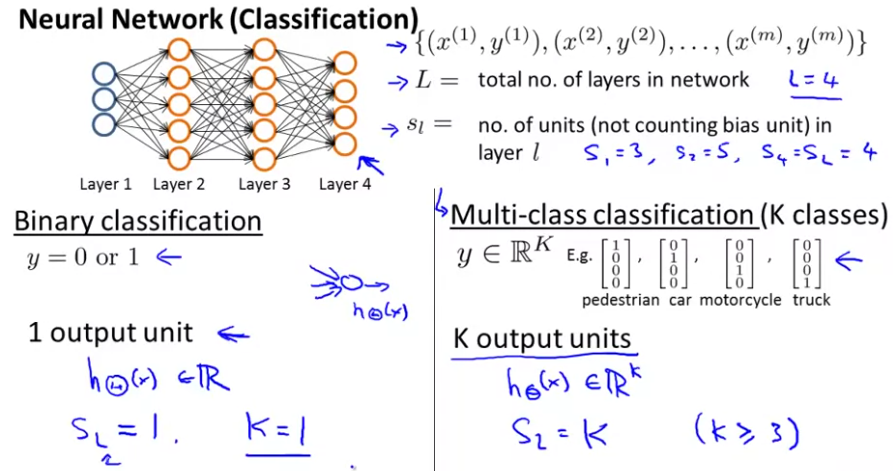
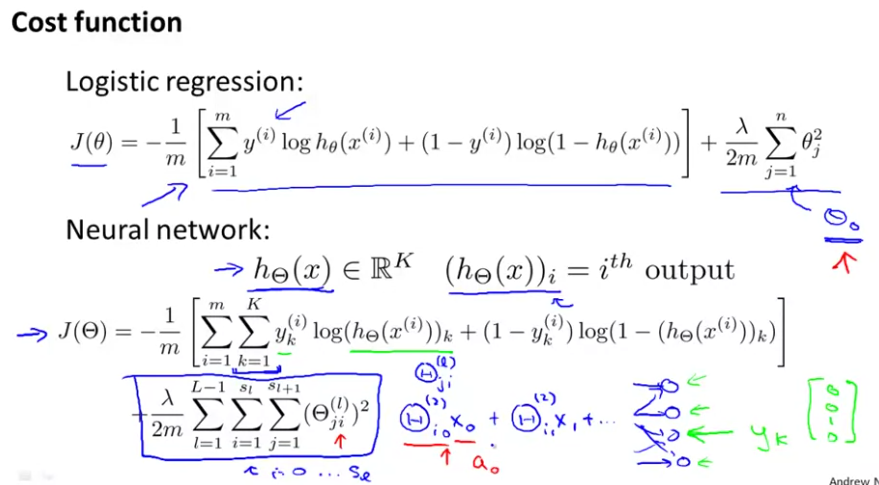
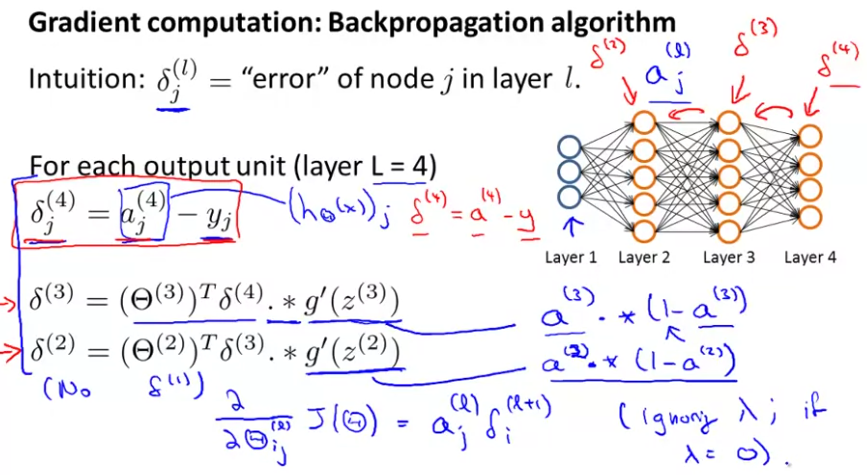
**Cost function**



