Ian MacDonald

CIS 530

Homework 6 Writeup

City Name Prediction

Description of Models Tried

For the city name prediction part of the homework, I experimented with a few different types of models. I tried varying the learning rates, number of epochs, loss functions, optimizers, weight initializations, and architectures. Below is a table documenting all of the different combinations of networks I tried, along with the resulting final training/validation accuracies.

LR	Epochs	Loss	Optim	Weights	Architecture	Train	Val
.002	75k	NLL	SGD	Zeros	Hidden (Size=50), Linear, Softmax	45%	43%
.0005	200k	NLL	SGD	Zeros	Hidden (50), Linear, Softmax	46.2%	45.6%
.0005	250k	NLL	Adam	Zeros	Hidden (50), Linear, Softmax	55.94%	54.2%
.0005	200k	CE	Adam	Zeros	Hidden (50), Linear	56.4%	53.56%
.0005	200k	NLL	Adam	Zeros	Dropout, Hidden (100), Linear, Softmax	38.35%	39%
.0005	200k	NLL	Adam	Zeros	Hidden (100), Linear, Softmax	54.16%	53.3%
.0005	200k	NLL	Adam	Random	Hidden (75), Linear, Softmax	53.27%	49.4%
.0005	200k	NLL	Adam	Zeros	LSTM (50), Linear, Softmax	76.95%	70.2%
.0005	200k	NLL	Adam	Zeros	GRU (50), Linear, Softmax	77.12%	70.2%
.0005	200k	NLL	Adam	Zeros	Two Layer GRU with Dropout (size=50, drop=.3), Linear, Softmax	76.49%	71.2%
.0005	200k	NLL	Adam	Zeros	Three Layer GRU with Dropout (size=50, drop=.5), Linear, Softmax	68.5%	65%
.0005	200k	NLL	Adam	Zeros	Two Layer GRU with Dropout (size=50, drop=.5), Linear, Softmax	74.37%	68.1%

Discussion on Hyperparameters and Effect on Various Models

The above table shows all the different combinations of model architectures and hyperparameters I tried. While it would have been best to adjust hyperparameters/architectures one at a time to isolate the effect it had on the final accuracy, each run took about 15 minutes, give or take depending on the epochs, layers, etc., which meant doing a true hyperparameter sweep would have been relatively cumbersome and time consuming, so you can see above there are cases where two or sometimes three elements of the model are changed from run to run to mitigate this. I used the resulting plots and accuracies from the previous models, along with general knowledge of RNNs/Neural Nets to decide what models to implement at each step.

I found that changing certain elements of the model lead to larger changes in accuracy. Firstly, decreasing the learning rate and increasing the epochs led to an increase in validation accuracy, which is why most of the models above use a LR of .0005 and at least 200k epochs. I used SGD optimization at first, but saw a large increase in accuracy when switching to Adam, so I kept Adam for the rest of the runs. Switching the loss function to cross entropy and getting rid of the softmax layer didn’t change much, but it dropped the validation accuracy a little bit so I just went back to using NLL and a softmax layer.

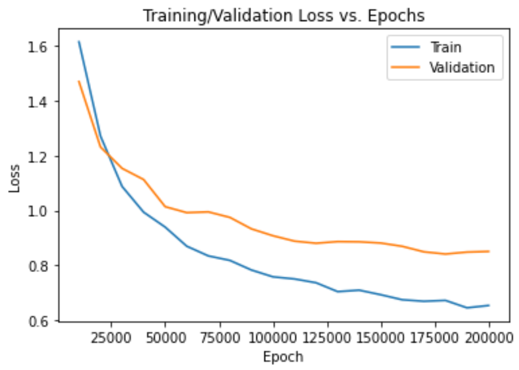
My earlier models all seemed to be overfitting a little bit, so I added a dropout layer to try and combat this. However, it ended up dropping my accuracy by a substantial amount, so I decided to go without it. Increasing the size of the hidden layer, as well as changing the initial weights to be small random numbers sampled from a normal distribution did not change much, but it also dropped the accuracy a little bit, so I decided to go back to zeroes and a hidden size of 50. The biggest increase in accuracy came when switching from a basic RNN to a LSTM RNN, which caused the validation accuracy to go from around 50% to around 70%. After the success of the LSTM RNN, I gave a GRU RNN a shot, and got identical validation accuracy along with a slightly higher training accuracy. It seemed that there wasn't much of a difference in performance between LSTM and GRU, but both of them performed substantially better than a simple RNN, so I decided to go with a GRU RNN for the rest of the models.

