Russel Stuart Daries
UCABRSD

STUDENT

FIRST LETTER OF SURNAME: D

PERSONAL TUTOR

Coursework Coversheet

Submission Details

Please ensure that the details you give are accurate and completed to the best of your knowledge.

COURSEWORK TITLE	Assignment 1
LECTURER	Mr. Thore Graepel
MODULE CODE	COMPGI13
MODULE NAME	Advanced Topics in Machine Learning
DEADLINE	7 / 02 / 2017

Declaration

I have read and understood the UCL and Departmental statements and guidelines concerning plagiarism.
I declare that:

1. This submission is entirely my own unaided work.
2. Wherever published, unpublished, printed, electronic or other information sources have been used as a contribution or component of this work, these are explicitly, clearly and individually acknowledged by appropriate use of quotation marks, citations, references and statements in the text.
3. In preparing this coursework, I discussed the general approach to take with the person(s) named below, however the content of the submission is my own work alone:

Person(s) with whom I have discussed this coursework:

Signature: R.Daries

For Departmental Office Use

DATE AND TIME RECEIVED

INITIALS OR STAMP

COMPGI13: Advanced Topics in Machine Learning Assignment #1

Due on Monday, February 7, 2017

Thore Graepel and Koray Kavukcuoglu

Russel Stuart Daries: UCABRSD

February 7, 2017

Contents

P1:

(a)

The following Neural Network (NN) was created in order to train and accurately classify the MNIST digits.

$$input \rightarrow linear\ \ layer \rightarrow softmax \rightarrow class\ \ probabilities$$

This resultant neural network achieved a training accuracy rate of 92.0% and a test accuracy of 92.72%. Therefore, given 100 test examples of handwritten digits from the MNIST test corpus only 7 would be classified incorrectly. This NN model and all the subsequent models were trained using the Cross-Entropy Loss function as shown in Equation 1 below using stochastic gradient descent for optimizing this objective,

$$loss = -\sum_{i=1}^{N} log\left(\frac{exp(z_i[y_i])}{\sum_{c=1}^{10} exp(z_i[c])}\right) = -\sum_{i=1}^{N}\left(-z_i[y_i] + log\left(\sum_{c=1}^{10} exp(z_i[c])\right)\right) \tag{1}$$

Figure 1 and 2 shown below was created in order to visualise the learning process as the Neural Network continuously learns and improves its classification performance [1] and reduces the loss.

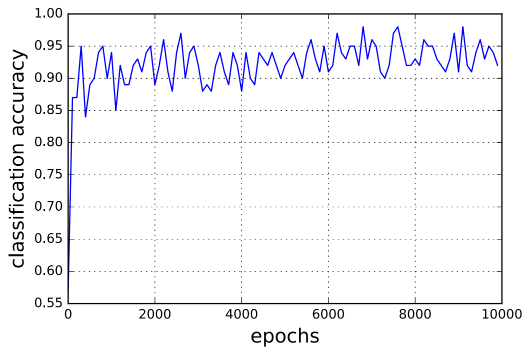

Figure 1: Classification accuracy on training set.

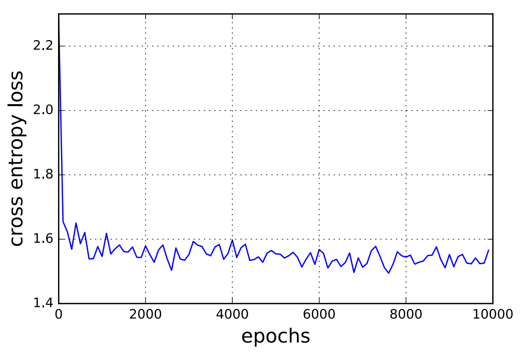

Figure 2: Cross entropy loss on training set.

After 10000 epochs on the training set, the classification error was reduced to 7% and a training cross-entropy loss of 1.56526. The normalised confusion matrix describing the classification can be seen in Figure 3 shown on the following page.

The table describing each of the test cases can be seen below.

Experiment	**P1:a**	**P1:b**	**P1:c**	**P1:d**
Training Error Rate	0.92	1.00	1.00	1.0
Test Error Rate	0.9272	0.9758	0.980	0.9733

Table 1: Error rates for P1.

P1: [(a)] continued on next page. . .

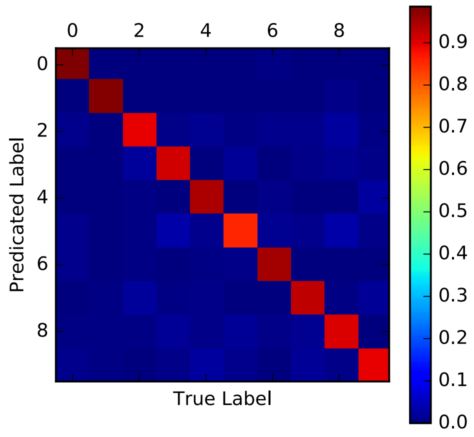

Figure 3: Confusion matrix on MNIST test set for P1b.

