**ECE 09495/09595**

**Assignment 3 Jacob Matteo**

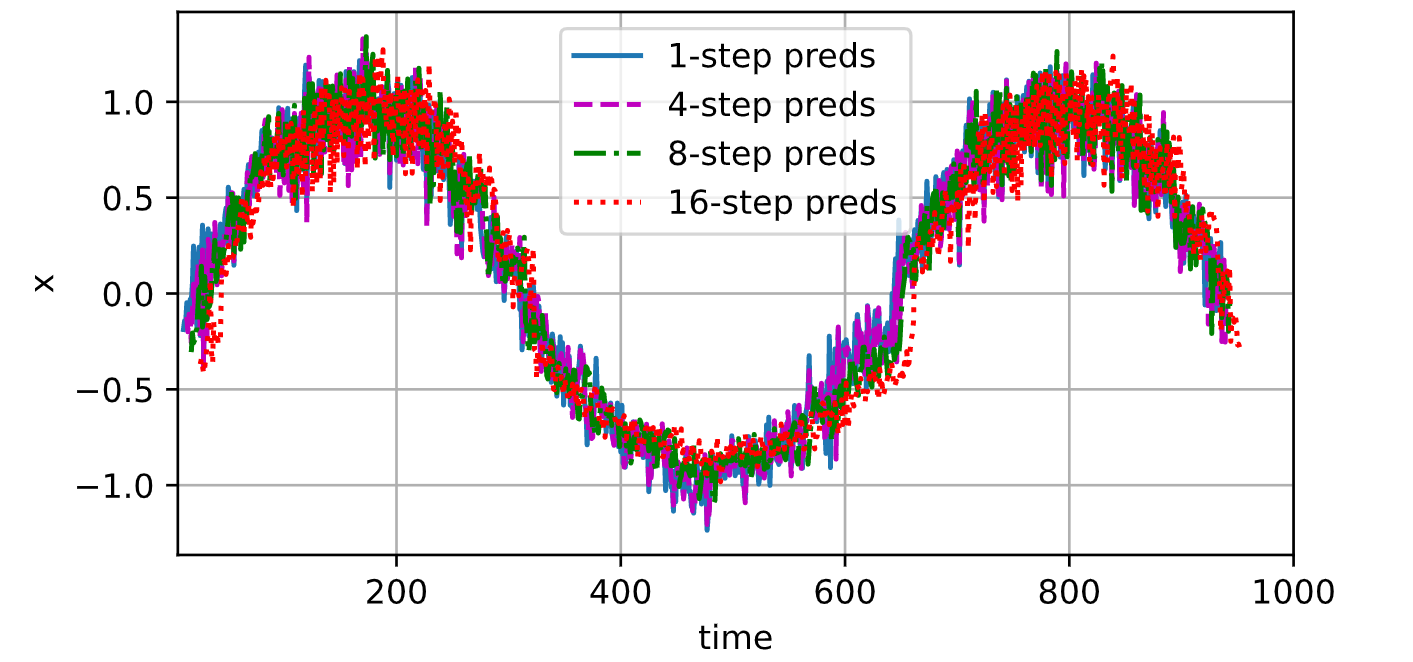
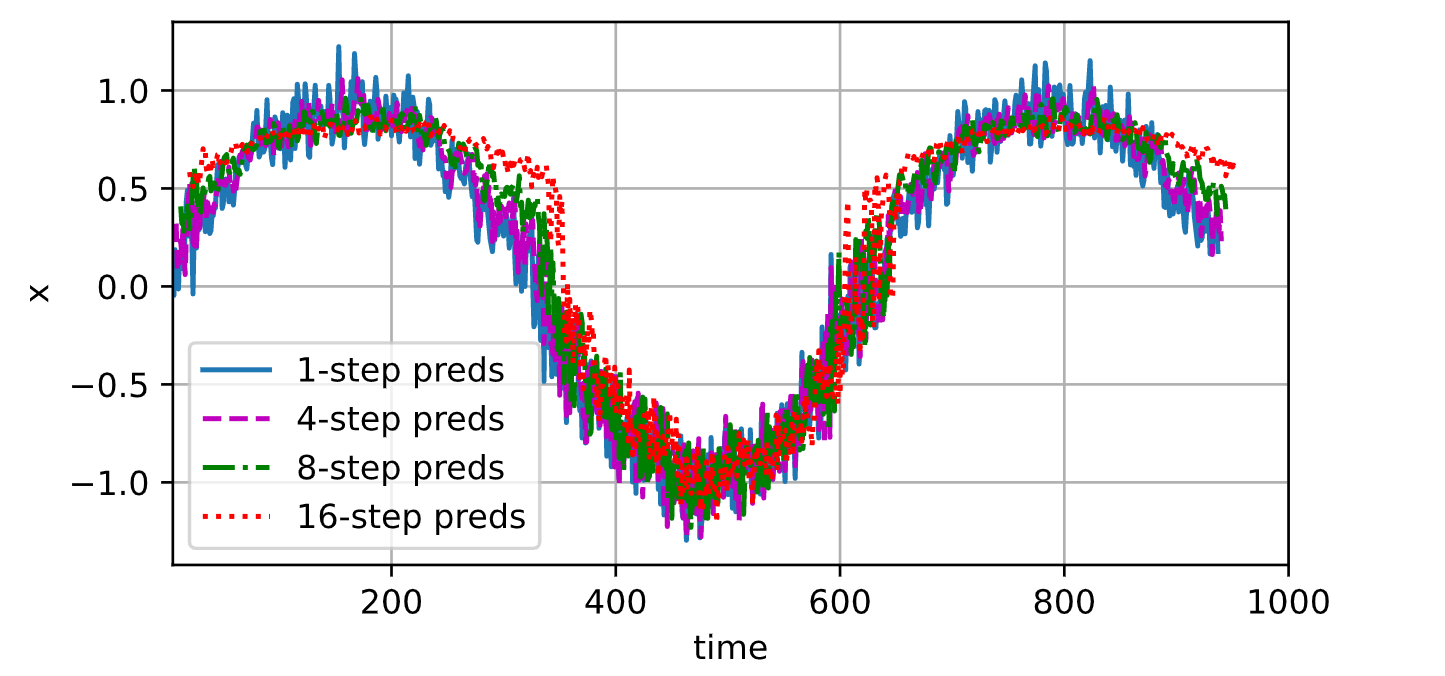
**Instructions**

