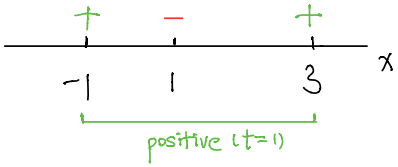
1.

a) Proof by contradiction.



By property of linearity, if 2 points lie in a half-space, the line segment connecting them must also lie in the same half-space.

Suppose, there exists some feasible weights, and a function $f: \mathbb{R} \to t \in \{0,1\}$, We know then $f(-1) = 1$ and $f(3) = 1$ implies that $\forall x \in [-1, 3]$, $f(x) = 1$.

However, we know $f(1) = 0$, so this contradicts and the dataset is not linearly separable.

b) For data point $x = -1$, $t = 1$

constraint: $-w_1 + w_2 > k$, $k \in \mathbb{R}$

For data point $x = 1$, $t = 0$

constraint: $w_1 + w_2 < k$, $k \in \mathbb{R}$

For data point $x = 3$, $t = 1$

constraint: $w_1 + 9w_2 > k$, $k \in \mathbb{R}$

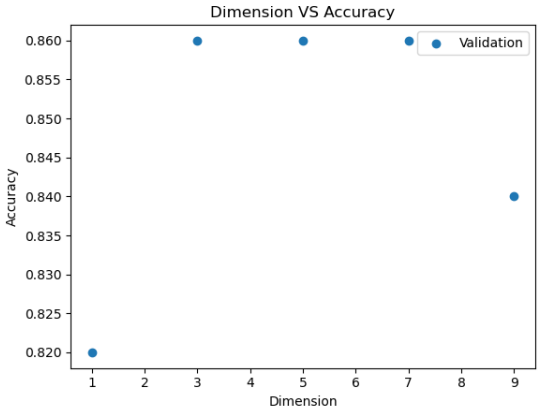
* Conventionally, we often take $k = 0$ to separate into positive/negative halfspace so the constraints are.

$$\begin{cases} -w_1 + w_2 > 0 \\ w_1 + w_2 < 0 \\ 3w_1 + 9w_2 > 0 \end{cases}$$

Take $w_1 = -2$, $w_2 = 1$ would satisfy the constraints

2.1

a)

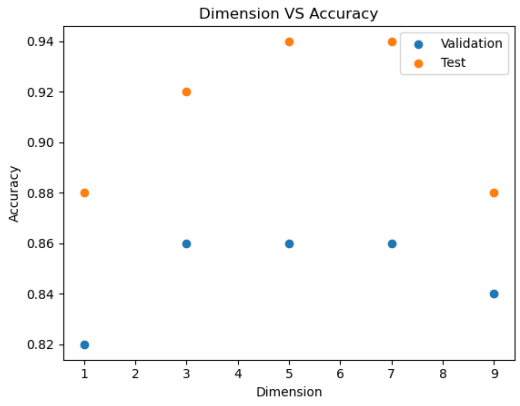


b) We can see that the classifier performs equally well when k=3 or k=5, k=7, with a classification rate of 86.0%.

I would choose k=5 because it has the best classification rate and its dimensionality is in the middle (capturing sufficient features without the pitfall of the curse of higher dimensionality).

$k^* = 5$, classification rate for k^*-2, k^*+2 are both 86.0%.

The test performance corresponds to the validation performance. For $k^* = 5$, test performance is optimal at 94%. For k^*-2, the classification rate is 92% and for k^*+2, the rate is 94%. We see that for k-values with lower validation performance, test performance is also lower.



