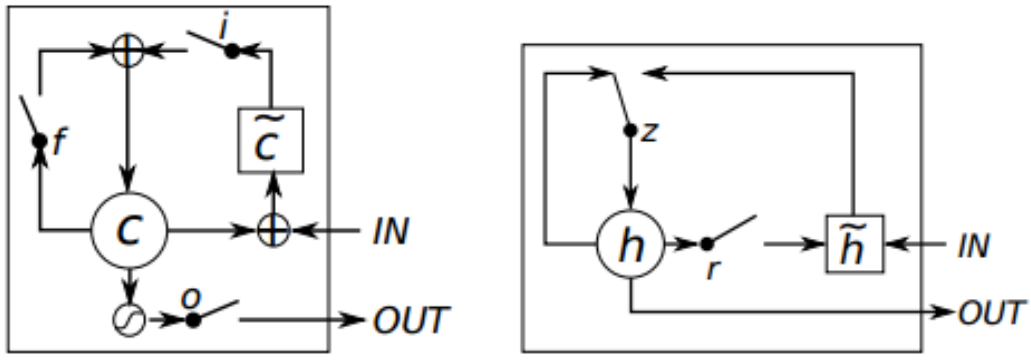
REPORT ADVANCED ML ICA 2

Task 1: Understanding LSTM vs GRU

Feed forward neural networks have no memory i.e. the only input considered is the current one and there is no notion of the previous inputs nor order in time. In contrast, recurrent neural nets are made to recognise patterns in sequences of data. Thus, they take as input not just the current data instance but also what they perceived in the previous instance(s). This 'memory' of the past enables for inference of dependancies in sequential data. LSTMs and GRUs are two types of relatively sophisticated recurrent cells, which avoid the vanishing gradient problem by introducing a gating mechanism (Chung et all. 2014) and are thus commonly used to build RNNs. Below is a figure depicting the recurrent cells (from Chung et all. 2014)



(a) Long Short-Term Memory (b) Gated Recurrent Unit

Figure 1: Illustration of (a) LSTM and (b) gated recurrent units. (a) i, f and o are the input, forget and output gates, respectively. c and $\tilde{c}$ denote the memory cell and the new memory cell content. (b) r and z are the reset and update gates, and h and $\tilde{h}$ are the activation and the candidate activation.

(a) Can LSTMs (and respectively GRUs) just store the current input for the next step? If so, give the gates activation that would enable this behaviour? If not, explain why that cannot be done using this model

Yes these calls can 'forget' the previous states and 'just store the current input' for the next step. This could be beneficial in certain scenarios where we are at start/end of distinct sequences and want to forget what we have learned in the past as it will no longer apply.

In the LSTM, the forget gate is a sigmoid layer outputting values between 0 and 1 for each number in the cell state C(t-1), for the past inputs to be completely forgotten the gates activation needs to be 0 throughout. In the update step the gate needs to be closed and the scaling factor set to one so that all new input is added to the current cell state.

In the GRU 'forgetting' can be achieved by closing the reset gate effectively enabling the cell to reset its memory (r = 0), with the values from sigmoid equal to zero. If the new input is to be 'remembered' the update gate should be closed too, with non-zero output values form sigmoid. In other words, the activation should be such so as to reset the cell and update with new input.

(b) Can LSTMs (and respectively GRUs) store a previous state (say currently in present in ct−1, respectively ht) and ignore the current input? If so, give the gates' activation that would enable this? If not, explain why that can't be done using this model.

Yes both cells have the option of ignoring current input.

An LSTM can ignore current input by setting the forget gate activation to a non-zero value (i.e remember all (=1) or some of the past) and setting the update gate activation to 0 (i,e do not update any value). Similarly, a GRU would achieve this by having an open reset gate (i.e do not reset memory) equivalent to an non-zero activation function. Simultaneously the update gate needs to be open (i.e do not allow update) with the activation function set to 0 throughout (r = 0).

(c) Are GRUs as special case of LSTMs? If so, give the expression of the GRU gates in term of LSTM's gates(ot, it, ft). If not, give a counter-example. Assume here the same input/output.

This question is rather ambiguous, as the answer depends on the definition of 'similarity' or being a 'special case' of something. One could say that these cells are similar as they both achieve comparable results in most RNN tasks (Chung et all. 2014). In terms of architecture the cells both have gates, thus one could say that they are special cases of gated cells, with the difference that LSTMs have 3 gates and GRUs two gates. The GRUs reset gate is similar to the LSTMs forget gate and the GRUs update gate is similar to the combination of LSTMs input/output gates. Despite similarity in the mathematics behind each cell type, one cannot *express* a GRU in terms of an LSTM. Thus, the cells are not spacial cases of one another. One could also argue the cells are fundamentally different because of the way they 'remember'.After all, an LSTM has a separate memory cell and a GRU does not. Thus, with an LSTM you can copy the input to the output and at the same time keep the memory, while in the GRU everything that is outputted is also stored in the memory.
After having done the assignment, in retrospect, one could expect the LSTM to perform better due to the presence of the separate memory cell, but it does not. This is another argument for why a GRU is not necessarily a special case of an LSTM.

