Assignment3-chang

May 31, 2024

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

This assignment contains 2 questions. The first question probes understanding of deep learning for classification. The second question requires you to write a short description of a Computer Vision method. You wil lneed to submit two separate files, one for each question.

1 Question 1: A simple classifier, 20 marks (70%)

For this exercise, we provide demo code showing how to train a network on a small dataset called Fashion-MNIST. Please run through the code "tutorial-style" to get a sense of what it is doing. Then use the code alongside lecture notes and other resources to understand how to use pytorch libraries to implement, train and use a neural network.

For the Fashion-MNIST dataset the lables from 0-9 correspond to various clothing classes so you might find it convenient to create a python list as follows:

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

You will need to answer various questions about the system, how it operates, the results of experiments with it and make modifications to it yourself. You can change the training scheme and the network structure.

Organize your own text and code cell to show the answer of each questions.

1.1 Question 1.1

Q1.1 (1 point)

Extract 3 images of different types of clothing from the training dataset, print out the size/shape of the training images, and display the three with their corresponding labels.

Please see the images displayed below.

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
```

Add in a code cell to inspect the training data, as per Q1.1. Each element of the training_data structure has a greyscale image (which you can use plt.imshow(img[0,:,:]) to display, just like you did in previous assignments.

```
Size and shape of training images: torch.Size([1, 28, 28])
```

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.

```
Epoch 1
-------------------------------
loss: 2.305552  [    0/60000]
loss: 2.292030  [ 6400/60000]
loss: 2.283248  [12800/60000]
loss: 2.279021  [19200/60000]
loss: 2.248799  [25600/60000]
loss: 2.236767  [32000/60000]
loss: 2.239798  [38400/60000]
loss: 2.211297  [44800/60000]
```

```
loss: 2.212178  [51200/60000]
loss: 2.183519  [57600/60000]
Test Error:
 Accuracy: 50.1%, Avg loss: 2.179195

Epoch 2
-------------------------------
loss: 2.186590  [    0/60000]
loss: 2.176636  [ 6400/60000]
loss: 2.137857  [12800/60000]
loss: 2.151737  [19200/60000]
loss: 2.094538  [25600/60000]
loss: 2.055787  [32000/60000]
loss: 2.079300  [38400/60000]
loss: 2.015861  [44800/60000]
loss: 2.023423  [51200/60000]
loss: 1.960742  [57600/60000]
Test Error:
 Accuracy: 59.5%, Avg loss: 1.959181

Epoch 3
-------------------------------
loss: 1.983575  [    0/60000]
loss: 1.956841  [ 6400/60000]
loss: 1.868469  [12800/60000]
loss: 1.899364  [19200/60000]
loss: 1.784761  [25600/60000]
loss: 1.749277  [32000/60000]
loss: 1.766319  [38400/60000]
loss: 1.676385  [44800/60000]
loss: 1.693626  [51200/60000]
loss: 1.588847  [57600/60000]
Test Error:
 Accuracy: 63.1%, Avg loss: 1.604347

Epoch 4
-------------------------------
loss: 1.664897  [    0/60000]
loss: 1.620766  [ 6400/60000]
loss: 1.490449  [12800/60000]
loss: 1.542536  [19200/60000]
loss: 1.409874  [25600/60000]
loss: 1.418659  [32000/60000]
loss: 1.422417  [38400/60000]
loss: 1.351770  [44800/60000]
loss: 1.379244  [51200/60000]
loss: 1.268798  [57600/60000]
Test Error:
 Accuracy: 64.3%, Avg loss: 1.299689

Epoch 5
-------------------------------
loss: 1.379926  [    0/60000]
loss: 1.349978  [ 6400/60000]
loss: 1.197700  [12800/60000]
loss: 1.282175  [19200/60000]
loss: 1.154815  [25600/60000]
loss: 1.190748  [32000/60000]
loss: 1.201745  [38400/60000]
loss: 1.142816  [44800/60000]
loss: 1.174767  [51200/60000]
loss: 1.083401  [57600/60000]
Test Error:
 Accuracy: 65.4%, Avg loss: 1.110647

