CS699 DeepLearning HW3

mijanur palash, mpalash@purdue.edu

February 2020

1 Q0:

(1) Student interaction with other students / individuals:
No (a) I have copied part of my homework from another student or another person (plagiarism).
Yes (b) Yes, I discussed the homework with another person but came up with my own answers. Their name(s) is (are): Siddhartha Das
(c) No, I did not discuss the homework with anyone

(2) On using online resources:
NO (a) I have copied one of my answers directly from a website (plagiarism).
Yes (b) I have used online resources to help me answer this question, but I came up with my own answers (you are allowed to use online resources as long as the answer is your own). Here is a list of the websites I have used in this homework: stackoverflow.com
(c) I have not used any online resources except the ones provided in the course website

2 Q1.1- Theoretical Part:

2.1 Q.1.1.1:

The noisy gradient needs to be un-biased. The Robbins-Monro stochastic approximation algorithm:

$$\theta_t = \theta_{t-1} - \gamma_t \times h_t \times (\theta_{t-1})$$

here

$$E[h_t((\theta)] = h(\theta)$$

and γ_t is a learning rate. without loss of generality, start from θ_0, sampling Z_n i.i.d with average E[z]=z. Now we perform:

$$\theta_t = \theta_{t-1} - \gamma_t \times (\theta_{t-1} - z_t)$$

now when $\gamma_t = \frac{1}{t}$ then

$$\theta_t = \frac{1}{t} \times \sum_{i=1}^{t} z_i$$

which asymptotically converges to E[z].

But if noise is not unbiased we get

$$\theta_t = \theta_{t-1} - \gamma_t \times (\theta_{t-1} - (z_t + b_t))$$

now when $\gamma_t = \frac{1}{t}$ then

$$\theta_t = \frac{1}{t} \times \sum_{i=1}^{t} (z_i + b_i)$$

which asymptotically converges to E[z]+E[b], but we needed it to converge to E[z]. So with a bias in noise we will not converge to the right point.

2.2 Q.1.1.2

For some learning rate decay function γ_t we will not converge. sampling Z_t i.i.d with average $E[z_t] = z$:

$$\theta_t - z = (\theta_0 - z) \prod_{i=1}^{n} t \times (1 - \gamma_i) + \sum_{j=1}^{t} (\gamma_j) \prod_{i=j+1}^{t} (1 - \gamma_i)(z_j - z)$$

We need both terms of this equation to converge to our θ_n to converge to z. First for our approximation to converge we assume the initial condition θ_0 to be forgotten over time. Suppose $\gamma_n = O(1)$, then

$$log \prod_{i=1}^{n} t \times (1 - \gamma_i)$$

can be written as $-\sum_{i=1}^{n} \gamma_i$

Now,

$$\lim_{n \to \infty} \prod_{i=1}^{n} (1 - \gamma_i) = 0$$

needs

$$\lim_{n \to \infty} -\sum_{i=1}^{n} \gamma_i = -\alpha$$

or

$$\lim_{n \to \infty} \sum_{i=1}^{n} \gamma_i = \alpha$$

this is the first condition.

We want that the MSE or our estimator converges to zero as n goes to infinity. We want

$$\lim_{n\to\infty} E[(euclidean\_norm(\theta_n - z))^2] = 0$$

after several mathematical steps a upper bound can be established for this MSE as

$$exp(-\mu \sum_{j=m+1}^{n} \gamma_j) \sum_{i=1}^{m} (\gamma_i)^2 + \frac{\gamma_m}{\mu}$$

we get MSE =0 at $n \to \infty$ when we assume $m \propto n$ and $\lim_{n\to\infty} \gamma_n = 0$ and $\lim_{n\to\infty} \sum_{i=1} n(\gamma)^2 < \infty$

This is the second condition. Now if any of this condition does not meet the model will not converge and MSE would not be zero.

2.3 Q.1.1.3

Loss flatness means large poodle of parameter space. This means that the loss remain similar for a wide variation of model parameter in many dimension. So we can reach to that loss level from different direction i.e. different values of model parameters. Generally local minima found in SGD tends to be wider hence generalize better. It also faster to reach one. In case of global minima it may be narrow and hard to reach.

