ECE421S – Introduction to Machine Learning

Assignment 1

Linear and Logistic Regression

New Deadline: February 6, 2019 @ BA3014, 4:00-5:00 PM EST

Code Submission: ece421ta2019@gmail.com **February 6, 2019 @ 5:00 PM EST**

General Notes:

- Attach this cover page to your hard copy submission
- For assignment related questions, please contact Matthew Wong (matthewck.wong@mail.utoronto.ca)
- For general questions regarding Python or Tensorflow, please contact Tianrui Xiao (tianrui.xiao@mail.utoronto.ca) or see him in person in his office hours, Tuesdays, 4:00-6:00 PM in BA-3128 (Robotics Lab)

Please circle section to which you would like the assignment returned

Tutorial Sections

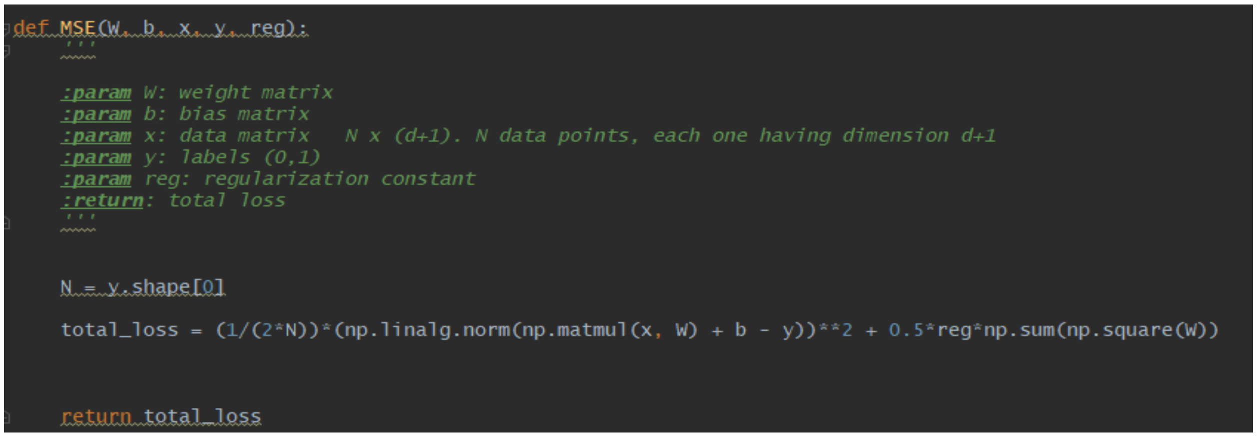
001	002	003	[✓ 004]
005	006	007	Graduate

Group Members		
Names	StudentID	Contribution %
Andy Linzi Zhou	**1002296730**	**55**
Ryan Do	**1001254117**	**45**

1. Linear Regression

1.1 Loss Function and Gradient

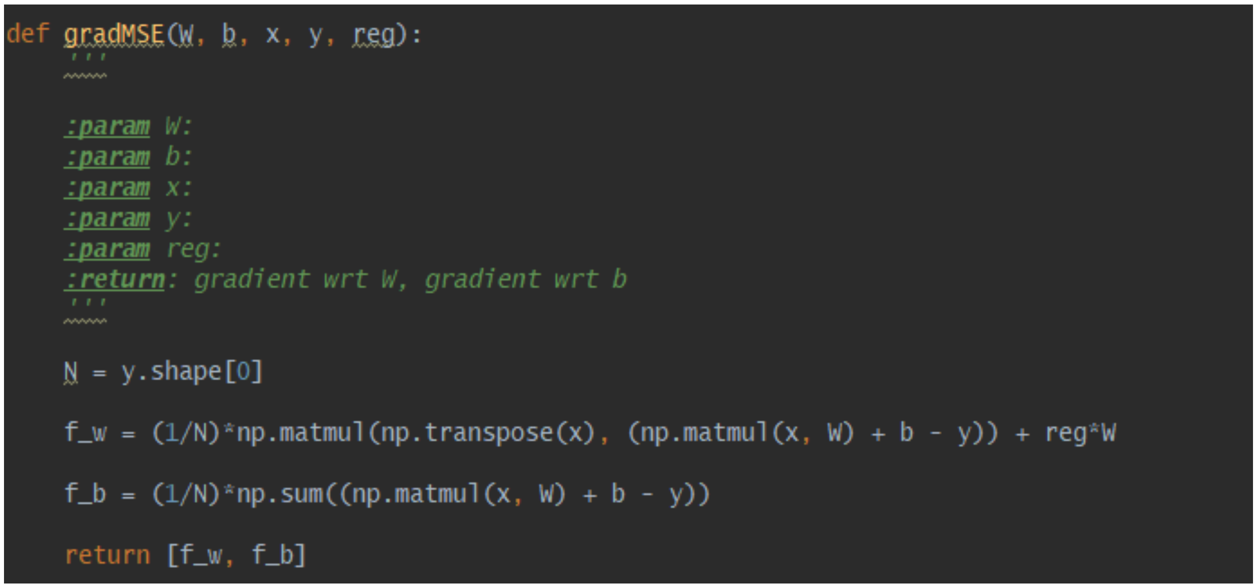
$MSE_{loss} = \frac{1}{2N}||xW + b - y||_2^2 + \frac{\lambda}{2}||W||_2^2$ ***MSE Loss Equation, Returns total loss***



```
def MSE(W, b, x, y, reg):
    '''

    :param W: weight matrix
    :param b: bias matrix
    :param x: data matrix    N x (d+1). N data points, each one having dimension d+1
    :param y: labels (0,1)
    :param reg: regularization constant
    :return: total loss
    '''

    N = y.shape[0]

    total_loss = (1/(2*N))*(np.linalg.norm(np.matmul(x, W) + b - y))**2 + 0.5*reg*np.sum(np.square(W))


    return total_loss
```

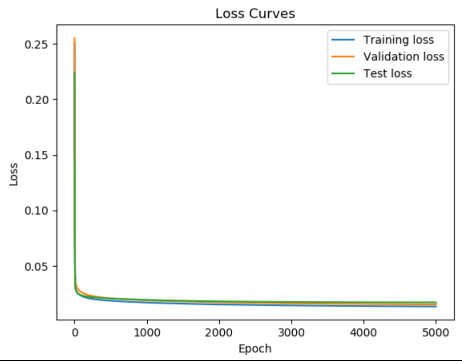
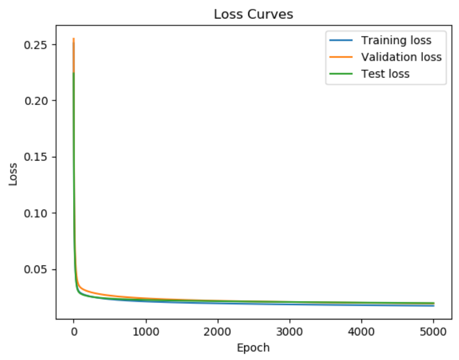
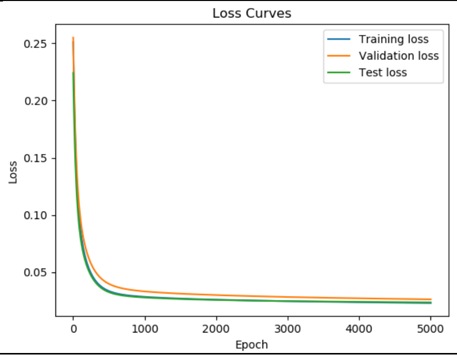
$\nabla f(W) = \frac{1}{N}x^T(xW + b \; - \; y) + \; \lambda W$ ***Gradient with respect to weights***

