Assignment 3 Report

Can Gözpınar [1]

1. RNNs for Classification

1.0.1. Question 1

My implementation of *RNNClassifier* extends *torch.nn.Module* to form the Deep Neural Network model. It consists of an embedding layer of type *torch.nn.Embedding*, GRU model of type *torch.nn.GRU*, ReLU layer of type *torch.nn.ReLU*, Linear layer of type *torch.nn.Linear*, and a softmax layer of type *torch.nn.Softmax*. Given an input, it first converts the sequence of characters of type *char* to a sequence of its corresponding indices using the *Indexer* object passed to it. Then this sequence is passed through the embedding layer to get the word embeddings. This sequence of word embeddings is then passed through the GRU model which possesses some sense of memory. Then its final output (one generated after processing the whole sequence) is extracted and passed through a ReLU activation layer. Then its output is passed through the linear layer. This layer (linear layer) has its output dimension equal to two since we have two classes in this task namely, vowel and consonant. This layer produces the logits which are then passed through the softmax layer to output corresponding prediction probabilities of the classes. As the loss function Cross Entropy from *torch.nn.CrossEntropyLoss* is used. For measuring the time it takes to train the model with different hyperparameters I have created a decorator function that times the function execution which then I used to decorate training functions. For reproducibility purposes I have manually set seeds to *random, numpy, torch* libraries. My final model has achieved 77.6% accuracy. I share my experiments and my thought process along the way at A.1 section. There I share both the hyperparameters I have used and the corresponding model performance as pairs along with my final model that achieved the 77.6% accuracy. Please refer to that section for further clarification.

*Equal contribution [1]Department of Computer Engineering. Correspondence to: Can Gözpınar <cgozpinar18@ku.edu.tr>.

COMP541 Deep Learning, Fall 2022.