REFERENCE:

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

Task 1: Classification

General Remarks:

1) For this task binarization of pixels makes results marginally worse. This is not surprising as slightly changing pixel colour, say around edges of the image, contains information which is lost upon binarizing the image into black and white.
2) It is regrettable that we could not perform this task on batches of 28 pixels (or some other size either than one) as this significantly improves training time, accuracy and the stability of the solution. Furthermore, the edges of the image contain no information and 'confuse' the RNN before meaningful pixels are reached. Therefore, clipping the images to remove the edges, whist still feeding the RNNs with pixel-by-pixel input would have also allowed for better and faster training.
3) Generally the choice of hyperparameters was crucial for the Networks to give satisfactory results. If the wrong parameters were used the network never converged to a solution. This occurred at relatively low batch sizes (100 and below) and it was found that higher batch sizes were not only faster to run but also yelled improved results (given a suitable learning rate)
4) Dropout was found to make results worse thus it was not used in the final version of the model. With the exception of the 3xLSTM model in part C. Batch normalisation was used instead, which is also a form of a regulariser and yelled better results.
5) It should be noted that for all models epoch length for this assignment was chosen to achieve results above 80% accuracy and to minimise training time. Alternatively, the maximum epoch length was limited to 20 due to time constraints.

Model Outline/ Experimental set-up:

Despite changing the RNN's units as specified in parts (a)-(d) certain model architecture remained the same throughout; batch normalisation as well as gradient clipping were used. Before these two enhancements were added to the 'baseline' model the network did not train well.
Gradient clipping was implemented together with the AdamOptimiser to minimise the loss function and to avoid the vanishing/ exploding gradient problem. Gradients were clipped by relatively high min/max values of -0.9 to 0.9. This gradient clipping allowed for the use of a higher learning rate and training in less epochs.
Significant improvement in training time and accuracy was achieved through applying batch normalisation to the first linear layer (before feeding into ReLu) of the output layer. Batch normalisation changes the distributions of the input to the hidden layer, so that the input to the activation function across each training batch has mean and variance of 1. This has been successfully used by Ioffe & Szegedy (2015), whereby introducing changes to the distribution outputs of the hidden layer forces later layers to adapt to this changes during training.
AdamOptimiser was the used to perform optimisation to find the mime cross-entropy loss yielding combination at each iteration.

Choice of hyperparameters:

Hyperparameters were found by trail and error as well as by applying some basic knowledge; for instance that higher batch sizes allow for higher learning rates.

Batch Size and Learning Rate:
Batch sizes ranging from 30 to 1000 were trailed as well as learning rates ranging from 0.0001 to 0.1. It was discovered that higher batch sizes gave better results and final models were trained with batch sizes ranging between 250 and 500. In terms of learning rate it was discovered that relatively high rates, in the order of 0.01 performed well and learned faster than lower rates (due to

gradient clipping). However, for 128 units this did not work and lower learning rates yelled better solutions. Similar rates were used for LSTMs and GRUs. Specific learning parameters used for each case are given in the name of the model in the format: BN_Model_PartX_64.250.0.02, where 64 refers to the unit size, 250 to the batch size and 0.02 to the learning rate (look tables below).

AdamOptimiser parameters:
In part 1 A and B default optimiser parameters were used, but to improve results in part 1 C these were adjusted. The exponential decay rate for the 1st moment estimates was set to 0.95. The exponential decay rate for the 2nd moment estimates was set to 0.99, the epsilon constant was made smaller by one order of magnitude (epsilon = 1e-09) and the lock for update operations was kept at default (false). For part D the default optimiser parameters were used again.

Other:
When it comes to the linear layer initialisation this was sampled from a truncated normal distribution, limiting the sampling space to two standard deviations from the mean. It was found that setting the standard deviation value to 0.1 gave the best results. The batch normalisation decay constant was set to 0.999.

Below tables summarise the final cross-entropy loss and the classification accuracy (expressed as a decimal) results for all architectures considered:

LSTM unit	1 layer, 32 units	1 layer, 64 units	1 layer, 128 units	3 layers, 32 units
Training Loss	0.276512	0.253544	0.330994	0.791111
Training Accuracy	0.913909	0.931382	0.896436	0.696491
Testing Loss	0.268405	0.220628	0.30806	0.815706
Testing Accuracy	0.9165	0.9354	0.9054	0.6979
Model Name	BN_Model_Part1A _32.250.0.018	BN_Model_Part1A _64.400.0.004	BN_Model_Part1A _128.500.0.0005	BN_Model_Part1C _32.500.0.0005
Num. of Epochs	12	18	18	20

Given the lowest training time, LSTM 32 performed very well as compared to the more complex units. If it were trained up to 18 epochs it would likely outperform the LSTM with 64 units. Given the same training epochs, LSTM 64 outperformed LSTM 128 and both these architectures do better than the 3 LSTM cell combination. Generally, for LSTM units there is a trend that performance reduces with complexity. Some of this can be attributed to the need for better hyperparameters tuning for these complex units. However, it is also due to the need for much more training time, which was not possible due to tie constraints. Similarity between testing and training results indicates that models are generally good and did not overfit.

