Question 1:

No, t is shared with all the nodes so its equivalent ~~to~~ a t- effect layer with vanilla rnn, so this doesn't add any expressive power to our model

Question 2.2?

~~The derivative of sigmoid is $\frac{\partial \sigma(x)}{\partial x}$~~

derivation of $\sigma(x)$?

$$\frac{\partial}{\partial x}\sigma(x) = \frac{\partial}{\partial x}\left(\frac{1}{1+e^{-x}}\right) = \frac{\partial}{\partial x}\left(1+e^{-x}\right)^{-1} =$$

$$= -(1+e^{-x})^{-2}(-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{x}} \cdot \frac{e^{-x}}{1+e^{-x}}$$

$$= \frac{1}{1+e^{-x}} \cdot \frac{(1+e^{-x})-1}{1+e^{-x}} = \frac{1}{1+e^{-x}} \cdot \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right)$$

$$= \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right) = \sigma(x)\cdot(1-\sigma(x))$$

So remembering previous sigmoid values is used to calculate the derivative of current sigmoids, this can be done is backwards and using forward pass

Question 1.3:

for u it wouldnt be problem since we get $\frac{\partial l}{\partial u} = 2(p-y)m^T = -2ym^T$

however for w_is

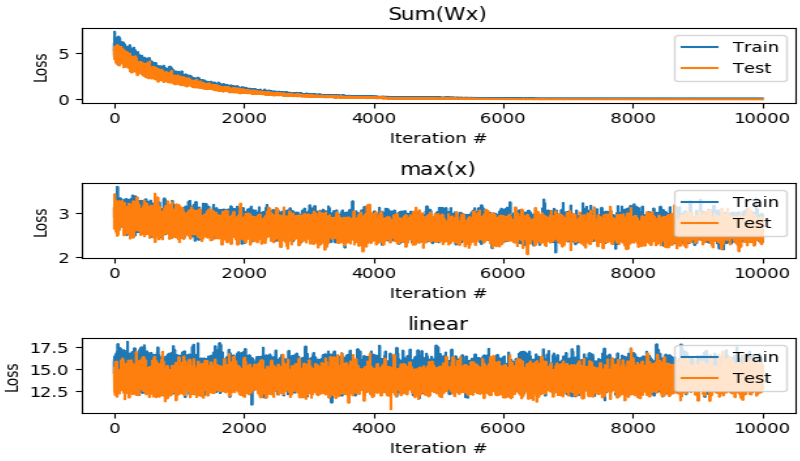
$\frac{\partial l}{\partial w_i} = \frac{\partial l}{\partial p} \frac{\partial p}{\partial m_{3j}} \frac{\partial m_{3j}}{\partial o_{2j}} \frac{\partial o_{2j}}{\partial w_j}$, in particular we saw that $o_{ij}\{x_{i-1}, \ldots, x_{i+1}, w_j\} = \max\{w^T x, 0\}$ and since we initialize w to 0, we get O_{ij}

so $\frac{\partial o_{ij}}{\partial w_i} = 0$ (chain rule) $\Rightarrow \frac{\partial l}{\partial w_i} = 0$, and in updating the value of w_i we get $w_i -= \lambda \frac{\partial l}{\partial w_i} \Rightarrow w_i = 0$

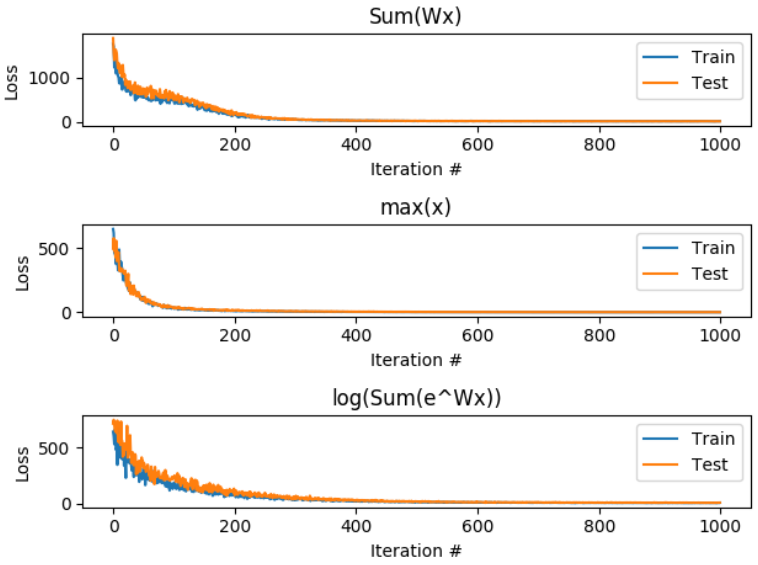
so in second iteration we would get same values of 0, and never change it using random initializations or xavier will solve this.

