Sample Neural Network model

Dataset:

The sample neural network model for our class performs wheat kernel seed classification which I got from http://archive.ics.uci.edu/ml/datasets/seeds. The seed classes are of Type 1, 2 and 3. Each seed has seven geometric features, all of which are real-valued, which include

(1) area, A
(2) perimeter, P
(3) compactness, C = 4*pi*A/(P^2)
(4) length of kernel,
(5) width of kernel,
(6) asymmetry coefficient
(7) length of kernel groove

The dataset has 210 samples, 70% of which are used as training dataset and 30% are used as test dataset, saved in the train_seeds.csv and test_seeds.csv, respectively. Each line in the CSV files corresponds to one sample, with the first entry being the label and the next entries being the seven geometric features.

The model:

The model that I used is a 3-layer NN with the first hidden and second hidden layers having 5 hidden nodes each, and the last layer having, of course, 3 output nodes. For each hidden layer, I used hyperbolic tangent as the non-linear activation function. The bias weights for the input and hidden layers are initialized with 0 while the synaptic weights are initialized randomly. See the train.prototxt file for other details.

Implementation:

The implementation of the model is done using Caffe framework with the train.protoxt describing the initialization of the NN model and the deploy.protoxt being exactly the same as train.prototxt without the learning hyperparameters. Other learning parameters are stored in the solver.protoxt.

One observation that can be seen from the train.protoxt and deploy.protoxt which are typically not in Caffe framework models, are the python layers – these python layers allow the user to come up with their own layers. The custom python layer that is introduced is the CustomDaya layer which is basically an input layer.

Two python programs are also provided namely (1) the finetuneMLP.py to train the train.protoxt and to save the learned weights called the caffemodel inside the train_output folder; and (2) the MLPnet.py to perform classification of the test dataset by calling the saved caffemodel and the deploy.prototxt.

Some notes on setting the parameters (i.e. synaptic weights) and hyperparameters (e.g. number of nodes, layers, and learning parameters):

As I have mentioned in class, NN is more of an art rather than a science. You should know how each parameter influence the learning of your NN model.

For example, if you have more number of hidden layers or more number of hidden nodes, then it means you more synaptic weight parameters that need to adjust or learn. Therefore, you need more training time – you may need to increase the number of epochs and/or the number of steps, in case your learning policy is “step”.

Learning rate affects how fast your system learns. If during the training, you observed that your system loss does not decrease, one reason is maybe you don’t have enough number of layers or hidden nodes to model your input-output training set. What I do is to gradually increase these hyperparameters and see if the training loss improves. Another reason is maybe you have a very big learning rate (hyperparameters that are involved include: base_lr in the solver.prototxt and the lr_mult in the train.prototxt). What I do is to set the lr_mult to 1 first and use base_lr equal to 0.1 as a starting value.

I encourage you to understand these hyperparameters, although the discussion is mathematical, because this is the only way that you could design your NN model efficiently, and not seeing NN as entirely a black box. Details of the hyperparameters can be seen from this link: http://caffe.berkeleyvision.org/tutorial/solver.html

Some of these parameters will be discussed in our next meeting on backpropagation algorithm.