Deep Learning in Data Science, DD2424

Short report on lab assignment 1 bonus
One-layer neural network with multiple outputs

Hamid Dimyati

1. Introduction

The focus of this report is to implement several methods to achieve higher accuracy on the same datasets as the main report on lab assignment 1. Some methods that will be presented in this report cover random shuffling data before each epoch, implementing learning rate decay, hyperparameter grid search, and applying to the full data. Further, the SVM multi-class loss function is also used, replacing the entropy-cross loss function to know how good it is in improving the performance.

The tool used for this bonus assignment is also `python 3.7.3`, along with several packages including `numpy 1.16.4`, `pandas 0.25.3`, and `matplotlib 3.1.0`.

2. Results and discussion

This experiment runs a single-layer neural network with different hyperparameters, as provided in Table 1.

Table 1. Hyperparameter settings for four models

Model	**Hyperparameters**			
	Total epoch	**Batch size**	**Lambda**	**Eta**
Model 1	40	100	0	0.1
Model 2	40	100	0	0.001
Model 3	40	100	0.1	0.001
Model 4	40	100	1	0.001

The datasets mainly used for this assignment are CIFAR-10 batch 1 for training, CIFAR-10 batch 2 for validation, and CIFAR-10 test batch for testing. The detail results for the first two methods are presented in Table 2:

Table 2. Accuracy improvement as a result of different treatment on the dataset

Model	**Treatment**	**Accuracy improvement**			**Accuracy gap (train – test)**
		Training	**Validation**	**Testing**	
Model 1	No treatment	40.85%	27.87%	27.72%	13.13%
	Random shuffling	58.94%	34.58%	34.86%	24.08%
	Learning rate decay	57.32%	32.56%	33.01%	24.31%
	Combined	59.83%	32.86%	33.09%	25.74%

Model 2	No treatment	45.40%	38.14%	38.76%	6.64%
	Random shuffling	41.08%	36.95%	37.59%	4.09%
	Learning rate decay	42.90%	38.11%	38.43%	4.47%
	Combined	43.00%	37.97%	38.57%	4.43%
Model 3	No treatment	44.75%	38.37%	39.20%	5.55%
	Random shuffling	40.92%	37.17%	37.95%	2.97%
	Learning rate decay	42.74%	38.15%	38.85%	3.89%
	Combined	42.51%	38.25%	38.67%	3.84%
Model 4	No treatment	39.93%	36.41%	37.54%	2.39%
	Random shuffling	39.78%	36.25%	37.34%	2.44%
	Learning rate decay	40.22%	36.35%	37.50%	2.72%
	Combined	40.11%	36.42%	37.47%	2.64%

2.1. Effect of random shuffling

The first technique to use is employing random shuffling data before each epoch. Table 1 summarizes the result comparison between training a single neural network with and without random shuffling data to know how much improvement caused by employing this technique to the model accuracy for all parameter settings. Although it does not guarantee an accuracy improvement, it could improve the generalization capability of the model by lowering the accuracy gap between training and testing accuracy.

As the random shuffling contributes much improvement to the Model 1 accuracy, this technique could help the cost reduction progress significantly, as seen in Figure 1.

Model 1

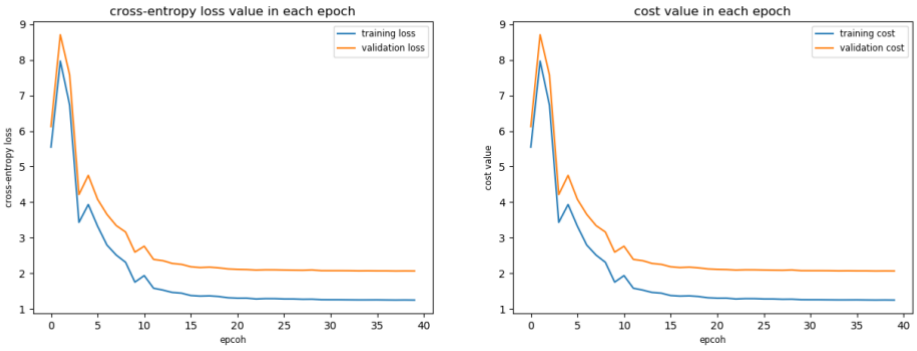

Figure 1. Cross-entropy loss (left) and cost (right) value by epoch from Model 1 employing random shuffle

