training. Moreover, as the demand for low-latency neural computation for mobile platforms increases, some neural-based models are expected to run on mobile devices. Thus, it is becoming more important to compress the size of NLP models for deployment to devices with limited memory or storage capacity.

5.2 Representing Words with Discrete Codes

In this study, we attempt to reduce the number of parameters used in word embeddings without hurting the model performance. Neural networks are known for the significant redundancy in the connections (Denil et al., 2013). In this work, we further hypothesize that learning independent embeddings causes more redundancy in the embedding vectors, as the inter-similarity among words is ignored. Some words are very similar regarding the semantics. For example, "dog" and "dogs" have almost the same meaning, except one is plural. To efficiently represent these two words, it is desirable to share information between the two embeddings. However, a small portion in both vectors still has to be trained independently to capture the syntactic difference.

Following the intuition of creating partially shared embeddings, instead of assigning each word a unique ID, we represent each word w with a code $C_w = (C_w^1, C_w^2, ..., C_w^M)$. Each component C_w^i is an integer number in $[1, K]$. Ideally, similar words should have similar codes. For example, we may desire $C_{\text{dog}} = (3, 2, 4, 1)$ and $C_{\text{dogs}} = (3, 2, 4, 2)$. Once we have obtained such compact codes for all words in the vocabulary, we use embedding vectors to represent the codes rather than the unique words. More specifically, we create M codebooks $E_1, E_2, ..., E_M$, each containing K codeword vectors. The embedding of a word is computed by summing up the codewords corresponding to all the components in the code as

$$E(C_w) = \sum_{i=1}^{M} E_i(C_w^i), \tag{5.1}$$

$$\tag{5.2}$$

where $E_i(C_w^i)$ is the C_w^i-th codeword in the codebook E_i. In this way, the number of vectors in the embedding matrix will be $M \times K$, which is usually much smaller than the vocabulary size. Fig. 5.1 gives an intuitive comparison between the compositional approach and the conventional approach (assigning unique IDs). The codes of all the words can be stored in an integer matrix, denoted by C. Thus, the storage footprint of the embedding layer now depends on the total size of the combined codebook E and the code matrix C.

Although the number of embedding vectors can be greatly reduced by using such coding approach, we want to prevent any serious degradation in performance

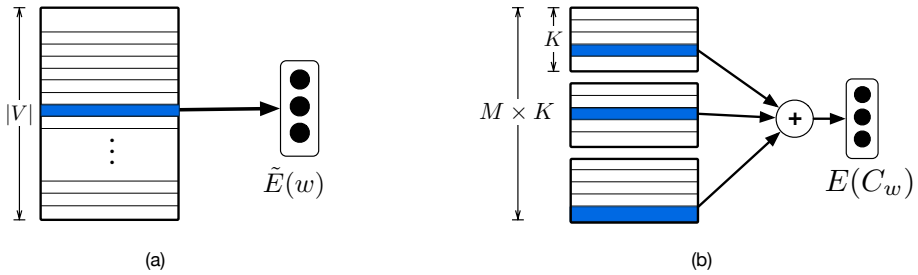


Fig. 5.1 Comparison of embedding computations between the conventional approach (a) and compositional coding approach (b) for constructing embedding vectors

compared to the models using normal embeddings. In other words, given a set of baseline word embeddings $\tilde{E}(w)$, we wish to find a set of codes $\hat{C}$ and combined codebook $\hat{E}$ that can produce the embeddings with the same effectiveness as $\tilde{E}(w)$. A safe and straight-forward way is to minimize the squared distance between the baseline embeddings and the composed embeddings as

$$(\hat{C}, \hat{E}) = \underset{C,E}{\operatorname{argmin}} \frac{1}{|V|} \sum_{w \in V} ||E(C_w) - \tilde{E}(w)||^2 \tag{5.3}$$

$$= \underset{C,E}{\operatorname{argmin}} \frac{1}{|V|} \sum_{w \in V} || \sum_{i=1}^{M} E_i(C_w^i) - \tilde{E}(w)||^2 \,, \tag{5.4}$$

where $|V|$ is the vocabulary size. The baseline embeddings can be a set of pre-trained vectors such as word2vec (Mikolov et al., 2013) or GloVe (Pennington et al., 2014a) embeddings.

In Eq. (5.3), the baseline embedding matrix $\tilde{E}$ is approximated by M codewords selected from M codebooks. The selection of codewords is controlled by the code C_w. Such problem of learning compact codes with multiple codebooks is formalized and discussed in the research field of compression-based source coding, known as product quantization (Jégou et al., 2011) and additive quantization (Babenko and Lempitsky, 2014, Martinez et al., 2016). Previous works learn compositional codes so as to enable an efficient similarity search of vectors. In this work, we utilize such codes for a different purpose, that is, constructing word embeddings with drastically fewer parameters.

Due to the discreteness in the hash codes, it is usually difficult to directly optimize the objective function in Eq. (5.3). In this paper, we propose a simple and straight-forward method to learn the codes in an end-to-end neural network. We utilize the

Gumbel-softmax trick (Maddison et al., 2016, Jang et al., 2016) to find the best discrete codes that minimize the loss. Besides the simplicity, this approach also allows one to use any arbitrary differentiable loss function, such as cosine similarity.

The contribution of this work can be summarized as follows:

- We propose to utilize the compositional coding approach for constructing the word embeddings with significantly fewer parameters. In the experiments, we show that over 98% of the embedding parameters can be eliminated in sentiment analysis task without affecting performance. In machine translation tasks, the loss-free compression rate reaches $94\% \sim 99\%$.
- We propose a direct learning approach for the codes in an end-to-end neural network, with a Gumbel-softmax layer to encourage the discreteness.
- The neural network for learning codes will be packaged into a tool[1]. With the learned codes and basis vectors, the computation graph for composing embeddings is fairly easy to implement, and does not require modifications to other parts in the neural network.

5.3 Related Work

Existing works for compressing neural networks include low-precision computation (Vanhoucke et al., 2011, Hwang and Sung, 2014, Courbariaux et al., 2014, Anwar et al., 2015), quantization (Chen et al., 2015, Han et al., 2016, Zhou et al., 2017), network pruning (LeCun et al., 1989, Hassibi and Stork, 1992, Han et al., 2015, Wen et al., 2016) and knowledge distillation (Hinton et al., 2015). Network quantization such as HashedNet (Chen et al., 2015) forces the weight matrix to have few real weights, with a hash function to determine the weight assignment. To capture the non-uniform nature of the networks, DeepCompression (Han et al., 2016) groups weight values into clusters based on pre-trained weight matrices. The weight assignment for each value is stored in the form of Huffman codes. However, as the embedding matrix is tremendously big, the number of hash codes a model need to maintain is still large even with Huffman coding.

Network pruning works in a different way that makes a network sparse. Iterative pruning (Han et al., 2015) prunes a weight value if its absolute value is smaller than

[1]The code can be found in https://github.com/zomux/neuralcompressor

a threshold. The remaining network weights are retrained after pruning. Some recent works (See et al., 2016, Zhang et al., 2017) also apply iterative pruning to prune 80% of the connections for neural machine translation models. In this paper, we compare the proposed method with iterative pruning.

The problem of learning compact codes considered in this paper is closely related to learning to hash (Weiss et al., 2008, Kulis and Darrell, 2009, Liu et al., 2012), which aims to learn the hash codes for vectors to facilitate the approximate nearest neighbor search. Initiated by product quantization (Jégou et al., 2011), subsequent works such as additive quantization (Babenko and Lempitsky, 2014) explore the use of multiple codebooks for source coding, resulting in compositional codes. We also adopt the coding scheme of additive quantization for its storage efficiency. Previous works mainly focus on performing efficient similarity search of image descriptors. In this work, we put more focus on reducing the codebook sizes and learning efficient codes to avoid performance loss. Joulin et al. (2016) utilizes an improved version of product quantization to compress text classification models. However, to match the baseline performance, much longer hash codes are required by product quantization. This will be detailed in Section 5.6.2. Concurrent to this work, Chen et al. (2017) also explores the similar idea and obtained positive results in language modeling tasks. Also, Raunak (2017) tried to reduce dimension of embeddings using PCA.

To learn the codebooks and code assignment, additive quantization alternatively optimizes the codebooks and the discrete codes. The learning of code assignment is performed by Beam Search algorithm when the codebooks are fixed. In this work, we propose a straight-forward method to directly learn the code assignment and codebooks simutaneously in an end-to-end neural network.

Some recent works (Xia et al., 2014, Liu et al., 2016, Yang et al., 2017) in learning to hash also utilize neural networks to produce binary codes by applying binary constrains (e.g., sigmoid function). In this work, we encourage the discreteness with the Gumbel-Softmax trick for producing compositional codes.

As an alternative to our approach, one can also reduce the number of unique word types by forcing a character-level segmentation. Kim et al. (2016) proposed a character-based neural language model, which applies a convolutional layer after the character embeddings. Botha et al. (2017) propose to use char-gram as input features, which are further hashed to save space. Generally, using character-level inputs requires modifications to the model architecture. Moreover, some Asian languages such as Japanese and Chinese retain a large vocabulary at the character

level, which makes the character-based approach difficult to be applied. In contrast, our approach does not suffer from these limitations.

5.4 Advantage of Compositional Codes

In this section, we formally describe the compositional coding approach and analyze its merits for compressing word embeddings. The coding approach follows the scheme in additive quantization (Babenko and Lempitsky, 2014). We represent each word w with a compact code C_w that is composed of M components such that $C_w \in Z_+^M$. Each component C_w^i is constrained to have a value in $[1, K]$, which also indicates that $M \log_2 K$ bits are required to store each code. For convenience, K is selected to be a number of a multiple of 2, so that the codes can be efficiently stored.

If we restrict each component C_w^i to values of 0 or 1, the code for each word C_w will be a binary code. In this case, the code learning problem is equivalent to a matrix factorization problem with binary components. Forcing the compact codes to be binary numbers can be beneficial, as the learning problem is usually easier to solve in the binary case, and some existing optimization algorithms in learning to hash can be reused. However, the compositional coding approach produces shorter codes and is thus more storage efficient.

As the number of basis vectors is $M \times K$ regardless of the vocabulary size, the only uncertain factor contributing to the model size is the size of the hash codes, which is proportional to the vocabulary size. Therefore, maintaining short codes is cruicial in our work. Suppose we wish the model to have a set of N basis vectors. Then in the binary case, each code will have $N/2$ bits. For the compositional coding approach, if we can find a $M \times K$ decomposition such that $M \times K = N$, then each code will have $M \log_2 K$ bits. For example, a binary code will have a length of 256 bits to support 512 basis vectors. In contrast, a 32×16 compositional coding scheme will produce codes of only 128 bits.

A comparison of different coding approaches is summarized in Table 5.1. We also report the number of basis vectors required to compute an embedding as a measure of computational cost. For the conventional approach, the number of vectors is identical to the vocabulary size and the computation is basically a single indexing operation. In the case of binary codes, the computation for constructing an embedding involves a summation over $N/2$ basis vectors. For the compositional approach, the number

	#vectors	computation	code length (bits)
conventional	$\|V\|$	1	-
binary	N	$N/2$	$N/2$
compositional	MK	M	$M \log_2 K$

Table 5.1 Comparison of different coding approaches. To support N basis vectors, a binary code will have $N/2$ bits and the embedding computation is a summation over $N/2$ vectors. For the compositional approach with M codebooks and K codewords in each codebook, each code has $M \log_2 K$ bits, and the computation is a summation over M vectors.

of vectors required to construct an embedding vector is M. Both the binary and compositional approaches have significantly fewer vectors in the embedding matrix. The compositional coding approach provides a better balance with shorter codes and lower computational cost.

5.5 Code Learning with Gumbel-Softmax

Let $\tilde{\mathbf{E}} \in \mathbb{R}^{|V| \times H}$ be the original embedding matrix, where each embedding vector has H dimensions. By using the reconstruction loss as the objective function in Eq. (5.3), we are actually finding an approximate matrix factorization $\tilde{\mathbf{E}} \approx \sum_{i=0}^{M} \boldsymbol{D^i A_i}$, where $\boldsymbol{A_i} \in \mathbb{R}^{K \times H}$ is a basis matrix for the i-th component. $\boldsymbol{D^i}$ is a $|V| \times K$ code matrix in which each row is an K-dimensional one-hot vector. If we let $\boldsymbol{d_w^i}$ be the one-hot vector corresponding to the code component C_w^i for word w, the computation of the word embeddings can be reformulated as

$$E(C_w) = \sum_{i=0}^{M} \boldsymbol{A_i^\top d_w^i}. \tag{5.5}$$

Therefore, the problem of learning discrete codes C_w can be converted to a problem of finding a set of optimal one-hot vectors $\boldsymbol{d_w^1}, ..., \boldsymbol{d_w^M}$ and source dictionaries $\boldsymbol{A_1}, ..., \boldsymbol{A_M}$, that minimize the reconstruction loss. The Gumbel-softmax reparameterization trick (Maddison et al., 2016, Jang et al., 2016) is useful for parameterizing a discrete distribution such as the K-dimensional one-hot vectors $\boldsymbol{d_w^i}$ in Eq. (5.5). By applying

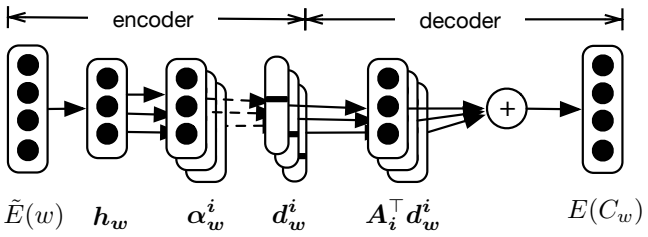


Fig. 5.2 The network architecture for learning compositional compact codes. The Gumbel-softmax computation is marked with dashed lines.

the Gumbel-softmax trick, the k-th elemement in $\boldsymbol{d_w^i}$ is computed as

$$\left(\boldsymbol{d_w^i}\right)_k = \text{softmax}_\tau(\log \boldsymbol{\alpha_w^i} + G)_k \tag{5.6}$$

$$= \frac{\exp((\log\ (\boldsymbol{\alpha_w^i})_k + G_k)/\tau)}{\sum_{k'=1}^{K} \exp((\log\ (\boldsymbol{\alpha_w^i})_{k'} + G_{k'})/\tau)}\ , \tag{5.7}$$

where G_k is a noise term that is sampled from the Gumbel distribution by computing $-\log(-\log(\text{Uniform}[0,1]))$, whereas τ is the temperature of the softmax. In our model, the vector $\boldsymbol{\alpha_w^i}$ is computed by a simple neural network with a single hidden layer as

$$\boldsymbol{\alpha_w^i} = \text{softplus}(\boldsymbol{\theta_i'}^\top \boldsymbol{h_w} + \boldsymbol{b_i'}), \tag{5.8}$$

$$\boldsymbol{h_w} = \tanh(\boldsymbol{\theta}^\top \tilde{E}(w) + \boldsymbol{b})\ . \tag{5.9}$$

In our experiments, the hidden layer $\boldsymbol{h_w}$ always has a size of $MK/2$. We found that a fixed temperature of $\tau = 1$ just works well. The Gumbel-softmax trick is applied to $\boldsymbol{\alpha_w^i}$ to obtain $\boldsymbol{d_w^i}$. Then, the model reconstructs the embedding $E(C_w)$ with Eq. (5.5) and computes the reconstruction loss with Eq. (5.3). The model architecture of the end-to-end neural network is illustrated in Fig. 5.2, which is effectively an auto-encoder with a Gumbel-softmax middle layer. The whole neural network for coding learning has five parameters $(\boldsymbol{\theta}, \boldsymbol{b}, \boldsymbol{\theta'}, \boldsymbol{b'}, \boldsymbol{A})$.