$\nabla f(b) = \frac{1}{N}\sum_{i=1}^{N}(xW + b - y)_i$ ***Gradient with respect to bias scalar***



```
def gradMSE(W, b, x, y, reg):
    '''

    :param W:
    :param b:
    :param x:
    :param y:
    :param reg:
    :return: gradient wrt W, gradient wrt b
    '''

    N = y.shape[0]

    f_w = (1/N)*np.matmul(np.transpose(x), (np.matmul(x, W) + b - y)) + reg*W

    f_b = (1/N)*np.sum((np.matmul(x, W) + b - y))

    return [f_w, f_b]
```

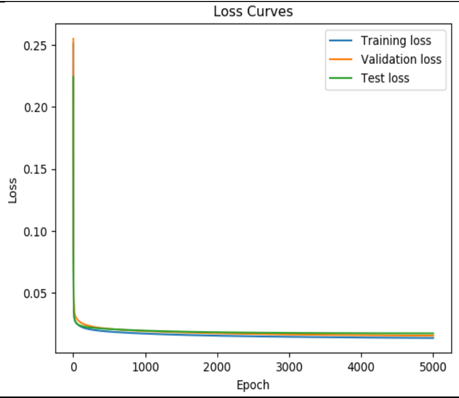
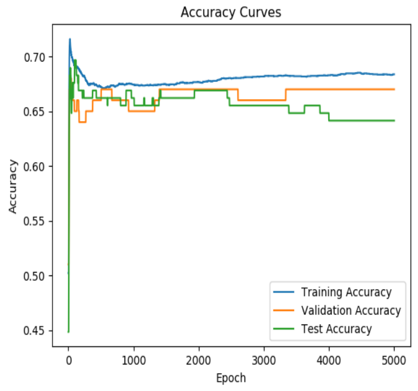
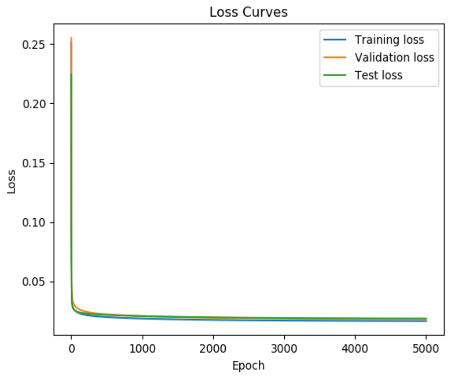
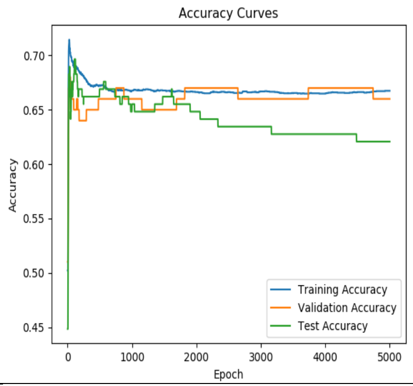
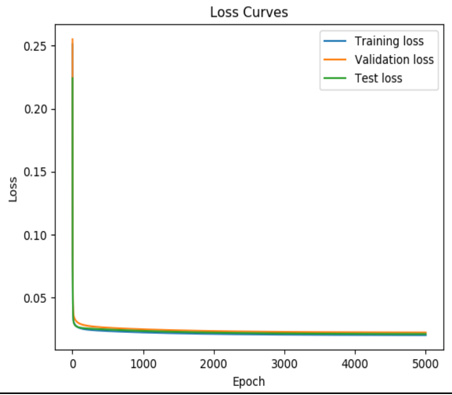
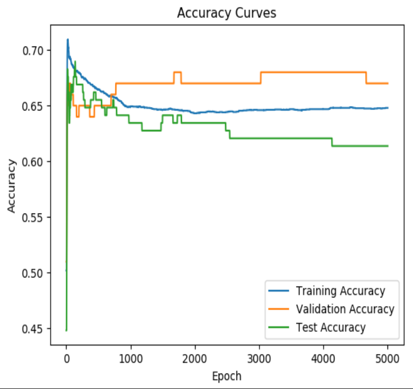
1.3 Tuning the learning rate

Learning Rate α	**Loss Curves**	**Training Time**	**Final Classification Accuracy**
0.005		00:00:36	Training Accuracy: 0.68371 Validation Accuracy: 0.67000 Test Accuracy: 0.64138
0.001		00:00:32	Training Accuracy: 0.67429 Validation Accuracy: 0.65000 Test Accuracy: 0.66207
0.0001		00:00:33	Training Accuracy: 0.69086 Validation Accuracy: 0.65000 Test Accuracy: 0.68966

The effect of α on the training time isn’t very pronounced. It takes roughly the same time to train since the number of epochs were the same for all three scenarios. The change of α only impacts the rate of convergence towards an optimal solution, not the time it takes to compute the gradients for descent.

The effect of α on the final classification accuracy can be observed. It seems that decreasing α leads to a greater final classification accuracy on the training set. On the other hand, a larger α leads to faster progress but lower accuracy. The lower accuracy is due to oscillations around the optimal point due to a too large step size to get closer to optimality.

1.4 Generalization

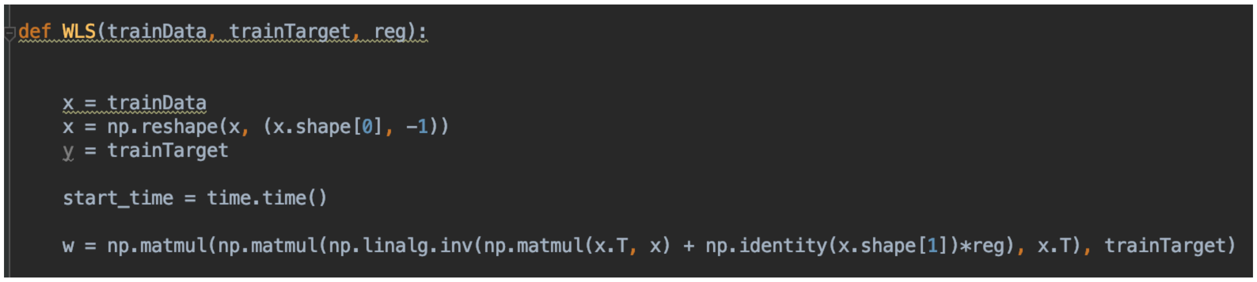
λ	**Loss curves**	**Accuracy Curves**	**Final Classification Accuracy**
0.001			Training Accuracy: 0.68371 Validation Accuracy: 0.67000 Test Accuracy: 0.64138
0.1			Training Accuracy: 0.66743 Validation Accuracy: 0.66000 Test Accuracy: 0.62069
0.5			Training Accuracy: 0.64800 Validation Accuracy: 0.67000 Test Accuracy: 0.61379

As regularization increases, the performance of the model on the training and testing set declines. The purpose of regularization is to induce generalization of the model to the validation set. Tuning λ to higher values allows the model to generalize and have good performance on the validation set despite worse performance on the training and testing sets. Tuning λ allows the model to have good generalization abilities even if training performance is not optimal.

