1 Boosting (20 Points)

In this problem, you will develop an alternative way of forward stagewise boosting. The overall goal is to derive an algorithm for choosing the best weak learner h_t at each step such that it best approximates the gradient of the loss function with respect to the current prediction of labels. In particular, consider a binary classification task of predicting labels $y_i \in \{+1, -1\}$ for instances $\mathbf{x}_i \in \mathbb{R}^d$, for $i = 1, \ldots, n$. We also have access to a set of weak learners denoted by $\mathcal{H} = \{h_j, j = 1, \ldots, M\}$. In this framework, we first choose a loss function $L(y_i, \hat{y}_i)$ in terms of current labels and the true labels, e.g. least squares loss $L(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$. Then we consider the gradient g_i of the cost function $L(y_i, \hat{y}_i)$ with respect to the current predictions $\hat{y}_i$ on each instance, i.e. $g_i = \frac{\partial L(y_i, \hat{y}_i)}{\partial \hat{y}_i}$. We take the following steps for boosting:

(a) **Gradient Calculation** (4 points) In this step, we calculate the gradients $g_i = \frac{\partial L(y_i, \hat{y}_i)}{\partial \hat{y}_i}$.

(b) **Weak Learner Selection** (8 points) We then choose the next learner to be the one that can best predict these gradients, i.e. we choose

$$h^* = \arg\min_{h \in \mathcal{H}} \left(\min_{\gamma \in \mathbb{R}} \sum_{i=1}^{n} (-g_i - \gamma h(\mathbf{x}_i))^2 \right)$$

We can show that the optimal value of the step size γ can be computed in the closed form in this step, thus the selection rule for h^* can be derived independent of γ.

(c) **Step Size Selection** (8 points) We then select the step size α^* that minimizes the loss:

$$\alpha^* = \arg\min_{\alpha \in \mathbb{R}} \sum_{i=1}^{n} L(y_i, \hat{y}_i + \alpha h^*(\mathbf{x}_i)).$$

For the squared loss function, α^* should be computed analytically in terms of y_i, $\hat{y}_i$, and h^*. Finally, we perform the following updating step:

$$\hat{y}_i \leftarrow \hat{y}_i + \alpha^* h^*(\mathbf{x}_i).$$

In this question, you have to derive all the steps for squared loss function $L(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$.

2 Neural Networks (20 Points)

(a) (8 points) Show that a neural network with a single logistic output and with linear activation functions in the hidden layers (possibly with multiple hidden layers) is equivalent to the logistic regression.

(b) (12 points) Consider the neural network in figure 1 with one hidden layer. Each hidden layer is defined as $z_k = \tanh(\sum_{i=1}^{3} w_{ki} x_i)$ for $k = 1, \ldots, 4$ and the outputs are defined as $y_j = \sum_{k=1}^{4} v_{jk} z_k$ for $j = 1, 2$. Suppose we choose the squared loss function for every pair, i.e. $L(\mathbf{y}, \widehat{\mathbf{y}}) = \frac{1}{2}\left((y_1 - \widehat{y}_1)^2 + (y_2 - \widehat{y}_2)^2\right)$, where y_j and $\widehat{y}_j$ represent the true outputs and our estimations, respectively. Write down the backpropagation updates for estimation of w_{ki} and v_{jk}.

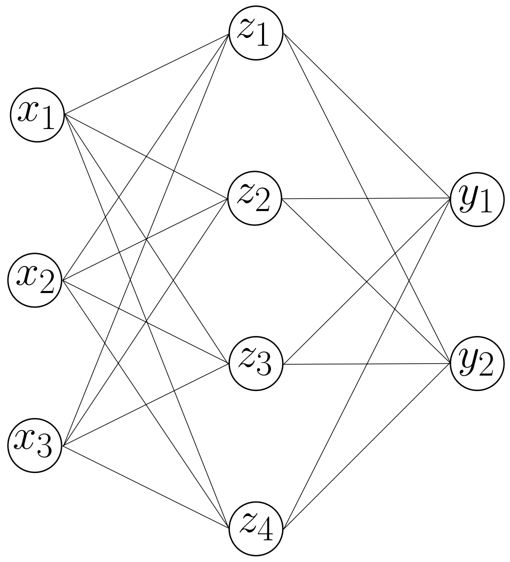

Figure 1: A neural network with one hidden layer

Programming

Deep Learning (60 Points)

In this programming problem, you will be introduced to deep learning via hands on experimentation. We will explore the effects of different activation functions, training techniques, architectures and parameters in neural networks by training networks with different architectures and hyperparameters for a classification task.
For this homework, we highly recommend using the Google Cloud to run your code since training neural networks can take several tens of hours on personal laptops. You will need all the multi-core speedup you can get, to speed things up. We will only work with Python this time (no MATLAB), since all the deep learning libraries we need, are freely available only for Python.
There is an accompanying code file along with this homework titled `hw_utils.py`. It contains four functions which are all the functions you will need for the homework. You will not have to write any deep learning code by yourself for this homework, instead you will just call these helper functions with different parameter settings. Go over the file `hw_utils.py` and understand what each of the helper functions do.

