博士論文

Mechanisms for Combining Character and Word Level Representations in Natural Language Processing

(自然言語処理における文字レベルと単語レベルの表現を組み合わせるメカニズムの研究)

[指導教員　松尾 豊 教授]

東京大学大学院 工学系研究科
技術経営戦略学専攻

バラズ　テノ
ホルヘ　アンドレス　ジャン　ミシェル

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Meaning

In order to create truly intelligent machines, we need to teach them first how to use Language. Alas, what is the proof that humans themselves are intelligent, if not the use of Language? By Language I do not mean a specific instantiation of it such as the Japanese or Spanish languages, nor the modality with which they are used, such as writing, speaking, signaling, or touching. I mean the ability humans possess for manipulating symbols capable of communicating meaning.

A symbol, by definition, is something that can be *perceived* by the human senses – letters, ideograms, drawings, sounds, currency –, and *represents* something other than itself. "Meaning", however, is harder to define. As Sahlgren (2006) stated:

> *In one sense, everyone knows what meaning is* [...] *but in another sense, no one seems to be able to pin down exactly what this "meaning" is.*

I offer the following definition:

> Meaning is the agreed-upon representation of an aspect of reality shared by a group of humans.

Note that this agreement is *tacit*, meaning that humans did not explicitly chose how to map symbols to reality, and *emergent*, meaning that it raises from local interactions between humans, and cannot be controlled in a centralized way.

Indeed, Saussure (2013) defined Language as a system of *signs* composed of two elements: the *signifier*, which he described as a "sound image" or "sound pattern", and the *signified*, corresponding to the *psychological* concept related to it. He further suggested that the correspondence between signifier and signified, or symbol and the aspect it represents, was purely arbitrary. In other words, there is no underlying reason for any symbol to mean what it means, other than *convention* and *tradition*:

> *It is because the linguistic sign is arbitrary that it knows no other law than that of tradition, and because it is founded upon tradition that it can be arbitrary.* (p. 108)

This fact can be further abstracted to the systems we use for creating arrangements of symbols to create new concepts; not only the symbols are arbitrary but also the rules used for combining them into more complex ones. A simple example of this is that there are languages, such as Spanish, that use a Subject-Verb-Object (SVO) ordering for composing words into sentences, whereas others, such as Japanese, use a Subject-Object-Verb (SOV) ordering. Humans' ability to use Language seems to be independent of the arbitrary symbols and rules used in a specific instance of it.

1.2 Computational Models of Meaning

All this being said, how can we model meaning computationally? Sahlgren (2006) discusses the vector-space model, in which words are represented as vectors, and the spatial properties of these vectors represent information about their meaning (semantics). Simple algebraic operations on these vectors allow the creation of sentence and document representations useful for applications such as information retrieval, word-sense disambiguation, and machine translation, among others.

In the simplest realization of this model, every known word is represented as a vector containing 1 in a single dimension, and 0s in every other dimension (this is also called a *one-hot encoding*). Different words will be represented by vectors containing a 1 in different dimensions. However, a drawback of this particular encoding is that the only semantic information it contains is that *different words mean different things*.

That is to say, any two different words will be equally different to any two other words. For example the difference between "house" and "home", when one-hot encoded, will be exactly the same as the difference between "house" and "deoxyribose", despite the first pair being semantically closer than the second.

Words can be related to other words in different ways. These relationships exist both when words are considered independently of context (*paradigmatic* or *associative* relationship), and when they are co-present in a sentence (*syntagmatic* relationship) (Saussure, 2013).

Paradigmatic relationships relate to the meaning of words themselves. Some examples include:

- *synonymy*: Words that have similar meaning, e.g., "ball" and "sphere".
- *antonymy*: Words that have opposite meanings, e.g., "happiness" and "sadness".
- *holonymy*: Represents a *has-a* relationship. For example "car" is a *holonym* of

"wheel" ("wheel" is a *meronym* of "car").

- *hyponymy*: Represents a *is-a* relationship. For example "cat" is a *hyponym* of "mammal" ("mammal" is a *hypernym* of "cat").

Syntagmatic relationships, on the other hand, concern changes in meaning rising from composing single words into sequences of words; for example the role a word plays in a phrase (*part of speech* (POS)), or how it interacts with neighboring words (*dependency relations*).

One-hot encodings are unable to capture this type of semantic information. How then can we create vector representations of words that encode a notion of meaning? Harris (1954) suggested that words with different meaning are used in different ways, resulting in different distributions:

> *difference of meaning correlates with difference of distribution.* (p. 156)

The previous statement is roughly equivalent to saying that words with similar distributions have similar meanings. This fact is known as the **distributional hypothesis**.

Methods exploiting the distributional nature of meaning for creating word-space models, nowadays commonly called word embeddings, evolved from using purely-statistical methods, from term frequency and inverse document frequency (TF-IDF) (Jurafsky and Martin, 2018), co-occurrence statistics (Schütze, 1993), to more recent neural models such as `word2vec` (Mikolov et al., 2013b) and contextualized word representations (Peters et al., 2018a).

What all these methods have in common is that they create a mapping between words and vectors that encode semantic information. The type of vectors they create fall into two categories: *sparse* and *dense* (Jurafsky and Martin, 2018). Classical methods such as TF-IDF create sparse vectors, with a number of dimensions close to the size of the vocabulary (tens of thousands), only a few of which are nonzero, and where each dimension represents a word of the vocabulary. Neural methods, on the other hand, create dense representations: low-dimensional vectors having around 300 dimensions, most of which are nonzero, and generally represent an uninterpretable quantity. These vectors are also called *distributed representations*.

1.3 Neural Methods and Distributed Representations

Distributed representations are often defined in opposition to *local* representations (Hinton et al., 1986), also called *symbolic* representations (Goodfellow et al., 2016). In local representations, each feature represents a single property of the element being

represented; one-hot vectors being the canonical example. On the other hand, in distributed representations a single feature can represent several properties, and a single property can be represented by several features. Further, they are called distributed, because the concept they encode is spread out in a computational network, as opposed to condensed in a single computing unit.

An obvious disadvantage of preferring distributed rather than local representations is the added difficulty when trying to interpret what each feature encodes; nowadays we even have specific workshops devoted to interpreting neural networks and the representations they create[1]. However, the benefits of using distributed representations outweighs the downsides. Distributed representations are capable of efficiently using the processing abilities of neural networks, resulting in the appearance of beneficial emergent properties, such as being able to capture the similarity between concepts according to the similarity of their representations, and automatically generalizing (Hinton, 1986).

1.4 Broader Impact

The ultimate application of Natural Language Processing (NLP) is arguably an agent that can communicate naturally. This agent could either work to solve user requests in a manner similar to how a personal assistant would, or it could pose as another human for both purpose-driven and open-ended conversations.

Spoken or written communication, however is just an interface for transmitting knowledge to and from humans. This interface represents the highest level of abstraction of a hierarchical structure of sorts, since in order to have natural conversations the agent must first solve a plethora of other problems to simulate understanding and knowledge. In order for there to be anything to communicate, there have to be mechanisms able to extract and generate information.

These mechanisms can be broadly categorized as Natural Language Understanding (NLU) and Natural Language Generation (NLG). NLU deals with all the machinery necessary for encoding natural language into a machine-actionable representation, whereas NLG attempts to solve the problem of decoding a machine representation into human language.

The latest surge of interest in NLP has been caused, in part, by the migration from manually engineering these representations, or feature vectors, to manually engineering the neural architectures capable of learning them. The new ultimate desideratum therefore is to gain a deeper understanding of how to create systems capable of producing good representations.

[1] https://blackboxnlp.github.io/

Representations are good when they provide good performance in the end task, naturally, and when they are invariant to the task at hand, or in other words, generalizable. The recent success of pre-trained language models is due precisely to this reason; the contextualized representations they produce are easily adaptable to all kinds of downstream NLU tasks, while providing unprecedented performance.

These pre-trained language models perform so well in fact, that they are beginning to uncover how prone models are to latch onto annotation artifacts and biases present in training data, sparking interest in studying how to prevent this.

Creating better representations would bring NLP systems closer to human performance, enabling limitless applications. In their ideal form these systems should be able to answer questions pertaining to any domain, from the mundane "Is it going to rain today?" to the more complex "What is the standard prophylactic regime of oral amoxicillin for dental procedures for male adults allergic to penicillin?" Consumers would be able to buy through these agents, get recommendations, and find items suiting their specific needs; while professionals in need of critical data such as medical doctors and heavy machinery operators would be able to quickly get the information they need.

These systems could also have a wide range of humanitarian applications. Artificial teachers could teach almost any topic one on one at almost zero cost, while adapting to the specific needs of each learner. The deepest benefits of such a teacher would be mostly felt by those without access to good education, giving them better opportunities and helping them bridge the income gap with those better off. Similarly, these agents would be deployed in areas with worse access to healthcare to advise health practitioners, and even non-professionals, in preemptive and immediate emergency care.

Making these systems perform closer to and better than humans would clearly have a deep economic impact, it would potentially save lives and probably make the world a better a place.

1.5 Creating Better Representations

Every improvement of a pre-existing machine learning or deep learning model, whether for NLP, Computer Vision (CV), or any other related field, is accompanied by an improvement of the representations it relies upon. In fact, one could argue that "creating better representations" and "improving a deep learning model" mean essentially the same.

Now the question we face is "how can we create better representations, in the context of Natural Language Processing?"

Recent Breakthroughs

The deep learning breakthrough that propelled the field into what it is today, the invention of AlexNet (Krizhevsky et al., 2012), was a paradigm shift concerning the way of creating representations of images; from manually crafted features, to an end-to-end Neural Network (NN) architecture possessing a Convolutional Neural Network (CNN) capable of learning the features by itself. CNNs, among other things, encode the knowledge that pixels in images do not occur uniformly at random, that local neighborhoods of pixels are often semantically related, and that representations should be translation and rotation invariant. If we were to use a NN composed of dense layers instead, the architecture not only would contain a significantly greater number of parameters but it would also ignore this knowledge and have to discover it on its own, for which there are no guarantees. The astonishing result of AlexNet on the ImageNet Challenge (Russakovsky et al., 2015), was a strong indicator that this new method, encoding these previously-mentioned inductive biases, was better at creating image representations than manually engineering them.

A similar breakthrough in NLP was the invention of pre-trained word embeddings. This breakthrough is often referred to as *word2vec*[2] (Mikolov et al., 2013a,b), and it consisted on training a NN to predict the context of a word (Skip-gram), or predicting a word given its context (Continuous Bag of Words (CBOW)). This task resulted in portable and highly performant word representations that could be used in totally different domains while still providing significant boosts in performance.

The principle guiding the creation of this system was the Distributional Hypothesis, or the fact that similar words occur in similar contexts. The same principle guides the most recent breakthrough in NLP: contextualized word embeddings, also known as pre-trained language models. These contextualized word embeddings are similar to traditional word embeddings in the sense that their final product is vector representations for words, but different in that they rely on a specific encoding architecture, and are trained for longer in bigger datasets. This makes them perform better than simple word embeddings, but with the downside of being less portable given their dependence on a specific implementation (Devlin et al., 2019; Howard and Ruder, 2018; Peters et al., 2018a; Radford et al., 2019; Yang et al., 2019).

In one of the most popular implementations of an architecture producing contextualized word representations, BERT (Devlin et al., 2019), the model was trained in a masked language model task, where it had to learn how to predict a masked word given its context, and whether a sentence followed from another sentence. Even though

[2]word2vec is the name of the repository containing the code implementing the ideas presented in the papers. It can be found here: `https://code.google.com/archive/p/word2vec/` and is mirrored in `https://github.com/tmikolov/word2vec`.

the authors do not mention the Distributional Hypothesis at all, the similarities with word2vec are clear.

From the inductive biases encoded by CNNs to the Distributional Hypothesis in word embeddings, the most recent breakthroughs in Deep Learning stem from simple ideas encoded into specific NN architectures.

Combining Different Modalities in NLP

In an attempt to find general methods for creating better word representations in NLP, I will investigate the interactions between different modalities for representing words, and their influence in the performance of several models in diverse tasks.

A "modality" in this context is simply a method for obtaining a computational representation of a word; in our case a vector representation (word representations are discussed in detail in Section 2.1). For example, word embeddings are a common modality for representing words, and are often implemented as a mapping between words and distributed word representations. I will refer to this modality simply as the "word modality". Representations obtained through this modality will be referred to as "word representations", or "pre-trained word representations" in case they were initialized from pre-trained embeddings such as GloVe (Pennington et al., 2014).

Another modality is creating word representations by modeling the interactions among patterns occurring within words, or in other words, learning word representations from "subwords", henceforth the "subword modality". The simplest instance of such modality is modeling the interactions between characters, where each character representation is stored in a lookup table, and word representations are built by aggregating them in some way (subword representations are discussed in detail in Sections 2.4 and 5.3). We will refer to this modality as the "character modality" and word representations built with it as "character-derived word representations". Other instances include modeling morphemes (Botha and Blunsom, 2014), character n-grams (Bojanowski et al., 2017), byte-pair encoded tokens (Sennrich et al., 2016), wordpieces (Devlin et al., 2019), or combinations of them (Kudo, 2018).

In this thesis I will focus in studying how character-derived word representations contribute to model performance in several tasks, first without controlling for the way in which the character and word modalities are combined; to understand how they perform in isolation (Chapters 3 and 4). Then I will experiment with different methods combining both character-derived and pre-trained word representations, and study how they impact performance in several NLP tasks (Chapters 5 and 6).

Particularly, in Chapter 2 I will explain some concepts necessary for understanding remainder of this work, separated into background knowledge for word representations,

modeling interactions between them, and aggregating them for producing sentence representations. This chapter will also introduce recent work incorporating the character modality in several scenarios.

In Chapter 3 I will study whether incorporating character-derived word representations into the Natural Language Inference (NLI) task has any benefits. We chose this task because it requires models to capture high-level semantic properties of sentences (Bowman et al., 2015). Therefore it will give us insights about how characters contribute to the overall meaning representation of a sentence.

In Chapter 4 I will study how an architecture that relies purely on character-derived word representations, and pre-trained in a self-supervised fashion, performs in implicit emotion classification with data from Twitter. To perform well in this task, models need to be capable of handling the inherently noisy data from the microblogging platform (Aisopos et al., 2012; Martínez-Cámara et al., 2014). This task will show us how well character-derived representations can help in handling the high number of alternative spellings, and uncommon words frequently found in this context.

In Chapter 5, I will investigate the role that different ways of combining character and word representations play in creating final word and sentence representations. Specifically, I will study how using gating mechanisms impacts word representation quality when combining character-derived and pre-trained word representations, measured with both intrinsic and extrinsic metrics.

Finally, in Chapter 6, I will test the hypothesis that word representations, no matter the modality for obtaining them, should be similar since they represent the same underlying meaning. To do so, I will apply adversarial regularization to force character-derived representations to be similar to pre-trained word representations, and will study how these new representations perform when compared to traditional ones.

Chapter 2

Technical Background

2.1 Word-Level Representations in NLP

Distributed word representations, also known as word embeddings, are vectors associated to words that encode syntactic and semantic information. This means that words that have similar meanings will be "close" in the embedding space, according to a certain distance metric. In GloVe embeddings (Pennington et al., 2014), for example, the nearest neighbors of the word `Japan`, according to the cosine distance, are `Tokyo`, `Japanese`, and `Korea`.

Moreover, arithmetic operations in this space often have an intuitive semantic equivalent:

$$\texttt{houses} - \texttt{house} + \texttt{dog} \approx \texttt{dogs}$$

In other words, the difference between vector representations can encode inflectional phenomena such as the plural, in the example above, or even more complex semantic relationships such as gender in the following:

$$\texttt{princess} - \texttt{prince} + \texttt{king} \approx \texttt{queen}$$

There are several methods for obtaining mappings from words to vectors, but I will focus only in neural-based ones, namely, those that rely in predicting the relationship between a word and its neighborhood. Mikolov et al. (2013a) proposed explicitly to learn how to represent words by either predicting a word given its neighborhood, method which they called CBOW, or by predicting a word given another word in the same sentence, referred to as Skip-gram. They called this method `word2vec`[1].

The skip-gram model is as follows. Given a vocabulary of words $\mathcal{V} = \{v_1, \ldots, v_{|\mathcal{V}|}\}$

[1] https://code.google.com/archive/p/word2vec/.

and a sequence of words $w_1, \ldots, w_n \subset \mathcal{V}$, we want to predict words in the vicinity of w_t for $t = 1, \ldots, n$. Borrowing the notation used by Bojanowski et al. (2017), let us define $\mathcal{C}_t$ as the context of word w_t, containing the k previous and k following elements of w_t, such that:

$$\mathcal{C}_t = \{w_{t-k}, w_{t-k+1}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+k-1}, w_{t+k}\}$$

Note that $\mathcal{C}_t$ will contain $|\mathcal{C}_t| = 2k$ elements. Predicting $c \in \mathcal{C}_t$ given w_t is equivalent to maximizing the probability $p(c|w_t)$. Usually independence is assumed between words, so we can say that predicting the whole context of word w_t amounts to maximizing the probability:

$$p(\mathcal{C}_t|w_t) = \prod_{c \in \mathcal{C}_t} p(c|w_t) \tag{2.1}$$

Further, computing sums is less expensive than computing multiplications, thus instead of directly maximizing Eq. (2.1), we instead maximize:

$$\log p(\mathcal{C}_t|w_t) = \log \prod_{c \in \mathcal{C}_t} p(c|w_t) \tag{2.2}$$

$$= \sum_{c \in \mathcal{C}_t} \log p(c|w_t) \tag{2.3}$$

Finally, solving the problem of predicting the context of each word in a sequence is equivalent to maximizing the average log probability of predicting a context $\mathcal{C}_t$ given word w_t:

$$\frac{1}{n} \sum_{t=1}^{n} \log p(\mathcal{C}_t|w_t) = \frac{1}{n} \sum_{t=1}^{n} \sum_{c \in \mathcal{C}_t} \log p(c|w_t) \tag{2.4}$$

In order to solve this problem we need to parameterize the probability $p(c|w_t)$ of observing a context word c around w_t. To do so, we represent each word $v_i \in \mathcal{V}$ as a parameter vector $\boldsymbol{v}_i \in \mathbb{R}^d$. Analogously we denote the vector representations of words w_t in a generic sentence as $\boldsymbol{w}_t \in \mathbb{R}^d$, and context words $c_j \in \mathcal{C}_t$ as $\boldsymbol{c}_j \in \mathbb{R}^d$. We also need to define a scoring function $s : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, such that $s(\boldsymbol{w}_t, \boldsymbol{c})$ returns a single scalar value as a proxy for the probability $p(c|w_t)$. This function is defined as the dot product of the vector representations of the two words being compared $s(\boldsymbol{w}_t, \boldsymbol{c}) = \boldsymbol{w}_t \cdot \boldsymbol{c}$. Finally, to make $s(\boldsymbol{w}_t, \boldsymbol{c})$ behave like an actual probability we normalize it by feeding

it to a softmax function:

$$p(c|w_t) = \text{softmax}(s(\boldsymbol{w}_t, \boldsymbol{c})) = \frac{\exp(s(\boldsymbol{w}_t, \boldsymbol{c}))}{\sum_{j=1}^{|\mathcal{V}|} \exp(s(\boldsymbol{w}_j, \boldsymbol{c}))} \tag{2.5}$$

The problem with this formulation, however, is that obtaining the denominator requires summing over the whole vocabulary $\mathcal{V}$ for each pair (w_t, c) which is computationally intensive. Instead, the problem of predicting context words for a single word is reframed as a set of binary classification tasks. A word in the vocabulary belongs to the positive class if it appears in the context of w_t, otherwise it belongs to the negative one. However, using the whole negative class would be, again, quite expensive, so instead we take a random sample $\mathcal{N}_{t,c}$ of negative examples from the corpus. Then, for each context word c the binary logistic loss is defined as[2]:

$$\log(1 + \exp(-s(\boldsymbol{w}_t, \boldsymbol{c}))) + \sum_{\boldsymbol{n} \in \mathcal{N}_{t,c}} \log(1 + \exp(s(\boldsymbol{w}_t, \boldsymbol{n}))) \tag{2.6}$$

Therefore for each context word associated with w_t the logistic binary loss will be:

$$\sum_{\boldsymbol{c} \in \mathcal{C}_t} \log(1 + \exp(-s(\boldsymbol{w}_t, \boldsymbol{c}))) + \sum_{\boldsymbol{n} \in \mathcal{N}_{t,c}} \log(1 + \exp(s(\boldsymbol{w}_t, \boldsymbol{n}))) \tag{2.7}$$

Finally, defining the binary logistic loss function as $l(x) = \log(1 + e^{-x})$, the overall loss for the corpus is:

$$\sum_{t=1}^{|\mathcal{V}|} \left[\sum_{\boldsymbol{c} \in \mathcal{C}_t} l(-s(\boldsymbol{w}_t, \boldsymbol{c})) + \sum_{\boldsymbol{n} \in \mathcal{N}_{t,c}} l(s(\boldsymbol{w}_t, \boldsymbol{n})) \right] \tag{2.8}$$

Minimizing Eq. (2.8) will thus make the word representations of words appearing frequently in similar contexts similar, and those of words appearing in different contexts dissimilar.

2.2 Modeling Interactions Between Words

The method described in Section 2.1 produces word representations based on the distributional hypothesis. That is, they encode semantic information as a by-product of encoding information about the neighborhoods in which words are often found. In this section we present the most widely-used methods for modeling syntagmatic relationships between words, i.e., the changes in meaning when in presence of other words.

[2]We make abuse of notation to refer to the vector representation of $n \in \mathcal{N}_{t,c}$ as $\boldsymbol{n} \in \mathcal{N}_{t,c}$.

2.2.1 Recurrent Neural Networks

Plain RNN

Recurrent Neural Networks (RNNs) are useful for modeling dependencies between elements of a sequence. For each element of the sequence they take two inputs: the current element to be modeled and the output of the previous time step. By repeating this operation for each element, this network effectively learns how to accumulate knowledge about the elements in the sequence.

Elman (1990), proposed allowing time to be represented by the effect it has on processing the elements of the input sequence, instead of treating it as an additional dimension of the input. Among other things, this formulation can obtain comparable representations of variable-length inputs (such as sentences).

Assume we have a sequence of words $w_1, \ldots, w_n$, represented by a sequence of vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ where $\boldsymbol{x}_i \in \mathbb{R}^{d_{in}}, \ \forall i$. The Elman RNN is defined as:

$$\boldsymbol{h}_t = \sigma_h(\boldsymbol{W}^h \boldsymbol{h}_{t-1} + \boldsymbol{U}^h \boldsymbol{x}_t + \boldsymbol{b}^h) \tag{2.9}$$

$$\tag{2.10}$$

Where $\boldsymbol{W}^h \in \mathbb{R}^{d_h \times d_h}$ and $\boldsymbol{U}^h \in \mathbb{R}^{d_h \times d_{in}}$ are trainable weight matrices; $\boldsymbol{b}^h \in \mathbb{R}^{d_h}$ is a trainable bias vector; and σ_h is an activation function. The initial state vector $\boldsymbol{h}_0$ is a hyperparameter of the model chosen by the user. The resulting state vector $\boldsymbol{h}_t \in \mathbb{R}^{d_h}$ for word w_t encodes paradigmatic information represented by $\boldsymbol{x}_t$ and syntagmatic information from past elements represented by $\boldsymbol{h}_{t-1}$. From this it follows that $\boldsymbol{h}_n$ encodes semantic information for the whole sentence.

Long Short-Term Memory Network

A downside of plain RNNs is that they do not perform well at modeling long-term dependencies between elements of the sequence. Further, these are difficult to train because of vanishing and exploding gradients when performing Back Propagation Through Time (BPTT). Hochreiter and Schmidhuber (1997) proposed a variation on the previously-described RNN model precisely to address this issue. They called their architecture "Long Short-Term Memory Networks" (LSTMs).

