Advanced Machine Learning – ELL888

Assignment 1

Anish Mahishi Adarsh Onkar
February 15, 2018

1 PROBLEM STATEMENT

Train a multi-layer perceptron or a feed-forward network to classify the images from EMNIST dataset as belonging to one of the nine alphabet classes. Alphabets chosen are (a-i).

2 CONTENTS

3 APPROACH

- We have the input csv file contains 784 feature vectors which contain values between [0, 255] and a class label between [1, 26].
- Since we need only 9 alphabets, we filter the dataset which has class labels between [1, 26].
- We scale the feature vectors between [0, 1] and then subtract the mean and divide by variance.
- Also, for the class labels, we create one-hot encoding vectors for the output layer of the network.
- In all our experiments, we use softmax function at the output layer
- We use 3 activation functions (sigmoid, tanh, relu), 3 weight initialization methods (Gaussian, Xavier and He) and 2 cost functions (Mean-Squared Error (MSE) and Cross Entropy).
- After tuning, the learning rate is around the order of 1.0 - 5.0 for sigmoid with Gaussian initialization and 0.1 for sigmoid, relu with Xavier and He initialization.
- We use these learning rates for most of the time.
- We use mini-batch gradient descent for optimization. Mostly, we use 100 mini-batches because of its good convergence which is described in the experiment.

4 CODE DESCRIPTION

i. **Class Name : Main.py**
Description: First function to be called. Reads and preprocesses data.

Purpose: Reads data. Creates one-hot encoding. Normalizes Training and Testing Data. Initializes the hyper-parameters.
Refer: *Main.py*

ii. **Class Name : Network.py**
Description: The Neural Network is stored as list of HiddenLayer() and list of numpy arrays for weights (all parameters). We always use softmax as our output layer.
Purpose: Construct the Network, Initialize the Network, Perform forward and backward propagation using matrix multiplication of numpy arrays.
Refer: *Network.py*

iii. **Class Name : HiddenLayer.py**
Description: This class is an abstraction of 'Layers' file.
Purpose: Based on the arguments passed to it, it creates a BatchNormalLayer() or a NormalLayer().
Refer: *HiddenLayer.py*

iv. **Class Name : Layers.py**
Description: Contains the definition of a Layer in a Neural Network.
Purpose: Implements BatchNormalLayer() and NormalLayer()
Refer: *Layers.py*

v. **Class Name : Activations.py**
Description: Contains 3 classes for activation functions (1 each) i.e Relu, Sigmoid, Tanh. Each class contains a calc(x) and derivative(x). calc(x) contains function value at x.
Purpose: Contains the definition of various activation functions and their derivatives used in the assignment.
Refer: *Activation.py*

5 PERFORMANCE METRICS

i. **Mean-Squared Error**
ii. **Cross Entropy**

If flag isMSE = true in Main.py then Mean Square Error is used, else Cross Entropy is used by default. The comparison in performance between the two is shown in Section 6.3.
Implementation: Implemented as calc_loss in Main.py
Refer: *Main.py*

6 EXPERIMENTS

1. Sigmoid vs. Tanh

- ## Settings

Component	***Parameter***	***State***	***Interpretation***
Weight Initialization	`weight_initialization`	`'Gaussian'`	Gaussian Initialization
Network Structure	`layers`	`[(784, 'input'), (32, 'sigmoid'/'tanh'), (16, sigmoid/'tanh'), (9, 'output')]`	3-layer network with sigmoid or tanh activation functions
Epochs	`epochs`	`60`	60 epochs
Learning Rate	`learning_rate`	`1.0`	1.0
Mini-Batch size	`mini_batch_size`	`100`	100
Decay Rate	`lmbda`	`0.0`	No regularization
Regularization Technique	`reg`	`'None'`	No regularization
Loss Function	`IsMSE`	`False`	Cross-Entropy(Default)
Batch Normalization	`isBatchNorm`	`False`	No
Dropout	`isDropout`	`False`	No

- ## Observations

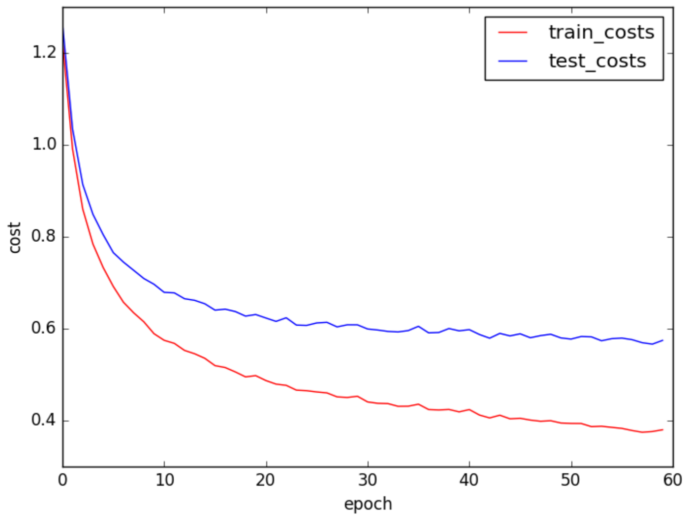


Figure 1: Sigmoid Loss

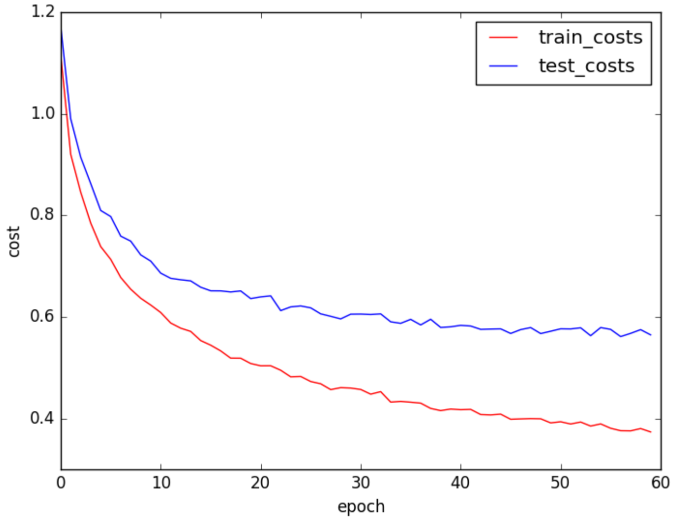


Figure 2: Tanh Loss

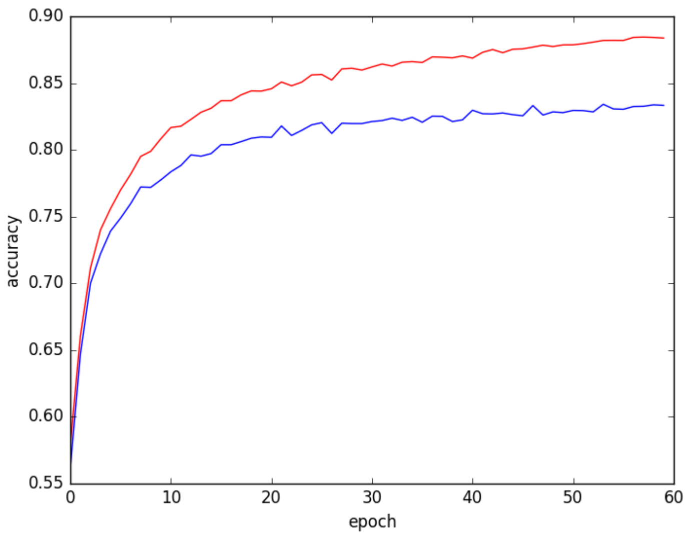


Fig.3: Sigmoid Accuracy

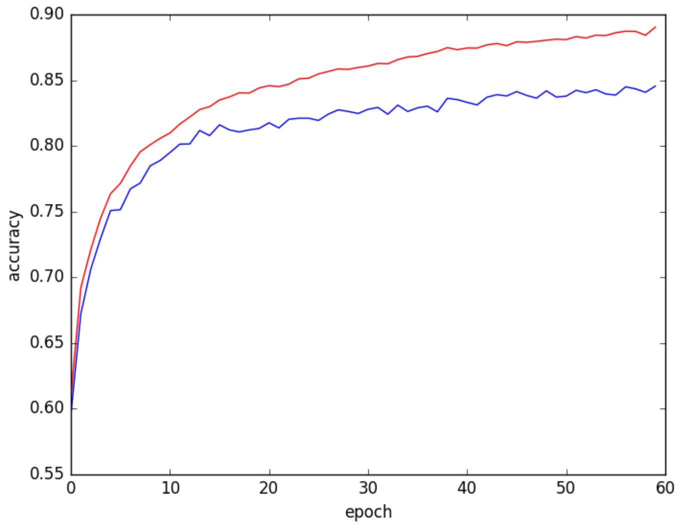


Figure 4: Tanh Accuracy

Saturation Values

Activation Function	Observation	Training	Testing
Sigmoid	**Accuracy**	88.20	83.16
Relu	**Accuracy**	88.92	84.78

Inference

Both tanh and sigmoid activation functions performs almost equally with tanh slightly better.
Possible Reason: Tanh is zero-centered.

Let us next compare Sigmoid and Relu activation functions.

2. Sigmoid vs. Relu

- **Settings**

Component	*Parameter*	*State*	*Interpretation*
Weight Initialization	`weight_initialization`	`'Xavier'`	Xavier Initialization
Network Structure	`layers`	`[(784, 'input'), (32,`	2-layer network

		'sigmoid'/'relu') , (9, 'output')]	
Epochs	epochs	70	70 epochs
Learning Rate	learning_rate	0.1	0.1
Mini-Batch size	mini_batch_size	100	100
Decay Rate	lmbda	0.0	No regularization
Regularization Technique	reg	'None'	No regularization
Loss Function	IsMSE	False	Cross-Entropy(Default)
Batch Normalization	isBatchNorm	False	No
Dropout	isDropout	False	No

