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.

Answer The forward pass:

$x \in \mathbb{Z}_+^{1x3}$	[Input is a list of 3 integers, one for each word]
$e = \texttt{embed}(x)$	$\mathbb{R}^{1\times 3\times D}$
$e[i,j] = W_{\text{emb}}[x[i,j]]$	$W_{\text{emb}} \in \mathbb{R}^{V\times D}$
$a_1 = [e[:,0,:]; e[:,1,:]; e[:,2,:]]$	$\mathbb{R}^{1\times 3\cdot D}$ [Where ; denotes concatenation]
$z_1 = a_1 \cdot W^{(1)} + b^{(1)}$	$W^{(1)} \in \mathbb{R}^{3\cdot D\times H}, b^{(1)} \in \mathbb{R}^H, z_1 \in \mathbb{R}^{1\times H}$
$a_2 = \texttt{tanh}(z_1)$	$\mathbb{R}^{1xH}$
$z_2 = a_2 \cdot W^{(2)} + b^{(2)}$	$W^{(2)} \in \mathbb{R}^{H\times V}, b^{(1)} \in \mathbb{R}^V, z_2 \in \mathbb{R}^{1\times V}$
$a_3 = \texttt{softmax}(z_3)$	$\mathbb{R}^{1\times V}$
$o = \texttt{one-hot}(o)$	$\mathbb{R}^{1\times V}$ [The one hot representation of the target word]
$L = \texttt{cross-entropy}(o, a_3)$	

From the previous homework, we know that

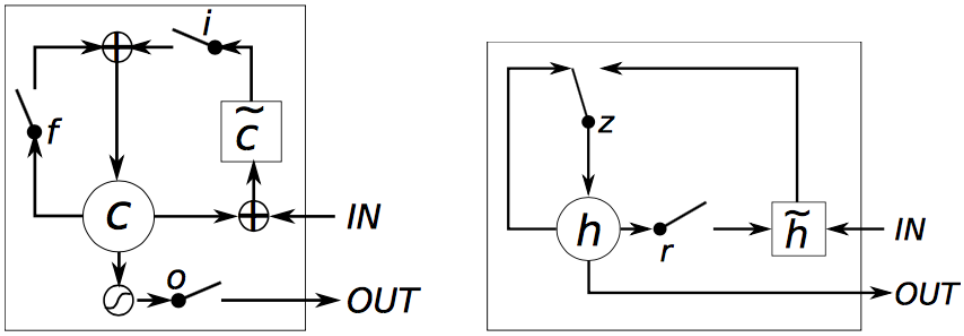
$$\frac{\partial L}{\partial z_2} = (a_3 - o) \quad \mathbb{R}^{1\times V}$$

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

The backward pass is hence defined as

$$
\begin{aligned}
\frac{\partial L}{\partial z_2} &= (a_3 - o) && \mathbb{R}^{1\times V} \\
\frac{\partial L}{\partial W^{(2)}} &= a_2^T \cdot \frac{\partial L}{\partial z_2} && \mathbb{R}^{H\times V} \\
\frac{\partial L}{\partial b^{(2)}} &= \frac{\partial L}{\partial z_2} && \mathbb{R}^{V} \\
\frac{\partial L}{\partial a_2} &= \frac{\partial L}{\partial z_2} \cdot W^{(2)T} && \mathbb{R}^{1\times H} \\
\frac{\partial L}{\partial z_1} &= \frac{\partial L}{\partial a_2} \cdot \texttt{tanh}'(z_1) && [\texttt{tanh}'(\text{x}) = 1 - \texttt{tanh}^2(\text{x})] \\
\frac{\partial L}{\partial W^{(1)}} &= a_1^T \cdot \frac{\partial L}{\partial z_1} && \mathbb{R}^{3\cdot D\times H} \\
\frac{\partial L}{\partial b^{(1)}} &= \frac{\partial L}{\partial z_1} && \mathbb{R}^{H} \\
\frac{\partial L}{\partial a_1} &= \frac{\partial L}{\partial z_1} \cdot W^{(1)T} && \mathbb{R}^{1\times 3\cdot D} \\
\frac{\partial L}{\partial e} &= \texttt{reshape}(\frac{\partial L}{\partial a_1}, [:, 3, D]) && \mathbb{R}^{1\times 3\times D} \\
\forall_{j\in\{1,2,3\}} \frac{\partial L}{\partial W_{\text{emb}}}[x[1,j]] &+= \frac{\partial L}{\partial e}[1,j] && \frac{\partial L}{\partial W_{\text{emb}}} \in \mathbb{R}^{V\times D}
\end{aligned}
$$

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:

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

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.