Done!
```

1.2 Question 1.2

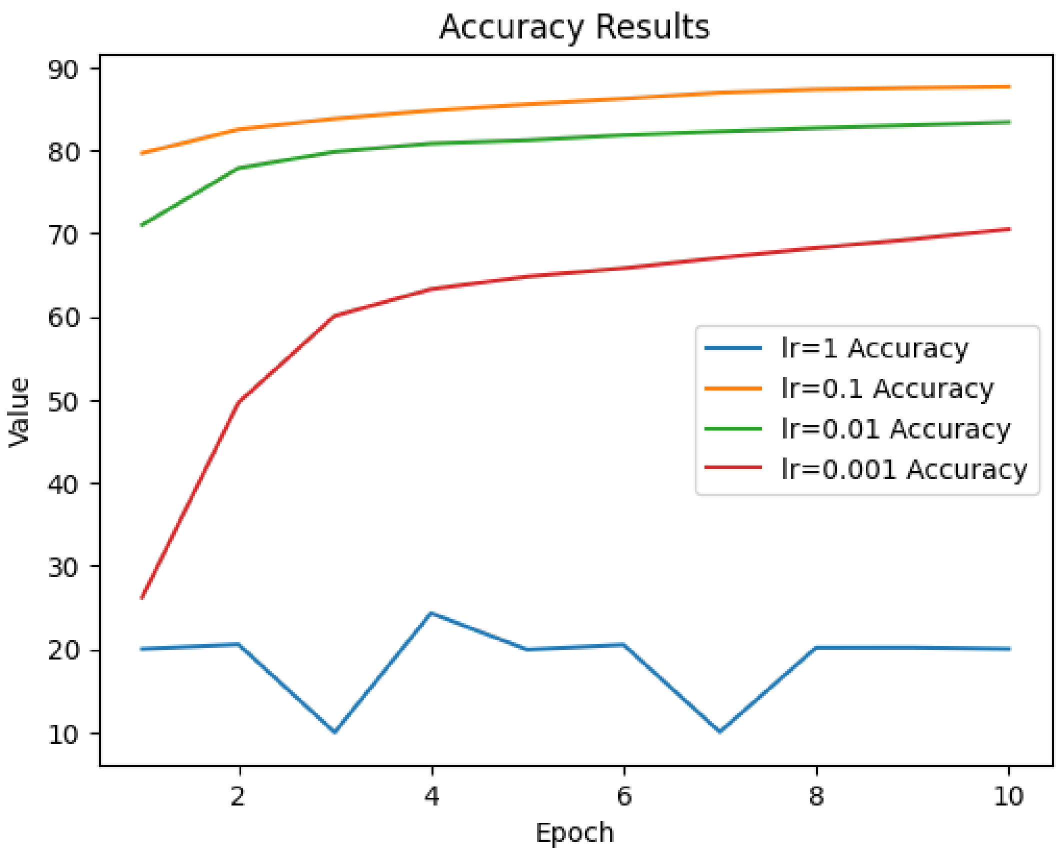
Q1.2 (2 point) Run the training code for 10 epochs, for different values of the learning rate. Fill in the table below and plot the loss curves for each experiment:

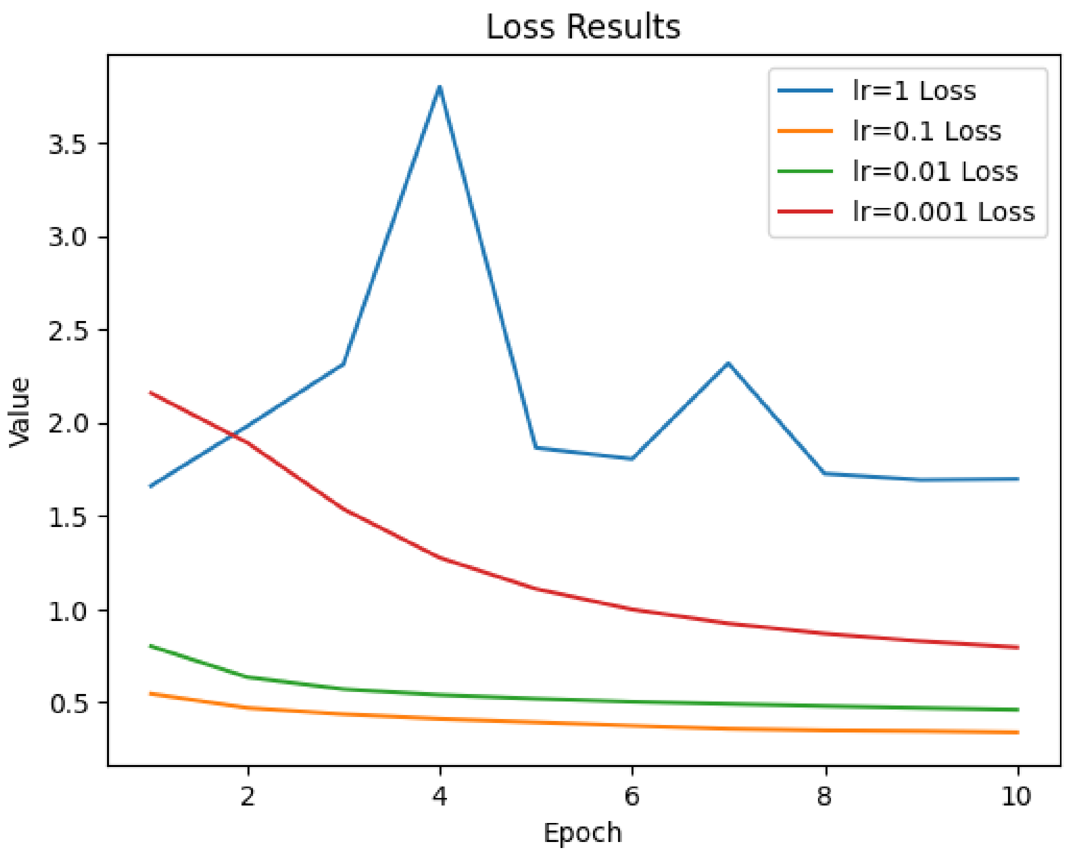
Lr	Accuracy
1	20.0%
0.1	87.7%
0.01	83.4%
0.001	70.5%

(Please see extended results bellow)

Through the following I'm going to be duplicating the code from the example and modifying it to get the results for this question.

