CMSC 25400 Homework 6

Ruolin Zheng

March 9, 2019

The submission zip file include:
bobsue.seq2seq.pred.tsv - the prediction result with highest accuracy for *bobsue.seq2seq.test.tsv*
bobsue.seq2seq.pred.interesting.tsv - the prediction result with a lower accuracy but slightly more interesting and diverse word choice
lstm.accu0.06 - the highest-accuracy model state dict created with torch.save()
lstm.interesting - the more interesting model state dict created with torch.save()
embed - the random embedding dict created with torch.save()
glove - the GloVe embedding dict created with torch.save()
lstm.py - the LSTM, LSTMCell classes and training functions
utils.py - helper functions for pre and post processing
driver.py - driver code for training and plotting
hw6.pdf

1 Program Description

1.1 Input and Output

The program reads in a TSV file of sentence pairs, splits the pairs and trains the LSTM model on the training set. During each epoch of the training, the program validates the model on a holdout validation set. Finally, after training for several epochs, the LSTM model is tested to predict the second sentence given the first sentence.

1.2 Class Definition

LSTMCell is a class which computes f, i, g, o, c_t, h_t given the hidden and the context as h_t, c_t. Instead of concatenating x, h_t and using one linear layer for each of f, i, g, o, I used two different linear layers for x, h_t. In other words, my **LSTMCell** has 16 parameters (8 weights and 8 biases) as opposed to the regular 4 weights and 4 biases $W_f, W_i, W_g, W_o, b_f, b_i, b_g, b_o$. Admittedly this may take more computation power during backpropagation, but the effort is justified in giving the model slightly more flexibility.
All of the **LSTMCell**'s attributes are tensors on which backpropagation is performed:
self.w_if, self.w_ii, self.w_ig, self.w_io, self.w_hf, self.w_hi, self.w_hg, self.w_ho.

LSTM is a class which serves to encode the input string word by word. For each word x_t, it obtains h_t, c_T from the encoder (an **LSTMCell** object,) and feeds the next word x_{t+1} as well as h_t, c_t to obtain the next pair of h_{t+1}, c_{t+1}. Finally, it decodes $[h_1, \ldots, h_k]$ with a linear layer to produce the output. The class has the following attributes: *self.n_hid, self.lstmcell, self.fc.* The second is the encoder and the third is the decoder. Backpropagation is performed on both the encoder and the decoder.

1.3 Pre-Processing and Word Embedding

As defined in *utils.py*, there are various helper functions for generating a random word embedding as vectors of length 200 from a uniform distribution $[-0.1, 0.1]$, given a vocabulary; for reading and parsing files; and for converting a string of words into its tensor representation. Please refer to the docstrings of **rand_embedding(), line_pairs(), line_to_tensor()**. The random word embedding file *embed* is stored as a dictionary, mapping each word in *bobsue.voc.txt* to its word vector of shape (1, 200). Pre-computed word embedding file *glove* is also stored as a dictionary and includes all words in *bobsue.voc.txt.* For simplicity's sake, all words and sentences used during training have been lowercased.

1.4 Training

The training algorithm is as follows:

```
1  for each training example do
2      forward the first sentence to obtain hidden and context;
3      for word in the second sentence do
4          forward the ground truth word to obtain new hidden and context;
5          apply linear transformation to hidden to generate raw prediction;
6          find the word vector in our dictionary closest to raw prediction;
7          record both raw prediction and predicted word;
8      end
9      compute loss based on ground truth sentence and raw predictions;
10     compute accuracy based on ground truth sentence and predicted words;
11     back propagate on the loss;
12     let optimizer update the gradients;
13 end
```

For class and function definition, please also refer to the docstrings in *lstm.py.*

1.5 Validating

In every epoch, after the training is completed, the model is then validated on *bobsue.seq2seq.dev.tsv*, a validation set of size 750. Reasonably enough, the loss on the validation set is usually observed to be higher than that on the training set, whereas the accuracy is lower. If the loss on the validation set is noticeably higher than the previous epoch, this may indicate overfitting and calls for the end of training.

