Machine Learning & Discovery *ECS 271, Fall 2022*

HW3

- You will learn how to train a deep network using PyTorch tool. Please read the following tutorial. You may skip the data parallelism section.

https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

- CIFAR10 is a dataset of $60,000$ color images of size 32×32 from 10 categories. Please download the PyTorch tutorial code for CIFAR10 to start:

https://pytorch.org/tutorials/_downloads/cifar10_tutorial.py

- When you run the tutorial code, it will download CIFAR10 dataset for you. Please follow the instructions in the following link to install PyTorch: *https://pytorch.org/*

- To learn more, you can also find a tutorial for MNIST here:

https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html#sphx-glr-beginner-blitz-neural-networks-tutorial-py

and the sample model for MNIST here:

https://github.com/pytorch/examples/blob/master/mnist/main.py

and the sample code for Imagenet here:

https://github.com/pytorch/examples/blob/master/imagenet/main.py

- For all the following sections, train the model for 50 epochs and plot the curve for loss, training accuracy, and test accuracy evaluated every epoch.

1. Run the tutorial code out of the box and make sure you get reasonable results. You will report these results in Section 4, so no report needed here.

   Note: All graphs can be found in the submitted .ipynb file

2. Change the code to have only a single fully connected layer. The model will have a single layer that connects the input to the output. What is the number of parameters? In PyTorch, "nn.Linear" can be used for fully connected layer.

   The number of parameters can be calculated using:

   $$n_{parameters} = \underbrace{(3 \times 32 \times 32)}_{\text{input}} \times \underbrace{10}_{\text{output}} + \underbrace{10}_{\text{bias}} = 30,730 \quad (1)$$

3. Change the code to have multiple fully connected layers. Try having a layer from input to 110 neurons and then a layer to 74 neurons, and finally a layer to 10 neurons, one for each category. What happens if you do not use ReLU? Describe why.

In the absence of the ReLU activation function, the neural network may be seen as a simple sequence of repeated multiplications, eventually learning just the linear relationships present in the data. The ReLU activation function is a piecewise linear function, i.e., it is linear for values greater than zero however it is nonlinear as the output of negative values is zero. This non-linearity introduced by the RelU activation function urges the network to learn hidden and more complex patterns in the data. The figures below are the test accuracy v/s epochs graphs for the case without ReLU and with ReLU respectively.[1] [2]

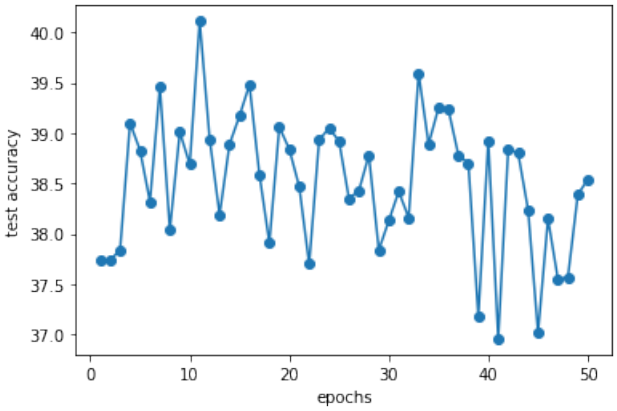

Figure 1: Without ReLU

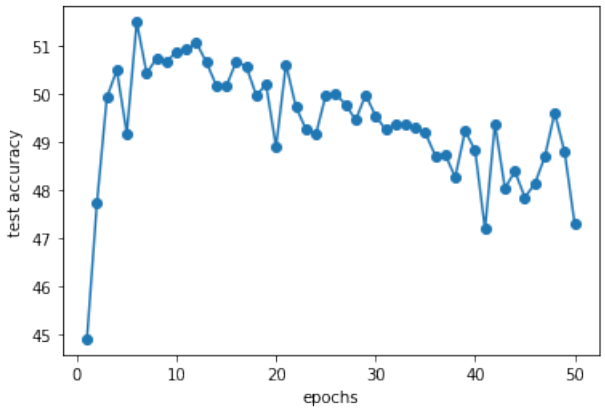

Figure 2: With ReLU

4. Change the code by adding two convolutional layers along with maxpooling layers before the fully connected layers. This will be similar to the example in the tutorial. Use this model for the following sections.

5. Try multiple batch sizes to see the effect and describe the findings. Please use batch size of 1, 4, and 1000. If 1000 does not fit into the memory of your machine, please feel free to reduce it to the largest possible number.

