1. nAccomplished
2. Mistakes
3. New knowledge

First -- go through each file and compare with an answer key, add corrections and keep track of them here.

Assignment 1

* Training KNN classifier
  + Quick summary: Take every training image, calculate distance (using l2 or l1) between the test image and every training image. Take the k closest images, and use their classes to predict the class of the test image.
  + (super slow) pixel by pixel with nested for loop, find knn with np.argsort
  + Cross validation -- hardest part was taking out the test fold and then concatenating the remaining folds together. Use np.delete and np.concatenate
    - The way they do it straight from the data set is to create a ‘mask’ that stores the indices of the fold that we want in a range, and then uses normal array selection to get the right points out
    - Ex. mask = list(range(500)); sample = X[mask]
  + np.bincount creates bins equal to one more than the maximum value in given array, and then returns the count of the number corresponding to the indexes -- used to count the number of occurrences of each class in the k nearest neighbors
  + Most clever part was calculating the l2 distances without using any loops. What we did was expand the squared term (x2 - x1) \*\*2 into x2\*\*2 - 2x1x2 + x1\*\*2 and then manipulated the matrix multiplications so we got each of those terms separately, then added them all together
* Training Support Vector Machine
  + Quick summary: SVM classifier works by using multiclass SVM for the loss, then using SGD to train the weights. Predictions are made by taking the argmax of scores once the weight matrix has been trained. We use the bias trick on the weight matrix (i.e. adding a column of 1s to X and b as the last row of W). (What is multiclass SVM?) Uses a ‘fencing’ idea to ensure that the correct class score is far enough away from the other class scores. We take max(0, s\_j - s\_y\_i + delta) as our margin. If the j\_th score is far enough away, it does not contribute to the loss.
  + Funnily enough, the mistake that I made the most in general was forgetting to divide by N (the number of training examples) after calculating the loss and gradient (before adding the regularization terms). Why do we divide by N?
    - The The full loss is L = (1/N) \* sum(L\_i) + lambda \* R(W)
    - Average over training data for gradient because each element in the gradient consists of the desires of all the training examples to shift the weights, so we use the ‘average desire’
    - This means that dL has a 1/N term in there (d(ca) = c)
    - Will carry over to the two layer neural network, where we must divide everything by N again because all the gradients are of L, with respect to each weight and bias.
  + Using -1 as index to np.reshape -- this lets numpy figure out what dimension to put there based on whatever makes the most sense
  + Using np.arange to select values -- to select the y\_th values from each row, we can use A[np.arange(len(A)), y] -- this matches each element in the arange array with the corresponding element in the y vector -- numpy assumes that y is an index vector
* Training Softmax classifier
  + My vectorized implementation can be improved a ton -- just use the same dimension matching method that we use in backprop to calculate the matrix derivatives. Also recognize that you can just decrement the y\_ith values beforehand manually.
    - Probably would’ve lost a ton of points on that one
* Two-Layer Neural Network
  + Is possible to get away with less intermediate terms in the backprop, the rest of the neural network implementation looks ok
* Higher Level Representations
  + Using the features instead of the raw pixels led to about 5% bare minimum improvement with svm linear classifier and neural net. We used histogram of gradients and color histogram.

Major problems and lessons

* At the very end, thought the two layer net implementation was broken, turned out that I just did not tune the hyperparams properly. Turns out nets suck when the hyperparams aren’t tuned!
* Some vectorized implementations could have been better, basically try to get it out in one matrix multiplication
* The derivative of softmax is in terms of itself, became even simpler because of the (-log)
* Derivative of svm was indicators on the margin
* Neural net was matrix multiplication plus bias term, followed by activation function. Activation function tells you if the ‘neuron’ has a meaningful enough value to continue being considered in the forward pass.
* Basic steps in neural net: forward pass → loss (we used log softmax) → backprop to get gradients → update the weights → keep going until you hit num\_iterations
  + The point of the gradient is to tell you how to change the weights in order to lower the cost
  + We use Stochastic gradient descent, which samples batches from the data to calculate the gradient, batch size of about 200
* Definitely going to need to work on optimizing the training process -- what are you supposed to do while the network is training? Did not really know what to do this time, just watched youtube. We can be more productive. Probably good answer is reading.
* The person that I checked my implementations against (Aman Chadha, at Apple getting MS in AI from Stanford) had much more detailed documentation of his code than I did. This will likely be important, try much harder to include comments on all the technical details of your thought process. Although a lot of it did seem like busy work that wouldn’t have much return on learning.