Assignment 3: Logistic Regression and Neural Networks

For due date see `https://courses.cs.sfu.ca`

This assignment is to be done individually.

Important Note: The university policy on academic dishonesty (cheating) will be taken very seriously in this course. You may not provide or use any solution, in whole or in part, to or by another student.

You are encouraged to discuss the concepts involved in the questions with other students. If you are in doubt as to what constitutes acceptable discussion, please ask! Further, please take advantage of office hours offered by the instructor and the TA if you are having difficulties with this assignment.

DO NOT:

- Give/receive code or proofs to/from other students
- Use Google to find solutions for assignment

DO:

- Meet with other students to discuss assignment (it is best not to take any notes during such meetings, and to re-work assignment on your own)
- Use online resources (e.g. Wikipedia) to understand the concepts needed to solve the assignment.

Theory: Gradient Descent Update for Logistic Regression Using Cross-Entropy (10 Marks)

For a probabilistic classifier, it is natural and effective to take the negative class label log-likelihood as the empirical loss function, which is known as the **cross-entropy** loss. The cross-entropy is defined as follows. Let $p_{\boldsymbol{w}}(y = 1|\boldsymbol{x})$ be the probability that the class label $y = 1$ given input features $\boldsymbol{x}$, where $y = 0, 1$. The cross-entropy of data $(\boldsymbol{x}_j, y_j)$ is then defined as

$$Loss(\boldsymbol{w}) \equiv -\frac{1}{N}\sum_{j=1}^{N} y_j \times ln(p_{\boldsymbol{w}}(y_j = 1|\boldsymbol{x}_j)) + (1 - y_j) \times ln(1 - p_{\boldsymbol{w}}(y_j = 1|\boldsymbol{x}_j))$$

Notice that either y_j or $1 - y_j$ is 0. For logistic regression, the conditional probability is $p_{\boldsymbol{w}}(y_j = 1|\boldsymbol{x}) \equiv \frac{1}{1+e^{-\boldsymbol{w}\cdot\boldsymbol{x}}}$.

1. Write down the cross-entropy loss for logistic regression.

2. Prove that for logistic regression, the cross-entropy gradient is

$$\nabla Loss(\boldsymbol{w}) = \frac{1}{N}\sum_{j=1}^{N}(p_{\boldsymbol{w}}(y_j = 1|\boldsymbol{x}_n) - y_j)\boldsymbol{x}_n$$

   Hint: recall that $\frac{d\sigma(z)}{dz} = \sigma(z)\sigma(1-z)$ where $\sigma = \frac{1}{1+e^{-z}}$ is the sigmoid function.

3. Following the notation in the text, write down the stochastic gradient descent update formula for minimizing the cross-entropy with logistic regression.

Practice: Implementing Logistic Regression (60 Marks)

Use the data set as before, with the modifications indicated on-line. Using *GP > 0* as the target class variable and drop the *sum_7yr_GP* column. We will go through a few typical steps for a regression analysis: data preprocessing, weight learning, and model evaluation.

Data Preprocessing

This is very similar to what we did in Assignment 2. For completeness and clarity, I repeat some instructions.

Convert discrete to continuous The hockey data form a *hybrid* data set, meaning that it mixes discrete with continuous variables. Regression is a technique for continuous input variables. A common approach to using regression with discrete variables is to first convert all discrete variables to binary (Boolean) variables, then use 1 and 0 to represent variables. Boolean variables that are treated as continuous regressors are called "dummy" variables or indicator variables. So the first step is to replace all discrete variables by dummy variables. For example, instead of having a single variable *Country_Group* with three possible values *Canada, USA, Europe*, you should introduce three binary variables *Country_Group = Canada, Country_Group = USA, Country_Group = Europe*. Change the data matrix so all values for the dummy variables are 1 or 0, depending on where the player is from.

Standardize Input Variables As discussed in class, in a multivariate regression it is often a good idea to standardize continuous variables to the same scale. In this assignment, we use gradient descent with a single step size, so the input variables should be on a single scale.

1. Subtract the mean of the column from each entry (**centering**).
2. Divide each entry by the standard deviation of the column.

Standardize the columns for continuous variables as described (leave alone the discrete columns with dummy variables).

Implement Stochastic Gradient Descent

To reduce your workload, the code we have posted on the website does almost everything you need to implement. If you prefer, you can also build your own implementation and use the posted code for guidance only. Given that you can build on your solution from Assignment 2, this should not be too much work. Here are some specific components we will check.

1. In the posted code, fill in the part that performs a single gradient descent update, using your solution formula from the previous question. If you submit your own code, please highlight the part that performs the single gradient descent update.

2. Adjust the provided code (or build your own) to perform stochastic gradient descent with the hockey dataset. (E.g., you need to fix some of the dimensions that are hard-coded into the code. And you may want to change some of the notation). For the initial weight vector, set the bias weight $w_0 := 0.1$ and all others to 0.

3. Write code that takes as input a weight vector $\boldsymbol{w}$ and a dataset and outputs the following:

   (a) The logistic regression cross-entropy (negative log-likelihood). (This is already implemented in the sample code.)

   (b) The classification accuracy of $\boldsymbol{w}$: Label an example $\boldsymbol{x}$ as positive if the predicted probability $p_{\boldsymbol{w}}(y = 1|\boldsymbol{x}) > 0.5$, and as negative otherwise. Then output the percentage of correctly labelled examples.

Experiments

Congratulations, you have a working implementation that learns weights for logistic regression. Now we can try things out. Try three different step sizes (aka learning rates), $\alpha = 0.5, 0.3, 0.1, 0.05, 0.01$, each for 300 training epochs, i.e. 300 passes through the entire dataset (batches).

