#1

1. **[4pts] Feature Maps.** Suppose we have the following 1-D dataset for binary classification:

x	t
-1	1
1	0
3	1

(a) **[2pts]** Argue briefly (at most a few sentences) that this dataset is not linearly separable. (Your argument should resemble the one we used in lecture to prove XOR is not linearly separable.)

(b) **[2pts]** Now suppose we apply the feature map

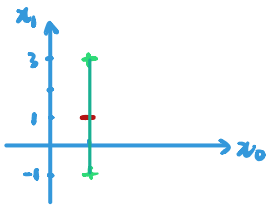
$$\boldsymbol{\psi}(x) = \begin{pmatrix} \psi_1(x) \\ \psi_2(x) \end{pmatrix} = \begin{pmatrix} x \\ x^2 \end{pmatrix}.$$

Assume we have no bias term, so that the parameters are w_1 and w_2. Write down the constraint on w_1 and w_2 corresponding to each training example, and then find a pair of values (w_1, w_2) that correctly classify all the examples. Remember that there is no bias term.

a) Suppose our training set, with the dummy feature x_0 included. (taking the value of 1 always).

input

x_0	x_1	t
1	-1	1
1	1	0
1	3	1



Proof by Contradiction:

- The training examples of the two different types all lies on the same straight line alternately.
- If these three pts lie in a half-space, the green line segment connecting them also lie in the same halfspace.
- Suppose there were some feasible weights (Hypothesis). If the positive examples are in the positive half-space, then the line thru (1, -1) and (1, 3) which is $x_0 = 1$ must be in the positive half-space as well.
- With the claim the given 1-D dataset is linearly separable, there should exist a line segment thru (1, 1) lie within the negative half-space, which would never intersect with the segment in the positive half-space.
- However any segment thru (1, 1) must interset the green line segment, and intersection cannot lie in both half-spaces. Contradiction!
- Therefore, the give 1-D dataset is not linearly separable.

b)

		$\psi_1(x)$	$\psi_2(x)$	
x_0	x_1	x_1	x_1^2	t
1	-1	-1	1	1
1	1	1	1	0
1	3	3	9	1

$z = \vec{w}^T \phi(\vec{x}) + b$ (no bias term)

$z = w_1\psi_1(x) + w_2\psi_2(x)$

$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$

$w_1(-1) + w_2(1) = -w_1 + w_2 \geq 0$

$w_1(1) + w_2(1) = w_1 + w_2 < 0$

$w_1(3) + w_2(9) \Rightarrow w_1 + 3w_2 \geq 0$

} constraints on (w_1, w_2)

Find a pair of valid (w_1, w_2) :

Assume $w_1 = -3w_2$ which satisfies ③

Then, we plug it into ① and ②, we get

$3w_2 + w_2 = 4w_2 \geq 0$

$-3w_2 + w_2 = -2w_2 < 0$

And we can see if $w_2 > 0$, ① & ② will also be satisfied.

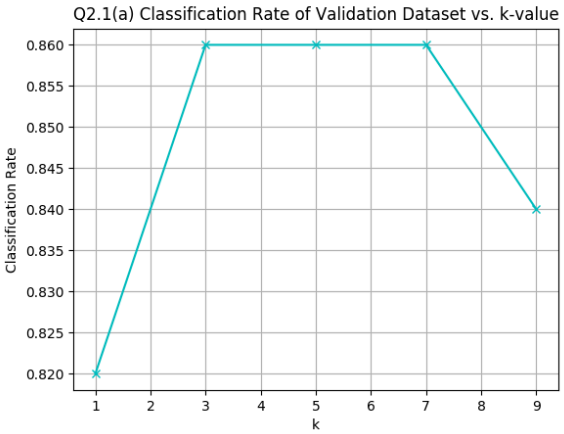
Thus, let $w_2 = 1$, then $w_1 = -3$

We get a valid (w_1, w_2) pair : $(-3, 1)$

2.1 (a)

See "`run_knn`" in "`run_knn.py`".

Graph: plots the classification rate on the validation set vs. k-values, where k = 1, 3, 5, 7, 9



Report the plot in the write up:
- the plot gives a tendency of first increase of the classification rate and then decrease, as k-value increases
- the classification rate has the highest among k=3, 5, 7 all with a rate of 0.860
- k=1, 9 has a much lower classification rate in comparison with the other k-values
- It suggests that k between 3 and 7 are best choices of k-value for this classification.

2.1 (b) Comments on the performance of the classifier:

- the classifier works well, but it still has a quite significant classification error even when it performs the best. (Such as given the k value of 3 or 5 or 7, the classification error is of 14% which is not negligible)

The k-value I would choose:

- k=5 accommodates the algorithm speed due to the potentially large-sized data, and the accuracy of classification

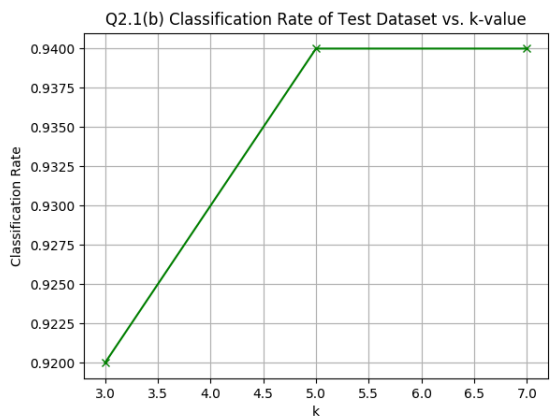
The classification rate for chosen k value, k* = 5, for the test dataset:

classification rate for k* = 0.94
classification rate for k*-2 = 0.92, where k*-2 = 3
classification rate for k*+2 = 0.94, where k*+2 = 7

The test performance is much better than the validation performance of these k values.

k	3	5	7
test performance	0.92	0.94	0.94
validation performance	0.86	0.86	0.86

Graph: plots the classification rate on the test set vs. k-values, where k = 3, 5, 7 and k* = 5



2.2(a) see 3 functions in “logistic.py”
2.2(b, c)