2. Implementing a Language Model

2.0.1. Question 2

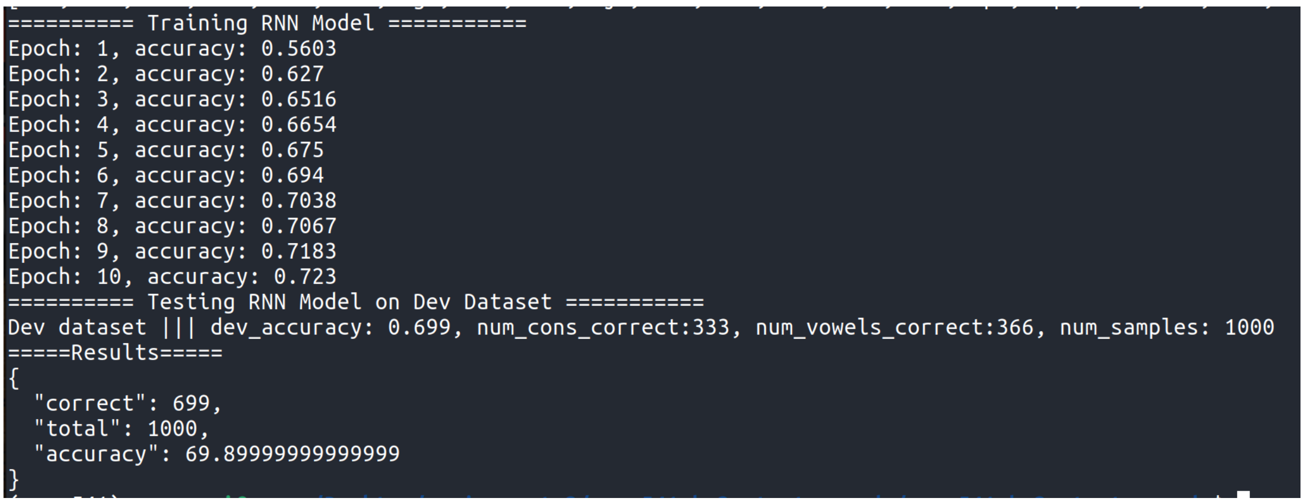
My implementation of *RNNLanguageModel* extends *torch.nn.Module* to form the Deep Neural Network model. It consists of an embedding layer of type *torch.nn.Embedding*, GRU model of type *torch.nn.GRU*, ReLU layer of type *torch.nn.ReLU*, Linear layer of type *torch.nn.Linear*, and a softmax layer of type *torch.nn.Softmax*. In its *forward* method, given an input, it first converts the sequence of characters of type *char* to a sequence of its corresponding indices using the *Indexer* object passed to it. Then this sequence is passed through the embedding layer to get the word embeddings. This sequence of word embeddings is then passed through the GRU model which possesses some sense of memory. It produces outputs for each step of the sequence it was passed. Then we pass these outputs through ReLU activation. The the outputs are passed to the linear layer. This linear layer produces outputs of dimension that is equal to the vocabulary size (i.e. output_dim = vocab_size = 27). This is because, we will need the probability over elements of the vocabulary for the language modeling tasks. Then it returns this final output. We will use this *forward* functions in both *get_next_char_log_probs* and *get_log_prob_sequence* functions. In the *get_next_char_log_probs* function, we pass the input through the aforementioned *forward* method, pass it through softmax layer to obtain probabilities. Then we extract the probability predictions for the last step of the sequence (i.e. output for the overall sequence). Then we take its log and return it as a *numpy* arrray. This constitutes a probability distribution over the characters inside the vocabulary. For the *get_log_prob_sequence* function, we implement a similar logic to the *get_next_char_log_probs* function. We iteratively extend the given *context* with characters from the *next_chars* sequence one by one and accumulate log probabilities for having a corresponding char from the *next_char* sequence come after the updated *context* according to our language model. Finally we return this accumulation of the log probabilities. As explained in the 1.0.1 section, we again manually set seeds for reproducibility reasons and use a decorator to time certain function executions. For forming the dataset we introduce a *chunk_size* parameter. It determines the size of sequences that are used to divide the training dataset into separate sequences used for training the model. To form the training dataset used for training, I divided the given dataset into strings of length *chunk_size*. The target labels for the corresponding chunks of sentences are the same sequences from the dataset shifted one character right with the same length. In other words, the input sequence (X)'s corresponding target for every sequence is the character that comes next in the sequence. During training I iterated over the input sequences I have generated and obtained probability distributions over the vocabulary for the next character. Then using the cross entropy loss (*torch.nn.CrossEntropyLoss*) I trained my language model. My model has made predictions on every step of the given input sequences during its training. During training I have tracked loss, perplexity and accuracy of the model. For tuning the hyperparameters I have stuck with the hyperparameters I found that worked best for part 1 (i.e. I used hyperparameters from experiment 12 of part 1's experiments). Also, I used *chunk_size* of 20. Even though this was my first experiment for this part, it has passed the *sane*, and *normalizes* checks and achieved 6.233 perplexity score which surpassed the $<= 7$ perplexity requirement for this part of the assignment. As a result I stuck with this model. Further information on the hyperparameters I used and the performance of the model can be found in the A.2.1 section.

A. Appendix

A.1. Part1: Experiments

A.1.1. EXPERIMENT 1

```
# Hyperparameters ==============================
word_embedding_dim = 10
gru_hidden_dim = 20
lr = 1e-3
epochs = 10
model = RNNClassifier(vocab_index, word_embedding_dim, gru_hidden_dim)
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr)
# ==============================================
```



```
========== Training RNN Model ===========
Epoch: 1, accuracy: 0.5603
Epoch: 2, accuracy: 0.627
Epoch: 3, accuracy: 0.6516
Epoch: 4, accuracy: 0.6654
Epoch: 5, accuracy: 0.675
Epoch: 6, accuracy: 0.694
Epoch: 7, accuracy: 0.7038
Epoch: 8, accuracy: 0.7067
Epoch: 9, accuracy: 0.7183
Epoch: 10, accuracy: 0.723
========== Testing RNN Model on Dev Dataset ===========
Dev dataset ||| dev_accuracy: 0.699, num_cons_correct:333, num_vowels_correct:366, num_samples: 1000
=====Results=====
{
  "correct": 699,
  "total": 1000,
  "accuracy": 69.89999999999999
}
```

Figure 1. Part1: Experiment 1

This was my first experiment. I started with small *word_embedding_dim, gru_hidden_dim*.