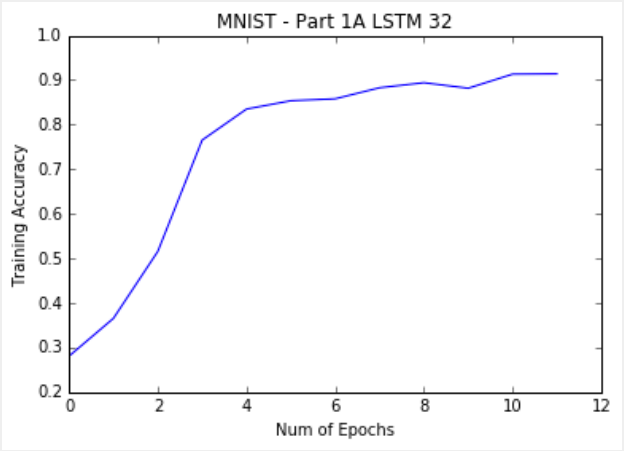
GRU unit	1 layer, 32 units	1 layer, 64 units	1 layer, 128 units	3 layers, 32 units
Training Loss	0.112446	0.123332	0.551495	0.0405764
Training Accuracy	0.965564	0.962855	0.809255	0.988927
Testing Loss	0.108343	0.11783	0.523989	0.0429978
Testing Accuracy	0.9659	0.9627	0.8141	0.9871
Model Name	BN_Model_Part1B _32.250.0.02	BN_Model_Part1B _64.400.0.004	BN_Model_Part1B _128.500.0.0005	BN_Model_Part1D _32.400.0.005

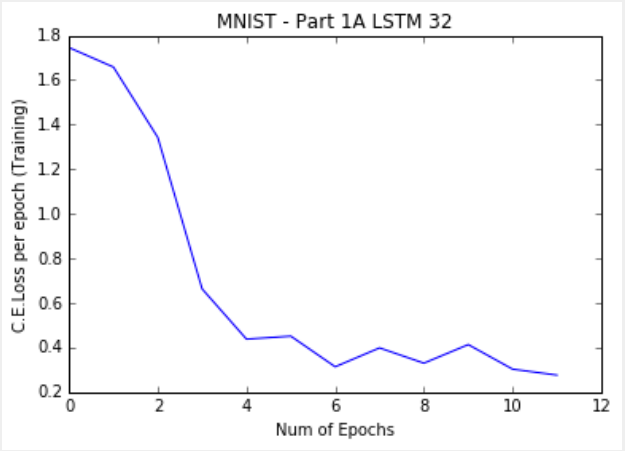
Num. of Epochs	12	10	15	20

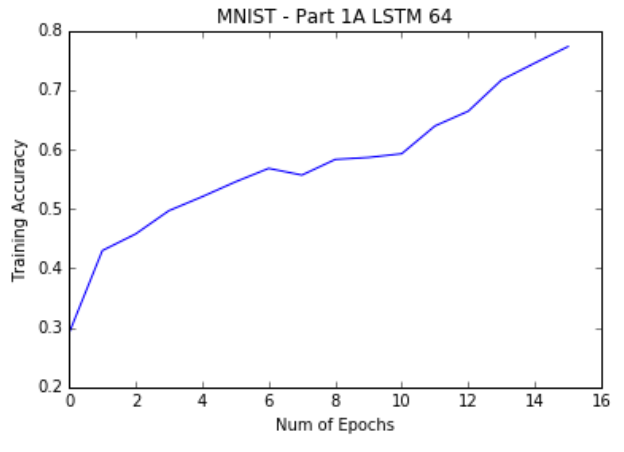
It can be sen that GRU units generally outperform the LSTM units and train faster, or yield higher accuracies with the same amount of epochs. GRU with 64 units is marginally better than that with 32 units as it achieves the same accuracy results in less epochs. I expected a trend to emerge whereby GRU architectures with increasing complexity achieve better results, this has not emerged in the case of GRU with 128 units. However, the most complex architecture (3 times GRU with 32 units) yields the best score out of all the models. Thus it is my belief that the GRU 128 can be made to perform better using different hyperparameters and given more training time. It is interesting to note that the 3 xGRU architecture achieves results of 98% accuracy already after 10 epochs, the model was just left to run for longer (refer to graphs below). Therefore, it is unquestionably the best model in terms of results and speed out of the ones tested in this part of the assignment. Again, similarity between testing and training results indicates that models are generally good and did not overfit.

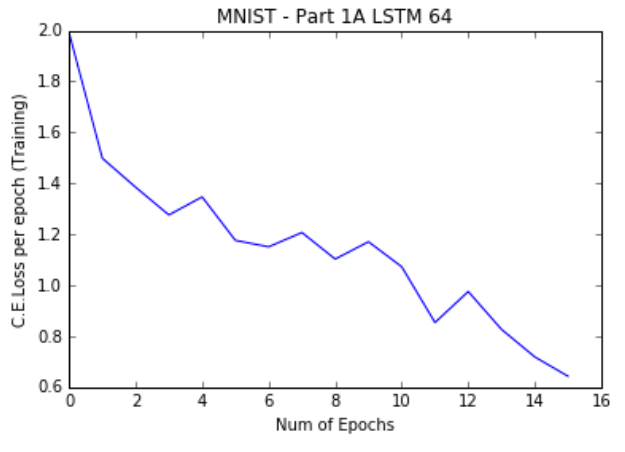
Below are the learning curves - graphs of Accuracy and Cross Entropy Loss for all the models:

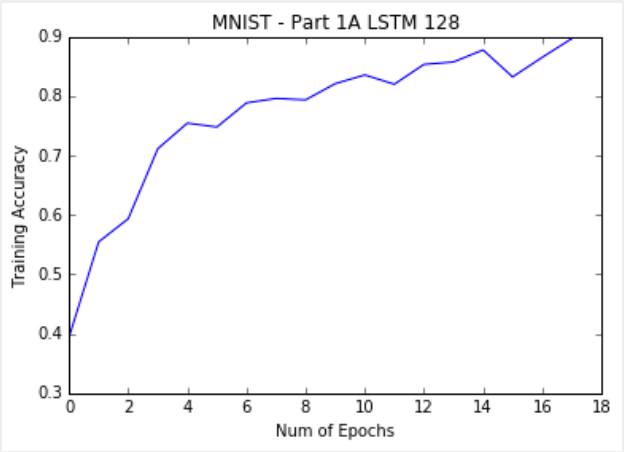
PART 1 A. SINGLE LSTM RESULTS ILLUSTRATED