1.6 Predicting and Post-Processing

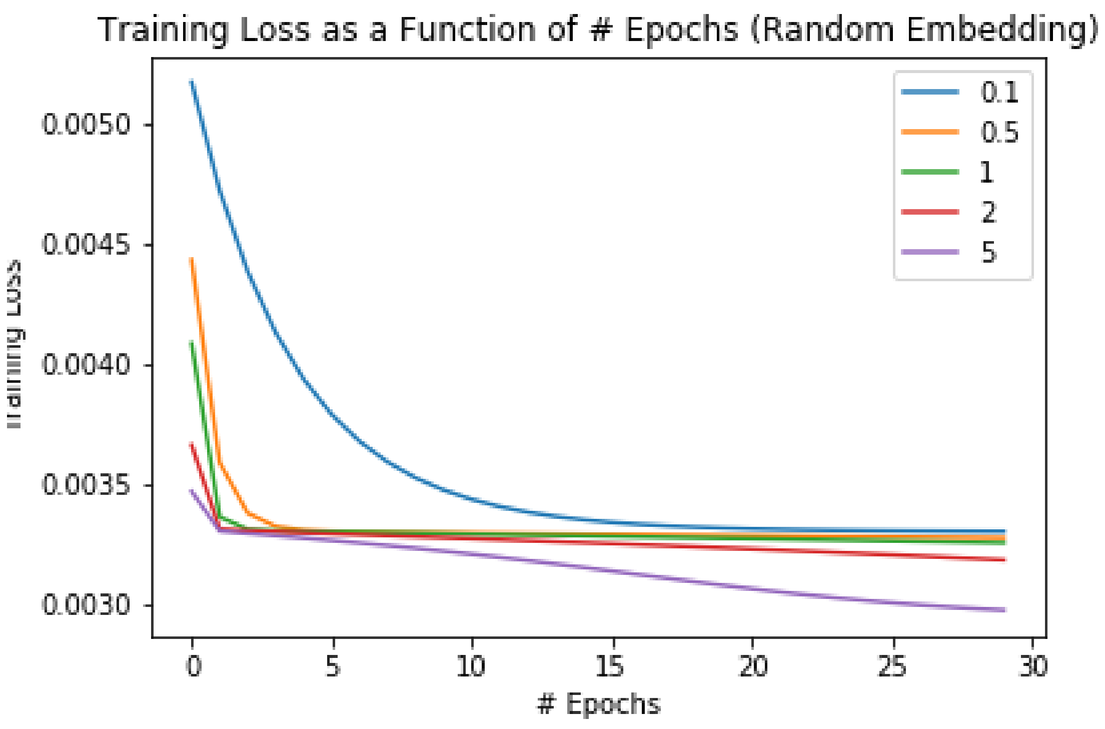
After several training epochs, the model is tested on *bobsue.seq2seq.test.tsv*, a test set of size 750. To generate a prediction for the second sentence, the model encodes the first sentence to obtain h_t, c_t. Then, we feed in the start token <s> to obtain h_{t+1}, c_{t+1}. The model decodes h_{t+1} as the output x_t and searches for the closest word vector $closest(x_t)$ in the word embedding; $closest(x_t)$ is thus the predicted first word of the second sentence, and we feed $closest(x_t), (h_{t+1}, c_{t+1})$ into the model. Repeat this procedure of encoding a single word, obtaining hidden and context, decoding to get the output, finding the closest word vector and then encoding this vector, until we encounter the end token </s>. We now have a complete sentence $closest(x_t), \ldots, closest(x_{t+k})$, each element corresponding to a word vector in our embedding.

