LSTM and attention for Natural Language Inference

Abstract—Natural Language Inference, which means recognizing entailment relations between pairs of natural language sentences, has been crucial to some NLP domains like machine translation. In this project, we refer to methods in *Reasoning about Entailment with Neural Attention* to construct LSTM models with attention and word-by-word attention respectively. Also, we constructed ESIM model referring to work of Qian Chen. As for tools, nn.LSTM model of *Pytorch* was used and attention was self-defined referring to the article. Finally, the best accuracy for test dataset was achieved at 80.59% with ESIM model.

I. Introduction

Natural Language Inference is a task to determine relationship between a pair of sentences. There are mainly three kinds of relationships we need to take into account: i) entailment—the first sentence(called premise) entails the second sentence(called hypothesis) ,ii)not related, iii) they are contradicting each other. This task is important as many NLP problems such as Information Extraction, relation extraction and machine translation rely on accurate text inference.

In the past few years, some researchers[1-2] used enhanced LSTM models to deal with this task. In their models, neural attention plays an important role in reasoning about entailment. Therefore, we try to draw lessons from their work and utilize attention to complete this task.

When it comes to toolikit, we used *pytorch*. *Pytorch* is a deep learning framework, which can replace *numpy*. It not only inherits many advantages of *numpy*, but also supports GPUs computing. So it excels *numpy* in computing efficiency. As it can quickly build and train the deep neural network model, we used *pytorch* in this project.

II. Methods

In this section, we discuss main methods used in the models. LSTM and attention mechanism are described to pay the way for model construction later.

A.LSTM

As LSTM can relieve the problem of gradient vanishing and gradient explosion, we used LSTM to encode our premises and hypothesis. LSTM can capture and store information for a long period of time, using three types of gates that control the flow of information into and out of these cells: input gates, forget gates and output gates. Given an input vector x_t at time step *t*, the previous output h_{t-1} and cell state c_{t-1}, an LSTM with hidden size *k* computes the next output h_t and cell state c_t as:

$$\begin{bmatrix} \tilde{c}_t \\ o_t \\ i_t \\ f_t \end{bmatrix} = \begin{bmatrix} tanh \\ \sigma \\ \sigma \\ \sigma \end{bmatrix} \left(W \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b \right)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$h_t = o_t \odot \tanh(c_t)$$

B. Attention

When using neural network to process massive input information, we can learn from the attention mechanism of human brain to only select some key information input for processing. In attention mechanism, query vector is introduced to select task-related information from N input vectors $[x_1, x_2 \dots x_N]$. A scoring function is used measure similarity between each input vector and query vector.

In this case, attention distribution α_i can be calculated as:

$$\alpha_i = p(z = i|\boldsymbol{X}, \boldsymbol{q})$$

$$= softmax(s(\boldsymbol{x_i}, \mathbf{q}))$$

$$= \frac{exp(s(\boldsymbol{x_i}, \mathbf{q}))}{\sum_{j=1}^{N}(s(\boldsymbol{x_j}, \mathbf{q}))}$$

Then a soft information selection mechanism is used to gather input information together:

$$\mathrm{att}(\mathrm{X}, \mathrm{q}) = \sum_{i=1}^{N} \alpha_i x_i$$

In Natural Language Inference task, we don't need to capture the whole semantics of the premise in its cell state. Instead, it's sufficient to output vectors while reading the premises and focusing on the key information and accumulating a representation in the cell state. Then these information can be transformed to the second LSTM and help determine the class of relationship.

III. Models

In this section, we mainly refer to work of Tim Rocktäschel etal.[2] to construct models to complete recognizing textual entailment task. A traditional conditional encoding LSTM, an extension of an LSTM with attentional and one with word-by-word attention are introduced.

A. Conditional Encoding

Firstly, we don't take attention into account and focus on LSTM only. Two LSTM are used to process premises and hypothesises respectively. We call it Model A here. It's noticeable that initial state of the second LSTM depends on the last hidden state of the first LSTM. In this way, the second LSTM with different parameters can focus more on premise-related information when processing the hypothesis.

We use word2vec vectors as word representations and we don't optimize them during training. As for OOV problem, out-of-vocabulary words in training set are randomly initialized and optimized during training. And, they keep the same fixed random vectors for validation and test set corpus.