2.4 Toy Convnet

1) In the linear model we have only learned the first function, the linear one, this result was expected since the model uses only linear operations hence it could not learn non-linear functions.



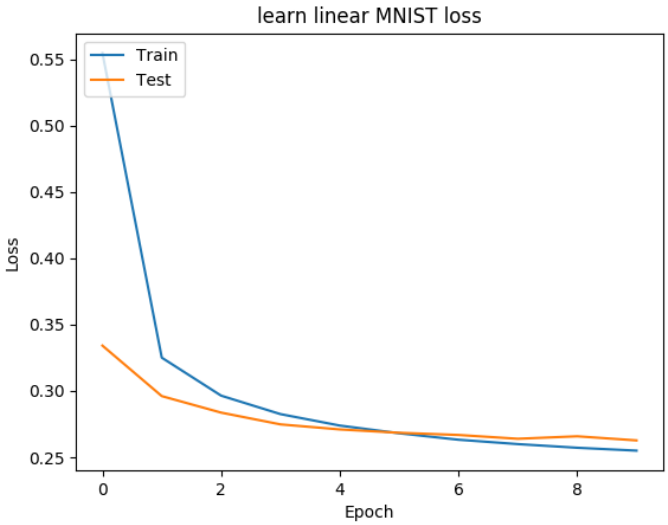
2) In the Toy model we learned all the functions as we see in the graph below, the reason we could learn with this model comes from its non-linearity as we discussed in Tirgul, however the linear model always converges to the optimal solution since its convex, meanwhile our toy model results depends on its weights initializations which is random and doesn't converge to the same optimal solution.

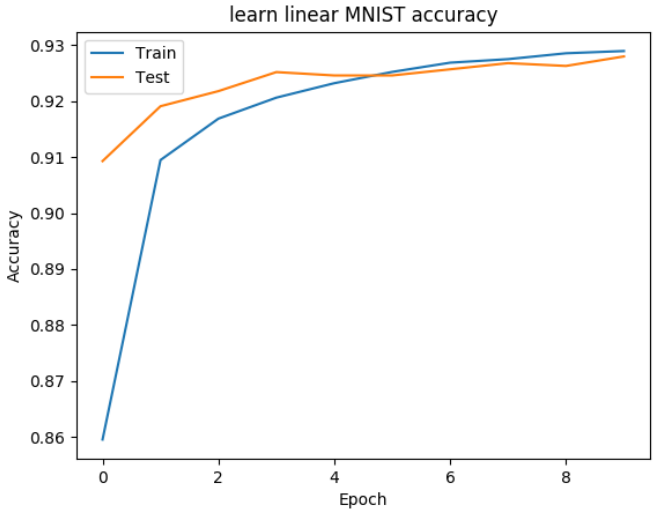


2.5 MNIST Classifier

2.5.1 Linear Model:

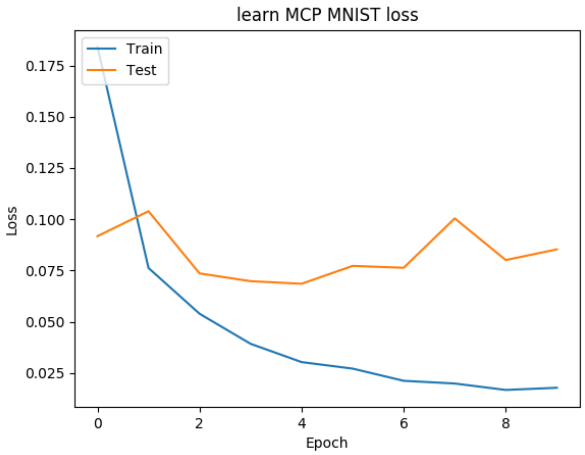
basic linear model, the loss/accuracy converges to optimal values, the phase of convergence might not be very impressive however given the simplicity of this model its good choice for basic tasks.
I choose Adam optimizer based on this, also running SGD with specified momentum gives better results most of the time, however specifying the momentum and decay for batch sizes and number of samples can be much work so I used Adam.(for the next classifiers I will use Adam for same reasons)

