SPOKEN LANGUAGE UNDERSTANDING USING LONG SHORT-TERM MEMORY NEURAL NETWORKS

E4040.2016Fall.MOON.report

Xiang Li xl2533, Yunfan Zhang yz2866, Zhuangfei Yang zy2233
Columbia University

Abstract

In this project we reproduce the result of the paper *SPOKEN LANGUAGE UNDERSTANDING USING LONG SHORT-TERM MEMORY NEURAL NETWORKS* [2] and add attention mechanism to research on its effect. Our results are quite similar with the original paper coordinating all the conclusions it gets. In our experiment, we find that if we use Stochastic Gradient Descent instead of the mini-batch size as the original paper, we can achieve better results. Then We applied the attention mechanism in the paper NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE [3]. We achieved comparable result using attention with the LSTM model in original paper and we believe that it can be improved further.

1. Introduction

RNNs (Recurrent Neural Networks) have achieved outstanding performance in speech recognition, but a simple RNN suffer from the problem of Long-Term Dependencies. LSTM (Long Short Term Memory) Networks are one solution to this problem. An LSTM add gates to control the cell state to avoid the gradient diminishing and exploding problem. Moreover, attention mechanism is an another approach to augment RNNs. RNNS with attention can focus on different positions of the information they're given.

Our project goal is to reproduce the paper *SPOKEN LANGUAGE UNDERSTANDING USING LONG SHORT-TERM MEMORY NEURAL NETWORKS* [2], then add attention mechanism referring to the paper *NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE* [3]. We implemented them in theano.

This project is similar to assignment 4. We built the models on top of it with modifications.

2. Summary of the Original Paper

The paper makes several modifications on original LSTM to train a sequence to sequence word-labeling task, which mainly targets at output-label dependence

2.1 Methodology of the Original Paper

The base model is a 1 layer LSTM, with memory cell, input, output, and forgetting gates. The structure is shown in the Figure 1.

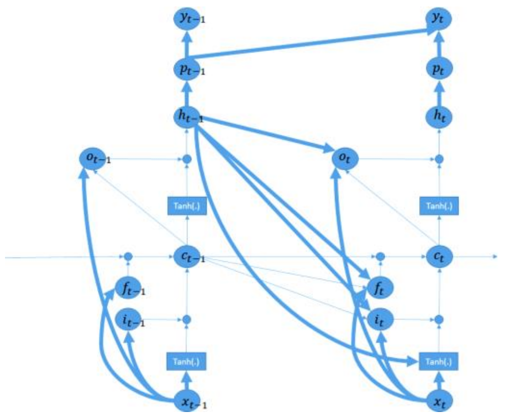

Figure 1. Graphical structure of the LSTM and its moving av- erage extension unrolled at time t − 1 and t. (From *SPOKEN LANGUAGE UNDERSTANDING USING LONG SHORT-TERM MEMORY NEURAL NETWORKS* [2])

The following are the composite function:

$$i_t = \sigma(w_{xi}x_t + w_{hi}h_{t-1} + w_{ci}c_{t-1} + b_i)$$
$$f_t = \sigma(w_{xf}x_t + w_{hf}h_{t-1} + w_{cf}c_{t-1} + b_f)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \, tanh(w_{xc}x_t + w_{hc}h_{t-1} + b_c)$$
$$o_t = \sigma(w_{xo}x_t + w_{ho}h_{t-1} + w_{co}c_{t-1} + b_o)$$
$$h_t = o_t \odot \tanh(c_t)$$

Where σ is the sigmoid function, and $\odot$ is the element-wise product.

Given the LSTM base model, the author conduct following trials:

- Regression: use past predictions as additional input to the output layer. To further include the dependency among output labels. Here author use the lag-3 moving average.
- Auto regression: apply auto regression on the predictions.
- Deep LSTM: a 2-layer LSTM, with the input of the upper layer also input to the lower layer.
- LSTM simplification: by separately set the gate to constant, the author tests the significance of each gate in the LSTM.

2.2 Key Results of the Original Paper

By using the ATIS dataset as training data, the author gets the testing scores in Table 1.