At an abstract level, LSTMs receive and return elements in the same format as RNNs. They receive a sequence of vectors as input, and return a sequence of vectors encoding both semantic information about each element and information about their interactions as output. Specifically, they follow the following equations:

$$\boldsymbol{g}_t = \tanh(\boldsymbol{W}^g \boldsymbol{h}_{t-1} + \boldsymbol{U}^g \boldsymbol{x}_t + \boldsymbol{b}^g) \tag{2.11}$$

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}^i \boldsymbol{h}_{t-1} + \boldsymbol{U}^i \boldsymbol{x}_t + \boldsymbol{b}^i) \tag{2.12}$$

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}^f \boldsymbol{h}_{t-1} + \boldsymbol{U}^f \boldsymbol{x}_t + \boldsymbol{b}^f) \tag{2.13}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{g}_t \tag{2.14}$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}^o \boldsymbol{h}_{t-1} + \boldsymbol{U}^o \boldsymbol{x}_t + \boldsymbol{b}^o) \tag{2.15}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t) \tag{2.16}$$

$$\tag{2.17}$$

Where $\boldsymbol{W}^i, \boldsymbol{W}^f, \boldsymbol{W}^g, \boldsymbol{W}^o \in \mathbb{R}^{d_h \times d_h}$, and $\boldsymbol{U}^i, \boldsymbol{U}^f, \boldsymbol{U}^g, \boldsymbol{U}^o \in \mathbb{R}^{d_h \times d_{in}}$ are trainable weight matrices; $\boldsymbol{b}^i, \boldsymbol{b}^f, \boldsymbol{b}^g, \boldsymbol{b}^o \in \mathbb{R}^{d_h}$ are trainable bias vectors; and $\odot$ corresponds to the element-wise product of vectors (Hadamard product). The sigmoid (σ) and hyperbolic tangent (tanh) functions are applied element-wise to each of their arguments.

$\boldsymbol{g}_t \in \mathbb{R}^{d_h}$ (Eq. (2.11)) corresponds to the same vector obtained by a plain RNN modulo the activation function, as shown in Eq. (2.9). $\boldsymbol{i}_t$ and $\boldsymbol{f}_t \in [0,1]^{d_h}$, called input and forget gates, correspond to sigmoidal gates controlling how much the previous cell state $\boldsymbol{c}_{t-1}$, and current pre-gated state $\boldsymbol{g}_t$, should influence the current cell state $\boldsymbol{c}_t \in \mathbb{R}^{d_h}$, conditioned on the current input element $\boldsymbol{x}_t$ and the previous output $\boldsymbol{h}_{t-1}$ (Eqs. (2.12) to (2.14)). $\boldsymbol{o}_t \in [0,1]^{d_h}$ corresponds to a gate for controlling how much of the activated cell state $\tanh(\boldsymbol{c}_t)$ should be returned (Eqs. (2.15) and (2.16)). The LSTM will finally return $\boldsymbol{h}_t$ for word w_t, which will correspond to a word representation roughly representing the same as one produced by a plain RNN, with the added guarantees of being easier to train, and capable of modeling long-range dependencies.

Bidirectional LSTM

At each time step, a normal LSTM only encodes information from previous elements, that is, $\boldsymbol{h}_t$ encodes information from $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{t-1}$. However some NLP applications, such as dependency parsing, require knowledge about future elements in the sequence. To achieve this goal Graves and Schmidhuber (2005) proposed the Bidirectional Long Short-Term Memory Network (BiLSTM).

A BiLSTM (Graves et al., 2013; Graves and Schmidhuber, 2005), is composed of two LSTMs, one that reads the input sequence from left to right (i.e., from w_1 to w_n), and produces the representations $\overrightarrow{\boldsymbol{h}_1}, \ldots, \overrightarrow{\boldsymbol{h}_n}$, and another that reads the input from right to left (i.e., from w_n to w_1), and produces $\overleftarrow{\boldsymbol{h}_1}, \ldots, \overleftarrow{\boldsymbol{h}_n}$, where each $\overleftarrow{\boldsymbol{h}_t}$ encodes information about $\overleftarrow{\boldsymbol{h}_{t+1}}, \ldots, \overleftarrow{\boldsymbol{h}_n}$. The former is often referred to as *forward* LSTM, and the latter as *backward* LSTM. The final output of the BiLSTM will be the concatenation of the

forward and backward outputs for each time step: $\boldsymbol{h}_t = [\overrightarrow{\boldsymbol{h}_t}; \overleftarrow{\boldsymbol{h}_t}]$, which will ensure that each $\boldsymbol{h}_t$ encodes information from both the past and future elements of the sequence.

2.2.2 Convolutional Neural Networks

Another way to encode sequence information into each sequence element is through the use of CNNs. Collobert and Weston (2008) proposed using a specific instance of CNN called Time Dilated Neural Network (TDNN) (Waibel et al., 1989), for achieving this. The output of a TDNN at time t is defined as:

$$\boldsymbol{h}_t = \sum_{j=1-t}^{n-t} \boldsymbol{L}_j \cdot \boldsymbol{x}_{t+j} \tag{2.18}$$

Where $\cdot$ corresponds to the dot product between vectors, and $\boldsymbol{L}_j \in \mathbb{R}^{d_h \times d_{in}} (-n \leq j \leq n)$ is a trainable parameter matrix ($\mathbf{L} \in \mathbb{R}^{2n \times d_h \times d_{in}}$ is a trainable parameter tensor).

The previous definition can be constrained by defining a *kernel size* or *window size* w to consider only a limited vicinity of each element instead of the whole sequence at each time step. Imposing a window size implies enforcing the following conditions:

$$|j| > \frac{(w-1)}{2} \implies \boldsymbol{L}_j = \mathbf{0} \tag{2.19}$$

$$\iff -\frac{(w-1)}{2} \leq j \leq \frac{(w-1)}{2} \implies \boldsymbol{L}_j \neq \mathbf{0} \tag{2.20}$$

$$\tag{2.21}$$

Which simplifies the convolution operation (Eq. (2.18)) to:

$$\boldsymbol{h}_t = \sum_{|j| \leq \frac{(w-1)}{2}} \boldsymbol{L}_j \cdot \boldsymbol{x}_{t+j} \tag{2.22}$$

For example, setting $w = 3$ would result in:

$$\boldsymbol{h}_t = \sum_{-1 \leq j \leq 1} \boldsymbol{L}_j \cdot \boldsymbol{x}_{t+j} = \boldsymbol{L}_{-1} \cdot \boldsymbol{x}_{t-1} + \boldsymbol{L}_0 \cdot \boldsymbol{x}_t + \boldsymbol{L}_1 \cdot \boldsymbol{x}_{t+1} \tag{2.23}$$

In general, setting a window size of w will result in a parameter tensor $\mathbf{L} \in \mathbb{R}^{w \times d_h \times d_{in}}$. The final output of the TDNN will be analogous to that of the RNNs: a sequence of vectors $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n$ encoding syntagmatic and paradigmatic information for each word $w_1, \ldots, w_n$.

2.3 Sentence-level Representations

It is also possible to represent whole sentences as vectors that encode a notion of meaning. As we saw in Section 2.2, RNNs and CNNs are useful for imbuing word-level vector representations with information about their context. However, the resulting sequence of vector representations will have a variable length depending on the amount of elements in the sequence. Dense (or classical) NNs though, are not able to deal with inputs of varying length (Collobert and Weston, 2008), thus the next question to ask is: How can we combine word representations to form meaningful sentence representations of fixed length?

Assuming we have a sequence of words $w_1, \ldots, w_n$, and their corresponding vector representations $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n$ where $\boldsymbol{h}_i \in \mathbb{R}^d \ \forall i$, there are several ways in which we can aggregate or *pool* them. The most straightforward is to use one of the `mean`, `sum` or `max` pooling operations over the sequence dimension. Another option when using unidirectional RNNs is to take the last hidden state $\boldsymbol{h}_n$ for representing the whole sentence. When using bidirectional RNNs the common practice is to concatenate the last hidden states of the forward and backward passes: $\boldsymbol{h}_n = [\overrightarrow{\boldsymbol{h}_n}; \overleftarrow{\boldsymbol{h}_1}]$. Using any of these methods will return a single vector $\boldsymbol{h} \in \mathbb{R}^d$ representing the whole sentence.

To illustrate how this can be achieved we will explain the work by Conneau et al. (2017), who attempted to create a universal sentence encoder by answering the following questions:

1. What is the best neural network architecture for achieving such task?
2. How should this network be trained?

Being universal means being general enough to be trained in a single dataset and performing well tasks and datasets for which it was not explicitly trained. They tested 4 different architectures trained on the Natural Language Inference task and showed that a BiLSTM followed by a max-pooling layer trained on a combination of SNLI and MultiNLI datasets performed better than the other methods.

In the following sections we will first introduce the NLI task and later present the sentence encoding architectures that Conneau et al. (2017) tested.

2.3.1 The Natural Language Inference Task

NLI, also known as Recognizing Textual Entailment (RTE), is a task designed for testing the ability of models to capture high level semantic properties of sentences. It consists in predicting whether a *premise* sentence and a *hypothesis* sentence are related by *entailment*, *neutral*, or *contradiction* relationships. If the hypothesis can be inferred from the premise we say that the premise entails the hypothesis. If the premise being

true means that the hypothesis cannot be true, then they contradict each other. If the truth value of the premise is independent of that of the hypothesis we say they are neutral to each other. Examples for each relationship include:

Entailment:

`Premise`: At the other end of Pennsylvania Avenue, people began to line up for a White House tour.

`Hypothesis`: People formed a line at the end of Pennsylvania Avenue.

Neutral

`Premise`: The new rights are nice enough.

`Hypothesis`: Everyone really likes the newest benefits.

Contradiction

`Premise`: This site includes a list of all award winners and a searchable database of Government Executive articles.

`Hypothesis:` The Government Executive articles housed on the website are not able to be searched.

To perform well at the NLI task a model has to be capable of capturing sentence meaning, which in turn requires handling lexical ambiguity, coreference, belief, tense and modality, among other phenomena (Williams et al., 2018). The SNLI dataset[3] (Bowman et al., 2015), was the first large scale NLI dataset, containing 570,000 examples; two orders of magnitude larger than the other NLI datasets at the time, making it ideal for training and testing NN models.

SNLI was later supplemented by MultiNLI[4] (Williams et al., 2018) which was created from several sources: fiction novels, government-related documents, magazine articles, telephone conversation transcripts, and travel guides. Moreover, it comes with two validation datasets, the *matched* validation dataset containing examples from domains matching the training data, and the *mismatched* validation set containing examples from five different domains: A 9/11 report, face-to-face conversation transcriptions, letters, non-fiction works, and articles on linguistics.

Additionally, the XNLI[5] dataset (Conneau et al., 2018b), was recently released as an additional validation set for MultiNLI for testing the cross-lingual abilities of NLI models.

Other recent examples include the dataset presented by Lai et al. (2017) where the task is to predict the relationship between several premises and a single hypothesis[6],

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

[4] https://www.nyu.edu/projects/bowman/multinli/

[5] https://www.nyu.edu/projects/bowman/xnli/

[6] At the time of writing, however, almost two years after the paper was published, the dataset is still not available.

and the SciTail dataset (Khot et al., 2018), created from multiple-choice science-related questions.

Predating SNLI, there was the SICK dataset[7] (Marelli et al., 2014), created from the Flickr-8k image caption dataset[8] (Rashtchian et al., 2010), and the MSR video description dataset[9]. Also worth mentioning: SNLI was created from the extension to Flickr-8k, the Flickr-30k dataset (Young et al., 2014).

2.3.2 Architectures

Conneau et al. (2017) performed their universal sentence encoder study on MultiNLI. This section briefly describes the architectures they experimented with.

LSTM and GRU

The first and simplest models Conneau et al. (2017) trained were an LSTM and a Gated Recurrent Unit (GRU) (Cho et al., 2014), where the final sentence representation corresponded to the last hidden vector $\boldsymbol{h}_n$ for a sequence of length n. They also tried using a Bidirectional Gated Recurrent Unit (BiGRU), which is analogous to a BiLSTM, and represented each sentence as the concatenation of the last forward and backward hidden states (see. Section 2.2.1 on BiLSTMs).

GRUs rely on the same gating mechanism concept found in LSTMs. They are defined by the following equations:

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}^r \boldsymbol{h}_{t-1} + \boldsymbol{U}^r \boldsymbol{x}_t + \boldsymbol{b}^r) \tag{2.24}$$

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}^z \boldsymbol{h}_{t-1} + \boldsymbol{U}^z \boldsymbol{x}_t + \boldsymbol{b}^z) \tag{2.25}$$

$$\boldsymbol{g}_t = \tanh(\boldsymbol{W}^g (\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) + \boldsymbol{U}^g \boldsymbol{x}_t + \boldsymbol{b}^g) \tag{2.26}$$

$$\boldsymbol{h}_t = \boldsymbol{z}_t \odot \boldsymbol{h}_{t-1} + (\boldsymbol{1} - \boldsymbol{z}_t) \odot \boldsymbol{g}_t \tag{2.27}$$

Where the output at time step t, $\boldsymbol{h}_t$, can be interpreted as containing information analogous to that produced by an LSTM.

BiLSTM with Mean and Max Pooling

This architecture is equivalent to the BiLSTM described in Section 2.2.1. To create a single sentence representation from the word-level outputs $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n$ returned by the BiLSTM, Conneau et al. (2017) tried max-pooling and mean-pooling them. Assuming

[7] http://clic.cimec.unitn.it/composes/sick.html
[8] http://nlp.cs.illinois.edu/HockenmaierGroup/8k-pictures.html
[9] https://www.cs.york.ac.uk/semeval-2012/task6/index.php%3Fid=data.html

we concatenate the word representations into a matrix $\boldsymbol{H} = [\boldsymbol{h}_1; \ldots; \boldsymbol{h}_n] \in \mathbb{R}^{d_h \times n}$, max-pooling corresponds to picking the maximum element across each row. In other words, each element $\boldsymbol{s}_j$ of the sentence representation $\boldsymbol{s} \in \mathbb{R}^{d_h}$ will be defined as: $\boldsymbol{s}_j = \max(\boldsymbol{H}_j)$, where $\boldsymbol{H}_j \in \mathbb{R}^n$ is the j-th row of $\boldsymbol{H}$. Mean-pooling is analogous: $\boldsymbol{s}_j = \text{mean}(\boldsymbol{H}_j)$.

Self-attentive Network

The self-attentive network uses a BiLSTM to produce context-aware word representations $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n$, and then defines the following operations for obtaining the sentence representation $\boldsymbol{s}$:

$$\boldsymbol{g}_t = \tanh(\boldsymbol{W}^g \boldsymbol{h}_t + \boldsymbol{b}^g) \tag{2.28}$$

$$z_t = \boldsymbol{g}_t \boldsymbol{u} \tag{2.29}$$

$$\alpha_t = \frac{\exp(z_t)}{\sum_j \exp(z_j)} \iff \boldsymbol{\alpha} = \text{Softmax}(\boldsymbol{z}) \tag{2.30}$$

$$\boldsymbol{s} = \sum_t \alpha_t \boldsymbol{h}_t \tag{2.31}$$

Where $\boldsymbol{W}^g \in \mathbb{R}^{d_u \times d_h}$, $\boldsymbol{b}^g \in \mathbb{R}^{d_u}$ and $\boldsymbol{u} \in \mathbb{R}^{d_u}$ are parameters, and $\boldsymbol{\alpha} \in [0, 1]^n$ can be interpreted as a probability distribution over the input sequence. Consequently this architecture learns a weighting scheme for each element of the input sequence conditioned on the other elements.

Hierarchical CNN

This architecture stacks L CNN layers (see Section 2.2.2), obtaining a representation $\boldsymbol{h}_t^l$ for each element of the sequence $l = 1, \ldots, L$. Each layer obtains a representation $\boldsymbol{H}^l = [\boldsymbol{h}_1^l, \ldots, \boldsymbol{h}_n^l] \in \mathbb{R}^{d_l \times n}$ which is later max-pooled for obtaining an aggregated representation of the input: $\boldsymbol{h}^l = \max(\boldsymbol{H}^l) \in \mathbb{R}^{d_l}$. The final sentence representation is the concatenation of these aggregated input representations: $\boldsymbol{s} = [\boldsymbol{h}^1, \ldots, \boldsymbol{h}^L] \in \mathbb{R}^{Ld_l}$.

Conneau et al. (2017) concluded that the BiLSTM with max-pooling was the best architecture overall. They called this model *InferSent*. I used this sentence encoding architecture throughout the work leading to the writing of this manuscript given its proven superior performance.

2.4 Subword-Level representations

Similar to how CNNs and LSTMs can be used for modeling contextual word information within sentences, they can also be used to model contextual *subword* information within words. The specific type of subword to use for a given problem or dataset is not trivial to choose, and is in and of itself an unsolved research problem (Kudo, 2018).

Besides representing phenomena occurring within words, subwords also allow us to represent words not seen during training. These unknown words are often referred to as Out-of-Vocabulary (OOV) or UNK words. Throughout this thesis we use both terms interchangeably. Below we describe a few works at the forefront of subword-level representations research.

Wu et al. (2016) use what they call "wordpieces," a set of common sub-word units, based on previous work by Schuster and Nakajima (2012). They claim their method provides a good balance between the flexibility of character-based models and the efficiency of word-based models, and is capable of naturally handling the translation of rare words, resulting in overall better performance.

Sennrich et al. (2016) propose encoding OOV words, and rare words as sequences of subword units (*byte-pair encoding*). They claim subword models achieve higher accuracy than large-vocabulary models and back-off dictionaries, and are capable of generating words not seen at training time. Further, these models are able to learn transliteration and compounding from subword representations.

Vulić et al. (2017) use a post-processing system (ATTRACT-REPEL) for refining word vectors based on a simple set of morphological constraints. They were able to beat the state of the art in the intrinsic evaluation task of semantic similarity of several pre-trained word vectors by using hand-crafted rules, and obtained good results in the downstream task of Dialogue State Tracking. Their method differs from the previously-mentioned works in that the Morph-fitting method is decoupled from the training process by incorporating morphological knowledge in the form of linguistic constraints, obtained from inflectional and derivational rules.

Avraham and Goldberg (2017) show that the base-form of words is related to semantic aspects of word similarity in embedding space, whereas affixes are related to their morphological similarity. Additionally they show that there is a trade-off between morphological and semantic performance.

There are also hybrid systems that combine both word and character-level representations. However, there is no theory or experimental data supporting any specific way of combining both hierarchies. For example Luong and Manning (2016) fall back to the last hidden state of a character-level RNN when they encounter an unknown word, thus ignoring character-level information for common words. Others just add the vector

representations at both hierarchies (Bojanowski et al., 2017; Botha and Blunsom, 2014) without justifying this choice.

In Section 5.3 we provide a more formal explanation of how to combine character subwords with word representations.

Chapter 3

The Importance of Characters and Self-Attention in the NLI Task

3.1 Abstract

In this chapter we tackle the task of Natural Language Inference. We use an architecture that separately encodes a pair of sentences into variable-length representations with a BiLSTM. Then, it creates a fixed-length raw representation by means of simple aggregation functions, which are later refined with a self-attention mechanism. Finally, it combines the representations for both the premise and hypothesis into a single vector to be classified. We also experimented with adding character-level information by concatenating character-derived and pre-trained word representations, however this proved not to be very helpful.

Our best model we obtained test accuracies of 72.057% and 72.055% respectively in the matched and mismatched evaluation tracks of the RepEval 2017 shared task, outperforming an LSTM baseline, and obtaining performances similar to a model that relies on shared information between sentences. When using an ensemble both accuracies increased to 72.247% and 72.827% respectively. Code for replicating this chapter is available at `https://github.com/jabalazs/repeval_rivercorners`.

3.2 Introduction

The task of Natural Language Inference (NLI) aims at characterizing the semantic concepts of entailment and contradiction, and is essential in tasks ranging from information retrieval to semantic parsing to commonsense reasoning, as both entailment and contradiction are central concepts in natural language meaning (Katz, 1972; van Benthem, 2008).

The aforementioned task has been addressed with a variety of techniques, including those based on symbolic logic, knowledge bases, and neural networks. With the advent of deep learning techniques, NLI has become an important testing ground for approaches that employ distributed word and phrase representations, which are typical of these models.

In this chapter we work exclusively with the MultiNLI dataset (Williams et al., 2018). As mentioned in Section 2.3.1, this dataset features two evaluation sets; a standard in-domain (matched) evaluation in which the training and test data were drawn from the same sources, and a cross-domain (mismatched) evaluation in which the training and test data differ substantially. This cross-domain evaluation is aimed at testing the ability of models to learn representations of sentence meaning that capture broadly useful features.

3.3 Proposed Model

Our proposed model draws inspiration from intra-sentence attention models for sentence representation such as the ones described by Liu et al. (2016) and Lin et al. (2017). In particular, our architecture is based on the notion that it is usually necessary to re-read certain portions of text in order to obtain a comprehensive understanding of it. To model such phenomenon, we rely on an attention mechanism able to iteratively obtain a richer and more expressive version of a raw sentence representation.

The architecture is composed of 5 layers mapping a premise and a hypothesis to one of the three characteristic classes of the NLI task. The Word Representation Layer in charge of creating vector representations from character-derived and pre-trained word embeddings; the Context Representation Layer modeling the interactions between word representations; the Pooling Layer tasked with creating a sentence representations from the contextualized representations; an Inner Attention Layer for refining these; an Aggregation Layer for combining both the premise and hypothesis representations into a single pair representation vector; and finally a Dense Layer projecting this vector into three dimensions corresponding to the size of the classification space. Below we describe each layer in more detail.

Word Representation Layer: This layer is in charge of generating a comprehensive vector representation of each token for a given sentence. We construct this representation based on up to two basic components:

- Pre-trained word embeddings: We take pre-trained word embeddings and use them to generate a raw word representation. This can be seen as a simple lookup-layer that returns a word vector for each provided word index.

- Character embeddings: We generate a character-derived representation of each word, which we concatenate to the pre-trained word vectors. We start by generating a randomly initialized character embedding matrix C. Then, we split each word into its component characters, get their corresponding character embedding vectors from C and feed them into a unidirectional LSTM (Hochreiter and Schmidhuber, 1997). We then choose the last hidden state returned by the LSTM as the fixed-size character-based vector representation for each token. Our embedding matrix C is trained with the rest of the model (Wang et al., 2017).

Context Representation Layer: This layer complements the vectors generated by the Word Representation Layer by incorporating contextual information into them. To do this, we utilize a BiLSTM that reads the embedded sequence and returns the hidden states for each time step. Formally, let $\mathcal{S}$ be a sentence such as $\mathcal{S} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, where each $\boldsymbol{x}_i$ is an embedded word vector as returned by the Word Representation Layer. The contextualized word representation $\boldsymbol{h}_i$ for each time step $i = 1, \ldots, n$ is calculated as follows:

$$\overrightarrow{\boldsymbol{h}}_i = LSTM(\boldsymbol{x}_i, \overrightarrow{\boldsymbol{h}}_{i-1}) \tag{3.1}$$

$$\overleftarrow{\boldsymbol{h}}_i = LSTM(\boldsymbol{x}_i, \overleftarrow{\boldsymbol{h}}_{i+1}) \tag{3.2}$$

$$\boldsymbol{h}_i = [\overrightarrow{\boldsymbol{h}}_i; \overleftarrow{\boldsymbol{h}}_i] \tag{3.3}$$

Where $\overrightarrow{\boldsymbol{h}}_i$ is the forward contextual vector representation of $\boldsymbol{x}_i$, $\overleftarrow{\boldsymbol{h}}_i$ the backward one, and $[\ \cdot\ ;\ \cdot\ ]$ represents the concatenation of two vectors. The output of this layer is a variable-length sentence representation for both the premise and hypothesis. We then define a pooling layer in charge of a generating a raw fixed-size representation of each sentence.

