CMSC 510 HW2 – due Friday October 14th at 5pm

The goal of the homework is to gain familiarity with PyTorch (to install it, see: https://pytorch.org/get-started/locally/), a machine learning library for python that allows for defining the machine learning model and performing gradient descent for it in an automated way.

Complete 4 exercises described below, and submit via Canvas a zip file with four Jupyter Notebook files, one per each exercise. Each notebook should contain the code, as well as short reports on the results of experiments.

Exercise 1.

Train a linear classifier for the Iris dataset (a 3-class classification problem, file iris.csv in Canvas), using Mean Squared Error as loss (see pytorch_linear_Iris_MSE.py file on Canvas). Perform an analysis of the behavior of training risk and accuracy for different learning rates.

Detailed steps:
a) Use pandas to load the iris dataset. Create dummy variables for the classes

b) Define pytorch tensors for the dataset using:
torch.tensor

c) Define pytorch tensors (with gradient) for weights and biases (W & b). W should be n_features x n_classes, b should be 1 x n_classes. Initialize b to zeros (torch.zeros), and W to random values sampled from a normal distribution with null mean – try different values for the standard deviation and observe changes in the training behavior.

d) Define pytorch optimizer over variables W & b
torch.optim.SGD or torch.optim.Adam

e) Create the main loop that goes over the dataset in multiple epochs. In each epoch
e1) clear gradients (using optimizer.zero_grad)
e2) calculate linear predictions: pred=X W + b using
torch.matmul
e3) pass the linear predictions through the unipolar sigmoid: sigmoid(pred)=1/(1+exp(-pred)). Use these functions:
torch.log, torch.exp
e4) calculate the squared difference between the predictions (after sigmoid) and the true classes, for all three output neurons. Use:
torch.pow
e5) calculate risk = average the squared difference over the training samples. Use:
torch.mean

e6) calculate gradients of risk with respect to W & b (call risk.backwards)
e7) make optimizer step (using optimizer.step)
e8) calculate accuracy

Experiment with different learning rates for the two optimizers and report the behavior of the training loss and accuracy.

Exercise 2.

Train a linear classifier for the Iris dataset, using CrossEntropy as loss. Perform an analysis of the behavior of training risk and accuracy for different learning rates.

Detailed steps - follow Exercise 1, but replace MSE with CrossEntropy:

e3) pass the linear predictions through softmax (i.e., normalize the unipolar sigmoids for classes i=1,...,3 to sum up to 1 for each sample)

e4) calculate the cross entropy after softmax (- sum_{i=1}^3 y_i ln(softmax_i)).
torch.multiply, torch.log, torch.sum

e5) calculate risk = average the cross entropy over the training samples

Experiment and report results as in Exercise 1.

Exercise 3.

Starting from Exercise 2, add a split of the Iris dataset into a training set and a test set. Also, in the training loop, go over small batches of samples (e.g. 20 samples) instead of always over the whole training set. Experiment with batch size and learning rate.

Exercise 4:

Linear classifier for MNIST Digits dataset. Explore the behavior of the code from Exercise 3 on a larger, more complicated dataset and report the results.
The number of training samples is 50,000 - analyze training behavior if a random subset of 100, 500, 1000, 2000 samples is used instead. Also, experiment with the learning rate and the batch size.

For loading the dataset, use:

```
import torchvision.datasets as datasets
full_train_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=None)
full_test_dataset = datasets.MNIST(root='./data', train=False, download=True, transform=None)
x_train = full_train_dataset.data.numpy().reshape(-1,n_features).astype(dtype=np.float)/255.0;
x_test = full_test_dataset.data.numpy().reshape(-1,n_features).astype(dtype=np.float)/255.0;
y_train_cat = full_train_dataset.targets.numpy()
y_test_cat = full_test_dataset.targets.numpy()
```

Note that the download of the dataset may take long time. As with Iris, convert categorical variables for classes into dummy variables (there are 10 classes).