It can be seen due to the high values associated with the diagonal elements in Figure 3 the classifier works well in predicting the correct label. The optimal learning rate was found to be $\lambda_{optimal} = 0.8$.

The code used to complete this task can be found in *P1a.py*. The saved weights for this model can be found in the directory *model_training_weights*\P1a.

(b)

Although Q1a discussed yielded favourable results by adding a hidden layer to the neural network improved classification performance was possible. The NN architecture described below was implemented.

$$input \rightarrow non-linear\ \ layer \rightarrow linear\ \ layer \rightarrow softmax \rightarrow class\ \ probabilities$$

The resultant training process can be seen below in Figure 4 and Figure 5 shown below.

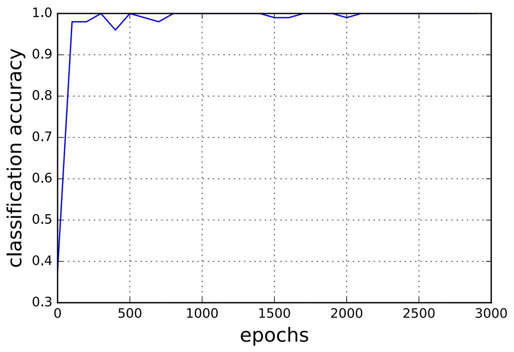

Figure 4: Hidden Layer NN classification accuracy on training set without dropout.

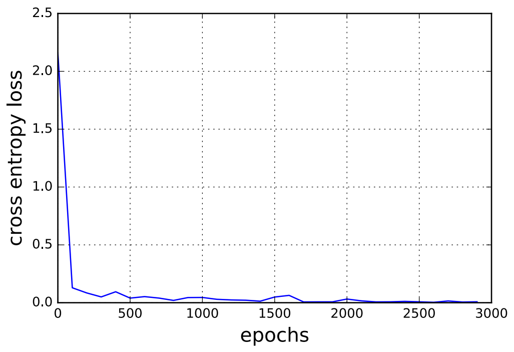

Figure 5: Hidden Layer NN cross entropy loss on training set without dropout.

By adding the hidden layer the classification performance improved to 97.58% shown in Table 1. The test cross-entropy was 0.0814. This model can be seen in Figure 4 and Figure 5, the model over trained rapidly. The optimal learning rate was found to be $\lambda_{optimal} = 0.8$. The training accuracy for this model was 100% which suggest potential overfitting of the data. However, in this instance the model performed well on the test set.

The confusion matrix associated with the new NN model can be seen in Figure 6 shown below.

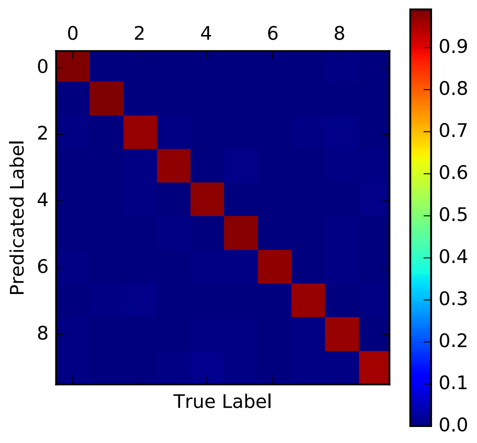

Figure 6: Confusion matrix on MNIST test set for P1b without dropout.

It can be seen when comparing Figure 3 with 6, the diagonal elements of Figure 6 are brighter, indicating better performance.

The model was also tested by adding dropout in the output layer. This was done in order to reduce overfitting. The resultant test accuracy was 97.54% and cross-entropy loss of 0.0860. The training of the model with dropout can be seen in Figure 7 and 8 shown below. The training accuracy (86.0%) and cross-entropy error was 0.4645 which is lower than the original model, thus reducing overfitting.

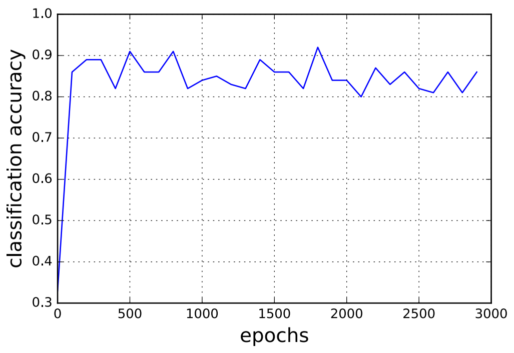

Figure 7: Hidden Layer NN classification accuracy on training set with dropout.