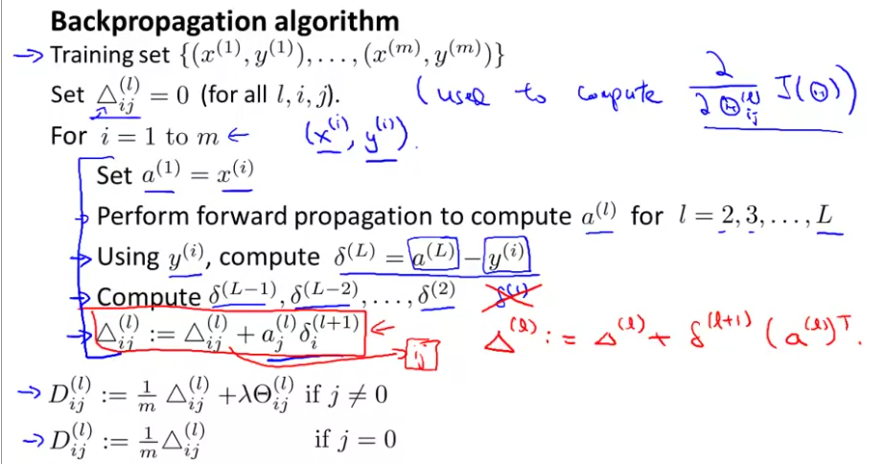


from the K classes

**Cost function minimisation – Backpropagation algorithm**



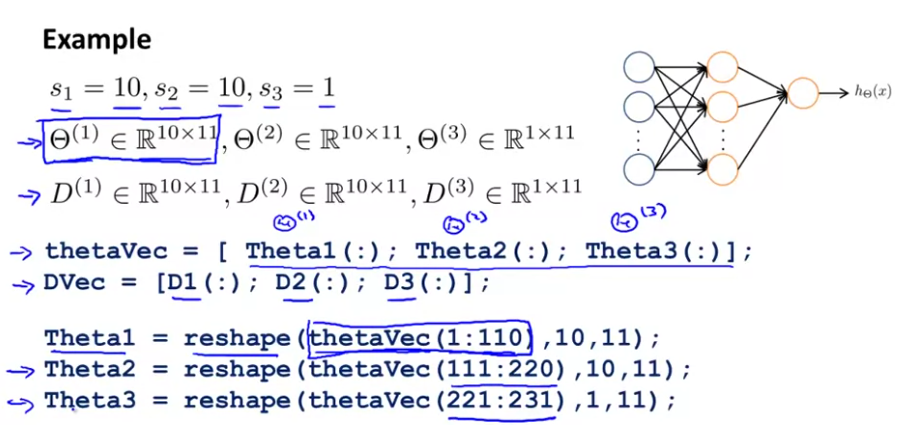
Summary of the backpropagation algorithm implementation:

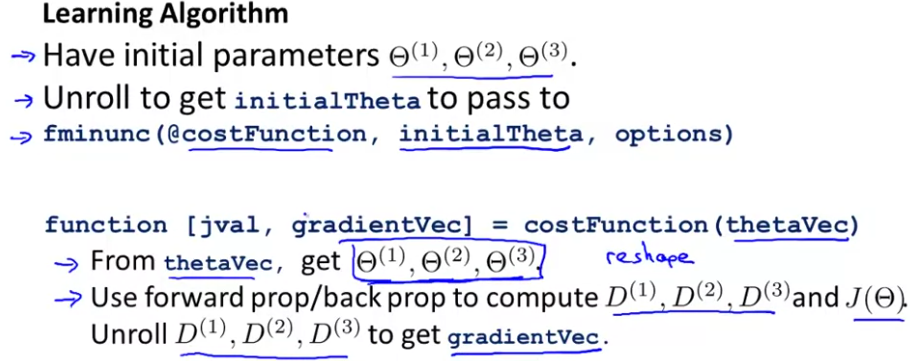


We compute the final gradient matrices after the for loop

**Backpropagation implementation – Unrolling parameters**

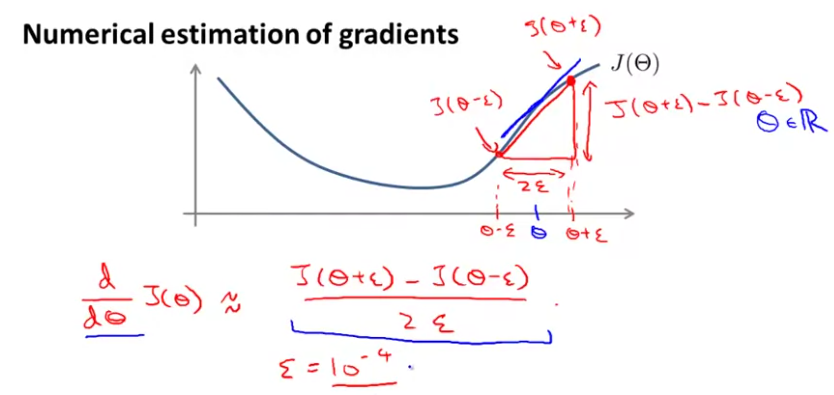
When we use the advanced optimisation algorithms, we may need to introduce only vectors as inputs. For converting the theta and gradient matrices into vectors and vice versa:





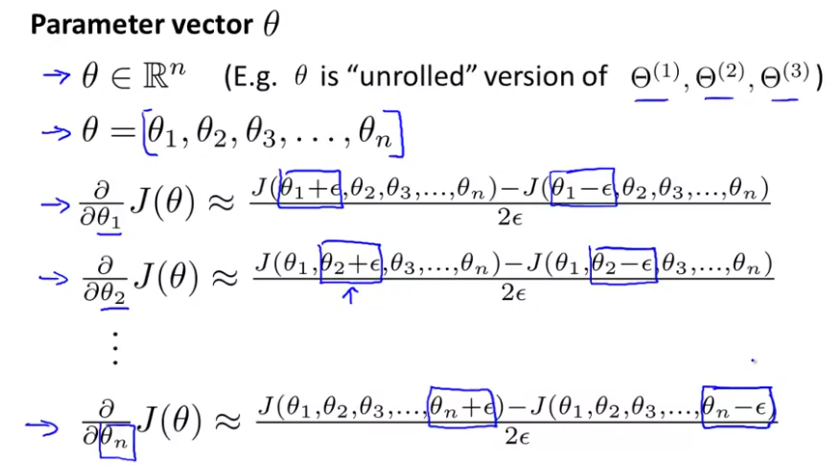
**Gradient Checking**

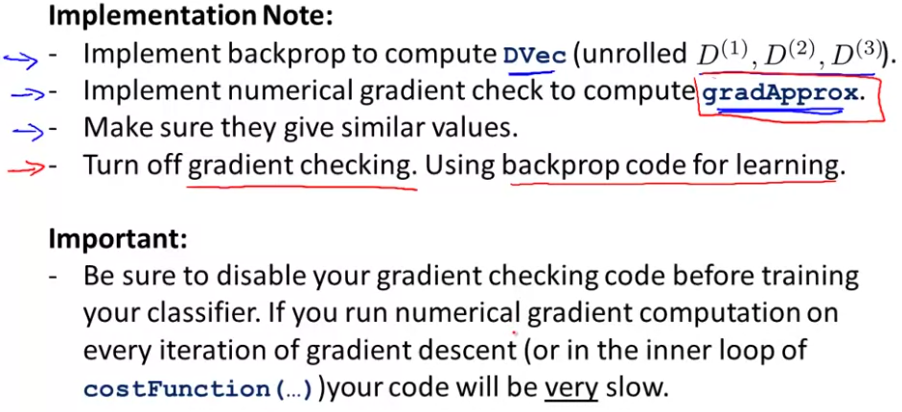
It is useful to see if your backpropagation algorithm is working correctly. You calculate a numerical approximation of the gradient and then you compare this value to the one obtained by the backpropagation algorithm.



