Assignment 2

Neha Iyer
SR No. 14942

1 Task 1

To build the best token level LSTM-based language model:
The token-level LSTM language model was implemented under following settings:

2 Datasets used

D2 Gutenberg corpus

3 Handling of unknown words

1. To handle the cases of unknown words, the vocabulary size was fixed to 40000 unique tokens.
2. The entire corpus was read and only the top 40000 most frequent words were retained and rest of the tokens was replaced with UKN token.
3. After replacing with UKN token the data set was split into train,validation and test sets.

4 Data Split

The dataset was divided into train and test sets in following manner:
For D2 Gutenberg corpus, the train set was created by taking 80% from each file and the test set was created by taking remaining 20% of each file. However due to the large time taken for training the model the training set size was reduced to 3,39,238 tokens out of which 10,733 where unique tokens. Correspondingly the test set size was reduced to total 84,810 tokens.

5 Data Preprocessing

Following preprocessing steps were performed in order to clean the raw text before feeding it to the training model:

1. Removal of numeric characters
2. Removal of text punctuations
3. Transform the text case to lower case
4. Removal of apostrophes from words e.g. one's, that's, they'd

6 Approach followed

The approach followed to develop token-level language model is as follows:
1. The train set was divided into sequences of length 51.
2. These sequences were then fed to the LSTM language model in batches of size 30.
3. Given an input sequence of length 50, the language model was required to predict the 51 th word.

7 Data Representation

1. An embedding layer is used to learn the dense vector representation for each word as part of training.
2. The size of dense vector representation used is 50.
3. Before feeding into the embedding layer, the input sequences were encoded to integer values using Tokenizer class of Keras API.

8 Hidden layers

1. Two hidden layers are used with 100 memory cells each. The hidden layers will extract useful features from the input sequence.
2. The output layer outputs a single vector of the size of the vocabulary with a probability for each word in the vocabulary.
3. The Relu activation function is used to convert

the output to valid probablities.

9 Evaluation metric

Evaluation Metric used for computing the training set loss is "$sparse\_categorical\_crossentropy''$. Wheareas the evaluation metric used to compute test set loss is perplexity which is computed as 2^{loss}.

10 Results

Please note that the language model was implemented using Keras library and all the following results were obtained purely by executing the code without GPU support.

- Number of epochs: 60
- Batch size: 30
- Time steps: 50
- Training accuracy: 17%
- Test loss: 8.511

11 Task 2

To build the best character level LSTM-based language model:
The character-level LSTM language model was implemented under following settings

12 Data Preprocessing

Preprocessing steps followed are same as that for Task 1.

13 Data Split

The dataset was divided into train, validation and test sets in following manner:
For D2 Gutenberg corpus, the train set was ceated by taking 80% from each file and the test set was created by taking remaining 20% of each file. However due to the large time taken for training the model the training set size was reduced to 17,80,584 characters out of which 27 where unique characters. Correspondingly the test set size was reduced to total 449836 characters.

14 Approach followed

The approach followed to develop character-level language model is as follows:
1. The raw text is split up into character sequences of length 120.
2. Each training pattern consists of 120 time steps of one character (X) followed by one character output (y). The sequences are created by sliding the window one character at a time allowing each character to be learned.
3. Total of 17,80,464 sequences of 120 characters each are generated and fed for training.
4. The input is normalized and the output vector is converted into a one-hot vector of size vocab size.

15 Hidden layers

1. One hidden layer is used with 200 memory cells each. The hidden layers will extract useful features from the input sequence.
2. The output layer outputs a single vector of the size of the vocabulary with a probability for each character in the vocabulary.
3. The Relu activation function is used to convert the output to valid probablities.

16 Evaluation metric

Evaluation Metric used for computing the training set loss is "$categorical\_crossentropy''$. Whareas the evaluation metric used to compute test set loss is perplexity which is computed as $2^(loss)$.

17 Results

Please note that the language model was implemented using Keras library and all the following results were obtained purely by executing the code without GPU support.

- Number of epochs: 20
- Batch size: 128
- Time steps: 120
- Training loss: 2.8582
- Test loss: 8.7430

Model	Perplexity
Classical Bigram	30.29
Token-level LSTM	364.80
Character-level LSTM	428.48

Table 1: Perplexity measure of LMs

18 Performance comparison with the best classical LM

The best classical language model under the same setting was obtained using the bigram model.
Table 1 shows the perfomance comparison of all the three language models.

19 Observations/Analysis

1. The classical bigram language model gives best perplexity as compared to char-based and token-level language models.

2. Token-level language model displays higher accuracy and lower computational cost than char-based language model.

3. This is because char-based language model require much bigger hidden layer to successfully model long-term dependencies which results in higher computational costs. Character language models need to learn spelling in addition to syntax and semantics which means in order to perform better than word-level models, char-based models have to be trained on a larger corpus with more number of hidden layers and time-steps. Also the char-based model was trained for relatively very epochs as compared to token-level model.

4. The sentences generated by classical bigram and word-level model seems to be almost comparable. This is due to the fact that the word-level model was trained for relatively less number of epochs and on a smaller training corpus due to training time constraints. However the accuracy and perplexity of word-level model is known to beat the classical bigram model if trained on a larger training set and for higher number of epochs.

20 Task 3

To generate a sentence of 10 tokens
Out of token-level and char-level, the best model as observed is token-level. So using this model a sentence of 10 tokens is generated given a seed text as input.
On executing the script generate_sentence.sh, the sentence is generated in a file called "output.txt".
An example sentence generated is shown below:

Seed text: "will not it be beautiful in her dark hair very beautiful indeed replied emma and she spoke so kindly that he gratefully burst out how delighted i am to see you again and to see you in such excellent looks i would not have missed this meeting for the world i"

Generated text: "have been a very much and he will be a very"