ECO 352: Machine learning: Neural networks

Serguei Maliar

Stony Brook University

ECO 352: Machine learning: Neural networks

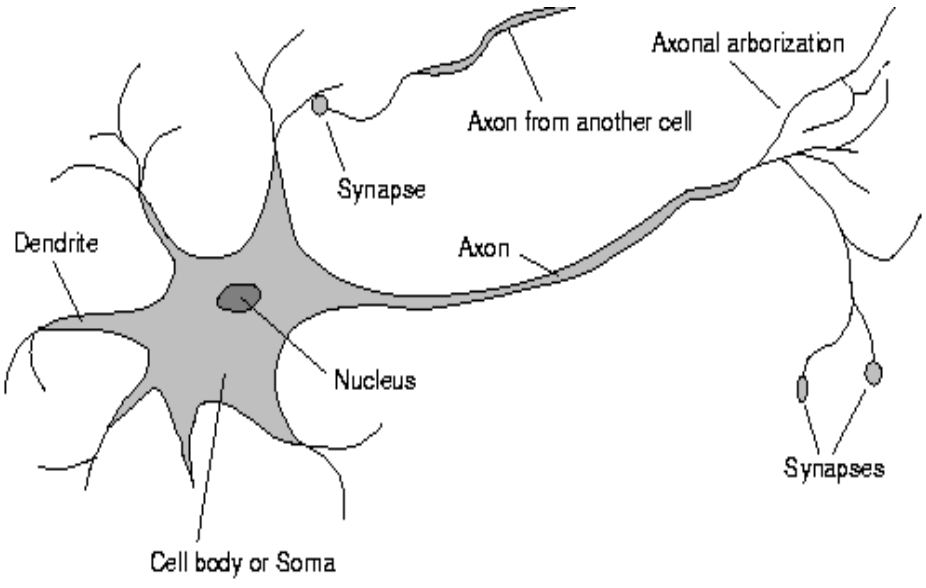
Serguei Maliar

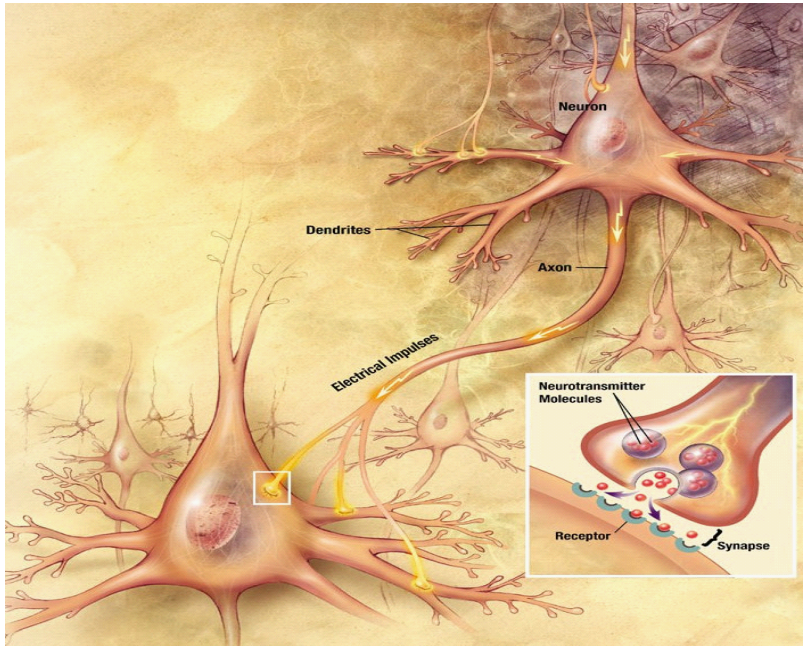
Stony Brook University

Brain function (thought) occurs as the result of the firing of neurons

- Neurons connect to each other through synapses, which propagate action potential (electrical impulses) by releasing neurotransmitters
- Synapses can be excitatory (potential--increasing) or inhibitory (potential--decreasing), and have varying activation thresholds
- Learning occurs as a result of the synapses' plasticicity: They exhibit long-term changes in connection strength
- There are about 1011 neurons and about 1014 synapses in the human brain!

Biology of a neuron





Brain Structure

Different areas of the brain have different functions

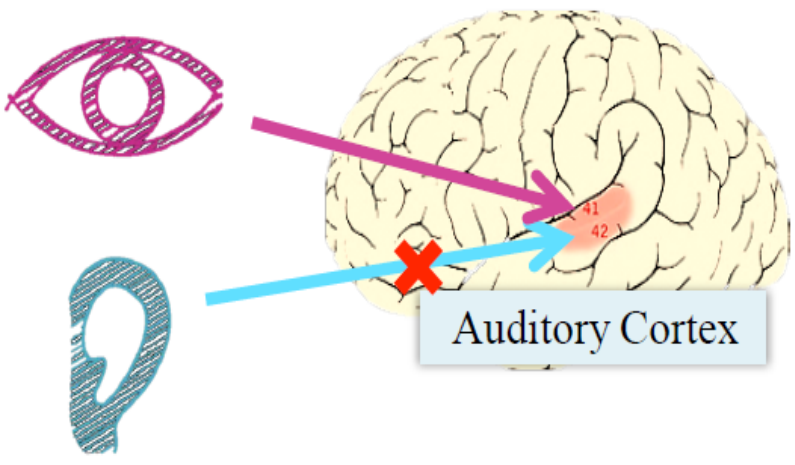
- Some areas seem to have the same function in all humans (e.g., Broca's region for motor speech); the overall layout is generally consistent
- Some areas are more plastic, and vary in their function; also, the lower--level structure and function vary greatly

We don't know how different functions are "assigned" or acquired

- Partly the result of the physical layout / connection to inputs (sensors) and outputs (effectors)
- Partly the result of experience (learning)

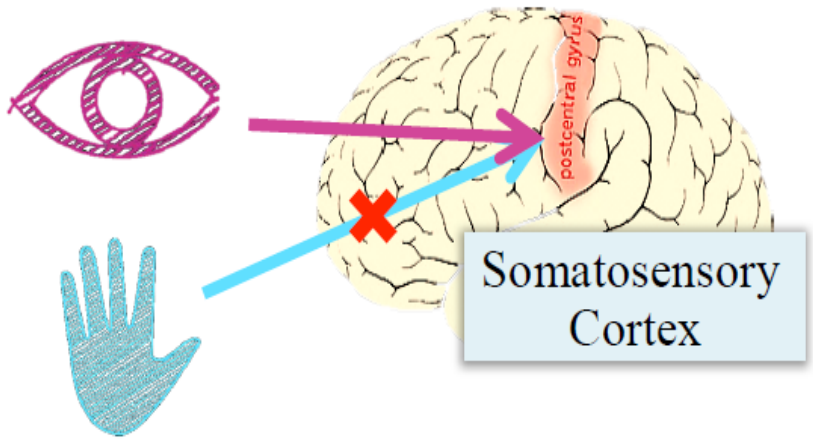
We really don't understand how this neural structure leads to what we perceive as "consciousness" or "thought"

The "One Learning Algorithm" Hypothesis



Auditory cortex learns to see

[Roe et al., 1992]



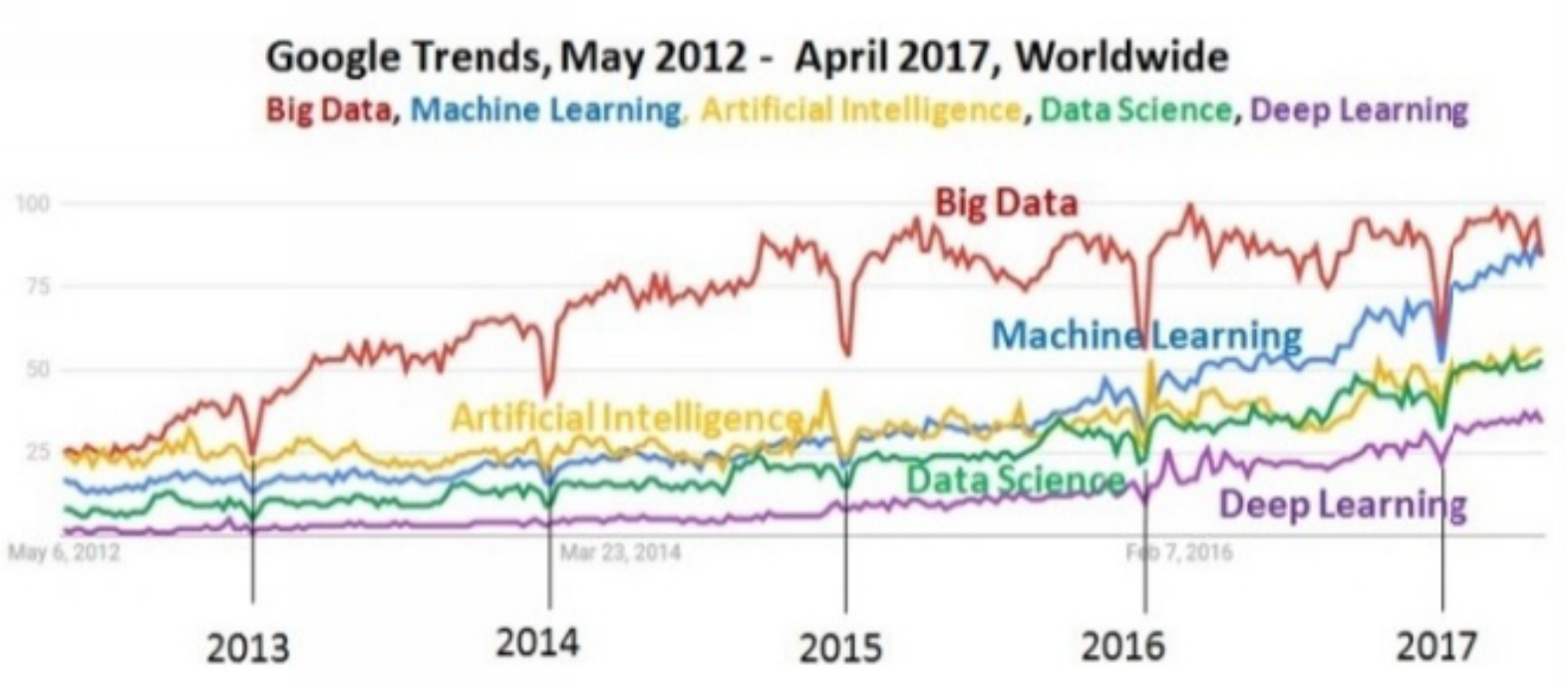
Somatosensory cortex learns to see

[Metin & Frost, 1989]

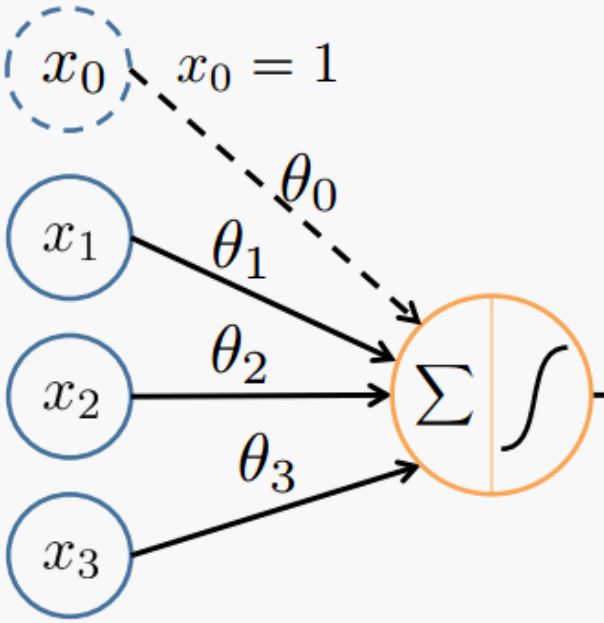
Neural networks

- Origins: Algorithms that try to mimic the brain.
- Very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications
- Artificial neural networks are not nearly as complex or intricate as the actual brain structure