- ## Observations

<u>Loss</u>

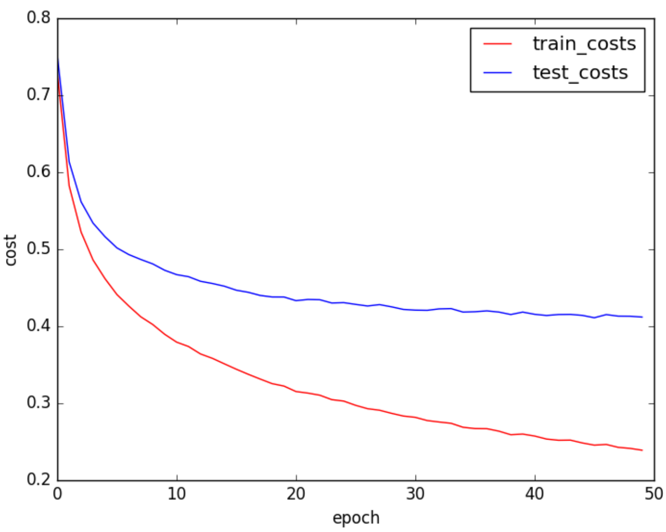


Figure 2: Sigmoid Loss

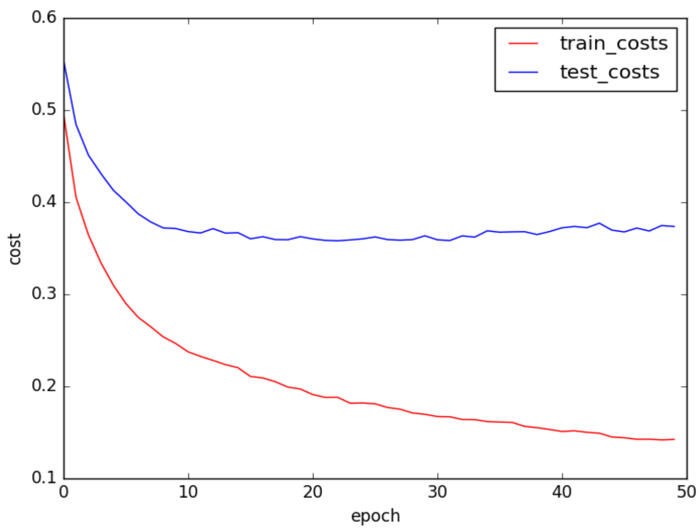


Figure 2: Relu Loss

Accuracy

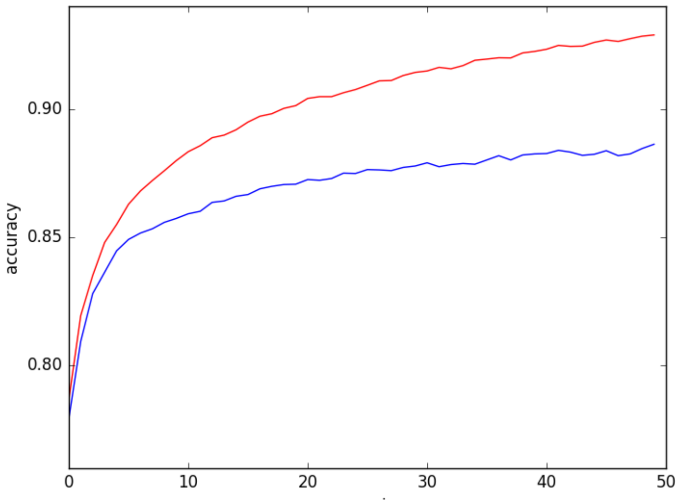


Figure 3: Sigmoid Accuracy

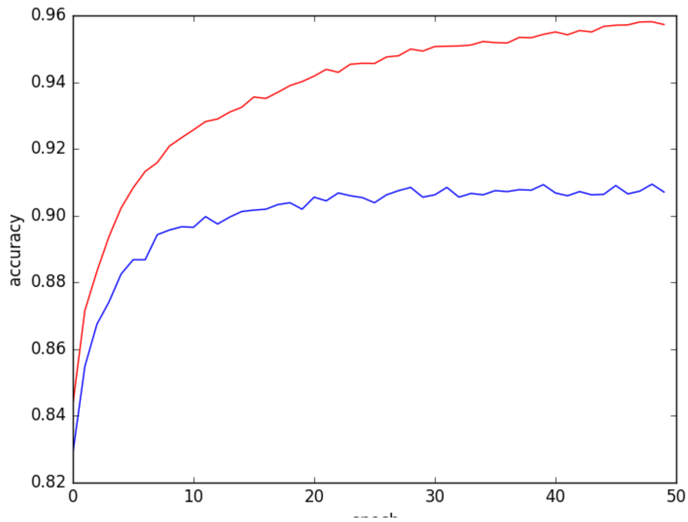


Figure 4: Relu Accuracy

Saturation Values

Activation Function	Observation	Training	Testing
Sigmoid	**Accuracy**	92.94	88.12
Relu	**Accuracy**	95.87	90.23

Inference

Clearly, accuracy for Relu is greater than that for Sigmoid. So, **Relu performs better.**
Possible Reason: Non-saturation of Relu's gradient which accelerates the convergence of the gradient descent used for back-propagation.

As Relu has proved to be much better than sigmoid (which was almost equivalent to tanh by Experiment.1), we will use Relu for most of the further experiments.

3. Mean-Squared Error vs. Cross-Entropy

- **Settings**

Component	*Parameter*	*State*	*Interpretation*
Weight Initialization	`weight_initialization`	`'Xavier'`	Xavier Initialization
Network Structure	`layers`	`[(784, 'input'), (128, 'relu'),`	3-layer network with sigmoid or

		(64, 'relu'), (9, 'output')]	tanh activation functions
Activation Function	Layers	'relu'	Relu
Epochs	epochs	100	100 epochs
Learning Rate	learning_rate	0.1	0.1
Mini-Batch size	mini_batch_size	100	100
Decay Rate	lmbda	0.0	No regularization
Regularization Technique	reg	'None'	No regularization
Batch Normalization	isBatchNorm	False	No
Dropout	isDropout	False	No

- **Observations**

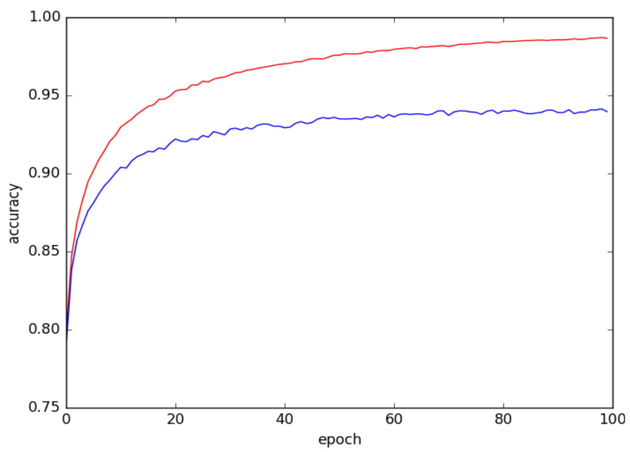


Figure 1: MSE Accuracy

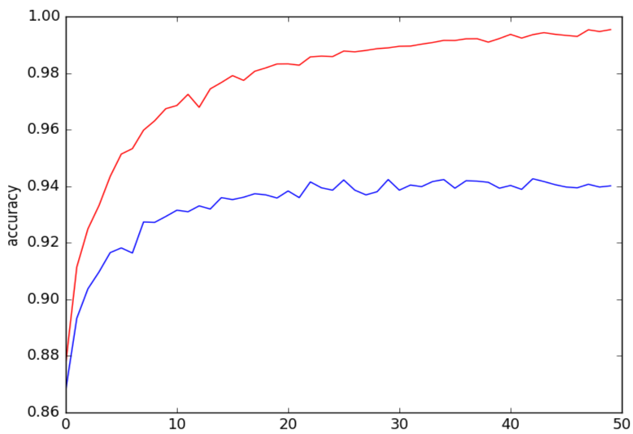


Figure 2: Cross Entropy Accuracy

Saturation Values

Loss Function	Observation	Training	Testing
MSE	**Accuracy**	98.05	93.28
Cross Entropy	**Accuracy**	99.28	93.72

Inference

The performance is equivalent with both the loss functions. While MSE looks to gives smoother convergence, Cross-Entropy converges much faster. So, **cross-entropy can be inferred to be a better loss function** to be used in our case.
Possible Reason: This is due to the softmax derivative in the output layer. In cross-entropy, the error propagated from the output layer is proportional to the difference between the true_value and output_value. In MSE, the error is proportional to (output_value – true_value)*(output_value)*(1 – output_value). So, the error in MSE have lower values and have higher chances of killing a neuron but this is not the case in Cross Entropy.

As using Cross-Entropy gives faster convergence, we will use Cross-Entropy loss for all the further experiments.

4. Varying the Number of Neurons in a single-hidden layer Neural Network.

- ### Settings

Component	***Parameter***	***State***	***Interpretation***
Weight Initialization	`weight_initialization`	`'Xavier'`	Xavier Initialization
Network Structure	`layers`	`[(784, 'input'), (16/32/64, 'relu'), (9, 'output')]`	2-layer network
Activation Function	`Layers`	`'relu'`	Relu
Loss Function	`IsMSE`	`False`	Cross-Entropy(Default)
Epochs	`epochs`	`50`	100 epochs
Learning Rate	`learning_rate`	`0.1`	0.1
Mini-Batch size	`mini_batch_size`	`100`	100
Decay Rate	`lmbda`	`0.0`	No regularization
Regularization Technique	`reg`	`'None'`	No regularization
Batch Normalization	`isBatchNorm`	`False`	No
Dropout	`isDropout`	`False`	No

