Neural Network Joint Language Model: An Investigation and An Extension With Global Source Context

Yuhao Zhang
Computer Science Department
Stanford University
zyh@stanford.edu

Charles Ruizhongtai Qi
Department of Electrical Engineering
Stanford University
rqi@stanford.edu

Abstract

Recent work has shown success in using a neural network joint language model that jointly model target language and its aligned source language to improve machine translation performance. In this project we first investigate a state-of-the-art joint language model by studying architectural and parametric factors through experiments and visualizations. We then propose an extension to this model that incorporates global source context information. Experiments show that the best extension setting achieves 1.9% reduction of test set perplexity on a French-English data set. [1]

1 Introduction

The construction of language model has always been an important topic in NLP. Recently, language models trained by neural networks (NNLM) have achieved state-of-the-art performance in a series of tasks like sentiment analysis and machine translation. The key idea of NNLMs is to learn distributive representation of words (aka. word embeddings) and use neural network as a smooth prediction function. In a specific application like translation, we can build a stronger NNLM by incorporating information from source sentences. A recent work from ACL 2014 (Devlin et al., 2014) achieved a 6+ BLEU score boost by using both target words and source words to train a neural network joint model (NNJM).

In this project, we implement the original NNJM and design experiments to understand the model's strengths and weaknesses as well as how hyper parameters affect performance and why they affect performance in specific ways. While the original paper on NNJM focuses on presenting the model and its performance gains, our project focuses on gaining a deep and well-rounded understanding of the model.

As an important part of the work, we also extend the current NNJM with global context of source sentences, based on the intuition that long range dependency in source language is also an important information source for modelling target language. Besides target words and source words, we compute sentence vectors from source sentences in various ways and incorporate the sentence vectors as an extra input into neural networks.

Our contribution mainly lies in three aspects: First, we present a deep dive into a state-of-the-art joint language model, and discuss the factors that influence the model with experimental results; second, we propose a new approach that incorporates global source sentence information into the original model, and present our experimental results on a French-English parallel dataset; third, as a side contribution, we have open-sourced[2] our implementation of both the two models, which could be run on both CPU and GPU with no additional effort.

The rest of this report is organized as follows. We first give a brief introduction on NNJM in Section 2. Then in Section 3 we present our extensions: We introduce how we compute source sentence vectors and why we make these design choices. We then spend more space present our insights on NNJM gained from experiments, and evaluation of our extended NNJM model in Section 5. We summarize related work in Section 6 and explore future directions for extending our

[1] This project is advised by Thang Luong and it is a solo CS229 co-project for one of the authors.

[2] http://goo.gl/WizzCF

current work in Section 7.

2 Neural Network Joint Model

Language model, in its essence, is assigning probability to a sequence of words. For machine translation application, language model is evaluating translated target sentence in terms of how likely or reasonable it is as a sentence in target language. The intuition for a joint language model is to utilize source sentence information to help increase quality of the target language model. Note that it is a privilege for machine translation task since there is always a source sentence available. BBN paper has shown and the NNJM we implemented has also shown that by utilizing source language information, a very significant quality improvement of target language model can be achieved.

In terms of how to make use of the extra information of source sentence, an effective approach proposed in the BBN paper is to extend normal NNLMs by concatenating a context window of source words with target *n*-gram as the input to the model and train word representations (or embeddings) for both source and target languages. In Section 3 we will also describe another extension of NNLM of using source sentence vector as the extra source of information.

2.1 Model Description

We use a similar model as the original neural network joint model. To be concrete, we provide mathematical formulation for the model together with a model illustration in Figure 1. For more details please refer to the original BBN paper.

One sample input to the model is a concatenated list of words composed of both target context words (n-1 history words for n-gram) T_i and source context words S_i. Source words are selected by looking at which source word target word t_i is aligned with, say it's s_{a_i}, then we take a context window of source words surrounding this aligned source word. When the window width is $\frac{m-1}{2}$, we have m source words in the input.

