Unsupervised Neural Network approaches for Morphological Analysis

Daisuke Yoda, Kentaro Asaba, Samuel Dedeurwaerdere, Tatsuro Yamamoto
Keio University
Introduction to AI - Team 4

Abstract

In the field of morphology, many researchers have attempted to conduct or improve their analyses in a quantitative way. However, there is plenty of scope for improvement by employing the latest deep learning techniques. This paper presents three different unsupervised morphemes discovery models and compares the performances with the state-of-the-art model presented by Soricut and Och (2015)[13]. Among the three of them, we can especially say that Character Prediction Model based on the concept of Confidence has great potential and that Augmented Attention Model performs an outstanding result. Our newly proposed ideas and techniques will enable us to conduct more precise analyses in the linguistic field.

1 Introduction

In the past years, many discoveries have been done in linguistic analysis. Indeed, new AI models have given new opportunities to understand better the essence of our languages. Many types of research have emphasized the importance of linguistic analysis using machine learning techniques because it will lead to the next big step in linguistics and artificial intelligence. Not only it will lead to the discovery in the infant learning process (Saffran et al., 1996 [12]), but also many other concrete implementations such as machine translation (Cho, et al., 2014 [2]), speech recognition (Graves, et al., 2013 [6]) etc.

Morphological analysis is a broad topic. The aim of this paper is to focus on exploring new ways of finding morphological rules in an unsupervised way. By understanding what has already been discovered around this question, we will propose new ways for morphemes analysis. Before introducing and comparing our models it's important to understand the methodology we have used for this paper.

The starting point to build our models was a paper from Soricut and Och (2015)[13] answering the question of unsupervised morphology discovering by analyzing the vector representation of pairs of words in the Word2Vec or GloVe vector space. Their research shows that these two vector spaces contain information about grammar. For example, we can hope that the vector between the pair (create; created) is close to the vector between the pair (change; changed) in some direction of the embedding space. This feature understanding of these vector spaces can be used to find morphological rules. Model 1, described in details in section 3 of this paper reproduces this idea. We find candidate pairs by character similarity that could describe a morphological rule (prefix, suffix). We then compute the cosine similarity between the two vectors to discover if we keep the pair. We finally map the entire set of pairs around a base word.

By exploiting state-of-art methods including LSTM (Long-Short-Term Memory), we will propose new ways (model 2, model 3 and model 4) of answering the same question as model 1.

The main idea behind the model 2 is to predict the next character of a word. For instance, the probabilities that the next character in the word "chang" is "e" or "i" are high (change, changing).

Overall, by training our Character Prediction Model, we try to discover a new way to find prefix and suffix.

Later on, in the exploration process, we reoriented our morpheme exploration to a more intuitive level. Indeed, model 3 tries to separate a base word and its prefix or suffix by learning morphemes, rather than predicting the next letter, or taking word similarities into account. In model 3, we divided the morphological search into three interconnected steps. Firstly, DNN (Deep Neural Network) decides after each character to split or not the word. Secondly, each split is embedded separately into a vector on a character-based approach. Thirdly, through a final network, the model combines the vectors of split words and compares the reconstruction to the original GloVe or Word2Vec vector of the word.

In order to build a more robust and compact network than model 3, model 4 will be presented. In model 4, firstly, pre-trained GloVe (or Word2Vec) vectors are used in the initial hidden state of a character-based LSTM. Secondly, we concatenate all the hidden states of this LSTM into a 1 column vector. Thirdly, this attention vector is then the input of a one layer network that outputs the separation points of words. Model 1 results (found in an unsupervised way) are used to train model 4, to impose the separation points of words.

Each model employs latest Deep Learning techniques, so reviews of related researches and description of main existing models used in our work (LSTM, attention mechanism, etc.) will be presented in section 2.

Finally, section 4 we will present the main advantages and disadvantages of each model and discuss if our exploration of the topic has led to new ways of approaching this AI-based morpheme exploration. Moreover, we will discuss if it has led to new ideas and open some curiosity for further papers.

To summarize this introduction the main goal of this paper is to explore new ways of unsupervised morphological analysis.

2 Related Works

In this section, we will introduce the significantly related works in the field and main machine learning techniques that we implemented in our models.

2.1 Word2Vec and GloVe

To use machine learning to analyze a language, we have first to encode characters, words, sentences or paragraphs into vectors that can be trained. One famous way of embedding words is by using linguistic context. Word2Vec and GloVe are the two main models that use this context-based word embedding approach. This means that by analyzing in which context words appear, we can extract the meanings of them in an unsupervised way. There are some differences between Word2Vec, developed at Google (Mikolov, 2013 [9]), and GloVe, an open-source project developed at Stanford (Pennington, 2014 [11]).

The Word2Vec model represents each word regarding other words nearby in the context. Words that have similar meanings have similar vector representations. Two architectures are possible to choice nearby words: SkipGram or CBOW. Respectively, the first one is more accurate but computationally heavier than the second because SkipGram uses a fixed amount of words around each word while CBOW uses a fixed "bag" of words without considering the order.

The GloVe model uses a statistical approach. The co-occurrence matrix is obtained by counting frequency of words in a given context. Then, word embedding is calculated by using the co-occurrence matrix.

These context-based embedding models lead to the following results: the vector representing "queen" minus the vector for "women" plus the vector for "men" equals to a vector of "king". Many other meaning descriptions can be found in these vector spaces. Moreover, these representations could be used for different linguistic tasks like translation, picture description, understand the meaning of new words, etc.

Furthermore, Soricut and Och (2015)[13] claim that one can uncover morphological transformation such as suffix and prefix from context-based vector representations and proved that by showing the

results in six languages. More precisely, they used Skip-gram algorithm. The main contributions of their study can be summarized as follows:

1. Their methods were language agnostic.
2. Their method works for known words.

This paper is unique compared to the previous works for that they do not assume any knowledge about the morphology, and do not use any parser before embedding. This unsupervised method will be our starting point for our model 1.

