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 3. 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.

Your Answer

Forward process

(1) Input to Word Embedding Layer: This Layer we use lookup table to select weights of 3 words. And reshape the three weights to a matrix with the dimension $[3 * D, 1]$, which we will represented as: $z_{embedding}$

(2) Word Embedding Layer to Hidden Layer: This layer we use a weight matrix with the dimension $[H, 3 * D]$, and a bias matrix with dimension $[H, 1]$. And the forward process can be represented as:

$$a_{hidden} = W_{hidden} * z_{embedding} + b_{hidden}$$
$$z_{hidden} = tanh(a_{hidden})$$

(3) Hidden Layer to Output Layer: This layer we use a weight matrix with the dimension $[V, H]$ and a bias matrix with dimension $[V, 1]$. And the forward process can be represented as:

$$a_{output} = W_{output} * z_{hidden} + b_{output}$$
$$z_{output} = softmax(a_{output})$$

Backpropagation Process

(1) The error of the output layer:
The ΔW_{output} can be derived as:

$$\begin{aligned}\Delta W_{output} &= \frac{\partial Loss}{\partial W_{output}} \\ &= \frac{\partial Loss}{\partial z_{output}} \cdot \frac{\partial z_{output}}{\partial a_{output}} \cdot \frac{\partial a_{output}}{\partial W_{output}}\end{aligned}$$

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

The Δb_{output} can be derived as:

$$\begin{aligned}\Delta b_{output} &= \frac{\partial Loss}{\partial b_{output}} \\ &= \frac{\partial Loss}{\partial z_{output}} \cdot \frac{\partial z_{output}}{\partial a_{output}} \cdot \frac{\partial a_{output}}{\partial b_{output}}\end{aligned}$$

Here, we define that:

$$\begin{aligned}\delta_{output} &= \frac{\partial Loss}{\partial z_{output}} \cdot \frac{\partial z_{output}}{\partial a_{output}} \\ &= target - z_{output}\end{aligned}$$

Then we can get that ΔW_{output} is :

$$\begin{aligned}\Delta W_{output} &= \delta_{output} \cdot \frac{\partial a_{output}}{\partial W_{output}} \\ &= \delta_{output} \cdot z'_{hidden}\end{aligned}$$

Then we can get that Δb_{output} is :

$$\Delta b_{output} = \delta_{output} \cdot \frac{\partial a_{output}}{\partial b_{output}} \quad = \delta_{output}$$

(2) The error of the hidden layer:
The ΔW_{hidden} can be derived as:

$$\Delta W_{hidden} = \frac{\partial Loss}{\partial W_{hidden}} \quad = \frac{\partial Loss}{\partial z_{output}} \cdot \frac{\partial z_{output}}{\partial a_{output}} \cdot \frac{\partial a_{output}}{\partial z_{hidden}} \cdot \frac{\partial z_{hidden}}{\partial a_{hidden}} \cdot \frac{\partial a_{hidden}}{\partial W_{hidden}}$$

The Δb_{hidden} can be derived as:

$$\Delta b_{hidden} = \frac{\partial Loss}{\partial b_{hidden}} \quad = \frac{\partial Loss}{\partial z_{output}} \cdot \frac{\partial z_{output}}{\partial a_{output}} \cdot \frac{\partial a_{output}}{\partial z_{hidden}} \cdot \frac{\partial z_{hidden}}{\partial a_{hidden}} \cdot \frac{\partial a_{hidden}}{\partial b_{hidden}}$$

And here we define that:

$$\begin{aligned}\delta_{hidden} &= \delta_{output} \cdot \frac{\partial a_{output}}{\partial z_{hidden}} \cdot \frac{\partial z_{hidden}}{\partial a_{hidden}} \\ &= W'_{output} \cdot \delta_{output} \odot (1 - z^2_{hidden})\end{aligned} \tag{1}$$

Then we can get ΔW_{hidden} is:

$$\begin{aligned}\Delta W_{hidden} &= \delta_{hidden} \cdot \frac{\partial a_{hidden}}{\partial W_{hidden}} \\ &= \delta_{hidden} \cdot z'_{embedding}\end{aligned} \tag{2}$$