Once the coding learning model is trained, the code C_w for each word can be easily obtained by applying argmax to the one-hot vectors $\boldsymbol{d_w^1}, ..., \boldsymbol{d_w^M}$. The basis vectors (codewords) for composing the embeddings can be found as the row vectors in the weight matrix $\boldsymbol{A}$.

For general NLP tasks, one can learn the compositional codes from publicly available word vectors such as GloVe vectors. However, for some tasks such as

machine translation, the word embeddings are usually jointly learned with other parts of the neural network. For such tasks, one has to first train a normal model to obtain the baseline embeddings. Then, based on the trained embedding matrix, one can learn a set of task-specific codes. As the reconstructed embeddings $E(C_w)$ are not identical to the original embeddings $\tilde{E}(w)$, the model parameters other than the embedding matrix have to be retrained again. The code learning model cannot be jointly trained with the machine translation model as it takes far more iterations for the coding layer to converge to one-hot vectors.

5.6 Experiments

In our experiments, we focus on evaluating the maximum loss-free compression rate of word embeddings on two typical NLP tasks: sentiment analysis and machine translation. We compare the model performance and the size of embedding layer with the baseline model and the iterative pruning method (Han et al., 2015). Please note that the sizes of other parts in the neural networks are not included in our results. For dense matrices, we report the size of dumped numpy arrays. For the sparse matrices, we report the size of dumped *compressed sparse column matrices* (`csc_matrix`) in scipy. All float numbers take 32 bits storage. We enable the "compressed" option when dumping the matrices, without this option, the file size is about 1.1 times bigger.

5.6.1 Code Learning

To learn efficient compact codes for each word, our proposed method requires a set of baseline embedding vectors. For the sentiment analysis task, we learn the codes based on the publicly available GloVe vectors. For the machine translation task, we first train a normal neural machine translation (NMT) model to obtain task-specific word embeddings. Then we learn the codes using the pre-trained embeddings.

We train the end-to-end network described in Section 5.5 to learn the codes automatically. In each iteration, a small batch of the embeddings is sampled uniformly from the baseline embedding matrix. The network parameters are optimized to minimize the reconstruction loss of the sampled embeddings. In our experiments, the batch size is set to 128. We use Adam optimizer (Kingma and Ba, 2014) with a

fixed learning rate of 0.0001. The training is run for 200K iterations. Every 1,000 iterations, we examine the loss on a fixed validation set and save the parameters if the loss decreases. We evenly distribute the model training to 4 GPUs using the *nccl* package, so that one round of code learning takes around 15 minutes to complete.

5.6.2 Sentiment Analysis

Dataset: For sentiment analysis, we use a standard separation of IMDB movie review dataset (Maas et al., 2011), which contains 25k reviews for training and 25K reviews for testing purpose. We lowercase and tokenize all texts with the *nltk* package. We choose the 300-dimensional uncased GloVe word vectors (trained on 42B tokens of Common Crawl data) as our baseline embeddings. The vocabulary for the model training contains all words appears both in the IMDB dataset and the GloVe vocabulary, which results in around 75K words. We truncate the texts of reviews to assure they are not longer than 400 words.

Model architecture: Both the baseline model and the compressed models have the same computational graph except the embedding layer. The model is composed of a single LSTM layer with 150 hidden units and a softmax layer for predicting the binary label. For the baseline model, the embedding layer contains a large $75K \times 300$ embedding matrix initialized by GloVe embeddings. For the compressed models based on the compositional coding, the embedding layer maintains a matrix of basis vectors. Suppose we use a 32×16 coding scheme, the basis matrix will then have a shape of 512×300, which is initialized by the concatenated weight matrices $[\boldsymbol{A_1}; \boldsymbol{A_2}; ...; \boldsymbol{A_M}]$ in the code learning model. The embedding parameters for both models remain fixed during the training. For the models with network pruning, the sparse embedding matrix is finetuned during training.

Training details: The models are trained with Adam optimizer for 15 epochs with a fixed learning rate of 0.0001. At the end of each epoch, we evaluate the loss on a small validation set. The parameters with lowest validation loss are saved.

Results: For different settings of the number of components M and the number of codewords K, we train the code learning network. The average reconstruction loss on a fixed validation set is summarized in the left of Table 5.2. For reference, we also report the total size (MB) of the embedding layer in the right table, which includes the sizes of the basis matrix and the hash table. We can see that increasing either M or K can effectively decrease the reconstruction loss. However, setting M

loss	M=8	M=16	M=32	M=64	size	M=8	M=16	M=32	M=64
K=8	29.1	25.8	21.9	15.5	K=8	0.28	0.56	1.12	2.24
K=16	27.0	22.8	19.1	11.5	K=16	0.41	0.83	1.67	3.34
K=32	24.4	20.4	14.3	9.3	K=32	0.62	1.24	2.48	4.96
K=64	21.9	16.9	12.1	7.6	K=64	0.95	1.91	3.82	7.64

Table 5.2 Reconstruction loss and the size of embedding layer (MB) of difference settings

to a large number will result in longer hash codes, thus significantly increase the size of the embedding layer. Hence, it is important to choose correct numbers for M and K to balance the performance and model size.

To see how the reconstructed loss translates to the classification accuracy, we train the sentiment analysis model for different settings of code schemes and report the results in Table 5.3. The baseline model using 75k GloVe embeddings achieves an accuracy of 87.18 with an embedding matrix using 78 MB of storage. In this task, forcing a high compression rate with iterative pruning degrades the classification accuracy.

	#vec	**vec size**	**code**	**code size**	**total size**	**acc.**
GloVe baseline	75K	78 MB	-	-	78 MB	87.18
prune 80%	75K	21 MB	-	-	21 MB	86.25
prune 90%	75K	11 MB	-	-	11 MB	84.96
NPQ (10×256)	256	0.26 MB	80 bits	0.71 MB	0.97 MB	86.21
NPQ (60×256)	256	0.26 MB	480 bits	4.26 MB	4.52 MB	87.11
8×64 coding	512	0.52 MB	48 bits	0.42 MB	0.94 MB	86.66
16×32 coding	512	0.52 MB	80 bits	0.71 MB	**1.23 MB**	87.37
32×16 coding	512	0.52 MB	128 bits	1.14 MB	1.66 MB	87.80
64×8 coding	512	0.52 MB	192 bits	1.71 MB	2.23 MB	**88.15**

Table 5.3 Trade-off between the model performance and the size of embedding layer on IMDB sentiment analysis task

We also show the results using normalized product quantization (NPQ) (Joulin et al., 2016). We quantize the filtered GloVe embeddings with the codes provided by the authors, and train the models based on the quantized embeddings. To make the results comparable, we report the codebook size in numpy format. For our proposed methods, the maximum loss-free compression rate is achieved by a 16×32 coding scheme. In this case, the total size of the embedding layer is 1.23 MB, which is

equivalent to a compression rate of 98.4%. We also found the classification accuracy can be substantially improved with a slightly lower compression rate. The improved model performance may be a byproduct of the strong regularization.

5.6.3 Machine Translation

Dataset: For machine translation tasks, we experiment on IWSLT 2014 German-to-English translation task (Cettolo et al., 2014) and ASPEC English-to-Japanese translation task (Nakazawa et al., 2016). The IWSLT14 training data contains 178K sentence pairs, which is a small dataset for machine translation. We utilize moses toolkit (Koehn et al., 2007) to tokenize and lowercase both sides of the texts. Then we concatenate all five TED/TEDx development and test corpus to form a test set containing 6750 sentence pairs. We apply byte-pair encoding (Sennrich et al., 2016b) to transform the texts to subword level so that the vocabulary has a size of 20K for each language. For evaluation, we report *tokenized BLEU* using "multi-bleu.perl".

The ASPEC dataset contains 300M bilingual pairs in the training data with the automatically estimated quality scores provided for each pair. We only use the first 150M pairs for training the models. The English texts are tokenized by moses toolkit whereas the Japanese texts are tokenized by kytea (Neubig et al., 2011). The vocabulary size for each language is reduced to 40K using byte-pair encoding. The evaluation is performed using a standard kytea-based post-processing script for this dataset.

Model architecture: In our preliminary experiments, we found a 32×16 coding works well for a vanilla NMT model. As it is more meaningful to test on a high-performance model, we applied several techniques to improve the performance. The model has a standard bi-directional encoder composed of two LSTM layers similar to Bahdanau et al. (2015). The decoder contains two LSTM layers. Residual connection (He et al., 2016) with a scaling factor of $\sqrt{1/2}$ is applied to the two decoder states to compute the outputs. All LSTMs and embeddings have 256 hidden units in the IWSLT14 task and 1000 hidden units in ASPEC task. The decoder states are firstly linearly transformed to 600-dimensional vectors before computing the final softmax. Dropout with a rate of 0.2 is applied everywhere except the recurrent computation. We apply Key-Value Attention (Miller et al., 2016) to the first decoder, where the query is the sum of the feedback embedding and the previous decoder state and the keys are computed by linear transformation of encoder states.

Training details: All models are trained by Nesterov's accelerated gradient (Nesterov, 1983) with an initial learning rate of 0.25. We evaluate the smoothed BLEU (Lin and Och, 2004) on a validation set composed of 50 batches every 7,000 iterations. The learning rate is reduced by a factor of 10 if no improvement is observed in 3 validation runs. The training ends after the learning rate is reduced three times. Similar to the code learning, the training is distributed to 4 GPUs, each GPU computes a mini-batch of 16 samples.

We firstly train a baseline NMT model to obtain the task-specific embeddings for all in-vocabulary words in both languages. Then based on these baseline embeddings, we obtain the hash codes and basis vectors by training the code learning model. Finally, the NMT models using compositional coding are retrained by plugging in the reconstructed embeddings. Note that the embedding layer is fixed in this phase, other parameters are retrained from random initial values.

Results: The experimental results are summarized in Table 5.4. All translations are decoded by the beam search with a beam size of 5. The performance of iterative pruning varies between tasks. The loss-free compression rate reaches 92% on ASPEC dataset by pruning 90% of the connections. However, with the same pruning ratio, a modest performance loss is observed in IWSLT14 dataset.

For the models using compositional coding, the loss-free compression rate is 94% for the IWSLT14 dataset and 99% for the ASPEC dataset. Similar to the sentiment analysis task, a significant performance improvement can be observed by slightly lowering the compression rate. Note that the sizes of NMT models are still quite large due to the big softmax layer and the recurrent layers, which are not reported in the table. Please refer to existing works such as Zhang et al. (2017) for the techniques of compressing layers other than word embeddings.

5.7 Qualitative Analysis

5.7.1 Examples of Learned Codes

In Table 5.5, we show some examples of learned codes based on the 300-dimensional uncased GloVe embeddings used in the sentiment analysis task. We can see that the model learned to assign similar codes to the words with similar meanings. Such

	coding	#vec	vec size	code	code size	total size	BLEU
De-En	baseline	40K	35 MB	-	-	35 MB	29.45
	prune 90%	40K	5.21 MB	-	-	5.21 MB	29.34
	prune 95%	40K	2.63 MB	-	-	2.63 MB	28.84
	32×16	512	0.44 MB	128 bits	0.61 MB	1.05 MB	29.04
	64×16	1024	0.89 MB	256 bits	1.22 MB	**2.11 MB**	**29.56**
En-Ja	baseline	80K	274 MB	-	-	274 MB	37.93
	prune 90%	80K	41 MB	-	-	41 MB	38.56
	prune 98%	80K	8.26 MB	-	-	8.26 MB	37.09
	32×16	512	1.75 MB	128 bits	1.22 MB	**2.97 MB**	38.10
	64×16	1024	3.50 MB	256 bits	2.44 MB	5.94 MB	**38.89**

Table 5.4 Trade-off between the model performance and the size of embedding layer in machine translation tasks

a code-sharing mechanism can significantly reduce the redundancy of the word embeddings, thus helping to achieve a high compression rate.

category	word	8×8 code	16×16 code
animal	dog	0 7 0 1 7 3 7 0	7 7 0 8 3 5 8 5 B 2 E E 0 B 0 A
	cat	7 7 0 1 7 3 7 0	7 7 2 8 B 5 8 C B 2 E E 4 B 0 A
	penguin	0 7 0 1 7 3 6 0	7 7 E 8 7 6 4 C F D E 3 D 8 0 A
verb	go	7 7 0 6 4 3 3 0	2 C C 8 2 C 1 1 B D 0 E 0 B 5 8
	went	4 0 7 6 4 3 2 0	B C C 6 B C 7 5 B 8 6 E 0 D 0 4
	gone	7 7 0 6 4 3 3 0	2 C C 8 0 B 1 5 B D 6 E 0 2 5 A

Table 5.5 Examples of learned compositional codes based on GloVe embedding vectors

5.7.2 Analysis of Code Efficiency

Besides the performance, we also care about the storage efficiency of the codes. In the ideal situation, all codewords shall be fully utilized to convey a fraction of meaning. However, as the codes are automatically learned, it is possible that some codewords are abandoned during the training. In extreme cases, some "dead" codewords can be used by none of the words.

