COMPGI13

Advanced Topics in Machine Learning

Assignment 1

Nitish Mutha

UCABMUT

Git repo: https://github.com/compgi13/assignment-1-tensorflow-nn-models-NitishMutha

Note: All the trained weights have been stored in **source folder/trained_weights.**

P1 MNIST with TensorFlow

a) 1 linear layer, followed by a softmax

Following neural network was created to train MNIST digits with Tensforflow.

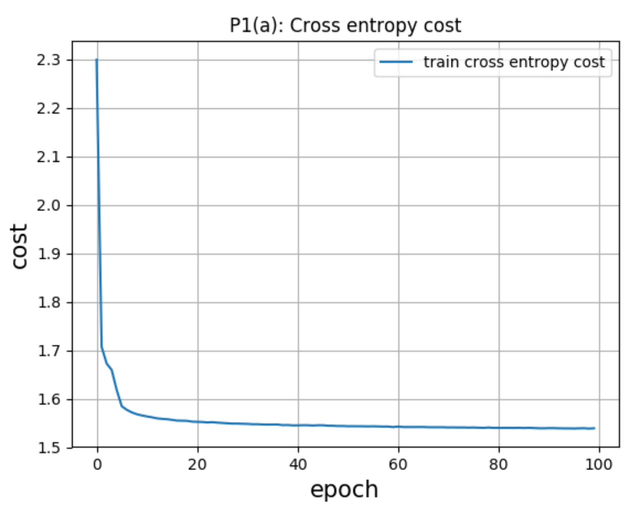
input → linear layer → softmax → class probabilities

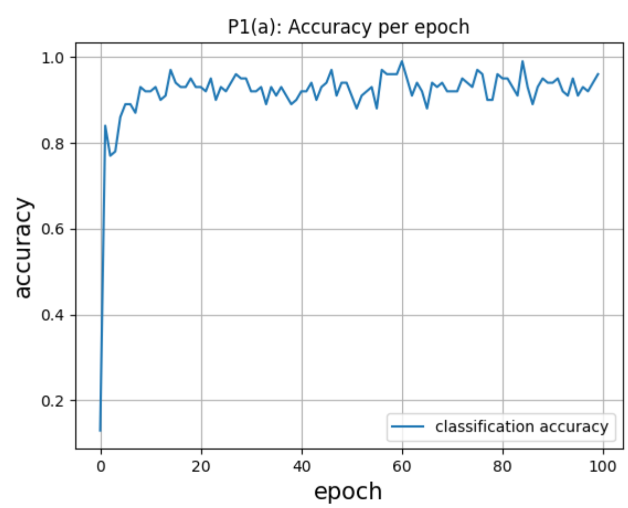
Training accuracy = 96.6%
Test accuracy = 92.9%

Optimal Learning rate: 0.6
EPOCH = 20000
BATCH_SIZE = 100

Code: tf_P1_a.py

Accuracy and Cross entropy plots:





Figures: The plots are plotted with epoch scale of 1 unit = 200 epoch.

Confusion Matrix:

```
[[ 963    0    3    2    0    2    8    1    1    0]
 [   0 1110    4    3    0    2    4    2   10    0]
 [  12    1  931    8   19    4   10   13   29    5]
 [   3    0   25  926    0   20    3    9   16    8]
 [   1    2    4    1  924    0   11    2    5   32]
 [   9    3    4   30   11  775   12    9   33    6]
 [  12    3    4    2   10   14  911    1    1    0]
 [   3    6   27    5    9    1    0  950    6   21]
 [   4    4    5   19   10   20   10   13  887    2]
 [  11    5    3   11   29   12    0   15   12  911]]
```

b) 1 hidden layer (128 units) with a ReLU non-linearity, followed by a softmax

Following neural network was created to train MNIST digits with Tensforflow.