```
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)
  )
)

Learning Rate 1
Epoch 1: accuracy - 20.0%
Epoch 2: accuracy - 20.6%
Epoch 3: accuracy - 10.0%
Epoch 4: accuracy - 24.3%
Epoch 5: accuracy - 19.9%
Epoch 6: accuracy - 20.5%
Epoch 7: accuracy - 10.1%
Epoch 8: accuracy - 20.2%
Epoch 9: accuracy - 20.2%
Epoch 10: accuracy - 20.0%
Learning Rate 0.1
Epoch 1: accuracy - 79.7%
Epoch 2: accuracy - 82.5%
Epoch 3: accuracy - 83.8%
Epoch 4: accuracy - 84.8%
Epoch 5: accuracy - 85.5%
Epoch 6: accuracy - 86.2%
Epoch 7: accuracy - 86.9%
Epoch 8: accuracy - 87.3%
Epoch 9: accuracy - 87.5%
Epoch 10: accuracy - 87.7%
Learning Rate 0.01
Epoch 1: accuracy - 71.0%
Epoch 2: accuracy - 77.8%
Epoch 3: accuracy - 79.8%
Epoch 4: accuracy - 80.8%
Epoch 5: accuracy - 81.2%
Epoch 6: accuracy - 81.8%
Epoch 7: accuracy - 82.3%
Epoch 8: accuracy - 82.7%
Epoch 9: accuracy - 83.0%
Epoch 10: accuracy - 83.4%
Learning Rate 0.001
Epoch 1: accuracy - 26.2%
Epoch 2: accuracy - 49.6%
Epoch 3: accuracy - 60.1%
Epoch 4: accuracy - 63.3%
