Due Date: February 12th (11pm), 2019

- Name: Dhaivat Jitendra Bhatt
- Date: 09/02/2020
- UdeM ID: 20146667

Problem 2

Note: All the codes to reproduce the report results can be found on this github repository.

Initialization (report) [15] In this sub-question, we consider different initial values for the weight parameters. Set the biases to be zeros, and consider the following settings for the weight parameters:

- **Zero**: all weight parameters are initialized to be zeros (like biases).
- **Normal**: sample the initial weight values from a standard Normal distribution; $w_{i,j} \sim \mathcal{N}(w_{i,j}; 0, 1)$.
- **Glorot**: sample the initial weight values from a uniform distribution; $w^l_{i,j} \sim \mathcal{U}(w^l_{i,j}; -d^l, d^l)$ where $d^l = \sqrt{\frac{6}{h^{l-1}+h^l}}$.

1. Train the model for 10 epochs [1] using the initialization methods above and record the average loss measured on the training data at the end of each epoch (10 values for each setup).

   Answer:

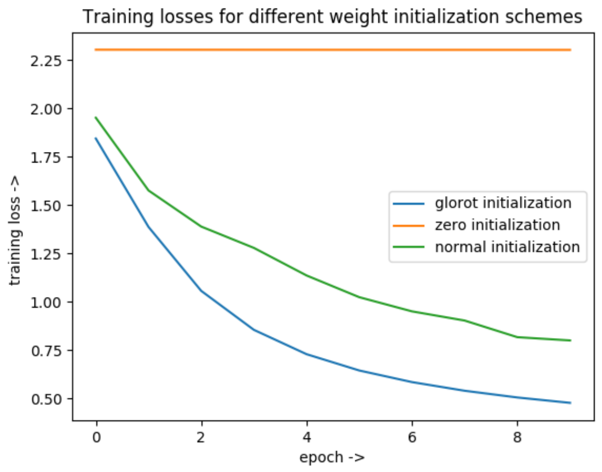
Table 1: The number corresponds to average training loss at every epoch recorded. Corresponding code to reproduce the results can be found here.

Epoch	**Glorot (training loss)**	**Zero (training loss)**	**Normal (training loss)**
1	1.8435794766083398	2.302421418707069	1.9508511146793015
2	1.386797650896656	2.302274667119889	1.5752362356561744
3	1.0566605762797445	2.302274667119889	1.388801946619586
4	0.8552241274677513	2.3020252225102973	1.2785801731179915
5	0.7294575570267213	2.3019195827656724	1.136102644635233
6	0.6453375377356994	2.3018249417288428	1.0239160676441028
7	0.5854366903072762	2.301740167711852	0.9507500528647017
8	0.540656433662026	2.3016642427439344	0.903311692865815
9	0.505921687711295	2.301596251583872	0.8173677021929033
10	0.4782115510088706	2.3015353717321325	0.8006123637784317

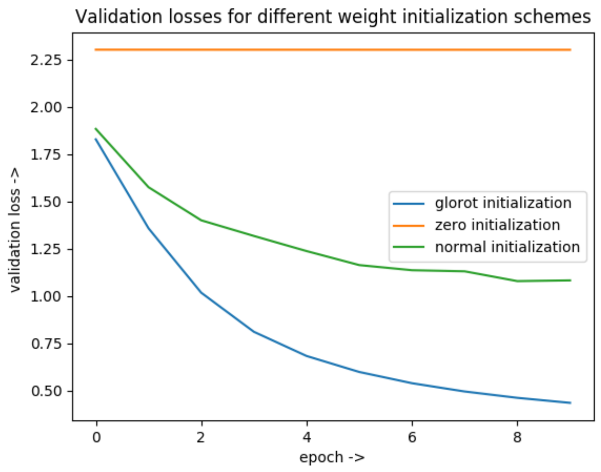
[1] One epoch is one pass through the whole training set.