1.5 Comparing Batch GD with Normal Equation

<u>Analytical solution</u>

$$w = (x^T x + \lambda I)^{-1} x^T y$$



```
def WLS(trainData, trainTarget, reg):


    x = trainData
    x = np.reshape(x, (x.shape[0], -1))
    y = trainTarget

    start_time = time.time()

    w = np.matmul(np.matmul(np.linalg.inv(np.matmul(x.T, x) + np.identity(x.shape[1])*reg), x.T), trainTarget)
```

	Batch GD	**Normal Equation**
Training MSE loss	0.01366	0.011582
Accuracy	0.68371	0.733143
Computation Time	00:00:36	00:00:01

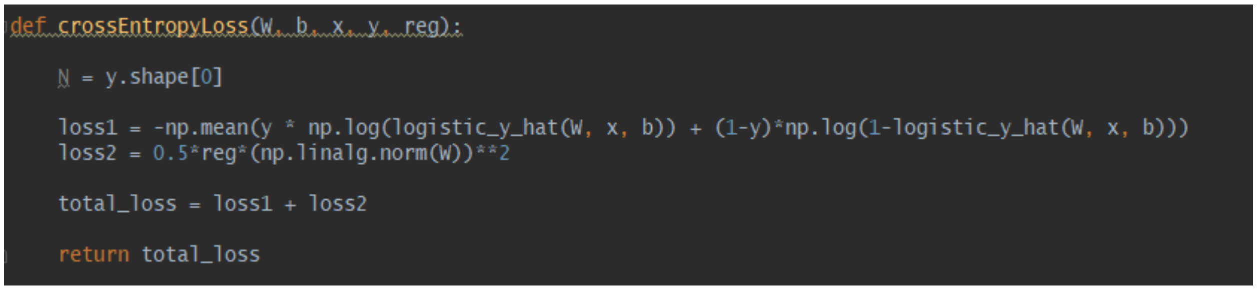
Part 2: Logistic Regression

2.1 Loss Function and Gradient

$\sigma(z) = \frac{1}{1+e^{-z}}$ ***sigmoid function***

$\hat{y}(x) = \sigma(xW + b)$ ***logistic y_hat***

$f(x, y, W, \lambda) = -\frac{1}{N}\left(y \log(\hat{y}(x)) + (1-y)\log(1-\hat{y}(x))\right) + \frac{\lambda}{2}||W||_2^2$ ***Cross Entropy loss***

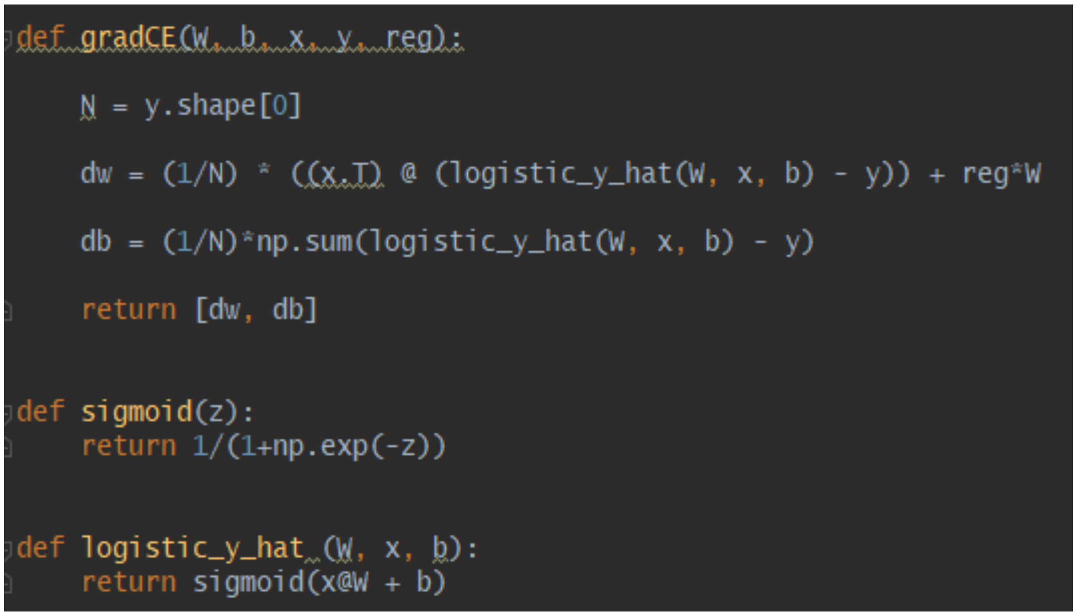


```
def crossEntropyLoss(W, b, x, y, reg):

    N = y.shape[0]

    loss1 = -np.mean(y * np.log(logistic_y_hat(W, x, b)) + (1-y)*np.log(1-logistic_y_hat(W, x, b)))
    loss2 = 0.5*reg*(np.linalg.norm(W))**2

    total_loss = loss1 + loss2

    return total_loss
```

$\nabla f(W) = \frac{1}{N}\left(x^T(\hat{y}(x) - y)\right) + \lambda W$ ***Gradient with respect to weights***

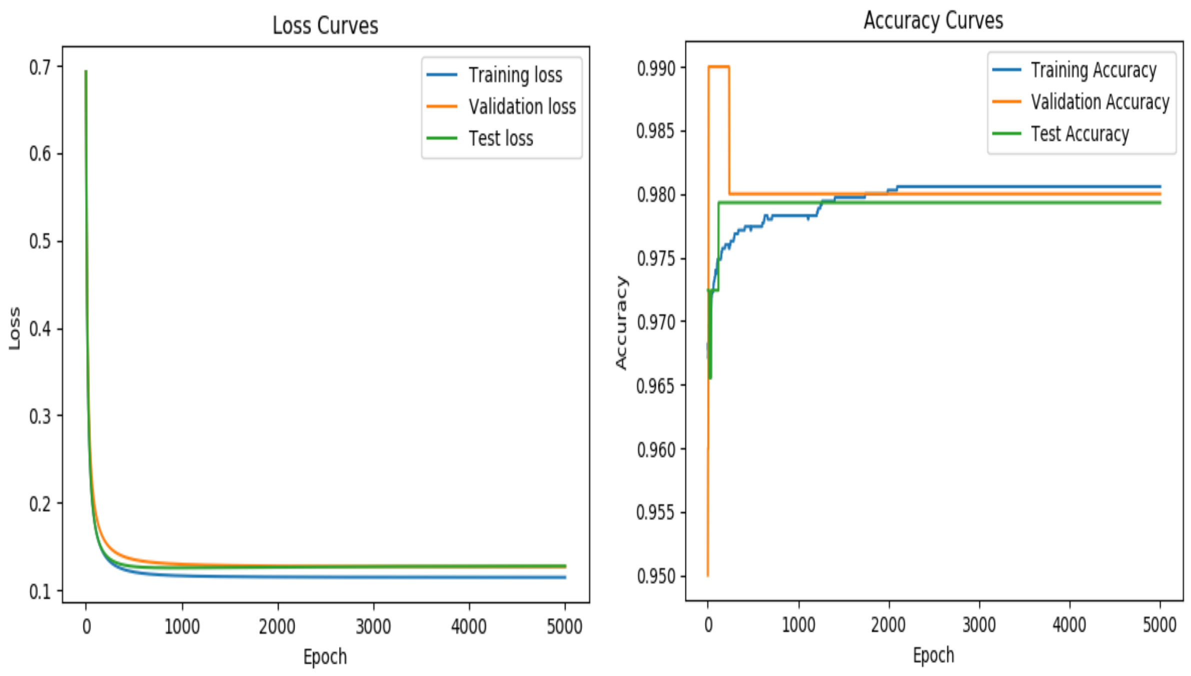
$\nabla f(b) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}(x) - y)_i$ ***Gradient with respect to bias scalar***

Derivation: https://github.com/Exquisition/ECE421-Projects/blob/master/a1/bceloss_gradient_derivation.jpg



