Word Representation Using A Deep Neural Network

Yunpeng Li
Faculty of Information
140 St George St
Toronto, ON M5S 3G6, Canada
yunpeng.li@mail.utoronto.ca

Kelly Lyons
Faculty of Information
140 St George St
Toronto, ON M5S 3G6, Canada
kelly.lyons@utoronto.ca

ABSTRACT

A growth in the number of applications that make use of cognitive computing has increased the need for algorithms that can parse and understand natural language. Most modern systems rely on machine learning algorithms that take a large corpus of text as input and identify features of the language in the input data. Deep learning is a recently-developed type of machine learning that uses a network with several layers (a deep network) to identify and process features in a given data set automatically. In this paper, we introduce a deep learning neural network model called the Character-Morphology-Word network (CMW), to solve the word embedding problem. We implemented our CMW model and trained it on the Wikipedia corpus. We compared our model against two past approaches in a number of word-embedding tasks and found that, while the average performance of our model is not as good as the past approaches, our model achieves comparable precision and, in one task, our model outperforms these approaches. We also used our model to generate random text and found that our model produces words in the vocabulary and can produce lexical-correct text. The results of our research indicate that a deeper neural network architecture can be used to represent words and ultimately, help solve related Natural Language Processing (NLP) tasks.

CCS Concepts

•**Computing methodologies** → **Natural language processing; Information extraction; Neural networks;** *Phonology / morphology;*

Keywords

Word Embedding; Natural Language Processing; Recurrent Neural Networks; Recursive Neural Networks

1. INTRODUCTION

Being able to represent the words of a language is an important task in NLP [50]. Word embedding is a word representation method that maps words into a continuous space where the similarity between words can be measured by their Euclidean distance [50]. A good representation of words can reduce the complexity of the language model by making use of the redundancy in natural languages to improve the performance of NLP applications.

A key to word representation is to find the structures, or features, of the words in a language. Letters form words, words form sentences, and sentences express concepts and meanings. The combinations of letters and words could be vast. However, some underlying principles rule out most of these combinations, making the distribution highly sparse. For example, there are $26^3 = 17,576$ possible combinations for three-letter-words in English, but only 1,015 of them are commonly used [14]. Similarly, in the formation of a sentence, the number of actual meaningful sentences are far less than the total number of possible combinations of English words. Just like identifying edges and texture helps in recognizing images, automatically identifying the patterns in words can help in understanding how the concepts form; recognizing patterns in sentence structures can help in understanding how languages work.

Deep learning is a recently-developed technique that has been applied to automatically detect features in images [33]. As an unsupervised auto-encoding method for feature extraction, deep learning is a candidate method for solving the word representation task [35]. It performs as an automatic encoder to extract higher level patterns and features from a given training data set. These features are generated statistically and can be automatically optimized according to independency and representativeness, which is usually difficult or even impossible in manually-created hand-crafted features.

Deep learning has been widely applied in the field of NLP [2, 4, 26, 27, 29, 30, 36, 47]. Traditional neural network language models (NNLM) usually use word [2, 4, 26, 27, 29, 36] or character level input [7, 30]. Traditional neural network based word embedding models take words (tokens) as the input [19, 42]. In this paper, we introduce a model called the Character-Morphology-Word Network (CMW), that is composed of two deep neural networks (a character-level network and a word-level network) and a morphological layer in between that uses information obtained from the character level network to break words into stems. Our model captures character-level information in the spelling of words, a feature that is ignored in more traditional models. Our goal is to determine if our model can perform as well as or better than more traditional word-level networks.

Using Theano [3, 6], we implemented our deep learning algorithms and trained them on the Wikipedia corpus. We evaluated our model by comparing its performance with two approaches in Levy *et al.* [37]. The results shows that, al-

CASCON 2016 October 31-November 2, 2016, Markham, Ontario, Canada.

though the average performance of our model is not as good as that in [37, 44], our model achieves comparable precision in most tasks. In one task, our model even achieves even better performance. We also used our model to generate random text.

The rest of this thesis is organized as follows, in Section 2, we present relevant literature. In Section 3, we describe our CMW model and explain each of its components in detail. In Section 4, we present the results of our evaluation and conclude in Section 5.

2. LITERATURE REVIEW

In common NLP tasks such as Part-of-Speech (POS) tagging and machine translation, words used to be treated as atomic entities using a "one-hot representation" [11, 12, 42, 59] which regards different words as completely isolated items and ignores the relations and similarity among them. However, due to the complex relationships among words such as overlapping concepts, and variations in spelling and grammatical cases in real world languages, the one-hot representation is inefficient, and overlooks useful latent information hidden in the structure of a language.

Word embedding is a word representation method which maps words into dense vectors in a continuous space [36]. Word embedding can be used to find the syntactic and semantic latent structure of a language. For example, if the language model has been trained with a corpus containing the sentence, "The cat is walking in the bedroom", it should also recognize the sentence "A dog was running in a room" because "dog" and "cat", "the" and "a", "room" and "bedroom" have similar semantic and grammatical roles [4]; that is, they have similar word embedding vectors. Word embedding can augment the existing NLP algorithms by connecting unseen words to known ones, by detecting common behaviors between similar words, and by directly identifying lexicon features. This is especially useful when the corpus is not big enough to solve the data sparsity problem [1]. Word embedding features have been successfully applied to Part-of-Speech (POS) Tagging [32], Named Entity Recognition (NER) [45] and parsing [15, 32, 58].

Current methods for word embedding include neural networks [43], dimensionality reduction on the word co-occurrence matrix [36], and explicit representation in terms of the context in which words appear [50]. The most popular methods and packages for word embedding are word2vec [42] and GloVe [50]. These methods use a neural network language model to find the representation of words in a continuous vector space [4, 46, 47].

