CIS 530 Assignment 6: Neural Language Models

Ignacio Arranz
Yezheng Li

Part I

Part II: Classification Using Character-Level Recurrent Neural Networks

Problem 2.1. *Write code to output accuracy on the validation set. Include your best accuracy in the report. (For a benchmark, the TAs were able to get accuracy above 50%) Discuss where your model is making mistakes. Use a confusion matrix plot to support your answer.*

Answer.

- ***Write code to output accuracy on the validation set. Include your best accuracy in the report.***

  Our best accuracy on the validation set is about 61.4 - 62.9%.

- ***Discuss where your model is making mistakes. Use a confusion matrix plot to support your answer.***

  We perform best in 'cn' (Chinese, I think most of the team also perform best in 'cn'), while our second best label (true value) is 'de' (Deutsch, German). Our worst predicted labels are 'in' and 'fi'.

  To see our mistakes in city-name level, for cityname wit true value 'in', 'fi' (worse predicted label), for example, we predict 'aanaar' as 'af' ('fi'), 'kinigi' as 'za' ('in'). For better predicted label (like 'cn', 'de'), we correctly predict 'qiujiapo', 'baoguo' as 'cn', but predict 'priego', 'chiadede' as 'de', but incorrectly predict 'cukurdere' as 'fr' ('de'). My explanation for 'de' being predicted incorrectly is that 'fr' and 'de' are to some extent similar.

  See below for one of our final confusion matrices in table 2 and figure 1.

Problem 2.2. *Modify the training loop to periodically compute the loss on the validation set, and create a plot with the training and validation loss as training progresses. Is your model overfitting? Include the plot in your report.*

True label/ predicted label	'in'	'pk'	'fr'	'af'	'cn'	'za'	'fi'	'ir'	'de'
'in'	0.51	0.07	0.05	0.05	0.09	0.02	0.07	0.05	0.09
'pk'	0.07	0.62	0.01	0.15	0.02	0.03	0.02	0.06	0.02
'fr'	0.05	0.00	0.67	0.04	0.00	0.04	0.05	0.01	0.14
'af'	0.05	0.17	0.04	0.50	0.01	0.07	0.04	0.08	0.04
'cn'	0.09	0.01	0.00	0.00	0.87	0.00	0.02	0.01	0.00
'za'	0.08	0.04	0.08	0.08	0.04	0.60	0.04	0.03	0.01
'fi'	0.03	0.03	0.03	0.12	0.00	0.00	0.72	0.02	0.05
'af'	0.01	0.18	0.03	0.19	0.01	0.07	0.03	0.39	0.09
'de'	0.06	0.03	0.08	0.02	0.00	0.03	0.01	0.04	0.73

Table 1: Confusion matrices (table). Each row represents for each true label class (summation of each row is 1). Each column represents predicted label.