2.4 Q.1.1.4

Early stopping helps us to reach better generalization of the model. If we run the fixed number of epoch there is a chance of over fitting if we run a large number of epoch, or model under performing if we do not run enough epoch.

We should not use training data because accuracy tends to be higher in training data and training accuracy and test accuracy does not always be the same or near.

2.5 Q.1.1.5

CNNs are sensitive to the image rotations. We learn filter values in correlation of the corresponding image patches. If we rotate image this correlations becomes irrelevant.

Max pooling helps in case of small rotation of the image. When we take max value over a certain area, max remain same over that area and the model will result in same output even though we may move within that area.

If we use a large image patch and takes max over it, we lose information. This lose of information here can never be regained later.

2.6 Q.1.1.6

Min-pooling is never applied to CNNs with ReLU activations. This is because when we perform the convolution between image patch and the filter, if the filter has negative values in its parameters space, and after hadamard product and summation, a negative value arises, min pooling will pick the lowest and negative value. After ReLU this negative will become zero. A parameter value with zero in any dimension will not learn anything in that dimension and if most of the values of the parameter dimensions are zero, the model will die.

3 Q.1.2.3 Gradient Descent Variants:

3.1 Task 1a: Minibatch Stochastic Gradient Descent (SGD)

3.1.1 Task 1a.4-tables:

BatchSize$\|\|$*LearningRate*	1e-3	1e-4	1e-5
100	0.001	0.052	0.33
500	0.02	0.2	0.75
3000	0.14	0.55	1.96
5000	0.23	0.7	2.74
full-batch	0.43	0.6	5.3

Table 1: Loss

BatchSize$\|\|$*LearningRate*	1e-3	1e-4	1e-5
100	1	0.99	0.91
500	1	0.94	0.79
3000	0.96	0.85	0.55
5000	0.93	0.81	0.44
full-batch	0.86	0.6	0.13

Table 2: Training Accuracy

BatchSize$\|\|$*LearningRate*	1e-3	1e-4	1e-5
100	0.96	0.94	0.89
500	0.95	0.91	0.78
3000	0.93	0.83	0.55
5000	0.91	0.81	0.44
full-batch	0.84	0.6	0.14

Table 3: Testing Accuracy

3.1.2 Task 1a.5-Observation:

1. Accuracy decreases with reduced learning rate
2. Accuracy decreases with increase of mini-batch size, and no-mini-batch gradient descent performs worst. When we use small mini-batch the model gets more random noise and hence going over a puddle becomes more possible and eventually finds a more wide puddle/flatness. The bigger batch size becomes, the randomness is averaged more and more and it gets more stuck in the puddle

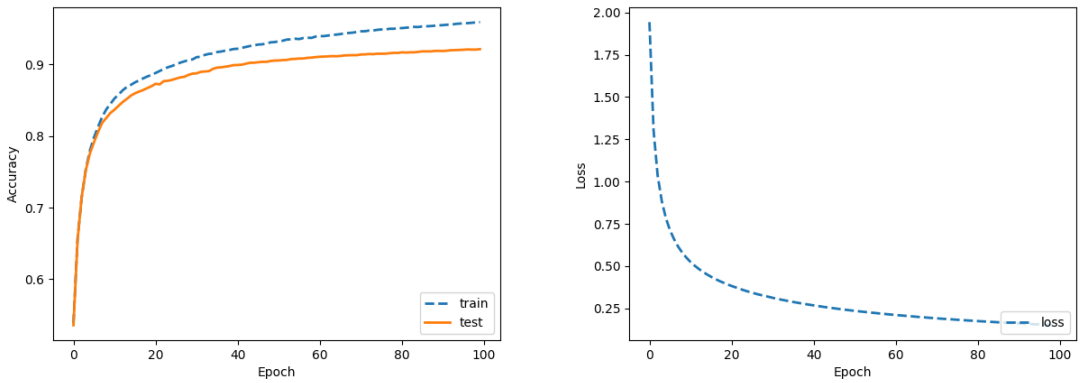
3.1.3 Task 1a.6-GD vs minibatch SGD:

I put the result of the regular GD as full-batch in the table 1,2 and 3 with different mini batch sizes. We can observe that regular GD performs worse than any mini-batch SGD in all learning rates.