(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.

Answer

(a) There are 3 gates in an LSTM network (input gate i_t, forget gate f_t, output gate o_t), while there are 2 gates in a GRU network (reset gate r_t and update gates z_t)

(b) The following are the functions of the gates in both the LSTM cell and the GRU cell:

- LSTM
  - Input Gate i_t: Controls how much of new content needs to be added to the memory cell (c).
  - Forget Gate f_t: Controls how much of the previous memory cell content (c) is to be retained.
  - Output Gate o_t: Controls how much of the memory cell content should be exposed outside of the unit.
- GRU
  - Update Gate z_t: Controls what fraction of the memory update comes from the previous memory value and what fraction comes from the new content. If the value is close to 1, then the unit effectively ignores the input. Conversely, if it is close to 0, then it somewhat ignores the previous memory value (however, the new content also contains the contribution of the memory cell, controlled by the reset gate).
    Broadly, z_t has the same function as f_t for the LSTM, while i_t can be considered to be equal to $1 - z_t$, i.e z_t is obtained by tying the input gate and forget gate of the LSTM network.
  - Reset Gate r_t: Controls how much of the memory cell influences the new update content. This could be thought of being equivalent to the output gate o_t in the sense that the output gate also controls how much of the previous cell state value does the update see (note however that the output gate also monitors how much of the memory cell goes out of the LSTM cell, while the GRU has no such method, it exposes the entire memory at every timestep).

(c) The output gate controls how much of the memory cell is exposed outside the LSTM cell, which is different in the case of the GRU unit, since the GRU the output gate, and consequently exposes the contents of the entire memory unit.

(d) For the LSTM, we have $\texttt{Num-Units} = 4 \times (m \cdot n + n \cdot n + n)$, while for the GRU, we have $\texttt{Num-Units} = 3 \times (m \cdot n + n \cdot n + n)$

(e) Given that the GRU has fewer parameters than the LSTM, GRU's would train faster than LSTM's. As far as generalization goes, since GRU's have fewer parameters, they have a lower chance of over-fitting. However, since GRU's expose the entire memory state, and have tied input and forget gates, it is possible that the class of GRU's may not be able to model a complex enough distribution (i.e GRU's may under-fit). In practice, it depends heavily on the problem and datasets - in literature, people have reported LSTM's to outperform GRU's and vice versa, depending on the task.

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 ?

Answer:
Number of trainable parameters : (8000 x 16) + ((3 * 16) x 128 + 128) + (128 x 8000 + 8000) = 1166272

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 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.
Answer:
Figure 2 shows the distribution of the four gram frequency. As can be seen from the figure, most four-grams occur very infrequently (most of them are encountered just once). This by and large follows a zipf distribution. Also, given that most words are finance related, I assume the dataset is a portion of the Penn Tree Bank data on WSJ articles.
The top 50 four grams, have been mentioned below:

```
0 *t*-1 . <END>
$ <UNK> million *u*
said 0 *t*-1 .
million *u* . <END>
, '' says *t*-1
million *u* , or
0 *t*-2 . <END>
```