input → non - linear layer → linear layer → softmax → class probabilities

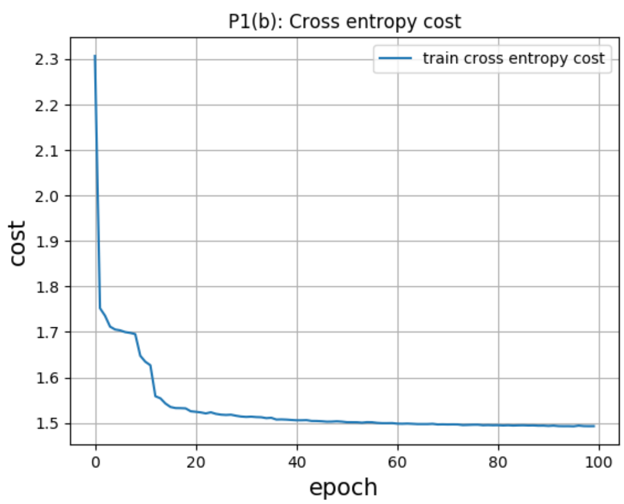
Training accuracy = 97.6%
Test accuracy = 97.11%

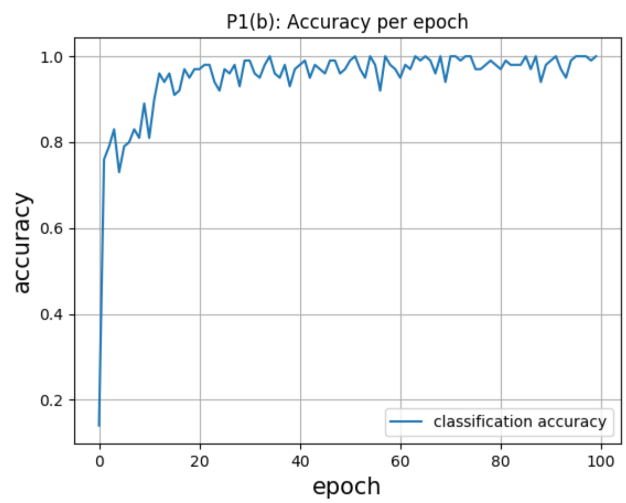
Optimal Learning rate: 0.5
EPOCH = 10000
BATCH_SIZE = 100

Code: tf_P1_b.py

Accuracy and Cross entropy plots

Figures: The plots are plotted with epoch scale of 1 unit = 100 epoch.





Confusion Matrix:

```
[[ 966    0    3    1    0    4    3    1    2    0]
 [   0 1119    3    2    0    1    5    2    3    0]
 [   5    2 1000    8    2    0    1    8    6    0]
 [   0    0    5  987    0    2    1    6    6    3]
 [   1    0    3    0  948    0    7    6    2   15]
 [   6    1    0   11    0  859    6    2    4    3]
 [   9    2    1    1    2    8  932    1    2    0]
 [   1    5   13    1    0    1    0 1001    2    4]
 [   3    0    5    9    3    7    6    5  932    4]
 [   6    5    2   10    8    2    1    8    0  967]]
```

c) 2 hidden layers (256 units) each, with ReLU non-linearity, follow by a softmax

Following neural network was created to train MNIST digits with Tensforflow.

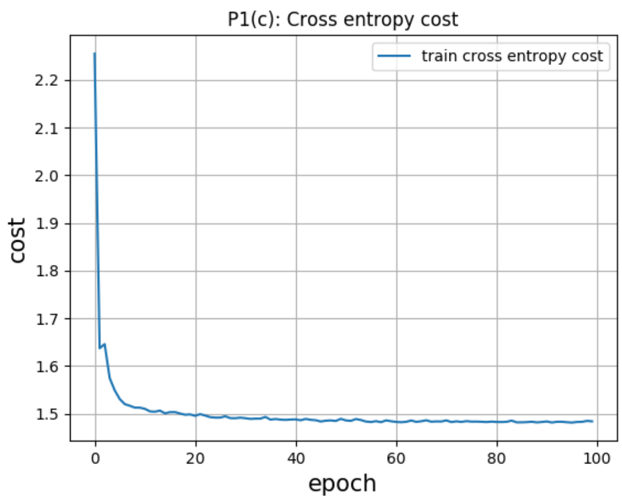
input → non-linear layer → non-linear layer → linear layer → softmax → class probabilities