CRF	RNN	CNN	LSTM	LSTM-ma(3)	Deep
92.94	94.11	94.35	94.85	94.92	95.08

Table 1. F1 scores (in %) on ATIS with different modeling techniques. LSTM-ma(3) denotes using moving average of order 3. Deep denotes deep LSTM. (From *SPOKEN LANGUAGE UNDERSTANDING USING LONG SHORT-TERM MEMORY NEURAL NETWORKS* [2])

Given the score, he came to the following conclusion:

- LSTM is generally better than all the other models, by dealing with the gradient vanishing and long dependency.
- LSTM with moving average is a little bit better than original LSTM, by including the label dependency using moving average.
- Deep LSTM is the best model

Furthermore, in the LSTM simplification, he shut down gate one by one and trains the model as shown in the Figure 2.

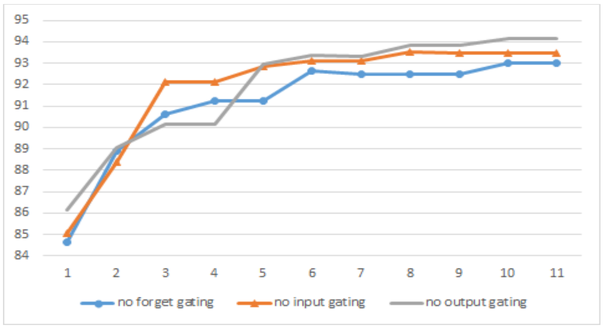

Figure 2. F1 score versus training iterations by simplified LSTM with forget gate fixed to 1.0 in round-marked curve, input gate fixed to 1.0 in triangle-marked curve, and output gate fixed to 1.0 in real curve. (From *SPOKEN LANGUAGE UNDERSTANDING USING LONG SHORT-TERM MEMORY NEURAL NETWORKS* [2])

He further came to the conclusion:

- Without forget gate, the model have the worst performance, indicating that the forget gate is the most important one.

3. Methodology

We break down our objectives into 2 parts: the original paper replication about LSTM and the modification using attention mechanism.

3.1. Objectives and Technical Challenges

First in paper replication, we replicate all the trials that the author implemented, including: 1 layer LSTM, 2 layers LSTM, LSTM with regression, LSTM with one gate closed. And we encounter the following technical challenges:

- Gradient descent: The author use mini-batch with truncated back propagation in optimization, so that the sentence will need 0 padding or truncation to have the same length. However we think this normalization to some extent contradict with the idea of attention, which utilize the dependency of output label.
- Embedding: the paper doesn't mention what is the size of the embedding they choose. And in our implementation, it seems that the train result is very sensitive to the embedding size.
- LSTM with regression: when we implemented the LSTM, by adding the previous hidden layer into the current hidden layer. Sometimes the F1 score will stick around 0 and doesn't improve.

Second in adding attention mechanism into the original paper, we encounter a lot more problems:

- Attention Structure: The attention mechanism was first event to deal with the long memory and label dependence along with RNN. Although there are a few ne papers about adding attention to LSTM, the author all made some modification on how to implement attention. Therefore, we need to find out the fitted attention structure, given the feature of our data.
- Bi-direction LSTM: The original attention model is based on a bi-direction encoder-decoder model. And we think this 'bi-direction' idea will to some extent improve our original LTSM by including the future information of sentence. Given the F1 score in the original paper is already very high, we not be able to identify whether the attention model really can improve LSTM.
- Theano scan function: The most complicated part about attention mechanism is that after computing feedforward propagation, it will iterate over the hidden layer again to compute the weight for all the hidden layers. However theano scan doesn't support to iterate over a symbolic tensor within the computation graph.
- Dynamic input length: Because we didn't normalize the sentence length to make the best of

all the data, we will need to deal with the different sentence size.

- Attention training time: The attention mechanism is inherently very time-consuming to train, because it iterates over the hidden layer for every input with all the other hidden layers to compute the importance weight between word pairs.