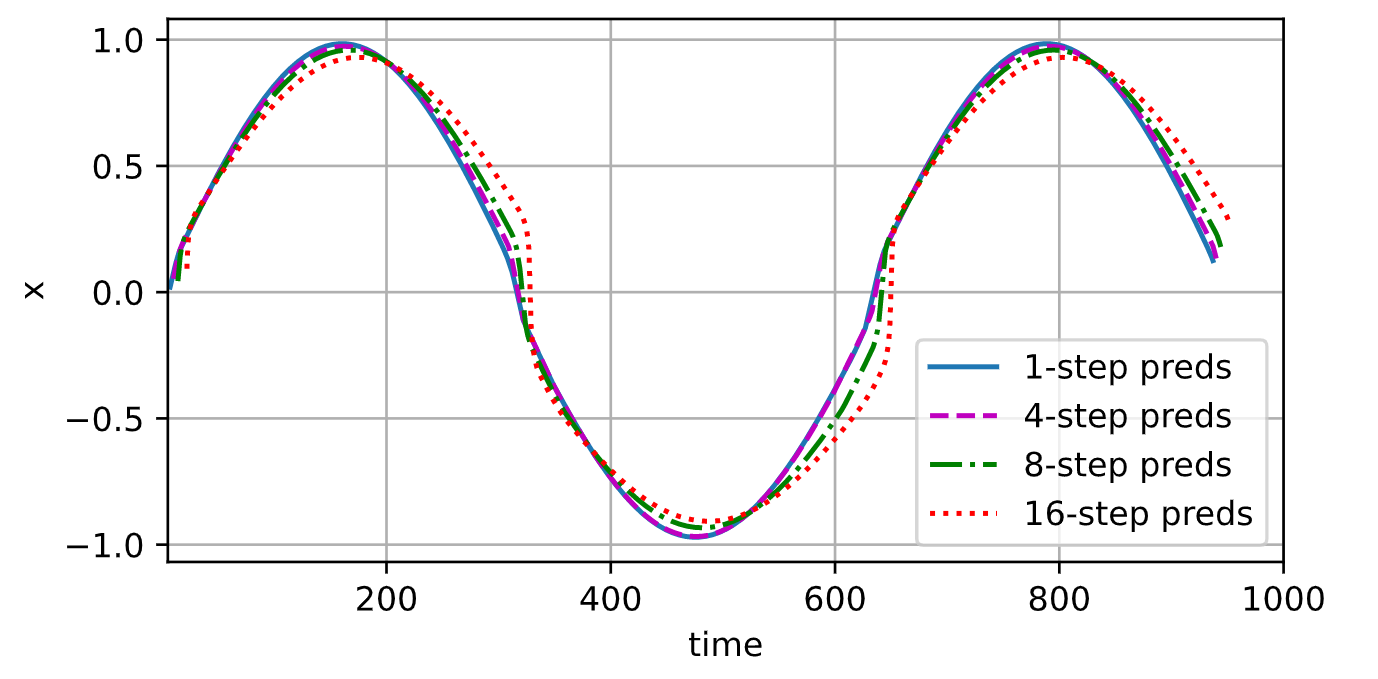
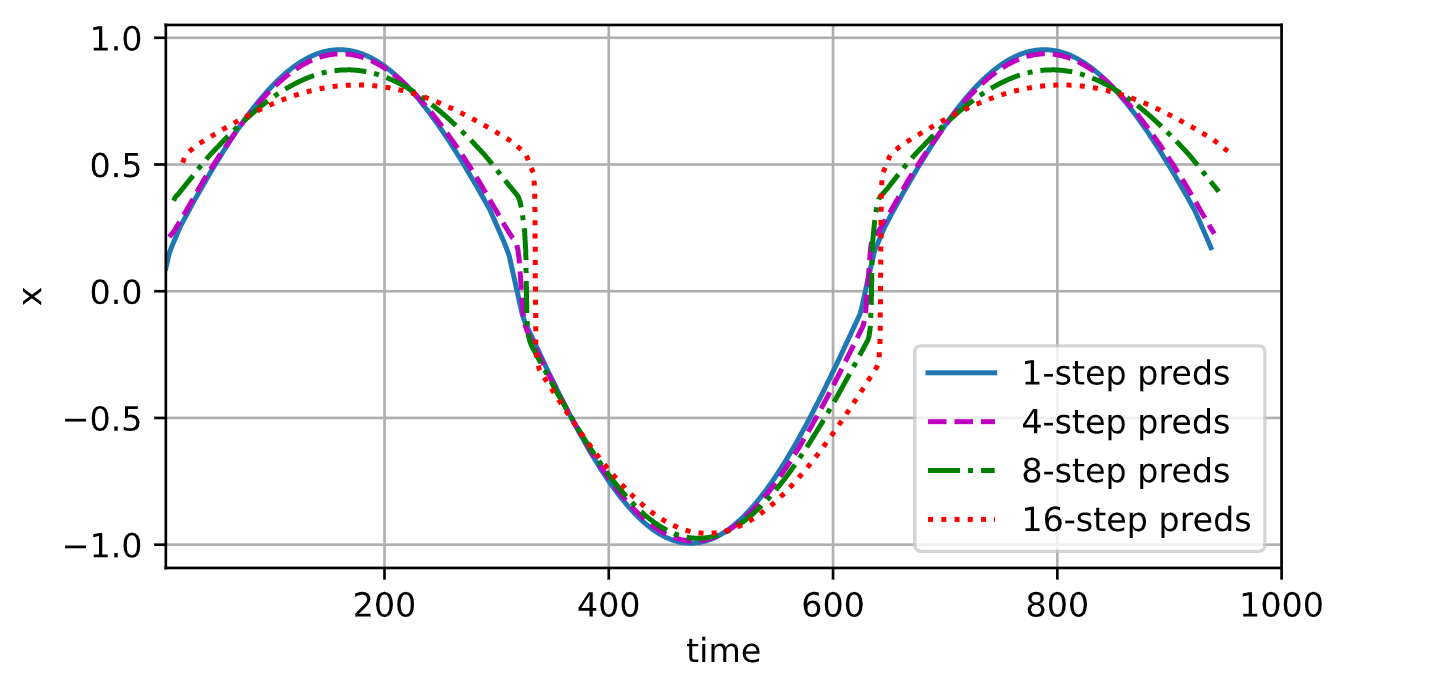
1. Max Credit: 100 Points
2. All questions are from the Textbook – Dive into Deep Learning (<https://d2l.ai/>).
3. Submit a single PDF.
4. Please do not include code. Upload the code to your GitHub, share it publicly and add link in the assignment.