Two sided derivative numerical approximation

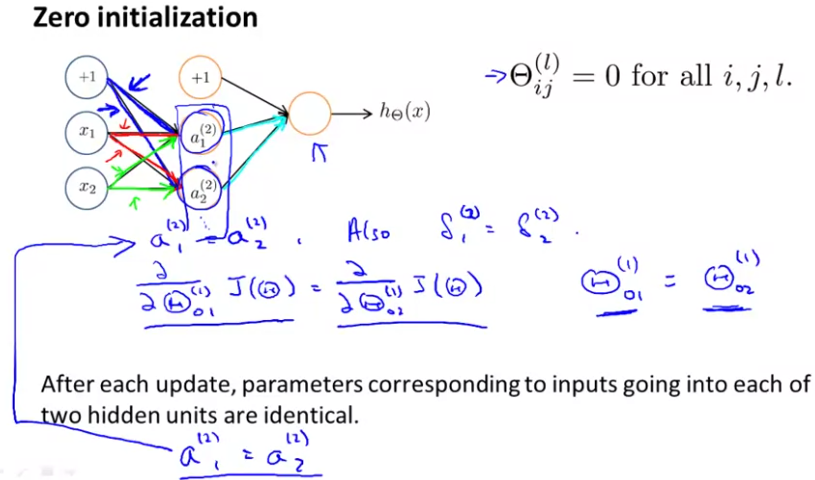
For obtaining the partial derivatives of the cost function with respect to each theta:

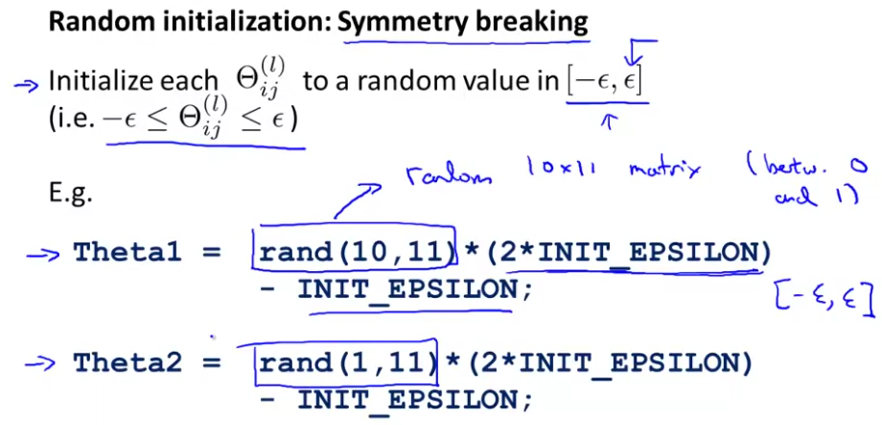




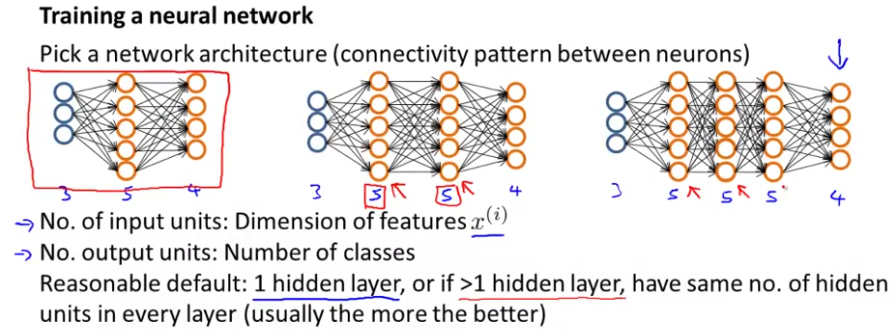
**Random initialization**

Initialising the theta parameters to zero works fine in logistic regression but not with the neural network parameters.