Epoch 5: accuracy - 64.8%
Epoch 6: accuracy - 65.8%
Epoch 7: accuracy - 67.1%
Epoch 8: accuracy - 68.2%
Epoch 9: accuracy - 69.3%
Epoch 10: accuracy - 70.5%
Done!
```


Accuracy Results
Value
Epoch
lr=1 Accuracy
lr=0.1 Accuracy
lr=0.01 Accuracy
lr=0.001 Accuracy



Looking at the results of this, we are able to see that it is most likely that Learning rate equal to 0.1 would be the best for this model. As that is going to balance both speed and the accuracy.

1.3 Question 1.3

Q1.3 (3 point) Report the number of epochs when the network converges (or nukber of epochs for the best accuracy, if it fails to converge). Fill in the table below and plot the loss curve for each experiment:

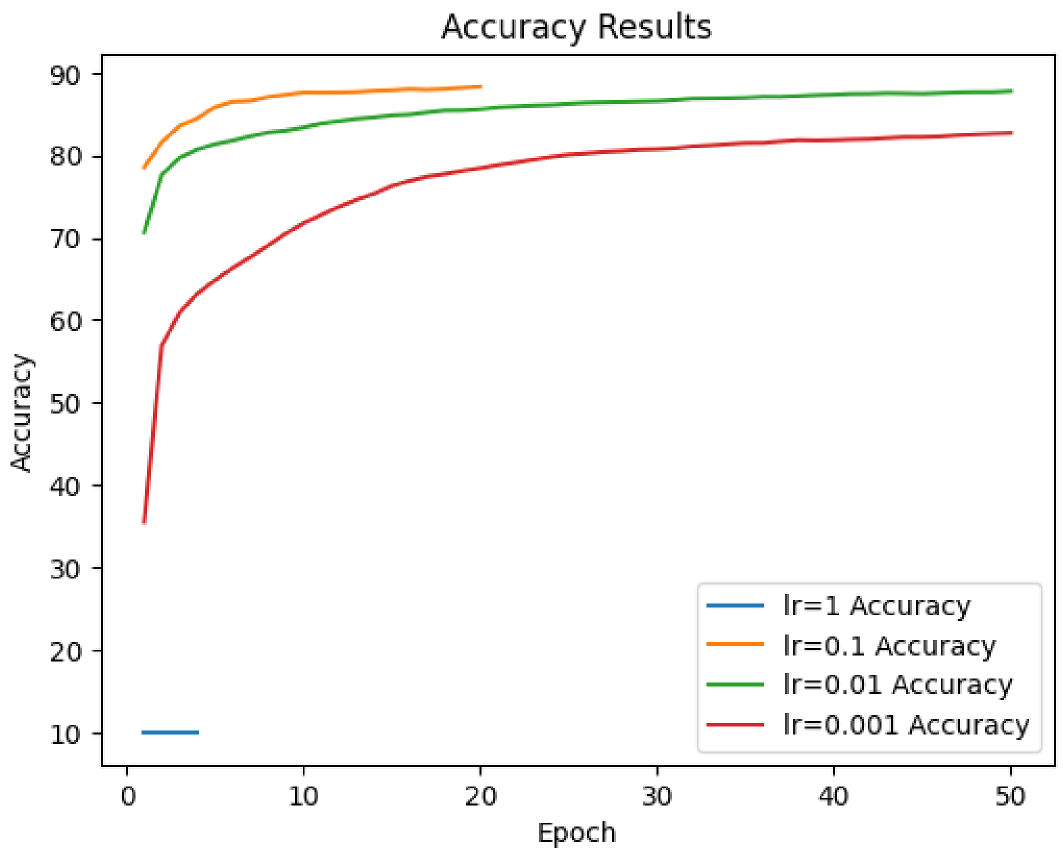
Lr	Accuracy	Epoch
1	10.0%	1
0.1	88.0%	17
0.01	87.8%	50+
0.001	82.7%	50+

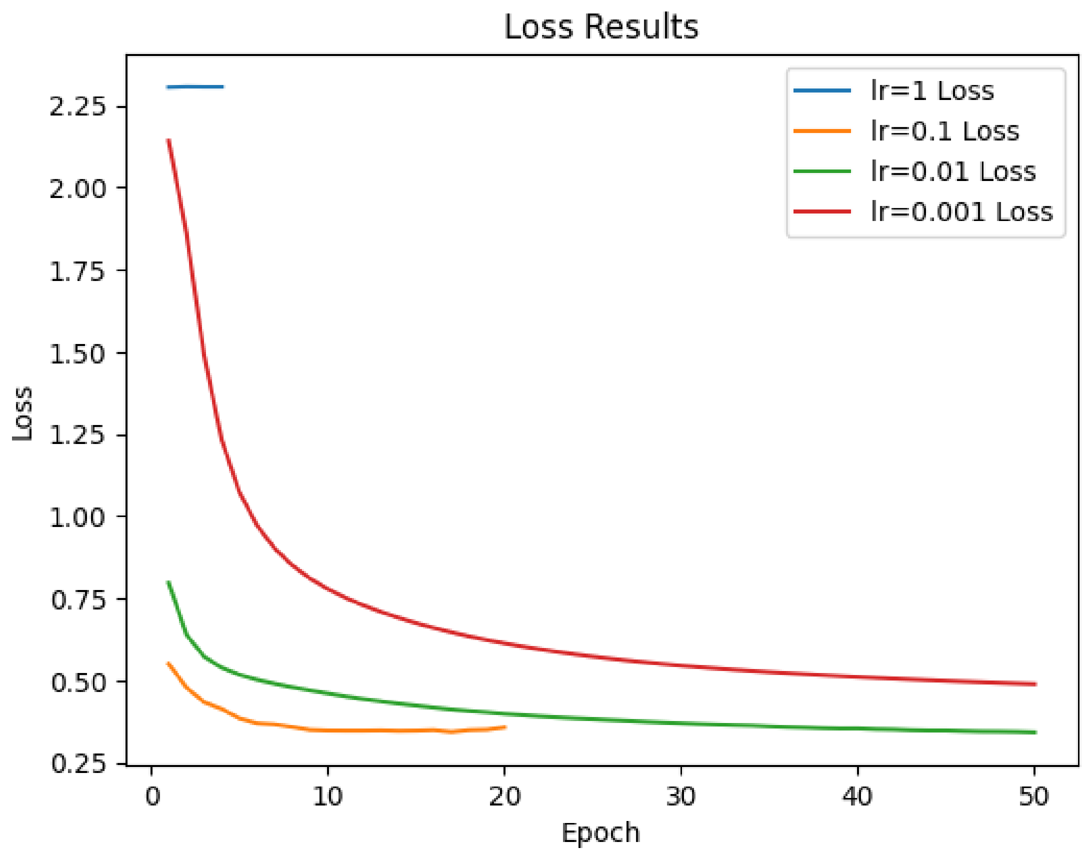
(Please see extended results bellow)

Through the following I'm going to be duplicating the code from the example and modifying it to get the results for this question.