What's more, a linear layer is used to project word vectors to the dimensionality of the hidden size of the LSTM. Finally, for classification, a softmax layer over the output is constructed and we train with cross-entropy loss.

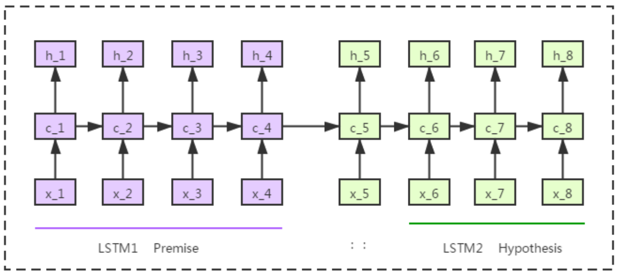

Fig 1. Model A

B. Attention

Next, we try to build attention mechanism between hypothesis and premise. In this part, we use the last hidden state to build attention with each word of premise. Additive model $s(\boldsymbol{x_i}, \boldsymbol{q}) = \boldsymbol{v}^T \tanh(W\boldsymbol{x_i} + U\boldsymbol{q})$ is used as the scoring function here. The model structure can be depicted as follows:

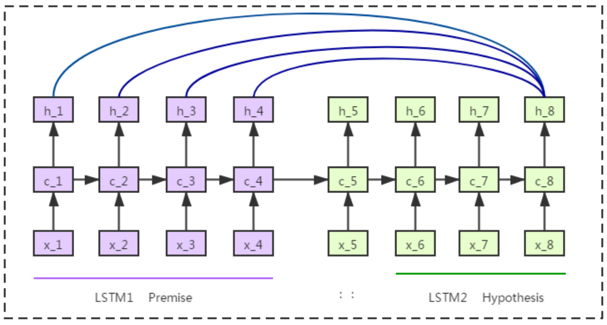

Fig 2.Model B

In this case, we can compute attention of the final hidden state on premise as follows:

Use $\mathrm{Y} \in R^{k \times L}$ be a matrix consist of the first LSTM's output vectors $[h_1 \ldots h_L]$ when processing the L words of premise, where k is a hyperparameter denoting the size of embeddings and hidden layers. e_L and h_N represent a vector of 1s and last hidden state processed by the two LSTMs respectively. Then we can produce a vector $\boldsymbol{\alpha}$ of attention weights and a weighted representation $\mathbf{r}$ via:

$$\mathrm{M} = \tanh(W^y Y + W^h h_N \odot e_L)$$

$$\alpha = \mathrm{softmax}(w^T M)$$

$$r = Y\alpha^T$$
$$h^* = \tanh(W^p r + W^x h_N)$$

Where W^y, W^h, W^p, W^x are trained matrices, w is a trained parameter vector. h^* represents the final sentence-pair representation used for classification.

C.Word-by-word Attention

Although LSTM can model long distance information, some details can not be captured when the sentences are relatively long. So we decide to compute attention at each output of the second LSTM, and each attention is depend on attention of last step. As result, we can get word-by-word attention. For example, if we have three words(A,B,C) in hypothesis, then we need to construct attention :A-premise, B-premise, C-premise, just as the figure shows:

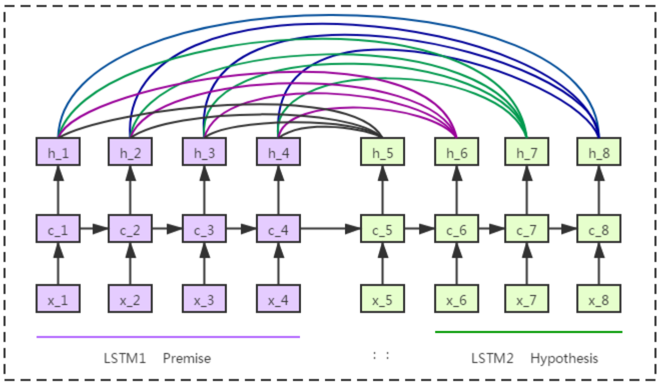

Fig 3.Model C

In this case, we can generate an attention weights α_t and a weighted representation r_t at each step of hypothesis. This can be modeled as follows:

$$M_t = \tanh(W^y Y + (W^h h_t + W^r r_{t-1}) \odot e_L)$$
$$\alpha_t = softmax(w^T M_t)$$
$$r_t = Y{\alpha_t}^T + \tanh(W^t r_{t-1})$$