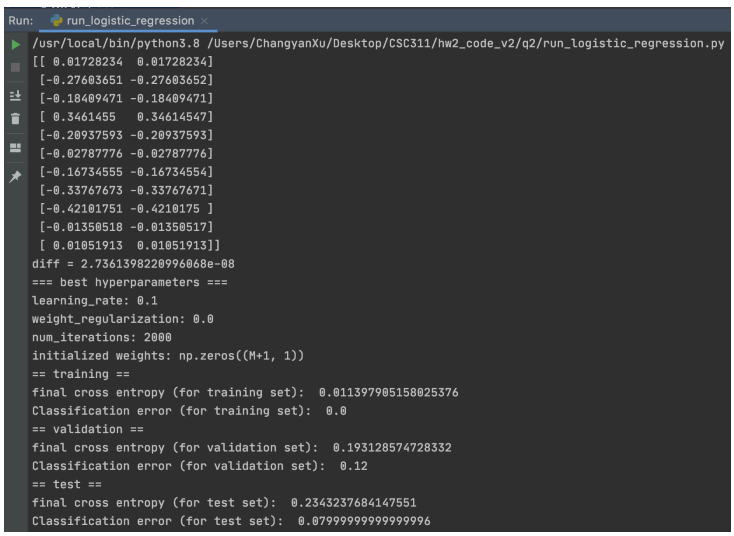
See run_logistic_regression in run_logistic_regression.py

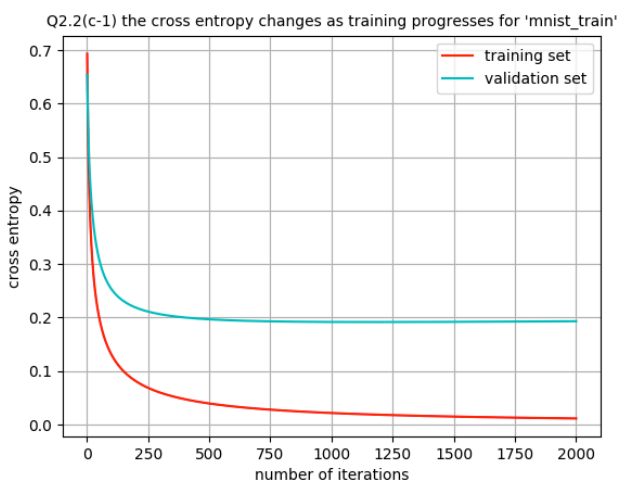
for "mnist_train":

By checking the returned value of run_check_grad, I got
diff = 2.7...e-08, which is smaller than 1e-06

The hyperparameter setting I found the best and chose for "mnist_train" are
learning_rate: 0.1
weight_regularization: 0.0 (not changed)
num_iterations: 2000
initialized weights: np.zeros((M+1, 1))

For training set, (for "mnist_train")
final cross entropy (for training set): approx. 0.01
Classification error (for training set): 0.0
For validation set, (for "mnist_train")
final cross entropy (for validation set): approx. 0.19
Classification error (for validation set): 0.12
For test set, (for "mnist_train")
final cross entropy (for test set): approx. 0.23
Classification error (for test set): approx. 0.08





with “mnist_train”

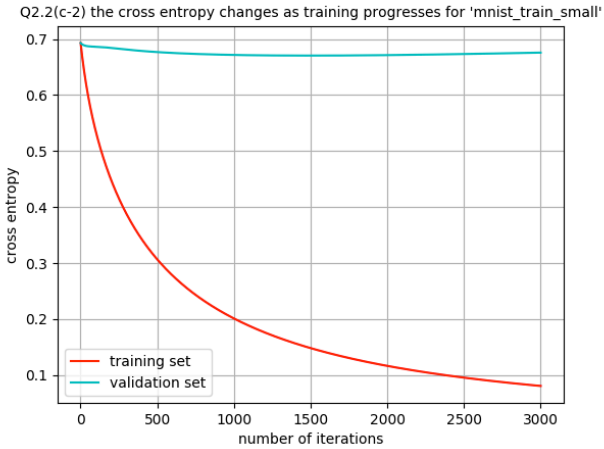
for "mnist_train_small":

By checking the returned value of run_check_grad, I got
diff = 1.98...e-08, which is smaller than 1e-06

The hyperparameter setting I found the best and chose for "mnist_train_small" are
learning_rate: 0.001
weight_regularization: 0.0 (not changed)
num_iterations: 3000
initialized weights: np.zeros((M+1, 1))

For training set, (for "mnist_train")
final cross entropy (for training set): approx. 0.08
Classification error (for training set): 0.0
For validation set, (for "mnist_train")
final cross entropy (for validation set): approx. 0.68
Classification error (for validation set): 0.36
For test set, (for "mnist_train")
final cross entropy (for test set): approx. 0.55
Classification error (for test set): approx. 0.24

```
Run:  run_logistic_regression
/usr/local/bin/python3.8 /Users/ChangyanXu/Desktop/CSC311/hw2_code_v2/q2/run_logistic_regression.py
[[-0.29517417 -0.29517415]
 [-0.18072567 -0.18072567]
 [ 0.25458931  0.2545893 ]
 [-0.23944381 -0.2394438 ]
 [ 0.03564982  0.03564982]
 [-0.02045323 -0.02045323]
 [ 0.25138206  0.25138205]
 [-0.15839253 -0.15839253]
 [-0.01107337 -0.01107336]
 [-0.02752779 -0.02752779]
 [-0.10596855 -0.10596855]]
diff = 1.9843818303821517e-08
=== best hyperparameters ===
learning_rate: 0.001
weight_regularization: 0.0
num_iterations: 3000
initialized weights: np.zeros((M+1, 1))
== training ==
final cross entropy (for training set):  0.08062842007165902
Classification error (for training set):  0.0
== validation ==
final cross entropy (for validation set):  0.6758913745859328
Classification error (for validation set):  0.36
== test ==
final cross entropy (for test set):  0.5486756128475324
Classification error (for test set):  0.24
```



with “mnist_train_small”

2.3(a) See `logistic_pen` in `logistic.py`

2.3(b) See `run_pen_logistic_regression` in `run_logistic_regression.py`

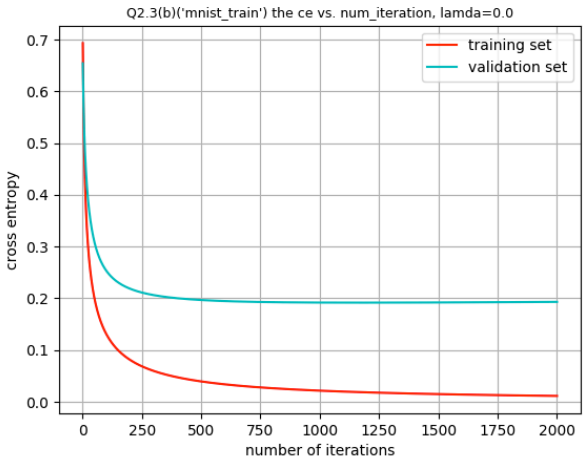
-- for "`mnist_train`"

