Homework 5, Due May 17

MPCS 53111 Machine Learning, University of Chicago

Practice problem, do not submit

1. There are several activation functions other than the logistic function (also called the *sigmoid* function). Please research (e.g., see Stanford CS231 notes) the following functions: *tanh, ReLU,* and *leaky ReLU* and answer the following:

   (a) What is the advantage of tanh over the logistic function?

   (b) What is the advantage of the leaky ReLU over ReLU?

Graded problems, submit

2. Implement a Python class `CNNLayer` for a convolutional layer in the convolutional neural networks. In particular, implement the following methods:

   - `__init__(n, filter_shape, stride, activation)`: Initializes an convolutional layer object where `n` is the number of filters, `filter_shape` is the 3-dimensional shape of each filter, `(height, width, depth)`, and `activation` is the type of activation to use. You should implement at least `"relu"` and `"no_act"`, the ReLU and *no activation* (for debugging).
   - `forward_step(X, pad)`: Perform a forward convolution step using the convolution layer. X is the input data with shape (N, h, w, d), where N is the number of input data examples, and h, w, and d, are the height, width, and depth of the input data (for each example)[1], and `pad` is the padding to be applied. Return the appropriate output data.
   - `backward_step(out_delta)`: Perform a backward step during the backpropagation, based on `delta`, the Δ at the next layer, and return the Δ at this layer. Here you should check that the

[1] In practice, height and width are invariably the same; here we simply want to specify them explicitly.

shape of `delta` is consistent with what you observed during the `forward_step`.

- `update()`: Perform a step of gradient descent based on the Δ computed in `backward_step()`.
- `print()`: Print the current weights of the filters.
- `set_filters(filters)`: Set the filters to the given values. This is usefully for testing.

Please include other parameters and functions as needed, such as those for maintaining and updating bias values.

3. Use the above `CNNLayer` implementation to implement a Python class CNN for a convolution neural network. Include the following methods:

   - `__init__()`: Initialize a convolutional neural network.
   - `fit(X, y, alpha, t)`: Train the network using back propagation, in which `X` is the training data, `y` is the set of labels for the training data, `alpha` is the learning rate, and `t` is the number of iterations. Use multi-class hinge loss.
   - `predict(T)`: Return the predicted labels of test examples.

   Include other methods and parameters as needed.

4. Use your convolutional neural network for classifying the MNIST dataset of handwritten digits. The network should contain two convolutional layers, each with 32 filters of size 5×5, followed by a fully-connected layer of 10 nodes for ten-class classification. Report your results in a discussion section using appropriate plots.

5. Now we want to get hand-on experience with deep learning in real application. In this problem, we will use Tensorflow, a open source software library for numerical computation and deep learning developed by Google.

   Please complete the tutorial (https://www.tensorflow.org/tutorials/deep_cnn), and learn how to build a convolutional neural network using Tensorflow and how to train/test the model. Then build a convolutional neural network using the following architecture to solve the CIFAR-10 classification problem:

Layer 1: Convolutional layer with 64 filters of size 3×3, use rectify linear unit as activation function.
Layer 2: Convolutional layer with 64 filters of size 3×3, use rectify linear unit as activation function.
Layer 3: 2×2 max-pooling layer.
Layer 4: Convolutional layer with 128 filters of size 3×3, use rectify linear unit as activation function.
Layer 5: Convolutional layer with 128 filters of size 3×3, use rectify linear unit as activation function.
Layer 6: 2×2 max-pooling layer.
Layer 7: Convolutional layer with 256 filters of size 3×3, use rectify linear unit as activation function.
Layer 8: 2×2 max-pooling layer.
Layer 9: Fully-connected layer with 256 nodes, use rectify linear unit as activation function.
Layer 10: Fully-connected layer with 10 nodes for classification, use softmax as activation function.

Put your implementation in a separate python file `hw5-tf.py`. The python file needs to parse the command-line options and arguments to specify the execution mode. More specifically, `python hw5-tf.py -train` executes the code for training and `python hw5-tf.py -test` executes the code for testing. The following functions need to be implemented:

- `main`: The program startup function.
- `build_cnn`: Construct the ten-layer CNN described above following the Tensorflow tutorial.
- `train`: Train the CNN on the CIFAR-10 training dataset. Print out the training loss and accuracy at every training epoch.
- `test`: Test the CNN on the CIFAR-10 testing dataset. Print out the final test accuracy.