Pooling Layer: This layer is in charge of generating a crude sentence representation vector by reducing the sequence dimension using one of four simple operations, all of which are fed the context-aware token representations $\{\boldsymbol{h}_i\}_{i=1}^{n}$ obtained previously:

$$\bar{\boldsymbol{h}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{h}_i \tag{3.4}$$

$$\bar{\boldsymbol{h}} = \sum_{i=1}^{n} \boldsymbol{h}_i \tag{3.5}$$

$$\bar{\boldsymbol{h}} = [\overrightarrow{\boldsymbol{h}}_n; \overleftarrow{\boldsymbol{h}}_1] \tag{3.6}$$

$$\bar{\boldsymbol{h}} = \max_{i=1,\ldots,n} \boldsymbol{h}_i \tag{3.7}$$

These operations correspond to the *mean* of the word representations (Eq. (3.4)), their *sum* (Eq. (3.5)), the concatenation of the *last* hidden state for each direction (Eq. (3.6)), and the *maximum* one (Eq. (3.7)).

Inner Attention Layer: To refine the representations generated by the pooling strategy, we use a global attention mechanism (Luong et al., 2015; Vinyals et al., 2015) that compares each context-aware token representation $\boldsymbol{h}_i$ with the raw sentence representation $\bar{\boldsymbol{h}}$. Formally,

$$u_i = \boldsymbol{v}^\top \tanh(\boldsymbol{W}[\bar{\boldsymbol{h}}; \boldsymbol{h}_i]) \tag{3.8}$$

$$\alpha_i = \frac{\exp u_i}{\sum_{k=1}^{n} \exp u_k} \tag{3.9}$$

$$\bar{\boldsymbol{h}}' = \sum_{i=1}^{n} \alpha_i \boldsymbol{h}_i \tag{3.10}$$

Where both $\boldsymbol{v}$ and $\boldsymbol{W}$ are trainable parameters and $\bar{\boldsymbol{h}}'$ is the refined sentence representation[1].

Aggregation Layer: To produce a single pair representation from the premise and hypothesis representations we apply the sentence matching mechanism proposed by Mou et al. (2015). Concretely, we concatenate the representations of the premise $\bar{\boldsymbol{h}}'_P$ and hypothesis $\bar{\boldsymbol{h}}'_H$ in addition to their element-wise product (Eq. (3.11)) and their absolute difference (Eq. (3.12)), obtaining the vector $\boldsymbol{r}$ (Eq. (3.13)).

[1]The refined sentence representation $\bar{\boldsymbol{h}}'$ for both premise and hypothesis is the final representation in which both are treated as separate entities. The representations produced by our best-performing model are available in `https://zenodo.org/record/825946`.

$$\boldsymbol{h}_{mul} = \bar{\boldsymbol{h}}'_P \odot \bar{\boldsymbol{h}}'_H \tag{3.11}$$

$$\boldsymbol{h}_{dif} = |\bar{\boldsymbol{h}}'_P - \bar{\boldsymbol{h}}'_H| \tag{3.12}$$

$$\boldsymbol{r} = [\bar{\boldsymbol{h}}'_P; \bar{\boldsymbol{h}}'_H; \boldsymbol{h}_{mul}; \boldsymbol{h}_{dif}] \tag{3.13}$$

Dense Layer: Finally, $\boldsymbol{r}$ is fed to a fully-connected layer whose output is a vector containing the logits for each class, which are then fed to a softmax function for obtaining their probability distribution. The class with the highest probability is chosen as the predicted relationship between premise and hypothesis. The model is trained through backpropagation by minimizing the Cross Entropy between the output distribution and the real class distribution.

3.4 Experiments

To make our results comparable to those obtained by similar models we randomly sampled 15% of the SNLI corpus (Bowman et al., 2015) and added it to the MultiNLI corpus.

We used the pre-trained 300-dimensional GloVe vectors trained on 840B tokens (Pennington et al., 2014). These embeddings were not fine-tuned during training and unknown word vectors were initialized by randomly sampling from the uniform distribution in $(-0.05, 0.05)$.

Each character embedding was initialized as a 20-dimensional vector and the character-level LSTM output dimension was set to 50. The word-level LSTM output dimension was set to 300, which means that after concatenating word-level and character-level representations the word vectors for each direction are 350-dimensional (i.e., $\boldsymbol{h}_i \in \mathbb{R}^{700}$).

For the Inner Attention Layer we defined the parameter $\boldsymbol{W}$ as a square matrix matching the dimension of the concatenated vector $[\bar{\boldsymbol{h}}; \boldsymbol{h}_i]$ (i.e., $\boldsymbol{W} \in \mathbb{R}^{1400\times1400}$), and $\boldsymbol{v}$ as a vector matching the same dimension (i.e., $\boldsymbol{v} \in \mathbb{R}^{1400}$). Both $\boldsymbol{W}$ and $\boldsymbol{v}$ were initialized by randomly sampling from the uniform distribution on the interval $(-0.005, 0.005)$.

The final layer was created as a 3-layer Multi-Layer Perceptron (MLP) with 2000 hidden units each, and with ReLU activations (Nair and Hinton, 2010).

Additionally, we used the `rmsprop` optimizer[2] with an initial learning rate of 0.001. We applied dropout of 0.25 only between the layers of the MLP.

[2]This optimizer has no associated peer-reviewed publication, and was first proposed in one of Geoffrey Hinton's lectures: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

Further, we found out that normalizing the capitalization of words by making all characters lowercase, and transforming numbers into a specific numeric token improved the model's performance while reducing the size of the embedding matrix. We also ignored the sentence pairs with a premise longer than 200 words during training (for improved memory stability), and those without a valid label ("-") both during training and validation.

Since one of the most conceptually important parts of our model was the raw sentence representation created in the Pooling Layer, we used four different methods for generating it (Eqs. (3.4) to (3.7)). Results are reported in Table 3.2.

We also tried using other architectures that rely on some variant of "inner" attention such as the *self-attentive* model proposed by Lin et al. (2017) and the *co-attentive* model by Xiong et al. (2016), but our preliminary results were not promising so we did not attempt to fine-tune them.

All the experiments were repeated without using the character modality (i.e., $\boldsymbol{h}_i \in \mathbb{R}^{600}$), to evaluate their contribution to overall performance.

3.5 Results

In Table 3.1 we report the accuracies obtained by our best model in both matched (first 5 genres) and mismatched (last 5 genres) development sets. We can observe that our implementation performed like ESIM overall, however ESIM relies on the parse tree of the input sentences, and on a TreeLSTM (Tai et al., 2015), to process them (Chen et al., 2017; Williams et al., 2018), while ours works in a more constrained setting, only with access to the their surface representations.

It is also worth noting that the ESIM model presented in table Table 3.1, was trained in MultiNLI only, while ours was trained in MultiNLI with an added slice of SNLI, so both columns are not perfectly comparable. However, Williams et al. (2018) also reported the overall performance of an ESIM model trained with this setting: it obtained 72.4% and 71.9% in the matched and mismatched setting respectively.

We picked the best model based on the best validation accuracy score obtained on the matched development set (72.257%). We submitted these predictions to the RepEval 2017 competition (Nangia et al., 2017), and obtained test accuracies of 72.057% in the matched and 72.055% in the mismatched settings, on private test sets for each setting. Additionally, we created an ensemble by training 4 models as described earlier but initialized with different random seeds. The prediction was made by averaging the probability distributions returned by each model and then picking the class with the highest probability for each example. This improved our test results to 72.247%

Genre	**CBOW**	**ESIM**	**InnerAtt**
Fiction	67.5	73.0	73.2
Government	67.5	74.8	75.2
Slate	60.6	67.9	67.2
Telephone	63.7	72.2	73.0
Travel	64.6	73.7	72.8
Avg. Matched Acc.	64.8	72.3	72.3
9/11	63.2	71.9	70.5
Face-to-face	66.3	71.2	74.5
Letters	68.3	74.7	75.4
Non-fiction	62.8	71.7	71.5
Linguistics	62.7	71.9	69.5
Avg. Mismatched Acc.	64.7	72.3	72.3
MultiNLI Overall	**64.7**	**72.3**	**72.3**

Table 3.1 – Validation accuracies (%) for our best model broken down by genre. Both CBOW and ESIM results are reported as in (Williams et al., 2018).

(+0.19%) in the matched, and to 72.827% (+0.77%) in the mismatched evaluation tracks.

Effect of Pooling Method and Character Modality

To assess the importance of character-derived word representations and pooling mechanisms, we performed an ablation study on these components. We can see in Table 3.2 that that both the *mean* method, and picking the last hidden state for both directions performed slightly better than the two other strategies, however at 95% confidence we cannot assert that any of these methods is statistically different from one another.

This could be interpreted as any of the four methods being good enough for capturing the overall meaning of the sentence, and letting the attention mechanism do the heavy lifting. As future work we intend to test these four strategies without the presence of attention to see whether it really plays an important role in this task or whether the predictive power lies within the sentence matching mechanism.

Another interesting result, as shown by Tables 3.2 and 3.3, is that the model seemed to be insensitive to the usage of character embeddings. To test whether this effect was tied to our architecture or whether it was a more general trend we added character-derived word representations to a reimplementation of the ESIM model[3], without depending on the sentences' parse trees, but sharing information between premise and hypothesis. Without the character modality it obtained matched and mismatched ac-

[3] https://github.com/coetaur0/ESIM

Method	w/o. chars	w. chars
mean	71.3 ± 1.2	71.3 ± 0.7
sum	70.7 ± 1.0	70.9 ± 0.8
last	70.9 ± 0.6	71.0 ± 1.2
max	70.6 ± 1.1	71.0 ± 1.1

Table 3.2 – Mean matched validation accuracies (%) broken down by type of pooling method and presence or absence of character embeddings. Confidence intervals are calculated at 95% confidence over 10 runs for each method.

Method	w/o. chars	w. chars
mean	**72.3**	71.8
sum	71.6	71.6
last	71.4	**72.1**
max	71.1	71.6

Table 3.3 – Best matched validation accuracies (%) obtained by each pooling method in presence and absence of character embeddings.

curacies of 75.94% and 75.95% respectively. After adding the character-derived representations, these accuracies slightly increased to 76.10% (+0.16%) and 76.15% (+0.2%) respectively. These results show that character-level representations do carry information useful for the task at hand, but their usefulness might be dependent on the downstream modules of the architecture. A possible explanation for this is that English is not a morphologically rich language, hence does not contain meaningful patterns at the subword level, and therefore complex architectures at the word level are enough for capturing syntactic and semantic information.

3.6 Conclusions and Future work

We presented a model for tackling the NLI task. Despite being conceptually simple and not relying on parse trees for encoding each sentence, our implementation achieved results as good as the ESIM model.

Future venues for improvement include incorporating part-of-speech embeddings by concatenating them with the pre-trained word embeddings as we did with the character embeddings. Incorporating pre-trained character embeddings could also have a positive impact in performance. We also did not perform an exhaustive hyperparameter search, and think results could be further improved by finding a better hyperparameter configuration. Specifically, we did not try using different types of attention; for this implementation we used the *concat* scoring scheme (Eq. (3.8)), as described by Luong et al. (2015), but there are several others that could provide better results.

Finally, we were surprised at character embeddings not providing performance im-

provements in this task with our architecture, despite our preliminary observations that they did improve results in a different model (Wang et al., 2017). This might be due to the fact that the architecture following the character-aware word-encoding mechanism in our model is not sophisticated enough to exploit character-level information. Indeed, later experiments showed that enhancing the ESIM model with characters did provide performance improvements in MultiNLI, suggesting that benefits provided by the character modality might be dependent on specific architectural details.

In the next chapter we study another architecture exploiting the character modality, pre-trained in a self-supervised fashion, and evaluated in the task of implicit emotion classification.

Chapter 4

Contextualized Word Representations for Predicting Implicit Emotion

4.1 Abstract

In this chapter we introduce an architecture for tackling the task of implicit emotion recognition. The system is composed of a single pre-trained ELMo layer for obtaining word representations, a Bidirectional Long-Short Memory Network BiLSTM for enriching word representations with context, a max-pooling operation for creating sentence representations from them, and a Dense Layer for projecting the sentence representations into label space. Our model obtained 69.23% validation accuracy. We submitted our system to the WASSA 2018 Implicit Emotions Shared Task (IEST) and obtained an f1-score of 71.05 on a held-out test set, with an ensemble of 6 models, obtaining 2nd place out of 30 teams. Code for replicating this chapter is available at https://github.com/jabalazs/implicit_emotion.

4.2 Introduction

Although the definition of emotion is still debated among the scientific community, the automatic identification and understanding of human emotions by machines has long been of interest in computer science. It has usually been assumed that emotions are triggered by the interpretation of a stimulus event according to its meaning.

As language usually reflects the emotional state of an individual, it is natural to study human emotions by understanding how they are reflected in text. We see that many words indeed have affect as a core part of their meaning, for example, *dejected*

and *wistful* denote some amount of sadness, and are thus associated with sadness. On the other hand, words like *failure* and *death*, despite not having affect in themselves, are usually used in sad contexts and thus could be considered as words conveying sad affect.

The task of automatically recognizing emotions from text has recently attracted the attention of researchers in Natural Language Processing. This task is usually formalized as the classification of words, phrases, or documents into predefined discrete emotion categories. Some approaches have also aimed at predicting to which degree an emotion is expressed in text (Mohammad and Bravo-Marquez, 2017).

In light of this, the WASSA 2018 Implicit Emotions Shared Task (Klinger et al., 2018) was proposed to help find ways to automatically learn the link between situations and the emotion they trigger. The task consisted in predicting the emotion of a masked word in the context of a tweet. Masked words, or *trigger-words*, included the terms "sad", "happy", "disgusted", "surprised", "angry", "afraid" and their synonyms, and the task was to predict the emotion they conveyed, specifically sadness, joy, disgust, surprise, anger and fear.

From a machine learning perspective, this problem can be seen as sentence classification, in which the goal is to classify a sentence, into one of several categories. In the case of IEST, the problem is specially challenging since tweets contain informal language, and use of emoji, hashtags and username mentions.

Our system did not require manual feature engineering and relied on minimal use of external data. Concretely, our approach is composed of a single pre-trained Embeddings from Language Models (ELMo) layer for encoding words (Peters et al., 2018a), a BiLSTM, for modeling word context, a max-pooling operation for aggregating contextualized word representations into sentence representations, and finally a Dense Layer for projecting sentence representations into label space. To the best of our knowledge our system was the first to utilize ELMo (or any pre-trained language model), for implicit emotion recognition.

4.3 Proposed Approach

4.3.1 Preprocessing

As our model is purely character-based, we performed little data preprocessing. Table 4.1 shows the special tokens found in the datasets, and how we substituted them.

Furthermore, we tokenized the text using a variation of the `twokenize.py`[1] script, a

[1] https://github.com/myleott/ark-twokenize-py

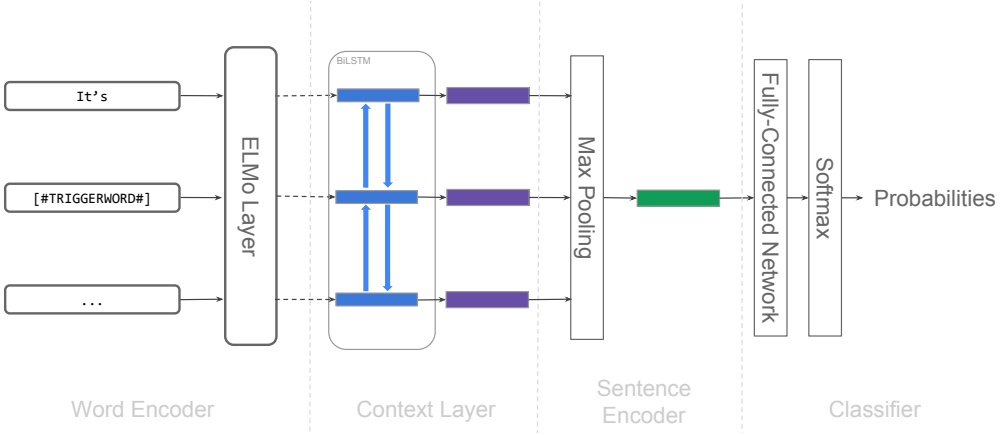


Figure 4.1 – Proposed architecture.

Original	Replacement
[#TRIGGERWORD#]	__TRIGGERWORD__
@USERNAME	__USERNAME__
[NEWLINE]	__NEWLINE__
http://url.removed	__URL__

Table 4.1 – Preprocessing substitutions.

Python port of the original `Twokenize.java` (Gimpel et al., 2011). Concretely, we created an emoji-aware version of it by incorporating knowledge from an emoji database,[2] which we slightly modified for avoiding conflict with emoji sharing unicode codes with common glyphs used in Twitter,[3] and for making it compatible with Python 3.

4.3.2 Architecture

Figure 4.1 summarizes our proposed architecture. Our input is based on ELMo by Peters et al. (2018a). ELMo uses a set of convolutional neural networks to extract features from character embeddings, and builds word vectors from them. These are then fed to a multi-layer Bidirectional Language Model (BiLM) which returns context-sensitive vectors for each input word.

We used a single-layer BiLSTM as context fine-tuner on top of the pre-trained ELMo encoder, and then max-pooled the returned hidden states to form the sentence vector representation, method which has been shown to perform well on classification tasks (Conneau et al., 2017).

Finally, we used a single-layer fully-connected network for projecting the sentence representation into a vector corresponding to the label logits for each predicted class.

Note that the ELMo word encoding architecture is essentially different to that eval-

[2] https://github.com/carpedm20/emoji/blob/e7bff32/emoji/unicode_codes.py

[3] For example, the hashtag emoji is composed by the unicode code points `U+23 U+FE0F U+20E3`, which include `U+23`, the same code point for the `#` glyph.

uated in Chapter 3 in that it only relies in the character modality, and does not keep a separate word embedding lookup-table, therefore avoiding the problem of having to combine different modalities of the same concept. This has the advantage that the model does not depend on a single word vocabulary and can consequently deal with any input containing characters seen during training. On the other hand it has the downside that it is unable to exploit pre-trained word embeddings encoding associative relationships between words.

4.3.3 Implementation Details and Hyperparameters

ELMo Layer: We used the official AllenNLP implementation of the ELMo model[4], with the official weights pre-trained on the 1 Billion Word Language Model Benchmark, which contains about 800M tokens of news crawl data from WMT 2011 (Chelba et al., 2014).

Dimensionalities: By default the ELMo layer outputs a 1024-dimensional vector, which we then feed to a BiLSTM with output size 2048, resulting in a 4096-dimensional vector when concatenating forward and backward directions for each word of the sequence[5]. After max-pooling the BiLSTM output over the sequence dimension, we obtain a single 4096-dimensional vector corresponding to the tweet representation. This representation is finally fed to a single-layer fully-connected network with input size 4096, 512 hidden units, output size 6, and a ReLU nonlinearity after the hidden layer. The output of the dense layer is a 6-dimensional logit vector for each input example.

Loss Function: As this corresponds to a multiclass classification problem (predicting a single class for each example, with more than 2 classes to choose from), we used the Cross-Entropy Loss as implemented in PyTorch (Paszke et al., 2017).

Optimization: We optimized the model with Adam (Kingma and Ba, 2014), using default hyperparameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$), following a slanted triangular learning rate schedule (Howard and Ruder, 2018), also with default hyperparameters ($cut\_frac = 0.1$, $ratio = 32$), and a maximum learning rate $\eta_{\max} = 0.001$, over $T = 23,970$ iterations[6].

Regularization: we used a dropout layer (Srivastava et al., 2014), with probability of 0.5 after both the ELMo and the hidden fully-connected layer, and another one with probability of 0.1 after the max-pooling aggregation layer. We also reshuffled the

[4] https://allenai.github.io/allennlp-docs/api/allennlp.modules.elmo.html

[5] A BiLSTM is composed of two separate LSTMs that read the input in opposite directions and whose outputs are concatenated at the hidden dimension. This results in a vector double the dimension of the input for each time step.

[6] This number is obtained by multiplying the number of epochs (10), times the total number of batches, which for the training dataset corresponds to 2396 batches of 64 elements, and 1 batch of 39 elements, hence $2397 \times 10 = 23,970$.

training examples between epochs, resulting in a different batch for each iteration.

Model Selection: To choose the best hyperparameter configuration we measured the classification accuracy on the validation (trial) set.

4.3.4 Ensembles

Once we found the best-performing configuration we trained 10 models using different random seeds, and tried averaging the output class probabilities of all their possible $\sum_{k=1}^{9} \binom{9}{k} = 511$ combinations. As Figure 4.2 shows, we empirically found that a specific combination of 6 models yielded the best results (70.52%), providing evidence for the fact that using a number of independent classifiers equal to the number of class labels provides the best results when doing average ensembling (Bonab and Can, 2016).

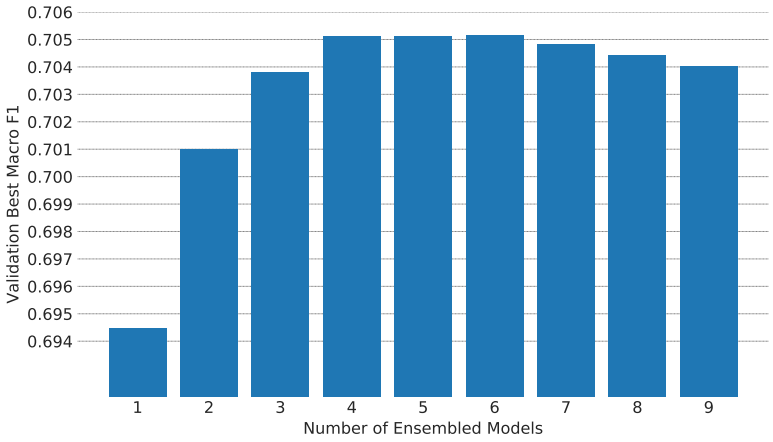


Figure 4.2 – Effect of the number of ensembled models on validation performance.

4.4 Experiments and Analyses

We performed several experiments to gain insights on how the proposed model's performance interacts with the shared task's data. We performed an ablation study to see how some of the main hyperparameters affect performance, and an analysis of tweets containing hashtags and emoji to understand how these two types of tokens help the model predict the trigger-word's emotion. We also observed the effects of varying the amount of data used for training the model to evaluate whether it would be worthwhile to gather more training data.

4.4.1 Ablation Study

We performed an ablation study on a single model having obtained 69.23% accuracy on the validation set. Results are summarized in Table 4.2.

Variation	Accuracy (%)	Δ%
Submitted	**69.23**	-
No emoji	68.36	−0.87
No ELMo	65.52	−3.71
Concat Pooling	68.47	−0.76
LSTM hidden=4096	69.10	−0.13
LSTM hidden=1024	68.93	−0.30
LSTM hidden=512	68.43	−0.80
POS emb dim=100	68.99	−0.24
POS emb dim=75	68.61	−0.62
POS emb dim=50	69.33	+0.10
POS emb dim=25	69.21	−0.02
SGD optim lr=1	64.33	−4.90
SGD optim lr=0.1	66.11	−3.12
SGD optim lr=0.01	60.72	−8.51
SGD optim lr=0.001	30.49	−38.74

Table 4.2 – Ablation study results.

