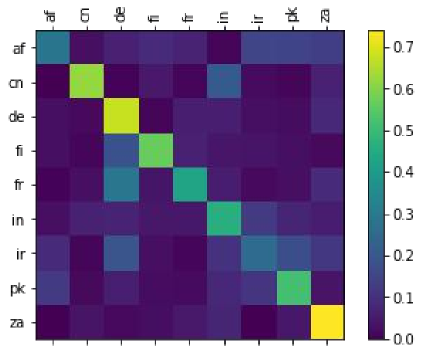
Part 1

(1) (**10 points**) Write code to output accuracy on the validation set. Include your best validation accuracy in the report. Discuss where your model is making mistakes and use a confusion matrix plot to support your answer.

For this part, my program computes both the micro and macro accuracy. Up to now, the best model I get has the **accuracy = (micro_acc: 0.50410002 macro_acc: 0.50733662)**, which uses the configuration:

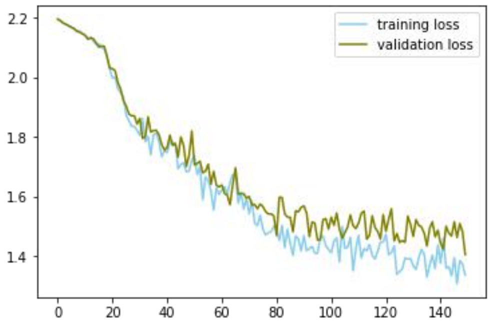
Hidden layer size = 512, LR = 0.001, Optimizer = SGD, n_epochs=100000



The Confusion Matrix is shown above. According to the matrix, the model performs very well classifying categories like de, za(> 0.6), but does badly on ir (< 0.4). In terms of ir, it might even confuse ir with pk since they have similar heat colors on the map.

(2) (**10 points**) 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.

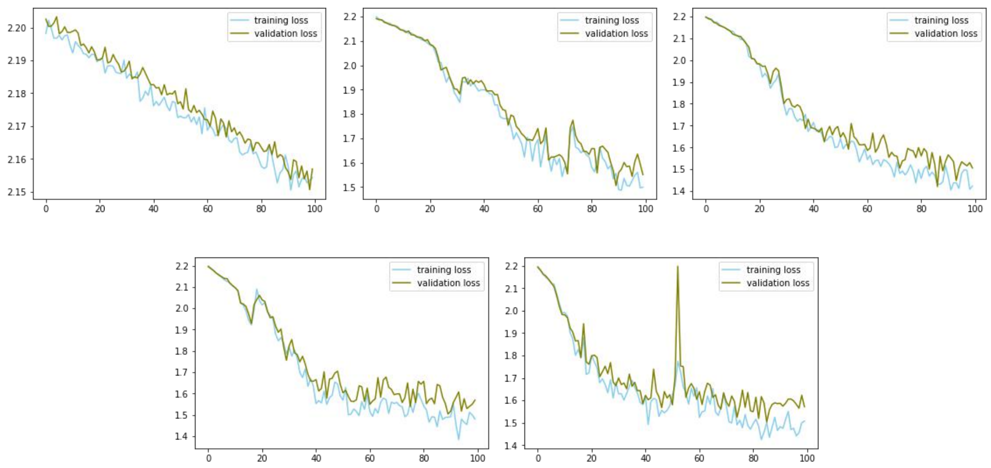
Here is the plot of training and validation losses from my best models. The training and validation losses are shown in the plot below.



According to the plot, it seems that the model works well since both the training and validation losses are reducing with the training process moves on. However, there might be a trend of overfitting since we can see the validation loss becomes more stable while the training loss still keeps decreasing, which causes two lines becomes separated in the late training process. In other words, if we ran more training on the model, it would be likely to become overfit.

(3) (**10 points**) Experiment with the learning rate (at least 5 different learning rates). You can try a few different learning rates and observe how this affects the loss. Use plots to explain your experiments and their effects on the loss.