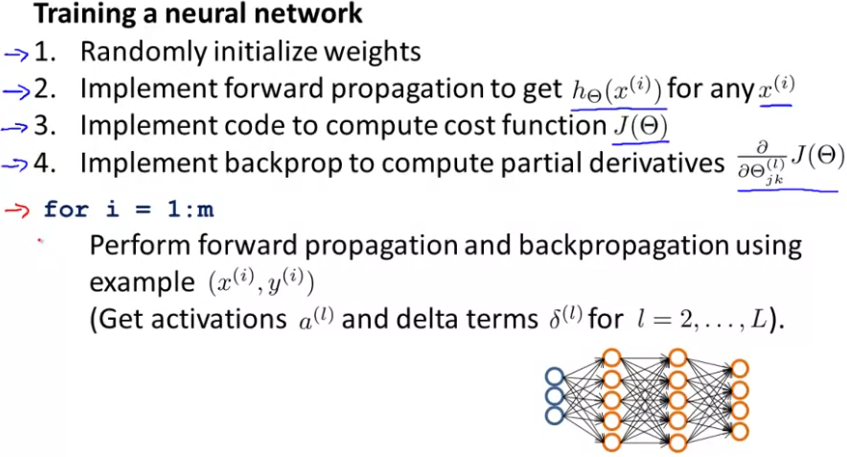


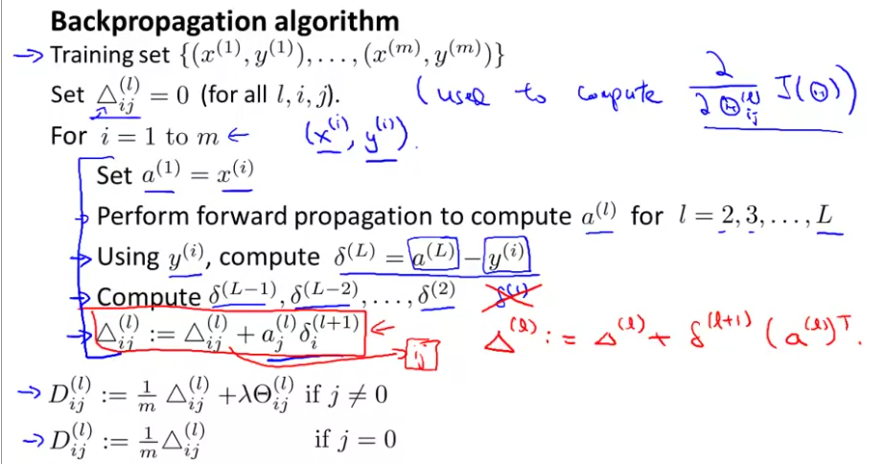


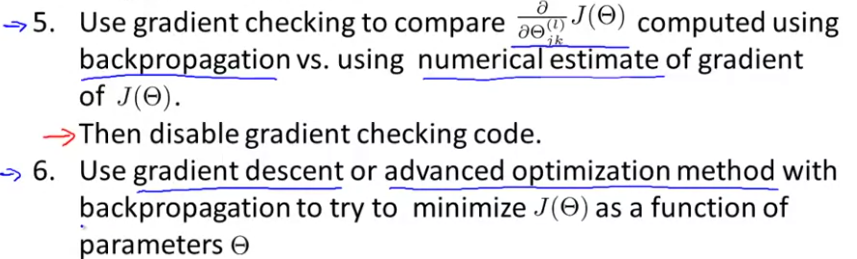
They must be values close to 0



The number of hidden units usually can be between the number of features and just up to 3 or 4 times this number. More hidden units are better but it increases the computational cost.







Note: The cost functions in neural networks are non-convex, which means we can get stuck to a local minimum. However, that’s not really a great problem and they usually give good results.