Accuracies were obtained from the validation dataset. Each model was trained with the same random seed and hyperparameters, save for the one listed. "No emoji" is the same model trained on the training dataset with no emoji, "No ELMo" corresponds to having switched the ELMo word encoding layer with a simple pre-trained GloVe embedding lookup table, and "Concat Pooling" obtained sentence representations by using the pooling method described by Howard and Ruder (2018). "LSTM hidden" corresponds to the hidden dimension of the BiLSTM, "POS emb dim" to the dimension of the part-of-speech embeddings, and "SGD optim lr" to the learning rate used while optimizing with the schedule described by Conneau et al. (2017).

We can observe that the architectural choice that had the greatest impact on our model was the ELMo layer, providing a 3.71% increase in performance as compared to using GloVe pre-trained word embeddings.

We can further see that emoji also contributed significantly to the model's performance. In Section 4.4.4 we give some pointers to understanding why this is so.

Additionally, we tried using the concatenation of the max-pooled, average-pooled and last hidden states of the BiLSTM as the sentence representation, following Howard and Ruder (2018), but found out that this impacted performance negatively. We hypothesize this is due to tweets being too short for needing such a rich representation. Also, the size of the concatenated vector was $4096 \times 3 = 12,288$, which probably could not be properly exploited by the 512-dimensional fully-connected layer.

Using a greater BiLSTM hidden size did not help the model, probably because of the reason mentioned earlier; the fully-connected layer was not big or deep enough to

exploit the additional information. Similarly, using a smaller hidden size neither helped.

We found that using 50-dimensional part-of-speech embeddings slightly improved results, which implies that better fine-tuning this hyperparameter, or using a better Part-of-Speech (POS) tagger could yield an even better performance.

Regarding optimization strategies, we also tried using SGD with different learning rates and a step-wise learning rate schedule as described by Conneau et al. (2018a), but we found that doing this did not improve performance.

Finally, Figure 4.3 shows the effect of using different dropout probabilities. We can see that having higher dropout after the word-representation layer and the fully-connected network's hidden layer, while having a low dropout after the sentence encoding layer yielded better results overall.

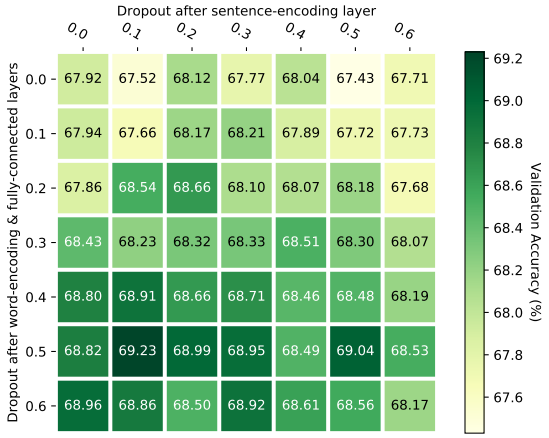


Figure 4.3 – Dropout Ablation.
Rows correspond to the dropout applied both after the ELMo layer (word encoding layer) and after the fully-connected network's hidden layer, while columns correspond to the dropout applied after the max-pooling operation (sentence encoding layer.)

4.4.2 Error Analysis

Figure 4.5 shows the confusion matrix of a single model evaluated on the test set, and Table 4.3 the corresponding classification report. In general, we confirm what Klinger et al. (2018) report: `anger` was the most difficult class to predict, followed by `surprise`, whereas `joy`, `fear`, and `disgust` are the better performing ones.

To observe whether any particular pattern arose from the sentence representations encoded by our model, we projected them into 3d space through Principal Component Analysis (PCA), and were surprised to find that 2 clearly defined clusters emerged (see Figure 4.4), one containing the majority of datapoints, and another containing `joy` tweets exclusively. Upon further exploration we also found that the smaller cluster was composed only by tweets containing the pattern `un __TRIGGERWORD__`, and further, that all of them were correctly classified.

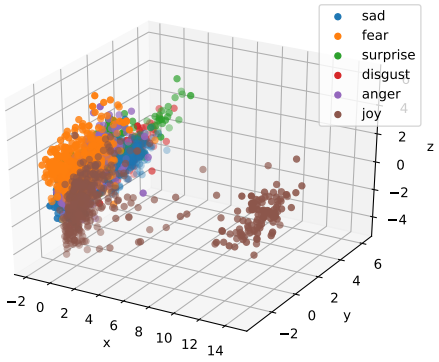


Figure 4.4 – 3d Projection of the Test Sentence Representations.

It is also worth mentioning that there are 5827 tweets in the training set with this pattern. Of these, 5822 (99.9%) correspond to the label `joy`. We observe a similar trend on the test set; 1115 of the 1116 tweets having the `un __TRIGGERWORD__` pattern correspond to `joy` tweets. We hypothesize this is the reason why the model learned this pattern as a strong discriminating feature.

Finally, the only tweet in the test set that contained this pattern and did not belong to the `joy` class, originally had *unsurprised* as its triggerword[7], and unsurprisingly, was misclassified.

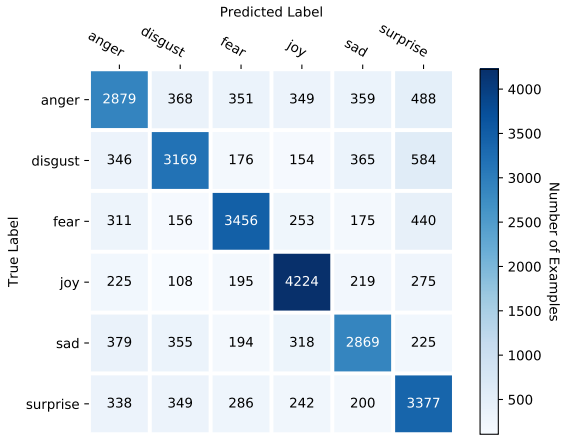


Figure 4.5 – Confusion Matrix (Test Set).

4.4.3 Effect of the Amount of Training Data

As Figure 4.6 shows, increasing the amount of data with which our model was trained consistently increased validation accuracy and validation macro F1 score. The trend

[7]We manually searched for the original tweet.

	Precision	Recall	F1-score
anger	0.643	0.601	0.621
disgust	0.703	0.661	0.682
fear	0.742	0.721	0.732
joy	0.762	0.805	0.783
sad	0.685	0.661	0.673
surprise	0.627	0.705	0.663
Average	0.695	0.695	0.694

Table 4.3 – Classification Report (Test Set).

suggests that the proposed model is expressive enough to learn from more data, and is not overfitting the training set.

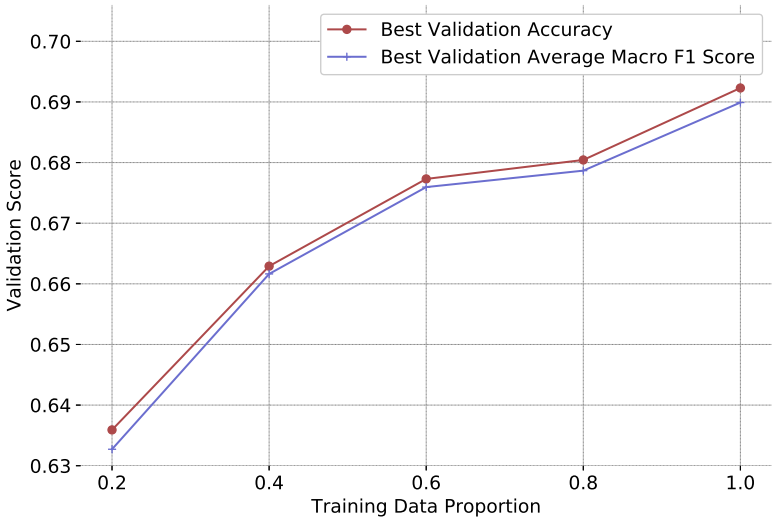


Figure 4.6 – Effect of the amount of training data on classification performance.

4.4.4 Effect of Emoji and Hashtags

	Present	Not Present
Emoji	4805 (76.6%)	23952 (68.0%)
Hashtags	2122 (70.5%)	26635 (69.4%)

Table 4.4 – Number of tweets on the test set with and without emoji and hashtags. The number between parentheses is the proportion of tweets classified correctly.

Table 4.4 shows the overall effect of hashtags and emoji on classification performance. Tweets containing emoji seem to be easier for the model to classify than those without. Hashtags also have a positive effect on classification performance, however it is less significant. This implies that emoji, and hashtags in a smaller degree, provide tweets with a context richer in sentiment information, allowing the model to better guess the emotion of the *trigger-word*.

Emoji alias	N	emoji		no-emoji		Δ%
		#	%	#	%	
mask	163	154	94.48	134	82.21	- 12.27
two_hearts	87	81	93.10	77	88.51	- 4.59
heart_eyes	122	109	89.34	103	84.43	- 4.91
heart	267	237	88.76	235	88.01	- 0.75
rage	92	78	84.78	66	71.74	- 13.04
cry	116	97	83.62	83	71.55	- 12.07
sob	490	363	74.08	345	70.41	- 3.67
unamused	167	121	72.46	116	69.46	- 3.00
weary	204	140	68.63	139	68.14	- 0.49
joy	978	649	66.36	629	64.31	- 2.05
sweat_smile	111	73	65.77	75	67.57	1.80
confused	77	46	59.74	48	62.34	2.60

Table 4.5 – Fine grained performance on tweets containing emoji, and the effect of removing them. **N** is the total number of tweets containing the listed emoji, **#** and **%** the number and percentage of correctly-classified tweets respectively, and **Δ%** the variation of test accuracy when removing the emoji from the tweets.

Table 4.5 shows the effect specific emoji have on classification performance. It is clear some emoji strongly contribute to improving prediction quality. The most interesting ones are `mask`, `rage`, and `cry`, which significantly increase accuracy. Further, contrary to intuition, the `sob` emoji contributes less than `cry`, despite representing a stronger emotion. This is probably due to `sob` being used for depicting a wider spectrum of emotions.

Finally, not all emoji are beneficial for this task. When removing `sweat_smile` and `confused` accuracy increased, probably because they represent emotions other than the ones being predicted.

4.4.5 Model Comparison

Table 4.6 shows an overview of the information sources and methodologies used by each team in the shared task.

It is clear that relying on Language Models (LMs) was key to performance in this task. Indeed, the top 4 teams made use of them. The 1st team pre-trained a LM architecture in 5×10^9 tweets, while the 3rd team did so in three different Twitter corpora with 2 million, 3 million and 5 million examples respectively. Both our team and the 4th team relied on a pre-trained ELMo architecture. Every team not using LMs obtained worse results (Klinger et al., 2018).

The usage of characters was spread among teams along the whole rank spectrum, meaning that using characters alone was not a good predictor of performance. Further, in our case, the usage of characters along with the pre-trained ELMo architecture confounds the contribution of each, therefore it is not possible to accurately assess to which extent the pre-training procedure or the character modality explain performance.

Team	Rank	F1-Score	Words	Lexicons	Characters	Emoji	Unlabeled Corpora	Emotion Emb.	Embeddings	LSTM/RNN/GRU	Ensemble	CNN/Capsules	Transfer Learning	Language model	SemEval	ELMo
Amobee	1	71.45	✓	✓			✓	✓	✓	✓	✓	✓	✓	✓	✓	
Ours	2	71.05	✓		✓	✓	✓		✓	✓	✓	✓	✓	✓		✓
NTUA-SLP	3	70.29	✓				✓		✓	✓	✓		✓	✓	✓	
UBC-NLP	4	69.28	✓			✓	✓		✓	✓	✓		✓	✓		
Sentylic	5	69.20	✓						✓	✓	✓	✓				
HUMIR	6	68.64	✓	✓				✓	✓	✓	✓		✓			
nlp	7	68.48	✓						✓	✓	✓					
DataSEARCH	8	68.04	✓		✓	✓			✓	✓		✓				
YNU1510	9	67.63	✓						✓	✓	✓	✓				
EmotiKLUE	10	67.13	✓				✓		✓	✓						
wojtek.pierre	11	66.15	✓	✓	✓				✓	✓		✓				✓
hgsgnlp	12	65.80	✓	✓	✓		✓		✓		✓	✓				
UWB	13	65.70	✓			✓			✓	✓						
NL-FIIT	14	65.52	✓			✓			✓	✓			✓			
TubOslo	15	64.63	✓		✓											
YNU_Lab	16	64.10	✓						✓	✓						
Braint	17	62.61	✓			✓			✓	✓						
EmoNLP	18	62.11	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓				
RW	19	60.97	✓													
Baseline	20	59.88	✓	✓		✓			✓	✓						
			20	6	6	8	7	3	18	17	10	8	6	4	2	2

Table 4.6 – Overview of information sources and methods employed by different teams in the WASSA 2018 Implicit Emotions Shared Task. Table adapted from (Klinger et al., 2018) with modifications.

The 1st and 3rd teams used LMs that did not rely on the character modality, showing that it might actually not be essential when copious amount of data from the same domain are available. In such scenario the amount of potential OOV words is effectively reduced, and pre-training might be enough to produce highly-performant word representations. ELMo, however, was pre-trained on a dataset containing data significantly dissimilar to that found in Twitter while obtaining competitive performance. This fact entertains the possibility that using the character modality might ease the knowledge transfer between different domains.

4.5 Conclusions and Future Work

We described an architecture for tackling the Implicit Emotion Classification task and obtained 2nd place in WASSA 2018 IEST. Despite its simplicity, and low amount of dependencies on libraries and external features, it performed almost as well as the system that obtained the first place.

Our ablation study revealed that our hyperparameters were indeed quite well-tuned for the task, which agrees with the good results obtained in the official submission. The ablation study also showed that we could obtain better performance by incorporating

POS embeddings as additional features, however, further experiments are required to accurately measure their impact. We also think performance can be improved by adding more elaborate components to the architecture, concretely, by using a BiLSTM with multiple layers and skip connections in a way akin to (Peters et al., 2018a), or by making the fully-connected network deeper.

We also showed that, what was probably an annotation artifact, the `un __TRIGGERWORD__` pattern, resulted in increased performance for the `joy` label. This pattern was probably originated by a heuristic naïvely replacing the ocurrence of *happy* by the trigger-word indicator. We think the dataset could be improved by replacing the word *unhappy*, in the original examples, by `__TRIGGERWORD__` instead of `un __TRIGGERWORD__`, and labeling it as `sad`, or `angry`, instead of `joy`.

Further, our studies regarding the importance of hashtags and emoji showed that both of them carry a strong sentiment signal, as they contributed significantly to implicit emotion classification performance.

The significance of the character modality was again proven to be ambiguous as its usage provided mixed results among different teams. The usage of LMs on the other hand proved to be key in obtaining the highest performances, as the four teams that used them obtained the four top ranks in the shared task leaderboard.

Finally, even though it was not possible to accurately estimate how useful the character modality was in this scenario, we argued that exploiting character-level information might explain how ELMo was able to transfer its knowledge and apply it to inherently noisy data from Twitter.

In the next chapter we attempt to gain further insights into the character modality. Specifically, we will study how combining both character and word modalities affects representation quality in several scenarios.

Chapter 5

Vector Gates for Combining Character-derived and Pre-trained Word Representations

5.1 Abstract

In this chapter we study how different ways of combining character and word-level representations affect the quality of both final word and sentence representations. We provide strong empirical evidence that modeling characters improves the learned representations at the word and sentence levels, and that doing so is particularly useful when representing less frequent words. We further show that a feature-wise sigmoid gating mechanism is a robust method for creating representations that encode semantic similarity, as it performed reasonably well in several word similarity datasets. Finally, our findings suggest that properly capturing semantic similarity at the word level does not consistently yield improved performance in downstream sentence-level tasks. Code for replicating this chapter is available at https://github.com/jabalazs/gating.

5.2 Introduction

Incorporating sub-word structures like substrings, morphemes and characters to the creation of word representations significantly increases their quality as reflected both by intrinsic metrics and performance in a wide range of downstream tasks (Bojanowski et al., 2017; Ling et al., 2015; Luong and Manning, 2016; Wu et al., 2016).

The reason for this improvement is related to sub-word structures containing information that is usually ignored by standard word-level models. Indeed, when representing words as vectors extracted from a lookup table, semantically related words

resulting from inflectional processes such as *surf*, *surfing*, and *surfed*, are treated as being independent from one another[1]. Further, word-level embeddings do not account for derivational processes resulting in syntactically-similar words with different meanings such as *break*, *breakable*, and *unbreakable*. This causes derived words, which are usually less frequent, to have lower-quality (or no) vector representations.

Previous works have successfully combined character-level and word-level word representations, obtaining overall better results than using only word-level representations. For example Luong and Manning (2016) achieved state-of-the-art results in a machine translation task by representing unknown words as a composition of their characters. Botha and Blunsom (2014) created word representations by adding the vector representations of the words' surface forms and their morphemes ($\overrightarrow{\text{perfectly}} = \overrightarrow{perfectly} + \overrightarrow{perfect} + \overrightarrow{ly}$), obtaining significant improvements on intrinsic evaluation tasks, word similarity and machine translation. Lample et al. (2016) concatenated character-level and word-level representations for creating word representations, and then used them as input to their models for obtaining state-of-the-art results in Named-Entity Recognition (NER) on several languages.

What these works have in common is that the models they describe first learn how to represent subword information, at character (Luong and Manning, 2016), morpheme (Botha and Blunsom, 2014), or substring (Bojanowski et al., 2017) levels, and then combine these learned representations at the word level. The incorporation of information at a finer-grained hierarchy results in higher-quality modeling of rare words, morphological processes, and semantics (Avraham and Goldberg, 2017).

There is no consensus, however, on which combination method works better in which case, or how the choice of a combination method affects downstream performance, either measured intrinsically at the word level, or extrinsically at the sentence level.

In this paper we aim to provide some intuitions about how the choice of mechanism for combining character-level with word-level representations influences the quality of the final word representations, and the subsequent effect these have in the performance of downstream tasks. Our contributions are as follows:

- We show that a feature-wise sigmoidal gating mechanism is the best at combining representations at the character and word-level hierarchies, as measured by word similarity tasks.
- We provide evidence that this mechanism learns that to properly model increasingly infrequent words, it has to increasingly rely on character-level information.
- We finally show that despite the increased expressivity of word representations it offers, it has no clear effect in sentence representations, as measured by sentence

[1]Unless using pre-trained embeddings with a notion of subword information such as `fastText` (Bojanowski et al., 2017)

evaluation tasks.

5.3 Background

We are interested in studying different ways of combining word representations, obtained from different hierarchies, into a single word representation. Specifically, we want to study how combining word representations (1) taken directly from a word embedding lookup table, and (2) obtained from a function over the characters composing them, affects the quality of the final word representations.

Let $\mathcal{W}$ be a set, or vocabulary, of words with $|\mathcal{W}|$ elements, and $\mathcal{C}$ a vocabulary of characters with $|\mathcal{C}|$ elements. Further, let $\boldsymbol{x} = w_1, \ldots, w_n;\ w_i \in \mathcal{W}$ be a sequence of words, and $\boldsymbol{c}^i = c_1^i, \ldots, c_m^i;\ c_j^i \in \mathcal{C}$ be the sequence of characters composing w_i. Each token w_i can be represented as a vector $\boldsymbol{v}_i^{(w)} \in \mathbb{R}^d$ extracted directly from an embedding lookup table $\boldsymbol{E}^{(w)} \in \mathbb{R}^{|\mathcal{W}| \times d}$, pre-trained or otherwise, and as a vector $\boldsymbol{v}_i^{(c)} \in \mathbb{R}^d$ built from the characters that compose it; in other words, $\boldsymbol{v}_i^{(c)} = f(\boldsymbol{c}^i)$, where f is a function that maps a sequence of characters to a vector.

The methods for combining word and character-level representations we study, are of the form $G(\boldsymbol{v}_i^{(w)}, \boldsymbol{v}_i^{(c)}) = \boldsymbol{v}_i$ where $\boldsymbol{v}_i$ is the final word representation.

5.3.1 Mapping Characters to Character-level Word Representations

The function f is composed of an *embedding* layer, an optional *context* function, and an *aggregation* function.

The **embedding layer** transforms each character c_j^i into a vector $\boldsymbol{r}_j^i$ of dimension d_r, by directly taking it from a trainable embedding lookup table $\boldsymbol{E}^{(c)} \in \mathbb{R}^{|\mathcal{C}| \times d_r}$. We define the *matrix* representation of word w_i as $\boldsymbol{C}^i = [\boldsymbol{r}_1^i, \ldots, \boldsymbol{r}_m^i],\ \ \boldsymbol{C}^i \in \mathbb{R}^{m \times d_r}$.

The **context function** takes $\boldsymbol{C}^i$ as input and returns a context-enriched matrix representation $\boldsymbol{H}^i = [\boldsymbol{h}_1^i, \ldots, \boldsymbol{h}_m^i],\ \ \boldsymbol{H}^i \in \mathbb{R}^{m \times d_h}$, in which each $\boldsymbol{h}_j^i$ contains a measure of information about its context, and interactions with its neighbors. In particular, we chose to do this by feeding $\boldsymbol{C}^i$ to a BiLSTM (Graves et al., 2013; Graves and Schmidhuber, 2005)[2].

Informally, we can think of an LSTM (Hochreiter and Schmidhuber, 1997) as a function $\mathbb{R}^{m \times d_r} \rightarrow \mathbb{R}^{m \times d_h}$ that takes a matrix $\boldsymbol{C} = [\boldsymbol{r}_1, \ldots, \boldsymbol{r}_m]$ as input and returns a context-enriched matrix representation $\boldsymbol{H} = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_m]$, where each $\boldsymbol{h}_j$ encodes

[2]Other methods for encoding the characters' context, such as CNNs (Kim et al., 2016), could also be used.

information about the previous elements $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{j-1}$[3].

A BiLSTM is simply composed of 2 LSTMs, one that reads the input from left to right (forward), and another that does so from right to left (backward). The output of the forward and backward LSTMs are $\overrightarrow{\boldsymbol{H}} = [\overrightarrow{\boldsymbol{h}}_1, \ldots, \overrightarrow{\boldsymbol{h}}_m]$ and $\overleftarrow{\boldsymbol{H}} = [\overleftarrow{\boldsymbol{h}}_1, \ldots, \overleftarrow{\boldsymbol{h}}_m]$ respectively. In the backward case the LSTM reads $\boldsymbol{r}_m$ first and $\boldsymbol{r}_1$ last, therefore $\overleftarrow{\boldsymbol{h}}_j$ will encode the context from $\overleftarrow{\boldsymbol{h}}_{j+1}, \ldots, \overleftarrow{\boldsymbol{h}}_m$.

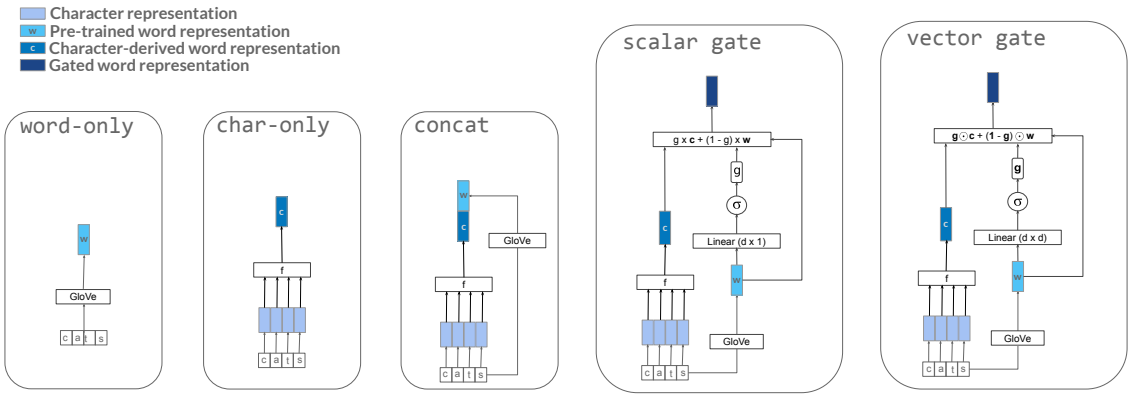


Figure 5.1 – Character and Word-level combination methods.

