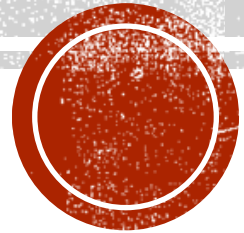
NEURAL NETWORK TRAINING PARAMETERS

Dr. Brian Mc Ginley



FULL IMPLEMENTATION OF TRAINING A 2-LAYER NN

- Can be done in 11 lines

```
01. X = np.array([ [0,0,1],[0,1,1],[1,0,1],[1,1,1] ])
02. y = np.array([[0,1,1,0]]).T
03. syn0 = 2*np.random.random((3,4)) - 1
04. syn1 = 2*np.random.random((4,1)) - 1
05. for j in xrange(60000):
06.     l1 = 1/(1+np.exp(-(np.dot(X,syn0))))
07.     l2 = 1/(1+np.exp(-(np.dot(l1,syn1))))
08.     l2_delta = (y - l2)*(l2*(1-l2))
09.     l1_delta = l2_delta.dot(syn1.T) * (l1 * (1-l1))
10.     syn1 += l1.T.dot(l2_delta)
11.     syn0 += X.T.dot(l1_delta)
```

from @iamtrask, http://iamtrask.github.io/2015/07/12/basic-python-network/

- The above uses a logistic regression loss
- We might need validation to choose the loss function, the activation functions, size of neural network etc.

NUMBER OF EPOCHS

- How many Epochs should we do when training our network.
- As usual it depends!
- Too many epochs takes longer, and it can also cause overfitting to the training data.
- It's another thing to consider.
- A technique called *Early Stopping* could be used.

EARLY STOPPING

- One strategy is called *Early Stopping*.
- As usual we can split training data and test data.
- Instead of doing a kfold cross validation (or similar), just hold back some of the training data as validation data.
  - kfold Cross-validation could take far too long
- So, we have 3 sets, training data, validation data and test data.
- Training data to train, Validation data to pick our parameters, test data for overall evaluation of our model.

EARLY STOPPING

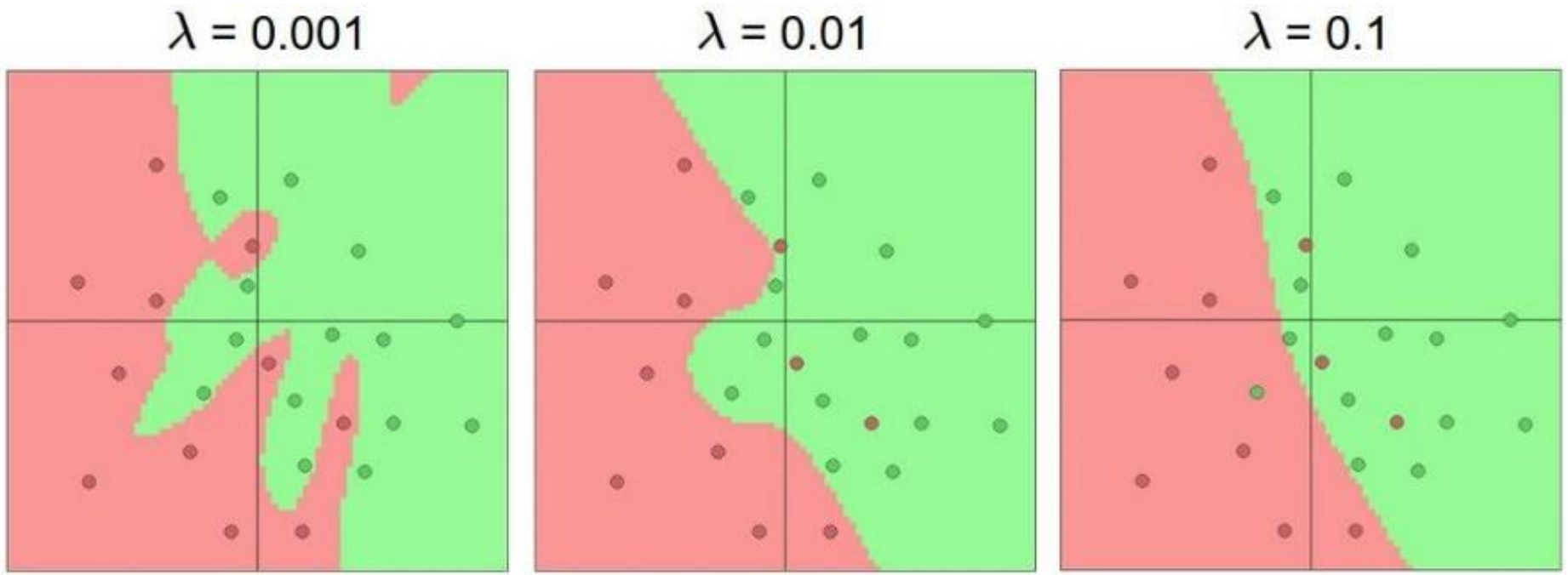
1. Train the model using the training data as usual
2. At the end of each epoch compute the classification accuracy on the validation data
3. Once the classification accuracy on the validation data has saturated, stop training
   - How you determine it has saturated is another question. Just because the accuracy has decreased from one epoch to the next doesn't mean it can't improve later. (accuracy often jumps around a bit)
   - Sometimes networks plateau near a particular accuracy score for a while and then start improving again
   - Some people do a "no-improvement in 10 epochs" rule. Or 20, or 100 . . ..