Traditional approaches take the context of the word that appears in a sentence (often called the "bag of words" or "BOW") as the input to the models. In the neural network approaches [43, 50], a neural network is trained to use the BOW to represent the word by predicting the missing word w_i between two bags of words (previous BOW $w_{i-n}, w_{i-n+1}, \cdots, w_{i-1}$ and next BOW $w_{i+1}, w_{i+1}, \cdots, w_{i+n}$). This is called the Continuous Bag-of-Words model (CBOW) [42]. The context is usually two bags of words of the same length n. While increasing n gives a better performance, it increases the complexity of the model dramatically and the model becomes untrainable quickly. In matrix based models [36], the co-occurrence matrix is obtained for the words in sentences, and the matrix is factorized. Again, matrix factorization is a computationally intensive task which becomes impractical with a large vocabulary. Moreover, these models all take words as the input which makes it impossible to handle words that have not been seen in the training examples.

In this research, we use deep neural networks to perform the word embedding task. Our character-morphology-word (CMW) network is composed of two deep neural networks with different structures on character-level and word-level. Our Character-Level Neural Network Language Model (CLNNLM) is a Bidirectional Long Short-term Memory Network (Bi-LSTM) and our Word-Level Neural Network Language Model (WLNNLM) is a Tree-Structured Recursive Neural Network (TRNN). The CLNNLM takes each character in the words as input and produces the embedding of the word. It uses Bi-LSTM to retain the letter order in words so that stems can be correctly processed. The WLNNLM uses the parsing tree of the sentence to combine the words recursively and produce the embedding of the sentence. It uses TRNN to represent long distance dependencies or discontinuities which occur when there is a separation between a word or phrase and the word or phrase that it modifies such as those that happen with wh- words (whom, what, which, whose, etc.).

A multi-layer neural network includes an input layer, one or more hidden layers, and an output layer with several neurons in each layer [38]. Neurons of neighboring layers are interconnected by weighted edges. The layers are parallelly arranged in a hierarchy such that information can flow via links between layers (and get processed) before being sent to outputs [49]. Once trained, the weights of the links are adjusted to adapt to the features of the inputs and outputs. One of the popular and efficient training algorithms for traditional neural networks is Back Propagation (BP) [34, 35, 53]. BP uses the chain rule to revise the weights according to the errors of each prediction during training. However, BP fails in training deep networks due to a decrease of the error signal in the propagation through layers [23]. This causes the training process to get stuck in some saddle points, the so called Vanishing Gradient Problem [5, 35].

Around 2006, an unsupervised pre-training procedure was introduced to solve this problem [35]. In a deep neural network, raw data (such as Wikipedia in our case) is first used to pre-train the layers. The pre-training process is applied to the weighted matrix between the first two layers w_{ij}, while keeping other layers "frozen". The weights w_{ij} between the first two layers i and j are then adjusted by an optimization function, which compares the output of the neural network with the actual answers. Once a local minimum is reached, w_{ij} is frozen, and the weight matrix between higher layers w_{jk} is adjusted using the same method. By applying this greedy algorithm to the deep neural network, a representative model of the original input data can be obtained [22].

A successful application of the deep learning is image recognition. A deep Convolutional Neural Network (CNN) based image recognition system first tries to combine the neighboring pixels and identify the features (strokes and texture), and combine them into structures (a tail, an ear, a face), and finally the meaningful representation (a cat) [33, 35].

CNN based methods have also been widely applied to NLP tasks such as text classification and sentiment analysis [26, 27, 29, 30]. CNN has been used for embedding character, morpheme and word level inputs. However, one of the keys

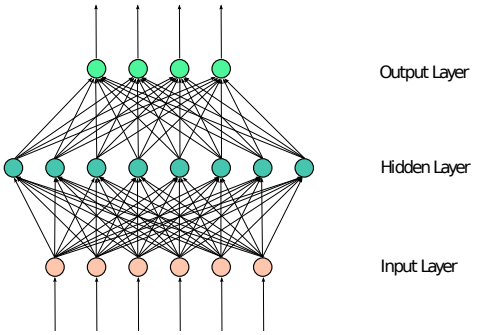

Figure 1. The typical structure of a Artificial Neural Network with one hidden layer.

to the success of CNN is the translational, rotational and scale invariances in image recognition tasks which are not applicable in NLP tasks. Another popular and successful model of deep learning in NLP is the Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) [24]. Unlike the CNN which is deep in space, the LSTM-RNN is deep in time. Another successful model is the Tree-structured Recursive Neural Network (TRNN) [57] which takes advantage of the parsing tree structure of the sentence. These models were used in our CMW network. A more detailed description is presented in the next section.

3. CHARACTER-MORPHOLOGY-WORD NETWORK

In this section, we describe the architecture of our Character-Morphology-Word Network (CMW). The model is a combined model composed of two deep neural networks (Character Level and Word Level) and a transition layer (Morphological Level) in between. In order to present the detailed description of our CMW Network, it is first necessary to describe the kinds of deep neural networks that underlie it. Fig. 2 shows the architecture of a general Neural Network Language Model [2]. Given a sequence of n words, the first $(n-1)$ words are used to predict the n^{th} word. The first $(n-1)$ words in the sequence are input to the projection layer. In this layer, the one-hot representations of these previous words are projected into vectors in a continuous space (word embedding). Then the continuous feature vectors are concatenated into a big vector and processed by one or more hidden layers. The final layer, which has the same number of output neurons as the size of the vocabulary, is the probability of each word appearing in a natural sentence following the first $(n-1)$ words [2]. In [2], the authors obtained a trained projection layer by optimizing the output probability with the training data from the Wikipedia corpus, which gives the word vector of each word in the vocabulary (Wikipedia in this case). In order to minimize the prediction errors, the words with similar distribution of successive words are mapped to similar vectors, so that the input of layers above the projection layer are similar, and hence the output (the estimated distribution) is similar to the actual distribution [2].