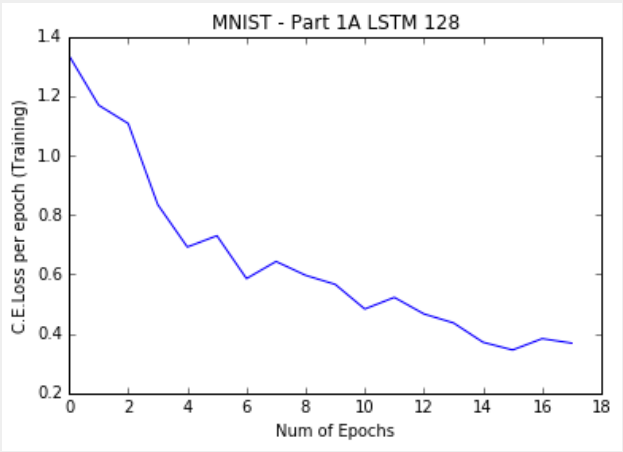




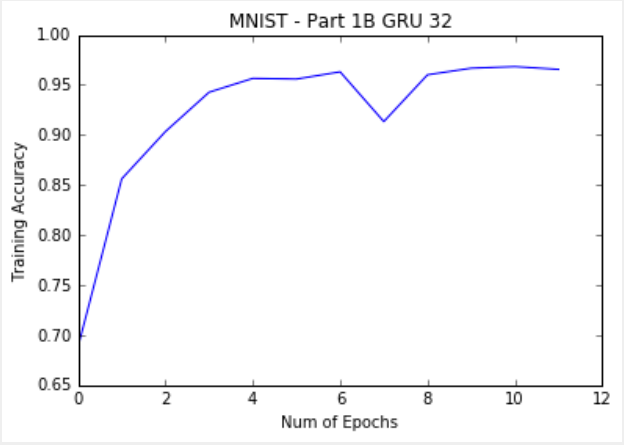


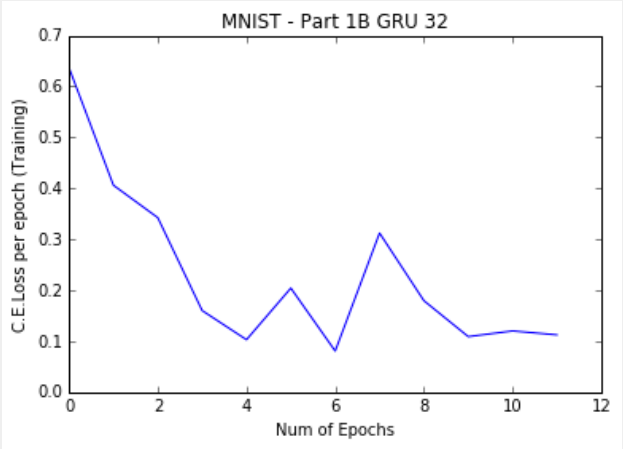


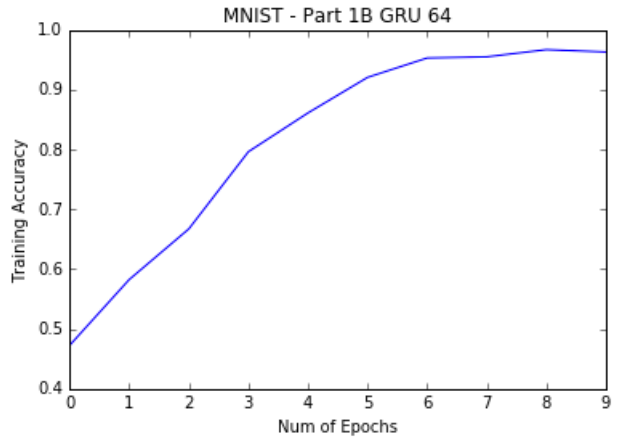


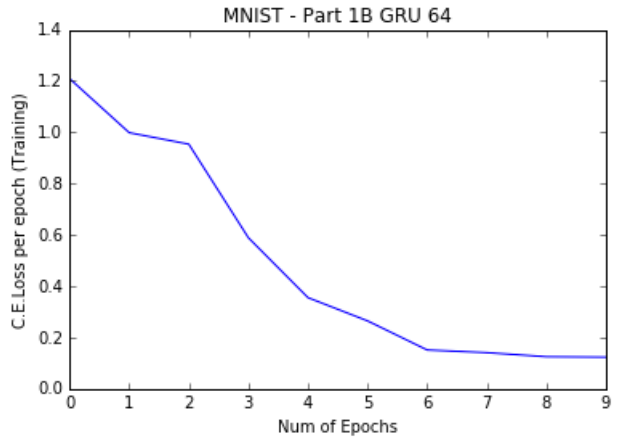


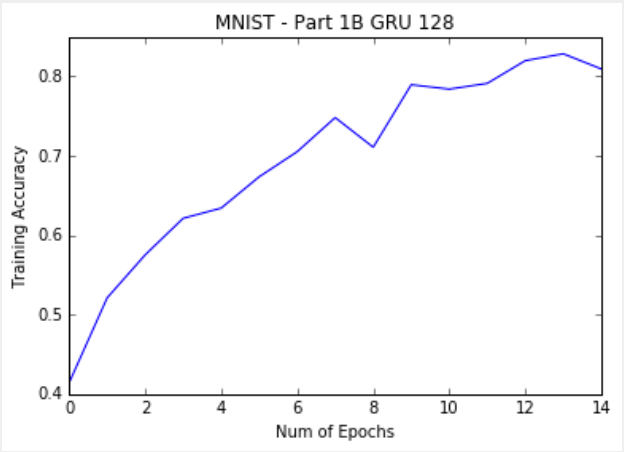
PART 1 B. SINGLE GRU RESULTS ILLUSTRATED