The rise of machines



Neuron Model: Logistic Unit

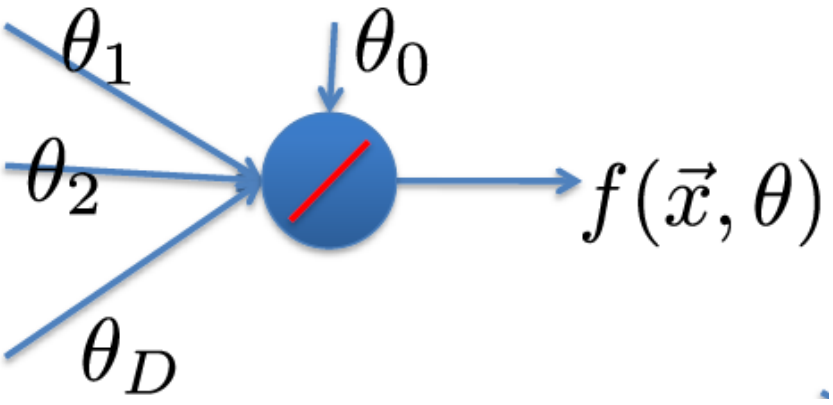


$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

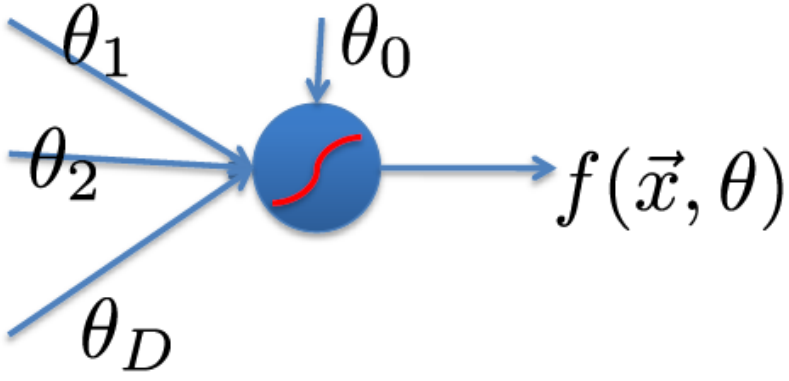
$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}}}$$

Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

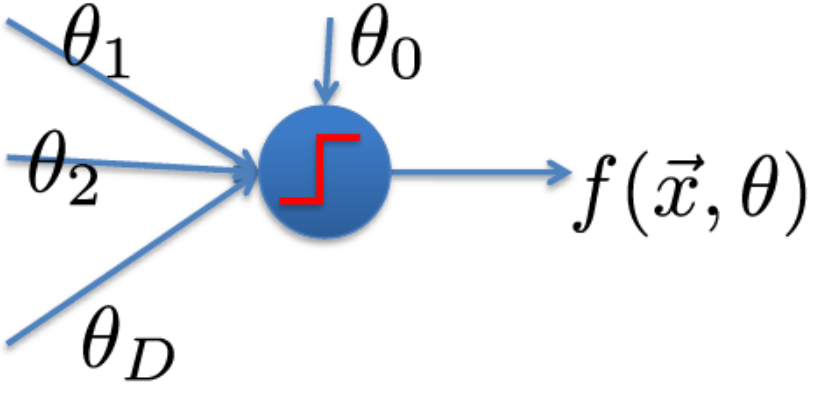
Types of Neurons



Linear Neuron



Logistic Neuron



Perceptron

Potentially more. Require a convex loss function for gradient descent training.

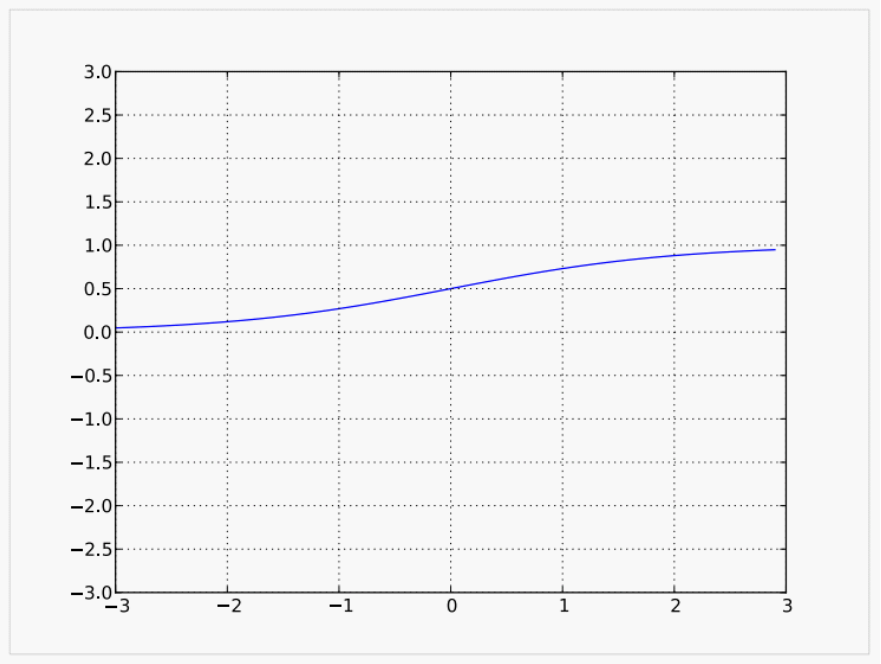
Activation - sigmoid

– Squashes the neuron's pre-activation between 0 and 1

– Always positive

– Bounded

– Strictly increasing



$$g(x) = \frac{1}{1 + e^{-x}}$$

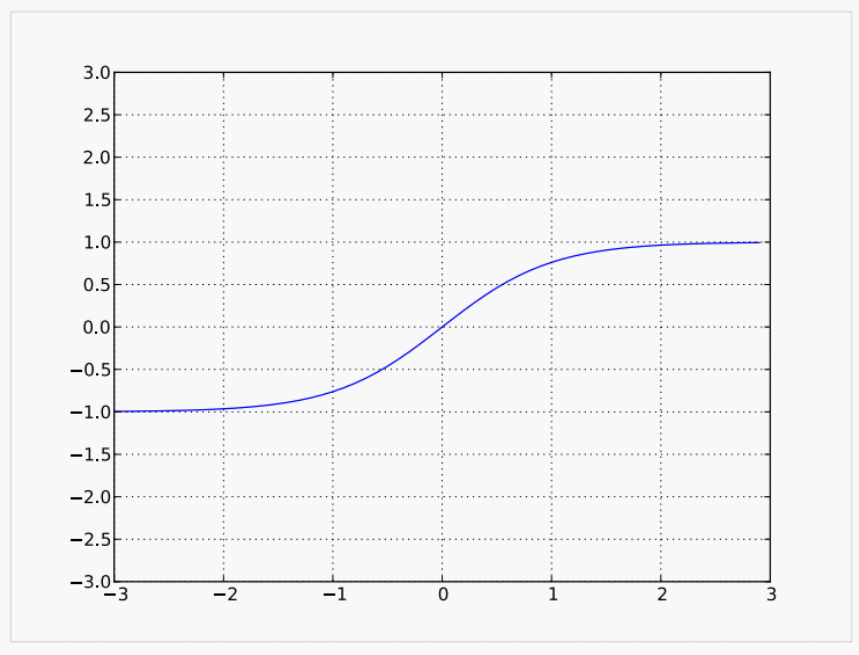
Activation - hyperbolic tangent (tanh)

– Squashes the neuron's pre-activation between -1 and 1

– Can be positive or negative

– Bounded

– Strictly increasing



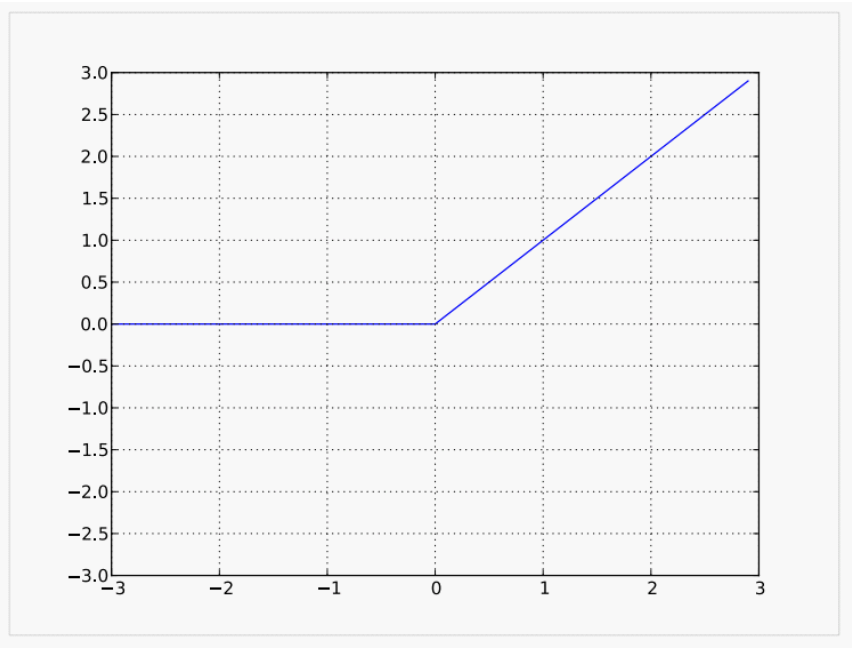
$$g(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Activation - rectified linear (relu)

– Bounded below by 0, always non-negative

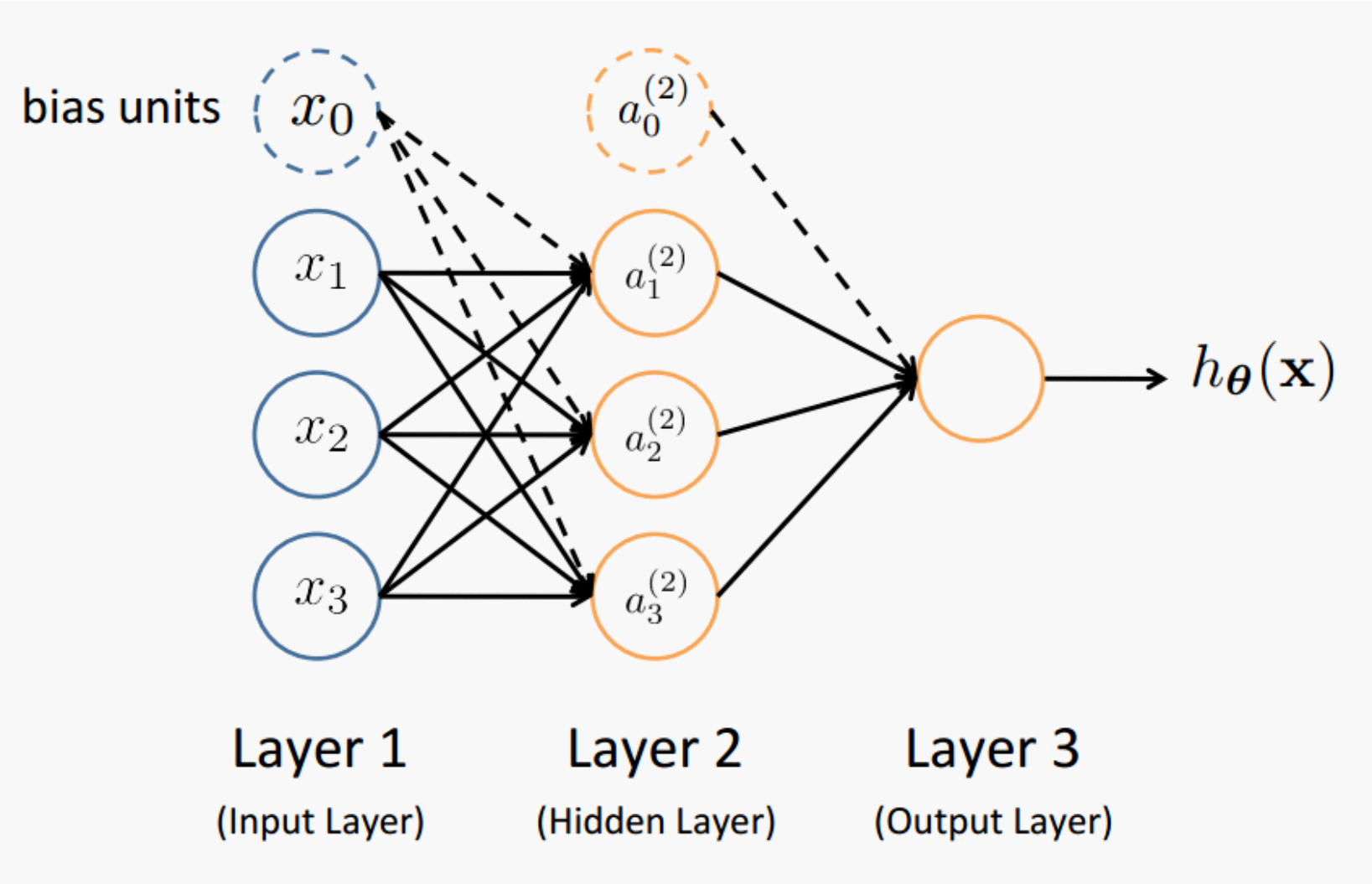
– Not upper bounded

– Tends to give neurons with sparse activities



$$g(x) = relu(x) = \max(0, x)$$

Deep Neural Network



Feed-Forward Process

- Input layer units are set by some exterior function (think of these as sensors), which causes their output links to be activated at the specified level