A.1.2. EXPERIMENT 2

```
# Hyperparameters ==============================
word_embedding_dim = 4
gru_hidden_dim = 20
lr = 1e-3
epochs = 10
model = RNNClassifier(vocab_index, word_embedding_dim, gru_hidden_dim)
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr)
# ==============================================
```

```
========== Training RNN Model ===========
Epoch: 1, accuracy: 0.5237
Epoch: 2, accuracy: 0.5741
Epoch: 3, accuracy: 0.6058
Epoch: 4, accuracy: 0.6207
Epoch: 5, accuracy: 0.6284
Epoch: 6, accuracy: 0.6314
Epoch: 7, accuracy: 0.6505
Epoch: 8, accuracy: 0.6461
Epoch: 9, accuracy: 0.6622
Epoch: 10, accuracy: 0.6778
=====Results=====
{
  "correct": 682,
  "total": 1000,
  "accuracy": 68.2
}
```

Figure 2. Part1: Experiment 2

With this experiment I reduced the *word_embedding_dim* which resulted in learning smaller vectors for representing the words. As a result it gave smaller accuracy compared to the prior experiment.

A.1.3. EXPERIMENT 3

```
# Hyperparameters ================================
word_embedding_dim = 20
gru_hidden_dim = 20
lr = 1e-3
epochs = 10
model = RNNClassifier(vocab_index, word_embedding_dim, gru_hidden_dim)
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr)
# ===============================================
```

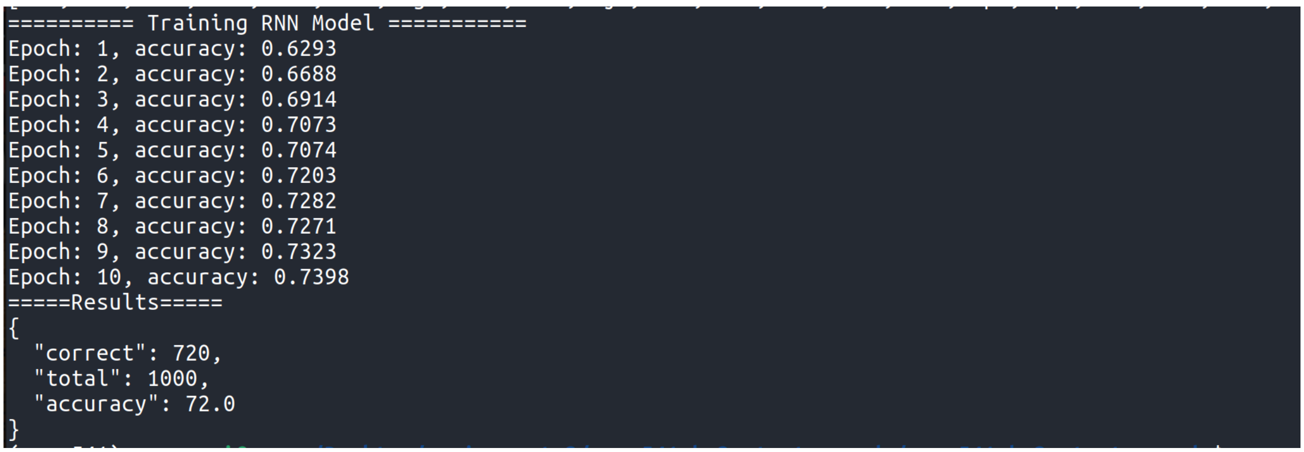
```
========== Training RNN Model ===========
Epoch: 1, accuracy: 0.579
Epoch: 2, accuracy: 0.638
Epoch: 3, accuracy: 0.6816
Epoch: 4, accuracy: 0.704
Epoch: 5, accuracy: 0.7078
Epoch: 6, accuracy: 0.7137
Epoch: 7, accuracy: 0.7251
Epoch: 8, accuracy: 0.7243
Epoch: 9, accuracy: 0.724
Epoch: 10, accuracy: 0.7362
=====Results=====
{
  "correct": 719,
  "total": 1000,
  "accuracy": 71.89999999999999
}
```

Figure 3. Part1: Experiment 3

With this experiment I increased the *word_embedding_dim* which resulted in learning larger vectors for representing the words. Larger vector meant an increased capacity for representing information. As a result it gave higher accuracy compared to the first two experiments.