To analyze the code efficiency, we count the number of words that contain a specific subcode in each component. Figure 5.3 gives a visualization of the code

balance for three coding schemes. Each column shows the counts of the subcodes of a specific component. In our experiments, when using a 8×8 coding scheme, we found 31% of the words have a subcode "0" for the first component, while the subcode "1" is only used by 5% of the words. The assignment of codes is more balanced for larger coding schemes. In any coding scheme, even the most unpopular codeword is used by about 1000 words. This result indicates that the code learning model is capable of assigning codes efficiently without wasting a codeword.

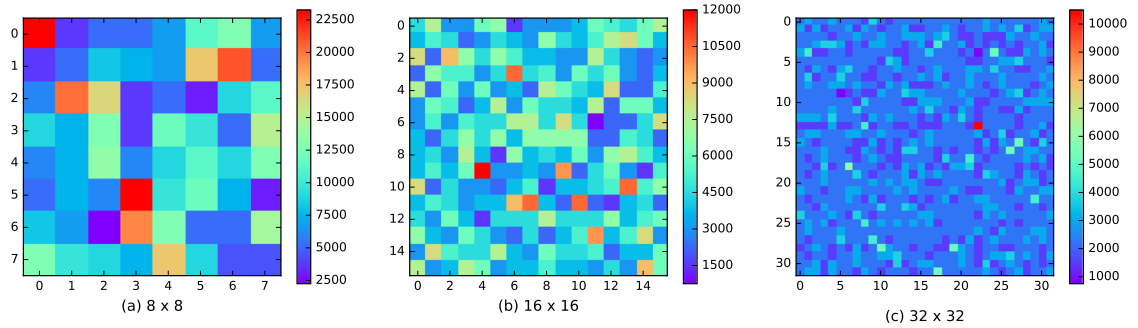


Fig. 5.3 Visualization of code balance for different coding scheme. Each cell in the heat map shows the count of words containing a specific subcode. The results show that any codeword is assigned to more than 1000 words without wasting.

5.7.3 Shared Codes

In both tasks, when we use a small code decomposition, we found some hash codes are assigned to multiple words. Table 5.6 lists some samples of shared codes with their corresponding words from the sentiment analysis task. This phenomenon does not cause a problem in either task, as the words only have shared codes when they have almost the same sentiments or target translations.

shared code	words
4 7 7 0 4 7 1 1	homes cruises motel hotel resorts mall vacations hotels
6 6 7 1 4 0 2 0	basketball softball nfl nascar baseball defensive ncaa tackle nba
3 7 3 2 4 3 3 0	unfortunately hardly obviously enough supposed seem totally ...
4 6 7 0 4 7 5 0	toronto oakland phoenix miami sacramento denver ...
7 7 6 6 7 3 0 0	yo ya dig lol dat lil bye

Table 5.6 Examples of words sharing same codes when using a 8×8 code decomposition

5.7.4 Semantics of Codes

In order to see whether each component captures semantic meaning. we learned a set of codes using a 3 x 256 coding scheme, this will force the model to decompose each embedding into 3 vectors. In order to maximize the compression rate, the model must make these 3 vectors as independent as possible.

word	code
man	210 153 153
woman	232 153 153
king	210 180 039
queen	232 180 039
British	118 132 142
London	185 126 142
Japan	118 056 021
Tokyo	185 036 021

Table 5.7 Some code examples using a 3×256 coding scheme.

As we can see from Table 5.7, we can transform "man/king" to "woman/queen" by change the subcode "210" in the first component to "232". So we can think "210" must be a "male" code, and "232" must be a "female" code. Such phenomenon can also be observed in other words such as city names.

5.8 Conclusion

In this work, we propose a novel method for reducing the number of parameters required in word embeddings. Instead of assigning each unique word an embedding vector, we compose the embedding vectors using a small set of basis vectors. The selection of basis vectors is governed by the hash code of each word. We apply the compositional coding approach to maximize the storage efficiency. The proposed method works by eliminating the redundancy inherent in representing similar words with independent embeddings. In our work, we propose a simple way to directly learn the discrete codes in a neural network with Gumbel-softmax trick. The results show that the size of the embedding layer was reduced by 98% in IMDB sentiment analysis task and 94% $\sim$ 99% in machine translation tasks without affecting the performance.

Our approach achieves a high loss-free compression rate by considering the semantic inter-similarity among different words. In qualitative analysis, we found the learned codes of similar words are very close in Hamming space. As our approach maintains a dense basis matrix, it has the potential to be further compressed by applying pruning techniques to the dense matrix. The advantage of compositional coding approach will be more significant if the size of embedding layer is dominated by the hash codes.

Chapter 6

Learning Syntactic Latent Variables for Diverse Translation

6.1 Motivation

When using machine translation systems, users may desire to see different candidate translations other than the best one. In this scenario, users usually expect the system to show candidates with different sentence structures.

To obtain diverse translations, conventional neural machine translation (NMT) models allow one to sample translations using the beam search algorithm, however, they usually share similar sentence structures. Recently, various methods (Li et al., 2016, Xu et al., 2018) are proposed for diverse generation. These methods encourage the model to use creative vocabulary to achieve high diversity. Although producing creative words benefits tasks in the dialog domain, when applied to machine translation, it can hurt the translation quality by changing the original meaning.

In this work, we are interested in generating multiple valid translations with high diversity. To achieve this, we propose to construct the codes based on semantics-level or syntax-level information of target-side sentences.

To generate diverse translations, we constrain the generation model by specifying a particular code as a semantic or syntactic assignment. More concretely, we prefix the target-side sentences with the codes. Then, an NMT model is trained with the original source sentences and the prefixed target sentences. As the model generates tokens in left-to-right order, the probability of emitting each word is predicted

conditioned on the assigned code. As each assignment is supposed to correspond to a sentence structure, the candidate translations sampled with different assignments are expected to have high diversity.

We can think such model as a mixture-of-expert translation model where each expert is capable of producing translations with a certain style indicated by the code. In the inference time, code assignments are given to the model so that a selection of experts are picked to generate translations.

The key question is how to extract such sentence codes. Here, we explore two approaches. First, a simple unsupervised method is tested, which clusters the sentence embeddings and use the cluster ids as the code assignments. Next, to capture only the structural variation of sentences, we turn to syntax. We encode the structure of constituent parse trees into discrete codes with a tree auto-encoder.

Experiments on two machine translation datasets show that a set of highly diverse translations can be obtained with reasonable mechanism for extracting the sentence codes, while the sampled candidates still have BLEU scores on par with the baselines. As our approach only modifies the training data, but does not require modification to the NMT models, it can be easily adopted in existing machine translation systems.

The contributions of this work can be summarized as follows:

1. We propose a novel approach for obtaining candidate translation with only syntactic diversity. Our approach encodes the syntactic tags into few discrete codes to minimize negative impact on translation speed.

2. Through experiments, we show that the quality of the best translation still remains intact with the discrete codes as a syntactic prior.

3. To quantatively evaluate our approach, we propose a syntactic diversity metric. The evaluation results show that our approach is capable of producing translations with drastically different structures.

6.2 Proposed Approach

6.2.1 Extracting Sentence Codes

Our approach produces diverse translations by conditioning sentence generation with the sentence codes. Ideally, we would like the codes to capture the information about the sentence structures rather than utterances. To extract such codes from target sentences, we explore two methods.

Semantic Coding Model The first method extracts sentence codes from unsupervisedly learned semantic information. We cluster the sentence embeddings produced by pre-trained models into a fixed number of clusters, then use the cluster ids as discrete priors to condition the sentence generation. In this work, we test two semantic coding models. The first model is based on BERT (Devlin et al., 2018), where the vectors corresponding to the "[CLS]" token are clustered.

The second model produces sentence embeddings by averaging FastText word embeddings (Bojanowski et al., 2017). Comparing to the hidden states of BERT, word embeddings are expected to contain less syntactic information as the word order is ignored during training.

Syntactic Coding Model To explicitly capture the syntactic diversity, we also consider to derive the sentence codes from the parse trees produced by a constituency parser. As the utterance-level information is not desired, the terminal nodes are removed from the parse trees.

To obtain the sentence codes, we use a TreeLSTM-based auto-encoder similar to Socher et al. (2011a), which encodes the syntactic information into a single discrete code. As illustrated in Fig. 6.1 (a), a TreeLSTM cell (Tai et al., 2015) computes a recurrent state based on a given input vector and the states of N_i child nodes:

$$h_i = f_{cell}(x_i, h_{i1}, h_{i2}, ..., h_{iN_i}; \theta). \tag{6.1}$$

The tree auto-encoder model is shown in Fig. 6.1 (c), where the encoder computes a latent tree representation. As the decoder has to unroll the vector representation following a reversed tree structure to predict the non-terminal labels, the standard TreeLSTM equation cannot be directly applied. To compute along the reversed tree, we modify Eq. (6.1) for computing the hidden state of the j-th child node given the

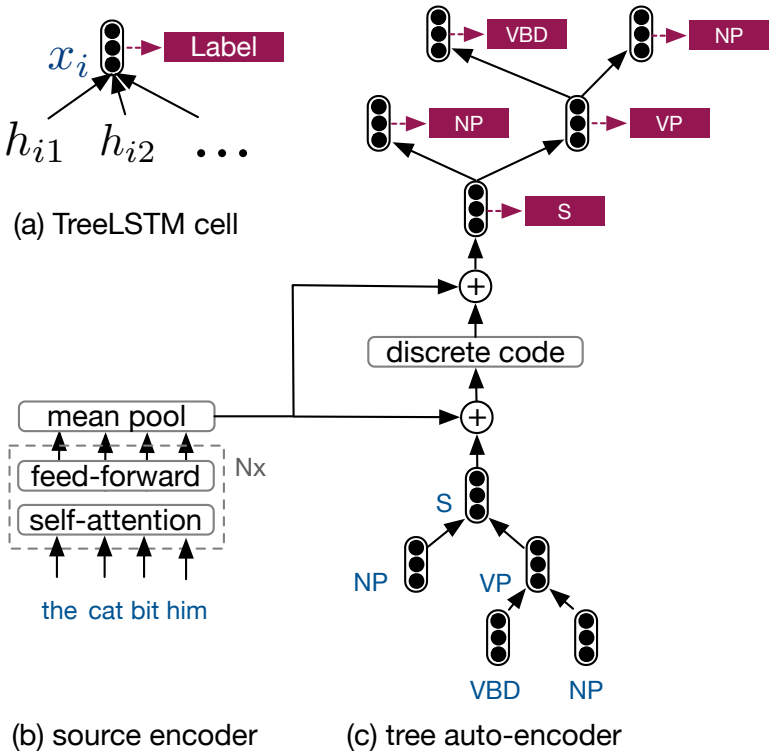


Fig. 6.1 Architecture of the TreeLSTM-based auto-encoder with a discretization bottleneck for learning the sentence codes.

parent-node state h_i:

$$h_{ij} = f_{dec}(h_i; \theta_j), \tag{6.2}$$

where the internal implementation of the recurrent function is same as Eq. (6.1), however, each node has a different parameterization depending on its position among siblings. Note that in the decoder side, no input vectors are fed to the recurrent computation. Finally, the decoder states are used to predict target labels, whereas the model is optimized with cross-entropy loss.

As the source sentence already provides hints on the target-side sentence structure, we feed the source information to the tree auto-encoder to encourage the latent representation to capture the syntax that cannot be inferred from the source sentence.

As some target-side syntactic information is obvious given the source sentence, we do not want the latent representation to encode such information, but rather substitutable syntactic choices. For example, the structure of "NP VP" is obvious given the source sentence in Fig. 6.1 (b). Therefore, we use an attention-based

encoder followed by a mean pool operation to create a source representation and feed it to the decoder.

To obtain the sentence codes from the latent tree representation, we apply *improved semantic hashing* (Kaiser and Bengio, 2018) to the hidden state of the root node, which discretizes the vector into a 8-bit code (binary vector). When performing improved semantic hashing, the forward pass computes two operations: binarization and saturated sigmoid, resulting in two vectors. One of these two vectors are randomly selected for the next computation. However, in the backward pass, the gradient always flows through the vector produced by saturated sigmoid. As the model is trained together with the bottleneck, the codes are optimized directly to minimize the loss function.

6.2.2 Diverse Generation with Code Assignment

Once we obtain the sentence codes, we prefix the target-side sentences in the training data with the corresponding codes. The resultant target sentence has a form of "⟨c12⟩ ⟨eoc⟩ Here is a translation.". The "⟨eoc⟩" token separates the code and words.

