* 7/28/18
  + Allow y to be a 1D array and automatically convert to a 2D array with 1 column
  + This might happen anyways with how numpy coerces types
  + There are many problems that need to be resolved:
  + Changing all arrays to 2D arrays
  + Making sure the activation function works correctly
  + Adding in the 1 offset neuron
  + Recursive activation now working
  + Temporarily removing the bias 1 term; will add in later once dimensionality problems are resolved
  + The standard will be for all arrays to be stored as n x 1 and then if necessary the transposes will be taken to make the dimensionality work out
  + The dimensions map up but the performance is highly varied and generally atrocious for a linearly separable problem...something isn't right
  + Maybe we need the bias 1 term for it to work?
  + Added the bias term but still not working - classification error is 50%
  + 7/29/18
  + Going to first look at the test part and see if anything looks unusual
  + All of the predictions come out to zero! All the neurons are dead!
  + The dimensions of the weights may be off
  + Modified the weight vector dimensions so it follows the slides; may not have actually made a difference though.....
  + Changed the predict method so that the predictions are stored in a fixd numpy array and not a list that gets converted
  + For whatever reason it's biased towards generating -1 predictions - the error isn't shrinking!
  + 8/11/18
  + Looked at the code briefly - attempted to multiply the weights by the predicted value to change direction of training but this had no effect
  + The error is growing for some test points which is not good....
  + Should add to github and also check what cases do need to be multiplied by the actual value
  + Should look at another description of backpropagation and see if it's consistent
  + Should also definitely push this to a lower dimension space where it's more visible what's going on
  + 8/13/18
  + Created a new test\_simple function that tests a one dimensional fixed data set
  + When subtracted weights, it works without a hidden layer! But two problems: why is subtraction not working, and why does the hidden layer ruin it?
  + There’s a weird two wrongs make a right scenario – if the weight is updated by addition AND it’s multiplied by the output then it works
  + There was an extra square that I didn’t reduce the power by when I took chain rule! That takes care of the direction problem and now it works without a hidden layer
  + Still a problem with the hidden layer, but likely because the indexing is wrong and one set of weights never gets updated
  + 8/21/18
  + Played around with indexing of backpropagation and looked at why the dimensions are mismatched
  + The final x\_l should have a dummy output 1 as well for consistent indexing
  + The current indexing is changing the shape of the delta in the last step, which is why the delta array for the last layer is misshapen and not aligning with the weight matrix
  + Changed the final x\_l layer so it’s no longer special; also has an extra 1 x\_l “input” to the next layer
  + This indexing treats all layers equally, and the output layer is just like any other layer
  + 8/22/18
  + Fixed the indexing so that it propagates and can run the predict method without generating an error
  + Still not predicting correctly with hidden layers – it always gives the same answer regardless of input
  + Answer is whatever was the last trained point
  + Unclear why…..
  + The weight for the bias term is far larger than the weight for the hidden term so that the bulk of the final output is the bias terms times the single weight….why is it not updating correctly then?
  + 8/23/18
  + As test, eliminated bias term and now the tests are much more successful!
  + With one hidden layer test\_simple is over 90% and test\_train is in the 80-90% range
  + Bias terms were all set to 0; will later set the x\_0 bias term to 1
  + Possibly rewrite code to get rid of all the zeros since we don’t have a bias term anymore
  + It seems to be working!
  + 9/3/18
  + Going to work on strela on the flight
  + Strela\_v1 moved to the archive – working version but with the additional unnecessary index
  + Going to go through and remove the extra index
  + First removing the extra index
  + Something seems off – there isn’t a delta for the bias term in the zeroth layer
    - Oh there doesn’t need to be one
    - All of the input terms are updated because the delta dimensions match to the outputs; the delta is applied to all input to output weights
  + Successfully updated and extra weights removed
  + Next to clean up comments and add some helper functions
  + Then more tests and some regularization
  + 9/4/18
  + Added some print statements to display info when initializing
  + Should write a function that visualizes how it’s making decisions in two dimensions to see if it’s actually classifying functions correctly
  + 9/8/18
  + Played around more with the plotter and the neural net is only fitting linear functions for some reason
  + Tried a parabolic function restricted in Q1; it seems to work fin, but it’s because it’s fitting a linear function
  + Could be that it’s operating in the regime where the activation function is linear?
  + Could try using ReLu…first going to set the test function to a circle since that’s wildly non-linear and only do this in Q1 for now
  + ReLu didn’t really help, and even then it shouldn’t make a difference
  + Should try doing a one dimensional toy example again and try and get it to learn
  + Adding more layers is NOT improving the in sample error meaning something is still wrong with having many layers
  + Possible that the third layer failing is simply vanishing gradient
  + More pressing concern is why predictions are linear when they definitely should NOT be
* 9/9/18
  + First let’s try printing out the x\_l to see if any of the nodes are obviously dead
  + If not then two things to try – do a one dimensional toy example and see why it won’t separate
  + Other is set the weights into what’s known to not be a linear function and see if it’s something wrong with the way predictions are made
  + The way classifications are done may not be correct; should maybe be using categorical cross entropy as error function
    - That may not explain the non-linearity of it
  + Changed so that during training multiple passes are made through data
  + Increased starting random weight values
* 9/10/18
  + Wrote function to set weights to something non linear and see if the neural net is not training into non-linear functions or if it’s not able to predict non linear functions
  + Did not test the new train square function – ran the train\_plot function and the classification started looking non-linear!
  + Ran more test – it is working! It is able to do non-linear classifications!
  + It’s the passes! Only one pass through the data doesn’t work – must do multiple passes to get non-linear fits
  + Set up neural net to massively overfit and it DOES fit correctly now with multiple passes!
  + Tested on a circle centered around 0 in all four quadrants – it works!
* 9/17/18
  + First clean up the code and add little things like progress bars
  + Then clean up test functions
  + Maybe read about regularization tonight and plan for multiple types of error like categorical cross entropy
  + Added some small features – starting to split off the error and delta calculation to be more flexible
* 9/28/18
  + The multiclass classification does not work, but it also doesn’t work for binary classifications
  + Suggests that problem is with all the stuff we added on top and not the neural net being unable to learn the data set
  + The multi-class classification setup (multiple output heads) works when squared loss is used and not categorical cross entropy
    - Suggests that we have the wrong derivative…..
  + Let’s see if multiclass classification works with three classes and squared loss
    - Also works! 90% test accuracy
  + Must be something wrong with implementation of categorical cross entropy
* 9/29/18
  + Wow the categorical cross entropy looks very wrong – it’s all 1.5 or 3!
  + Now the arrays are aligned and appears to be “working” (80% correctly classified) but the errors should all be positive
  + Or maybe not, because it’s the derivative
  + No it makes sense that it shouldn’t always be positive, otherwise it’d always train in the same direction
  + I think CCE is working now as well
  + Next we’ll implemented regularization and then move to final cleanup and demo code
  + Hard to demonstrate that regularization is working though…
  + Wrote regularization test function – looks like it’s working!