- Working forward through the network, the input function of each unit is applied to compute the output value

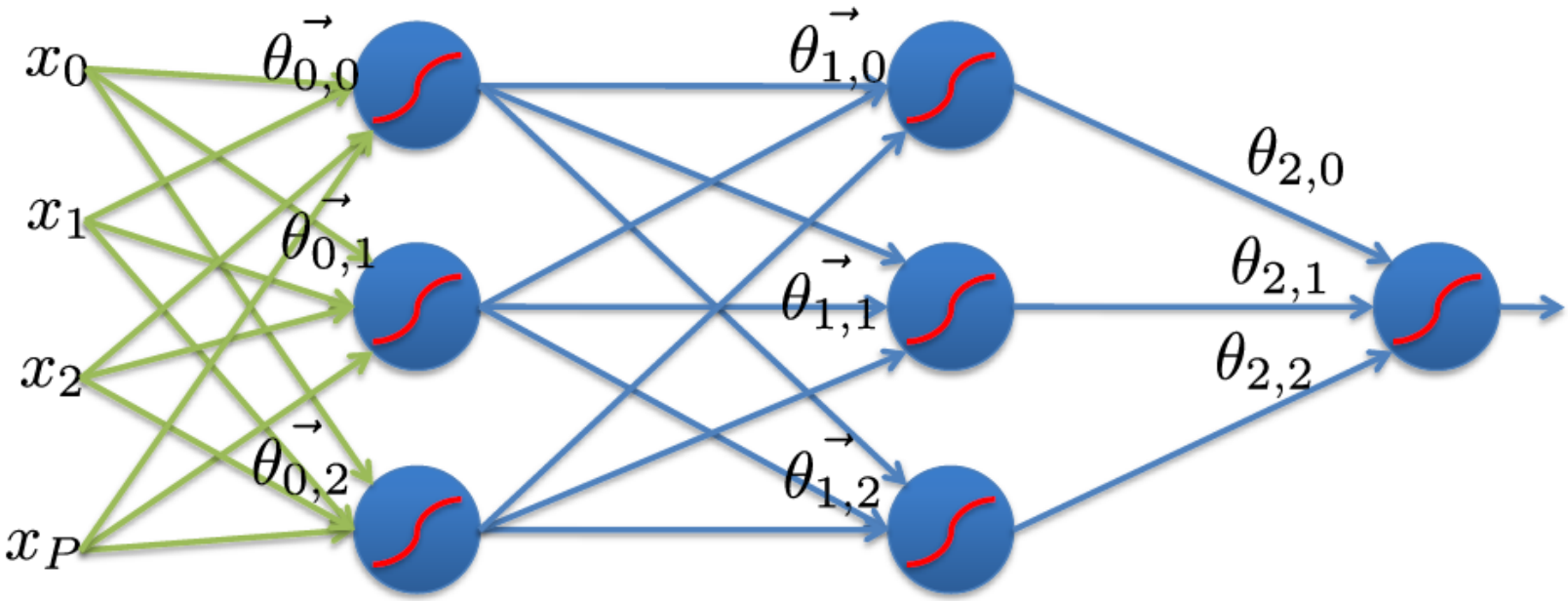
  – Usually this is just the weighted sum of the activation on the links feeding into this node

- The activation function transforms this input function into a final value

  – Typically this is a nonlinear function, often a sigmoid function corresponding to the "threshold" of that node

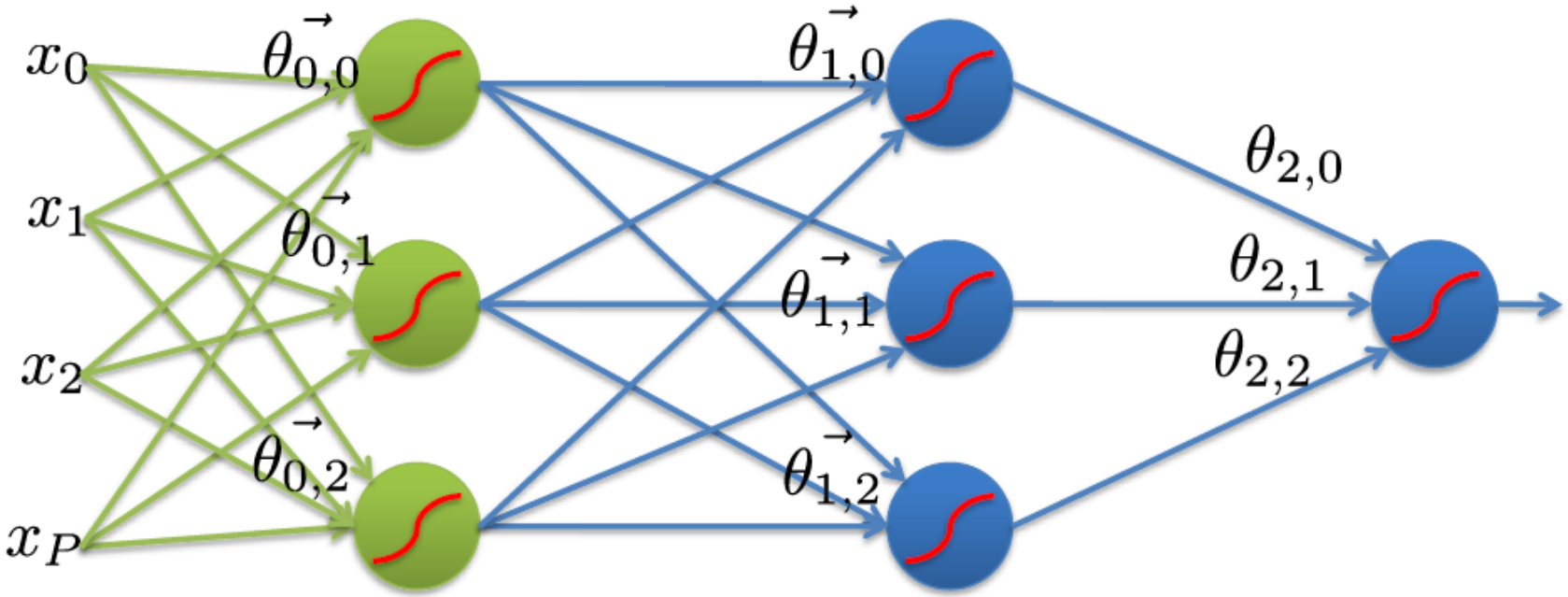
Feed-Forward Networks

Predictions are fed forward through the network to classify.