A Recurrent Neural Network (RNN) is a special kind of deep neural network which, instead of laying multiple layers upon each other to create a deep structure, it is a network that is "deep in time" [21, 25]. As can be seen in Fig. 3,

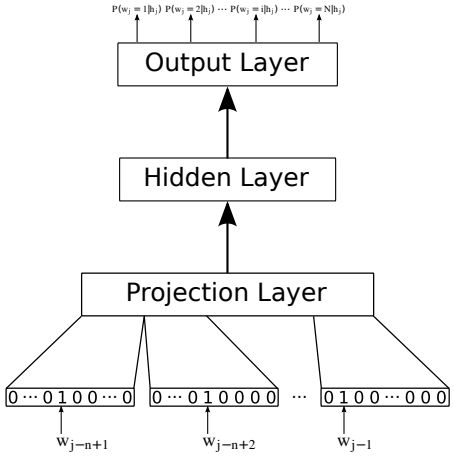

Figure 2. The architecture of a general Neural Network Language Model for word embedding. Note that in a Deep Neural Network there are many hidden layers (adapted from [2]).

while the step inputs ($x = \{x_0, x_1, x_2, \cdots, x_t, \cdots\}$) are fed to the network one by one, the status of hidden neurons ($A = \{A_0, A_1, A_2, \cdots, A_t, \cdots\}$) are passed through time and adjusted according to the input x_t and the last hidden status A_{t-1}. And the output ($h = \{h_0, h_1, h_2, \cdots, h_t, \cdots\}$) is generated form the hidden state A. This gives an RNN the ability to take an unpredetermined length of input and output which makes it able to handle words and sentences of different lengths. This is especially useful in NLP tasks.

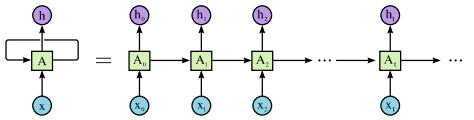

Figure 3. Recurrent Neural Network is a deep neural network in time (reproduced with permission from [48]).

A Long Short-term Memory Network (LSTM) is a special kind of RNN which has an updating rule that is more complex than a simple RNN [18]. In a simple RNN (see Fig. 4), the hidden states and outputs (both h_t in this case) are generated by applying a simple $tanh$ function to the merged vector of the hidden state of previous step h_{t-1} and the input of this step x_t. But in an LSTM (see Fig. 5), different restrictions called "gates" are used to regulate the update behavior which allows the model to determine which neurons in the hidden layer should be "kept" and which should be "forgotten" or "updated".

In an LSTM network called the "Peephole Network" [16], input x_t, output h_{t-1} and the hidden state C_{t-1} are used to decide the update neurons. The updating functions can be seen in Equation 1. The sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ is a function with value between $(0, 1)$. The inputs of each gate are weighted and biased vectors. The weights and bias of each gate are parameters to be learned in the training. By applying σ to the weighted vector, it creates a mask vector of real numbers in the range $(0, 1)$. These mask vectors are the forget gate f, input gate i and output gate o. Weights for the gates are W_f, W_i and W_o, and biases are b_f, b_i,

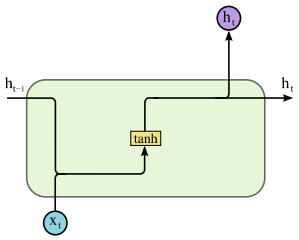

Figure 4. The updating operations in a neuron of Simple RNN (reproduced with permission from [48]).

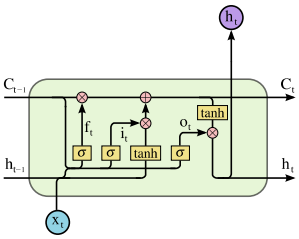

Figure 5. The updating operations in a neuron of LSTM (reproduced with permission from [48]).

b_o, respectively. Then the hidden state C_{t-1} is first multiplied with the forget gate point-wise, so that some of the neurons are regulated towards zero. Then the input vector $tanh(W_c \cdot [x_t, h_{t-1}] + b_c)$ is multiplied with the input gate, then added to the hidden state to get the new C_t. Similarly, the output is obtained by applying $tanh$ to the hidden state, and then going through the output gate. The use of these states helped the LSTM keep long distance information. The application of LSTM to NLP tasks increases the representability of the Neural Network Language Models [17].

$$
\begin{aligned}
i_t &= \sigma(W_i \cdot [x_t, h_{t-1}, C_{t-1}] + b_i) \\
f_t &= \sigma(W_f \cdot [x_t, h_{t-1}, C_{t-1}] + b_f) \\
o_t &= \sigma(W_o \cdot [x_t, h_{t-1}, C_t] + b_o) \\
C_t &= f_t \odot C_{t-1} + i_t \odot tanh(W_c \cdot [x_t, h_{t-1}] + b_c) \\
h_t &= o_t \odot tanh(C_t)
\end{aligned} \tag{1}
$$

In our Character-Level Neural Network Language Model (CLNNLM), we use a Bidirectional Long Short-term Memory (Bi-LSTM) Network to encode the context of the word [40]. As shown in Fig. 6, the Bi-LSTM is actually a combination of two LSTMs which are in different directions. This means that the complete contextual information is available

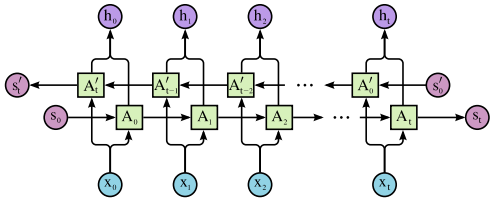

Figure 6. The architecture of Bidirectional RNN (reproduced with permission from [48]).

at each step, where as in the simple LSTM only information about the previous context is known at the current step.

As shown in Fig. 7, our model first embeds each character into a vector and then the vector is put in the Bi-LSTM to generate the predictions for the next letter in the same direction. All the weights are trained jointly including the embedding vectors of each character. The hidden states carry the information obtained from the previous steps. By combining the final hidden state in both directions, the embedding of a word is obtained.