3.2. Problem Formulation and Design

Given all the problems that I mentioned before, we use the following methods to solve them

- Gradient descent: we change to stochastic gradient descent so that we don't have to truncate or add zero-padding to the sentence. And it won't contradict with attention later.
- Embedding: We tried different embedding dimension from 50 to 400. And in the end, we choose 2 most significant one to show the difference of performance with different embeddings.
- LSTM with regression: compared to all the other structure, here we can only use smaller learning rate to make sure we won't be stuck in saddle area.
- Attention Structure: we refer to 3 of the papers about attention and a make a combination of their implementation. First we train a bi-direction LSTM to get both the feed forward hidden layer and the feed backward hidden layer, $(\overleftarrow{h}_1, \overleftarrow{h}_2 \dots \overleftarrow{h}_T)$, $(\vec{h}_1, \vec{h}_2 \dots \vec{h}_T)$ like the Figure 3.

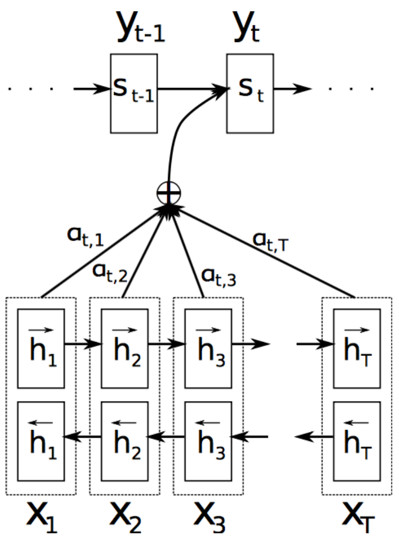

Figure3. The graphical illustration of the proposed model trying to generate the t-th target word yt given a source sentence (x1,x2,...,xT). (From NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE [3])

And then we concatenate these 2 hidden layers to get all the information for the sentence.

$$\tilde{h}_i = \begin{bmatrix} \overleftarrow{h}_T \\ \vec{h}_T \end{bmatrix}$$

Next, in order to compute the weight between word pairs, we will iterate over the feed forward hidden layer to compare the current target hidden layer wit h the bi-direction source hidden layer, where the weight is calculated as below:

$$e_{ij} = v_a^T \tanh(w_a h_i + u_a \tilde{h}_j)$$

Next, we use the softmax function to normalize the weight:

$$a_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})}$$

Now this normalized weight is like the correlation between word pairs. And we further use this weight of different hidden layer for each input words to compute a weight average of all the bi-direction hidden layers:

$$c_i = \sum \tilde{h}_i * a_{ij}$$

In the end we include this weighted averaged matrix, which contains the emphasized information about the entire sentence into the current feed forward hidden layer and reshape it to compute the output layer, like following:

$$\hat{h}_i = \begin{bmatrix} h_i \\ c_i \end{bmatrix}$$

$$s_i = softmax(w\hat{h}_i + b)$$

$$\hat{y}_i = argmax(\hat{p}(y|s))$$

- Bi-direction: To check whether it is the attention mechanism that is improving the model, here we independently train a bi-direction LSTM, see if it will yield better result.
- Theano scan function: we unroll the built-in scan function, and manually write a for loop to compute the iteration over the output feed forward hidden layer in the bi-direction LSTM. However later we figure that removing the bracket from the sequence make it possible for us to loop over symbolic variable.
- Dynamic input length: Except for scan, other computation graph complied before take in the input data. Therefore neither global variable nor private class member can pass-in the dynamic sentence. In the endl we utilize the stochastic iteration depth to break out the loop to make this happen.
- Attention training time: To avoid nested loop in computing the weight between every words pair,

we use matrix computation in calculating the cross-relation between h_i and $\tilde{h}_j$. Therefore only 1 loop will be needed. However the training of attention is indeed incredibly slow.

4. Implementation

We will discuss our model architecture, training algorithm and dataset here.

4.1. Deep Learning Network

1. Architectural block diagram(s).
   The architecture for LSTM model is shown in figure__. The architecture for attention mechanism is shown as following:
2. Training algorithm details
   We use Adagrad algorithm to optimize gradient descent.
3. Data used
   We use the same dataset, ATIS ((Airline Travel Information System)) as the original paper. The ATIS offical split contains 4,978/893 sentences for a total of 56,590/9,198 words (average sentence length is 15) in the train/test set. The number of classes (different slots) is 128 including the O label (NULL). According to the original paper, the author only have train and validation set. Therefore we separate the original ATIS data into training group and validation group.