$$p(t_i \mid T_i, S_i)$$

$$T_i = t_{i-1}, ..., t_{i-n+1}$$

$$S_i = s_{a_i - \frac{m-1}{2}}, ..., s_{a_i}, ..., s_{a_i + \frac{m-1}{2}}$$

Here we regard t_i as output, i.e. $y \in \mathbf{R}$ as one of the target words, and concatenation of T_i and S_i as input, i.e. $x \in \mathbf{R}^{n+m-1}$ of $n-1$ target words and m source words. The mathematical relation between input and output is as follows, where $\Theta = \{L, W, b^{(1)}, U, b^{(2)}\}$. Linear embedding layer $L \in \mathbf{R}^{d \times (V_{src} + V_{tgt})}$ which converts words to word vectors by lookup, where d is word vector dimension. In hidden layer, $W \in \mathbf{R}^{h \times (d*(n+m-1))}$, $b^{(1)} \in \mathbf{R}^h$. In softmax layer, $U \in \mathbf{R}^{V_{tgt} \times h}$, $b^{(2)} \in \mathbf{R}^{V_{tgt}}$ and

$$g_i(v) = \frac{\exp(v_i)}{\sum_{k=1}^{V_{tgt}} \exp(v_k)}$$

$$p(y = i \mid x; \Theta) = g_i(Uf(WL(x) + b^{(1)}) + b^{(2)})$$

Optimization objective is to maximize the log-likelihood of the model.

$$\ell(\Theta) = \sum_{i=1}^{m} log(p(y^{(i)} \mid x^{(i)}; \Theta))$$

2.2 Evaluation Metric

We use *perplexity* as the metric to evaluate quality of a language model.

$$PP(W) = p(w_1, w_2, ..., w_N)^{\frac{-1}{N}}$$

3 Neural Network Joint Model with Global Source Context

An *n*-gram language model is based on Markov assumption and sacrifices long-term dependencies. The NNJM studied in the previous section suffers from a similar problem: When utilizing the source sentence information, the model only incorporates source words in a small window range around the aligned source word, thus the long-term dependencies in the source language is missing. In this section, we show our attempts in pushing the state-of-the-art of NNJM by utilizing global source context (global source sentence information). For simplicity, we will use NNJM-Global to refer to this extension in the following sections.

Intuitively, the optimal approach for incorporating the long-term dependencies in the source sentence is to exploiting the dependency information, by utilizing the results of dependency

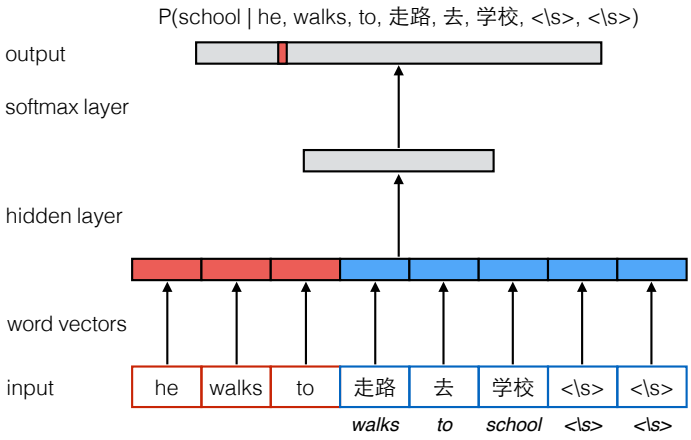

Figure 1: Neural network joint model with an example (illustrated with Chinese-English) where we use 4-gram target words (3 words history) and source context window size of 2. We want to predict the next word following *he, walks, to* and hopefully estimated probability of the next word being *school* would be high.

parsing of the source sentence. However, this direct approach requires a parsing phase which is both language-dependent and time-consuming. As a result, it is difficult to scale to corpus that is large in size and consists of various languages. Thus, instead we attempt to explore methods that are both language-independent and time-efficient in this project.

3.1 Weighted Sum Source Sentence

Our first attempt is to include sentence vector directly into the input layer of the neural network. However, since the source sentence vectors are various in length, we need a way to adapt the global input vector into having uniform length. Thus, we calculate the weighted sum of word vectors in the source sentence, and feed the result into our input layer, as shown in Figure 2.

There are various ways to determine the weights used for different source words. Specifically, we experimented with two different approaches:

1. **Uniform weights** We assign each word a uniform weight in the source sentence. In another word, we take the mean of all the word vectors to form the global context vector.

2. **Zero weights for stop words** Instead of giving all words the same weight, we identify top N frequent words in the vocabulary as stop words, and assign each of them with a zero weight. For all the rest words in the vocabulary, we still assign them with a uniform weight. The intuition is that stop words are over-frequent in the corpus, and instead of providing useful information, they may bring a lot of noise to the global context vector when we compress them together with other less frequent words.

