EE599 Deep Learning – Homework 3

©K.M. Chugg

February 22, 2020

assigned: Feb. 22, 2020
due: March 4, 2020.

1 Backprop in Numpy: Training MLP to classify MNIST

The purpose of this problem is to have you program, using numpy, the training for a multilayer perceptron (MLP) deep neural network. You cannot use tf.keras, pytorch, scikitlearn, etc. – just use numpy. The purpose of this problem is to make sure you understand the basic backprop operations before we move onto using tf.keras.

This is based on the same MNIST image classification problem as HW2, problem 1, so you can build upon the Python code you wrote for the feedforward (inference) processing.

(a) The file `mnist_traindata.hdf5` contains 60,000 images in the key 'xdata' and their corresponding labels in the key 'ydata'. Format is same as before. Extract these. Split them into 50,000 images for training and 10,000 for validation.

(b) Design a neural network. It must have 784 neurons in the input layer and 10 neurons in the output layer. *You can have any number of intermediate layers with any number of neurons.* It must be an MLP (don't use convolutional or other layers). Train this network on the 50,000 training set with the following settings:

(c) Hidden layer activations: Try tanh and ReLU. You'll need to write your own functions for these, and their derivatives. For ReLU, derivative at 0 can be any number in $[0, 1]$.

(d) Cost function: Cross entropy.

(e) Optimizer: Stochastic gradient descent. Try 3 different values for initial learning rate. Momentum is optional. Remember that you can only use numpy, so momentum might make things too complicated.

(f) Minibatch size: Your choice. Pick a number between 50 and 1000 which is divisible by 50,000 (for example, don't pick 123). Remember to average the gradients for a minibatch before doing 1 update to all the weights and biases. So if your minibatch size is 100, there will be 50,000/100 = 500 updates per epoch.

(g) Regularizer: Your choice. If you decide to use it, be careful in calculating gradient of cost.

(h) Parameter initialization: Your choice. Don't initialize weights with zeroes. You can use random values, or one of the techniques mentioned in class like He normal (will be covered in next lecture).

(i) Input preprocessing, batch normalization: None

(j) **Train the network for 50 epochs**. At the end of each epoch, run feedforward on the validation set and note the accuracy (Eg: 9630 correct out of 10000 means accuracy is 96.3%). Remember that the validation set should NOT be used for training. You should only run feedforward on the validation set, like what you did in HW2 prob1.

(k) Learning rate decay: Divide your 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). So your final learning rate should be 1/4th of the initial learning rate.

(l) Repeat training and validation for all 6 configurations – 2 activations and 3 initial learning rates. Make 6 plots, each with 2 curves – training accuracy (y-axis) and validation accuracy (y-axis) vs number of epochs (x-axis). Mark the epochs where you divided learning rate.

(m) Choose the configuration (activation and initial learning rate) which gave best validation accuracy. Now *train this for all 60,000 images = training+validation.* Remember to apply learning rate decay same as before. After all 50 epochs are done, test on the test set defined in `mnist_testdata.hdf5` from the previous homework. Note your test accuracy.

Remember: Do not touch the test set from `mnist_testdata.hdf5` until the very end, when you have decided which configuration is the best.

Report:

- 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 mentioned
- Final test accuracy for the best network

You are given too much freedom in this problem! We have not specified anything about the number of layers and the number of neurons in each. Feel free to play around and don't worry too much if your accuracy is bad. A reasonable number for final test accuracy is 95%–99%. **Note/Hint:** the fashion-MNIST dataset and classification problem was introduced because this MNIST problem became “too easy” as deep learning techniques advanced. Since you have seen training results for Fashion-MNIST, this should give you some insight into this problem.

2 Confusion Matrix for Fashion MNIST

Train an MLP for Fashion MNIST with one hidden layer with 100 nodes and ReLu activations. Use a dropout layer with 30% dropout rate at the hidden layer output and an L2 regularizer with coefficient 0.0001. You can use the sample code from lecture (learning curves for these network and training configuration are in the slides).

Evaluate this trained model on the test data and produce the so-called "confusion matrix" – *i.e.,* find the rate of classification decisions conditioned on the true class. Element (i, j) of the confusion matrix is the rate at which the network decides class j when class i is the label (ground truth).

1. Print a well-formatted table of the confusion matrix. You may also consider plotting a heat map of the confusion matrix for visualization purposes (this is not required).
2. Consider class m – what is the class most likely confused for class m? List this for $m \in \{0, 1, \ldots 9\}$.
3. Overall, what two classes (types of clothing) are most likely to be confused?

3 Exploring Regularization on Fashion MNIST

For this problem, you will train two models on Fashion MNIST:

1. One hidden layer with ReLu activation, 128 nodes. No regularization, no dropout.
2. One hidden layer with ReLu activation, 48 nodes. L2 regularization with coefficient 0.0001 and doprout with rate 0.2 at the hidden layer.

You can use the example scripts from lecture to train these networks (learning curves for these network and training configuration are in the slides). Train for 40 epochs.

Produce histograms for the weights of these two networks – a separate histogram for the two junctions (layers) for each case. Describe the qualitative differences that you see in these histograms.

4 MLP for the Human/Machine Binary Data

In this problem, you will use tf.keras to train a simple MLP that takes as input a (20×1) binary vector and outputs a probability that the vector was generated by a human – *i.e.,* this is one minus the probability that it was generated by a machine. Guidelines for this

- Use a single hidden layer with 20 neurons and ReLu activation.
- Use a single output neuron with logistical sigmoid function and binary cross-entropy loss. The data label should be 1 when the sequence was human-generated and 0 if it was machine-generated.
- Use an 80/20 training and validation split.
- Use a batch size of 16.

- Plot learning curves for your training as you go – *i.e.,* probability of correct classification vs. number of epochs. You should have two of these curves on one plot; one for training and the other for validation. Ensure that you are not over-fitting by monitoring these curves.
- Explore the use of various optimizers and regularizers as you see fit. You may also explore other neural network architectures and activations, but this is not required.

When your training is complete, print out the (20×20) weight matrix for the junction between the input and hidden layers. Does this give you any insight into feature design for this problem?

You will submit:

1. A tf.keras model file in hdf5 format for the auto-grader.
2. A learning curve for your training – *i.e.,* probability of correct classification vs. number of epochs. You should have two of these curevs on one plot; one for training and the other for validation.