The **aggregation function** takes the context-enriched matrix representation of word w_i for both directions, $\overrightarrow{\boldsymbol{H}^i}$ and $\overleftarrow{\boldsymbol{H}^i}$, and returns a single vector $\boldsymbol{v}_i^{(c)} \in \mathbb{R}^{d_h}$. To do so we followed Miyamoto and Cho (2016), and defined the character-level representation $\boldsymbol{v}_i^{(c)}$ of word w_i as the linear combination of the forward and backward last hidden states returned by the context function:

$$\boldsymbol{v}_i^{(c)} = \boldsymbol{W}^{(c)}[\overrightarrow{\boldsymbol{h}^i}_m; \overleftarrow{\boldsymbol{h}^i}_1] + \boldsymbol{b}^{(c)} \tag{5.1}$$

where $\boldsymbol{W}^{(c)} \in \mathbb{R}^{d_h \times 2d_h}$ and $\boldsymbol{b}^{(c)} \in \mathbb{R}^{d_h}$ are trainable parameters, and $[\circ; \circ]$ represents the concatenation operation between two vectors.

5.3.2 Combining Character and Word-level Representations

We tested three different methods for combining $\boldsymbol{v}_i^{(c)}$ with $\boldsymbol{v}_i^{(w)}$: simple concatenation, a learned scalar gate (Miyamoto and Cho, 2016), and a learned vector gate (also referred to as feature-wise sigmoidal gate). Additionally, we compared these methods to two baselines: using pre-trained word vectors only, and using character-only features for representing words. See Fig. 5.1 for a visual description of the proposed methods.

[3]In terms of implementation, the LSTM is applied iteratively to each element of the input sequence regardless of dimension m, which means it accepts inputs of variable length, but we will use this notation for the sake of simplicity.

`word-only (w)` considers only $\boldsymbol{v}_i^{(w)}$ and ignores $\boldsymbol{v}_i^{(c)}$:

$$\boldsymbol{v}_i = \boldsymbol{v}_i^{(w)} \tag{5.2}$$

`char-only (c)` considers only $\boldsymbol{v}_i^{(c)}$ and ignores $\boldsymbol{v}_i^{(w)}$:

$$\boldsymbol{v}_i = \boldsymbol{v}_i^{(c)} \tag{5.3}$$

`concat (cat)` concatenates both word and character-level representations:

$$\boldsymbol{v}_i = [\boldsymbol{v}_i^{(c)}; \boldsymbol{v}_i^{(w)}] \tag{5.4}$$

`scalar gate (sg)` implements the scalar gating mechanism described by Miyamoto and Cho (2016):

$$g_i = \sigma(\boldsymbol{w}^\top \boldsymbol{v}_i^{(w)} + b) \tag{5.5}$$
$$\boldsymbol{v}_i = g_i \boldsymbol{v}_i^{(c)} + (1 - g_i)\boldsymbol{v}_i^{(w)} \tag{5.6}$$

where $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ are trainable parameters, $g_i \in (0, 1)$, and σ is the sigmoid function.
`vector gate (vg)`:

$$\boldsymbol{g}_i = \sigma(\boldsymbol{W}\boldsymbol{v}_i^{(w)} + \boldsymbol{b}) \tag{5.7}$$
$$\boldsymbol{v}_i = \boldsymbol{g}_i \odot \boldsymbol{v}_i^{(c)} + (\mathbf{1} - \boldsymbol{g}_i) \odot \boldsymbol{v}_i^{(w)} \tag{5.8}$$

where $\boldsymbol{W} \in \mathbb{R}^{d \times d}$ and $\boldsymbol{b} \in \mathbb{R}^d$ are trainable parameters, $\boldsymbol{g}_i \in (0, 1)^d$, σ is the element-wise sigmoid function, $\odot$ is the element-wise product for vectors, and $\mathbf{1} \in \mathbb{R}^d$ is a vector of ones.

The vector gate is inspired by Miyamoto and Cho (2016) and Yang et al. (2017), but is different to the former in that the gating mechanism acts upon each dimension of the word and character-level vectors, and different to the latter in that it does not rely on external sources of information for calculating the gating mechanism.

Finally, note that `word only` and `char only` are special cases of both gating mechanisms: $g_i = 0$ (scalar gate) and $\boldsymbol{g}_i = \mathbf{0}$ (vector gate) correspond to `word only`; $g_i = 1$ and $\boldsymbol{g}_i = \mathbf{1}$ correspond to `char only`.

5.3.3 Obtaining Sentence Representations

To enable sentence-level classification we need to obtain a sentence representation from the word vectors $\boldsymbol{v}_i$. We achieved this by using a BiLSTM with max pooling, which

was shown to be a good universal sentence encoding mechanism (Conneau et al., 2017).

Let $\boldsymbol{x} = w_1, \ldots, w_n$, be an input sentence and $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n]$ its matrix representation, where each $\boldsymbol{v}_i$ was obtained by one of the methods described in Section 5.3.2. $\boldsymbol{S} = [\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n]$ is the context-enriched matrix representation of $\boldsymbol{x}$ obtained by feeding $\boldsymbol{V}$ to a BiLSTM of output dimension d_s[4]. Lastly, $\boldsymbol{s} \in \mathbb{R}^{d_s}$ is the final sentence representation of $\boldsymbol{x}$ obtained by max-pooling $\boldsymbol{S}$ along the sequence dimension.

Finally, we initialized the word representations $\boldsymbol{v}_i^{(w)}$ using GloVe embeddings (Pennington et al., 2014), and fine-tuned them during training. In the following section we specify the hyperparameters we used for our experiments.

5.3.4 Hyperparameters

We only considered words that appear at least twice, for each dataset. Those that appeared only once were considered UNK. We used the Treebank Word Tokenizer as implemented in NLTK[5] for tokenizing the training and development datasets.

In the same fashion as Conneau et al. (2017), we used a batch size of 64, an SGD optmizer with an initial learning rate of 0.1, and at each epoch divided the learning rate by 5 if the validation accuracy decreased. We also used gradient clipping when gradients where > 5.

We defined character vector representations as 50-dimensional vectors randomly initialized by sampling from the uniform distribution in the $(-0.05; 0.05)$ range.

The output dimension of the character-level BiLSTM was 300 per direction, and remained of such size after combining forward and backward representations as depicted in eq. 5.1.

Word vector representations where initialized from the 300-dimensional GloVe vectors (Pennington et al., 2014), trained in 840B tokens from the Common Crawl[6], and finetuned during training. Words not present in the GloVe vocabulary where randomly initialized by sampling from the uniform distribution in the $(-0.05; 0.05)$ range.

The input size of the word-level LSTM was 300 for every method except `concat` in which it was 600, and its output was always 2048 per direction, resulting in a 4096-dimensional sentence representation.

[4] $\boldsymbol{s}_i = [\overrightarrow{\boldsymbol{s}_i}; \overleftarrow{\boldsymbol{s}_i}]$ for each i, and both $\overrightarrow{\boldsymbol{s}_i}$ and $\overleftarrow{\boldsymbol{s}_i} \in \mathbb{R}^{\frac{d_s}{2}}$.

[5] https://www.nltk.org/

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

5.4 Experiments

5.4.1 Experimental Setup

We trained our models for solving the Natural Language Inference (NLI) task in two datasets, SNLI (Bowman et al., 2015) and MultiNLI (Williams et al., 2018), and validated them in each corresponding development set (including the matched and mismatched development sets of MultiNLI).

For each dataset-method combination we trained 7 models initialized with different random seeds, and saved each when it reached its best validation accuracy[7]. We then evaluated the quality of each trained model's word representations $\boldsymbol{v}_i$ in 10 word similarity tasks, using the system created by Jastrzebski et al. (2017)[8].

Finally, we fed these obtained word vectors to a BiLSTM with max-pooling and evaluated the final sentence representations in 11 downstream transfer tasks (Conneau et al., 2017; Subramanian et al., 2018).

5.4.2 Datasets

Word-level Semantic Similarity A desirable property of vector representations of words is that semantically similar words should have similar vector representations. Assessing whether a set of word representations possesses this quality is referred to as the semantic similarity task. This is the most widely-used evaluation method for evaluating word representations, despite its shortcomings (Faruqui et al., 2016).

This task consists of comparing the similarity between word vectors measured by a distance metric (usually cosine distance), with a similarity score obtained from human judgements. High correlation between these similarities is an indicator of good performance.

A problem with this formulation though, is that the definition of "similarity" often confounds the meaning of both *similarity* and *relatedness*. For example, *cup* and *tea* are related but dissimilar words, and this type of distinction is not always clear (Agirre et al., 2009; Hill et al., 2015).

To face the previous problem, we tested our methods in a wide variety of datasets, including some that explicitly model relatedness (WS353R), some that explicitly consider similarity (WS353S, SimLex999, SimVerb3500), and some where the distinction is not clear (MEN, MTurk287, MTurk771, RG, WS353). We also included the RareWords (RW) dataset for evaluating the quality of rare word representations. In the next sec-

[7] We found that models validated on the matched development set of MultiNLI, rather than the mismatched, yielded best results, although the differences were not statistically significant.

[8] https://github.com/kudkudak/word-embeddings-benchmarks/tree/8fd0489

tions we describe both word and sentence level datasets in detail.

Word Similarity Datasets

Dataset	Reference	URL
MEN	Bruni et al. (2014)	https://staff.fnwi.uva.nl/e.bruni/MEN
MTurk287	Radinsky et al. (2011)	https://git.io/fhQA8 (Unofficial)
MTurk771	Halawi et al. (2012)	http://www2.mta.ac.il/~gideon/mturk771.html
RG	Rubenstein and Goodenough (1965)	https://git.io/fhQAB (Unofficial)
RareWords (RW)	Luong et al. (2013)	https://nlp.stanford.edu/~lmthang/morphoNLM/
SimLex999	Hill et al. (2015)	https://fh295.github.io/simlex.html
SimVerb3500	Gerz et al. (2016)	http://people.ds.cam.ac.uk/dsg40/simverb.html
WS353	Finkelstein et al. (2002)	http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/
WS353R	Agirre et al. (2009)	http://alfonseca.org/eng/research/wordsim353.html
WS353S	Agirre et al. (2009)	http://alfonseca.org/eng/research/wordsim353.html

Table 5.1 – Word similarity and relatedness datasets.

Table 5.1 lists the word-similarity datasets and their corresponding reference. As mentioned in Section 5.4.2, all the word-similarity datasets contain pairs of words annotated with similarity or relatedness scores, although this difference is not always explicit. Below we provide some details for each.

MEN contains 3000 annotated word pairs with integer scores ranging from 0 to 50. Words correspond to image labels appearing in the ESP-Game[9] and MIRFLICKR-1M[10] image datasets.

MTurk287 contains 287 annotated pairs with scores ranging from 1.0 to 5.0. It was created from words appearing in both DBpedia and in news articles from The New York Times.

MTurk771 contains 771 annotated pairs with scores ranging from 1.0 to 5.0, with words having synonymy, holonymy or meronymy relationships sampled from WordNet (Fellbaum, 1998).

RG contains 65 annotated pairs with scores ranging from 0.0 to 4.0 representing "similarity of meaning".

RW contains 2034 pairs of words annotated with similarity scores in a scale from 0 to 10. The words included in this dataset were obtained from Wikipedia based on their frequency, and later filtered depending on their WordNet synsets, including synonymy, hyperonymy, hyponymy, holonymy and meronymy. This dataset was created with the purpose of testing how well models can represent rare and complex words.

SimLex999 contains 999 word pairs annotated with similarity scores ranging from 0 to 10. In this case the authors explicitly considered similarity and not relatedness, addressing the shortcomings of datasets that do not, such as MEN and WS353. Words include nouns, adjectives and verbs.

[9] http://www.cs.cmu.edu/~biglou/resources/
[10] http://press.liacs.nl/mirflickr/

SimVerb3500 contains 3500 verb pairs annotated with similarity scores ranging from 0 to 10. Verbs were obtained from the USF free association database (Nelson et al., 2004), and VerbNet (Kipper et al., 2008). This dataset was created to address the lack of representativity of verbs in SimLex999, and the fact that, at the time of creation, the best performing models had already surpassed inter-annotator agreement in verb similarity evaluation resources. Like SimLex999, this dataset also explicitly considers similarity as opposed to relatedness.

WS353 contains 353 word pairs annotated with similarity scores from 0 to 10.

WS353R is a subset of WS353 containing 252 word pairs annotated with relatedness scores. This dataset was created by asking humans to classify each WS353 word pair into one of the following classes: synonyms, antonyms, identical, hyperonym-hyponym, hyponym-hyperonym, holonym-meronym, meronym-holonym, and none-of-the-above. These annotations were later used to group the pairs into: *similar* pairs (synonyms, antonyms, identical, hyperonym-hyponym, and hyponym-hyperonym), *related* pairs (holonym-meronym, meronym-holonym, and none-of-the-above with a human similarity score greater than 5), and *unrelated* pairs (classified as none-of-the-above with a similarity score less than or equal to 5). This dataset is composed by the union of related and unrelated pairs.

WS353S is another subset of WS353 containing 203 word pairs annotated with similarity scores. This dataset is composed by the union of similar and unrelated pairs, as described previously.

5.4.3 Sentence Evaluation Datasets

Dataset	Reference	URL
CR	Hu and Liu (2004)	https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html#datasets
MPQA	Wiebe et al. (2005)	https://mpqa.cs.pitt.edu/corpora/mpqa_corpus/
MR	Pang and Lee (2005)	http://www.cs.cornell.edu/people/pabo/movie-review-data/
SST2	Arora et al. (2017)	https://github.com/PrincetonML/SIF/tree/master/data
SST5	See caption.	https://git.io/fhQAV
SUBJ	Pang and Lee (2004)	http://www.cs.cornell.edu/people/pabo/movie-review-data/
TREC	Li and Roth (2002)	http://cogcomp.org/Data/QA/QC/
SICKE	Marelli et al. (2014)	http://clic.cimec.unitn.it/composes/sick.html
SICKR	Marelli et al. (2014)	http://clic.cimec.unitn.it/composes/sick.html
STS16	Agirre et al. (2016)	http://ixa2.si.ehu.es/stswiki/index.php/Main_Page
STSB	Cer et al. (2017)	http://ixa2.si.ehu.es/stswiki/index.php/STSbenchmark

Table 5.2 – Sentence representation evaluation datasets. SST5 was obtained from a GitHub repository with no associated peer-reviewed work.

Table 5.2 lists the sentence-level evaluation datasets used in this paper. The provided URLs correspond to the original sources, and not necessarily to the URLs where SentEval[11] got the data from[12].

[11] https://github.com/facebookresearch/SentEval/tree/906b34a

[12] A list of the data used by SentEval can be found in its data setup script: https://git.io/fhQpq

The version of the CR, MPQA, MR, and SUBJ datasets used in this paper were the ones preprocessed by Wang and Manning (2012)[13]. Both SST2 and SST5 correspond to preprocessed versions of the Stanford Sentiment Treebank (SST) dataset by Socher et al. (2013)[14]. SST2 corresponds to a subset of SST used by Arora et al. (2017) containing flat representations of sentences annotated with binary sentiment labels, and SST5 to another subset annotated with more fine-grained sentiment labels (very negative, negative, neutral, positive, very positive).

Sentence-level Evaluation Tasks Unlike word-level representations, there is no consensus on the desirable properties sentence representations should have. In response to this, Conneau et al. (2017) created SentEval[15], a sentence representation evaluation benchmark designed for assessing how well sentence representations perform in various downstream tasks (Conneau and Kiela, 2018).

Some of the datasets included in SentEval correspond to sentiment classification (CR, MPQA, MR, SST2, and SST5), subjectivity classification (SUBJ), question-type classification (TREC), recognizing textual entailment (SICK E), estimating semantic relatedness (SICK R), and measuring textual semantic similarity (STS16, STSB). The datasets are described by Conneau et al. (2017), and we provide pointers to their original sources in Table 5.2.

To evaluate these sentence representations SentEval trained a linear model on top of them, and evaluated their performance in the validation sets accompanying each dataset. The only exception was the STS16 task, in which our representations were evaluated directly.

5.5 Word-level Evaluation

5.5.1 Word Similarity

Table 5.3 shows the quality of word representations in terms of the correlation between word similarity scores obtained by the proposed models and word similarity scores defined by humans.

First, we can see that for each task, `character only` models had significantly worse performance than every other model trained on the same dataset. The most likely explanation for this is that these models are the only ones that need to learn word representations from scratch, since they have no access to the global semantic knowledge encoded by the GloVe embeddings.

[13] https://nlp.stanford.edu/~sidaw/home/projects:nbsvm
[14] https://nlp.stanford.edu/sentiment/
[15] https://github.com/facebookresearch/SentEval/tree/906b34a

		MEN	MT287	MT771	RG65	RW	SL999	SV3500	WS	WSR	WSS
SNLI	w	71.78	35.40	49.05	61.80	18.43	19.17	10.32	39.27	28.01	53.42
	c	9.85	-5.65	0.82	-5.28	17.81	0.86	2.76	-2.20	0.20	-3.87
	cat	71.91	**35.52**	48.84	62.12	18.46	19.10	10.21	39.35	28.16	53.40
	sg	70.49	34.49	46.15	59.75	18.24	17.20	8.73	35.86	23.48	50.83
	vg	**80.00**	32.54	**62.09**	**68.90**	**20.76**	**37.70**	**20.45**	**54.72**	**47.24**	**65.60**
MNLI	w	68.76	50.15	68.81	65.83	18.43	42.21	25.18	61.10	58.21	70.17
	c	4.84	0.06	1.95	-0.06	12.18	3.01	1.52	-4.68	-3.63	-3.65
	cat	68.77	50.40	68.77	65.92	18.35	42.22	25.12	61.15	58.26	70.21
	sg	67.66	49.58	68.29	64.84	18.36	41.81	24.57	60.13	57.09	69.41
	vg	**76.69**	**56.06**	**70.13**	**69.00**	**25.35**	**48.40**	**35.12**	**68.91**	**64.70**	**77.23**

Table 5.3 – Word-level evaluation results. Each value corresponds to average Pearson correlation of 7 identical models initialized with different random seeds. Correlations were scaled to the $[-100; 100]$ range for easier reading. **Bold** values represent the best method per training dataset, per task; **underlined** values represent the best-performing method per task, independent of training dataset. For each task and dataset, every best-performing method was significantly different to other methods ($p < 0.05$), except for `w` trained in SNLI at the MTurk287 task. Statistical significance was obtained with a two-sided Welch's t-test for two independent samples without assuming equal variance (Welch, 1947). MT corresponds to MTurk, SL999 Corresponds to SimLex999 and SV3500 to SimVerb3500

Further, **bold** results show the overall trend that `vector gates` outperformed the other methods regardless of training dataset. This implies that learning how to combine character and word-level representations at the dimension level produces word vector representations that capture a notion of word similarity and relatedness that is closer to that of humans.

Additionally, results from the MNLI row in general, and **underlined** results in particular, show that training on MultiNLI produces word representations better at capturing word similarity. This is probably due to MultiNLI data being richer than that of SNLI. Indeed, MultiNLI data was gathered from various sources (novels, reports, letters, and telephone conversations, among others), rather than the single image captions dataset from which SNLI was created.

Exceptions to the previous rule are models evaluated in MEN and RW. The former case can be explained by the MEN dataset[16] containing only words that appear as image labels in the ESP-Game[17] and MIRFLICKR-1M[18] image datasets (Bruni et al., 2014), and therefore having data that is more closely distributed to SNLI than to MultiNLI.

More notably, in the RareWords dataset (Luong et al., 2013), the `word only`, `concat`, and `scalar gate` methods performed equally, despite having been trained in different datasets ($p > 0.1$), and the `char only` method performed significantly worse when trained in MultiNLI. The `vector gate`, however, performed significantly better than its counterpart trained in SNLI. These facts provide evidence that this method is capable of capturing linguistic phenomena that the other methods are unable to model.

[16] https://staff.fni.uva.nl/e.bruni/MEN
[17] http://www.cs.cmu.edu/~biglou/resources/
[18] http://press.liacs.nl/mirflickr/

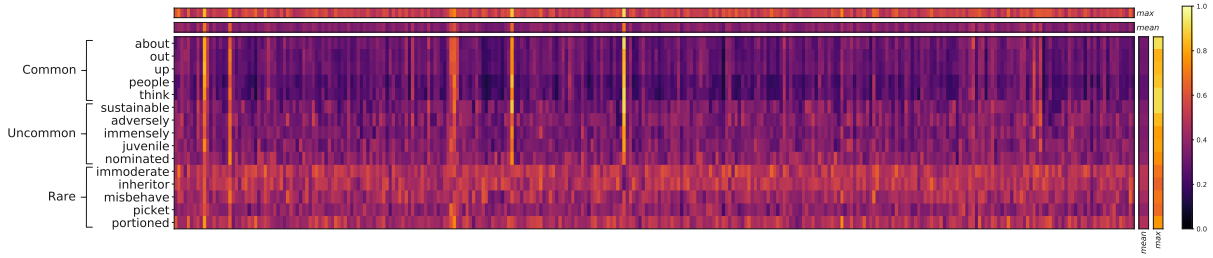


Figure 5.2 – Visualization of gating values for 5 common words (freq. $\sim$ 20000), 5 uncommon words (freq. $\sim$ 60), and 5 rare words (freq. $\sim$ 2), appearing in both the RW and MultiNLI datasets.

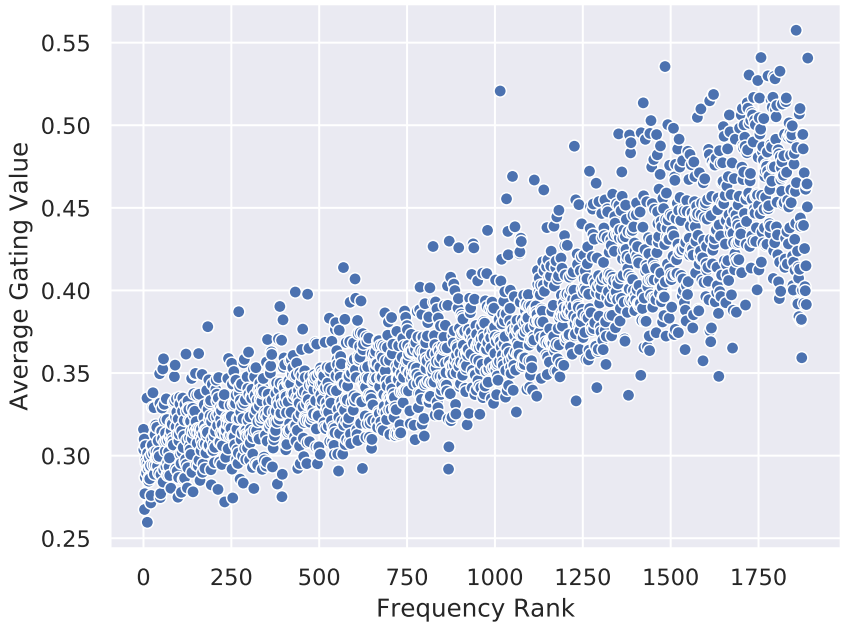


Figure 5.3 – Average gating values for words appearing in both RW and MultiNLI. Words are sorted by decreasing frequency in MultiNLI.

5.5.2 Word Frequencies and Gating Values

Figure 5.2 shows that for more common words the `vector gate` mechanism tends to favor only a few dimensions while keeping a low average gating value across dimensions. On the other hand, values are greater and more homogeneous across dimensions in rarer words. Further, Fig. 5.3 shows this mechanism assigns, on average, a greater gating value to less frequent words, confirming the findings by Miyamoto and Cho (2016), and Yang et al. (2017).