3.2 Splitting Source Sentence Into Sections

The previous approach of taking the weighted sum of the whole sentence vector suffers from a problem: Compressing a whole sentence vector into a single word vector length may cause a non-trivial information loss. In order to solve this problem and in the mean time does not slow down the model training significantly, we experimented with an approach where we split the source sentence into sections before taking the weighted sum and feeding the results into the next layer, as shown in Figure 3. We treat the number of sections as a hyper-parameter for this model.

Specifically, we experimented with two variants of this approach:

1. **Fixed section length splitting** The sentence vector is first extended with end-of-sentence tokens so that all the input source sentences are of the same length. Then the splitting is done on the populated source sentences. For instance, if we extend all the sentence to a

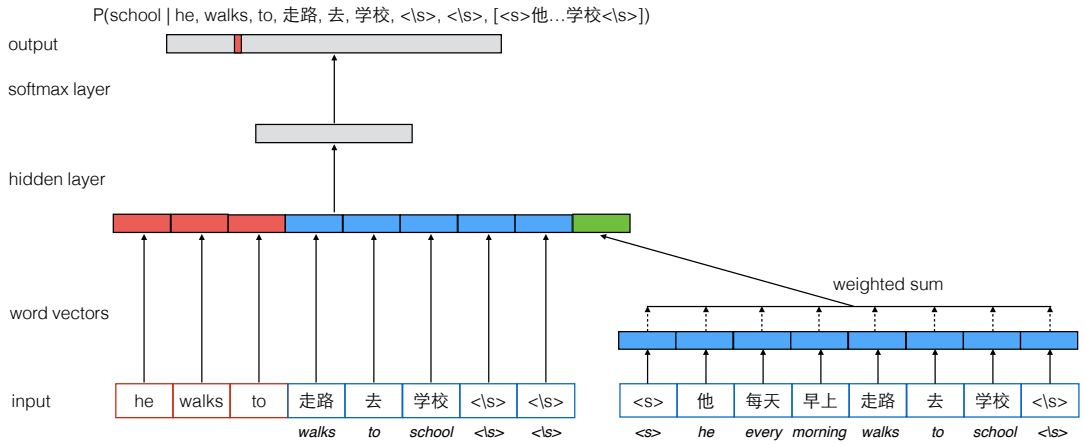

Figure 2: An example for the NNJM with global context, where an additional source sentence is fed into the model, while the source window and the target n-gram remains the same for the input. The linear layer first takes all the word embedding vectors (blue) in the source sentence, and calculate a weighted sum of the these vectors to form the global context vector (green). It is then concatenated with the original input layer and fed into hidden layer.

length of 100 and we split the sentence and get 10 global context vectors, each section will have a fixed length of 10. This approach is computationally more efficient since it can be easily vectorized.

2. **Adaptive section length splitting** We use the original source sentence instead of extending all sentence vectors into a uniform length. Thus, each section will have a variable length dependent on the length of the entire sentence. This approach is difficult to vectorized for efficient GPU computation, but we expect it to give us a performance boost over the fixed section length approach.

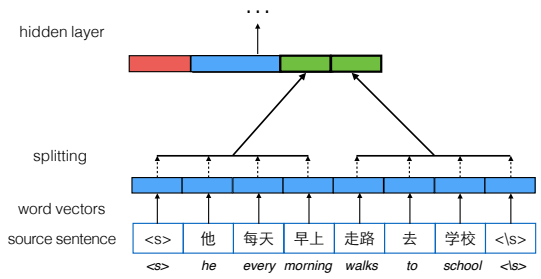

Figure 3: An example of splitting source sentence into 2 sections before calculating the global context vectors. Weighted sum is calculated on the first half of the sentence to form the first global context vector, and then on the second half.

3.3 Global-only Non-linear Layer

Different dimensions of the global context vector and different sections in the source sentences are independent before the global context vector is fed into the neural network in the previous approaches. We add non-linearity to the model by adding another global-only non-linear layer between the global linear layer and the downstream hidden layer, as it is illustrated in Figure 4.

Note that this non-linear layer is only added for the global part of the model, and has no effect on the local part. We use the same non-linear function for this layer as in other layers of the model.

3.4 Bootstrapping NNJM-Global with Pre-trained NNJM Parameters

The previous methods train the word embedding vectors and all other parameters in the neural network together. An natural extension to this is to first train the NNJM, and then use the pre-trained model parameters to bootstrap the NNJM-Global model on the same dataset. Since NNJM and NNJM-Global only differs in the global sentence vector part and share architecture for the rest of the neural network, this pre-training process might be helpful.

