ECS 189G-001

Deep Learning

Winter 2024

Course Project: Stage 4 Report

Team Information

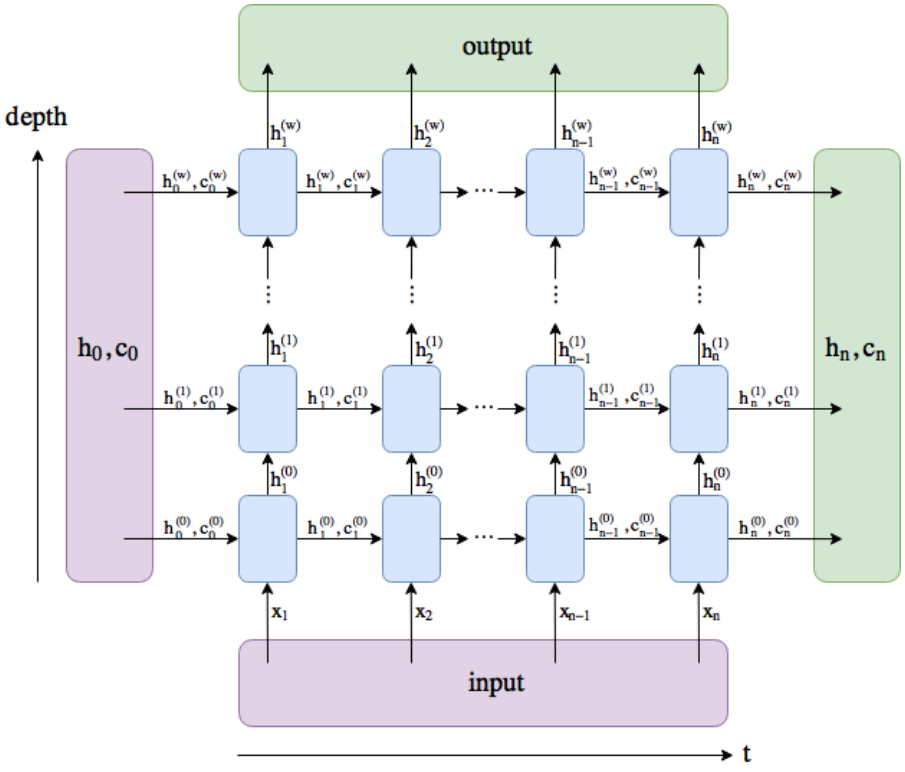
Denis Savytski	920000274	dsavytski@ucdavis.edu

Section 1: Task Description

Train 3 RNN models: regular RNN, LSTR, and GRU on the text Movie reviews and Jokes dataset. Movies reviews dataset aims to teach the model to distinguish negative vs positive reviews of the movies, and the Jokes dataset's goal is to teach the model how to create jokes. Experiment with different activation functions, loss functions, adding dropouts, different learning rates, and batch sizes affect accuracy and loss

Section 2: Model Description

All of the models are versions of RNN, a general image is provided below:



RNN and GRU do not have extra c0, and cT vectors that they have to carry around. All of the models initially utilize only one hidden layer. The generation dataset has only three inputs and the corresponding number of blocks in the hidden layer is also 3. The model tries to predict the fourth word based on the previous three. Also, the basic model only utilizes one output vector from the model, the very last one generated by h3, and applies a linear transformation to convert it to the probabilistic vector, to further determine the generated word. Text generation uses longer chains, 50 in our case. Therefore only 50 first

words of each sentence are taken into account, while the rest is ignored. (Even though I would have preferred to have longer chains, my machine does not possess enough RAM, and the length of chains directly affects the time it takes to train the model. Also, I assume that 50 words should be enough to understand whether the label is positive or negative). The basic model also discards all of the previous output vectors and works only with the last h50 vector, which is then transformed using Linear Multiplication into a probabilistic vector of size 2 (pos /neg), also sigmoid is applied to convert this into probability. All of these basic models act as a test ground for the model, experiments with hyperparameters are demonstrated in 3.7.