In other words, the less frequent the word, the more this mechanism allows the character-level representation to influence the final word representation, as shown by Eq. (5.8). A possible interpretation of this result is that exploiting character information becomes increasingly necessary as word-level representations' quality decrease.

Another observable trend in both figures is that gating values tend to be low on

average. Indeed, it is possible to see in Fig. 5.3 that the average gating values range from 0.26 to 0.56. This result corroborates the findings by Miyamoto and Cho (2016), stating that setting $g = 0.25$ in Eq. (5.6), was better than setting it to higher values.

In summary, the gating mechanisms learn how to compensate the lack of expressivity of underrepresented words by selectively combining their representations with those of characters.

5.6 Sentence-level Evaluation

Table 5.4 and Table 5.4 show the impact that different methods for combining character and word-level word representations have in the quality of the sentence representations produced by our models.

We can observe the same trend mentioned in Section 5.5.1, and highlighted by the difference between **bold** values, that models trained in MultiNLI performed better than those trained in SNLI at a statistically significant level, confirming the findings of Conneau et al. (2017). In other words, training sentence encoders on MultiNLI yields more general sentence representations than doing so on SNLI.

The two exceptions to the previous trend, SICKE and SICKR, benefited more from models trained on SNLI. We hypothesize this is again due to both SNLI and SICK (Marelli et al., 2014) having similar data distributions[19].

Additionally, there was no method that significantly outperformed the `word only` baseline in classification tasks. This means that the added expressivity offered by explicitly modeling characters, be it through concatenation or gating, was not significantly better than simply fine-tuning the pre-trained GloVe embeddings for this type of task. We hypothesize this is due to the conflation of two effects. First, the fact that morphological processes might not encode important information for solving these tasks; and second, that SNLI and MultiNLI belong to domains that are too dissimilar to the domains in which the sentence representations are being tested.

On the other hand, the `vector gate` significantly outperformed every other method in the STSB task when trained in both datasets, and in the STS16 task when trained in SNLI. This again hints at this method being capable of modeling phenomena at the word level, resulting in improved semantic representations at the sentence level.

[19]SICK was created from Flickr-8k (Rashtchian et al., 2010), and SNLI from its expanded version: Flickr30k (Young et al., 2014).

		CR	MPQA	MR	SST2	SST5	SUBJ	TREC
SNLI	w	80.50	84.59	74.18	78.86	42.33	**90.38**	**86.83**
	c	74.90*	78.86*	65.93*	69.42*	35.56*	82.97*	83.31*
	cat	80.44	84.66	74.31	78.37	41.34*	90.28	85.80*
	sg	**80.59**	84.60	**74.49**	**79.04**	41.63*	90.16	86.00
	vg	80.42	**84.66**	74.26	78.87	**42.38**	90.07	85.97
MNLI	w	83.80	**89.13**	79.05	83.38	45.21	91.79	89.23
	c	70.23*	72.19*	62.83*	64.55*	32.47*	79.49*	74.74*
	cat	**83.96**	89.12	**79.23**	83.70	45.08*	**91.92**	**90.03**
	sg	83.88	89.06	79.22	83.71	45.26	91.66*	88.83*
	vg	83.45*	89.05	79.13	**83.87**	**45.88**	91.55*	89.49

Table 5.4 – Experimental results on classification tasks. Each value shown in the table is the average result of 7 identical models initialized with different random seeds. Values represent accuracy (%). **Bold** values represent the best method per training dataset, per task; **underlined** values represent the best-performing method per task, independent of training dataset. Values marked with an asterisk (*) are significantly different to the average performance of the best model trained on the same dataset ($p < 0.05$). Results for every best-performing method trained on one dataset are significantly different to the best-performing method trained on the other. Statistical significance was obtained in the same way as described in Table 5.3.

		Entailment	Relatedness	Semantic Textual Similarity	
		SICKE	**SICKR**†	**STS16**†	**STSB**†
SNLI	w	86.37	88.52	59.90*	71.29*
	c	84.13*	83.89*	59.33*	67.20*
	cat	**86.40**	88.44	59.90*	71.24*
	sg	86.10*	**88.57**	60.05*	71.34*
	vg	85.67	88.31*	**60.92**	**71.99**
MNLI	w	84.92	86.33	66.08	71.96*
	c	81.53*	75.92*	51.47*	61.74*
	cat	**85.06**	86.45	**66.17**	71.82*
	sg	84.96	86.40	65.49*	71.87*
	vg	84.82	**86.50**	65.75	**72.82**

Table 5.5 – Experimental results on semantic tasks. The same conditions specified in Table 5.4 apply here. In addition, columns marked with † represent Pearson correlation scaled to the range $[-100, 100]$ for easier reading.

5.7 Relationship Between Word- and Sentence-level Evaluation Tasks

It is clear that the better performance the `vector gate` had in word similarity tasks did not translate into overall better performance in downstream tasks. This confirms previous findings indicating that intrinsic word evaluation metrics are not good predictors of downstream performance (Chiu et al., 2016; Faruqui et al., 2016;

Gladkova and Drozd, 2016; Tsvetkov et al., 2015).

Figure 5.4b shows that the word representations created by the `vector gate` trained in MultiNLI had positively-correlated results within several word-similarity tasks. This hints at the generality of the word representations created by this method when modeling similarity and relatedness.

However, the same cannot be said about sentence-level evaluation performance; there is no clear correlation between word similarity tasks and sentence-evaluation tasks. This is clearly illustrated by performance in the STSBenchmark, the only in which the `vector gate` was significantly superior, not being correlated with performance in any word-similarity dataset. This can be interpreted simply as word-level representations capturing word-similarity not being a sufficient condition for good performance in sentence-level tasks.

In general, Fig. 5.4 shows that there are no general correlation effects spanning both training datasets and combination mechanisms. For example, Fig. 5.4a shows that, for both `word-only` and `concat` models trained in SNLI, performance in word similarity tasks correlates positively with performance in most sentence evaluation tasks, however, this does not happen as clearly for the same models trained in MultiNLI (Fig. 5.4b).

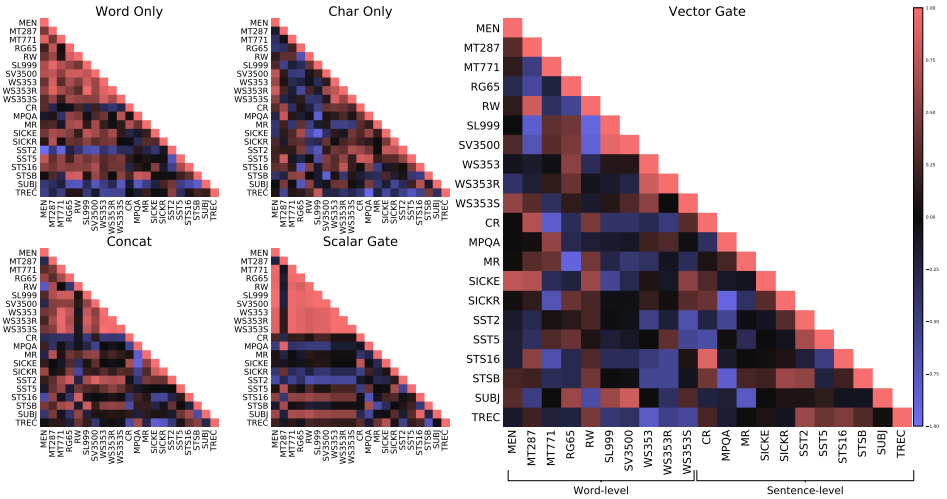
5.8 Related Work

5.8.1 Gating Mechanisms for Combining Characters and Word Representations

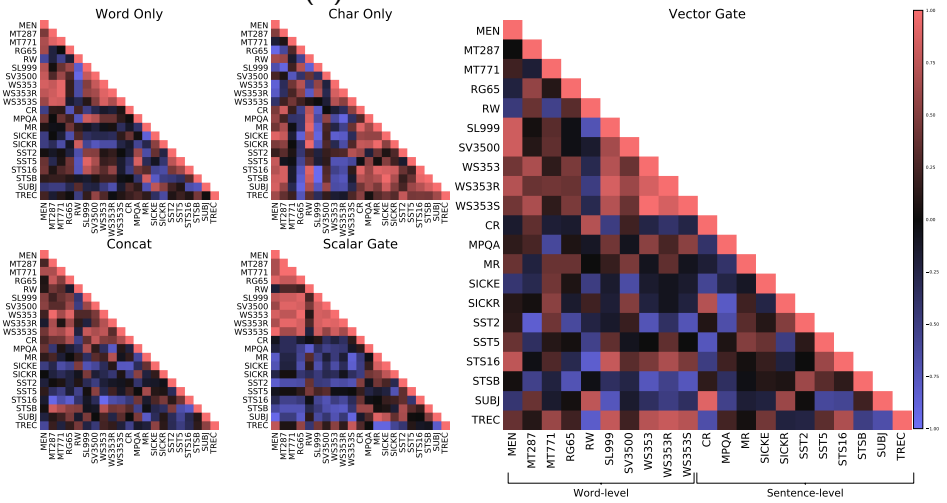
To the best of our knowledge, there are only two recent works that specifically study how to combine word and subword-level vector representations.

Miyamoto and Cho (2016) propose to use a trainable scalar gating mechanism capable of learning a weighting scheme for combining character-level and word-level representations. They compared their proposed method to manually weighting both levels; using characters only; words only; or their concatenation. They found that in some cases taking the weighted average of the character and word-level representations, with weights of 25% and 75% respectively, yielded the best results, while in others the learned scalar gate performed better.

Yang et al. (2017) further expand the gating concept by making the mechanism work at a finer-grained level, learning how to weight each vector's dimensions independently, conditioned on external word-level features such as part-of-speech and named-entity tags. Similarly, they compared their proposed mechanism to using words only, characters only, and a concatenation of both, with and without external features. They found



(a) Models trained in SNLI.



(b) Models trained in MultiNLI.

Figure 5.4 – Spearman correlation between performances in word and sentence level evaluation tasks.

that their vector gate performed better than the other methods in all the reported tasks, and beat the state of the art in two reading comprehension tasks.

Both works showed that the gating mechanisms assigned greater importance to character-level representations in rare words, and to word-level representations in common ones, reaffirming the previous findings that subword structures in general, and characters in particular, are beneficial for modeling uncommon words.

5.8.2 Sentence Representation Learning

The problem of representing sentences as fixed-length vectors has been widely studied.

Zhao et al. (2015) suggested a self-adaptive hierarchical model that gradually composes words into intermediate phrase representations, and adaptively selects specific hierarchical levels for specific tasks. Kiros et al. (2015) proposed an encoder-decoder model trained by attempting to reconstruct the surrounding sentences of an encoded passage, in a fashion similar to Skip-gram (Mikolov et al., 2013b). Hill et al. (2016) overcame the previous model's need for ordered training sentences by using autoencoders for creating the sentence representations. Jernite et al. (2017) implemented a model simpler and faster to train than the previous two, while having competitive performance. Similar to Kiros et al. (2015), Gan et al. (2017) suggested predicting future sentences with a hierarchical CNN-LSTM encoder.

Conneau et al. (2017) trained several sentence encoding architectures on a combination of the SNLI and MultiNLI datasets, and showed that a BiLSTM with max-pooling was the best at producing highly transferable sentence representations. More recently, Subramanian et al. (2018) empirically showed that sentence representations created in a multi-task setting (Collobert and Weston, 2008), performed increasingly better the more tasks they were trained in. Zhang et al. (2018) proposed using an autoencoder that relies on multi-head self-attention over the concatenation of the max and mean pooled encoder outputs for producing sentence representations. Finally, Wieting and Kiela (2019) show that modern sentence embedding methods are not vastly superior to random methods.

The works mentioned so far usually evaluate the quality of the produced sentence representations in sentence-level downstream tasks. Common benchmarks grouping these kind of tasks include SentEval (Conneau and Kiela, 2018), and GLUE (Wang et al., 2019). Another trend, however, is to *probe* sentence representations to understand what linguistic phenomena they encode (Adi et al., 2017; Conneau et al., 2018a; Linzen et al., 2016; Perone et al., 2018; Zhu et al., 2018).

5.8.3 General Feature-wise Transformations

Dumoulin et al. (2018) provide a review on feature-wise transformation methods, of which the mechanisms presented in this paper form a part of. In a few words, the g parameter, in both `scalar gate` and `vector gate` mechanisms, can be understood as a *scaling parameter* limited to the $(0, 1)$ range and conditioned on word representations, whereas adding the scaled $\boldsymbol{v}_i^{(c)}$ and $\boldsymbol{v}_i^{(w)}$ representations can be seen as *biasing* word representations conditioned on character representations.

The previous review extends the work by Perez et al. (2018), which describes the Feature-wise Linear Modulation (FiLM) framework as a generalization of Conditional Normalization methods, and apply it in visual reasoning tasks. Some of the reported findings are that, in general, scaling has greater impact than biasing, and that in a setting similar to the `scalar gate`, limiting the scaling parameter to $(0, 1)$ hurt performance. Future decisions involving the design of mechanisms for combining character and word-level representations should be informed by these insights.

5.9 Conclusions

We presented an empirical study showing the effect that different ways of combining character and word representations has in word-level and sentence-level evaluation tasks.

We showed that a vector gate performed consistently better across a variety of word similarity and relatedness tasks. Additionally, despite showing inconsistent results in sentence evaluation tasks, it performed significantly better than the other methods in semantic similarity tasks.

We further showed through this mechanism, that learning character-level representations is always beneficial, and becomes increasingly so with less common words.

In the future it would be interesting to study how the choice of mechanism for combining subword and word representations affects the more recent language-model-based pretraining methods such as ELMo (Peters et al., 2018b), GPT (Radford et al., 2018, 2019) and BERT (Devlin et al., 2019).

Chapter 6

The Interplay Between Gating Mechanisms and Adversarial Learning

6.1 Introduction

Vector representations of words are supposed to capture syntactic and semantic phenomena occurring between words. The way to obtain these vary from purely statistical-based approaches, such as the GloVe embeddings (Pennington et al., 2014), to neural approaches such as word2vec (Mikolov et al., 2013a) and FastText (Bojanowski et al., 2017). GloVe and word2vec capture paradigmatic information from corpus level interactions, following the distributional hypothesis; while the FastText further incorporates modeling phenomena occurring within words.

In Chapter 5 we obtained insights on the question of how combining word representations obtained by different methods affected downstream performance in several NLP tasks. We showed that explicitly learning how to combine word representations obtained with different methods, i.e. using a vector gate, provided consistent performance increases in word-level semantic similarity tasks. We also showed that these increases did not propagate to downstream tasks, with a single exception, the STSBenchmark. This means that for most tasks, properly modeling lexical semantic similarity is not a sufficient condition for increased performance.

This does not mean, however, that modeling other phenomena at the word level could not be beneficial. In this chapter we attempt to incorporate character-level information into word representations, guided by the intuition that since they represent the same underlying meaning, character-derived and pre-trained word representations should be similar. In other words, vectors representing the same word should be similar

no matter the method used for obtaining them.

6.2 Preliminaries

In Chapter 5 we showed the differences in performance between models that simply concatenate character and word level representations (`cat` models), and those that learn how to combine them through a vector gate (`vg` model). To motivate this chapter, we look deeper into the representations learned by these modalities.

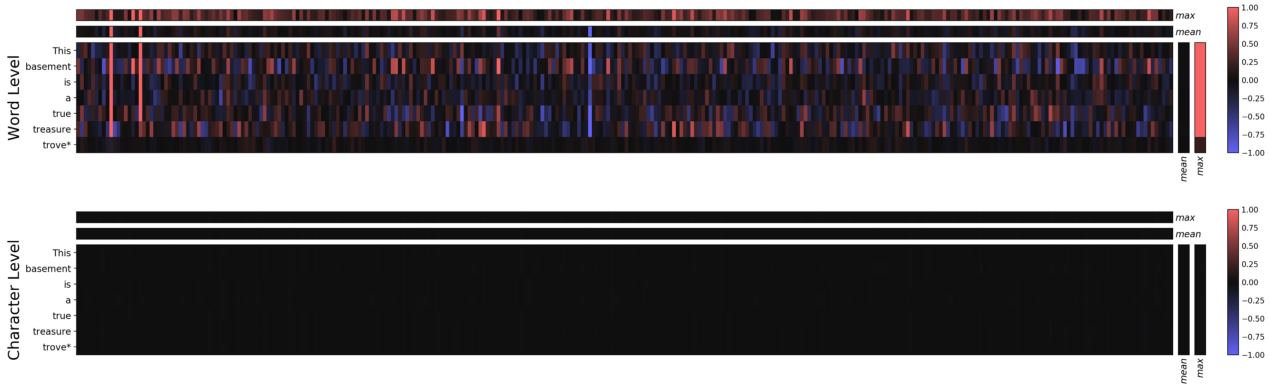


Figure 6.1 – Token level representations of an example sentence created by a `cat` model trained on SNLI. * denotes an UNK word.

Figure 6.1 shows how a `cat` model encodes a simple sentence at the word and character levels. The main rectangles represent the encoded sentence, where the vertical axis represents the words, and the horizontal one their vector representations. We can see that the activations in the character-level setting (average norm: ~ 0.3), are barely visible when using the same scale as the GloVe word vectors (average norm: ~ 6).

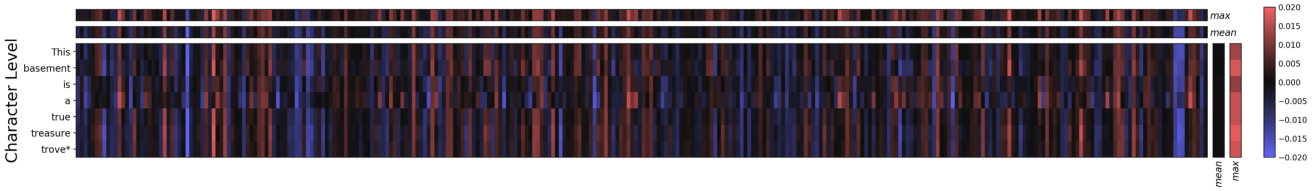


Figure 6.2 – Rescaled character-level representations created by a `cat` model trained on SNLI

Figure 6.2 shows that character representations are in fact being learned, but at a much smaller scale. At this point it could be argued that simply normalizing representations would get rid of this phenomenon. Preliminary experiments showed that this was in fact the case, but performance in downstream tasks dropped drastically, which is why I did not pursue this direction further.

On the other hand Fig. 6.3 shows that character-level representations learned by a `vg` model are closer in magnitude to word-level ones, as their activations are visible,

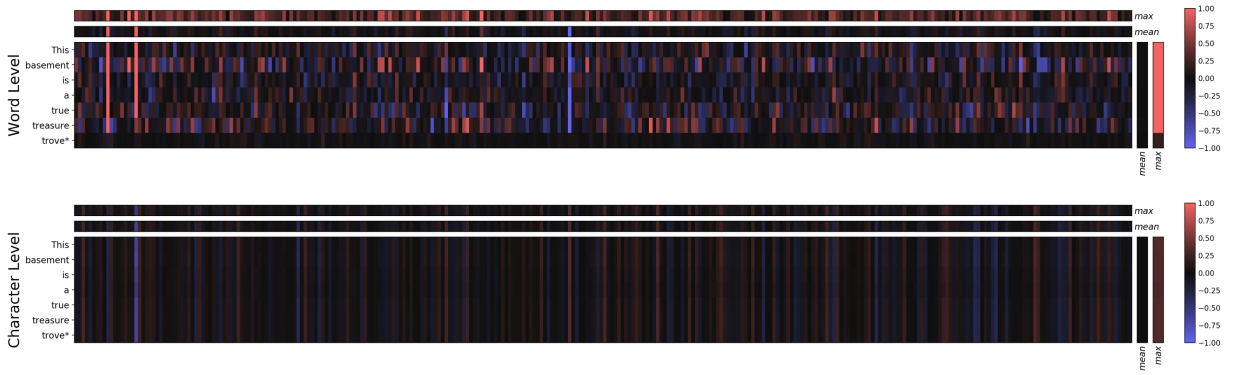


Figure 6.3 – Token level representations of an example sentence created by a `vg` model trained on SNLI

but they are still smaller in general. It is also possible to observe that a limited number of specific dimensions seem to be significant across words, resulting in a clearly visible banded pattern that occurs only to a lesser extent in the character representations obtained by the `cat` model.

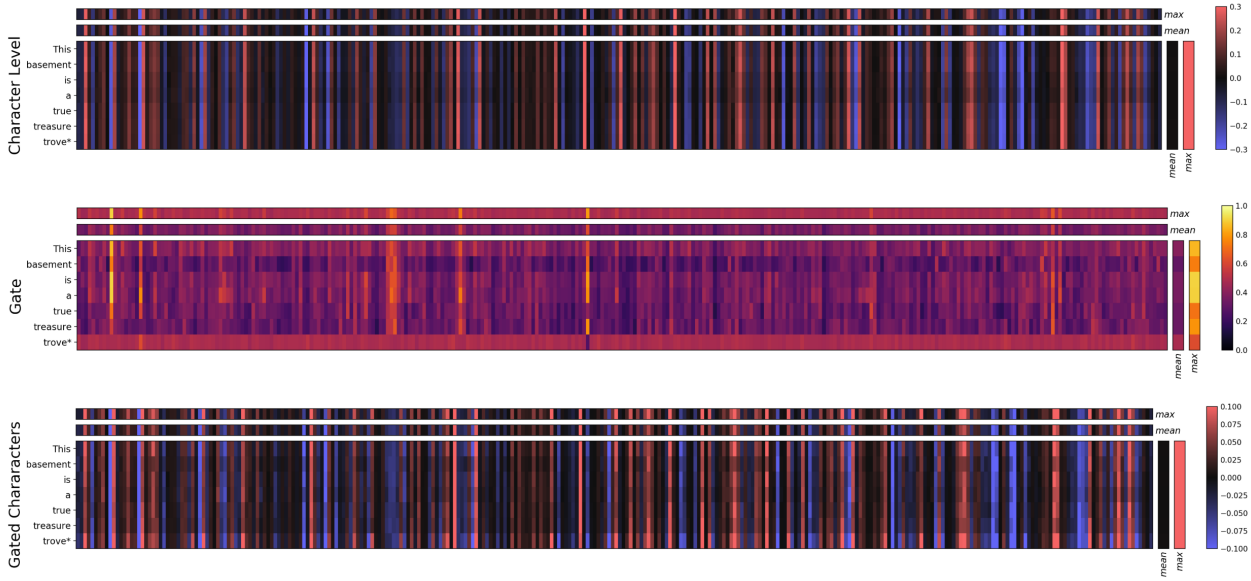
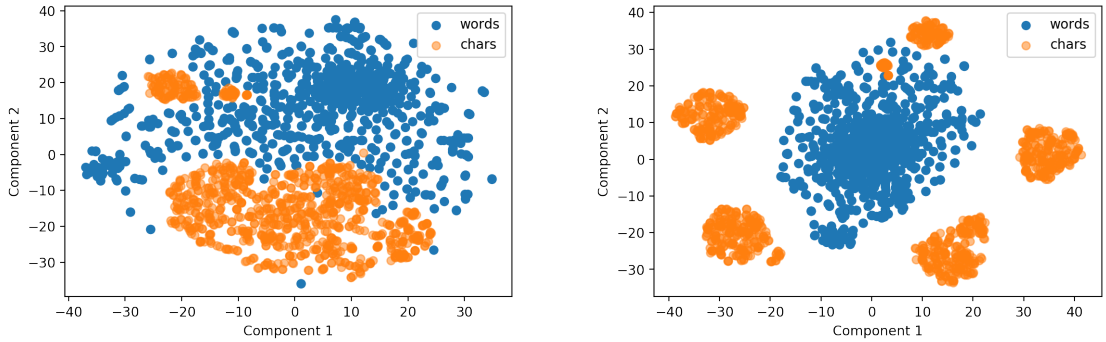


Figure 6.4 – Character-level representations, vector gate and gated character representations obtained by `vg` model trained on SNLI. Gated characters are obtained by multiplying element-wise the vector gate values with the character-level representations: $\boldsymbol{g} \odot \boldsymbol{w}^{(c)}$. See Section 5.3.2 for more details. Also note that the scale for the gated characters is different to that of the un-gated ones; this was done deliberately to highlight the differences between them.

