Deep Learning Assignment 4

Student Number:13064947 - John Goodacre

March 28, 2018

1 Introduction

This is the write up of the submission for Deep Learning assignment 4. The associated code and implementations may be found in the emailed Colab link and the Jupyter notebook included in the zipped submission. The assignment was fully implemented for parts one and three, however for part 2 of the in-painting, formulae and discussion was put in this document, but that part of the code wasn't implemented.

2 Question 1: Many to many pixel prediction

2.1 Question 1 - Implement and train model

2.1.1 Implement and train model:

The model was implemented as per the following specification:

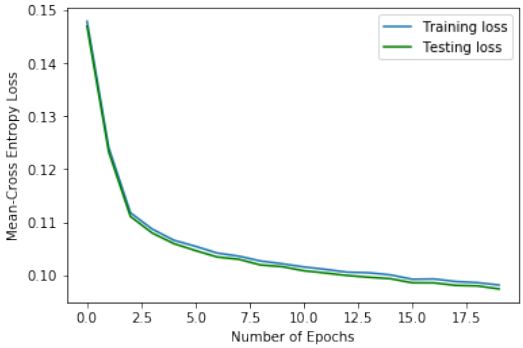
- Architecture: Input- GRU(32) - Relu - FC(64) - Relu - FC(64) - Output
- Number of Epochs 0,1,5 (for reporting), 20 for graphs of train/ test evaluation.
- Learning rate 0.001, optimiser Adam
- Batch size 256

2.1.2 Report cross-entropy and accuracy on train test set

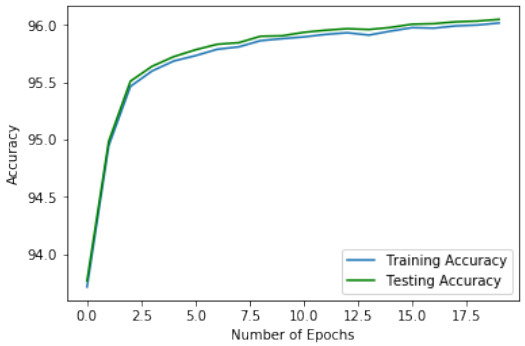
The model was trained for 20 epochs (only used for illustrative purposes for this chart - the rest of the assignment is based off the 5 epochs requested). Fig 1 shows the results. The cross-entropy reduced to 0.1 and the one step ahead accuracy peaked at around 95%.

2.1.3 Plot summary table

The table for losses and accuracy for the trained model is given in figure 2



(a) Cross Entropy by Epoch



(b) Accuracy by Epoch

Figure 1: Question 1: Train/ Test Xent, Acc by Epoch

GRU	(Beginning - 0 epochs)	(Mid-training - 1 epoch)	(End of training - 5 epochs)
Test loss	0.15	0.12	0.10
Test accuracy	93.77	94.99	95.78

Figure 2: Question 1: Epoch 0,1,5 Loss and Accuracy, GRU model

2.2 Question 1 - Visualise predictions

2.2.1 Generate pixel prediction dataset

Please see Jupyter notebook for the implementation. The model was saved for each epoch. 100 test images were taken and masked. The 0,1,28,and 300 step prediction were then produced.

2.2.2 Predict missing parts and compare with ground truth

Although this work was done on 100 test images, some parts of the question requires 10 samples. Thus to simplify I simply reported cross-entropies across 1000 test images. The question felt slightly ambigious as regards requirements (at least to me!), but the thrust of my answer is to save the model at different epochs and create comparable cross-entropy loss results with different look-aheads.

Report cross-entropy of trained model after 0,1,5 epochs, for 1,10,28, 300 step predictions

This exercise was reported **after** 0,1 and 5 epochs. If it was done beforeany training then weights would be simply random for the zeroth epoch as would the results, which I presume wasn't actually wanted? Even with 1000 samples, the results differed by small percentages for a few tries of this part of the exercise.

The interpretation is that looking ahead one pixel has an accuracy of around 96% (and of course more training helps - indeed with this model it keeps improving beyond 20 epochs - but takes a long time to train). As we look further ahead, the results decrease but not necessarily as one would expect. For example, the bottom part of the images has lots of zero values and we are predicting from pixel 484. So the prediction of the next row can be worse on average than to the end of the image in some cases. Results are given below.

- Epochs 0: Accuracy and Loss:
  - 1 step prediction : 0.872, 0.295
  - 10 step prediction : 0.746, 0.531
  - 28 step prediction : 0.846, 0.353
  - 300 step prediction : 0.80, 0.64
- Epochs 1: Accuracy and Loss:
  - 1 step prediction: 0.89, 0.24
  - 10 step prediction: 0.79,0.49
  - 28 step prediction: 0.87, 0.32

  - 300 step prediction: 0.81, 0.69

- Epochs 5: Accuracy and Loss:
  - 1 step prediction: 0.933, 0.176
  - 10 step prediction: 0.826, 0.41
  - 28 step prediction: 0.90, 0.24
  - 300 step prediction: 0.83, 0.62

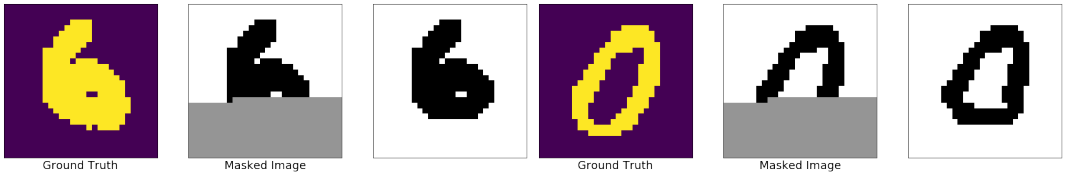
Visualise examples, (1,10,28,300) pixel predictions - 3 examples - Good/ Fail/ Other.

Here 5 samples are asked for the longer time step predictions and one for the 1-step. Again there is some ambiguity over what exactly a sample comprises. To show I know what I am doing - clearly we get a probability over each pixel from the trained model. But using that we can treat a sample as an image, and thus do an argmax over the probabilities and come up with a single new image each time. Or we can take a single test image and sample from the probability outputs from the RNN each time to create multiple predicted 'sample images' from the single test image.

