Deep Learning Assignment 3

Student Number:13064947 - John Goodacre

February 23, 2018

1 Introduction

This is part of the submission for Deep Learning assignment 3. The associated code and implementations may be found in the emailed Colab link and the Jupyter notebook included in the zipped submission. To save space but also help orient the reader, in each section I have included the very first part of the question referred to followed by (...).

2 Question 1: Understanding LSTM vs GRU

2.1 Question 1, Part 1

Can LSTMs (and respectively GRUs) just store the current input in the state? ...

First for LSTM's. Using the notation from the lectures, the definitions for our various gates and two states (with W's, b's comprising the respective weights and biases):

$$
\begin{aligned}
i_t &= W_{ix}x_t + W_{ih}h_{t-1} + b_i \\
j_t &= W_{jx}x_t + W_{jh}h_{t-1} + b_j \\
f_t &= W_{fx}x_t + W_{fh}h_{t-1} + b_f \\
o_t &= W_{ox}x_t + W_{oh}h_{t-1} + b_o
\end{aligned}
$$

$$
\begin{aligned}
c_t &= \sigma(f_t) \odot c_{t-1} + \sigma(i_t) \odot \tanh(j_t) \\
h_t &= \sigma(o_t) \odot \tanh(c_t)
\end{aligned}
$$

The question really asks if we can make $c_t = x_t$. Clearly for this to happen we would at least need $\sigma(f_t) = 0$ for all entries and $\sigma(i_t) = 1$ for all entries. Given these are sigmoid functions we would effectively need to ensure very low negative values for f_t, regardless of the inputs x_t and h_{t-1}. So let's cheat and

just force the gates $\sigma(f_t) = 0$ and $\sigma(i_t) = 1$.

Presuming we did this then we would have $c_t = \tanh(W_{jx}x_t + W_{jh}h_{t-1} + b_j)$. So now we would now hope to 'force/ train' the weights $W_{jh} = 0$ to remove h_{t-1} and then force the weights $W_{jx} = I$ and have a zero bias. This would leave us with $c_t = \tanh(x_t)$. So we still have a problem, a squashing function and so are still not storing the current input. If we changed the activation function to linear then yes we could, but I would argue this is no longer an LSTM (one gate forced to 0, one to 1, a set of weights changed to the identity, another to zero and a linear activation instead of tanh). So a perhaps long winded way of saying no for the LSTM unless you brutalise it so much, that it is not an LSTM anymore.

GRUS: The equations for a GRU are,

$$
\begin{aligned}
z_t &= \sigma(x_t U_z + h_{t-1} W_z + b_z) \\
r_t &= \sigma(x_t U_r + h_{t-1} W_r + b_r) \\
\tilde{h} &= \tanh(x_t U_h + (h_{t-1} \odot r_t) W_h + b_h) \\
h_t &= (1 - z_t) \odot \tilde{h} + z_t \odot h_{t-1}
\end{aligned}
$$

Here the question asks if we can make $h_t = x_t$, again by a similar argument to the LSTM we would need in this case $z_t = 0$, under all circumstances. Even if we forced the gate to this value and again 'forced/ learned' weights such that $b_h = 0$, and $W_h = 0$, and $U_h = I$, we would still end up with $h_t = \tanh(x_t)$.

Again we have a squashing function and the only way we could make these values equal would be to remove this non-linearity. So again by a similar argument we cannot achieve this in general. Without fundamentally changing the GRU.

2.2 Question 1, Part 2

Can LSTMs (and respectively GRUs) just store a previous state into the current state and ignore the current input? ...

LSTMs, Yes - but perhaps not 100% exactly ! If we reduce the problem to storing the previous state c_{t-1} and ignoring the current input, then we see that if we can do this then we can also do this for previous states. To do this we would require $\sigma(f_t) = 1$ and $\sigma(i_t) = 0$, given these are sigmoid activations they only exactly equal these values at $\pm\infty$, but in practice given an existing state, if subsequent activations f_t are very high and i_t very low, then the LSTM will keep the current state and ignore the new inputs. Capturing some of these long term dependencies is part of what the LSTM is designed for.

GRUs, Yes again - by a similar but simpler argument we need $z_t = 0$, then $h_t = h_{t-1}$, again in practice we might approach these limits rather than hit them exactly, but essentially for practical purposes the state can be stored and inputs essentially ignored.

2.3 Question1, Part 3

Are GRUs a special case of LSTMs? ...