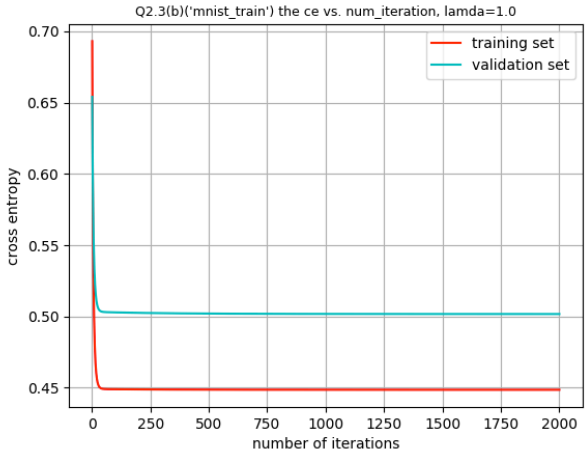
with hyperparameter settings in 2.2 for the same datasets

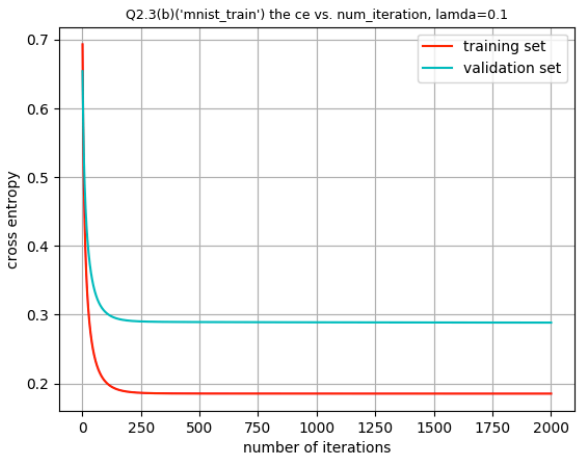
learning_rate: `0.1`

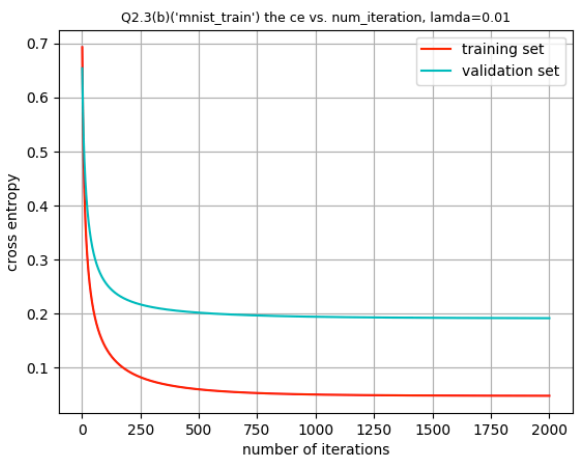
num_iterations: `2000`

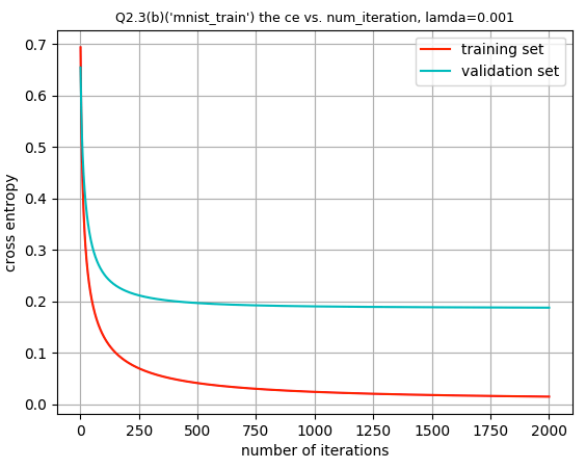
initialized weights: `np.zeros((M+1, 1))`











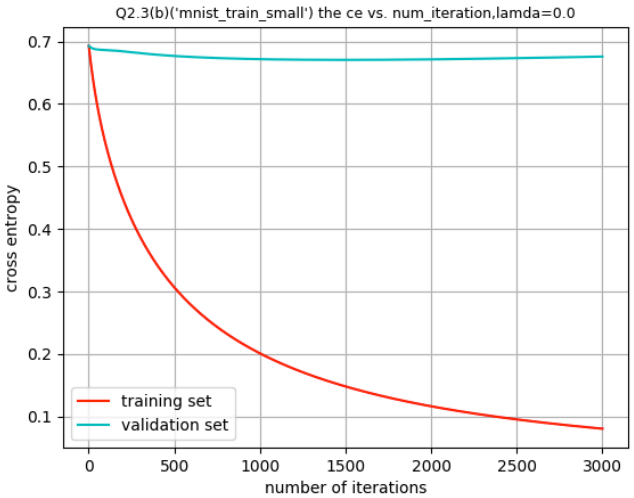
run_logistic_regression

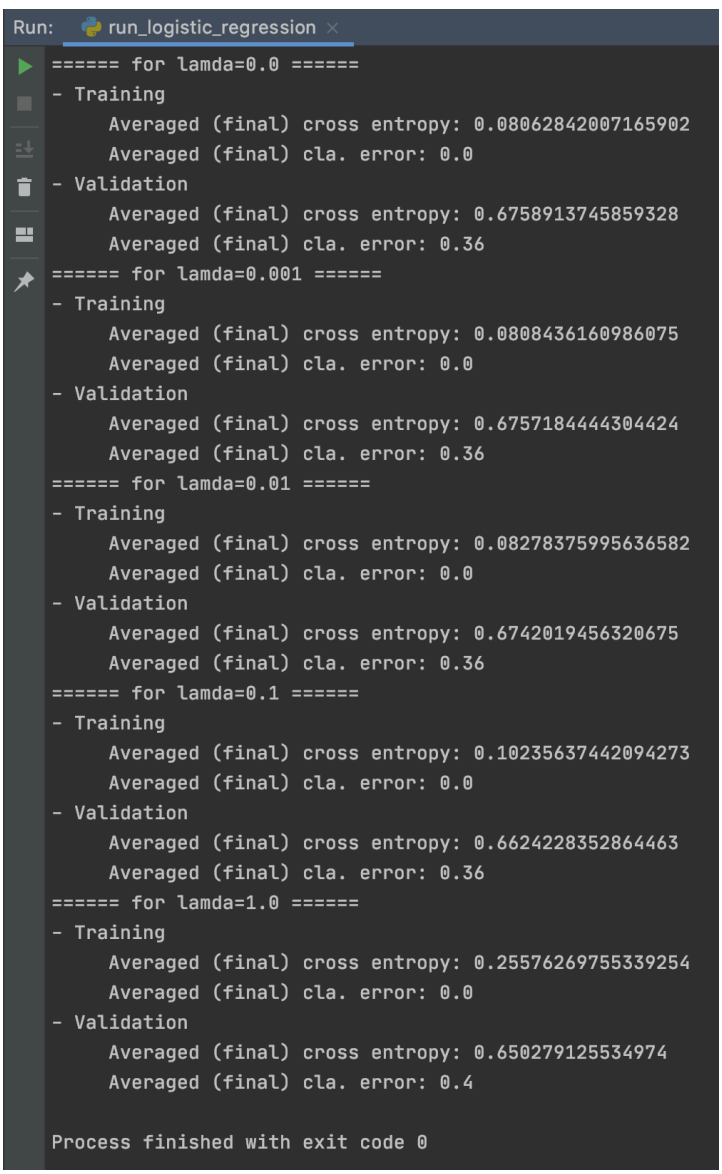
```
====== for lamda=0.0 ======
- Training
     Averaged (final) cross entropy: 0.011397905158025376
     Averaged (final) cla. error: 0.0
- Validation
     Averaged (final) cross entropy: 0.193128574728332
     Averaged (final) cla. error: 0.12
====== for lamda=0.001 ======
- Training
     Averaged (final) cross entropy: 0.015049664137655554
     Averaged (final) cla. error: 0.0
- Validation
     Averaged (final) cross entropy: 0.18777495091023905
     Averaged (final) cla. error: 0.12
====== for lamda=0.01 ======
- Training
     Averaged (final) cross entropy: 0.04812209856773088
     Averaged (final) cla. error: 0.0
- Validation
     Averaged (final) cross entropy: 0.191712795442279
     Averaged (final) cla. error: 0.12
====== for lamda=0.1 ======
- Training
     Averaged (final) cross entropy: 0.185178067183286
     Averaged (final) cla. error: 0.0
- Validation
     Averaged (final) cross entropy: 0.28848267409452333
     Averaged (final) cla. error: 0.12
====== for lamda=1.0 ======
- Training
     Averaged (final) cross entropy: 0.44853054013214494
     Averaged (final) cla. error: 0.05625000000000002
- Validation
     Averaged (final) cross entropy: 0.5017020190660223
     Averaged (final) cla. error: 0.16000000000000003

Process finished with exit code 0
```