2.2 (Simple) RNN

One of the characteristics of language modeling is that it has to take the word or character orders into account in order to understand the flow and the order. It has been shown that texts can be encoded efficiently using RNN (Recurrent Neural Network) and LSTM (Long Short-Term Memory) (Pascanu, 2013 [10]). Figure 1 displays the character-based RNN architecture. Under char-RNN, the probability of word $\boldsymbol{w}$ appears can be calculated as:

$$p(\boldsymbol{w}) = \prod_{t=1}^{|\boldsymbol{w}|} p\left(x_t | \left\{x_1, \cdots, x_{t-1}\right\}\right). \tag{1}$$

As in the figure, RNN creates outputs y based on inputs x and hidden states $\mathbf{h}$. RNN has hidden states $\mathbf{h}$ that is recursively updated as follows:

$$\mathbf{h}_t = f\left(\mathbf{h}_{t-1}\mathbf{W}_{\mathbf{h}} + \mathbf{x}_t\mathbf{W}_{\mathbf{x}} + \mathbf{b}\right). \tag{2}$$

where $f(\cdot)$ is an activation function, $\mathbf{W}_{\mathbf{x}}$ and $\mathbf{W}_{\mathbf{h}}$ are trainable weights, and $\mathbf{b}$ is a trainable bias. Using the hidden states in every time-steps, a vector c which represents the whole inputs can be derived as with another activation function g:

$$c = g\left(\left\{\mathbf{h}_1, \cdots, \mathbf{h}_T\right\}\right). \tag{3}$$

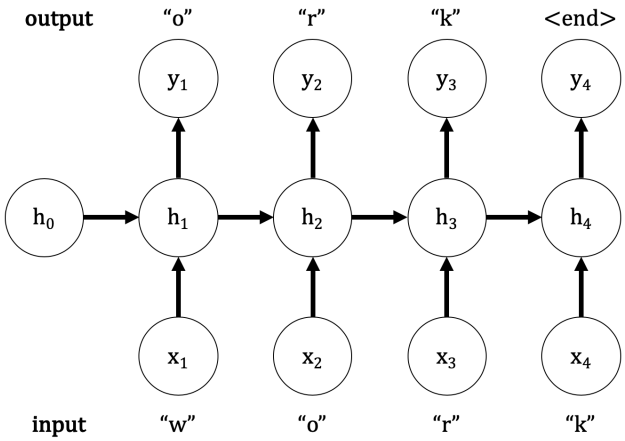

Figure 1: **Visualization of the Char-RNN**. Char-RNN is trained to predict the next character. The inputs are one-hot vectors of characters, and the outputs are probability distributions of each character.

One of the advantages of char-RNN is that it is robust to create vector representations for unseen words. However, Bahdanau, et al. (2014)[1] argue that since common RNN encoding method can remember only short-term information, the prediction performance decreases as a length of the input gets larger. Therefore, RNN is ineffective for particular tasks such as translations of long sentences. Many researchers have attempted to improve RNN models so that they can understand morphological information better than rule-based morphology, and developed useful tools such as machine translation. Chung et al.(2017)[3] invented Hierarchical Multiscale Recurrent Neural Networks (HM-RNN) which can derive a sentence's hierarchical information by using character encoders, word encoders, character decoders, and a softmax output layer.

2.3 LSTM

LSTM is a typical example of a gated RNN, and this can significantly improve simple RNN's performance. The reason for this is that simple RNN is not good at learning long-term dependency of time series data. This is caused by a vanishing gradient problem or a gradient explosion when BPTT (Back Propagation Through Time) is used in a simple RNN.

LSTM is a model that can deal with the unstable gradient problem by introducing a storage cell ($\mathbf{C}_t$) that is described later. The flow of memory $\mathbf{C}_t$ and hidden layer $\mathbf{h}_t$ is controlled by the three gates (the input gate, the forget gate, and the output gate).

In order for the storage cell to propagate and memorize correctly, the concept of "gate" is necessary. The gate refers to the role of controlling the flow of data. Each gate has its own weights and is controlled by using the sigmoid function to output a real number between 0.0 and 1.0. The following describes the definition of each gate and what the role is.

The newly introduced storage cell ct in LSTM is a matrix that contains all the information required from the past to the time t. The memory cell ct can be determined by calculating the three elements of $\mathbf{h}_{t-1}$, $\mathbf{c}_{t-1}$ and $\mathbf{x}_{t-1}$ through the gate. First, the memory cell $\mathbf{C}_t$ forgets the unnecessary memory from $\mathbf{c}_{t-1}$ by passing through the forget gate, and then update the memory by adding new information passing through the input gate.

The forget gate can be formulated as:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right). \tag{4}$$

The newly added information can be formulated as:

$$\widetilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right). \tag{5}$$

The input gate can be formulated as:

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right). \tag{6}$$

Therefore, the memory cell $\mathbf{C}_t$ can be formulated as:

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t. \tag{7}$$

The output gate is formulated as: $\tanh\left(C_t\right)$

$$o_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_o\right). \tag{8}$$

Finally, the hidden gate is formulated as:

$$h_t = \tanh\left(C_t\right). \tag{9}$$

To summarize, compared to RNN, LSTM has an extra input C_t which enables LSTM to keep long-term memory. LSTM also has a mechanism to link C_t with $\mathbf{h_t}$ through newly introduced gates. This system is applicable to morphological analysis and shows better results than the simple RNN. The reason is that as the dataset and word length get larger, simple RNN cannot tolerate the computation. Moreover, because languages have been created over thousands of years and have complex structures, LSTM is useful in finding back the original dependencies.

2.4 Attention

In this section, we first explain what attention is, and then describe how it works. The mechanism of learning the link between two time-series data is called the attention mechanism. By implementing it to LSTM, the performance of the model is expected to be improved.

Attention is particularly introduced for the purpose of improving seq2seq, which is a method for encoding/decoding a sentence. In the concept of seq2seq, the encoder can only transfer the last element of hidden state to the decoder. However, this structure often deteriorates the performance of seq2seq since the model has the possibility of failing to retain its states. On the other hand, the

encoder of the attention mechanism can transfer all the hidden state vectors to the decoder. This is why the attention is theoretically superior to the seq2seq.