   We refer to the data processing method in theano DeepLearning Recurrent Neural Networks with Word Embeddings tutorial [4]. A token corresponds to a word. Each token in the ATIS vocabulary is associated to an index. Each sentence is an array of indexes. Then, each set (train, valid, test) is a list of arrays of indexes. A python dictionary is defined for mapping the space of indexes to the space of words. Then Given a sentence i.e. an array of indexes, and a window size i.e. 1,3,5,..., we need to convert each word in the sentence to a context window surrounding this particular word. To summarize, we started with an array of indexes and ended with a matrix of indexes. Each line corresponds to the context window surrounding this word. Finally we associate these indexes to the embeddings (real-valued vector associated to each word).

4.2. Software Design

The code framework we use is quite similar to the assignment 4. We modified the LSTM class to reproduce the original paper as well as add attention mechanism. We have discussed the model architecture in the previous sections. For the detailed implementation, please refer to our Bitbucket link [1].

5. Results

5.1. Project Results

For the paper "Spoken Language Understanding Using Long Short-Term Memory Neural Networks", we implemented the experiments of normal LSTM, Deep LSTM, LSTM-ma(3), and LSTM with simplification using Adagrad optimization.

We use same parameters as the paper. Context window is 3, hidden layer dimension for one-layer LSTM is 300, while for deep LSTM we use 200 hidden layer dimension for the lower LSTM and 300 hidden layer dimension for the upper LSTM. Learning rate was 0.1 per sample.

Rather than training on minibatch as what they did in the paper, the input x_t we trained every time is one sentence. Due to time limitation, we just run 40 epochs for each experiment. Since the word embedding dimension is unknown in the paper, we tried 60 and 400 for these experiments, the results are shown in Table 2.

F1 Score	Emb = 60	Emb = 400
LSTM	94.68	95.89
LSTM-ma(3)	93.42	96.45
Deep LSTM	94.37	96.76
No input gating	89.15	96.4
No forget gating	79.02	94.76
No output gating	78.63	97.17

Table 2. F1 scores (in %) on ATIS with different modeling techniques and word embedding dimensions. LSTM-ma(3) denotes using moving average of order 3.

As for the LSTM with simplification, the parameters we used are different with what the paper used (hidden layer dimension 100, learning rate 0.01). The parameters of our work are same with other one-layer LSTM experiments in our work. The result is shown in Figure 4.

We also explored the attention models based on the papers mentioned in appendix. We tried to implement a simpler model with ideas in the paper "Neural Machine Translation by Jointly Learning to Align and Translate" using the ATIS dataset. The encoder of attention layer is a bidirectional LSTM hidden layer. Hidden layer and word embedding configurations are the same as the single layer LSTM experiments with word embedding dimension of 400, learning rate 0.01/0.02. Because of time and computation capacity limitation, we tested several activation functions of attention models in 0 epochs and list their best F1 score in Table 3.

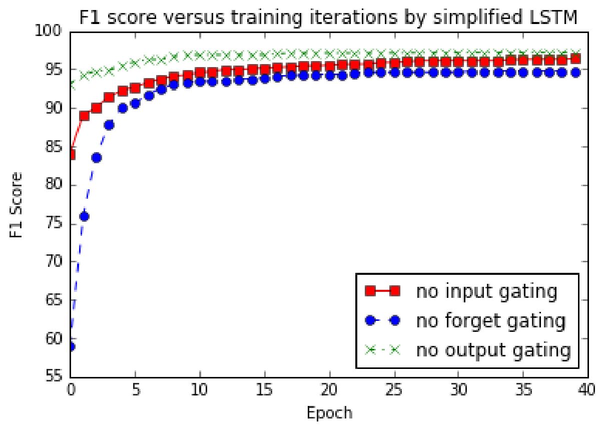

Figure 4. F1 score versus training iterations by simplified LSTM with forget gate fixed to 1.0 in round-marked curve, input gate fixed to 1.0 in square-marked curve, and output gate fixed to 1.0 in cross-marked curve.