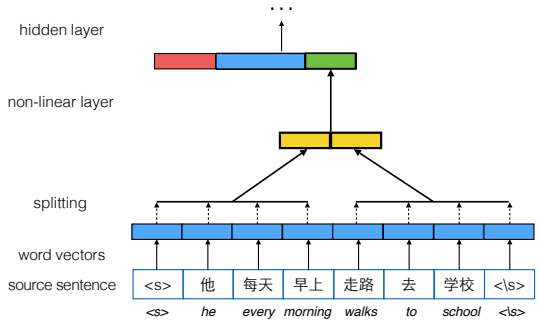

Figure 4: An example for the non-linearity on the global source sentence. A weighted sum is calculated to form the intermediate global context vectors (yellow), and then these intermediate vectors are fed into a global-only non-linear layer.

4 Model Training

Following a similar strategy with BBN paper in training the neural network, we use mini-batch gradient descent to maximize the log-likelihood on training set. Each batch contains 128 input samples, each of which is a sequence of target words plus source context words. There are around 22K mini-batches per epoch. Model parameters are randomly initialized in the range of [-0.05, 0.05].

For hyper parameter tuning in NNJM model, training runs for 5 epochs if not noted otherwise. To evaluate the NNJM-Global model and compare its different variants, instead of using a maximum epoch number to limit the training time, we use convergence check with the goal to exploit the power of each model. Specifically, we check for convergence after each epoch, and if in 5 consecutive epochs the model achieves the same validation set perplexity, we identify the learning process as converged and stop the learning process. We then use the same parameters at the best validation perplexity to evaluate the test set perplexity.

Instead of adding regularization terms, we use the early stopping technique to pick the model with least validation set perplexity. At the end of every epoch we do a validation set test and see if the validation set perplexity becomes worse from last time, if it is worse we halve the learning rate.

The data set we use is from European Parallel Corpus. Our training set contains 100,000 pairs of parallel French-English sentences. Validation and test set each contains 1000 pairs of French-English sentences. For analyzing the NNJM model, we use all the 100,000 pairs of sentences. However, since the implementation of NNJM-Global model contains code that is hard to vectorize, and we need to run for more epochs to exploit the power of each model on the training data, the training of NNJM-Global models takes relatively longer time to finish. Due to time limit, we use a subset (1/4) of the full corpus, which contains 25,000 sentence pairs of parallel French-English sentences, to evaluate each variant of NNJM-Global, and compare the result with NNJM trained under the same settings.

Both training and testing are implemented using Python. We use Theano Library for neural network modeling. The training process is run on a single GPU on Stanford rye machine. Training speed is around 1,500 samples/second and training on one epoch of data (128*22K) takes around half an hour. For reference, total training time for a basic NNJM model over the entire corpus is thus around 2.5 hours when the full GPU power is utilized.

5 Experimental Results

5.1 NNJM

In this subsection, we focus on showing our understanding of the joint language model. Evaluation results will be combined with NNJM+Global model in Subsection 5.2.

5.1.1 Effects of Hyperarameters

In this part, we study how model hyper parameters affect system performance and show insight on our understanding of why they affect performance in specific ways. Among all hyper parameters, word vector dimension, source window size, target n-gram size are specific to our language model while network architecture (hidden layer size and number of hidden layers) and learning rate, number of epochs are general for neural network training. By examining effects of those hyper parameters we expect to get a better understanding of both NNJM and neural network training.

Tuning of hyper parameters is done on the full 100K training set as described above unless

noted otherwise. Since a full grid search is too time consuming we will start from a default hyper parameter setting and change one of them each time. In default setting, learning rate is 0.3, target *n*-gram size is 5 (4 history words), source window width is 5 (thus $2 * 5 + 1 = 11$ source words), vocab size is 20K for both target and source language, epoch number is 5 (though the model may not fully converge in just 5 epochs, it's enough to show the general trends of hyper parameter's influence), word vector size is 96 and there is one hidden layer of 128 units.

Word Vector Dimension

Generally, it helps to increase word vector dimensions. As shown in Figure 5, as we have larger word vector sizes, validation perplexity decreases monotonically. The disadvantage of large word vector size is more training time and more evaluation cost.

