Assignment_3

June 10, 2022

0.1 Computer vision 2022 Assignment 3: Deep Learning for Perception Tasks

This assignment contains 2 questions. The first question gives you a basic understanding of the classifier. The second question requires you to write a simple proposal.

1 Question 1: A simple classifier (60%)

For this exercise, we will provide a demo code showing how to train a network on a small dataset called FashionMinst. Please go through the following tutorials first. You will get a basic understanding about how to train an image classification network in pytorch. You can change the training scheme and the network structure. Please answer the following questions then. You can orginaze your own text and code cell to show the answer of each questions.

Note: Please plot the loss curve for each experiment (2 point).

Requirement:

Q1.1 (1 point) Change the learning rate and train for 10 epochs. Fill this table:

Lr	Accuracy
1	
0.1	
0.01	
0.001	

Q1.2 (2 point) Report the number of epochs when the network is converged. Hint: The network is called "converged" when the accuracy is not changed (or the change is smaller than a threshold).

Fill this table:

Lr	Accuracy	Epoch
1		
0.1		
0.01		
0.001		

Q1.3 (2 points) Compare the results in table 1 and table 2, what is your observation and your understanding of learning rate?

Q1.4 (3 point) Build a deeper/ wider network. Report the accuracy and the parameters for each structure. Parameters represent the number of trainable parameters in your model, e.g. a 3 x 3 conv has 9 parameters.

Structures	Accuracy	Parameters
Base		
Deeper		
Wider		

Q1.5 (2 points) Choose to do one of the following two tasks:

a. Write a code to calculate the parameter and expian the code.

OR

b. Write done the process of how to calculate the parameters by hand.

Q1.6 (1 points) What are your observations and conclusions for changing network structure?

Q1.7 (2 points) Calculate the mean of the gradients of the loss to all trainable parameters. Plot the gradients curve for the first 100 training steps. What are your observations? Note that this gradients will be saved with the training weight automatically after you call loss.backwards(). Hint: the mean of the gradients should be decreased.

For more exlanation of q1.7, you could refer to the following simple instructions: https://colab.research.google.com/drive/1XAsyNegGSvMf3_B6MrsXht7-fHqtJ7OW?usp=sharing

```
Mounted at /content/drive
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting torchinfo
  Downloading torchinfo-1.7.0-py3-none-any.whl (22 kB)
Installing collected packages: torchinfo
Successfully installed torchinfo-1.7.0
```

```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-
images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-
images-idx3-ubyte.gz to data/FashionMNIST/raw/train-images-idx3-ubyte.gz
```

```
  0%|          | 0/26421880 [00:00<?, ?it/s]
```

```
Extracting data/FashionMNIST/raw/train-images-idx3-ubyte.gz to
data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-
labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-
labels-idx1-ubyte.gz to data/FashionMNIST/raw/train-labels-idx1-ubyte.gz
```

```
  0%|          | 0/29515 [00:00<?, ?it/s]
```

```
Extracting data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to
data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-
central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-
central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to
data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz
```

```
  0%|          | 0/4422102 [00:00<?, ?it/s]
```

```
Extracting data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to
data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-
central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-
central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to
data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz
```

```
  0%|          | 0/5148 [00:00<?, ?it/s]
```

```
Extracting data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to
data/FashionMNIST/raw
```

We pass the Dataset as an argument to DataLoader. This wraps an iterable over our dataset and supports automatic batching, sampling, shuffling, and multiprocess data loading. Here we define a batch size of 64, i.e. each element in the dataloader iterable will return a batch of 64 features and labels.

```
Shape of X [N, C, H, W]:  torch.Size([64, 1, 28, 28])
Shape of y:  torch.Size([64]) torch.int64
```

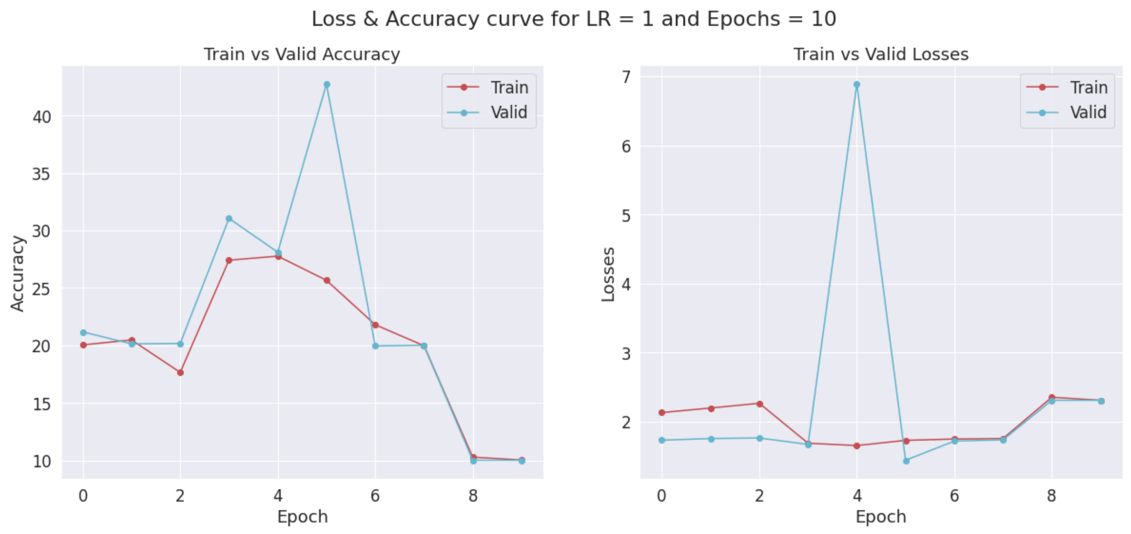
To define a neural network in PyTorch, we create a class that inherits from nn.Module. We define the layers of the network in the init function and specify how data will pass through the network in the forward function. To accelerate operations in the neural network, we move it to the GPU if available.

```
Using cpu device
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

In a single training loop, the model makes predictions on the training dataset (fed to it in batches), and backpropagates the prediction error to adjust the model's parameters.

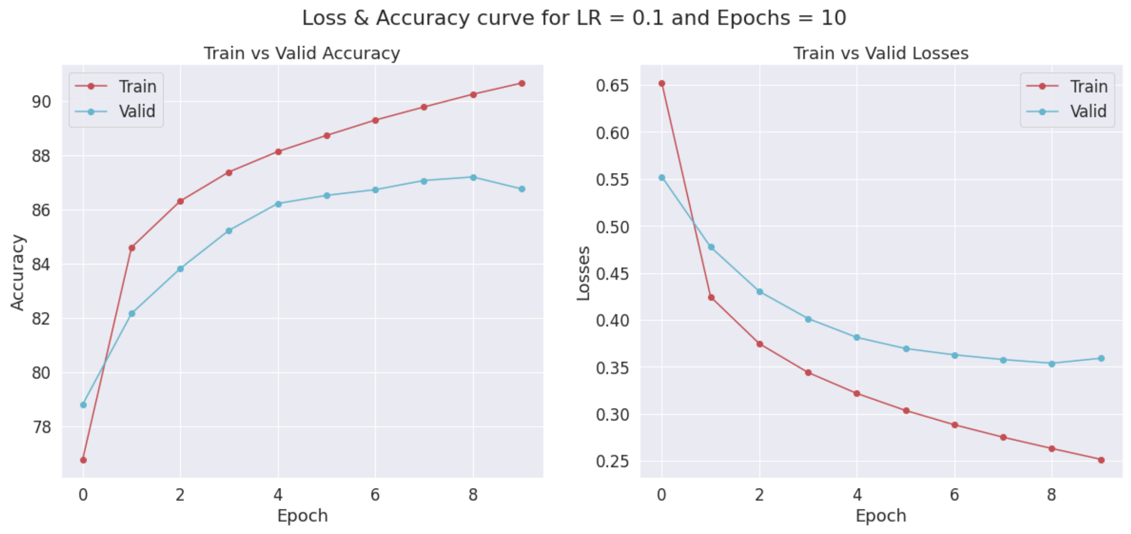
Q1.1 Change the learning rate and train for 10 epochs.

`Learning Rate: 1, Number of Epochs: 10 :`



Epochs	Accuracy	Avg Loss
1	20.3%	1.659739
2	21.6%	1.651419
3	16.0%	1.750029
4	27.0%	1.754452
5	28.6%	6.795542
6	26.9%	1.720640
7	23.8%	1.728050
8	20.0%	1.720005
9	12.9%	2.712244
10	11.0%	2.332313
Avg Accuracy:	21.81%	

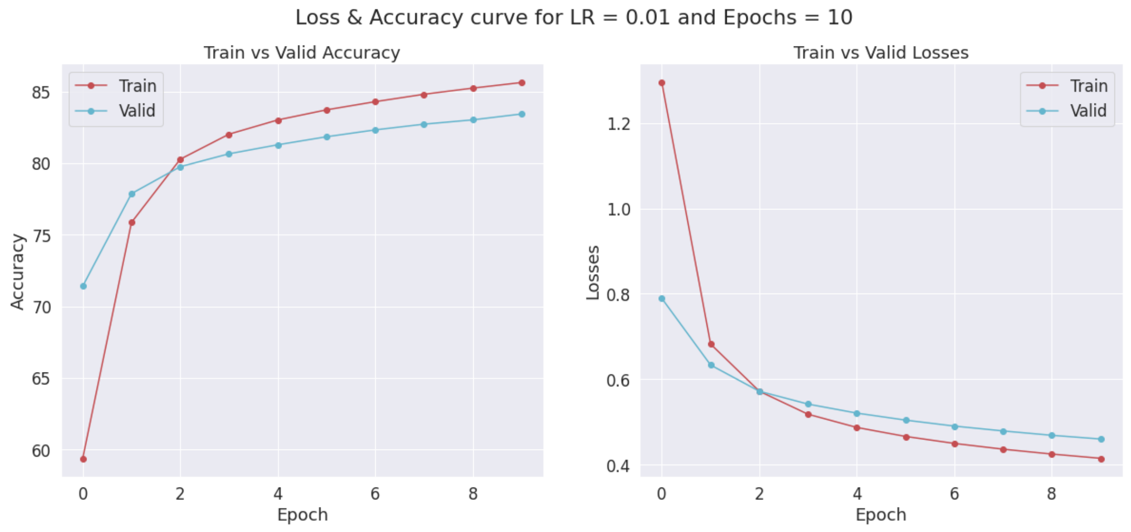
`Learning Rate: 0.1, Number of Epochs: 10 :`



Epochs	Accuracy	Avg Loss
1	78.8%	0.552627
2	81.8%	0.478842
3	83.5%	0.438541
4	84.7%	0.416869
5	84.9%	0.404906
6	85.6%	0.386635
7	86.4%	0.366515
8	86.8%	0.358582
9	87.1%	0.352089
10	87.2%	0.347195
Avg Accuracy:	84.68%	

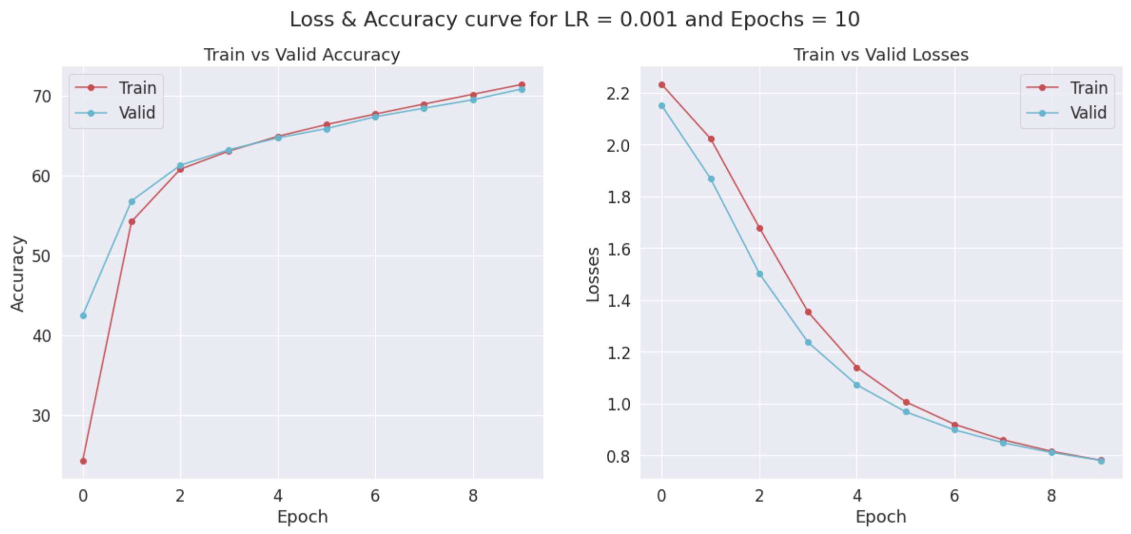
```
--------------------------------------------------------------------------------
----------------------
```

`Learning Rate: 0.01, Number of Epochs: 10 :`



Epochs	Accuracy	Avg Loss
1	71.4%	0.789093
2	78.1%	0.630266
3	80.0%	0.567738
4	80.6%	0.537489
5	81.1%	0.518184
6	81.7%	0.503154
7	82.1%	0.491174
8	82.5%	0.479639
9	82.9%	0.468684
10	83.4%	0.460194
Avg Accuracy:	80.38%	