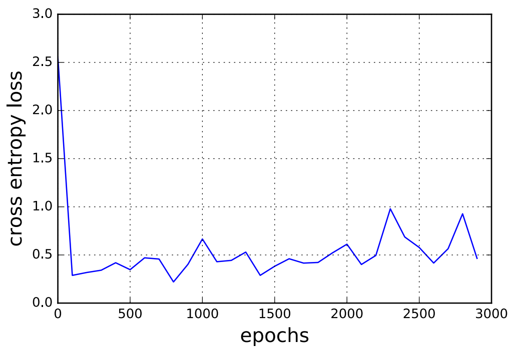

Figure 8: Hidden Layer NN cross entropy loss on training set with dropout.

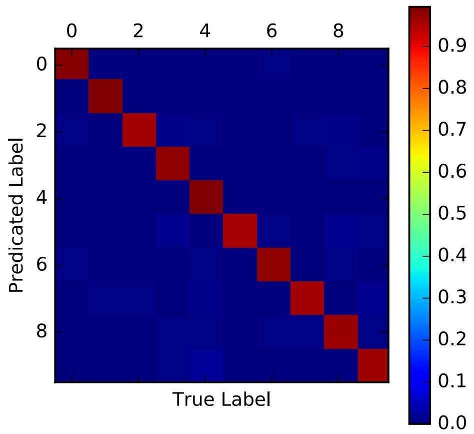

Figure 9: Confusion matrix on MNIST test set for P1b with dropout.

The code used to complete this task can be found in *P1b.py*. The saved weights for this model can be found in the directory *model_training_weights*\P1b.

(c)

The single hidden layer NN shown earlier was improved upon further by creating a NN with two hidden layers as described below.

$$input \rightarrow non-linear\ \ layer \rightarrow non-linear\ \ layer \rightarrow linear\ \ layer \rightarrow softmax \rightarrow class\ \ probabilities$$

The 2 hidden layer NN architecture achieved an accuracy of 98.0% on the test set as shown in Table 1. This is a further improvement when compared to Q1b as can be seen in Figure 10 and 11 shown below.

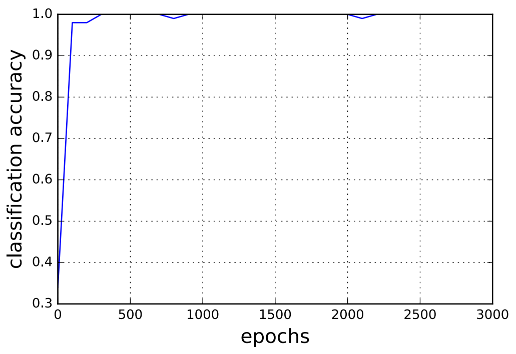

Figure 10: Two Hidden Layer NN classification accuracy on training set without dropout.

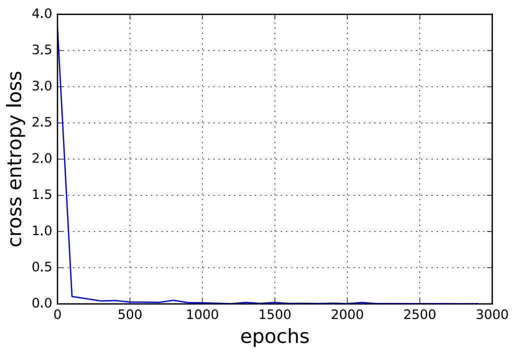

Figure 11: Two Hidden Layer NN cross entropy loss on training set without dropout.

The confusion matrix for the two hidden layer NN can be seen in Figure 12 below.

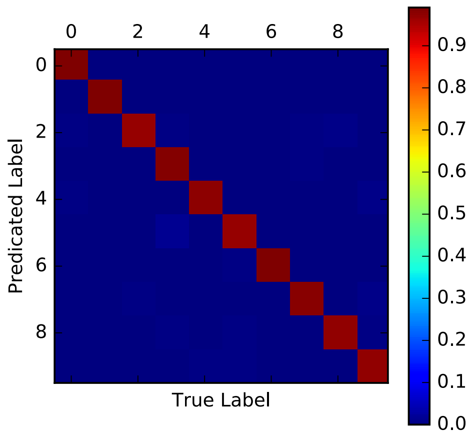

Figure 12: Confusion matrix on MNIST test set for P1c without dropout.

The code used to complete this task can be found in *P1c.py*. The saved weights for this model can be found in *model_training_weights\P1b*. The optimal learning rate was found to be $\lambda_{optimal} = 0.45$. Due to the fact that is model was overfitting as can be seen in Figure 10, dropout was added. The resultant figures with the NN with dropout can be seen on the following page.

P1: [(c)] continued on next page...