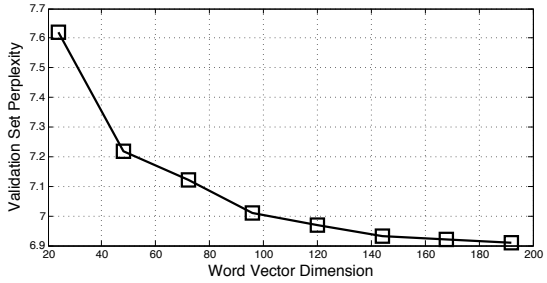

Figure 5: Effect of Word Vector Dimension

Source Window Width

While having no source window degrades the NNJM to NNLM, having a very small source window (say include only one source word) can greatly boost performance. From Figure 6 we can see for our data set and model, source window width 3 ($3 * 2 + 1 = 7$ source words) achieves the best validation set perplexity in 5 epochs. Possible explanation is that source words distant from the aligned one add less information to predicting target word and it also takes more epochs to converge.

Target N-gram Size

As we can see in Figure 7, the general trend is that as we increase target *n*-gram size perplexity drops, yet after some turning point perplexity stays roughly stable. Since larger *n*-gram size increases model complexity, we'd conclude that *n*-gram size of 4 is good for our case.

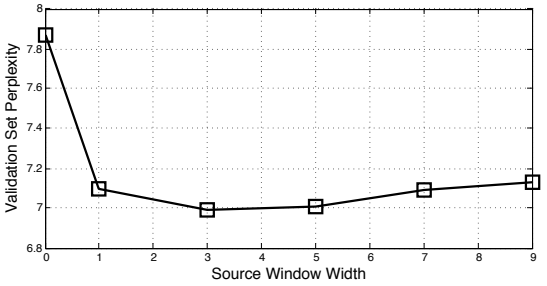

Figure 6: Effect of Source Window Width

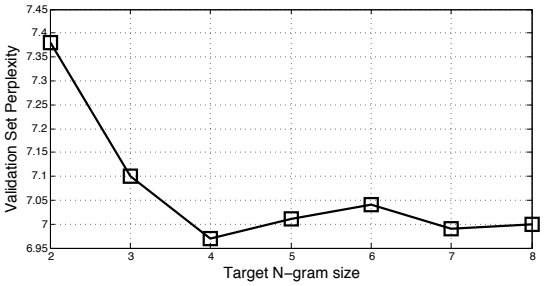

Figure 7: Effect of Target N-gram Size

Hidden Layer Size

The effect of hidden layer size is similar to word vector dimension, as seen in Figure 8, as we have larger hidden layers the perplexity drops monotonically. Although extremely large hidden layer may overfit the training set, we do not observe such situation for hidden layer sizes we have tried. Therefore, we can choose hidden layer size of 256 for higher performance.

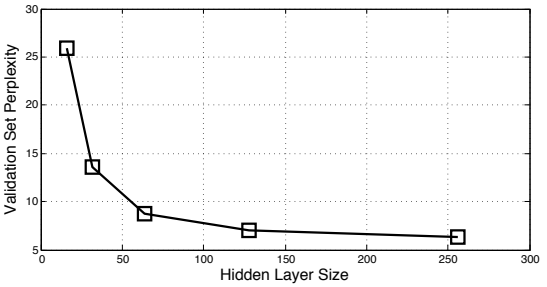

Figure 8: Effect of Hidden Layer Size

Number of Epochs

An epoch of training means going through the entire training set once in mini-batch gradient descent training. Strictly speaking, it does not belong to hyper parameters since, in theory, we can

always train the model until convergence. However for real-world case, it might take too long to reach convergence and we may want to get a sense of how fast the model converges and how number of epochs affect model quality so that we can make informed decision to stop training earlier than convergence. In Figure 9, we can see that our default model converges in around 25 epochs. Since 5 to 10 epochs, the decrease of perplexity becomes quite slow, thus it's applicable to train for 5 to 10 epochs to get a decent result.

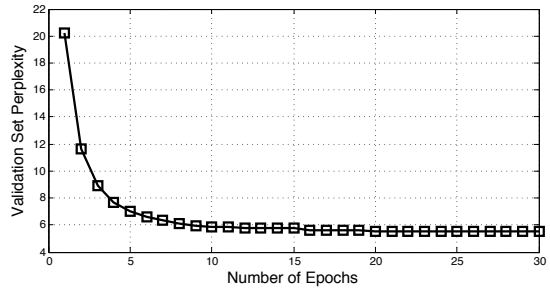

Figure 9: Effect of Number of Epochs

Learning Rate

While very large learning rate such as 1.0 and 3.0 leads to quick convergence yet unsatisfactory local minimums (the loss stabilized at around 2 while for lr=0.3, though not shown in the figure, can reach around 1.5), very small learning rate such as 0.03 converges too slow. Therefore, we think learning rate around 0.3 with balance of convergence speed and training quality. Note that in our training method, we will halve the learning rate at the end of a epoch if necessary. Here the validation set loss is negative log likelihood, which is what we want to minimize. Due to time limit, experiment for this part is using a 25K training set.