2.2

b) I experimented with different hyperparameters

Iterations: 50, 100, 250, 500

Learning rate: 0.001, 0.01, 0.1, 1.0

Weights: zero-weights, randomly initialized

Train:

Optimal params: iterations = 250, learning rate = 0.1, weights = zero-weights

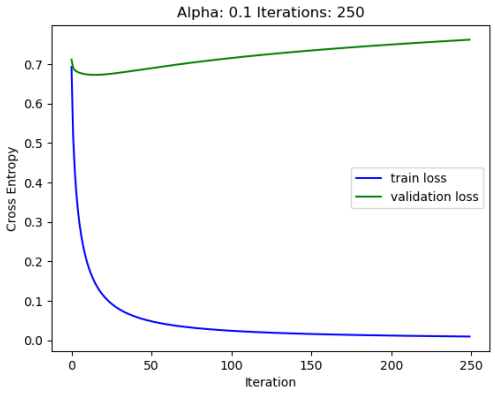
	Cross-Entropy	Error
Train set	0.068	0%
Validation set	0.211	12%
Test set	0.204	8%

Train small:

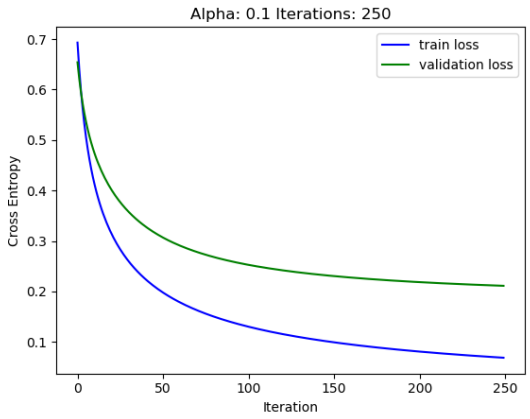
Optimal params: iterations = 250, learning rate = 0.1, weights = zero-weights

	Cross-Entropy	Error
Train set	0.009	0%
Validation set	0.762	30%
Test set	0.690	22%

c) Train small



Train



The result does not change because the weight initialized are zero-weights.

For the small training set, we see that cross-entropy for validation does not decrease significantly over iterations. Instead, it starts to increase after 30th iteration, indicating overfitting on the training set.

For the large training set, both train and validation loss decreases significantly in the early iterations. Validation loss begins to flatten after 200th iteration, shaving that 250th is an appropriate threshold to stop the gradient descent to avoid overfitting.

2.3

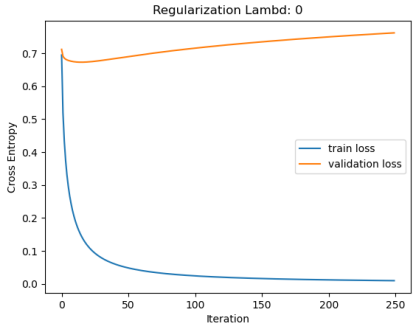
b) Train (mnist_train): learning rate = 0.1, iteration = 250

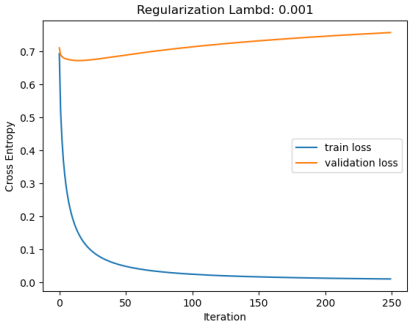
λ	Train CE	Valid CE	Train Accuracy	Valid Accuracy
0.0	0.068	0.211	100%	88%
0.001	0.070	0.212	100%	88%
0.01	0.082	0.217	100%	88%
0.1	0.186	0.290	100%	90%
1.0	0.449	0.502	94.4%	84%

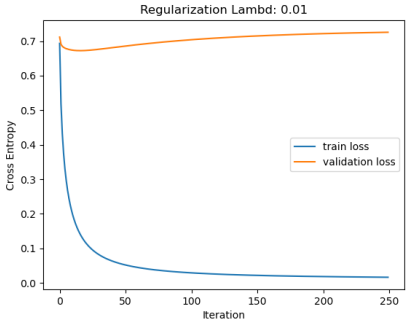
Train Small (mnist_train_small): learning rate = 0.1, iteration = 250