A.1.4. EXPERIMENT 4

```
# Hyperparameters ==============================
word_embedding_dim = 20
gru_hidden_dim = 30
lr = 1e-3
epochs = 10
model = RNNClassifier(vocab_index, word_embedding_dim, gru_hidden_dim)
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr)
# ==============================================
```



```
========== Training RNN Model ===========
Epoch: 1, accuracy: 0.6293
Epoch: 2, accuracy: 0.6688
Epoch: 3, accuracy: 0.6914
Epoch: 4, accuracy: 0.7073
Epoch: 5, accuracy: 0.7074
Epoch: 6, accuracy: 0.7203
Epoch: 7, accuracy: 0.7282
Epoch: 8, accuracy: 0.7271
Epoch: 9, accuracy: 0.7323
Epoch: 10, accuracy: 0.7398
=====Results=====
{
  "correct": 720,
  "total": 1000,
  "accuracy": 72.0
}
```

Figure 4. Part1: Experiment 4

With this experiment I experimented with increasing the *gru_hidden_dim*. Increasing it meant more capacity to learn better representations for the hidden representations of the GRU which ideally meant better memory. As expected it resulted in increased accuracy.

A.1.5. EXPERIMENT 5

```
# Hyperparameters ==============================
word_embedding_dim = 20
gru_hidden_dim = 30
lr = 5e-4
epochs = 10
model = RNNClassifier(vocab_index, word_embedding_dim, gru_hidden_dim)
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr)
# ==============================================
```

```
========== Training RNN Model ===========
Epoch: 1, accuracy: 0.6032
Epoch: 2, accuracy: 0.6498
Epoch: 3, accuracy: 0.6713
Epoch: 4, accuracy: 0.678
Epoch: 5, accuracy: 0.6833
Epoch: 6, accuracy: 0.6791
Epoch: 7, accuracy: 0.7071
Epoch: 8, accuracy: 0.7145
Epoch: 9, accuracy: 0.7114
Epoch: 10, accuracy: 0.7261
=====Results=====
{
  "correct": 719,
  "total": 1000,
  "accuracy": 71.89999999999999
}
```

Figure 5. Part1: Experiment 5

Up to this point, I have observed that increasing both the *word_embedding_dim* and *gru_hidden_dim* results in better accuracy performance. Now I wanted to experiment with *lr* (learning rate). For this experiment I tried reducing it. It resulted in slightly smaller accuracy performance compared to the prior experiment. In the earlier experiments, I observed that increase in the accuracy between epochs reduced significantly after around epoch 7. As a resulted I though decreasing the learning rate would result in a slower but more steady learning which I can use to train for longer to obtain higher accuracy scores.

A.1.6. Experiment 6

```
# Hyperparameters ==============================
word_embedding_dim = 30
gru_hidden_dim = 50
lr = 5e-4
epochs = 10
model = RNNClassifier(vocab_index, word_embedding_dim, gru_hidden_dim)
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr)
# ==============================================
```