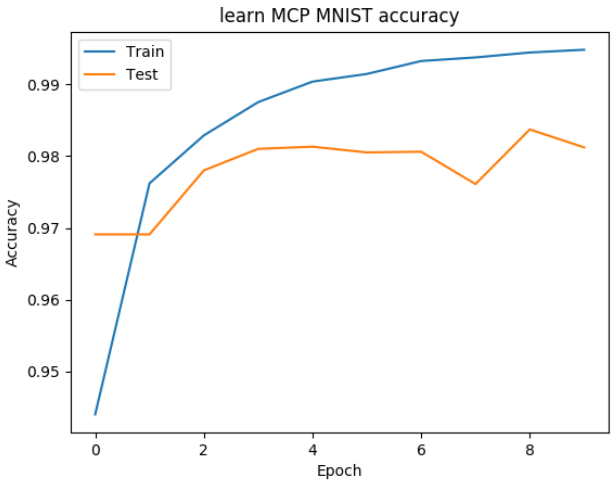




2.5.2 Multi-Layer Perceptron Classifier:

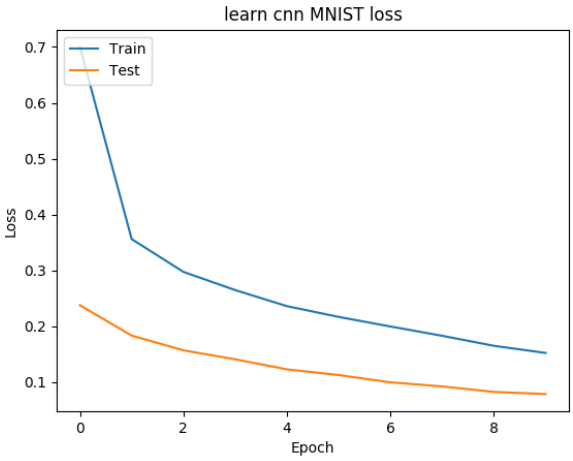
I choose minimal depth for this classifier, I have tried using more deep depths and it wasn't worthy at all given the huge increase in running time and the small increase of learning, I choose to add relu activation for each layer based on this, and when applying sigmoid it give slightly worse results.
Since its an classifying problem in all classifiers we use softmax as an output activation function.

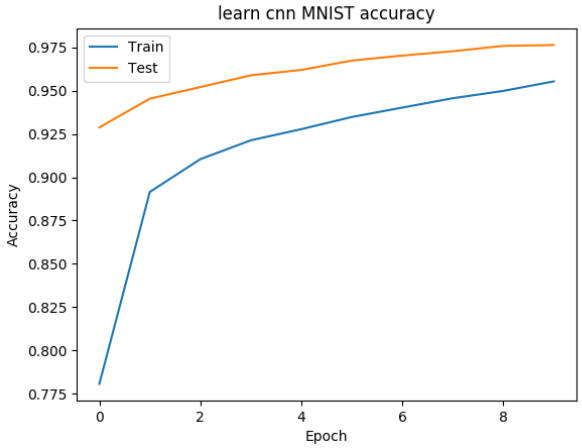




2.5.3 Convnet

Choosing the activation/optimization was done in the same manner as above, in the architecture I used dropouts to get better performance on training, max pooling was applied to reduce dimension and to avoid over-fitting.