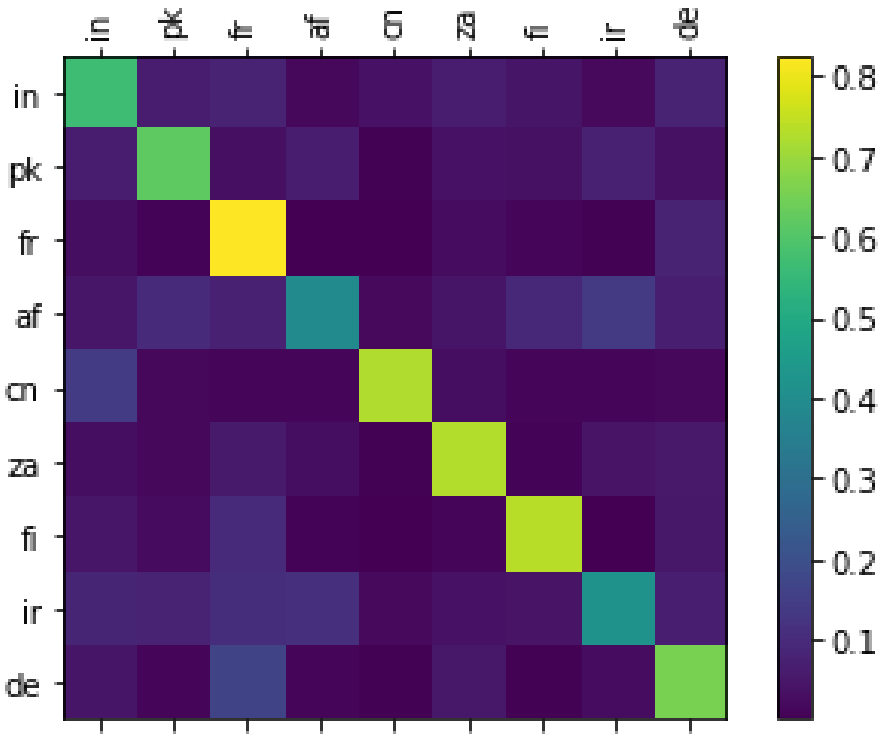

Figure 1: One of our resulting confusion matrices (plot).

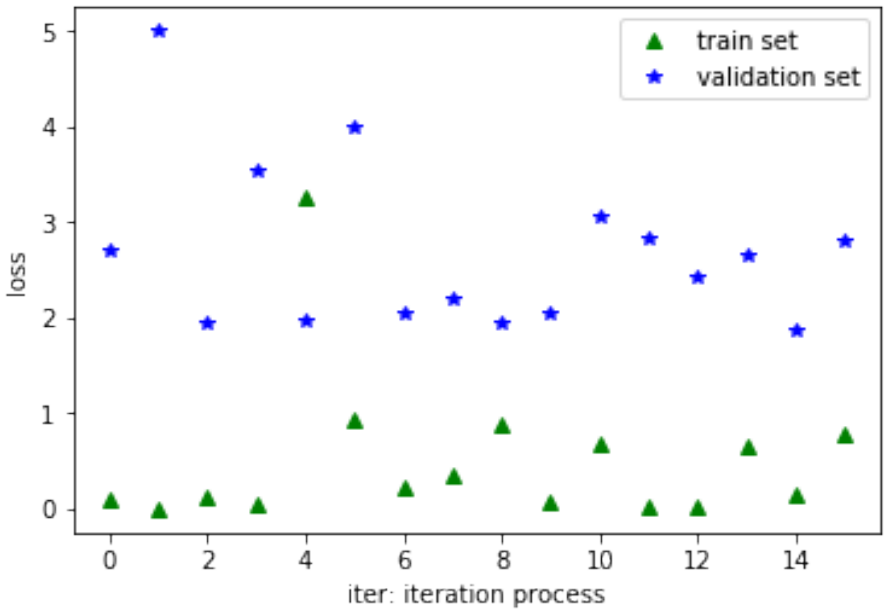

Figure 2: Negative log-likelihood Losses from training set and validation set versus training epochs. Losses from validation set are generally larger than ones from the training set.

Answer. ***Modify the training loop to periodically compute the loss on the validation set.***

We did it. By the way, in order to train neural network on 9 countries fairly with respect to order during the procedure, we train first 1-100th city names in 'in','pk',..., 'de' first, then train 101-200th city names in 'in','pk',..., 'de' in order, etc.

Create a plot with the training and validation loss as training progresses. Is your model overfitting? Include the plot in your report.

According to figure 2, losses from validation set are generally larger than ones from the training set. As a result, I think there is certain level of overfitting – however, this seems do not affect accuracy (on dev set, training set, test test, our RNN performs about the same – all about 62% accuracy See confusion matrix figure - model is not overfiting. By comparison, an overfiting seems happens in LSTM where when evaluating the accuracy from the sample just trained (see default code), the accuracy is very high.

Problem 2.3. *Experiment with the learning rate. You can try a few different learning rates and observe how this affects the loss. Another common practice is to drop the learning rate when the loss has plateaued. Use plots to explain*

your experiments and their effects on the loss.

Answer. ***Experiment with the learning rate. You can try a few different learning rates and observe how this affects the loss.***

Figure 2 plots negative log-likelihood Losses from training set versus training epochs. Comparison are made for various learning rates. Due to existing oscillations for each epoch, we smooth the losses by

- *splitting 100,000 epochs into 100 groups, each consisting 1,000 consecutive epochs' losses.*
- *Take the average over each group.*

Two comments are worthing mentioning:

(a) $\gamma \simeq 0.001$ *is about the optimal choice of learning rate.*

(b) Our losses do not decrease in the way as ones in default codes in Part II, Part III. An explanation is that we have much better pretrained character's vector embedding so that out plots generally skip some earlier period of decreasing.