Firstly, the attention mechanism calculates a matrix "$\mathbf{a}$" which represents the similarity between the whole hidden state vector $\mathbf{h} = [\mathbf{h_1} \cdots \mathbf{h_T}]$ and a hidden vector of a single character $\mathbf{h}_i$. "$\mathbf{a}$" can be formulated as:

$$\mathbf{a} = (softmax(\mathbf{h} \cdot \mathbf{h_1}) \cdots softmax(\mathbf{h} \cdot \mathbf{h}_i) \cdots softmax(\mathbf{h} \cdot \mathbf{h}_T)) \,. \tag{10}$$

This matrix "a" is calculated by using activation function (to be specific softmax function which is a multivariable version of sigmoid function as mentioned in the previous section.). Second attention calculates the dot product "c" of the vector "hidden state" and "a". This "c " is often called a "context vector" and is the output of the attention mechanism.

In morphological analysis, attention mechanisms can be interesting because backpropagating attention allows us to influence the LSTM hidden layers. Moreover, we can influence it because all the processes in the attention are differential functions, thus the loss can be backpropagated in the LSTM.

3 Models

In this section we will present our 4 models independently. The results and discussion about the performance of each model will be the aim of section 4.

- Model 1: Word Embedding Space Model
- Model 2: Character Prediction Model
- Model 3: Three-net Model
- Model 4: Augmented Attention Model

As already introduced, the models have been developed in an experimental approach to answer the following topic: unsupervised morphemes discovery.

3.1 Model 1: Word Embedding Space Model

As already introduced in the Related Works, this first model is inspired by the work of (Soricut & Och, 2015)[13]. By using word embedding trained through unsupervised learning, it avoids mistakenly create meaningless morphological transformation, such as "*only*" $\rightarrow$ "*on*" + "*ly*".

Step 1: Create a vocabulary list
At first, vocabulary list V is created from a monolingual row text. In our research, Brown Corpus [1] is used as a row text. Brown Corpus has enough credibility because it has sufficient text size and is widely used in linguistics.

Step 2: Create prefix and suffix groups
Then, groups are created around suffix and prefix rules. For example, in (suffix : $\epsilon : ed$) group, we have words such as ({work $\rightarrow$ worked}, {talk $\rightarrow$ talked}, etc.). This is simply done by comparing characters of the beginning and ending of each word in the vocabulary list. To avoid the creation of irrelevant rules, we eliminate groups less than five pairs. For instance, the word pair (an $\rightarrow$ anaconda) has rule: (suffix : ϵ : $aconda$), but this group is removed because it is a very rare case and not considered as a grammatical rule.

Moreover, to be sure the words inside each pair are connected in meaning, we eliminate pairs with low cosine similarity. In our research, the threshold of cosine similarity is set as 0.7. For example, this avoids to put scream and cream in the same group.

Step 3: Create groups around each word
For each word of the vocabulary list (called as a target word), we extract all the words that construct a suffix/prefix rules around the target word and make a group.

To go further, hit rate could be calculated like in the Soricut and Orh (2015) research [13] to find the stem words. We will show mapping examples around words in the Result section.

[1] http://www.nltk.org/nltk_data/

We summarize the model 1 step by step in the Algorithm 1.

Algorithm 1 Making word pairs in Word Embedding Space Model

```
procedure CREATE_PAIRS(Vocabulary_list)
    List = []
    Pairs = { }
    for pair in all_pairs do                ▷ all_pairs: all combination of 2 words
        if the rule of pair is not minor AND similarity(pair) > 0.7 then
            APPEND pair to list
    pairs[word] = similar_words
    for word in Vocabulary_list do
        if word in list then
            extract the pairs of word from list, and add it to Pairs
    return Pairs
```

3.2 Model 2: Character Prediction Model

The second model is an alternative to the first one. While the first model relies on the character-based word similarities and cosine similarities of word embedding to find stem words and morphological information, this model uses two Char-LSTM to extract the information.

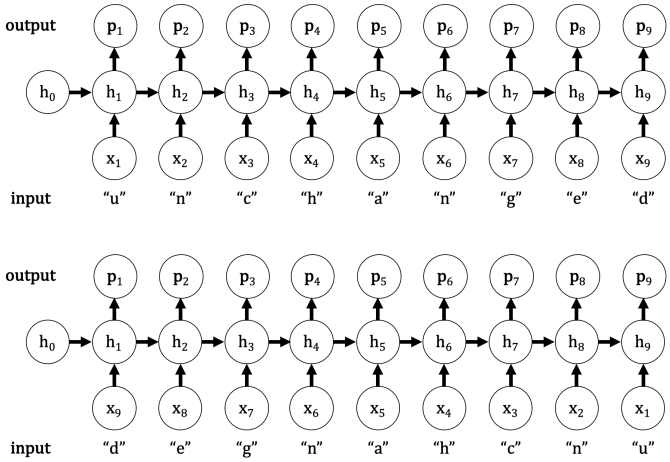

Figure 2: **Example of Character Prediction Model on word "unchanged"**. By reading the words from the beginning and the ending, the model can extract both the suffix and prefix information. The top figure displays the LSTM that reads characters in normal order, and it aims to obtain suffix "*d*". Conversely, the second LSTM shown at bottom reads the characters in reverse order, and attempts to find prefix "*un*".

Step 1: Create LSTM models and set GloVe vector as initial hidden state
In this step, LSTM is trained in order to approximate the probability distribution of the next character from a given set of vocabulary. Besides, GloVe vectors are employed as initial hidden state of the LSTM, which is expected to enable the model to learn more smoothly. Also, another LSTM will be introduced later for the purpose of technical problem explained in step 2.