```
--------------------------------------------------------------------------------
----------------------

Learning Rate: 0.001, Number of Epochs: 10 :
```

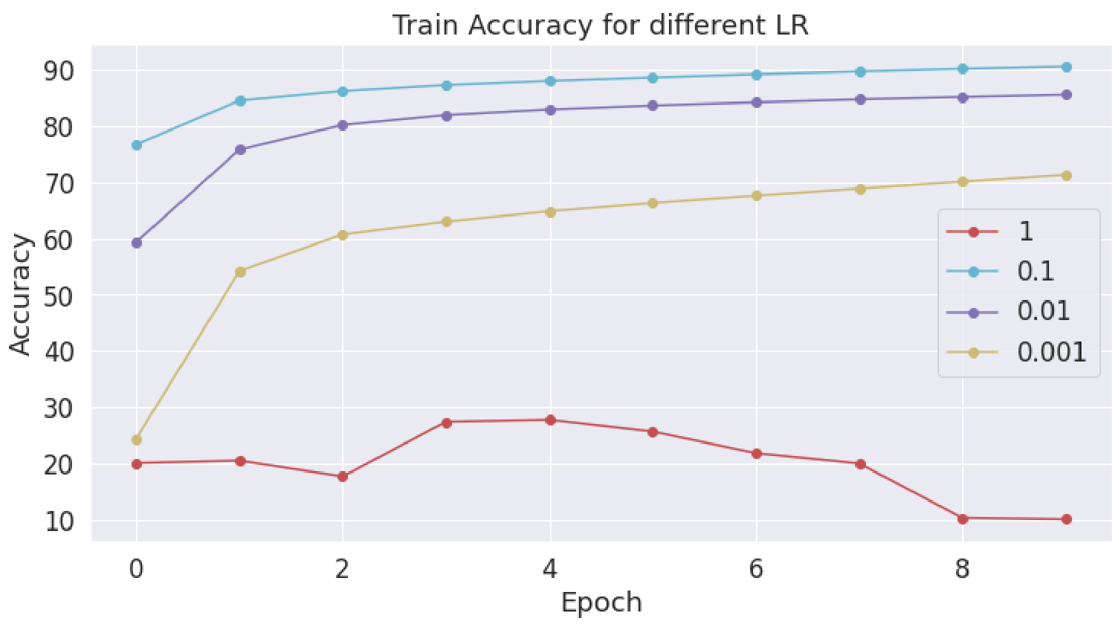


Epochs	Accuracy	Avg Loss
1	48.5%	2.168470
2	57.0%	1.909972
3	59.6%	1.533187
4	62.9%	1.257567
5	64.8%	1.089994
6	66.0%	0.982977
7	67.1%	0.910346
8	68.4%	0.858210
9	69.5%	0.818771
10	70.5%	0.787442
Avg Accuracy:	63.43%	

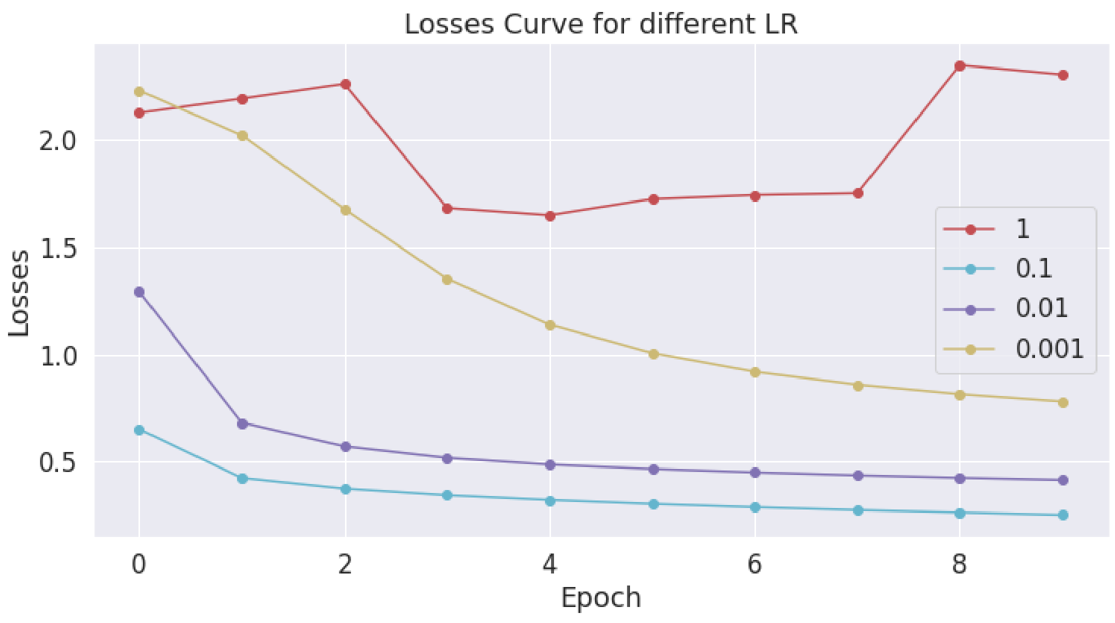
```
-------------------------------------------------------------------------
--------------------

