



# Putting it Together

First, pick a network architecture; choose the layout of your neural network, including how many hidden units in each layer and how many layers in total you want to have.

- Number of input units = dimension of features  $x^{(i)}$
- Number of output units = number of classes
- Number of hidden units per layer = usually more the better (must balance with cost of computation as it increases with more hidden units)
- Defaults: 1 hidden layer. If you have more than 1 hidden layer, then it is recommended that you have the same number of units in every hidden layer.

## Training a Neural Network

1. Randomly initialize the weights
2. Implement forward propagation to get  $h_{\Theta}(x^{(i)})$  for any  $x^{(i)}$
3. Implement the cost function
4. Implement backpropagation to compute partial derivatives
5. Use gradient checking to confirm that your backpropagation works. Then disable gradient checking.
6. Use gradient descent or a built-in optimization function to minimize the cost function with the weights in theta.

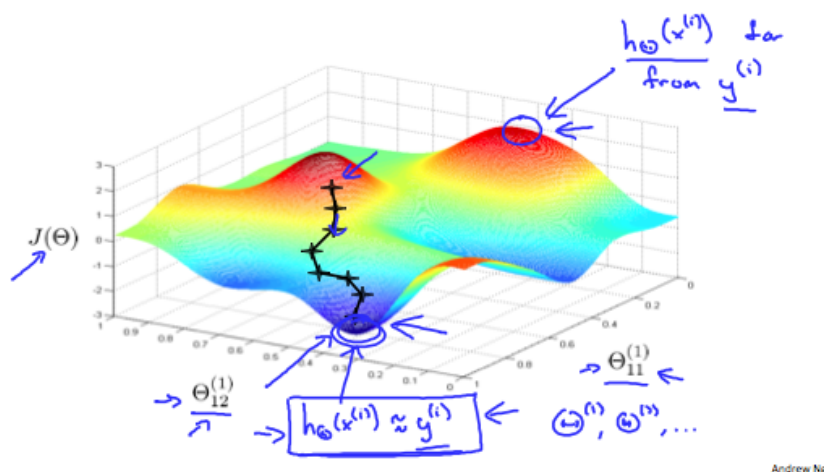
When we perform forward and back propagation, we loop on every training example:

```

1 for i = 1:m,
2     Perform forward propagation and backpropagation using example (x(i),y(i))
3     (Get activations a(l) and delta terms d(l) for l = 2,...,L)

```

The following image gives us an intuition of what is happening as we are implementing our neural network:



Ideally, you want  $h_{\Theta}(x^{(i)}) \approx y^{(i)}$ . This will minimize our cost function. However, keep in mind that  $J(\Theta)$  is not convex and thus we can end up in a local minimum instead.

✓ Complete