HOMEWORK 3

CMU 10-707: DEEP LEARNING (FALL 2017)
https://piazza.com/cmu/fall2017/10707
OUT: Nov 1
DUE: Nov 15
TAs: Dimitris Konomis, Dheeraj Rajagopal, Yao-Hung (Hubert) Tsai

1 Problem 1 (20 pts)

Assume a network that computes a 4-gram language model[1], which computes $P(w_i|w_{i-1}w_{i-2}w_{i-3})$ as shown in figure 2. Consider a dataset of natural language sentences with a vocabulary size V. The network contains H hidden units in the hidden layer and each word w in the input X is of the embedding dimension D. Specifically, your embedding layer will have $D*$number_of_words (which is three for this specific question) for this language model. In addition to the architecture in the figure, assume that the hidden layer is followed by a tanh non-linearity layer.
Assuming a softmax layer at the end with a cross-entropy loss for prediction, derive the backpropagation equations for the loss, with respect to the weights and biases shown in the figure (word embedding weights, embedding to hidden weights, hidden to output weights and their corresponding biases). While deriving, clearly mention the dimensions of each weight matrix and the bias terms.

My Answer Here

Denote word embedding weights as $\mathbf{w}^{(1)}$ (dimension: 3D vector); embedding to hidden weights as $\mathbf{w}^{(2)}$(dimension: H rows, 3D columns), bias as $\mathbf{b}^{(2)}$ (dimension: size H vector); hidden to output weights as $\mathbf{w}^{(3)}$ (dimension: V rows, H columns), bias as $\mathbf{b}^{(3)}$ (dimension: size V vector).
In forward propagation:

$$\mathbf{h}^{(1)} = \mathbf{w}^{(1)}$$

$$\mathbf{a}^{(2)} = \mathbf{b}^{(2)} + \mathbf{w}^{(2)}\mathbf{h}^{(1)}$$
$$\mathbf{h}^{(2)} = tanh(\mathbf{a}^{(2)})$$
$$\mathbf{a}^{(3)} = \mathbf{b}^{(3)} + \mathbf{w}^{(3)}\mathbf{h}^{(2)}$$

$$f = softmax(\mathbf{a}^{(3)})$$

In backward propagation:

$$l = log(f)_y$$

$$\nabla_{\mathbf{a}^{(3)}} l = -(e(y) - f)$$

$$\nabla_{\mathbf{w}^{(3)}} l = (\nabla_{\mathbf{a}^{(3)}} l)(\mathbf{h}^{(2)})^T$$

$$\nabla_{\mathbf{b}^{(3)}} l = \nabla_{\mathbf{a}^{(3)}} l$$

[1]https://en.wikipedia.org/wiki/N-gram

$$\nabla_{\mathbf{h}^{(2)}} l = ((\mathbf{w}^{(3)})^T \nabla_{\mathbf{a}^{(3)}} l)$$

$$\nabla_{\mathbf{a}^{(2)}} l = (\nabla_{\mathbf{h}^{(2)}} l) \odot [..., tanh^{'}(\mathbf{a}_j^{(2)}), ...]$$

$$\nabla_{\mathbf{w}^{(2)}} l = (\nabla_{\mathbf{a}^{(2)}} l)(\mathbf{h}^{(1)}))^T$$

$$\nabla_{\mathbf{b}^{(2)}} l = \nabla_{\mathbf{a}^{(2)}} l$$

$$\nabla_{\mathbf{w}^{(1)}} l = ((\mathbf{w}^{(2)})^T \nabla_{\mathbf{a}^{(2)}} l)$$

2 Problem 2 (20 pts)

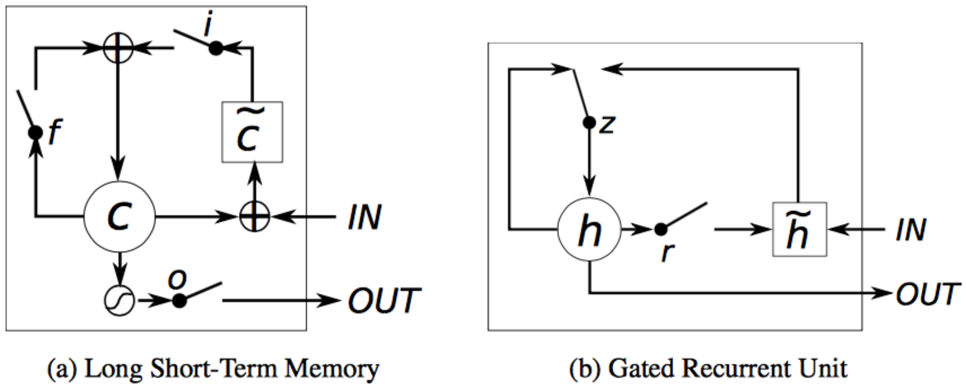


Figure 1: Illustration of (a) LSTM and (b) GRU.