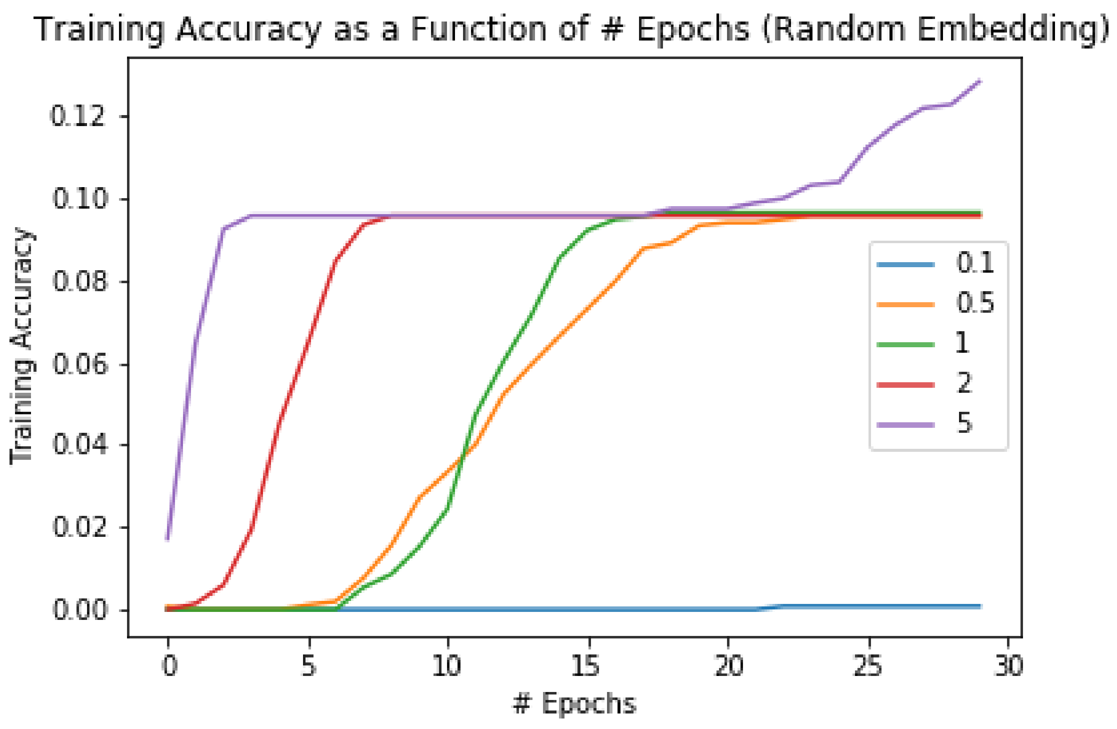
2 Performance Study

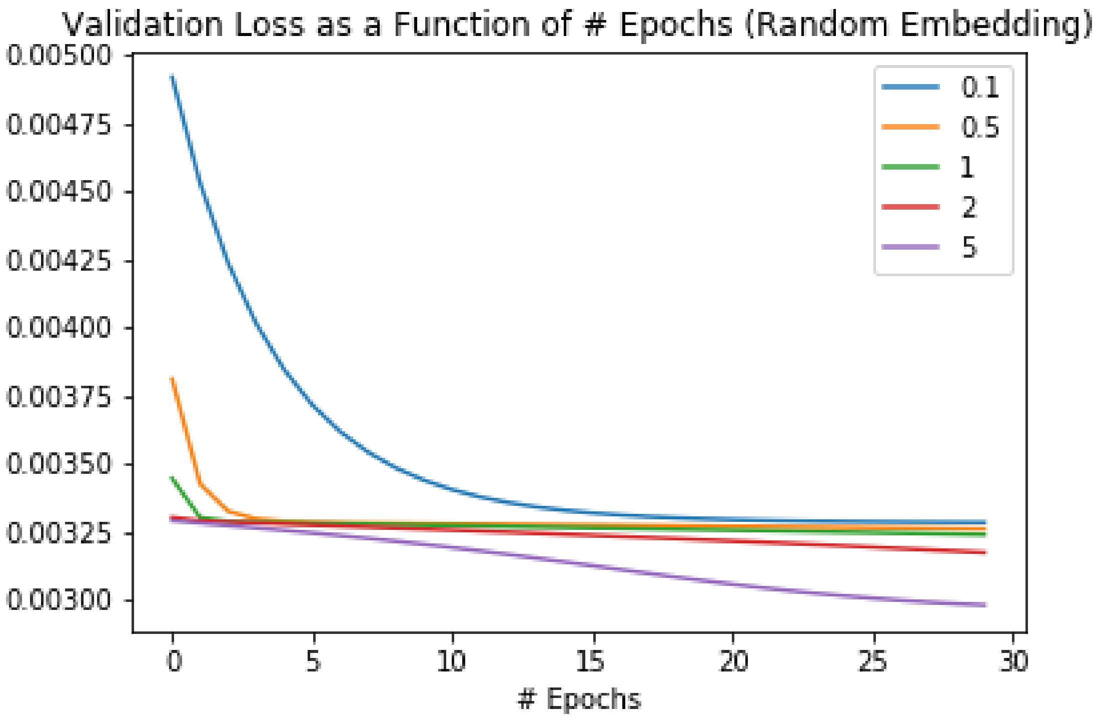
The plots shown are generated on a smaller subset of the training set, and serve as a performance study of varying learning rate. During the actual training, I use a scheduler to gradually decay my learning rate and sometimes manually adjust the learning rate to avoid overfitting; these techniques are not reflected in these plots.