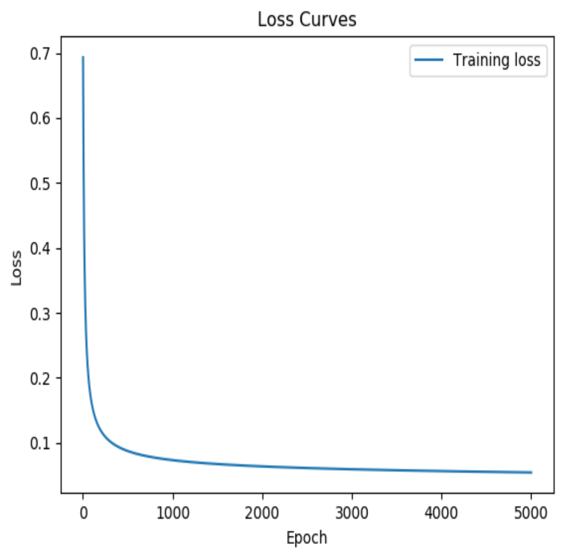
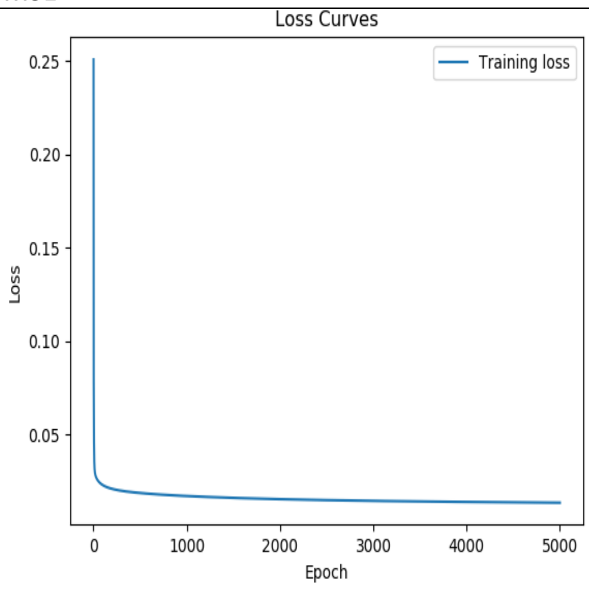
```
def gradCE(W, b, x, y, reg):

    N = y.shape[0]

    dw = (1/N) * ((x.T) @ (logistic_y_hat(W, x, b) - y)) + reg*W

    db = (1/N)*np.sum(logistic_y_hat(W, x, b) - y)

    return [dw, db]


def sigmoid(z):
    return 1/(1+np.exp(-z))


def logistic_y_hat (W, x, b):
    return sigmoid(x@W + b)
```

2.2 Learning



Loss and accuracy curves for logistic regression and cross entropy loss. Lambda = 0.1, 5000 epochs.

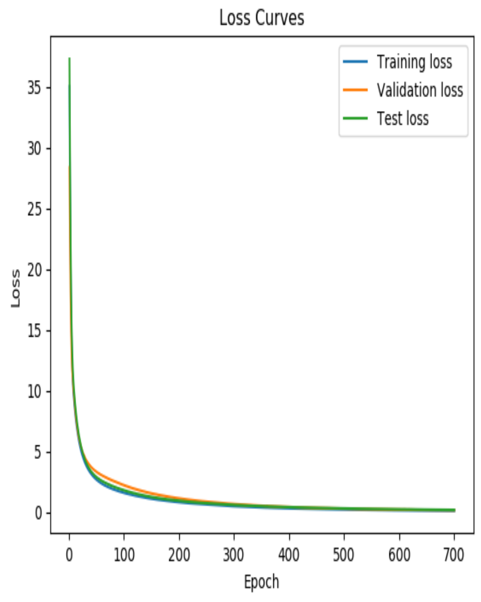
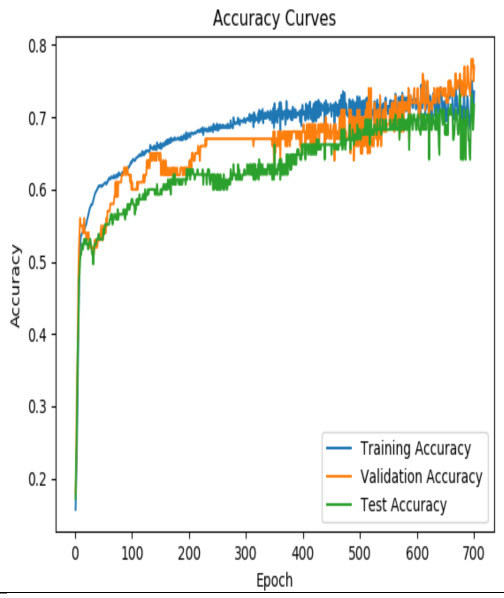
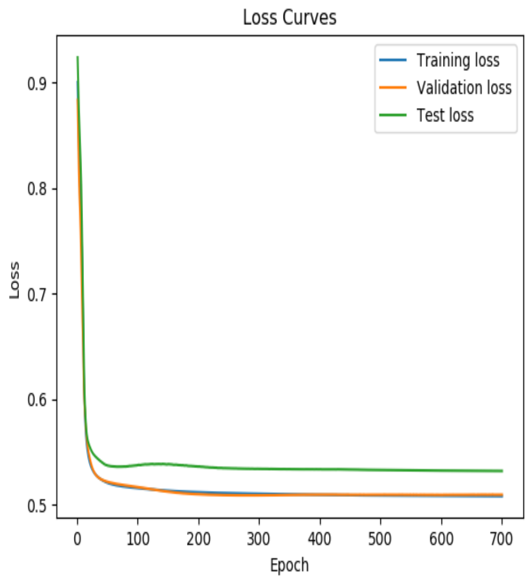
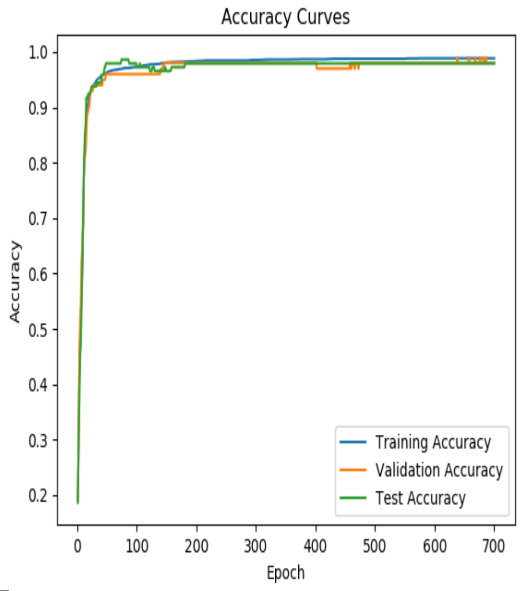
2.3 Comparison to Linear Regression

Cross Entropy	**MSE**
	

The convergence of cross entropy loss is less abrupt than the MSE loss. Also, cross entropy loss converges slower than MSE loss. The convergence behavior of cross entropy is smoother and resembles a continuous, differentiable function that has its derivative approaching zero.