(a) **Libraries**: Launch a virtual machine on the Google Cloud (please use a 64-bit machine with Ubuntu 16.04 LTS and the maximum number of CPU cores you can get). You need to install the following libraries:

- **Python Package Manager (pip)**: `sudo apt-get install python-pip`
- **Numpy and Scipy**: Standard numerical computation libraries in Python. Install with:

```
sudo apt-get install python-numpy python-scipy
```

- Theano: Analytical math engine. Install with:

```
sudo apt-get install python-dev python-nose g++ libopenblas-dev git
sudo pip install Theano
```

- Keras: A popular Deep Learning library. Install with:

  ```
  sudo pip install keras
  ```

- **Screen**: For saving your session so that you can run code on the virtual machine even when you are logged out of it. Install with:

  ```
  sudo apt-get install screen
  ```

Next, configure Keras to use **Theano** as its backend (by default, it uses **TensorFlow**). Open the **Keras** config file and change the `backend` field from `tensorflow` to `theano`. The Keras config file `keras.json` can be edited on the terminal with nano:

```
nano ~/.keras/keras.json
```

(b) **Useful information for homework**: We will only use fully connected layers for this homework in all networks. We will refer to network architecture in the format: $[n_1, n_2, \cdots, n_L]$ which defines a network having L layers, with n_1 being the input size, n_L being the output size, and the others being hidden layer sizes, e.g. the network in figure 1 has architecture: $[3, 4, 2]$.

Checkout the various activation functions for neural networks namely linear ($f(x) = x$), sigmoid, ReLu and softmax. In this homework we will always use the softmax activation for the output layer, since we are dealing with a classification task and output of softmax layer can be treated as probabilities for multi-class classification.

Have a look at the last part of the homework (hints and tips section) before you start the homework to get some good tips for debugging and running your code fast.

A brief description of the functions in the helper file `hw_utils.py` is as follows:

- `genmodel()`: Returns a neural network model with the requested shape, activation function and L2-regularizer. You won't need to call this method at all.
- `loaddata()`: Loads the dataset for this homework, shuffles it, generates labels, bifurcates the data and returns the training and test sets.
- `normalize()`: Normalizes the training and test set features.
- `testmodels()`: It takes the following parameters: your training and test data, a list of model architectures, activation function (hidden layers and last layer), list of regularization coefficients, number of epochs for stochastic gradient descent (SGD), batch size for SGD, learning rate for SGD, list of step size decays for SGD, list of SGD momentum parameters, boolean variable to turn nesterov momentum on/off, boolean variable to turn early stopping on/off and another boolean variable to turn the verbose flag on/off. The method generates a model of appropriate size and trains it on your training data. It prints out the test set accuracy on the console. In case of list of parameters, it trains networks for all possible combinations of those parameters and also reports the best configuration found (i.e. the configuration which gave the maximum test accuracy). This is the method that you will have to call a lot in your code.

Lastly, try running the experiments multiple times if needed, since neural networks are often subject to local minima and you might get suboptimal results in some cases.

(c) **Dataset and preprocessing**: We will use the MiniBooNE particle identification dataset from the UCI Machine Learning Repository. It has 130065 instances with 50 features each and each instance has to be classified as either "signal" or "background".

Download the dataset and call `loaddata()` in your code to load and process it. The function loads the data, assigns labels to each instance, shuffles the dataset and randomly divides it into training (80%) and test (20%) sets. It also makes your training and test set labels categorical i.e. instead of a scalar "0" or "1", each label becomes a two-dimensional tuple; the new label is (1,0) if the original label is "0" and it is (0,1) if the original label is "1". The dimension of every feature is $d_{in} = 50$ and the dimension of output labels is $d_{out} = 2$. Next, normalize the features of both the sets by calling `normalize()` in your code.