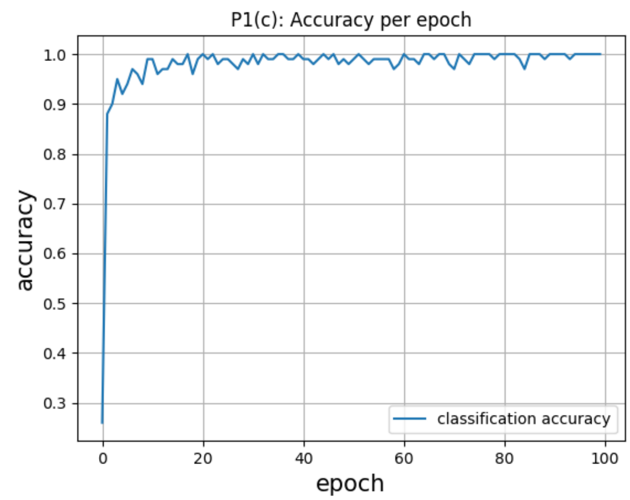
Training accuracy = 99.5%
Test accuracy = 97.8500%

Optimal Learning rate: 0.95
EPOCH = 10000
BATCH_SIZE = 100

Code: tf_P1_c.py

Accuracy and Cross entropy plots





Figures: The plots are plotted with epoch scale of 1 unit = 100 epoch

Confusion Matrix:

```
[[ 972    1    1    0    1    0    3    1    0    1]
 [    0 1122    2    2    0    2    2    1    4    0]
 [    4    4 1005    1    3    0    1    7    7    0]
 [    0    0    3  983    0   13    0    3    5    3]
 [    1    1    4    0  958    0    4    0    0   14]
 [    4    0    0    1    1  877    2    1    4    2]
 [    4    2    1    0    4    3  942    0    2    0]
 [    1    5    6    5    0    0    0 1004    5    2]
 [    2    0    3    3    3    3    4    2  953    1]
 [    4    2    0    4   12    4    2    4    8  969]]
```

d) 3-layer convolutional model (2 convolutional layers followed by max pooling) + 1 non-linear layer (256 units), followed by softmax.

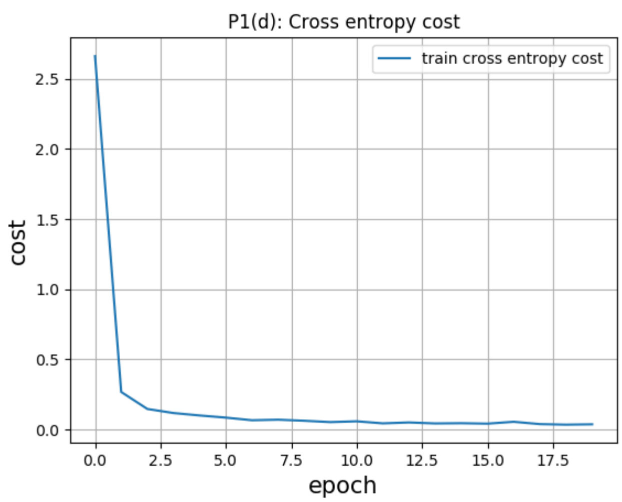
input(28x28) → conv(3x3x16) + maxpool(2x2) → conv(3x3x16) + maxpool(2x2) → atten → non- linear → linear layer → softmax → class probabilities

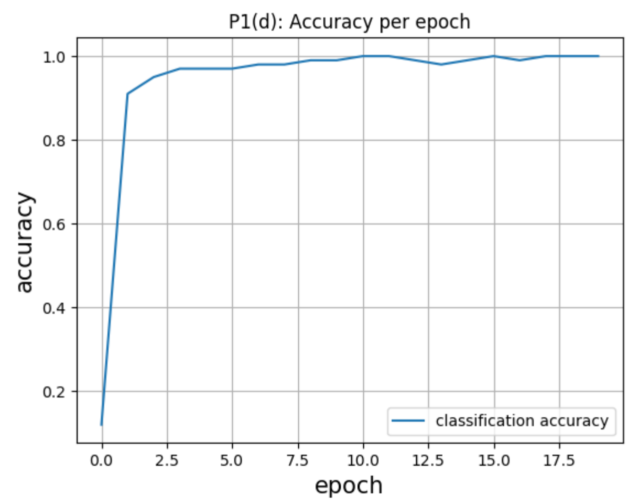
Training accuracy = 100%
Test accuracy = 98.54%