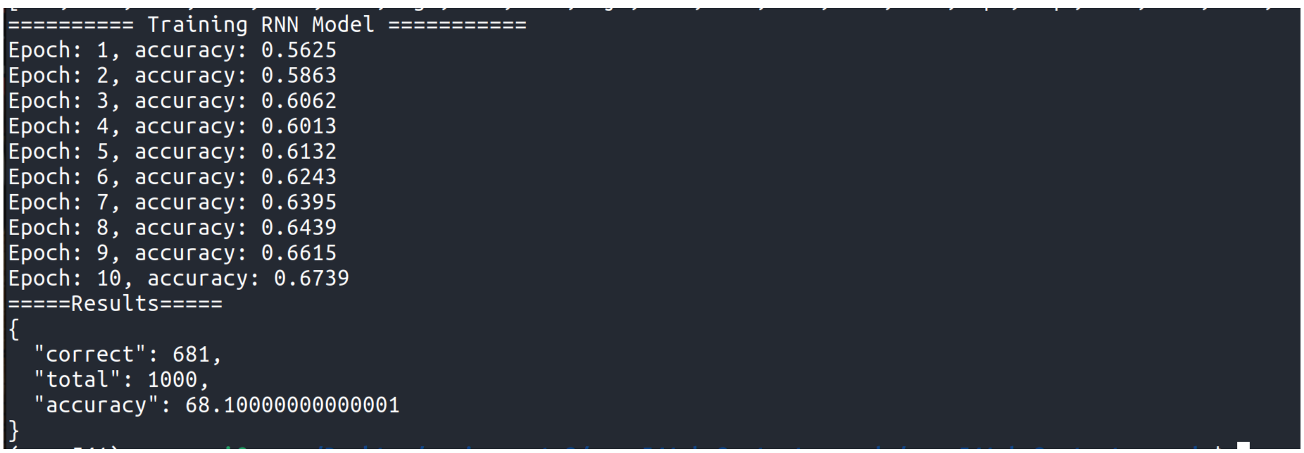
```
========== Training RNN Model ===========
Epoch: 1, accuracy: 0.6014
Epoch: 2, accuracy: 0.642
Epoch: 3, accuracy: 0.693
Epoch: 4, accuracy: 0.7068
Epoch: 5, accuracy: 0.71
Epoch: 6, accuracy: 0.7119
Epoch: 7, accuracy: 0.7229
Epoch: 8, accuracy: 0.7167
Epoch: 9, accuracy: 0.7222
Epoch: 10, accuracy: 0.7299
=====Results=====
{
  "correct": 713,
  "total": 1000,
  "accuracy": 71.3
}
```

Figure 6. Part1: Experiment 6

In this experiment I stuck with the smaller learning rate I found from the earlier experiment and increased the *word_embedding_dim* further. It resulted in higher training accuracy whereas, provided reduced test accuracy. Even though the differences were small, I suspected that it was suffering from the overfitting problem.

A.1.7. Experiment 7

```
# Hyperparameters ==============================
word_embedding_dim = 30
gru_hidden_dim = 50
lr = 1e-4
epochs = 10
model = RNNClassifier(vocab_index, word_embedding_dim, gru_hidden_dim)
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr)
# ==============================================
```



```
========== Training RNN Model ===========
Epoch: 1, accuracy: 0.5625
Epoch: 2, accuracy: 0.5863
Epoch: 3, accuracy: 0.6062
Epoch: 4, accuracy: 0.6013
Epoch: 5, accuracy: 0.6132
Epoch: 6, accuracy: 0.6243
Epoch: 7, accuracy: 0.6395
Epoch: 8, accuracy: 0.6439
Epoch: 9, accuracy: 0.6615
Epoch: 10, accuracy: 0.6739
=====Results=====
{
  "correct": 681,
  "total": 1000,
  "accuracy": 68.10000000000001
}
```

Figure 7. Part1: Experiment 7

Since in the earlier experiment I suspected overfitting the training data, I used smaller learning rate for this iteration. Unfortunately, resulting models accuracy performance was even lower.

A.1.8. Experiment 8

```
# Hyperparameters ==============================
word_embedding_dim = 100
gru_hidden_dim = 100
lr = 1e-3
epochs = 10
model = RNNClassifier(vocab_index, word_embedding_dim, gru_hidden_dim)
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr)
# ==============================================
```

```
========== Training RNN Model ===========
Epoch: 1, accuracy: 0.6738
Epoch: 2, accuracy: 0.7156
Epoch: 3, accuracy: 0.7256
Epoch: 4, accuracy: 0.7348
Epoch: 5, accuracy: 0.7383
Epoch: 6, accuracy: 0.7516
Epoch: 7, accuracy: 0.7505
Epoch: 8, accuracy: 0.752
Epoch: 9, accuracy: 0.7596
Epoch: 10, accuracy: 0.7648
=====Results=====
{
  "correct": 745,
  "total": 1000,
  "accuracy": 74.5
}
```

Figure 8. Part1: Experiment 8

After previous experiments failure, I decided to increase the models capacity further. To this end I increased the *word_embedding_dim*, and *gru_hidden_dim*. These resulted in bigger vectors to use for representations which could potentially learn to embed information better. I also increased the learning rate (*lr*) to learn quicker and reduce required training time by training for smaller epochs. Results came out to be better on both the training and the test set. It has achieved 74.5% accuracy which was the best performance I had up to this iteration and it indicated me that the approach I took in this experiment was towards something better.