REGULARISATION

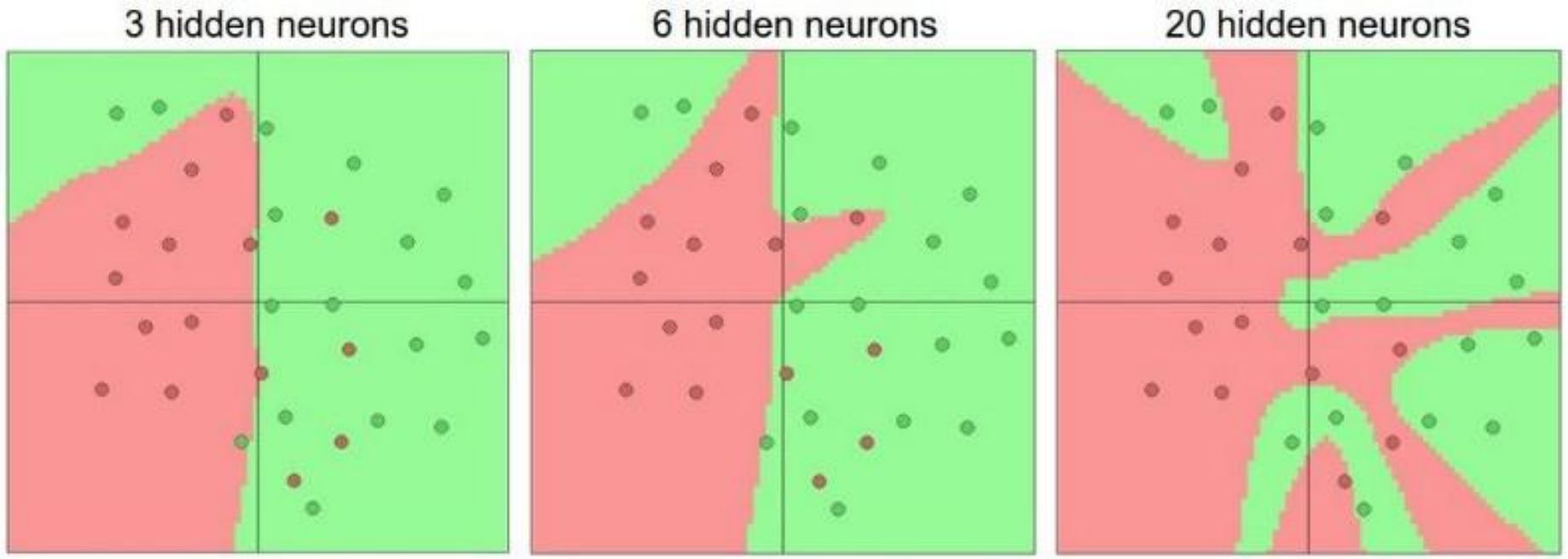
- Overfitting is a consideration with Neural Networks as it is in other ML techniques
- Increasing the amount of training data is one way of reducing overfitting. (Probably the best way)
- L2 Regularisation (sometimes known as weight decay) can be applied to Neural Networks
  - You have seen Regularisation mentioned before during the course.

