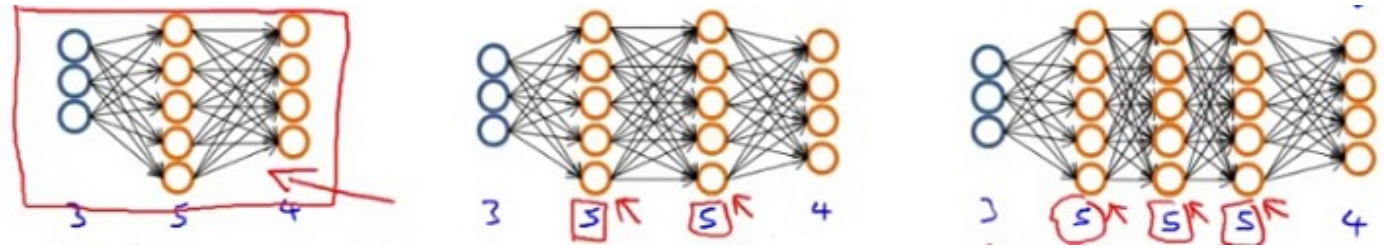
## ~ | Putting it Together | ~

When training a neural network, the first thing we need to do is pick some network architecture and by architecture just we mean connectivity pattern between the neurons.

These choices of how many hidden units in each layer and how many hidden layers, those are architecture choices. So, how do we make these choices?




**1. Training a neural network: Choosing a Neural Network architecture**

Pick a network architecture (connectivity pattern between neurons).

- **No. of input units:** Dimension of features $x^{(i)}$
- **No. output units:** Number of classes (if we have a multiclass classification where $y \in \{1, 2, 3, \ldots, 10\}$, so we have ten possible classes.)
- Reasonable default: 1 hidden layer, or if > 1 hidden layer, have some no. of units in every layer (usually the more the better).

So the choice of number of input units and number of output units is maybe somewhat reasonably straightforward, and as for the number of hidden units and the number of hidden layers, a reasonable default is to use a single hidden layer, or if we use more than one hidden layer, again the reasonable default will be to have **the same number of hidden units in every single layer**.

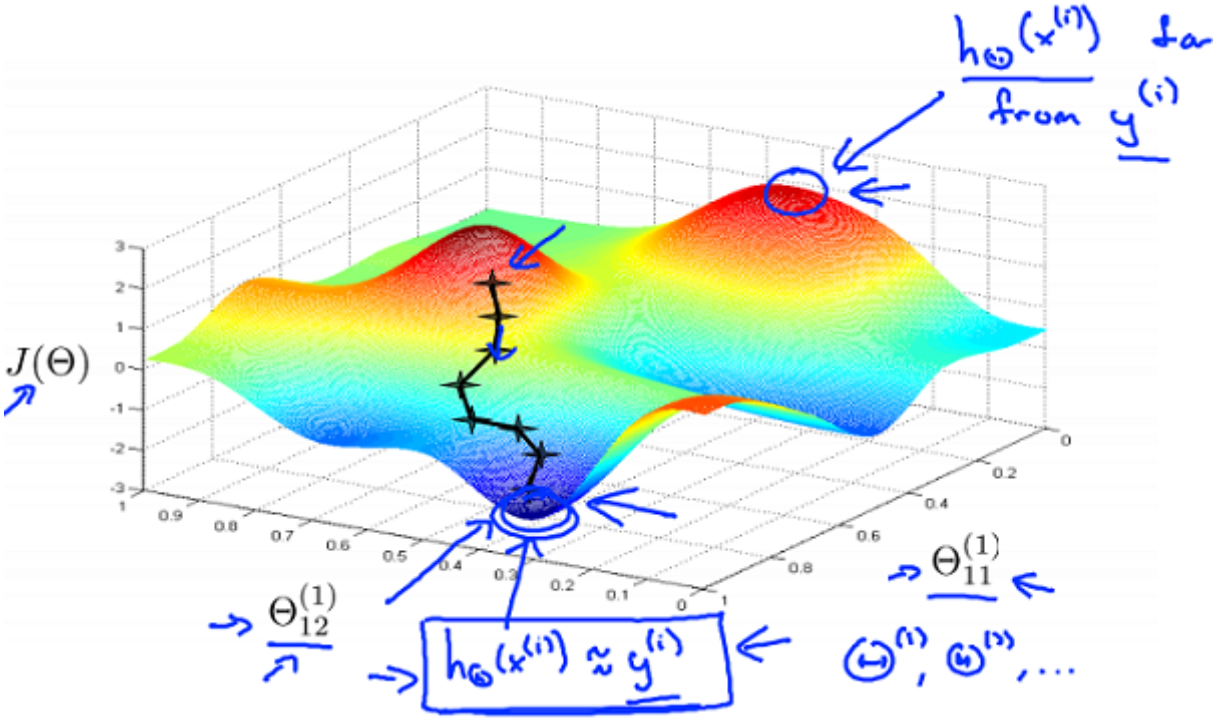
And as for the number of hidden units - usually, the more hidden units the better; it's just that if we have a lot of hidden units, it can become more computationally expensive, and usually the number of hidden units in each layer will be maybe comparable to the dimension of $x$, comparable to the number of features, or it could be any where from same number of hidden units of input features to maybe so that three or four times of $x$.

**2. Training a neural network: How to implement a Neural Network?**

1. Randomly initialize weights (we usually initialize the weights to small values near zero).
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$.
3. Implement code to compute cost function $J(\Theta)$.
4. Implement backpropagation to compute partial derivates $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$.
5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backpropagation vs. using numerical estimate of gradient of $J(\Theta)$. Then disable gradient checking code.
6. Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters $\Theta$.