Overal Results:
```

Lr	Accuracy
1	21.81%
0.1	84.68%
0.01	80.38%
0.001	63.43%

```
-------------------------------------------------------------------------
--------------------

Overall accuray comparison :
```



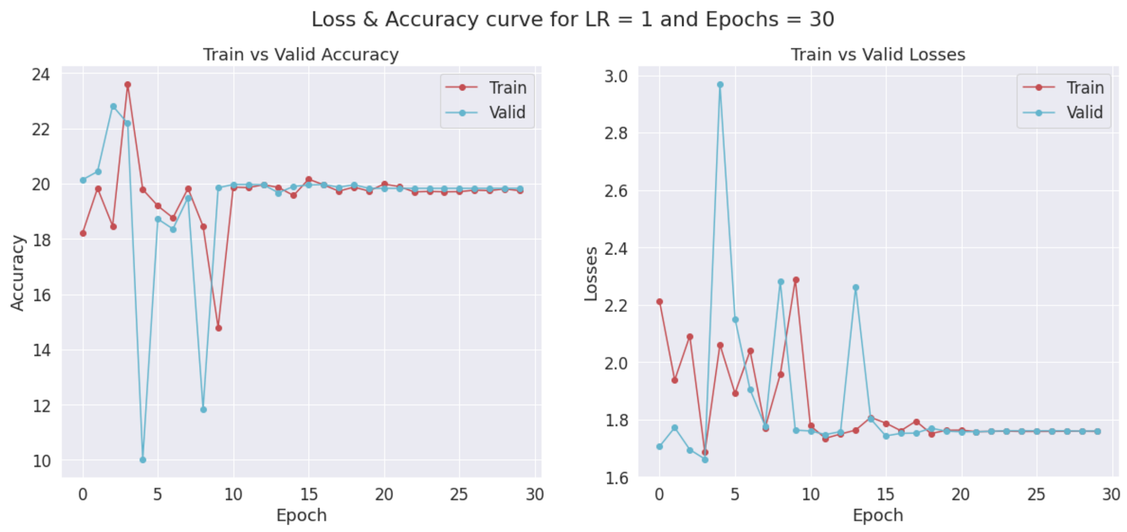
```
--------------------------------------------------------------------------------
---------------------

Overall loss comparison :
```



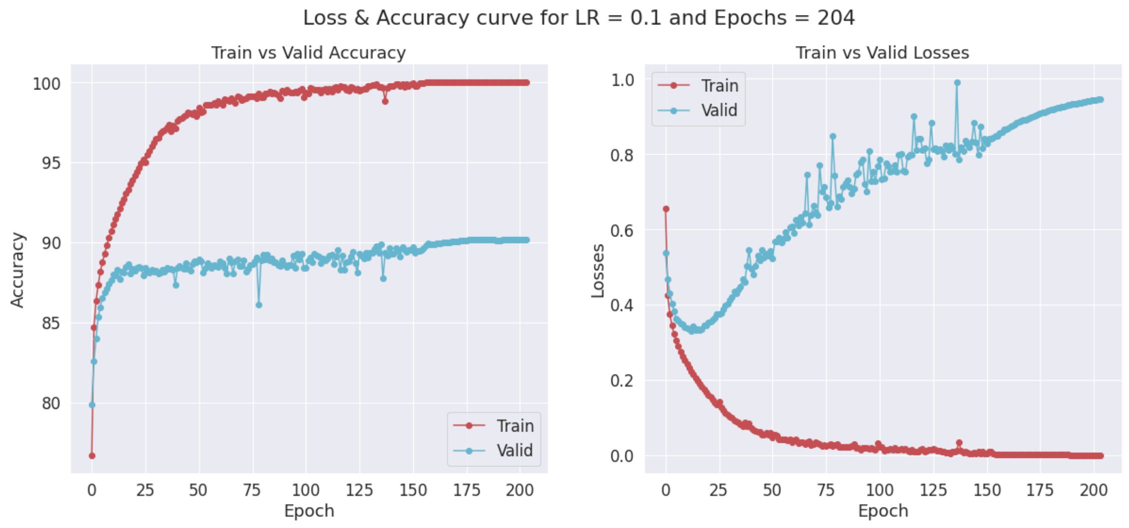
```
--------------------------------------------------------------------------------
---------------------
```

Q1.2 Report the number of epochs when the network is converged. Hint: The network is called "converged" when the accuracy is not changed (or the change is smaller than a threshold).

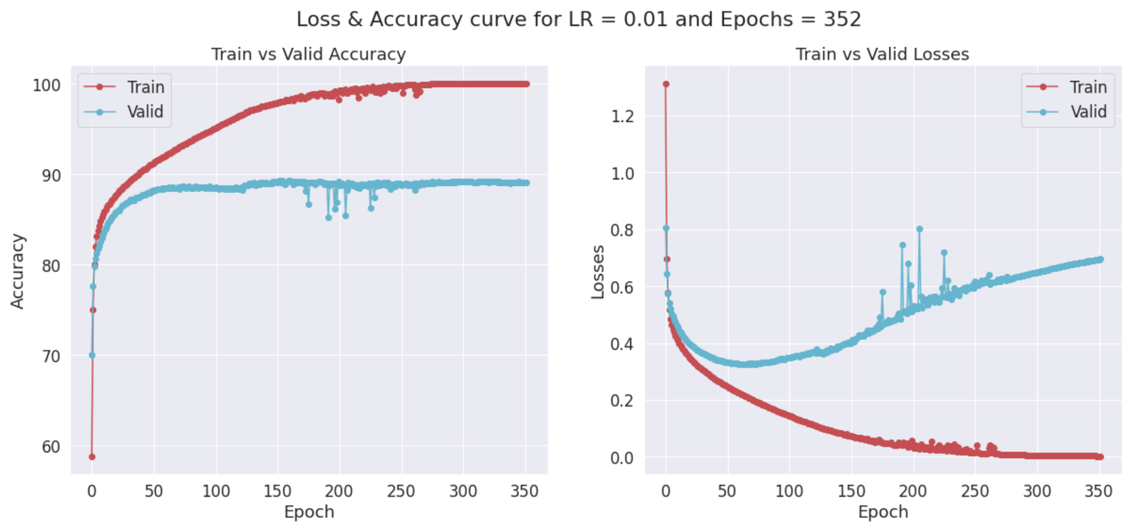
```
The network is converged after 30 epochs, the accuracy is 19.830000000000002
```



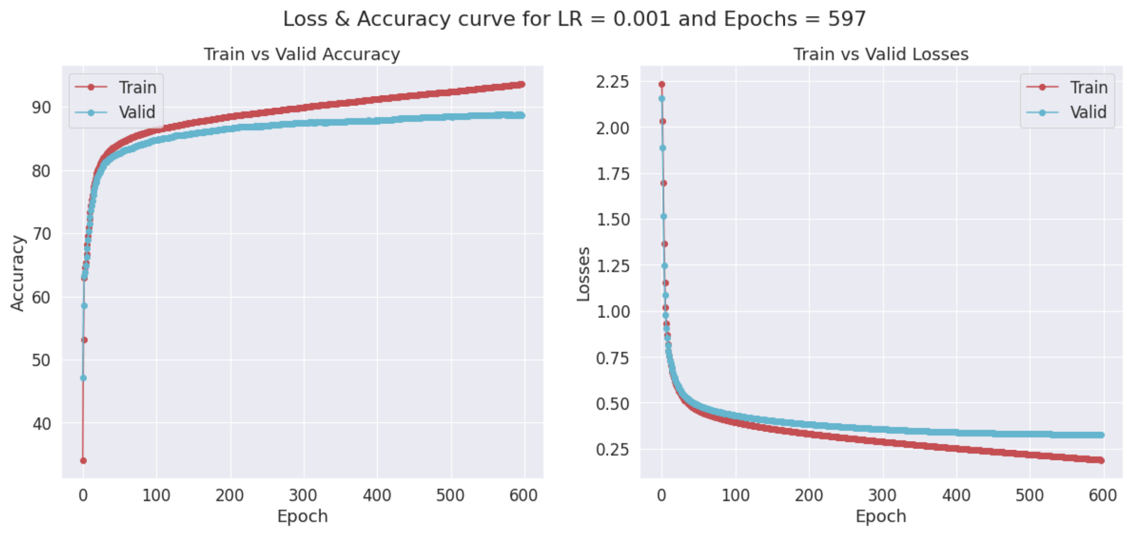
```
The network is converged after 204 epochs, the accuracy is 90.18
```



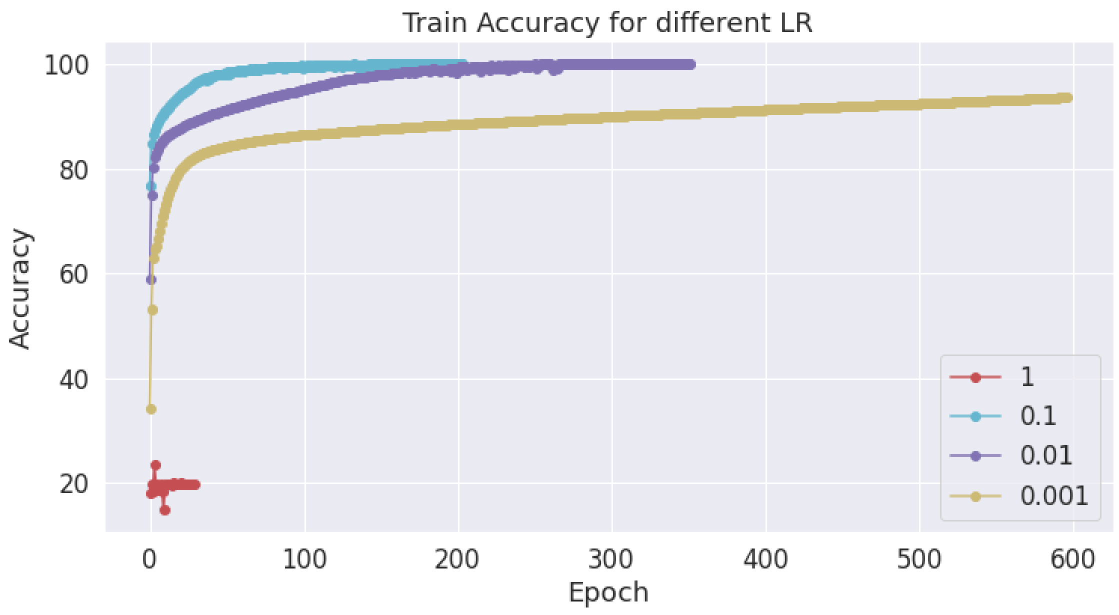
```
The network is converged after 352 epochs, the accuracy is 89.13
```