The final sentence-pair representation is obtained from the last attention-weighted representation r_L of the premise and the last output vector h_N using:

$$h^* = \tanh(W^p r_L + W^x h_N)$$

D.Enhanced Sequential Inference Model (ESIM)

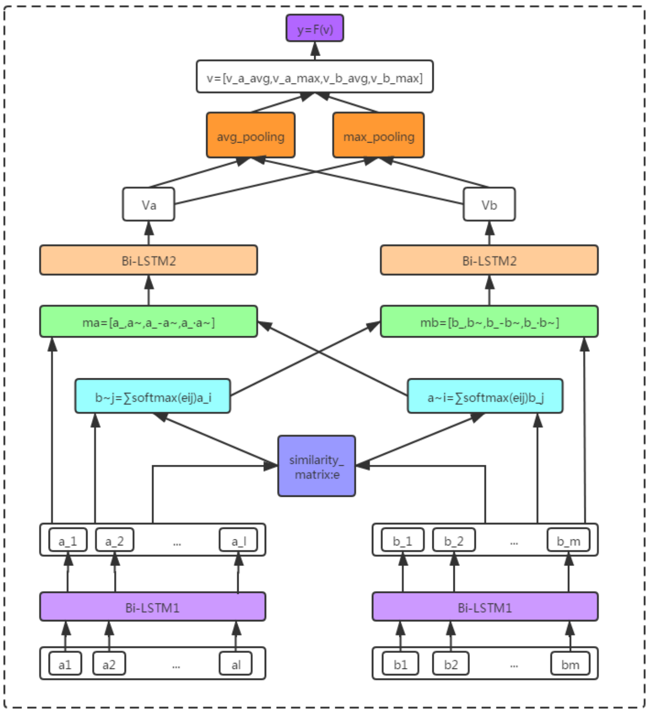

Fig 4. ESIM structure

Referred to work of Qian Chen etal[1], ESIM model is constructed as Fig 4 shows. We mainly have 4 layers to process given premises and hypothesises.

- Input Encoding Layer
  A Bi-LSTM, which enforces contextual meaning of every word, was used to re-encode the premise and hypothesis.
- Local Inference Layer
  Here, Local Inference is attention mechanism, in fact. A similarity matrix e of premise and hypothesis was constructed first. Then, combine matrix e and encoded premise $\bar{a}$ and encoded hypothesis $\bar{b}$ to calculate the mutual expression between premise and hypothesis. When we get the weighted sum $\tilde{a}$ and $\tilde{b}$, we enforce the local inference by concatenating several tensors and get m_a and m_b.
- Inference Composition Layer
  At this layer, a Bi-LSTM is firstly exerted to enforce the contextual information of m_a and m_b to get v_a and v_b. Then an average pooling and a max pooling are used to help form the final tensor v.
- Prediction Layer
  Finally, a prediction layer with two fully-connected layers of *tanh* activation function and *softmax* function are built to process the final tensor v.

Utilizing *pytorch*, input encoding layer and composition layer were constructed by nn.LSTM, local inference layer was built according to formulas mentioned above.

IV. Experiments

Our dataset comes from the Stanford Natural Language Inference corpus [3]. All sentence-pairs in SNLI stem from human annotators. So the size and quality of SNLI make it a suitable resource for training neural architectures. It's noticeable that we only use corpus that labeled "contradiction"," neutral" and "entailment" in our project.

For word embedding, glove[4] pretrained word vectors of 300 dim are used. We found 39k words in SNLI for which we could not obtain glove embeddings, resulting in 11.4M tunable parameters.

The corpus is divided into Development set and Test set, go a step further, we divided the Development set into Training set and Dev-Test set. Hence, we do training on training set, adjusting parameters and choosing the best classifier base on the results from Dev-Test set.