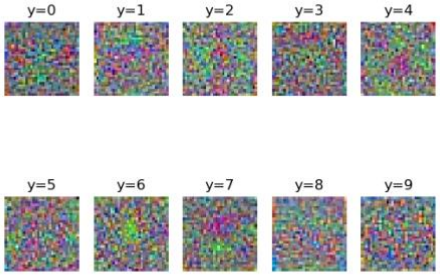

Figure 2. Image representation of learned weight from Model 1 employing random shuffle

Model 2

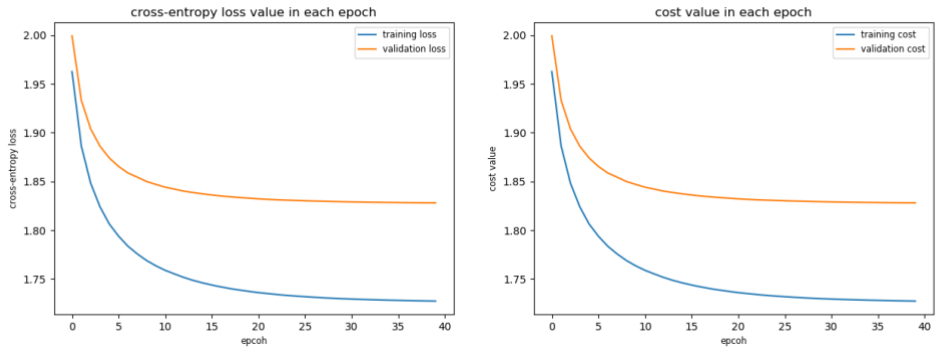

Figure 3. Cross-entropy loss (left) and cost (right) value by epoch from Model 2 employing random shuffle

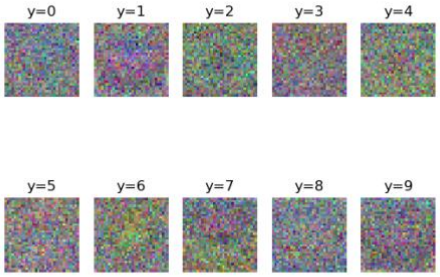

Figure 4. Image representation of learned weight from Model 2 employing random shuffle

Model 3

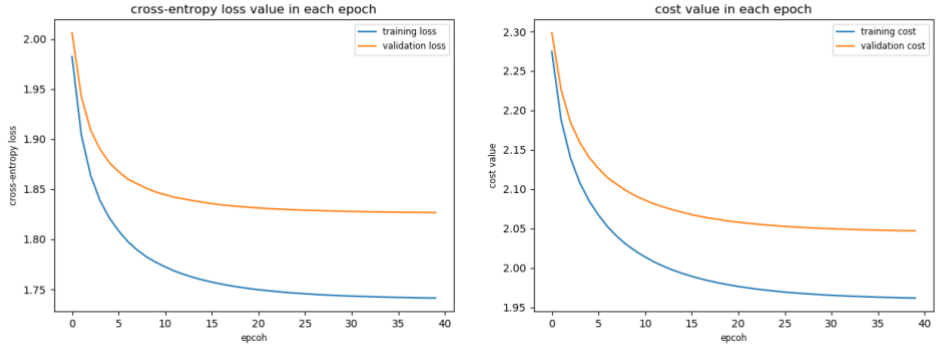

Figure 5. Cross-entropy loss (left) and cost (right) value by epoch from Model 3 employing random shuffle

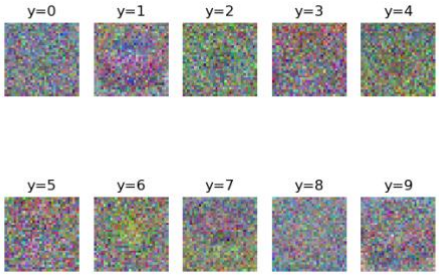

Figure 6. Image representation of learned weight from Model 3 employing random shuffle

