1.1

$$L = \frac{1}{N}\sum_{n=1}^{N}[-y^{(n)}log\widehat{y}(x^{(n)}) - (1-y^{(n)})log(1-\widehat{y}(x^{(n)}))] + \frac{\lambda}{2}||w||_2^{\ 2}$$

To get the gradient in terms of w and b, apply definition and chain rule:

$$\triangledown L(w) = \triangledown\frac{1}{N}\sum_{n=1}^{N}[-y^{(n)}log\widehat{y}(x^{(n)}) - (1-y^{(n)})log(1-\widehat{y}(x^{(n)}))] + \frac{\lambda}{2}||w||_2^{\ 2}$$

$$\triangledown L(w) = \frac{1}{N}\triangledown\sum_{n=1}^{N}[-y^{(n)}log\widehat{y}(x^{(n)}) - (1-y^{(n)})log(1-\widehat{y}(x^{(n)}))] + \frac{\lambda}{2}\triangledown||w||_2^{\ 2}$$

$$\begin{aligned}\triangledown L(w) =& -\frac{1}{N}\triangledown[y^{(1)}log\widehat{y}(x^{(1)}) + y^{(2)}log\widehat{y}(x^{(2)}) + ... + y^{(N)}log\widehat{y}(x^{(N)})] \\ & - \frac{1}{N}\triangledown[(1-y^{(1)})log(1-\widehat{y}(x^{(1)})) + ... + (1-y^{(N)})log(1-\widehat{y}(x^{(N)}))] \\ & + \lambda w\end{aligned}$$

$$\begin{aligned}\triangledown L(w) =& -\frac{1}{N}[\frac{y^{(1)}}{\widehat{y}}\triangledown\widehat{y}(x^{(1)}) + ... + \frac{y^{(N)}}{\widehat{y}}\triangledown\widehat{y}(x^{(N)})] \\ & - \frac{1}{N}[\frac{1-y^{(1)}}{1-\widehat{y}(x^{(1)})}\triangledown(1-\widehat{y}(x^{(1)})) + ... + \frac{1-y^{(N)}}{1-\widehat{y}(x^{(N)})}\triangledown(1-\widehat{y}(x^{(N)}))] \\ & + \lambda w\end{aligned}$$

$$\begin{aligned}\triangledown L(w) =& \frac{1}{N}[\frac{y^{(1)}e^{-w^Tx^{(1)}-b}}{1+e^{-w^Tx^{(1)}-b}}\triangledown(-w^Tx - b) + ... + \frac{y^{(N)}e^{-w^Tx^{(N)}-b}}{1+e^{-w^Tx^{(N)}-b}}\triangledown(-w^Tx - b)] \\ & - \frac{1}{N}[...] + \lambda w\end{aligned}$$

$$\triangledown L(w) = -\frac{1}{N}\sum_{n=1}^{N}[\frac{y^{(n)}x^{(n)}}{1+e^{w^Tx^{(1)}+b}} + \frac{(y^{(n)}-1)x^{(n)}}{1+e^{-w^Tx^{(1)}-b}}] + \lambda w$$

Similarly, $\triangledown L(b) = -\frac{1}{N}\sum_{n=1}^{N}[\frac{y^{(n)}}{1+e^{w^Tx^{(1)}+b}} + \frac{y^{(n)}-1}{1+e^{-w^Tx^{(1)}-b}}]$