```
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)
  )
)
```

```
Learning Rate 1
Epoch 1: accuracy - 10.0%, loss - 2.304739
Epoch 2: accuracy - 10.0%, loss - 2.306479
Epoch 3: accuracy - 10.0%, loss - 2.305841
Epoch 4: accuracy - 10.0%, loss - 2.305844
Learning Rate 0.1
Epoch 1: accuracy - 78.5%, loss - 0.550608
Epoch 2: accuracy - 81.6%, loss - 0.478730
Epoch 3: accuracy - 83.6%, loss - 0.434355
Epoch 4: accuracy - 84.5%, loss - 0.413341
Epoch 5: accuracy - 85.8%, loss - 0.384923
Epoch 6: accuracy - 86.5%, loss - 0.370060
Epoch 7: accuracy - 86.6%, loss - 0.366587
Epoch 8: accuracy - 87.1%, loss - 0.358751
Epoch 9: accuracy - 87.3%, loss - 0.349779
Epoch 10: accuracy - 87.6%, loss - 0.347901
Epoch 11: accuracy - 87.6%, loss - 0.347509
Epoch 12: accuracy - 87.6%, loss - 0.347526
Epoch 13: accuracy - 87.7%, loss - 0.348295
Epoch 14: accuracy - 87.8%, loss - 0.346894
Epoch 15: accuracy - 87.9%, loss - 0.347529
Epoch 16: accuracy - 88.0%, loss - 0.349170
Epoch 17: accuracy - 88.0%, loss - 0.343744
Epoch 18: accuracy - 88.0%, loss - 0.349200
Epoch 19: accuracy - 88.2%, loss - 0.350568
Epoch 20: accuracy - 88.3%, loss - 0.357596
Learning Rate 0.01
Epoch 1: accuracy - 70.6%, loss - 0.796759
Epoch 2: accuracy - 77.6%, loss - 0.639434
Epoch 3: accuracy - 79.7%, loss - 0.572653
Epoch 4: accuracy - 80.7%, loss - 0.538961
Epoch 5: accuracy - 81.3%, loss - 0.517621
Epoch 6: accuracy - 81.8%, loss - 0.502352
Epoch 7: accuracy - 82.3%, loss - 0.490395
Epoch 8: accuracy - 82.8%, loss - 0.479692
Epoch 9: accuracy - 83.0%, loss - 0.470137
Epoch 10: accuracy - 83.4%, loss - 0.460765
Epoch 11: accuracy - 83.9%, loss - 0.451925
Epoch 12: accuracy - 84.1%, loss - 0.443726
Epoch 13: accuracy - 84.4%, loss - 0.436756
Epoch 14: accuracy - 84.6%, loss - 0.429857
Epoch 15: accuracy - 84.8%, loss - 0.423754
Epoch 16: accuracy - 84.9%, loss - 0.417716
Epoch 17: accuracy - 85.2%, loss - 0.412010
Epoch 18: accuracy - 85.4%, loss - 0.407365
Epoch 19: accuracy - 85.5%, loss - 0.403407
Epoch 20: accuracy - 85.6%, loss - 0.398725
Epoch 21: accuracy - 85.8%, loss - 0.395039
Epoch 22: accuracy - 85.9%, loss - 0.391196
Epoch 23: accuracy - 86.0%, loss - 0.388043
Epoch 24: accuracy - 86.1%, loss - 0.385149
Epoch 25: accuracy - 86.2%, loss - 0.382690
Epoch 26: accuracy - 86.4%, loss - 0.379952
Epoch 27: accuracy - 86.4%, loss - 0.377572
Epoch 28: accuracy - 86.5%, loss - 0.374683
Epoch 29: accuracy - 86.5%, loss - 0.372278
Epoch 30: accuracy - 86.6%, loss - 0.370049
Epoch 31: accuracy - 86.7%, loss - 0.367766
Epoch 32: accuracy - 86.9%, loss - 0.365945
Epoch 33: accuracy - 86.9%, loss - 0.364365
Epoch 34: accuracy - 86.9%, loss - 0.363063
Epoch 35: accuracy - 87.0%, loss - 0.360583
Epoch 36: accuracy - 87.1%, loss - 0.358498
Epoch 37: accuracy - 87.1%, loss - 0.357078
Epoch 38: accuracy - 87.2%, loss - 0.355210
Epoch 39: accuracy - 87.3%, loss - 0.354079
Epoch 40: accuracy - 87.4%, loss - 0.354190
Epoch 41: accuracy - 87.5%, loss - 0.352025
Epoch 42: accuracy - 87.5%, loss - 0.351102
Epoch 43: accuracy - 87.5%, loss - 0.349038
Epoch 44: accuracy - 87.5%, loss - 0.348000
Epoch 45: accuracy - 87.5%, loss - 0.347900
Epoch 46: accuracy - 87.5%, loss - 0.346335
Epoch 47: accuracy - 87.6%, loss - 0.345106
Epoch 48: accuracy - 87.7%, loss - 0.344799
Epoch 49: accuracy - 87.7%, loss - 0.344042
Epoch 50: accuracy - 87.8%, loss - 0.342506
Learning Rate 0.001
Epoch 1: accuracy - 35.5%, loss - 2.141357
Epoch 2: accuracy - 56.9%, loss - 1.860004
Epoch 3: accuracy - 60.9%, loss - 1.491046
Epoch 4: accuracy - 63.2%, loss - 1.230688
Epoch 5: accuracy - 64.8%, loss - 1.072695
Epoch 6: accuracy - 66.3%, loss - 0.971305
Epoch 7: accuracy - 67.6%, loss - 0.901307
Epoch 8: accuracy - 69.0%, loss - 0.849903
Epoch 9: accuracy - 70.5%, loss - 0.810244
Epoch 10: accuracy - 71.8%, loss - 0.778346
Epoch 11: accuracy - 72.8%, loss - 0.751792
Epoch 12: accuracy - 73.7%, loss - 0.728989
Epoch 13: accuracy - 74.6%, loss - 0.708931
Epoch 14: accuracy - 75.3%, loss - 0.691006
Epoch 15: accuracy - 76.3%, loss - 0.674822
Epoch 16: accuracy - 76.9%, loss - 0.660100
Epoch 17: accuracy - 77.4%, loss - 0.646696
Epoch 18: accuracy - 77.7%, loss - 0.634461
Epoch 19: accuracy - 78.1%, loss - 0.623276
Epoch 20: accuracy - 78.4%, loss - 0.613033
Epoch 21: accuracy - 78.8%, loss - 0.603635
Epoch 22: accuracy - 79.1%, loss - 0.594990
Epoch 23: accuracy - 79.5%, loss - 0.587026
Epoch 24: accuracy - 79.8%, loss - 0.579666
Epoch 25: accuracy - 80.1%, loss - 0.572858
Epoch 26: accuracy - 80.2%, loss - 0.566547
Epoch 27: accuracy - 80.4%, loss - 0.560687
Epoch 28: accuracy - 80.5%, loss - 0.555238
Epoch 29: accuracy - 80.7%, loss - 0.550154
Epoch 30: accuracy - 80.7%, loss - 0.545408
Epoch 31: accuracy - 80.9%, loss - 0.540960
Epoch 32: accuracy - 81.1%, loss - 0.536788
Epoch 33: accuracy - 81.2%, loss - 0.532864
Epoch 34: accuracy - 81.3%, loss - 0.529167
Epoch 35: accuracy - 81.5%, loss - 0.525677
Epoch 36: accuracy - 81.5%, loss - 0.522377
Epoch 37: accuracy - 81.7%, loss - 0.519249
Epoch 38: accuracy - 81.8%, loss - 0.516279
Epoch 39: accuracy - 81.8%, loss - 0.513460
Epoch 40: accuracy - 81.8%, loss - 0.510771
Epoch 41: accuracy - 81.9%, loss - 0.508203
Epoch 42: accuracy - 82.0%, loss - 0.505748
Epoch 43: accuracy - 82.1%, loss - 0.503396
Epoch 44: accuracy - 82.2%, loss - 0.501141
Epoch 45: accuracy - 82.2%, loss - 0.498982
Epoch 46: accuracy - 82.3%, loss - 0.496912
Epoch 47: accuracy - 82.4%, loss - 0.494920
Epoch 48: accuracy - 82.5%, loss - 0.493000
Epoch 49: accuracy - 82.6%, loss - 0.491154
Epoch 50: accuracy - 82.7%, loss - 0.489368
Done!
```


Accuracy Results
Accuracy
Epoch
lr=1 Accuracy
lr=0.1 Accuracy
lr=0.01 Accuracy
lr=0.001 Accuracy