Multiple Hidden Layers

We have tried to extend the single hidden layer NNJM to multiple hidden layers. Using two hidden layers with 128 units in each of them achieves a boost in performance yet longer training time (we train until convergence for this case). Using three hidden layer with 128 units each takes too long to converge and tends to overfit the training set - we observe training set loss is much less than validation set and while training loss keeps decreasing, validation set perplexity stays the same.

Activation Function

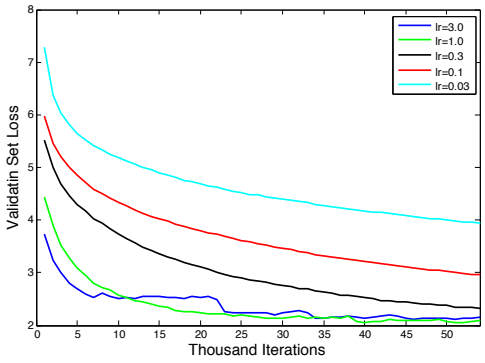

Figure 10: Effect of Learning Rate

l	1	2
Perplexity	8.05	6.98

Table 1: Effect of Hidden Layer Number

Table 2 shows that rectifier activation function achieves better performance. Leaky rectifier's performance is similar to rectify.

$$\text{rect}(x) = x\mathbb{1}[x > 0]$$

$$\text{leaky-rect}(x) = x\mathbb{1}[x > 0] + 0.01x\mathbb{1}[x < 0]$$

	tanh	rectify	leaky rectify
Perplexity	8.05	7.35	7.35

Table 2: Effect of Activation Function

5.1.2 Visualizations and Insights

In this subsection we use network parameter visualization to show how the neural network take advantage of source context. Specifically, we will look at the linear transformation matrix W in the hidden layer, which can be thought as a way to measure how much certain part of input contribute to predicting the next word.

In Figure 11 we see that regions corresponding to certain word positions have stronger intensity. By averaging the absolute values of the weights in each region of dimension of word vector size by hidden layer units number, we get results in Figure 12. It's clear that the source word in the middle (word index number 6), i.e. the one aligned with the next target word, contributes most to predicting the next word. There is a quick trend of attenuating importance for source words far from the middle one. We can also observe that the second

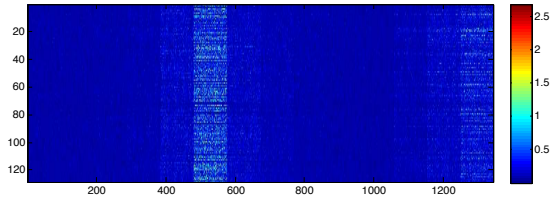

Figure 11: Heat map of absolute element values of hidden layer matrix W. Input dimension is of $92 * 14 = 1344$ where 96 is word vector dimension and for n-gram size of 4 and source context window width 5, there are 14 words in involved in each input sample. Output put dimension is 128.

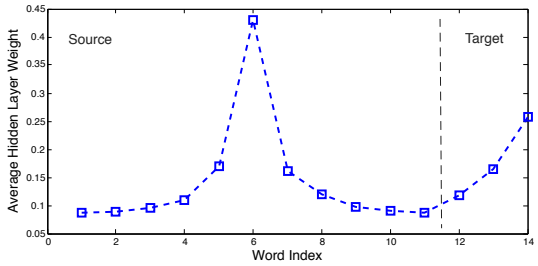

Figure 12: Average of absolute values of hidden layer matrix W elements corresponding to each of the 14 words. Left 11 words $(2 * 5 + 1)$ are from source window whose center is the source word aligned with the next target word. Right 3 words are the history words for target n-gram.

last target word (word index 14) in the n-gram (the next target word/to-be-predicted one is the last) contribute a lot for the prediction though with a less weight than the middle souce word.

5.2 NNJM-Global

In this subsection we demonstrate experimental results for each variant of the NNJM-Global model, and compare their results with the vanilla NNJM model. Note that all the models in this part are trained with the same strategy described in previous section. By default, we use a vocabulary size of 10000, a source window size of 3, a target n-gram size of 4, an embedding dimension of 96, a hidden layer size of 128, and a learning rate of 0.3 to train the models.