Also we only have information up to pixel 484. We use that information to predict one pixel ahead. (1-step). Now because the rest of the image is masked we use that prediction to predict the next pixel and so on. So it appears to me the way the question is asked that the 1-step,10-step and 28 step predictions are already included in the 300 step prediction. Thus, I have generated lots of images and generated the data required but to attempt to reduce the document length, rather than put in 48 images! I will put in the images for the 300 step predictions (which include the others anyway). Apologies if this was not what was intended, but I believe the essence of the question has been solved and implemented and the interpretation of the question I was again not entirely sure of.

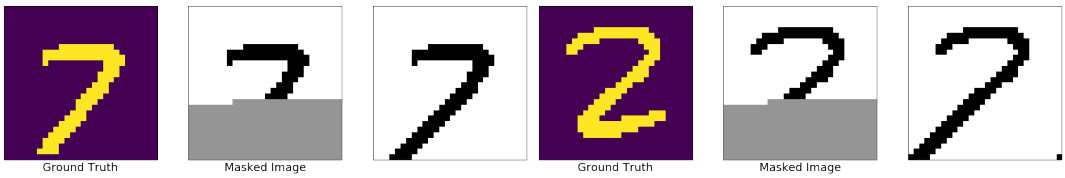
Finally, the images shown are the worst possible, as they are basically the full image with 300 pixels all predicted. The success rate on a fully trained model (not stopped after 5 epochs for a 1 step prediction is over 96%). Thus, the images are near perfect if we were allowed to for example, step forward each time with 1 pixel from the ground truth. Predicting further ahead and showing 'failure' cases is perhaps more informative. I hope this explanation shows that the work has been done regardless of the implementation interpretation.

In terms of explaining the page of images. Fig 3, shows a variety of ground truth's, masked images and predicted images. I have deliberately chosen success and fail cases. There is a brief explanation below each image.



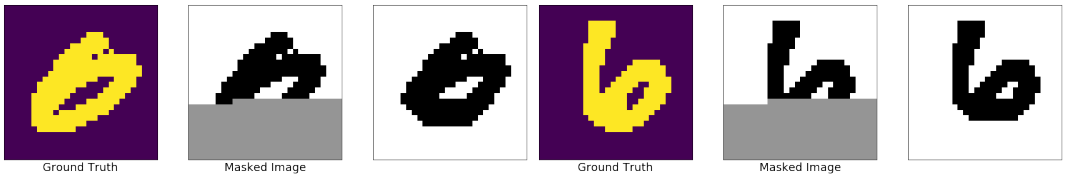
(a) Success - 1,10,28, 300 pixel predictions mostly good

(b) Moderate success - 1,10,28 and some of the 300 ok.



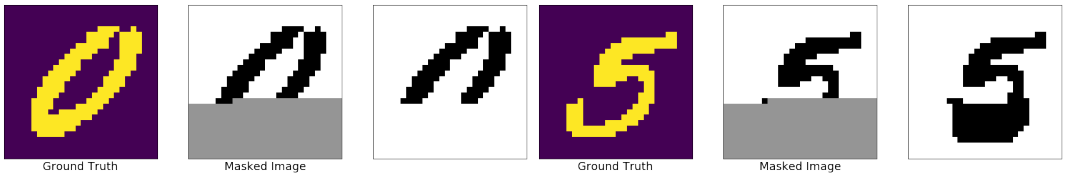
(c) Success example - 1,10,28,300 well predicted

(d) Fail - Similar mask, but predicts akin to the previous model



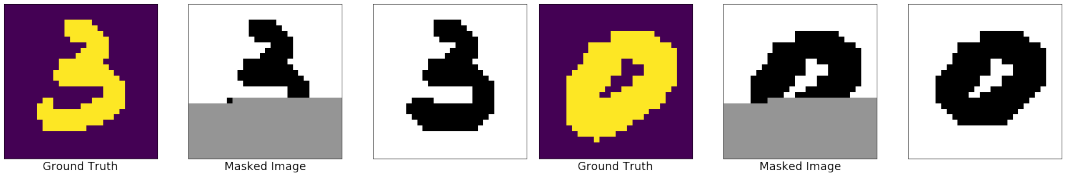
(e) Strong success on almost most lookaheads

(f) Relative success - but figure again cut short for 300 preds



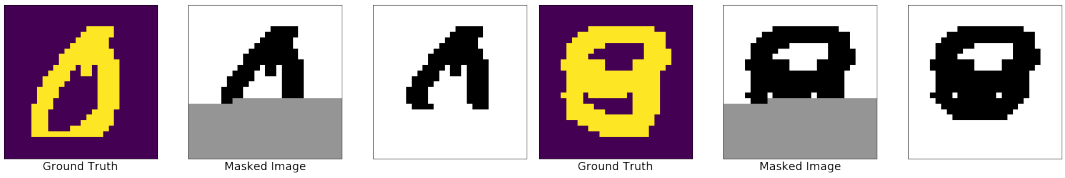
(g) Complete fail - starts with zeros and continues

(h) Mixed success,but 1,10,28 pixel pred poor - just fills in



(i) Fairly good success at most look-aheads

(j) Moderate success - 1,10,28 pixel - cuts figure short for 300



(k) Complete fail - all zeros, and not the right kind

(l) Semi success - but 1,10,28 poor

Figure 3: Ground truth/Mask/ Prediction - model never sees beyond pixel 484 and can only predict using its own predictions - thus 1,10,28 and 300 pixel predictions included

3 Question 2 - In painting

3.1 Question 2 - Write probability for missing pixel/ patch

Using the notation given in the notebook we are training a recurrent network to output the following:

$$\hat{p}(x_{t+1}|x_{1:t}) = g(x_t, h_t, c_t)$$

In the previous examples, we have 'past' pixel information and can then use that to predict one step forward (in the models trained), when we predict further forward we can bootstrap from our own predictions. In this case we not only have past pixel information but future pixel information.