Another common practice is to drop the learning rate when the loss has plateaued. Use plots to explain your experiments and their effects on the loss.

We did this, our figure 2 has two $\gamma = 0.001$*, three* $\gamma = 0.0001$*, without or with different levels of drop when the loss has plateaued. Our observation is that from loss perspectives (figure 2), we have not seen substantial benefit of dropping. However, when it comes to accuracy, dropping makes about 1% improvement in both dev set and test set.*

Problem 2.4. *Experiment with the size of the hidden layer or the model architecture How does this affect validation accuracy?*

Answer.

- *The larger the number of hidden layers, the more accuracy we get, the slower our program. when our accuracy reaches 58%. Accuracy does not increase much (for example, increase number of hidden layers from 128 to 200, we possibly about 0.01% on validation set and no much improvement for test set (leaderboard). On the other hand, the speed of training doubles.*
- *We put the pretrained feature vectors from trigram model (HW5). This drastically improves our result.*
- *We failed to implement LSTM correctly. As a result, LSTM leads to very bad accuracy result (14% for test set).*

Problem 2.5 (Leaderboard)**.** *Here are some ideas for improving your leaderboard performance:*

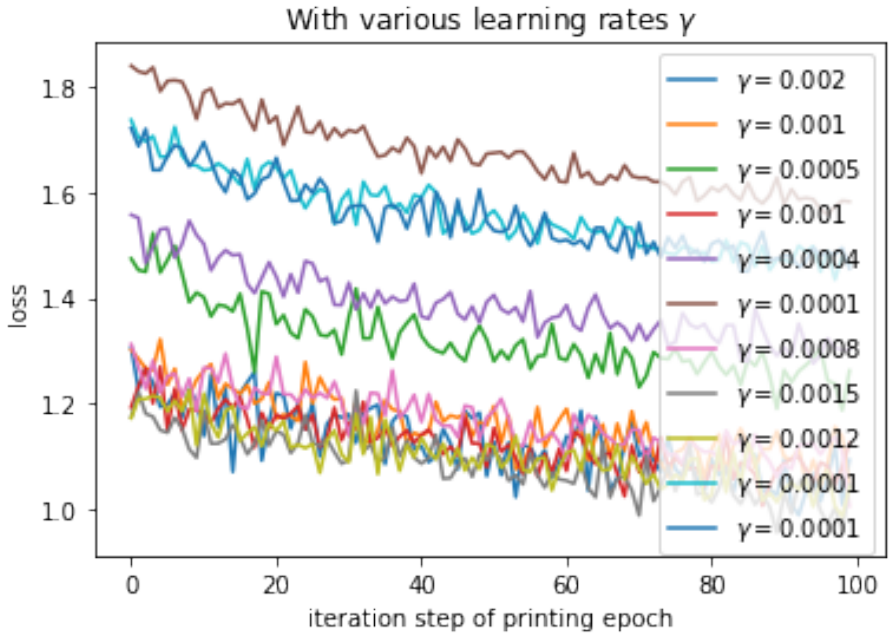

Figure 3: Negative log-likelihood Losses from training set versus training epochs. Comparison are made for various learning rates. Certain smoothing technique is applied to avoid serious oscillations of losses.

- *Play around with the vocabulary (the all_letters variable), for example modifying it to only include lowercase letters, apostrophe, and the hyphen symbol.*
- *Test out label smoothing.*
- *Try a more complicated architecture, for example, swapping out the RNN for LSTM or GRU units.*
- *Use a different initialization for the weights, for example, small random values instead of 0s*

In your report, describe your final model and training parameters.

Answer.

- ***Play around with the vocabulary (the all_letters variable), for example modifying it to only include lowercase letters, apostrophe, and the hyphen symbol.***

  We turn upper letters into lower letters; issues involving apostrophes, hyphen symbols, are treated intrinsically inside trigram model from HW5.

