**Exercises for 9/29 (heavily based on HOML Ch. 4 exercises)**

*Submission instructions:*

* *Make a copy of this document and rename it with your FIRSTNAME.LASTNAME.04*
* *Share it with* [*denys.katerenchuk@gmail.com*](mailto:denys.katerenchuk@gmail.com) *and* [*rebeccalevitan@share.brooklyn.edu*](mailto:rebeccalevitan@share.brooklyn.edu)
* *Submission closes on* ***Friday 6pm*** *(contact me if you need religious accommodation)*

1. Which Linear Regression training algorithm can you use if you have a training set with millions of features?

Stochastic Gradient Descent

1. Suppose the features in your training set have very different scales. Which algorithms might suffer from this, and how? What can you do about it?

Gradient Descent (batch, stochastic, & mini-batch) will take a long time to find the feature weights with minimum cost because the different scales will elongate the “bowl”. You can scale the data before training (with StandardScaler for example).

1. Can Gradient Descent get stuck in a local minimum when training a Logistic Regression model?

No, because there’s only a global minimum, no local ones.

1. Do all Gradient Descent algorithms lead to the same model, provided you let them run long enough?

Stochastic GD and Mini-Batch GD bounce around the global minimum (in convex cost functions), so unless you make sure to deal with learning rate issues they will be ever so slightly different. Also, you can’t make batch [Full set] GD converge properly if you set the learning rate too high.

1. Suppose you use Batch Gradient Descent and notice that the validation error consistently goes up at every epoch. What is likely going on? How can you fix this?

You’re either past the ideal training point in a program that will overfit with enough training data (in which case, use early stopping to end the training before it goes through the whole set), or your learning rate is too high and the algorithm is changing the feature weights way too much each time. In that case simply reduce the learning rate (you can use a grid search for this).

1. Is it a good idea to stop Mini-batch Gradient Descent immediately when the validation error goes up?

Not necessarily, because it’s picking its mini-batches randomly from the set. However, on average it should be lowering the cost function; if it gets worse for a few batches in a row you should stop and roll-back to the hyperparameters you had at the best score.

1. Which GD algorithm of the ones we discussed will reach the vicinity of the optimal solution the fastest? Which will actually converge? How can you make the others converge as well?

Stochastic GD is usually fastest, but will bounce around the optimal solution. Batch GD will converge with enough time and a low enough learning rate. You can modify Mini-Batch and Stochastic GD to include a decreasing learning schedule over time that will eventually converge as well.

1. Suppose you are using Polynomial Regression. You plot the learning curves and you notice there is a large gap between the training error and the validation error. What is happening? What are three ways to solve this?

A large gap there indicates overfitting; it trained too well and tried very hard to fit the training data that it can’t generalize as well as it should. You can:

1. Lower the degrees of freedom
2. Penalize the algorithm for large feature weights
3. Give it more training data (so that it casts a wider net when trying to fit the data)
4. Suppose you are using Ridge Regression and you notice that the training error and the validation error are almost equal and fairly high. Would you say that the model suffers from high bias or high variance? Should you increase the regularization hyperparameter *α* or reduce it?

High bias; the model is underfitting, which prevents it from giving consistent accurate predictions. To deal with underfitting you should reduce the regularization (for *α,* lower value = lower regularization)

1. Why would you want to use:
   1. Ridge Regression instead of plain Linear Regression (without any regularization)?

Ridge regression will help prevent overfitting since it penalizes very large feature weights (usually turns out better able to generalize).

* 1. Lasso instead of Ridge Regression?

Lasso helps eliminate insignificant features (by setting their feature weights to 0).

* 1. Elastic Net instead of Lasso?

Elastic Net provides a nice balance between the two. Lasso goes a little crazy sometimes [oversimplifying the model], and ridge may force feature weights below their real-world value. Also, you can change the r value to choose how much of each you want, so you get the best of both worlds with a little more work.