Step 2: Calculate Confidence Fluctuation
Assume that p^i is the probability of the character i, then *Confidence Fluctuation* is defined as:

$$\mathrm{CF}(t) := \mathrm{Var}[\mathbf{p}_t] - \mathrm{Var}[\mathbf{p}_{t-1}]. \tag{11}$$

Where

$$\mathrm{Var}[\mathbf{p}_t] \quad = \quad \frac{1}{|\mathbf{p}_t|} \sum_{i=1}^{|\mathbf{p}_t|} \left(p^i - \bar{p}\right). \tag{12}$$

$$\bar{p} \quad = \quad \frac{1}{|\mathbf{p}_t|} \sum_{i=1}^{|\mathbf{p}_t|} p^i. \tag{13}$$

When $\mathrm{Var}[\mathbf{p}_t]$ is small, it can be interpreted that the model is unsure about the next letter. In other words, we hypothesize that words can be separated at the points where the change in $\mathrm{Var}[\mathbf{p}]$ is high. Therefore, we can call the Variance as *Confidence*. However, there is possibility that $\mathrm{Var}[\mathbf{p}]$ is small at the beginning of the words, because there is not enough information given to predict the next character. This problem makes it difficult to extract prefix, since it appears in the beginning of words. To avoid this problem, we create another LSTM (shown on the bottom of Figure. 2) that reads the character from the end of the words, so that the model reads prefix at the end of inputs.

Step 3: Determine threshold to extract suffix and prefix
The next step is to determine the threshold c so that we can regard the point where $c < \mathrm{CF}(t)$ as splitting point.

3.3 Model 3: Three Net Model

For model 1 and 2, we had similar problem; it could not conduct morphological analysis on the unseen words. This model tries to split words and recombine those sub-words so that the combined vector is close to the pre-trained GloVe vector. By training the network, the model would understand how to break words and it will lead to the discoveries of morphemes. This model's process is presented in Figure 3.

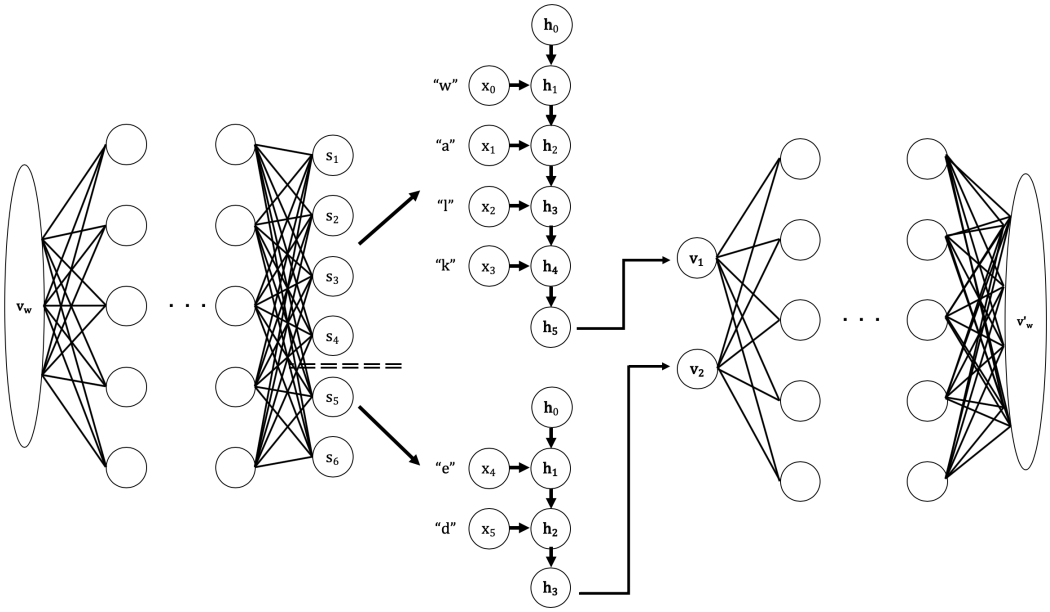

Figure 3: **Illustration of Three-net Model**. On the first NN, pre-trained word embedding is used as an input and separate the word into stem and prefix/suffix. Then, the sub-words are vectorized using char-RNN. Finally, the two vectors are combined using another NN so that the produced vector is close to GloVe.

Step 1: Find splitting points with DNN using GloVe vector as inputs
Firstly, DNN is used to find the splitting points. Here, pre-trained GloVe vector $\boldsymbol{v_w}$ is used as an input, and outputs s_i are used to identify a splitting point of words. Using this splitting point, words are divided into sub-words. When a stem word is used as an input, it should be divided into two parts: the original word and a null word ϵ.

Compare to LSTM, DNN can learn the loss more efficiently because its algorithm is more simple. Therefore, DNN is used to the first net of the model instead of LSTM.

Step 2: character-based LSTM on each sub-sequences due to step 1 slits
Then, the word embedding of sub-words are calculated using character-based LSTM. We assume that the hidden state at the last time step of the LSTM expresses the sub-words' embedding.

Step 3: word reconstruction network to be as close as possible to the original vector of step1
Finally, the sub-words vectors are combined using another DNN, to reproduce the original GloVe vector.

3.4 Model 4: Augmented Attention Model

Because of the fact that the learning of model 3 could be unstable due to its large network size, Augmented Attention Model is proposed.

The Augmented Attention Model is our final and the best performing model showing 91.70% accuracy on test data, outperforming a classic attention mechanism that achieves 82.42%. We will go more in depth in the performance analysis in section 4. This model is inspired by the Attention mechanism, explained in Related Works, that utilizes the whole hidden states of LSTM to extract attention features. Figure 4 shows the differences between our model and a normal attention model. While the normal attention mechanism sees attention weight as subsidiary, our Augmented Attention Model focuses directly on the attention weight itself.