2. Compare the three setups by plotting the losses against the training time (epoch) and comment on the result.

   Here, we employed 3 different weight initialization techniques and trained the model for 10 epochs.



(a) Training loss profile



(b) Validation loss profile

Figure 1: We can see how loss profile evolves for training and validation data as training progresses. Code to reproduce the image can be found here.

1. Glorot intialization: In Glorot intialization, we are intializing weights by calibrating variance of the weights by using fan-in and fan-out. As evident from the plots, Glorot initialization is a good strategy that drastically reduces training and validation loss as training progresses. The idea is to calibrate variance of the weights

2. Zero initialization: If we initialize all the weights with 0, the network refuses to train itself. One reason for unsuccessful training is that every neuron will end up computing the same output, and since biases are also initialized with 0, the output is going to be 0, when passes through non-linearity(ReLU), the activation function won't fire up and training won't happen. This is evidently visible in 1a. Thus, the validation loss also remains same as no training is happening.

3. Normal distribution: When initializing weights of the neural networks, we don't want weights to be 0, but we want it to be close to 0. In this strategy, we initialize weights with standard normal distribution. That also breaks asymmetry which can be introduced by initializing the weights to 0. While this works well, this still has uncalibrated variance problem, as number of inputs increases, the variance will grow. From the figure, we can clearly see that this method works better than zero initialization but poorer than Glorot initialization. We can also see that validation loss is not reducing as quickly as training loss, pointing towards possibility of overfitting.

Hyperparameter Search (report) [10] From now on, use the Glorot initialization method.

1. Find out a combination of hyper-parameters (model architecture, learning rate, nonlinearity, etc.) such that the average accuracy rate on the validation set ($r^{(valid)}$) is at least 97%.

Answer: We can find multiple different configurations where validation accuracy crosses 97%. For the experiments, the seed was fixed to 500, and the models were trained for 10 epochs for every configuration. Batchsize of 64 was used for all the experiments. Hyperparameter search was done over hidden layer dimensions, learning rate and activation function. Glorot intialization was used to initialize weights. The table below is the set of best hyperparameters for which validation accuracy is above 97%. In table 2, all those configurations are presented.

Table 2

hidden dims	**learning rate**	**activation function**	**training accuracy**	**validation accuracy**	**#model parameters**
(784, 256)	0.05	relu	0.9911	0.9781	818970
(784, 256)	0.05	tanh	0.97832	0.9709	818970
(512, 384)	0.05	relu	0.98894	0.9763	602762
(512, 384)	0.05	tanh	0.97608	0.9705	602762
(666, 333)	0.05	relu	0.9902	0.9764	748261
(666, 333)	0.05	tanh	0.97724	0.97	748261
(666, 666)	0.05	relu	0.98838	0.976	973702

2. Report the hyper-parameters you tried and the corresponding $r^{(valid)}$.

To find optimal set of hyper-parameters, hyperparameters search was performed. Below, possible value of each hyperparameter is mentioned.

(a) hidden_dims = [(784, 256), (512, 384), (666, 333), (666,666)]
(b) activations = [”relu”, ”sigmoid”, ”tanh”]
(c) learning rates = [5e-2, 1e-2, 5e-3]
(d) seed = 500
(e) batch_size = 64
(f) epochs = 10

All the experiments were performed over the space of above mentioned hyperparameters. Extensive results can be seen in table 3. Code to reproduce the results can be found here. From the extensive experimentation and subsequent results in Table 3, we can make several conclusions.

- Sigmoid activation function consistently performs poorly across all configurations. It’s possible that the learning is not happening because of saturating characteristics of Sigmoid function, which could lead to 0 gradients.
- ReLU is able to learn in all the configurations. Infact, for all possible values of hidden dimensions, there is atleast 1 network with ReLU activation that crosses desired validation accuracy of 97%.
- Tanh activation has been able to perform well for a number of configurations.
- In all possible configurations of hidden layer dimensions, we are able to find network which crosses validation accuracy of 97%. Infact, as we increase number of parameters in the model, the performance improvement is not that significant.