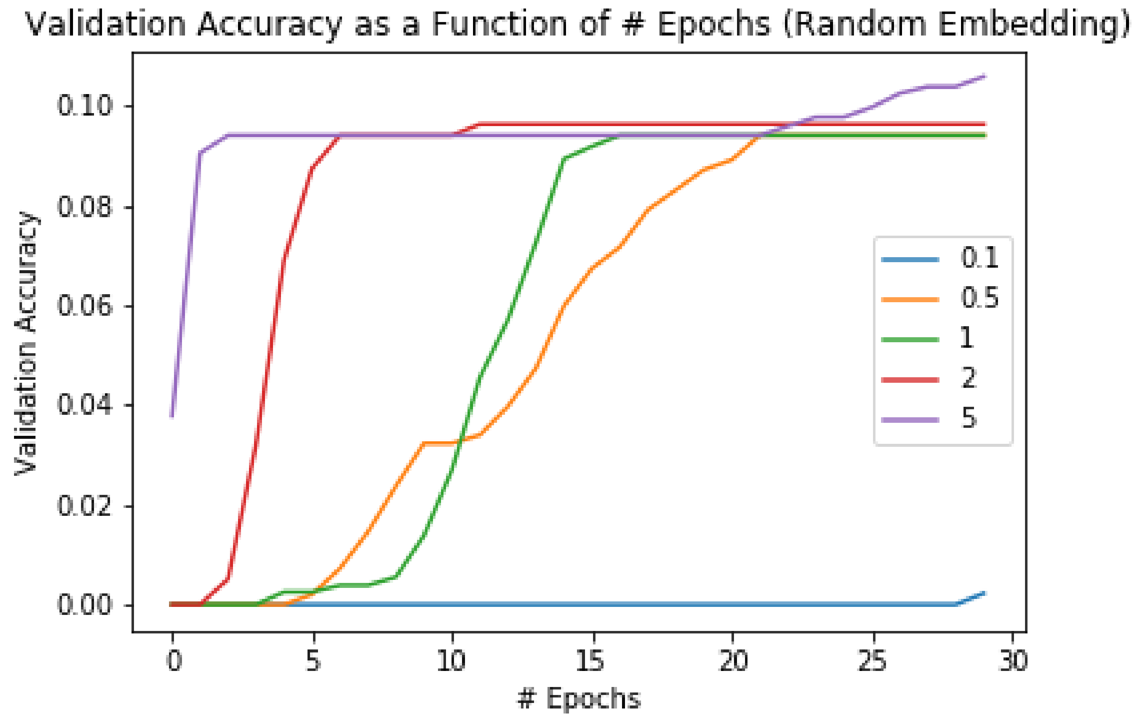
2.1 Random Word Embedding

The plots below are generated over 30 epochs on a training set of size 200 and a validation set of size 50; using the random word embedding (uniformly distribution on $[-0.1, 0.1]$.) With varying learning rates from 0.1 to 5, the loss on the training and the validation sets decreases faster and reaches a lower value for larger learning rate; the accuracy on the training and the validation sets increases slightly faster and reaches a higher value for larger learning rate.


Training Loss as a Function of # Epochs (Random Embedding)
Training Loss
# Epochs
0.1
0.5
1
2
5

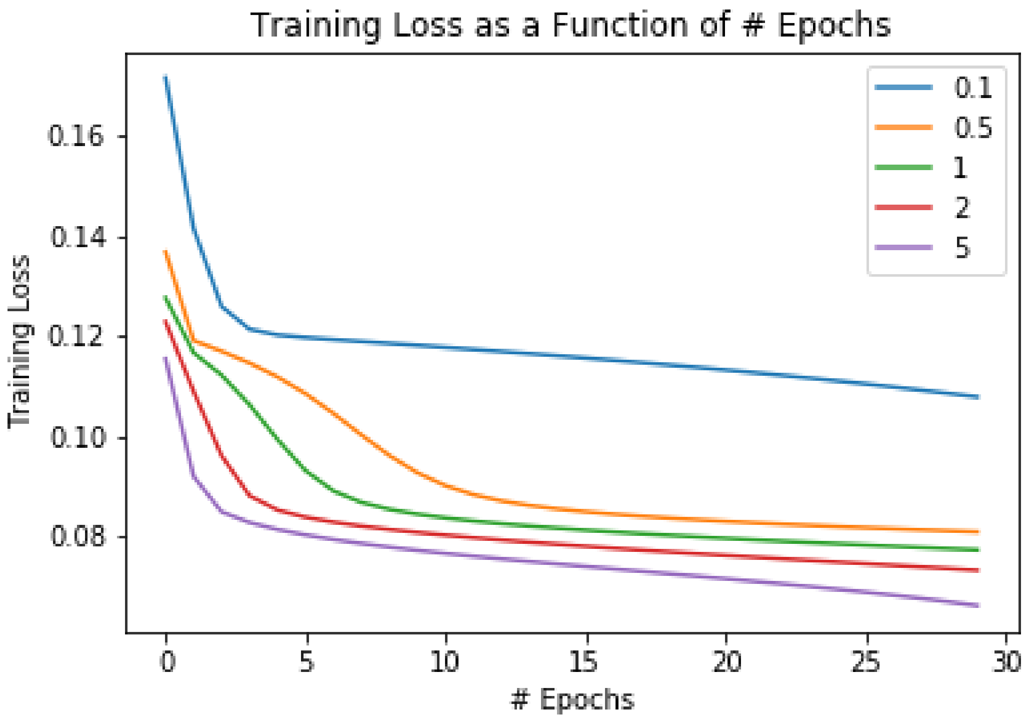

Training Accuracy as a Function of # Epochs (Random Embedding)
Training Accuracy
# Epochs
0.1
0.5
1
2
5