(d) **Linear activations**: (5 Points) First we will explore networks with linear activations. Train models of the following architectures: $[d_{in}, d_{out}]$, $[d_{in}, 50, d_{out}]$, $[d_{in}, 50, 50, d_{out}]$, $[d_{in}, 50, 50, 50, d_{out}]$ each having linear activations for all hidden layers and softmax activation for the last layer. Use 0.0 regularization parameter, set the number of epochs to 30, batch size to 1000, learning rate to 0.001, decay to 0.0, momentum to 0.0, Nesterov flag to False, and Early Stopping to False. Report the test set accuracies and comment on the pattern of test set accuracies obtained. Next, keeping the other parameters same, train on the following architectures: $[d_{in}, 50, d_{out}]$, $[d_{in}, 500, d_{out}]$, $[d_{in}, 500, 300, d_{out}]$, $[d_{in}, 800, 500, 300, d_{out}]$, $[d_{in}, 800, 800, 500, 300, d_{out}]$. Report the observations and explain the pattern of test set accuracies obtained. Also report the time taken to train these new set of architectures.

(e) **Sigmoid activation**: (5 Points) Next let us try sigmoid activations. We will only explore the bigger architectures though. Train models of the following architectures: $[d_{in}, 50, d_{out}]$, $[d_{in}, 500, d_{out}]$, $[d_{in}, 500, 300, d_{out}]$, $[d_{in}, 800, 500, 300, d_{out}]$, $[d_{in}, 800, 800, 500, 300, d_{out}]$; all hidden layers with sigmoids and output layer with softmax. Keep all other parameters the same as with linear activations. Report your test set accuracies and comment on the trend of accuracies obtained with changing model architectures. Also explain why this trend is different from that of linear activations. Report and compare the time taken to train these architectures with those for linear architectures.

(f) **ReLu activation**: (5 Points) Repeat the above part with ReLu activations for the hidden layers (output layer = softmax). Keep all other parameters and architectures the same, except change the learning rate to 5×10^{-4}. Report your observations and explain the trend again. Also explain why this trend is different from that of linear activations. Report and compare the time taken to train these architectures with those for linear and sigmoid architectures.

(g) **L2-Regularization**: (5 Points) Next we will try to apply regularization to our network. For this part we will use a deep network with four layers: $[d_{in}, 800, 500, 300, d_{out}]$; all hidden activations ReLu and output activation softmax. Keeping all other parameters same as for the previous part, train this network for the following set of L2-regularization parameters: $[10^{-7}, 5 \times 10^{-7}, 10^{-6}, 5 \times 10^{-6}, 10^{-5}]$. Report your accuracies on the test set and explain the trend of observations. Report the best value of the regularization hyperparameter.

(h) **Early Stopping and L2-regularization**: (5 Points) To prevent overfitting, we will next apply early stopping techniques. For early stopping, we reserve a portion of our data as a validation set and if the error starts increasing on it, we stop our training earlier than the provided number of iterations. We will use 10% of our training data as a validation set and stop if the error on the validation set goes up consecutively six times. Train the same architecture as the last part, with the same set of L2-regularization coefficients, but this time set the Early Stopping flag in the call to `testmodels()` as `True`. Again report your accuracies on the test set and explain the trend of observations. Report the best value of the regularization hyperparameter this time. Is it the same as with only L2-regularization? Did early stopping help?

(i) **SGD with weight decay**: (5 Points) During gradient descent, it is often a good idea to start with a big value of the learning rate (α) and then reduce it as the number of iterations progress i.e.

$$\alpha_t = \frac{\alpha_0}{1 + \beta t}$$

In this part we will experiment with the decay factor β. Use the network [d_{in}, 800, 500, 300, d_{out}]; all hidden activations ReLu and output activation softmax. Use a regularization coefficient = 5 × 10−7, number of epochs = 100, batch size = 1000, learning rate = 10^{-5}, and a list of decays: $[10^{-5}, 5 \times 10^{-5}, 10^{-4}, 3 \times 10^{-4}, 7 \times 10^{-4}, 10^{-3}]$. Use no momentum and no early stopping. Report your test set accuracies for the decay parameters and choose the best one based on your observations.

(j) **Momentum**: (5 Points) Read about momentum for Stochastic Gradient Descent. We will use a variant of basic momentum techniques called the Nesterov momentum. Train the same architecture as in the previous part (with ReLu hidden activations and softmax final activation) with the following parameters: regularization coefficient = 0.0, number of epochs = 50, batch size = 1000, learning rate = 10^{-5}, decay = best value found in last part, Nesterov = True, Early Stopping = False and a list of momentum coefficients = [0.99, 0.98, 0.95, 0.9, 0.85]. Find the best value for the momentum coefficients, which gives the maximum test set accuracy.