On training the network with the various batch sizes, we see that the batch size of 4 performed the best based on these test accuracy v/s epochs graphs for batch size 1, 4, and 1000 respectively.

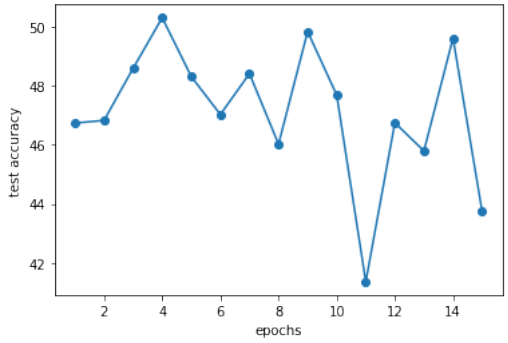

Figure 3: Batch size 1

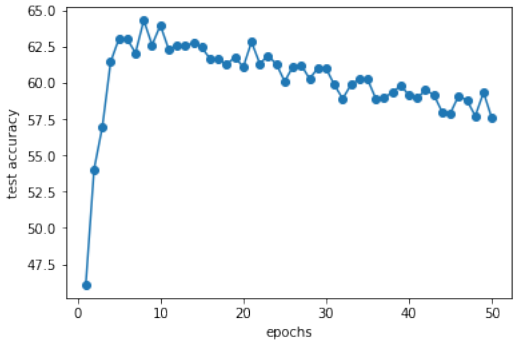

Figure 4: Batch size 4

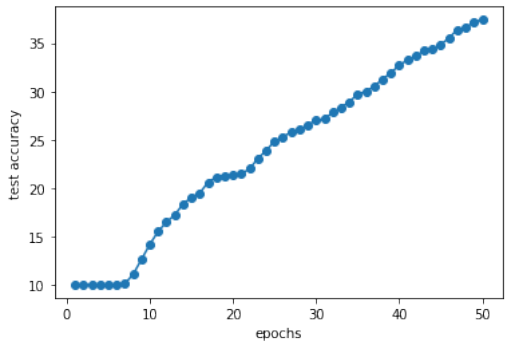

Figure 5: Batch size 1000

6. Try multiple learning rates to see the effect and describe the findings. Please use learning rates of 10, 0.1, 0.01, and 0.001.

The network gave varied results on using different learning rates. From the graphs we can infer that the model using a learning rate of 0.001 outperformed the rest.