And we can get that Δb_{hidden} is:

$$\begin{aligned}\Delta b_{hidden} &= \delta_{hidden} \cdot \frac{\partial a_{hidden}}{\partial b_{hidden}} \\ &= \delta_{hidden}\end{aligned} \tag{3}$$

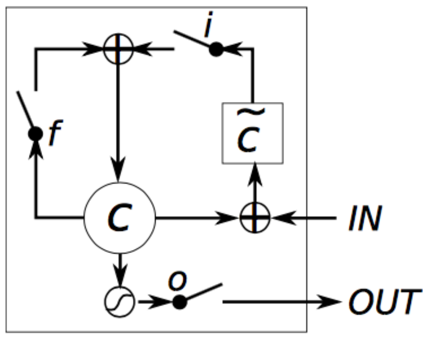
(3) The error of the embedding layer:
The derivation of $\Delta W_{embedding}$ is very similar to the derivation we have done, so we can directly define that:

$$\delta_{embedding} = W'_{hidden} \cdot \delta_{hidden} \tag{4}$$

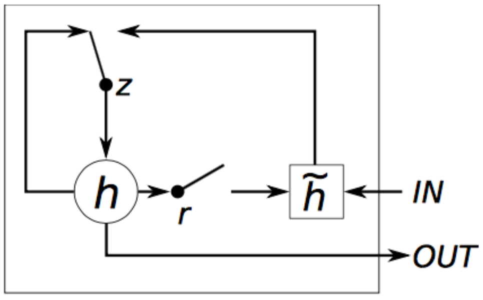
And we only need to update the rows in the $W_{embedding}$ corresponding the three input words with the $\Delta W_{embedding}$, which can be calculated as:

$$\Delta W_{embedding} = \delta_{embedding} \tag{5}$$

2 Problem 2 (20 pts)



(a) Long Short-Term Memory



(b) Gated Recurrent Unit

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:

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

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 [**?**].
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.

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

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

(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.

(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.

Your Answer

(a) There are 3 gates in LSTM. And the name of the three gates are: Input gates, Forget gates and output gates respectively.
There are 2 gates in GRU. And the name of the two gates are: Update gate, Reset gate.

(b) LSTM:

- Input gate: Input gate is a sigmoid layer which decides what values the network will update. It gates the connection between the input unit and the memory cell. Here, input unit isa a $tanh$ layer which receives inputs from other blocks in the network.
- Forget gate: Forget gate controls how much information should the memory cell remember from previous. It computes a linear function of its inputs, followed by a activation function(usually sigmoid). If it is on, i.e the value of forget gate is 1?the memory cell remembers its previous value. If it is off, ie the value is 0, the memory cell forgets it.
- Output gate: The output gate is a sigmoid layer that decides what parts of the memory cell state the block is going to output. Here, the memory cell states are the output values of the memory cell after passing through a $tanh$ function.

GRU:

- Update gate: It defines how much of the previous memory to keep around and used as a coefficient in the final output. It use a sigmoid function.
- Reset gate: It is a sigmoid layer that determines how to combine the new input with the previous memory. The hidden layer outputs will be added a tuned previous hidden states. And previous units are tuned by element-wise multiplied by Reset gate.

Difference between gates:

- For LSTM, the function to remember previous state and getting new information from current state is controlled by forget gate and inout gate respectively. But the similar function in GRU is done by the update gate. And the reset gate in GRU is applied directly to the previous hidden states. But in LSTM, the tuning functions are applied to memory cell states.
- Compared to LSTM, GRU doesn't apply a $tahn$ function to the input, and GRU also doesn't have a memory cell.

(c) Differences between LSTM and GRU in terms of outputs:

- Compared to LSTM, GRU don't have the memory cell and there's no output gate to tune the output from the whole block.
- Compared to LSTM, GRU doesn't apply a $tanh$ function to the output. But LSTM use a $tanh$ to squash the output value from the memory cell to the range 0 1.

(d) Suppose input $\mathbf{x}$ is a matrix that has m rows and 1 columns. And suppose $\mathbf{h}$ is a matrix that has n rows and 1 columns.
LSTM:

- Forget gate:
W_f will be a matrix that has the dimension $[n, m]$. U_f will be a matrix that has the dimension $[n, n]$. b_f will be a matrix that has the dimension $[n, 1]$.
So, the number of parameters of forget gate is: $n * (m + n + 1)$.
- Input gate:
W_i will be a matrix that has the dimension $[n, m]$. U_i will be a matrix that has the dimension $[n, n]$. b_i will be a matrix that has the dimension $[n, 1]$.
So, the number of parameters of input gate is: $n * (m + n + 1)$.