Using expression L, $\triangledown L(w)$, and $\triangledown L(b)$ gives below codes:

```
# loss = cross-entropy loss + regularization term
def loss(w, b, x, y, reg):
  loss = ce_loss(w, b, x, y) + (reg / 2) * np.linalg.norm(w) ** 2
  return loss

'''some help functions for loss(*)'''
# sigmoid function
def sigmoid(z):
  return 1 / (1 + np.exp(-z))

# predict y value and returns the probability
# the return value is a [n * d][d * 1] = [n * 1] array
def predict(w, b, x):
```

```
  return sigmoid(x @ w + b)

# calculate cross-entropy loss
def ce_loss(w, b, x, y):
  yHat = predict(w, b, x)
  ceLoss = np.sum(- y * np.log(yHat) - (1 - y) * np.log(1 - yHat))
  ceLoss /= yHat.size
  return ceLoss
'''help functions for loss(*) end here'''

# calculate gradient for loss function in respect to weight & bias
def grad_loss(w, b, x, y, reg):
  gradW, gradB = grad_ce_loss(w, b, x, y)
  gradW += reg * w
  return gradW, gradB

'''some help functions for grad_loss(*)'''
# calculate gradient in terms of weight, bias of cross-entropy loss
def grad_ce_loss(w, b, x, y):

  pos = -np.sum(y * (1 - predict(w, b, x)) * x) / y.size
  neg = -np.sum((y - 1) * predict(w, b, x) * x) / y.size
  gradCeW = pos + neg

  pos = -np.sum(y * (1 - predict(w, b, x))) / y.size
  neg = -np.sum((y - 1) * predict(w, b, x)) / y.size
  gradCeB = pos + neg

  return gradCeW, gradCeB
'''help functions for grad_loss(*) end here'''
```

1.2
Pass

1.3/4
Unfortunately, there exist two unsolved errors in pa1_1.ipynb. If they are solved, pa1_1.ipynb provides the full code for the analysis in these two sections.

2.1
Note: pa1_2.ipynb uses tensorflow 2.x approach so there is no computational graph.
2.2
Given the size of minibatch is 500, the number of minibatchs in each training epoch is 3500/500=7. Also given that the model trains at least 700 epochs, there are total 7*700=4900 distinct minibatchs created from shuffle and split.