Section 3: Experiment Settings

3.1 Dataset Description

Data Generation:

The dataset contains 1622 pieces of short jokes. We process the data using regexp and other built-in modules for working with text in Python (Script is provided as part of the custom data set class). We transform the data, remove all of the punctuation signs, lowercase all of the remaining text, and stem the words. Furthermore, the data is divided into arrays, each containing a consecutive sequence of 4 words from each sentence. For example “the bird has wings.” -> [[“the”, “bird”, “has”, “wings”], [“bird”, “has”, “wings”, “.”], etc., the first three words of each sequence become the input for the model, while the last word becomes the label. The only separation that was left is the “.” sign, which indicates the end of the sentence and is also predicted by the model, so it knows when to stop text generation. The transformed dataset is then split into training and test data sets with a ratio of 70:30, with a fixed seed.

Data Classification:

The dataset contains 50000 files (25000/25000 train/test split). Each file contains a label of the review - its score, which can either be positive or Negative and the review itself. The data is processed, all of the punctuation marks have been removed, the text is lowercase, and a stemming was also applied. Furthermore, each review was transformed into an array of words.

3.2 Detailed Experimental Setups

We work with 3 different models, which are RNN, LSTM, and GRU. Each has similar hyperparameters. Number of layers, number of parameters in each hidden layer, number of layers in input vectors (which are adjusted throughout built-in embedding), and length of chain (number of blocks in each layer). Default models used in the project have the same parameters for each of the three models: RNN, LSTM, and GRU

	Generation	Classification
Number of layers	1	1
Chain length	3	50
Input size	250	250
Hidden size	500	500

CrossEntropyLoss function and Adam optimizer are utilized for every model, the learning rate and number of epochs differ slightly from model to model. Also, the dropout has not been utilized for the first version of the model (with 1 layer)

3.3 Evaluation Metrics

In the experiment, we will use Accuracy, Precision, Recall, and F1 score as our evaluation metrics.

The generation models are not expected to have very high accuracy, instead, they are expected to produce a readable, meaningful sentence. Even though the metrics for them are also provided, the main criteria are the generated joke produced by the machine would be judged by the human who is testing the model.

3.4 Source Code

https://github.com/BoyyyxD/ecs189_w2024

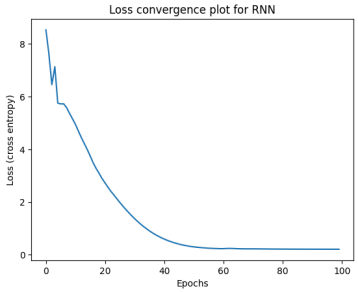
(All of the data loaders work with preloaded files of transformed data, therefore to run the code, one would have to run helper functions inside of each of the dedicated datasets)

3.5 Training Convergence Plot

3.6 Model Performance

Generation:

1) **RNN**




```
Metrics for model Generation RNN on train data:
-- Accuracy: 92.40 %
-- Recall: 97.43 %
-- Precision: 97.22 %
-- F1: 97.33 %
-- Loss: 0.20
```

```
Metrics for model Generation RNN on Test data:
-- Accuracy: 17.79 %
-- Recall: 41.49 %
-- Precision: 65.05 %
-- F1: 50.67 %
-- Loss: 21.88
```

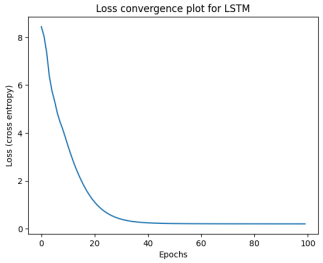
Joke: “`fish walks into a bar and orders a martini the bartender asks the ghost of a chicken a poultrygeist`”