1. For each learning rate, plot a learning curve that shows the cross-entropy after each epoch. So the x-axis is the epoch number and the y-axis is the cross-entropy (negative log-likelihood) for that epoch. Put the learning curve for all rates into a single plot and compare the results. What are the advantages and disadvantages of different learning rates? Figure 18.25 in the text shows you an example of a similar plot for a single learning rate.

2. From your plot, find the learning rate that converges to the weight vector $\boldsymbol{w}$ with the best fit to the training data (lowest cross-entropy). What is the accuracy of $\boldsymbol{w}$ on the test data? Using your results from previous assignments, compare the classification accuracies of Naive Bayes, decision tree ID3 and logistic regression.

Here are some **optional** experiments that are instructive, but we won't grade you on them.

1. Try gradient descent rather than stochastic gradient descent. Which is faster?
2. Try using the original features rather than the standardized ones.
3. Try adding interaction terms as in the previous assignment.

Exercise: Trace Backpropagation (20 Marks)

Consider a neural net with one hidden layer, two inputs a and b, one hidden unit c, and one output unit d. The activation function is the sigmoid function for each node. This network has five weights $(w_{ac}, w_{bc}, w_{0c}, w_{cd}, w_{0d})$, where w_{0x} represents the bias or threshold weight for unit x. Initialize these weights to the values (.1,.1,.1,.1), then give their values after each of the first two training epochs of Backpropagation algorithm. Assuming learning rate (step size) of 0.3, stochastic gradient descent, and the following training examples:

Table 1: Data Table for Backpropagation Trace

	a	b	d
$\boldsymbol{x}_1$	1	0	1
$\boldsymbol{x}_2$	0	1	1

Fill in the following tables. The tables use the notation from the slides. You can expand these to include more information (e.g. derivatives of activation functions) if you like.

Table 2: Backpropagation Trace

Iteration	a_c	δ_c	a_d	δ_d
$\boldsymbol{x}_1$				
$\boldsymbol{x}_2$				

Updated Weights	w_{0c}	w_{ac}	w_{bc}	w_{cd}	w_{0d}
$\boldsymbol{x}_1$					
$\boldsymbol{x}_2$					

Vanishing Gradients (10 Marks)

Consider a simple linear neural net with the structure *input* $\rightarrow$ *node1* $\rightarrow$ *node2* $\rightarrow$ *output*, as shown in Figure 1. There are 6 weight parameters; assume they have all been initialized to 0.1. Apply backpropagation to the training data $(1, 1)$, input =1 is mapped to output = 1.

- Assume a *sigmoid* activation function for each node (other than the input). Write down the gradient for each weight parameter for the training data $(1, 1)$.
- Assume a *rectified linear unit* activation function for each node (other than the input). Write down the gradient for each weight parameter for the training data $(1, 1)$.

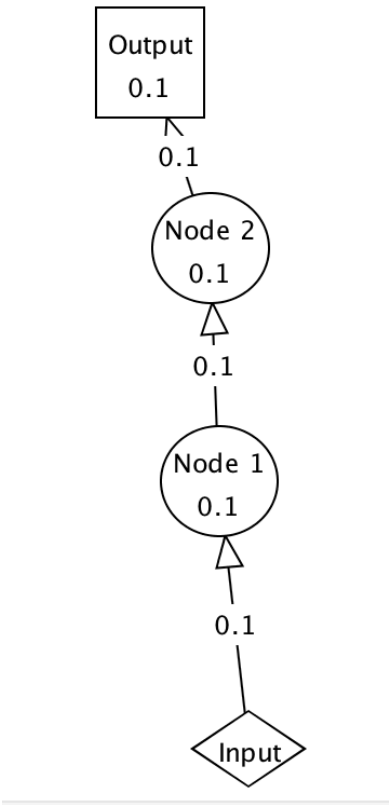

Figure 1: A simple linear neural network

Which activation function leads to higher gradients? What if we extended the network with additional layers (intermediate nodes in the line). Would the difference in gradients between sigmoid and ReLU activations become greater or smaller?

Deep Learning (80 Marks)

Install a deep learning package. I have listed a number on the course website. Most students have liked Keras and Caffee (which interfaces with Torch). Try to use your package to predict the number of games an NHL player will have played after 7 years. (The same problem we tackled before with linear regression). For data preprocessing, drop the $GP > 0$ column and use sum_7yr_GP as the target column. Try to find deep learning settings for training on the test set such that the trained model gives the best prediction on the test set. (In terms of the textbook discussion, we are using the test set as a validation set for hyperparameters.) There will be a prize for the student(s) who get the lowest squared error on the test set!

You should use a linear activation function for the output node. (Why not sigmoid?) There are some settings you must try and some that I suggest you try.

Things you must try

1. Try using rectified linear units for hidden nodes.
2. Try learning with 0,1,2 hidden layers.

Things you can try

1. Try sigmoid units for hidden units.
2. Try learning with more than 2 layers.
3. Experiment with different numbers of nodes per layer.
4. Try different learning rates. (Initial learning rates if you use an option that adjusts learning rates during learning, like ADAM.)
5. Try different learning options, e.g. Batch Normalization and Dropout.

Report the 2 or 3 best options you have tried. Discuss which factors lead to the most improvement. (E.g. is the number of hidden nodes per layer more important than the number of layers?) It's okay just to turn some knobs but it's even better if you can provide motivation for why you think some options will work better than others.

Submitting Your Assignment

You should create a report with the answers to questions and figures described above in PDF format. Make sure it is clear what is shown in each figure. We would also like you to submit your source code.

Submit your assignment using the online assignment submission server at: `https://courses.cs.sfu.ca`.