**Github:** https://github.com/jmatteo/Machine-Learning-Fall-20

**Questions**

* **Part – I 60 points**
  1. Q1 – Section 8.1.5: Exercise 1 (all parts)

1) After incorporating more past-observations, I found that 9 is optimal for τ. τ=9  τ=4

2) Based on the code, I found that without noise, the optimal τ is 6. τ=6 τ=4

3) Because the amount of features in a model is dependant on the number of observations made per step (tau), it isn’t possible to increase the previous observations without increasing feature size. However, looking back farther in time does increase accuracy in the predictions made by the network.

4) Performace if this neural network increases when the amount of future steps predicted are minimum and the amount of past observations observed are increased. Running the training for longer also increases the end performance.

* 1. Q2 – Section 8.2.6: Exercise 1

In this section, words were tokenized by using the python split() command and making a token for every word or character. Three other options for tokenizing words are through the Natural Language ToolKit (NLTK) which can tokenize sentences or words, the spaCy library which can tokenize only English words and sentences, or the Keras tool which can only tokenize words.

* 1. Q3 – Section 8.3.6: Exercises 1, 4, and 6

1) In a 4-gram model, P(x1,x2,x3,x4,x5)=P(x1)P(x2)P(x3)P(x4)+(x5),=P(x1)P(x2∣x1)P(x3∣x2)P(x4∣x3)P(x5|x4),=P(x1)P(x2∣x1)P(x3∣x1,x2)P(x4∣x2,x3)P(x5|x3,x4)=P(x1)P(x2|x1)P(x3|x2,x1)P(x4|x3,x2,x1)P5(x4|x4,x3,x2). The word frequency for a 4-gram is -n2 + (w+1)n (a formula I obtained from outside the textbook) = 399988.