- Output gate:
  W_o will be a matrix that has the dimension $[n, m]$. U_i will be a matrix that has the dimension $[n, n]$. b_i will be a matrix that has the dimension $[n, 1]$.
  So, the number of parameters of output gate is: $n * (m + n + 1)$.
- Memory cell:
  W_c will be a matrix that has the dimension $[n, m]$. U_c will be a matrix that has the dimension $[n, n]$. b_i will be a matrix that has the dimension $[n, 1]$.
  So, the number of parameters of memory cell is: $n * (m + n + 1)$. W_c will be a matrix that has the dimension

And for LSTM, there are $4 * n * (m + n + 1)$ parameters for one unit.

GRU:

- Update gate:
  W_z will be a matrix that has the dimension $[n, m]$. U_z will be a matrix that has the dimension $[n, n]$. b_z will be a matrix that has the dimension $[n, 1]$. So, the number of parameters of update gate is: $n * (m + n + 1)$.
- Reset gate: W_r will be a matrix that has the dimension $[n, m]$. U_r will be a matrix that has the dimension $[n, n]$. b_r will be a matrix that has the dimension $[n, 1]$. So, the number of parameters of reset gate is: $n * (m + n + 1)$.
- $\tilde{\mathbf{h}}_t$ layer: $W_{\tilde{\mathbf{h}}_t}$ will be a matrix that has the dimension $[n, m]$. $U_{\tilde{\mathbf{h}}_t}$ will be a matrix that has the dimension $[n, n]$. $b_{\tilde{\mathbf{h}}_t}$ will be a matrix that has the dimension $[n, 1]$. So, the number of parameters of $\tilde{\mathbf{h}}_t$ layer is: $n * (m + n + 1)$.

And for GRU, there are $3 * n * (m + n + 1)$ parameters for one unit.

(e) The network consisting of GRU cells may take less time to train and generalize. Because GRU unit is simpler than LSTM unit. And from the problem d we can know that GRU has less parameters than LSTM. Less parameters and simpler structure of unit will make the network 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 3. 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.
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 ?

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.'

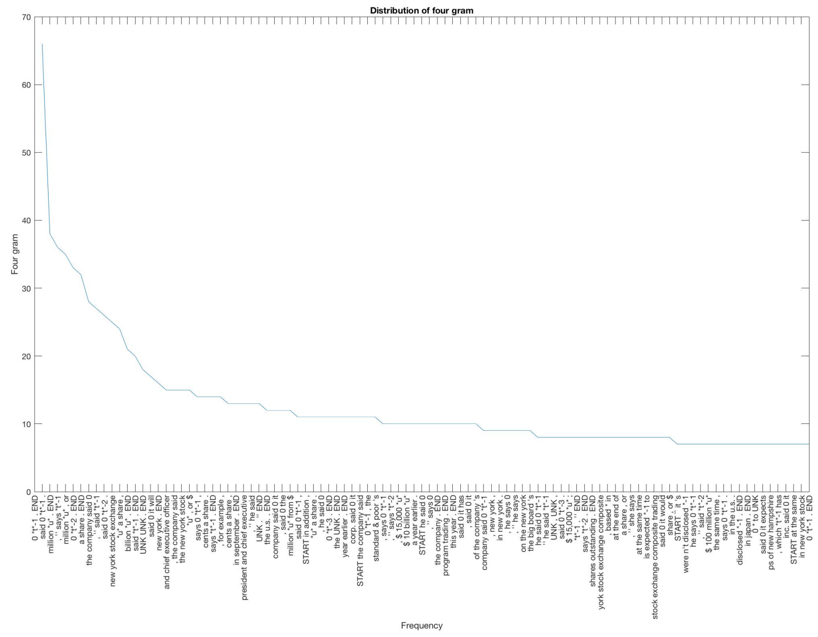


Figure 2: The distribution of 4-grams and their counts.

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.

Your Answer

1. Number of parameters in network.
   Suppose the word vector has the dimension $|D|$. The hidden layer has $|H|$ neurons. The size of vocabulary is $|V|$. And we use 4-gram as training data, which means there are 3 words used as input, and output one word.