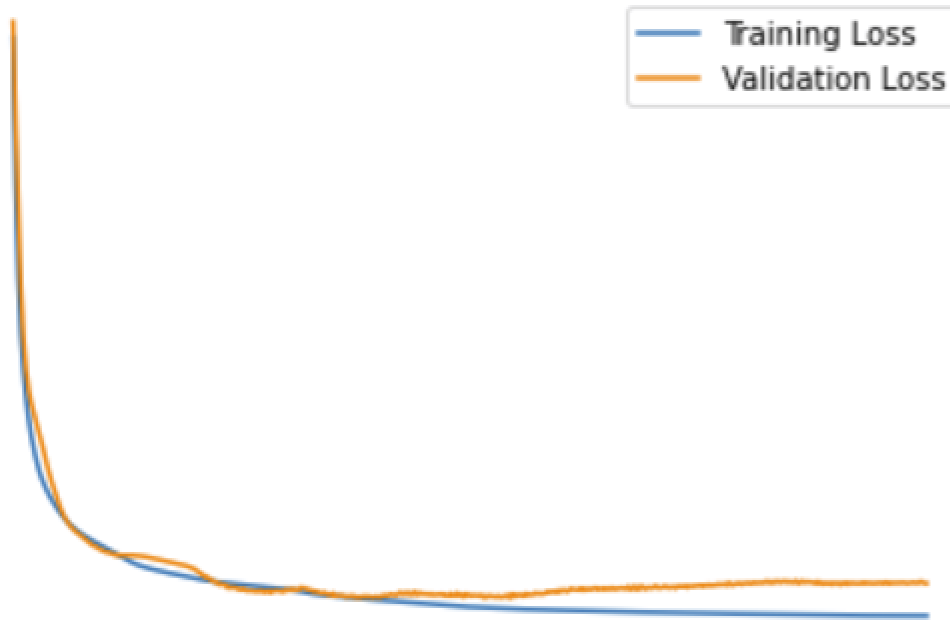

Figure 1: Training Loss vs Epochs @ batch size = 500

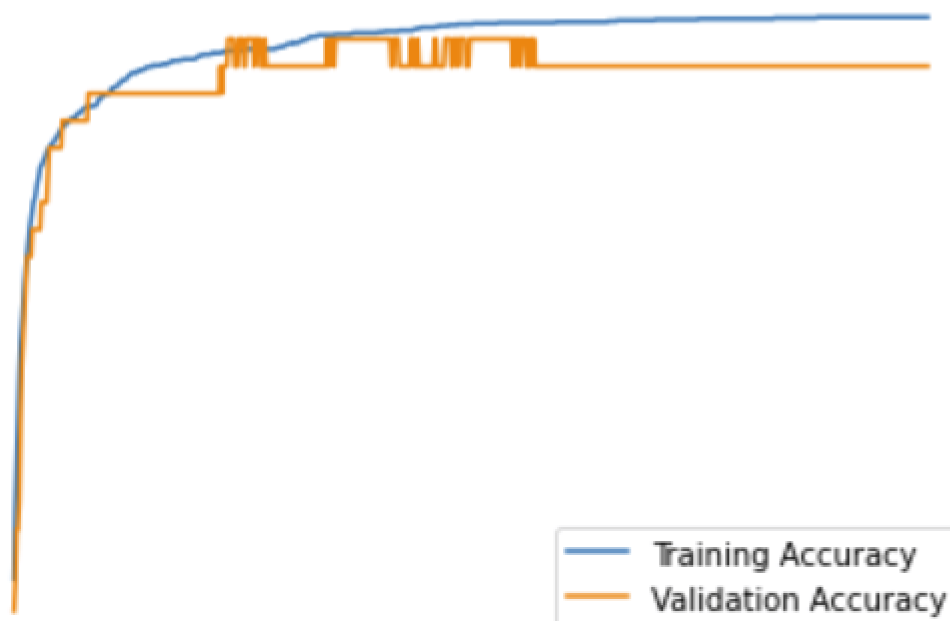

Figure 2: Training Accuracy vs Epochs @ batch size = 500

2.3

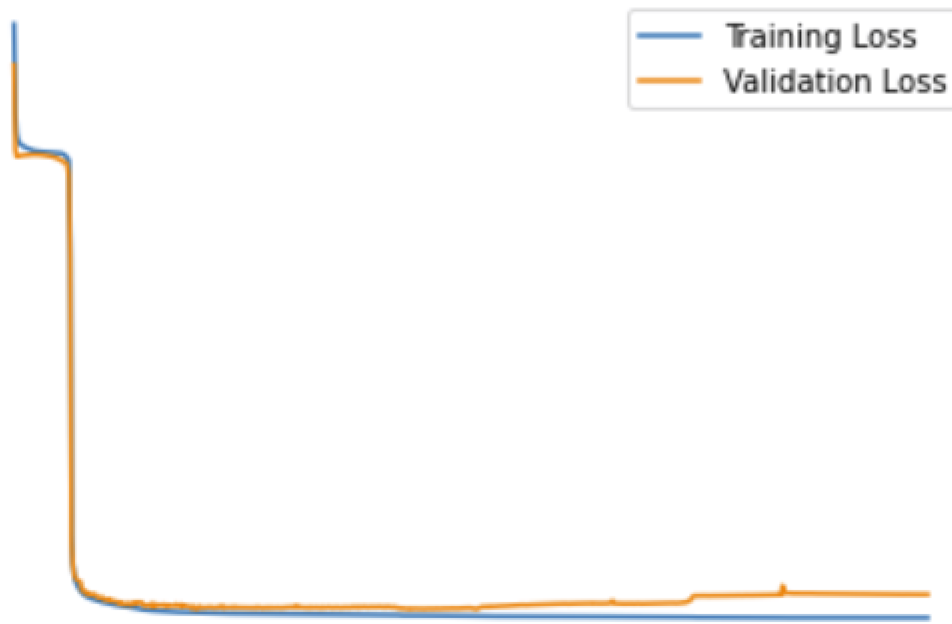

Figure 3: Training Loss vs Epochs @ batch size = 100

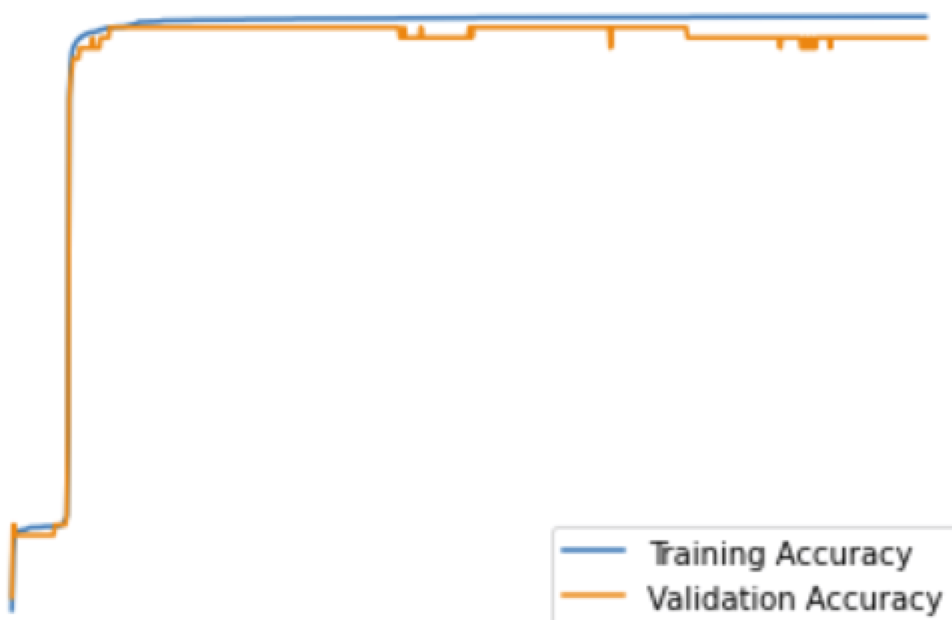

Figure 4: Training Accuracy vs Epochs @ batch size = 100

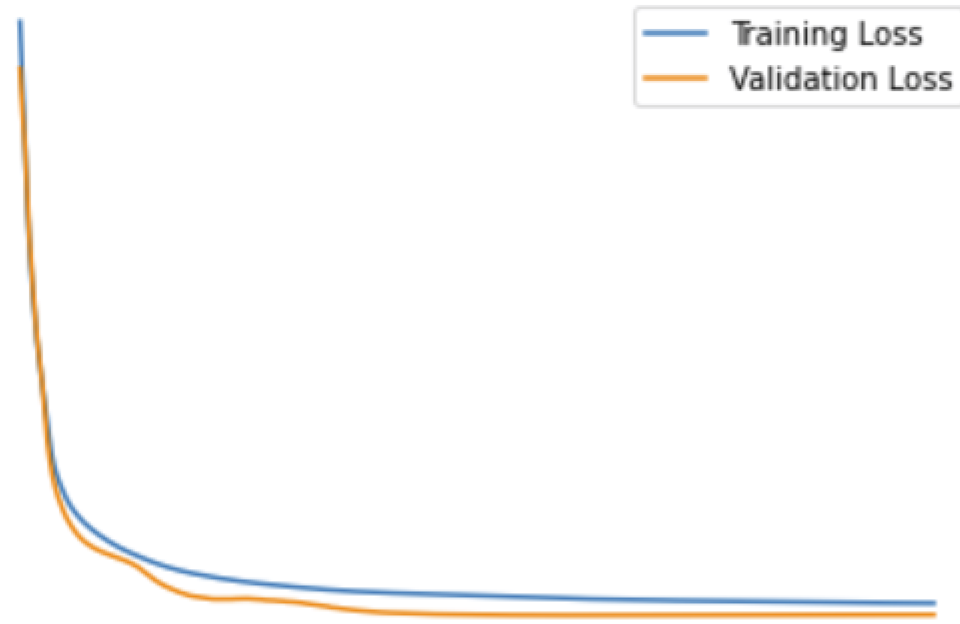