REGULARISATION

- You can play with this demo over at ConvNetJS: http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html



NUMBER OF LAYERS/NEURONS



more neurons = more capacity

NUMBER OF LAYERS/NEURONS

- There are no definitive rules about it. If your dataset is relatively small and it doesn't take too long to train, then you can experiment with a different number of layers and neurons. So, it depends! Trial and error is a big part of machine learning which hopefully you've learnt by now.
- Couple of points:
  - If the training score is very low (underfit), then maybe your model is not complex enough.
  - Use the validation score to prevent overfitting.

NUMBER OF LAYERS/NEURONS

- Number of neurons (units):
  - Input units is obvious, it is the number of features.
  - Output units should be obvious too. One for regression or as many neurons as classes for classification.
- For Hidden Layers – from "An Introduction to Neural Networks for Java, Second Edition by Jeff Heaton"
  - There are many rule-of-thumb methods for determining the correct number of neurons to use in the hidden layers, such as the following:
    - The number of hidden neurons should be between the size of the input layer and the size of the output layer.
    - The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
    - The number of hidden neurons should be less than twice the size of the input layer.

NUMBER OF LAYERS/NEURONS

- The more units and layers the higher the complexity. Too many of these can result in overfitting.
- Many problems (we'll see Convolutional NNs soon which use a lot of layers) can be solved by only one layer. If that is complex enough, why not use that?
  - This is true about CNNs but deeper networks can train more efficiently (to hierarchically extract and combine features of increasing complexity)
- From the previously mentioned book:

Table 5.1: Determining the Number of Hidden Layers

Number of Hidden Layers	Result
none	Only capable of representing linear separable functions or decisions.
1	Can approximate any function that contains a continuous mapping from one finite space to another.
2	Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.

OPTIONS

- What are all the options now?
  - Activation Function
  - Depth of Network (number of hidden layers)
  - Number of Units in a Layer
  - Loss Function
    - MLPClassifier uses log-loss which is related to cross-entropy
  - Regularisation Parameter
  - Learning Rate
  - Number of Epochs
  - Whether to do early stopping
  - Even the ”Learning Algorithm”
    - SGD, lbfgs or Adam in MLPClassifier
    - Adam is an ”optimised” version of SGD - default
    - lbfgs can work better on small datasets
  - Batch size
  - Weight initialisation

SUMMARY

- We arrange neurons into fully-connected layers
- The abstraction of a layer has the nice property that it allows us to use efficient vectorized code (e.g. matrix multiplies)
- Neural networks: bigger = better (but might have to regularise more strongly)
  - The bigger the NN, the better it's able to model different functions

SOME CLOSING REMARKS

- Neural Networks/Deep Learning is the current "State of the Art" of Machine Learning. However, the amount of data needed to train a neural network can be a deal-breaker.
- If a right-sized neural network would need 10000 points to train it properly, it's useless if you have 1000 and no way to get more.
  - But maybe data augmentation/synthesis is possible?
- Thus Naive Bayes, kNN, SVM, Random Forests still have their place. They might not be the cutting edge, but are (in general) workable in simpler cases, when you have smaller training sets. Sometimes they may outperform Neural Networks.