(k) **Combining the above**: (10 Points) Now train the above architecture: [d_{in}, 800, 500, 300, d_{out}] (hidden activations: ReLu and output activation softmax) again, but this time we will use the optimal values of the parameters found in the previous parts. Concretely, use number of epochs = 100, batch size = 1000, learning rate = 10^{-5}, Nesterov = True and Early Stopping = True. For regularization coefficient, decay and momentum coefficient use the best values that you found in the last few parts. Report your test set accuracy again. Is it better or worse than the accuracies you observed in the last few parts?

(l) **Grid search with cross-validation**: (15 Points) This time we will do a full fledged search for the best architecture and parameter combinations. Train networks with architectures [d_{in}, 50, d_{out}], [d_{in}, 500, d_{out}], [d_{in}, 500, 300, d_{out}], [d_{in}, 800, 500, 300, d_{out}], [d_{in}, 800, 800, 500, 300, d_{out}]; hidden activations ReLu and final activation softmax. For each network use the following parameter values: number of epochs = 100, batch size = 1000, learning rate = 10^{-5}, Nesterov = True, Early Stopping = True, Momentum coefficient = 0.99 (this is mostly independent of other values, so we can directly use it without including it in the hyperparameter search). For the other parameters search the full lists: for regularization coefficients = $[10^{-7}, 5 \times 10^{-7}, 10^{-6}, 5 \times 10^{-6}, 10^{-5}]$, and for decays = $[10^{-5}, 5 \times 10^{-5}, 10^{-4}]$. Report the best parameter values, architecture and the best test set accuracy obtained.

Hints and tips:

- You can use FTP clients like FileZilla for transferring code and data to and fro from the virtual machine on the Google Cloud.
- Always use a `screen` session on the virtual machine to run your code. That way you can logout of the VM without having to terminate your code. A new screen session can be started on the console by: `screen -S <session_name>`. An existing attached screen can be detached by pressing `ctrl+a` followed by `d`. You can attach to a previously launched screen session by typing: `screen -r <session_name>`. Checkout the basic screen tutorial for more commands.
- Don't use the full dataset initially. Use a small sample of it to write and debug your code on your personal machine. Then transfer the code and the dataset to the virtual machine, and run the code with the full dataset on the cloud.
- While running your code, monitor your CPU usage using the `top` command on another instance of terminal. Make sure that if you asked for 24 CPUs, your usage for the python process is showing up to be around 2400% and not 100%. If `top` consistently shows 100%, then you need to setup your numpy, scipy and theano to use multiple cores. Theano (and Keras) by default make use of all cores, but numpy and scipy will not. Since raw numpy and scipy computations will only form a very small part of your program, you can ignore their parallelization for the most part, but if you so require you can google how to use multiple cores with numpy and set it up.
- Setting the `verbose` flag for `fit()` and `evaluate()` methods in Keras as `1` gives you detailed information while training. You can tweak this by passing `verbose=1` in the `testmodels()` method in the `hw_utils.py` file.
- Remember to save your results to your local machine and turn off your machine after you have finished the homework to avoid getting charged unnecessarily.

Submission Instructions: You need to submit a soft copy and a hard copy of your solutions.

- All solutions must be typed into a `pdf` report (named `CSCI567_hw4_fall16.pdf`). If you choose handwriting instead of typing, you will get 40% points deducted.
- For code, the only acceptable language is Python2.7. You MUST include a main script called `CSCI567_hw4_fall16.py` in the root directory of your code folder. After running this main script, your program should be able to generate all the results needed for the programming assignment, either as plots or console outputs. You can have multiple files (i.e your sub-functions), however, once we execute the main file in your code folder, your program should execute correctly.
- The soft copy should be a single `zip` file named `[lastname]_[firstname]_hw4_fall16.zip`. It should contain your `pdf` report (named `CSCI567_hw4_fall16.pdf`) having answers to all the problems, and the folder containing all your code. It must be submitted via Blackboard by **11:59pm** of the deadline date.
- The hard copy should be a printout of the report `CSCI567_hw4_fall16.pdf` and must be submitted to locker #19 at PHE building 1st floor by **5:00pm** of the deadline date.

Collaboration You may collaborate. However, collaboration has to be limited to discussion only and you need to write your own solutions and submit separately. You also need to list the names of people with whom you have discussed.