Report INF265 project03

Task 2.1 – Word embedding

1) Approach and design choices

This time we decided to stick to simple architectures, and then tune hyperparameters to obtain different models. Thus, with the CBOW model, we trained three models with the same architecture.

2) Models and hyperparameters

CBOW Model 1: The first model was a simple CBOW implementation with one embedding layer and one linear fully connected layer. We used a batch size of 128, a learning rate of 0.01, 21 epochs for training, and an embedding dimension of 10. The vocabulary size was constant throughout the project, 1880.

CBOW Model 2: The second model had the same architecture, but different hyperparameters. We decreased the learning rate to 0.001 and increased the embedding dimension to 12. The rest of the hyperparameters remained unchanged from model 1.

CBOW Model 3: The third model also had the same architecture, but now we increase the embedding size even more, to 16. We also increased the batch size to 512. The rest of the hyperparameters remained unchanged from model 2.

3) Performance of selected model

Model 2 had the highest validation accuracy and so we went with that one. It had a validation accuracy of approximately 73.1%, but a much lower test accuracy of approximately 25.6%.

4) Results

In terms of accuracy, the model performed quite poorly. However, the objective was not to train a good predictor, but to have a representation of the semantics of the words in the vocabulary. For the example words (*me, white, man, have, be, child, yes, what*), we found that the ten most similar words actually were somewhat similar to the respective example words. It least for most of the words (maybe except *yes*) we could place the ten most similar words into the same group of words.

Task 2.2 – Conjugating *be* and *have*

1) Approach and design choices

Also here we decided to use the same architecture for the different models, that is one MLP architecture and one RNN architecture. Unfortunately, we did not manage to implement an MLP with an attention layer.

2) Models and hyperparameters

MLP Model 1: The first model was a simple MLP architecture with two fully connected linear layers. We used a hidden dimension of 10, a learning rate of 0.01, and a batch size of 512. The input and output dimensions of the network were already decided from the task.

RNN Model 1: The second model was an RNN architecture with the trained embedding layer from task 2.1, then a long-term-short-term layer (nn.LSTM), and then a fully connected linear layer. We used a hidden dimension of 10, a batch size of 512 and a learning rate of 0.01.

RNN Model 2: The third model had the same architecture as the first RNN. We used a hidden dimension of 16 this time, the rest remained unchanged from the previous.

3) Performance of selected model

Model 3 had the highest validation accuracy, with an accuracy of approximately 45.8%. It had a test accuracy of approximately 40.1%.

4) Results

The model performed quite poorly, with a test accuracy of 40.1%. Maybe a deeper MLP or an MLP with the attention layer would perform better. Adding more layers to the RNN might also have helped. It performs quite poorly on the training set as well, so the model is most likely an underfit.

Task 2.3 – Text generation

1) Approach and design choices

We used the same RNN architecture for both models we trained, with an embedding layer to start off, then an RNN layer, and a fully connected linear layer in the end.

2) Models and hyperparameters

Model 1: For the first model, we used a hidden dimension of 16, a learning rate of 0.01, and a batch size of 512.

Model 2: For the second model, we used a hidden dimension of 32, to try and decrease the training loss. The rest of the hyperparameters remained the same.

3) Performance of selected model

Model 2 performed best in terms of validation accuracy, and so we chose this one. It had a validation accuracy of approximately 21.8%. It had a test accuracy of approximately 26.3%.

4) Results

The model performed poorly in terms of accuracy. However, after implementing the beam search algorithm, it managed to produce some sentences that kind of made sense.