A. Files

Structure of our project is depicted like this:

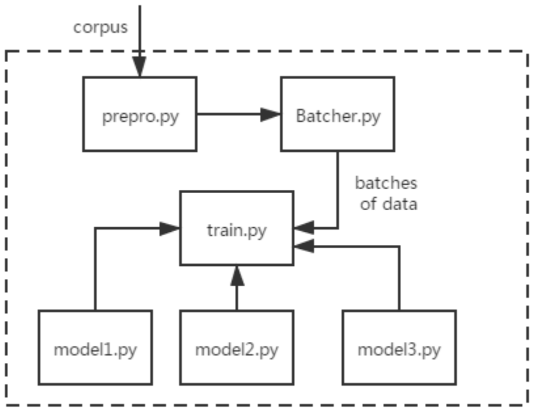

Fig 4. File structure

Prepro.py is a file for preprocessing. We read the corpus, tokenize the sentences and integrate them into a dataset, according to types of data: train, dev, test and premises, hypothesis and targets. Also, in order to train via batches of data, *Batcher.py* was created to generate batches of data for further training. Sequence padding and word embedding are also done in this file. As a result, we can easily get word vectors of batches of samples. LSTM models with different attention of premises are respectively constructed in files of *model1.py*, *model2.py* and *model3.py*. LSTM structure, attention and forward propagation are defined in these models. Finally, we load data and train the models in *train.py*.

B. Results

Data was trained and tested with above four models. The results show that ESIM model is fast and effective .The best results of our ESIM model are as follows:

criterion	Train_acc	Dev_acc	Test_acc
	81.25%	80.29%	80.59%

Tab 1. Experiments results

The results show that sequential inference models based on chain LSTMs can help capture information between premises and hypothesis.

C. Debug Experience

In experiments of this project, I come across many problems while coding and debugging. As some of them may be inspiring and meaningful for further study, they are recorded.

Problem:
I found that loss don't decrease obviously after long time training. After printing, I found that the net parameters didn't change at all, which means the network didn't learn anything. So I started to investigate problems.

Debug:

- Dataset related problems
  - Check input data
    There is no problem with the data dimension, but it is found that each premise vector is exactly the same as its corresponding hypothesis vector, so the training must be random. I finally found that the word segmentation was not successful in the preprocessing, and embed the sentence as a whole.
  - Make sure input is connected to output
    I print several pairs and make sure they are connected.

    - Check the data loader
    - Shuffle the dataset

      I shuffle the dataset to make sure sequence of the sentences pairs is random, to reduce the influence of order.
- Normalization/ Training problems
    - Normalization for input data

      I used batch normalization before inputting the vectors into the LSTM models.
    - Have too much data augmentation?

      I try to remove regularization and dropout first and adjust the models
    - Verify loss input

      I used nn.CrossEntropyLoss as loss function, that means the input probability distribution don’t need softmax, because the function has softmax itself.
    - Monitor other metrics

      Besides loss, I also pay attention to metrics like accuracy.
    - Check for hidden dimension errors
    - Explore Gradient checking/ Exploding / Vanishing gradients

      After setting a very large value for “learning rate”, I printed the grad value of net weights and found that grad of the first LSTM was quite small, while the second LSTM has normal grad value. This means, during back propagation, gradient vanishing happened on the first LSTM. This may be the reason why the parameters can’t be upgraded.

      After studying source code of nn.LSTM, I found that it doesn’t have Super parameter of ‘nonlinearity’, so I can’t assign ‘relu’ to its activation function very easily. So I tried some other methods. After the first LSTM, I add a batch normalization and the weights can change obviously.
    - Check weights initialization

      For self-defined weight matrixs, I firstly initialize them as zeros. This can lead to difficulty of learning for the network . So I used random initialization.
    - Switch from Train to Test mode

      There are some difference for train-set and dev-set while processing. For example, dropout is needed for train-set, not for dev-set and test-set.
    - Give it more time

V. Conclusions

In this report, LSTM models with attention mechanism were implemented to recognize entailment relations between pairs of natural language sentences. Also, ESIM model of Chen etal[1] was constructed with reference to their paper. With many bugs found and solved, models perform well and the final accuracy for test dataset was achieved at 80.59%. Further improvement is needed, for example, tree-LSTM can be discussed.

Appendix

[1] Chen, Q. , Zhu, X. , Ling, Z. H. , Wei, S. , & Inkpen, D. . (2017). Enhanced LSTM for Natural Language Inference. Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).

[2] Rocktäschel, Tim, Grefenstette, Edward, Hermann, Karl Moritz, Kočiský, Tomáš, & Blunsom, Phil. . Reasoning about entailment with neural attention.

[3] Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. In Conference on Empirical Methods in Natural Language Processing (EMNLP), 2015.

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