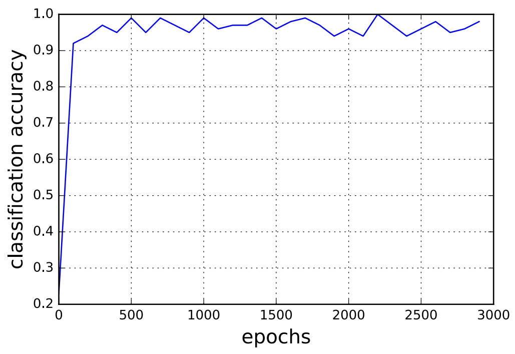

Figure 13: Two Hidden Layer NN classification accuracy on training set with dropout.

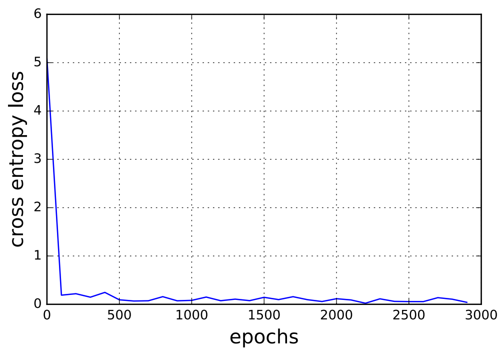

Figure 14: Two Hidden Layer NN cross entropy loss on training set with dropout.

The confusion matrix for the two hidden layer NN with dropout can be seen on the following page in Figure 15 below.

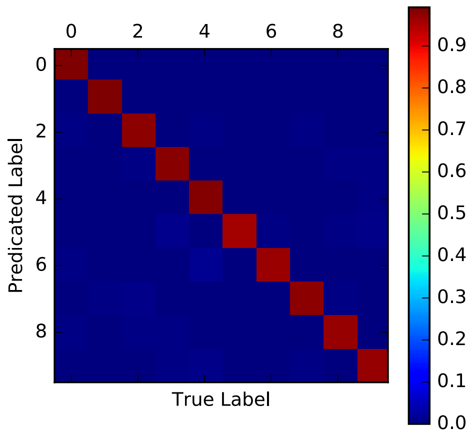

Figure 15: Confusion matrix on MNIST test set for P1c with dropout.

The resultant model with dropout does not overfit the training data as can be seen in Figure 13 and achieves a test accuracy of 97.84%.

P1: continued on next page...

(d)

Following the favourable results shown in Q1a-Q1c a most sophisticated classification model was created that had the following structure.

$$input(28 \times 28) \rightarrow conv(3 \times 3 \times 16) + maxpool(2 \times 2) \rightarrow conv(3 \times 3 \times 16) + maxpool(2 \times 2) \rightarrow flatten \rightarrow$$

$$non-linear\ \ layer \rightarrow linear\ \ layer \rightarrow softmax \rightarrow class\ \ probabilities$$

The resultant training visualization for this NN can be seen in Figure 16 and 17 shown below.

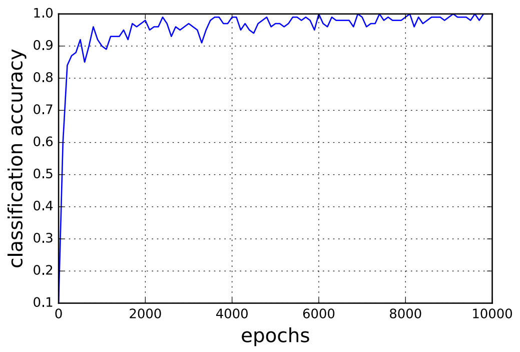

Figure 16: Convolutional NN classification accuracy on training set without dropout.

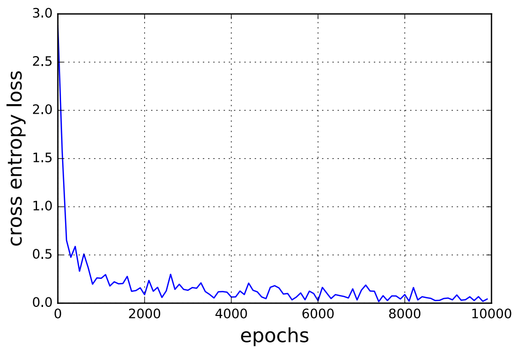

Figure 17: Convolutional NN cross entropy loss on training set without dropout.

The confusion matrix for the Convolutional NN can be seen on the following page in Figure 18.

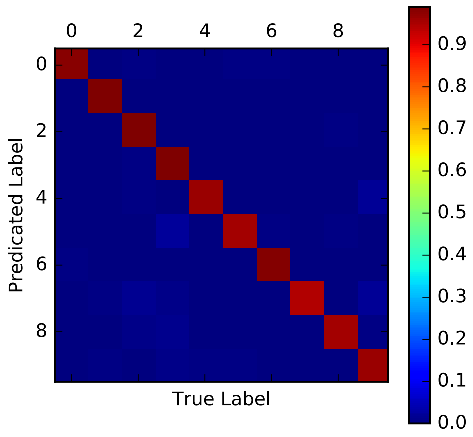

Figure 18: Confusion matrix on MNIST test set for P1d without dropout.