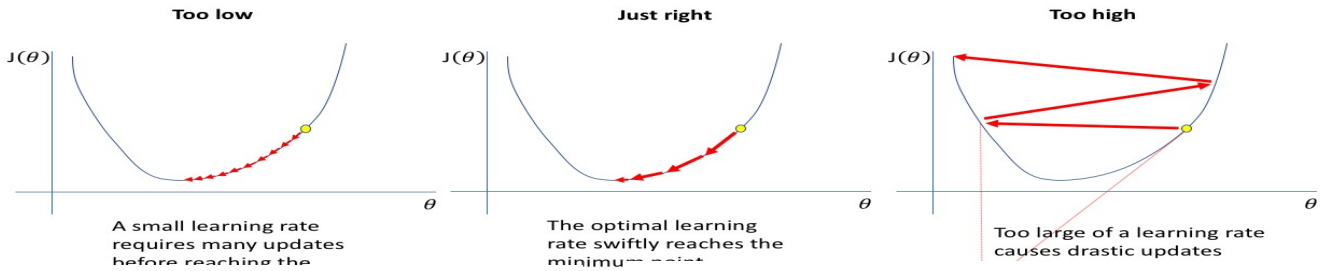


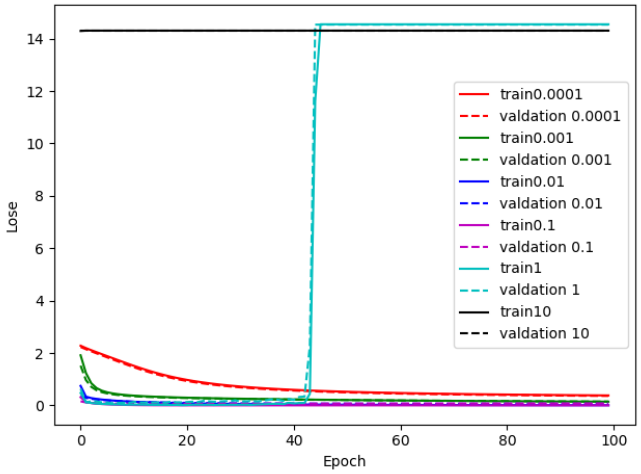


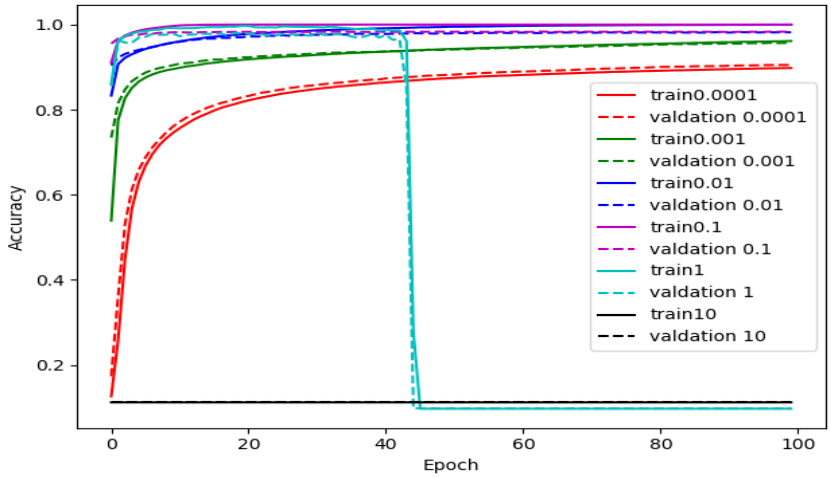
2.5.4 Hyper-Paramter Exploration

I have researched learning rate for MLP, and we can see according to epoch the accuracy kept increasing for low learinng rate values, and for 1 it dropped to 0 around 50 epoch, the reason for this is the same as the reason why for learning rate 10 we had constant 0 accuracy which is the we learn too much that we start missing information, for learnin rate 1 it took a while to happen cause at first we needed to learn almost all information but when we advanced in epoch we needed to start looking at small things and for such high learning rates we cant so the loss increased and accuracy decreased, the same explanation goes for loss, for same learning rates the train and test accuracy/lose didnt differ too much since our algorithm doesn't over-fit.

Illustration to what happened in high learning rates can be seen in most right picture.







2.6 Autoencoders:

I've tried first using two-layers encoding $784 \rightarrow 2$, two-layers decoding, $2 \rightarrow 784$, however it worked but gave very bad results, after some research I have discovered that adding fully-dense layers in middle helps "transport" more features and learning them better since the jump wouldn't be too high, this article helped me out.

As we can see in comparison Autoencoder representation looks more spare, also running my implementation gives better results than PCA