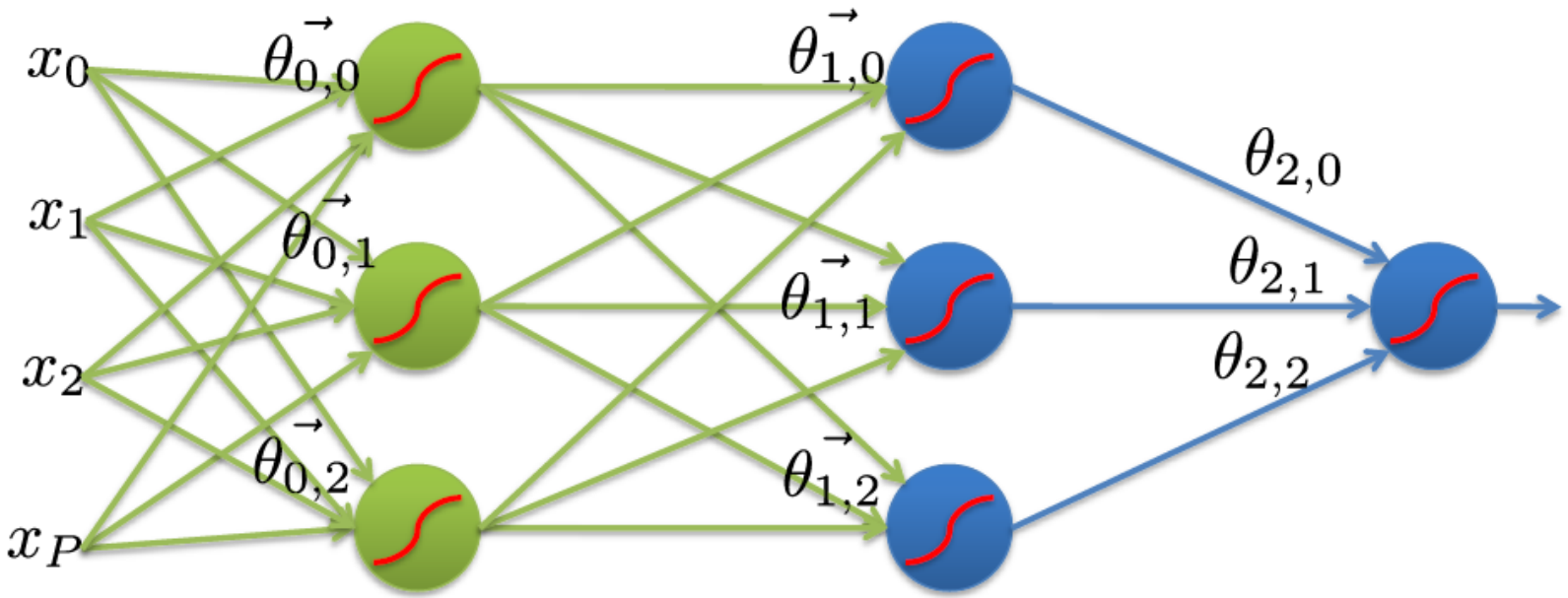
Feed-Forward Networks

Predictions are fed forward through the network to classify



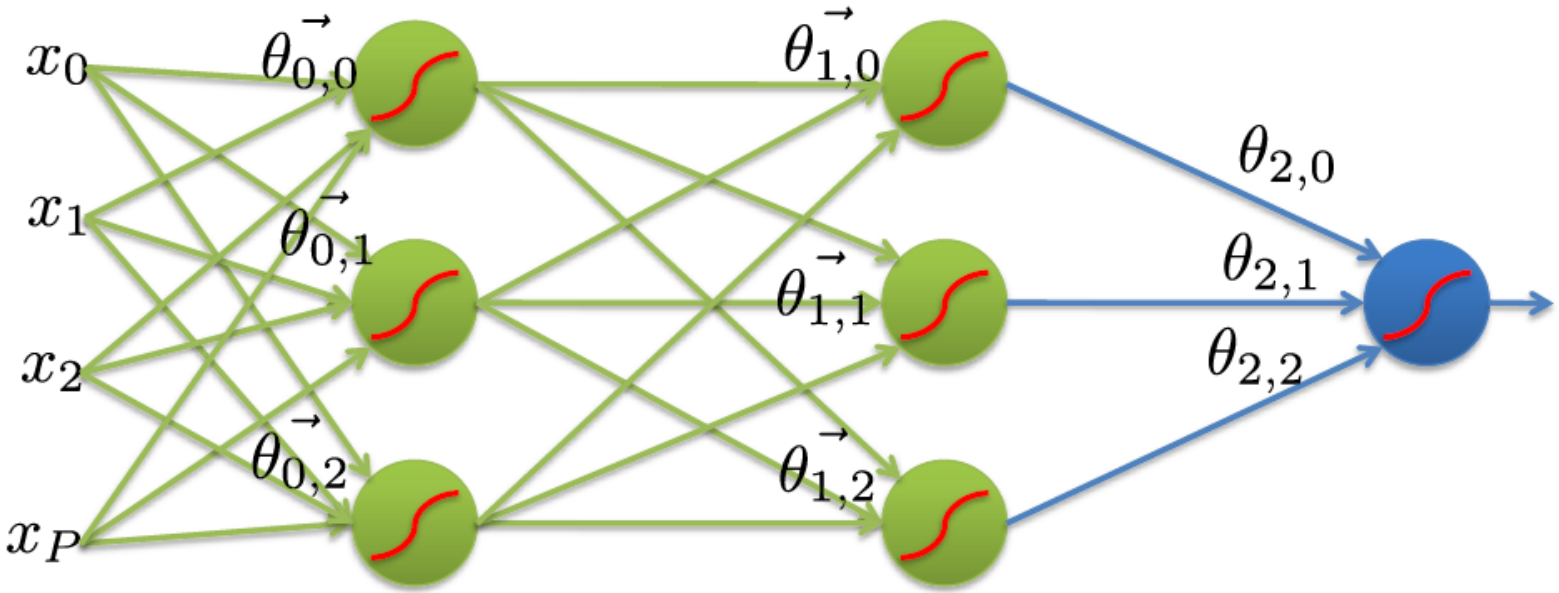
Feed-Forward Networks

Predictions are fed forward through the network to classify



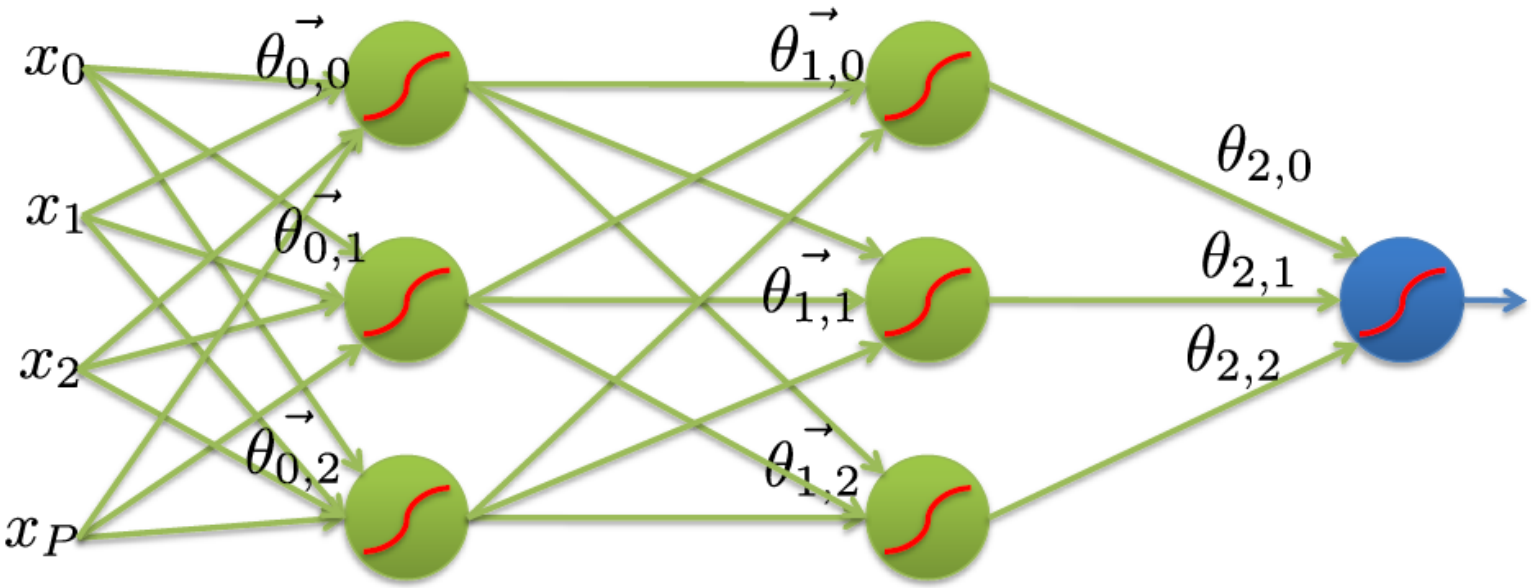
Feed-Forward Networks

Predictions are fed forward through the network to classify



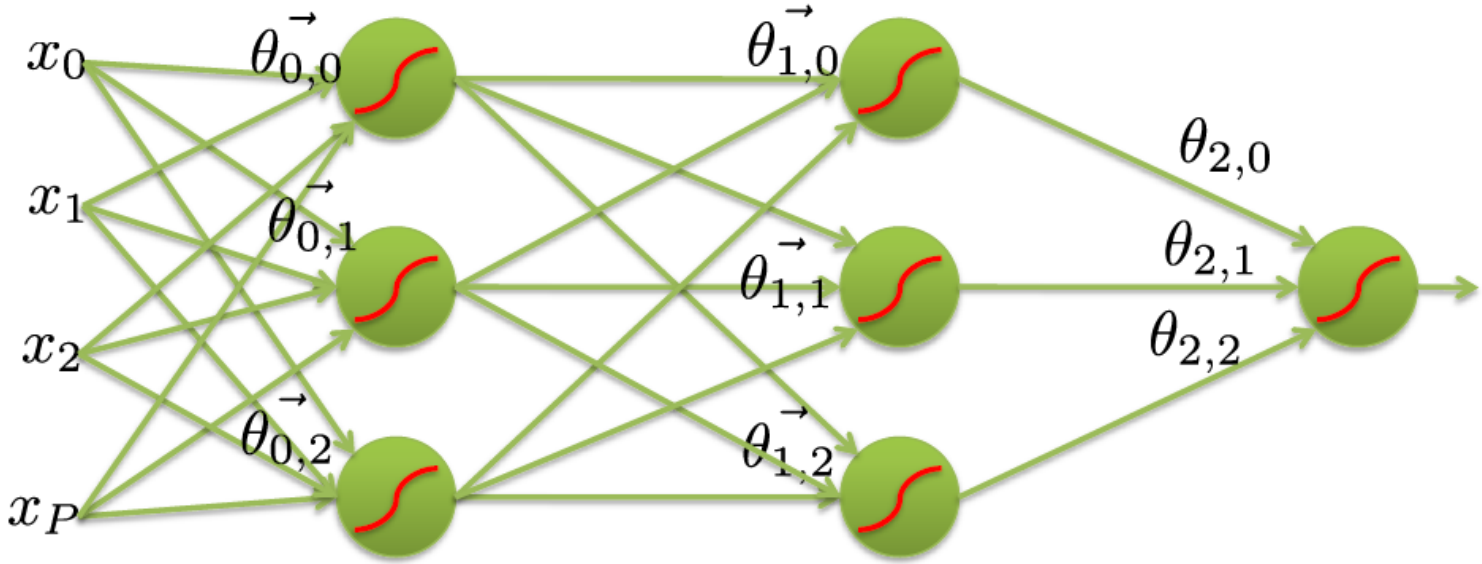
Feed-Forward Networks

Predictions are fed forward through the network to classify

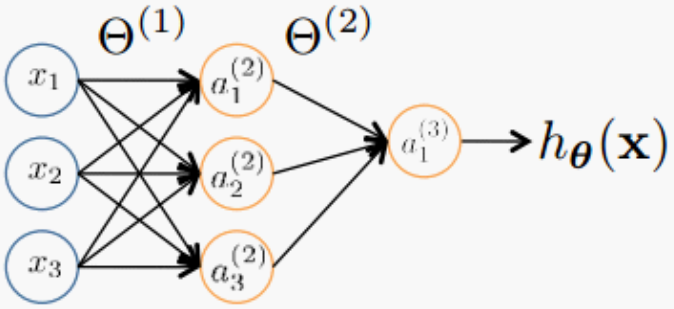


Feed-Forward Networks

Predictions are fed forward through the network to classify



Neural Network



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = weight matrix controlling function mapping from layer j to layer $j+1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

If network has s_j units in layer j *and* s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j+1)$.

$$\Theta^{(1)} \in \mathbb{R}^{3\times 4} \qquad \Theta^{(2)} \in \mathbb{R}^{1\times 4}$$

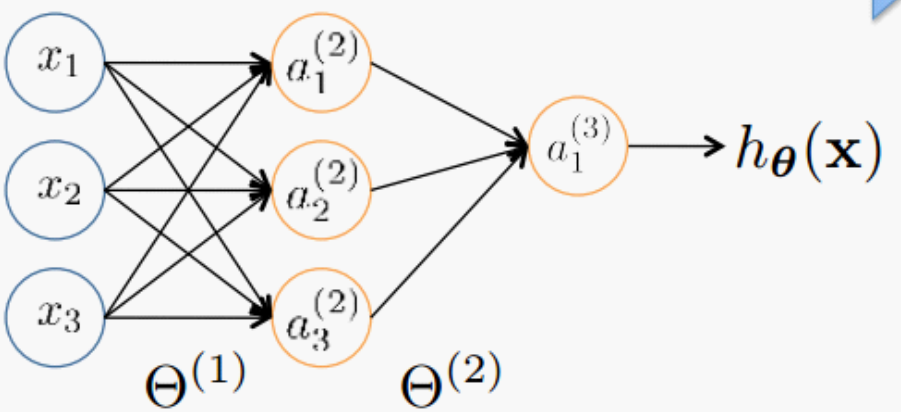
Vectorization

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right) = g\left(z_2^{(2)}\right)$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right) = g\left(z_3^{(2)}\right)$$

$$h_\Theta(\mathbf{x}) = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right) = g\left(z_1^{(3)}\right)$$



Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$

$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

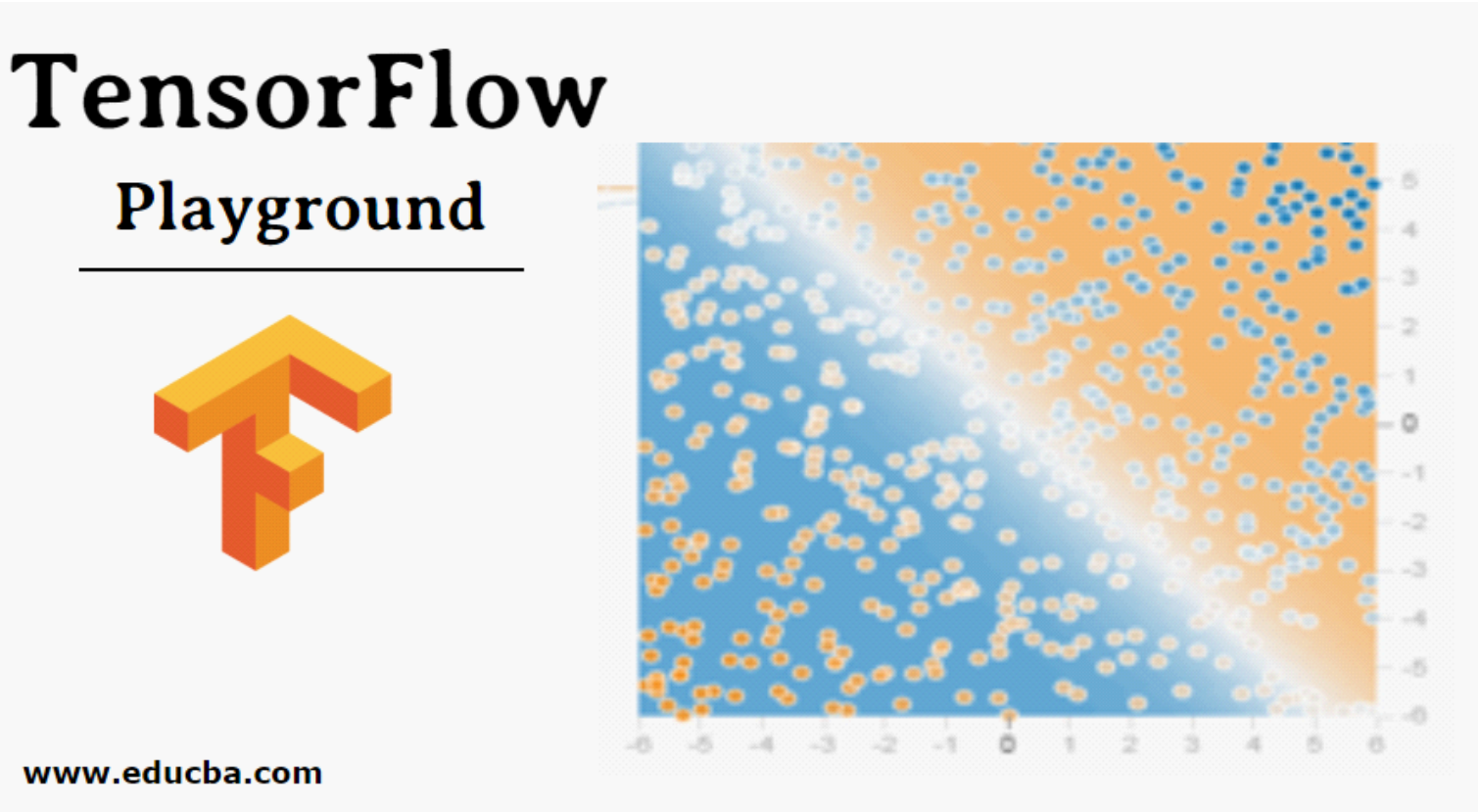
Add $a_0^{(2)} = 1$

$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$

$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

TensorFlow

TensorFlow Demonstration
`https://playground.tensorflow.org`



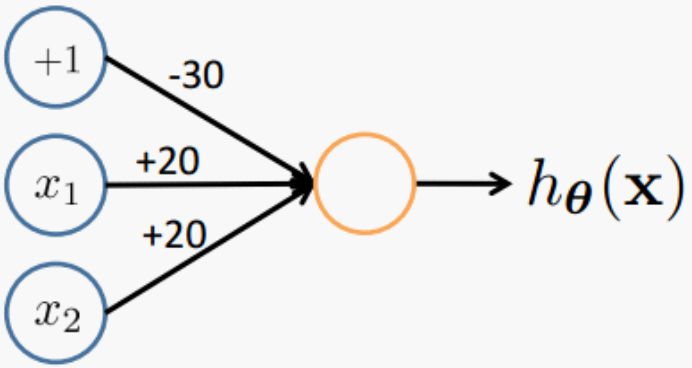
Understanding Representations

Representing Boolean Functions

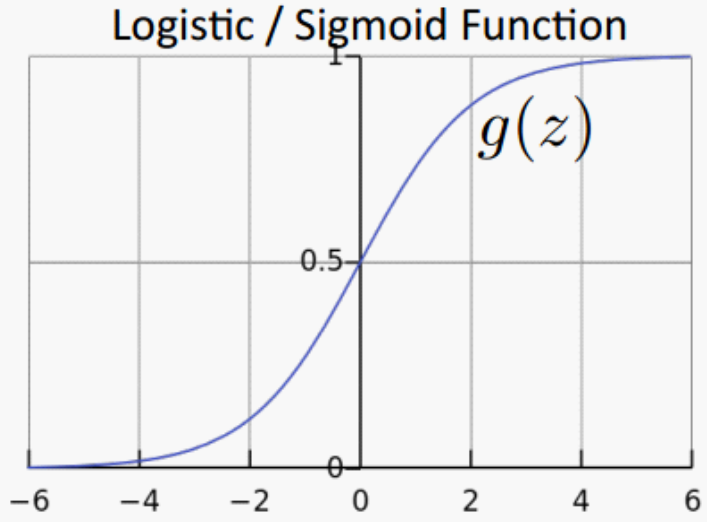
Simple example: AND

$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$

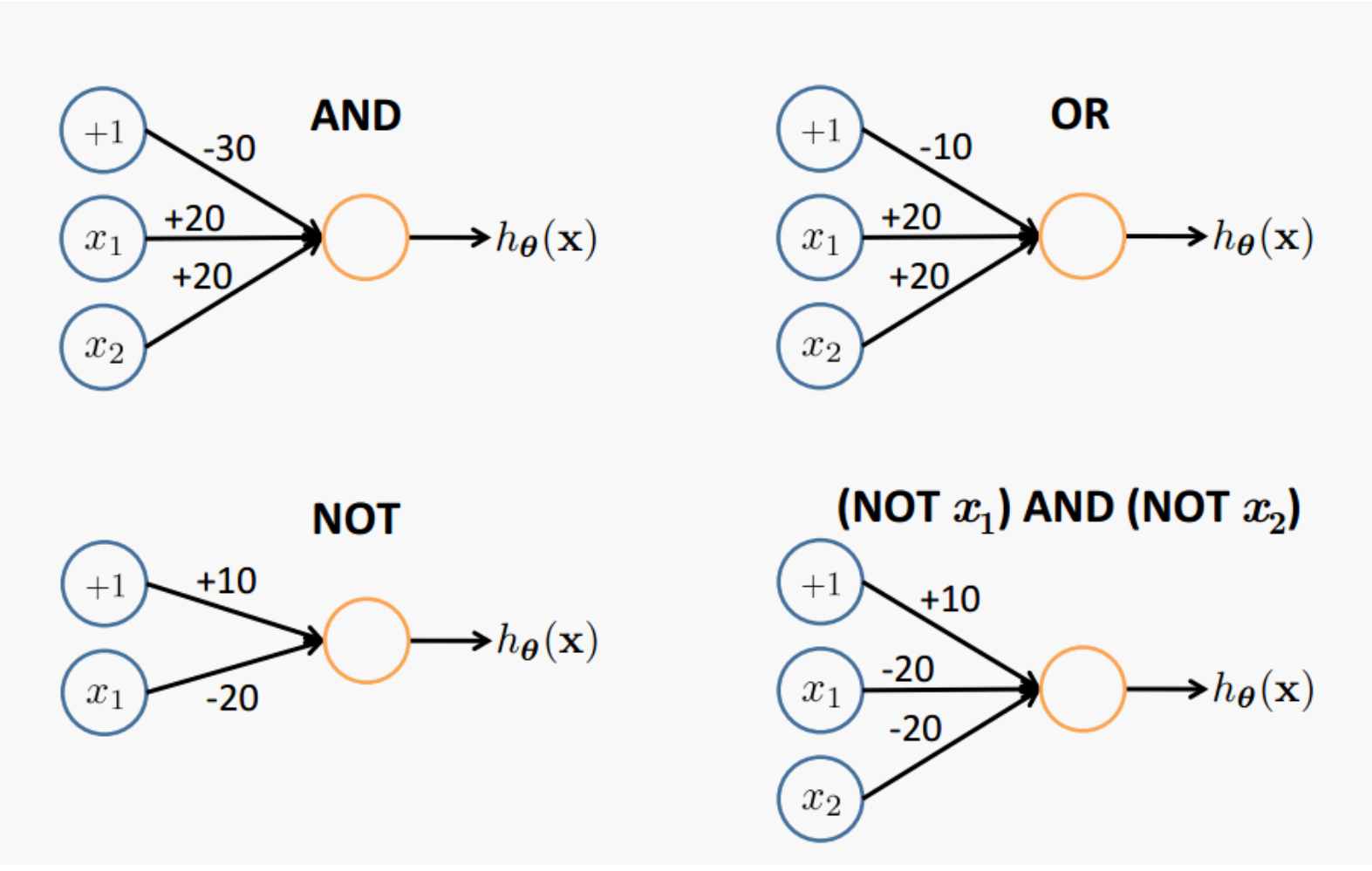


$\mathrm{h}_\Theta(\mathbf{x}) = g(\text{-}30 + 20x_1 + 20x_2)$

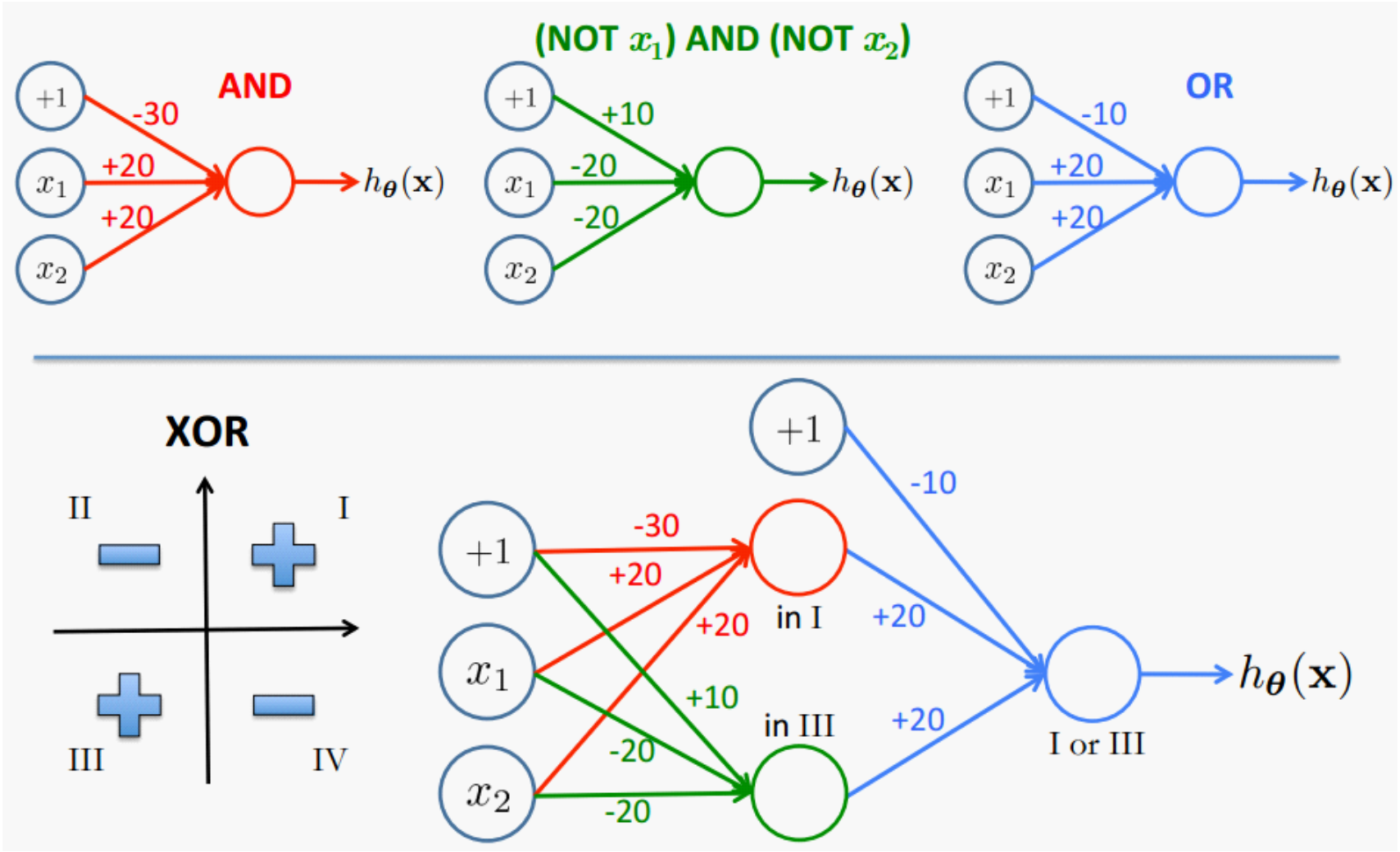


x_1	x_2	$\mathrm{h}_\Theta(\mathbf{x})$
0	0	$g(\text{-}30) \approx 0$
0	1	$g(\text{-}10) \approx 0$
1	0	$g(\text{-}10) \approx 0$
1	1	$g(10) \approx 1$