- ***Test out label smoothing.***

  We do not have to do much on this piece since our pretrained trigram vector embeddings are weighted (I mean, not just 0, 1).

- ***Try a more complicated architecture, for example, swapping out the RNN for LSTM or GRU units.***

  We tried both LSTM and GRU – neither of them have improvement, but worse performances (both for one-hot vectors and trigram vectors). We think we fail to implement either of the two correctly and have a guess that the three teams beating us (all above 70% in leaderboard) should have all successfully implemented either of the two.

- ***Use a different initalization for the weights, for example, small random values instead of 0s.***

  I think our trigram initialization/ pretrain is just working on this piece.

In your report, describe your final model and training parameters.

For the leaderboard we used pretrained trigram feature vectors in the default RNN. Training parameters are almost by default:

- *number of hidden layers is 128;*
- *learning rate is initially 0.001 but with certain drop: lr_temp =*

  $$learning\_rate * \max(pre\_err\_rate - curr\_err\_rate, 0.1) * 3$$

- *Input size is n_letters *n_categories which is essentially concatenation of each category's trigram feature embedding.*

Part III

Figure 4: MJ lyrics

Problem 3.1. *Include a sample of the text generated by your model, and give a qualitative discussion of the results. Where does it do well? Where does it seem to fail? Report perplexity on a couple validation texts that are similar and different to the training data. Compare your models results to that of an n-gram language model.*

We used Michael Jackson lyrics as input.

Song 1
Input Chunk 200, output size 300, start with 'A', temperature 0.4.

> All you
> That I dream
> The love want to race
>
> Dom Shanger
>
> Dom Shanget be me
> I want to could
> The on
> I am the are the falling
> The man is the love
>
> Dom Shance a could
> The one way you
> The on
> Dom Sheldon that's me
> Then I can the more chance a be man
> I want to rack
> Dance

We see in this song a large repetition of "Dom Sh..". This is likely because the chunk happened to contain the song D.S., which repeats "Dom Sheldon". The irony is that Dom Sheldon is said to be a reference to Tom Sneddon, the D.A. who investigated Michael Jackson for sexual abuse, so further variations of Dom Sheldon would continue to be references to Tom Sneddon.

The average length of each line is very much in line with Michael Jackson songs. This is basically representing how frequently the new line character appears.

The fact that the artificial song ends with a line saying "Dance" makes it great.

The generation seems to fail in always creating real words. It is surprisingly good despite being trained on bigrams.

Song 2
Input Chunk 1000, output size 300, start with 'A', temperature 0.4.

> Ad baby, baby
> To one wants to be down
>
> I'll be there to could be rear
> Perearth and out to face
> And is girl (yeah)
> He's and there
> And I was want
> I love what you got to get you
> I love with you
> I wanna saw fight
> I want it can see
>
> I am the one
> I want to be down
> I just can't

Calculations of perplexity:
The calculation of perplexity was done using the neural network model, calculating the probability that the neural network assigned to each character (following another character).
Our results for perplexity were as follows:

Perplexity for VERY POPULAR phrases in MJ songs:
"Oh oh oh oh": 1.043
"Billie Jean": 1.063
"beat it": 1.045

" I ": 1.015
"my girl": 1.04

Perplexity for phrases in artificial MJ songs:
"I wanna saw fight": 1.11
"Ad baby, baby": 1.09

Perplexity for phrases in MJ songs:
"where men are free": 1.14
"shake that thing girl": 1.14

We can see that logically perplexity increases as the length of the phrase (in characters) increases. It's interesting to see that the lowest perplexity is reached by words such as " I " (appears 254 times in this subset of lyrics). "my girl" has a low value, given the high rate of appearance of the words "my" (150) and "girl" (40). "beat it" as a sequence is repeated 49 times. So these low values are logical.
Some of the phrases generated by our neural network, also have relatively low values of perplexity, logically not as low as the TOP phrases we saw in the first place. To double check, we compared against other phrases that did show up in Michael Jackson songs, such as "where men are free" (appears once) and "shake that thing girl" (twice), both of which had higher perplexity. So this can help us understand that the neural network generated very reasonable results for its language model. The perplexity depends a lot on the specific chunk used for the modeling, so it's understandable if there are variations on the value of perplexity for different iterations of the same model.