Assignment 1 Report

This is an outline for your report to ease the amount of work required to create your report. Jupyter notebook supports markdown, and I recommend you to check out this cheat sheet. If you are not familiar with markdown.

Before delivery, **remember to convert this file to PDF**. You can do it in two ways:

1. Print the webpage (ctrl+P or cmd+P)
2. Export with latex. This is somewhat more difficult, but you'll get somehwat of a "prettier" PDF. Go to File -> Download as -> PDF via LaTeX. You might have to install nbconvert and pandoc through conda; `conda install nbconvert pandoc`.

Task 1

task 1a) Part 1

task 1_1

task 1_2

task 1_3

task 1_4

task 1a) Part 2

task 1_5

task 1_6

Task 2

Task 2c)

Task 2d)

i*h + h*o + i + o = 785*64 + 64*10 + 64 + 10 = 50954

Task 3

Task 3a: Improve weight initialization

We initialize the weight matrix using a normal distribution where we use a mean of 0 and a standard deviation of $\frac{1}{\sqrt{fan-in}}$ being "fan-in" = 785 in the hidden layer and 64 in the output layer.

As we can see in the **Training and validation loss** figure the gap betweeen the validation and the training loss show us that the model with improved weights is more generalized, and as a consecuence has less overfiting.

In the **Training and validation accuracy** figure we can see that the model with improved weights converge faster to a solution using less steps than the normal model and also it provides a more accurate solution.

The normal model bump out at epoch 31 and the improved weights has a faster convergence bumping at the epoch 29.

task 3a loss

task 3a accuracy

Task 3b: Improve Sigmoid

The new improvement was changing the sigmoid activation function in the hidden layers. The results of this function are not centered arround zero so we use the sigmoid activation function discussed by **LeCun**.

For the forward activation we use the following sigmoid function:

$$f(x)=1.7159*tanh\left(\frac{2}{3}x\right)$$

And for the backpropagation we use the following sigmoid function derivation:

$$f(x)=\frac{1.7159*2}{3}*\left(1-tanh\left(\frac{2}{3}*x^2\right)\right)$$

As we can see the in the validation loss the sigmoid model validation loss the accuracy goes even further. But as we can appreciate the the is a large gap between the validation loss and the training loss, meaning that the model begging to overfit.

This model has a improvement in speed and convergence rate in comparison with the other models.

For this improvement we get a significant improvement in the number of epoch going from 29 to 8.

task 3b loss

task 3b accuracy

Task 3c: Momentum

The last addition is adding the momentum for the gradient descent. Momentum makes that the gradient convergence faster. The momentum is given by:

$$\Delta w(t) = \frac{\alpha * \partial C}{\partial w} + \gamma \Delta w(t-1)$$

The training with all this show some overfiting. The validation loss is more that the improved sigmoid. The model is the fastest one. And the accuracy its almost perfect, this could be becouse the model was trained with a small data set.

task 3c loss

task 3c accuracy

Task 4

Task 4a,b)

Here we compare the results of the network with different numbers of hidden layers. We use a 32 layer network, 64 and a 128. The base of our comparison is the 64 layer network because is the one that we use for the hole assignment.

The 32 layer network perfoms worse than the 64 layer as we can see in due to the differeces between the training loss or the validation loss. The speed of the network also is prety slow. So we can say that a small number of hidden units will perform bad.

The 128 layer network performs similary than the 64 layer network but it's slower in comparision. So The 128 layer network show that increasing the number of hidden units only provide a small improvement but it compromise the speed of the network.

Task 4ab loss Task 4ab accuracy

Task 4d)

Task 3: i*h + h*o + i + o = 785*64 + 64*10 + 64 + 10 = 50954 Task 4d.1: i*h + hPOWERHiddenLayersAmount + h*HiddenLayersAmount + h*o + i + o = 785*64 + 64POWER2 + 64*2 + 64*10 + 64 + 10 Task 4d.2: i*h + hPOWERHiddenLayersAmount + h*HiddenLayersAmount + h*o + i + o = 785*64 + 64POWER10 + 64*10 + 64*10 + 64 + 10 The 4d.1 probabilly behaves better than the original one. 4d.2, despite having a greater amount of parameters, would perform worse (overfitting)

Task 4e)

There would be a better training loss and a worse validation loss