- ### Observations

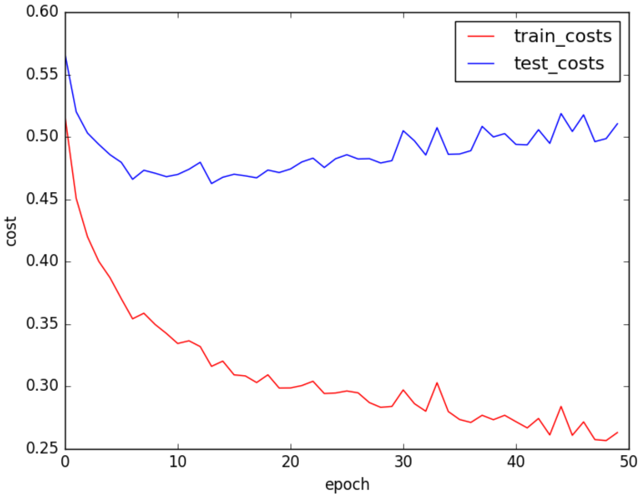


Figure 1: Cost for 16 neurons

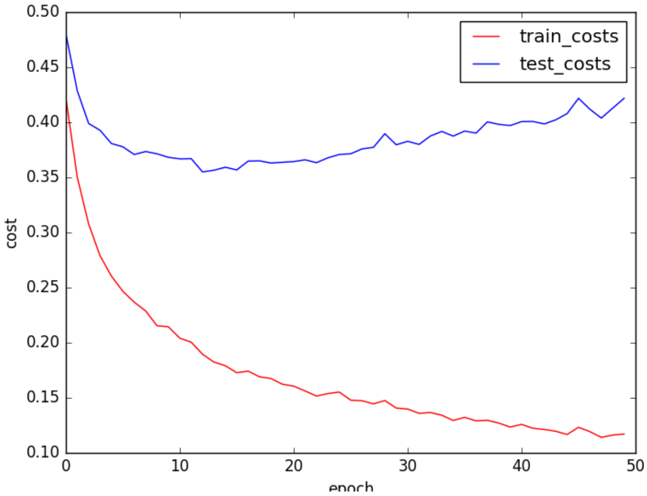


Figure 2: Loss for 32 Neurons

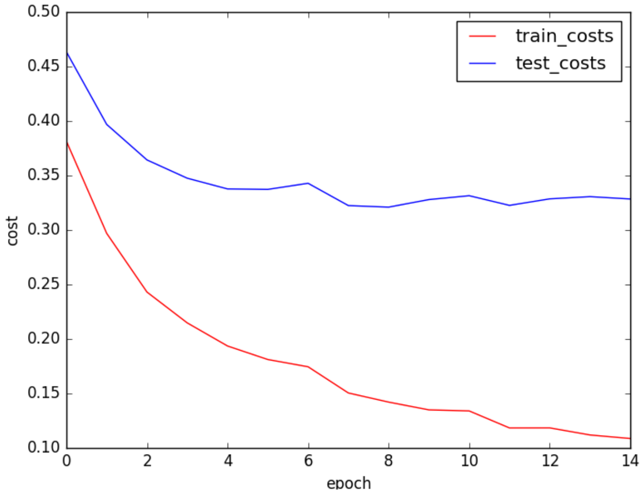


Figure 3: Loss for 64 Neurons

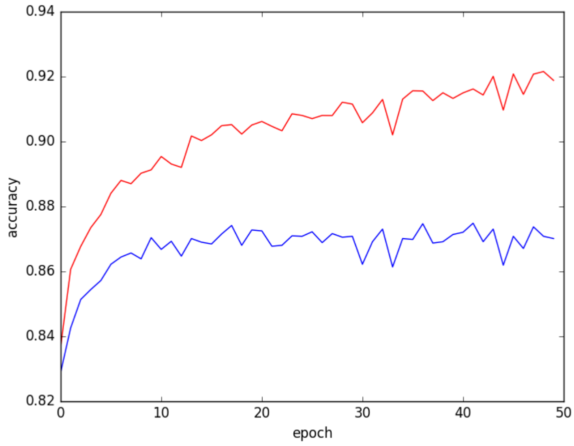


Figure 4: Accuracy for 16 neurons

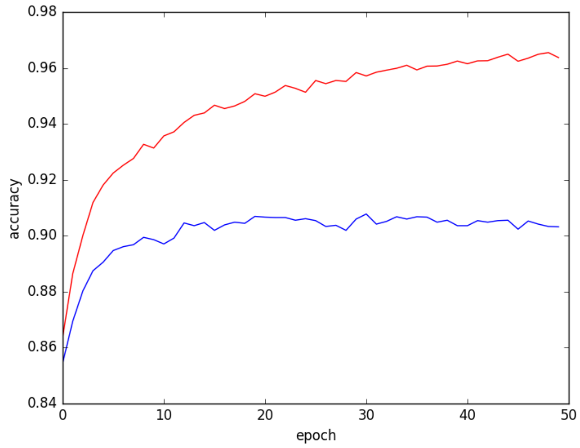


Figure 5: Accuracy for 32 neurons

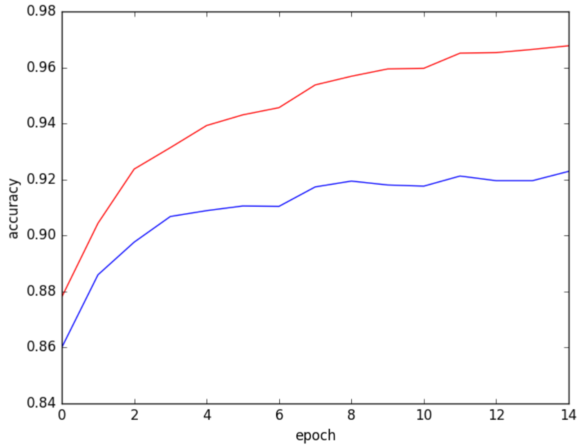


Figure 6: Accuracy for 64 neurons

<u>Saturation Values</u>

Number of Neurons	**Observation**	**Training**	**Testing**
16	**Accuracy**	91.84	86.73
32	**Accuracy**	96.15	89.96
64	**Accuracy**	96.93	92.03

Inference

Increasing the number of neurons **increases** the accuracy.
Possible Reason: Increasing the number of neurons in a shallow NN increases the capability of the network to learn more number of features. So, performance gets better, but if we further increase the number of neurons, there is a risk of overfitting the data.

We have seen that increasing the number of neurons in a shallow neural network increases the accuracy. Next, try to compare a shallow and deep neural network to motivate the use of deep NN.

5. Shallow Networks vs. Deep Networks for Sigmoid Activation

- ### Settings for Shallow Network

Component	***Parameter***	***State***	***Interpretation***
Weight Initialization	`weight initialization`	`'Gaussian'`	Gaussian Initialization
Network Structure	`layers`	`[(784, 'input'), (128, 'sigmoid'), (9, 'output')]`	2-layer network
Epochs	`epochs`	`60`	60 epoch
Learning Rate	`learning_rate`	`1.0`	1.0
Mini-Batch size	`mini_batch_size`	`100`	100
Decay Rate	`lmbda`	`0.0`	No regularization
Regularization Technique	`reg`	`‘None’`	No regularization
Loss Function	`IsMSE`	`False`	Cross-Entropy(Default)
Batch Normalization	`isBatchNorm`	`False`	No
Dropout	`isDropout`	`False`	No

- ### Settings for Deep Network

Component	***Parameter***	***State***	***Interpretation***
Weight Initialization	`weight initialization`	`'Gaussian'`	Gaussian Initialization
Network Structure	`layers`	`[(784, 'input'), (64, 'sigmoid'), (32,`	4-layer network

		'sigmoid'), (16, 'sigmoid'), (9, 'output')]	
Epochs	epochs	60	60 epoch
Learning Rate	learning_rate	1.0	1.0
Mini-Batch size	mini_batch_size	100	100
Decay Rate	lmbda	0.0	No regularization
Regularization Technique	reg	'None'	No regularization
Loss Function	IsMSE	False	Cross-Entropy(Default)
Batch Normalization	isBatchNorm	False	No
Dropout	isDropout	False	No

- ## Observations

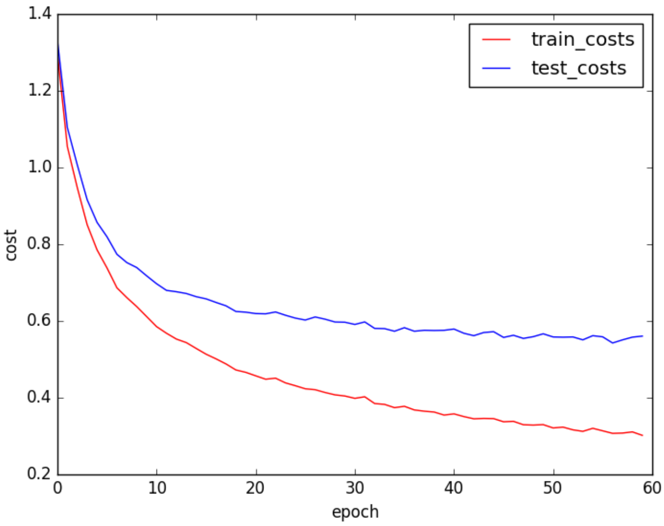


Figure 4: Loss for shallow network

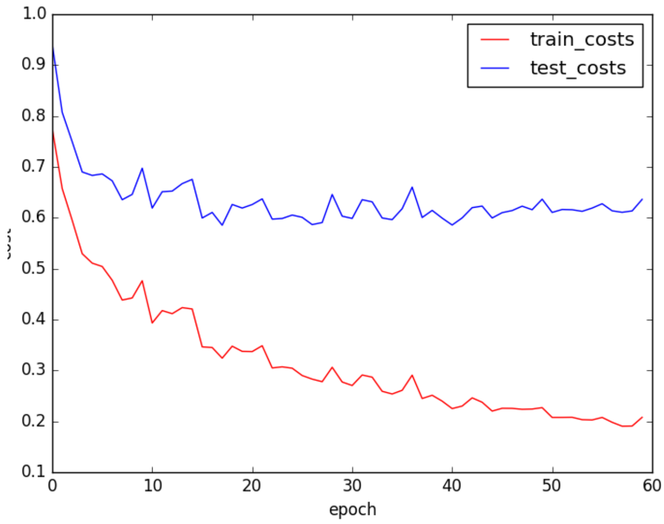


Figure 2: Loss for deep network

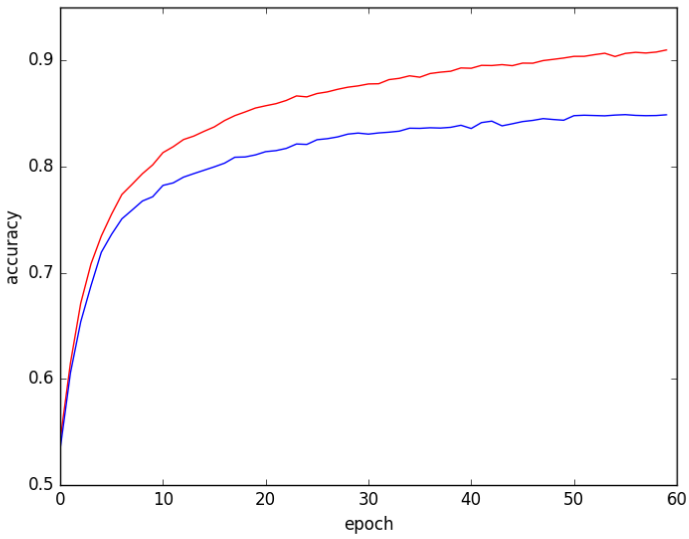


Figure 3: Accuracy for Shallow Network

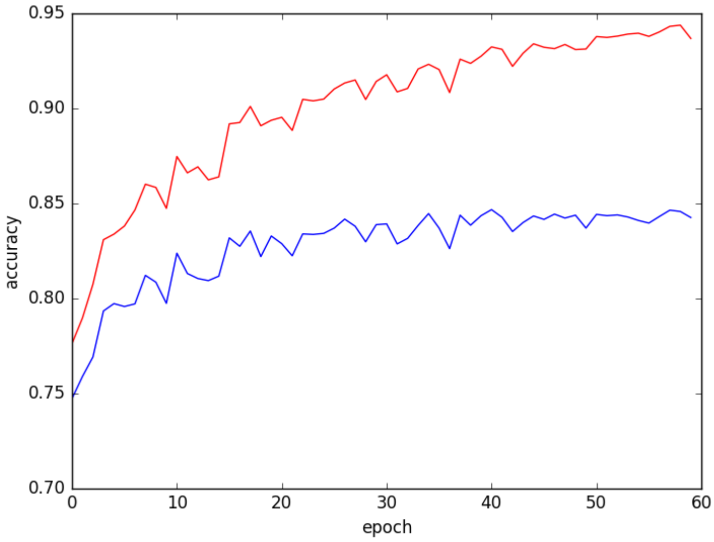


Figure 4: Accuracy for Deep Network

Saturation Values

Deep/Shallow	Observation	Training	Testing
Shallow	**Accuracy**	91.9	84.81
Deep	**Accuracy**	93.6	84.78

Inference

Deep NN performs slightly better than Shallow NN in accuracy.
Possible Reason: Deep Neural networks can learn composite features better although here the difference is not distinct enough

Sometimes, there are very high chances of a vanishing gradient in the deep Neural Network that might happen due to a bad weight initialization. Therefore, we next try to test deep neural network with Zero Initialization and Xavier initialization.