Like the previous models, the GRU RNN seemed to have some issues with overfitting, as the training accuracy was a few points higher than the validation accuracy. To combat this, I decided to change the GRU layer to a two-layer stacked GRU, and gave the dropout layer another go, this time adding a dropout probability of .3 to the first GRU layer. This worked far better than the first dropout layer regularization attempt, as it decreased the training accuracy slightly but increased the validation accuracy slightly as well. Because it worked, I gave it another run, this time increasing the dropout probability to .5 and adding a third GRU layer. However, this dropped the validation accuracy to about 65%, so for the final model I tested, I went back to a two-layer stacked GRU, with a dropout probability of .5, which gave a slightly higher validation accuracy, but still not the best.

Final Model Architecture, Performance, and Error Analysis

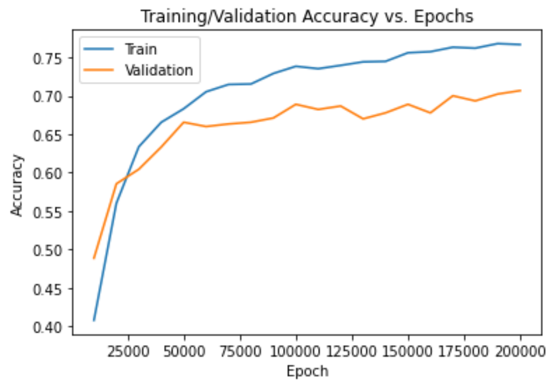
Ultimately, the final model I decided on had a learning rate of .0005, NLL loss, Adam optimization, initial weights of 0 for the network's hidden layers, a model architecture consisting of a two-layer stacked GRU RNN (with dropout=.3 and hidden size=50) with a Linear and Softmax layer following it, and was trained for 200k epochs. I chose this model because it had the highest accuracy on the validation set. Below, are the training curves for this model, as well as a confusion matrix.

Loss

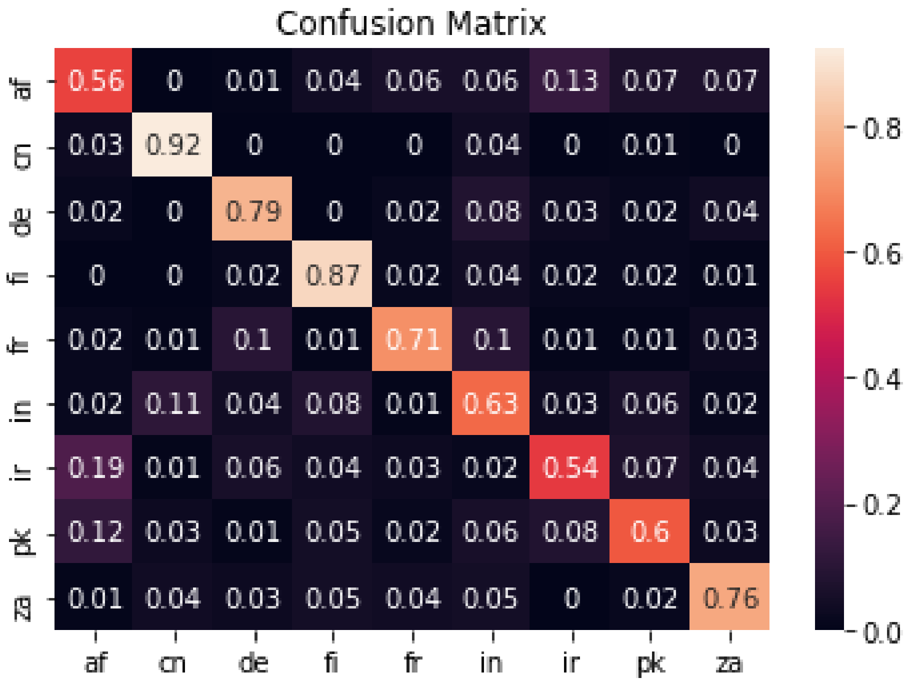


Accuracy

Note: Training accuracy was not calculated on the entire training set at the specified epochs, as that increased the run time substantially. Instead, the training accuracy is calculated example by example as the model trains, and is reset every time the training accuracy is logged, so it represents the training accuracy from the previous calculation epoch to the current calculation epoch.