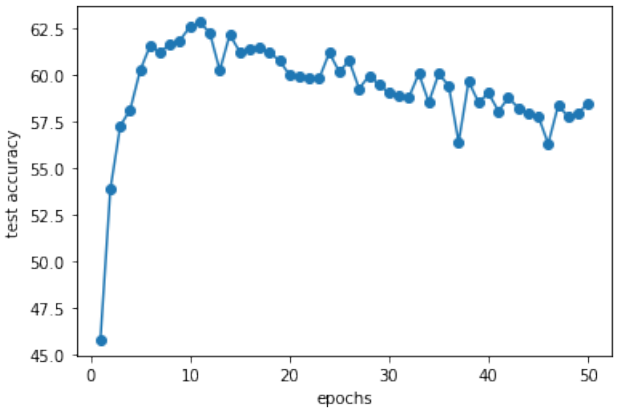

Figure 6: Learning rate 0.001

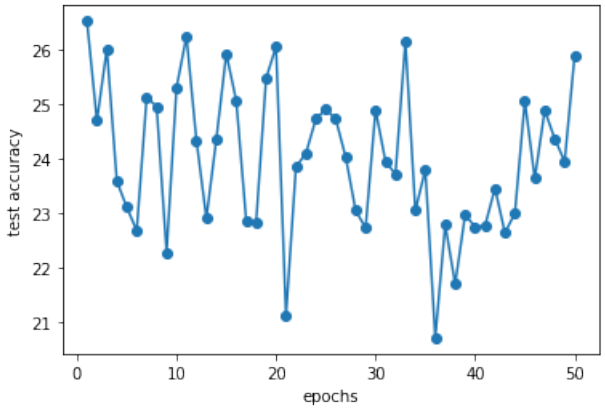

Figure 7: Learning rate 0.01

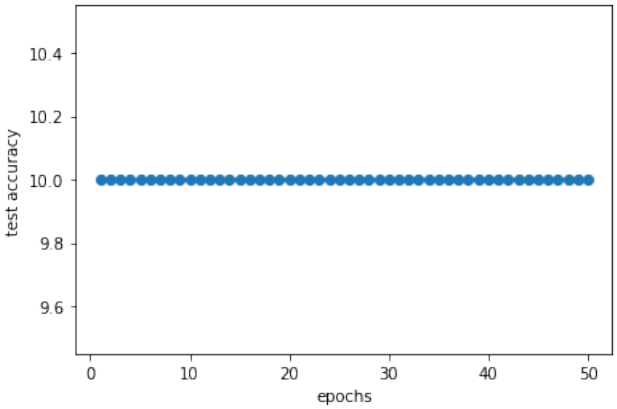

Figure 8: Learning rate 0.1

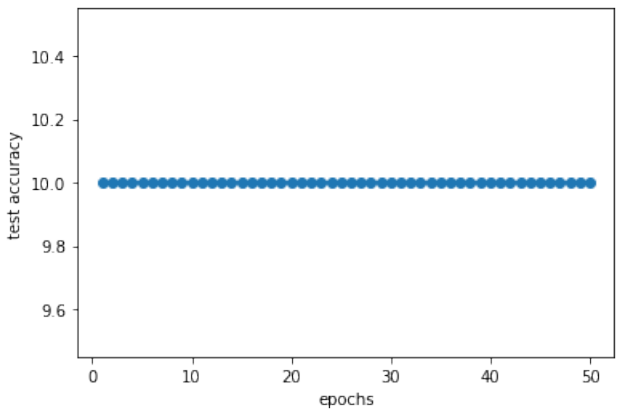

Figure 9: Learning rate 10

7. Please add some data augmentation to avoid overfitting. Note that you need to do this only for the trainnig and not the testing. You may use line 233-253 from Imagenet sample code:

   https://github.com/pytorch/examples/blob/master/imagenet/main.py

   "RandomResizedCrop" samples a random patch from the image to train the model on. "RandomHorizontalFlip" flips randomly chosen images horizontally.

8. Change the loss function from Cross Entropy to Mean Squared Error and report the effect.

For classification problems, cross-entropy is preferred over mean squared error as the loss function. On the contrary, mean squared error is preferred for regression problems. The expected behavior after changing the loss function from cross-entropy loss to mean squared error in our classification problem is that the accuracy would decrease. However, the accuracy after changing the loss function to mean squared error results in increased accuracy.

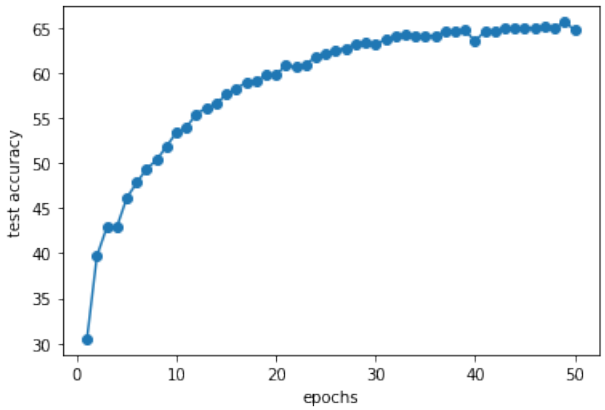

Figure 10: With mean squared error

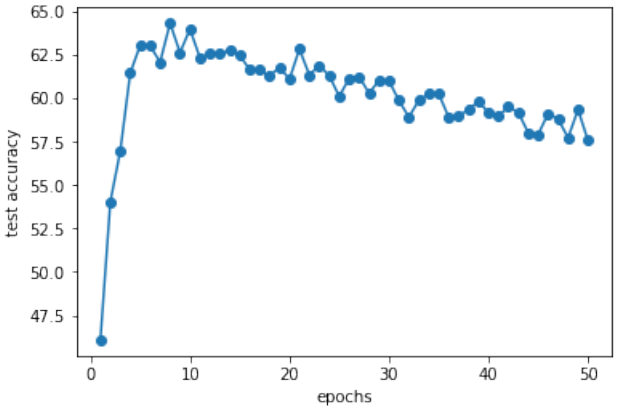

Figure 11: With cross-entropy loss

References

[1] Jason Brownlee. *A gentle introduction to the rectified linear unit (ReLU)*. Aug. 2020. URL: `https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/`.

[2] Rukshan Pramoditha. *What happens if you do not use any activation function in a neural network's hidden layer(s)?* Jan. 2022. URL: `https://rukshanpramoditha.medium.com/what-happens-if-you-do-%20not-use-any-activation-function-in-a-neural-networks-hidden-layer-s-f3ce089e4508`.