The learning rates I tried during the experiment are: 0.0001, 0.001, 0.0015, 0.002, 0.003 while other configurations are remaining identical. Here are the loss plots I got. As a result, the best model among them is the one with RL=0.0015 (micro_acc: 0.4817, macro_acc: 0.48259291). The plots of them are shown below.

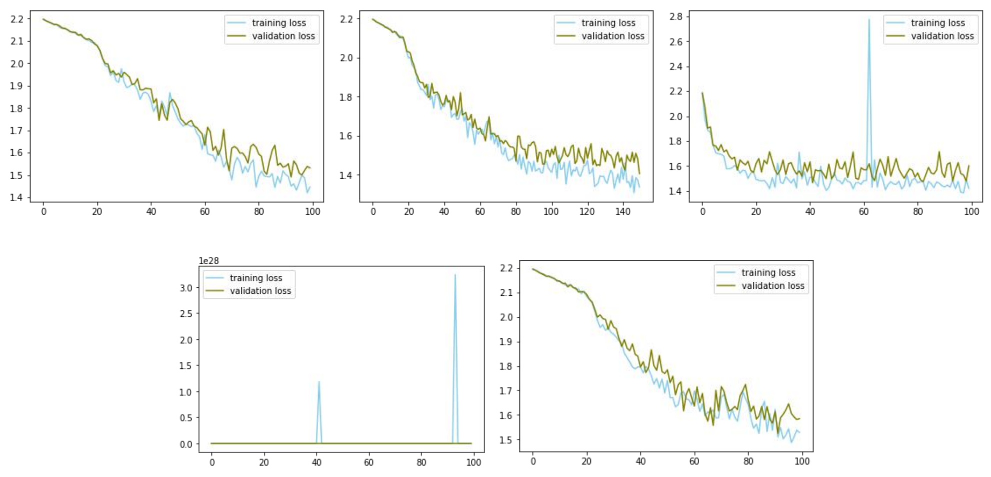


From left top to right bottom: RL=0.0001, 0.001, 0.0015, 0.002, 0.003

As we can see, when the RL is small, the gradient of the loss is slight. It's obvious in the first plot where it takes a long time but still stuck in the loss value around 2.16 while other models normally have reached below 2.0 at that time. However, if the RL becomes too large, then the losses become very unstable. Like in the final plot, there are huge turbulences in both training and validation losses and they may even jump back to the initial loss values (1.8 - 2.2), which indicates that we may overshoot during the optimization process. Overall, the RL between 0.001 and 0.003 seems to be a good choice in this case.

(4) (**10 points**) Experiment with the size of the hidden layer or the model architecture (at least 5 different sizes and/or modifications). How does this affect validation accuracy? (**Bonus 5 points** for those whose validation accuracies are in the top-3 of the class)

During the experiment, I have tried two different hidden layer sizes, 256 and 512 (FYI 128 is the default setting), as well as three different optimizers (SGD, Adam, RMSprop, and ASGD) while other configurations are remaining identical. Here are the results I got.



From left top to right bottom: hidden layer size = 256 and 512, optimizer = RMSprop, Adam, ASGD

	hidden layer size = 256	hidden layer size = 512	optimizer = RMSprop	optimizer = Adam	optimizer = ASGD
Accuracy (micro, macro)	0.47530001, 0.47619072	**0.50410002, 0.50733662**	0.49259999, 0.49076638	0.1547, 0.15520222	0.46160001, 0.45946229

As we can see, the performance (accuracy) is increasing while we add extra hidden layer nodes. Meanwhile, however, we have to run more epochs so that the model could convergent to a good result. In terms of the optimizer, RMSprop seems to work better than other optimizers under the current configuration, which reaches the accuracy of 0.49.