3.1.1 Provide the formula used to compute the probability over the missing pixel and the missing patch respectively

Missing Pixel and Patch. If we presume an image has T pixels and that without loss of generality pixel t+1 is missing then we seek:

$$p(x_{t+1}|x_{1:t}, x_{t+2:T})$$

My first thought was to use a bi-directional LSTM here, but the rules of the game are such that we are only allowed to use our existing model which is training on past pixels only. So we rejig the formula to include future information under these constraints (notice that the numerator is the $p(\mathbf{X})$, the likelihood of the image). Given this likelihood then we can unwind this joint probability into conditional probabilities based on past pixel data, thus fitting the format of the model. Also notice that the denominator is the joint distribution but with the missing pixel marginalised out. So for the denominator we can take the joint but summing over the two cases comprising the missing pixel examples.

$$p(x_{t+1}|x_{1:t}, x_{t+2:T}) = \frac{1}{p(x_{1:t}, x_{t+2:T})} p(x_{t+2:T}|x_{1:t+1}) p(x_{t+1}|x_{1:t}) p(x_{1:t})$$

Therefore,

$$p(x_{t+1}|x_{1:t}, x_{t+2:T}) = \frac{p(x_{1:t})}{p(x_{1:t}, x_{t+2:T})} p(x_T|x_{1:T-1}) p(x_{T-1}|x_{1:T-2}) \dots p(x_{t+2}|x_{1:t+1}) p(x_{t+1}|x_{1:t})$$

In this example we wish to either maximise the likelihood over the whole image or maximise the conditional probability of the missing pixel x_{t+1} given the rest. The above formula enables us to calculate this as the other pixels $x_{1_t}, x_{t+2:T}$ are given.

Missing Patch

The probability over the missing patch (here we are using 2x2, and assuming the image has dimension dxd) is very similar and is given by:

$$p(x_{t+1,t+2,t+d+1,t+d+2}|x_{1:t,t+3:t+d,t+d+3:T}) = p(\mathbf{X})\frac{1}{p(x_{1:t,t+3:t+d,t+d+3:T})}$$

Again the numerator is the joint distribution (which can be unwound in the right order for the RNN - i.e. conditional probability based on past pixels). The denominator we can have the same thing as long as we do the sum over the cases of the missing pixels as for the above individual pixel case.

After having completed questions 1 and 3 of this assignment. The length of the assignment, number of graphs to produce, other assignments and the exam period unfortunately means that I didn't get to implement this in code. However, my initial and highly inelegant plan was to do the following.

First, I was happy to maximise the likelihood of the overall image, or the likelihood of the missing pixel given the rest of the image (I basically considered it an equivalent question). The above formulae give the likelihood in the format needed from the original trained model. My inelegant plan was to find the missing pixel (np.where(image==-1)). Then because the this is either 0 or 1, I could simply test the two cases. By in-filling the rest of the image as per part one I could then calculate the cross-entropy of the two inferred images with the rest of the image. The selected pixel would be the one which resulted in the lowest cross-entropy.

As I said this is inelegant. This can be applied to the patch example but would require checking 2^4 different examples. An alternative method that I was looking to implement was to use sampling.

Sampling:

Here the idea was similar but instead of generating an image using an argmax for each missing pixel eventuality, instead the idea is to use the output probabilities from the RNN. Then sample from these probabilities to generate many sampled images. Given those sampled images we subdivide between those where the missing pixel is on/off. We could then calculate the mean cross-entropy between the generated images of each category and the future pixels and select the pixel activation resulting in the lowest mean cross-entropy.

The visualisation of the image and cross-entropies of the 'best' images would then be calculated in exactly the same manner as in the rest of the assignment.

4 Question 3 - Learning multiple tasks

4.1 Question 3 - pre experiment

The first question is a 'feel' and intuition question, thus I have been a little wordy. The problem setting is Categorical/ Dirichlet - giving a Dirichlet conjugate back in the Bayesian world. Where I am unsure is exactly how 'Bayesian' the model can be (trained parameters on one 'meta-distribution' but then online predictions as new data comes in, for our 'posterior'). Thus I have given long-run and short-run answers to the 'intuition' bit.

4.1.1 Pre Experiment - What will the LSTM learn?

It makes little difference, but I used a GRU throughout the assignment. My interpretation of the task is as follows. We will have a sequence of values for a fixed time period (5 or 20), those values will come from a fixed categorical probability distribution. However, after the fixed time period, we hit the reset button and this probability distribution is resampled. The underlying parameters for the probability distribution come from a Dirichlet with the alpha vector provided.

I will initially discuss long-run behaviours. For $\alpha = (10, 1, 1)$, I would expect samples from the categorical distribution to be heavily biased to the first character. Thus, my intuition would expect the RNN to 'learn' to be heavily biased to the first character and would hope that given the number of samples it would also learn the fixed time reset perfectly. Every 5 or 20 steps the actual probabilities will change but if we resampled the parameters repeatedly I would expect the overall expected values of our parameters to approach probabilities of 10/12,1/12,1/12 respectively (the mean values of this Dirichlet). Over the long run I would thus expect the RNN softmax outputs to approach this distribution.

A semi-subtlety as regards implementation is that if we choose our most likely character (argmax), then I would expect the RNN to only choose character 1 and the 'Reset', because the other characters are always going to be less likely. If however one chose instead to 'sample' from the RNN softmax outputs then it would of course select the other characters. (Argmax of course also gives a maximum a posteriori and not the distribution that a bayesian would output).

For $\alpha = (1.1, 1.1, 1.1)$, the long run behaviour would be a fixed reset time and otherwise an equally likely probability of each character. Again I would expect the RNN to learn the fixed reset correctly and the long run behaviour as the number of samples approaches infinity for the other symbols to be equally likely. In reality though, I would expect the new data to not be perfectly balanced and the RNN base its next prediction based not only on this equally likely trained 'prior' but also the new data.

4.1.2 How does the learning ability and generalisation depend on the length of the LSTM and the speed of resetting?

I am imagining the behaviour and problem setting of the LSTM as follows. The long-run trained behaviour, being some sort of 'prior', however when new sequence data comes in, then the LSTM will predict according to that data (giving a 'posterior' based on the parameterised model and our new data), so if the reset time is long enough and the model has a long enough memory then I would expect the LSTM to predict in the direction of the new data distribution as it comes in.

