HW3 – Report

Gal Barak – 204233688

Shahar Stahi – 305237257

In this HW we were asked to build a network that creates sentences.

Phase 1: Data Loaders

For this practice we used Penn Tree Bank as a sentences source. The data comes in text files and as a start we created data frames (one for each set: train, validation and test) using Panda. Then we converted each data frame into a json format in order to tokenize it using SpaCy vocabulary.

The corpus we build will contain sentences where each starts with <sos> and end with <eos>.

Now we use the generic field we created in order to create 3 datasets and then the Bucket Iterator will use them to create the 3 data loaders.

Finally, we will create our vocabulary out of the data loaders. In order to lessen the data and improve performance, we chose to ignore tokens that appears less than twice. Additionally, in accordance to the longest sequence, we added padding to shorter sequence in order to have all vectors in the same size.

Phase 2: Net Construction

The network we'll use is LSTM based.

We chose to build a net with 2 layers where each's hidden dim is 256.

Additionally, we added an embedding layer that converts token indices to dense tensor with dimension of 256 and of course a fully connected linear layer which will convert dense dimensions to token dimensions.

The optimizer we used is Adam and in the loss function we'll ignore <pad> tokens.

```
LSTM_Predictor(
  (embedding): Embedding(9701, 128)
  (lstm): LSTM(128, 256, num_layers=2, dropout=0.3)
  (out_fc): Linear(in_features=256, out_features=9701, bias=True)
  (log_softmax): LogSoftmax(dim=2)
  (loss_func): CrossEntropyLoss()
)

number of parameters = 4,656,485
```

Phase 3: Network Train

The train process includes 30 epochs where each use a batch size of 32.

In each epoch, we'll perform a full analysis of the data by batches in the following way:

Each batch includes a tensor as the following:

```
tensor([[  2,   2,   2,  ...,   2,   2,   2],
        [  8,   8,   8,  ...,   8,   8,   8],
        [291,   4,  25,  ..., 214,  21,  37],
        ...,
        [  1,   1,   1,  ...,   1,   1,   1],
        [  1,   1,   1,  ...,   1,   1,   1],
        [  1,   1,   1,  ...,   1,   1,   1]])
```

Each column represents a sentence from our train set. We can see that each sentence starts with index 2 = <sos>, then we have different words and finally all sentences will end in index 1 = <pad>. The sentences length is being determined by the longest sentence. The amount of sentences is being determined by the batch size.

Now, for each batch we'll start with the embedding process, its output we'll move through the LSTM Network and the results we'll move through the linear layer.

The forward function output is a 3 dim matrix with 'batch size' columns where each contains a sentence in length 'max sequence' where each contains a mapping to a score vector in size 'vocabulary length'. Each cell in this vector has a grade for this token.

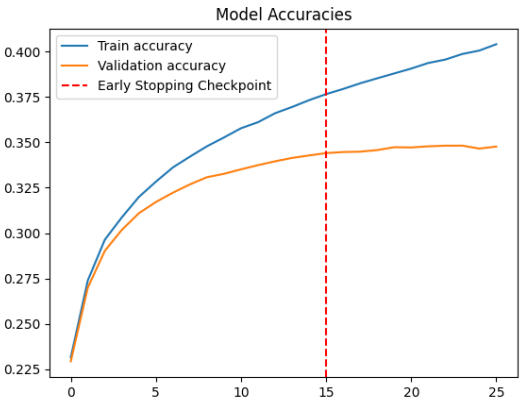
We'll notice that we need to compare Xi+1 to Yi and hence will use slicing to the prediction vector.

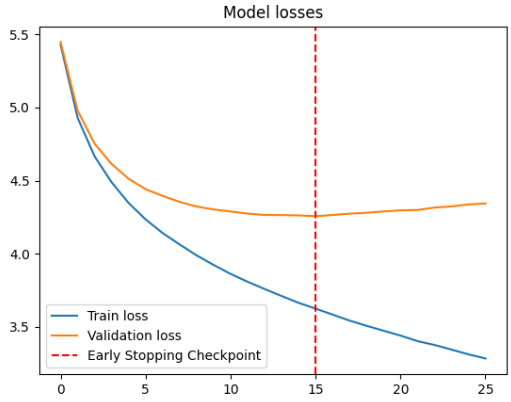
We'll run the loss function between the predictions to the original labels and update the net.

Additionally, since in RNN networks we can encounter a state where all gradients increasing with no control, we'll use the 'clip grad norm' component in order to give an upper bound of 1 to the gradients values.

At the end of each epoch we'll check the early stopping mechanism in order to avoid over fitting.

Finally, we'll load the best model we got and will calculate its test accuracy.






```
Train: Accuracy = 37.65%, Avg Loss = 3.63
Validation: Accuracy = 34.41%, Avg Loss = 4.26
Test: Accuracy = 35.57%, Avg Loss = 4.19
```

Phase 4: Words Creation

Method 1: the next word is based on the previous word estimation.

In this method, we randomly chose the first word in the sentence and based on that the network we trained evaluates the first word. Now, based on the prediction to this word and network will predict the next one and so on.

Eventually we'll get a sentence with 50 words (or less if the network predicted <eos>).

In the first iteration as said we'll pass the words we chose randomly to the embedding and to the LSTM we'll pass 2 random vectors (H and C).

From the second iteration and forward, the embedding will get the previous estimation and the LSTM will pass the parameters we got from the LSTM in the previous iteration.

At the end of each iteration we'll pass the output to linear layer, pass its score into argmax which will return the entry with the highest score. This entry will represent the token with the highest score.

After we performed this process for 50 iterations we got a matrix where each row represents a token indices column. Number of columns = batch size.

Then we'll transpose this matrix and now we have 'batch size' rows where each contains an indices vector. Each of this row will be translated back to tokens based on our vocabulary as can be seen in the following image:

```
Sentence 1:
['drink' 'improvements' 'stream' 'whichever' 'refineries' 'whichever'
 'refineries' 'policyholders' 'momentum' 'abroad' 'widens' 'artificially'
 'format' 'incurred' 'artificially' 'vacancy' 'widens' 'artificially'
 'format' 'incurred' 'artificially' 'refined' 'redemptions' 'refined'
 'widens' 'wires' 'rain' 'widens' 'artificially' 'refined' 'morale'
 'widens' 'dependents' 'refineries' 'audio' 'refineries' 'widens'
 'satisfactory' 'refineries' 'whichever' 'refineries' 'whichever' 'widens'
 'whichever' 'widens' 'momentum' 'dependents' 'reinvest' 'temperatures'
 'widens']
```

Method 2: the next word is based on the ground truth.

In this method we took sentences from the test set. We gave our network a full sentence and the output (a sentence the network created) is a concatenation of each component's output. Using this method we were able to produce sentences while using the original sentence context.

The translation from tokens to words was made in the same way of method 1.

Method 3: using Beam Search

The third method we generated sentences is by using Beam search as we learned in class.

We also used the explanations form the provided link in the assignment.

Conclusions:

The way that generates a more similar text to the real word:

2.b provides the most similar text to the real world since it does take reference from the real text and looks at the previous output.

The way that generates the most “non-sense" text:

2.a provides the most "non-sense" text since it doesn’t take any reference from the real text and only look at the previous output.

The way that generates a new text that makes sense:

2.c generates new text that makes sense since it takes various options into consideration and takes the one with the highest probability.