Table 3

hidden dims	**learning rate**	**activation function**	**training accuracy**	**validation accuracy**	**#model parameters**
(784, 256)	0.05	relu	0.9911	0.9781	818970
(784, 256)	0.01	relu	0.95728	0.9579	818970
(784, 256)	0.005	relu	0.93828	0.9423	818970
(784, 256)	0.05	sigmoid	0.90488	0.914	818970
(784, 256)	0.01	sigmoid	0.86294	0.8766	818970
(784, 256)	0.005	sigmoid	0.7932	0.8158	818970
(784, 256)	0.05	tanh	0.97832	0.9709	818970
(784, 256)	0.01	tanh	0.93662	0.94	818970
(784, 256)	0.005	tanh	0.92222	0.9256	818970
(512, 384)	0.05	relu	0.98894	0.9763	602762
(512, 384)	0.01	relu	0.9534	0.9557	602762
(512, 384)	0.005	relu	0.9344	0.937	602762
(512, 384)	0.05	sigmoid	0.90516	0.9137	602762
(512, 384)	0.01	sigmoid	0.8631	0.8767	602762
(512, 384)	0.005	sigmoid	0.78956	0.8147	602762
(512, 384)	0.05	tanh	0.97608	0.9705	602762
(512, 384)	0.01	tanh	0.9334	0.9357	602762
(512, 384)	0.005	tanh	0.92018	0.9228	602762
(666, 333)	0.05	relu	0.9902	0.9764	748261
(666, 333)	0.01	relu	0.95472	0.9568	748261
(666, 333)	0.005	relu	0.9361	0.9398	748261
(666, 333)	0.05	sigmoid	0.90464	0.9131	748261
(666, 333)	0.01	sigmoid	0.86386	0.877	748261
(666, 333)	0.005	sigmoid	0.7971	0.8196	748261
(666, 333)	0.05	tanh	0.97724	0.97	748261
(666, 333)	0.01	tanh	0.9356	0.9386	748261
(666, 333)	0.005	tanh	0.92154	0.9234	748261
(666, 666)	0.05	relu	0.98838	0.976	973702
(666, 666)	0.01	relu	0.9542	0.9564	973702
(666, 666)	0.005	relu	0.93608	0.9392	973702
(666, 666)	0.05	sigmoid	0.9004	0.9094	973702
(666, 666)	0.01	sigmoid	0.8606	0.8746	973702
(666, 666)	0.005	sigmoid	0.7996	0.8256	973702
(666, 666)	0.05	tanh	0.9741	0.9697	973702
(666, 666)	0.01	tanh	0.93172	0.9347	973702
(666, 666)	0.005	tanh	0.92014	0.9242	973702

Validate Gradients using Finite Difference (report) [15] The finite difference gradient approximation of a scalar function $x \in \mathbb{R} \mapsto f(x) \in \mathbb{R}$, of precision ε, is defined as $\frac{f(x+\varepsilon)-f(x-\varepsilon)}{2\varepsilon}$. Consider the second layer weights of the MLP you built in the previous section, as a vector $\theta = (\theta_1, \ldots, \theta_m)$. We are interested in approximating the gradient of the loss function L, evaluated using **one** training sample, at the end of

Table 4: We show finite difference between analytical gradient and numerical gradient in this table. For various values of N, we can see the gradient difference values. As we move towards larger value of N, we can see the gradient difference reducing. The 10 numbers below value of N are difference between analytical and numerical gradients for the 10 weights being considered.