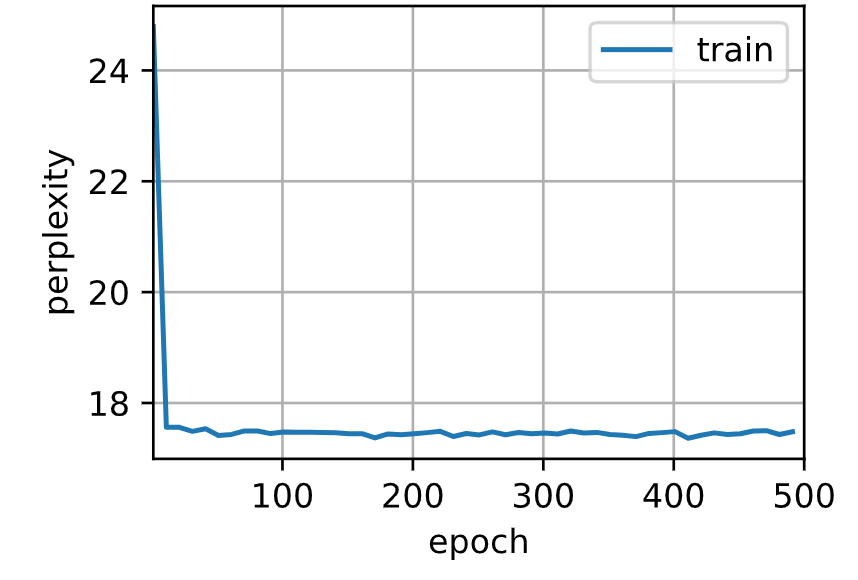
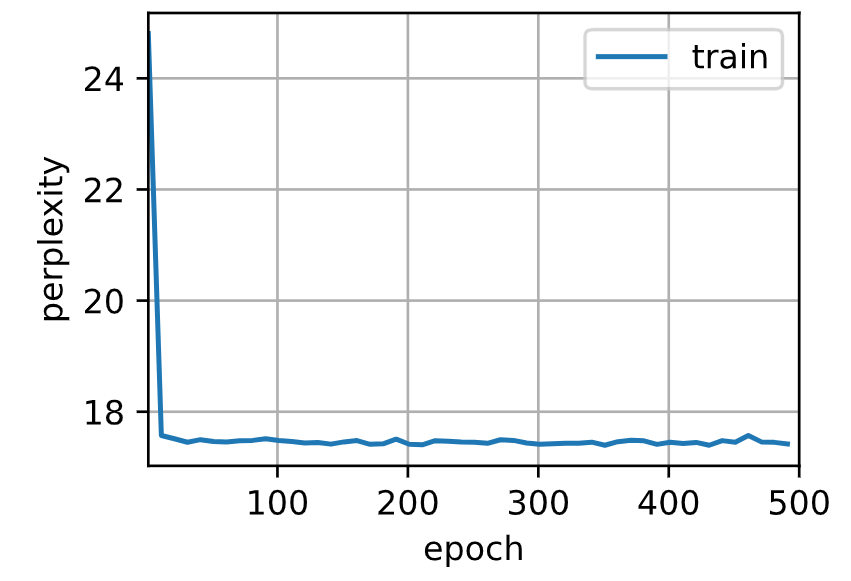
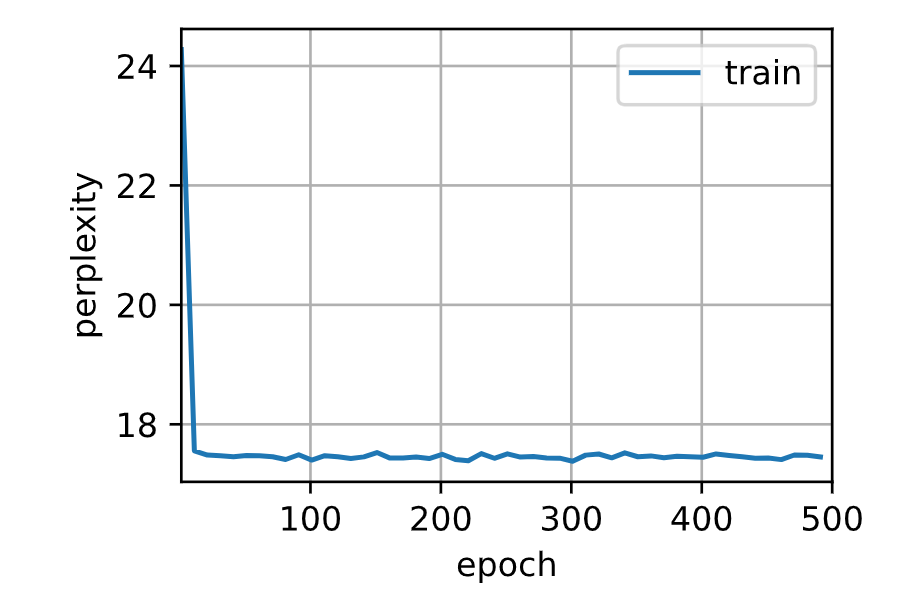
4) Some other methods I could think of for reading large sequences of data are to read in a string from and to each space and compare the string with a tree of learned data to find out the meaning of the data. Another way would be to get a list of learned data and compare it to every possible snippet of data of the same length and go down the entire list of words until every possible word is found (which would work but would take a long time to process and doesn’t account for unknown words)

6) The issue with using an entire sentence as the example for minibatch sampling is that a complete sentence is so large, the frequency of repetitious sentences is not likely to be said more than a handful of times. We can fix this problem by either having such a large dataset to choose from that sentence repetition is inevitable, or by not using entire sentences as examples at all.