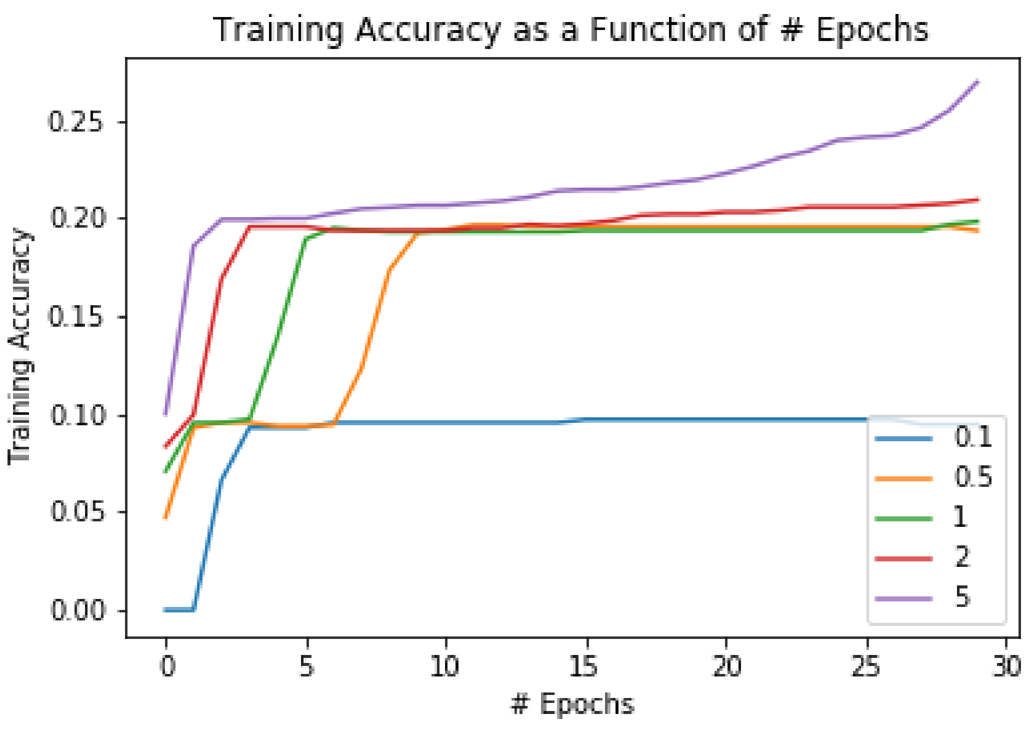

Validation Loss as a Function of # Epochs (Random Embedding)
0.00500
0.00475
0.00450
0.00425
0.00400
0.00375
0.00350
0.00325
0.00300
0
5
10
15
20
25
30
# Epochs
0.1
0.5
1
2
5

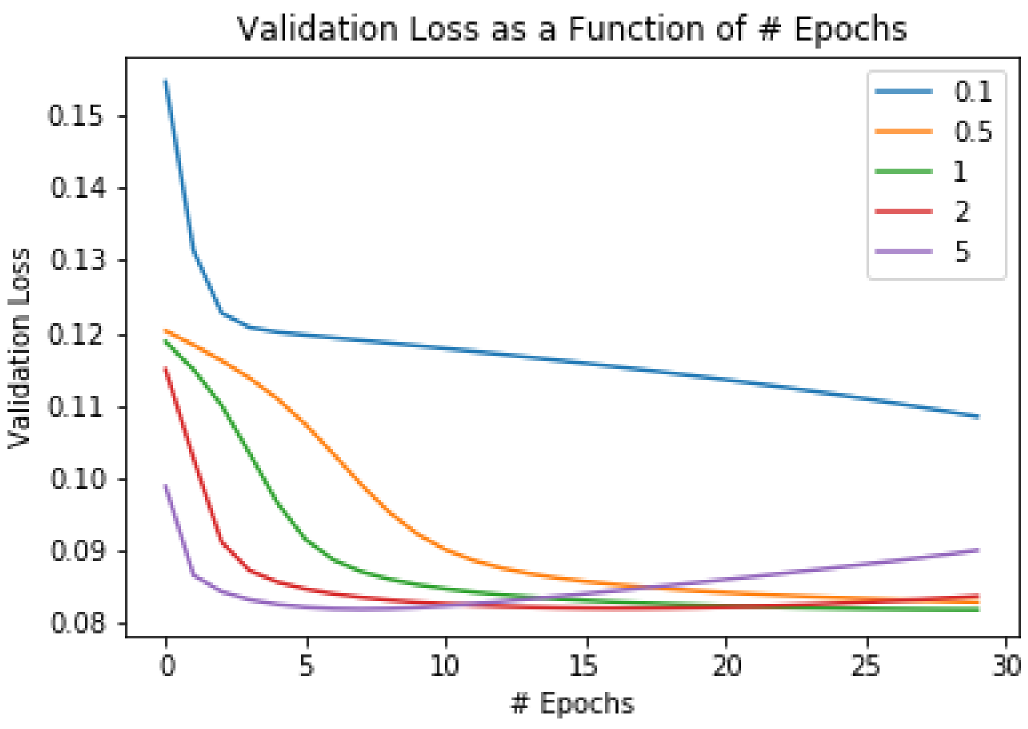

Validation Accuracy as a Function of # Epochs (Random Embedding)
Validation Accuracy
0.10
0.08
0.06
0.04
0.02
0.00
0
5
10
15
20
25
30
# Epochs
0.1
0.5
1
2
5