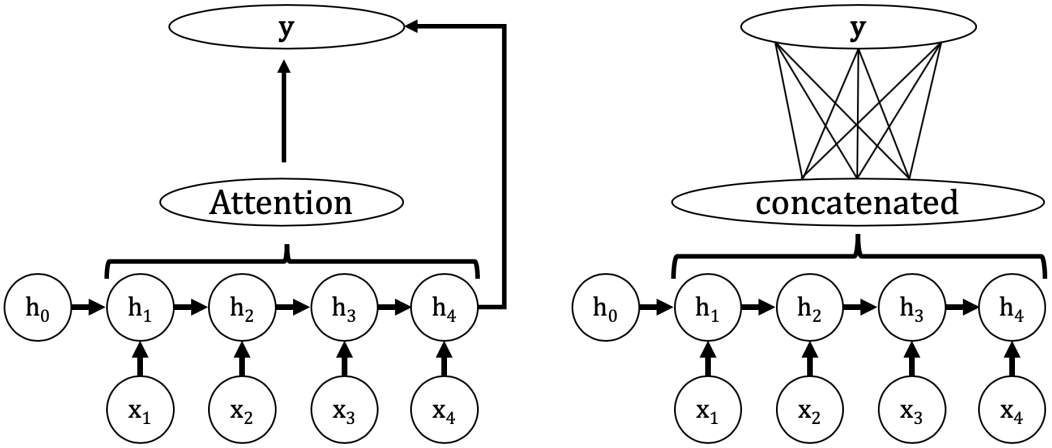

Figure 4: **Comparison between Attention Model and Augmented Attention Model**.

Aside from that, the result from Three-net Model suggests that a loss can not be easily backpropagated through different networks. As it's more efficient to split the models independently, we use model 1's results as the teacher data of model 4. This brings the teacher data closer to the output.

Augmented Attention Model attempts to address the issues we faced so far in the previous models and has successfully cleared these obstacles. The main feature of Augmented Attention Model is that we focus on the concatenated hidden states (= attention weights) to find a result. This leads to extract attention features differently than a normal attention mechanism. This model's mechanism is presented in Figure 5.

This augmented attention model is very similar to a so-called self-attention mechanism presented by Lin et al.(2017)[8]. The main difference relies in the purpose which our model regards the outputs as just the answer of a given problem, while Lin et al.(2017)[8] aims to grasp the features of each inputs. Therefore we don't employ a soft-max function as an activate function.

This is the process step by step of our Augmented Attention Model:

Step 1: Character based LSTM using GloVe vector as initial hidden state
The GloVe vector of the evaluated word is used as the initial hidden state in a character based LSTM. The initial hidden state could also be a vector of zeros but this leads to inferior results as shown in Result section.

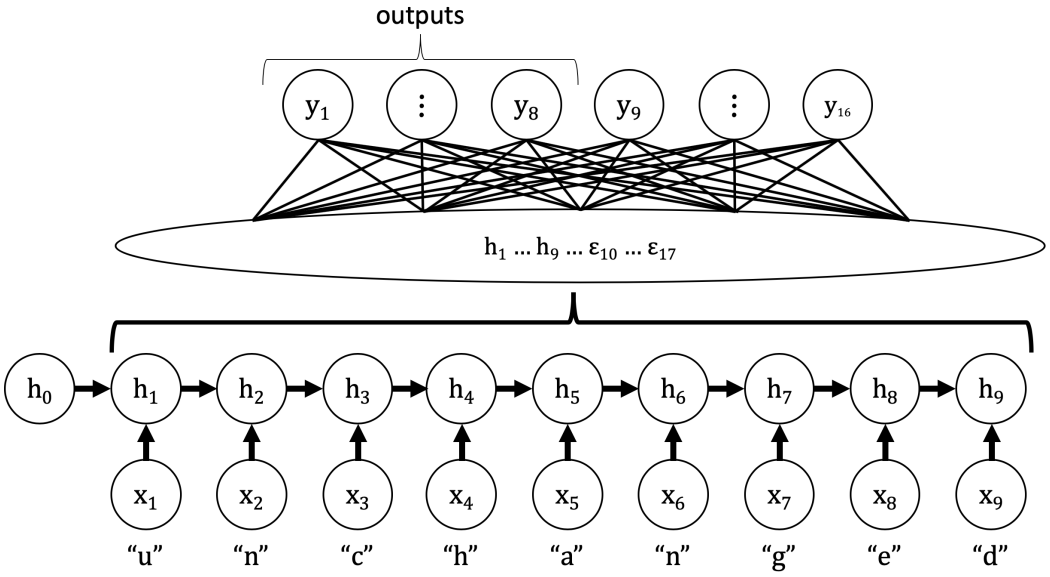

Figure 5: **Visualization of Augmented Attention Model**.

Step 2: Concatenation of the hidden states of the LSTM
We store all the hidden states of the character-based LSTM and then concatenated them in a column vector. Thanks to this step, the model understands the connections between all the hidden states of the LSTM.

Step 3: Use one layer network to find split points
Finally, we use a linear network to output the splitting results (a vector of zeros and ones). The input of this network is the concatenated vector of step 2.

This structure enables the model to extract the features of the LSTM hidden states. These hidden states have information about the context (because of GloVe) and about the order of characters (because of char-LSTM) so we expect that the model will learn to split the words in an efficient way.

As the teacher data, word pairs created in the Word Embedding Space Model (model 1 of this paper) is utilized. We assume that the difference in a stem word and its word family are separation points. For instance, if we have the stem word "*cover*" and its family {*covers, covered, covering, uncovered*}, the separation points used as teacher data will be:

$$\begin{aligned} \text{cover} &\rightarrow [0,0,0,0,1] \\ \text{covers} &\rightarrow [0,0,0,0,1,0] \\ \text{covered} &\rightarrow [0,0,0,0,1,0,0] \\ \text{covering} &\rightarrow [0,0,0,0,1,0,0,0] \\ \text{uncovered} &\rightarrow [0,1,0,0,0,0,1,0,0] \end{aligned}$$

To sum up, our model 4 Augmented Attention Model uses GloVe as inital hidden state of the LSTM. And we will compare this model with variations:

- Attention Model
- Augmented Attention Model (without GloVe)

4 Results & Discussion

In this section we will firstly analyze the performance of each model individually and then make a comparison between the models that have led to significant results.

4.1 Performance analysis

4.1.1 The performance of model1 (Word Embedding Space Model)

The outputs of model 1 are pairs that have a common prefix/suffix rule and a high similarity. And because we use GloVe as a reference for the model, results have been found by our model in an unsupervised way. We can group the pairs and make group of rules around base words which found by the hit-rates. The outputs are similar to the research of Soricut and Och (2015)[13].