```
The network is converged after 597 epochs, the accuracy is 88.73
```

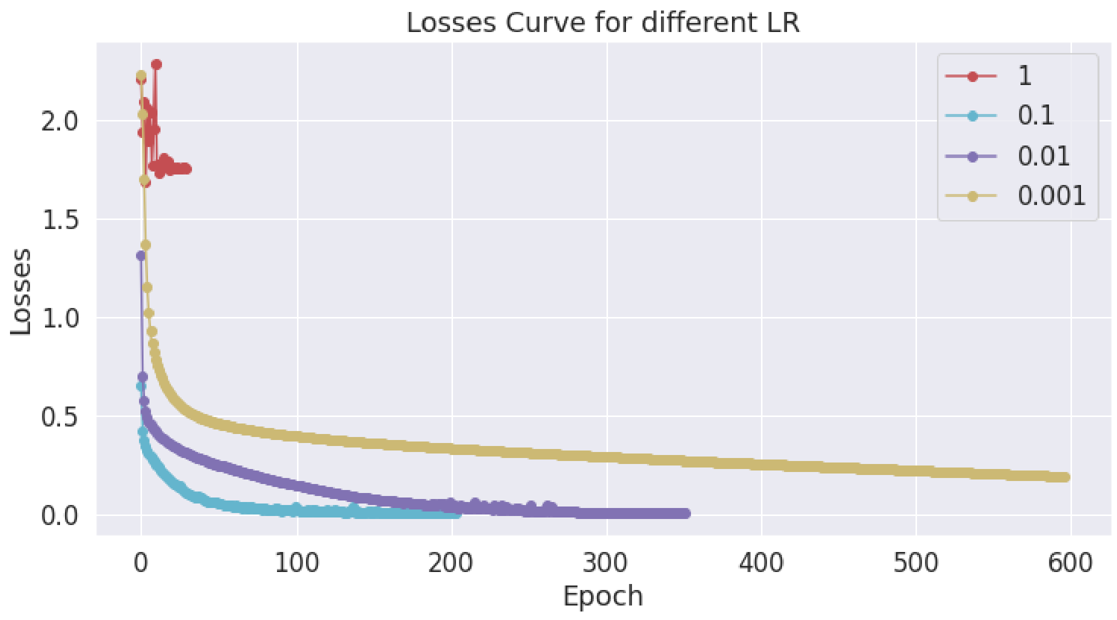


```
--------------------------------------------------------------------------------
----------------------

Overall accuracy comparison :
```



```
--------------------------------------------------------------------------------
----------------------
```