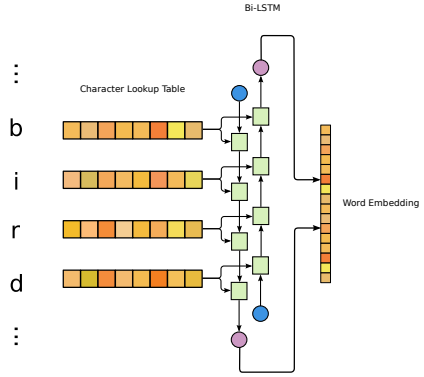

Figure 7. The architecture of our Character-Level Neural Network Language Model (adapted from [40]).

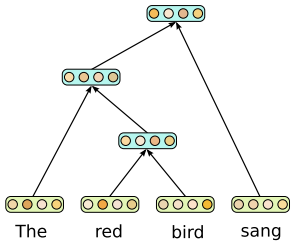

Figure 8. The architecture of Tree-Structured Recursive Neural Network (adapted from [57]).

A TRNN is a recursive neural network on the parsing tree of a sentence [57] (see Fig. 8). Each node takes inputs only from its children nodes and feeds the output to its parent node. This mechanism utilizes the information of sentence structure to eliminate the long-range dependency or discontinuity problem [57]. The parsing tree can be obtained via various popular methods. In our model we used the Stanford Dependency Parser to parse the sentences [9].

The same architecture can be used for word level embeddings as well. We only need to replace the characters in the previous model with words and the embeddings of words can be obtained directly with the output of the CLNNLM. However, as mentioned previously, there are discontinuities in many languages [54], such as "**put** your red hat **on**", "**give** the girl over there **some help**", "He does **not** feel tired after such a long run **at all**", where the phrases in **bold** are split by other words. In such cases, treating the sentence in bulk provides better performance than n-gram methods which only consider the n continuous words [43]. So instead of using the continuous n-gram as the input of the Word-Level Neural Network Language Model (WLNNLM), we use

a Tree-Structured Recursive Neural Network (TRNN) to model the sentences at the word level.

It is also useful to note that the spelling of a word is not a random sampling over the set of possible characters since the words are formed with smaller character groups (like etyma, prefix and suffix) in a more organized way. The probability distribution of the next letter is actually a measurement of the cohesiveness between the current and the next letters. For example, in the case of the word, "preventing", if the model is very confident that the next letter after "i" is "n" and "g", it indicates that "ing" is possibly a character group (in this case, a suffix). By mining the patterns of letters in a word, we find the prefixes and suffixes in English words, and use them as additional features in sentence understanding [40, 41, 55]. In order to identify these features, we added an additional morphological-level layer between the CLNNLM and WLNNLM which takes the states of each hidden neuron in the CLNNLM and splits the words into character groups before feeding them into the WLNNLM.

Fig. 9 shows the architecture of our entire model, the Character-Morphology-Word (CMW) Network. The sentence is first parsed using the Stanford Dependency Parser [9] and the parsing tree is fed to the WLNNLM which is a tree structured neural network. For each word in the parsing tree, the characters are fed to the CLNNLM, which is a Bi-LSTM. The Bi-LSTM predicts the probability distribution of the next character at each step. In the Morphological Layer, the information entropy of the distribution is calculated which represents the uncertainty of the prediction. Since the stems appears in words more often statistically, the uncertainty of the prediction drops within a stem. For example, in the word "biomineralization", the model is more confident that the next letter is "o" after seeing "bi" in the beginning, but less confident about the next letter after seeing "bio" since the stem is complete here and a lot of different stems may follow this. As can be seen in Fig. 9, we cut the word when the uncertainty increases (lighter neurons), and combine the final hidden states of the stem in both directions into a representation of the stem. The stems are also organized in a tree-structured way, in which the stem in the middle is taken as the root of the tree, and all the other stems are dependents of their neighbors that are closer to the root, which forms a "V" shape. Then this revised tree structure is processed using the TRNN of the WLNNLM to combine the information from the children recursively in the tree and form the embedding of the whole sentence at the root of the dependency parsing tree.

4. EXPERIMENTS AND EVALUATION

In this section, we describe the corpus and libraries used for building our model as well as the evaluations used to test the performance of our model.

4.1 Environment Setup

Theano is a linear algebra compiler that optimizes symbolically-specified mathematical computations to produce efficient low-level implementations [3, 6]. Theano provides a high-level mathematical description language to define mathematical functions and then compile and automatically differentiate the functions with existing libraries. Theano is currently used as a development toolkit in neural networks, with many software packages and tools built upon it [60].

Keras is a highly modular neural network library in Python. It supports both Theano and TensorFlow backends [10]. Its optimizers were used in our experiment as training functions.

The experiments were run on a Linux server with two Intel E5-2620 CPUs, 256 GB memory and NVIDIA Tesla Fermi M2090 cGPUs with 6GB memory and 512 CUDA cores. The version of CUDA is 6.0. The version of Theano is 0.7.0. And the version of Keras is 0.2.0.

4.2 Training

Wikipedia [52, 63] is composed of millions of articles in different languages, making it a useful corpus for language model training [39]. As of February 2016, the EnglishWiki has 5,149,619 articles and over 2.6 billion words [61]. The Wikimedia Foundation provides dumps of the English Wikipedia almost every month. We used the 2016-07-01 dump of English Wikipedia.

The copy of Shakespeare's works used in our training was obtained via Project Gutenberg [56], which includes 38 plays written by William Shakespeare (1564 âĂŞ 1616). The corpus has 5,338,964 tokens, 883,331 words and 122,690 lines.

Kiros *et al.* [31] propose a sentence based training procedure which uses the current sentence s_i to reconstruct the previous sentence s_{i-1} and the next sentence s_{i+1} (skip-thought framework). We used a training procedure similar to this. However, since it is difficult to reconstruct sentences using a TRNN, we added an extra softmax layer to the top of the model during the training process to translate the sentence embedding output by our model into the probability distribution of words in the previous and next sentences (the word orders in these sentences are ignored for simplicity).