Optimal Learning rate: 0.001
EPOCH = 2000
BATCH_SIZE = 100

Code: tf_P1_d.py

Accuracy and Cross entropy plots





Figures: The plots are plotted with epoch scale of 1 unit = 100 epoch

Confusion Matrix:

```
[[ 974    1    1    0    0    1    0    2    1    0]
 [    0 1130    1    0    0    2    1    0    1    0]
 [    0    4 1016    3    1    0    0    7    1    0]
 [    0    0    0 1004    0    4    0    1    1    0]
 [    0    0    1    0  977    0    0    0    0    4]
 [    2    0    0    6    0  883    1    0    0    0]
 [    9    3    0    0    2    7  937    0    0    0]
 [    0    6    4    2    0    0    0 1014    1    1]
 [    3    0    6   10    1    3    1    3  945    2]
 [    0    3    0    4   15    6    0    4    3  974]]
```

P2 MNIST without TensorFlow

a) **Derivations**:

i. **Derivative of the loss function wrt. the scores z**:

Let softmax with j[th] element be

$$s_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

Taking derivative $\frac{\delta s_j}{\delta z_i}$ by splitting into two conditions: i = j and i ≠ j

For i = j:

$$\frac{\delta s_j}{\delta z_i} = \frac{e^{z_i} \sum_m e^{z_m} - e^{z_i} e^{z_i}}{(\sum_m e^{z_m})^2}$$

$$= \frac{e^{z_i}}{\sum_m e^{z_m}} - \left(\frac{e^{z_i}}{\sum_m e^{z_m}}\right)^2$$

$$= s_i - (s_i)^2$$

$$= s_i(1 - s_i)$$

For i ≠ j:

$$\frac{\delta s_j}{\delta z_i} = -\frac{e^{z_i} e^{z_j}}{(\sum_m e^{z_m})^2}$$

$$= -\frac{e^{z_i}}{\sum_m e^{z_m}}\left(\frac{e^{z_i}}{\sum_m e^{z_m}}\right)$$

$$= -s_j s_i$$

So $\frac{\delta L}{\delta z_i}$, where $L = -\sum_k y_k \log(s_k)$

$$\frac{\delta L}{\delta z_i} = -\sum_k y_k \frac{\delta log s_k}{\delta z}$$

$$= -\sum_k y_k \frac{1}{s_k} \frac{\delta s_k}{\delta z}$$

$$= -y_i(1 - s_i) + \sum_{k \neq i} y_k s_i$$

$$= -y_i - y_i s_i + s_i \sum_{k \neq i} y_k$$

$$= -y_i + s_i \sum_k y_k$$

As $\sum_k y_k = 1$

$$\frac{\delta L}{\delta z_i} = s_i - y_i$$

= $\boldsymbol{s}-\boldsymbol{y}$ (vector form)

ii. **Given the model in (P1:a), compute the derivative of the loss wrt input x, derivative of the loss with to the layer's parameters W, b.**

Consider the general form of neural network equation:

$$z_j^l = \sum_m w_{mj}^l\, a_m^{l-1} + b_j^l$$

$l: layer, a^{l-1}: activation\ output, m: nodes\ in\ layer, w_{ij}: weight\ from\ i\ to\ j\ layer$

Loss wrt to x:

From chain rule we can write:

$$\delta_i^j = \frac{\delta L}{\delta z_i^l} = \sum_j \frac{\delta L}{\delta z_j^{l+1}} \cdot \frac{\delta z_j^{l+1}}{\delta z_i}$$

From previous equation:

$$z_j^{l+1} = \sum_m w_j^{l+1} a_m^l + b_j^{l+1}$$

$$z_j^{l+1} = \sum_m w_j^{l+1} \sigma(z_m^l) + b_j^{l+1}$$

$$\frac{\delta z_j^{l+1}}{\delta z_i} = w_{ij}^{l+1} \sigma'(z_i^l)$$

Now $\frac{\delta L}{\delta z_i^l} = \sum_j \delta_j^{l+1} . w_{ij}^{l+1} \sigma'(z_i^l)$ can be written as $\delta^l = \delta^{l+1} \left(W^{l+1}\right)^T \cdot \sigma'(z_i^l)$