3.2 Task 1b: Adaptive Learning Rate Algorithms

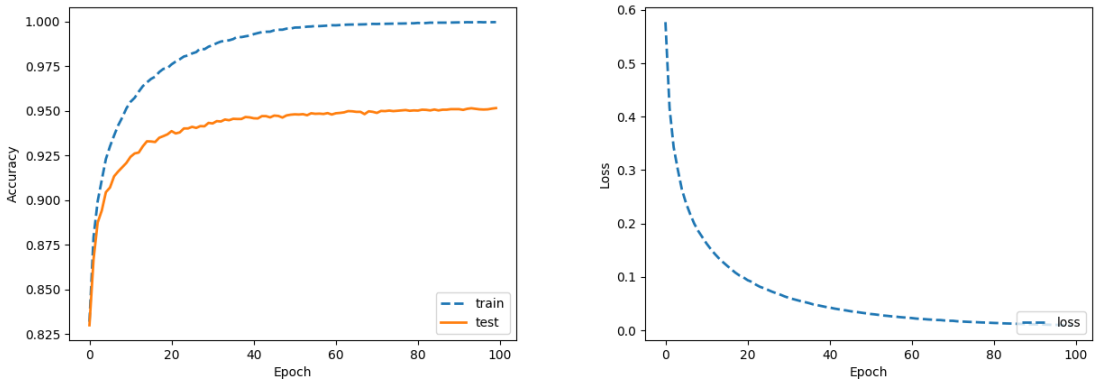
3.2.1 Task 1b.3- plots:

The accuracy and loss graphs of SGD, Momentum, Nestorev and Adam adaptive learning rate networks are given here:



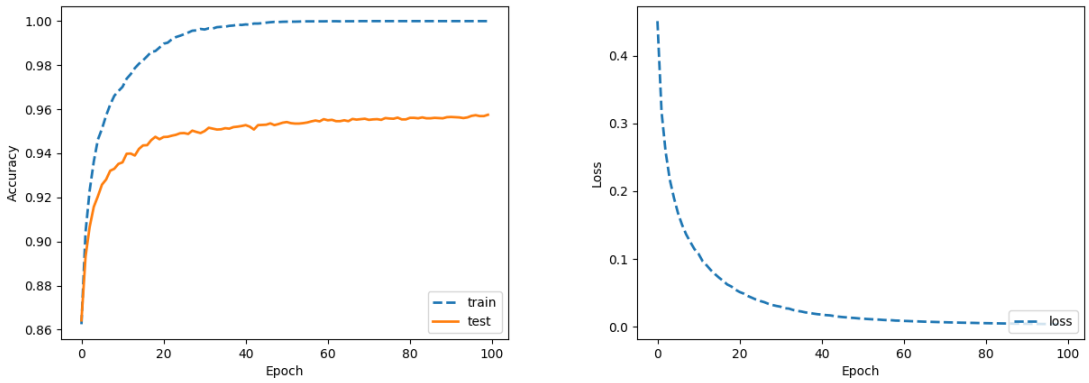
(a) Accuracy (b) Loss

Figure 1: Accuracy and loss in SGD



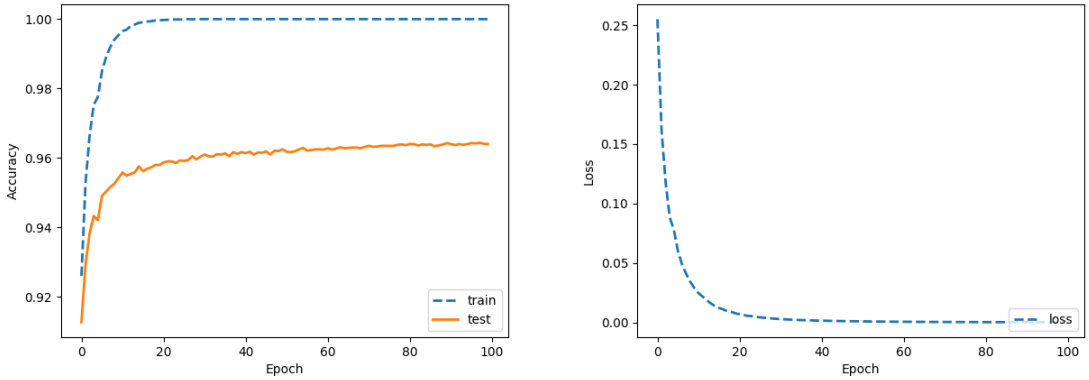
(a) Accuracy (b) Loss

Figure 2: Accuracy and loss in Momentum



(a) Accuracy (b) Loss

Figure 3: Accuracy and loss in Nesterov



(a) Accuracy (b) Loss

Figure 4: Accuracy and loss in Adam

3.2.2 Observation:

1. Accuracy: SGD<Momentum<Nesterov<Adam.
2. Adam has the highest accuracy while SGD is the lowest.

3.2.3 Comparison between the Algorithms:

Stochastic gradient descent computes the gradient of the cost function with respect to the parameters and update the parameter value for each training example. Mini-batch SGD update parameters for every mini-batch of n training examples.

Momentum helps SGD to accelerate in the relevant direction and dampen fluctuations. It adds a fraction of last gradient to the current gradient. This increases change the value of parameters of dimensions whose gradients point in the same directions and reduces for dimension whose gradients change directions. This ensures faster convergence and reduced oscillation.

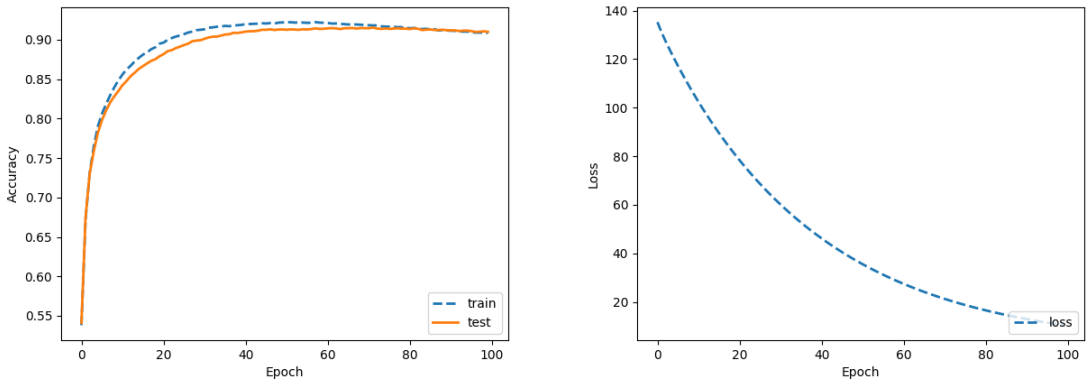
In nesterov we look ahead for a future point based on the previous gradient and computes gradients for that point and uses parts of this with current gradient to update parameter values. It results in better responsiveness and prevent to reach to the to fast.

Adam uses the squared gradients to scale the learning rate and it uses moving average of the gradient. Adam uses estimations of first and second moments of gradient to adapt the learning rate for each weight of the neural network

