



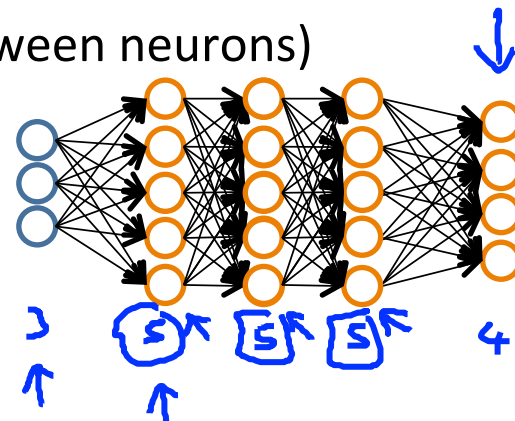
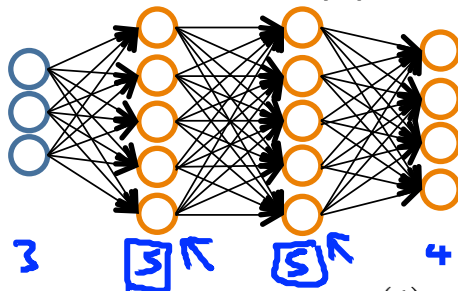
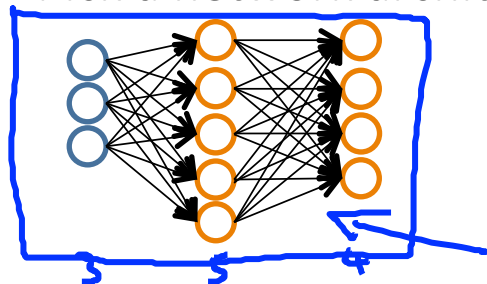
Machine Learning

Neural Networks: Learning

Putting it together

Training a neural network

Pick a network architecture (connectivity pattern between neurons)



→ No. of input units: Dimension of features $\underline{x^{(i)}}$

→ No. output units: Number of classes

Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)

$y \in \{1, 2, 3, \dots, 10\}$
 ~~$y = 5$~~

$y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \text{ or } \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \text{ or } \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$

Training a neural network

- 1. Randomly initialize weights
- 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 backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

→ for $i = 1:m$ { $(x^{(1)}, y^{(1)})$ $(x^{(2)}, y^{(2)})$, ..., $(x^{(m)}, y^{(m)})$

→ Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$

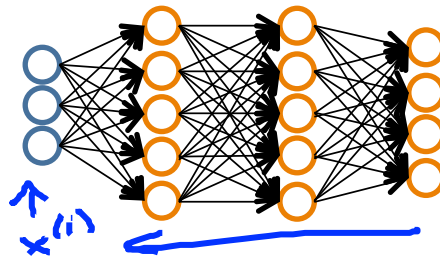
(Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \dots, L$).

→ $\Delta^{(2)} := \Delta^{(2)} + \delta^{(L)} (a^{(2)})^T$

...

}

compute $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$.



Training a neural network

- 5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{ik}^{(l)}} J(\Theta)$ 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 Θ

$\frac{\partial}{\partial \Theta_{ik}^{(l)}} J(\Theta)$

$J(\Theta)$ — non-convex.