N = 1	N = 50	N = 1000	N = 50000	N = 500000
-3.79e-02	-3.52e-02	0.000e+00	0.000e+00	0.000e+00
3.233e-02	3.573e-09	8.867e-12	3.316e-12	-6.329e-11
-1.586e-02	3.506e-09	8.721e-12	9.499e-13	-9.897e-11
3.560e-02	8.535e-09	2.159e-11	3.612e-12	2.581e-11
-1.366e-02	0.000e+00	0.000e+00	0.00e+00	0.00e+00
-3.598e-02	0.000e+00	0.000e+00	0.000e+00	0.000e+00
3.9924e-02	6.032e-09	1.518e-11	7.414e-12	6.292e-11
-1.582e-02	4.211e-09	1.065e-11	-4.447e-12	-2.66e-11
-2.870e-02	-1.850e-09	-4.547e-12	-5.436e-12	-7.204e-11
2.755e-02	-6.391e-09	-1.597e-11	2.674e-12	-8.428e-12

training, with respect to $\theta_{1:p}$, the first $p = \min(10, m)$ elements of θ, using finite differences.

1. Evaluate the finite difference gradients $\nabla^N \in \mathbb{R}^p$ using $\varepsilon = \frac{1}{N}$ for different values of N

$$\nabla_i^N = \frac{L(\theta_1, \ldots, \theta_{i-1}, \theta_i + \varepsilon, \theta_{i+1}, \ldots, \theta_p) - L(\theta_1, \ldots, \theta_{i-1}, \theta_i - \varepsilon, \theta_{i+1}, \ldots, \theta_p)}{2\varepsilon}$$

   Use at least 5 values of N from the set $\{k10^i : \ i \in \{0, \ldots, 5\}, \ k \in \{1, 5\}\}$.

2. Plot the maximum difference between the true gradient and the finite difference gradient ($\max_{1 \leq i \leq p} |\nabla_i^N - \frac{\partial L}{\partial \theta_i}|$) as a function of N. Comment on the result.

   - As we can see, as value of N increases, the finite difference between analytical gradient and numerical gradient reduces. The reason is that as value of ε gets smaller and smaller, the numerical gradient comes more closer to that of analytic gradient. As $\varepsilon \to 0$, we get value very close to that of gradient coming through analytic formula(through differentiation). Hence, we can clearly observe the phenomena of reduction in error as we increase value of N. In figure 2, the plot is for 12 different values of N as asked.

Convolutional Neural Networks (report) [25] Many techniques correspond to incorporating certain prior knowledge of the structure of the data into the parameterization of the model. Convolution operation, for example, was originally designed for visual imagery. For this part of the assignment we will train a convolutional network on MNIST for 10 epochs using PyTorch. Plot the train and valid errors at the end of each epoch for the model.

1. Come up with a CNN architecture with more or less similar number of parameters as MLP trained in Problem 1 and describe it.