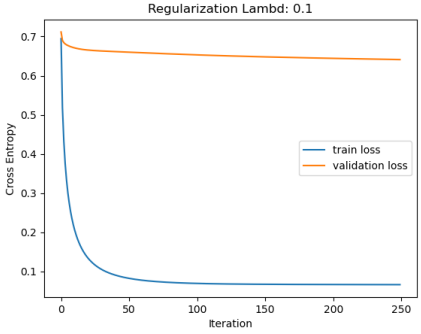
λ	Train CE	Valid CE	Train Accuracy	Valid Accuracy
0.0	0.009	0.762	100%	70%
0.001	0.010	0.758	100%	70%
0.01	0.014	0.725	100%	70%
0.1	0.059	0.641	100%	66%
1.0	0.237	0.625	100%	68%

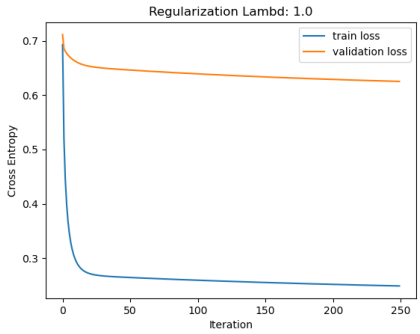
Train Small (to clarify, the values plotted are cross entropy (does not include penalty)



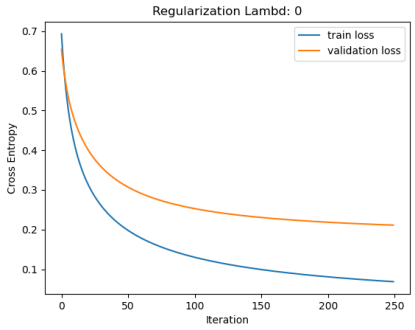


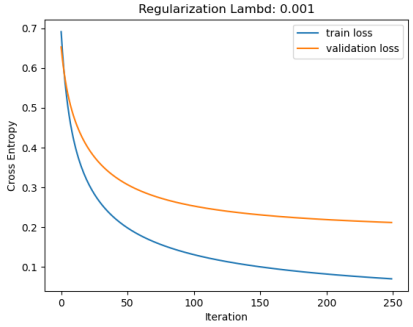


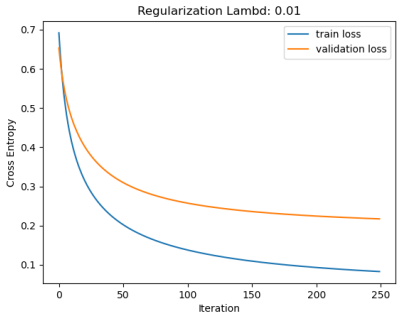


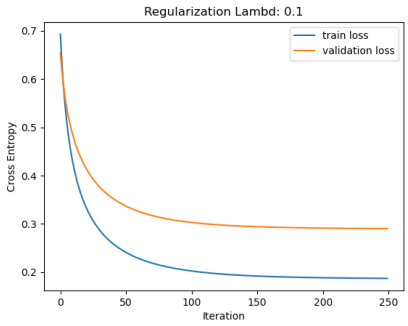


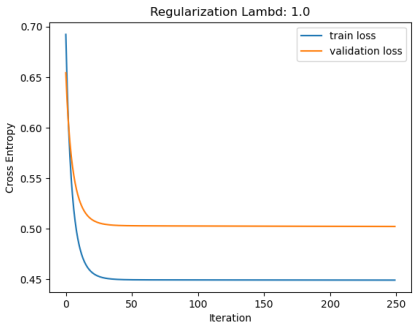
Train (to clarify, the values plotted are cross entropy (does not include penalty)











c) For train (mnist_train), train cross entropy increases as λ increases because the purpose of the regularizer is to prevent overfitting. Validation cross-entropy stays flat and then increases when $\lambda \geq 0.1$. This shows that the penalization factor does not boost the model's generalization significantly. We see a slight improvement with $\lambda = 0.1$ in validation accuracy. The reason that validation accuracy is not significantly increased may be due to λ being too small ($\lambda = 0.001$ / 0.01) or too large ($\lambda = 1.0$). Regularization changes the objective function and we may need more values of λ to find an optimal regularizer. I also observed that the convergence is faster with larger regularization terms.