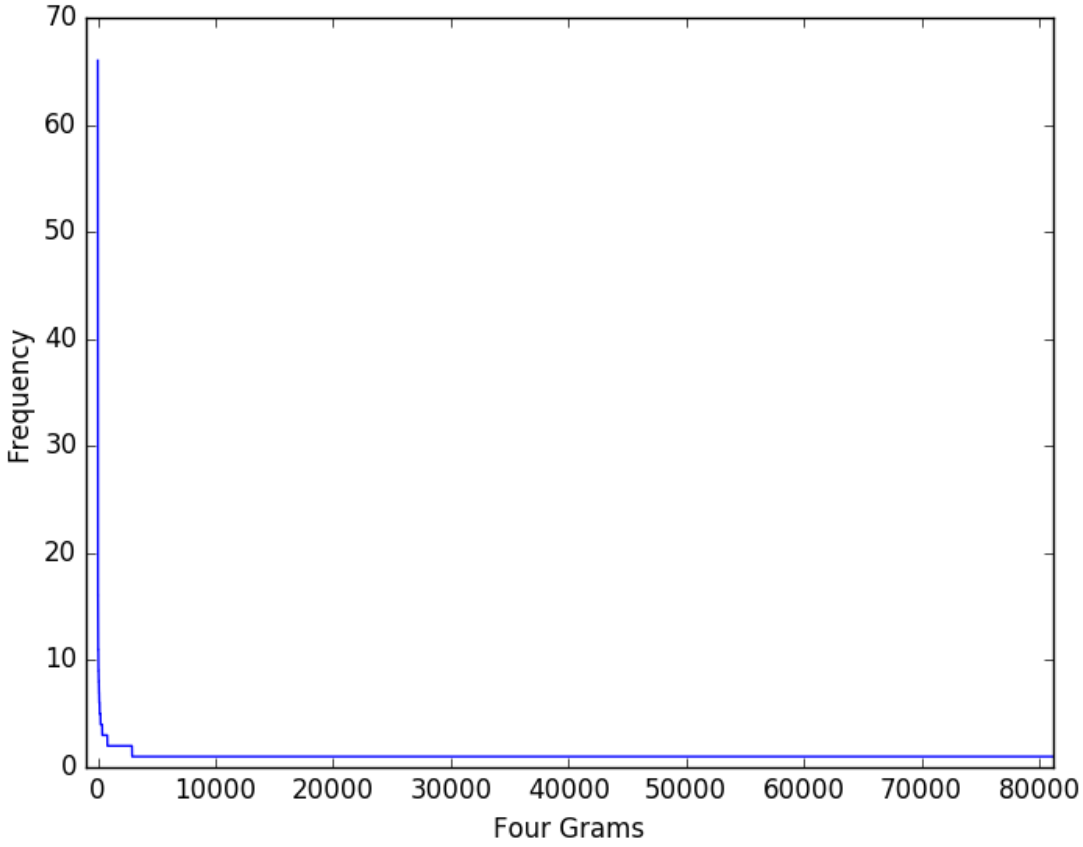


Figure 2: Four Gram Frequency Distribution

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

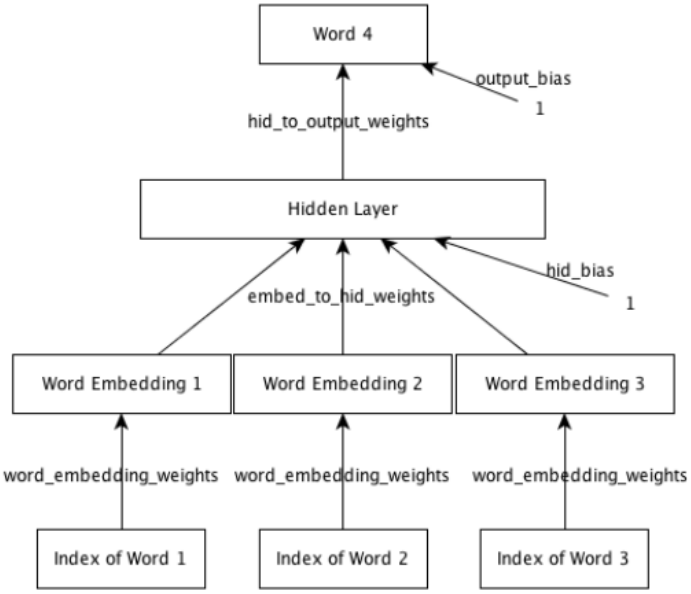


Figure 3: Language Model Architecture

```
<UNK> million *u* ,
, '' he said
, for example ,
in september . <END>
president and chief executive
cents a share ,
the u.s. . <END>
, said 0 the
company said 0 it
<START> the company said
million *u* from $
corp. said 0 it
said 0 *t*-1 ,
, he said 0
year earlier . <END>
*u* a share .
0 *t*-3 . <END>
<START> in addition ,
, or $ <UNK>
0 *t*-1 , the
this year . <END>
, said 0 it
program trading . <END>
```