4 1.2.4: Regularization

4.1 Task 2a: L2-Regularization

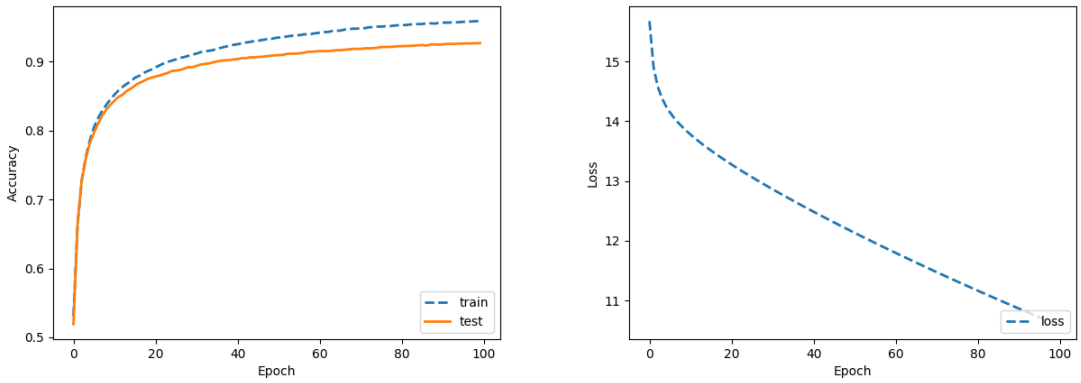
4.1.1 Task 2a.4: plots



(a) Accuracy

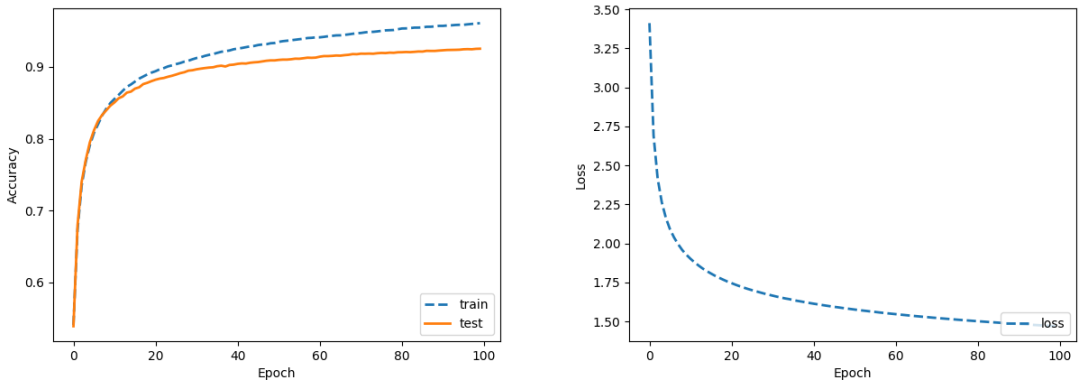
(b) Loss

Figure 5: Accuracy and loss in $\lambda = 1$



(a) Accuracy (b) Loss

Figure 6: Accuracy and loss With $\lambda = 0.1$



(a) Accuracy (b) Loss

Figure 7: Accuracy and loss in $\lambda = 0.01$

4.1.2 Task 2a.5-Observations:

1. Difference between training and test accuracy decreases with increasing λ.
2. Regularizer prevents overfitting. Thats why increased λ decreases the gap between the training and test accuracies.

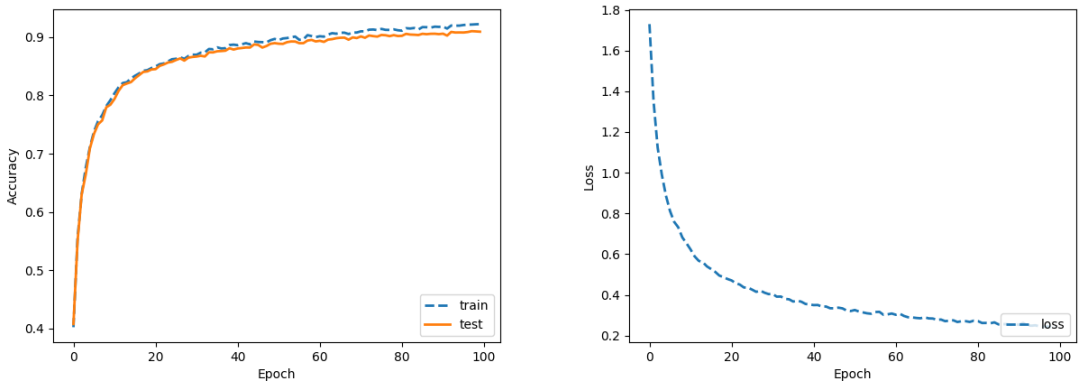
5 Task 3: Implement a CNN

5.1 Task 3.1:- Describe the neural network architecture:

So this neural network starts with two convolution layer and then it has two fully connected layer. First convolution layer has filter of size 5x5 and stride 1 while second convolution layer has same filter size and stride. ReLU is used as activation function in the hidden layers while output uses softmax. Max pooling of 2 has been used in the convolution layer.