For train_small, train cross entropy increases as λ increases because the regularizer penalizes to prevent overfitting. Validation cross-entropy decreases as λ increases, showing that a larger λ improves model generalization. However, the improvement in generalization is not reflected in higher validation accuracy. From 1st glance, this seems counterintuitive, but loss and accuracy are not strictly positively correlated, especially when doing binary classification.

Example: Suppose that I use 0.5 as the threshold. for binary classification
prior to regularization: predicted score = 0.7
After regularization: predicted score = 0.8
target = 1
In this case, accuracy won't increase but loss would decrease.

The two datasets show that when training dataset size is small, applying regularization can significantly prevent overfitting and make the model generalize better. When dataset is sufficiently large, the benefits of regularisation starts to diminish.

I'll pick $\lambda = 0.1$ for both datasets, because it decreases valid CE loss for train_small while maintaining the same accuracy and it increases valid accuracy for train dataset.

For train_small: Test CE = 0.549, Test Accuracy = 78%

For train: Test CE = 0.270, Test Accuracy = 92%

3.

b) Train CE: 0.345

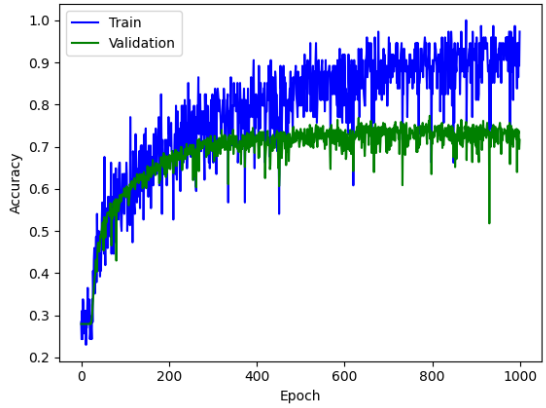
Error: 12.2%

Valid CE: 1.130

Error: 28.2%

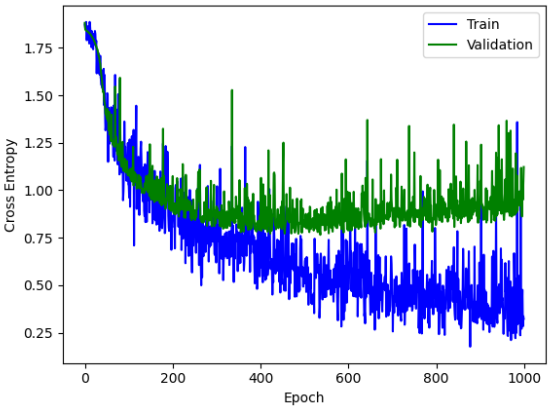
Test CE: 0.913

Error: 27.0%



We can see that as the iteration (epoch) increases, training CE keeps decreasing and training accuracy keeps increasing. However, validation CE keeps decreasing and validation accuracy keeps increasing until ~800th epoch. Then both metrics flatten out, with CE slightly going up again.

This shows that the model's training in the later epochs does not add to the model's generalization power. We can also say that the model has converged at ~350 - 400th epoch.



c) In general, smaller batch size leads to faster convergence.

Holding $\alpha = 0.01$, we can see from the 5 graph below:

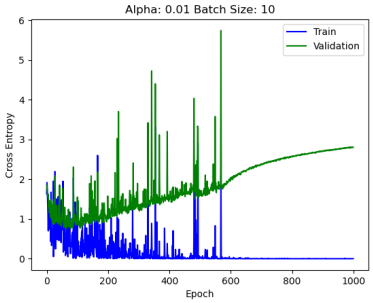
batch size = 10, the model converges at epoch < 100.

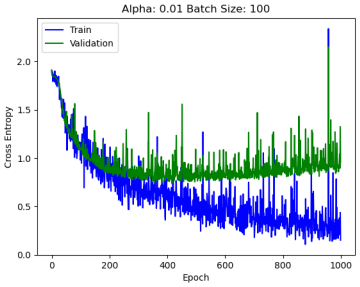
batch size = 100, model converges at epoch ≈ 200 ~ 300.