For simple neural network in P1 a, input layer with activation function $\sigma(x) = x$ and $\delta^L = \boldsymbol{s}-\boldsymbol{y}$

$$\delta^l = \frac{\delta L}{\delta \boldsymbol{X}} = (\boldsymbol{s}-\boldsymbol{y}\,)\boldsymbol{W}^T$$

Loss with respect to the weights:

$$\frac{\partial L}{\partial w_{ij}^l} = \frac{\partial L}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{ij}^l}$$

$$\frac{\partial L}{\partial w_{ij}^l} = \delta_j^l \frac{\partial z_j^l}{\partial w_{ij}^l}$$

Also:

$$\frac{\partial z_j^l}{\partial w_{ij}^l} = a_i^{l-1}$$

Hence, $$\frac{\partial L}{\partial w_{ij}^l} = \delta_j^l a_i^{l-1}$$

And $$\frac{\partial L}{\partial W} = \left(a^{l-1}\right)^T \delta^l = \boldsymbol{x}^T(\boldsymbol{s} - \boldsymbol{y})$$

Loss wrt bias:

$$\frac{\partial L}{\partial b_j^l} = \frac{\partial L}{\partial z_j^l} \frac{\partial z_j^l}{\partial_j^l}$$

since $\frac{\partial z_j^l}{\partial_j^l} = 1$

we can write: $$\frac{\partial L}{\partial b_j^l} = \delta^l = \mathbf{s} - \mathbf{y}$$

iii. **Compute the derivative of a convolution layer wrt. to its parameters W and wrt. to its input (4-dim tensor).**

Let's consider input to the neuron as

$$z_j^l = \sum_i w_{ji}^l a_i^{l-1} + b_j^l$$

Activation σ in layer l as

$$a_j^l = \sigma(z_j^l)$$

For convolution we rewrite the input equation as

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

With the previously derived results we can write following for convolution C layer

$$\frac{\partial C}{\partial z_{x,y}^l} = \sum_{x'}\sum_{y'} \frac{\partial C}{\partial z_{x',y'}^{l+1}} \frac{\partial z_{x',y'}^{l+1}}{\partial z_{x,y}^l}$$

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

We can substitute

$$x' = x + a$$
$$y' = y + b$$

Hence:

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

Using following we can write above as

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

Hence convolution wet its parameters

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

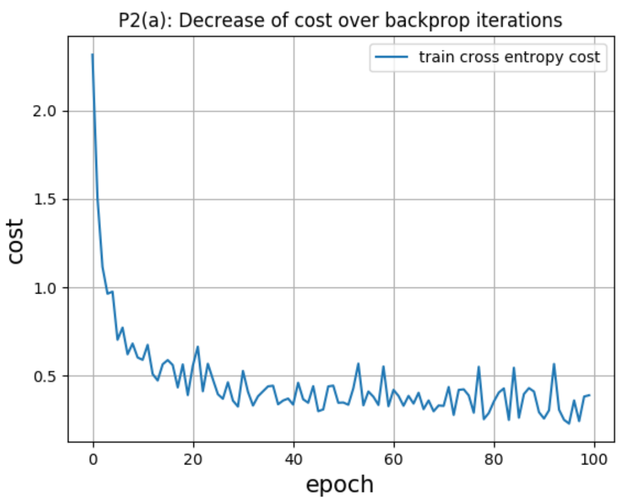
b) Implement and train the model in (P1:a)

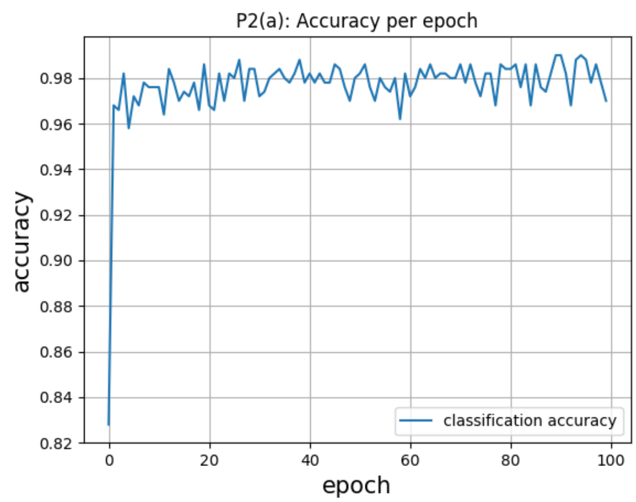
Training accuracy = 98.2%
Test accuracy = 98.19%