Representing Boolean Functions



Combining Representations to Create Non-Linear Functions



Multiclass Classification

Multiple Output Units: One-vs-Rest



Pedestrian



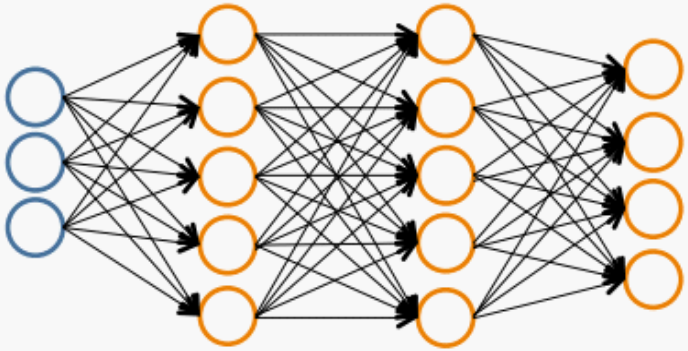
Car



Motorcycle



Truck



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

We want:

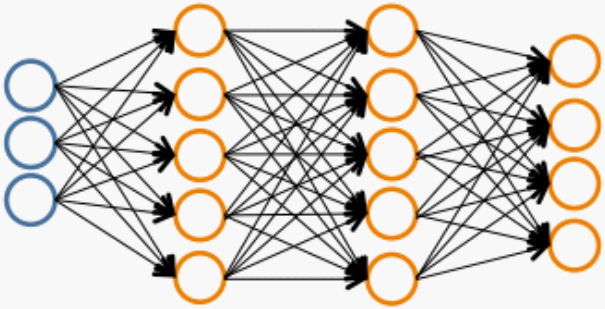
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$ when pedestrian

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$ when car

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$ when motorcycle

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$ when truck

Multiple Output Units: One-vs-Rest



$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$

We want:

$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ when pedestrian

$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ when car

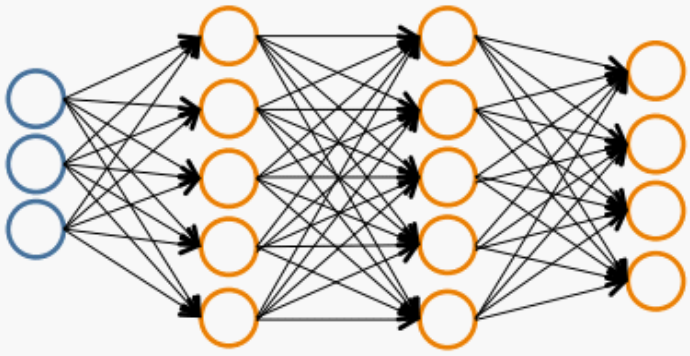
$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ when motorcycle

$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ when truck

---

- Given $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$
- Must convert labels to 1-of-K representation
  - e.g., $\mathbf{y}_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ when motorcycle, $\mathbf{y}_i = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ when car, etc.

Neural Network Classification



Given:

$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$

$\mathbf{s} \in \mathbb{N}^{+L}$ contains # nodes at each layer

- $s_0 = d$ (# features)

Binary classification

y = 0 or 1

1 output unit $(s_{L-1} = 1)$

Multi-class classification (K classes)

$\mathbf{y} \in \mathbb{R}^K$ e.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

K output units $(s_{L-1} = K)$

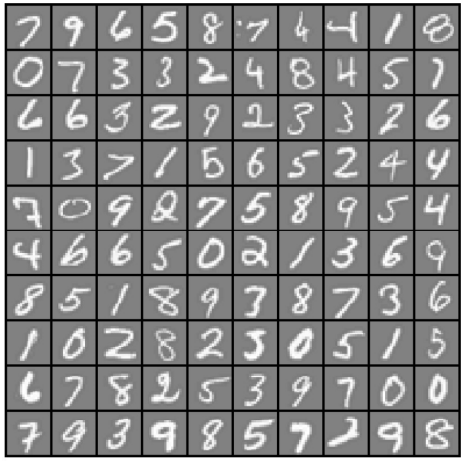
Activation - softmax

- For multi-class classification:
  - we need multiple outputs (1 output per class)
  - we would like to estimate the conditional probability $p\left(y = c \mid x\right)$

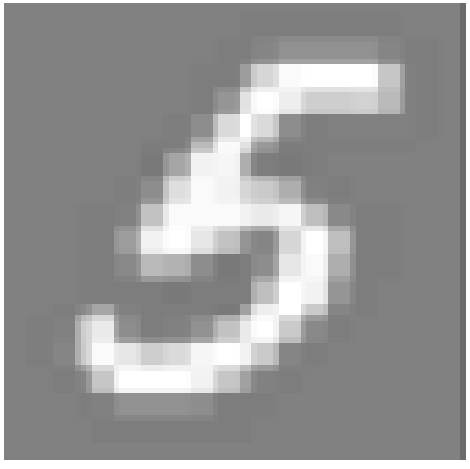
- We use the softmax activation function at the output:

$$g\left(x\right) = \text{softmax}\left(x\right) = \left[\frac{e^{x_1}}{\sum_c e^{x_c}} \ldots \frac{e^{x_c}}{\sum_c e^{x_c}}\right]$$

Layering Representations



20×20 pixel images

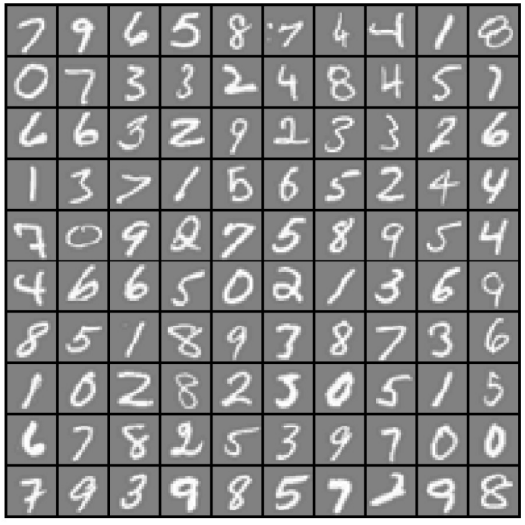


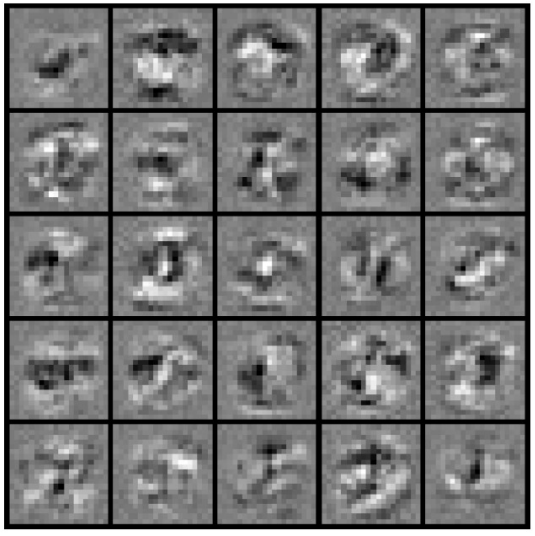
$d = 400$ $\quad$ 10 classes

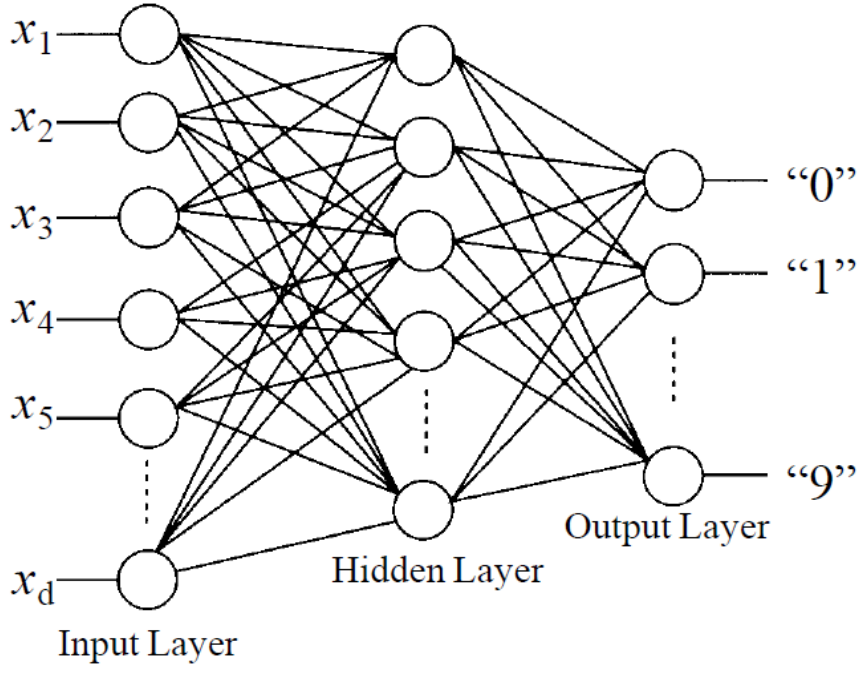
Each image is "unrolled" into a vector x of pixel intensities.

$$\begin{matrix} x_1 \dots x_{20} \\ \vdots \\ x_{381} \dots x_{400} \end{matrix}$$

Layering Representations





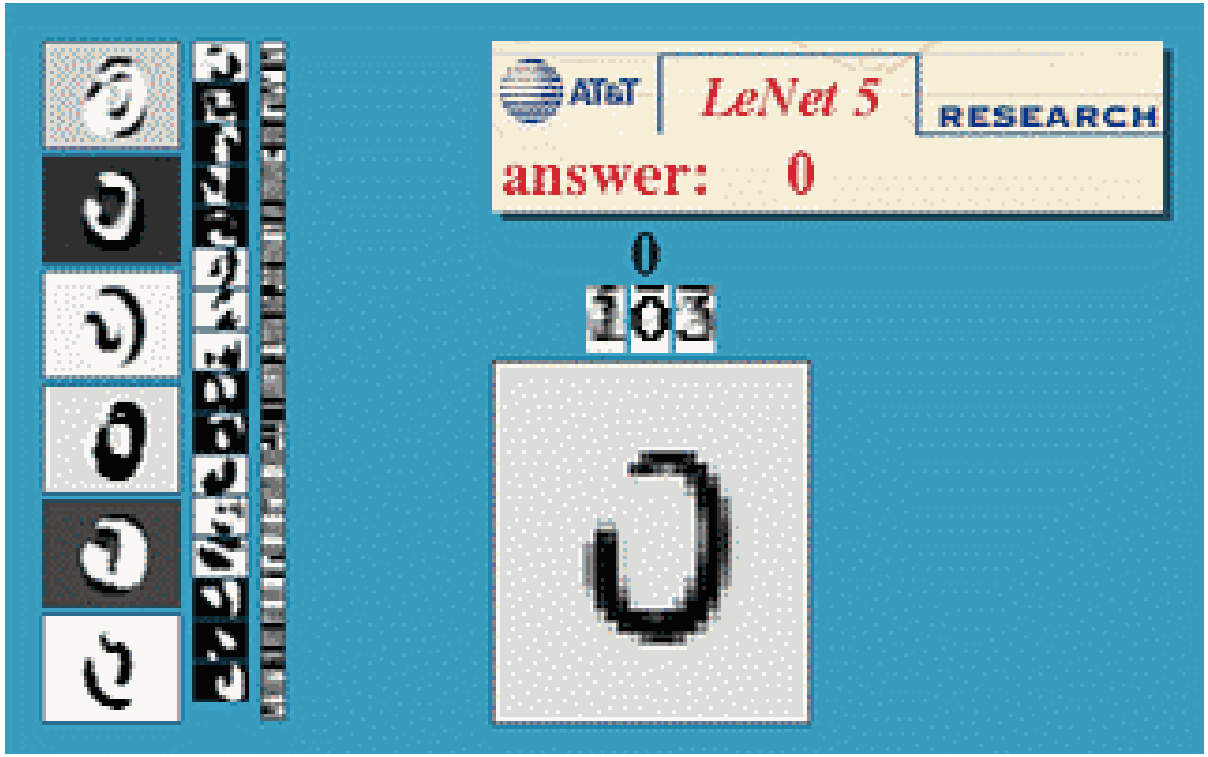


Visualization of Hidden Layer

LeNet 5

- LeNet 5 Demonstration

`http://yann.lecun.com/exdb/lenet`



Neural Network Learning

Universal approximation theorem

"a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units"