We train a regular NMT model with the modified training dataset. To generate diverse translations, we first obtain top-K codes from the probability distribution of code prediction. In detail, we select K sentence codes with the highest probabilities. Then, conditioning on each code, we let the beam search continue to generate the sentence, resulting in K translations conditioned on different codes.

6.3 Related Work

Our work is related to other works of diverse language generation. Though differing in purpose, our approach is also related to NMT models with syntactic constraint.

Style Transfer for Natural Language As our task learns the structural representation, the problem setting is closely related to style transfer for language generation tasks. Hu et al. (2017) simultaneously learns a disentangled representation along with entangled representations within the same latent vector. The disentangled representation is trained with a style-specific discriminator. Shen et al. (2017) learns a cross-aligned auto encoder to tackle the style transfer problem without parallel text data. Prabhumoye et al. (2018) utilizes back-translation to create a style

agnostic representation of sentence. Then multiple style transfer decoders is trained to generate sentences in specific styles according to the latent representation. The parameters are optimized with style classifiers using adversarial loss. The main difference between our task and previous works in style transfer is in the objectives. In the aforementioned works, the styles have finite categories. For example, the label for sentiment style can only be "positive" or "negative". In our task setting, the objective is to learn a representation that captures the global sentence structure, which has infinite variations.

Theoretically, we can still train a variational auto-encoder (VAE) that learns a categorical latent code or a continuous latent variable with a Gaussian prior to capture the structure. We can extend the VAE model described in Hu et al. (2017) with an enhanced discriminator $q_D(c|x)$ to map the global structure of a sentence x to the space of a latent variable c, and ensure that sentences with similar structures are mapped to same or close latent representations. However, as the structural information is highly entangled with the utterances, such a discriminator is fairly difficult to implement.

Diverse Language Generation Existing works for diverse text generation can be categorized into two major categories. The approaches in the first categoriy sample diverse sequences by varying a hidden representation. Jain et al. (2017) generates diverse questions by injecting Gaussian noise to the latent in a VAE for encouraging the creativity of results. Xu et al. (2018) learns K shared decoders, conditioned on different pattern rewriting embeddings. The former method is evaluated by assessing the ability of generating unique and unseen results, whereas the latter is evaluated with the number of unique uni/bi-grams and the divergence of word distributions produced by different decoders. Independent to this work, Shen et al. (2019) also explores mixture-of-expert models with an ensemble of learners. The paper discusses multiple training strategies and found the multiple choice learning works best.

The second category of approaches attempts to improve the diversity by improving decoding algorithms. Li et al. (2016) modifies the scoring function in beam search to encourage the algorithm to promote hypotheses containing words from different ancestral hypotheses, which is also evaluated with the number of unique uni/bi-grams. Kulikov et al. (2018) uses an iterative beam search approach to generate diverse dialogs.

Comparing to these works, we focus on generating translations with different sentence structures. We still use beam search to search for best words in every

decoding steps under the constraint of code assignment. Our approach also comes with the advantage that no modification to the NMT model architecture is required.

NMT with Syntactic Constraint To our knowledge, there are no existing works control the sentence structure ahead of translation. However, our approach is related to NMT models under syntactic constraint. Stahlberg et al. (2016) syntactically guides the NMT decoding using the lattice produced by Hiero, a statistical machine translation system allowing hierarchical phrase structure. Eriguchi et al. (2017) parse a dependency tree and combine the parsing loss with the original loss, which improves the syntactic trace of translations in a soft way. The following works also enhance the target sentence with extra syntactic tags. Nadejde et al. (2017) interleaves CCG supertags with normal output words in the target side. Instead of predicting words, Aharoni and Goldberg (2017) trains an NMT model to generate linearized constituent parse trees. Wu et al. (2017) proposed a model to generate words and parse actions simultaneously. The word prediction and action prediction are conditioned on each other. These works focus on improving the syntactic correctness, whereas our work focuses on syntactic diversity.

Similar to our code learning approach, some works also learn the discrete codes for different purposes. Shu and Nakayama (2018) compresses the word embeddings by learning the concept codes to represent each word. Kaiser et al. (2018b) breaks down the dependency among words with shorter code sequences. The decoding can be faster by predicting the shorter artificial codes.

6.4 Experiments

6.4.1 Experimental Settings

We evaluate our models on two machine translation datasets: ASPEC Japanese-to-English dataset (Nakazawa et al., 2016) and WMT14 German-to-English dataset. The datasets contain 3M and 4.5M bilingual pairs respectively. For the ASPEC Ja-En dataset, we use the Moses toolkit (Koehn et al., 2007) to tokenize the English side and Kytea (Neubig et al., 2011) to tokenize the Japanese side. After tokenization, we apply byte-pair encoding (Sennrich et al., 2016b) to segment the texts into subwords, forcing the vocabulary size of each language to be 40k. For WMT14 De-En dataset,

code setting	capacity	IWSLT14 De-En		ASPEC Ja-En	
		tag acc.	code acc.	tag acc.	code acc.
N=1, K=4	2 bits	27%	63%	35%	40%
N=2, K=2	2 bits	23%	67%	27%	55%
N=2, K=4	4 bits	35%	41%	58%	25%
N=4, K=2	4 bits	22%	44%	18%	79%
N=4, K=4	8 bits	44%	27%	14%	58%

Table 6.1 A comparison of different code settings on IWSLT14 De-En and ASPEC Ja-En datset. For each code setting, we report the accuracy of recovering the syntactic tag sequence using the codes (tag accuracy), and the accuracy of predicting the codes using the source sentence (code accuracy).

we use sentencepiece (Kudo and Richardson, 2018) to segment the words to ensure a vocabulary size of 32k.

In evaluation, we report *tokenized BLEU* for ASPEC Ja-En dataset. For WMT14 De-En dataset, BLEU scores are generated using SacreBleu toolkit (Post, 2018b). For models that produce sentence codes during decoding, the codes are removed from translation results before evaluating BLEU scores.

6.4.2 Obtaining Sentence Codes

For the semantic coding model based on BERT, we cluster the hidden state of "[CLS]" token into 256 clusters with k-means algorithm. The cluster ids are then used as sentence codes. For models using FastText Embeddings, pre-trained vectors (Common Crawl, 2M words) are used. Please note that the number of clusters is a hyperparameter, here we choose the number of clusters to match the number of unique codes in the syntax-based model.

To train the syntax coding model, we parse target-side sentences with Stanford CFG parser (Klein and Manning, 2003). The TreeLSTM-based auto-encoder is implemented with DGL,[1] which is trained using AdaGrad optimizer for faster convergence. We found it helpful to pre-train the model without the discretization bottleneck for achieving higher label accuracy.

[1] https://www.dgl.ai/

Model	BLEU	Oracle	DP
ASPEC Ja → En			
Transformer Baseline	27.1	-	22.4
+Diverse Dec (Li et al., 2016)	26.9	-	26.2
+ Random Codes	27.0	-	4.9
+ Semantic Coding (BERT)	26.8	28.8	30.6
+ Semantic Coding (FastText)	27.3	28.5	31.1
+ Syntactic Coding	27.4	29.5	**39.8**
WMT14 De → En			
Transformer Baseline	29.4	-	28.2
+Diverse Dec (Li et al., 2016)	29.1	-	31.0
+ Random Codes	29.5	-	3.8
+ Semantic Coding (BERT)	29.3	29.4	21.7
+ Semantic Coding (FastText)	28.5	29.2	28.8
+ Syntactic Coding	29.3	30.7	**33.0**

Table 6.2 Results for different approaches. The BLEU(%) are reported for the first sampled candidate. Oracle BLEU scores are produced with reference codes. Diversity scores (DP) are evaluated with Eq. (6.3).

6.4.3 Quantitive Evaluation of Diversity

As we are interested in the diversity among sampled candidates, the diversity metric based on the divergence between word distributions (Xu et al., 2018) can not be applied in this case. In order to qualitatively evaluate the diversity of generated translations, we propose to use a BLEU-based discrepancy metric. Suppose Y is a list of candidate translations, we compute the diversity score with

$$\mathrm{DP}(Y) = \frac{1}{|Y|(|Y|-1)} \sum_{y \in Y} \sum_{y' \in Y, y' \neq y} 1 - \Delta(y, y'), \tag{6.3}$$

where $\Delta(y, y')$ returns the BLEU score of two candidates. The equation gives a higher diversity score when each candidate contains more unique n-grams.

6.4.4 Experiment Results

We use Base Transformer architecture (Vaswani et al., 2017) for all models. The results are summarized in Table 6.2. We sample three candidates with different models, and report the averaged diversity score. The BLEU(%) is reported for the candidate with highest confidence (log-probability). A detailed table with BLEU scores of all three candidates can be found in supplementary material, where the BLEU scores of the second and third candidates are on par with the baseline.

We compare the proposed approach to three baselines. The first baseline samples three candidates using standard beam search. We also tested the diverse decoding approach (Li et al., 2016). The coefficient γ is chosen to maximize the diversity with no more than 0.5 BLEU degradation. The third baseline uses random codes for conditioning.

As shown in the table, the model based on BERT sentence embeddings achieves higher diversity in ASPEC dataset, which contains only formal texts. However, it fails to deliver similar results in WMT14 dataset, which is more informal. This may be due to the difficulty in clustering BERT vectors which were never trained to work with clustering. The model using FastText embeddings is shown to be more robust across the datasets, although it also fails to outperform the diverse decoding baseline in WMT14 dataset.

In contrast, syntax-based models achieve much higher diversity in both datasets. We found the results generated by this model has more diverse structures rather than word choices. By comparing the BLEU scores, no significant degradation is observed in translation quality. As a control experiment, using random codes does not contributes to the diversity. As a confirmation that the sentence codes have strong impact on sentence generation, the models using codes derived from references (oracle codes) achieve much higher BLEU scores.

6.5 Analysis and Conclusion

Table 6.3 gives samples of the candidate translations produced by the models conditioning on different discrete codes, compared to the candidates produced by beam search. We can see that the candidate translations produced by beam search has only minor grammatical differences. In contrast, the translation results sampled with

	Tg 以上 の 温度 で I を 消去 できた。 (Japanese)
A	1. It is possible to eliminate I at temperatures above Tg . 2. It is possible to eliminate I at temperatures higher than Tg . 3. It is possible to eliminate I at the temperature above Tg .
B	1. above Tg , I was able to be eliminated . 2. It was found that the photoresists were eliminated at temperatures above Tg . 3. at the temperature above Tg , I was able to be eliminated .
C	1. I could be eliminated at temperatures above Tg . 2. I was removed at temperatures above Tg . 3. It was possible to eliminate I at temperatures above Tg .

Table 6.3 A comparison of candidates produced by beam search (A), semantic coding model based on BERT (B) and syntactic coding model (C) in Ja-En task

the syntactic coding model have drastically different grammars. By examining the results, we found the syntax-based model tends to produce one translation in active voice and another in passive voice.

We also observe that the resultant diversity is higher in the Ja-En task as the target-side structures are less relevant to the input sentences.

To summarize, we show a diverse set of translations can be obtained with sentence codes when a reasonable external mechanism is used to produce the codes. When a good syntax parser exists, the syntax-based approach works better in terms of diversity. The source code for extracting discrete codes from parse trees will be publicly available.

Chapter 7

Summary

7.1 Contributions and Insights

Contributions In this thesis, we present a study to show that the latent-variable models are capable of learning multiple levels of linguistic features to benefit natural language generation models such as neural machine translation. The chapters of this thesis are organized according to the framework we use to train the latent-variable models.

In the first contribution, by learning sentence-level features using a generative model with continuous latent variables, we show the resultant model can be applied to obtain translation results with comparable quality comparing to autoregressive baseline models. We propose a deterministic inference procedure for this type of model to update the prediction of latent variables. On WMT'14 English-to-German translation task, the proposed inference algorithm helps to close the performance gap between the baseline model down to 1.0 BLEU point with a 6.8x faster decoding speed.

The second contribution aims to learn a compact representation using discrete latent variables. The proposed model represent the words with several discrete codes, where the codes and the code vectors are optimize simultaneously in a neural network. Such a joint training paradigm is enabled by Gumbel-softmax reparameterization trick for discrete latent variables. Experiments show that the compression rate achieves 98% in a sentiment analysis task and $94\% \sim 99\%$ in machine translation tasks without performance loss.

The third contribution takes a more challenging task of capturing syntactic structures of sentences. We tackle this problem by learning discrete latent variables to encode linguistic parse trees. In detail, we build a generative model with a pair of TreeLSTM-based encoder and decoder. The latent variables are learnt through the discretization bottleneck implemented by improved semantic hashing. Using this model, we are able to generate translations with drastically higher syntactical diversity.

Insights Through out the study, the experimental results provide insightful observations on applying latent-variable models to neural machine translation models. These insights may also be generalized to other natural language generation tasks. The study on non-autoregressive generation and diverse translation in chapter 4 and chapter 6 show that the latent variables can be applied to effectively control the text generation. In chapter 4, we show an example of improving the quality of generation by manipulating the latent variable prediction. In chapter 6, we show an example of controlling the syntactic structure of output sentences by imposing structural constraint using discrete latent variables. In both cases, the output sentences are generated from a decoder distribution $p(y|x, z)$. However, the two works adopt drastically different modeling strategies for controlling the generation process. In the first case, continuous latent variables are passed to the decoder as an additional condition, which is more suitable for non-autoregressive models. The second case treats the discrete latent variables as discrete symbols. By merging the artificially created discrete symbols with natural language text, the strategy naturally fits into the autoregressive model.

The study on word embedding compression in chapter 5 then demonstrate that applying discretization bottleneck in neural networks can create compact discrete representation of vectors. The major difference comparing to conventional quantization methods is that the neural networks learns the codes (i.e. assignment of quantized clusters) and the codebooks (i.e. vectors of codes) simultaneously through optimization. Thus it scales better comparing to conventional EM algorithms. Experiment results demonstrate superior performance of compression. Although the empirical experiments only cover the case on word embedding. The insights may also be applied to quantize the softmax embeddings other learned continuous vectors. If no empirical degradation of model performance is found, then such an additive quantization scheme trades computational complexity for memory footprint. To con-

struct word vectors, additive quantization requires to perform an advanced indexing on the codebooks, which may slow down the neural network computation depending on the platform.