```
Overall loss comparison :
```



```
--------------------------------------------------------------------------------
----------------------
```

Q1.3 Compare the results in table 1 and table 2, what is your observation and your understanding of learning rate?

What I observed:

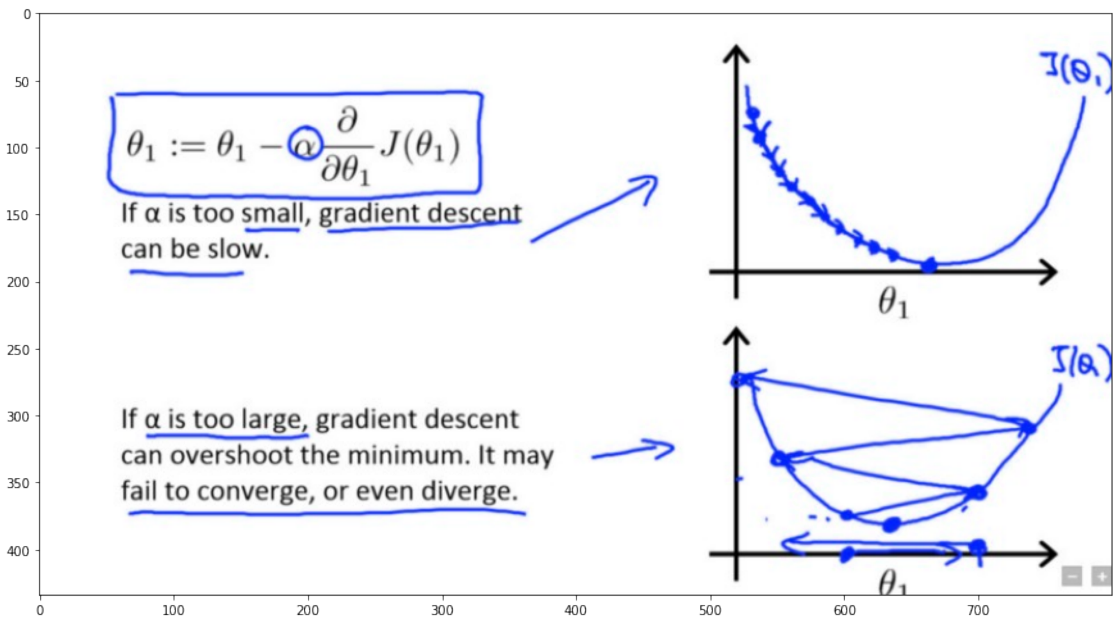
A neural network learns or approximates a function to best map inputs to outputs from examples in the training dataset. The learning rate hyperparameter controls the rate or speed at which the model learns. Specifically, it controls the amount of apportioned error that the weights of the model are updated with each time they are updated, such as at the end of each batch of training examples.During training, the backpropagation of error estimates the amount of error for which the weights of a node in the network are responsible. Instead of updating the weight with the full amount, it is scaled by the learning rate.

This means that a learning rate of 0.1, would mean that weights in the network are updated 0.1 * (estimated weight error) or 10% of the estimated weight error each time the weights are updated. Generally, a large learning rate allows the model to learn faster, at the cost of arriving on a sub-optimal final set of weights. A smaller learning rate may allow the model to learn a more optimal or even globally optimal set of weights but may take significantly longer to train.

At extremes, a learning rate that is too large will result in weight updates that will be too large and the performance of the model (such as its loss on the training dataset) will oscillate over training epochs. Oscillating performance is said to be caused by weights that diverge (are divergent). A learning rate that is too small may never converge or may get stuck on a suboptimal solution.

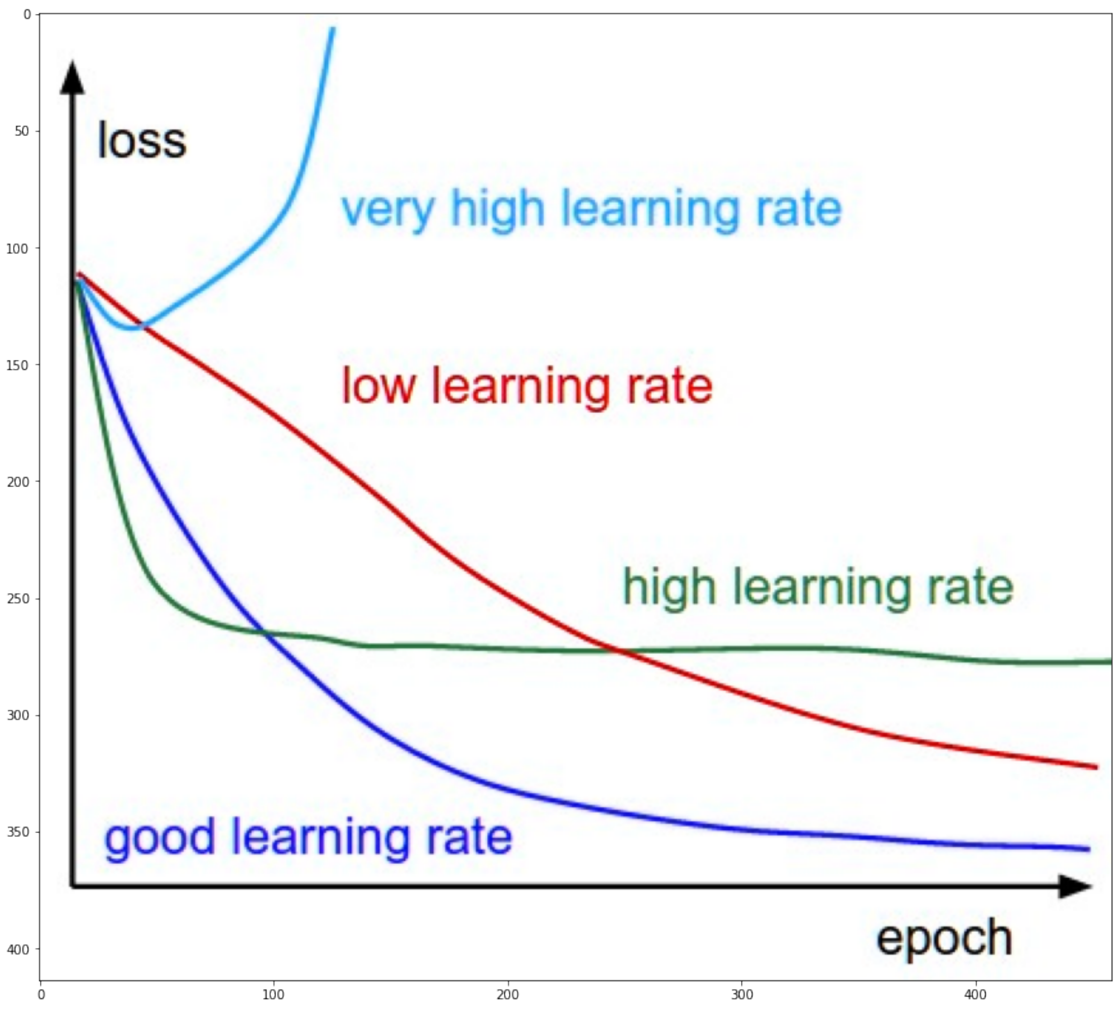
The figure below will show the gradient desent with small and large learning rates.

```
<matplotlib.image.AxesImage at 0x7fae9e894590>
```


$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$
If α is too small, gradient descent can be slow.
If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.
J(θ₁)
J(α)
θ₁
θ₁

The below diagram demonstrates the different scenarios one can fall into when configuring the learning rate.

```
<matplotlib.image.AxesImage at 0x7fae9e80ae50>
```

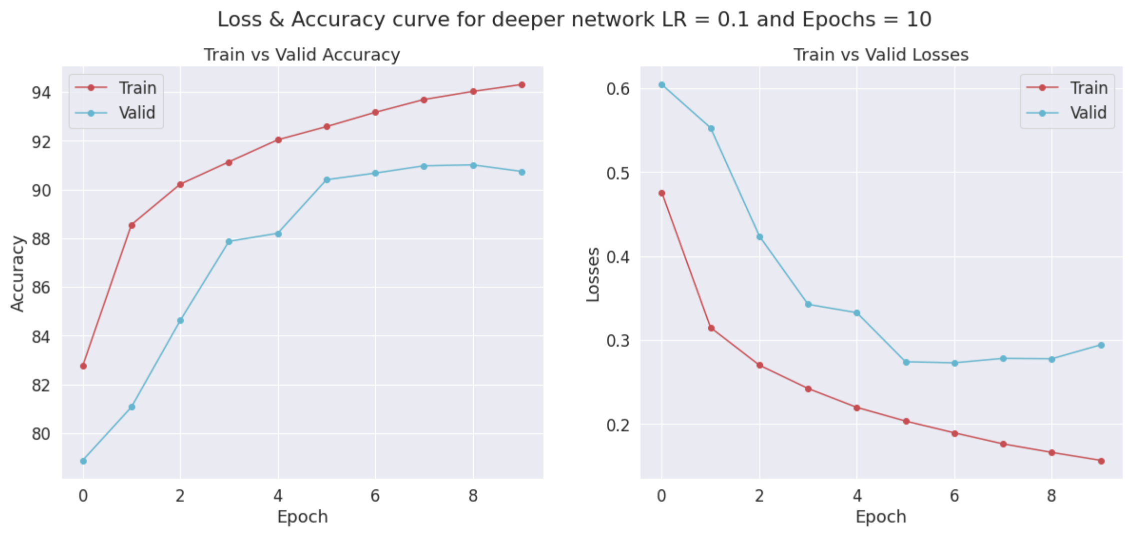


Q1.4 Build a deeper/ wider network. Report the accuracy and the parameters for each structure. Parameters represent the number of trainable parameters in your model, e.g. a 3 x 3 conv has 9 parameters.

The deeper netwrok structure shown below contains several layers, 1 Convolutional layer, 1 baych-

Norm2d layer, 1 Relu layer and one Maxpool layer are included in the first sequential, and same layer strcture has also been included in the second sequential. Lastly, 3 Linear layers and 1 Dropout layser have been included. The total number of trainable paramters found in the deeper network is 1,475,338. The average accuracy found to be 81.79% when the learning rate is 0.1 and epoches is 10.