2.3(b) console prints with “mnist_train”

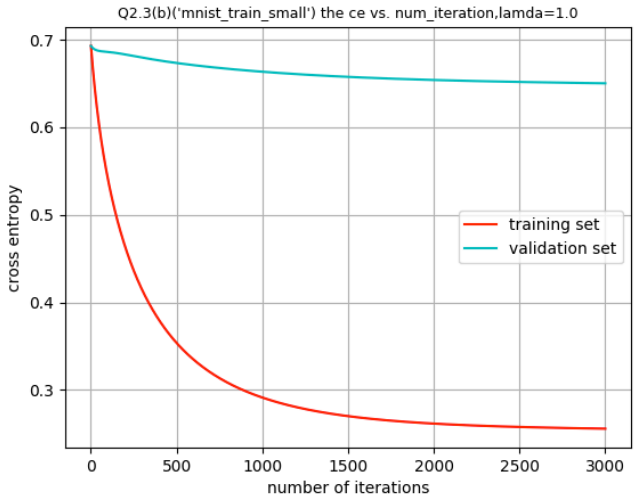
-- for "`mnist_train_small`"

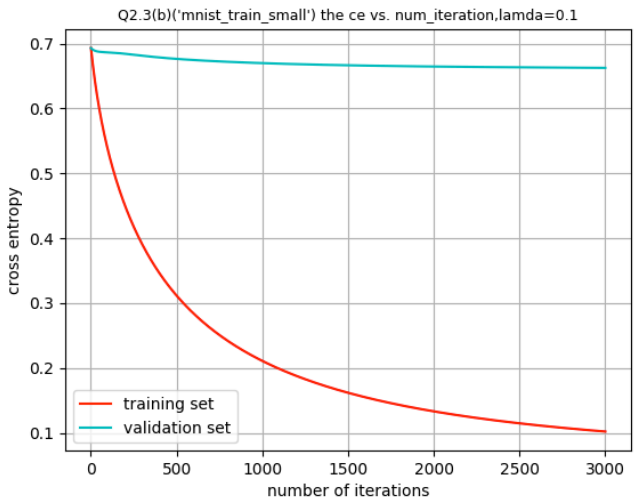
with hyperparameter settings in 2.2 for the same datasets
learning_rate: `0.001`
num_iterations: `3000`
initialized weights: `np.zeros((M+1, 1))`

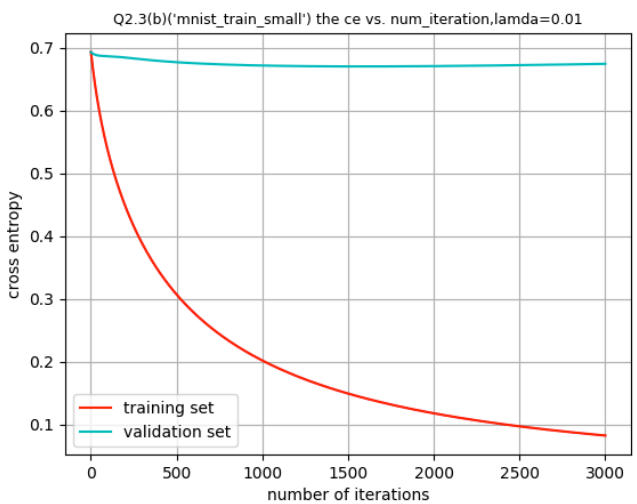


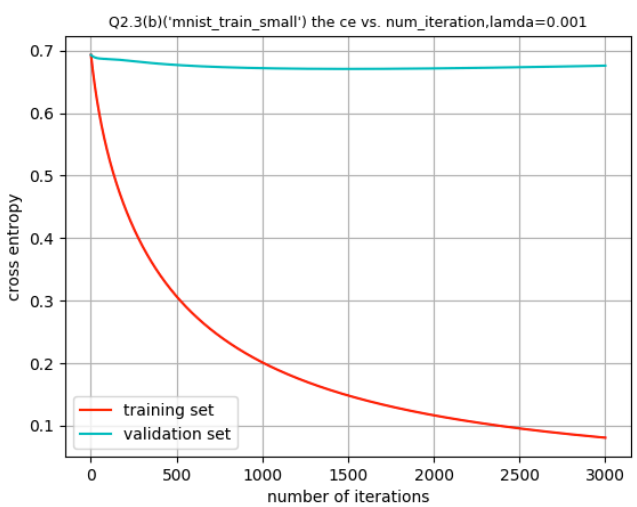


2.3(b) console prints with "mnist_train_small"









2.3(c)

How CE change when you increase lamda? .Explain why the CE behaved like this.

For dataset "mnist_train":

The cross entropy goes **up** as lambda increases.

For dataset "mnist_train_small":

The cross entropy goes **up** as lambda increases.

For validation dataset:

The cross entropy goes **down** as lambda increases.