Figure 6.4 shows this phenomenon more clearly. A possible explanation is that the interactions between the gating mechanism and the character-level representations might have enough modeling power allowing the model to create similar character representations for different words and relying on the gating mechanism for modeling their differences. Indeed Fig. 6.4 shows that the gate, learning a non-banded pattern, ends up producing gated character representations that are less banded than the ones output

by the encoder. Another explanation is that, similar to what Bahdanau et al. (2015) hypothesize, the gating mechanism is relieving the character encoder from the burden of having to encode all the information in the character-level word representations.



(a) `cat` model **(b)** `vg` model

Figure 6.5 – t-SNE projection of the representations for the 1000 most frequent words of SNLI

Further, Fig. 6.5 displays a 2D t-SNE projection (van der Maaten and Hinton, 2008) of character and word level representations where it is possible to see that character representations cluster together among themselves, but are clearly separated from word representations. This could be interpreted as character and word representations being different enough that the simple t-SNE projection is capable of telling them apart.

All this being said, it is clear that the character and word representations being created by these models differ greatly. Our hypothesis is that they might benefit from being forced closer together, given that they represent the same underlying meaning.

6.3 Related Work

Miyamoto and Cho (2016) and Yang et al. (2017), introduced in Chapter 5 propose gating mechanisms for combining character and word representations.

Rei et al. (2016) implemented a gating mechanism similar to the vector gate introduced in Chapter 5, with a slightly different gating architecture, and an added loss component for making non OOV character-level word representations similar to pre-trained word representations. During training, the cosine distance between character- and word-level word representations is minimized, but only character-level gradients are backpropagated. This effectively forces the character-level encoder to produce representations similar to the pre-trained word embeddings, without interfering with them. The downside to their approach is that the authors do not report ablation results on adding this loss term, so it is impossible to tell how it contributes to the overall performance. Further, the choice of cosine distance as distance metric is not well justified, as it ignores the vectors' norms, which might have a significant impact in representation

quality.

Ganin et al. (2016) introduce the "Gradient Reversal Layer" (GRL), a simple method for forcing Neural Network representations from different domains to be similar to each other. This mechanism implements the insight from Domain Adaptation that in order to have effective domain transfer, the features representing elements from different domains should not contain information about the domains themselves. In other words, successful domain adaptation relies on features being domain-invariant. For example, a successful speech recognition system should be able to transcribe a speech signal into text regardless of the speaker and the characteristics that identify them, such as the pitch of their voice. Therefore, these characteristics should not be encoded into the features representing the signal.

To achieve this, the authors propose pairing a gradient reversal layer with a domain classifier whose task is to discriminate between domains. During backpropagation, the gradients beneficial to the discriminator, i.e. those useful for discriminating between domains, are reversed by the GRL, effectively updating the feature extractors with gradients working against distinguishable features. This results in feature extractors creating feature representations that are difficult for the domain discriminator to classify.

On a similar vein, Adi et al. (2018) found out that reversing the gradient was the same as not doing so, or in other words, representations learned by a deep architecture on big data created indistinguishable features (the potentially discriminating factor was the speaker, in the context of speech recognition).

To test test how "unrecognizable" their features were, they trained classifiers on top on the representations created by a baseline model. They found out that the earliest features (those closest to the data) were good at predicting the speaker (the domain), but the later ones were not, meaning that the architecture itself was able to generate domain-invariant features.

6.4 Method

Our goal is to make character-derived and pre-trained word representations closer to each other, with the hope that this inductive bias will produce better final word representations. As an additional requirement, we want to enforce this closeness without specifying any particular distance metric, like Rei et al. (2016) did when minimizing the cosine distance.

In both `concatenation` (`cat`) and `vector gate` (`vg`) settings, introduced in Chapter 5, we study the effect of pushing character-derived and pre-trained closer by means

of adversarial regularization. Figure 6.6 shows the adversarial regularization setting, inspired by Ganin et al. (2016).

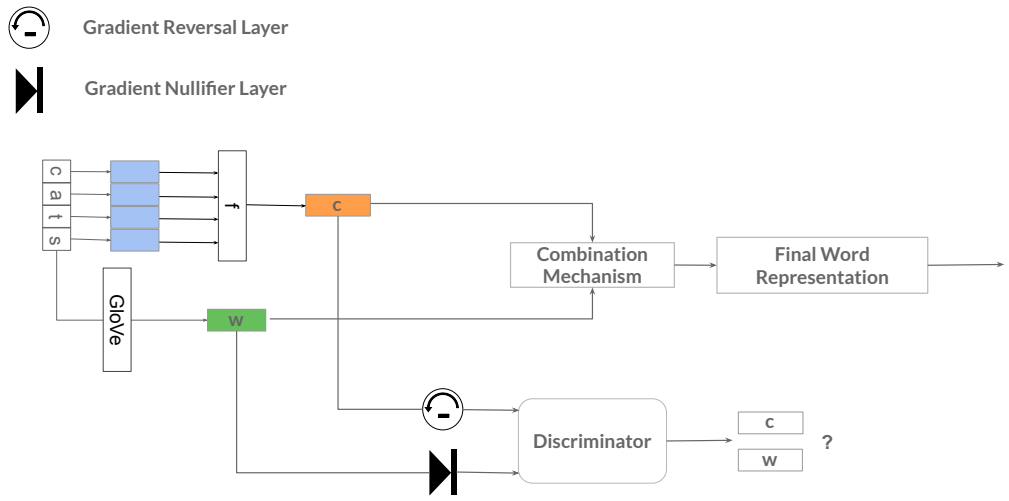


Figure 6.6 – Adversarial Regularization Architecture

The main purpose of adversarial regularization in this context is to make character-derived representations harder for the discriminator to distinguish from word-level ones, or in other words to make them similar. To achieve this we train a discriminator designed for telling the difference between word- and character-level representations, while manipulating the gradients during backpropagation.

Data flows normally during the forward pass, the discriminator produces an output from which we obtain the binary cross-entropy when comparing it to the real class of each representation. This loss value is then backpropagated normally through this component, which will make it better at classifying them. After the gradients produced by the word-level representations flow back through the discriminator we multiply them by 0, effectively nullifying their effect on the rest of the architecture. On the other hand, the gradients produced by the character-level representations are flipped (multiplied by -1), right after flowing back from the discriminator. This will make the previous components from the architecture, specifically the f layer and the character embeddings themselves, produce character-level representations that are harder for the discriminator to tell apart.

The rationale behind pushing the character-level word representations closer to the pre-trained ones through adversarial means, as opposed to only adding a term to the loss function, is that this gives a higher degree of freedom to the model. Indeed, since we are not explicitly defining any measure of similarity between this representations, other than "easy for the discriminator to classify", we are letting the architecture learn whatever works best for itself. Moreover, by not modifying the mechanisms to produce word-level word representations we ensure that no information acquired during pre-

training is lost.

We follow a procedure similar to the one introduced in Chapter 5 for training our models: we pre-train them in SNLI and test them in several test sets. Again, the choice of this task is justified by it allegedly requiring models to produce good semantic representations in order to have good performance.

6.5 Results

6.5.1 Effect of Adversarial Regularization in NLI tasks

	SNLI	MNLIm	MNLImis
Majority	33.33	33.33	33.33
Hyp-only	68.62	55.92	55.43
Infersent	84.57	70.81	70.99
+ `cat`	**84.95**	**71.58**	71.15
+ adv	84.15	71.06	71.00
+ `vg`	84.23	71.09	71.16
+ adv	84.06	71.35	**71.51**

Table 6.1 – Performance of `cat` and `vg` models in the test set of SNLI and the matched and mismatched evaluation sets of MultiNLI

Table 6.1 shows that incorporating character-level information through the `cat` method is beneficial in every case, as doing this beats the Infersent baseline. The same cannot be said about `vg` models, which are only slightly beneficial in the MultiNLI dataset. Further, adversarial regularization is not beneficial when applied to `cat` models, but it is when applied when applied to `vg` in MultiNLI.

6.5.2 Effect of Adversarial Regularization in NLI Diagnostic tasks

We test our models in two NLI diagnostic tasks. Breaking NLI (Glockner et al., 2018) is designed to show the deficiencies of NLI models that require both lexical and world knowledge. Similarly, HANS (McCoy et al., 2019) tests whether NLI models are "cheating" by relying on specific heuristics categorized as *lexical overlap*, *subsequence* and *constituent*.

We compare our models to four state-of-the-art architectures. The "Enhanced Sequential Inference Model" (ESIM, Chen et al. (2017)), which is a BiLSTM that uses an inter-sentence attention mechanism for creating the premise-hypothesis pair representation. The version of ESIM that does not share information between hypothesis

and premise, i.e. that has to encode both sentences individually first and then combine them, which we denote as ENC_NLI. The "Knowledge-based Inference Model" (KIM, Chen et al. (2018)), which is essentially an ESIM model with added external WordNet information. Finally, BERT (Devlin et al., 2019), a pre-trained language model, trained on 700 million words from English books and 2.5×10^9 words from Wikipedia.

	BERT	KIM	ESIM	ENC_NLI	Infersent	+ `cat`	+ adv	+ `vg`	+ adv
antonyms	90.1	86.5	70.4	40.5	76.9	66.1	61.0	56.2	83.0
cardinals	97.4	93.4	75.5	47.0	71.1	68.6	64.7	58.9	79.3
nationalities	100.0	73.5	35.9	16.4	60.0	53.6	42.4	62.5	82.3
drinks	93.6	96.6	63.7	81.5	92.9	95.8	87.6	91.0	93.8
antonyms (WN)	83.4	78.8	74.6	59.2	73.9	73.1	65.4	67.0	75.2
colors	97.3	98.3	96.1	72.8	95.0	94.3	94.3	91.0	95.3
ordinals	77.5	56.6	21.0	5.9	9.0	15.8	14.8	8.6	20.4
countries	99.3	70.8	25.4	57.6	89.9	82.4	86.1	76.2	87.1
rooms	94.8	77.6	69.4	51.9	76.5	80.5	72.1	77.3	74.3
materials	96.0	98.7	89.7	41.8	96.0	90.2	90.9	89.4	91.7
vegetables	70.6	79.8	31.2	24.8	47.7	49.5	45.9	41.3	43.1
instruments	98.5	96.9	90.8	83.1	96.9	96.9	98.5	96.9	98.5
planets	91.7	5.0	3.3	5.0	85.0	91.7	81.7	80.0	78.3
synonyms	99.7	92.1	99.7	99.4	86.0	92.2	98.8	95.3	91.7
BNLI acc.	93.2	83.5	65.6	52.6	74.7	73.3	70.0	69.4	**79.5**
SNLI test acc.	90.7	88.6	87.9	85.0	84.6	85.0	84.2	84.2	84.1

Table 6.2 – Performance of `cat` and `vg` models trained in SNLI and tested in BreakingNLI

Table 6.2 shows our results in the Breaking NLI dataset. The setting that is closest to ours in terms of size, architecture and training setting is the ENC_NLI that does not share information between sentences and does not rely on external information. The table shows that a simple Infersent baseline already beats this model by a margin of more than 20%, despite it having a test accuracy in SNLI 0.4% lower. Similarly, the baseline beats ESIM by 8.9%, despite it having a test accuracy in SNLI 3.3% lower. This is indicative that there is a trade-off between relying in heuristics and having good performance in SNLI.

Our models with added characters were not able to surpass the baseline, and again the adversarial setting did not contribute in the `cat` model's performance. However the `vg` + adversarial setting provided a significant improvement over the baseline, beating it by 4.8% while trading-off only 0.5% in test accuracy. What is more, our setting is only 4% short of achieving the same performance as KIM which has both external knowledge and shared knowledge between sentences. This supports the hypothesis that our model creates representations that are less reliant on annotation heuristics.

Similarly, Table 6.3 shows that our vector gate model paired with adversarial regularization significantly beats every model by more than 10% in the Non-Entailment case, except BERT. This increased performance, however, comes at a cost in perfor-

		BERT	KIM	ESIM	ENC_NLI	Infersent	+ cat	+ adv	+ vg	+ adv
E	LO	0.97	1.00	0.99	0.99	0.98	0.99	0.99	0.99	0.92
	S	1.00	0.99	0.99	1.00	0.98	0.99	0.99	0.99	0.96
	C	1.00	0.96	0.98	0.97	0.97	0.98	0.99	0.98	0.95
	avg	*0.99*	*0.98*	*0.99*	*0.98*	*0.98*	*0.99*	*0.99*	*0.99*	*0.94*
NE	LO	0.42	0.01	0.01	0.01	0.07	0.02	0.05	0.05	0.28
	S	0.06	0.00	0.00	0.01	0.02	0.02	0.01	0.01	0.06
	C	0.02	0.02	0.04	0.03	0.06	0.05	0.03	0.05	0.09
	avg	*0.17*	*0.01*	*0.02*	*0.02*	*0.05*	*0.03*	*0.03*	*0.04*	***0.14***

Table 6.3 – Performance of `cat` and `vg` models trained in SNLI and tested in HANS. E stands for "Entailment", NE for "Non-Entailment", LO for "Lexical Overlap", S for "Subsequence", and C for "constituent"

mance in the Entailment case, where every model performs almost perfectly our model drops to 94%.

Chapter 7

Conclusions

In this thesis we studied the character modality, and how to combine it with word representations. We began by proposing two models; one for tackling the NLI task, and another for the Implicit Emotion Recognition task. The former was competitive to other systems developed in the same context, while the latter was superior. We learned from these studies that the usefulness of the character modality is likely to depend on other parts of the architecture, on the task being tackled, and on the dataset that embodies it.

Later, we proposed letting the model learn how to combine both character and word modalities through gating mechanisms and showed that this was beneficial in some cases. Finally, we cast the problem of making character-derived representations similar to pre-trained word embeddings as multi-modal domain adaptation, an addressed it with an adversarial regularization setting. We showed that imbuing models with this inductive bias significantly reduced their reliance on dataset annotation artifacts.

7.1 Contributions

In Chapter 3 we showed that a self-attentive architecture that uses the output of a max-pooled BiLSTM as context query vector, is capable of performing as well as a more complex model with access to more information, in the NLI task. We further provided evidence that such architecture does not benefit from having access to character-derived word representations in the NLI task, despite this modality being helpful, to a limited extent, in more complex models. We argued this might be due to English not being a morphologically rich language, hence not containing meaningful patterns at the sub-word level, and therefore complex architectures at the word level might be enough for capturing syntactic and semantic information.

In Chapter 4 we presented an architecture that relies on character-derived repre-

sentations only, and was pre-trained in a self-supervised manner. We showed that it performed comparably well to a more complex model despite having been trained on significantly less data. This model also proved to be good at domain transfer, since it was trained in data from a vastly different domain, and performed as well as models pre-trained in data from the same domain. We argue this high transfer capability might be attributable to the character modality. Additionally, we show that emoji and hashtags are important features when attempting to predict implicit emotion, and that different emoji contribute differently and often counter-intuitively to this task. This architecture got 2nd place out of 30 teams in the IEST at WASSA 2018.

In Chapter 5 we proposed a feature-wise sigmoidal gating mechanism for selectively combining character and word representations, and showed that it is better than other commonly-used methods, as measured by word-similarity tasks. We also observed that this mechanism learns that to build good word representations it has to increasingly rely on character-level information as it models increasingly infrequent words. We further demonstrated that despite the increased expressivity of word representations it offers, it has no clear effect in sentence representations as measured by sentence evaluation tasks, which could be interpreted as these specific sentence-level tasks not requiring complex modeling at the subword level.

In Chapter 6 we showed that combining character-derived and pre-trained word representations with a vector gate, while forcing them to be similar, obtains state-of-the-art results in datasets aimed at measuring how much models rely in annotation artifacts. This means that our models encode representations of meaning better than comparable architectures.

In all, we provided evidence that the character modality is often useful for obtaining better meaning representations, but the way in which it is combined with the word modality plays an essential role. Further, we demonstrated that letting models selectively combine character-derived and pre-trained word representations through gating mechanisms leads to better word representations. Moreover, we showed that the assumption that the character and word modalities should produce similar word representations, based on the fact that they represent the same meaning, proved to be true in some experimental settings.

Finally, since our findings pertain to the domain of learning word representations, they are roughly task-independent. The same gating architectures and adversarial methods used in combining character-derived and pre-trained word representations could be used in diverse applications such as SPAM detection, Part-of-Speech tagging, Named-Entity Recognition, Machine Translation, or Question Answering. In short, any system relying on word representations could potentially benefit from our findings.

Chapter 8

Publications

8.1 First Author

- (*Planned*) **Jorge A. Balazs**, Edison Marrese-Taylor and Yutaka Matsuo. Adversarial training of word and subword representations.
- **Jorge A. Balazs** and Yutaka Matsuo. Gating Mechanisms for Combining Character and Word-level Word Representations: An Empirical Study, in Proceedings of NAACL-HLT: Student Research Workshop, Minneapolis, USA, Association for Computational Linguistics (ACL), June 2019. https://aclweb.org/anthology/N19-3016/
- **Jorge A. Balazs**, Edison Marrese-Taylor, and Yutaka Matsuo. IIIDYT at IEST 2018: Implicit Emotion Classification With Deep Contextualized Word Representations, in Proceedings of the 9th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis (WASSA) collocated with EMNLP, Brussels, Belgium, Association for Computational Linguistics (ACL), November 2018. **(Best System Analysis Paper)** https://aclweb.org/anthology/W18-6208/
- **Jorge A. Balazs**, Edison Marrese-Taylor, Pablo Loyola, and Yutaka Matsuo. Refining Raw Sentence Representations for Textual Entailment Recognition via Attention, in Proceedings of the 2nd Workshop on Evaluating Vector Space Representations for NLP (RepEval) collocated with EMNLP, Copenhagen, Denmark, Association for Computational Linguistics (ACL), September 2017. https://aclweb.org/anthology/W17-5310/

8.2 Others

- Pablo Loyola, Edison Marrese-Taylor, **Jorge A. Balazs**, Yutaka Matsuo, and Fumiko Satoh, Content Aware Source Code Change Description Generation, in Proceedings of the 11th International Conference on Natural Language Generation (INLG), Tillburg, The Netherlands, Association for Computational Linguistics, November 2018. https://aclweb.org/anthology/W18-6513/
- Suzana Ilic, Edison Marrese-Taylor, **Jorge A. Balazs**, and Yutaka Matsuo, Deep contextualized word representations for detecting sarcasm and irony, in Proceedings of the 9th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis (WASSA), Brussels, Belgium, Association for Computational Linguistics (ACL), November 2018. https://aclweb.org/anthology/W18-6202/
- Edison Marrese-Taylor, Suzana Ilic, **Jorge A. Balazs**, Yutaka Matsuo, Helmut Prendinger, IIIDYT at SemEval-2018 Task 3: Irony detection in English tweets, in Proceedings of the 12th International Workshop on Semantic Evaluation (SemEval-2018), New Orleans, USA, Association for Computational Linguistics (ACL), June 2018. https://aclweb.org/anthology/S18-1087/
- Edison Marrese-Taylor, **Jorge A. Balazs**, and Yutaka Matsuo, Mining fine-grained opinions on closed captions of YouTube videos with an attention-RNN in Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment & Social Media Analysis (WASSA) collocated with EMNLP, Copenhagen, Denmark, Association for Computational Linguistics (ACL), September 2017. https://aclweb.org/anthology/W17-5213/

Bibliography

Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. 2017. Fine-grained Analysis of Sentence Embeddings Using Auxiliary Prediction Tasks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*. Toulon, France. https://openreview.net/pdf?id=BJh6Ztuxl. 58

Yossi Adi, Neil Zeghidour, Ronan Collobert, Nicolas Usunier, Vitaliy Liptchinsky, and Gabriel Synnaeve. 2018. To reverse the gradient or not: An empirical comparison of adversarial and multi-task learning in speech recognition. *arXiv e-prints* abs/1812.03483. http://arxiv.org/abs/1812.03483. 64

Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Pasca, and Aitor Soroa. 2009. A Study on Similarity and Relatedness Using Distributional and WordNet-based Approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, Boulder, Colorado, pages 19–27. http://aclweb.org/anthology/N09-1003. 48, 49, 49

Eneko Agirre, Carmen Banea, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Rada Mihalcea, German Rigau, and Janyce Wiebe. 2016. SemEval-2016 Task 1: Semantic Textual Similarity, Monolingual and Cross-Lingual Evaluation. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*. Association for Computational Linguistics, San Diego, California, pages 497–511. http://www.aclweb.org/anthology/S16-1081. 50

Fotis Aisopos, George Papadakis, Konstantinos Tserpes, and Theodora Varvarigou. 2012. Content vs. context for sentiment analysis: a comparative analysis over microblogs. In *Proceedings of the 23rd ACM Conference on Hypertext and Social Media*. ACM, pages 187–196. https://dl.acm.org/citation.cfm?id=2310028. 8

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2017. A Simple but Tough-to-Beat Baseline for Sentence Embeddings. In *International Conference on Learning Representations*. https://openreview.net/pdf?id=SyK00v5xx. 50, 51

Oded Avraham and Yoav Goldberg. 2017. The Interplay of Semantics and Morphology in Word Embeddings. *arXiv preprint arXiv:1704.01938* https://arxiv.org/abs/1704.01938. 19, 43

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. San Diego, California, USA. 63

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics* 5:135–146. https://transacl.org/ojs/index.php/tacl/article/view/999. 7, 10, 20, 42, 43, 43, 60

Hamed R. Bonab and Fazli Can. 2016. A theoretical framework on the ideal number of classifiers for online ensembles in data streams. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, New York, NY, USA, CIKM '16, pages 2053–2056. https://doi.org/10.1145/2983323.2983907. 34

Jan Botha and Phil Blunsom. 2014. Compositional Morphology for Word Representations and Language Modelling. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*. PMLR, Bejing, China, volume 32 of *Proceedings of Machine Learning Research*, pages 1899–1907. http://proceedings.mlr.press/v32/botha14.html. 7, 20, 43, 43

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A Large Annotated Corpus for Learning Natural Language Inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Lisbon, Portugal, pages 632–642. http://aclweb.org/anthology/D15-1075. 8, 16, 25, 48

Elia Bruni, Nam-Khanh Tran, and Marco Baroni. 2014. Multimodal Distributional Semantics. *Journal of Artificial Intelligence Research* 49:1–47. https://www.jair.org/index.php/jair/article/view/10857/25905. 49, 52

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Association for Computational Linguistics, Vancouver, Canada, pages 1–14. https://doi.org/10.18653/v1/S17-2001. 50

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2014. One billion word benchmark for measuring progress in statistical language modeling. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH* pages 2635–2639. https://doi.org/10.1016/j.csl.2015.07.001. 33

Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Diana Inkpen, and Si Wei. 2018. Neural Natural Language Inference Models Enhanced with External Knowledge. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Melbourne, Australia, pages 2406–2417. https://doi.org/10.18653/v1/P18-1224. 67

Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. 2017. Enhanced lstm for natural language inference. In *Proc. ACL*. 26, 66