No GRUs are not a special case of LSTM's. One example might be our previous example. Imagine a very poor input and both the LSTM and the GRU wish to store their previous state and that this was the same for both, and they both receive this same input. The GRU will also output this previous state (by the arguments given above). However in the case of an LSTM even if its output gate is set to one. Then $h_t = \tanh(c_t)$, thus there is an extra squashing function on the output for the LSTM and they are not the same.

This proves they are not the same but in fact the argument is more general than this specific example. If $f_t = z_t$, $i_t = 1 - f_t$ and as before we force the output $o_t = 1$, then this would be the same as $r_t = 1$ for the GRU. But again the output would be squashed by the tanh activation for the LSTM, but not the GRU. They are just different, and indeed their behaviour in practice can be different too. The GRU can be faster due to its simpler architecture. Often however, results can be quite similar. As far as I know the definitive clear answer as to when to use an LSTM and GRU is not yet known and we remain in a world of heuristics.

3 Question 2, Train Models

A line by line implementation of MNIST was trained, using the respective architectures and hyper-parameters specified in the Jupyter notebook question. (Adam - LR-0.001, Batch-256, Epochs-10, Fully connected units-64, Loss - reduce sum xent, this is following Assignment 1, I prefer mean for reporting purposes). Please see the code for the implementation.

Clearly hyper-parameters and training can be improved. But bar adding batch normalisation, I simply ran the with parameters as asked for in the question. **Please see the results in fig 1**. With these settings the GRU with 128 neurons happened to perform the best on the test set with an accuracy of 0.9864.

LSTM	(1 layer, 32 units)	(1 layer, 64 units)	(1 layer, 128 units)	(3 layers, 32 units)
Test loss	877.76276	702.0221	510.27527	605.4952
Test accuracy	0.972	0.979	0.9838	0.9812

GRU	(1 layer, 32 units)	(1 layer, 64 units)	(1 layer, 128 units)	(3 layers, 32 units)
Test loss	822.5758	573.5083	412.55176	539.20386
Test accuracy	0.9736	0.9819	0.9864	0.9836

Figure 1: Accuracy and Loss (sum), LSTM/ GRU model results

4 Question 3: Analyse the Results

4.1 Question 3, part 1

How does this compare with the results you obtained in the first assignment(DL1)...

We tested 4 models in DL1. A pure softmax, a one layer feed-forward net with 32 hidden units. A two layer net with the same amount of units for each hidden layer and a convolutional neural network.

There is not much to say about model one, it lacks power and is not really expressive enough for the problem. Its maximum accuracy on the test set is somewhere around 0.92. The feed-forward networks managed a maximum accuracy of around 0.965. The convolutional neural network a maximum accuracy of 0.985.In this assignment, the results for both the LSTM and GRU were both excellent, even the small models with 32 neurons 'out of the box' managed an accuracy of over 0.97, and the larger models quickly reached 0.985, comparable to the LSTM.

Explanation. The feed-forward networks treat each pixel independently.

Given a big enough model, enough data and enough layers - they 'should' be able to have an excellent classification accuracy, but it is extremely inefficient. With a CNN we have a network designed for the problem using kernels, parameter sharing etc. utililising what we already know about natural images and embedding this as a prior on the architecture of the model.

The LSTM/ GRU take a quite different approach. None of the other models have a memory or state. They take an image or batch of images and feedforward to give an output. The LSTM/ GRU do too, but only if unrolled through time, so at any given time point they have an additional state. As rows are fed in for an image the state of the LSTM/ GRU changes and outputs from the neurons get stronger. It is simply a different architecture and approach. My only slight surprise was that the LSTM/ GRU does just as well in this simple set up as the CNN which I felt is 'designed' for this particular task.

4.2 Question 3, part 2

Take a look closer look at one of the trained models: say GRU-32. Plot the outputs of the RNN layer and hidden state over time...