3.2 Backpropagation with Linear Hidden Layer (20 pts)

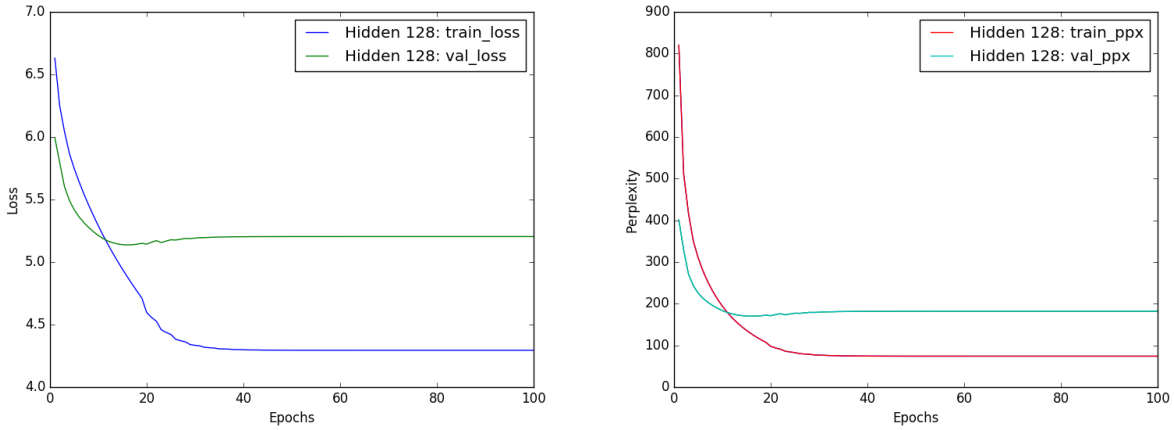
You will now implement the model shown in figure 3 with the linear hidden layer. Each word will be represented by a 16-dimensional embedding followed by hidden layer of 128 units. Your network's output will be the softmax over the vocabulary which tries to predict the next word. You are going to minimize 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.

Answer:
All models were trained with a momentum of 0.9, and an lr of 0.1, scheduled to be halved every time the validation



(a) Cross Entropy Loss

(b) Perplexity

Figure 4: Loss vs Epochs for hidden layer size of 128

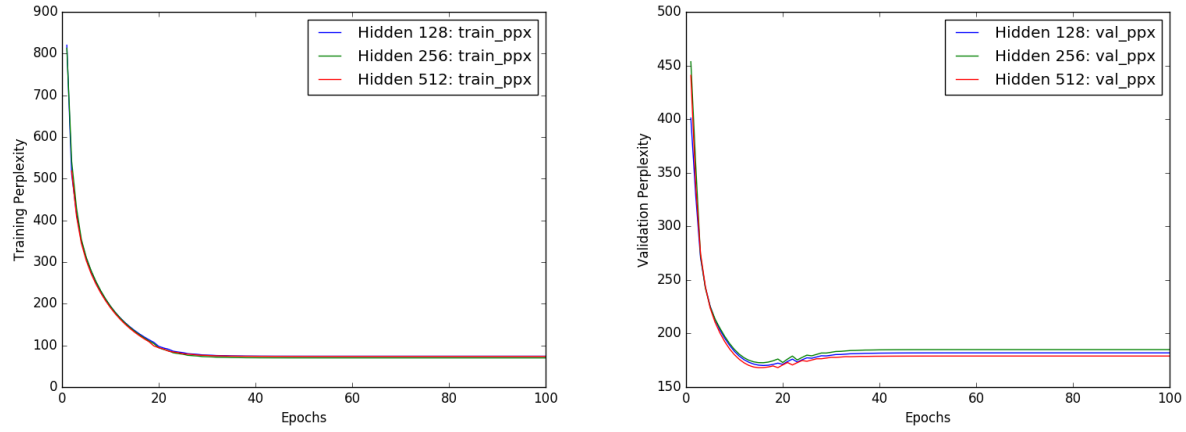
loss didn't improve across 2 epochs.
Figures 4a and 4b show the training and validation loss vs epochs for a network with 128 hidden layers. Since perplexity is in a sense a scaled version of the entropy loss ($2^{\frac{\text{entropy\_loss}}{\log 2}}$), the trend is the same in both the graphs. We observe that the perplexity decreases, and then stagnates. The model reaches the minimum really quickly, and then starts over-fitting. However, the over-fitting is not extreme because of the lr scheduling. By the 100th epoch, the lr was very small (of the order of 1e-7).
Figures 5a and 5b show the training and validation perplexity for varying hidden size. The training perplexity is roughly the same, however the best performing model on the validation set was the one with hidden size of 512. Hence, increasing the hidden size seems to help, but very significantly.