5.2.1 Comparing NNJM-Global with NNJM

The resulting perplexity achieved by different models on the test set is shown in Table 3. Note that we also include the result for a basic neural network language model (NNLM) where only target words are utilized for making predictions, to demonstrate the effect of global source context information.

Model	SrcWin	Perplexity
NNLM	-	95.06
NNLM-Global	-	94.73
NNJM	7	10.09
NNJM-Global	7	10.05
NNJM	5	9.89
NNJM-Global	5	9.71
NNJM	3	9.51
NNJM-Global	3	9.45
NNJM-Global + SW-10	3	**9.44**
NNJM-Global + SW-25	3	**9.44**

Table 3: Test set perplexity for different models. *SrcWin* represents the source window size that is used in the model. *SW-N* represents that *N* most frequent stop words are removed from the global sentence vector. Results for the NNLM model where only target words are used for prediction are also included.

It is easily observed that for each setting of source window size, the NNJM-Global model achieves smaller (better) test set complexity compared to its corresponding NNJM model. For the settings shown in the table, the best performance is achieved when the source window size is set ot be 3. Under this setting, a marginally better result is achieved when we use a zero-weights-for-stop-words weighted sum strategy. There is no noticeable difference between the different settings of number of stop words in the NNJM-Global model.

5.2.2 Effect of Splitting Source Sentence

Both the two approaches for splitting the global source sentence vectors are evaluated and compared to the basic NNJM and NNJM-Global models. The results are shown in Table 4.

The fixed section length splitting strategy with section number of 2 gives reduction of the test set perplexity when compared to the basic NNJM-Global model, while the adaptive section

Model	NumSec	Perplexity
NNJM	-	9.51
NNJM-Global	1	9.45
NNJM-Global + FixSplit	4	9.54
NNJM-Global + FixSplit	2	**9.38**
NNJM-Global + AdaSplit	2	9.46

Table 4: Test set perplexity for models with different global context vector section numbers. *NumSec* represents the section number in the resulting global context vector. We use *FixSplit* to denote the model where the fixed section length splitting method is used; we use *AdaSplit* to denote the model where the adaptive section length splitting method is used. All models included in this table use a source window size of 3.

length splitting strategy gives almost the same result as the basic NNJM-Global model, and also achieves better result compared to the original NNJM model. The performance is observed to deteriorate when the section number increases.

5.2.3 Effect of Global-only Non-linear Layer

Generally, adding a non-linear layer could add expression power to the neural network. We evaluate different architectures for adding the global-only non-linear layer in the NNJM-Global model and demonstrate the result in Table 5. Specifically, we compare adding the non-linear layer to the basic NNJM-Global model, and to the NNJM-Global model with the two splitting strategies. We also evaluate the effect of different non-linear layer sizes. For better interpreting the architecture, we use non-liear layer sizes that are integral multiple of the word embedding size.

One observation is that the effect of global-only non-linear layer depends on the size of it and the architecture of the rest part of the model. In most cases adding the non-linear layer size would boost the performance, but the scale of this performance boost depends on the architecture of the model. The best test set perplexity is observed when a non-linear layer with the size of double the word embedding vector is added to the model where the global source context vector is splitted into two sections. This best perplexity is 1.9% lower than the basic NNJM model. One possible explanation for this is that while the fixed section size splitting approach allows more global context information, the non-linear layer adds a non-linear combination of this global information, and without compromising the dimension used to express this information. The model gains additional expressive power from this combination of architecture settings.

6 Related Work

In ACL 2014, BBN published a paper on neural network join model for statistical machine translation (Devlin et al., 2014), which is based on neural network language model (Bengio et al., 2003), and uses source language information to augment target language model. In this project, instead of focusing on efficiency and MT result presentation, we investigate deep into the original NNJM by study on hyper parameters and visualization of hidden layer weights. We also extend the model with global source context and achieves improvement in terms of perplexity scores.

In another work published in ACL 2012, sentence vector generated by weighted average of source words is used for learning word embeddings with multiple representations per word (Huang et al., 2012). Our project have taken similar strategy in generating sentence vector but have also developed more complex models. Besides, while their work forcus on representation learning, we forcus on designing good architecture to improve joint language model quality.

