National University of Singapore
School of Computing
CS5242: Neural Networks and Deep Learning
Tutorial 3: Multilayer Networks,
Pytorch Datasets and Optimizers

Pytorch has a wide variety of datasets that one can leverage. In this tutorial, we will use one such dataset and build a multilayer neural network to fit the data. We will use PyTorch optimizers to perform the optimization process.

Q1. Use the following snippet to get a dataloader from the already available Mnist dataset from PyTorch. Mnist dataset has 60,000 digit images with its corresponding labels.
This snippet will download the dataset if not downloaded already and transform each image before returning it. Dataloaders are iterable over a dataset. For further reading follow pytorch dataloader

```
from torchvision import datasets, transforms

transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])
trainset = datasets.MNIST('~/.pytorch/MNIST_data/', download=True, train=True, transform=transform)
traintloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)
```

Q2. Wrap the dataloader with python `"iter"` class which will give you an iterable object from which you can get new images and labels using `.next()`. After applying iter, iterate through a few samples, check the shape, the datatype of the image, label tuple that you are getting. Plot some of the images. Also, print the flattened shape of each image. (hint: use `.flatten()`)

Q3. Build a multilayer neural network with 3 fully connected layers of shape `(784, 128)`, `(128, 64)` and then `(64, 10)`. Use ReLu activation in between each layer, and at the end of the output layer.
The class should have a forward function that you will manually create. The forward will take image batch as an input (image_batch * flattened_image_784_size) and return (image_batch x 10) size vectors.
Hint: Use `torch.nn.Linear` for linear layers. Follow this pytorch module to understand how to inherit the pytorch module class.

Q4. Create a cross-entropy loss. You can create it yourself (just like the last tutorial) or create it by using `torch.nn.CrossEntropyLoss` (pytorch cross-entropy loss uses logits, not softmax probabilities, so we had not applied softmax at the last layer). Initialize an SGD optimizer from `torch.optim`.

Q5. **(1 pass of the data)** Write a training loop which will

1. Get data batches from dataloader, feed it into the network and get outputs.
2. Get loss using outputs and ground truth labels.
3. Call backward on the loss (remember to clear grads before calling gradients)

4. Call optimizer.step to perform SGD update.
5. Print the accuracy, loss and amount of data used till now.

Q6. **(Multiple Passes of Data)** Write a loop to perform the training pass for 50 training iterations (50 epochs, this is outer for loop of data loader). Print and save accuracy and loss after each epoch.

Q7. Play around with the momentum of the SGD optimizer and then replace the SGD optimizer with Adam and RMSProp and analyze the loss trajectory (save the value of loss in an array after each update) of 50 training iterations for each of these types of optimizers.

Q8. Play around with the initialization function like uniform and gaussian learnt in the lecture slides.

Q9. Add batch normalization to each of the layers before ReLu activation function. Check how your results or iterations vary.