```
Learning Rate 1 converged at epoch 1
Learning Rate 0.1 converged at epoch 17
Learning Rate 0.01 converged at epoch 50
Learning Rate 0.001 converged at epoch 50
```

1.4 Question 1.4

Q1.4 (2 points) Compare the results in table 1 and table 2, what is your observation and your understanding of learning rate?

First comparing the results in the table for each of the learning rates.

Learning rate 1: This high learning rate results in poor performace and convergence. Table 1 after 10 epochs the accuracy is 20% while table 2 after 1 epochs the accuracy is 10%. With a learning rate of 1 the network is struggling to learn effiecently which could likely be due to overshooting the optimal soltion.

Learning rate 0.1: With this learning rate the network achieves a high accuracy relatively quickly. Table 1 shows 87.7% after 10 epochs and table 2 shows 88.0% after 17 epochs. This learning rate converges fast with a high accuracy and is the optimal learning rate for this network.

Learning rate 0.01: With this learning rate the network converges slower. Table 1 shows 87.8% accuracy after 10 epochs and table 2 shows 87.8% after 50 epochs. Showing that the network is taking longer to converge. This learning rate is requiring more epochs to reach simlar accuracy to the network with learning rate 0.1

Learning rate 0.001: This learning rate is very slow to converge. After running 100 epochs the network was unable to converge. Table 1 after 10 epochs produced and accuracy of 70.5% and table 2 shows an accuracy of 82.7% after 50 epochs. This is showing that the network is able to produce a good result but needs to time to train to reach a similar place to the other networks.

The higher learning rate leads to the rapid changes and causes fluctuations between the epochs producing a result that is unfavourable and not the optimal solution that is most likely that it has overshow the optimal solution. The moderate learning rate resulted in the best for this model, as it provides relatively fast learning and converges to result in a good solution. The low learning rate resulting in the slowest learning, but after a while it would results is a relativly accuracte solution but will need more time to converge.

1.5 Question 1.5

Q1.5 (5 points) Build a wider network by modifying the code that constructs the network so that the hidden layer(s) contain more perceptrons, and record the accuracy along with the number of trainable parameters in your model. Now modify the oroginal network to be deeper instead of wider (i.e. by adding more hidden layers). Record your accuracy and network size findings. Plot the loss curve for each experiment. Write down your conclusions about changing the network structure?

Structures	Accuracy	Parameters
Base	87.1%	669706
Deeper	86.7%	1195018
Wider	87.8%	1863690

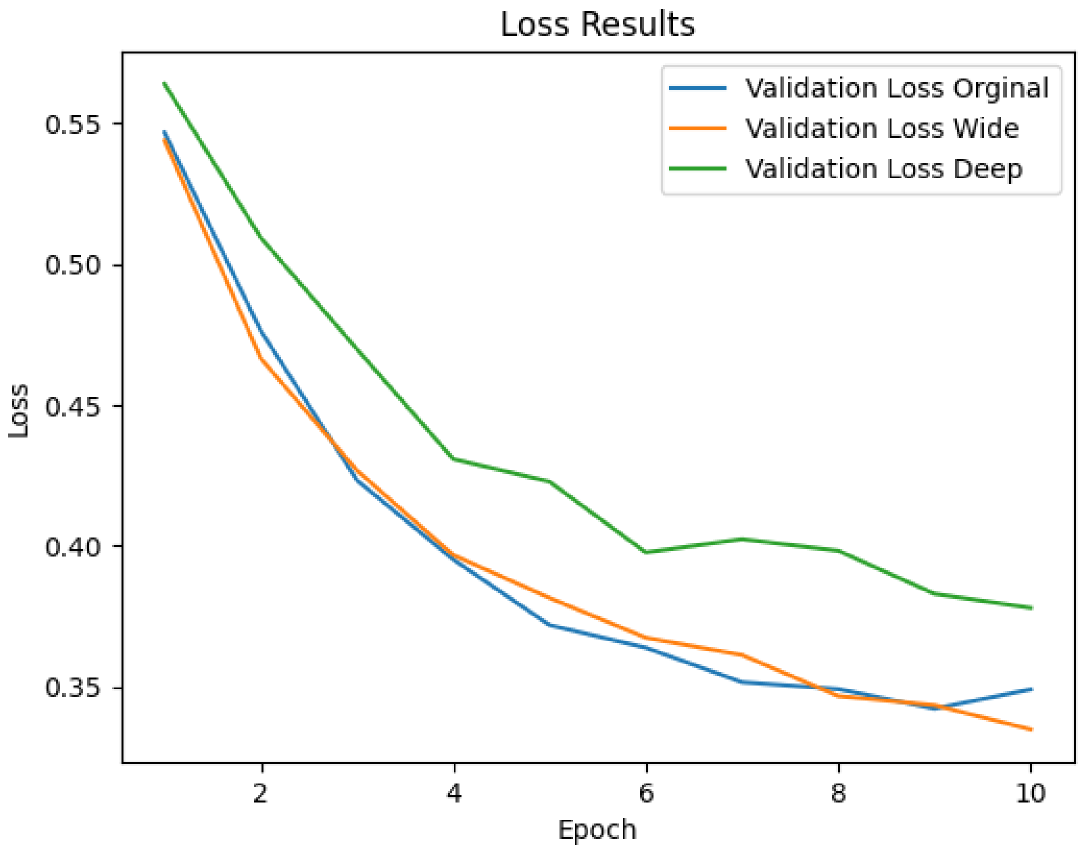
(Please see extended results bellow)

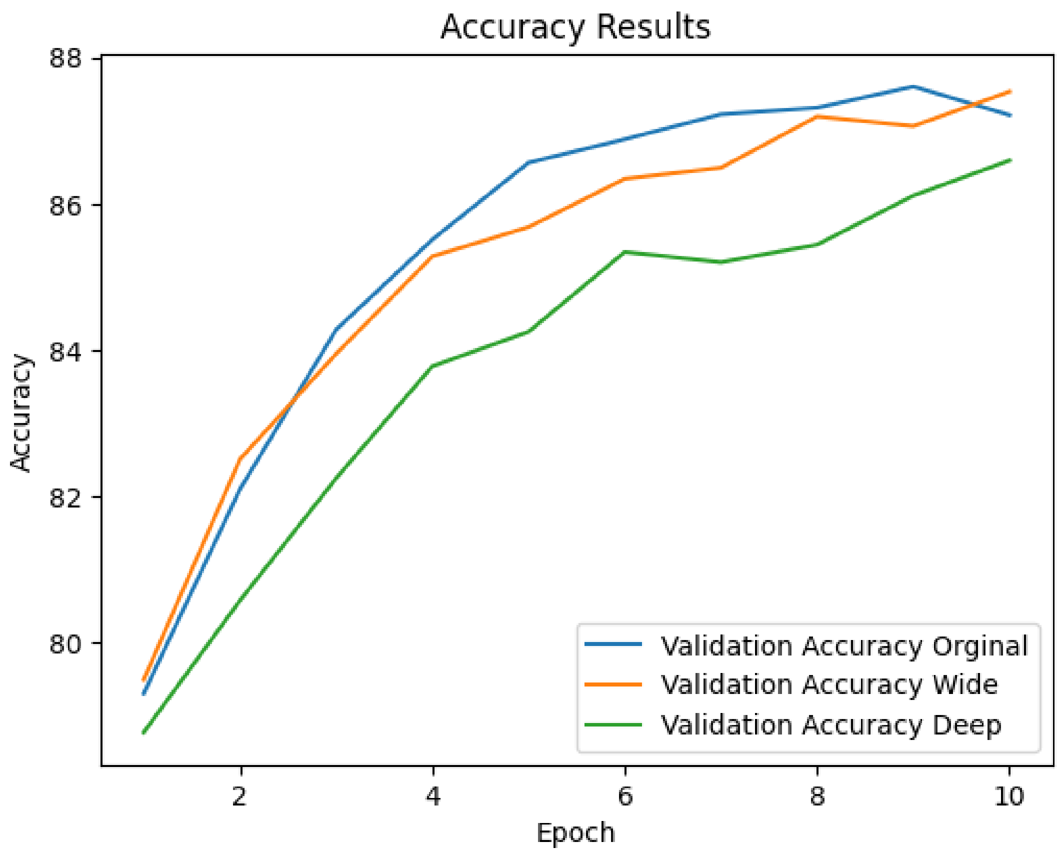
More of the models