The 2016-07-01 dump of English Wikipedia training corpus has 59,784,453 sentences. All the sentences that do not have previous or next sentences in the same paragraph are discarded. This leaves 26,719,225 valid triples (s_{i-1}, s_i, s_{i+1}). We split the triples into training (80%) and validation (20%) sets. The object function in training is the cross entropy [51] between the predicted and actual distributions of words. The optimization algorithm used is Stochastic Gradient Descent (SGD) [8]. We used batches of 51,200 triples to train the model. The total number of batches is 5000 so that the model is trained on the whole corpus for roughly 10 times. The Shakespeare corpus only has 57,881 valid triples so we used batches of 128 instead of 51,200 while kept other setups the same as the Wikipedia model.

The models in Levy *et al.* [37] were also trained on the Wikipedia corpus. The versions of Wikipedia used in [37] and our experiments may be different. Furthermore, instead of the 5 word context window which is used in Levy *et al.* [37], the context used for training our CMW network is the 3 sentences context according to the skip-thought framework in Kiros *et al.* [31]. Due to the limitation of space, in Levy *et al.* [37] only words appearing 100 times or more are included in the vocabulary which includes 189,553 words. This may result in the Out-Of-Vocabulary (OOV) problem in their model. However, it is not an issue in our character-based network since the total number of characters appearing in the corpus is 205. For comparison reasons, we embedded the words into a 600-dimensional space which is the same as the embedding model in Levy *et al.* [37].

In both the models in Levy *et al.* [37] and our Wikipedia CMW network, the functional and meta information in the training corpus such as hyperlinks, titles, section names and references has been removed, but some of the prepossessing

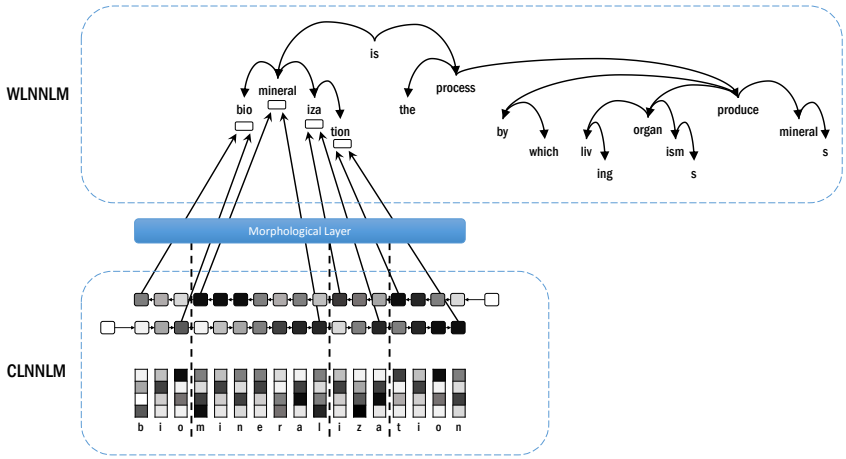

Figure 9. The architecture of our whole CMW network model.

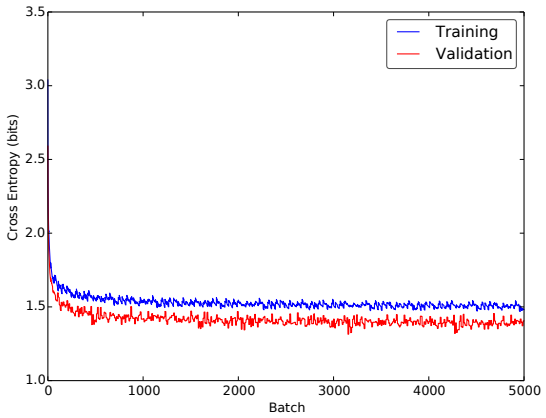

Figure 10. The cross entropy over training batches.

in [37] such as lowercasing all characters & filtering non-alphanumeric tokens are not used in our model. Because our CMW model uses character level input, we wanted to keep the modifications on the original corpus to the minimum. Preserving the case and special tokens such as punctuation in the training corpus can help the model learn lexical phenomenons such as proper names so that it can distinguish between "Smith" and "smith", but this might also negatively affect the performance by mapping "The" in the beginning of a sentence and "the" in the middle of a sentence to different words. During the training process, Levy *et al.* [37] also used negative sampling to regularize the model while we used the skip-thought framework in Kiros *et al.* [31] which did not utilize this trick. These differences might negatively impact the performance of the CMW model on the word based tests. Compared to the neural network based embedding model in [37], our CMW network is more complex and has larger number of parameters, thus requires more time for training. We had hoped this would help our model achieve better results but as seen below we see better results in only one domain and comparable results in the others.

The change of training and validation cross entropies at each batch during the training on Wikipedia can be seen in Fig. 10. The lower cross entropy basically means that the predictions of the model matches the training cases better. The training functions from Keras library was used in our training process which has a known issue of reporting lower loss in the validation sets than training sets due to different evaluation methods[28]. It can be seen that the model quickly converged during the first 500 batches and stayed fluctuating during the rest of training. This may suggest that the model is not taking advantage of the whole training corpus. The reason for this could be that either the corpus is too big for the model, or the training algorithm is not optimizing the model effectively. We tried several different training algorithms such as AdaGrad [13] and AdaDelta [62]. The trainings using these algorithms also converged early around 500-1000 batches. As a result, we believe that the model is well trained using the corpus.

4.3 Word Embedding

One of the benefits of using a character level model for word embedding is that it can process any words based on their spelling, even if the word has never appeared in the vocabulary during training, the so-called the out-of-vocabulary words. Words such as the names of people, places and animals are usually very rare while following some patterns, such as "Dogville". By using character level inputs, we can identify such patterns and use it for embedding unseen words.