Optimal Learning rate: 0.01
EPOCH = 10000
BATCH_SIZE = 100

Code: tf_P2_a.py

Accuracy and Cross entropy plots





Figures: The plots are plotted with epoch scale of 1 unit = 100 epoch.

Confusion Matrix:

```
[[ 957    0    2    2    1    4   10    1    3    0]
 [   0 1106    2    3    1    2    4    1   16    0]
 [  11    9  897   15   15    1   16   18   41    9]
 [   5    2   19  907    1   27    3   14   21   11]
 [   1    4    6    1  910    1   10    1    8   40]
 [  11    5    5   44   12  742   17   10   37    9]
 [  15    3    5    2   14   15  900    1    3    0]
 [   3   19   24    6   10    0    0  926    3   37]
 [   9    9    8   27    8   19   12   14  854   14]
 [  11    8    5   12   41   10    0   20    6  896]]
```

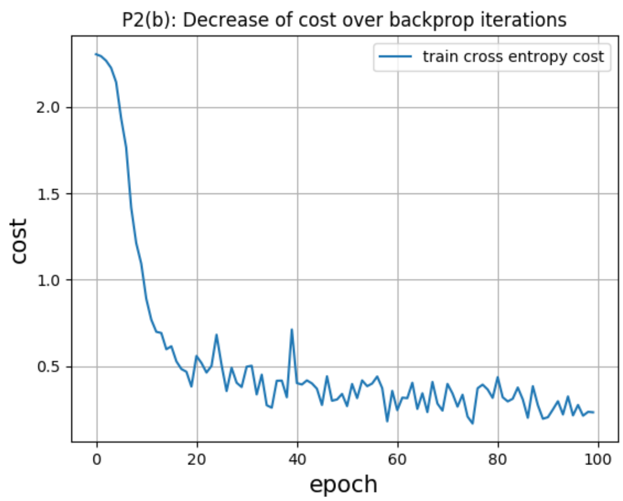
c) Implement and train the model in (P1:b)

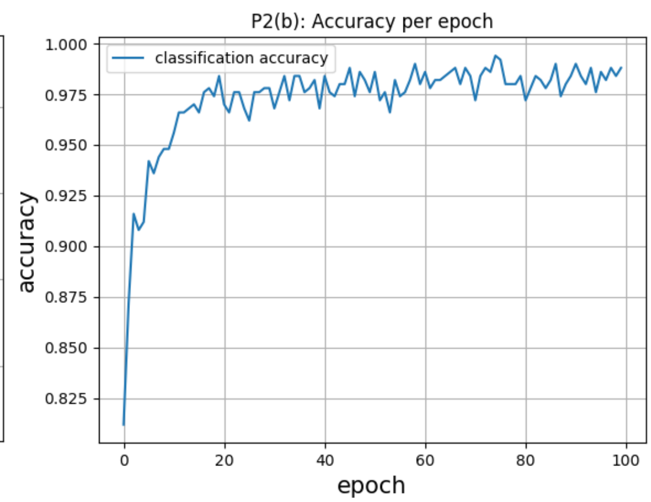
Training accuracy = 99.6%
Test accuracy = 98.53%

Optimal Learning rate: 0.01
EPOCH = 10000
BATCH_SIZE = 100
HIDDEN_LAYER_1 = 128

Code: tf_P2_b.py

Accuracy and Cross entropy plots





Figures: The plots are plotted with epoch scale of 1 unit = 100 epoch.