   - From word layer to word embedding layer: There is a word distribution matrix with the dimension $[V, D]$. And this layer has $V * D$ parameters to train.
   - From word embedding layer to hidden layer: There is a weight matrix with the dimension $[3 * D, H]$. And this layer also has a bias matrix with the dimension $[H, 1]$.
     And this layer has $3 * D * H + H$ parameters to train.
   - From hidden layer to output layer: There is weight matrix with the dimension $[H, V]$ and a bias matrix that has dimension $[V, 1]$.
     And this layer has $H * V + V$ parameters to train.

   To sum up, there are $V(1 + D) + H(1 + V + 3 * D)$ parameters to train.

2. The 4-grams distribution graph is shown in figure 2. Here I plot the most frequent 100 4-grams. We can see that the frequency of the first 4-gram is much higher than the 100th 4-gram. And most 4-gram appears less than 10 times.

3. The most 50 frequent 4-grams are shown in the table 1. And we can see that the most frequent 4-gram is "0 *t*-1 . END". And we can see that there are many useless word in 4-gram like some punctuations and stop words "the", which may affect the training accuracy.

Order	4-grams	Frequency	Order	4-grams	Frequency
1	0 *t*-1 . END	66	2	said 0 *t*-1 .	38
3	million *u* . END	36	4	, " says *t*-1	35
5	million *u* , or	33	6	0 *t*-2 . END	32
7	a share . END	28	8	the company said 0	27
9	, " said *t*-1	26	10	said 0 *t*-2 .	25
11	new york stock exchange	24	12	*u* a share ,	21
13	billion *u* . END	20	14	said *t*-1 . END	18
15	UNK UNK . END	17	16	said 0 it will	16
17	new york . END	15	18	and chief executive officer	15
19	, the company said	15	20	the new york stock	15
21	*u* , or $	14	22	says 0 *t*-1 ,	14
23	cents a share .	14	24	says *t*-1 . END	14
25	, for example ,	13	26	cents a share ,	13
27	in september . END	13	28	president and chief executive	13
29	, " he said	13	30	UNK . " END	12
31	the u.s. . END	12	32	company said 0 it	12
33	, said 0 the 12	12	34	million *u* from $	11
35	said 0 *t*-1 ,	11	36	START in addition ,	11
37	*u* a share .	11	38	, he said 0	11
39	0 *t*-3 . END	11	40	the UNK . END	11
41	year earlier . END	11	42	corp. said 0 it	11
43	START the company said	11	44	0 *t*-1 , the	11
45	standard poor 's	10	46	, says 0 *t*-1	10
47	, " says *t*-2	10	48	, $ 15,000 *u*	10
49	$ 10 billion *u*	10	50	a year earlier .	10

Table 1: Most 50 frequent 4-grams

3.2 Backpropagation with Linear Hidden Layer (20 pts)

You will now implement the model shown in figure 3 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.

Your Answer

In this problem, I use learning rate 0.1. And experiment on different number of hidden units.

- Linear Hidden layer with 128 neurons. The loss of training data and validation data is shown in figure 4. We can see that the training loss is continue decreasing. But the perplexity of validation data first decrease then increase at about 30 epochs. The overfitting is very obivous.
- Linear Hidden layer with 256 neurons. The loss of training data and validation data is shown in figure 5. Similar to with 128 hidden neurons, the loss of training data keep decreasing. But there's overfitting of validation data.

