Assignment 3 "Classification" for ML4BIO 2016 (there will be no assignment 2)

Due Date: *May 7th, 2016*

Submit to ML4BIO@gmail.com before 11:59pm on May 7th (Eastern Daylight Time)

Classification

You will be using http://playground.tensorflow.org to answer questions for this assignment. When you need to show your work, please submit screen captures of the interface. Feel free to keep your response as short as possible but no shorter than necessary to explain your answer. Playground is non-deterministic, so please ensure that the behaviour that you report is at least qualitatively reproducible.

The playground uses Stochastic Gradient Descent (or SGD to its friends) as the optimization method to train the neural network. This method is related to the gradient (or steepest) descent optimization method that we talked about in class. In gradient descent and SGD, you initialize the parameters to some random values and iteratively improve them in an attempt to minimize the cost function (Because the goal here is to minimize the objective function, we can it a cost function). There is one parameter for each connection in the neural network, these parameters are called weights. The current settings of the parameters are represented by a vector and you update the parameters in each iteration by adding another vector, the *update vector*, that has a small magnitude (i.e. length). Intuitively, what you are doing is taking a small step in the direction of the update vector – the size of the step is the length/magnitude of the update vector. The *learning rate* is proportional to the size of the step

In gradient descent, this update vector points in the direction opposite to the gradient of the cost function (i.e. if the gradient is a vector $\boldsymbol{v}$ then the update vector points in the direction of $-\boldsymbol{v}$). Thus the name *gradient descent*. Why is this a good idea? Well, a small step in the direction opposite the gradient will give you the largest decrease of the objective function of any direction you can choose (for the same sized step). That's why gradient descent is sometimes called *steepest descent*.

SGD is just like gradient descent, except that instead of computing the gradient using all the training set, you compute it with a random subset of the training set. This is why it is called *stochastic*. The *batch size* is the size of this subset. If the batch is big enough, then the stochastic gradient will point in nearly the same direction as the gradient computed using the whole training set. Because the amount of time it takes to compute the gradient scales linearly in the size of the batch, training is much faster if you can use smaller batches but get more or less the same gradient. Also, surprisingly, it turns out that using a stochastic gradient sometimes gives better solutions than the actual gradient, because it can prevent the optimization method from getting stuck in bad local minima.

Choosing a good learning rate is important because sometimes the gradient can vary a lot for nearby points. So, if you take too big of a step, you can sometimes increase

the cost function rather than decreasing it. But if you take too small of a step, training will take forever.

The four parts of ML

In class, we talked about four parts of an ML method: the Data, the Model, the Objective Function, and the Optimization Method.

1. For each of the four parts, indicate two choices that the playground gives you that affect that part. Bonus question: can you find one more choice for each of the four parts? Note, I think that one of the parts only is associated with two choices.

Logistic regression

In the classification setting, if you choose zero hidden layers, playground will train a logistic regression (LR) classifier using SGD.

1. What datasets are linearly separable using LR with only X1 and X2 inputs?

2. What datasets are separable using some subset of the inputs? In these cases, what is minimum set of inputs that make the datasets separable and what is the function of the decision boundary? Just write down the non-zero terms and their sign.

3. What is the impact of the learning rate on learning?

4. What is the impact of using L1 regularization? How about L2? How does it depend on the regularization rate?

Neural networks

The multiple layer perceptron that we talked about in class corresponds to the classification setting, with a single hidden layer.

5. What datasets can you achieve good classifier for using MLP with only X1 and X2 as input? Show screen captures of the datasets you were able to learn. How does using the other inputs help this?

6. What is the impact of the learning rate on learning? What happens if the learning rate is too high (esp. with small batches)? What happens if it is too low? How does this depend on the number of hidden layers in the network, and their size? Does regularization help? Just give one or two examples for these questions.

7. Can you find any examples of overfitting? (Hint: try playing around with the amount of noise you add to the data) When, qualitatively, does this happen?

8. If you use all the inputs, all problems are learnable for some settings of the model. Can you give three examples of input choices that lead to one or more unlearnable classification problems? Why are these bad choices for the inputs?