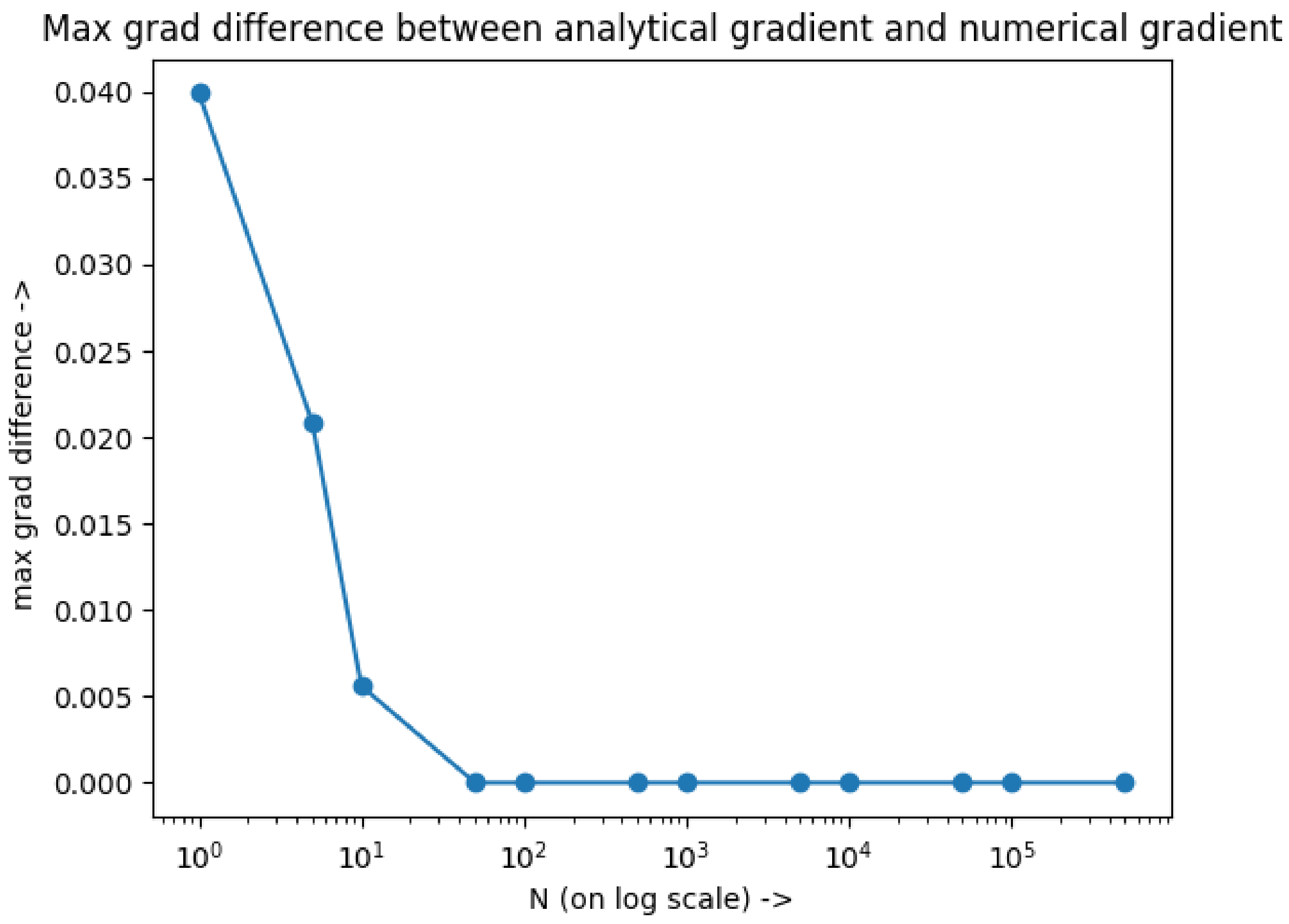
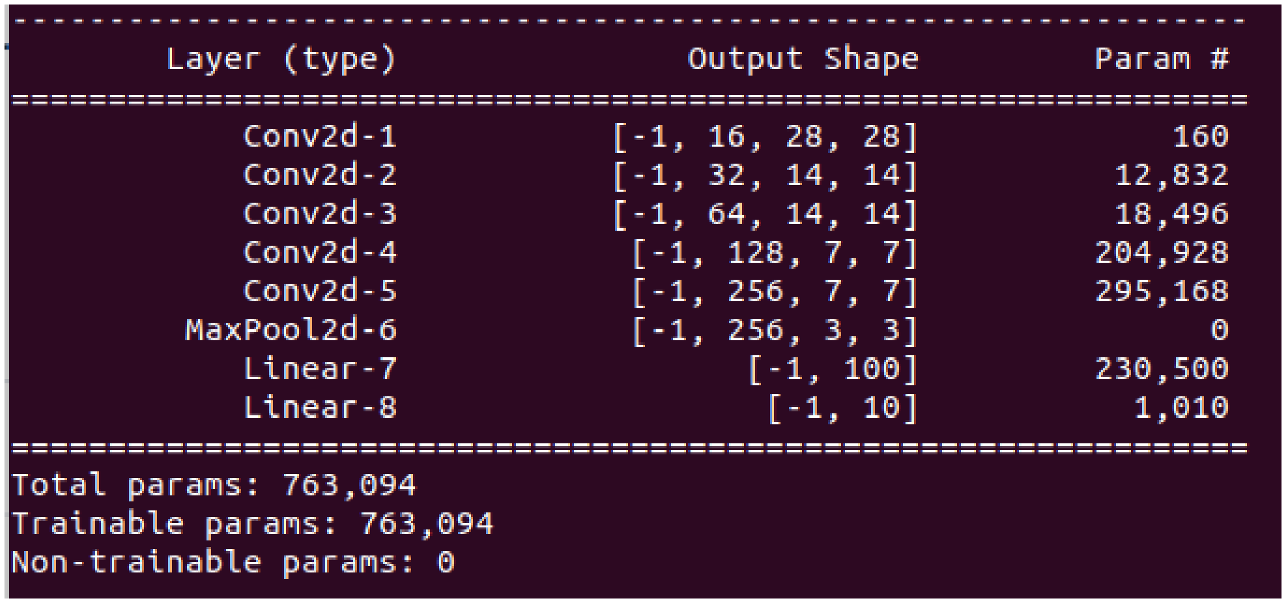


