Assigned: 23 October

Homework #5

EE 541: Fall 2023

Due: Friday, 03 November at 23:59. Submission instructions will follow separately on canvas.

1. **Backprop Initialization for Multiclass Classification**

   The softmax function $\mathbf{h}(\cdot)$ takes an M-dimensional input vector $\mathbf{s}$ and outputs an M-dimensional output vector $\mathbf{a}$ as

$$\mathbf{a} = \mathbf{h}(\mathbf{s}) = \frac{1}{\sum_{m=1}^{M} e^{s_m}} \begin{bmatrix} e^{s_1} \\ e^{s_2} \\ \vdots \\ e^{s_M} \end{bmatrix}$$

   and the multiclass cross-entropy cost is given by

$$C = -\sum_{i=1}^{n} y_i \ln a_i$$

   where $\mathbf{y}$ is a vector of *ground truth* labels. Define the error (vector) of the output layer as:

$$\delta = \nabla_{\mathbf{s}} C = \dot{\mathbf{A}} \nabla_{\mathbf{a}} C$$

   where $\dot{\mathbf{A}}$ is the matrix of derivatives of softmax, given as

$$\dot{\mathbf{A}} = \frac{d\mathbf{h}(\mathbf{s})}{d\mathbf{s}} = \begin{bmatrix} \frac{\partial p_1}{\partial s_1} & \cdots & \frac{\partial p_M}{\partial s_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial p_1}{\partial s_M} & \cdots & \frac{\partial p_M}{\partial s_M} \end{bmatrix}.$$

   (denominator convention with the left-handed chain rule.). Show that $\delta = \mathbf{a} - \mathbf{y}$ if $\mathbf{y}$ is one-hot.

2. In this problem you manually train a multilayer perceptron (MLP) deep neural network using only `numpy`. Do not use `tf.keras`, `PyTorch`, `scikitlearn`, *etc.* – use only `numpy`. The purpose of this problem is to understand backprop and its implementation before we hand the calculation to PyTorch Autograd.

   This problem uses the same MNIST image classification problem as the previous homework. So you should build upon the Python code you wrote for the feedforward (inference) processing.

   The file `mnist_traindata.hdf5` contains 60,000 images in the key `xdata` and the corresponding labels in the key `ydata`. Split into 50,000 training images and 10,000 validation images.

   Create a neural network for classification. It must have 784 neurons in the input layer and 10 neurons in the output layer. You may use any number of intermediate (hidden) layers with any number of neurons. It must be a fully connected MLP (don't use convolutional or other layers). Construct your MLP with:

a. **Hidden layer activations:** Try tanh and ReLU. **Write your own functions for these**, and their derivatives. Note: derivative of ReLU at 0 can be any number in [0,1].

b. **Cost function:** Cross entropy.

c. **Optimizer:** Stochastic gradient descent. Try 3 different values for learning rate. Momentum is optional. Remember that you can only use numpy, so momentum complicates the implementation.

d. **Minibatch size:** Your choice. Pick a number between 50 and 1000 that divides by 50,000 (*i.e.,* don't pick 123). Average the gradients for weight and bias updates over each minibatch – *e.g.,* minibatch size of 100 means 50,000/100 = 500 updates per epoch.

e. **Regularizer:** Your choice. If you choose to use it be mindful when calculating cost gradient.

f. **Parameter initialization:** Your choice. Don't initialize weights with zeroes. You can use random values like $w_{ij} \sim U[0,1]$ or $w_{ij} \sim \mathcal{N}(0,1)$ or use a standard algorithm like He or Xavier normal.

g. **Input preprocessing, batch normalization:** None

Train the network for 50 epochs using the 50,000 training images. At the end of each epoch complete a forward pass with the validation set and record the accuracy (*e.g.,* 9630 correct out of 10000 means accuracy is 96.3%). DO NOT use the validation set for training. Perform only forward inference with the validation set.

(a) Learning rate decay: Divide the initial learning rate by 2 twice during training. You can do this after any epoch, for example, after epochs 20 and 40. (Do not do this in the middle of an epoch). The final learning rate should be 1/4th of the initial value.

(b) Repeat training and validation for each of the 6 configurations — using 2 different activation functions and 3 initial learning rates. Make 6 plots each with 2 curves – training accuracy (y-axis) and validation accuracy (y-axis) vs. epoch number (x-axis). Mark the epochs where you divided learning rate.

(c) Choose the configuration (activation and initial learning rate) with the best validation accuracy. Train this for all 60,000 images = training + validation. Remember to apply a learning rate decay as before. After all 50 epochs are done test with the data in `mnist_testdata.hdf5` from the previous assignment. Record the final test accuracy.

Remember: Do not touch the test set from `mnist_testdata.hdf5` until the very end after you decide which configuration is best.

You have a lot of freedom in this problem — welcome to deep learning! This problem does not constrain the number of layers or the number of neurons in each layer. Experiment and don't worry about low accuracy. A reasonable number for final test accuracy is 95%-99%. Submit the following:

- Network configuration – how many layers and how many neurons.
- Batch size
- 3 different values of initial learning rate
- How you initialized parameters
- 6 curves, as described
- Final test accuracy for the best network

Note: Researchers constructed the Fashion-MNIST dataset for classification because MNIST became "too easy" with modern deep learning techniques. Use the in-lecture experiments on Fashion-MNIST to provide guidance for this problem.