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).

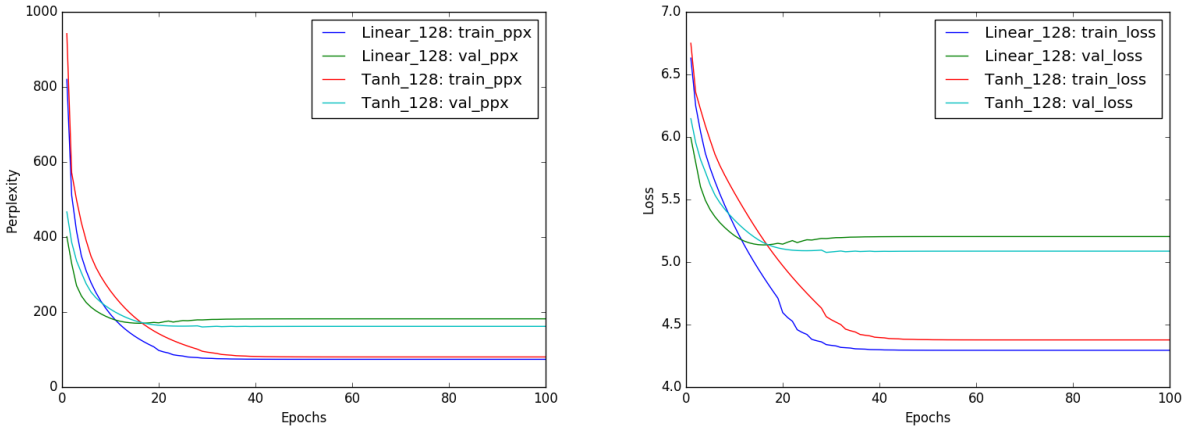
Answer:
Figures 6a and 6b show the perplexity and cross entropy loss for the linear and the tanh model. As expected, the tanh model performs better than the linear model; since the tanh model has more capacity. Note that all losses flat-line because of the lr scheduling; for both models, the final learning rate was of the order of 1e-7.
Figures 7a and 7b show the perplexity for different hidden layer sizes of the network. As can be seen, increasing the