As this model can identify the base words, we are able to visualize their word families. Figure 6 shows the mapping for the word *create*, *work* and *violation*.

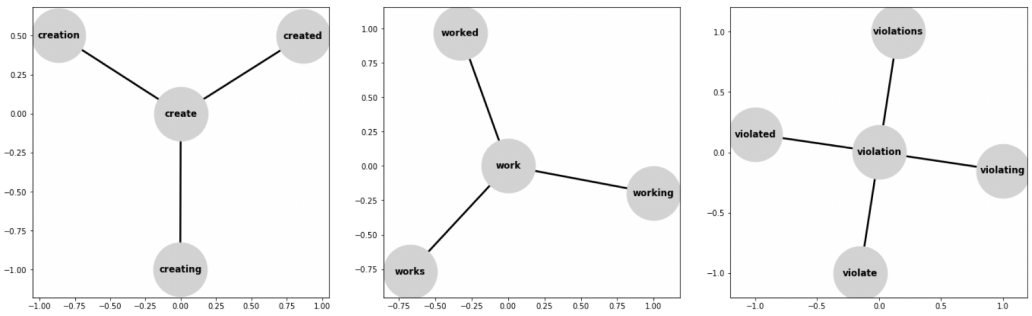

Figure 6: **Example Outputs from the Word Embedding Space Model**.

As already introduced, hit-rate is used to find the base word. For instance, the word *work* has a hit rate of 3. Nevertheless, one limitation is that we are not able to find pairs with low similarity even though they are grammatically in the same word family. For instance, since we set the threshold as 0.7, we are not able to categorize *worker* and *work* in the same group. Indeed, they have a similarity of 0.52674675.

Results are comparable to the paper of Soricut and Och (2015)[13]. The model is useful in finding morphemes in an unsupervised way thus it will be compared with our newly proposed model.

4.1.2 The performance of model2 (Character Prediction Model)

input	Prediction for next characters and its probabilities			Confidence Fluctuation
a	n :0.25881	r : 0.10419	l : 0.08208	5.98E-4
g	n : 0.46837	h : 0.21220	m : 0.05137	10.43E-4
e	s : 0.95386	d : 0.34660	t : 0.00390	194.83E-4
d	end : 0.73587	s : 0.28813	u : 0.00982	-146.78E-4

Table 1: Example output of model 2 on word "aged"

input	Prediction for next characters and its probabilities			Confidence Fluctuation
t	o :0.33372	e : 0.27056	h : 0.16133	1.0E-4
e	r : 0.79984	d : 0.04072	m : 0.04067	143.07E-4
s	t : 0.58885	h : 0.20045	t : 0.11182	-106.35E-4
t	u : 0.81860	r : 0.07511	u : 0.04182	100.48E-4
s	end : 0.59650	u : 0.28697	y : 0.05886	-114.24E-4

Table 2: Example output of model 2 on word "tests"

Figure 7 shows the change in the loss of this model, and it seems that learning is successfully conducted. Table 1 and 2 shows interesting prediction results. Indeed, when we look at the Confidence Fluctuation of the word "aged" (in Table 1), we observe a positive pick after inputting character

input	Prediction for next characters and its probabilities			Confidence Fluctuation
c	o :0.55676	h : 0.29889	e : 0.08934	16.62E-4
r	o : 0.41062	e : 0.30792	a : 0.27208	-13.99E-4
e	a : 0.76334	s : 0.12540	c : 0.03925	99.11E-4
a	s : 0.59073	c : 0.12110	b : 0.0817	-46.6E-4
t	l : 0.43951	u : 0.25349	h : 0.16147	-46.49E-4
i	n : 0.48250	l : 0.38697	e : 0.09153	-6.37E-4
o	n : 0.93703	end : 0.03603	w : 0.02378	200.67E-4
n	end : 0.99958	y : 0.00028	s : 0.00012	46.7E-4

Table 3: Example output of model 2 on word "creation"

"e". This implies that the model is confident about the next letter after receiving "e", thus the model suggests us that we might have a separation point before the character "d".

However, we also observe that the Confidence Fluctuation for the beginnings of words tend to be large and unstable. For instance after inputting the character "e" of the word "tests" (in Table 2). This is due to the fact that the prediction of the next character for the first characters of a word is almost at random. In other words, the prediction confidence increases as the LSTM receives more inputs. But we can process LSTM in both directions so we can avoid this issue.

Moreover, the main problem of this model is that the Confidence Fluctuation can be unstable and it can split the words at wrong points. Figure 3 shows the split result for the word "creation", and the model propose to split the word after the character "e" (this is because many words begin with "crea" like cream for example) and after the character "o" (this is because the training of the model can be improved). This leads us to mistakenly splitting the word as: $crea + tio + n$.

Therefore we will not compare this model with the others. Nevertheless, beside this confidence fluctuation instability the model shows promising results. Moreover, we will use the character-based LSTM method of this model for model 4.

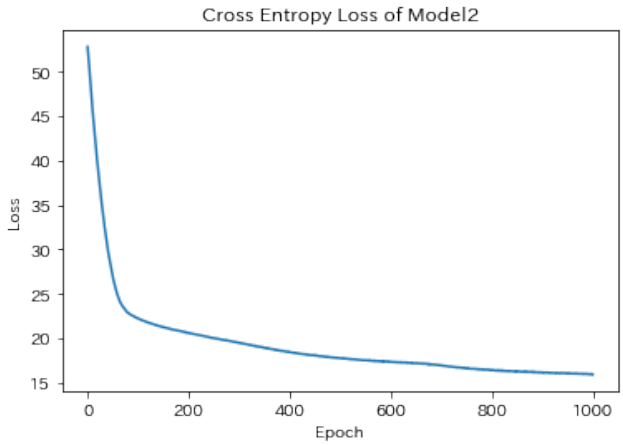

Figure 7: **Loss Graph for Character Prediction Model**.

4.1.3 The performance of Model 3 (Three Net Model)

Even though we have tried a large amount of model tuning and modifications, it was impossible to successfully train the Three Net Model. Although the loss was correctly backpropagated and decreased during training for the third network (Figure 8), the loss is not correctly backpropagated to the upstream NN (Figure 9). In other words, the model does not have enough ability to update weights so that the output is close to the initial GloVe vector.