I am not sure if I am being a bit too optimistic, as this implies the LSTM can replicate a Bayesian model (where I am unsure is if for example the model has been trained on a fixed distribution and now has fixed parameters, but we try and predict on a very different data distribution). So I believe it will come up with some sort of posterior, but within the limitations of its pre trained parameters and the length of the LSTM.

As regards the length of the LSTM. Clearly this also gives the LSTM a memory. If we had no length at all, we are back in the world of multi-layer perceptrons and doing a pure regression on the next value given the current.

In terms of the speed of reset, with a limited number of samples it seems intuitive to me that the RNN will have a tougher time if the data keeps changing distribution at shorter time intervals. Even for long run behaviour, at the very least this introduces more noise.

In terms of the ability to generalise to an unseen p, my intuition is weaker. I would hope that the model would given enough samples simply match this fixed p. Whether it will do this quicker or slower than the bayesian comparison I simply don't know intuitively.

4.2 Question 3 - Implementation

Train for 1000 mini-batches - record train/ test performance and plot curves.

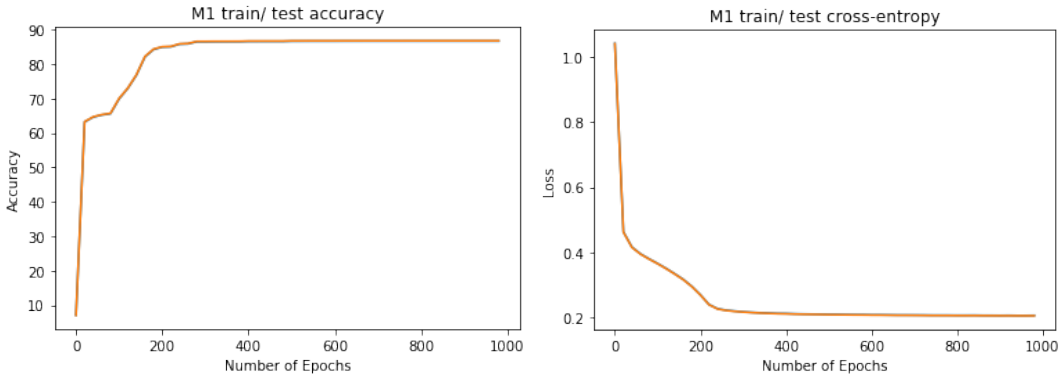
- Four models - M1,..,M4
- T=5, T=20, $\alpha = (10, 1, 1), \alpha = (1.3, 1.3, 1.3)$

The four different models were implemented in Jupyter notebook. With a heavily biased α the models reached a high accuracy of over 90% and a low cross-entropy. However, I was correct in my assumption that the argmax over such a biased alpha would mean that only symbol one and the reset would be selected. To avoid this we would need to sample from the posterior from the model.

The relative advantage of a short versus a long time to perform a reset were to my eyes less clear than I expected. With enough samples the RNN was able to learn both.

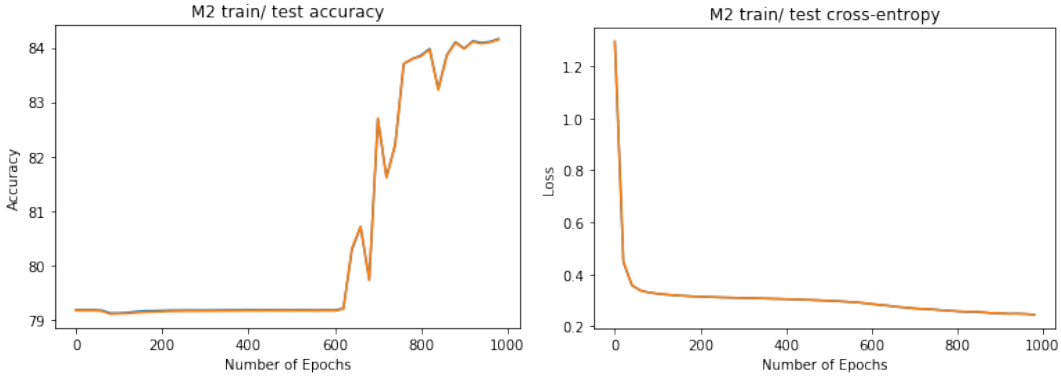
As expected the accuracy with the less informative prior was lower, hovering around the mid-50% range rather than the 90's with a strongly biased prior. Also as expected the model learned the 'reset' perfectly.

The train and test performances of all models are in Figures 4, 5, 6, 7. The distinction between train and test is so close as to be unclear (they are both there), however we are sampling from the same distribution (I chose to use the given mini-batches, and then the 'population train' results is over 10,000 samples and the 'population test' over 5000. Both curves are very similar.



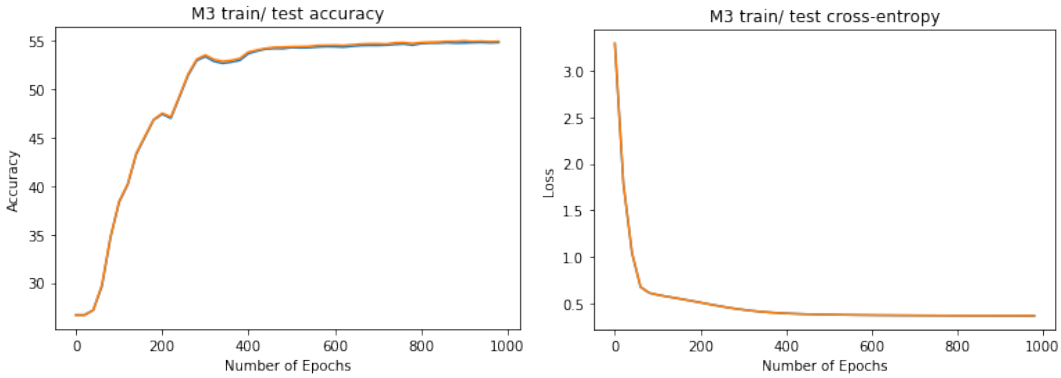
(a) Model 1 - Accuracy (b) Model 1 - Cross-entropy