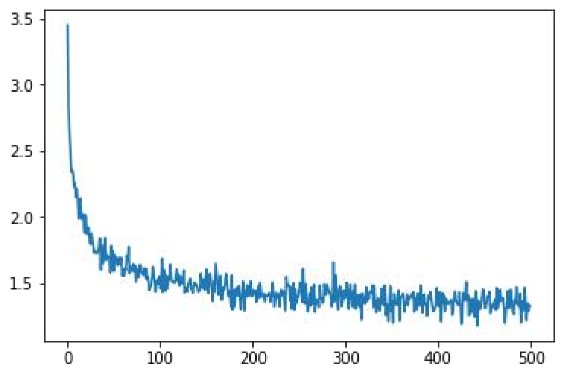
Part 2

(1) Include a sample of the text generated by your model (**5 points**), and give a qualitative discussion of the results.

For this part, the training data I used is a lyrics collection which contains lyrics from a number of different songs. To simplify the problem, the data contains only capital characters so that we only care about the relations among characters and words. The result is shown below, which with the prime_str="YOU".

YOU AND I CAN'T HAVE AND ELSE THAT JOLES
LIKE A SEE YOU WERE WITH THOUGH HER
BABY BLUT MINES
I GOT **I WANTED IT TO** SOLL IT AIN'T BE SOME YOU LOOKING THIS **BABY BABY BABY I WAN**
T YOU I DON'T YOU'RE HEER THE PAIN CUZ THEY'RE WISH I WOR THAT I WANNA THE SICKS
 I WANT YOUR BREAKY BE HER THINKNOW YOU'RE FREER TO SEE WHEN I KNOW YEAH
BE OF THE THINGS THINGS THAT I WANT YOU WELCOME
CAN THE WORLD NEVER AND YOU BETTER I WANNA DO EM I DON'T STARS **I DON'T MEAN T**
O THE THINGS UP IT I WANNA SBOMICAN **BE ALONE IF YOU HIDE YOUR** ASTER THAN SHE T
EAL IF YOU ARE YOU THOUGH TO ME
DECTANG SONE YOU **I WANT YOU AND I WOULD** FAIN WEELL YOU GOT ME OF THE SEA IN M
Y SECHE I SHE THINKING IT BREAKING EVEN YOU I DOWN ANY THAT WHAT I KNOW HIT MIG
HT GO SHAD ON YOU CALL THE LALDLY
YOU DON'T KNOW YOU TROUFONE ONE SO HOLD YOU AS LONG YOU LOVE A LITTLE JUST T
O BE MY STUP DON'T GOT
THE SEAL IT MAKE IT SHOW YOU WALK FROM I WIFA BE THIS YOU JUST IT IT TWILL WAY B
ECAUSE TIME THERE DAY OF AND THE SEET YOU **I'M HAPPY** CLAP ASK AND

As we can see, the generated result still lacks on logic and does not make much sense sometimes. However, if we see those bold words, such as "I want you and I would", they to some degrees express a feeling of romantic and can possibly be added to a song. Since there are lots of "love songs" in the training data, it makes sense that there are many emotions and expressions of feeling found in the generated text. And also, according to the plot of loss shown below, the model is converging well during the training process.



(2) Where does it do well and where does it seem to fail? (**5 points**)

Firstly, it's obvious that the model has some troubles handling generic prime_str like "wh" and "th" which frequently appear in the text and are derived from many different words. More specifically, the model sometimes performs weirdly on them even though it handles other words very well. Moreover, the generated sentences are still "broken" after several training processes, where there are lots of grammatical mistakes and ambiguous meanings.

One good thing is that the model is usually able to generate complete words inside a sentence (little spelling errors) even for the words in different formats (-ing, ed, etc.). And also, sometimes it does generate some sort of "romantic" lyrics with "love", "baby", "really feel" and "your world". It probably gains some insight from the training data (this kind of music genre).

(3) Report perplexity on a couple validation texts that are similar and different to the training data (**10 points**).

I used three different strings for the validation and perplexity calculation: sentence from training data, validation data and a random generated string. Based on the perplexity results, we could tell how our model performs. If the model convergent well, the perplexity for the training sentence should be very small. If the model generalized well, then the perplexity for the validation should be small as well. However, for the random / nonsense string, the perplexity should be very large since it's technically not a sentence we want at all.

The result is shown below.

training data	3.6023459094834127
validation data	4.140773127271072
random generated string	9.115670415785104