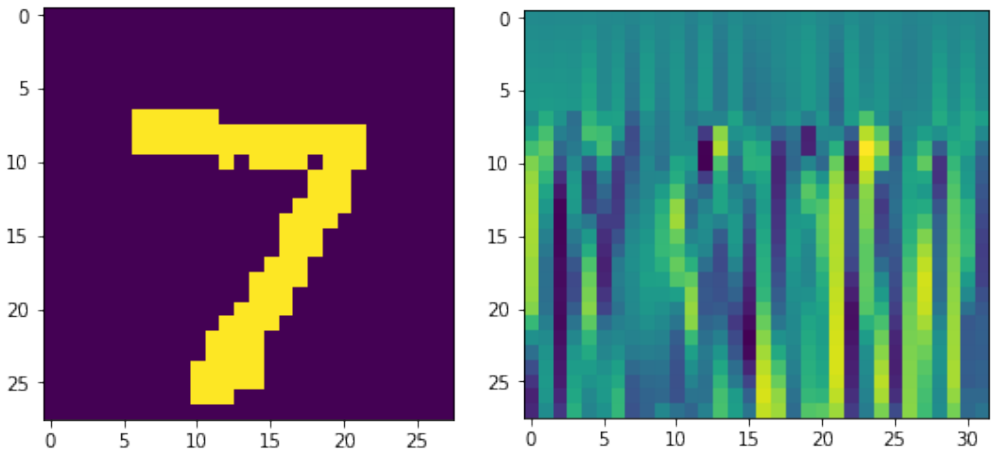
In fig 2, we can see an example binarized test image. For the GRU-32 the accuracy on the test-set was over 97%. This particular example was classified correctly. The image to the right shows the outputs from the 32 neurons through time as slices of this image are passed through the GRU. For the first few time-steps we see very low values. The image is dark. However as we move down to the bottom we see much higher outputs as the network becomes far more sure of the classification.

In fig 3, I show the first and last outputs by time explicitly. This part of the question displays the outputs. I will go into more detail with a further experiment and interpretation in the next section.

4.3 Question 3, part 3

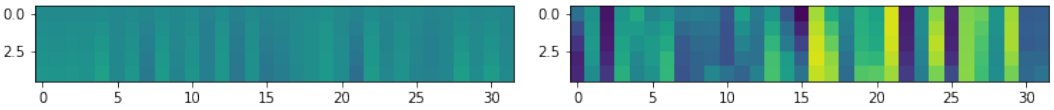
Now, look at the last 3-5 time steps...interpretation...(last 5 were shown in prior part - below is a further experiment for interpretation).

I stored the outputs from the GRU for every time-point, for every image. This was fed into the rest of the trained network. I calculated accuracies for the entire test-set, but based upon which output (at time t=1 to 28), was fed into the rest of the network. I also selected the max value of the softmax for every image, for every time-slice. This enabled me to show the average of the max of the softmaxes for the entire test-set by time, together with the corresponding average accuracy. Fig 4.



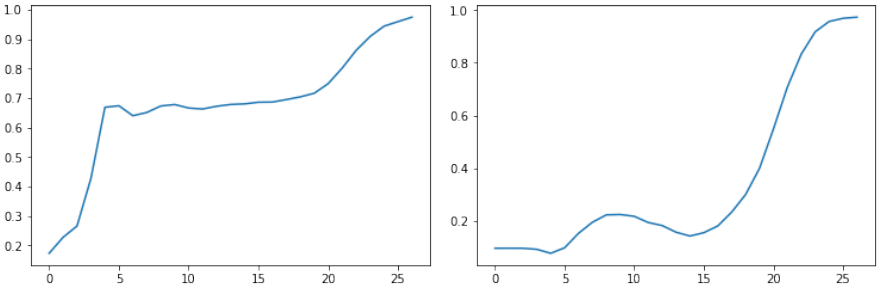
(a) Binarised Test example 28x28 (b) Neuron outputs by time-slice

Figure 2: Neuron Outputs for GRU-32 through time



(a) Neuron activations first 5 time slices (b) Neuron activations last 5 time slices

Figure 3: GRU32 comparison for single image - activations by time



(a) Average Max of Softmax by time-slice (b) Average accuracy by time-slice

Figure 4: GRU32 conf. and acc. by time slice on the whole test set

The final figure, Fig 4 shows the results. Note, that given 10 different categories, a random answer would be an accuracy and max softmax equal to 0.1. The model and accuracies begin here, however as more rows of the images are fed into the GRU the max softmax increases. Interestingly it is only later that there is a fairly dramatic corresponding increase in the classification accuracy.For the first few time-slices accuracy is essentially random. In contrast, for the last 5-10 timesteps we have a very sharp increase in both the max-softmax (the model is very confident in its prediction), and also in terms of the test set accuracy. I believe this shows how and when the model learns through time. Basically it becomes both very accurate and very confident over the last 5-10 timesteps as shown by the averages on the entire 10,000 image test set in fig 4.