Confusion Matrix:

```
[[ 957    0    4    1    0    3   10    2    3    0]
 [    0 1110    2    2    0    1    4    2   14    0]
 [   10    7  922   16   13    0   11   14   35    4]
 [    2    1   21  924    0   22    1   15   17    7]
 [    1    1    3    1  916    0   12    2    8   38]
 [    8    4    3   35    8  775   17    8   27    7]
 [   12    3    4    1   14   11  909    1    3    0]
 [    3    8   24    8    6    0    0  957    2   20]
 [    9    6    5   16   10   18   13   10  877   10]
 [   10    7    2   10   34    8    0   16    4  918]]
```

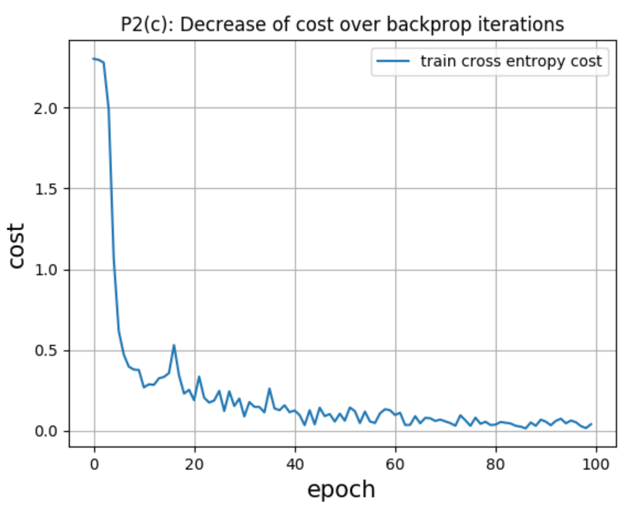
d) Implement and train the model in (P1:c)

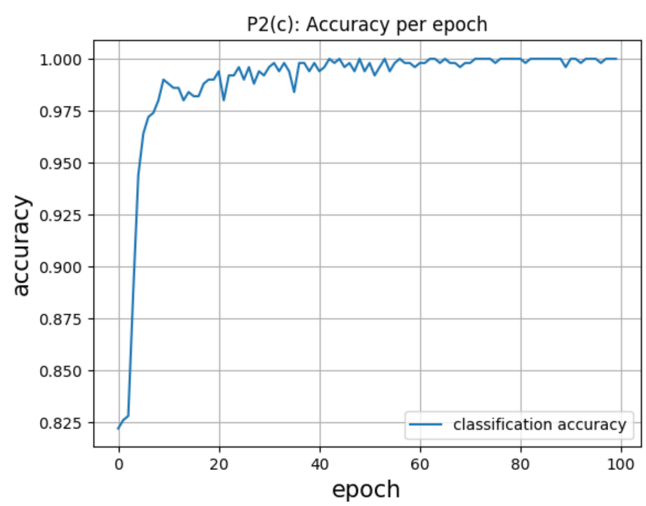
Training accuracy = 99.96%
Test accuracy = 99.51%

Optimal Learning rate: 0.07
EPOCH = 10000
BATCH_SIZE = 100
HIDDEN_LAYER_1 = 256
HIDDEN_LAYER_2 = 256

Code: tf_P2_c.py

Accuracy and Cross entropy plots





Figures: The plots are plotted with epoch scale of 1 unit = 100 epoch.

Confusion Matrix:

```
[[ 970    0    0    3    1    1    2    1    2    0]
 [    0 1123    3    3    0    1    2    1    2    0]
 [    4    1 1007    5    3    0    3    6    3    0]
 [    1    0    3  996    0    1    0    5    3    1]
 [    3    0    3    0  951    0    4    5    0   16]
 [    3    0    0   15    1  862    6    0    3    2]
 [    5    3    1    1    3    6  938    0    1    0]
 [    1    4    7    5    0    0    0 1003    1    7]
 [    5    1    3   16    3    6    3    4  931    2]
 [    2    3    0    9    7    5    1    7    0  975]]
```

e) Implement and train the model in (P1:d)

Table of comparison of the training and test error rates (1 - accuracy).

Experiment	P1:a	P1:b	P1:c	P1:d	P2:b	P2:c	P2:d	P2:e
Training error rate	3.4	2.4	0.5	0	1.8	0.4	0.04	
Testing error rate	7.1	2.89	2.15	1.46	1.81	1.47	0.49	

References:

[1] www.tensorflow.org

[2] Stanford University http://cs231n.github.io/