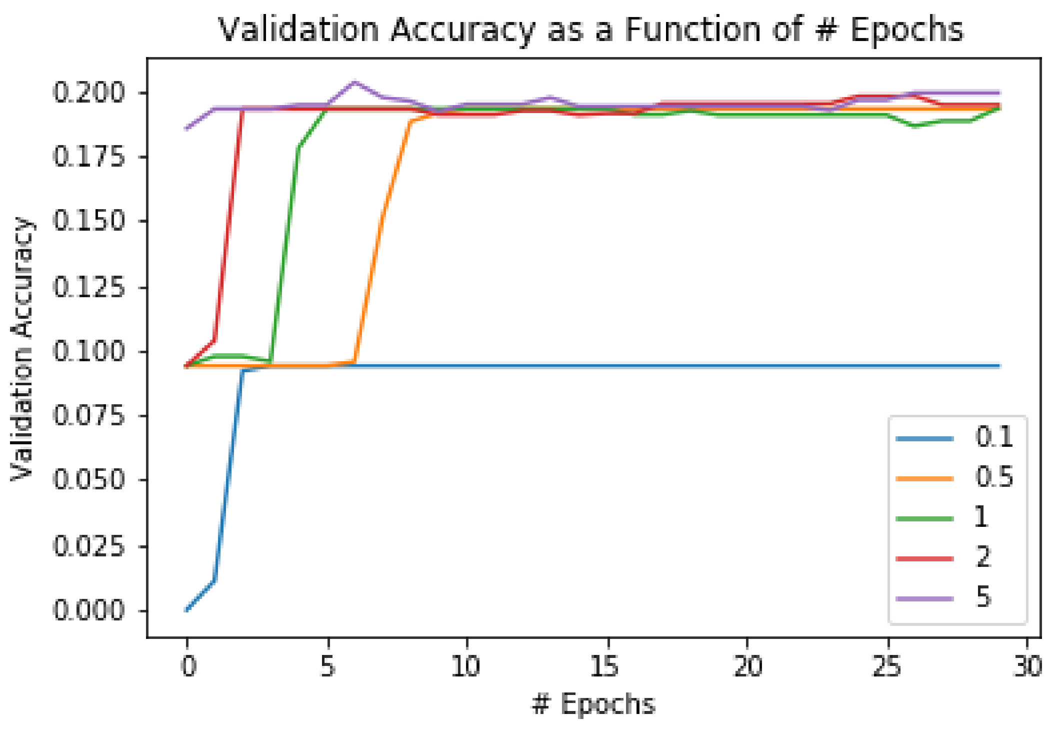
2.2 Pre-Computed Word Embedding

As illustrated in the plots below, using a pre-computed word embedding like GloVe helps the model to perform better than using a random word embedding. The trend is similar to that of the random embedding in that the loss decreases faster with a slightly higher learning rate. It is worth noting that the loss on the y-label is higher for the pre-computed embedding than the random; at convergence, 0.08 for the former and 0.0030 for the latter. This may be the result of generating the random embedding in the range $[-0.1, 0.1]$, causing the word vectors to be closer to each other than in the precomputed version (presumably -1.0 to 1.0). Therefore, the loss appears lower when we computes the MSE Loss. However, the rate of convergence is quite similar: after 30 epochs, at a learning rate of 1, the loss decreases by about 50% for both the random embedding and the pre-computed one.



