CSE244 HW2 Report

Zhengqi Liu
zliu194@ucsc.edu

1 Prepare

For the data preprocessing, first, build the vectorization function to convert the input words to vectors. Here I use the word embedding technique. The pretrained data set I choose is GloVe. GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. For the sake of efficiency, I use the 100 dimension version. I store them in a hasp map named 'word_to_index'. Because I will use padding in the training process, I manually set the word embedding of the word 'pad' to be all zero, it can avoid the padding cause unknown impact.

Second, build a function to convert the target, tagging, to index. I count the occurrence number of each target, and sort time from the highest to the lowest. Then map them to indices one by one. Also, I map the 'pad' tag to 0.

Third, build a function to convert the input sentence to vector matrix. Because I use the LSTM model in the training process, the input data should be sequential. What I did is traverse all the words in the sentence, converge them to word embedding, and concatenate all the words. The data structure of the input of LSTM model is shown as figure 1. It shows the input of sentence "The cow jumped" should look like.

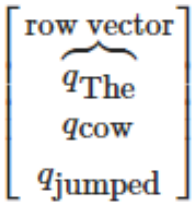

Figure 1: Input of LSTM model

2 Build Model

The model I build is an RNN model with LSTM cells. It's two layers neural network, one LSTM layer, and one linear layer. In the forward process, first, I converge the input sentence to a matrix, use the function I mentioned in the Prepare section. Then the matrix will be input to the LSTM layer and linear layer. The output layer uses the log softmax function, which can get the score of each label. The source code of the structure of the model is shown in figure 2.

```
class LSTMTagger(nn.Module):


    def __init__(self, embedding_dim, hidden_dim, tagset_size):
        super(LSTMTagger, self).__init__()
        self.hidden_dim = hidden_dim

        # The LSTM takes word embeddings as inputs, and outputs hidden states
        # with dimensionality hidden_dim.
        self.lstm = nn.LSTM(embedding_dim, hidden_dim)

        # The linear layer that maps from hidden state space to tag space
        self.hidden2tag = nn.Linear(hidden_dim, tagset_size)

    def forward(self, sentence):
        embeds = prepare(sentence)
        lstm_out, _ = self.lstm(embeds.view(len(sentence), 1, -1))
        tag_space = self.hidden2tag(lstm_out.view(len(sentence), -1))
        tag_scores = F.log_softmax(tag_space, dim=1)
        return tag_scores
```

Figure 2: LSTM Model

3 Training

In the training process, I choose the Binary Cross Entropy as the loss function and Adam as the optimizer. I meet a problem that the length of the input sentence is different, range from 3 to 21. The average is 6.7. So I choose 10 to be the length of each input. There are 256 sentences in the training set has a length greater than 10, which occupies about 10 percent in the training set. If the length of a sentence is greater than 10, it will be truncated; if the length of a sentence is less than 10, it will be padded to 10. Both the words and tags of the padded part is 'pad'.
The other problem I meet is, there exists a space before symbol "" in the training and test data. It's like " 's". If the word before " 's" is words like "what 's", "who 's" or "it 's", " 's" isn't an independent word. It will share the same tag as the word before it. But in some case when " 's" in the name of a movie, like "charlie 's angels" the tag of this part will have it's one tag. To solve this problem, I split the sentence with the keyword " " first. Then I compare the length

of words and tags. If the length of tags is less than words, I will concatenate " 's" with the word before it. Then continue doing the following process.

I run 30 epoch, and it seems to converge in the 25th epoch. The loss of each epoch is shown in figure 3.

Finally, the trained result I get is a [10, 26] matrix. Find the max value in each row, and its index is the predicted tag of the word.

```
epoch= 0 loss= tensor(0.4963, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 1 loss= tensor(0.2640, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 2 loss= tensor(0.1890, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 3 loss= tensor(0.1469, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 4 loss= tensor(0.1183, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 5 loss= tensor(0.0992, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 6 loss= tensor(0.0864, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 7 loss= tensor(0.0761, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 8 loss= tensor(0.0691, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 9 loss= tensor(0.0677, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 10 loss= tensor(0.0611, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 11 loss= tensor(0.0600, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 12 loss= tensor(0.0550, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 13 loss= tensor(0.0556, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 14 loss= tensor(0.0518, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 15 loss= tensor(0.0511, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 16 loss= tensor(0.0496, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 17 loss= tensor(0.0510, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 18 loss= tensor(0.0482, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 19 loss= tensor(0.0481, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 20 loss= tensor(0.0466, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 21 loss= tensor(0.0477, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 22 loss= tensor(0.0462, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 23 loss= tensor(0.0457, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 24 loss= tensor(0.0462, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 25 loss= tensor(0.0428, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 26 loss= tensor(0.0448, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 27 loss= tensor(0.0432, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 28 loss= tensor(0.0446, device='cuda:0', grad_fn=<DivBackward0>)
epoch= 29 loss= tensor(0.0441, device='cuda:0', grad_fn=<DivBackward0>)
```

Figure 3: Loss of Each Epoch