6. Comparison between Gaussian and Zero Initialization on Sigmoid Activation

- **Settings**

Component	***Parameter***	***State***	***Interpretation***
Weight Initialization	`weight_initialization`	`'Gaussian'/'Zero'`	Gaussian/Zero Initialization
Network Structure	`layers`	`[(784, 'input'), (128, 'sigmoid'), (64, 'sigmoid'), (32, 'sigmoid'), (9, 'output')]`	4-layer network
Epochs	`epochs`	`40`	40 epochs
Learning Rate	`learning_rate`	`5.0 (for Gaussian)/1.0 (for Zero)`	5.0/1.0
Mini-Batch size	`mini_batch_size`	`100`	100
Decay Rate	`lmbda`	`0.0`	No regularization
Regularization Technique	`reg`	`'None'`	No regularization
Loss Function	`IsMSE`	`False`	Cross-Entropy(Default)
Batch Normalization	`isBatchNorm`	`False`	No
Dropout	`isDropout`	`False`	No

- ## Observations

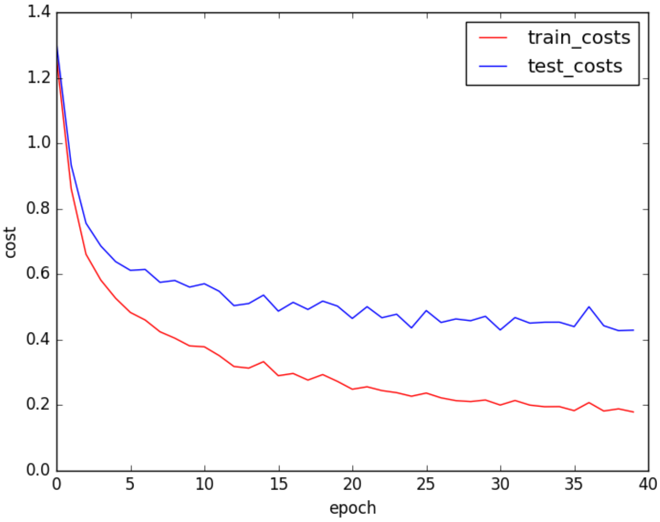


Figure 5: Loss for Gaussian Initialization

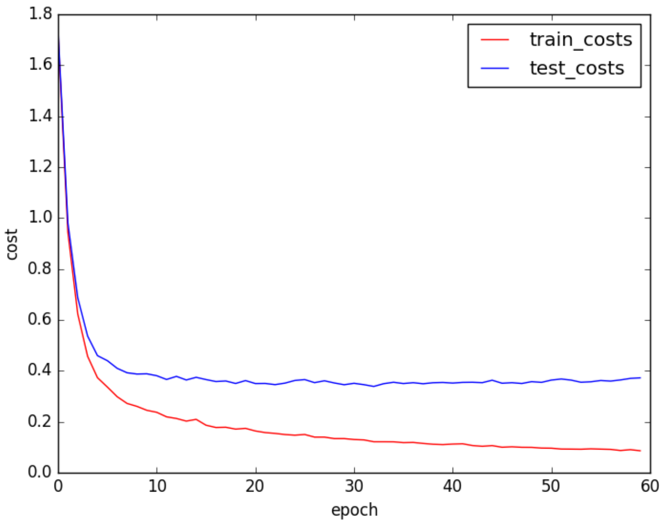


Figure 2: Loss for Zero Initialization

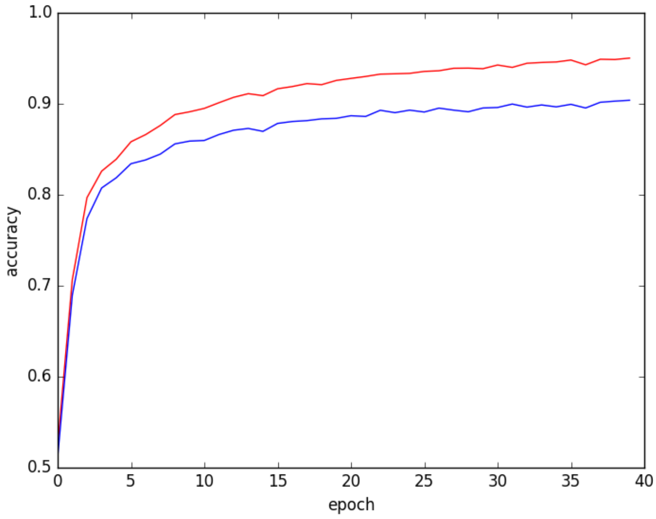


Figure 3: Accuracy for Gaussian Initialization

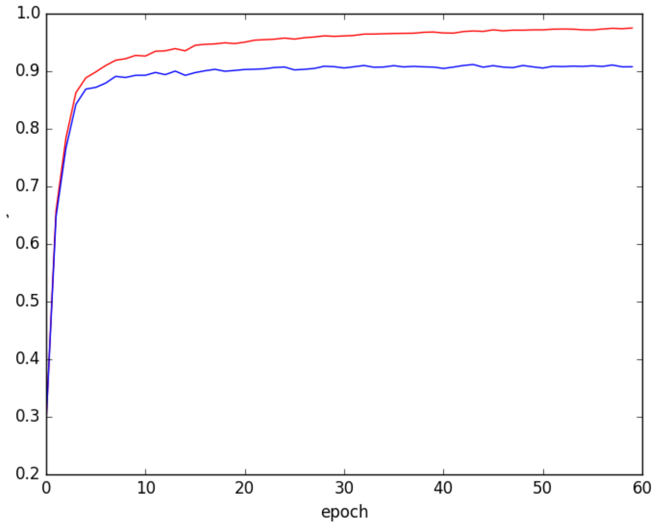


Figure 4: Accuracy for Zero Initialization

<u>Saturation Values</u>

Initialization	**Observation**	**Training**	**Testing**
Gaussian	**Accuracy**	95.30	90.05
Xavier	**Accuracy**	98.01	90.46

<u>Inference</u>

Surprisingly, **zero initialization works as good as Gaussian initialization and converges faster**.

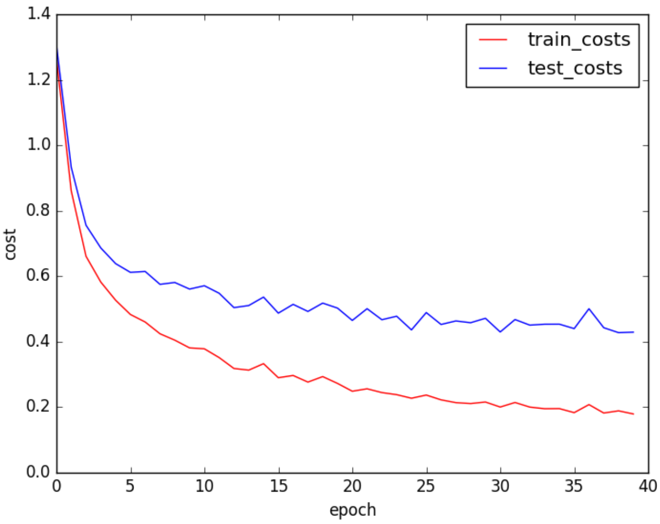
Possible Reason: Gaussian Random Initialization works much better than Zero initialization when Relu is used as it leads to killing most of the neurons. But, as Sigmoid is symmetric unlike Relu, zero-initialization does not have a negative impact when trained with Sigmoid acrivation.

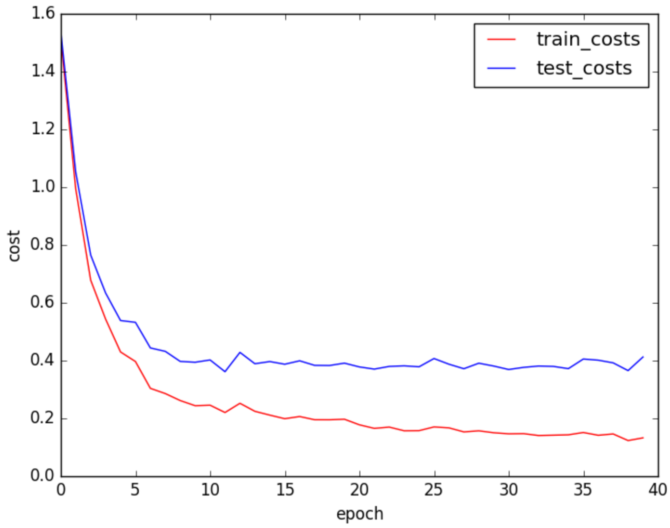
7. Comparison between Gaussian and Xavier Initialization on Sigmoid Activation

- **Settings**

Component	***Parameter***	***State***	***Interpretation***
Weight Initialization	`weight_initialization`	`'Gaussian'/'Xavier'`	Gaussian/Xavier Initialization
Network Structure	`layers`	`[(784, 'input'), (128, 'sigmoid'), (64, 'sigmoid'), (32, 'sigmoid'), (9, 'output')]`	4-layer network
Epochs	`epochs`	`40`	40 epochs
Learning Rate	`learning_rate`	`5.0 (for Gaussian)/0.1 (for Xavier)`	5.0/0.1
Mini-Batch size	`mini_batch_size`	`100`	100
Decay Rate	`lmbda`	`0.0`	No regularization
Regularization Technique	`reg`	`'None'`	No regularization
Loss Function	`IsMSE`	`False`	Cross-Entropy(Default)
Batch Normalization	`isBatchNorm`	`False`	No
Dropout	`isDropout`	`False`	No

- **Observations**





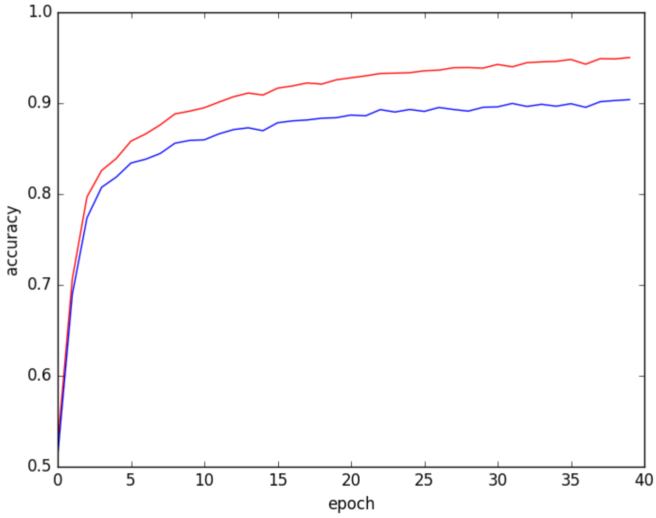


Figure 3: Accuracy for Gaussian Initialization

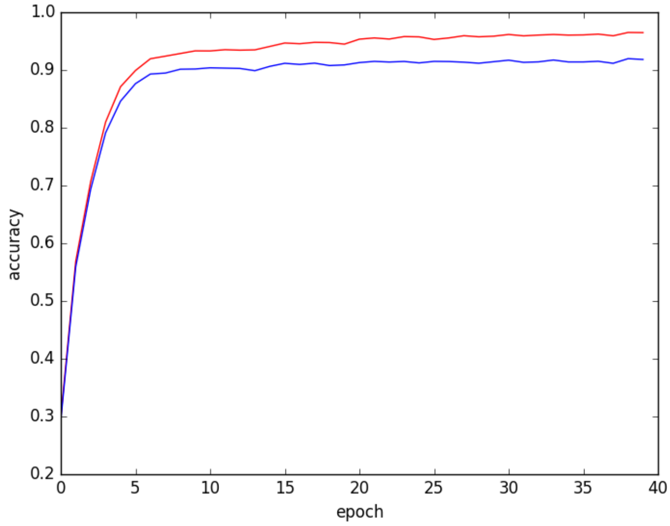


Figure 4: Accuracy for Xavier Initialization

Saturation Values

Initialization	Observation	Training	Testing
Gaussian	**Accuracy**	95.30	90.05
Xavier	**Accuracy**	96.48	91.83

Inference

Accuracy wise, both the initializations are almost equivalent. But **Xavier initialization converges much faster for Sigmoid Activation function**. So, Xavier Initialization is better than Gaussian Random Initialization.