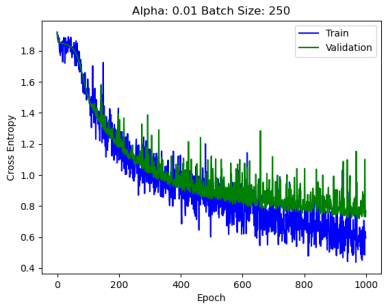
batch size = 250, model converges at epoch ≈ 900.

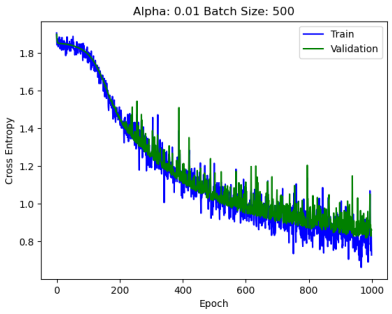
batch size = 500/1000, the model does not converge.

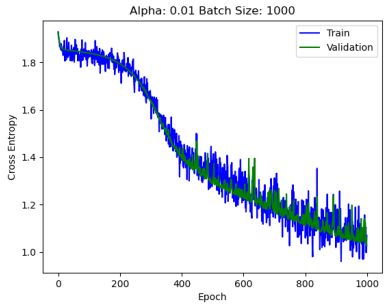
The intuition behind faster convergence is that the model starts learning by only seeing a small portion of the data. (weights move faster towards the optima)





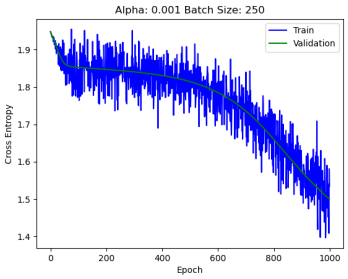


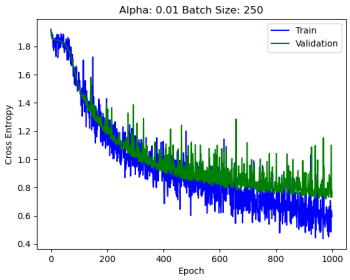


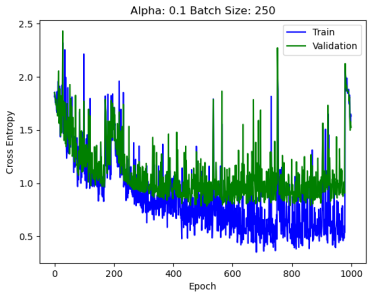


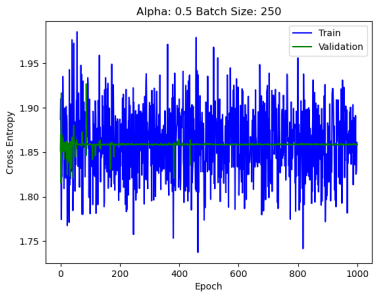
In general, larging learning rate up to a certain point leads to faster convergence.