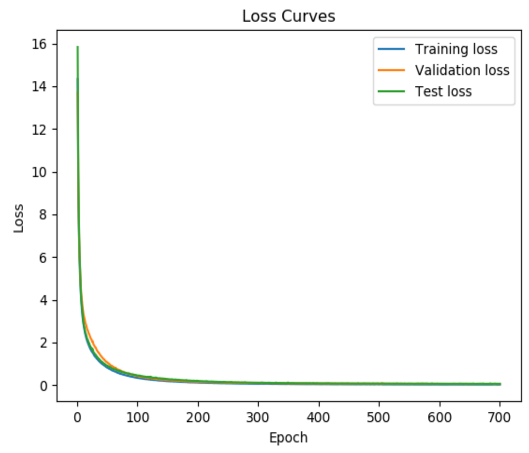
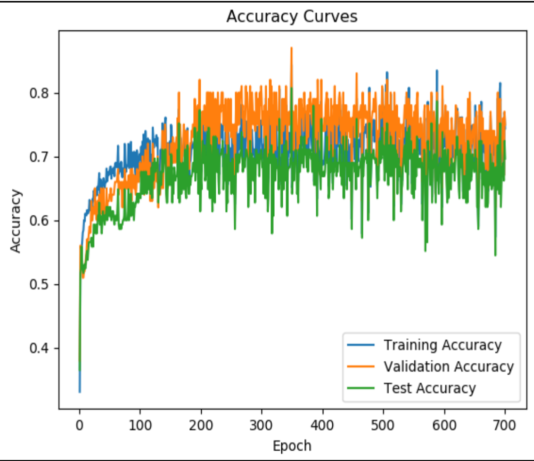
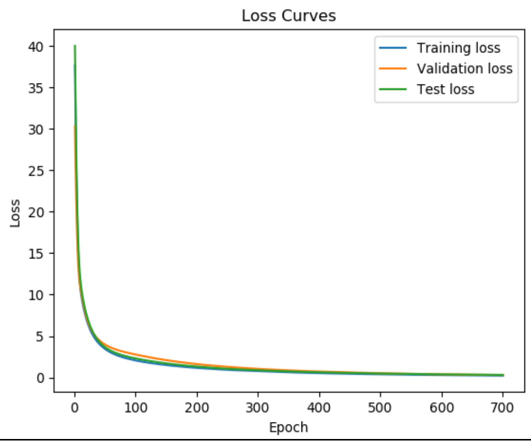
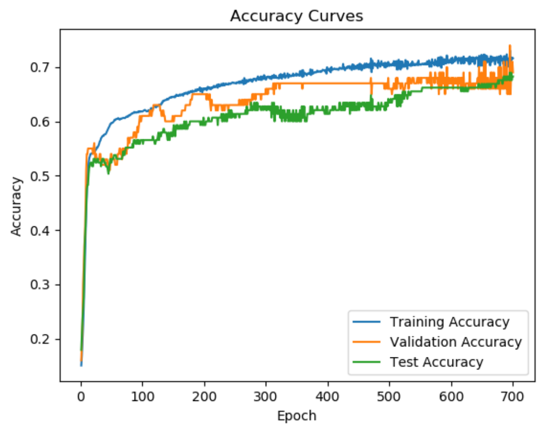
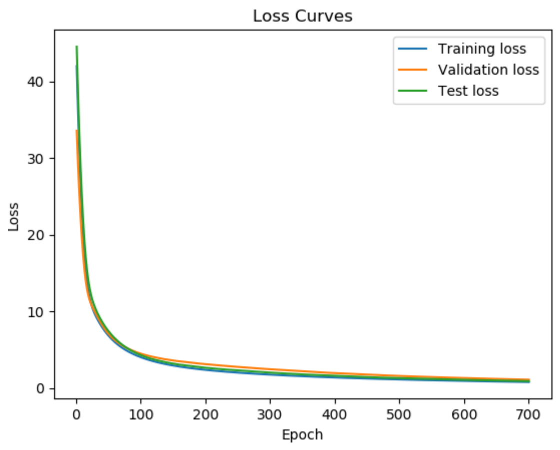
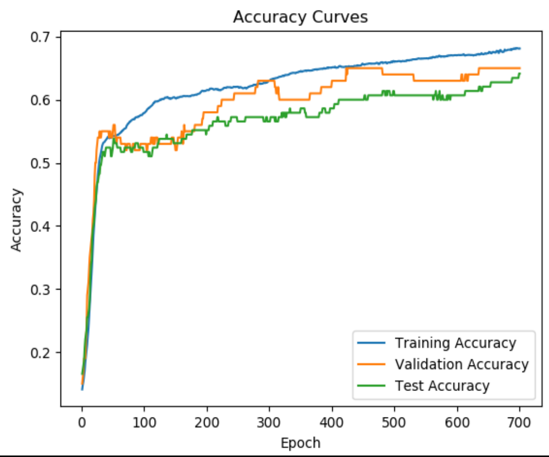
Batch Gradient Descent VS SGD & Adam

3.1.2 Implementing SGD Algorithm

	Loss Curves	**Accuracy Curves**
MSE Loss		
Binary Cross Entropy Loss		

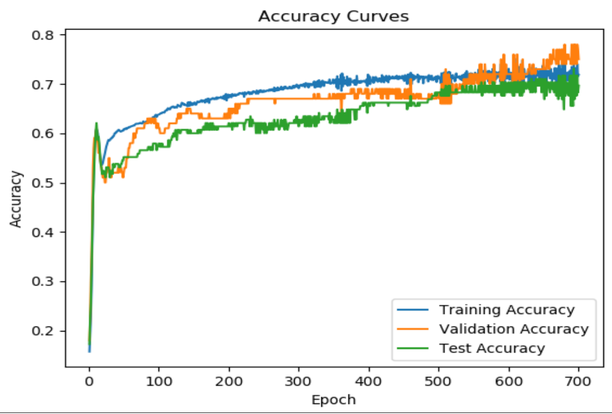
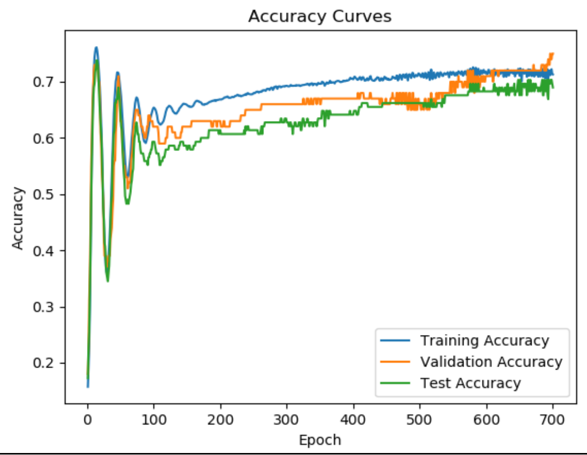
Lambda = 0, alpha = 0.001, minibatch = 500, 700 epochs.

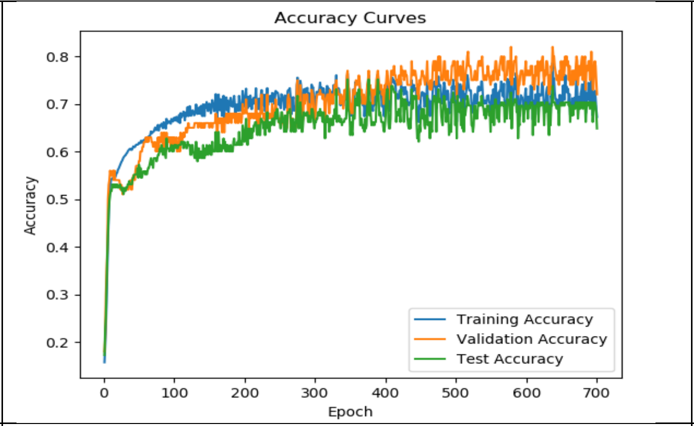
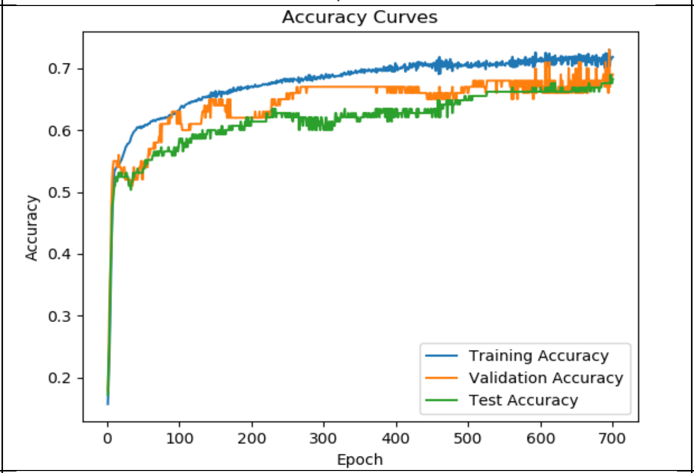
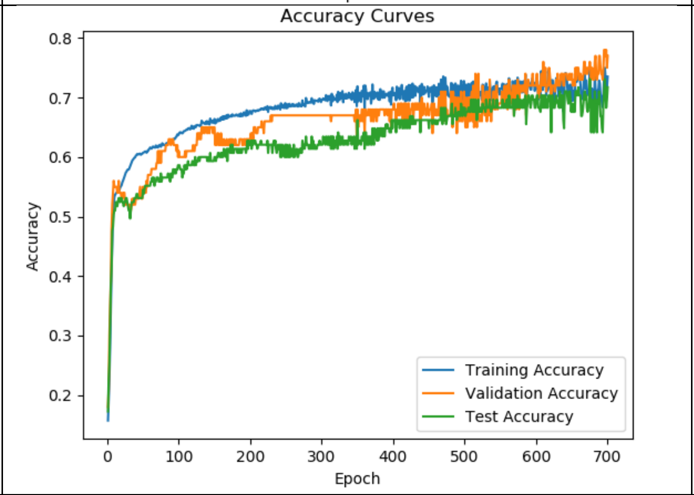
3.1.3 Batch Size Investigation