This backpropagation issue leads to bad splitting results. For instance, the word "$confirmed$" was split as: $confi + rmed$. Solutions to this bad performance model have been found in Model 4. Therefore, we don't compare the model 3 with the others.

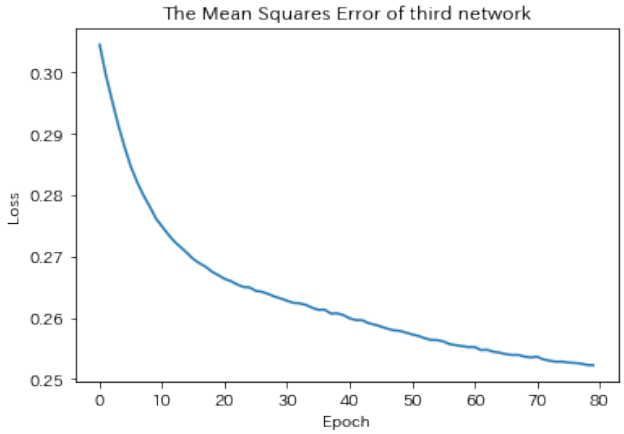

Figure 8: **Illustration of change in loss from the third network of the Three Net Model**.

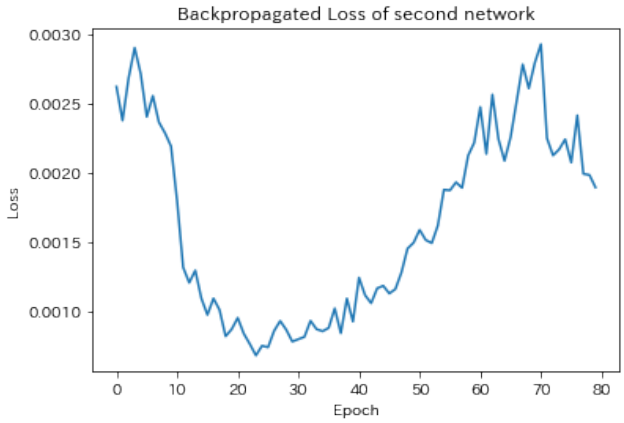

Figure 9: **Illustration of change in loss from the second network of the Three Net Model**.

4.1.4 Model 4 performance (Augmented Attention Model)

Fourthly, for our last models, we successfully trained them and we found some interesting outputs.

The change in loss of the Augmented Attention Model (with a GloVe vector as the initial state of the LSTM) shows success in training (Figure 10). The training accuracy achieves 100.00 % and the accuracy on the training test achieved 91.00 %.

We can compare those results with an alternative of our model 4 which is an Augmented Attention Model without a GloVe vector as initial state of the LSTM (see Figure 11 for graph loss). We observed a decrease in accuracy during the training (86.13 %) and a decrease of the test accuracy with 81.15 %, therefore we know that GloVe helps in this splitting task. From a morphological perspective, this makes sense because GloVe has information about grammar (e.g. tense, plural). Indeed, model 1 has already demonstrated this statement.

Table 4: The performances of model4

	On the training data	on the test data
Normal Attention	99.8	85.5
Augmented Attention with GloVe	**100.0**	**91.0**
Augmented Attention without GloVe	86.1	81.2

Finally, we have compared our Augmented Attention Model (the one with GloVe vector as initial state) with a model that uses a normal attention mechanism. Figure 12 shows the decrease in loss

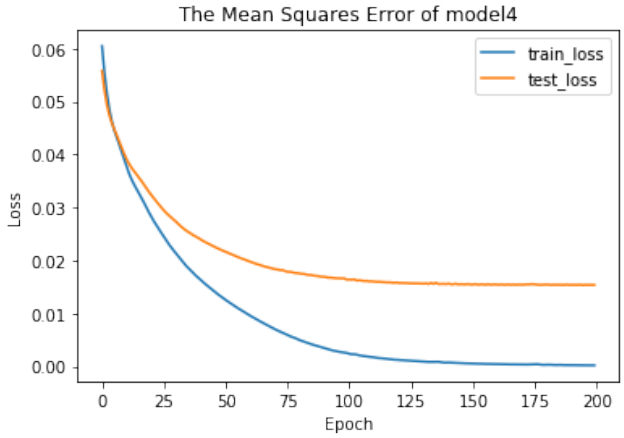

Figure 10: **Illustration of change in loss from the Augmented Attention Model**.

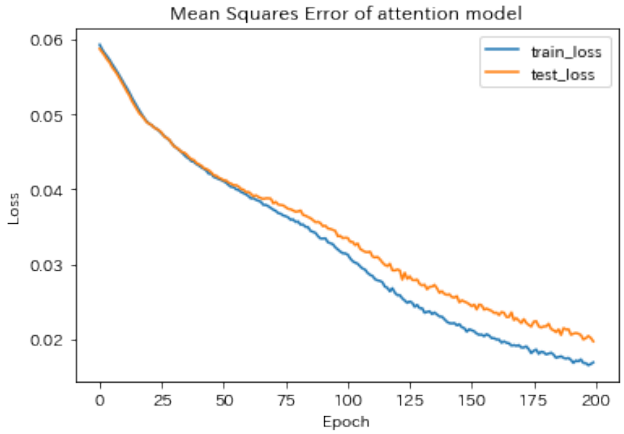

Figure 11: **Illustration of change in loss from the Augmented Attention Model without GloVe**.

during the training of this model. The accuracy of the training set is 99.80 % and the accuracy of the test set is 85.51 %. We again can conclude that our Augmented Attention Model accuracy outperform this normal attention model that as 85.51 % accuracy. In the Models section (section 3.4) we have highlighted some reasons for this.

Results for our Augmented Attention Model are significant, and showed high performance. This model is useful in finding morphemes in an unsupervised way. We will compare this model 4 with the Word Embedding Space Model (model 1) in the comparison section.