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

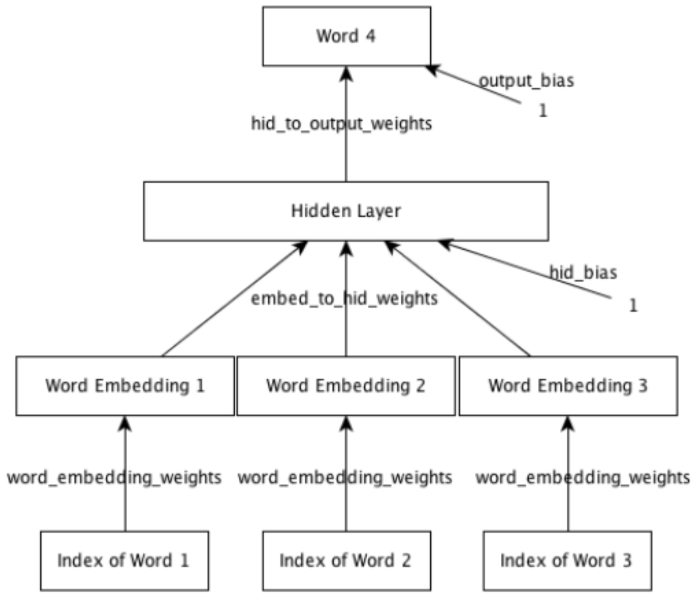
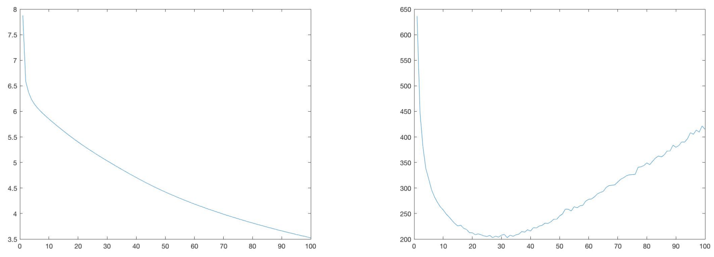


Figure 3: Language Model Architecture



(a) Training cross-entropy with 128 linear hidden neurons (b) Validation perplexity with 128 linear hidden neurons

Figure 4: Loss with 128 linear hidden neurons

And we can see that overfitting starts earlier than with 128 hidden neurons. Overfitting start at about 25 epochs.

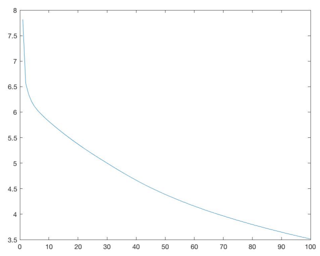
- Linear Hidden layer with 512 neurons. The loss of training data and validation data is shown in figure 6. And as we can see that, this time overfitting starts at about 20 epochs. And as I expected that, with more hidden neurons, the network is much more easier to suffer from overfitting.

3.3 Incorporating Non-linearity (10 pts)

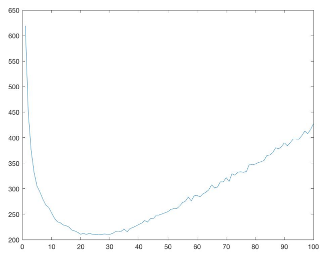
Now, you will be adding non-linear activations after the hidden layer. Include tanh activations for the hidden layer output. Do you see any changes in the trend of loss and perplexity (compared to network without non-linearity)? Describe your findings in a few sentences and show appropriate plots (after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

Your Answer

After including $tanh$ activations for the hidden layer, the training cross-entropy and validation perplexity is shown in figure 7. I still use learning rate 0.1 to do the experiment. And we can see that using non-linear activation function