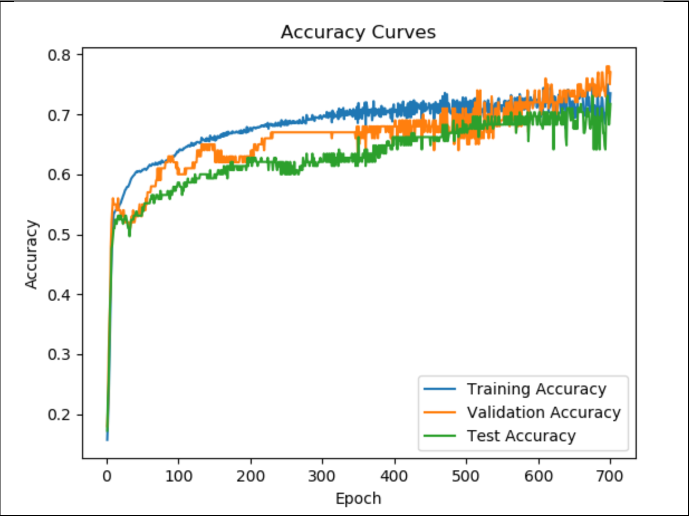
Minimizing MSE Loss		
Batch Size	**Loss Curves**	**Accuracy Curves**
100		
700		
1750		

As batch size increases, the final accuracies decrease and the losses tend to increase. The reason for this is that with larger batch sizes, the number of training steps per epoch decrease thus slower progression towards optimality. There is more variance/noise for lower batch sizes which slows down training for each step (since it's less direct), but more than makes up for it with the additional batches to be trained on per epoch.

3.1.4 Hyperparameter Investigation

Minimizing MSE Loss		
Hyperparameter	**Accuracy Curves**	**Final Accuracies**
β1 = 0.95		Training Accuracy: 0.71857 Validation Accuracy: 0.75000 Test Accuracy: 0.69655
β1 = 0.99		Training Accuracy: 0.71286 Validation Accuracy: 0.75000 Test Accuracy: 0.68966

β2 = 0.99		Training Accuracy: 0.67257 Validation Accuracy: 0.72000 Test Accuracy: 0.64828
β2 = 0.9999		Training Accuracy: 0.71829 Validation Accuracy: 0.69000 Test Accuracy: 0.68276
ϵ = 1e-9		Training Accuracy: 0.73514 Validation Accuracy: 0.77000 Test Accuracy: 0.71724

ϵ = 1e-4		Training Accuracy: 0.73514 Validation Accuracy: 0.77000 Test Accuracy: 0.71724

For each, what is the hyperparameter impact on the final training, validation and test accuracy? Why is this happening?

From the official tensorflow documentation (https://www.tensorflow.org/api_docs/python/tf/train/AdamOptimizer):

> The update rule for `variable` with gradient `g` uses an optimization described at the end of section2 of the paper:

$$t := t + 1$$

$$lr_t := extlearning\_rate * \sqrt{1 - beta_2^t}/(1 - beta_1^t)$$

$$m_t := beta_1 * m_{t-1} + (1 - beta_1) * g$$

$$v_t := beta_2 * v_{t-1} + (1 - beta_2) * g * g$$

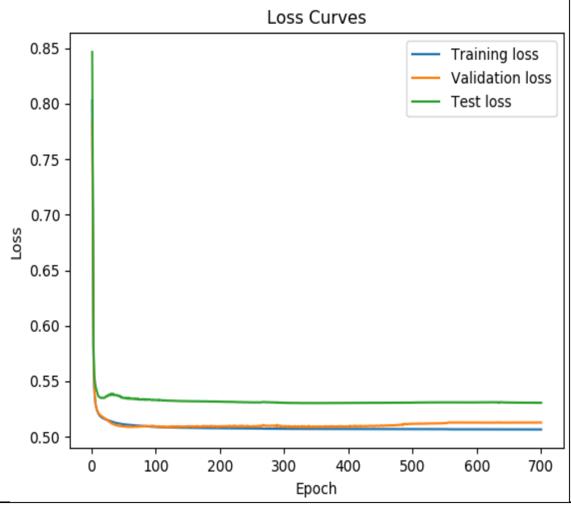
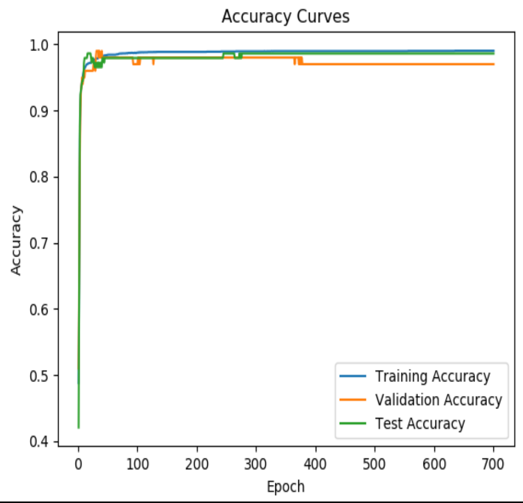
$$variable := variable - lr_t * m_t/(\sqrt{v_t} + \epsilon)$$