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



(a) Training Perplexity Loss (b) Validation Perplexity Loss

Figure 5: Training and validation perplexities with changing hidden layer size



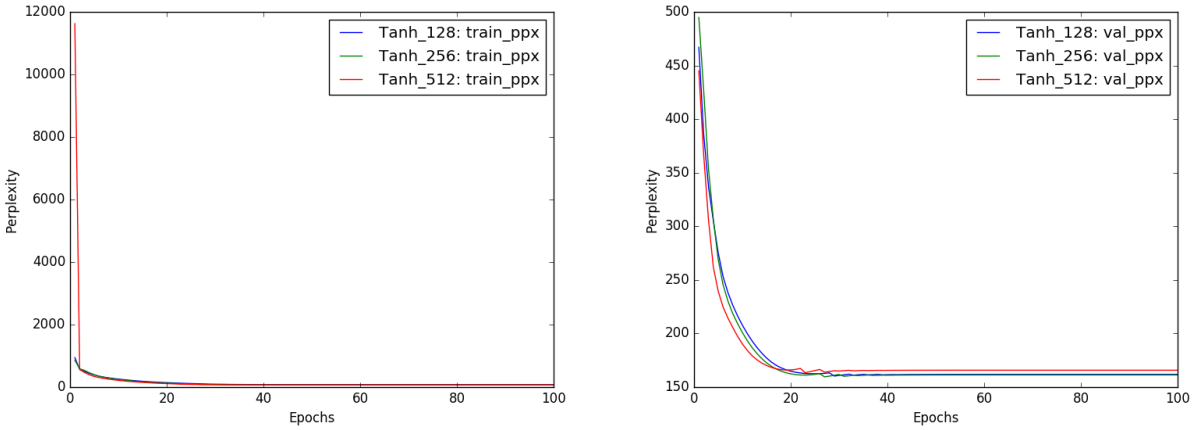
(a) Perplexity Loss (b) Cross Entropy Loss

Figure 6: Linear vs Tanh: Perplexity and Cross Entropy

hidden size does improve the accuracy, but again, not by a significant amount. Table 1 summarizes the results.

ACTIVATION	HIDDEN SIZE	TRAIN LOSS	VAL LOSS	TRAIN PPX	VAL PPX
linear	128	4.29	5.14	74.14	170.15
linear	256	4.24	5.15	70.06	172.75
linear	512	4.29	5.12	73.59	168.12
tanh	128	4.38	5.08	80.36	160.31
tanh	256	4.24	5.07	70.34	159.54
tanh	512	4.24	5.10	70.38	163.45
best RNN	128	4.55	5.122	126.84	167.73

Table 1: Table of Accuracies



(a) Training Perplexity Loss

(b) Validation Perplexity Loss

Figure 7: Linear vs Tanh: Perplexity and Cross Entropy

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.

Answer:
Sentence generation was done by sampling the softmax distribution instead of taking the argmax at every step. The best model (activation : tanh, hidden size: 256) was used to generate sentences in table 2. We see that common ngrams occur pretty frequently ("0 *t*-1", "said 0", ". END", "new york stock exchange") occur naturally. However, the sentence doesn't make sense with increasing length; which is expected, since we only condition on the previous 3 words.

Seed	Sentence
city of new	city of new england 's exchange and chief executive officer , " says
city of new	city of new york stock exchange , the company said 0 the nation
government of united	government of united and UNK the UNK of the UNK . END
life in the	life in the u.s. and the company 's largest UNK and UNK the
she is the	she is the UNK 0 *t*-1 to be a UNK in the UNK
START in addition	START in addition , the company said 0 the company 's UNK,

Table 2: Sentences generated by the language model

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.

Answer:
Table 3 shows the nearest neighbors of words using euclidean distance as the distance metric. For common words ("said") the model learns decent embeddings, and the nearest neighbors make sense. The bias of the corpus can also be seen, for example the word "exchange" has the nearest neighbors koito (Koito industries), sachs (Goldman Sachs), trust, Hutton (Shearson Lehman Hutton); implying that the word "exchange" is used more in the sense of

stock exchange than trading/ lending (the non financial sense). The nearest neighbors for settlements and accounting are particularly funny.