Confusion Matrix



	af	cn	de	fi	fr	in	ir	pk	za
af	0.56	0	0.01	0.04	0.06	0.06	0.13	0.07	0.07
cn	0.03	0.92	0	0	0	0.04	0	0.01	0
de	0.02	0	0.79	0	0.02	0.08	0.03	0.02	0.04
fi	0	0	0.02	0.87	0.02	0.04	0.02	0.02	0.01
fr	0.02	0.01	0.1	0.01	0.71	0.1	0.01	0.01	0.03
in	0.02	0.11	0.04	0.08	0.01	0.63	0.03	0.06	0.02
ir	0.19	0.01	0.06	0.04	0.03	0.02	0.54	0.07	0.04
pk	0.12	0.03	0.01	0.05	0.02	0.06	0.08	0.6	0.03
za	0.01	0.04	0.03	0.05	0.04	0.05	0	0.02	0.76

Error Analysis

As shown by the training curves, the training accuracy and training loss are consistently better than the validation accuracy and validation loss, indicating that this model is still overfitting, even after adding the dropout layers. Additionally, a look at the confusion matrix shows that the model is not performing evenly across all the classes, with some having significantly higher accuracies than others. The model performs extremely well on China and Finland (92%, 87% TP rates, respectively), but performs poorly on Afghanistan, Iran, and Pakistan (56%, 54%, 60% TP rates, respectively). A big reason the model performs poorly on those 3 countries in particular is that the model confuses them with each other. When the correct label is Iran, Afghanistan was predicted 19% of the time, the largest single miss rate of any country pairing. A similar problem is encountered for Afghanistan and Pakistan, as well as for Pakistan and Iran. I believe that the main cause of this is that some of the countries chosen are from similar geographic regions, and thus will have similar languages, making it hard to distinguish city names for those countries. The model performs very well on China, but there are also no other Eastern Asian countries included in the data. Similarly, the model performs very well on Finland, but there are also no other Scandinavian counties included either. Had the data included, for example, South Korea and Sweden, the model may have had a bit more trouble predicting China and Finland, respectively.

Final Model Discussion and Testing Accuracy

Ultimately, the final model had some issues, particularly with overfitting and imbalanced class accuracy, but overall still performed pretty well. When submitted to the Autograder, the model had a 72.78% accuracy rate on the testing set, which was pretty good. I think there is room to improve for the model, particularly in addressing overfitting, since I think that is an easier task to handle compared to addressing the imbalanced class accuracy, which I believe at least partly arises simply due to the nature of the data set we are using.

Text Generation

Description of Training/Testing Sets

For the text generation section of this homework, I decided to train a model using the script from the movie *Rush Hour 2*, an action-comedy film starring Jackie Chan and Chris Tucker. To see how well the model worked, I tested it against a couple of other movie scripts, as well as against a famous book. The movie scripts were all pulled from https://figshare.com/projects/imsdb_movie_scripts/18907, and it should be noted that they mention on their website that “the original scripts were uploaded on the website by individual users, so they might not correspond exactly to the movie scripts and typos may be present”, as well as “html formatting was not consistent in the website, so neither is the formatting of the resulting text files”. With both of those noted though, they do also say “the quality seems good on average and the dataset can easily be used for text-mining tasks”. After my experience with the website, I found that last statement to be relatively accurate. The scripts I decided on were a little messy formatting-wise, but otherwise pretty good. However, I did run into some issues with the text files that I will elaborate on below. The book text was pulled from http://www.gutenberg.org/ebooks/829, and seemed to be of pretty good quality.

The two movies I chose to test against were *Good Will Hunting* and *Pirates of the Caribbean*. Originally, the plan was to test *Rush Hour 2* against the original *Rush Hour*, as the sequel should be pretty similar to the original. However, after looking into their script for *Rush Hour*, it looked like someone had uploaded an incorrect script, so that plan had to be scrapped. Instead, *Good Will Hunting* was chosen as the “similar” script. While *Good Will Hunting* is a different genre (a drama), I figured they would still be somewhat comparable, especially compared to the other chosen texts. The second comparison movie I chose was *Pirates of the Caribbean*, because while it is still a movie script, the setting and general plot of the movie are pretty unique, and I figured this should be pretty different than *Rush Hour 2*. Finally, I chose the book *Gulliver’s Travels* so that I could have a comparison text that is of a completely different type of text.

Model Architecture

The model used was a GRU RNN with four layers. The first layer is a linear layer that takes as input a string represented as a tensor of character indices, then outputs the corresponding character embeddings. These embeddings are then fed into a stacked GRU layer, with two recurrent layers and a hidden size of 100. The output of the GRU layer is then fed into a dropout layer, with dropout probability .5. Finally, the output of that layer is fed into a linear layer that converts the output to a probability distribution that can be used to calculate the next character in the sequence. The network is trained for 3000 epochs, with a learning rate of .0002.

Example Generations

Each generation was done for 100 characters, with a random starting letter in the range A-Z.