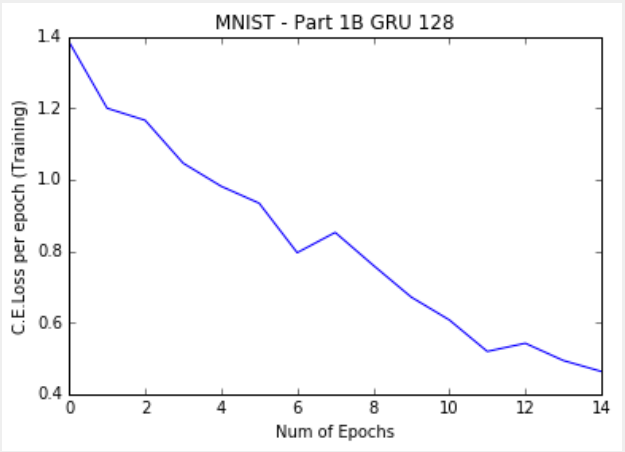




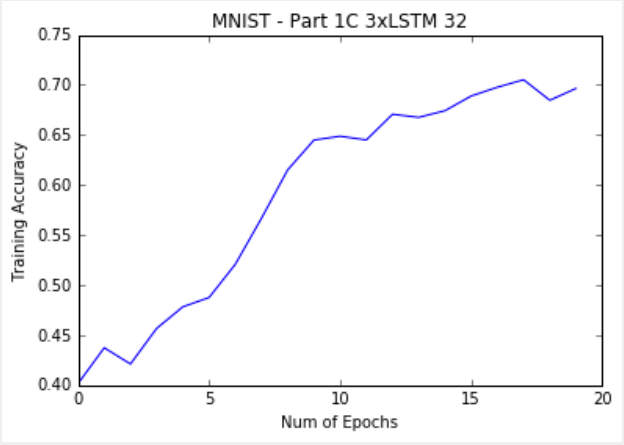


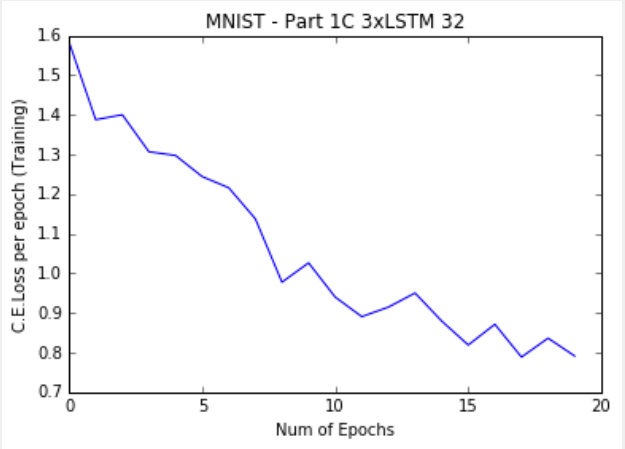




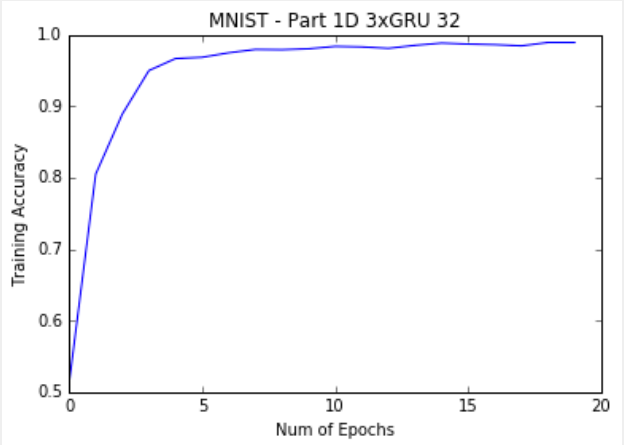


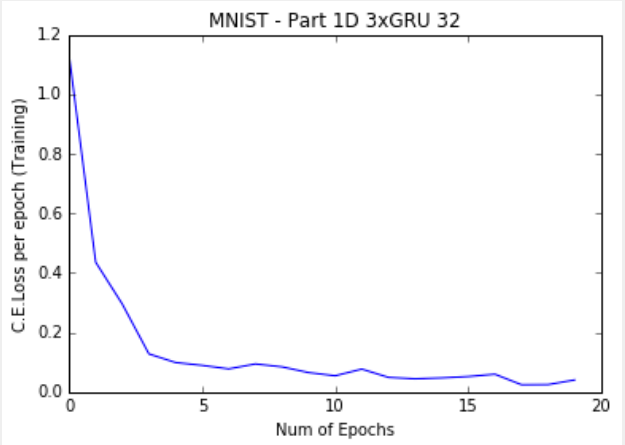
PART 1 C. 3 x LSTM RESULTS ILLUSTRATED





PART 1 D. 3 x GRU RESULTS ILLUSTRATED





Results can be considered stable if the loss and accuracy curves exhibit shapes similar to the on for Part D, where the curve flattens out with the amount of epochs. Thus form the graphs above certain models emerge as more stable than others, for instance, single layer GRUs with 32 and 64 cells are tending towards stability, so does the single LESTM 32 and 128. Thus for this models the amount of epochs and hyperparameters can be considered satisfactory. The 3 layer LSTM curve is hard to interpret as it is somewhat in between, the low accuracy of this model however suggest more training and parameter tuning is needed.

How does this compare with the results you obtained in the first assignment, when training a model that "sees" the entire image at once?

Generally, the results are worse in terms of accuracy and time needed to train as compared with assignment 1. This is due to a number of factors, firstly epoch length for this assignment was chosen to achieve results above 80% accuracy and to minimise training time. If models had more time to run or were run on GPUs in more epochs they would reach results comparable with those in assignment one. Secondly, the results could be improved by more hyperparameters tuning, which was not done to the full extent due to time and computational resource constraints. Furthermore, image classification is not a typical sequence classification task and using a pixel-by-pixel representation which carries long sequences of either 1's or 0's is hardly productive. In practice, RNNs do have a limited memory and the problem setup is not ideal for solving the problem. Sequence classification on groups of pixels (i.e. batches of 28), where information changes more dynamically achieves great results, with accuracy of above 97% within 2-3 epochs depending on the model architecture (parts (a)-(b)). Thus RNN's could have easily outperformed some techniques employed in assignment 1 provided they were given the right input. To illustrate, below is an extract of results obtained in Part 1A LSTM with 32 units using as input batches of 28 pixels at a time. However, it should also be noted that the very construction of CNN's is meant for image classification and it is hard for an RNN to compete with a well tuned CNN in this domain, so it is not certain that, even after much refinement, models developed in this assignment could outperform CNN results form assignment 1.