Table 1: Most similar words of Out-Of-Vocabulary tests

	Test Word			
	goood	misinformativeness	Xiatoujiao	Minemoto
Similar Words	good	informativeness	Shatoujiao	Minamoto
	Good	misinformation	Zhujiajiao	Minimoto
	great	Disinformation	Zhoushan	Fijimoto

As can be seen in Tab. 1, we coined four testing words that are not seen in the training corpus. We used our word embedding model to embed these OOV words using its characters, and compared the word vectors with every word in the corpus vocabulary. The similarity between the test and vocabulary word vectors $(\vec{v_t}, \vec{v_v})$ are measured by the cosine

distance $cos(\vec{v_t}, \vec{v_v})$. The most similar three words for each test are listed in the table. The result shows that our model can handle the character-level input of unseen words, and embed it into the word embedding space based on spelling similarity. In the case of "goood", it correctly linked it to the correct spelling "good" using character level information. And for "misinformativeness" which is a compositional word, it seems that our model correctly identified the roots "mis", "informat", "ive", "ness" which are matched or replaced in the three most similar words returned by the model. And for the Chinese and Japanese names, it can be clearly seen that the returned similar words are of similar forms.

4.3.1 Word Similarity

Our word embedding result can be compared with other methods using the Microsoft Research (MSR) data set [44] and the Google data set [42]. We used these data sets to compare our model to two models presented in [37].

The Word Similarity data sets by Microsoft and Google are word pair data sets with similarity scores between each pair assigned by humans. For each left side word a, several right side words $b_1 \cdots b_n$ are compared with a and the similarity scores $x_1 \cdots x_n$ are assigned where x_i measures the similarity between a and b_i. In order to evaluate the word embedding result, the similarity between word a and b_i is given by the cosine similarity between the word vectors $\vec{w_a}$ and $\vec{w_{bi}}$ of the words, *i.e.* $y(a, b_i) = \vec{w_a} \times \vec{w_{bi}} / ||\vec{w_a}|| \times ||\vec{w_{bi}}||$. The accuracy of the word embedding result can be measured using the Spearman's correlation between the orders of the output similarity scores and the human ratings. The state-of-the-art methods yield 60.5% to 69.1% accuracy [36].

The MSR and Google data sets are two analogy datasets, containing questions in the form of "a is to a* as b is to b*" [42]. In tests, a, a* and b are given to the system, and the result output by the system is compared with b*. The MSR data set contains 8000 syntactic analogy questions, such as "good is to best as smart is to smartest", and the Google data set contains 19,544 such questions, about half of which are syntactic questions and the other half are semantic questions such as capital of countries ("Paris is to France as Tokyo is to Japan") [42]. The analogy questions can be answered with Levy and Goldberg's similarity multiplication method [44] which is an analogy recovery algorithm as follows,

$$\underset{b' \in V_W \setminus (a, a^\star, b)}{\arg\max} cos(b', a^\star - a + b) = \frac{b' \cdot (a^\star - a + b)}{\|b'\| \cdot \|a^\star - a + b\|} \quad (2)$$

where $b^\star$ is the correct answer, $b\prime$ is the answer returned by the system, V_W is the vocabulary, ϵ is an infinitesimal which is used to avoid zero in the denominator. The accuracy rate for the analogy questions is defined as the percentage of questions for which the *argmax* result was the correct answer; that is, where $b\prime = b^\star$ [37]. State-of-the-art methods yield 14.55% to 99.41% accuracy, varying between different domains [37]. The average accuracy is 59. 09% on the MSR data set, and 68.24% on the Google data set [37].

Two models are used in [37], a neural network based *embedding model* and a matrix based *explicit model*. The embedding model is based on the word vectors obtained using word2vec as described in [42, 43]. The model is a single hidden layer feed forward neural network that takes the word at the center as the input and predicts its context words within the distance of 2 (*i. e.* 5 words in total). Then the word embedding is obtained from the neural network. Some tricks such as negative sampling are used in [37] to make the result more robust.

The explicit model is a matrix based model. The value of each element is the positive pointwise mutual information between a pair of words [11, 12, 59] based on a co-occurrence matrix of length 4 (2 words on each side), which is basically the maximum of zero and the mutual information between the pair of words. This sparse matrix of size $\|V_W\| \times 4\,\|V_W\|$offers the word embeddings used in the word similarity task using the argmax formula (Formula 2).

Tab. 2 shows the results of our test on the Google & MSR datasets. We compared our result to the two models by Levy *et al.* [37]. We also recalculated the number of correct and incorrect samples in [37] using the accuracy and the number of samples in each category.

As can be seen from the tables, our model performs comparatively well to [37]. The overall performance is around 11% worse than the two models of Levy *et al.* [37]. However, the performance of our method on the Google dataset is very close to Levy *et al.* [37], while the performance on MSR dataset is considerably worse. For the *gram1-adjective-to-adverb*, our model performs better than the two Levy models. It might be that our model has more parameters and could get stuck in local minimum or overfit, or the current method used in training is not sufficient to find an optimal result in the search space. Since we have tried several different optimization algorithms including SGD [8], AdaGrad [13] and AdaDelta [62] and the object functions always converge early in the training process, we think that might not be the main issue in this case. It is also possible that the size of neurons in each layer are not big enough for the model, which has reduced the ability of our model to represent the complexity of the language. However, the dimension of the character-level is restricted to 600 to be the same as [37] for comparison reasons, but we may use a higher dimension and use dimensional reducing methods such as Principle Component Analysis to resize to 600. This could be explored in the future.

4.3.2 Automatic Text Generation

A very popular demo for showing the representability of neural network language models is automatic text generation [20]. Although there is no public gold standard for evaluating the output text, the fluency and intelligibility of the auto-generated texts are often viewed as a demonstration of the generalizability of the language model. . We automatically generated text using our model trained on two different corpora, Wikipedia [52, 63] and Shakespeare's works [56] using a random initial hidden state.