Figure 4: Model 1 - Train/ test curves



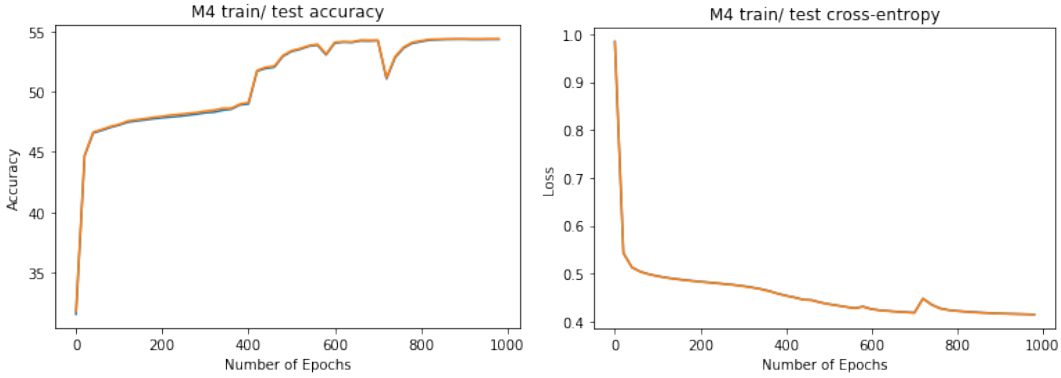
(a) Model 2 - Accuracy (b) Model 2 - Cross-entropy

Figure 5: Model 2 - Train/ test curves



(a) Model 3 - Accuracy (b) Model 3 - Cross-entropy

Figure 6: Model 3 - Train/ test curves



(a) Model 4 - Accuracy (b) Model 4 - Cross-entropy

Figure 7: Model 4 - Train/ test curves

4.3 Question 3 - Analyse Results

The three test sequences for the two α's above and for the fixed $p = (1, 0, 0)$, were implemented in Jupyter notebook. The train/ test accuracy and losses are given above, so we know the quality of predictions for model 1-4 already. Thus, for the following analysis I have concentrated on the comparison and particularly illustrating the probabilities of predicting each character through time, for each model given the generated sequences. First we derive and implement the Bayesian comparison.

Figure 8, shows the probability of each character for Models M1, M3. Recall that these are fast 'reset' models. i.e. the Reset button is hit every five time-steps. Recall also that M1 has a strong prior (biased towards character 1). M3 has an uninformative prior. The probability the model expects for each character is shown through time (this is the 100 time-steps for the RNN). The orange line gives the model's probability. The blue line, that expected from the Bayesian (implemented using the formulae above).

For the biased prior, the model seems to over-estimate the probability of character 1 and underestimate character 2 and 3. It gets the 'reset' perfectly (hopefully my implementation is bug-free- I am wondering if I have some inadvertent bias in there and given time would recheck!). For the unbiased prior the model is near perfect however. Its posterior over all characters is a perfect match to the bayesian case.

Note also in terms of implementation the model 'learned' the 'reset', this was 'hard-coded' as a rule for the bayesian update (else the bayesian would not be able to jump from probabilities of 0 to 1 for 'reset' and I wished to compare like for like without obfuscating with the 'reset').

4.4 Question 3 - Comparison to Bayesian Update

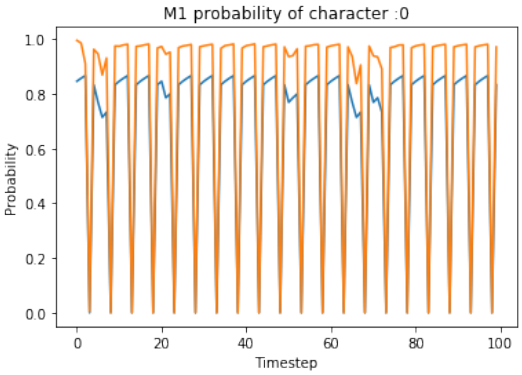
4.4.1 Derive the posterior update for each time-step, closed form

We know that we have 3 symbols $\mathbf{p} = (p_1, p_2, p_3)$ and that $\mathbf{p} \sim \text{Dirichlet}(\alpha)$. Thus, not including the regular resets we are generating symbols $(X_1, X_2, \ldots, X_{T-1})$, where $X_i \sim \text{Cat}(3, \mathbf{p})$.

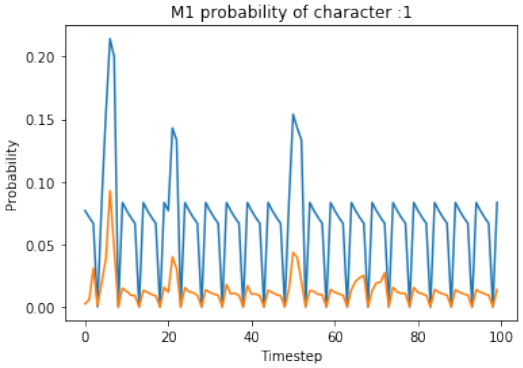
If we take a Bayesian approach we will calculate the posterior distribution over the parameters. The Dirchlet and Categorical are conjugate ditributions so:

$$P(p_1, p_2, p_3) = \frac{\prod_{i=1}^{I} P(x_i|p_1, p_2, p_3) P(p_1, p_2, p_3)}{P(x_{1,\ldots,I)}}$$