2) **GRU:**

```
Metrics for model Generation GRU conf 0 on Test data:
-- Accuracy: 19.19 %
-- Recall: 46.10 %
-- Precision: 61.67 %
-- F1: 52.76 %
-- Loss: 8.24
```

```
Metrics for model Generation GRU conf 0 on Train data:
-- Accuracy: 92.40 %
-- Recall: 97.51 %
-- Precision: 96.70 %
-- F1: 97.10 %
-- Loss: 0.25
```

3) **LSTM**:

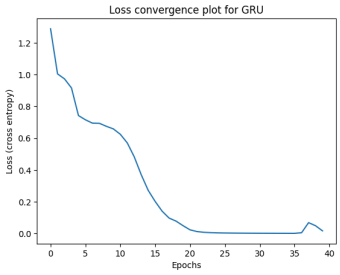



```
Metrics for model Generation LSTM on Test data:
-- Accuracy: 18.21 %
-- Recall: 42.02 %
-- Precision: 61.04 %
-- F1: 49.77 %
-- Loss: 9.89
```

```
Metrics for model Generation LSTM on Train data:
-- Accuracy: 92.40 %
-- Recall: 97.64 %
-- Precision: 96.99 %
-- F1: 97.31 %
-- Loss: 0.20
```

Classification

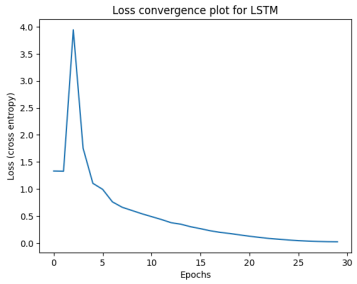
1) **GRU:**



```
Metrics for model Classification GRU on Test data:
-- Accuracy: 53.38 %
-- Recall: 76.61 %
-- Precision: 50.02 %
-- F1: 60.53 %
-- Loss: 2.24
```

```
Metrics for model Classification GRU on Train data:
-- Accuracy: 99.26 %
-- Recall: 99.97 %
-- Precision: 92.27 %
-- F1: 95.96 %
-- Loss: 0.01
```

2) LSTM




```
Metrics for model Classification LSTM on Test data:
-- Accuracy: 55.82 %
-- Recall: 77.84 %
-- Precision: 50.00 %
-- F1: 60.89 %
-- Loss: 1.85
```

```
Metrics for model Classification LSTM on Train data:
-- Accuracy: 99.06 %
-- Recall: 99.87 %
-- Precision: 75.26 %
-- F1: 85.83 %
-- Loss: 0.02
```

3.7 Ablation Studies

The length of the chain is bounded by our data, therefore we will not touch it. Instead, we will play with hidden layer sizes, embed sizes and number of layers.

First, we want to play around with the number of layers. We will change the number of layers in the GRU model to two, and add a .2 dropout to see how this will change the performance. While this hasn't changed the accuracy significantly, we got an improvement in the joke:

"`fish walks into a bar and says ill have a burger for a week`"

Therefore we will stick to at least 2 layers for the rest of the studies. Next, we want to understand how changing the embed size will affect the quality, we will reduce the embed size to 100. In my case, I struggled to have enough RAM to increase the size of the hidden layer, therefore if an improvement or at least nont significant decrease in performance can be seen after decreasing ember size, I could use "freed" memory to increase the hidden layer's size. This hasn't really led to any significant changes. Next, we will decrease the number of parameters in the hidden layer (to 100), while increasing the number of layers to 5. From the graph provided in the report, it seems that the model struggles to converge. Therefore decreasing the dimensions of the hidden layer leads to a decrease in the performance. Which also hasn't yield an imporvement in the performance. Overall, it is hard to work around with given models, as they are very comutationally expensive and trainign moedls takes very long. Over the period of working with the model, the best generated joke (generated using configuration 1 of GRU):

```
fish walks into a bar and orders a martini the bartender asks what is this some kind of joke
```