Gated Recurrent Unit (GRU) is a gating mechanism similar to Long Short-term Memory (LSTM) with similar performance.
LSTM has the following equations:

$$\begin{aligned}
\mathbf{f}_t &= \sigma\left(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + b_f\right) \\
\mathbf{i}_t &= \sigma\left(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + b_i\right) \\
\mathbf{o}_t &= \sigma\left(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + b_o\right) \\
\tilde{\mathbf{c}}_t &= \tanh\left(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1} + b_c\right) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh\left(\mathbf{c}_t\right),
\end{aligned}$$

where $\odot$ is element-wise product. Note that these equations are different from the ones in the course slides. In the slides, the *peephole* LSTM is introduced, whereas we use a simpler version of the LSTM. For details see [1].
On the other hand, GRU has the following equations:

$$\begin{aligned}
\mathbf{z}_t &= \sigma\left(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1} + b_z\right) \\
\mathbf{r}_t &= \sigma\left(W_r \mathbf{x}_t + U_r \mathbf{h}_{t-1} + b_r\right) \\
\tilde{\mathbf{h}}_t &= \tanh\left(W \mathbf{x}_t + U(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + b\right) \\
\mathbf{h}_t &= \mathbf{z}_t \odot \tilde{\mathbf{h}}_t + (\mathbf{1} - \mathbf{z}_t) \odot \mathbf{h}_{t-1}.
\end{aligned}$$

Fig. 1 illustrations the architecture for LSTM and GRU. Please briefly answer the following questions:

(a) How many gates are there in LSTM and GRU, respectively? Please also specify the names of gates in LSTM/GRU.

According to the sequnce of the above equations, a LSTM has forget gate, input gate,and output gate, in total 3 gates. A GRU has update gate and a reset gate, in total 2 gates.

(b) Summarize in your own words, the functions of the gates in the GRU and LSTM, and then compare between them.

In LSTM, forget gate is used to decide how much it would like to retain or forget about the previous cell state value according to the previous hidden state and current input x. The input gate decides how much, and which value we would like to update according to the previous hidden state and current input. The output gate servers as a filter so we only output the part we want.
In GRU, update gate basically combines the input gate and forget get in a LSTM, so it will decide how much previous state we want to keep. And reset get decides how we want to combine current state with previous state.

(c) In terms of outputs ("OUT" in Fig. 1 (a) and (b)), what are the differences between LSTM and GRU?

A LSTM has an internal memory $\mathbf{c}_t$ that is different from the exposed hidden state $\mathbf{h}_t$. However, a GRU does not have that, it only has one output state as it does not have the output gate.

(d) Suppose $\mathbf{x} \in \mathbb{R}^m$ and $\mathbf{h} \in \mathbb{R}^n$ ($\mathbf{c} \in \mathbb{R}^n$), please compute the numbers of parameters in each LSTM and GRU unit, respectively.

For LSTM, it has a set of $\mathbf{W}$ and $\mathbf{U}$ for each of the 3 gates and for computing $\tilde{\mathbf{c}_t}$, and the bias term as well. Therefore the size of the parameters is

$$4 * (\text{num\_of\_output} * \text{num\_of\_input} + \text{num\_of\_output} * \text{num\_of\_outpu} + \text{num\_of\_output}) = 4(mn + n^2 + n)$$

For GRU it only has 2 gates and an additional one sets of parameters for calculating $\tilde{\mathbf{h}_t}$. Therefore, the size of the parameters is

$$3 * (\text{num\_of\_output} * \text{num\_of\_input} + \text{num\_of\_output} * \text{num\_of\_outpu} + \text{num\_of\_output}) = 3(mn + n^2 + n)$$

(e) Suppose you have two networks consisting of a sequence of LSTM or GRU cells (with the same number of cells). Which networks might take less time to train and generalize? Please also explain why in a few sentences.