7.1.1 A Latent-variable Framework for Neural Machine Translation

In chapter 4, we show a new framework of formulating and training neural machine translation models. It is a continuous latent-variable model trained directly with evidence lowerbound in variational method. The distinct contribution of this proposed model is that we show the target prediction can be significantly improved by sharpening the prediction in the latent space. Hence, we come to an important hypothesis, the quality of generated text in the token space depends on the quality of latent variables. In chapter 4, we empirically demonstrates refining the latent prediction by $1 \sim 3$ iteration improves the quality of generation. However, one may argue that it does not directly shows the correlation of the quality of latent variables and generated text.

To confirm the hypothesis, we create some interpolated variables by computing weighted average of the prior mean and variational density mean (i.e. mean of $q(z|x, y)$). In Fig. 7.1, we show the BLEU scores produced by different interpolated latent variables on ASPEC Ja-En translation task.

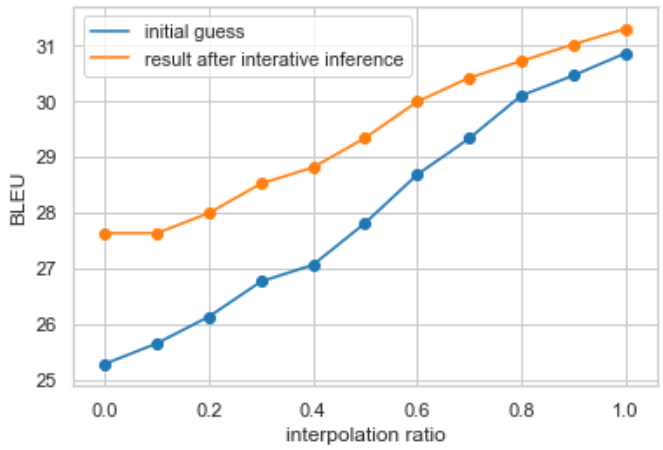


Fig. 7.1 BLEU scores on ASPEC Ja-En translation task produced by interpolated latent variables. The interpolation is created by computing weighted average of the prior mean and variational density mean.

When we move the latent variables from the prior mean to the center of the variational density, which indicates increasing quality in the latent space, we can see that the quality of generation improves monotonically. Therefore, this experimental result strongly confirms the hypothesis we mentioned above. Moreover, we also observe more upside gain which can be obtained by updating the latent prediction.

Following the result, we propose a hypothetic latent updating model $r(z'|z, x)$, which is an open opportunity for future researches to improve latent-variable text generation models. The model simply updates the latent predictions from prior to some areas in the latent space corresponding to higher-quality generation results. The model may be modeled more complicated approaches such as normalizing flow (Rezende and Mohamed, 2015). The advantage is that the computation can still be fast even advanced models are used. This is because the dimensionality of each latent variable is considerable small in our framework. In experiments, each continuous latent variable has only 8 dimensions.

7.2 On Interpretability of Latent Variables

In this thesis, we put our focus on learning latent variables to capture linguistic features and examine the application on neural machine translation. Therefore, much efforts are allocated to quantitatively evaluating the impact on translation results. We show with the latent-variables models, we can have a faster and more compact NMT model. We also show more diverse translations can be obtained.

However, we also notice the importance of analyzing the interpretability of learned latent variables. These variables might confirm the effectiveness of the learning model and provide further insights on the linguistic features. With the wide adoption of neural networks in natural language processing, the low interpretability of the highly non-linear model family is becoming problematic for computational linguistics. Without a straight-forward way to unbox the learned model and perform analysis, researchers generally struggle to check whether the neural networks correctly learns the linguistic feature of interest.

As part of the contribution of this thesis, we show that the latent variables produced by the model can indeed capture rich and meaningful linguistic features, forming interesting structures in the latent space.

7.2.1 Latent Variables for Sentence Generation

When we use a latent-variable model to directly generate sentences such as the example of non-autoregressive translation, the target sentence is generated with the decoder distribution $p(y|x, z)$. When we constrain the number of latent variables $|z|$ to be identical with the number of source tokens $|x|$, it indicates that each source word x_i is associated with a latent code z_i. Please refer to Fig. 1.7(c) for an illustration of the model. To understand what information do the latent codes capture, we randomly sample the codes from the prior distribution $p(z|x)$ and show the resultant translations in the following snippet:

```
Source: ＤＥＲＳ ソフトウエア を 用い て 「 ふげん 発電 所 」 の 線量
率 を 詳細 に 計算 でき る 。
Result 1: using the DERS software , the dose rate of ‘‘ Fugen plant ’’
can be calculated in detail .
Result 2: using DEDERS software , the dose rate of ‘‘ nuclear plant ’’
can be calculated in detail .
Result 3: using DEDE software software , the dose rate of ‘‘ nuclear
plant ’’ can be calculated in detail .
```

Here, we pick the first sentence in ASPEC test dataset as the example. We can see by randomly choose different latent codes, the NMT model gives different variations of translating the phrases “DERS” and “FUGEN Power Plant”.

As each code is associated with a word in our model, we further conjecture that each code controls the translation of the word it associates with. More concretely, we conjecture that changing the latent codes corresponding to “ふげん 発電 所” will alternate the translation of this specific phrase but not other parts. To perform the experiment, we sample a latent code $z_1, \ldots, z_n$ as our base code. Then as “ふげん 発電 所” are the 7th, 8th and 9th words in the sentence, we further alternate z_7, z_8, z_9 by sample from $p(z_7|x), p(z_8|x), p(z_9|x)$. The latent codes in other locations are kept unchanged in this process. The resultant translation results are shown in the following snippet:

```
Result 1:  using the DERS software , the dose rate of ‘‘ Fugen plant ’’
can be calculated in detail .
Result 2:  using DERS software , the dose rate rate ‘‘ Fugen power
plant ’’ can be calculated in detail .
Result 3:  using DEDERS software , the dose rate of ‘‘ nuclear plant ’’
can be calculated in detail .
```

We first observe the translation results contain more variations of ‘ふげん 発電 所”, which partially confirmed our hypothesis that each latent code controls the translations of their corresponding tokens. However, the selective latent code sampling did not prevent the translations of “DERS” to change. Therefore, the codes in the model are not perfectly disentangled.

Finally, we are curious to see how the translation changes when we move the latent variable in the latent space. We examine the results by interpolating the latent codes corresponding to two very different translations y_1 and y_2. The latent code can be obtained using the Q distribution $q(z|x, y)$. The interpolation can be done by computing $z_{\text{interpolate}} = rz_1 + (1 - r)z_2$, where r is the ratio of interpolation. We show the results in the following snippet:

```
Soruce:  リサイクル に 関 する 最近 の 話題 を 紹介 し た 。
Result  1 :  this paper introduces recent topics on recycling .
Ratio=0.20:  this paper introduces recent topics on recycling .
Ratio=0.24:  recent paper introduces recent topics on recycling .
Ratio=0.28:  recent paper introduces recent recycling are recycling .
Ratio=0.30:  recent topics introduces the recycling are recycling .
Ratio=0.32:  recent topics on the recycling are introduced .
Result  2 :  recent topics on the recycling are introduced .
```

As a result, we observe that the transformation happens with an interpolation ratio in the range $0.2 \sim 0.32$. Different part of the translations are interchanged during this process. To summarize the analysis presented in this section, we can see that the latent variables in our proposed translation model capture the utterance-level translations for each source token.

7.2.2 Qualitative Analysis of Discrete Word Codes

When learning word-level discrete variables, the model produces a set of codes for each word. By changing the hyperparameters, we force the model to learn 3 x 256 codes to quantize each word vector. In this setting, the discrete latent variable for each word is a composition of three numbers, each number can take a value from 1 to 256. We show the learned codes for some example words in the following snippet:

```
Man:   210 153 153   Woman:  232 153 153
King:  210 180 39   Queen:   232 180 39
Male:  208 11  219   Female:   232 11 219
```

We can observe that the code for "Man" and "Woman" has a common part "`153 153`". "King" and "Queen" share a common partial code "`180 39`". We thus conjecture that the former partial code can be the concept of man/woman, whereas the latter partial code is the concept of king/queen. Both "Man" and "King" share the first partial code "`210`", whereas "Woman" and "Queen" share the code "`232`" in the first position. Therefore, it is highly likely that "`210`" is a male concept and "`232`" is then a female concept.

However, such approach has a limitation. The analysis is feasible only when each word contains no more than three concepts and the model is able to disentangle them into three partial codes. However, this assumption does not hold generally. When a word has more than three concepts, the code learning model is forced to represent two or more concepts using one code. If multiple concepts are entangled, then such analysis cannot provide meaningful insights. In this thesis, we give a detailed discussion on the entanglement problem in section 7.3.2.

7.2.3 Discrete Coding for Syntactic Structures

In this thesis, we also describe a TreeLSTM-based model to perform discrete coding for syntactic structures. Similar to the word-level quantization model, we are interested in the latent structure captured by the model. After training the coding model, we obtain a discrete syntax encoder $f_{\text{enc}}(S_y; \theta)$, where S_y is the syntactic structure of the given sentence y. In our experiments, we adopt Stanford parse trees for structural representations. As a discrete encoder, the output of $f_{\text{enc}}(\cdot)$ is a K-bits binary vector.

We can also understand that the model clusters the syntactic representations into 2^K groups. Please note that the analysis in section is performed on a pure syntax coding model. However, the diverse translation model described in chapter 6 has a different form: $f_{\text{enc}}(S_y, x; \theta)$, which conditions also on the source sentence x to increases the efficiency of encoding.

Latent Code	Tree 1	Tree 2	Tree 3
1	(S S NP VP .)	(S ADVP NP VP .)	(S CC NP VP .)
31	(FRAG NP , PP .)	(NP DT NN SBAR .)	(FRAG NP , SBAR .)
61	(FRAG ADJP .)	(FRAG PP .)	(NP : NP PRN : NP .)
91	(X NP NP .)	(NP NP NP .)	(PP IN NP)
131	(S PP , INTJ VP .)	(S PP NP , VP .)	-
161	(S PP NP VP .)	(S SBAR NP VP .)	(S PP NP VP VP .)
191	(S PP , VP .)	(FRAG PP .)	(FRAG SBAR .)
231	(S NP VP .)	(S NP VP)	(S NP VP '' .)

Table 7.1 Most frequent parse trees in each clusters as a result of training the syntactic coding model with 256 clusters. We show the linearized form of top two levels of each tree.

Clustering Syntactic Structures To analyze the hidden structure captured by the model, we train a syntactic coding model on the English side corpus of WMT'14 that produces 256 clusters (i.e. latent codes). Then we analyze the most frequent tree structures in each cluster. The results are shown in Table 7.1.

For simplicity, we show the linearized form of top two levels in the parse trees. For example, `(S NP VP .)` indicates a parse tree with "S" as the root node, and three non-terminal nodes under it ("NP", "VP" and "."). By qualitatively analyzing the clustering results, we can see that the parse trees belong to each cluster tend to have similar global structures. For example, the top three parse trees grouped to cluster 1 have a common "`(S * NP VP .)`" structure.

To analyze the underlying vector space before discretization, we plot the continuous vector of each parse tree in the last hidden layer of $f_{\text{enc}}(S_y; \theta)$. The result is illustrated in Fig. 7.2. We use t-SNE to project the 256-dimensional vectors to the 2D space. Similar to the clustering experiment, we label each parse tree with the linearized form of top two levels. In Fig. 7.2, each dot is a projection of a parsee tree. Only 80 most frequent labels are shown in the figure. We can see that the model

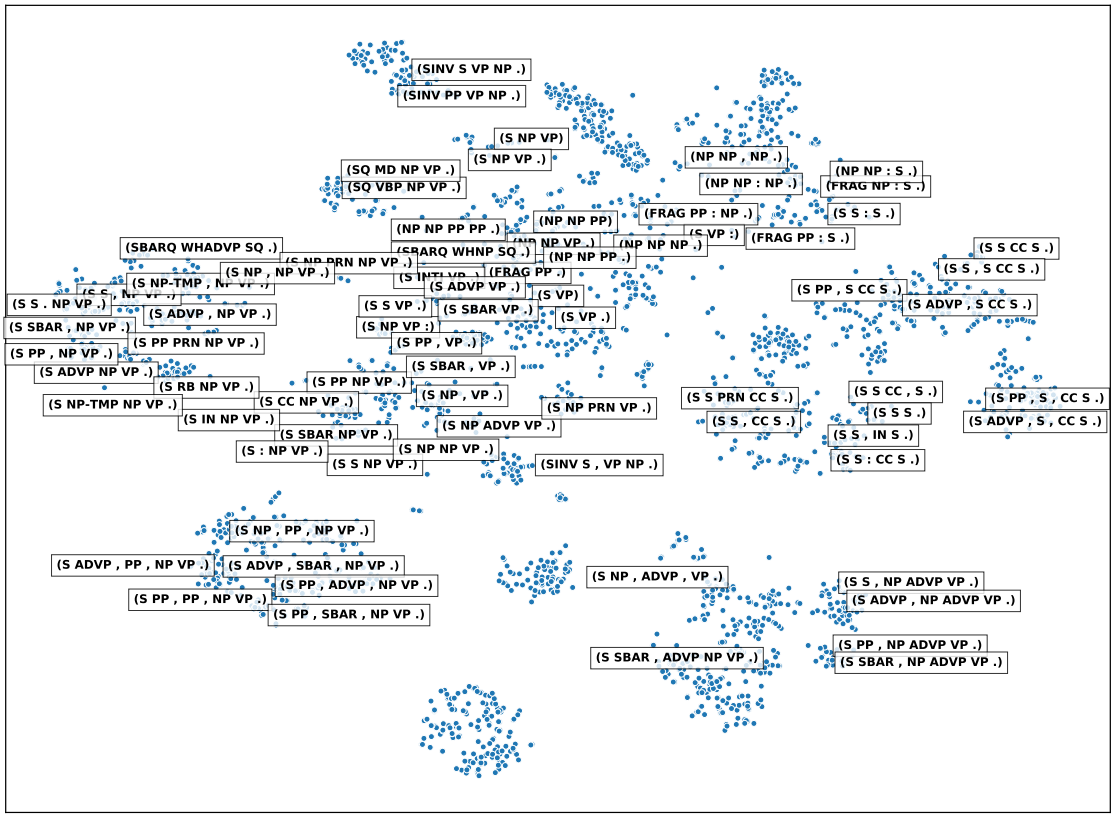


Fig. 7.2 Learned vector space of syntactic structures projected onto 2 dimensions by t-SNE. The labels for 80 most frequent trees are shown in the figure.

forms a certain structure in the hidden vector space. Similar trees are close to each other. For example, trees with a "NP" root are all located in the center of the figure. Trees with a "SQ" root are all located in a small cluster in the right side. Through the analysis, we confirmed that the model is capable of forming a interpretable latent space even for syntactic structures.