Hornik, 1991

Learning in NN: Backpropagation

- We cycle through our examples
  - If the output of the network is correct, no changes are made
  - If there is an error, weights are adjusted to reduce the error
- The trick is to assess the blame for the error and divide it among the contributing weights

Cost function for logistic regression

- Consider a classification problem with two labels $y \in \{0, 1\}$.
- To understand the cost function of the neural network, we first derive the cost function for logistic regression using the maximum likelihood estimator.
- Let us denote

$$\begin{aligned} p\left(y = 1 \mid x; \theta\right) &= h_\theta\left(x\right) \\ p\left(y = 0 \mid x; \theta\right) &= 1 - h_\theta\left(x\right) \end{aligned}$$

- Note that this formula can be written compactly as

$$p\left(y \mid x; \theta\right) = \left(h_\theta\left(x\right)\right)^y \left(1 - h_\theta\left(x\right)\right)^{1-y}$$

- Indeed if $y = 1$, then $p\left(y \mid x; \theta\right) = \left(h_\theta\left(x\right)\right)^1 \left(1 - h_\theta\left(x\right)\right)^0$
- Indeed if $y = 0$, then $p\left(y \mid x; \theta\right) = \left(h_\theta\left(x\right)\right)^0 \left(1 - h_\theta\left(x\right)\right)^1$

Cost function for logistic regression (cont.)

- Assume that n samples are generated independently, we get

$$\begin{aligned} L(\theta) &= p(\overline{\mathbf{y}} \mid \overline{\mathbf{x}}; \theta) = \prod_{i=1}^{n} p\left(y^{(i)} \mid x^{(i)}; \theta\right) \\ &= \prod_{i=1}^{n} \left(h_\theta\left(x^{(i)}\right)\right)^{y^{(i)}} \left(1 - h_\theta\left(x^{(i)}\right)\right)^{1-y^{(i)}} \end{aligned}$$

- Taking the log to simplify calculations, we get

$$\begin{aligned} \ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{n} y^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right) \end{aligned}$$

Cost Function: logistic regression versus neural network

Logistic Regression:

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log h_{\boldsymbol{\theta}}(\mathbf{x}_i) + (1-y_i)\log(1-h_{\boldsymbol{\theta}}(\mathbf{x}_i))] + \frac{\lambda}{2n}\sum_{j=1}^{d}\theta_j^2$$

Neural Network:

$$h_\Theta \in \mathbb{R}^K \qquad (h_\Theta(\mathbf{x}))_i = i^{th}\text{output}$$

$$J(\Theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik}\log(h_\Theta(\mathbf{x}_i))_k + (1-y_{ik})\log\left(1-(h_\Theta(\mathbf{x}_i))_k\right)\right]$$
$$+ \frac{\lambda}{2n}\sum_{l=1}^{L-1}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(\Theta_{ji}^{(l)}\right)^2$$

k^{th} class: true, predicted
not k^{th} class: true, predicted

Optimizing the Neural Network

$$J(\Theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik}\log(h_\Theta(\mathbf{x}_i))_k + (1-y_{ik})\log\Big(1-(h_\Theta(\mathbf{x}_i))_k\Big)\right]$$
$$+\frac{\lambda}{2n}\sum_{l=1}^{L-1}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(\Theta_{ji}^{(l)}\right)^2$$

Solve via: $\min_\Theta J(\Theta)$

$J(\Theta)$ is not convex, so GD on a neural net yields a local optimum

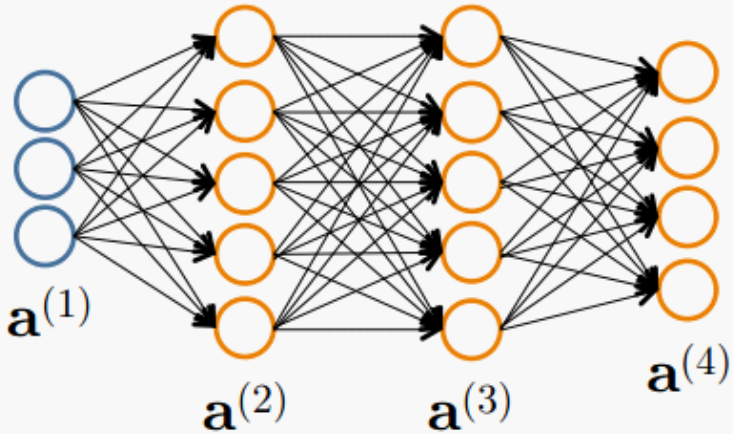
- But, tends to work well in practice

Need code to compute:

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

Forward Propagation

- Given one labeled training instance $(\mathbf{x}, y)$:



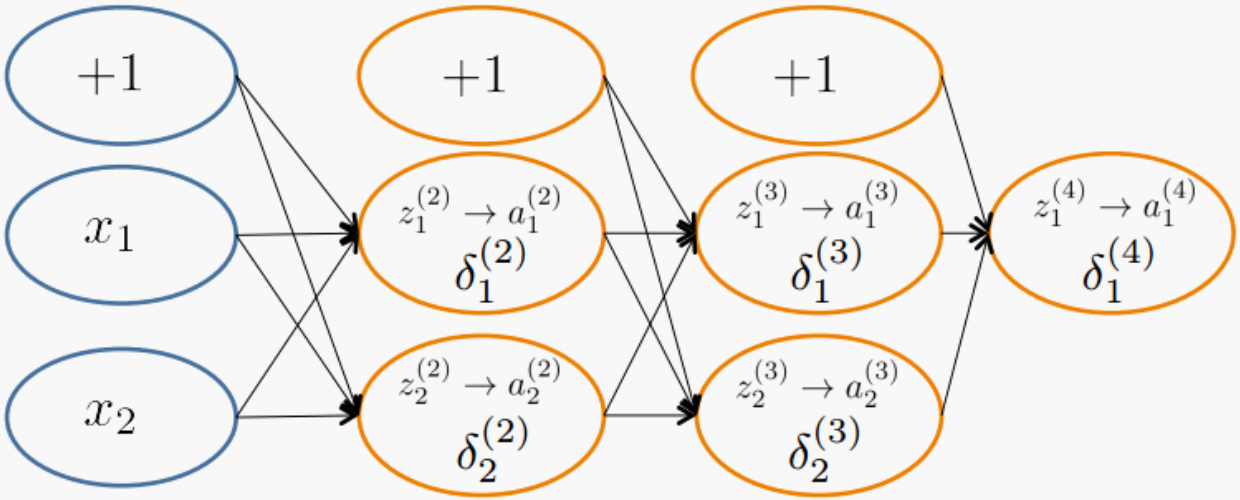
Forward Propagation

- $\mathbf{a}^{(1)} = \mathbf{x}$
- $\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{a}^{(1)}$
- $\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$ [add $a_0^{(2)}$]
- $\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$
- $\mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$ [add $a_0^{(3)}$]
- $\mathbf{z}^{(4)} = \Theta^{(3)}\mathbf{a}^{(3)}$
- $\mathbf{a}^{(4)} = h_\Theta(\mathbf{x}) = g(\mathbf{z}^{(4)})$

Backpropagation Intuition

- Each hidden node j is "responsible" for some fraction of the error $\delta_j^{(l)}$ in each of the output nodes j to which it connects
- $\delta_j^{(l)}$ is divided according to the strength of the j connection between hidden node and the output node
- Then, the "blame" is propagated back to provide the error values for the hidden layer.

Backpropagation Intuition

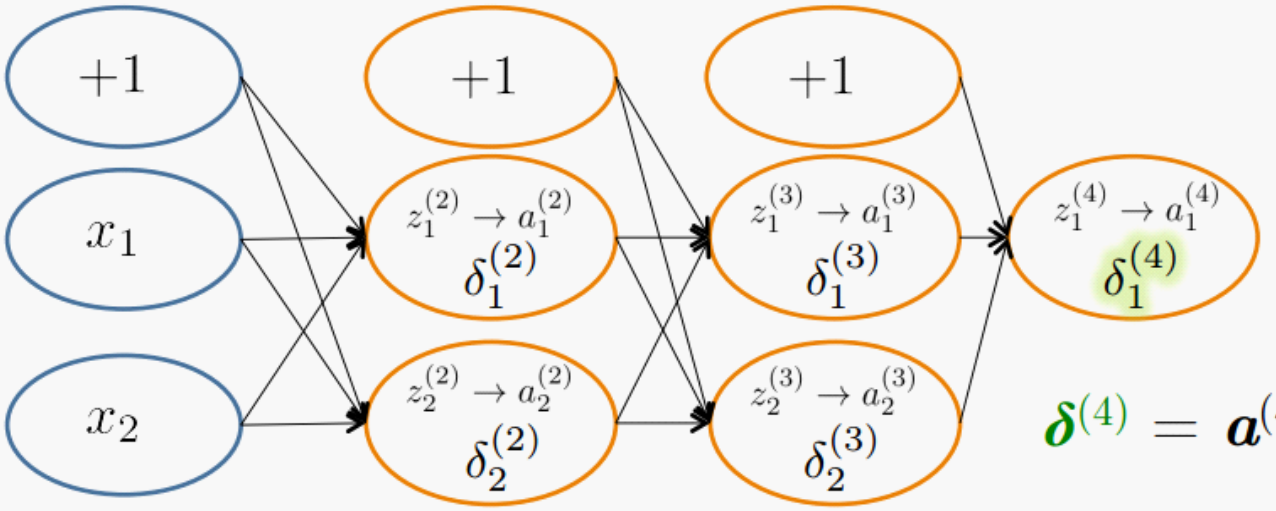


$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Backpropagation Intuition



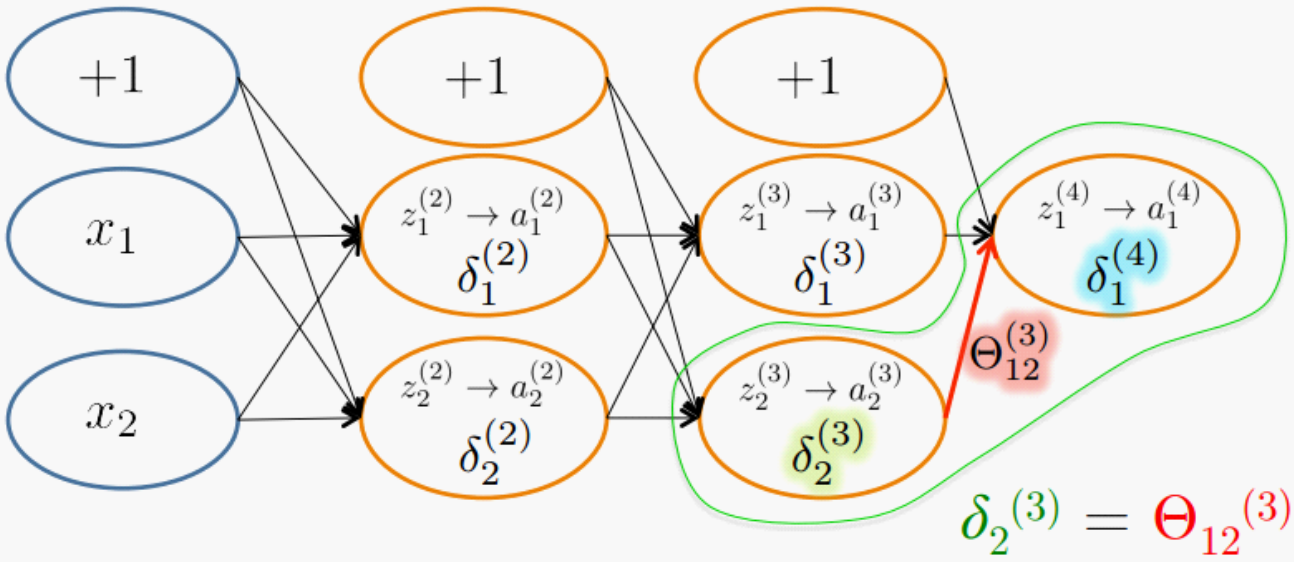
$$\boldsymbol{\delta}^{(4)} = \boldsymbol{a}^{(4)} - \mathbf{y}$$