GRU would take less time to train and generalize, as the number of parameters is only 3/4 of that of LSTM, so the computation is much faster.

3 Problem 3 (60 pts)

In this problem, you are going to build a 4-gram language model using a Multilayer Perceptron with an embedding layer as in figure 2. For each of the 4 word combinations, you will predict the next word given the first three words. Please do not use any toolboxes. We recommend you use Matlab or Python, but you are welcome to use any programming language of your choice

3.1 Data Preprocessing (10 pts)

In this question, you will learn to preprocess the data. The dataset is given in the files *train.txt* and *val.txt*. Each of the files contain one sentence per line. You are required to do the following

1. Create a vocabulary dictionary (i.e.) you are required to create an entry for every word in the training set (make the data lower-cased). This will serve as a lookup table for the words and their corresponding id. Note: Splitting it on space should do, there is no need for any additional processing.

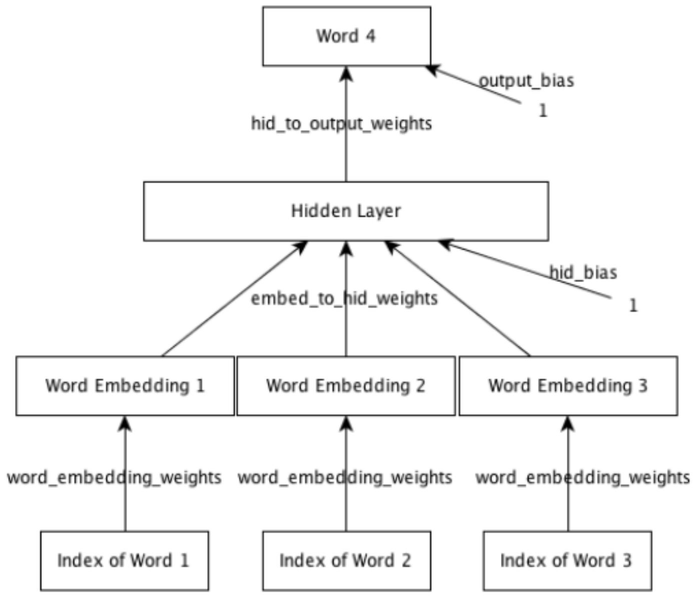


Figure 2: Language Model Architecture

2. For each sentence, include a START tag (before the sentence) and END tag (after the sentence).
3. How many trainable parameters are in the network including the weights and biases ?

   Let size of vocabulary be V, let the dimension of embedding be D, let size of hidden layer be H.

   For word embedding there are $V \times D = VD$ parameters.

   For hidden layer there are $H \times D \times 3 = 3HD$ parameters for the weights, and H parameters for the bias

   For the output layer, there are $V \times H = VH$ of parameters for weights and V parameters for bias.

   Therefore in total $VD + 3HD + H + VH + V$ number of parameters are trainable.

For language models, you will often encounter the problem of finding new words in the test/validation set. You should add an additional token 'UNK' in the vocabulary to accommodate this. This is also useful when you truncate your vocabulary size to avoid long-tail distributions. Everytime a word not from the truncated vocabulary appears, use 'UNK' instead.'
For this homework problem, limit your vocabulary size to 8000 (including the 'UNK', 'START' and 'END'). Plot the distribution of 4-grams and their counts. Report the most common 4-grams (top 50) and your observations from the distribution graph.

My Answer here

Figure 3 shows the distribution of the 4-grams. The horizontal axis is the rank of the 4-gram according to their count, and the vertical axis is the count of the 4-gram. The distribution shows that most of the 4-grams appeared only once, and only very few appeared more than 10 times. I would think this is reasonable, as if any 4-grams have a very high frequency, it may have been made into acronyms or a new word will be invented for it.