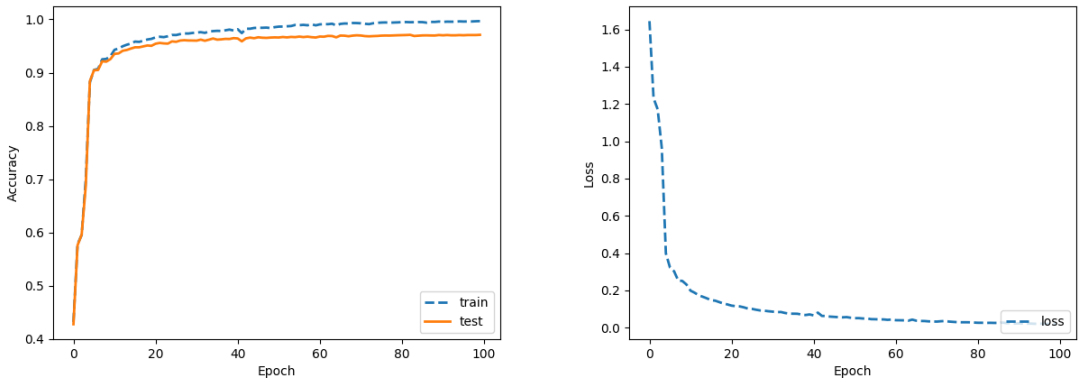
The model has first convolution layer. The filter has 5x5 size with stride 1 and no padding and traverse the image of 28x28. So we get 24x24 neurons in the output of it and which after applying maxpooling of 2 becomes 12x12.We have 10 of this 12x12 neurons for 10 different filters. We apply ReLU on this neurons and then apply second convolution on them. We get 8x8 neurons in the output and after maxpooling of 2 we get 4x4. We have 20 such filters so we get 16x20=320 neurons and then we feed these two the fully connected layer 1 which outputs 100 neurons and then fully connected layer 2. In the output layer we get 10 different neurons corresponding to 10 different labels.

5.2 Task 3.2 (kxk filter):



(a) Accuracy (b) Loss

Figure 8: Accuracy and loss in filter 3x3 and stride 3



(a) Accuracy

(b) Loss

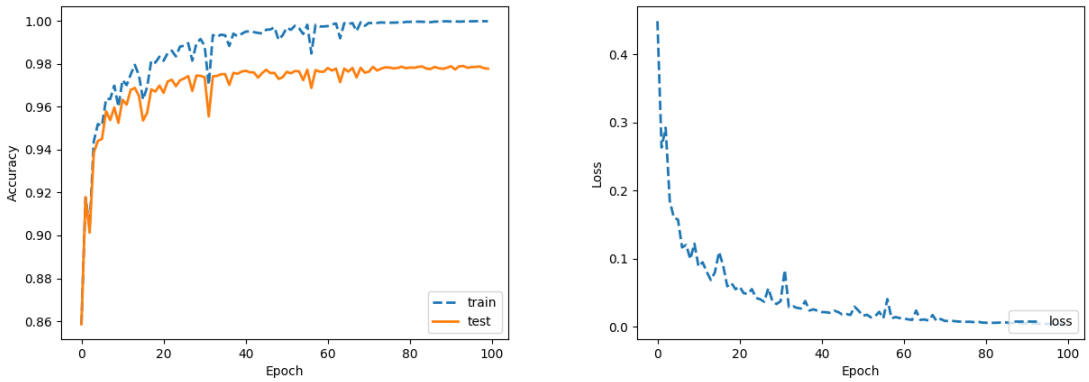
Figure 9: Accuracy and loss in 14x14 and stride 1

5.2.1 Observations:

1. filter 14x14 with stride 1 gives better accuracy then 3x3 with stride 3. This is because stride 3 makes bigger jumps over the data points which misses many features.
2. filter 14x14 takes much longer time to finish
3. with same stride 3x3 would do better than 14x14. It will finish faster and will give better accuracy.