Figure 2

- The CNN model consists of 5 convolution layers followed by a Max pooling layer and 2 fully connected layers. Square kernels are used for all convolutional layers. Kernel sizes from layer 1 to 5 are 3, 5, 3, 5, 3 respectively. Stride of 1 and padding of 1 was used for all layers with filter size 3. For layer-2 and layer-4, kernel size of 5 was used with stride of 2 and padding of 2, to downsample the feature-map by half. Eventually a maxpool 2d layer was used to reduce input resolution by half before going into fully connected layers. The final fully connected layer outputs a 10x1 vector for 10 classes. This network contains 0.76 million parameters and achieves test accuracy of 98.31% on MNIST test dataset.

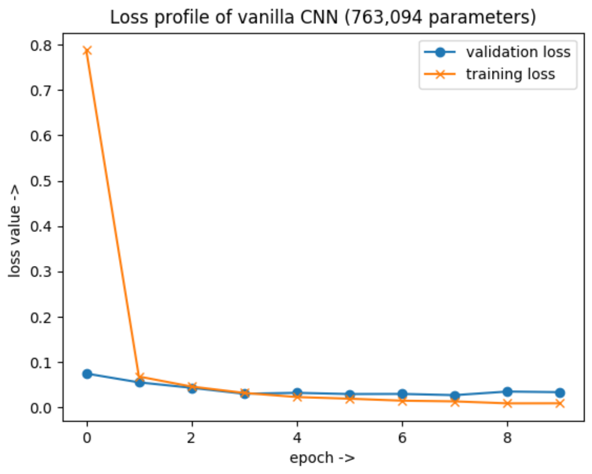
2. Compare the performances of CNN vs MLP. Plot the training loss and validation loss curve.

   - CNN and MLP perform fairly well over the validation data. In figure 4, we can see loss profile for MLP and Vanila CNN with similar number of parameters. The Vanila CNN is the one presented in previous question. The results in the graph can be reproduced through this code.