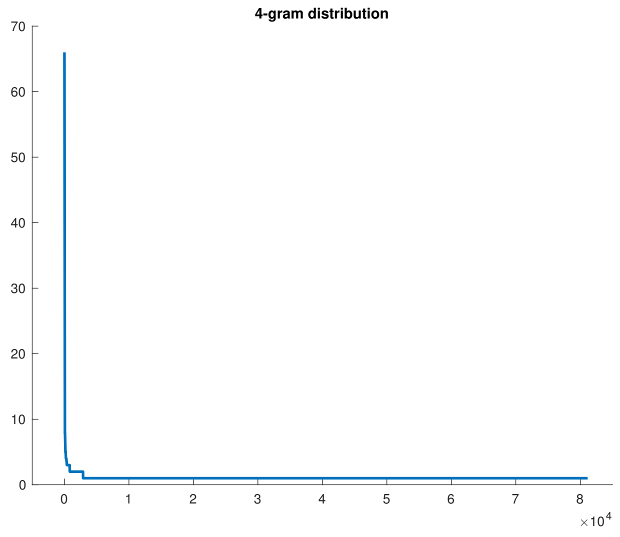


Figure 3: 4-gram distributon

Below are the top 50 4-grams I found in my data set. Notice that this will not be the only answer as it is greatly affected by how I choose the top 8000 words.

```
0 *t*-1 . <end>
$ <unk> million *u*
said 0 *t*-1 .
million *u* . <end>
, '' says *t*-1
million *u* , or
0 *t*-2 . <end>
a share . <end>
the company said 0
, '' said *t*-1
said 0 *t*-2 .
new york stock exchange
$ <unk> billion *u*
*u* a share ,
billion *u* . <end>
$ <unk> *u* a
said *t*-1 . <end>
<unk> *u* a share
said 0 it will
the new york stock
, the company said
new york . <end>
and chief executive officer
*u* , or $
says *t*-1 . <end>
cents a share .
says 0 *t*-1 ,
, for example ,
, '' he said
```

```
in september . <end>
president and chief executive
cents a share ,
$ <unk> *u* ,
, said 0 the
, or $ <unk>
<unk> million *u* ,
company said 0 it
the u.s. . <end>
or $ <unk> *u*
<start> in addition ,
<start> the company said
0 *t*-1 , the
corp. said 0 it
year earlier . <end>
<unk> . '' <end>
0 *t*-3 . <end>
million *u* from $
, he said 0
said 0 *t*-1 ,
*u* a share .
```

3.2 Backpropagation with Linear Hidden Layer (20 pts)

You will now implement the model shown in figure 2 with the linear hidden layer. Each word will be represented by a 16-dimensional embedding followed by hidden layer of 128 units. Your network's output will be the softmax over the vocabulary which tries to predict the next word. You are going to minimize cross entropy loss between the predicted softmax and next word.

For each epoch, plot your perplexity on the validation set (val.txt). Run your model for 100 epochs. Report your observation with graph of perplexity [2] and total loss.

Also, report the same observations on the same network with 256 and 512 hidden layer units. Note: When computing perplexity of the validation set, you will often encounter words that you have not seen in the training set.

My Answer Here:

Figure 4 shows the combined cross entropy and Figure 5 shows the combined perplexity for linear hidden layer with 128, 256, and 512 hidden units. For all 3 cases the training loss is constantly decreasing as more more epochs are run. However, the validation loss and the perplexity reaches a minimum value and start to increase. This shows that the model is able to over fit the data set with less than 100 epochs.

Comparing the cross-entropy results and perplexity results of different hidden layer size, it can be seen that the larger the hidden layer, the lower the training loss will be. However, the validation loss and perplexity is higher with more hidden units. This shows that the model complexity is of 128 hidden units is already enough for the given data set, and size of 256 and 512 will only starts to over fit the data set and does not generalize well.