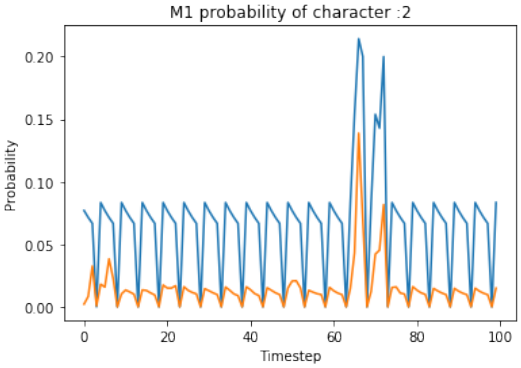
Therefore,



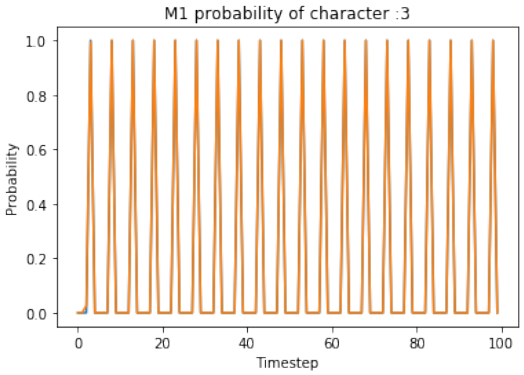
(a) Model 1 - Prob Character 0



(b) Model 1 - Prob Character 1

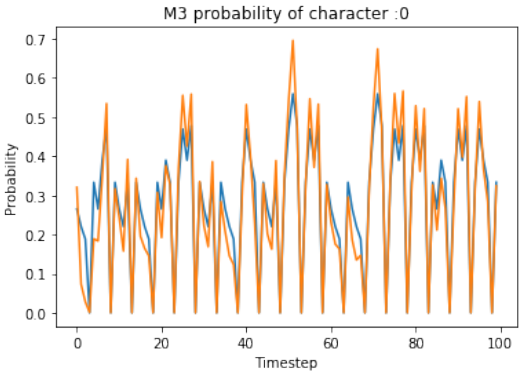


(c) Model 1 - Prob Character 2

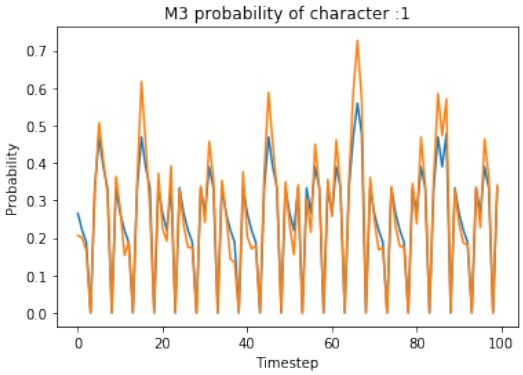


(d) Model 1 - Prob Character 3

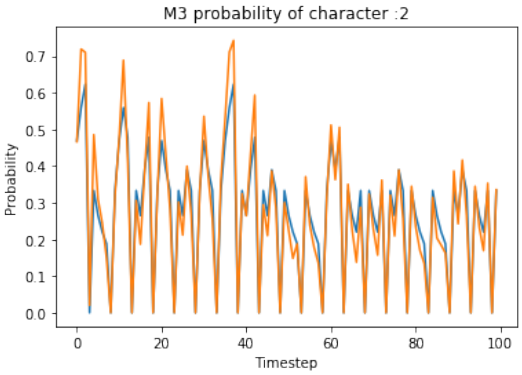
Figure 8: Model 1 - Character probabilities through time



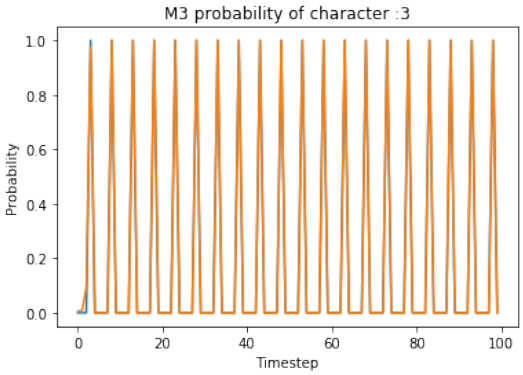
(a) Model 3 - Prob Character 0



(b) Model 3 - Prob Character 1



(c) Model 3 - Prob Character 2



(d) Model 3 - Prob Character 3

Figure 9: Model 3 - Character probabilities through time

$$P(p_1, p_2, p_3) = \frac{\prod_{i=1}^{I} Cat_{x_i}[\mathbf{p}] Dir_{p_1,p_2,p_3}[\alpha_{1,2,3}]}{P(x_{1,\ldots,I)}}$$

and,

$$P(p_1, p_2, p_3) = \frac{\kappa(\alpha_{1,2,3}, x_{1,\ldots,I}) Dir_{p_1,p_2,p_3}[\tilde{\alpha}_{1,2,3}]}{P(x_{1,\ldots,I)}}$$

which because the probability distribution must sum to one, means that we have:

$$P(p_1, p_2, p_3) = Dir_{p_1,p_2,p_3}[\tilde{\alpha}_{1,2,3}]$$

where $\tilde{\alpha}_k = N_k + \alpha_k$. Thus we can directly use the conjugate relationship to implement the parameter probabilities with an adjusted α. This α being the posterior over the prior α and the actual data coming in.

4.4.2 Implement the posterior update for the 3 sequences T=20, and for the last 2 different priors

This was implemented in the code using the above formula (indeed it was required for the previous graphs where I had already implemented the bayesian comparison). The basic idea being to start with our prior α and as characters come in, count them and adjust the posterior $\tilde{\alpha}$ according to this count. Note that with a full Bayesian treatment we would have a distribution over these parameters (the Dirichlet again), however for this implementation we want specfic probabilties which I did by taking the mean of the Dirichlet (thus giving an MAP for the probabilities).

This is a very long-winded way of saying add counts to the α prior as you go along, until you hit a reset, in which case go back to the prior ! For once things are easier in code than prose !!

4.4.3 Compare to LSTM predictions, failure cases?

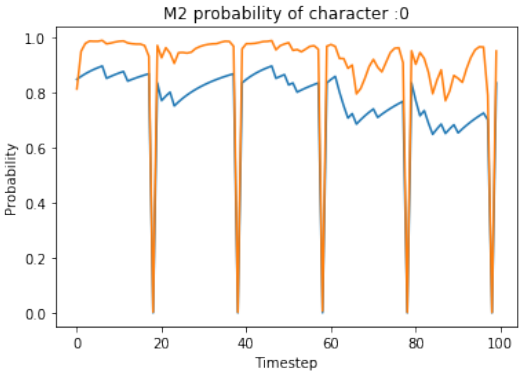
The final set of charts address this question. As requested this part of the question implements models M2 and M4 but also uses the case where the data comes from a fixed probability (both priors). The bayesian comparison is also shown on all charts.