A.1.9. Experiment 9

```
# Hyperparameters ==============================
word_embedding_dim = 150
gru_hidden_dim = 150
lr = 5e-4
epochs = 10
model = RNNClassifier(vocab_index, word_embedding_dim, gru_hidden_dim)
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr)
# ==============================================
```

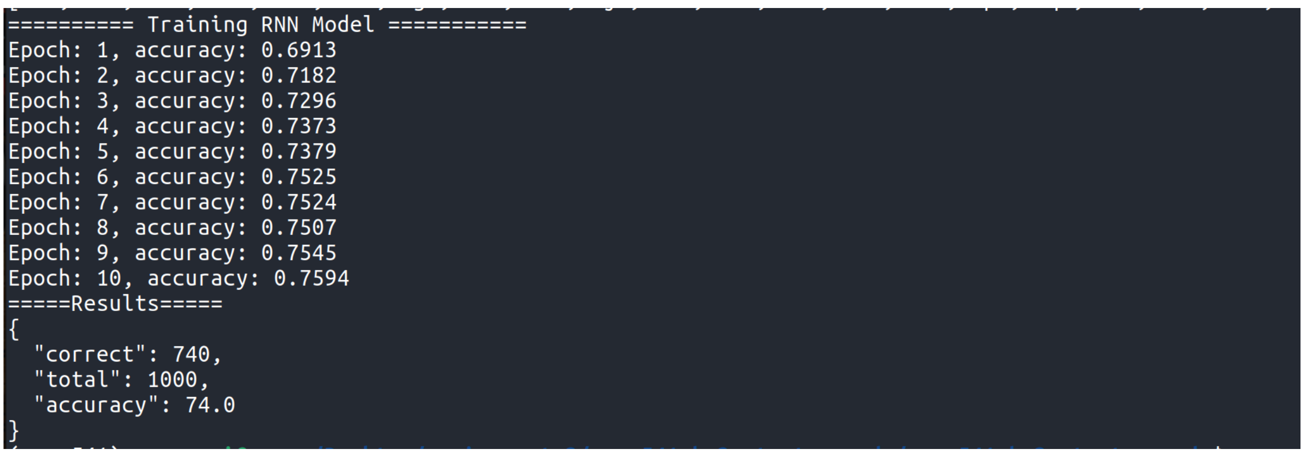
```
========== Training RNN Model ===========
Epoch: 1, accuracy: 0.6771
Epoch: 2, accuracy: 0.7117
Epoch: 3, accuracy: 0.725
Epoch: 4, accuracy: 0.7318
Epoch: 5, accuracy: 0.7317
Epoch: 6, accuracy: 0.7454
Epoch: 7, accuracy: 0.7531
Epoch: 8, accuracy: 0.7473
Epoch: 9, accuracy: 0.7493
Epoch: 10, accuracy: 0.7568
=====Results=====
{
  "correct": 740,
  "total": 1000,
  "accuracy": 74.0
}
```

Figure 9. Part1: Experiment 9

Inspired from the earlier experiment, I have decided to increase the model capacity even further by increasing both the *word_embedding_dim*, and *gru_hidden_dim*. I have also tried this experiment with a smaller learning rate value. The results came out to be slightly worse than the earlier experiment but I alluded it to using a smaller learning rate for this iteration.