Although the NN achieved great performance dropout was implemented in order to compare the performance as can be seen in Figure 19 - 21 shown on the following page.

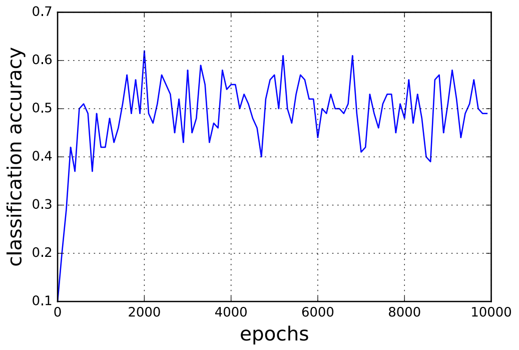

Figure 19: Convolutional NN classification accuracy on training set with dropout.

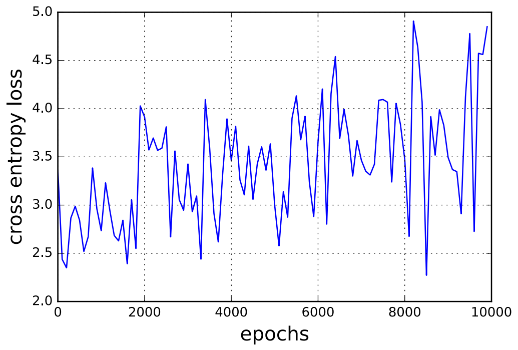

Figure 20: Convolutional NN cross entropy loss on training set with dropout.

It can be seen that even though with dropout the training accuracy seems low (49%) in Figure 19, the performance of the NN on the test set was 97.7%.

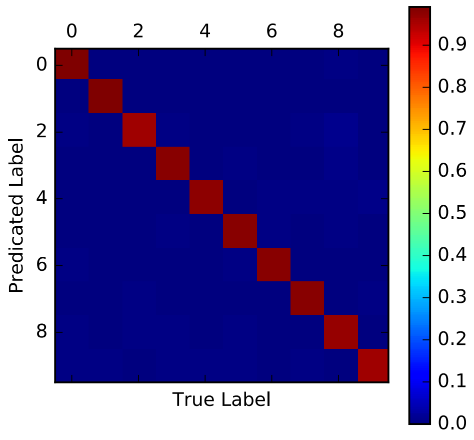

Figure 21: Confusion matrix on MNIST test set for P1d with dropout.

It can be seen from Table 1 that the Convolutional Model achieved an accuracy of 97.33%. This performance can be attributed to convolutional layers which confirms the workings of [2] and lecture notes [3],[4].

P2:

(a)

(i) The derivation of the loss function (Equation 1) wrt. to the scores z was computed as follows:

We define the softmax function:

$$y_i = \frac{e^{z_i}}{\sum_{c=1}^{nclass} e^{z_c}} \tag{2}$$

The resultant derivation can then be done as follows:

$$\frac{\partial l}{\partial z_i} = -\sum_{j=1}^{nclasses} \frac{\partial}{\partial z_i}\big(p_j log(y_j)\big) = -\sum_{j=1}^{nclasses} p_j \frac{\partial}{\partial z_i}\big(log(y_j)\big)$$

Separating the equation:

$$\frac{\partial l}{\partial z_i} = -\sum_{j=1}^{nclasses} p_j \frac{1}{y_i}\frac{\partial y_i}{\partial z_i} \tag{3}$$

$$\frac{\partial y_i}{\partial z_i} = \frac{\partial}{\partial z_i}\left(\frac{e^{z_i}}{\sum_{c=1}^{nclass} e^{z_c}}\right) = y_i(1-y_i)$$

Substituting this result into Equation 3.

$$\frac{\partial y_i}{\partial z_i} = -\frac{p_i}{y_i}y_i(1-y_i) - \sum_{j\neq i}^{10} \frac{p_j}{y_j} y_i(-y_i y_j)$$

$$\frac{\partial l}{\partial z_i} = -p_i + p_i y_i + \sum_{j\neq i}^{10} p_j y_i = -p_i + \sum_{j=1}^{10} p_j y_i = -p_i + y_i \sum_{j=1}^{10} p_j$$

$$\frac{\partial l}{\partial z_i} = y_i - p_i \tag{4}$$

Equation 4 can be written in vector form as:

$$\frac{\partial l}{\partial z} = Y - P \tag{5}$$

(ii) The derivative of the loss with respect to the layer's parameter W can be seen below:

We define the linear mapping of Z in terms of the weight vector W, X and the bias vector b.

$$Z = XW + b$$

$$Z_i = \sum_j W_{ji} X_j + b_i$$

The chain rule can be applied to Equation 1 defined earlier to form the following:

$$\frac{\partial l}{\partial W} = \frac{\partial l}{\partial Z}\frac{\partial Z}{\partial W}$$

P2: [(a)] continued on next page...