Possible Reason: Xavier is Gaussian divided by the square root of the number of neurons in the previous hidden layer. So if the number of neurons in the previous hidden layer is too large, initialization by Xavier will be small and if the number of neurons in the previous layer is too low, weights initialized will be higher. This prevents the input in to the Sigmoid Activation from being too low or loo large, thus giving a better learning and a faster convergence.

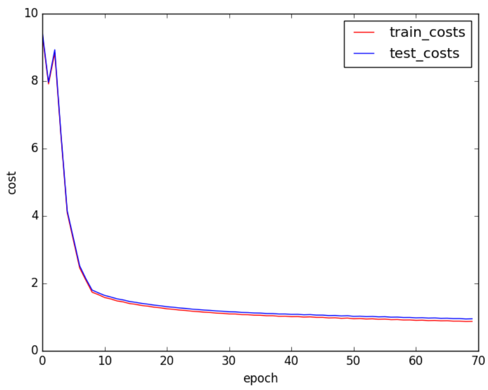
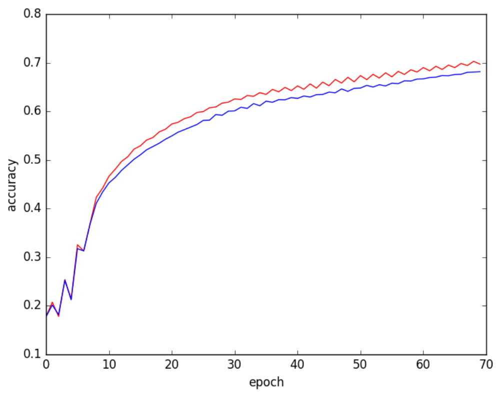
We move forward to check the effect of changing the Mini-Batch size in mini-batch gradient descent

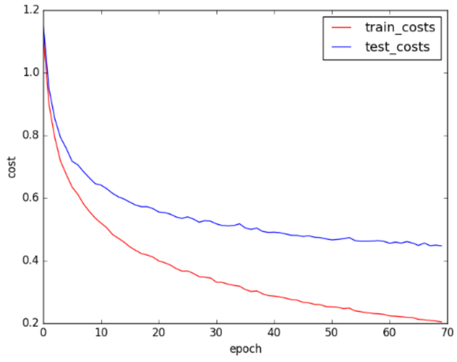
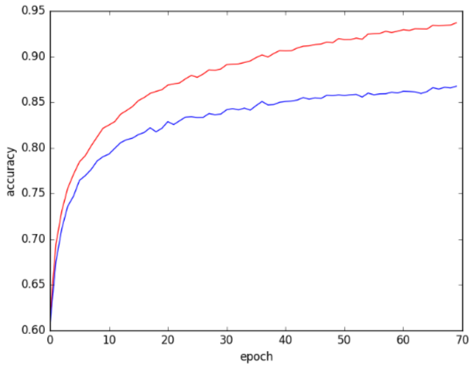
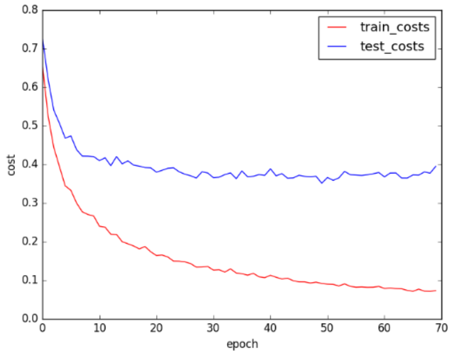
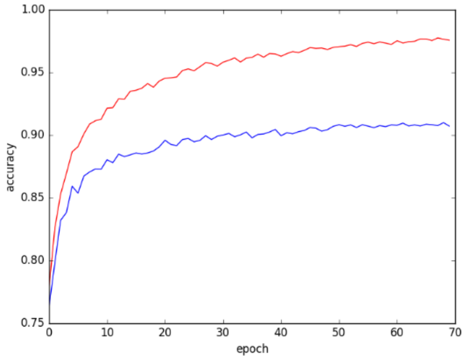
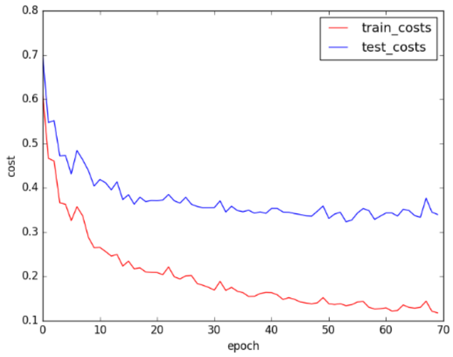
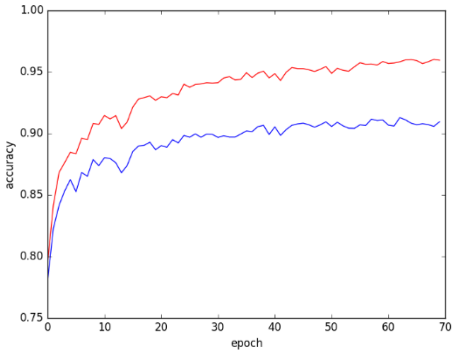
8. Effect of Mini-Batch size on Mini-Batch Gradient Descent

- **Settings**

Component	*Parameter*	*State*	*Interpretation*
Weight Initialization	`weight_initialization`	'Gaussian'	Gaussian Initialization
Network Structure	`layers`	[(784, 'input'), (128, 'sigmoid'), (64, 'sigmoid'),(9, 'output')]	3-layer network
Epochs	`epochs`	70	70 epochs
Learning Rate	`learning_rate`	5.0	5.0
Mini-Batch size	`mini_batch_size`	31799/1000/100/50	Same
Decay Rate	`lmbda`	0.0	No regularization
Regularization Technique	`reg`	'None'	No regularization
Loss Function	`IsMSE`	False	Cross-Entropy(Default)
Batch Normalization	`isBatchNorm`	False	No
Dropout	`isDropout`	False	No

- **Observations**

Batch-Size	**Loss**	**Accuracy**
31799		

1000		
100		
50		

Saturation Values

Mini-Batch Size	Observation	Training	Testing
31799	**Accuracy**	70.12	69.07
1000	**Accuracy**	94.23	87.61
100	**Accuracy**	96.44	92.20
50	**Accuracy**	96.12	91.11

Inference

Decreasing the batch-size increases the accuracy till a batch size of 100 and then decreased by further decreasing batch size to 50. This shows that mini-batch gradient descent gives

better result than normal gradient descent and **there exists an optimal batch size going below or above which accuracy decreases**.
Possible Reason: It takes a lot of memory to do normal gradient descent. The real issue is that a normal gradient trajectory may land you on a bad spot (saddle-point or local minima). Pure stochastic gradient descent which is based on one random data point, is very noisy and may go in a direction far from the batch-gradient. The mini-batch methodology is a compromise that injects enough noise to each gradient update which achieving relatively speedy convergence.

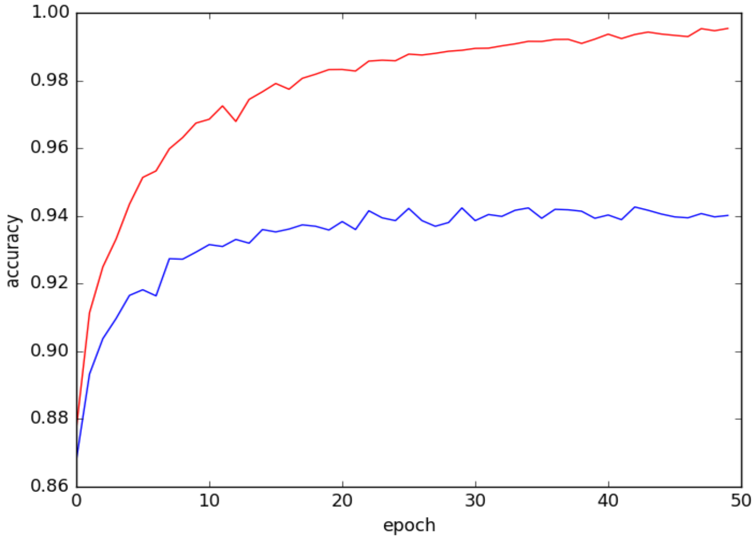
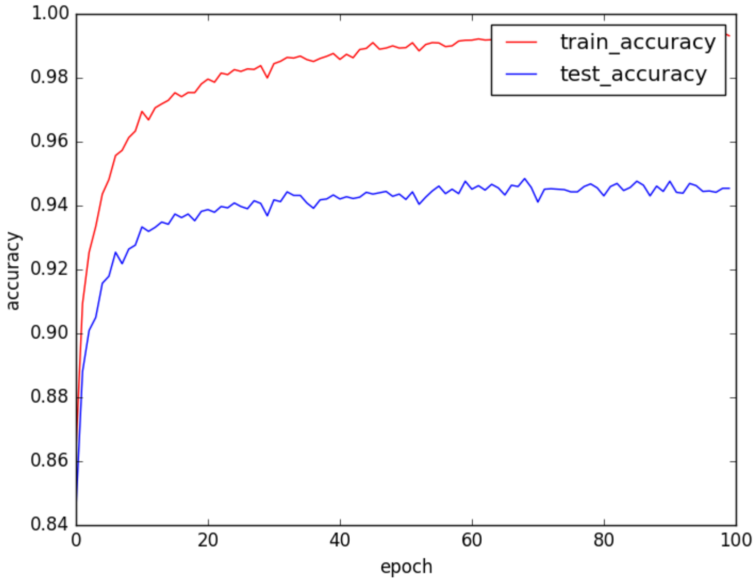
Next, we look at some popular ways to further optimize our model called Regularization methods.