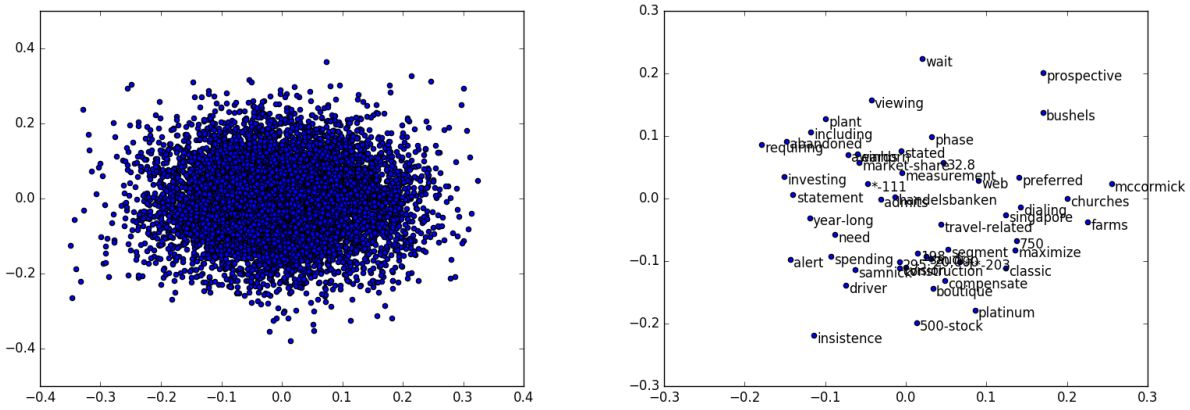
word	Nearest Neighbour
said	reported, added, say, noted, estimated
u.s.	futures, commodity, small, chicago, securities
exchange	koito, sachs, trust, hutton, meridian
settlements	mobster, incentives, fatalities, concentrated, vaccines
accounting	moneymakers, accounted, flirted, fledgling, apologize

Table 3: 5 Nearest neighbors using Euclidean distance

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.

Answer: Figure 8 shows the 2D plot of the embeddings. There is some structure (platinum and 500-stock are together, investing, statement, spending occur together etc.), but no obvious clusters are observed.



(a) All Embeddings

(b) Sample 50 words

Figure 8: Embeddings. Note that a Gaussian noise of mean 0, std 0.01 was added for better visualization

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

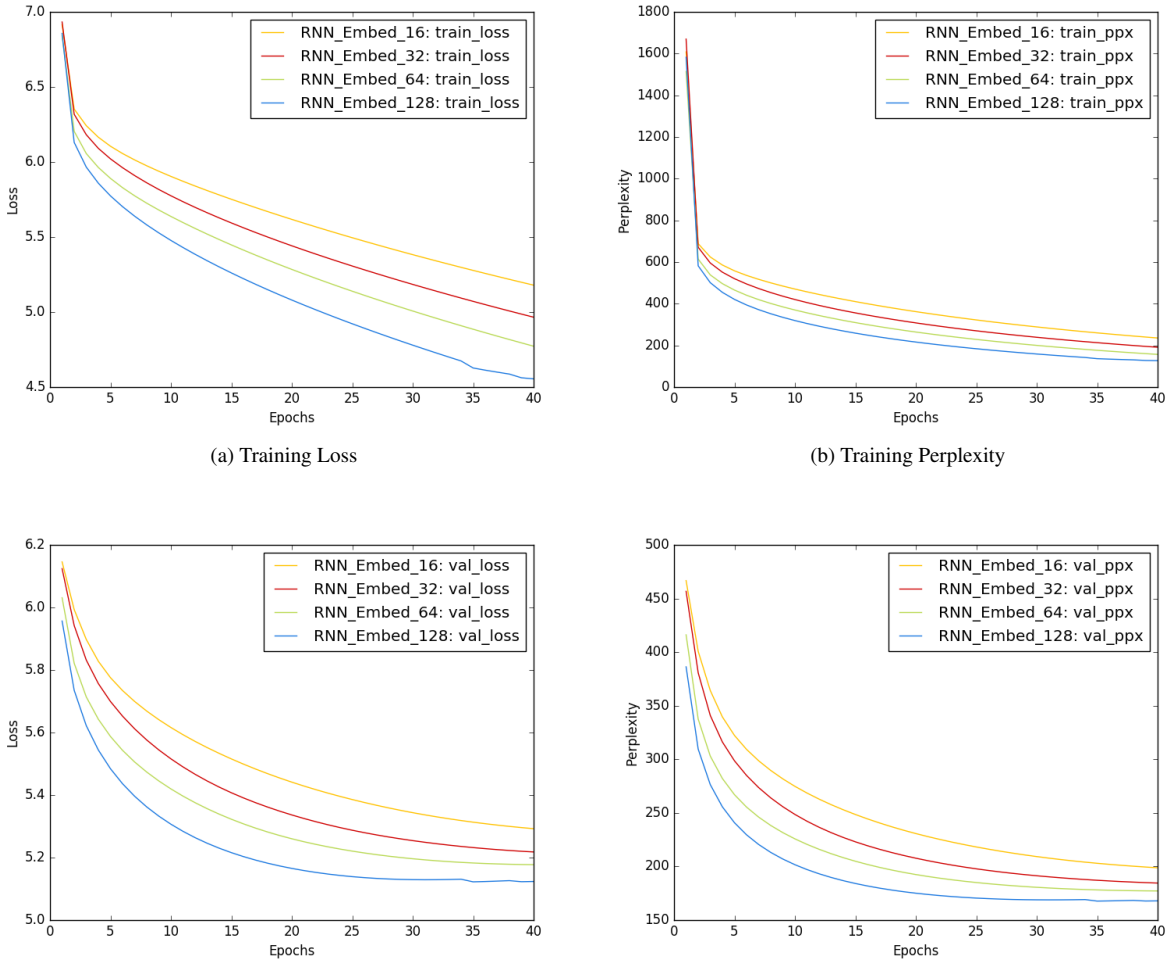
Each of the input will now go to an RNN cell that processes one word at a time and computes probabilities of the possible values for the 4-gram(time step = 4).