Model 4

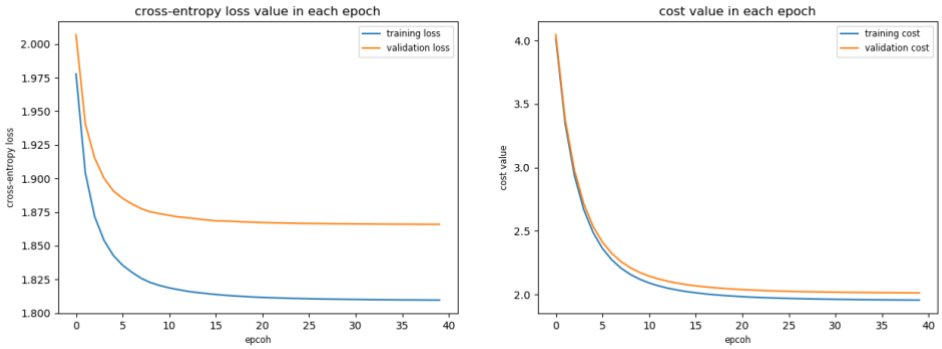

Figure 7. Cross-entropy loss (left) and cost (right) value by epoch from Model 4 employing random shuffle

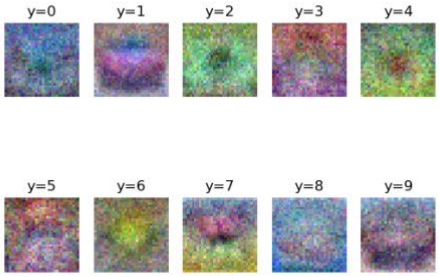

Figure 8. Image representation of learned weight from Model 4 employing random shuffle

2.2. Learning rate decay

The second technique to use is employing a learning rate decay. In detail, the technique tries to reduce the eta by 0.9 after every ten epochs. Table 2 summarizes the result after implementing a learning rate decay. Employing this technique could slightly enhance the model accuracy in the cost of model generalization power, which somewhat decreases. However, combining both techniques could produce the best model (higher accuracy and generalization).

Model 1

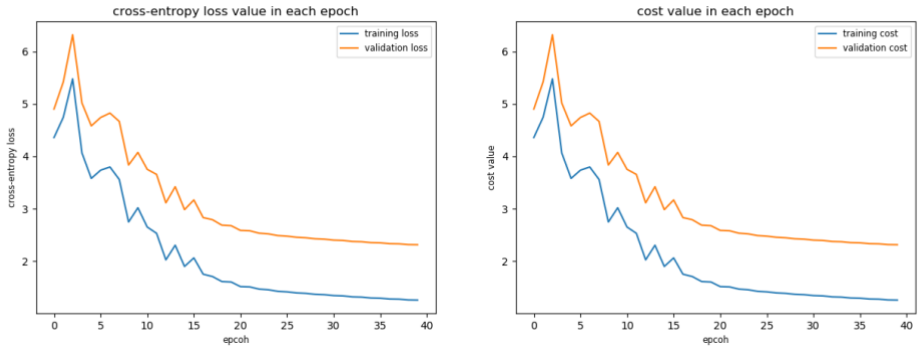

Figure 9. Cross-entropy loss (left) and cost (right) value by epoch from Model 1 employing learning rate decay

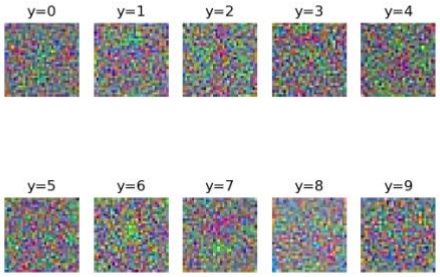

Figure 10. Image representation of learned weight from Model 1 employing learning rate decay

Model 2

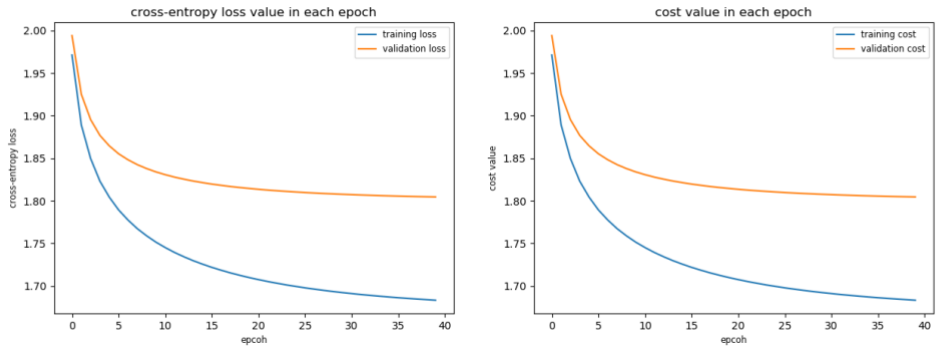

Figure 11. Cross-entropy loss (left) and cost (right) value by epoch from Model 2 employing learning rate decay