Billy Chiu, Anna Korhonen, and Sampo Pyysalo. 2016. Intrinsic Evaluation of Word Vectors Fails to Predict Extrinsic Performance. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*. Association for Computational Linguistics, Berlin, Germany, pages 1–6. http://anthology.aclweb.org/W16-2501. 55

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, pages 1724–1734. https://doi.org/10.3115/v1/D14-1179. 17

Ronan Collobert and Jason Weston. 2008. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In Andrew McCallum and Sam Roweis, editors, *Proceedings of the 25th Annual International Conference on Machine Learning (ICML 2008)*. Helsinki, Finland, pages 160–167. https://icml.cc/Conferences/2008/papers/391.pdf. 14, 15, 58

Alexis Conneau and Douwe Kiela. 2018. SentEval: An Evaluation Toolkit for Universal Sentence Representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)*. European Language Resource Association, Miyazaki, Japan. http://aclweb.org/anthology/L18-1269. 51, 58

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Copenhagen, Denmark, pages 670–680. https://www.aclweb.org/anthology/D17-1070. 15, 15, 17, 17, 17, 18, 32, 35, 47, 47, 48, 51, 51, 54, 58

Alexis Conneau, Germán Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. 2018a. What you can cram into a single $&!#* vector: Probing sentence embeddings for linguistic properties. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Melbourne, Australia, pages 2126–2136. https://www.aclweb.org/anthology/P18-1198. 36, 58

Alexis Conneau, Ruty Rinott, Guillaume Lample, Adina Williams, Samuel R. Bowman, Holger Schwenk, and Veselin Stoyanov. 2018b. Xnli: Evaluating cross-lingual sentence representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 16

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*

(Long and Short Papers). Association for Computational Linguistics, Minneapolis, Minnesota, pages 4171–4186. https://doi.org/10.18653/v1/N19-1423. 6, 6, 7, 59, 67

Vincent Dumoulin, Ethan Perez, Nathan Schucher, Florian Strub, Harm de Vries, Aaron Courville, and Yoshua Bengio. 2018. Feature-wise transformations. *Distill* https://doi.org/undefined. 59

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science* 14(2):179–211. https://doi.org/10.1207/s15516709cog1402_1. 12

Manaal Faruqui, Yulia Tsvetkov, Pushpendre Rastogi, and Chris Dyer. 2016. Problems With Evaluation of Word Embeddings Using Word Similarity Tasks. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*. Association for Computational Linguistics, Berlin, Germany, pages 30–35. http://anthology.aclweb.org/W16-2506. 48, 55

Christiane Fellbaum, editor. 1998. *WordNet: an Electronic Lexical Database*. MIT Press. 49

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2002. Placing Search in Context: The Concept Revisited. *ACM Transactions on Information Systems* 20(1):116–131. https://doi.org/10.1145/503104.503110. 49

Zhe Gan, Yunchen Pu, Ricardo Henao, Chunyuan Li, Xiaodong He, and Lawrence Carin. 2017. Learning Generic Sentence Representations Using Convolutional Neural Networks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Copenhagen, Denmark, pages 2390–2400. https://doi.org/10.18653/v1/D17-1254. 58

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-adversarial training of neural networks. *Journal of Machine Learning Research* 17(59):1–35. http://jmlr.org/papers/v17/15-239.html. 64, 65

Daniela Gerz, Ivan Vulić, Felix Hill, Roi Reichart, and Anna Korhonen. 2016. SimVerb-3500: A Large-Scale Evaluation Set of Verb Similarity. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, pages 2173–2182. https://aclweb.org/anthology/D16-1235. 49

Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A. Smith. 2011. Part-of-speech tagging for twitter: Annotation, features, and experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Portland, Oregon, USA, pages 42–47. http://www.aclweb.org/anthology/P11-2008. 32

Anna Gladkova and Aleksandr Drozd. 2016. Intrinsic Evaluations of Word Embeddings: What Can We Do Better? In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*. Association for Computational Linguistics, Berlin, Germany, pages 36–42. http://anthology.aclweb.org/W16-2507. 56

Max Glockner, Vered Shwartz, and Yoav Goldberg. 2018. Breaking NLI Systems with Sentences that Require Simple Lexical Inferences. *arXiv e-prints* http://arxiv.org/abs/1805.02266. 66

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. The MIT Press. 3

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech Recognition with Deep Recurrent Neural Networks. In *Proceedings of the 2013 International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, Vancouver, Canada, pages 6645–6649. https://ieeexplore.ieee.org/document/6638947/. 13, 44

Alex Graves and Jürgen Schmidhuber. 2005. Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures. *Neural Networks* 18(5-6):602–610. https://www.sciencedirect.com/science/article/pii/S0893608005001206. 13, 13, 44

Guy Halawi, Gideon Dror, Evgeniy Gabrilovich, and Yehuda Koren. 2012. Large-scale Learning of Word Relatedness with Constraints. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Beijing, China, KDD '12, pages 1406–1414. https://doi.org/10.1145/2339530.2339751. 49

Zellig S. Harris. 1954. Distributional structure. *Word* 10:146–162. 3

Felix Hill, Kyunghyun Cho, and Anna Korhonen. 2016. Learning Distributed Representations of Sentences from Unlabelled Data. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, San Diego, California, pages 1367–1377. https://doi.org/10.18653/v1/N16-1162. 58

Felix Hill, Roi Reichart, and Anna Korhonen. 2015. SimLex-999: Evaluating Semantic Models With (Genuine) Similarity Estimation. *Computational Linguistics* 41(4):665–695. http://aclweb.org/anthology/J15-4004. 48, 49

Geoffrey E. Hinton. 1986. Learning Distributed Representations of Concepts. In *Proceedings of the 8th Annual Conference of the Cognitive Science Society*. pages 1–12. 4

Geoffrey E. Hinton, James L. Mcclelland, and David E. Rumelhart. 1986. Distributed representations. In David E. Rumelhart and James L. Mcclelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, MIT Press, Cambridge, MA, pages 77–109. 3

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735. 12, 23, 44

Jeremy Howard and Sebastian Ruder. 2018. Universal Language Model Fine-tuning for Text Classification. *ArXiv e-prints* . 6, 33, 35, 35

Minqing Hu and Bing Liu. 2004. Mining and Summarizing Customer Reviews. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Seattle, Washington, KDD '04, pages 168–177. https://doi.org/10.1145/1014052.1014073. 50

Stanisław Jastrzebski, Damian Leśniak, and Wojciech Marian Czarnecki. 2017. How to evaluate word embeddings? on importance of data efficiency and simple supervised tasks. *arXiv preprint arXiv:1702.02170* https://arxiv.org/abs/1702.02170. 48

Yacine Jernite, Samuel R. Bowman, and David Sontag. 2017. Discourse-Based Objectives for Fast Unsupervised Sentence Representation Learning. *CoRR* abs/1705.00557. http://arxiv.org/abs/1705.00557. 58

John D. Hunter. 2007. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering* 9(3):90–95. https://doi.org/10.1109/MCSE.2007.55.

Eric Jones, Travis Oliphant, Pearu Peterson, et al. 2001–. SciPy: Open source scientific tools for Python. http://www.scipy.org/.

Daniel Jurafsky and James H. Martin. 2018. Speech and language processing (3rd edition draft). `https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf`. 3, 3

Jerrold J. Katz. 1972. *Semantic Theory*. Harper & Row, New York. 21

Tushar Khot, Ashish Sabharwal, and Peter Clark. 2018. SciTaiL: A Textual Entailment Dataset from Science Question Answering. In *AAAI Conference on Artificial Intelligence*. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17368. 17

Yoon Kim, Yacine Jernite, David Sontag, and Alexander Rush. 2016. Character-Aware Neural Language Models. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*. Phoenix, Arizona, pages 2741–2749. https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12489/12017. 44

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv e-prints* https://arxiv.org/abs/1412.6980. 33

Karin Kipper, Anna Korhonen, Neville Ryant, and Martha Palmer. 2008. A large-scale classification of English verbs. *Language Resources and Evaluation* 42(1):21–40. https://doi.org/10.1007/s10579-007-9048-2. 50

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-Thought Vectors. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, Curran Associates, Inc., pages 3294–3302. http://papers.nips.cc/paper/5950-skip-thought-vectors.pdf. 58, 58

Roman Klinger, Orphée de Clercq, Saif M. Mohammad, and Alexandra Balahur. 2018. IEST: WASSA-2018 Implicit Emotions Shared Task. In *Proceedings of the 9th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*. Association for Computational Linguistics, Brussels, Belgium. iv, 31, 36, 39, 40

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems*. pages 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf. 6

Taku Kudo. 2018. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Melbourne, Australia, pages 66–75. https://www.aclweb.org/anthology/P18-1007. 7, 19

Alice Lai, Yonatan Bisk, and Julia Hockenmaier. 2017. Natural language inference from multiple premises. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Asian Federation of Natural Language Processing, Taipei, Taiwan, pages 100–109. https://www.aclweb.org/anthology/I17-1011. 16

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural Architectures for Named Entity Recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, San Diego, California, pages 260–270. https://doi.org/10.18653/v1/N16-1030. 43

Xin Li and Dan Roth. 2002. Learning Question Classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*. http://aclweb.org/anthology/C02-1150. 50

Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A structured self-attentive sentence embedding. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*. https://arxiv.org/pdf/1703.03130.pdf. 22, 26

Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luis Marujo, and Tiago Luis. 2015. Finding Function in Form: Composi-

tional Character Models for Open Vocabulary Word Representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Lisbon, Portugal, pages 1520–1530. http://aclweb.org/anthology/D15-1176. 42

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies. *Transactions of the Association for Computational Linguistics* 4(1):521–535. https://www.aclweb.org/anthology/Q16-1037. 58

Yang Liu, Chengjie Sun, Lei Lin, and Xiaolong Wang. 2016. Learning natural language inference using bidirectional LSTM model and inner-attention. *CoRR* abs/1605.09090. http://arxiv.org/pdf/1605.09090.pdf. 22

Minh-Thang Luong and Christopher D. Manning. 2016. Achieving Open Vocabulary Neural Machine Translation with Hybrid Word-Character Models. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, pages 1054–1063. http://www.aclweb.org/anthology/P16-1100. 19, 42, 43, 43

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Lisbon, Portugal, pages 1412–1421. http://aclweb.org/anthology/D15-1166. 24, 28

Thang Luong, Richard Socher, and Christopher D. Manning. 2013. Better Word Representations with Recursive Neural Networks for Morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics, Sofia, Bulgaria, pages 104–113. http://www.aclweb.org/anthology/W13-3512. 49, 52

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella bernardi, and Roberto Zamparelli. 2014. A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*. European Language Resources Association (ELRA), Reykjavik, Iceland. http://www.lrec-conf.org/proceedings/lrec2014/pdf/363_Paper.pdf. 17, 50, 50, 54

Eugenio Martínez-Cámara, M. Teresa Martín-Valdivia, L. Alfonso Ureña-López, and Arturo Montejo-Ráez. 2014. Sentiment analysis in Twitter. *Natural Language Engineering* 20(1):1–28. https://doi.org/10.1017/S1351324912000332. 8

R. Thomas McCoy, Ellie Pavlick, and Tal Linzen. 2019. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. *arXiv e-prints* abs/1902.01007. http://arxiv.org/abs/1902.01007. 66

Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*. pages 51 – 56. https://conference.scipy.org/proceedings/scipy2010/pdfs/mckinney.pdf.

Tomas Mikolov, Kai Chen, Greg S Corrado, and Jeff Dean. 2013a. Efficient estimation of word representations in vector space. *ArXiv e-prints* https://arxiv.org/abs/1301.3781. 6, 9, 60

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*. Curran Associates, Inc., pages 3111–3119. http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf. 3, 6, 58

Yasumasa Miyamoto and Kyunghyun Cho. 2016. Gated Word-Character Recurrent Language Model. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, pages 1992–1997. https://aclweb.org/anthology/D16-1209. 45, 45, 46, 46, 53, 54, 56, 63

Saif M. Mohammad and Felipe Bravo-Marquez. 2017. WASSA-2017 Shared Task on Emotion Intensity. In *Proceedings of the EMNLP 2017 Workshop on Computational Approaches to Subjectivity, Sentiment, and Social Media (WASSA)*. Copenhagen, Denmark. 31

Lili Mou, Hao Peng, Ge Li, Yan Xu, Lu Zhang, and Zhi Jin. 2015. Discriminative neural sentence modeling by tree-based convolution. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Lisbon, Portugal, pages 2315–2325. http://aclweb.org/anthology/D15-1279. 24

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10, pages 807–814. https://icml.cc/Conferences/2010/papers/432.pdf. 25

Nikita Nangia, Adina Williams, Angeliki Lazaridou, and Samuel Bowman. 2017. The RepEval 2017 Shared Task: Multi-Genre Natural Language Inference with Sentence Representations. In *Proceedings of the 2nd Workshop on Evaluating Vector Space Representations for NLP*. Association for Computational Linguistics, pages 1–10. https://doi.org/10.18653/v1/W17-5301. 26

Douglas L. Nelson, Cathy L. McEvoy, and Thomas A. Schreiber. 2004. The University of South Florida free association, rhyme, and word fragment norms. *Behavior Research Methods, Instruments, & Computers* 36(3):402–407. https://doi.org/10.3758/BF03195588. 50

Travis E. Oliphant. 2015. *Guide to NumPy*. CreateSpace Independent Publishing Platform, USA, 2nd edition. http://web.mit.edu/dvp/Public/numpybook.pdf.

Bo Pang and Lillian Lee. 2004. A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts. In *Proceedings of the*

42nd Annual Meeting of the Association for Computational Linguistics (ACL-04). http://aclweb.org/anthology/P04-1035. 50

Bo Pang and Lillian Lee. 2005. Seeing Stars: Exploiting Class Relationships for Sentiment Categorization with Respect to Rating Scales. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*. Association for Computational Linguistics, Ann Arbor, Michigan, pages 115–124. http://aclweb.org/anthology/P05-1015. 50

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NeurIPS Autodiff Workshop*. Long Beach, California. https://openreview.net/pdf?id=BJJsrmfCZ. 33

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, pages 1532–1543. https://doi.org/10.3115/v1/D14-1162. 7, 9, 25, 47, 47, 60

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. 2018. FiLM: Visual Reasoning with a General Conditioning Layer. In *AAAI Conference on Artificial Intelligence*. New Orleans, Louisiana. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16528. 59

Christian S. Perone, Roberto Silveira, and Thomas S. Paula. 2018. Evaluation of sentence embeddings in downstream and linguistic probing tasks. *CoRR* abs/1806.06259. http://arxiv.org/abs/1806.06259. 58

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018a. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics, New Orleans, Louisiana, pages 2227–2237. http://www.aclweb.org/anthology/N18-1202. 3, 6, 31, 32, 41

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018b. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics, New Orleans, Louisiana, pages 2227–2237. https://doi.org/10.18653/v1/N18-1202. 59

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving Language Understanding by Generative Pre-Training. Technical report, OpenAI. https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf. 59

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. Technical report,

OpenAI. https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf. 6, 59

Kira Radinsky, Eugene Agichtein, Evgeniy Gabrilovich, and Shaul Markovitch. 2011. A Word at a Time: Computing Word Relatedness Using Temporal Semantic Analysis. In *Proceedings of the 20th International Conference on World Wide Web*. Hyderabad, India, WWW '11, pages 337–346. http://wwwconference.org/proceedings/www2011/proceedings/p337.pdf. 49

Cyrus Rashtchian, Peter Young, Micah Hodosh, and Julia Hockenmaier. 2010. Collecting Image Annotations Using Amazon's Mechanical Turk. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*. Association for Computational Linguistics, Los Angeles, California, pages 139–147. http://www.aclweb.org/anthology/W10-0721. 17, 54

Marek Rei, Gamal Crichton, and Sampo Pyysalo. 2016. Attending to Characters in Neural Sequence Labeling Models. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. The COLING 2016 Organizing Committee, Osaka, Japan, pages 309–318. https://www.aclweb.org/anthology/C16-1030. 63, 64

Herbert Rubenstein and John B. Goodenough. 1965. Contextual Correlates of Synonymy. *Communications of the ACM* 8(10):627–633. https://doi.org/10.1145/365628.365657. 49

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115(3):211–252. https://doi.org/10.1007/s11263-015-0816-y. 6

Magnus Sahlgren. 2006. *The Word-Space Model: Using distributional analysis to represent syntagmatic and paradigmatic relations between words in high-dimensional vector spaces*. Ph.D. thesis. 1, 2

Ferdinand de Saussure. 2013. *Course in General Linguistics*. Bloomsbury Academic, 1st edition. First published in 1915. Translated by Roy Harris. https://www.bloomsbury.com/uk/course-in-general-linguistics-9781472505385/. 1, 2

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, pages 5149–5152. 19

Hinrich Schütze. 1993. Word Space. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems*, Morgan-Kaufmann, pages 895–902. http://papers.nips.cc/paper/603-word-space.pdf. 3

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual*

Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics, Berlin, Germany, pages 1715–1725. http://www.aclweb.org/anthology/P16-1162. 7, 19

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Seattle, Washington, pages 1631–1642. http://aclweb.org/anthology/D13-1170. 51

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15:1929–1958. http://jmlr.org/papers/v15/srivastava14a.html. 33

Sandeep Subramanian, Adam Trischler, Yoshua Bengio, and Christopher J Pal. 2018. Learning General Purpose Distributed Sentence Representations via Large Scale Multi-task Learning. In *International Conference on Learning Representations*. https://openreview.net/pdf?id=B18WgG-CZ. 48, 58

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, Beijing, China, pages 1556–1566. https://doi.org/10.3115/v1/P15-1150. 26

Yulia Tsvetkov, Manaal Faruqui, Wang Ling, Guillaume Lample, and Chris Dyer. 2015. Evaluation of Word Vector Representations by Subspace Alignment. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Lisbon, Portugal, pages 2049–2054. http://aclweb.org/anthology/D15-1243. 56

Johan van Benthem. 2008. A brief history of natural logic. In M. Chakraborty, B. Löwe, M. Nath Mitra, and S. Sarukki, editors, *Logic, Navya-Nyaya and Applications: Homage to Bimal Matilal*, College Publications. 21

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9:2579–2605. http://www.jmlr.org/papers/v9/vandermaaten08a.html. 63

Guido van Rossum. 1995. Python Tutorial. Technical Report CS-R9526, Department of Computer Science, CWI, Amsterdam, The Netherlands. https://ir.cwi.nl/pub/5007.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*. pages 2773–2781. http://papers.nips.cc/paper/5635-grammar-as-a-foreign-language. 24

Ivan Vulić, Nikola Mrkšić, Roi Reichart, Diarmuid Ó Séaghdha, Steve Young, and Anna Korhonen. 2017. Morph-fitting: Fine-Tuning Word Vector Spaces with Simple Language-Specific Rules. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Vancouver, Canada, pages 56–68. 19

A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang. 1989. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37(3):328–339. https://doi.org/10.1109/29.21701. 14

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*. New Orleans, Louisiana. https://openreview.net/forum?id=rJ4km2R5t7. 58

Sida Wang and Christopher Manning. 2012. Baselines and Bigrams: Simple, Good Sentiment and Topic Classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, Jeju Island, Korea, pages 90–94. http://aclweb.org/anthology/P12-2018. 51

Zhiguo Wang, Wael Hamza, and Radu Florian. 2017. Bilateral multi-perspective matching for natural language sentences. *CoRR* abs/1702.03814. http://arxiv.org/abs/1702.03814. 23, 29

Michael Waskom, Olga Botvinnik, Drew O'Kane, Paul Hobson, Joel Ostblom, Saulius Lukauskas, David C Gemperline, Tom Augspurger, Yaroslav Halchenko, John B. Cole, Jordi Warmenhoven, Julian de Ruiter, Cameron Pye, Stephan Hoyer, Jake Vanderplas, Santi Villalba, Gero Kunter, Eric Quintero, Pete Bachant, Marcel Martin, Kyle Meyer, Alistair Miles, Yoav Ram, Thomas Brunner, Tal Yarkoni, Mike Lee Williams, Constantine Evans, Clark Fitzgerald, Brian, and Adel Qalieh. 2018. mwaskom/seaborn: v0.9.0 (july 2018). https://doi.org/10.5281/zenodo.1313201.

Bernard Lewis Welch. 1947. The Generalization of "Student's" Problem When Several Different Population Variances are Involved. *Biometrika* 34(1-2):28–35. https://doi.org/10.1093/biomet/34.1-2.28. 52

Janyce Wiebe, Theresa Wilson, and Claire Cardie. 2005. Annotating Expressions of Opinions and Emotions in Language. *Language Resources and Evaluation* 39(2):165–210. https://doi.org/10.1007/s10579-005-7880-9. 50

John Wieting and Douwe Kiela. 2019. No Training Required: Exploring Random Encoders for Sentence Classification. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*. New Orleans, Louisiana. https://openreview.net/pdf?id=BkgPajAcY7. 58

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In *Proceedings of the*

2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers). Association for Computational Linguistics, New Orleans, Louisiana, pages 1112–1122. http://www.aclweb.org/anthology/N18-1101. 16, 16, 22, 26, 26, 27, 48

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv preprint arXiv:1609.08144* http://arxiv.org/abs/1609.08144. 19, 42

Caiming Xiong, Victor Zhong, and Richard Socher. 2016. Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604* . 26

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. *arXiv e-prints* page arXiv:1906.08237. https://arxiv.org/abs/1906.08237. 6

Zhilin Yang, Bhuwan Dhingra, Ye Yuan, Junjie Hu, William W. Cohen, and Ruslan Salakhutdinov. 2017. Words or Characters? Fine-grained Gating for Reading Comprehension. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*. Toulon, France. https://openreview.net/pdf?id=B1hdzd5lg. 46, 53, 56, 63

Peter Young, Alice Lai, Micah Hodosh, and Julia Hockenmaier. 2014. From image descriptions to visual denotations. *Transactions of the Association for Computational Linguistics* 2:67–78. http://aclweb.org/anthology/Q14-1006. 17, 54

Minghua Zhang, Yunfang Wu, Weikang Li, and Wei Li. 2018. Learning Universal Sentence Representations with Mean-Max Attention Autoencoder. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium, pages 4514–4523. http://aclweb.org/anthology/D18-1481. 58

Han Zhao, Zhengdong Lu, and Pascal Poupart. 2015. Self-Adaptive Hierarchical Sentence Model. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*. AAAI Press, Buenos Aires, Argentina, pages 4069–4076. https://www.aaai.org/ocs/index.php/IJCAI/IJCAI15/paper/view/10828. 58

Xunjie Zhu, Tingfeng Li, and Gerard de Melo. 2018. Exploring Semantic Properties of Sentence Embeddings. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, Melbourne, Australia, pages 632–637. https://www.aclweb.org/anthology/P18-2100. 58

Index

Acronyms

BiGRU Bidirectional Gated Recurrent Unit
BiLM Bidirectional Language Model
BiLSTM Bidirectional Long Short-Term Memory Network
BPTT Back Propagation Through Time
CBOW Continuous Bag of Words
CNN Convolutional Neural Network
CV Computer Vision
ELMo Embeddings from Language Models
GRU Gated Recurrent Unit
IEST Implicit Emotions Shared Task
LM Language Model
LSTM Long Short-Term Memory Network
MLP Multi-Layer Perceptron
MT Machine Translation
NER Named-Entity Recognition
NLG Natural Language Generation
NLI Natural Language Inference
NLP Natural Language Processing
NLU Natural Language Understanding
NN Neural Network
OOV Out-of-Vocabulary
POS Part-of-Speech
QA Question Answering
RNN Recurrent Neural Network
SST Stanford Sentiment Treebank
TDNN Time Dilated Neural Network
WASSA Workshop on Computational Approaches to Subjectivity, Sentiment & Social Media Analysis