```
================================================================================
==========
Layer (type:depth-idx)                    Output Shape              Param #
================================================================================
==========
NeuralNetwork                             --                        --
 Sequential: 1-1                         [64, 32, 14, 14]          --
     Conv2d: 2-1                         [64, 32, 28, 28]          320
     BatchNorm2d: 2-2                    [64, 32, 28, 28]          64
     ReLU: 2-3                           [64, 32, 28, 28]          --
     MaxPool2d: 2-4                      [64, 32, 14, 14]          --
 Sequential: 1-2                         [64, 64, 6, 6]            --
     Conv2d: 2-5                         [64, 64, 12, 12]          18,496
     BatchNorm2d: 2-6                    [64, 64, 12, 12]          128
     ReLU: 2-7                           [64, 64, 12, 12]          --
     MaxPool2d: 2-8                      [64, 64, 6, 6]            --
 Linear: 1-3                             [64, 600]                 1,383,000
 Dropout: 1-4                            [64, 600]                 --
 Linear: 1-5                             [64, 120]                 72,120
 Linear: 1-6                             [64, 10]                  1,210
================================================================================
==========
Total params: 1,475,338
Trainable params: 1,475,338
Non-trainable params: 0
Total mult-adds (M): 279.73
================================================================================
==========
Input size (MB): 0.20
Forward/backward pass size (MB): 35.50
Params size (MB): 5.90
Estimated Total Size (MB): 41.60
================================================================================
==========
```

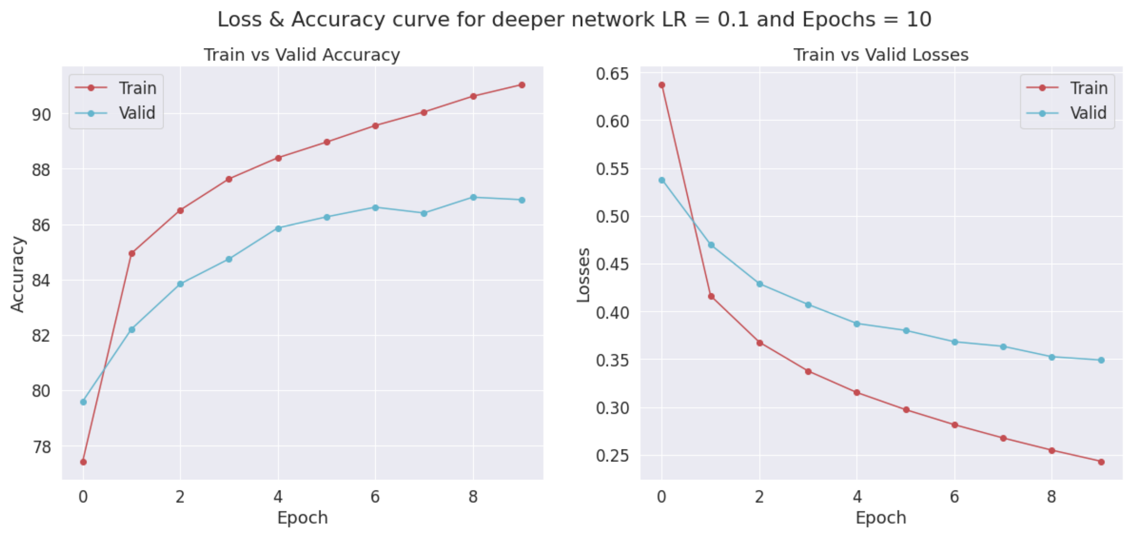


```
--------------------------------------------------------------------------------
----------------------
```

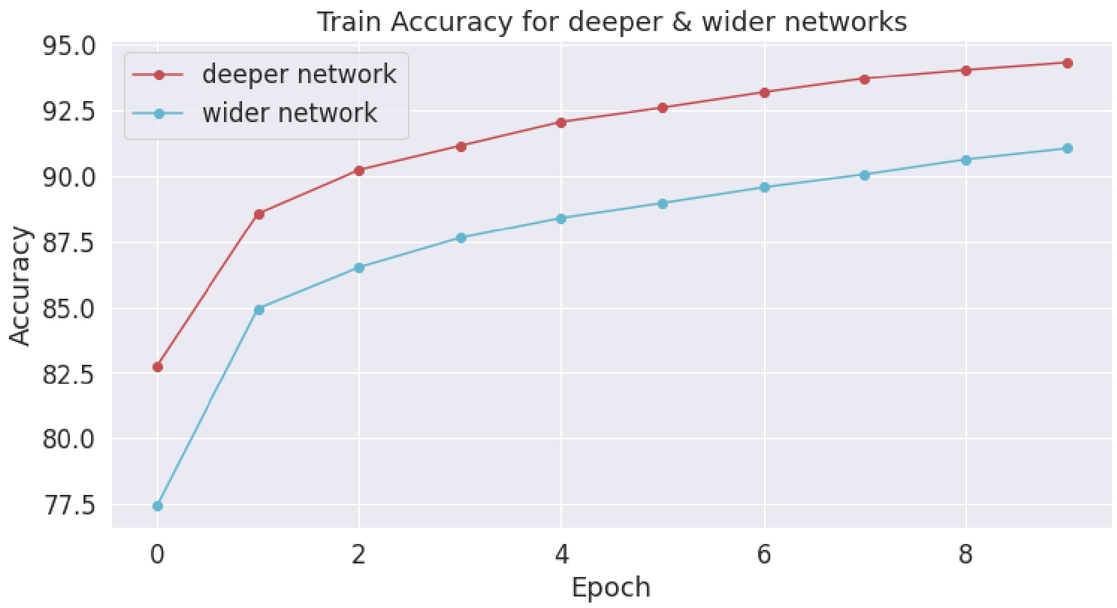
The wider netwrok structure shown below contains 5 layers: 3 Linear layers and 2 Relu layers. The total number of trainable paramters found in this wider network is 1,557,330. The average accuracy found to be 87.49% when the learning rate is 0.1 and epoches is 10.

```
================================================================================
==========
Layer (type:depth-idx)                   Output Shape              Param #
================================================================================
==========
NeuralNetwork                            --                        --
 Flatten: 1-1                           [64, 784]                 --
 Sequential: 1-2                        [64, 10]                  --
      Linear: 2-1                      [64, 1024]                803,840
      ReLU: 2-2                        [64, 1024]                --
      Linear: 2-3                      [64, 728]                 746,200
      ReLU: 2-4                        [64, 728]                 --
      Linear: 2-5                      [64, 10]                  7,290
================================================================================
==========
Total params: 1,557,330
Trainable params: 1,557,330
Non-trainable params: 0
Total mult-adds (M): 99.67
================================================================================
==========
Input size (MB): 0.20
Forward/backward pass size (MB): 0.90
```

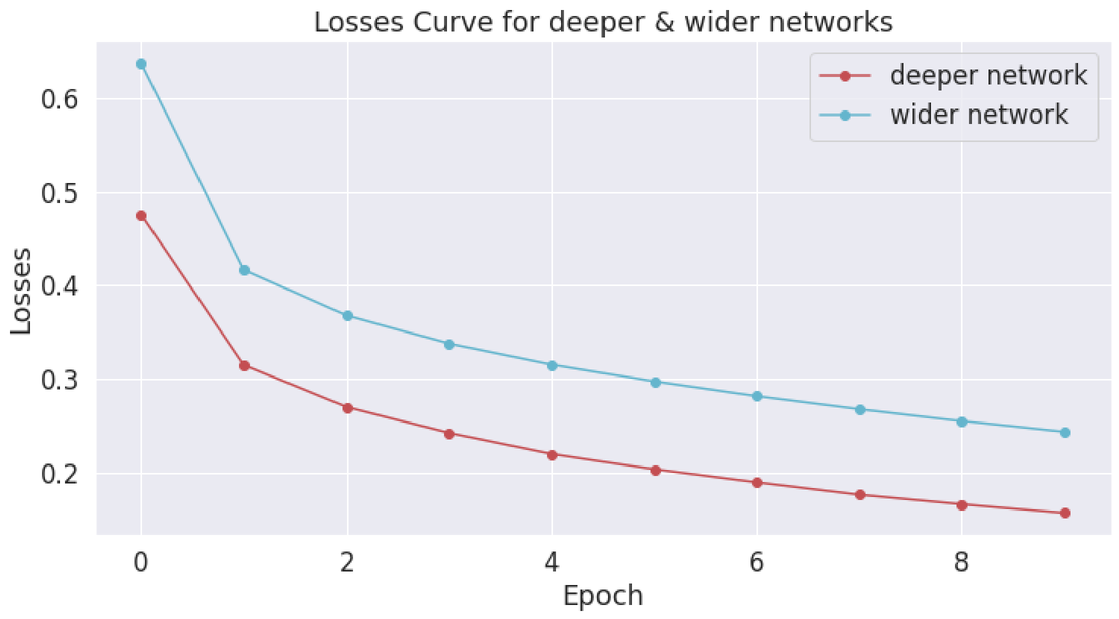
```
Params size (MB): 6.23
Estimated Total Size (MB): 7.33
================================================================================
==========
```



```
--------------------------------------------------------------------------------
----------------------

Overall accuracy comparison :
```



```
--------------------------------------------------------------------------------
----------------------