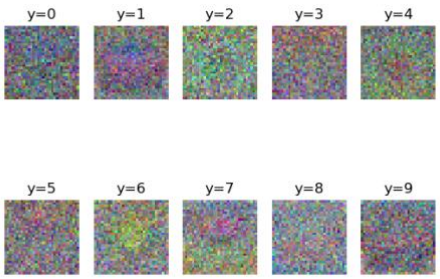

Figure 12. Image representation of learned weight from Model 2 employing learning rate decay

Model 3

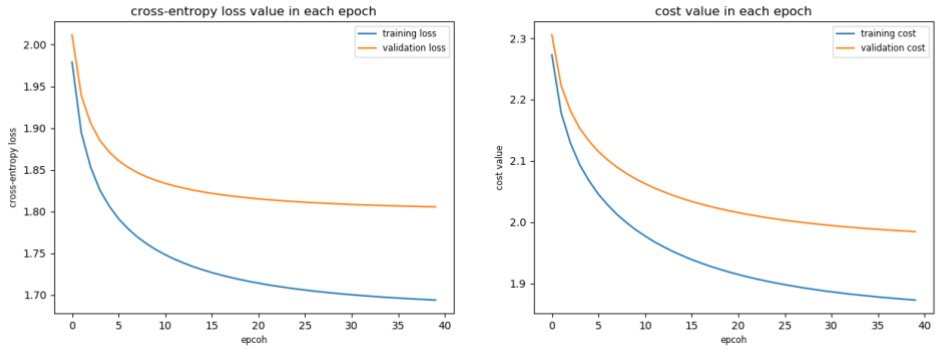

Figure 13. Cross-entropy loss (left) and cost (right) value by epoch from Model 3 employing learning rate decay

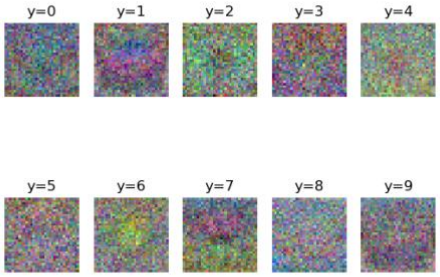

Figure 14. Image representation of learned weight from Model 3 employing learning rate decay

Model 4

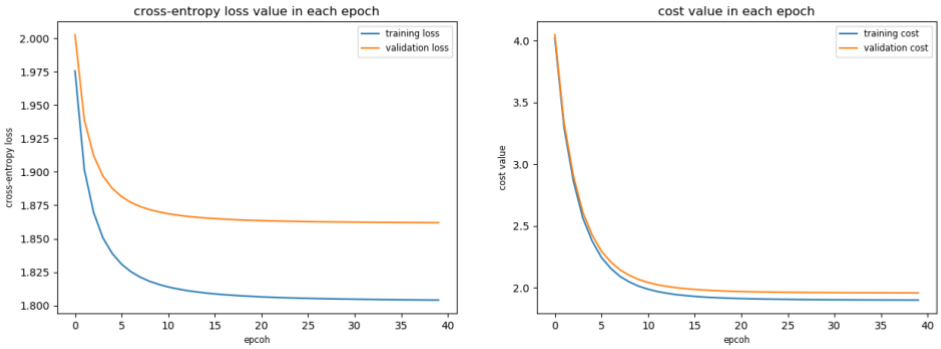

Figure 15. Cross-entropy loss (left) and cost (right) value by epoch from Model 4 employing learning rate decay

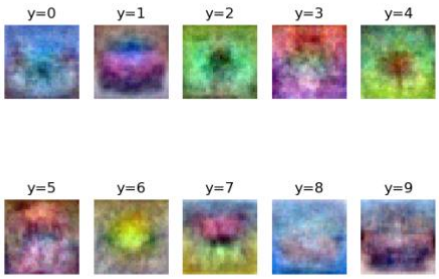

Figure 16. Image representation of learned weight from Model 4 employing learning rate decay

2.3. Parameter grid search

The third technique to employ is finding the best hyperparameters combinations through a grid search technique. Here I define different values for each parameter and run the learning algorithm with all combination of the three parameters: lambda = [0, 0.01, 0.1, 1], n_batch = [10, 50, 100], eta = [0.001, 0.01, 0.1, 1], and n_epoch = 40. Thus, there are 48 models, and their result summary by averaging all accuracies per different parameter value is provided in Table 3, Table 4, and Table 5. Table 3 suggests that a small value of the learning rate (0.001) will produce an excellent model. While in Table 4, one should take care to conclude whether to use which small lambda (0.01 or 0.1), since there is a trade-off between the testing accuracy and generalization capability. Nevertheless, too high lambda could also mean increasing the bias error. Lastly, Table 5 suggests 100 as the best mini-batch size.