Generation	Generated Text
1	`CARTER (CONT'D) You pertet prays. CARTER Waet the sty looks mac) As Stell - LEE Lor's back -- dekage.`
2	`CONS case. LEE The right cake on a most is their. LEE What the shoulding in the graps him. I ake out,`
3	`U PEADENTICLE as kids. MASTER FU, track clost buls. Carter still cinner. Lee partner Lee and Carter a`
4	`Y, my in his time somether. Konging in that dracks froff a prever the trignh. Carter track a prigfor`
5	`AS job of begul thousting the butn the LEODICE! Isatim you saman. LEE Srobe levill. Don't the it, Lee`
6	`X(you bodyelg. Lee stell chonese, is? I'm a way. All you deet Lee! Folds. You rew when grees to him w`
7	`HF ME RUGAW'S OMR Fu Carter! You and I understcans and wait Coume of him you do it/gons. Lee to Tan's`
8	`D. You along the rooms of a land. You's ever the try sboth. MAN Sut. EXT (CONT'D) It breath) Whan bef`
9	`FY Fu you go of a Carter and Lee are the have this in his are down His rich the trandon? EXT. CASINO`

Model Analysis and Perplexity Calculations

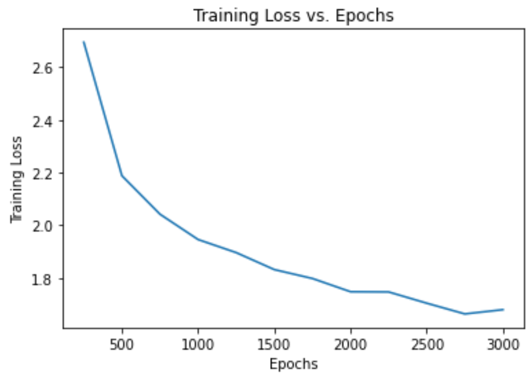
Clearly my model won't be replacing Hollywood script writers any time soon, but there is evidence that it is learning and outputting something at least somewhat reminiscent of the *Rush Hour 2* script. The generated text often has "Carter" and "Lee" written, which makes sense because Chris Tucker and Jackie Chan play the main characters, named James Carter and Inspector Lee. As they show up often in the training script, it makes sense they would show up often in the generated text as well. Additionally, there are a lot of actual words in the above text, and the text is structured relatively correctly. Sentences all pretty much begin with a capital letter, and punctuation marks are used to create somewhat believable sentence structures. However, many of the words above are still nonsensical.

Perplexity Table

Testing Set	Perplexity
Good Will Hunting	8.79366195573588
Pirates of the Caribbean	21.30371800778867

Gulliver's Travels	13.835917623383347

As expected, the perplexity was lowest when comparing the model to *Good Will Hunting*, which makes sense since both are movies with more or less "realistic" plots and settings, even though they fall under different movie genres. Additionally, *Gulliver's Travels* performs worse than *Good Will Hunting,* but still performs much better than *Pirates of the Caribbean.* This makes sense, as even though *Gulliver's Travels* is a different format and a different premise than *Rush Hour 2,* it's still not as wildly different as *Pirates of the Caribbean* is, which is filled with pirate-talk that just doesn't show up at all in any of the other texts. The model performed as expected in terms of perplexity, and somewhat well in terms of text generation, although there is still a ton of room to improve. Below is the loss curve for the model during training.



The training loss seems to level off at the end, and even spikes up a little bit, indicating that it is unlikely that further training with this model and training set will lead to much improvement. Therefore, to improve results, I think either different models should be tested, or more training data should be added. Additionally, this training data was a little messy formatting-wise, and cleaning that up may help improve the model's performance.

Comparison to N-gram Models

Compared to an N-gram model, the RNN performed significantly better. Below, is a snippet of text from the N-gram model trained on the *Rush Hour 2* script, followed by the perplexity calculations for both the N-gram model and the RNN for comparison.

```
RUSH HONG (O.S. RICAN I'm a crates, wall left. HOSTER Two fathrows at the
grange. LEE show I hank some to can frame off a hund. EYES the thas bait
FLIES Oh no isn't knoculous way assive you? Throught steeliever rough, who
want, tapes door, that's mor
```

Testing Set	Perplexity (RNN)	Perplexity (N-gram)
Good Will Hunting	8.79366195573588	21.350232210455403
Pirates of the Caribbean	21.30371800778867	34.51391081608932
Gulliver's Travels	13.835917623383347	24.733350818400552

While the text snippet doesn't look too bad, it doesn't have the same resemblance to the script that the RNN model's text snippets have. It has "Lee" mentioned, but no mention of "Carter". Additionally, this snippet is also filled with a large amount of nonsensical words. The difference is easier to see when looking at the perplexity calculations, where the perplexity is higher across the board for all test sets when using the N-gram model. Therefore, I think it is safe to conclude that the RNN is a much better model than the N-gram model in this case.