* 1. Q4 – Section 8.4.6: Exercise 3

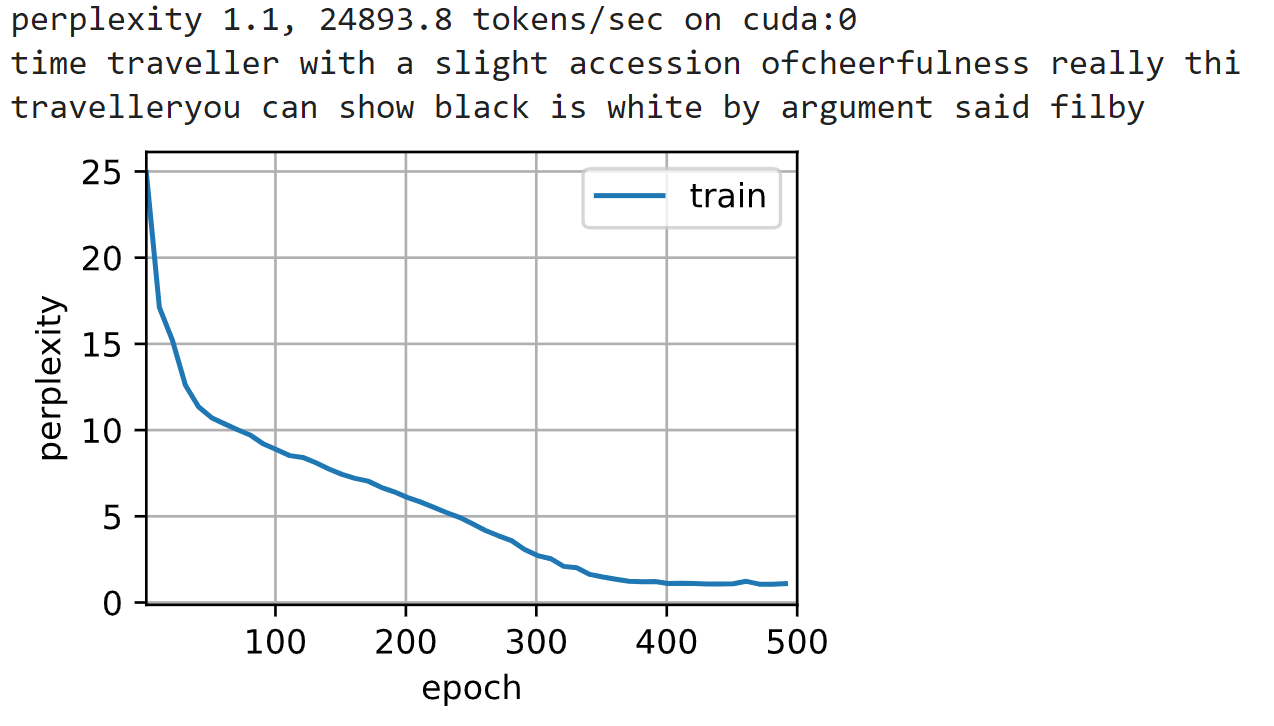
When you backpropogate through a long sequence, the gradient increases in size, is harder to compute, and therefore takes more time to calculate. This is because every layer has its own weights which need to be combined with the gradient. Every hidden state uses the hidden state of the previous step and therefore needs each previous step to be in themselves calculated and combined with the state before that state to create the next state. However, this long backpropogation also makes the gradient more accurate as it can learn from more previous solutions to create its own.