7 Discussion and Future Work

Reflecting on the limited power of source sentence vector on improving language model quality, we have the following insights. Firstly, we think sentence vector quality is restricted by the model generating it. While a simple average of sentence words' embeddings capture little about global context, architecture with non-linear layers can be more powerful. Secondly, since a single sentence vector is a highly compressed version of the original sentence of dozens of words, it may be more helpful on tasks relying on global context such as sentiment analysis and text classification and do less benefit to local tasks such as word prediction.

We have several ideas on future directions to

Model	NumSec	NonLinearSize	Perplexity
NNJM	-	-	9.51
NNJM-Global	1	-	9.45
NNJM-Global + NL	1	96 (1×)	9.45
NNJM-Global + NL	1	192 (2×)	9.45
NNJM-Global + FixSplit	2	-	9.38
NNJM-Global + FixSplit + NL	2	96 (1×)	9.61
NNJM-Global + FixSplit + NL	2	192 (2×)	**9.33**
NNJM-Global + AdaSplit	2	-	9.46
NNJM-Global + AdaSplit + NL	2	96 (1×)	9.55
NNJM-Global + AdaSplit + NL	2	192 (2×)	9.47

Table 5: Test set perlexity for models with global-only non-linear layers. Results for models with no global vector splitting, with fixed section length splitting, and with adaptable section length splitting are shown. *NL* represents the model with the non-linear layer in the global part. *NonLinearSize* represents the size of the global-only non-linear layer. For example, a *NonLinearSize* of *192 (2×)* shows that the global-only non-linear layer has a size of 96, which is 2 times of the word embedding vector size.

explore based on the discussion above. On one hand, we can push harder on sentence vector generation model by adding more free parameters and possibly use RNN model. On the other hand, while sentence vector has little idea on how to adapt itself to optimally predict local information like next target word, we can design network architecture to enable our model to learn this ability of adaption. For example, if we add target n-gram position as another input to the network, it may enable the model to automatically learn word alignment and source window length to optimize local prediction. In such way, we can also get rid of word alignment preprocessing on the parallel texts.

Model	#Sec	Perplexity
NNJM-Global	1	9.45
NNJM-Global+BS	1	9.53
NNJM-Global+FixSplit+BS	2	9.53
NNJM-Global+BS-W2V	1	9.45

Table 6: Test set perplexity for bootstrapping models. *BS* represents the bootstrapped model and all models are bootstrapped with the original NNJM-Global model. One exception is *BS-W2V*: this model is bootstrapped with Google Word2Vec word embeddings for English only.

Due to the limit of time, we are not able to tune hyperparameters especially the multilayer network architecture in enough resolution. Also we test our language model on moderate size of data. In the future, we can evaluate our model on larger data set and have more thourough hyperparameter tuning for each model.

As a part of our work, we also evaluate the effect of bootstrapping the NNJM-Global model by using word embeddings learned from training NNJM on the same dataset, and by using word embeddings from Google Word2Vec (Mikolov et al., 2013). Word embeddings in the model will then be fixed while we train the other parameters. As it is shown in Table 6, this extension does not work as well as we expected: bootstrapping with the pre-trained NNJM word embeddings degrades the NNJM-Global model performance, and bootstrapping with the Word2Vec word embeddings only gives similar results. By observing the learning process we find that, when starting with a pre-trained word vectors, the model can converge much faster than before (typically in less than 10 epochs). This fast convergence often leads the model into a local minimum, and the learned parameters will stay unchanged afterwards. Thus, the model performance will then be influenced by the choice of this starting point. Exploring more sophisticated ways to bootstrap this joint language model will be a possible future direction.

8 Conclusion

In this report we present our work in investigating a neural network joint language model and extending it with global source context. Our experimental analysis demonstrates that network architecture and multiple hyperparameters will influence the performance of this model in specific ways. We also show that visualization of the learned model parameters matches surprisingly well with our intuitions. Furthermore, evaluation shows that incorporating the weighted sum of the splitted source sentence and adding a non-linear layer into a local architecture can further improve the performance of the language model measured by perplexity. Finally, we open-sourced our implementation of both the original model and the extended model.

Acknowledgements

We sincerely acknowledge Thang Luong in the Stanford NLP Group for his advising on this project. We also thank CS224N TAs and Prof. Chris Manning for bringing us such a fruitful and rewarding class.

References

[Bengio et al.2003] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March.

[Devlin et al.2014] Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. 2014. Fast and robust neural network joint models for statistical machine translation. In *52nd Annual Meeting of the Association for Computational Linguistics, Baltimore, MD, USA, June*.

[Huang et al.2012] Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. 2012. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 873–882. Association for Computational Linguistics.

[Mikolov et al.2013] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.