4.2 Comparison (Word Embedding Space Model and Augmented Attention Model)

In this subsections we will compare the two most promising models presented in this paper: the Word Embedding Space Model (model 1) with the Augmented Attention Model (model 4).

Due to the fact that the types of output for Word Embedding Space Model and Augmented Attention Model are different, it is difficult to quantitatively compare the results. While Word Embedding Space Model produces word pairs as model's output, Augmented Attention Model separates words into a stem and prefix/suffix. Therefore, the analysis in this section will be conducted qualitatively.

While 88.92% of the two models proposed the same results using the test, table 5 shows examples where splitting points are different.

As we mentioned above, it is safe to say that Augmented Attention Model is more applicable in which it can deal with unseen words. In addition to that, Augmented Attention Model can ignore the

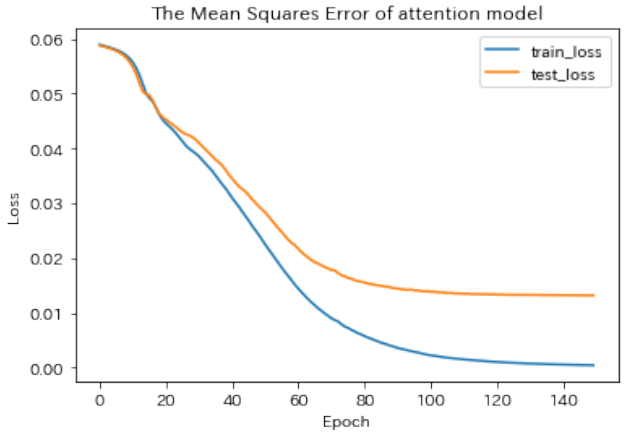

Figure 12: **Illustration of change in loss from the Attention Model**.

Table 5: Examples of different splitting between model 1 and model 4

	Model 1	Model 4
had	h + ad	ha + d
code	cod + e	co + de
pressures	pressure + s	pressu + res
suite	suit + e	sui + te
selection	selec + tion	selectio + n
varying	var + ying	vary+ ing
confrontation	confrontatio + n	confrontat+ ion
island	islan + d	isl + and
terminate	terminat+e	termina+te
pass	pas + s	pa + ss
estimated	estimate + d	estimat + ed
gives	giv + es	give + s
deeper	deep + er	deeper +
faces	fac + es	face + s
leave	leav + e	leave +
proposes	propos + es	propose + s

"noise" such as irregular conjugations. For instance, while model 1 splits the word 'varying' into 'var' + 'ying', model 4 splits it into 'vary' + 'ing'. Since languages are made in natural ways, no one can say which splitting point is correct without morphological rules made by humans. Yet, the example above shows that model 4 is obviously based on our intuition. The main differences are summarized on the table6

Table 6: The main differences between model 1 and model 4

	Model 1	Model 4
On the training data	Relatively Robust	Able to follow model 1 completely
On the test data	Not able to deal with	Able to predict based on its experience
Another advantage	Stable	Having some possibility to outperform model 1
Another disadvantage	Dependent upon cosine similarity	Vulnerability of neural network (overfitting issue)

5 Conclusion

To conclude, all over this research paper we have highlighted our main findings around unsupervised morphology analysis through several exploration models.

The Augmented Attention Model presented in this paper outperforms existing models in some areas. This model applies the attention mechanism in a slightly different way. The hidden states of the LSTM are fully used to split words and to find morphemes and performs an outstanding result. It overcomes the problems of unstable threshold and difficulty in backpropagation of large networks that have been observed in exploration models. Our model proves that by applying simplified attention LSTM to word splitting problem, significant progress can be made.

Moreover, the Character Prediction Model has shown some interesting results and sounds promising for future research. Indeed, the concept of this model is less complex compared to other models but still performs good results.

6 Further research

The aim of this paper was to compare unsupervised morphology discovering models by combining several existing ideas together. We mainly have used LSTM to discover new approaches to find morphological rules.

For further research on the topic, several additional improvements not mentioned in this paper could be explored to improve even more the models we have described.

Firstly, many researches have been done on external memory components to improve network performance, especially on large data sets (Weston, Chopra & Bordes, 2014)[14]. Additionally, memory networks can understand better the long-term dependencies (Gulcehre, Chandar & Bengio, 2017)[7]. Memory augmented networks could also help solving the data size issue (Grave, Joulin & Usunier, 2017)[5]. Moreover, we all know that the amount of data we are going to produce in the coming years is going to drastically increase (Forbes, 2017). To handle all these new linguistic data sources our models could be improved with memory concepts.

Secondly, GAN models could also be explored to increase performance in unsupervised linguistics features discovering. On one hand the first part of the model generates pairs and on the other hand the second part discriminates the pairs and decide to keep or not the morphological rule (Goodfellow, et al., 2014)[4]. We could train this kind of model with the results of the models we have presented in this paper.

Thirdly, our models are not able to investigate *Scriptio continua* languages (i.e. languages that are not separated by spaces), such as Chinese and Japanese. To tackle this issue, we can separate words by morphological analysis tools such as Mecab[2] in Japanese.

Finally, because morphology is closely linked to relationship between characters and words, we believe that the concept of order is important to solve morphological analysis problems. Therefore, memory models seem to be the most promising improvement for further research. In addition, improved attention mechanisms seem to be working well in complementary with memory.

[2]http://taku910.github.io/mecab/

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[3] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016.

[4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[5] Edouard Grave, Armand Joulin, and Nicolas Usunier. Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*, 2016.

[6] Alex Graves, Abdel-rahman Mohamed, and Geoffrey 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.

[7] Caglar Gulcehre, Sarath Chandar, and Yoshua Bengio. Memory augmented neural networks with wormhole connections. *arXiv preprint arXiv:1701.08718*, 2017.

[8] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017.

[9] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[10] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

[11] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[12] Jenny R Saffran, Richard N Aslin, and Elissa L Newport. Statistical learning by 8-month-old infants. *Science*, 274(5294):1926–1928, 1996.

[13] Radu Soricut and Franz Och. Unsupervised morphology induction using word embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1627–1637, 2015.

[14] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *CoRR*, abs/1410.3916, 2014.