Equation 5 gives the result for one of the products in this equation.

$$\frac{\partial l}{\partial W} = (Y - P)\frac{\partial Z}{\partial W}$$

$$\frac{\partial Z_i}{\partial W_{ji}} = \frac{\partial}{\partial W_{ji}}\left(\sum_j W_{ji}X_j + b_i\right) = X_j$$

Therefore, Equation 6 can be seen below:

$$\frac{\partial l}{\partial W} = X^T(Y - P) \tag{6}$$

The derivative of the loss with respect to the layer's parameter b can be seen below:

$$\frac{\partial l}{\partial b} = \frac{\partial l}{\partial Z}\frac{\partial Z}{\partial b} = (Y - P)\frac{\partial Z}{\partial b}$$

$$\frac{\partial Z_i}{\partial b_i} = \frac{\partial}{\partial b_i}\left(\sum_j W_{ji}X_j + b_i\right) = 1$$

Therefore, Equation 7 can be formed:

$$\frac{\partial l}{\partial b} = (Y - P) \tag{7}$$

Based on the model P1:a the derivative of the loss wrt. to X was derived as follows:

$$\frac{\partial l}{\partial X} = \frac{\partial l}{\partial Z}\frac{\partial Z}{\partial X} = (Y - P)\frac{\partial Z}{\partial X}$$

$$\frac{\partial Z_i}{\partial X_j} = \frac{\partial}{\partial X_j}\left(\sum_j W_{ji}X_j + b_i\right) = W_{ji}$$

Therefore Equation 8 can be seen below:

$$\frac{\partial l}{\partial X} = W^T(Y - P) \tag{8}$$

P2: [(a)] continued on next page...

(iii) The derivative of the convolutional layer (C) wrt. to its parameters w and wrt. to its inputs (4-dim tensor) can be seen below. In a typical NN, we could define the input to a neuron as follows [3]:

$$z^l{}_j = \sum_i w^l{}_{ji} a^{l-1}{}_i + b^l{}_j$$

The activation in the layer l would be described as follows:

$$a^l{}_j = \sigma(z^l{}_j)$$

The symbol σ symbol is simply the activation function for the layer l. In this application the activation function would be a ReLU function. However, in Convolutional NN the notation is different for the input to a node as can be seen below [5],[4]:

$$z_{x,y}{}^{l+1} = w^{l+1} * \sigma(z_{x,y}{}^l) + b_{x,y}{}^{l+1}$$

This equation is the convolution operation during forward propagation of the neural network.

$$z_{x,y}{}^{l+1} = \sum_a \sum_b w_{a,b}{}^{l+1} * \sigma(z_{x-a,y-b}{}^l) + b_{x,y}{}^{l+1} \tag{9}$$

Using the result above and applying the same thinking in the calculating the derivative of the convolutional layer C in back propogration Equation 10 as shown below.

$$\frac{\partial C}{\partial z_{x,y}{}^l} = \sum_{x'} \sum_{y'} \frac{\partial C}{\partial z_{x',y'}{}^{l+1}} \frac{\partial z_{x',y'}{}^{l+1}}{\partial z_{x,y}{}^l} = \sum_{x'} \sum_{y'} \delta_{x',y'}{}^{l+1} \frac{\partial \left(\sum_a \sum_b w_{a,b}{}^{l+1} \sigma(z_{x'-a,y'-b}{}^l) + b_{x',y'}{}^{l+1} \right)}{\partial z_{x,y}{}^l} \tag{10}$$

Equation 10 shown above can then be adjusted by manipulating and substituting the following equations:

$$x' = x + a$$

$$y' = y + b$$

Therefore Equation 10 becomes Equation 11 shown below:

$$\frac{\partial C}{\partial z_{x,y}{}^l} = \sum_{x'} \sum_{y'} \delta_{x',y'}{}^{l+1} w_{a,b}{}^{l+1} * \sigma'(z_{x,y}{}^l) \tag{11}$$

Then using the following equations defined below, Equation 11 can be re-written as shown in Equation 12:

$$a = x' - x$$

$$b = y' - y$$

$$\frac{\partial C}{\partial z_{x,y}{}^l} = \sum_{x'} \sum_{y'} \delta_{x',y'}{}^{l+1} w_{x'-x,y'-y}{}^{l+1} * \sigma'(z_{x,y}{}^l) \tag{12}$$

Finally, the derivative of the convolutional layer wrt to its parameters can be written as shown in Equation 13 on the following page.

$$\frac{\partial C}{\partial {z_{x,y}}^{l}} = \delta^{l+1} * {w_{-x,-y}}^{l+1} \sigma'({z_{x,y}}^{l}) \tag{13}$$

P2: continued on next page. . .