$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Backpropagation Intuition

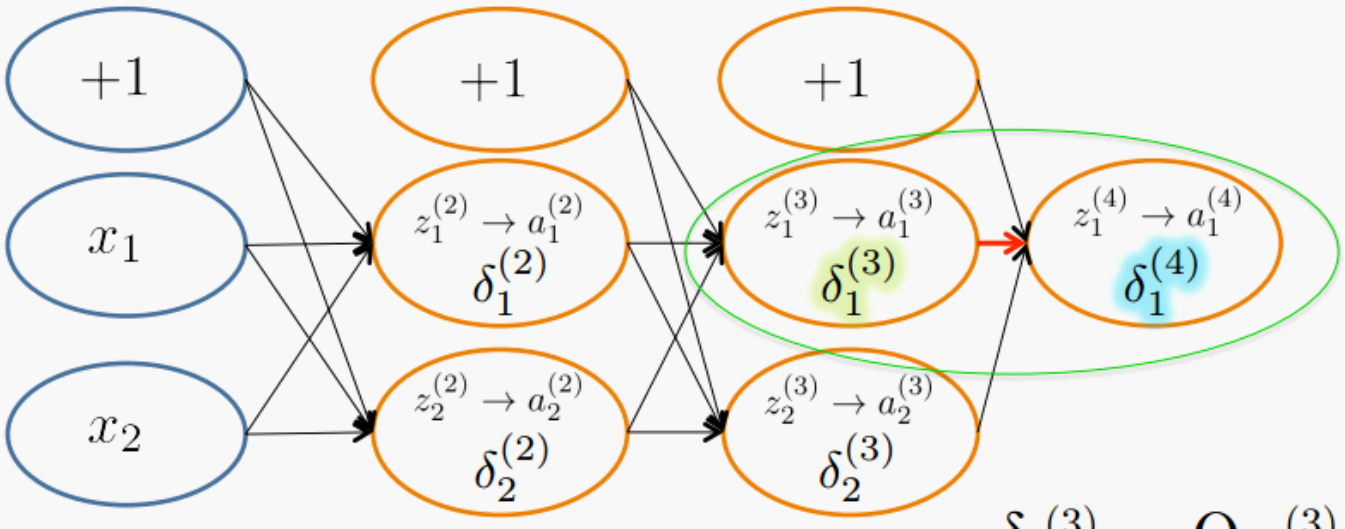


$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Backpropagation Intuition



$$\delta_2^{(3)} = \Theta_{12}^{(3)} \times \delta_1^{(4)}$$

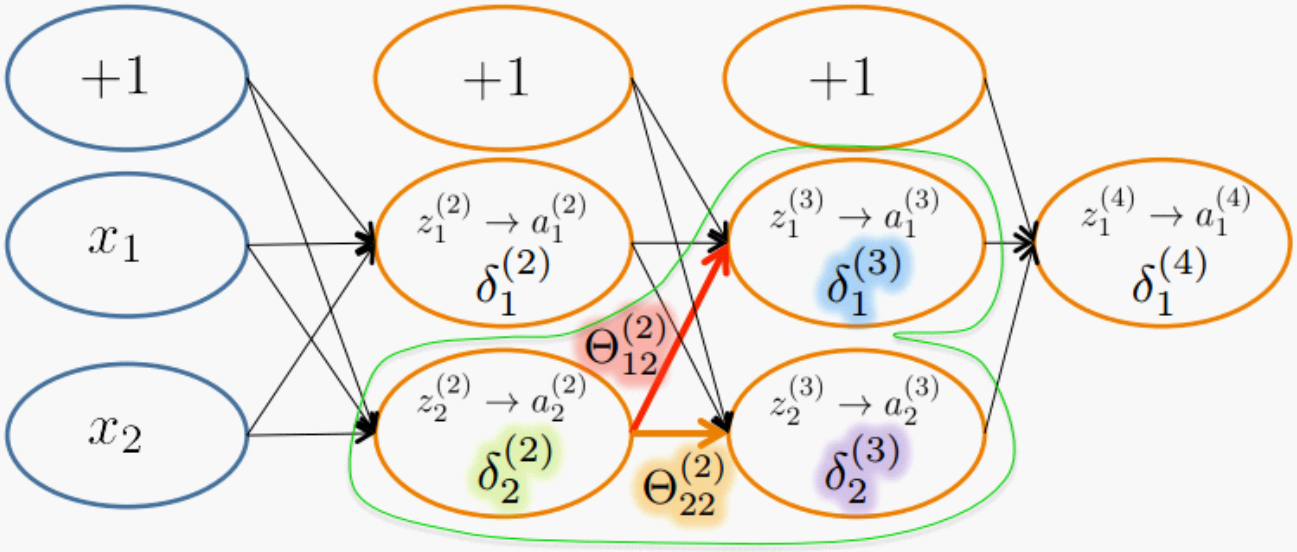
$$\delta_1^{(3)} = \Theta_{11}^{(3)} \times \delta_1^{(4)}$$

$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Backpropagation Intuition



$$\delta_2^{(2)} = \Theta_{12}^{(2)} \times \delta_1^{(3)} + \Theta_{22}^{(2)} \times \delta_2^{(3)}$$

$\delta_j^{(l)}$ = "error" of node j in layer l

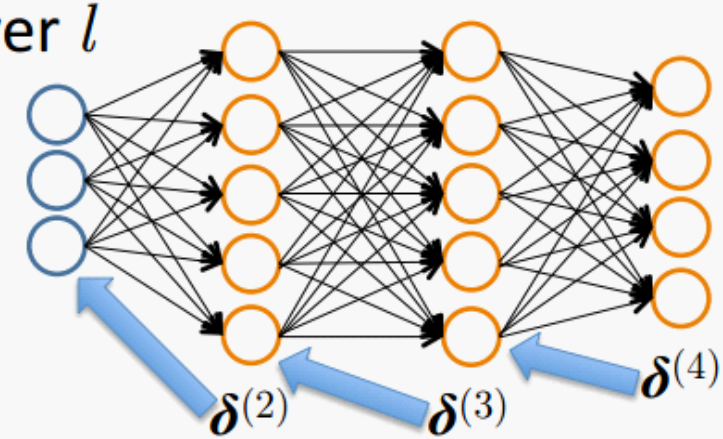
Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Backpropagation: Gradient Computation

Let $\delta_j^{(l)}$ = "error" of node j in layer l

(#layers $L = 4$)



Element-wise product .*

Backpropagation

- $\boldsymbol{\delta}^{(4)} = \boldsymbol{a}^{(4)} - \mathbf{y}$
- $\boldsymbol{\delta}^{(3)} = (\Theta^{(3)})^{\mathsf{T}}\boldsymbol{\delta}^{(4)}$.* $g'(\mathbf{z}^{(3)})$
- $\boldsymbol{\delta}^{(2)} = (\Theta^{(2)})^{\mathsf{T}}\boldsymbol{\delta}^{(3)}$.* $g'(\mathbf{z}^{(2)})$
- (No $\boldsymbol{\delta}^{(1)}$)

$$g'(\mathbf{z}^{(3)}) = \mathbf{a}^{(3)} .* (1–\mathbf{a}^{(3)})$$

$$g'(\mathbf{z}^{(2)}) = \mathbf{a}^{(2)} .* (1–\mathbf{a}^{(2)})$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

(ignoring λ; if $\lambda = 0$)

Training a Neural Network via Gradient Descent with Backprop

Given: training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$
Initialize all $\Theta^{(l)}$ randomly (NOT to 0!)
Loop // each iteration is called an epoch

Set $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$ (Used to accumulate gradient)

For each training instance $(\mathbf{x}_i, y_i)$:

Set $\mathbf{a}^{(1)} = \mathbf{x}_i$

Compute $\{\mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(L)}\}$ via forward propagation

Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_i$

Compute errors $\{\boldsymbol{\delta}^{(L-1)}, \ldots, \boldsymbol{\delta}^{(2)}\}$

Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

Compute avg regularized gradient $D_{ij}^{(l)} = \begin{cases} \frac{1}{n}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{n}\Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$

Update weights via gradient step $\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha D_{ij}^{(l)}$

Until weights converge or max #epochs is reached

Backpropagation

Backprop Issues

"Backprop is the cockroach of machine learning. It's ugly, and annoying, but you just can't get rid of it."

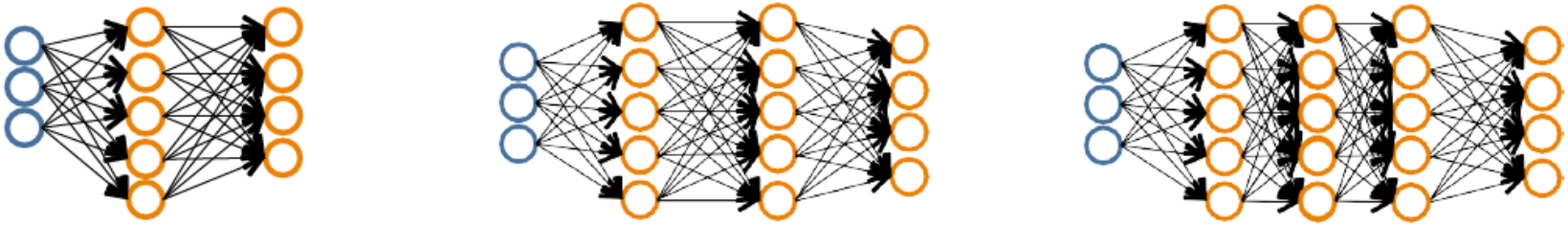
- Geoff Hinton

Problems:

– black box
– local minima

Choosing a neural network topology

Pick a network architecture (connectivity pattern between nodes)



- # input units = # of features in dataset
- # output units = # classes

Reasonable default: 1 hidden layer

- or if >1 hidden layer, have same # hidden units in every layer (usually the more the better)

Putting all together

Training a neural network

1. Randomly initialize weights
2. Implement forward propagation to get $h_\Theta(\mathbf{x}_i)$ for any instance $\mathbf{x}_i$
3. Implement code to compute cost function $J(\Theta)$
4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$
5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backpropagation vs. the numerical gradient estimate.
   - ✓ Then, disable gradient checking code
6. Use gradient descent with backprop to fit the network

Materials

For preparing this content, we used multiple sources including:

- Goodfellow, I., Y. Bengio and A. Courville, (2016). Deep Learning. The MIT Press.
- Judd, K. (1998). Numerical Methods in Economics. The MIT Press.
- Maliar, L. and S. Maliar, (2014). "Numerical Methods for Large Scale Dynamic Economic Models" in: Schmedders, K. and K. Judd (Eds.), Handbook of Computational Economics, Volume 3, Chapter 7, Amsterdam: Elsevier Science.
- Kevin P. Murphy (2012). Machine Learning: A Probabilistic Perspective. The MIT press.
- Andrew Ng (2019). Machine learning (online).
- https://developers.google.com/machine-learning/crash-course/framing/ml-terminology
- https://en.wikipedia.org/wiki
- https://www.geeksforgeeks.org
- https://medium.com
- https://www.jeremyjordan.me/nn-learning-rate/