Report - INF265 - Third Assignment

By Benjamin Mortensen Hansen & Vegard Berge

Q&A Answers

1. Give an explanation of your approach and design choices to help us understand how your particular implementation works

The data preprocessing is inspired on weekly exercise 14. We start by loading the training books as strings and start to preprocess the data. We remove elements like digits, newline characters, spaces and so on. After this operation, we count the word frequency so that we can build a vocabulary. To save some computation time, we have a minimum frequency at 100 occurrences. Then at the end we tokenize the entire dataset. We found this to be tedious and quite hardware limited, and that is why we choose to store the pytorch datasets as a file. This way, we only needed to "create" the datasets one time, and could simply load the datasets for each run.
Beam search takes in the text together with the length of words to add and the beam size. It converts the text into an integer array with mappings according to the already defined vocabulary. To search it selects the topK next words for the text at time T. It uses a buffer list to store the new sorted beams then takes those new beams the top k to the next iteration. When having composed text of sufficient lengths it returns the most probable one. Note is that we do not extend a new beam with <unk> or the same word as was previously added.

*Note: We assume that you should yield a text of given length, therefore we do not consider shorter sequences that the model is more confident in. As the task is to generate text, not translate or complete the sentence.

2. Report the different models and hyper - parameters you have used

Context size set to 2. Embedding dimension 16.
Models in task 2.1:

Learning rate 1e-3, ADAM Momentum and RSMProp Betas (0.9, 0.999), ADAM Vanishing and Exploding Gradients 1e-8 and L2 Regularization lambda set to 1e-8

NG - gram Model 1 (Deep): Two linear layers. ReLu first layer. Output size 64

NG - gram Model 2 (Middle): Two linear layers. ReLu first layer. Output size 48

NG - gram Model 3 (Shallow): Two linear layers. ReLu first layer. Output size 32

Models in task 2.2:

Learning rate 1e-3, ADAM Momentum and RSMProp Betas (0.9, 0.999), ADAM Vanishing and Exploding Gradients 1e-8 and L2 Regularization lambda set to 1e-8

All models use the previous best models embeddings.

MLP 1: One linear layer. ReLu.

RNN1 (Deep 2 Layers): One LSTM layer.

MLP 2: Two linear layers. Relu last layer.

RNN2 (Shallow 2 Layers): One LSTM layer.

Models in task 2.3:

Learning rate 1e-2, ADAM Momentum and RSMProp Betas (0.9, 0.999), ADAM Vanishing and Exploding Gradients 1e-8 and L2 Regularization lambda set to 1e-8

All models use the previous best models embeddings.

RNN1 (Deep 2 Layers): One LSTM layer.

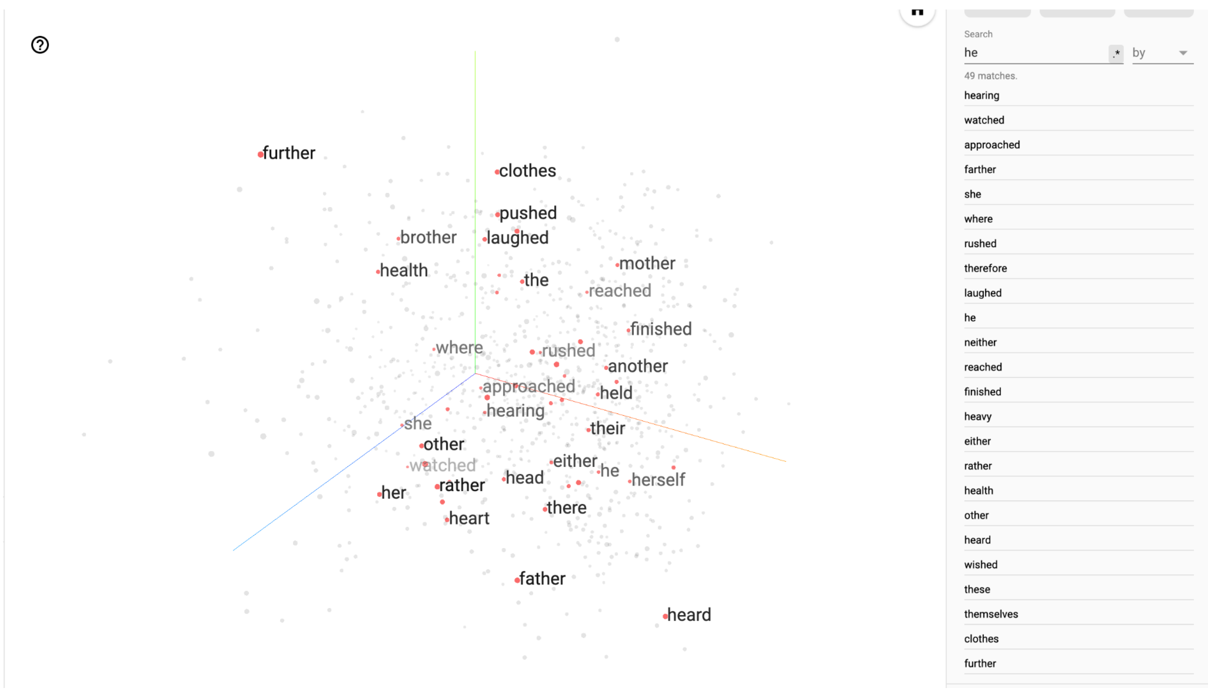
RNN2 (Shallow 2 Layers): One LSTM layer.