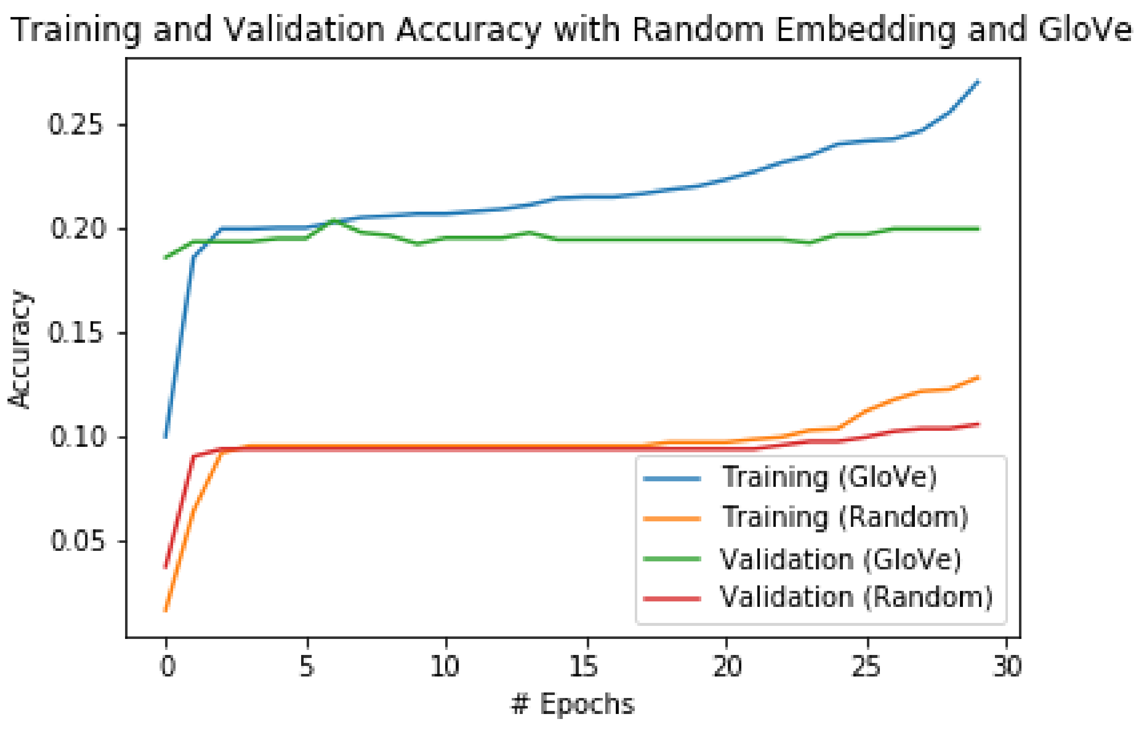

Training Accuracy as a Function of # Epochs
Training Accuracy
# Epochs
0.00
0.05
0.10
0.15
0.20
0.25
0
5
10
15
20
25
30
0.1
0.5
1
2
5


Validation Loss as a Function of # Epochs
Validation Loss
# Epochs
0.08
0.09
0.10
0.11
0.12
0.13
0.14
0.15
0
5
10
15
20
25
30
0.1
0.5
1
2
5



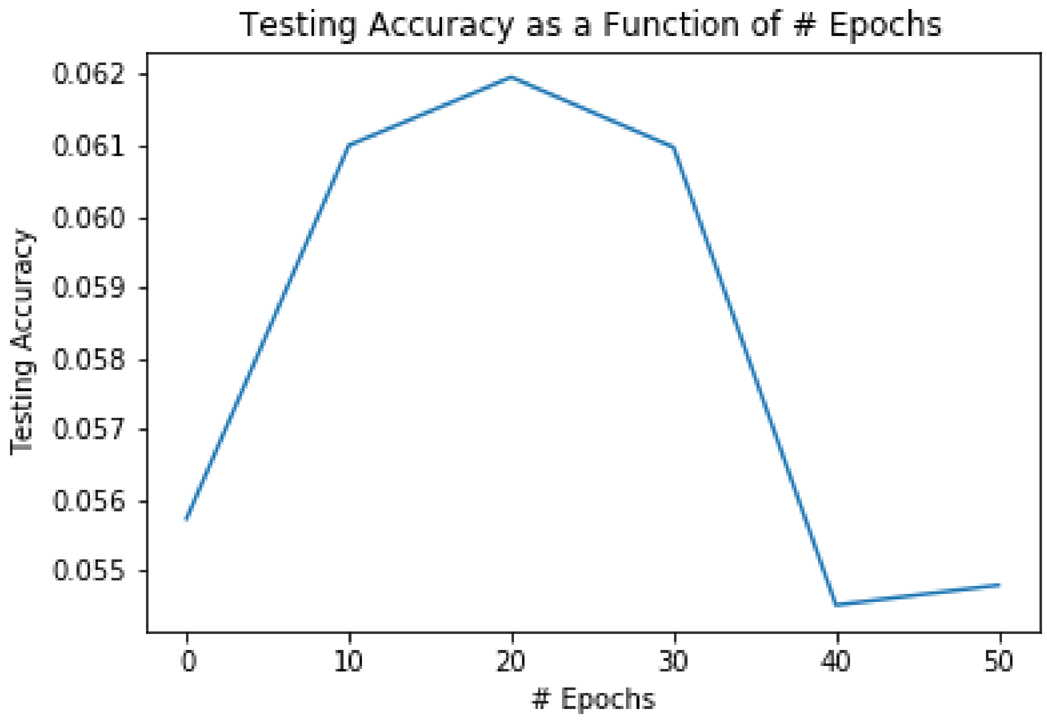
2.2.1 Training and Validation Accuracy Comparison on Random Embedding and GloVe

As illustrated in the plot, using the pre-computed word embedding GloVe results in a higher accuracy to start with for both the training and the validation set; as well as a slightly more steady increase on the training set.



2.3 Accuracy on the Test Set

I used GloVe in my production training; I trained my model on all 6000 samples, at an initial learning rate of 5, decaying by a factor of 0.5 every 5 epochs; I trained the model for a total of 50 epochs and tested it every 10 epochs. I found out that the performance peaked at around 20 epochs, at which point the learning rate should have decayed to about $5 * (0.5^4) = 0.3125$. The final prediction result *bobsue.seq2seq.pred.tsv* achieves an accuracy rate of 6.2% compared to the ground truth in *bobsue.seq2seq.test.tsv*. As illustrated in the plot, the accuracy decreases from epoch 20 to 30, and even more sharply from epoch 30 to 40; this may indicate overfitting. The small increase from epoch 40 to 50 may as well be a small amendment given the already decayed learning rate.



3 Result

3.1 20 Words with Highest Accuracy Rate

Below are the words that my model most often predicts correctly. Unsurprisingly, the model is quite good at predicting pronouns, names and some common words such as ‘but’ and ‘went’.

