Machine Learning *ENGG*6500, Fall 2016*

Assignment 1

Due Date: 9:40am Thursday, October 13, 2016
(Assignments should be submitted as PDFs to CourseLink Dropbox)

This assignment counts for 10% of the final grade.
There are a total of 50 points in this assignment.

Deliverables

Each student should submit the required materials to Dropbox before the start of the lecture on the due date. Late reports will not be accepted.

Reports should not exceed six (6) pages, including figures and tables. Material in excess of six pages will not be marked.

Technical Presentation Requirements

Include a cover page indicating your name, title of work, course, instructor's name and date. Assignments will be judged on the basis of visual appearance, the grammatical correctness and quality of writing, and the visual appearance and readability of any graphics, as well as their contents. Please make sure that the text of your report is well-structured, using paragraphs, full sentences, and other features of a well-written presentation. Use itemized lists of points where appropriate. Text font size should be between 10 and 12 point.

1 Theory

1. This question asks you to derive the gradient with respect to a softmax output unit:

$$y_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}.$$

We want to compute $\frac{\partial y_i}{\partial z_j}$. Note that a given y_i depends on $z_j \forall j$, not just z_i. Therefore, it's easier to split the analysis into two parts: when the indices match, and when the indices don't match.

(a) (4 points) Write $\frac{\partial y_i}{\partial z_j}$ for the case when $j = i$.

(b) (4 points) Write $\frac{\partial y_i}{\partial z_j}$ for the case when $j \neq i$.

(c) (2 points) Derive a more general expression which captures both of the above cases (Hint: use an indicator function).

2. This question asks you to reproduce Figures 5.2 and 5.5 in Goodfellow et al. Of course, you do not know their exact quadratic function that was used to generate the training points, nor do you know the exact points that were sampled. Try to select a function that appears close to the one used in the plots, and use the procedure they describe to generate a few reasonably spaced training points.

   (a) (5 points) Fit a linear, quadratic, and polynomial function to the points to approximately reproduce Figure 5.2.

   (b) (5 points) Incorporate three very large, medium, and nearly zero weight decay into your model to approximately reproduce Figure 5.5.

   Include both the replicated figures and the code you used to produce them in your report.

2 Practice

In this section, you will be implementing the backpropagation algorithm for a neural network with a single hidden layer of logistic units. You will then use your implementation to train a model with data from the 'StumbleUpon Evergreen Classification Challenge' at `kaggle.com`.

Download the code and data file (`A1.py` and `evergreen.pkl`) and ensure that the script runs on your machine (Hint: use the command `%run A1.py` within an IPython shell to execute the script). Read through the code to familiarize yourself with the conventions used. The model is represented by four `numpy.ndarrays`, each containing values for the model weights. If we define d as the number of input dimensions and h as the number of hidden units:

- `W_hid`: a $d \times h$ matrix where each column represents the incoming weights of a single hidden unit.
- `b_hid`: an h dimensional vector representing the biases for each of the hidden units.
- `W_out`: an h dimensional vector representing the weights from each hidden unit to the single output unit.
- `b_out`: an *array scalar* representing the bias for the output unit.

These weights are provided to the `train_model` method, along with the `numpy` arrays of the training and test data, the learning rate, and the number of training epochs. In addition, the `train_model` method also takes an optional `mlp_grads` parameter. This parameter expects a function which computes the derivatives of the error with respect to each of the weights. If not provided, `train_model` uses the auto-differentiation library `autograd` to compute these gradients[1].

1. (20 points) Fill in the method `my_grads` with your own implementation of the backpropagation algorithm. The method takes the training data, target labels, and model weights `W_hid`, `b_hid`, `W_out`, and `b_out`. It should return four `numpy` arrays of the same dimension as each

[1] We have provided an alternative version of the code `A1_tf.py`, that uses TensorFlow to auto-differentiate.

of the weight arrays. These arrays will contain the derivative values of the model error (in our case, the cross entropy) with respect to each of the weight dimensions.

You can check your implementation against the `autograd` gradients with the provided `check_gradients` method. Note that the `autograd` gradient computes the mean gradient over all training cases, not the sum[2]. In order to check your implementation with `check_gradients`, your `my_grads` implementation should scale your gradients by the number of examples in the training set. Include in your report your method `my_grads` as well as the output generated by `check_gradients`.

2. (5 points) The `train_model` method returns the cross entropy error computed on the training and test sets at each training epoch. Using 100 hidden units, train the model for 250 epochs at various settings of the learning rate. For each training run, plot the cross entropy on the training and test sets against the training epoch. How do the cross entropy plots change with different values of learning rate? How would you chose the *best* value for this parameter?

3. (5 points) Using the best setting for the learning rate that you found above, train a number of models while varying the number of hidden units and training epochs. Report your results. What criterion would you use to decide on 'good' settings of these parameters? Do the results change if the model is trained multiple times with the same "hyper-parameter" settings? Why or why not?

4. *[OPTIONAL]* Experiment with different activation functions. For example, how does training with `tanh` units in the hidden layer effect convergence time or accuracy? What about rectified linear units (RELUs)? Describe your experiments and report your results.

[2] The mean and sum differ by a constant factor which can be absorbed into the learning rate.