Learning rate	tanh/tanh	tanh/sigmoid	sigmoid/sigmoid
0.01	90.81	92.67	92.9
0.02	--	--	94.61

Table 3. F1 scores (in %) on ATIS with attention mechanism of different activation funtion.

5.2. Comparison of Results

Compared with the results and figures of original paper as we mentioned in section 2.2., our best results with word embedding dimension 400 are all better than original paper. The outperformance may come from the different word embedding configuration, and also from the difference that we used SGD instead of mini-batch. Deep LSTM performs much better than original LSTM model, while LSTM with moving average of 3 slightly improves the original LSTM model's performance.

F1 Score	Original Paper	Emb = 400
LSTM	94.85	95.89
LSTM-ma(3)	94.92	96.45
Deep LSTM	95.08	96.76

Table 4. F1 scores (in %) on ATIS with different modeling techniques in the original paper and our reproduction. LSTM-ma(3) denotes using moving average of order 3.

In the Figure 4, we plotted the curve of F1 score of LSTM with simplifications w.r.t. epochs. These curves' shapes are similar when curve trend becomes stable. LSTM without output gating is tend to underperform other two models, while LSTM without output gating performs best, which means the importance of three gates are different.

5.3. Discussion of Insights Gained

The word embedding dimension we used includes 60 and 400. Generally speaking, model with larger embedding dimension has a better performance than model with smaller embedding dimension.

Stochastic Gradient Descent can achieve better results than the mini-batch size as the original paper.

Attention mechanism model needs more time to train than simple LSTM.

6. Conclusion

We successfully reproduce the results of the original paper and prove all the conclusion of it. In our experiment, we find that if we use Stochastic Gradient Descent instead of the mini-batch size as the original paper, we can achieve better results.

We applied ideas of attention mechanism in the paper *NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE* [3] to ATIS dataset. The result (94.61) we got does not outperform the original methods but it is pretty close to the single LSTM model (95.89). We think it may be due to two possible reasons. 1. The attention mechanism is not better for the hyper-parameter configuration in the original paper, but is better for the optimal hyper-parameter configuration. 2. There are many kinds of attention mechanisms. The one we use is designed to solve a translation problem [3], so it may not be very suitable for this task. Therefore, we believe by optimizing hyper-parameters and trying different attention mechanism, we can obtain better result than the original paper.

6. Acknowledgement

We would like to thank professor and all TAs for the amazing course this semester. Without their help, we could not build a solid understanding of neural networks and handle complicated theano coding.

7. References

[1] Link to Bitbucket:
https://zy2233@bitbucket.org/e_4040_ta/e4040_project_moon.git
[2] Kaisheng Yao, Baolin Peng, Yu Zhang, Dong Yu, Geoffrey Zweig, and Yangyang Shi. 2014. "Spoken language understanding using long short-term memory

neural networks", In In Proceedings on IEEE SLT workshop. IEEE Institute of Electrical and Electron- ics Engineers.
[3] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.
[4] Theano DeepLearning Recurrent Neural Networks with Word Embeddings tutorial
http://deeplearning.net/tutorial/rnnslu.html
[5] Theano DeepLearning LSTM Networks for Sentiment Analysis tutorial
http://deeplearning.net/tutorial/lstm.html
[6] Colah's blog on Understanding LSTM Networks
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

8. Appendix

8.1 Individual student contributions - table

	xl2533	yz2866	zy2233
Last Name	Li	Zhang	Yang
Fraction of (useful) total contribution	1/3	1/3	1/3
What I did 1	Implemented Code for LSTM and LSTM-ma(3), and part of attention mechanism	Implemented Code for deep LSTM and part of attention mechanism	Implemented Code for testing gates and hyper-parameter optimization
What I did 2	Wrote Corresponding part of the report	Wrote Corresponding part of the report	Wrote Corresponding part of the report
What I did 3	Discussion	Discussion	Discussion