- M2 - (reset 20 time-steps, heavily biased prior). Similar to M1, the model has a higher probability of the first character, than the bayesian and less for characters 2 and 3. The 'reset' is again learned perfectly. See fig 10.
- M4 - (reset 20 time-steps, balanced prior). Again similar to M3, a successful set of graphs with an almost exact match to the bayesian case. See fig 11.

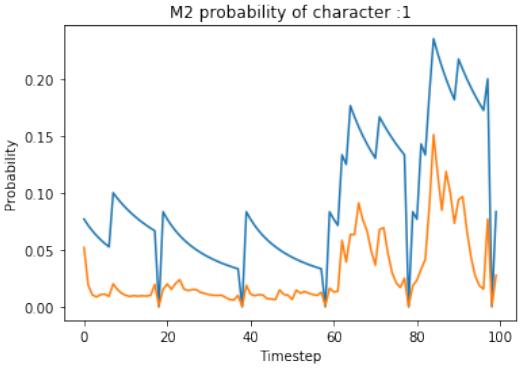
- Model 2 (biased prior - but data generated with a fixed p). In this case the model 'reacts' faster than the bayesian case. The bayesian prior is the biased *alpha* from the Dirchlet. And as more character 1's come through in the data (because of the fixed p), it does react but more slowly. The model reacts far quicker and so my conclusion is that it is similar to the bayesian example (very much so with a balanced prior) but there are differences in reaction as new data comes. Again 'reset's were learned perfectly. See fig 12.

- Model 4 (balanced prior - but data generated with a fixed p). Again the model 'reacts' faster than the bayesian comparison as new data comes in. However, it is slower to react than the RNN in the previous chart (where the RNN was trained with a prior already strongly biased to the first character). As expected 'reset's were learned exactly. See fig 13.

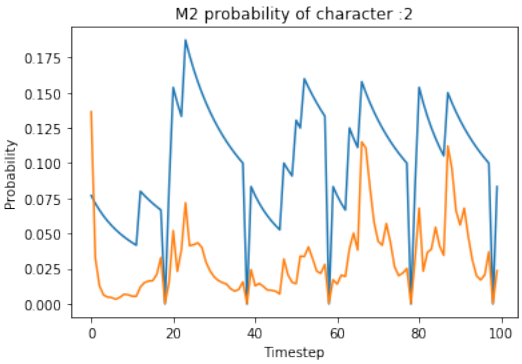
Apologies for the length of all this. I tried my best to keep it reasonable, but there are also a lot of charts as well as my sometimes wordy thoughts. Thanks again - John.



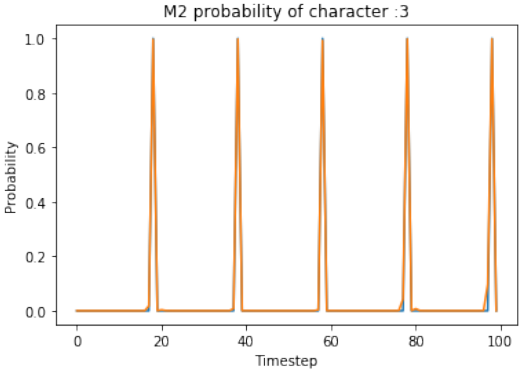
(a) Model 2 - Prob Character 0



(b) Model 2 - Prob Character 1

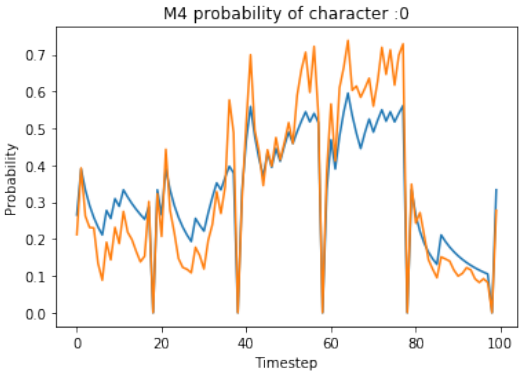


(c) Model 2 - Prob Character 2

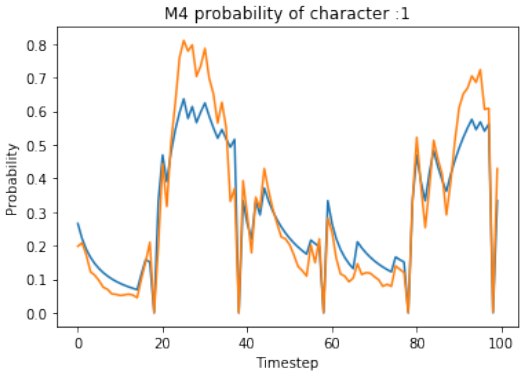


(d) Model 2 - Prob Character 3

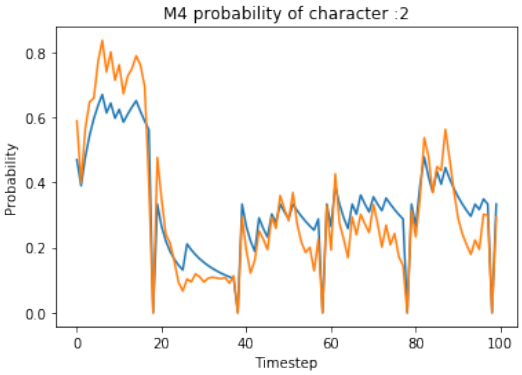
Figure 10: Model 2 - Character probabilities through time