Overall loss comparison :
```



Structures	Accuracy	Parameters
Base	84.68%	669,706
Deeper	86.79%	1,475,338
Wider	87.49%	1,557,330

```
--------------------------------------------------------------------------------
----------------------
```

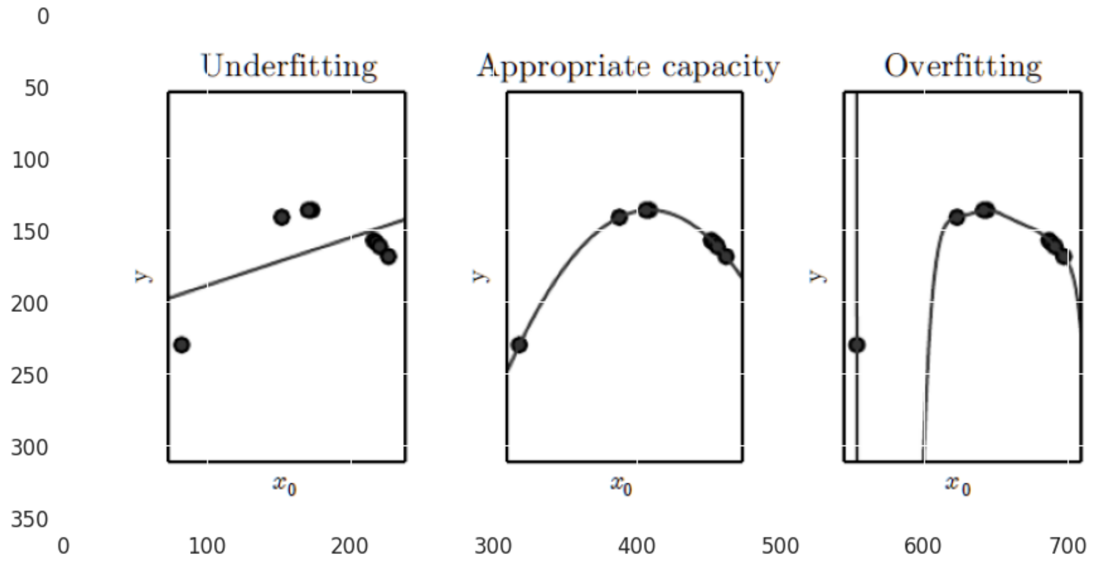
What I observed:

Adding layers to a neutral network adds dimensionality, we can also think of these as eigenvectors in a sense. As we increase dimensionality, we can solve more complex problems which are not linearly separable. We can think of a single hidden layer like a 2 dimensional space and when weights are trained it creates a line to separate the data points. Two hidden layers make a 3 dimensional space and we will create a plane to separate data.

Increasing the number of hidden layers much more than the sufficient number of layers will cause accuracy in the test set to decrease, yes. It will cause our network to overfit to the training set, that is, it will learn the training data, but it won't be able to generalize to new unseen data. The figure shows below provides a pretty good intuition for this concept:

Where in the left picture they try to fit a linear function to the data. This function is not complex enough to correctly represent the data, and it suffers from a bias(underfitting) problem. In the middle picture, the model has the appropriate complexity to accurately represent the data and to generalize, since it has learned the trend that this data follows (the data was synthetically created and has an inverted parabola shape). In the right picture, the model fits to the data, but it overfits to it, it hasn't learnt the trend and thus it is not able to generalize to new data.

```
<matplotlib.image.AxesImage at 0x7f7144f35350>
```



Q1.5 (b). Write done the process of how to calculate the parameters by hand.

The procedure of calculating number of learnable parameters: - Input layer: All the input layer does is read the input image, so there are no parameters could be learnt. - CONV layer: This is where CNN learns, so certainly we'll have weight matrices. To calculate the learnable parameters here, all we have to do is just multiply the by the shape of width m, height n, previous layer's filters d and account for all such filters k in the current layer. Also we cannot forget the bias term for each of the filter. Number of parameters in a CONV layer would be : ((m * n * d)+1)* k), added 1 because of the bias term for each filter. The same expression can be written as follows: ((shape of width of the filter * shape of height of the filter * number of filters in the previous layer+1)*number of filters). Where the term "filter" refer to the number of filters in the current layer. -POOL layer: This has got no learnable parameters because all it does is calculate a specific number, no backprop learning involved! Thus number of parameters = 0. - Batch Normalization layer: Batch Norm is a neural network layer that is now commonly used in many architectures. It often gets added as part of a Linear or Convolutional block and helps to stabilize the network during training. It has two learnable parameters called beta and gamma. To calculate the number of parameters in this layer, we only need to multiply the input number of features by 2 (number of learnable parameters). -Fully Connected Layer (FC): This certainly has learnable parameters, matter of fact, in comparison to the other layers, this category of layers has the highest number of parameters, why? because, every neuron is connected to every other neuron! In a fully-connected layer, all input units have a separate weight to each output unit. For n inputs and m outputs, the number of weights is nxm. Additionally, you have a bias for each output node, so you are at *(n+1)*m parameters.

```
Using cpu device
NeuralNetwork(
  (layer1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (fc1): Linear(in_features=2304, out_features=600, bias=True)
  (drop): Dropout2d(p=0.25, inplace=False)
  (fc2): Linear(in_features=600, out_features=120, bias=True)
  (fc3): Linear(in_features=120, out_features=10, bias=True)
)
```

Take the above Neural Network for example, it has following layers:

- Convolution layer that has kernel size of 3 * 3, padding = 1 (zero_padding) in 1st layer and padding = 0 in second one. Stride of 1 in both layer.
- Batch Normalization layer.
- Acitvation function: ReLU.
- Max Pooling layer with kernel size of 2 * 2 and stride 2.
- Flatten out the output for dense layer(a.k.a. fully connected layer).
- 3 Fully connected layer with different in/out features.
- 1 Dropout layer that has class probability p = 0.25.
- Our input image is changing in a following way:
  - First Convulation layer : input: 28 * 28 * 3, output: 28 * 28 * 32
  - First Max Pooling layer : input: 28 * 28 * 32, output: 14 * 14 * 32
  - Second Conv layer : input : 14 * 14 * 32, output: 12 * 12 * 64
  - Second Max Pooling layer : 12 * 12 * 64, output: 6 * 6 * 64
  - Final fully connected layer has 10 output features for 10 types of clothes.

Number of parameters in First Convulation layer: - m = 3, n = 3, d = 1, k = 32 - ((m * n * d)+1)* k) = ((3x3x1)+1)x32 = 320 ———-

Number of parameters in first Batch Normalization layer: - Number of leanable paramters = 2 - Input feature size = 32 - Number of paramters = 32x2 = 64 ———- Number of parameters in Second Conv layer: - m = 3, n = 3, d = 32, k =64 - ((m * n * d)+1)* k) = ((3x3x32)+1)x64 = 18,496 ———- Number of parameters in first Batch Normalization layer: - Number of leanable paramters = 2 - Input feature size = 64 - Number of paramters = 64x2 = 128 ————- Number of parameters in first Fully Connected Layer: - input layer neurons n = 64x6x6 = 2304 - output layer neurons m = 600 - Number of paramters = (n+1)m = (2304+1)*600 = 1,383,000 ————- umber of parameters in second Fully Connected Layer: - input layer neurons n = 600 - output layer neurons m = 120 - Number of paramters = (n+1)m = (600+1)*120 = 72,120 ————- umber of parameters in third Fully Connected Layer: - input layer neurons n = 120 - output layer neurons m = 10 - Number of paramters = (n+1)m = (120+1)*10 = 1,210 ————- Sum all parameters up: - 320 + 64 + 18,496 + 128 + 1,383,000 + 72,120 + 1,210 = 1,475,338 - Trainable params: 1,475,338

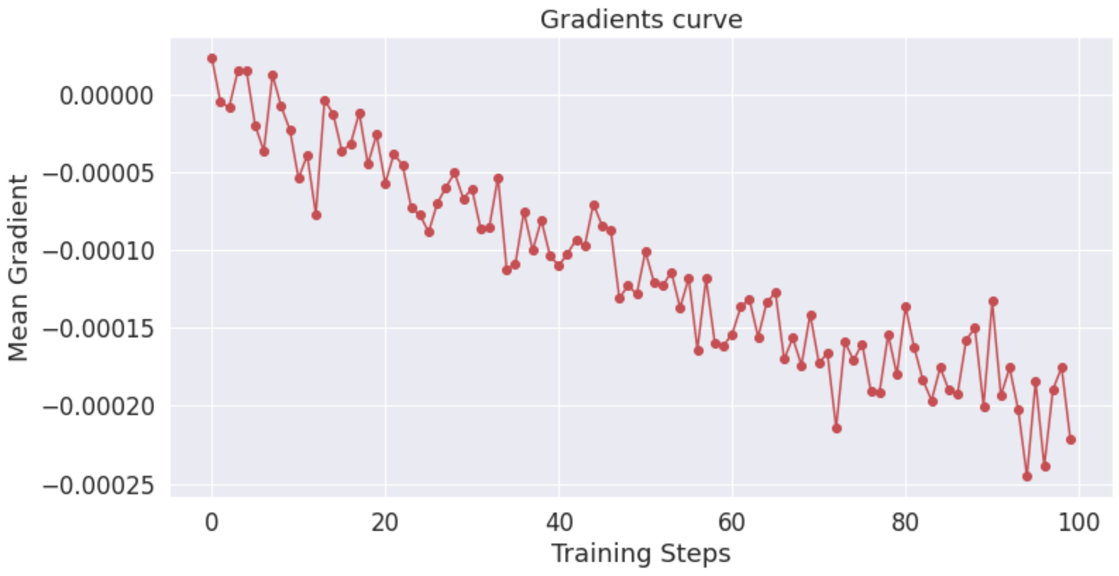
```
--------------------------------------------------------------------------------
----------------------