7.3 Discussions

7.3.1 Modeling Strategies of Latent-Variable Models

Comparison with Continuous Latent Variables The unique property of discrete bottleneck is the controllable information capacity. When we force a latent variable to be sampled from a categorical distribution with K categories, we know

that the amount of information counts up to $\log_2 K$ bits. This property also indicates that it takes $\log_2 K$ bits to store each latent variable.

	Continuous	Discrete
Posterior	Gaussian	Categorical
Backpropagation	Reparam. (Kingma and Welling, 2014)	Gumbel Softmax+[1]
Optimization	Easy	Difficult
Capacity	High	Low
Storage Efficiency	Low	High
Symbolic	No	Yes

Table 7.2 A high-level comparison of continuous and discrete latent-variable models.

In Table 7.2, we give an high-level comparison of continuous and discrete latent-variable models. Due to the limited capacity and difficulty in the training, we usually do not consider using discrete latent variables when our interest is maximizing the evidence lowerbound. Rather, we train discrete latent variables in order to obtain the discrete encoder, which is usually corresponding to the variational density $q(z|x)$ in the VAE setting.

Training Recall that we optimize the following evidence lowerbound (ELBO) when training generative models:

$$\text{ELBO}(x, q) = \mathbb{E}_q\big[\log p(x|z)\big] - \text{KL}(q(z|x)||p(z)). \tag{7.1}$$

When $q(z|x)$ is a categorical distribution with K categories and $p(z)$ is a uniform distribution, we can find that $\text{KL}(q(z|x)||p(z)) = -H(q) + \log K$. Therefore the ELBO for the discrete latent-variable model is now

$$\text{ELBO}(x, q) = \mathbb{E}_q\big[\log p(x|z)\big] + H(q) \underbrace{- \log K}_{\text{constant}}. \tag{7.2}$$

From the equation, we can see that the ELBO is a reconstruction objective function with an entropy-based regularization term. Similar to the posterior collapse in the continuous case, we may find that the entropy term is too strong for regularization

[1] Discrete bottlenecks also include Gumbel sigmoid, semantic hashing and VQVAE as discussed in section 3.4.

during training. In the discrete case, this problem is even more severe as we know the reconstruction objective takes more time to fully optimize due to the low capacity in the discretization bottleneck. One easy workaround is to drop the entropy from the objective and only optimize for $\mathbb{E}_q\big[\log p(x|z)\big]$. Such a training strategy has a risk that the resultant categorical distribution $q(z|x)$ may be unbalanced. In our experiments in chapter 5, we monitor $H(q)$ during training to see whether $q(z|x)$ is extremely skewed.

7.3.2 Evaluating Performance of Discrete Bottleneck

For discrete latent-variable models, it is important to evaluate the performance of the discrete bottleneck. In contrast to the continuous case where the de-facto way of reparameterization is to apply the trick introduced in Kingma and Welling (2014), there are multiple variations of discrete bottleneck. The performance of the bottleneck directly impact the model performance.

The angle of evaluation varies depending on our model and purposes. Here, we categorize the problem settings into two scenarios: auto-encoder and conditional auto-encoder. The regular auto-encoder has a prior $p(z|x)$ and a decoder $p(x|z)$, whereas the conditional auto-encoder has a prior $p(z|x)$ and a decoder $p(y|x,z)$. In both cases, we may care the balance of discrete codes, which can be measured by the entropy of variational density $H(q)$ as mentioned. If the codes are highly concentrated in one category, or some categories are not being used, we may guess that the capacity of the latent variables is not fully exploited.

Evaluation of Entanglement in Discrete Auto-Encoder In the auto-encoder case, where the computational graph can be illustrated by $x \to z \to x$, we care two metrics: ELBO and extent of entanglement of z. ELBO is a direct measurement of model performance, whereas the entanglement indicates that whether each latent variable learns an independent concept.

Suppose the discrete latent variables $\mathbf{Z} = z_1, \ldots, z_N$ are sampled from multivariate categorical distribution. Each variable z_i is a discrete K-way one-hot vector. Measuring the extent of entanglement in $\mathbf{Z}$ naturally translates to the problem of investigating the correlation between arbitrary two latent variables in $\mathbf{Z}$. Hence, it is a problem of finding the correlation matrix of categorical variables. In Niitsuma

and Okada (2005), Niitsuma and Lee (2016)[2], we can find the derivation of the covariance of categorical variables from Gini's definition (GINI, 1971) of covariance, where the authors also show that the derived results can be used to interpret the mechanism of word2vec. The method described in Niitsuma and Okada (2005), however, requires computing singular value decomposition, and there is no study of deriving the correlation matrix using this approach. We left the investigation of this approach to future works.

In statistics, we find that it is a common practice to measure the correlation between two discrete variables using Cramér's V (Cramir, 1946), which is based on Chi-Square test of independence. For two K-way categorical variables a and b, let $N(a=i,b=j)$ be the number of co-occurrence of $a=i$ and $b=j$, $N(a=i)$ be the number of occurrence of $a=i$ and the total number of data samples be N. We first compute the Chi-Square statistic with

$$\mathcal{X}^2_{ab} = \sum_{i=1}^{K}\sum_{j=1}^{K} \frac{[N(a=i,b=j) - \frac{1}{N}N(a=i)N(b=j)]^2}{\frac{1}{N}N(a=i)N(b=j)}. \tag{7.3}$$

In our setting, Cramér's V is then computed by

$$V_{ab} = \sqrt{\frac{\mathcal{X}^2_{ab}}{N*(K-1)}}. \tag{7.4}$$

As a byproduct of using Chi-Square test, Cramér's V is a symmetric measurement of correlation, which means $V_{ab} = V_{ba}$. Therefore, we are not expecting to observe causal relations from the results.

Using Cramér's V measurement, we analyze a set of 8×8 word codes learned for all words in IMDB vocabulary. The word embeddings are obtained from Glove (Pennington et al., 2014a). We encode the word embeddings using Gumbel-Softmax bottleneck with the model we described in chapter 5. In Fig. 7.3, we show the correlation matrix, where the number in each cell is produced by calculating $V_{z_i z_j}$. We can observe that the correlation between two discrete variables falls in the range from 0.2 to 0.3 in most cases. The results show that the learned discrete latent variables have balanced correlations. However, it also reveals that the information contained in each variable is entangled with all other variables.

[2]A better version of the formal paper can be found in https://arxiv.org/pdf/0711.4452.pdf

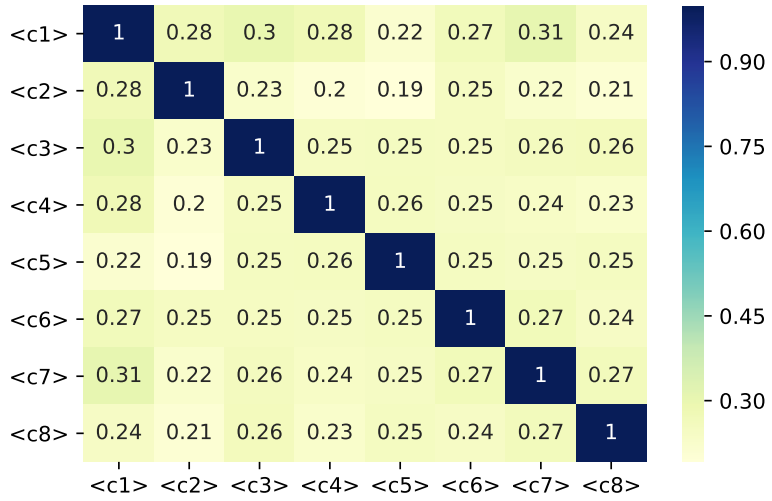


Fig. 7.3 Cramér's V correlation matrix produced by analyzing a set of 8x8 discrete codes for all words in IMDB vocabulary. The codes are produced by encoding using Gumbel-Softmax discretization bottleneck.

If we desire to learn independent discrete latent variables, we need to disentangle **Z** during the model training. Due to the discreteness, Cramér's V cannot be directly added to the loss function. However, we may consider to use REINFORCE algorithm to compute policy gradient to minimize the values of Cramér's V. We left further investigation of disentanglement to future works.

Evaluation of Conditional Discrete Auto-Encoder In the conditional discrete auto-encoder scenario, we learn latent-variable model for the conditional probability $p(y|x)$. With the discretization bottleneck, we produce latent variables with the variational density $q(z|x,y)$ and reconstruct the target using $p(y|x,z)$. An illustration of the computational graph is $x, y \to z \to y \leftarrow x$. The discrete latent variables in conditional auto-encoders learns distinct information, but not just encoding y. Let us first consider the case that the information capacity of z is infinite. That is, z can remember all information in both x and y. When we maximize the reconstruction objective $\mathbb{E}_{z\sim q(z|x,y)}\big[\log p(y|x,z)\big]$, z just need to remember y to perfectly reconstruct it.

However, when z is discrete, the information capacity of z is greatly limited. To maximize the reconstruction objective, the variational density $q(z|x,y)$ now has to

selectively remember partial information in y. What information will the $q(z|x, y)$ remember? As the decoder $p(y|x, z)$ has access to x, the best strategy is to remember information that exists in y but not in x. For example, in machine translation, z can capture information such as the syntactic structure or formalness in the target sentence y. Such information usually cannot be accurately predicted by only looking at x.

Now, we see that our purpose of learning conditional discrete auto-encoder is to force the latent variables z to capture information contained in y but not in x. We propose a simple metric for this purpose:

$$\text{METRIC}(q) = \min_{\phi} \max_{\theta} \log p_\theta(y|x, z_q) - \log p_\phi(y|x) - \log p_\phi(z_q|x). \tag{7.5}$$

Here, z_q are sampled discrete latent variables from $q(z|x, y)$. We train three probability models until they are fully optimized. The left-side term $\log p_\theta(y|x, z_q) - \log p_\phi(y|x)$ evaluates the extent that z_q contributes to the prediction of y. If z_q only captures information contained in x, this part of equation will be zero. $\log p_\phi(z_q|x)$ then evaluates the amount of information in z that is also contained in x. We want this value to be minimized so all the capacity in the latent variables can be used to capture things outside x. Note that because $\log p_\phi(z_q|x)$ is a log-probability on z but not y, we make add a weight α on this term to balance the metric.

7.4 Future Outlook: Learning Emergent Language

In chapter 6, we showed an application that use discrete latent variables for controlling text generation. Here, we articulate another important motivation for learning discrete representation, which is to learn an artificial language, which is referred to as *emergent language learning*. As natural languages can be seen as a collection of discrete symbols, the learned discrete representation can be naturally integrated into the vocabulary of natural languages.

To give a concrete example, we learn an independent code sequence that carries the same meaning of an original sentence in natural languages. The code sequence is supposed to have certain properties that we are interested in. For example, the code sequences can have a fixed length regardless of the original sentence length. Alternatively, it can have a small code vocabulary. Consider the following examples:

```
1.  The fox hiding behind the wall.
    <c16> <c31> <c62> <c86> <c42> <c14> <c15> <c26> <c74> <c26>
2.  He carefully went close to the wall and lean on it.
    <c74> <c33> <c26> <c73> <c53> <c85> <c76> <c67> <c48> <c63>
```

The sentences are encoded into code sequences with two constraints: (1) the code vocabulary contains 100 unique tokens, (2) each sequence is composed by 10 codes. If we can successfully learn such a emergent language, from which we can recover the original sentence, then it may benefit model training settings that have difficulty dealing with variable length. One example can be reinforcement learning based models.

Model Training and Evaluation In the aforementioned scenarios, the purpose of learning discrete codes is no longer obtaining descriptive latent variables, but to create an artificial language that satisfies specific requirements. Therefore, we are not required to formulate our code learning model as a generative model. The model is now trained solely to minimize task specific losses, where a discretization bottleneck is added to the computational graph. In the case of VAE, we can simply remove the KL-divergence term to let the model focus on minimizing the reconstruction loss.

One problem is that we are no longer able to evaluate the discretization methods with ELBO, which is the loss of VAE. To solve the problem, a recent work (Kaiser and Bengio, 2018) proposes to evaluate the effectiveness of the codes with the following metric:

$$\text{DSAE}(C) \propto \log_2(\text{pp}(X)) - \log_2(\text{pp}(X|C)), \tag{7.6}$$

where, the function $\text{pp}(X)$ computes the per-word perplexity of a sentence X with a neural language model. The perplexity has a low value when the model assigns a high probability to the sentence. Then, $\text{pp}(X|C)$ measures the per-word perplexity of the same sentences given the discrete codes C. Thus, Eq. (7.6) measures the improvement of the perplexity when conditioning on the codes. In other words, the discrete code has a high score if it can provide more information to recover the original sequence. The metric is referred to as *discrete sequence auto-encoding efficiency* (DSAE) in the Kaiser's paper.