```
Using cpu device

deepNeuralNetwork(
  (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=512, bias=True)
    (5): ReLU()
    (6): Linear(in_features=512, out_features=512, bias=True)
    (7): ReLU()
    (8): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

```
wideNeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=1024, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1024, out_features=1024, bias=True)
    (3): ReLU()
    (4): Linear(in_features=1024, out_features=10, bias=True)
  )
)

deepNeuralNetwork(
  (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=512, bias=True)
    (5): ReLU()
    (6): Linear(in_features=512, out_features=512, bias=True)
    (7): ReLU()
    (8): Linear(in_features=512, out_features=10, bias=True)
  )
)

Original Model Parameter: 669706
Wide Model Parameter: 1863690
Deep Model Parameter: 1195018

Results for the original Network
Epoch 1: accuracy - 79.3%
Epoch 2: accuracy - 82.1%
Epoch 3: accuracy - 84.3%
Epoch 4: accuracy - 85.5%
Epoch 5: accuracy - 86.6%
Epoch 6: accuracy - 86.9%
Epoch 7: accuracy - 87.2%
Epoch 8: accuracy - 87.3%
Epoch 9: accuracy - 87.6%
Epoch 10: accuracy - 87.2%

Results for the wide Network
Epoch 1: accuracy - 79.5%
Epoch 2: accuracy - 82.5%
Epoch 3: accuracy - 84.0%
Epoch 4: accuracy - 85.3%
Epoch 5: accuracy - 85.7%
Epoch 6: accuracy - 86.3%
Epoch 7: accuracy - 86.5%
Epoch 8: accuracy - 87.2%
Epoch 9: accuracy - 87.1%
Epoch 10: accuracy - 87.5%
```

```
Results for the deep Network
Epoch 1: accuracy - 78.8%
Epoch 2: accuracy - 80.6%
Epoch 3: accuracy - 82.2%
Epoch 4: accuracy - 83.8%
Epoch 5: accuracy - 84.2%
Epoch 6: accuracy - 85.3%
Epoch 7: accuracy - 85.2%
Epoch 8: accuracy - 85.4%
Epoch 9: accuracy - 86.1%
Epoch 10: accuracy - 86.6%
```





