Machine Learning from Data - IDC
HW7 – Deep Learning

Instructions: Read the following carefully before starting the assignment.

- Submitting in pairs is allowed.
- You must submit a zip file containing two folders – `out` and `src`. Do not submit the folder `res`.
  - The folder `out` should contain 2 folders and 1 file – the folder `my_vanilla` containing the weights saved for the vanilla architecture, the folder `conv` containing the weights saved for the convolutional architecture, and the text file `hw7.txt`.
  - The folder `src` should contain 4 files – `l1train.npz, l1valid.npz, my_mnist.py, ball_indicator.py`
  - Make sure you zipped the two folders with all 7 files required before submitting.
- Do not change anything in the code or the folder hierarchy you were not specifically instructed to. Points will be deducted in case of a non running solution.
- Due date – 04/07/2018.

1 Installing Python and TensorFlow

In this assignment, we will use the deep learning package TensorFlow, written in Python. Use the attached guide to install Python, TensorFlow and the PyCharm IDE.

2 Width vs Depth

In this part, we will begin by doing regression on fully-connected feed-forward neural networks of varying architecture on artificial data.

- The networks used will have ReLU activations in the hidden layers and a linear activation ($z \mapsto z$) in the output neuron.

- You are supplied with a training set file `l1train.npz` containing 10,000 instances and a validation set file `l1valid.npz` containing 2,000 instances. Both were generated randomly and labeled according to the concept

$$\mathbf{x} \mapsto \begin{cases} 1 & \sum_{i=1}^{10} |x_i| \leq 1 \\ 0 & \sum_{i=1}^{10} |x_i| > 1 \end{cases}.$$

  That is, the target value is 1 if the sum of the absolute values of the 10 features is at most 1, and is 0 otherwise.

- Train 5 networks on the given data for 200,000 epochs. Open the file `ball_indicator.py`, edit the architecture defined by the variable `arch` in line 5, and run your code. For example, defining `arch = [10,10,1]` will define a two-hidden layer network with hidden layers of width 10 and a single output neuron.

  - Train 4 one-hidden layer networks of widths $100, 200, 400, 800$ on the data.
  - Train a two-hidden layer network with first layer of size 100 and second layer of size 20 on the data.
  - How many weights does each of the 5 architectures above (including bias terms) have? Which is the architecture with the best training/validation error? Briefly explain the results you got. Provide the answers in a text file called `hw7.txt`.

3 Training a Vanilla Neural Network on MNIST

Train a vanilla feed-forward fully-connected neural network on MNIST our code splits the 60,000 images to training set of size 55,000 and a validation set of size 5,000. Open the file `my_mnist.py`, and edit the architecture of your choice by modifying the `my_vanilla_neural_net` function in line 65.

- The function currently defines a network with two-hidden layers, the first of size `N1`, and the second of size `N2`.

- To define a fully-connected layer of size `N` named `fc`, use the command `fc = tf.layers.dense(previous_layer, N)`. The first argument is the name of the layer preceding `fc`, and the second argument is the number of neurons in the defined layer.

- Experiment with different training parameters by changing the parameters appearing in lines 165-168 of the code.

  - `learning_rate` is the gradient step size, similar to what was learned in class in the gradient descent algorithm.

- `batch_size` is the number of instances the gradient is being computed over at each iteration. Since deep learning usually involves data comprising of millions of examples, it is common to run each update on a subset of the data. The larger the subset is, the more accurate the computation is, but also the more time consuming. If using a batch size of 100 for example, the next gradient descent update will use the next 100 instances in the data, until a full sweep over the data is completed, called an epoch.
- `n_epochs` specifies the number of epochs to train over. The more we use the better we can converge, but also the more time the algorithm will take to finish.
- `display_step` is a parameter allowing you to choose how often the current training progression is printed to console. setting it to 1 for example will result in an update once every epoch is completed.

- Once you find an architecture you are satisfied with, make sure that line 178 is commented out and that line 177 is not. Assert that the weights you produce are saved by setting line 198 to `train_test_model(save_final_model=True)`, and uncomment line 199 to verify the weights were indeed saved. Now train your network for a feasible amount of time, striving for the best result you can.

4 Training a ConvNet on MNIST

Train a convolutional neural network on MNIST. In the same file `my_mnist.py`, edit the architecture of your choice by modifying the `conv_net` function in line 37.

- The function currently defines a network with a single convolutional layer with `F1` filters (meaning there are `F1` different groups, each with a connected set of weights of its own) with a kernel (window) size of $K1 \times K1$, followed by a max pooling layer with window sizes $K2 \times K2$ and stride `S2`, which is then followed by a fully-connected layer of size `N`, and finally the output layer of size 10.

- To define a convolutional layer named `conv` with `F` filters, window size $K \times K$, and a ReLU activation, use the command `conv = tf.layers.conv2d(previous_layer, F, K, activation=tf.nn.relu)`. The first argument is the name of the layer preceding `conv`, the second argument is the number of filters in the layer, the third argument is the window size, and the fourth argument is the type of the activation function used.

- To define a max-pooling layer named `pool` with stride `S` and window size $K \times K$, use the command `pool = tf.layers.max_pooling2d(previous_layer, S, K)`. The first argument is the name of the layer preceding `conv`, the second argument is the stride size, and the third argument is the window size.

- Note that in order for us to define a fully connected layer following a conv/max-pooling layer, we need to flatten the output of the conv/max-pooling layers using the command in line 52.

- Experiment with different training parameters by changing the parameters appearing in lines 165-168 of the code.
- Once you find an architecture you are satisfied with, make sure that line 177 is commented out and that line 178 is not. Assert that the weights you produce are saved by setting line 198 to `train_test_model(save_final_model=True)`, and uncomment line 199 to verify the weights were indeed saved. Now train your network for a feasible amount of time, striving for the best result you can.
- Which architecture had a better performance, the one you chose in section 3 or in section 4? Explain your answer briefly in the text file `hw7.txt`.
- Use the confusion matrix produced at the end of the optimization to analyze the predictor you got. Where does it more commonly errs? Explain your findings briefly in the text file `hw7.txt`.