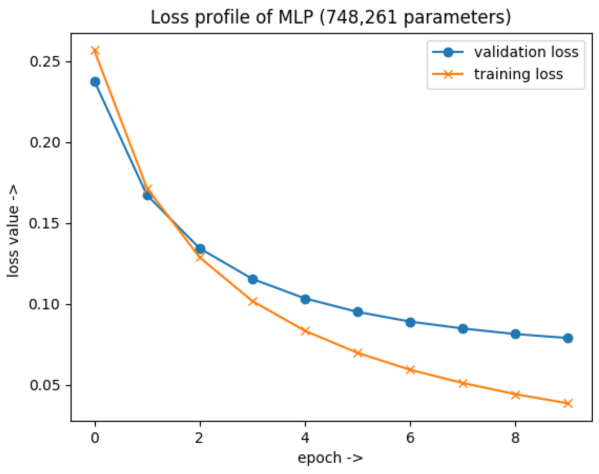


```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 16, 28, 28]             160
            Conv2d-2           [-1, 32, 14, 14]          12,832
            Conv2d-3           [-1, 64, 14, 14]          18,496
            Conv2d-4            [-1, 128, 7, 7]         204,928
            Conv2d-5            [-1, 256, 7, 7]         295,168
         MaxPool2d-6            [-1, 256, 3, 3]               0
            Linear-7                  [-1, 100]         230,500
            Linear-8                   [-1, 10]           1,010
================================================================
Total params: 763,094
Trainable params: 763,094
Non-trainable params: 0
```

Figure 3: The figure shows full architecture of the CNN used to train and classify MNIST dataset. As observed from the model summary, we can see that number of parameters in the model are in the range [0.5M, 1M].



(a) Loss profile of Vanilla CNN

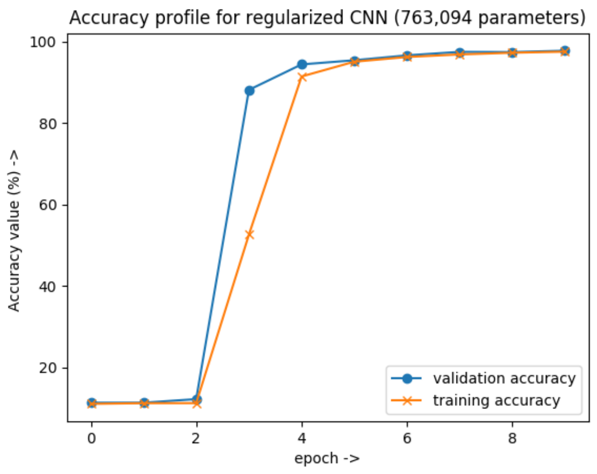


(b) Loss profile of MLP

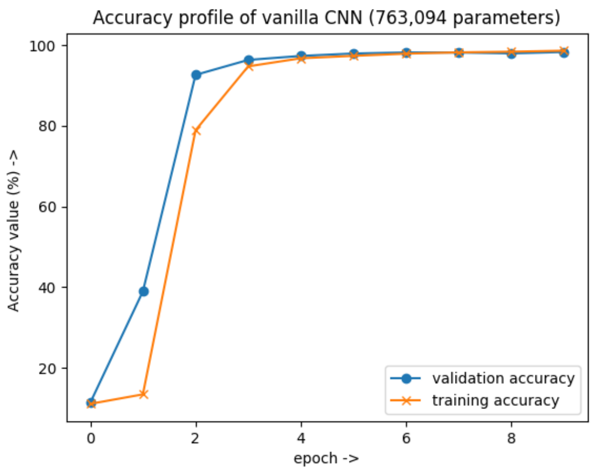
Figure 4: The vanila CNN plot here corresponds to the model in the previous answer.

3. Explore *one* regularization technique you've seen in class (e.g. early stopping, weight decay, dropout, noise injection, etc.), and compare the performance with a vanila CNN model without regularization.

   Here, I employ L2 regularization technique. L2 regularization technique reduces accuracy of equivalent vanila CNN classifier because of the additional constraints put on the model parameters. L2 regularization forces the weights to behave in certain fashion that reduces overall accuracy of the system. Below, we can see the performance of Vanila CNN and regularized CNN. Vanila CNN is able to achieve validation accuracy of 98.31% while regularized CNN stands slightly behind with validation accuracy at the end of training being 97.83%. Code corresponding to this experiment



(a) Accuracy profile of Vanilla CNN



(b) Accuracy profile of regularized CNN