3. /4. Report the performance of your selected model and comments

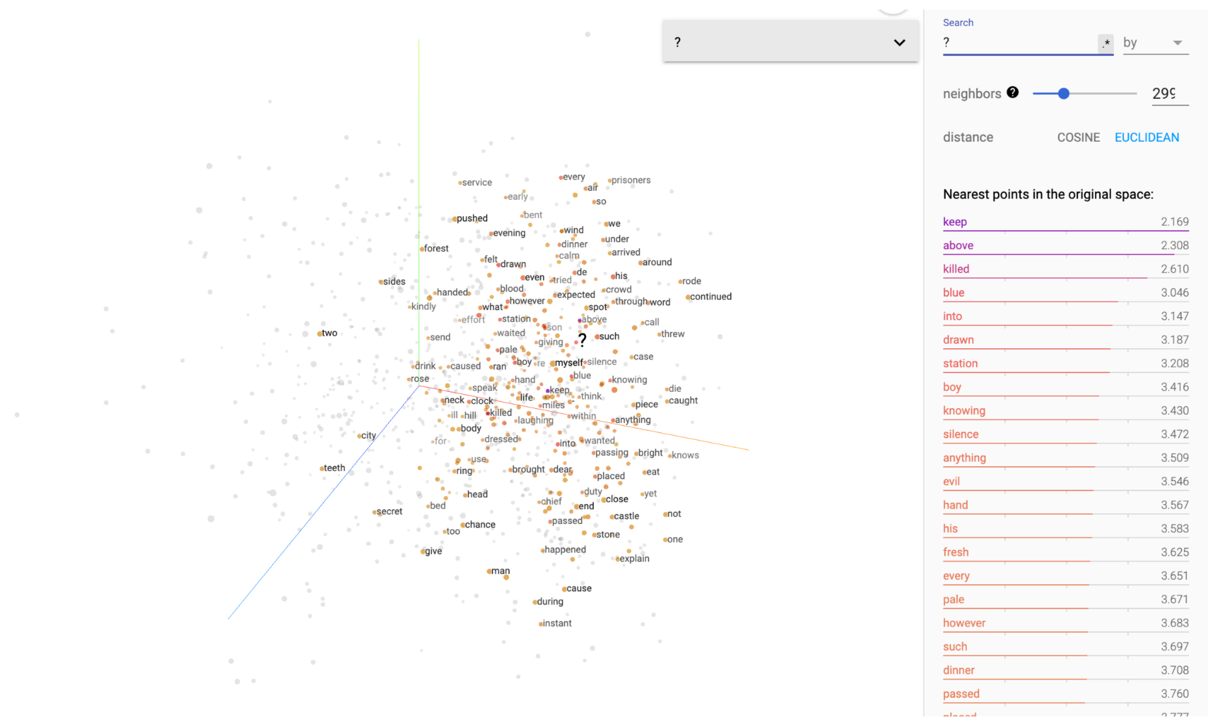
Note*: All results are heavily affected by poor model performance and accuracy due to restricted training on possessed hardware.

Models in task 3.1:

Results from 2.1 shows that the ‘Deep model’ performs the best. With a test accuracy of 18% with minimal training. From the cosine similarity matrix of the embeddings ‘he’ is similar to ‘she’, ‘in’ is similar to ‘into’, and ‘!’ is similar to ‘?’. Looking at projector.tensorflow.org agrees with these results:



Search
he
by
49 matches.
hearing
watched
approached
farther
she
where
rushed
therefore
laughed
he
neither
reached
finished
heavy
either
rather
health
other
heard
wished
these
themselves
clothes
further



Search: ?

neighbors 299

distance COSINE EUCLIDEAN

Nearest points in the original space:

keep	2.169
above	2.308
killed	2.610
blue	3.046
into	3.147
drawn	3.187
station	3.208
boy	3.416
knowing	3.430
silence	3.472
anything	3.509
evil	3.546
hand	3.567
his	3.583
fresh	3.625
every	3.651
pale	3.671
however	3.683
such	3.697
dinner	3.708
passed	3.760

Models in task 3.2:

When predicting targets from the context around we made two MLP's and two RNN's. The results are skewed since the MLP's are better trained. But the 'Shallow MLP' was the best model with a test accuracy of 30%. Training time between the MLP's and RNN's are significant as the average training time for the MLP's is 24 seconds and for the RNN it is 56 seconds. Which is double, meaning that it takes twice as long to train a RNN compared to MLP of the same size. This is due to the more demanding backprop calculations in the RNN.

Models in task 3.3:

We decided upon two models, one shallow and one deep.The training loss is high in both showing that to reduce the loss we'd have to train it for longer. The best model is the 'Deep RNN' with an accuracy of 1%. Even with poor performance the model still carries some language integrity as shown in the examples from the text generator.

"Sir arrived at noon with train and was happy" -> "Sir arrived at <unk> with train and was happy waiting am tomorrow anything waiting am".

We see some repetition due to bias to certain words, this comes from lack of training.

"A dream" -> " A dream understand anger"

"I have a dream a world where " -> "i have a dream a world where cannot woman commander world. darkness run wall reason. twenty ring . darkness run wall reason . darkness am"

Remarks about the code:

Due to either lack of code awareness or a library fault, we were not able to freeze the embeddings at task 2.2 and from the best model and onwards. For some reason the torch embedding library did not have attributes like .detach().clone() and neither nn.Embedding.from_pretrained(best_model.embeddings, freeze=True). We collaborated with TA, and were not able to resolve the matter. This will make the embeddings trainable for the other models -> which it was not supposed to.