(b)

The simple single layer NN model described in P1:a was implemented in *Python* without the use of the *TensorFlow* library.

This model achieved an accuracy of 92.23% as shown in Table 2 which is comparable to the NN in P1:a.

Experiment	**P2:b**	**P2:c**	**P2:d**
Training Error Rate	0.94	1.0	1.0
Test Error Rate	0.9223	0.9787	0.9743

Table 2: Error rates for P2.

This to be expected as it is the same NN with and without the *TensorFlow* library, however the underlying mathematical framework remains unchanged. The associated training errors and losses can be seen in the figures shown below.

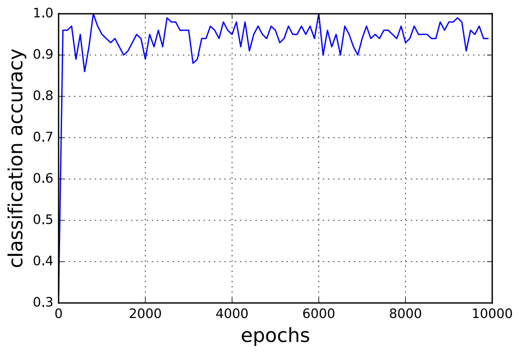

Figure 22: Accuracy accuracy on training set for P2b.

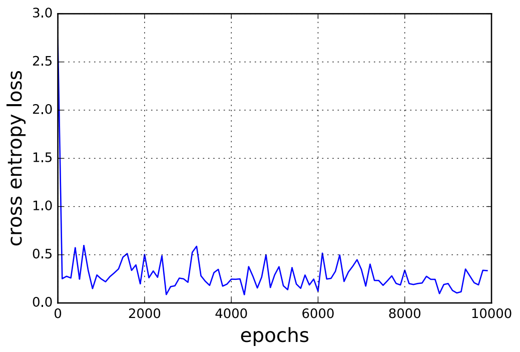

Figure 23: Cross entropy loss on training set for P2b.

The confusion matrix for P2:b can be seen in Figure 24 on the following page.

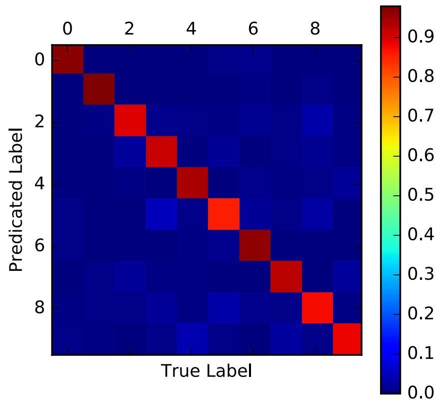

Figure 24: Confusion matrix on MNIST test set for P2b.

P2: [(b)] continued on next page...

The code used to complete this task can be found in *P2b.py*. The function files *misc_funcions.py* and *additional_functions.py* are described in the *README.txt* file attached to the assignment submission. The saved weights for this model can be found in *model_training_weights*\P2b.npz.

P2: continued on next page. . .

(c)

The model described in P1:b was implemented and trained as required. This model achieved an accuracy of 97.87% on the test set of the MNIST images. This result compared favourably to that presented in P1:b (Table 1) as shown in Table 2. The figures describing the results can be seen below.

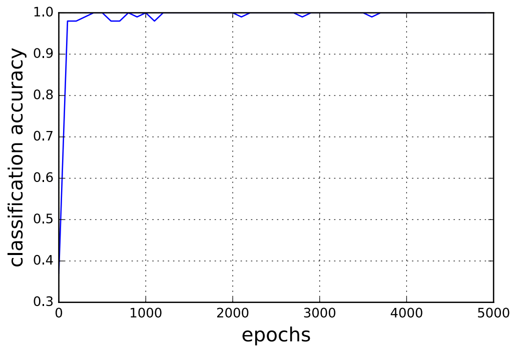

Figure 25: Accuracy accuracy on training set for P2c.

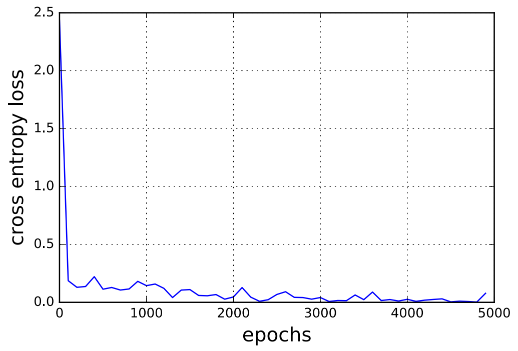

Figure 26: Cross entropy loss on training set for P2c.