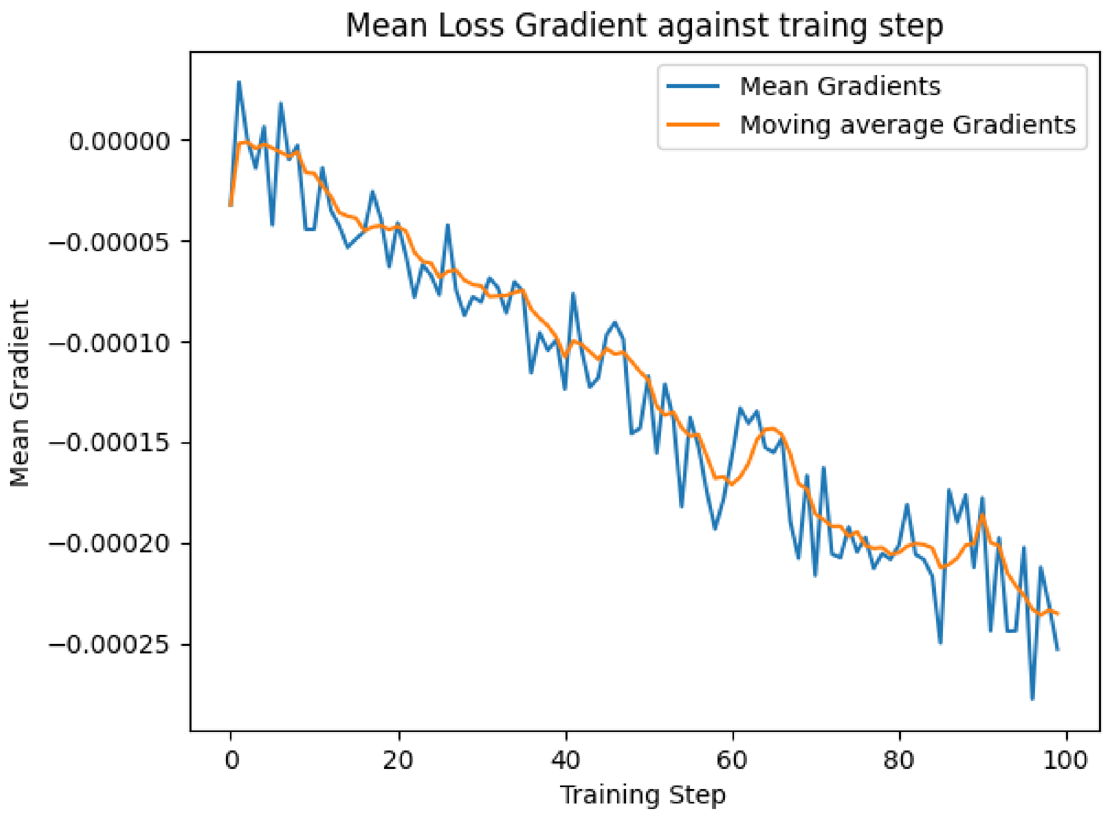
Looking at the overall results from the 3 different models, the wider model looks to be achieving the best results out of the three different models with the best accuracy and the loss decreacing the most. The worst one overall was the deep one producing the worst result.

1.6 Question 1.6

Q1.6 (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 decrease.

From the last question all the needs to be changes is the training function. So the orignal model will be called using the same functions with a modifyed trainging function.

```
Done!
```



The observations of this graph is that it is trending down, which is what we want to see. This is telling us that the model is moving toward more optimal values which means that our model is becomming more accurate. As we also have a faiarly consitant gratient we are able to duduce that we have picked a good learning rate for this model.

1.7 Question 1.7

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

Q1.7 (5 points) Modify the network structure and training/test to use a small convolutional neural network instead of an MLP. Discuss your findings with rehgard to convergence, accuracy and number of parameters, relative to MLPs.

Hint: Look at the structure of the CNN in the Workshop 3 examples.

```
CNN(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (fc1): Linear(in_features=3136, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=10, bias=True)
)
Total number of parameters 421834
Epoch 1
-------------------------------
Test Error:
 Accuracy: 82.2%, Avg loss: 0.428238

Epoch 2
-------------------------------
Test Error:
 Accuracy: 83.2%, Avg loss: 0.426585

Epoch 3
-------------------------------
Test Error:
 Accuracy: 85.9%, Avg loss: 0.368782

Epoch 4
-------------------------------
Test Error:
 Accuracy: 90.0%, Avg loss: 0.287261

Epoch 5
-------------------------------
Test Error:
 Accuracy: 90.6%, Avg loss: 0.275673

Epoch 6
-------------------------------
Test Error:
 Accuracy: 89.4%, Avg loss: 0.301058

Epoch 7
-------------------------------
Test Error:
 Accuracy: 90.8%, Avg loss: 0.279688

Epoch 8
-------------------------------
Test Error:
 Accuracy: 89.8%, Avg loss: 0.316344
```

```
Epoch 9
-------------------------------
Test Error:
 Accuracy: 89.5%, Avg loss: 0.340267

Epoch 10
-------------------------------
Test Error:
 Accuracy: 90.5%, Avg loss: 0.325097

Epoch 11
-------------------------------
Test Error:
 Accuracy: 90.5%, Avg loss: 0.330025

Epoch 12
-------------------------------
Test Error:
 Accuracy: 91.0%, Avg loss: 0.354739

Epoch 13
-------------------------------
Test Error:
 Accuracy: 90.9%, Avg loss: 0.371169

Done!
```

Looking at this convolutional neural network it can be wee that it is performing better then the MLP in all the following areas, convergence, accuracy, and the numher of parameters. This model above has a total of 421834 which is over 200000 parameters less than the original MLP network. In regards to the convergence it is also doing better then the MLP by converging after 12 epochs which is better then the MLP by 2 for the model that acheived the best accuracy. It also beat the best MLP model in accuracy also by achiving 91.0% which is better then the best MLP model by 3%. Overall this model is working better then the MLP model, and with some more optimizations it could be achiving a better accuracy. This optimisations could include adding more layers to the model, Decreasing the learing rate overtime, as the model is converging, and also some data augmentation to the model more generic. These optimisation could more the more accurate and produce a better result.