We used our model to generate the text automatically with a random initial hidden state. First we generate a seed sentence from the training corpus which ends at the first space after the 40^{th} character. The seed is fed into the CLNNLM model to initialize the hidden states. The characters after the seed are randomly picked according to the probability distribution predicted by the CLNNLM, and fed back to the CLNNLM to update the hidden states and generate the next character. We generated 100 samples of 450 characters for each of the models which have 12,781 words in total. The following are two samples of generated text:

Wikipedia: Zanzibar's economy is based primarily on a few

Table 2: Word similarity results comparing with related researches

	Domain	Levy		CMW Net	Difference	
		Embedding	Explicit		Embedding	Explicit
Google	capital-common-countries	90.51%	99.41%	89.71%	-0.80%	-9.70%
	capital-world	77.61%	**92.73%**	75.23%	-2.38%	-17.50%
	currency	56.95%	**64.69%**	50.20%	-6.75%	-14.49%
	city-in-state	**14.55%**	10.53%	6.32%	-8.23%	-4.21%
	family	**76.48%**	60.08%	65.61%	-10.87%	**5.53%**
	gram1-adjective-to-adverb	24.29%	14.01%	**33.22%**	**8.93%**	**19.21%**
	gram2-opposite	**37.07%**	28.94%	31.77%	-5.30%	**2.83%**
	gram3-comparative	**86.11%**	77.85%	65.36%	-20.75%	-12.49%
	gram4-superlative	56.72%	**63.45%**	24.15%	-32.57%	-39.30%
	gram5-present-participle	63.35%	**65.06%**	57.97%	-5.38%	-7.09%
	gram6-nationality-adjective	89.37%	**90.56%**	88.53%	-0.84%	-2.03%
	gram7-past-tense	**65.83%**	48.85%	47.16%	-18.67%	-1.69%
	gram8-plural	72.15%	**76.05%**	70.76%	-1.39%	-5.29%
	gram9-plural-verbs	**71.15%**	55.75%	67.14%	-4.01%	**11.39%**
	average	**59.09%**	56.82%	56.26%	-2.83%	-0.56%
MSR	adjectives	45.88%	**56.46%**	42.47%	-3.41%	-13.99%
	nouns	56.96%	**63.07%**	50.35%	-6.61%	-12.72%
	verbs	**69.90%**	52.97%	50.13%	-19.77%	-2.84%
	average	66.71%	**68.24%**	47.31%	-19.40%	-20.93%
Average		64.66%	**65.17%**	53.66%	-11.00%	-11.51%

of the recent notable control of his tribes have been proposed to be stored in the term in the candidate of the Catholic in the Medical and Stephens (1984), and the infraction of the Member of the Slave and Western Charles A. Bottle, Canadian form of since 1961 by the United States and proposed with the United States Art and Street for the Assembly of Consider Stone Bartare.

Shakespeare: Despite of wrinkles this thy golden tinow you stand not your presence and less in mine eyes are so and born and her oy the some state of HESTINGS and TROILUS, and MARCUS, with the master of the Paris, that I should be for the stone, the content, to the most sorrow in the behold of this bloody power and the strength and to be my son which when he shall be here and send the widow of the friends of the King.

Of the 5824 words in the randomly generated Wikipedia sample, only 171 of them did not appeare in the corpus. Of the 6057 words in the randomly generated Shakespeare sample, only 132 of them did not appeare in the corpus. Many of these out-of-vocabulary words are capitalized and appear as proper names such as Bartare. Only less than 2.37% of the character-by-character generated words are out of the training corpus. We also imported the automatically generated text into Word. The Word Spell Checker reports 534 misspelled words, making up 4.18% of the generated text. Considering it is generated with our character-level model one character by one character, it is interesting that most of the words produced are spelled correctly. Another thing to note is that for each model trained with different corpora, the writing style is different, which captures the way each author writes in the training text.

5. CONCLUSIONS AND FUTURE WORK

In this research we proposed a CMW neural network language model which takes character level input and encodes the sentences into vectors. The network is composed of two deep neural networks and a morphological layer in between which uses information obtained from the character level network to break words into stems. We trained the CMW network using the Wikipedia corpus and produced a set of vectors mapping English words into a continuous space. The vectors can be used in various scenarios in NLP, such as machine translation, knowledge extraction, information retrieval, and dialogue systems.

We demonstrated our model using word similarity tasks and automatic text generation. We compared our word embedding results with two previous models using word similarity tasks and achieved some comparable performance. Potential problems that may be affecting the performance of our model include inadequate training data, inadequate training mechanism to finding a satisfying convergence. It is also possible that some of the word similarity tasks require knowledge beyond the character and word levels, such as the country-currency pairs in the Google dataset. A knowledge system might answer such questions better and obtain higher accuracies. We believe this could be the most possible reason.

In the future, the model can be modified to utilize other training corpora, such as using the words and definitions from the dictionary as the output and input of the network. The current method of splitting words into stems was based on the probability distribution predicted by the CLNNLM. One area for future exploration is to study whether training or pretraining the CLNNLM with morphological datasets could enhance the performance of the model.

6. ACKNOWLEDGMENTS

Our thanks to Frank Rudzicz for feedback on this work. The research was partially funding by the GRAND NCE.

7. REFERENCES

[1] J. Andreas and D. Klein. How much do word embeddings encode about syntax. In *Proceedings of ACL*, 2014.

[2] E. Arisoy, T. N. Sainath, B. Kingsbury, and B. Ramabhadran. Deep neural network language models. In *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, pages 20–28. Association for Computational Linguistics, 2012.

[3] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*, 2012.

[4] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003.

[5] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994.

[6] J. Bergstra, F. Bastien, O. Breuleux, P. Lamblin, R. Pascanu, O. Delalleau, G. Desjardins, D. Warde-Farley, I. Goodfellow, A. Bergeron, et al. Theano: Deep learning on GPUs with python. In *NIPS 2011, BigLearning Workshop, Granada, Spain*, 2011.

[7] P. Bojanowski, A. Joulin, and T. Mikolov. Alternative structures for character-level rnns. *arXiv preprint arXiv:1511.06303*, 2015.