9. Effect of L2 Regularization with Cross-Entropy Cost

- **Settings**

Component	*Parameter*	*State*	*Interpretation*
Weight Initialization	`weight_initialization`	`'Xavier'`	Xavier Initialization
Network Structure	`layers`	`[(784, 'input'), (128, 'relu'), (64, 'relu'),(9, 'output')]`	3-layer network
Epochs	`epochs`	`100`	100 epochs
Learning Rate	`learning_rate`	`0.1`	0.1
Mini-Batch size	`mini_batch_size`	`100`	100
Decay Rate	`lmbda`	`0.1`	0.1
Regularization Technique	`reg`	`'L2'`	L2 regularizatiom
Loss Function	`IsMSE`	`False`	Cross-Entropy(Default)
Batch Normalization	`isBatchNorm`	`False`	No
Dropout	`isDropout`	`False`	No

- **Observations**

<table>
<tr><th>Regularization Technique</th><th>Accuracy</th></tr>
<tr><td>None</td><td>
</td></tr>
<tr><td>L2</td><td>
</td></tr>
</table>

<u>Saturation Values</u>

Cost Function	Observation	Training	Testing
Cross Entropy	**Accuracy**	99.28	93.72
Cross Entropy + L2	**Accuracy**	99.17	94.56

Inference

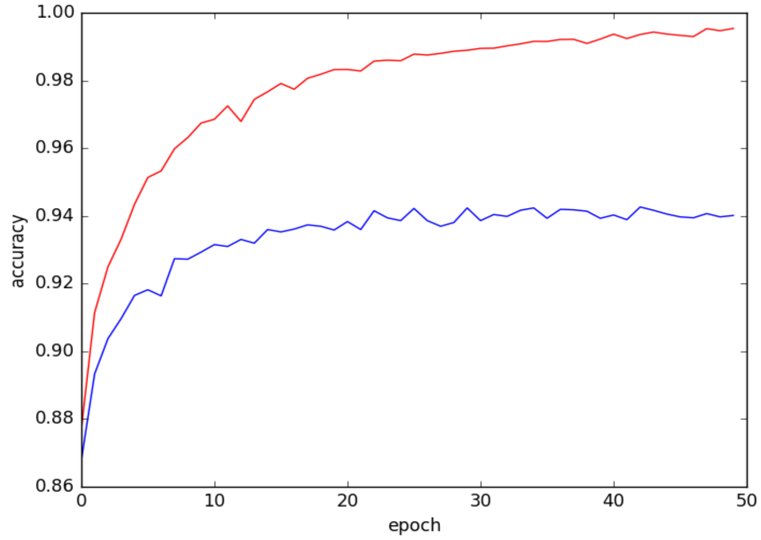
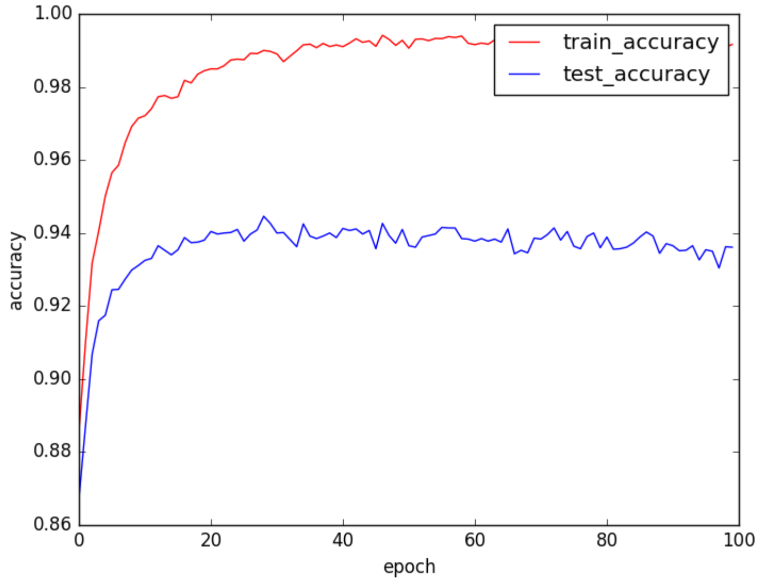
L2 regularization decreases the training accuracy slightly and improves the testing accuracy.
Possible Reason: L2 normalization increases bias and reduces variance, thus improves the performance.
Next, we look at L1 normalization.

10. Effect of L1 Regularization with Cross-Entropy Cost

- **Settings**

Component	***Parameter***	***State***	***Interpretation***
Weight Initialization	`weight_initialization`	`‘Xavier’`	Xavier Initialization
Network Structure	`layers`	`[(784, 'input'), (128, 'relu'), (64, 'relu'),(9, 'output')]`	3-layer network
Epochs	`epochs`	`100`	100 epochs
Learning Rate	`learning_rate`	`0.1`	0.1
Mini-Batch size	`mini_batch_size`	`100`	100
Decay Rate	`lmbda`	`0.1`	0.1
Regularization Technique	`reg`	`‘L1’`	L1 regularizatiom
Loss Function	`IsMSE`	`False`	Cross-Entropy(Default)
Batch Normalization	`isBatchNorm`	`False`	No
Dropout	`isDropout`	`False`	No

- **Observations**

Regularization Technique	Accuracy
None	
L1	

Saturation Values

Cost Function	Observation	Training	Testing
Cross Entropy	**Accuracy**	99.28	93.72
Cross Entropy + L1	**Accuracy**	99.21	93.76

Inference

L1 regularization does not bring much improvement to the results.
Possible Reason: Feature selection is not a good option with a neural network with the given size. A larger neural network may work better.

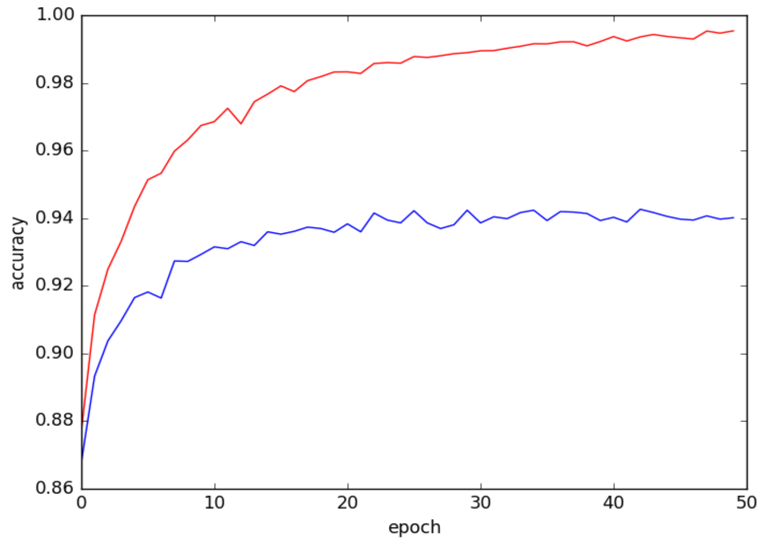
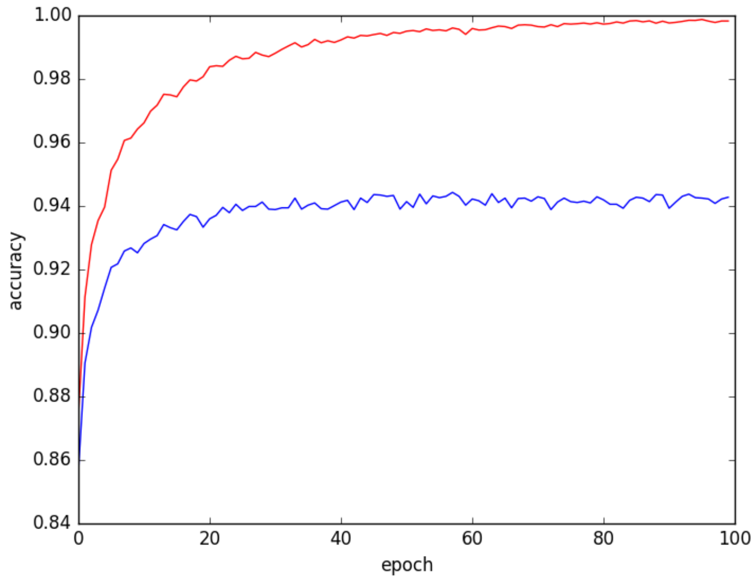
Next, we inspect the value of decay rates in normalization.

11. Effect of the magnitude of Decay Constant on Regularization with Cross-Entropy Cost

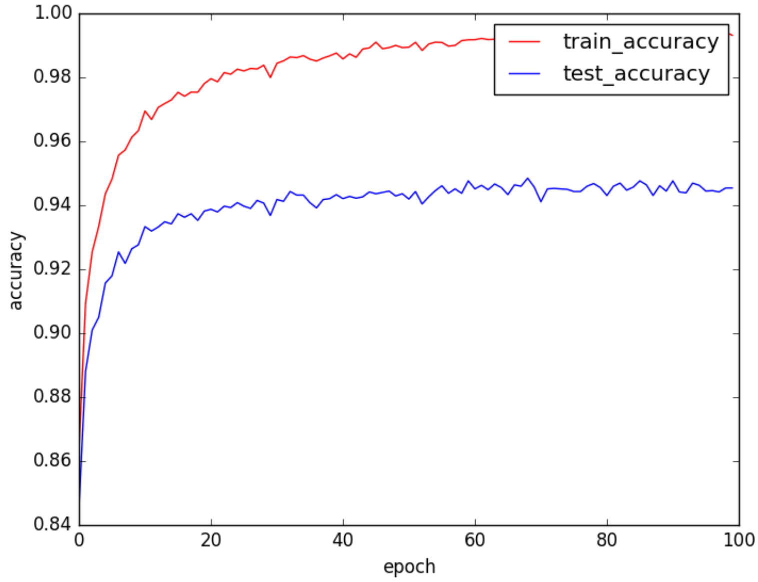
- **Settings**

Component	***Parameter***	***State***	***Interpretation***
Weight Initialization	`weight_initialization`	`'Xavier'`	Xavier Initialization
Network Structure	`layers`	`[(784, 'input'), (128, 'relu'), (64, 'relu'),(9, 'output')]`	3-layer network
Epochs	`epochs`	`100`	100 epochs
Learning Rate	`learning_rate`	`0.1`	0.1
Mini-Batch size	`mini_batch_size`	`100`	100
Decay Rate	`lmbda`	`0/0.0001/0.1/5.0`	Same
Regularization Technique	`reg`	`'L2'`	L2 regularizatiom
Loss Function	`IsMSE`	`False`	Cross-Entropy(Default)
Batch Normalization	`isBatchNorm`	`False`	No
Dropout	`isDropout`	`False`	No

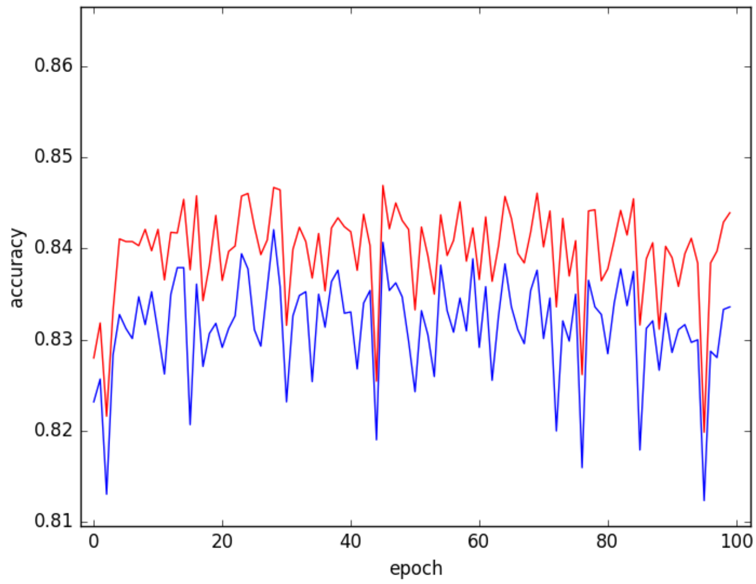
- **Observations**

Decay Constant	Accuracy
0.0 (No Regularization)	
0.0001 (Weak Regularization)	

0.1 (Moderate Regularization)



5.0 (Strong Regularization)



Saturation Values

Lambda	Observation	Training	Testing
0	**Accuracy**	99.28	93.72
0.0001	**Accuracy**	99.89	94.13
0.1	**Accuracy**	99.17	94.56
5.0	**Accuracy**	Fluctuating	Fluctuating