The total number of trainable parameters calculated by some in built functions
can varify that we got correct anwsers:

================================================================================
==========
Layer (type:depth-idx)                   Output Shape              Param #
================================================================================
==========
NeuralNetwork                             --                        --
 Sequential: 1-1                         [64, 32, 14, 14]          --
      Conv2d: 2-1                        [64, 32, 28, 28]          320
      BatchNorm2d: 2-2                   [64, 32, 28, 28]          64
      ReLU: 2-3                          [64, 32, 28, 28]          --
      MaxPool2d: 2-4                     [64, 32, 14, 14]          --
 Sequential: 1-2                         [64, 64, 6, 6]            --
      Conv2d: 2-5                        [64, 64, 12, 12]          18,496
      BatchNorm2d: 2-6                   [64, 64, 12, 12]          128
      ReLU: 2-7                          [64, 64, 12, 12]          --
      MaxPool2d: 2-8                     [64, 64, 6, 6]            --
 Linear: 1-3                             [64, 600]                 1,383,000
 Dropout: 1-4                            [64, 600]                 --
 Linear: 1-5                             [64, 120]                 72,120
 Linear: 1-6                             [64, 10]                  1,210
================================================================================
==========
Total params: 1,475,338
Trainable params: 1,475,338
Non-trainable params: 0
Total mult-adds (M): 279.73
================================================================================
==========
Input size (MB): 0.20
Forward/backward pass size (MB): 35.50
Params size (MB): 5.90
Estimated Total Size (MB): 41.60
================================================================================
```

==========

Q1.6 What are your observations and conclusions for changing network structure?

A conclusion can be drawn by comparing the performance based on base network, deeper netwrok and wider network. As the table from 1.4 shows, there has no clear performance gain or just a little performance gain from increasing the number of layers(make the network deeper). In fact, increasing the number of hidden layers much more than the sufficient number of layers will cause accuracy in the test set to decrease. It will cause network to overfit to the training set, that is, it will learn the training data, but it won't be able to generalize to new unseen data. Moreover, the time consumption to train the model will increase significantly. Similar to the case of increasing the number of layers, increasing the number of neurons has little impact on perfermance, but time consumed to train the claasifier still increases.

Q1.7 Calculate the mean of the gradients of the loss to all trainable parameters. Plot the gradients curve for the first 100 training steps. What are your observations?

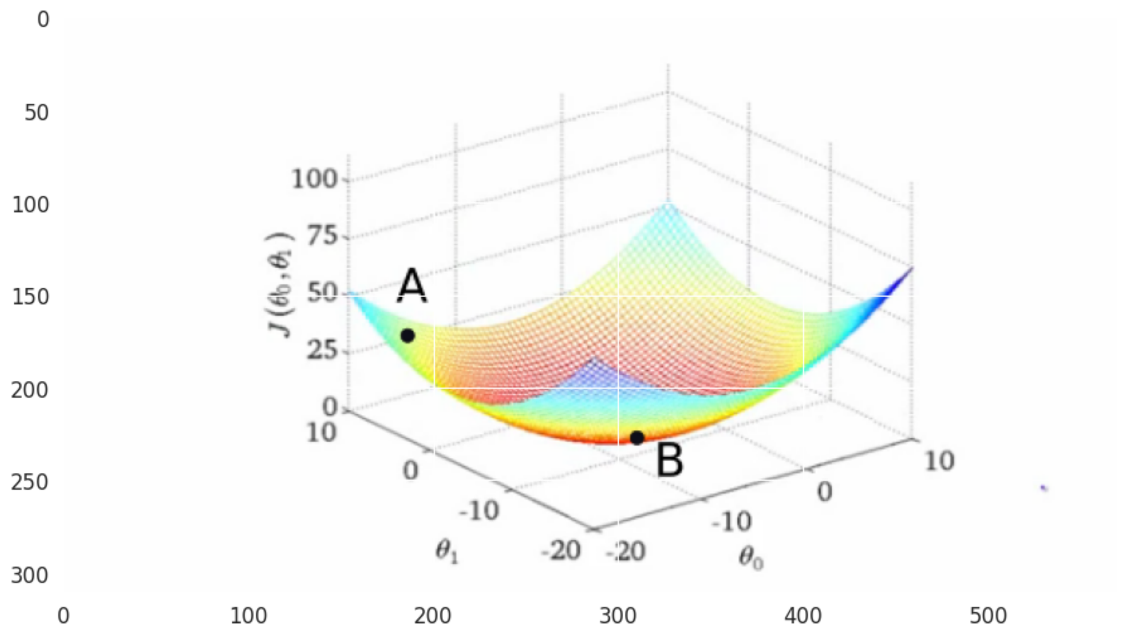


A few concepts can be clarified before we draw the conclusion. What is a loss function?

What is a loss function? - A loss function provides you the difference between the forward pass output and the actual output. Means a loss function value aka cost, or just loss, is the numeric representation of difference of network output with the actual one. This loss is not the actual delta, but rather a real number that gives you a notion of accuracy of the network. Lower the loss, means going towards more accuracy, or higher the loss means the accuracy is degrading.

What's that gradient descent? The gradient descent is an algorithm, which allows machines (computers) to minimize the error (loss) in an iterative way. As the below figure shows:

```
<matplotlib.image.AxesImage at 0x7f55778a8210>
```



When we initialize our weights, we are at point A in the loss landscape. The first thing we do is to check, out of all possible directions in the x-y plane, moving along which direction brings about the steepest decline in the value of the loss function. Now, once we have the direction we want to move in, we must decide the size of the step we must take. The the size of this step is called the learning rate. We must chose it carefully to ensure we can get down to the minima.

If we go too fast, we might overshoot the minima, and keep bouncing along the ridges of the “valley” without ever reaching the minima. Go too slow, and the training might turn out to be too long to be feasible at all. Even if that’s not the case, very slow learning rates make the algorithm more prone to get stuck in a minima, something we’ll cover later in this post.

Once we have our gradient and the learning rate, we take a step, and recompute the gradient at whatever position we end up at, and repeat the process.

In summary: 1. the network gives the output 2. => calculate the loss 3. ==> calculate gradients 4. ===> update weights with the gradients 5.Repeat again until we get exhaust all epochs, get to threshold loss or accuracy

2 Question 2: Proposal for Practical Applications (40%)

Look for a typical computer vision problem, such as: a. removing noise on the image

b. increasing the resolution of the image

c. identifying objects in the image

d. segmenting the area to which the image belongs
e. estimating the depth of an object
f. estimating the motion of two object in different frames
g. others

Discuss possible applications of this problem in life, e.g. image editing systems in your phone, improved quality of the old film, sweeping robot avoiding obstacles, unlocks the face of the mobile phone, identifies the cancer area according to the medical scan image, determines the identity according to the face, identifies the trash can on the road, and the detection system tracks the target object, etc.

In this question, you need to do 1. Clearly define the problem and describe its application scenarios 2. Briefly describe a feasible solution based on image processing and traditional machine learning algorithms. 3. Briefly describe a feasible deep learning-based solution. 4. Compare the advantages and disadvantages of the two options.

Hint1: Submit an individua report for question 2.

Hint2: Well orginaze your report.

Hint3: You can draw flow chart or inculde other figures for better understanding of your solution.

Please restrict your report within 800 words. In this question, you do not need to implement your solution. You only need to write down a proposal. Please submit this report in a seperate pdf.