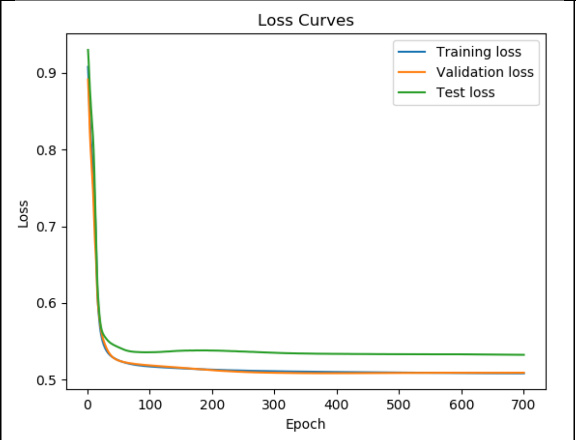
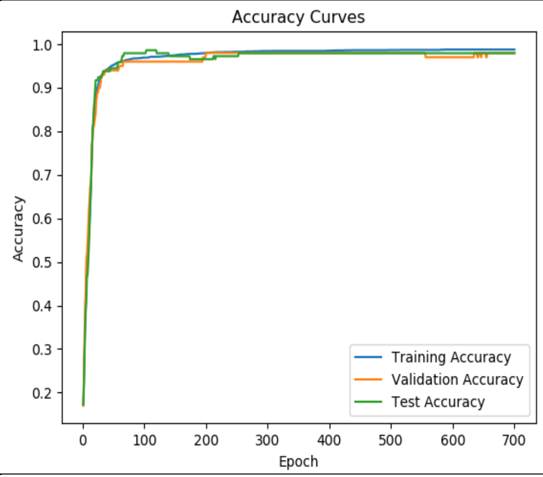
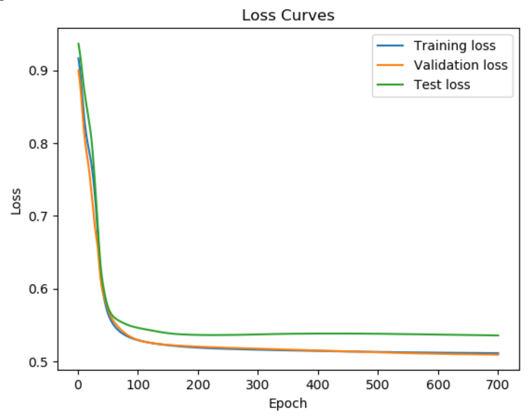
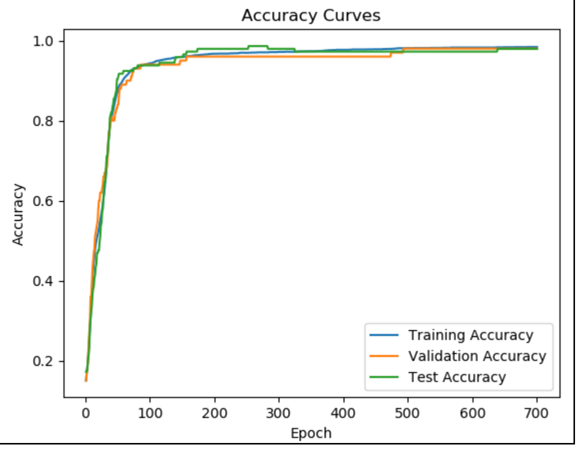
As shown in the plots, a higher beta1 value slightly increases time to convergence and introduces larger amplitude oscillations as the weights stabilize. As a momentum-based method, ADAM's beta1 represents the decay rate for first moment estimates (i.e the momentum). If simple momentum-based methods can be thought of a ball rolling down a slope, ADAM can be thought as a ball rolling down a slope with friction. With higher beta1, the momentum has a greater influence (i.e. higher "mass") and the gradient has less influence therefore oscillations are more pronounced thus increasing time to convergence, decreasing final accuracies.

Similarly, beta2 represents the decay rate for second moment estimates. As seen in the plots, a higher beta2 value decreases time to convergence (greater final accuracy) and decreases the oscillatory behaviour. This is opposite behaviour to beta1 changes. This is because the learning step is inversely dependent on the second order moment (influenced by beta2) but directly dependent on the first order moment (beta1). As a result, as beta2 increases, the gradient has more influence on learning steps and v (see above) has less influence. The result is decreased oscillations from valley-like movement and decreased time to convergence (greater final accuracy).

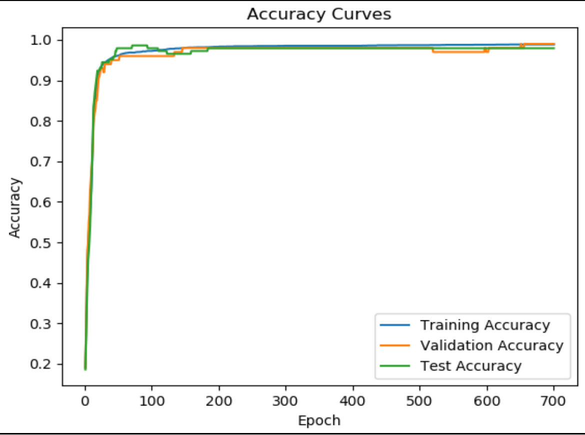
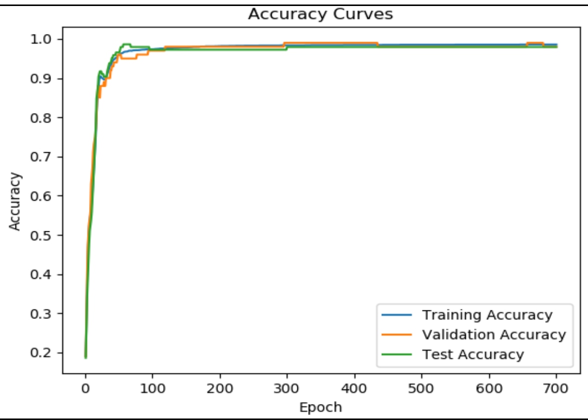
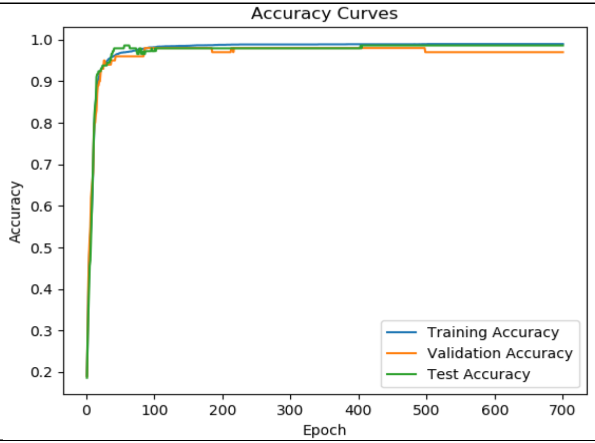
Epsilon had no effect on final accuracy. Epsilon is a very small value added to the second moment estimate which is likely usually negligible. It simply exists to prevent division from zero. As a result, an increase from 1e-9 to 1e-4 made no significant difference to final accuracy.

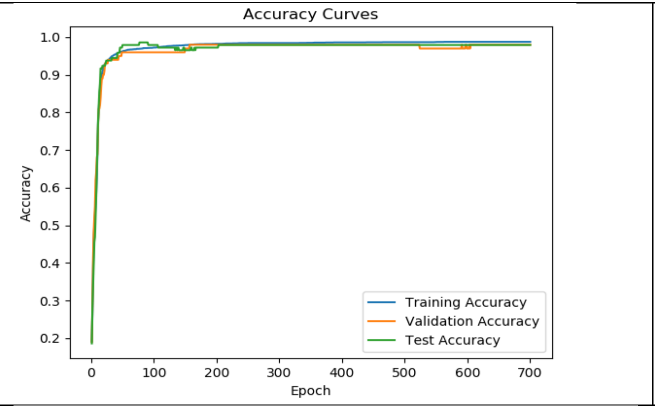
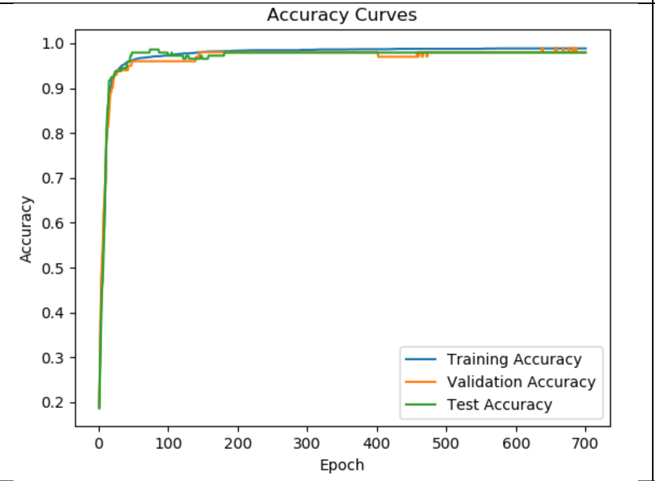
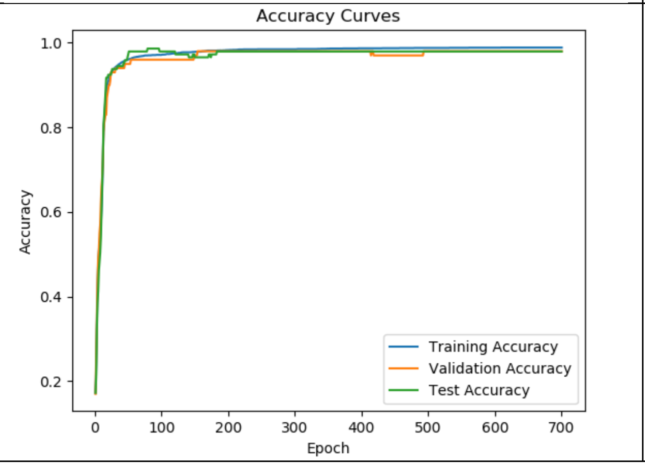
3.1.5 Cross Entropy Loss Investigation