Inference

For very small lambda (0.0001) the model with L2 regularization performs very similar to no regularization. A moderate value of lambda works well. When lambda is very high (5.0), the cost fluctuates heavily and the model performs worst.
Possible Reason: For very small lambda, the data cost dominates the regularization cost while optimization, **so the effect of regularization is not significant.** The reverse happens **for large values of lambda and the model underfits.**

12. Effect of adding noise to the Training Features and comparison with L2 normalization with Cross-Entropy Cost

- ### Settings for L2 Normalization with Cross-Entropy

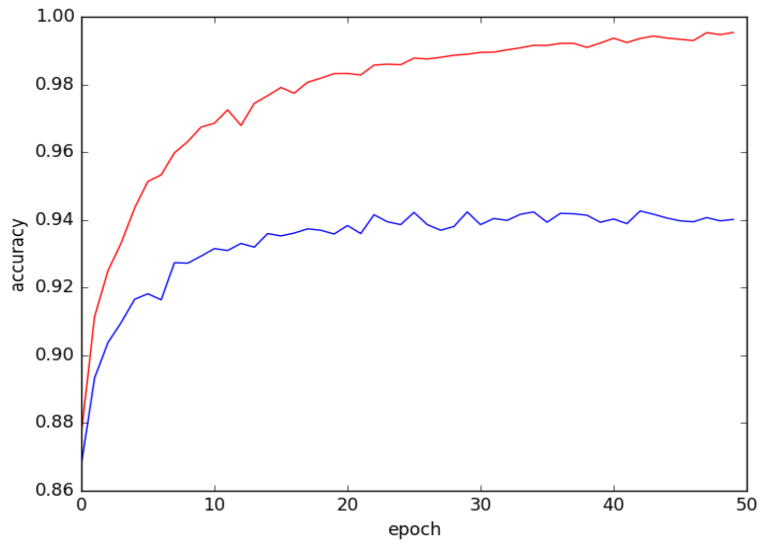
Component	*Parameter*	*State*	*Interpretation*
Weight Initialization	`weight_initialization`	`'Xavier'`	Xavier Initialization
Network Structure	`layers`	`[(784, 'input'), (128, 'relu'), (64, 'relu'),(9, 'output')]`	3-layer network
Epochs	`epochs`	`100`	100 epochs
Learning Rate	`learning_rate`	`0.1`	0.1
Mini-Batch size	`mini_batch_size`	`100`	100
Decay Rate	`lmbda`	`0.1`	0.1
Regularization Technique	`reg`	`'L2'`	L2 regularization
Loss Function	`IsMSE`	`False`	Cross-Entropy(Default)
Batch Normalization	`isBatchNorm`	`False`	No
Dropout	`isDropout`	`False`	No

- ### Settings for Noise Addition with Cross-Entropy

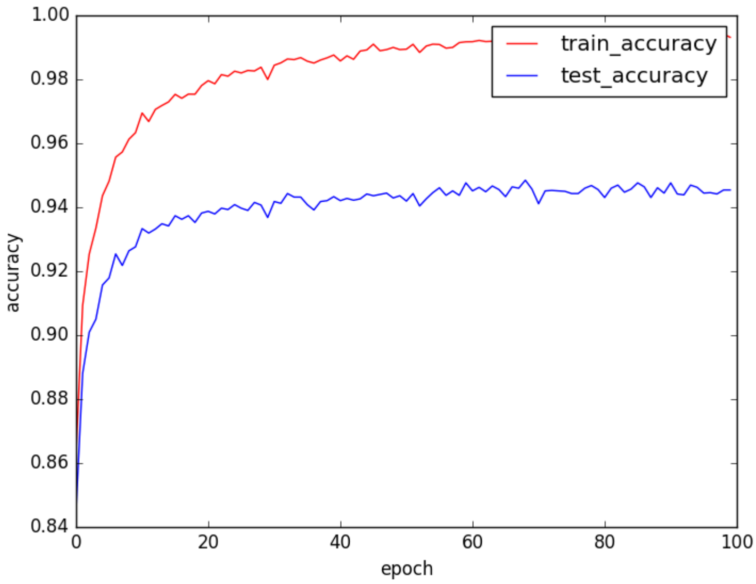
Component	*Parameter*	*State*	*Interpretation*
Weight Initialization	`weight_initialization`	`'Xavier'`	Xavier Initialization
Network Structure	`layers`	`[(784, 'input'), (128, 'relu'), (64, 'relu'),(9, 'output')]`	3-layer network
Epochs	`epochs`	`100`	100 epochs
Learning Rate	`learning_rate`	`0.1`	0.1
Mini-Batch size	`mini_batch_size`	`100`	100

Decay Rate	`lmbda`	`0.1`	0.1
Regularization Technique	`reg`	`'L2'`	L2 regularizatiom
Loss Function	`IsMSE`	`False`	Cross-Entropy(Default)
Batch Normalization	`isBatchNorm`	`False`	No
Dropout	`isDropout`	`False`	No
Noise Addition	`None`	`NA`	Yes (refer to Main.py)
Noise type	`NA`	`NA`	Gaussian
Noise (Mean, Variance)	`NA`	`(0, 0.1)`	(0, 0.1)

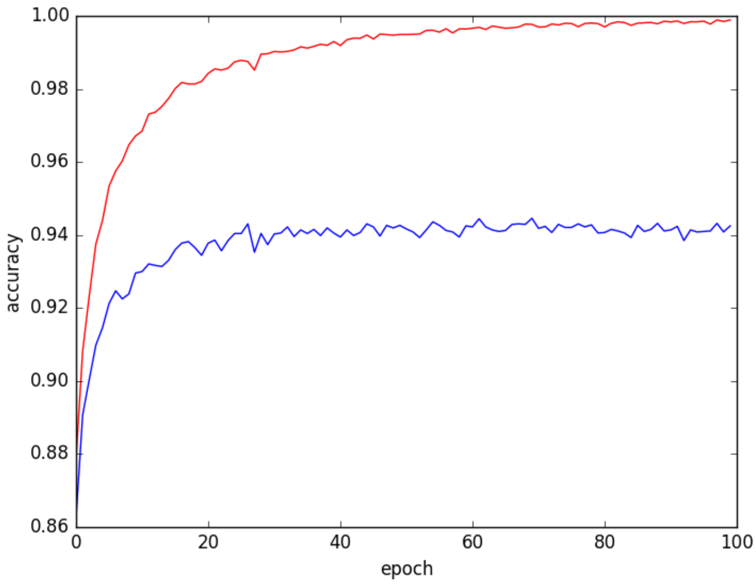
- ## Observations

Regularization Technique	**Accuracy**
None	

L2 (lambda = 0.1)



Adding Noise to training input without L2



<u>Saturation Values</u>

Cost Function	Observation	Training	Testing
Cross Entropy	Accuracy	99.28	93.72
Cross Entropy + L2	Accuracy	99.17	94.56
Cross Entropy (after adding Noise)	Accuracy	99.92	94.44

Inference

Adding noise to input while training **functions similar to L2 regularization** where variance of the noise is equal to the decay factor.
Possible Reason: The averaged optimal weight in the case of noisy inputs is same as in the case of no noise with L2 regularization.

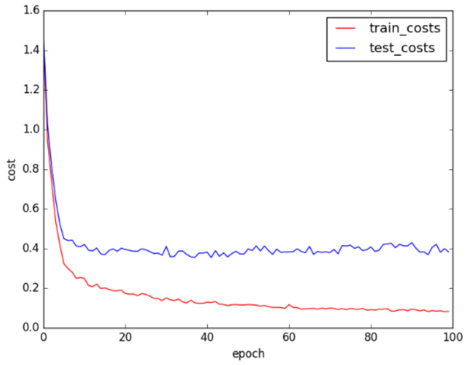
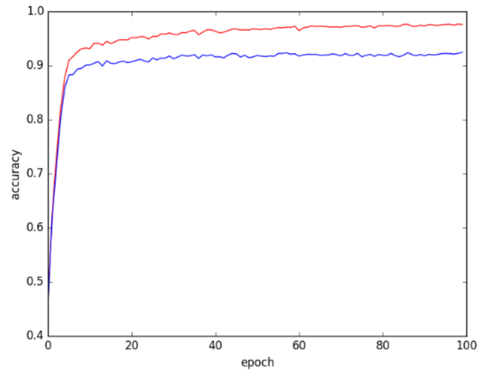
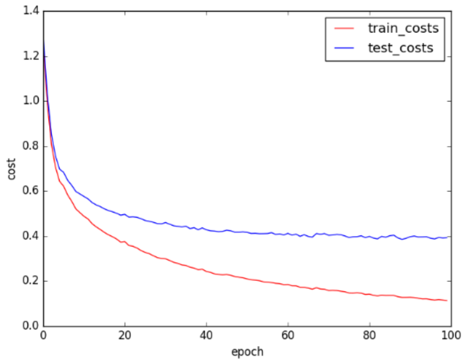
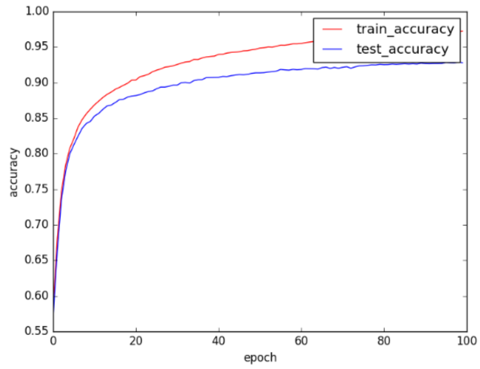
13. Effect of Dropout on Deep Neural Network

- ### Settings for Dropout

Component	***Parameter***	***State***	***Interpretation***
Weight Initialization	`Weight_initialization`	`'Xavier'`	Xavier Initialization
Network Structure	`Layers`	`[(784, 'input'), (128, 'relu'), (64, 'relu'), (32, 'relu') (16, 'relu'),(9, 'output')]`	5-layer network
Epochs	`Epochs`	`100`	100 epochs
Learning Rate	`learning_rate`	`0.1`	0.1
Mini-Batch size	`mini_batch_size`	`100`	100
Decay Rate	`Lmbda`	`0`	0
Regularization Technique	`Reg`	`'None'`	No regularization
Loss Function	`IsMSE`	`False`	Cross-Entropy(Default)
Batch Normalization	`isBatchNorm`	`False`	No
Dropout	`isDropout`	`True`	Yes
Probability	`self.p`	`0.5`	constant

- ### Observations

Regularization Technique	**Cost**	**Accuracy**

None		
Dropout		

Saturation Values

Regularization	**Observation**	**Training**	**Testing**
None	**Accuracy**	98.03	92.13
Dropout	**Accuracy**	97.4	92.79

Inference

There is a little bit improvement in the accuracy of the model.

Next, we can compare the benefit of dropout by varying depths. This may be the reason for poor results in the present experiment.