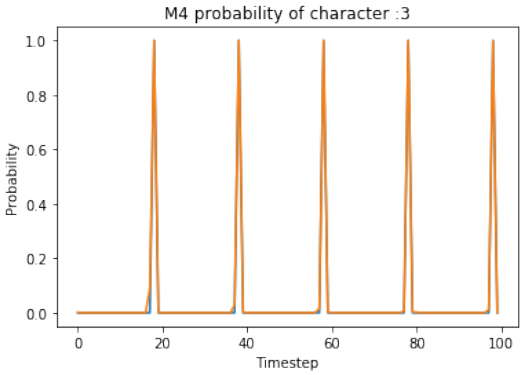
(a) Model 4 - Prob Character 0



(b) Model 4 - Prob Character 1

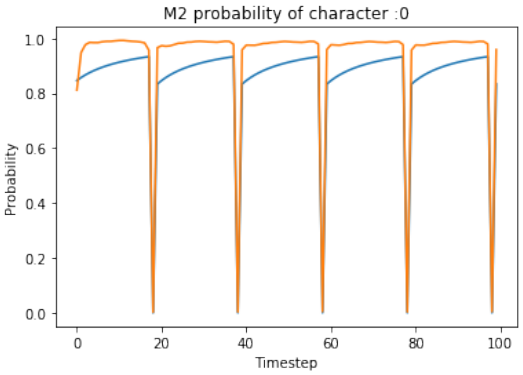


(c) Model 4 - Prob Character 2

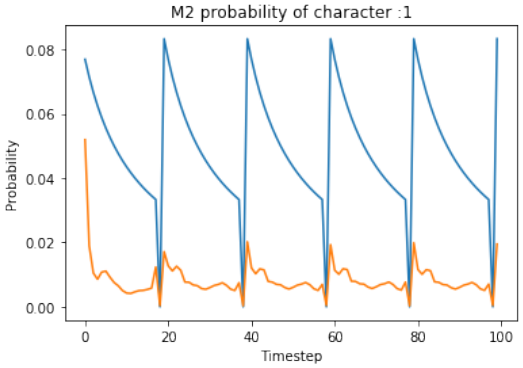


(d) Model 4 - Prob Character 3

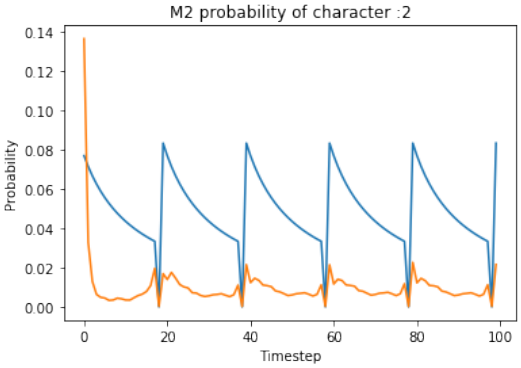
Figure 11: Model 4 - Character probabilities through time



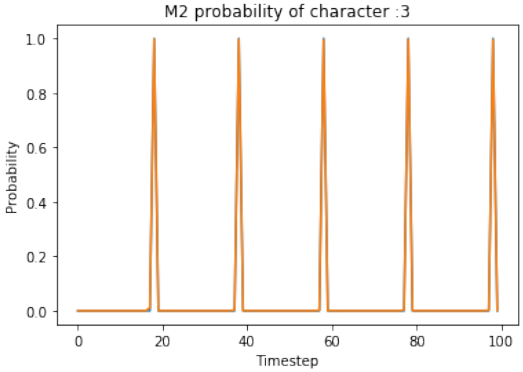
(a) Model 2 (p fixed) - Prob Character 0



(b) Model 2 (p fixed) - Prob Character 1

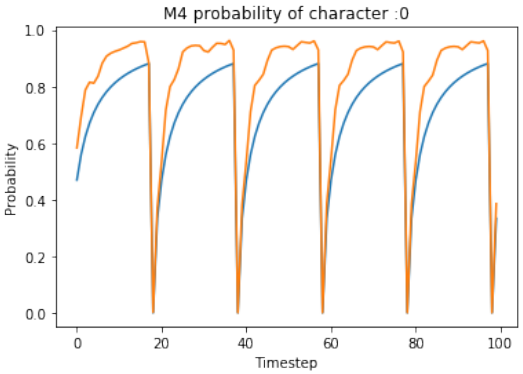


(c) Model 2 (p fixed) - Prob Character 2

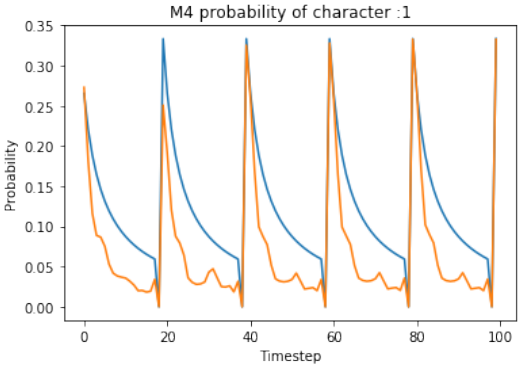


(d) Model 2 (p fixed) - Prob Character 3

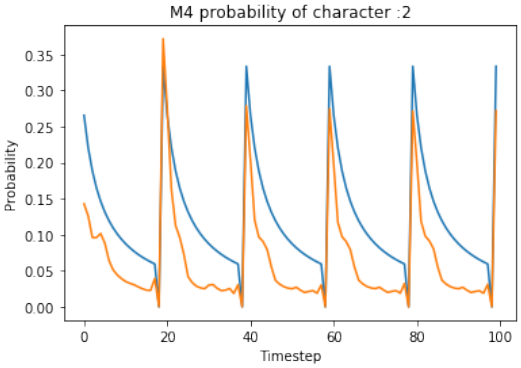
Figure 12: Model 2 - (p fixed) Character probabilities through time



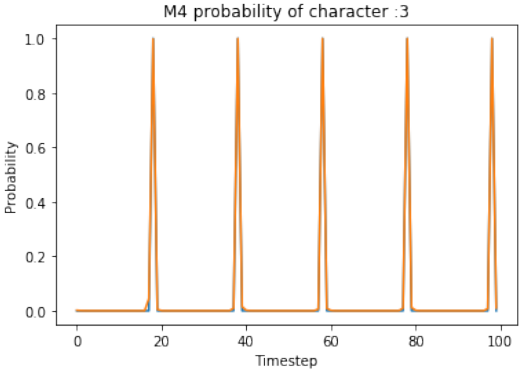
(a) Model 4 (p fixed) - Prob Character 0



(b) Model 4 (p fixed) - Prob Character 1



(c) Model 4 (p fixed) - Prob Character 2



(d) Model 4 (p fixed) - Prob Character 3

Figure 13: Model 4 - (p fixed) Character probabilities through time