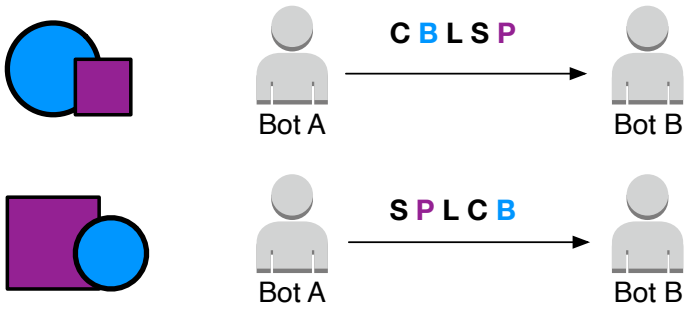


Fig. 7.4 Illustration of compositionality in emergent language during multi-agent communication. Here, /C B L S P/ renders the meaning of "A blue circle in the left of a purple square". /S P L C B/ means "A purple square in the left of a blue circle".

Applications Indeed, Kaiser and Bengio (2018) is already an application of emergent language learning. In the paper, the authors describe a discrete language encoding model that follows the conditional auto-encoder setting. Their model creates a emergent language to encode the target sentence while reducing the length by a factor of 8. Once the target sentence are completely encoded into artificially created codes, the authors then train an autoregressive sequence-to-sequence predictor for predicting the codes. Therefore, only eight generation steps are required in the autoregressive model, which results in faster generation speed.

To generalize this framework, we see that emergent language learning is helpful when we need to rewrite a sequence to comply specific constraints. Therefore, the applications in this setting can be developed in all NLP problems but not only machine translation.

7.4.1 Compositionality and Interpretability of Emergent Language

The problem becomes interesting when we want the artificially created language to be similar to human languages. Or even further, we may want to read and understand the artificial language created by neural networks. Such properties are sought after in mutlti-agent communications (Mordatch and Abbeel, 2017, Lazaridou et al., 2018, Chaabouni et al., 2019b,a).

The first property is known as *compositionality*, which is the key for human language to express infinite amount of meaning using a limited number of words. If the emergent language learned by our model can also express compositional structure,

the communication can be more efficient just like human languages. In Fig. 7.4, we demonstrate an example compositional language used by agents to communicate the color and position of objects. However, recent works point out that it is difficult for neural networks to naturally produce languages that expressing compositionality (Kottur et al., 2017, Baroni, 2019). As the first step of discussion, we have to find a way of evaluating the compositionality. Andreas (2019) proposes a tree-reconstruction measurement which calculates the distance between the model and a hypothetical model composes inferred representational primitive. Resnick et al. (2019) measures the compositionality using related entropy measured on the generated language. This work also shows the relation between model capacity and compositionality. When a model has limited capacity in the discretization bottleneck, it has a higher chance to develop compositional structure.

The second property is *interpretability* of emergent language. Such a property is important for human to understand the communication of neural networks. Both Kirby et al. (2015) and Chaabouni et al. (2019a) point out the trade-off between coding efficiency and interpretability. When we encourage the model to produce a storage-efficient representation, it tends to become a compression of encoded information and damage linguistic interpretability. Note the finding in Resnick et al. (2019) indicates reducing capacity, which essentially encourages coding efficiency, improves compositonality. We can see that it is considerable difficult to create a language that is both compositional and interpretable.

More recently, Lee et al. (2019) discusses the grounding problem of the learned emergent language. An obvious problem happens when we attempt to understand the meaning and compositionally of the emergent language. The meaning of each artificial symbol (or discrete code) does not carry the same meaning as that in natural language. For example, when we train two agents to communicate using human language, the agents can say "I'm hungry" but actually means "I'm moving forward". This is the meaning shift phenomenon. In the paper, the authors shows improvement of using visual information for grounding and counter the meaning shift problem. However, more generally, this is still an open problem when we try to learn an interpretable emergent language.

As the discrete representation learning is currently an active research topic, we are expecting to see more discretization methods and applications to be developed over time, especially in the scenario of learning an augmented or emergent languages.

7.4.2 Thoughts on Future Works for Natural Language Generation

Current natural language generation models and their training methods are large based on the original work of sequence-to-sequence learning proposed in 2014 (Sutskever et al., 2014). In this framework, the sequence generation model is essentially a conditional language model, or more formally, an autoregressive model. Autoregressive model is a typical solution for solving the structured prediction problem for sequence generation. In computer vision, image generation can be also considered as a structured prediction problem. The value of one pixel depends on other pixels. For example, for generated face images, the left eye and the right eye shall have the same color. However, except for PixelCNN (van den Oord et al., 2016), almost all recent state-of-the-art image generation models are not based on autoregressive models, but adversarial training (Goodfellow et al., 2014). Therefore, we can say that the NLP community and computer vision community were exploring the solutions for a single problem (i.e., structured prediction) with drastically different ways.

Hence, we have to consider this question: is autoregressive modeling the only solution for language generation task? If the answer is a "Yes", then what is the intrinsic reason? If the answer is a "No" or we cannot find a clear answer, then we have to think whether we can apply the knowledge acquired in training generative adversarial models to NLP. As highlighted in recent works seeking the connection between GANs and energy-based models (Finn et al., 2016), adversarial training can be treated as a method for minimizing the energy. In NLP community, energy-based models for language generation is largely underexplored. More future works can be done by training an energy-based model for solving the structured prediction problem, thus enabling fast non-autoregressive language generation.

Acknowledgements

I would like to acknowledge a number of people that supported or indirectly enabled the finish of this PhD work. First and most importantly, I would like to express gratitude towards my advisor in The University of Tokyo, Hideki Nakayama, for providing the opportunity to work in this beautiful campus and having the privilege of accessing the state-of-the-art computational resource. The utilization of such resources is costly especially in the early years when deep learning was booming. This PhD research cannot happen without the such a good environment and infrastructure in the lab. Hideki gave me great freedom from choosing research topics to decisions on paper submission. It greatly relieved me from the pressure of PhD course.

I would like to thank all collaborators. Thanks Hideki Nakayama for collaboration in all three works in this thesis. Thanks Kyunghyun Cho for collaboration on the topic in chapter 4 and chapter 6. Thanks Jason Lee for collaboration on the topic in chapter 4. Other than the co-authors, I also enjoyed and found it helpful when having conversation with members in Nakayama Lab. Special thanks go to Noriki Nishida for all NLPer discussions, Jiren Jin, Jin Zhang, Changhee Han for advising my presentations, Jan Zdenek for mixed conversations, Kento Masui, Rui Yang, Ryuichiro Hataya for giving an alternative research perspective from computer vision, Ikki Kishida, Masaki Baba for supporting from the hardware side, and Hong Chen. Also, thanks Weiyang Zhang for advices on coding interviews.

In the last year of my PhD, I was given the opportunity by prof. Kyunghyun Cho to collaborate with him and his students in New York University for around eight months. It was a great pleasure and an enjoyable process to work and talk to him. His advices were actionable and backed by sound theoretical proof. Even though I was a visiting student, he sometimes walk into my office and then we will brainstorm several research ideas. Special thanks go to Jason Lee, the collaboration with him directly enabled the research outcome of the non-autoregressive neural machine translation in chapter 4.

I would like to give thanks to the founder of Weblio Inc., Naoya Tsujimura for financially supporting my PhD. As the leader of a successful internet venture, maintaining one of top-50 most popular websites in Japan, he has no fear over exploring new technologies and business ideas. Even though I was young when joining the company, I had privilege to have lunch and dinner together with Naoya and discuss a variety of ideas freely. Backed by his great support, I was able to explore these new ideas using almost 100% of my working time. One of these idea was machine translation based on neural networks.

I would also like to thank for my wife, Gayoung Kim. We started our marriage life at the same time of the beginning of PhD. This thesis cannot be finished without her support and encouragement. Finally, I would like to conclude this PhD thesis with Romans 11:33, in King James Version it reads "O the depth of the riches both of the wisdom and knowledge of God! how unsearchable are his judgments, and his ways past finding out!". Back to 2015, although perusing the PhD in The University of Tokyo was not my plan until almost the last moment, the Lord responded to my prayer in the campus. He guided me to follow his plan by shutting down all other options. Finally, I was able to witness that his leading is beyond imagination. This PhD course really benefited me a lot not only in a worldly sense.

References

Aharoni, R. and Goldberg, Y. (2017). Towards string-to-tree neural machine translation. In *ACL*.

Andreas, J. (2019). Measuring compositionality in representation learning. *ArXiv*, abs/1902.07181.

Anwar, S., Hwang, K., and Sung, W. (2015). Fixed point optimization of deep convolutional neural networks for object recognition. *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1131–1135.

Arulkumaran, K., Creswell, A., and Bharath, A. A. (2017). Improving sampling from generative autoencoders with markov chains. *CoRR*, abs/1610.09296.

Auli, M. (2009). Ccg-based models for statistical machine translation.

Babenko, A. and Lempitsky, V. S. (2014). Additive quantization for extreme vector compression. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 931–938.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*.

Banchs, R. E. and Li, H. (2011). Am-fm: A semantic framework for translation quality assessment. In *ACL*.

Baroni, M. (2019). Linguistic generalization and compositionality in modern artificial neural networks. *ArXiv*, abs/1904.00157.

Bengio, Y. (2013). Estimating or propagating gradients through stochastic neurons. *ArXiv*, abs/1305.2982.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *ArXiv*, abs/1601.00670.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Bojar, O., Buck, C., Federmann, C., Haddow, B., Koehn, P., Leveling, J., Monz, C., Pecina, P., Post, M., Saint-Amand, H., Soricut, R., Specia, L., and Tamchyna, A. (2014). Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA. Association for Computational Linguistics.

Botha, J. A., Pitler, E., Ma, J., Bakalov, A., Salcianu, A., Weiss, D., McDonald, R. T., and Petrov, S. (2017). Natural language processing with small feed-forward networks. In *EMNLP*.

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., and Bengio, S. (2015). Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*.

Brown, P. F. and Piet, V. J. D. (2002). Hmm-based word alignment in statistical translation.

Cettolo, M., Niehues, J., Stüker, S., Bentivogli, L., and Federico, M. (2014). Report on the 11th iwslt evaluation campaign, iwslt 2014. In *Proceedings of the International Workshop on Spoken Language Translation, Hanoi, Vietnam*.

Chaabouni, R., Kharitonov, E., Dupoux, E., and Baroni, M. (2019a). Anti-efficient encoding in emergent communication. *ArXiv*, abs/1905.12561.

Chaabouni, R., Kharitonov, E., Lazaric, A., Dupoux, E., and Baroni, M. (2019b). Word-order biases in deep-agent emergent communication. In *ACL*.

Chen, T., Min, M. R., and Sun, Y. (2017). Learning k-way d-dimensional discrete code for compact embedding representations. *CoRR*, abs/1711.03067.

Chen, W., Wilson, J. T., Tyree, S., Weinberger, K. Q., and Chen, Y. (2015). Compressing neural networks with the hashing trick. In *ICML*.

Chiang, D. (2007). Hierarchical phrase-based translation. *Computational Linguistics*, 33:201–228.

Courbariaux, M., Bengio, Y., and David, J. (2014). Low precision arithmetic for deep learning. *CoRR, abs/1412.7024*, 4.

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1.

Cramir, H. (1946). Mathematical methods of statistics. *Princeton U. Press, Princeton*, page 500.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm.

Denil, M., Shakibi, B., Dinh, L., Ranzato, M., and de Freitas, N. (2013). Predicting parameters in deep learning. In *NIPS*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Devlin, J., Zbib, R., Huang, Z., Lamar, T., Schwartz, R. M., and Makhoul, J. (2014). Fast and robust neural network joint models for statistical machine translation. In *ACL*.

Dieng, A. B., Kim, Y., Rush, A. M., and Blei, D. M. (2018). Avoiding latent variable collapse with generative skip models. *CoRR*, abs/1807.04863.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14:179–211.

Eriguchi, A., Hashimoto, K., and Tsuruoka, Y. (2016). Tree-to-sequence attentional neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 823–833. Association for Computational Linguistics.

Eriguchi, A., Tsuruoka, Y., and Cho, K. (2017). Learning to parse and translate improves neural machine translation. In *ACL*.

Finn, C., Christiano, P. F., Abbeel, P., and Levine, S. (2016). A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *ArXiv*, abs/1611.03852.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. (2017). Convolutional sequence to sequence learning. *CoRR*, abs/1705.03122.

Gelfand, A. E. and Smith, A. F. M. (1990). Sampling-based approaches to calculating marginal densities.

Geman, S. and Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images.

Ghazvininejad, M., Levy, O., Liu, Y., and Zettlemoyer, L. S. (2019). Constant-time machine translation with conditional masked language models. *CoRR*, abs/1904.09324.

GINI, C. W. (1971). Variability and mutability, contribution to the study of statistical distributions and relations. studi cconomico-giuridici della r. universita de cagliari (1912). reviewed in : Light, r. j., margolin, b. h. : An analysis of variance for categorical data. *J. American Statistical Association*, 66:534–544.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. (2014). Generative adversarial nets. In *NIPS*.

Graves, A. (2013). Generating sequences with recurrent neural networks. *ArXiv*, abs/1308.0850.

Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., and Wierstra, D. (2015). Draw: A recurrent neural network for image generation. *ArXiv*, abs/1502.04623.

Gu, J., Bradbury, J., Xiong, C., Li, V. O. K., and Socher, R. (2018a). Non-autoregressive neural machine translation. *CoRR*, abs/1711.02281.