Why:

the learning rate increases as lambda increases. The training weight tends to more fit to the training data, leading into an overfit. As we then apply the learned weight to validation dataset, the CE then goes up after first going down.

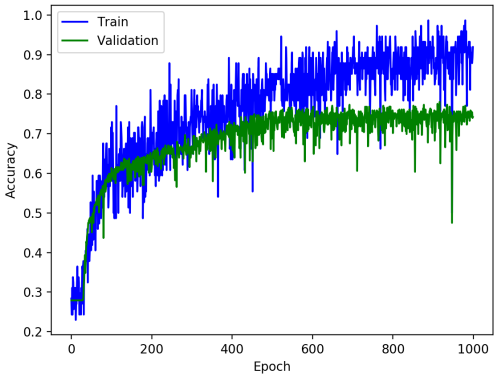
What is the best value of lamda, base on your experiment? Report the test CE and classification rate for the best of lamda.

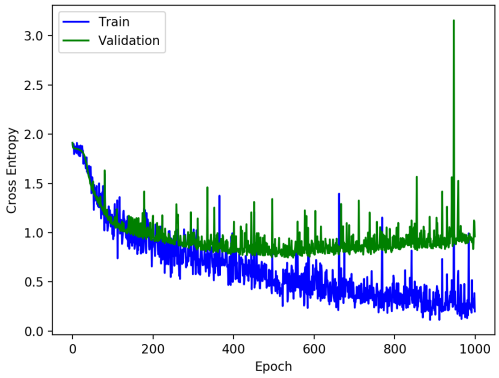
- best value of lamda: 0.1 for "mnist_train", 0.01 for "mnist_train_small"
- for test data, chose lamda=0.1 with "mnist_train"(larger dataset) as training data :
  - CE: approx. **0.27**
  -classification rate: approx. **0.92**

```
== Q2.3 (c) test data ==
final cross entropy (for test set):  0.2694053654024497
Classification error (for test set):  0.07999999999999996
```

3(a) See in `nn.py`

3(b) With the default hyperparameter settings, the graphs generated are as followed






```
CE: Train 0.24837 Validation 0.96157 Test 0.76612
Acc: Train 0.90545 Validation 0.74224 Test 0.74805
```

Since `error = 1 - acc`, we get

	Train	Validation	Test
error	approx. `0.10`	approx. `0.26`	approx. `0.25`

Examine the statistics:
- the validation error and test error are of similar values, which are all much greater than the training error
- the validation CE and training CE are significantly greater than the training CE

Examine both plots
- the training accuracy is generally greater than validation accuracy
  (i.e. the training error is generally lower than validation error)
- both training and validation accuracy increases as epoch increases,
  and they (tend to) converge to a fixed range of values
  (i.e.both training and validation error decreases as epoch increases)

How does the network's performance differ on the training set vs. the validation set during learning?
- both training error and training CE are lower than the counterparts in validation set.
- it performs worse for the validation sets comparing to the training set.

3(c)
(i) Observations on convergence properties for 5 different alpha settings, where alpha = 0.001, 0.01, 0.1, 0.5, 1.0

as the learning rate changed from 0.001 to 1.0, the plot of cross entropies and accuracy become more flat.
For alpha = 0.001, the cross entropies and accuracy of training data compared with the default set are similar.
For validation set, two curves are become more stable but the tendency to converge slowed down because of small learning rate.

For alpha = 0.1, the cross entropies and accuracy of training data compared with the default set is similar with a slightly larger vibration with the curvature. For validation set, both curves became more flatten and cross entropies became stable around 1 and accuracy rate barely reached 0.7.

For alpha = 0.5 and 1.0, the cross entropies and accuracy curve for training is basically flat which means it is not learning anything with an extremely poor accuracy around 0.27.

(ii) Observations on convergence properties for 5 different mini-batch sizes, where mini-batch sizes = 10, 100, 200, 500, 1000

as the mini batch sizes changed from 10 to 1000, the plot of cross entropies and accuracy become more flat.
For alpha = 0.001, the cross entropies and accuracy of training data compared with the default set are similar.
For validation set, two curves are become more stable but the tendency to converge slowed down because of small learning rate.

For alpha = 0.1, the cross entropies and accuracy of training data compared with the default set is similar with a slightly larger vibration with the curvature. For validation set, both curves became more flatten and cross entropies became stable around 1 and accuracy rate barely reached 0.7.

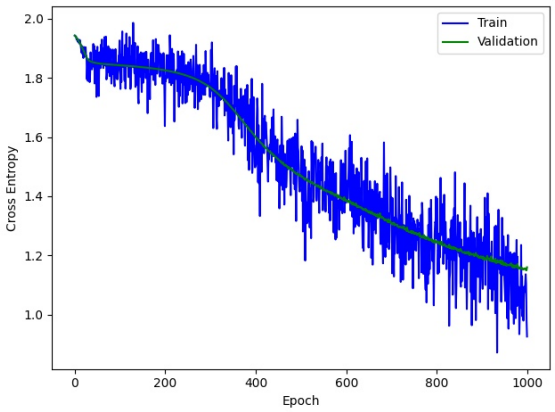
For alpha = 0.5 and 1.0, the cross entropies and accuracy curve for training is basically flat which means it is not learning anything with an extremely poor accuracy around 0.27.

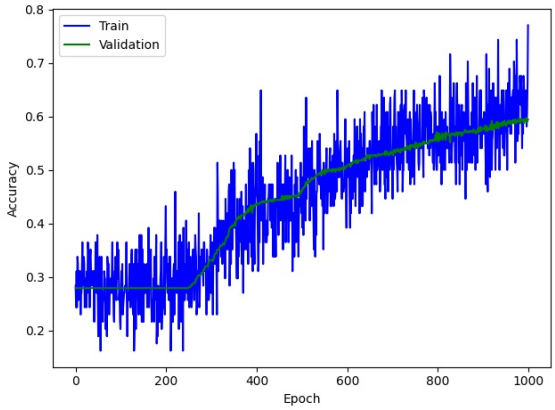
(iii)How would you choose the best value of these parameters?

Alpha:
the best value is **around 0.01**,
where cross entropy are generally low (less than 1),
and accuracy (greater than 70%) are generally high for validation set.

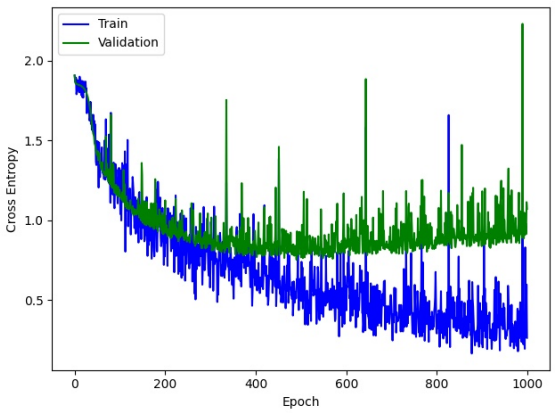
Batch_size:
the best value falls on **200**,
where the final cross entropy goes down to less than 0.8,
and accuracy (greater than 70%) are generally high for validation set.