A.1.10. EXPERIMENT 10

```
# Hyperparameters ==============================
word_embedding_dim = 200
gru_hidden_dim = 300
lr = 5e-4
epochs = 10
model = RNNClassifier(vocab_index, word_embedding_dim, gru_hidden_dim)
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr)
# ==============================================
```



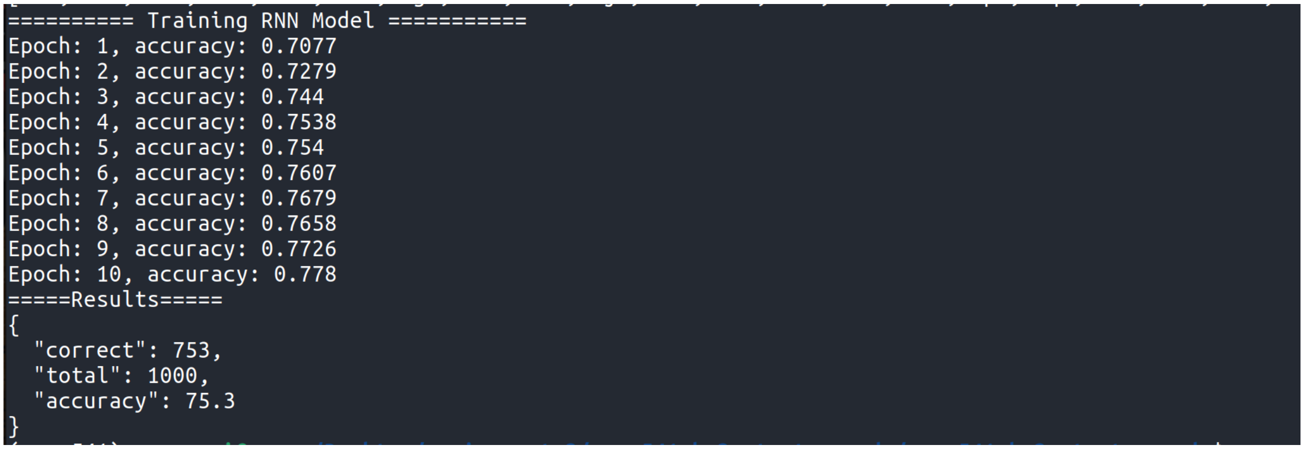
```
========== Training RNN Model ===========
Epoch: 1, accuracy: 0.6913
Epoch: 2, accuracy: 0.7182
Epoch: 3, accuracy: 0.7296
Epoch: 4, accuracy: 0.7373
Epoch: 5, accuracy: 0.7379
Epoch: 6, accuracy: 0.7525
Epoch: 7, accuracy: 0.7524
Epoch: 8, accuracy: 0.7507
Epoch: 9, accuracy: 0.7545
Epoch: 10, accuracy: 0.7594
=====Results=====
{
  "correct": 740,
  "total": 1000,
  "accuracy": 74.0
}
```

Figure 10. Part1: Experiment 10

In this experiment I wanted to see what would increasing (i.e. doubling) *word_embedding_dim*, and *gru_hidden_dim* even further could get me. The results came to be very close to the results of the earlier experiment. I suspected that by increasing the learning rate I could learn representations quicker and achieve better performance while being backed up by the increased representation capacity.

A.1.11. EXPERIMENT 11

```
# Hyperparameters ==============================
word_embedding_dim = 200
gru_hidden_dim = 300
lr = 5e-4
epochs = 10
model = RNNClassifier(vocab_index, word_embedding_dim, gru_hidden_dim)
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr)
# ==============================================
```



```
========== Training RNN Model ===========
Epoch: 1, accuracy: 0.7077
Epoch: 2, accuracy: 0.7279
Epoch: 3, accuracy: 0.744
Epoch: 4, accuracy: 0.7538
Epoch: 5, accuracy: 0.754
Epoch: 6, accuracy: 0.7607
Epoch: 7, accuracy: 0.7679
Epoch: 8, accuracy: 0.7658
Epoch: 9, accuracy: 0.7726
Epoch: 10, accuracy: 0.778
=====Results=====
{
  "correct": 753,
  "total": 1000,
  "accuracy": 75.3
}
```

Figure 11. Part1: Experiment 11

Inspired by the results of the previous experiment, in this experiment I tried using higher learning rate. As I suspected, the results improved and I surpassed the 75% accuracy threshold for the first time with this model. Even though, this model was enough for the purposes of this assignment I went ahead and tried to improve further with the my findings so far.

