**Final Project**

Github:

Part 1. 8x3x8 encoder (see: )

Part 2. Learning procedure for an ANN.

* Encode each nucleotide as binary (1 nt = 2 bits. A = 00, T = 01, C = 10, G = 11).
* Each DNA sequence (17 bp) is an entry, fed in as an array of shape (17,1).
* Input layers, nodes = 1, 17
  + (Since each nt is an independent variable, it is a feature.)
* Hidden layers, nodes = 1, 3
  + (I have ~220 training examples\*. To follow the rule of 10X more examples than weights I start with ~2 hidden nodes. To keep the number of hidden nodes < # of input nodes and > # output nodes, I should have between 2-17. These are just rules of thumb, however.)
* Output layers, nodes = 1, 2
  + (The probability of being a site and of not being a site. This could also be represented as a single node where 0<x<0.5 is not a site and 0.5<x<1 is a site.)

Part 3. Training regime.

* Format the negative data in the same manner as the positive data (text file with 17 nt per line). Before encoding, remove all examples that match the positive data.
  + translate nt to digits
  + turn each line into a 1D array
  + import to NN as a matrix of (17, len(file))
* Use an 80/10/10 scheme to split the data (80% as training data, 10% as testing data, and 10% as validation data)
  + Since there are 137 true positives, I can use ~109 as positive training examples. \*To not overweight the negative data, I would use the same number of negative training examples, thus giving me a pool of 218 training examples.
  + The stop criterion is when the change in the gradient is 0 over >1 iterations because that means the weights have collectively reached a (global) minima.

Part 4. Cross validation experiments.