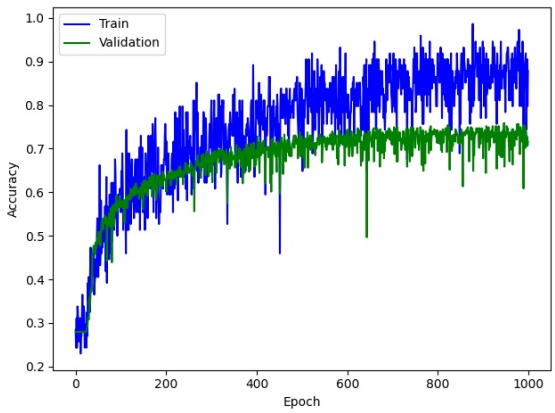
alpha = 0.001



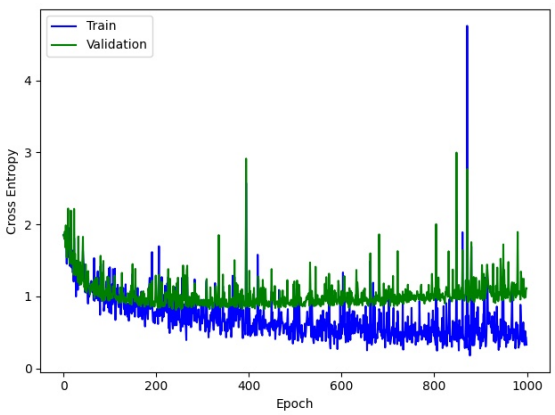


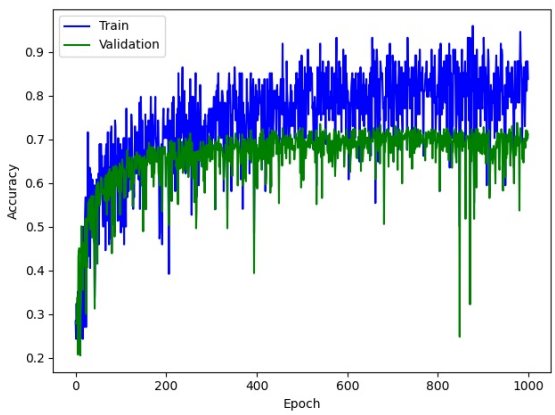
alpha = 0.01



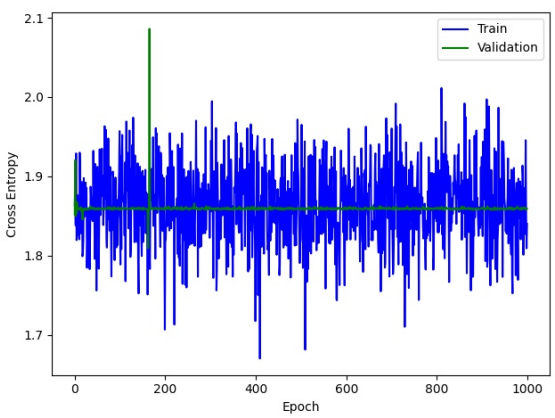


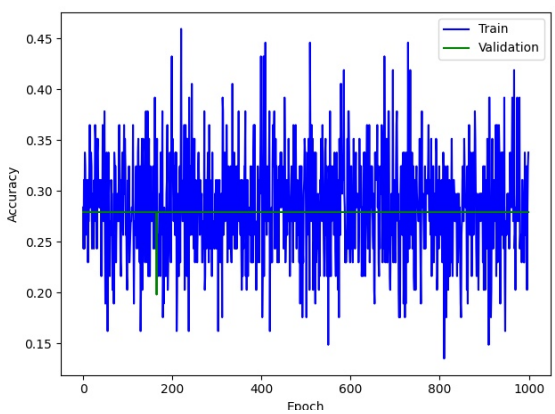
alpha = 0.1



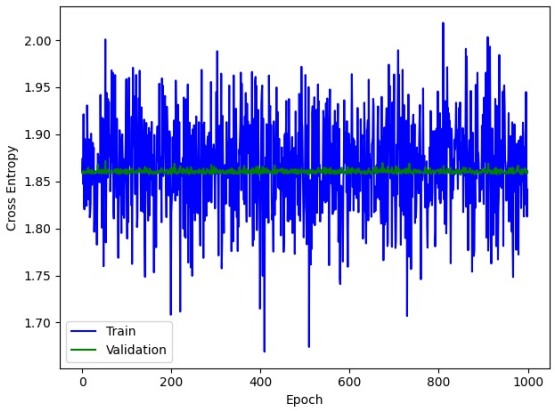


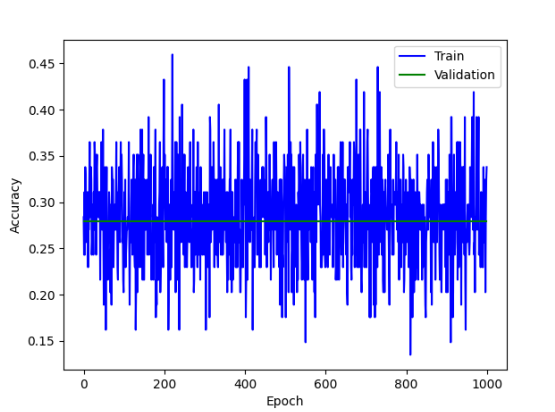
alpha = 0.5



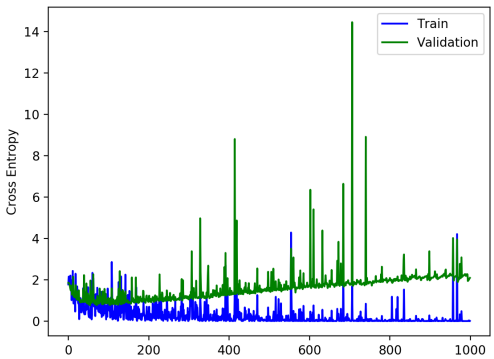


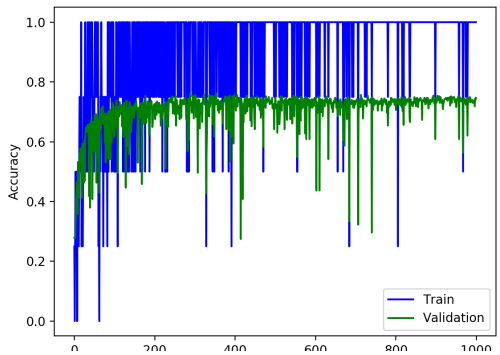
alpha = 1.0



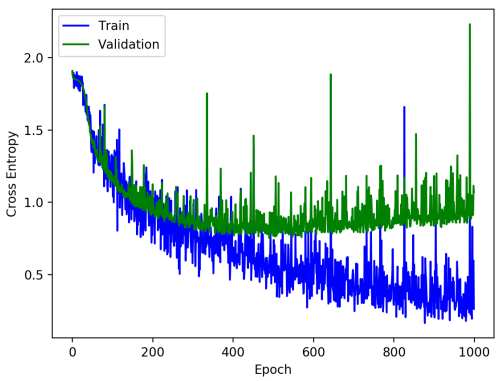


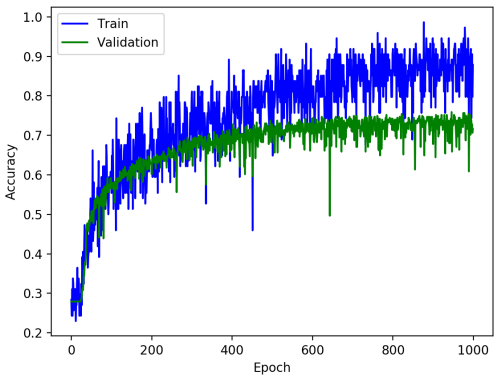
batch_size = 100



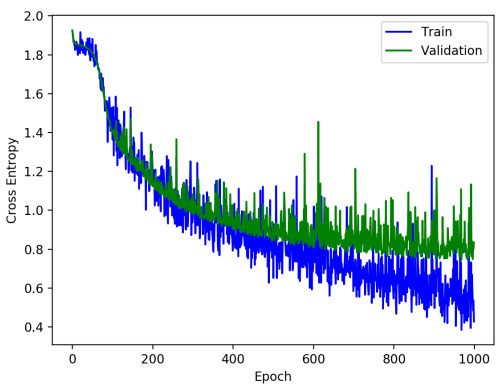


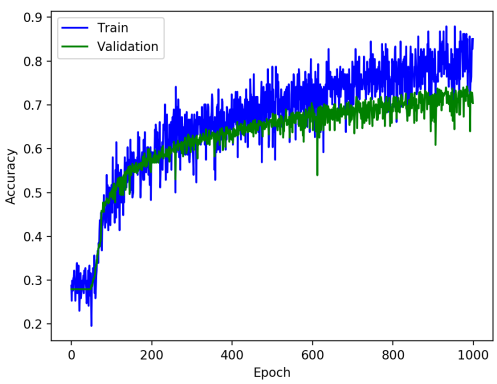
batch_size = 100



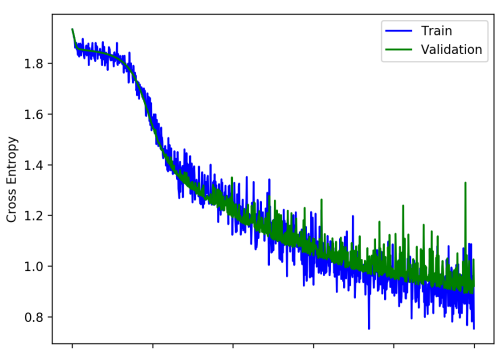


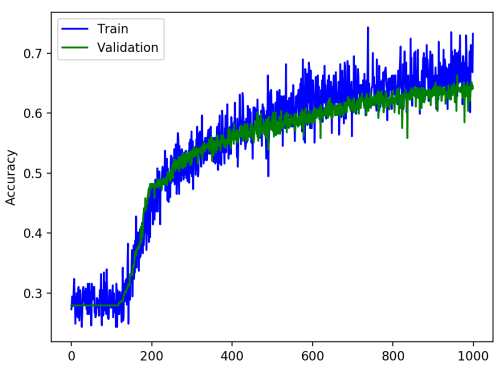
batch_size = **200**



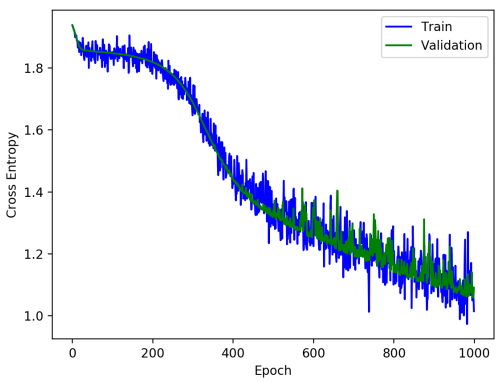


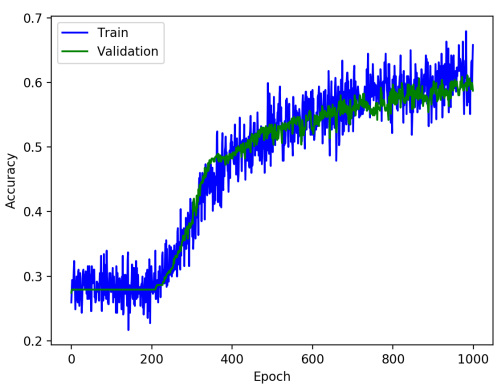
batch_size = 500





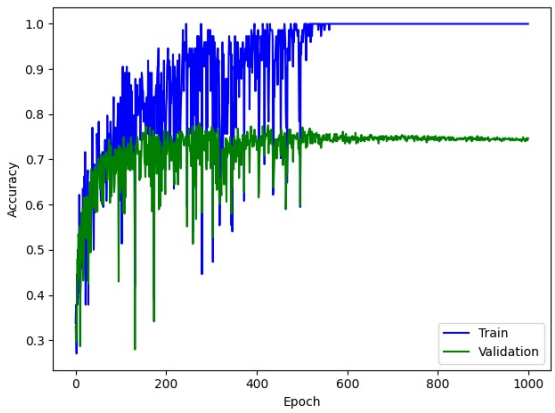
batch_size = 1000

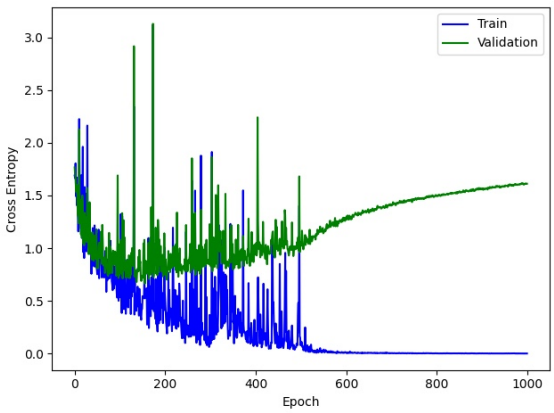




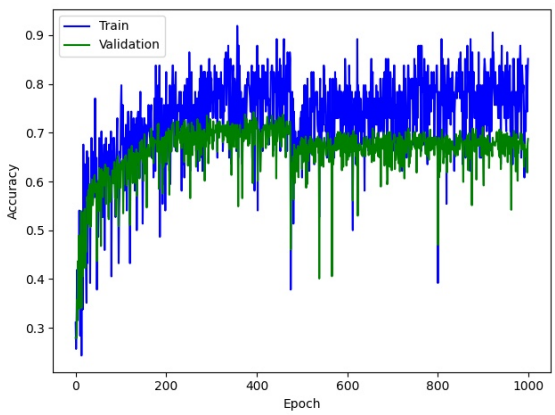
3(d)
For this question, the 3 (or 4) different set of values of hidden units for each layer of the Multilayer Perceptron is [[4, 10],[10,20],[10,90],[80,90]] and the learning rate was set to 0.05 with 1000 number of epochs for batch size at 200.