14. Dropout in Shallow NN vs. Dropout in deep NN

- **Settings for Dropout in deep NN**

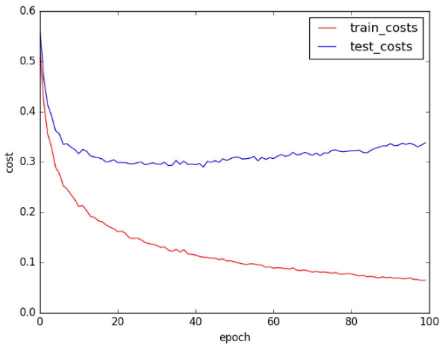
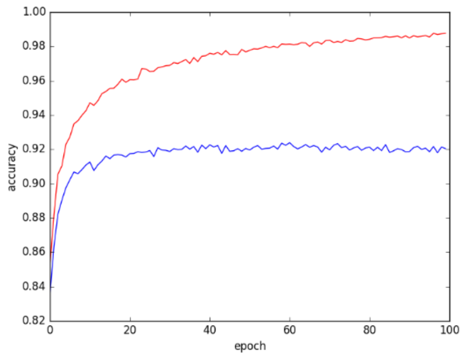
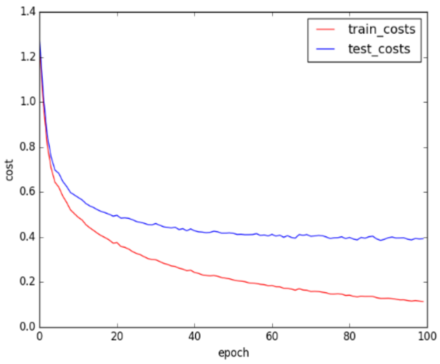
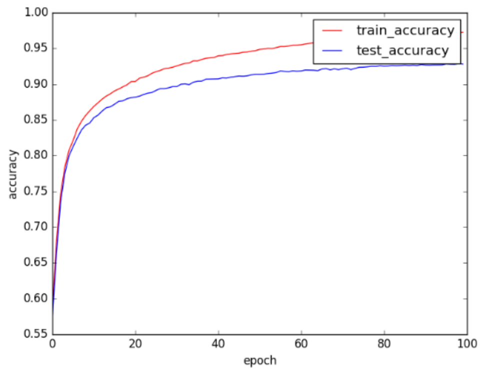
Component	***Parameter***	***State***	***Interpretation***
Weight Initialization	`Weight_initialization`	`'Xavier'`	Xavier Initialization

Network Structure	`Layers`	`[(784, 'input'), (128,'relu'),(9, 'output')]`	2-layer network
Epochs	`Epochs`	`100`	100 epochs
Learning Rate	`learning_rate`	`0.1`	0.1
Mini-Batch size	`mini_batch_size`	`100`	100
Decay Rate	`Lmbda`	`0`	0
Regularization Technique	`Reg`	`‘None’`	No regularization
Loss Function	`IsMSE`	`False`	Cross-Entropy(Default)
Batch Normalization	`isBatchNorm`	`False`	No
Dropout	`isDropout`	`True`	Yes
Probability	`self.p`	`0.5`	constant

- ## Settings for Dropout in deep NN

Component	***Parameter***	***State***	***Interpretation***
Weight Initialization	`Weight_initialization`	`‘Xavier’`	Xavier Initialization
Network Structure	`Layers`	`[(784, 'input'), (128, 'relu'), (64, 'relu'), (32, ‘relu’) (16, ‘relu’),(9, 'output')]`	5-layer network
Epochs	`Epochs`	`100`	100 epochs
Learning Rate	`learning_rate`	`0.1`	0.1
Mini-Batch size	`mini_batch_size`	`100`	100
Decay Rate	`Lmbda`	`0`	0
Regularization Technique	`Reg`	`‘None’`	No regularization
Loss Function	`IsMSE`	`False`	Cross-Entropy(Default)
Batch Normalization	`isBatchNorm`	`False`	No
Dropout	`isDropout`	`True`	Yes
Probability	`self.p`	`0.5`	constant

- ### Observations

NN Structure	Cost	Accuracy
Shallow		
Deep		

<u>Saturation Values</u>

Structure	Observation	Training	Testing
Shallow	**Accuracy**	98.87	92.00
Deep	**Accuracy**	97.46	93.03

<u>Inference</u>

Drop in deeper networks are more effective than dropout in shallow network.

15. Batch Normalization

- ### Settings for Batch Normalization in deep NN

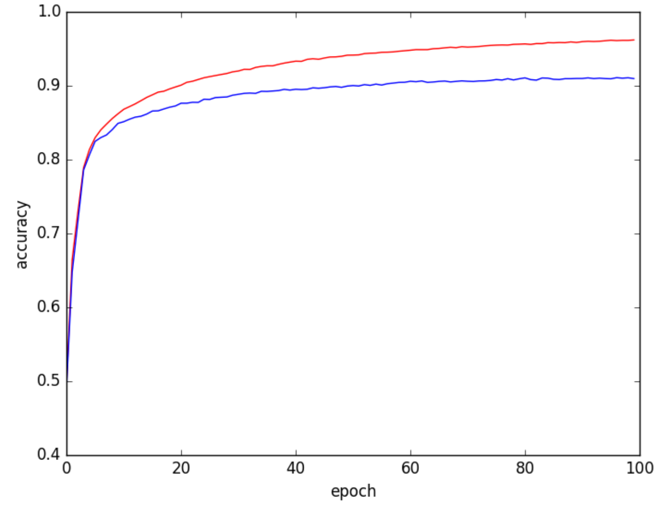
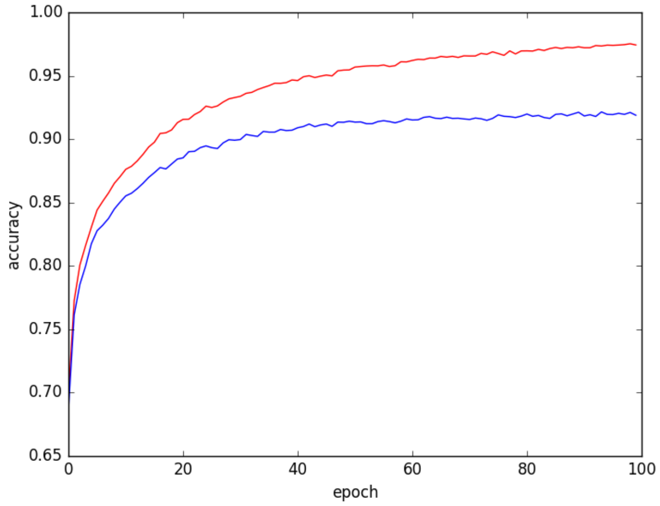
Component	*Parameter*	*State*	*Interpretation*
Weight Initialization	`Weight_initialization`	`'Xavier'`	Xavier Initialization

Network Structure	`Layers`	`[(784, 'input'), (64,'sigmoid'),(32, 'sigmoid'),(9, 'output')]`	3-layer network
Epochs	`Epochs`	`100`	100 epochs
Learning Rate	`learning_rate`	`0.1`	0.1
Mini-Batch size	`mini_batch_size`	`100`	100
Decay Rate	`Lmbda`	`0`	0
Regularization Technique	`Reg`	`'None'`	No regularization
Loss Function	`IsMSE`	`False`	Cross-Entropy(Default)
Batch Normalization	`isBatchNorm`	`True`	Yes
Dropout	`isDropout`	`False`	No
Probability	`self.p`	`1.0`	constant

- **Settings for without Batch Normalization in deep NN**

Component	***Parameter***	***State***	***Interpretation***
Weight Initialization	`Weight_initialization`	`'Xavier'`	Xavier Initialization
Network Structure	`Layers`	`[(784, 'input'), (64,'sigmoid'),(32, 'sigmoid'),(9, 'output')]`	3-layer network
Epochs	`Epochs`	`100`	100 epochs
Learning Rate	`learning_rate`	`0.1`	0.1
Mini-Batch size	`mini_batch_size`	`100`	100
Decay Rate	`Lmbda`	`0`	0
Regularization Technique	`Reg`	`'None'`	No regularization
Loss Function	`IsMSE`	`False`	Cross-Entropy(Default)
Batch Normalization	`isBatchNorm`	`False`	No
Dropout	`isDropout`	`False`	No
Probability	`self.p`	`0.5`	constant

- **Observations**

Normalization Technique	Accuracy
None	
Batch Norm	

Saturation Values

Normalization Technique	Observation	Training	Testing
None	**Accuracy**	97.81	91.43
Batch Normalization	**Accuracy**	97.48	92.3

Inference

Batch Normalization increases the test accuracy and decreases the training accuracy. Also the convergence is much faster in the Batch Normalization

Possible Reason: Batch Normalization improves the gradient flow through the network and adds a slight regularization effect into your network. It also reduces the dependence of your network to your weight initialization.

7 SUMMARY

Experiment	**Summary**
Sigmoid vs. Tanh	Both tanh and sigmoid activation functions performs almost equally with tanh slightly better.
Sigmoid vs. Relu	Clearly, accuracy for Relu is greater than that for Sigmoid. So, Relu performs better.
Mean-Squared Error vs. Cross-Entropy	The performance is equivalent with both the loss functions. While MSE looks to gives smoother convergence, Cross-Entropy converges much faster. So, cross-entropy can be inferred to be a better loss function to be used in our case.
Varying the Number of Neurons in a single-hidden layer Neural Network.	Increasing the number of neurons increases the accuracy.
Shallow Networks vs. Deep Networks for Sigmoid Activation	Deep NN performs slightly better than Shallow NN in accuracy.
Comparison between Gaussian and Zero Initialization on Sigmoid Activation	Surprisingly, zero initialization works as good as Gaussian initialization and converges faster.
Comparison between Gaussian and Xavier Initialization on Sigmoid Activation	Accuracy wise, both the initializations are almost equivalent. But Xavier initialization converges much faster for Sigmoid Activation function. So, Xavier Initialization is better than Gaussian Random Initialization.
Effect of Mini-Batch size on Mini-Batch Gradient Descent	Decreasing the batch-size increases the accuracy till a batch size of 100 and then decreased by further decreasing batch size to 50. This shows that mini-batch gradient descent gives better result than normal gradient descent and there exists an optimal batch size going below or above which accuracy decreases.
Effect of L2 Regularization with Cross-Entropy Cost	L2 regularization decreases the training accuracy slightly and improves the testing accuracy.

Effect of L1 Regularization with Cross-Entropy Cost	L1 regularization does not bring much improvement to the results.
Effect of the magnitude of Decay Constant on Regularization with Cross-Entropy Cost	For very small lambda (0.0001) the model with L2 regularization performs very similar to no regularization. A moderate value of lambda works well. When lambda is very high (5.0), the cost fluctuates heavily and the model performs worst.
Effect of adding noise to the Training Features and comparison with L2 normalization with Cross-Entropy Cost	Adding noise to input while training functions similar to L2 regularization where variance of the noise is equal to the decay factor.
Effect of Dropout on Deep Neural Network	There is a little bit improvement in the accuracy of the model.
Dropout in Shallow NN vs. Dropout in deep NN	Dropout in deeper networks are more effective than dropout in shallow network.
Batch Normalization	Batch Normalization increases the test accuracy and decreases the training accuracy

8 CONCLUSION AND LEARNINGS

This assignment provided a thorough knowledge of the implementation of Neural Networks. Although it is clear that a deep neural network with more than 5 hidden layers works very slow on normal CPUs but a 3-4 layer Neural Network can be implemented and is sufficient for basic understanding. This assignment gave insights and practical experience of very important concepts like cost function, activation function and various regularization techniques. In general, it can be concluded that for the given dataset and a neural network with 3-4 hidden layers:

i. Xavier Initialization
ii. ReLU Activation
iii. Cross Entropy Loss Function
iv. L2 Normalization / Adding Noise
v. Mini-batch of size 100

Works well.