Table 3. Accuracy using different learning rates

Eta	**Average Accuracy**			**Accuracy gap (train – test)**
	Training	**Validation**	**Testing**	
0.001	44.39%	37.92%	38.56%	5.83%
0.01	47.81%	36.81%	37.26%	10.55%
0.1	43.14%	30.86%	31.27%	11.87%
1	30.86%	22.07%	22.67%	8.19%

Table 4. Accuracy using different lambda values

Lambda	**Average Accuracy**			**Accuracy gap (train – test)**
	Training	**Validation**	**Testing**	
0	54.63%	33.25%	33.50%	21.13%
0.01	43.81%	33.71%	34.24%	9.57%
0.1	37.06%	32.28%	32.73%	4.33%
1	30.70%	28.43%	29.29%	1.42%

Table 5. Accuracy using different batch sizes

Batch size	Average Accuracy			Accuracy gap (train – test)
	Training	Validation	Testing	
10	40.70%	30.27%	30.81%	9.89%
50	41.99%	32.40%	32.88%	9.11%
100	41.96%	33.07%	33.63%	8.33%

Out of 48 models, a model X (lambda=0.1, n_batch=100, n_epoch=40, eta=0.001) has the best performance: 42.91% training accuracy, 38.25% validation accuracy, and 38.94% testing accuracy. Figure 17 and Figure 18 depicts the cross-entropy loss-cost value per epoch and the image from model X.

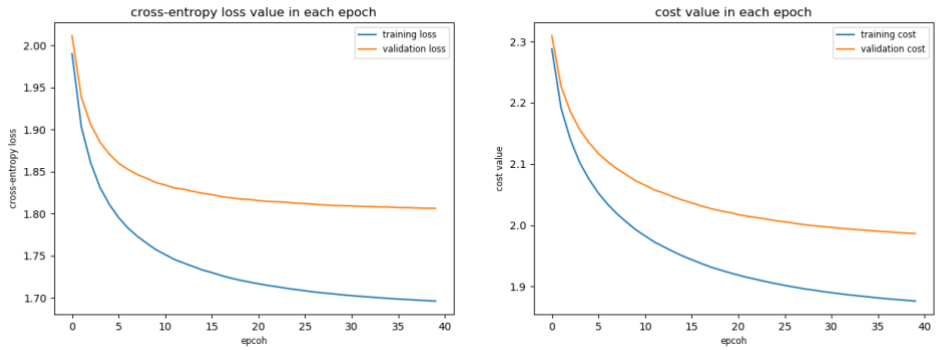

Figure 17. Cross-entropy loss (left) and cost (right) value by epoch from Model X

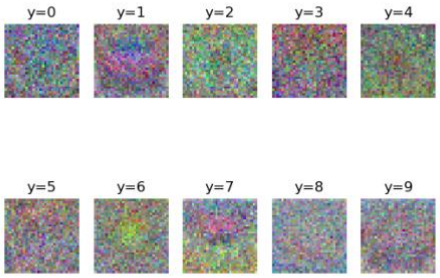

Figure 18. Image representation of learned weight from Model X

2.4. Applying to full datasets

The best model in 2.3 (Model X) is then applied to learn the full dataset. The datasets used for this section are CIFAR-10 batch 1 until batch 5, excluding the last 1,000 samples for training. The last 1,000 from CIFAR-10 batch 5 is used for validation, and CIFAR-10 test batch for testing. Interestingly, learning more data with the same parameters could achieve better performance. This model Y achieved 42.24% training accuracy, 41.20% validation accuracy, and 40.89% testing accuracy. This results mean that the model Y has better generalization capability with 1.35% accuracy gap between training and testing compared to model X (3.97% accuracy gap). It is understandable since adding more data could improve the generalization of the model and avoid overfitting. Figure 19 and Figure 20 depicts the cross-entropy loss-cost value per epoch and the image from model Y.