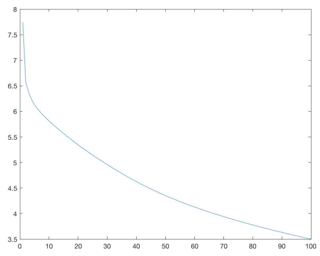


(a) Training cross-entropy with 256 linear hidden neurons

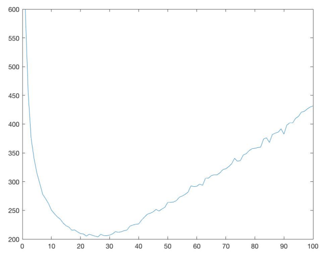


(b) Valid perplexity with 256 linear hidden neurons

Figure 5: Loss with 256 linear hidden neurons

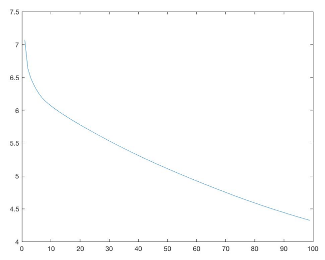


(a) Training cross-entropy with 512 linear hidden neurons



(b) Valid perplexity with 256 linear hidden neurons

Figure 6: Loss with 512 linear hidden neurons


(a) Training cross-entropy with 128 non-linear hidden neurons

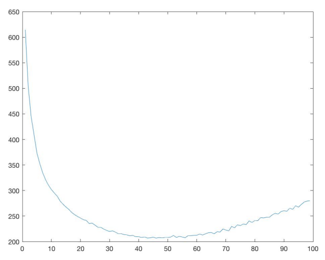

(b) Validation perplexity with 128 non-linear hidden neurons

Figure 7: Loss with 128 non-linear hidden neurons

make the network much more robust to the overfitting. And we can see from the figure 7(a), the cross-entropy is keeping going down. And perplexity starts going up at about 50 epochs which is much better than the situation in linearity.

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.

Implement a function that computes the distance between two embeddings. 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.

3.4.1 Your Anwser

Five sentences We can see that the prediction is not very reasonable. I think this is firstly because I use the weights after training 60 epochs, which may have overfitting. And secondly, I thing the training data is not very good. As we know that to generate training data, we just use 8000 words as vocabulary, which may cause a lot of "UNK" in the training data. And in training data, there are also many words that are meaningless, such as punctuations and stop words like "the", which also cause the training very difficult to give reasonable prediction. But for similar first three words, like example 1 and 2, the predictor gives similar prediction result. And for sentence that in training data, such as the 5th sentence here, the network give the prediction exactly same as the correct answer.

- (1) in new york , the UNK , " says 0 the u.s. market
- (2) city of new york, the UNK , " says 0 the u.s. market
- (3) life in the past, " says 0 the u.s. market . END
- (4) company said 0 of the UNK, " says 0 the.
- (5) shares outstanding . END

Distance of Words I choose ten pair of words to calculate the distance between each pair, which is shown in table 2. The first five pairs of words I chose are words that I think they are similar to each other. And the last five pairs of words I chose are words that I think they are very different from each other. And we can see that the distance between the first 5 pairs of words are all smaller than the last five pairs of words as we expected. And especially, we can see that the distance between word "do" and word "did" is much smaller than others', which justify that our training is successful. And it also justify that word embedding distance can represent the similarity between different words, which is very useful to be used as parameters to represent the meaning of words.

word	distance
say v.s. says	1.078025261556687
he v.s. she	1.287649125844409
do v.s. did	0.894210374377863
1988 v.s. 1990	1.066168435323900
big v.s. can	2.143830947582523
said v.s it	2.926188910836460
company v.s. do	1.778281570364671
before v.s. chief	1.831337241499255
'.' v.s. higher	3.040642655009188
an v.s. have	2.645622896464358

Table 2: Distance between words

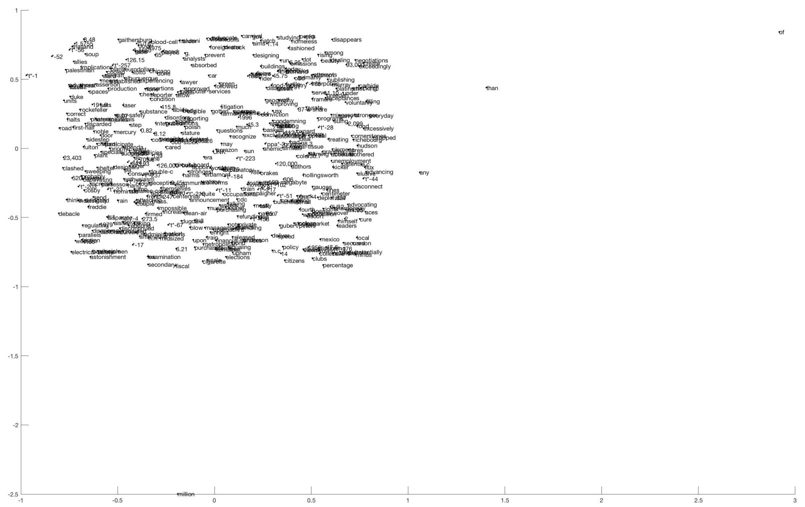


Figure 8: Visualization of word embeddings

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.

Your Answer

Figure 8 shows the visualization of the learnt word embedding for 500 randomly picked words. Here I use the weight trained by linear method, because it is faster. We can see that many common words aggregate together. And similar words like "say" and "says" are very close.

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 3 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 [**?**] 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."