Word	Accuracy Rate	Occurrences
he	0.386364	264
she	0.378238	193
bob	0.277778	198
sue	0.243056	144
but	0.157895	38
went	0.117647	34
.	0.047222	720
would	0.045455	22
was	0.043478	184
to	0.035032	314
decided	0.017544	57
had	0.014706	68
a	0.006623	151
the	0.003788	264

3.2 20 Words with Lowest Accuracy Rate

Below are 20 words of the greatest number of occurrences which the model never manages to learn. As we define accuracy in a very strict sense: the predicted word needs to occur at the exact same place in the sentence as the ground truth word, there may be case that the model predicts a correct instance but places it at the wrong position in the sentence. Therefore, it is not very surprising that the model experiences difficulty learning possessive forms, prepositions, punctuation and ‘ ’s’, ‘ ’t’.

Word	Accuracy Rate	Occurrences
her	1.000000	162
his	1.000000	131
and	1.000000	127
,	1.000000	106
it	1.000000	91
for	1.000000	68
in	1.000000	67
of	1.000000	57
on	1.000000	52
one	1.000000	45
day	1.000000	45
him	1.000000	44
got	1.000000	41
they	1.000000	40
't	1.000000	39
at	1.000000	36
's	1.000000	34
when	1.000000	33
with	1.000000	31
up	1.000000	30

3.3 Sample Predictions Compared to Ground Truth

Below are some sentence highlights from the test set, the predictions by the model, and the ground truths.

Test First Sentence	Predicted Second Sentence	Ground Truth
<s> bob had to make dinner for his family . </s>	<s> he decided to eat another instead . </s>	<s> he hated to cook . </s>
<s> sue moved away from home for college . </s>	<s> she convinced her mother did so but he wanted to come . </s>	<s> she had never lived away from home before .
<s> bob and sue had been married for many years . </s>	<s> bob was very happy . </s>	<s> he kept cheating on her </s>
<s> bob was eating dinner . </s>	<s> he wanted to eat but unfortunately . </s>	<s> he dropped his phone </s>
<s> at first sue told her coworkers that she was dating . </s>	<s> she decided to go instead . </s>	<s> then she convinced them that she was getting married </s>

Model Behavior: The first noteworthy thing is that the model has learned pronouns pretty well. From the design of the training algorithm, it seems to have picked up on the context c_t of the first sentence. Because of the long-term memory mechanism in LSTM, even if the names are not direct precedents of the pronouns to be predicted, the model retains this memory and uses it in its prediction. Another evidence that the model seems to have learned to use context is that it associates food-related words such as 'dinner', 'eating' with the verb 'eat'. Last but not least, the model also follows a certain grammatical structure; in most cases, it seems to place verbs after nouns, and recognizes 'decided to' and 'wanted to' as phrases.
(Although the prediction is quite a poor match for the actual ground truth sentence, we can still make some sense of the model's behavior. Natural language is 'messy', and the model is not very good at capturing context or logic shifts between the first and the second sentences.)

3.4 Interesting Sentence Highlights

The following test sentences are my own, and I designed them with no ground truth in mind; I include them here as they provide some interesting insights about the model's learning behavior.

Input First Sentence	**Predicted Second Sentence**
<s> sue turned to bob . </s>	<s> bob was very happy . </s>
<s> the pizza man is called bob . </s>	<s> bob was afraid . </s>
<s> bob came to his old school . </s>	<s> he wanted to come come . </s>
<s> sue went to visit her friends . </s>	<s> she realized when she turned come . </s>
<s> bob went to visit his friends . </s>	<s> he was so glad to but though . </s>
<s> sue wasn 't angry . </s>	<s> she luckily . </s>
<s> bob wasn 't angry . </s>	<s> he went to the store to come . </s>

Model Behavior: We again observe some interesting logical transitions from the first to the second sentence the model seems to have learned, i.e. 'friends' and 'glad'. Furthermore, from the last four pairs, it is quite interesting how the model learns to distinguish between 'sue' and 'bob' and outputs different pronouns. However, this also points to flaws in this model, since the only thing that changes is the name, the hidden and context will not change too much; the ideal language model should only swap the correct pronoun. (I assume this is possible only if we have various pairs of training sample, varying only the gender.)

4 Comment

4.1 Existing Model Optimizations

Per Steven's suggestions, I initialized the weights of the linear transformations in the **LSTM-Cell** class from a uniform distribution, and the biases to zeros. This may speed up convergence during training.
During the actual training, I used a **scheduler** which decays my learning rate by a factor of 0.5 every training epoch. This may reduce overfitting to some extent.

$$scheduler = optim.lr\_scheduler.StepLR(opt, step_size = 1, gamma = 0.5)$$

4.2 Possible Model Optimizations to be Performed

There are certainly many optimization techniques that will greatly boost the model's performance. For example, using one-hot vector encoding, softmax and cross-entropy loss may result in better accuracy. Furthermore, the choice of parameters such as the learning rate is crucial and worth closer examination.