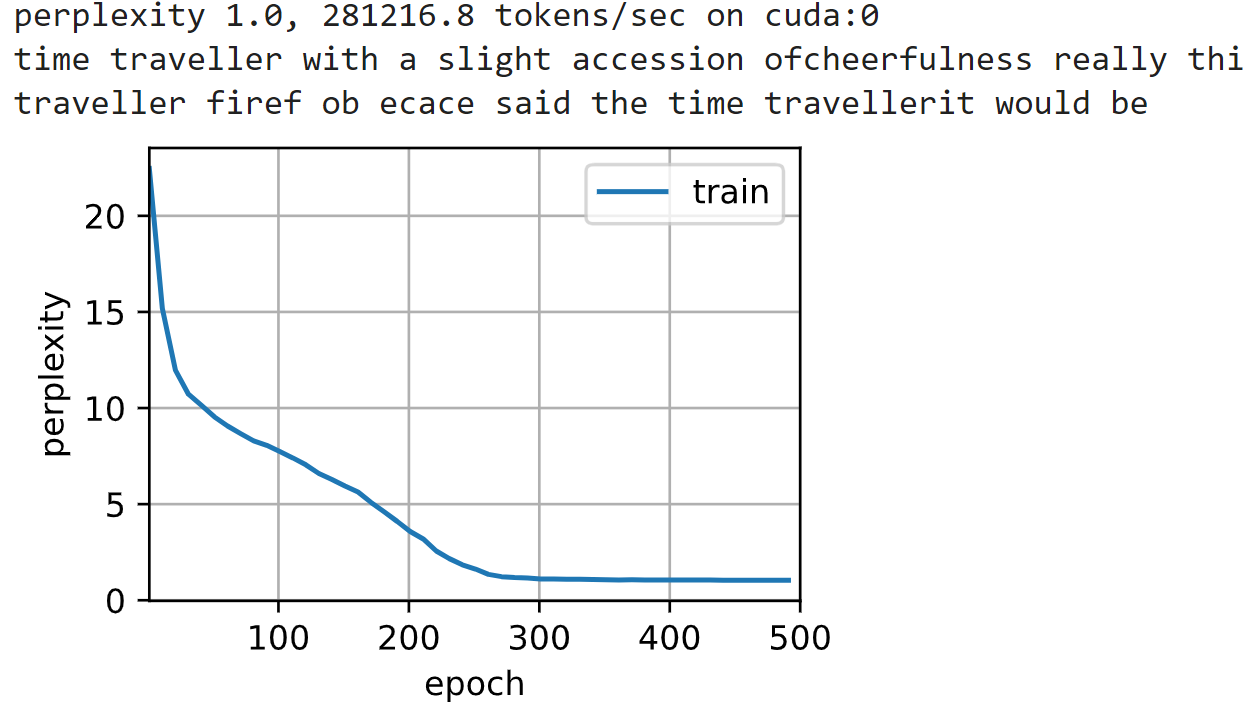
* 1. Q5 – Section 8.5.8: Exercise 3

Start: sampling:  alpha > 1:

* **Part – II 40 points**
  1. Q1 – Section 9.1.5: Exercise 1

Compared to a simple RNN, it seems as if a GRU rnn takes more time to train, but then runs faster and produces a better result at a lower perplexitivity.

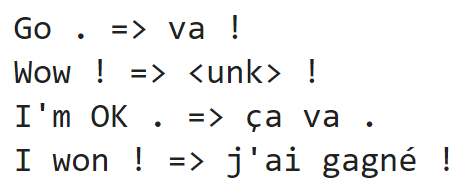
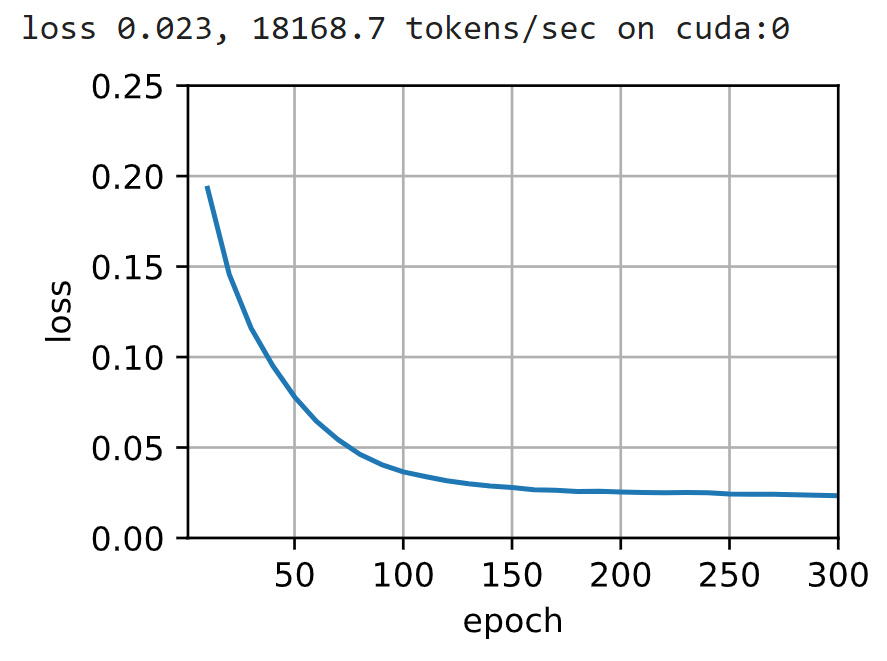
RNN: 