Figure 5: Training Loss vs Epochs @ batch size = 700

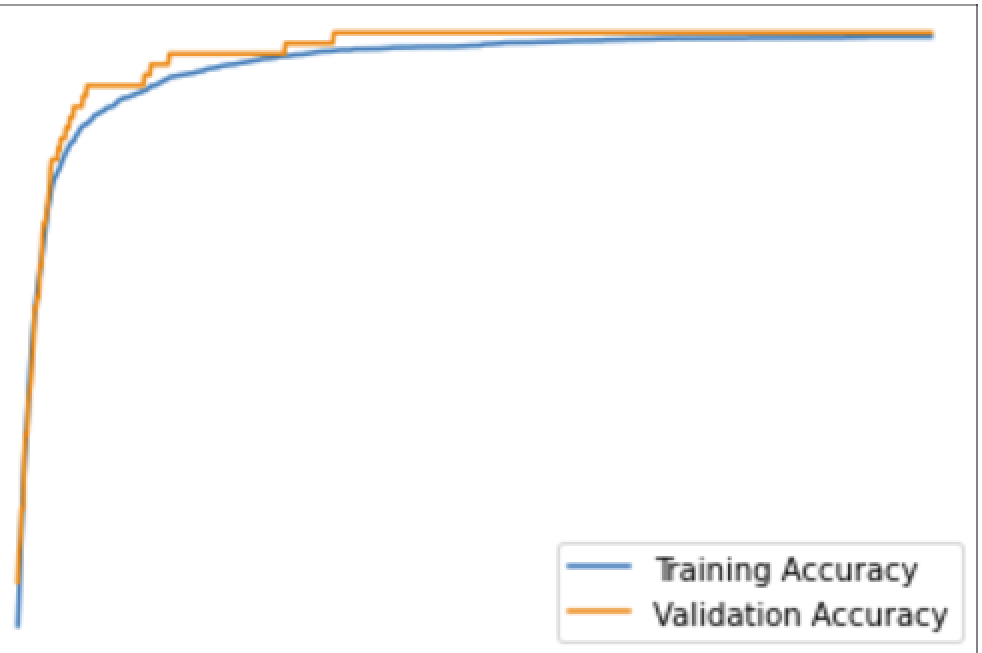

Figure 6: Training Accuracy vs Epochs @ batch size = 700

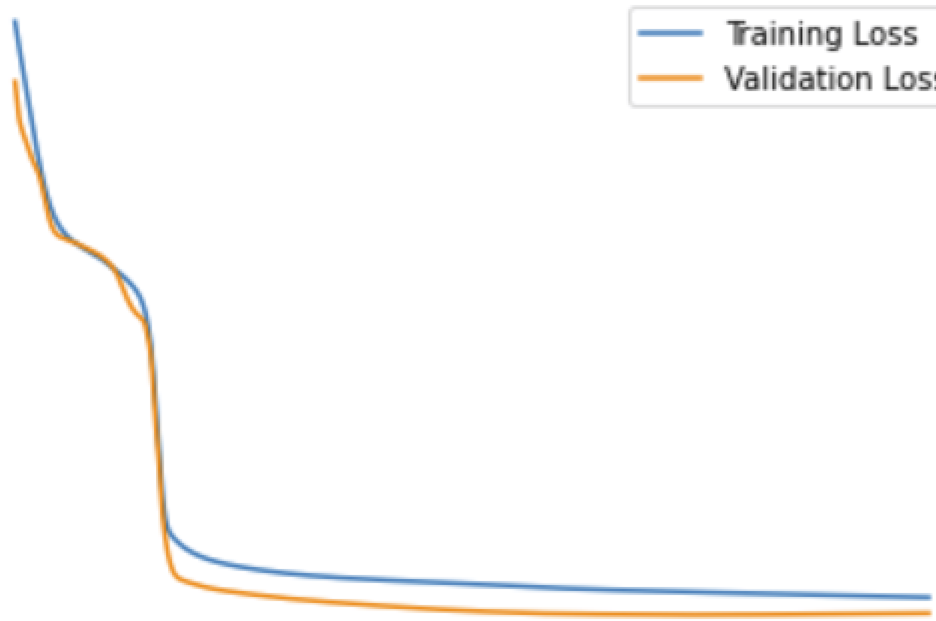

Figure 7: Training Loss vs Epochs @ batch size = 1750

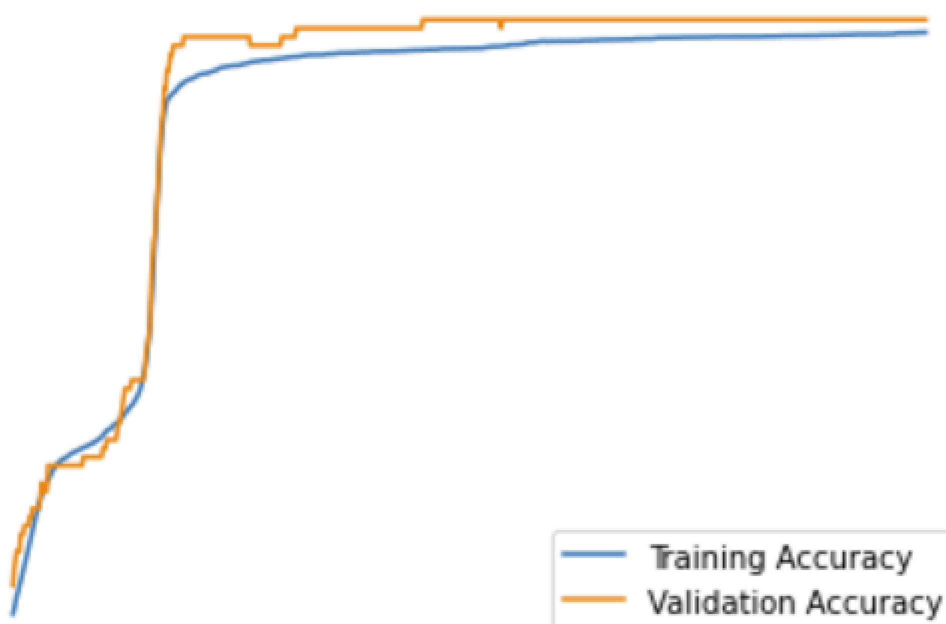

Figure 8: Training Accuracy vs Epochs @ batch size = 1750

Under 700 epochs, they perform equally well. However, the algorithm runs faster at batch size equals to 1750, which should be selected

2.4

beta1	default	0.95	0.99
Train accuracy	0.9886	0.9886	0.9883

Valid accuracy	0.99	0.99	0.98
Test accuracy	0.9655	0.9793	0.9862

Table 1: Final Accuracy @ beta1 = default, 0.95, 0.99

beta2	default	0.99	0.9999
Train accuracy	0.9886	0.9903	0.9871
Valid accuracy	0.99	0.97	0.98
Test accuracy	0.9655	0.9793	0.9655

Table 1: Final Accuracy @ beta2 = default, 0.99, 0.9999

epsilon	default	1e-09	1e-04
Train accuracy	0.9886	0.988	0.9857
Valid accuracy	0.99	0.97	0.97
Test accuracy	0.9655	0.9862	0.9586

Table 1: Final Accuracy @ epsilon = default, 1e-09, 1e-04

After comparing three tables, beta1 should select 0.99, because it has the highest test accuracy. Beta2 should select 0.99, because it has the highest test accuracy, and epsilon should select 1e-09, for the same reason

2.5

Adam is better because of shuffle, which gives randomize, and small batches.