A.1.12. Experiment 12

```
# Hyperparameters ==============================
word_embedding_dim = 200
gru_hidden_dim = 300
lr = 5e-3
epochs = 10
model = RNNClassifier(vocab_index, word_embedding_dim, gru_hidden_dim)
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr)
# ==============================================
```

```
========== Training RNN Model ===========
Epoch: 1, accuracy: 0.7246
Epoch: 2, accuracy: 0.7567
Epoch: 3, accuracy: 0.7626
Epoch: 4, accuracy: 0.7799
Epoch: 5, accuracy: 0.7786
Epoch: 6, accuracy: 0.7896
Epoch: 7, accuracy: 0.8023
Epoch: 8, accuracy: 0.8048
Epoch: 9, accuracy: 0.8166
Epoch: 10, accuracy: 0.824
train_rnn_classifier function took 583.090 seconds to execute.
=====Results=====
{
  "correct": 776,
  "total": 1000,
  "accuracy": 77.60000000000001
}
```

Figure 12. Part1: Experiment 12

Building on top of the last experiment, I wanted to try increasing the learning rate (*lr*) parameter. I though if I increased the learning rate and kept the rest of the parameters the same, then I could learn quicker and achieve higher performance. Things went according to my plan and it resulted in even better performance compared to my earlier experiment. This model achieved 77.6% accuracy with $583.090 seconds$ for executing the *train_rnn_classifier* function. This model is my final and best model which also surpasses the required accuracy threshold mentioned in the assignment instructions. Finally, through my experiments I observed that increasing the model capacity by increasing *word_embedding_dim*, and *gru_hidden_dim* results in longer training times for the model.

A.2. Part1: Experiments

A.2.1. Experiment 1

```
# Hyperparameters ================================
chunk_size = 20 # determines the sequence lengths of input samples used during training
word_embedding_dim = 200
gru_hidden_dim = 300
lr = 5e-3
epochs = 10
model = RNNLanguageModel(vocab_index, word_embedding_dim, gru_hidden_dim)
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr)
# ===============================================
```

```
========== Training RNN Model ===========
Epoch: 1, loss: 2.5611283367313034, perplexity: 13.879529783695695, accuracy: 0.2658231556415558
Epoch: 2, loss: 2.284076618156235, perplexity: 10.886357840175203, accuracy: 0.3169133961200714
Epoch: 3, loss: 2.1902959565206346, perplexity: 10.13365402537951, accuracy: 0.3426985442638397
Epoch: 4, loss: 2.1207262852306865, perplexity: 9.620592310416448, accuracy: 0.36384275555610657
Epoch: 5, loss: 2.0630730205021948, perplexity: 9.232225040526814, accuracy: 0.3817363381385803
Epoch: 6, loss: 2.013600833536649, perplexity: 8.916416587628627, accuracy: 0.39466893672943115
Epoch: 7, loss: 1.9704663440284074, perplexity: 8.651513648134404, accuracy: 0.40714141726493835
Epoch: 8, loss: 1.932074408121981, perplexity: 8.418822010504822, accuracy: 0.41744348406791687
Epoch: 9, loss: 1.897165869230937, perplexity: 8.210898117319974, accuracy: 0.426325261592865
Epoch: 10, loss: 1.8652049037999547, perplexity: 8.026777577600818, accuracy: 0.435787171125412
train_lm function took 1020.665 seconds to execute.
=====Results=====
{
  "sane": true,
  "normalizes": true,
  "log_prob": -914.96630859375,
  "avg_log_prob": -1.8299326171875,
  "perplexity": 6.233466615860848
}
```

Figure 13. Part2: Experiment 1

I decided to use the hyperparameters I found from A.1.12. It's training took (*train_lm function*) 1020.665 seconds to execute. It has passed the *sane*, *normalizes* checks. Achieved $log\_prob : -914.966$, $avg\_log\_prob : -1.83$, and $perplexity : 6.23$ which passes the requirements set in the instructions of the part 2 of the assignment. Due to this I stuck with this model and did not experiment further.