[8] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[9] D. Chen and C. Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar, October 2014. Association for Computational Linguistics.

[10] F. Chollet. keras. https://github.com/fchollet/keras, 2015.

[11] K. W. Church and P. Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990.

[12] I. Dagan, F. Pereira, and L. Lee. Similarity-based estimation of word cooccurrence probabilities. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 272–278. Association for Computational Linguistics, 1994.

[13] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[14] W. Finder. 3 letter words. http://www.wordfind.com/3-letter-words/.

[15] J. R. Finkel, A. Kleeman, and C. D. Manning. Efficient, feature-based, conditional random field parsing. In *ACL*, volume 46, pages 959–967, 2008.

[16] F. A. Gers and J. Schmidhuber. Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 189–194. IEEE, 2000.

[17] F. A. Gers and J. Schmidhuber. LSTM recurrent networks learn simple context-free and context-sensitive languages. *Neural Networks, IEEE Transactions on*, 12(6):1333–1340, 2001.

[18] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[19] Y. Goldberg and O. Levy. word2vec explained: Deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

[20] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[21] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE, 2013.

[22] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[23] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[24] Z. Huang, W. Xu, and K. Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[25] O. Irsoy and C. Cardie. Deep recursive neural networks for compositionality in language. In *Advances in Neural Information Processing Systems*, pages 2096–2104, 2014.

[26] R. Johnson and T. Zhang. Semi-supervised convolutional neural networks for text categorization via region embedding. In *Advances in neural information processing systems*, pages 919–927, 2015.

[27] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[28] Keras. Why is the training loss much higher than the testing loss? https://keras.io/getting-started/faq/why-is-the-training-loss-much-higher-than-the-testing-loss, 2016.

[29] Y. Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[30] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush. Character-aware neural language models. *arXiv preprint arXiv:1508.06615*, 2015.

[31] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler. Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302, 2015.

[32] T. Koo, X. Carreras, and M. Collins. Simple semi-supervised dependency parsing. 2008.

[33] Q. V. Le. Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8595–8598. IEEE, 2013.

[34] B. B. Le Cun, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*.

Citeseer, 1990.

[35] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[36] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems*, pages 2177–2185, 2014.

[37] O. Levy, Y. Goldberg, and I. Ramat-Gan. Linguistic regularities in sparse and explicit word representations. *CoNLL-2014*, page 171, 2014.

[38] Y. Li, J. Liu, Q. Bao, W. Xu, R. Sadiq, and Y. Deng. A new method of mapping relations from data based on artificial neural network. *International Journal of System Assurance Engineering and Management*, 5(4):544–553, 2014.

[39] W. Ling, C. Dyer, A. Black, and I. Trancoso. Two/too simple adaptations of word2vec for syntax problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2015.

[40] W. Ling, T. Luís, L. Marujo, R. F. Astudillo, S. Amir, C. Dyer, A. W. Black, and I. Trancoso. Finding function in form: Compositional character models for open vocabulary word representation. *arXiv preprint arXiv:1508.02096*, 2015.

[41] M.-T. Luong, R. Socher, and C. D. Manning. Better word representations with recursive neural networks for morphology. *CoNLL-2013*, 104, 2013.

[42] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[43] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[44] T. Mikolov, W.-t. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751, 2013.

[45] S. Miller, J. Guinness, and A. Zamanian. Name tagging with word clusters and discriminative training. In *HLT-NAACL*, volume 4, pages 337–342, 2004.

[46] A. Mnih and G. Hinton. Three new graphical models for statistical language modelling. In *Proceedings of the 24th international conference on Machine learning*, pages 641–648. ACM, 2007.

[47] A. Mnih and G. E. Hinton. A scalable hierarchical distributed language model. In *Advances in neural information processing systems*, pages 1081–1088, 2009.

[48] C. Olah. Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015.

[49] S. Oreski, D. Oreski, and G. Oreski. Hybrid system with genetic algorithm and artificial neural networks and its application to retail credit risk assessment. *Expert Systems with Applications*, 39(16):12605 – 12617, 2012.

[50] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12:1532–1543, 2014.

[51] R. Y. Rubinstein and D. P. Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media, 2013.

[52] M. Ruiz-Casado, E. Alfonseca, and P. Castells. Automatic assignment of wikipedia encyclopedic entries to wordnet synsets. In *International Atlantic Web Intelligence Conference*, pages 380–386. Springer, 2005.

[53] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[54] L. M. Santelmann and P. W. Jusczyk. Sensitivity to discontinuous dependencies in language learners: Evidence for limitations in processing space. *Cognition*, 69(2):105–134, 1998.

[55] C. D. Santos and B. Zadrozny. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1818–1826, 2014.

[56] W. Shakespeare. The complete works of william shakespeare. In *The Complete Works of William Shakespeare*. Project Gutenberg EBook, 2011.

[57] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.

[58] O. Täckström, R. McDonald, and J. Uszkoreit. Cross-lingual word clusters for direct transfer of linguistic structure. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 477–487. Association for Computational Linguistics, 2012.

[59] P. D. Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl. In P. De Raedt, Lucand Flach, editor, *Machine Learning: ECML 2001: 12th European Conference on Machine Learning Freiburg, Germany, September 5–7, 2001Proceedings*, pages 491–502. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[60] B. van Merriënboer, D. Bahdanau, V. Dumoulin, D. Serdyuk, D. Warde-Farley, J. Chorowski, and Y. Bengio. Blocks and fuel: Frameworks for deep learning. *arXiv preprint arXiv:1506.00619*, 2015.

[61] Wikipedia. Wikipedia:size comparisons. https://en.wikipedia.org/wiki/Wikipedia:Size_comparisons.

[62] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[63] T. Zesch, I. Gurevych, and M. Mühlhäuser. Analyzing and accessing wikipedia as a lexical semantic resource. *Data Structures for Linguistic Resources and Applications*, pages 197–205, 2007.