Improved results for a single LSTM with 32 units trained with batches of 28 pixels :

```
Training Epoch 0 loss: 95.6826269478 accuracy: 0.951691
Training Epoch 1 loss: 28.5760286599 accuracy: 0.973109
Training Epoch 2 loss: 22.0533351451 accuracy: 0.973836
Training Epoch 3 loss: 19.0052739289 accuracy: 0.977782
Training Epoch 4 loss: 17.0294087473 accuracy: 0.983964
Test loss: 0.073336 Accuracy: 0.9775
```

Task 2: Pixel prediction

In this part of the assignment the problem formulation was changed to train a many-to-many recurrent model. This received as input a pixel value and history and tried to predict the next pixel, based on this input and the recurrent cell state. Based on superior results in part 1, the GRU recurrent unit was chose above the LSTM.

As the images were binariesed, the problem setup changed so that the labels became categorical variables (0 or 1) instead of a vector of ten classes. However, the model was trained using an approach very similar to the one for part 1. An RNN was built as per the assignment specification, batch normalisation was added to the model but gradient clipping was not, as it did not have a substantial impact. Optimisation was done using AdamOptimiser with a changed learning rate and other parameters kept at the default settings. Similar Hyperparameters as in part 1 were kept and yielded satisfactory results of above 95% accuracy. However, if time allowed this could have been tuned further to achieve better results.

Below tables summarise the final cross-entropy loss and the classification accuracy (expressed as a decimal) results for all architectures considered:

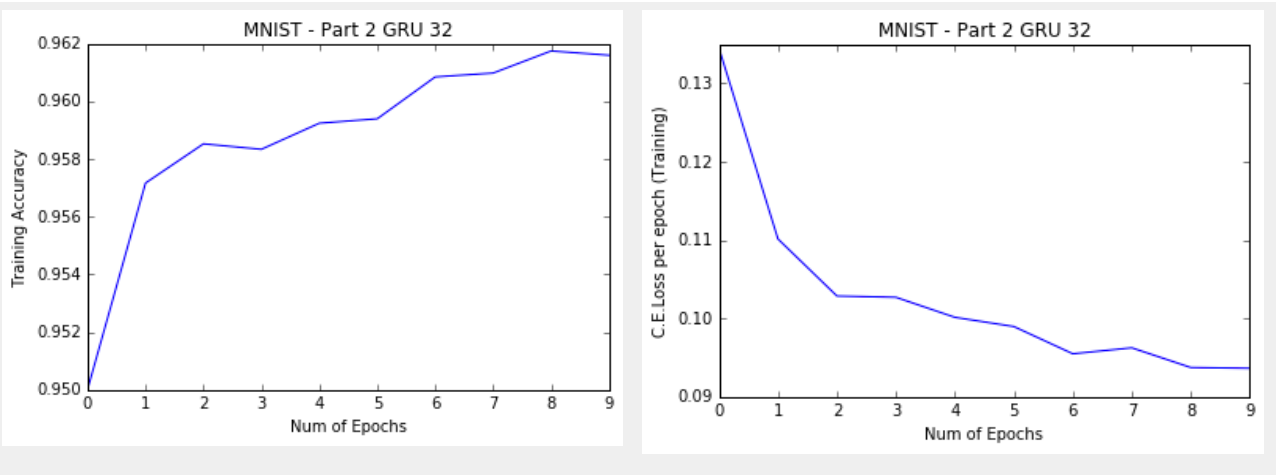
GRU unit	1 layer, 32 units	1 layer, 64 units	1 layer, 128 units	3 layers, 32 units
Training Loss	0.0948355	0.103025	0.291621	0.104101
Training Accuracy	0.959876	0.961232	0.950909	0.961057
Testing Loss	0.0953526	0.105305	0.291939	0.104921
Testing Accuracy	0.960231	0.961017	0.948764	0.961155
Model Name	BN_Model_Part2A_GRU_32.250.0.02	BN_Model_Part2A_GRU_64.400.0.004	BN_Model_Part2A_GRU_128.500.0.0005	BN_Model_Part2B_3GRU_32.400.0.004
Num. of Epochs	10	10	10	10