[2] http://www.cs.columbia.edu/ mcollins/lm-spring2013.pdf

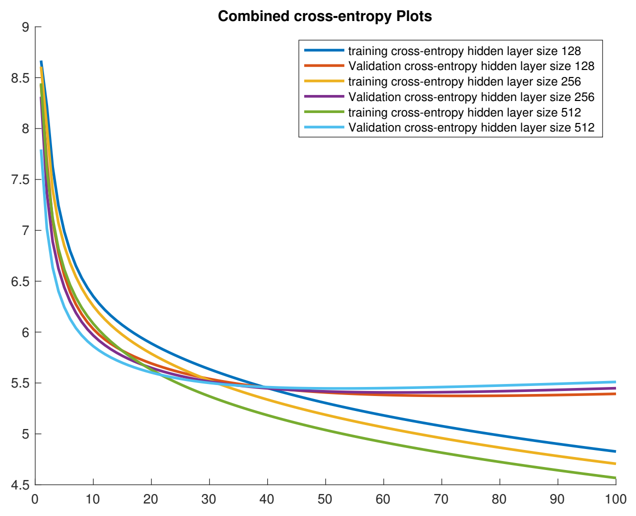
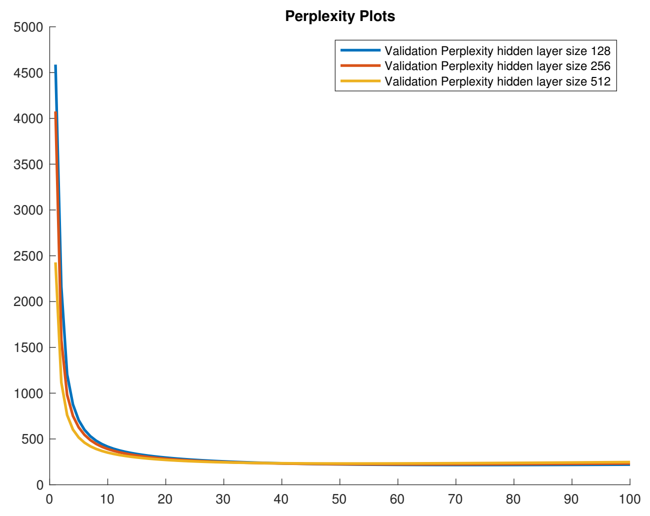
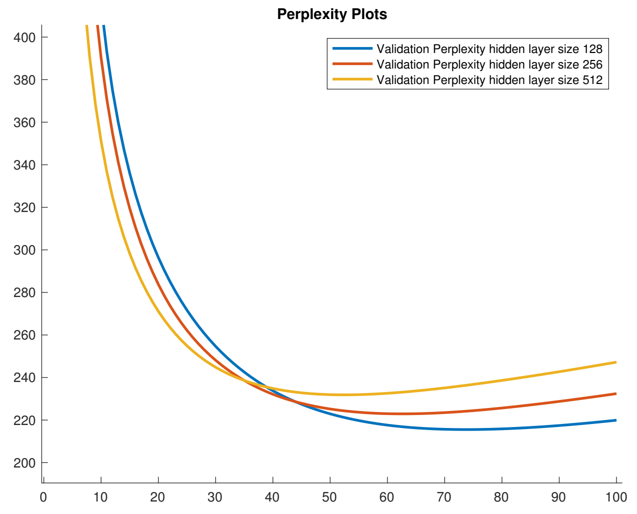


Figure 4: Combined Cross-Entropy with Linear Hidden Layer



(a) Overall View



(b) Zoomed View

Figure 5: Combined Perplexity with Linear Hidden Layer

3.3 Incorporating Non-linearity (10 pts)

Now, you will be adding non-linear activations after the hidden layer. Include tanh activations for the hidden layer output. Do you see any changes in the trend of loss and perplexity (compared to network without non-linearity)? Describe your findings in a few sentences and show appropriate plots (after running at least 100 epochs).

My Answer Here:

Figure 6 shows the cross-entropy and figure 7 shows the perplexity with tanh activation function in the hidden layer. Comparing to the linear hidden layer result, this model turn to result in lower perplexity, and the result does not show the sign of over-fitting in 100 epochs.