Gu, J., Bradbury, J., Xiong, C., Li, V. O. K., and Socher, R. (2018b). Non-autoregressive neural machine translation. In *Proceedings of the International Conference on Learning Representations 2018*.

Gu, J., Liu, Q., and Cho, K. (2019). Insertion-based decoding with automatically inferred generation order. *arXiv preprint arXiv:1902.01370*.

Gumbel, E. J. and Lieblein, J. (1954). Some applications of extreme- value methods.

Han, S., Mao, H., and Dally, W. J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*.

Han, S., Pool, J., Tran, J., and Dally, W. J. (2015). Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626.

Hassibi, B. and Stork, D. G. (1992). Second order derivatives for network pruning: Optimal brain surgeon. In *NIPS*.

Hastings, W. D. (1970). Monte carlo sampling methods using markov chains and their applications.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition supplementary materials.

Heafield, K., Pouzyrevsky, I., Clark, J. H., and Koehn, P. (2013). Scalable modified kneser-ney language model estimation. In *ACL*.

Hinton, G. (2012). Coursera video course of neural networks.

Hinton, G. E., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9:1735–1780.

Hu, Z., Yang, Z., Liang, X., Salakhutdinov, R., and Xing, E. P. (2017). Toward controlled generation of text. In *ICML*.

Hwang, K. and Sung, W. (2014). Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1. In *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*, pages 1–6. IEEE.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167.

Isozaki, H., Hirao, T., Duh, K., Sudoh, K., and Tsukada, H. (2010a). Automatic evaluation of translation quality for distant language pairs. In *EMNLP*.

Isozaki, H., Sudoh, K., Tsukada, H., and Duh, K. (2010b). Head finalization: A simple reordering rule for sov languages. In *WMT@ACL*.

Iyyer, M., Manjunatha, V., Boyd-Graber, J. L., and Daumé, H. (2015). Deep unordered composition rivals syntactic methods for text classification. In *ACL*.

Jain, U., Zhang, Z., and Schwing, A. G. (2017). Creativity: Generating diverse questions using variational autoencoders. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5415–5424.

Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *CoRR*, abs/1611.01144.

Jégou, H., Douze, M., and Schmid, C. (2011). Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:117–128.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine Learning*, 37:183–233.

Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., and Mikolov, T. (2016). Fasttext.zip: Compressing text classification models. *CoRR*, abs/1612.03651.

Kaiser, L. and Bengio, S. (2018). Discrete autoencoders for sequence models. *ArXiv*, abs/1801.09797.

Kaiser, L., Roy, A., Vaswani, A., Parmar, N., Bengio, S., Uszkoreit, J., and Shazeer, N. (2018a). Fast decoding in sequence models using discrete latent variables. *arXiv preprint arXiv:1803.03382*.

Kaiser, L., Roy, A., Vaswani, A., Parmar, N., Bengio, S., Uszkoreit, J., and Shazeer, N. (2018b). Fast decoding in sequence models using discrete latent variables. *CoRR*, abs/1803.03382.

Kalchbrenner, N. and Blunsom, P. (2013). Recurrent continuous translation models. In *EMNLP*.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. In *ACL*.

Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2016). Character-aware neural language models. In *AAAI*.

Kim, Y. and Rush, A. M. (2016). Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. In *NeurIPS*.

Kingma, D. P., Salimans, T., and Welling, M. (2016). Improving variational inference with inverse autoregressive flow. *CoRR*, abs/1606.04934.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. *CoRR*, abs/1312.6114.

Kirby, S., Tamariz, M., Cornish, H., and Smith, K. (2015). Compression and communication in the cultural evolution of linguistic structure. *Cognition*, 141:87–102.

Klein, D. and Manning, C. D. (2003). Accurate unlexicalized parsing. In *ACL*.

Koehn, P. (2004). Statistical significance tests for machine translation evaluation. In *EMNLP*.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., and Herbst, E. (2007). Moses: Open source toolkit for statistical machine translation. In *ACL*.

Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase-based translation. In *HLT-NAACL*.

Kottur, S., Moura, J. M. F., Lee, S., and Batra, D. (2017). Natural language does not emerge 'naturally' in multi-agent dialog. In *EMNLP*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *NIPS*.

Kudo, T. and Richardson, J. (2018). Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *EMNLP*.

Kulikov, I., Miller, A. H., Cho, K., and Weston, J. (2018). Importance of a search strategy in neural dialogue modelling. *CoRR*, abs/1811.00907.

Kulis, B. and Darrell, T. (2009). Learning to hash with binary reconstructive embeddings. In *NIPS*.

Lazaridou, A., Hermann, K. M., Tuyls, K., and Clark, S. (2018). Emergence of linguistic communication from referential games with symbolic and pixel input. *ArXiv*, abs/1804.03984.

LeCun, Y., Denker, J. S., and Solla, S. A. (1989). Optimal brain damage. In *NIPS*.

Lee, J., Cho, K., and Kiela, D. (2019). Countering language drift via visual grounding. *ArXiv*, abs/1909.04499.

Lee, J., Mansimov, E., and Cho, K. (2018). Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1173–1182.

Li, J., Monroe, W., and Jurafsky, D. (2016). A simple, fast diverse decoding algorithm for neural generation. *CoRR*, abs/1611.08562.

Lin, C.-Y. and Och, F. J. (2004). Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In *ACL*.

Liu, H., Wang, R., Shan, S., and Chen, X. (2016). Deep supervised hashing for fast image retrieval. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2064–2072.

Liu, W., Wang, J., Ji, R., Jiang, Y.-G., and Chang, S.-F. (2012). Supervised hashing with kernels. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2074–2081.

Liu, Y., Liu, Q., and Lin, S. (2006). Tree-to-string alignment template for statistical machine translation. In *ACL*.

Luong, T., Pham, H., and Manning, D. C. (2015). Effective approaches to attention-based neural machine translation. In *EMNLP*, pages 1412–1421.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning word vectors for sentiment analysis. In *ACL*.

Maddison, C. J., Mnih, A., and Teh, Y. W. (2016). The concrete distribution: A continuous relaxation of discrete random variables. *CoRR*, abs/1611.00712.

Martinez, J., Clement, J., Hoos, H. H., and Little, J. J. (2016). Revisiting additive quantization. In *ECCV*.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines.

Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 1045–1048.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *NIPS*.

Miller, A. H., Fisch, A., Dodge, J., Karimi, A.-H., Bordes, A., and Weston, J. (2016). Key-value memory networks for directly reading documents. In *EMNLP*.

Mnih, V., Heess, N. M. O., Graves, A., and Kavukcuoglu, K. (2014). Recurrent models of visual attention. In *NIPS*.

Mordatch, I. and Abbeel, P. (2017). Emergence of grounded compositional language in multi-agent populations. In *AAAI*.

Nadejde, M., Reddy, S., Sennrich, R., Dwojak, T., Junczys-Dowmunt, M., Koehn, P., and Birch, A. (2017). Predicting target language ccg supertags improves neural machine translation. In *WMT*.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *ICML*.

Nakazawa, T., Yaguchi, M., Uchimoto, K., Utiyama, M., Sumita, E., Kurohashi, S., and Isahara, H. (2016). Aspec: Asian scientific paper excerpt corpus. In *LREC*.

Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence o (1/k2). In *Doklady an SSSR*, volume 269, pages 543–547.

Neubig, G., Nakata, Y., and Mori, S. (2011). Pointwise prediction for robust, adaptable japanese morphological analysis. In *ACL*, pages 529–533.

Niitsuma, H. and Lee, M. (2016). Word2vec is a special case of kernel correspondence analysis and kernels for natural language processing.

Niitsuma, H. and Okada, T. (2005). Covariance and pca for categorical variables. In *PAKDD*.

Ott, M., Auli, M., Grangier, D., and Ranzato, M. (2018). Analyzing uncertainty in neural machine translation. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, pages 3953–3962.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2001). Bleu: a method for automatic evaluation of machine translation. In *ACL*.

Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Shazeer, N., Ku, A., and Tran, D. (2018). Image transformer. In *ICML*.

Pennington, J., Socher, R., and Manning, C. D. (2014a). Glove: Global vectors for word representation. In *EMNLP*.

Pennington, J., Socher, R., and Manning, C. D. (2014b). Glove: Global vectors for word representation. In *EMNLP*.

Post, M. (2018a). A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels. Association for Computational Linguistics.

Post, M. (2018b). A call for clarity in reporting bleu scores. In *WMT*.

Prabhumoye, S., Tsvetkov, Y., Salakhutdinov, R., and Black, A. W. (2018). Style transfer through back-translation. In *ACL*.

Raunak, V. (2017). Simple and effective dimensionality reduction for word embeddings. *CoRR*, abs/1708.03629.

Razavi, A., van den Oord, A., Poole, B., and Vinyals, O. (2019). Preventing posterior collapse with delta-vaes. *CoRR*, abs/1901.03416.

Resnick, C., Gupta, A., Foerster, J. N., Dai, A. M., and Cho, K. (2019). Capacity, bandwidth, and compositionality in emergent language learning. *ArXiv*, abs/1910.11424.

Rezende, D. J. and Mohamed, S. (2015). Variational inference with normalizing flows. *ArXiv*, abs/1505.05770.

Roy, A., Vaswani, A., Neelakantan, A., and Parmar, N. (2018). Theory and experiments on vector quantized autoencoders. *CoRR*, abs/1805.11063.

Salakhutdinov, R. and Hinton, G. E. (2009). Semantic hashing. *Int. J. Approx. Reasoning*, 50:969–978.

See, A., Luong, M.-T., and Manning, C. D. (2016). Compression of neural machine translation models via pruning. In *CoNLL*.

Sennrich, R., Haddow, B., and Birch, A. (2016a). Neural machine translation of rare words with subword units. In *ACL*, pages 1715–1725.

Sennrich, R., Haddow, B., and Birch, A. (2016b). Neural machine translation of rare words with subword units. *CoRR*, abs/1508.07909.

Shah, H. and Barber, D. (2018). Generative neural machine translation. In *NeurIPS*.

Shen, T., Lei, T., Barzilay, R., and Jaakkola, T. S. (2017). Style transfer from non-parallel text by cross-alignment. In *NIPS*.

Shen, T., Ott, M., Auli, M., and Ranzato, M. (2019). Mixture models for diverse machine translation: Tricks of the trade.

Shu, R. and Nakayama, H. (2018). Compressing word embeddings via deep compositional code learning. In *ICLR*.

Socher, R., Huang, E. H., Pennin, J., Manning, C. D., and Ng, A. Y. (2011a). Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in neural information processing systems*, pages 801–809.

Socher, R., Pennington, J., Huang, E. H., Ng, A. Y., and Manning, C. D. (2011b). Semi-supervised recursive autoencoders for predicting sentiment distributions. In *EMNLP*.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*.

Stahlberg, F., Hasler, E., Waite, A., and Byrne, B. (2016). Syntactically guided neural machine translation. *CoRR*, abs/1605.04569.

Stern, M., Chan, W., Kiros, J., and Uszkoreit, J. (2019). Insertion transformer: Flexible sequence generation via insertion operations. *arXiv preprint arXiv:1902.03249*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *NIPS*.

Tai, K. S., Socher, R., and Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. In *ACL*.

van den Oord, A., Kalchbrenner, N., Espeholt, L., Kavukcuoglu, K., Vinyals, O., and Graves, A. (2016). Conditional image generation with pixelcnn decoders. In *NIPS*.

van den Oord, A., Vinyals, O., and Kavukcuoglu, K. (2017). Neural discrete representation learning. In *NIPS*.

Vanhoucke, V., Senior, A., and Mao, M. Z. (2011). Improving the speed of neural networks on cpus. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, volume 1, page 4.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *NIPS*.

Wainwright, M. and Jordan, M. I. (2008). Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1:1–305.

Wang, Y., Tian, F., He, D., Qin, T., Zhai, C., and Liu, T.-Y. (2019). Non-autoregressive machine translation with auxiliary regularization. *CoRR*, abs/1902.10245.

Weiss, Y., Torralba, A., and Fergus, R. (2008). Spectral hashing. In *NIPS*.

Welleck, S., Brantley, K., Daumé III, H., and Cho, K. (2019). Non-monotonic sequential text generation. *arXiv preprint arXiv:1902.02192*.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. (2016). Learning structured sparsity in deep neural networks. In *NIPS*.

Wu, S., Zhang, D., Yang, N., Li, M., and Zhou, M. (2017). Sequence-to-dependency neural machine translation. In *ACL*.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Xia, R., Pan, Y., Lai, H., Liu, C., and Yan, S. (2014). Supervised hashing for image retrieval via image representation learning. In *AAAI*.

Xu, Q., Zhang, J., Qu, L., Xie, L., and Nock, R. (2018). D-page: Diverse paraphrase generation. *CoRR*, abs/1808.04364.

Yamada, K. and Knight, K. (2001). A syntax-based statistical translation model. In *ACL*.

Yang, H.-F., Lin, K., and Chen, C.-S. (2017). Supervised learning of semantics-preserving hash via deep convolutional neural networks. *IEEE transactions on pattern analysis and machine intelligence*.

Yin, P., Lyu, J., Zhang, S., Osher, S., Qi, Y., and Xin, J. (2019). Understanding straight-through estimator in training activation quantized neural nets. *ArXiv*, abs/1903.05662.

Zens, R., Och, F. J., and Ney, H. (2002). Phrase-based statistical machine translation. In *KI*.

Zhang, B., Xiong, D., and Su, J. (2016). Variational neural machine translation. In *EMNLP*.

Zhang, X., Chen, W., Wang, F., Xu, S., and Xu, B. (2017). Towards compact and fast neural machine translation using a combined method. In *EMNLP*.

Zhou, A., Yao, A., Guo, Y., Xu, L., and Chen, Y. (2017). Incremental network quantization: Towards lossless cnns with low-precision weights. *CoRR*, abs/1702.03044.