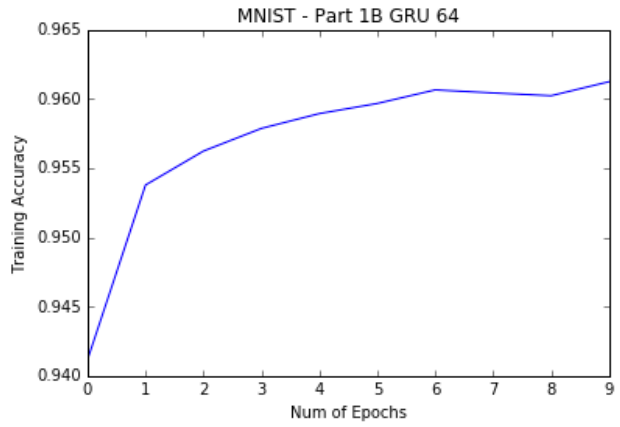
From the results presented in the table above, it can be seen that the many-to-many prediction problem can be modelled with RNNs to achieve better results (in terms of accuracy) than the many-to-one problem from part 1. It also requires considerably less running time. However, off course these are different problems.

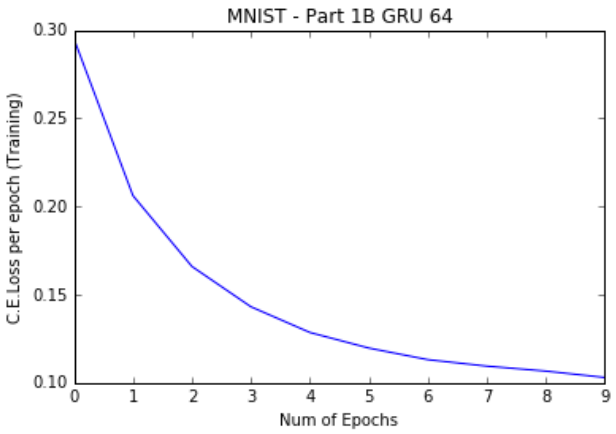
Assuming any accuracy above 95% is a good result, all models could be employed for prediction, as the differences in results from model to model are very slight. It should also be noted that all four models were run on 10 epochs to aid comparison. After some experimentation, it was also noted that models run for longer did not depart far form the 96% accuracy reached after 10 epochs either. In this event the best model is the simples one, the one layer GRU with 32 units. It is interesting to observe that although it performs slightly worse on the training set it achieves the best test set loss and accuracy. The most complex and computationally expensive model 3xGRU, which outperformed other models considerably in part 1, only performs slightly better for the many-to-many task at hand. Looking at the graphs below, learning curves generally exhibit good shapes and results can be considered stable, with the slight exception of the single layer GRU with 128 cells. This model perform worse, in terms of accuracy as well as the learning curve (loss curve) shape. One could argue that this model has not yet converged, and requires more training epochs, but in this case it should be discarded in favour of the simpler models.

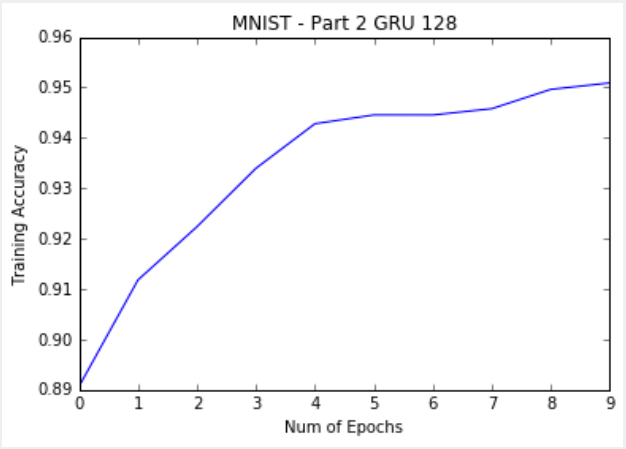
Below are the learning curves - graphs of Accuracy and Cross Entropy Loss for all the models:

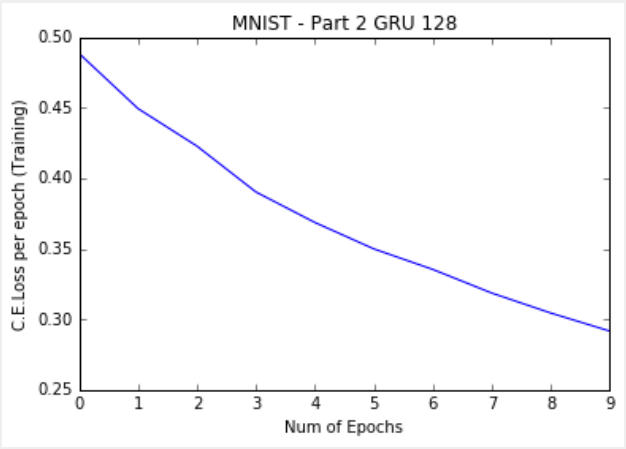
PART 2 GRU RESULTS ILLUSTRATED

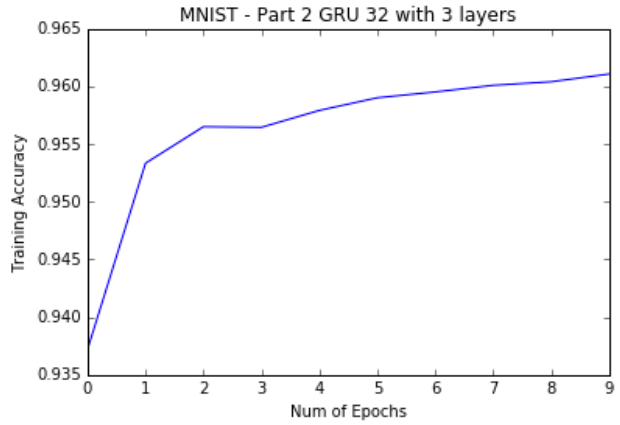


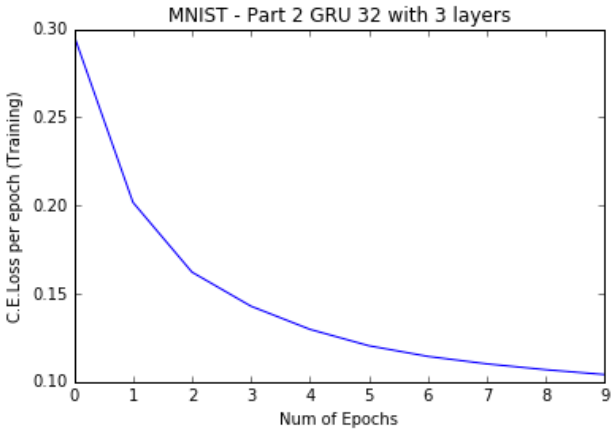












Task 3: In-painting

Refer to written solution on next page

Klaudia Ludwisiak ICA 2 Task 3A.

a) Provide formula for probability of missing pixel
→ call this pixel A, given the past pixels from time t-1 to time = t, denoted by P and also given the future pixels denoted as F.

Using Bayes rule:

$$P(A=1 \mid P,F) = \frac{P(A=1,P,F)}{P(P,F)} \quad \begin{array}{l} \leftarrow \text{this is known} \\ \leftarrow \text{this is unknown} \end{array} \quad (1)$$

similarly:

$$P(A=0 \mid P,F) = \frac{P(A=0,P,F)}{P(P,F)} \quad \begin{array}{l} \leftarrow \text{known} \\ \leftarrow \text{unknown} \end{array} \quad (2)$$

also know that: $P(A=1 \mid P,F) + P(A=0 \mid P,F) = 1$ (3)
as probability distribution needs to sum to 1.

→ Rearranging the first two equations to eliminate the unknown:

$$\frac{P(A=1,P,F)}{P(A=1 \mid P,F)} = \frac{P(A=0,P,F)}{P(A=0 \mid P,F)}$$ using the 3rd equation:

$$P(A=1,P,F)\,P(A=0 \mid P,F) = P(A=0,P,F)\left(1-P(A=0 \mid P,F)\right)$$

rearranging further to obtain $P(A=0 \mid P,F)$:

$$\frac{1-P(A=0 \mid P,F)}{P(A=0 \mid P,F)} = \frac{P(A=1,P,F)}{P(A=0,P,F)}$$

probability of missing pixel given past & future

$$\Rightarrow P(A=0 \mid P,F) = \frac{P(A=0,P,F)}{P(A=0,P,F) + P(A=1,P,F)}$$

analogously:

$$P(A=1 \mid P,F) = \frac{P(A=1,P,F)}{P(A=1,P,F) + P(A=0,P,F)}$$ (4)

- For the case of a 2×2 patch:
- The above presented logic can be extended to 4 pixels, which I will call A, B, C, D. ~~The spatial~~ These will also have a past denoted as P and future denoted as F.
- Before, we only had two cases: A=0 or A=1.
- Now, we have 4 missing pixels with a total of 16 possible combinations of values.

Therefore, we are now looking for the probability:

$P(A=0, B=0, C=0, D=0 \mid P, F)$ and the 15 other configurations of values of A, B, C & D.

Using the same logic as in ~~the~~ previous part we can write:

(i) $$P(A=0,B=0,C=0,D=0 \mid P,F) = \frac{P(A=0,B=0,C=0,D=0,P,F)}{\text{sum over 16 combinations}}$$

$$\text{sum over 16 combinations} = \begin{cases} P(A=0,B=0,C=0,D=0,P,F) + \\ P(A=1,B=0,C=0,D=0,P,F) + \\ P(A=0,B=1,C=0,D=0,P,F) + \\ \vdots \\ P(A=1,B=1,C=1,D=1,P,F) \end{cases}$$

Formula (i) can then be expressed in terms of the probability of the 16 different pixel states whereby the nominator is the joint probability of the state in question and the past (P) and future (F), and the denominator is the sum over all 16 states.