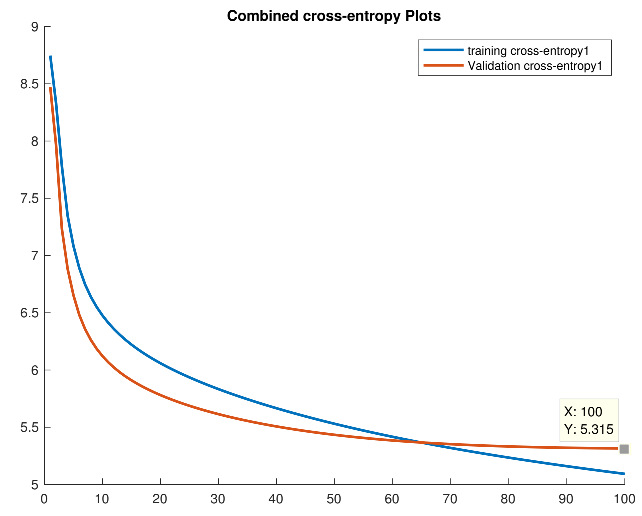


Figure 6: Cross-Entropy with tanh Hidden Layer

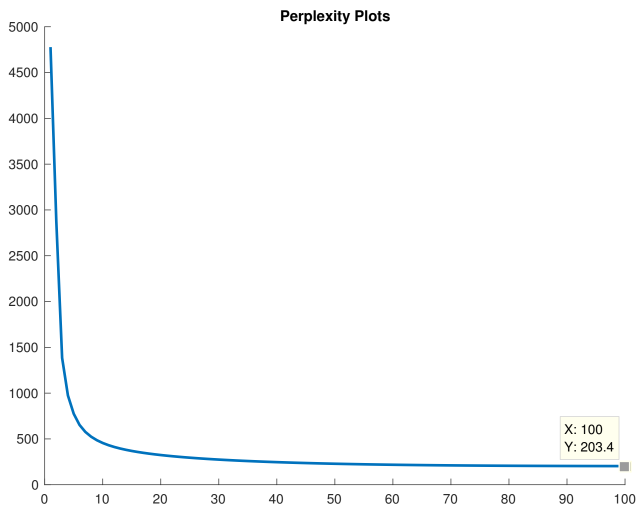


Figure 7: Perplexity with tanh Hidden Layer

3.4 Analysis (10 pts)

Language Generation: Pick three words from the vocabulary that go well together (for example,government of united, city of new, life in the, he is the etc.). Use the model to predict the next ten words (or stop at the END token). What do you notice as you generate more words ? Report your findings for five such ‘three word’ combinations.

My Answer Here:

Below are five such ’three words’ that I have observed. As the model continue to generate more words, it will get close to the common phrases that we hear everyday. For example, ’the study shows’ will be followed by ’that the <unk >of the <unk >. <end >is a very common structure observed in English, also for the case of ’at the end’.

```
the united states -> , '' says *t*-1 the <unk> of the <unk> .
city of new -> york , a <unk> of the <unk> . <end>
the study shows -> that the <unk> of the <unk> . <end>
the company has -> been a <unk> of the <unk> . <end>
at the end -> of the <unk> . <end>
```

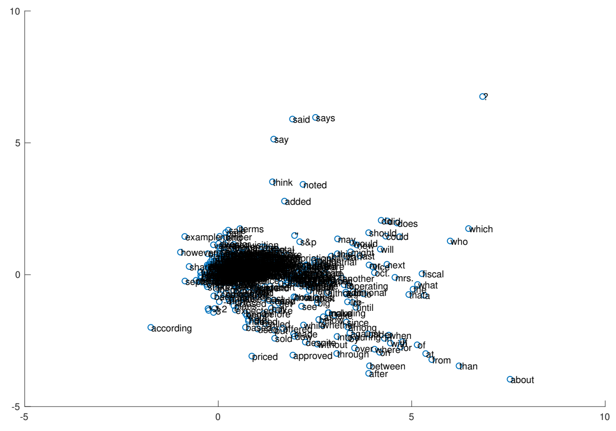
Implement a function that computes the distance between two embedding. Your function should compute the euclidean distance between the word vectors. Does your language model learn that the distances between similar words are smaller ? Report your findings.
As it can be seen from the below word pairs, more similar words have smaller distance, and unrelated words have larger distance.

```
city , town: 2.489170
business , company: 2.462060
administration , government: 1.985725
car , vehicle: 1.688506
big , city: 5.344898
no , yes: 3.462495
university , college: 2.054882
```

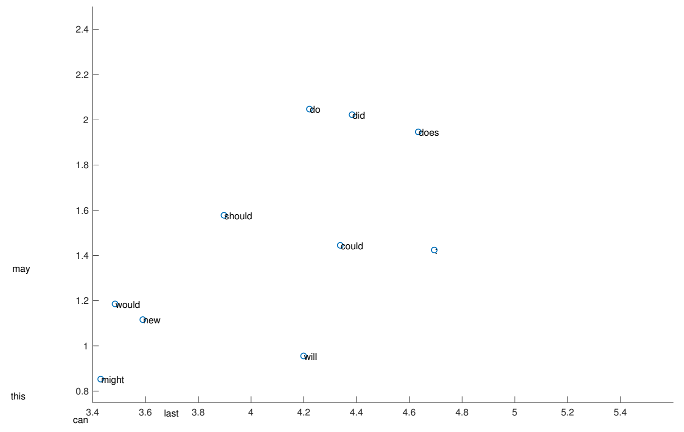
3.5 Embedding Layer Visualization (10 pts)

Reduce the embedding dimension to 2. Retrain the whole network. Visualize the learnt embeddings for 500 randomly picked words from the vocabulary and plot in a 2-D graph. What do you observe? Do similar words appear nearby in the plot? Report your findings with the plot.