Minimizing Binary Cross Entropy Loss		
Batch Size	**Loss Curves**	**Accuracy Curves**
100		

700		
1750		

Similarly to MSE error observations, higher batch size leads to longer time to convergence and higher final loss due to reduced number of batches per epoch thus less training steps taken. There is more variance/noise for lower batch sizes which slows down training for each step (since it's less direct), but more than makes up for it with the additional batches to be trained on per epoch

Minimizing Binary Cross Entropy Loss		
Hyperparameter	**Accuracy Curves**	**Final Accuracies**
β1 = 0.95		Training Accuracy: 0.98857 Validation Accuracy: 0.99000 Test Accuracy: 0.97931
β1 = 0.99		Training Accuracy: 0.98571 Validation Accuracy: 0.98000 Test Accuracy: 0.97931
β2 = 0.99		Training Accuracy: 0.98971 Validation Accuracy: 0.97000 Test Accuracy: 0.98621

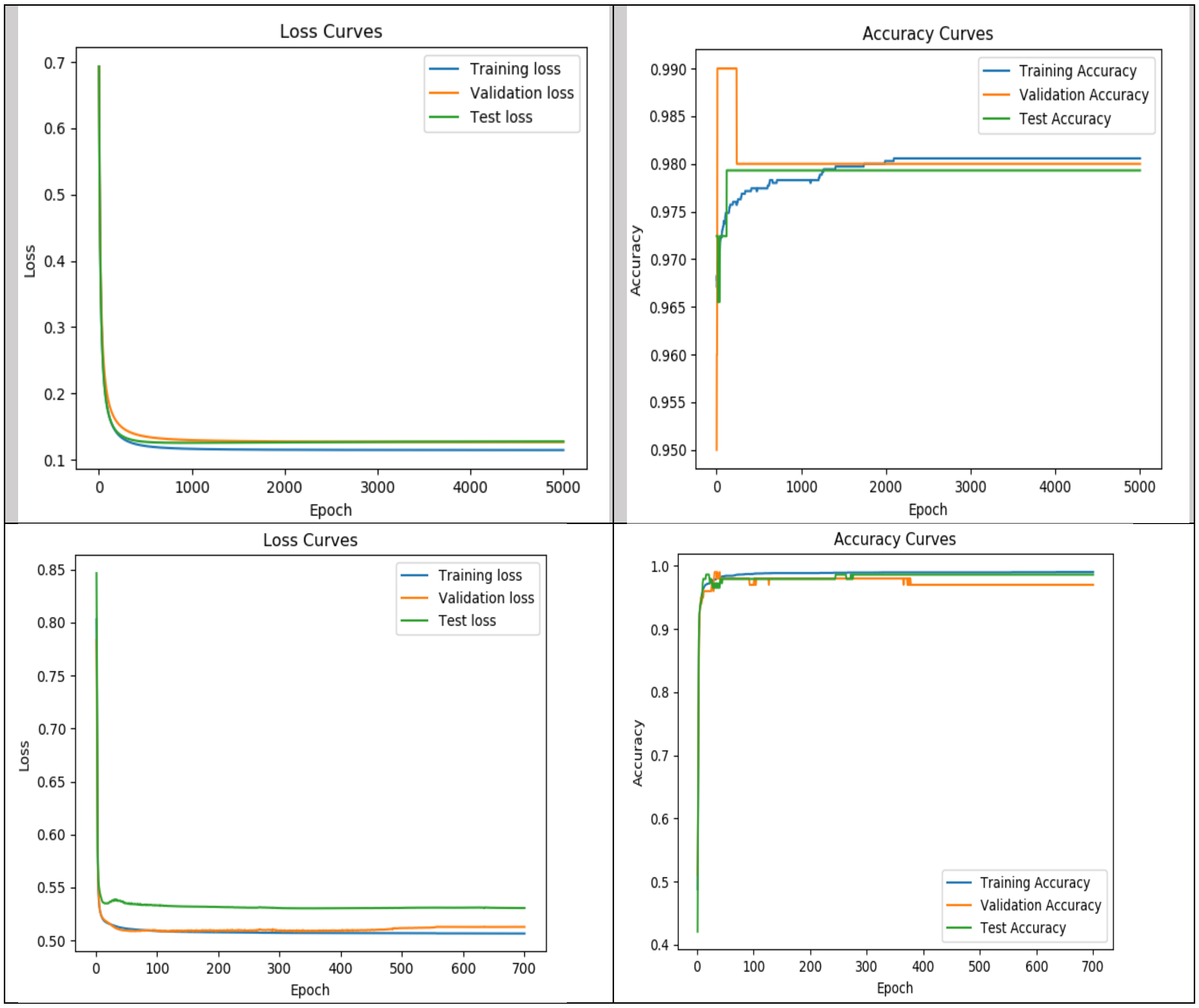
β2 = 0.9999		Training Accuracy: 0.98771 Validation Accuracy: 0.98000 Test Accuracy: 0.97931
ϵ = 1e-9		Training Accuracy: 0.98857 Validation Accuracy: 0.98000 Test Accuracy: 0.97931
ϵ = 1e-4		Training Accuracy: 0.98857 Validation Accuracy: 0.98000 Test Accuracy: 0.97931

Compared with MSE, beta1 and beta2 changes made a much smaller difference to final accuracy. The reason is mostly due to the accuracies being in a much higher range (>97%) therefore any influence to the final accuracies is relatively miniscule. However, similarly to the analysis of MSE, the increase of beta1 decreased final validation accuracy and the increase of beta2 increased final validation accuracy. The oscillatory behaviour in the transient state (pre-steady state) introduced from beta1=0.95 to 0.99 can still be seen as well. Also consistent with

MSE, the 5 order of magnitude jump in epsilon made no difference due to its negligibility compared to beta2.

In terms of final model performance, cross entropy loss universally performed significantly better, with a jump of 65-75% final accuracy to 97%+ final accuracy. The thresholding characteristic offered by the sigmoid function is much preferred for this application of binary classification. Incorrect results are given zero weight in training for cross entropy loss, while for mean squared error, lower probability incorrect results are given more weight. MSE is better for regression, for the same reason though. Using cross entropy is also more likely to lead to a convex optimization compared to mean square error.

3.1.6 Comparison against Batch GD



Top row: Batch Gradient Descent. Bottom row: Stochastic Gradient Descent (minibatch = 100) with ADAM. Both use cross entropy loss function.

There are a few clear differences between batch GD and SGD. The steady state loss reported by SGD is significantly higher than batch SGD. This is likely due to the variance in SGD giving an on average larger deviation from ground truth. Additionally, for batch GD, in one epoch the model was able to be trained to a very high accuracy. SGD takes a couple epochs at least to reach this ~95% accuracy. This is due to batch GD taking a more directly path to optimality while SGD wanders a bit before reaching it. This additional noise in SGD can be seen in the accuracy curve. The randomness in sampling in the batches is the cause of this.