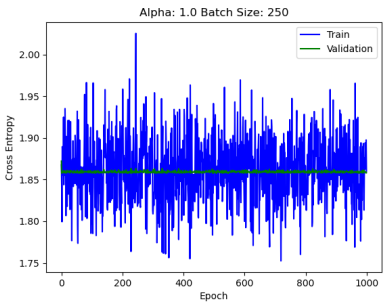
Holding batch size = 250









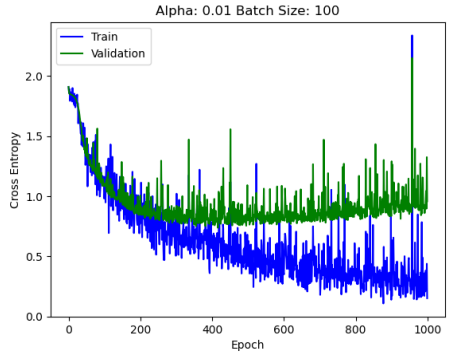


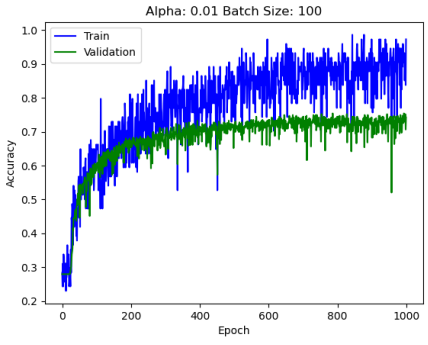
We can see that when $\alpha = 0.001$, the model does not converge because the gradient descent is too slow. When $\alpha = 0.01$, it converges near 900th epoch and when $\alpha = 0.1$, it converges much earlier. However, when α is too large, the model fails to train: the cross entropy for train/validation does not decrease over epochs.

To pick the best value, I would examine from 3 perspectives:

- Generalization (Is validation accuracy high?)
- Speed of Convergence
- CE curve (Is training happening? Do train/valid loss both decrease?)

- After filtering with the 3 criteria mentioned, I decided that the optimal hyperparameters are $\alpha = 0.01$, batch size = 100





- This is a good model because it converges quickly, has high validation accuracy and CE curve slopes downward and flattens.

d) I used the following number of units:

1st hidden layer: 16, 32, 64

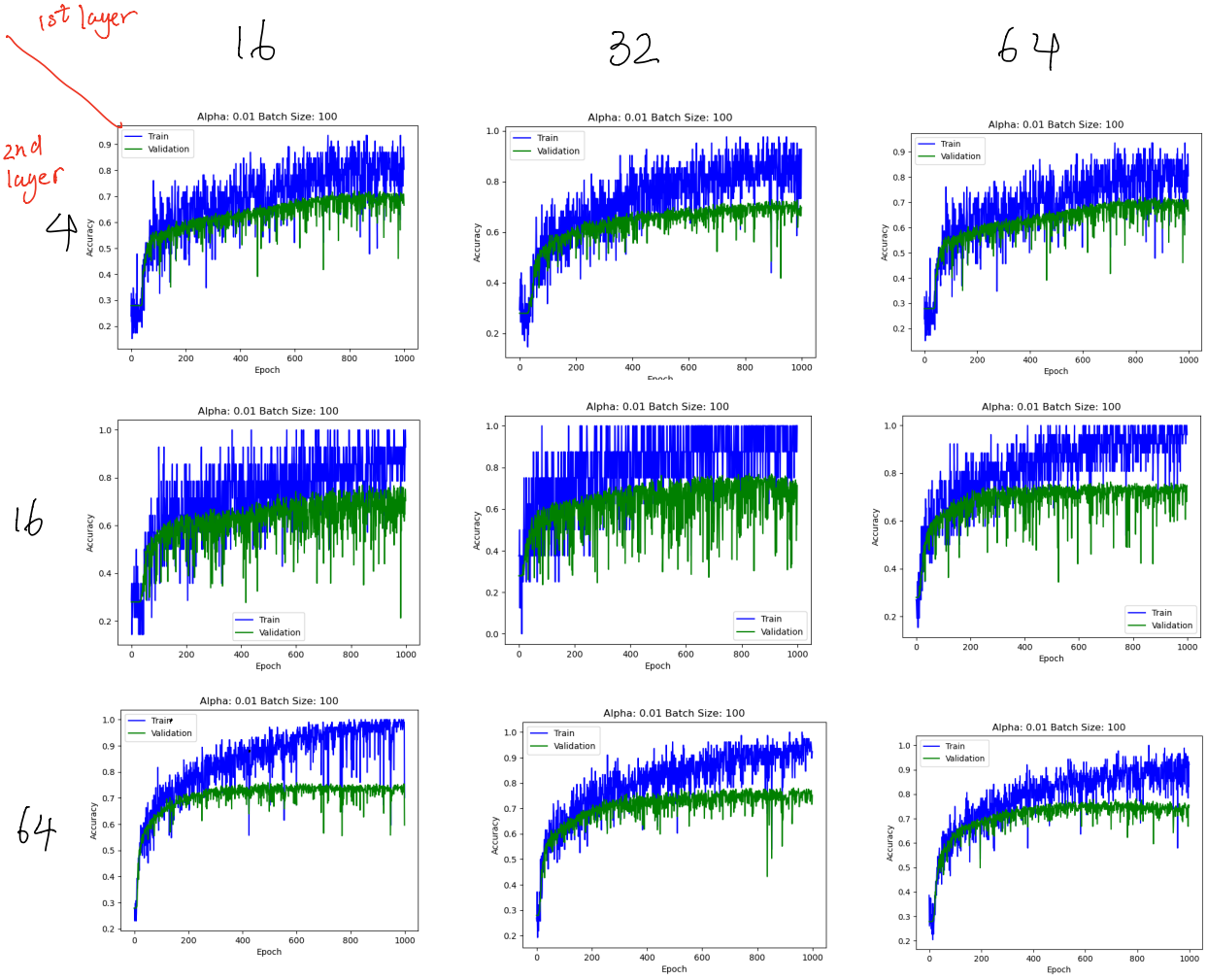
2nd hidden layer: 4, 16, 64

and a learning rate of 0.01 and a batch size of 100.

I created 9 combinations in total.

Observation 1: Convergence is faster when the 2nd layer has more units.

We can see from the graph below that convergence happens before 400th epoch when 2nd layer has 64 units, and convergence happens after 600th epoch when 2nd layer has 4 units.



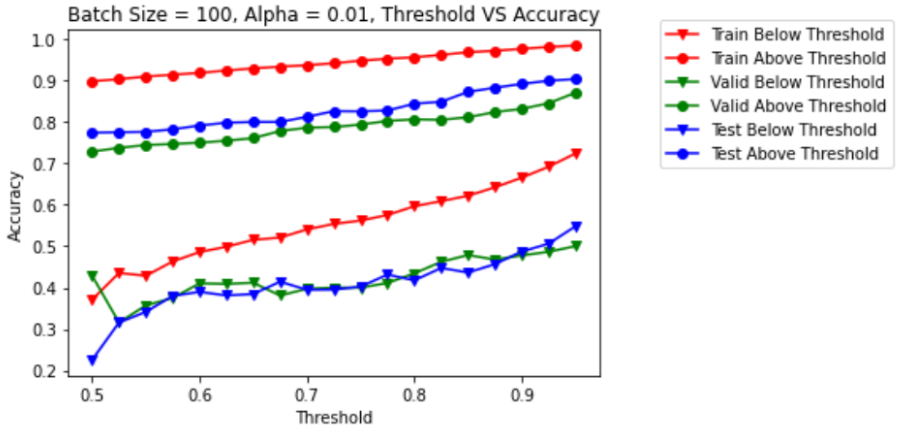
Observation 2: Generalization is better (higher validation accuracy) when the number of hidden units is high.
We see that when either layer has 64 units, validation accuracy converges above 70%. In contrast, when neither layer has 64 units, validation accuracy converges below 70%.

Observation 3: The model with the most hidden units see its validation accuracy decrease slightly in later epochs. This could be seen as overfitting, as the hidden units "overcomplicate" the problem. However, this observation is not fully confirmed because we need more models with high number of units to verify.

e)

* Trained with
$\alpha = 0.01$
batch = 100

This collection of images are from the test set where the top score is below 0.5. Out of these 31 examples, only 9 are predicted correctly. For the entire test set, 73% are identified correctly. This shows that the classifier is much less confident when top score is below a threshold. This is intuitive because the score somewhat resemble a probability. High scores indicate a high level of confidence. Visually with human judgement, it is difficult to identify those emotions correctly, as the emotion looks very ambiguous.



To further confirm my hypothesis, I looked at accuracy across train, valid and test datasets and divided the predictive accuracy by thresholds between 0.5 - 0.99.
We can see from the chart that: As threshold increases, the predictive accuracy increases for all 3 datasets.
Also, the accuracy gap between those with top score above the threshold and those below the threshold is consistent at all levels. This confirms the observation that higher top score implies more confidence in the model's predictive power.