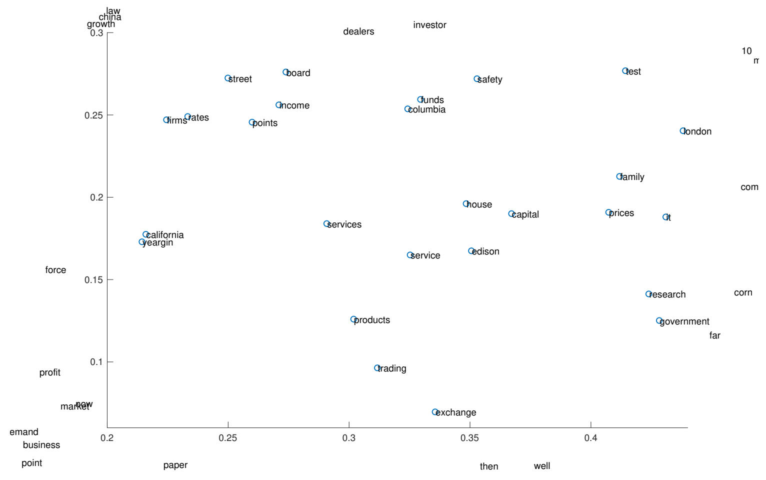
Figure 8 shows the word embedding with a dimension of 2. In figure 8 (a) we can see that ’say’, ’said’, ’says’ are close together. In figure 8 (b) which is a zoomed view of (a) we can see that ’do’, ’did’, ’does’ appeared to be together, and ’should’, ’cloud’, ’will’, and ’would’ appears to be close. If we further zoom in the mass of data points in (a) we can see a plot like (c) which shows that some similar words are together, such as ’service’ and ’services’, ’trading’ and ’exchange’. However, close words does not necessarily have similar meanings.



(a) Overall View



(b) Zoomed View 1



(c) Zoomed View 2

Figure 8: Word Embedding plot

3.6 Extra Points (20 pts)

For extra points, you will implement the same language model but this time with a Recurrent Neural Network. This takes care of the order in which the words are given as input to the model.
Note: For this extra points question, you are allowed to use a deep learning package of your choice (PyTorch, TensorFlow, Theano, etc.)

3.6.1 Recurrent Neural Network

Each of the input will now go to an RNN cell that processes one word at a time and computes probabilities of the possible values for the 4-gram(time step = 4).
The input to this network should be batched with a batch size of 16.
The architecture starting from the embedding layer, hidden layer (with tanh activation) and softmax remains the same as the architecture in 2 with hidden layer size = 128 and embedding size = 16.

3.6.2 Embedding layer

Try the following sizes for embedding dimension [32, 64, 128]. Report your findings based on cross-entropy error and perplexity of validation set.

3.6.3 Truncated Backpropagation

It is a usual practice to truncate backpropagation in RNNs [2] for long sequences. Now, you will try truncating the backprop(to 2 words) for randomly selected 10% of words. Plot the error and perplexity and report your findings.

Write up

Hand in answers to all questions above. For Problem 3, the goal of your write-up is to document the experiments you have done and your main findings, so be sure to explain the results. Be concise and to the point – do not write long paragraphs or only vaguely explain results.

- The answers to all questions should be in pdf form (please use LaTeX).
- Please include a README file with instructions on how to execute your code.
- Package your code and README document using `zip` or `tar.gz` in a file called `10707-A3-*yourandrewid*.[zip|tar.gz]`.
- Submit your PDF write-up to the Gradescope assignment "Homework 3" and your packaged code to the Gradescope assignment "Code for Homework 3."

References

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[2] Ronald J. Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, 2:490–501, 1990.