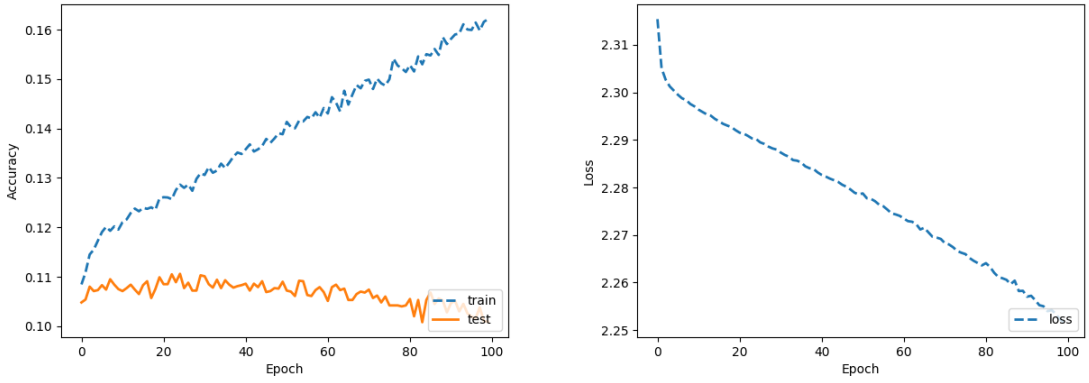
5.3 Task 3.3-Label suffling:

5.3.1 3.3.a: Plots:



(a) Accuracy (b) Loss

Figure 10: Accuracy and loss when no shuffling the labels (mini batch size 128, 5x5 filter, stride 1, no padding)



(a) Accuracy (b) Loss

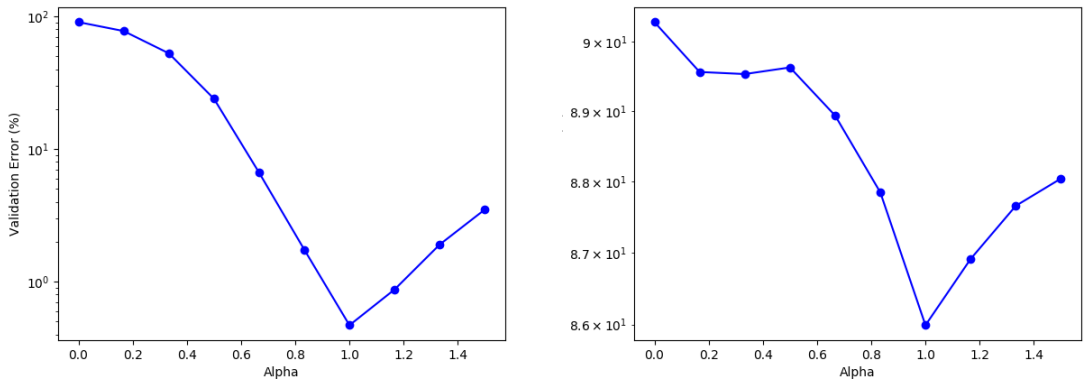
Figure 11: Accuracy and loss when shuffling the labels

5.3.2 3.3.b

Ability of this neural network to overfit this data: Deep neural network is a universal approximator. So given enough training examples and enough iterations it will be able to approximate any function. But this model we are using is not complex enough and we are not running enough iterations over enough examples so it does not overfits the data.

CNN can't naturally understands the images and will not try to classify all hand written 0 as same digit.

5.3.3 3.3.c- flatness plots:



(a) Not suffled (b) Shuffled

Figure 12: flatness of the model

These model are not generalized. It found a very narrow puddle (specific model) so the solution space is not wide/flat. The solution space is not generalized over a wide variety of parameter values.