The confusion matrix for P2:c can be seen in Figure 27.

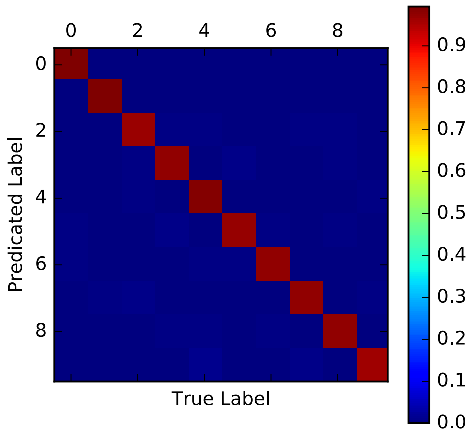

Figure 27: Confusion matrix on MNIST test set for P2b.

The code used to complete this task can be found in *P2c.py*. The saved weights for this model can be found in *model_training_weights\P2c.npz*.

(d)

The model described in P1:c was implemented and trained as required. This model achieved an accuracy of 97.43% on the test set of the MNIST images. This result compared favourably to that presented in P1:c (Table 1) as shown in Table 2.

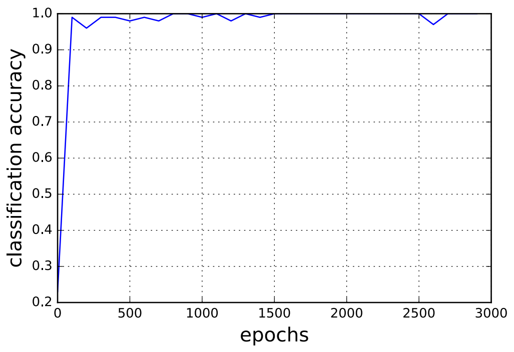

Figure 28: Accuracy accuracy on training set for P2d.

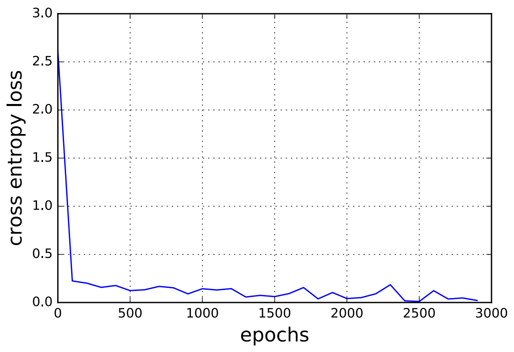

Figure 29: Cross entropy loss on training set for P2d.

The confusion matrix for P2:d can be seen in Figure 30.

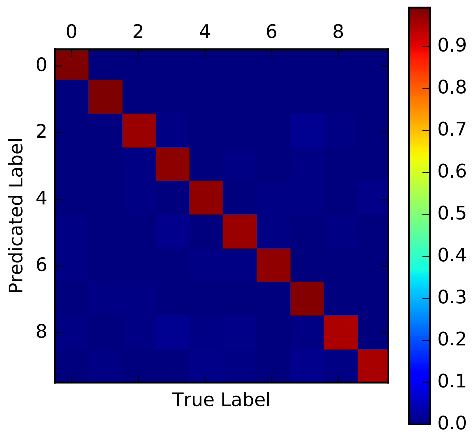

Figure 30: Confusion matrix on MNIST test set for P2d.

The code used to complete this task can be found in *P2d.py*. The saved weights for this model can be found in *model_training_weights\P2d.npz*.

(e)

The convolutional NN was implemented in Python without the *TensorFlow* library. Given the bottlenecking process of convolution as described [4], and not having the optimized packages available from TensorFlow these computation took far longer than P1:d to complete.

P2: [(e)] continued on next page...

It is to be expected that when the convolutional NN is implemented correctly in Python, the test accuracy and weights of the model should be similar as seen in P1:d.

It can be seen when comparing the results in Table 1 and Table 2, they are similar. Thereby confirming the implementations using *Python* with *TensorFlow* and without *TensorFlow* the image classification task is still possible.

References

[1] TensorFlow, *TensorBoard: Visualizing Learning*, 2016 (accessed January 15, 2017).

[2] TensorFlow, *Deep MNIST for Experts*, 2016 (accessed January 13, 2017).

[3] S. Osindero, "Neural network: Foundations," January 2017.

[4] K. Simonyan, "Image recognition with convolutional networks," February 2017.

[5] G. Gwardys, *Convolutional Neural Networks backpropagation: from intuition to derivation*, 2016 (accessed January 10, 2017).