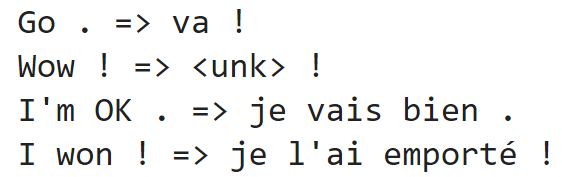
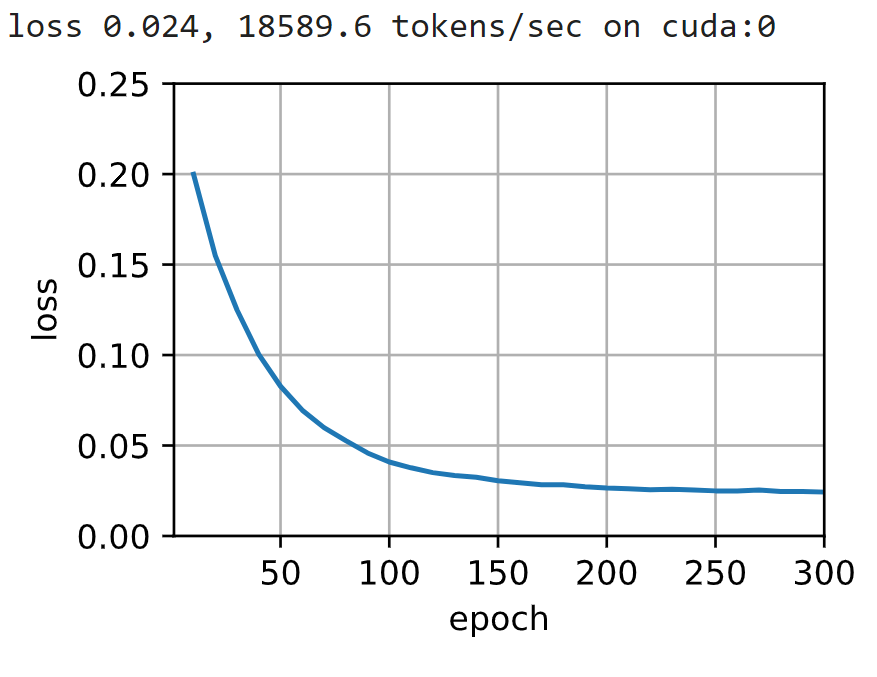
GRU: 

* 1. Q2 – Section 9.2.5: Exercise 4

The reason why the hidden states use the tanh function to ensure the output is between -1 and 1, even though the inputs can only be between the two, is because LSTM is based on logic gates which can only go from zero to one. This also makes sure the output matches the input of any following states as a 0 means it stays the same and the closer you get to 1 or -1, the more the next state changes. It makes sure no matter what changes are made, the output stays within bounds.

* 1. Q3 – Section 9.7.7: Exercise 3

With sequencemask: 

Without: 

Loss goes up as it gets a higher validation result than it should because it might learn to match with the zero elements of the sequences.

* 1. Q4 – Section 9.8.5: Exercise 2

In section 8.5, a greedy search is used. Mainly because it is very simple to implement as it takes the first best option it has. To increase performance, either a beam or an exhaustive search can be used. Beam search is based on trees and the greedy search so, while performing better and faster than a pure greedy, will not always end up with the most optimal result. An exhaustive search will always give the best result but takes a lot of time to compute. Beam is the best compromise between accuracy and speed while exhaustive is the best in accuracy and greedy is the best in speed.