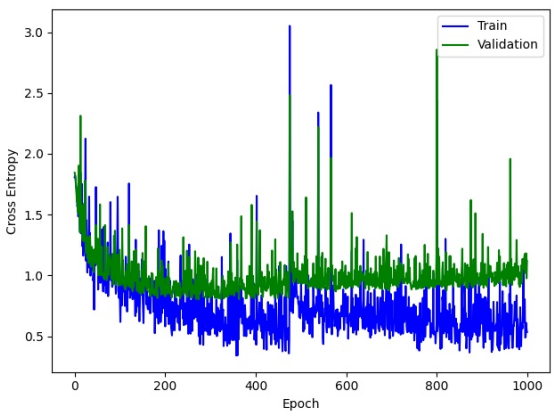
For the hidden units = [80,90]
CE: Train 0.00153 Validation 1.61115 Test 1.35626
Acc: Train 1.00000 Validation 0.74702 Test 0.80000
From the diagram below and the final ACC for train data above, as the weights become more fit to the training data around 500 epoch, the CE for validation data start to increase dramatically because of the overfit. The network become less general.



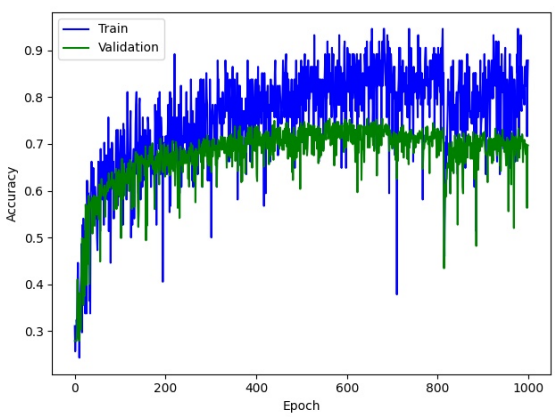


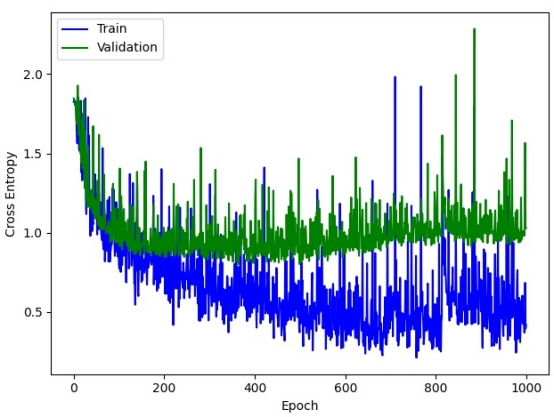
For the hidden units = [10,90]
CE: Train 0.74958 Validation 1.12062 Test 1.16229
Acc: Train 0.72199 Validation 0.68735 Test 0.64156
By choosing two hidden layers with rather large units, the weights are no longer overfits the training data with a more stable CE for training and validation by comparing with plots in 3c). However, around 500 epoch the accuracy for both training and validation data have an obvious decrease. That might because of certain pictures are hard to recognize.



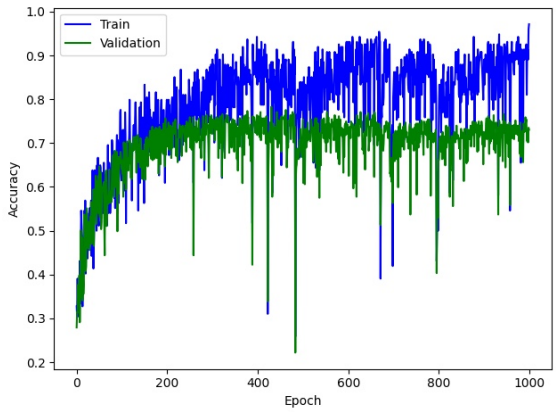


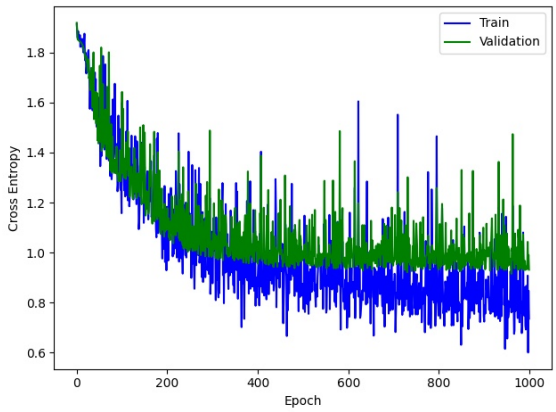
For the hidden units = [10,20]
CE: Train 0.52824 Validation 1.02715 Test 1.00963
Acc: Train 0.79727 Validation 0.69690 Test 0.69351
By choosing two relatively small hidden units, the accuracy performance is slightly less stable to compare with[10,90] hidden units. However, the CE for test and validation was still high compare to the default set up. There still exist a sudden increase on CE and sudden decrease on ACC for training and validation as the previous hidden units, but it happened after 800 epoch.





For the hidden units = [4,10]
CE: Train 0.74653 Validation 0.93181 Test 0.93354
Acc: Train 0.70806 Validation 0.63723 Test 0.63896





<u>Generalization:</u>

- If both hidden layer has a pair of smaller units, there will be a smaller difference between training and validation for both accuracy and CE.
- If both hidden layer has a pair of larger units, there will be a greater difference between training and validation for both accuracy and CE. It would lead to an overfit more easily.

- If the units of the two hidden layer has a greater difference, it will also lead to a better convergence (a smaller difference) between training and validation data for both accuracy and CE.