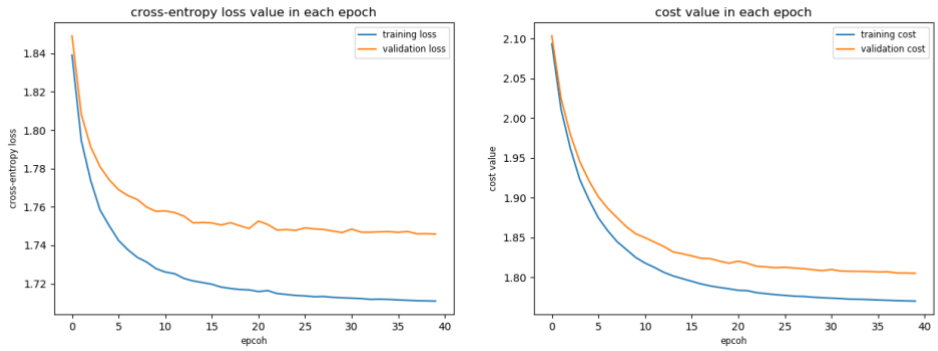

Figure 19. Cross-entropy loss (left) and cost (right) value by epoch from Model Y

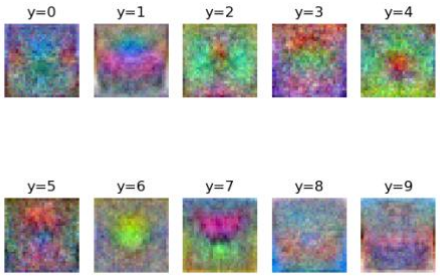

Figure 20. Image representation of learned weight from Model Y

2.5. SVM multi-class loss

The last experiment is to replace the cross-entropy loss with SVM multi-class loss function using the same parameters and datasets as Model X in 2.3. This model Z achieved 46.60% training accuracy, 36.20% validation accuracy, and 36.99% testing accuracy. It is learned that using SVM multi-class loss for this case is not as good as using cross-entropy loss. Model Z also has worse generalization capability with a 9.61% accuracy gap between training and testing compare to Model X (3.97% accuracy gap) in 2.3. Figure 21 and Figure 22 depicts the cross-entropy loss-cost value per epoch and the image from model Z.

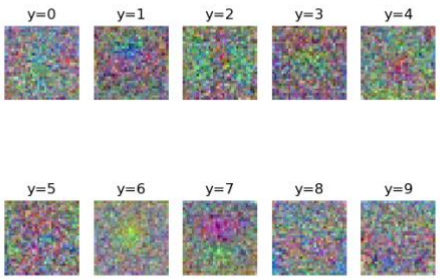

Figure 21. Cross-entropy loss (left) and cost (right) value by epoch from Model Z

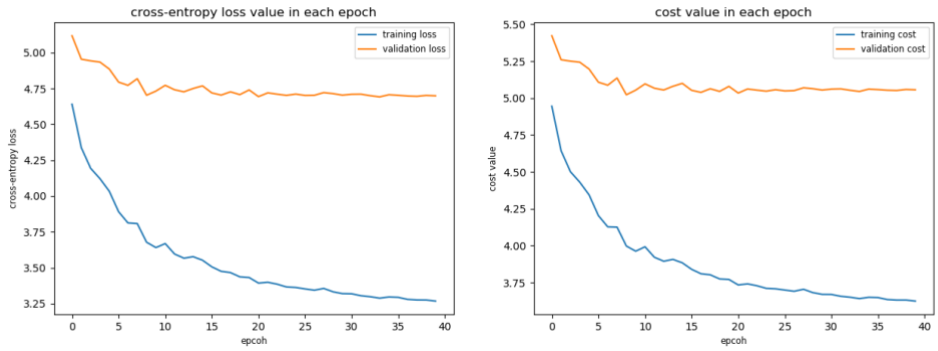

Figure 22. Image representation of learned weight from Model Z

3. Final remarks

From this assignment, it could be learned that performing both random shuffling and learning rate decay could increase the model performance. Moreover, the grid search technique could also be useful to find out the best hyperparameters through setting the learning rate, lambda, and batch size as small enough. In specific, increasing the lambda value will affect on reducing training accuracy but better generalization capability. Another way to improve the model generalization is to add more datasets. Lastly, besides cross-entropy loss, building a neural network could also use the SVM multi-class loss function. However, for this case, training a single neural network using SVM multi-class loss function does not perform as good as using the cross-entropy loss function.