Concretely, to implement backpropagation. Usually we will do compute partial derivates $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ with a for loop over the training examples.

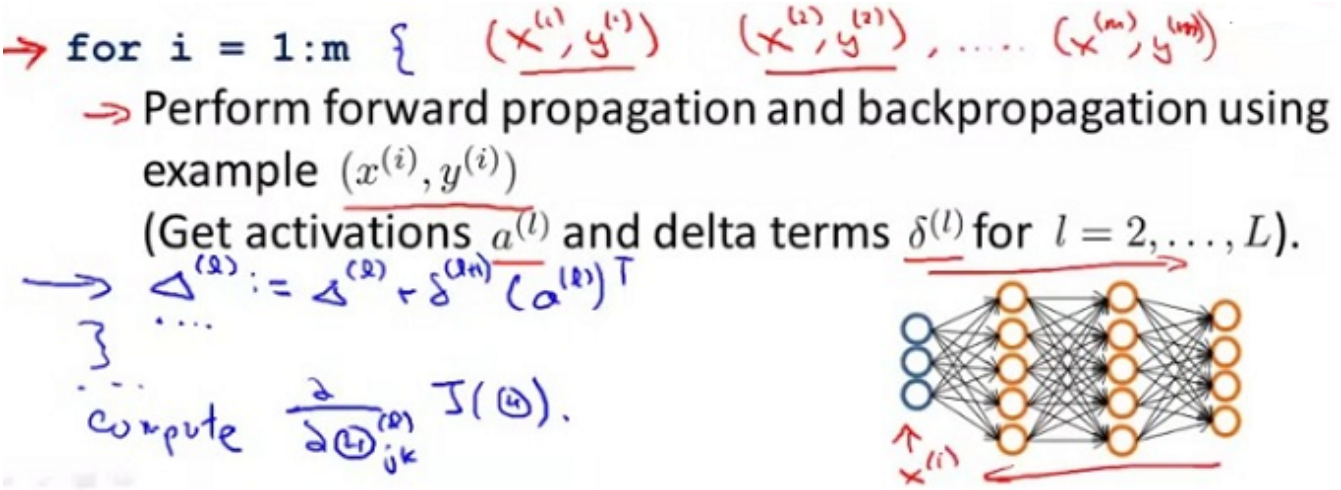
For neural networks, this cost function $J(\Theta)$ of theta is **non-convex**, and so it can theoretically be susceptible to local minima, but it turns out that in practice this is not usually a huge problem and even though we can't guarantee that these algorithms will find a global optimum.




- e.g. above pretending data only has two features to easily display what's going on
  - Our minimum here represents a hypothesis output ($h_\Theta(x^{(i)})$) which is pretty close to $y$
  - If we took one of the peaks hypothesis is far from $y$.
- Gradient descent will start from some random point and move downhill
  - Back propagation calculates gradient down that hill

**Notes on implementation:**

- Usually done with a for loop over training examples (for forward and back propagation)
- Can be done without a for loop, but this is a much more complicated way of doing things




**Video Question:** Suppose you are using gradient descent together with backpropagation to try to minimize $J(\Theta)$ as a function of $\Theta$. Which of the following would be a useful step for verifying that the learning algorithm is running correctly?

- Plot $J(\Theta)$ as a function of $\Theta$, to make sure gradient descent is going downhill.
- Plot $J(\Theta)$ as a function of the number of iterations and make sure it is increasing (or at least non-decreasing) with every iteration.

> Plot $J(\Theta)$ as a function of the number of iterations and make sure it is decreasing (or at least non-increasing) with every iteration.

- Plot $J(\Theta)$ as a function of the number of iterations to make sure the parameter values are improving in classification accuracy.

## Summary

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.

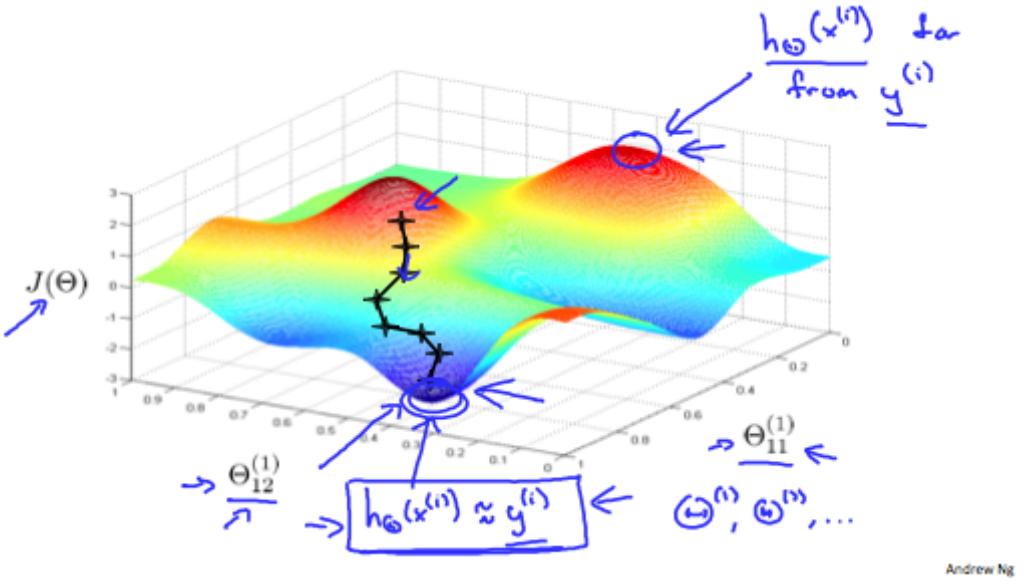
#### Training a Neural Network

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:

```
for i = 1:m,
   Perform forward propagation and backpropagation using example (x(i),y(i))
   (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.