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.

Answer:
Figure 9 shows the entropy loss and perplexity of the RNN language model as a function of the number of epochs.



(a) Training Loss

(b) Training Perplexity

(c) Validation Loss

(d) Validation Perplexity

Figure 9: Entropy Loss and Perplexity for RNN model with different embedding sizes

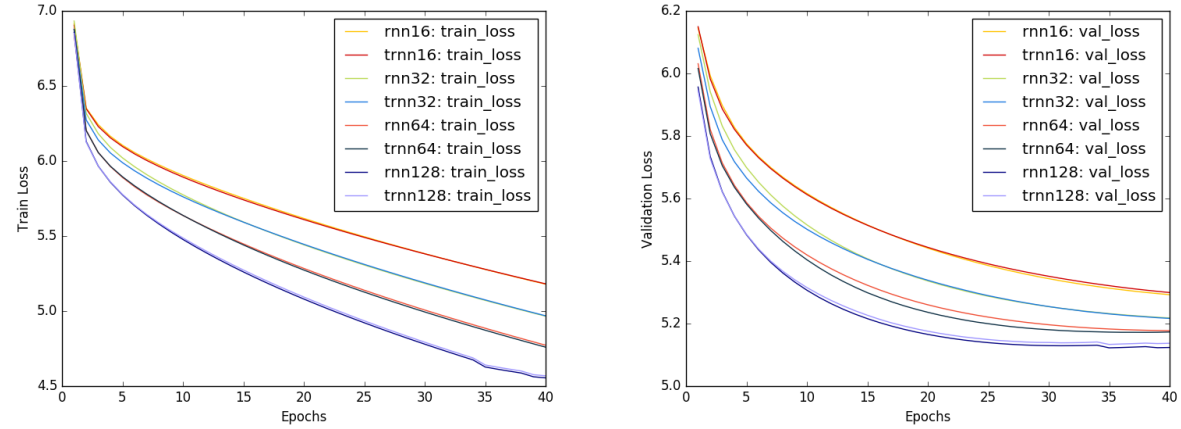
We see that increasing the embedding size helps improve both the training and the validation error. This makes sense, since increasing the embedding size increases the model capacity, which allows it to model the data distribution better. However, as can be seen from Table 1, the RNN network performs worse than the Tanh model. A reason for the same could be that in the case of the tanh model, we feed the entire 3-GRAM information to the network, whereas in the RNN network, it is possible that the network forgets the initial inputs (since the interactions b/w the hidden states is

multiplicative, this is known to happen).

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.

Answer:
Figure 10 shows the loss and perplexity of RNN and Truncated RNN (TRNN) models. As can be seen TRNN models



(a) Training Loss

(b) Training Perplexity

(c) Validation Loss

(d) Validation Perplexity

Figure 10: RNN vs TRNN

have a better validation perplexities initially; however, both converge to similar values. I believe TRNN would be useful for larger NGRAMS model, when BPTT would actually cause gradient to vanish / explode; their efficacy cannot be seen in a 4 GRAM language model.

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.