STA 141C - Big Data & High Performance Statistical Computing

Week 8-2: Neural Networks

Instructor: Bo Y.-C. Ning

May 19, 2022

Announcement

Last time

- Gradient descent

Today

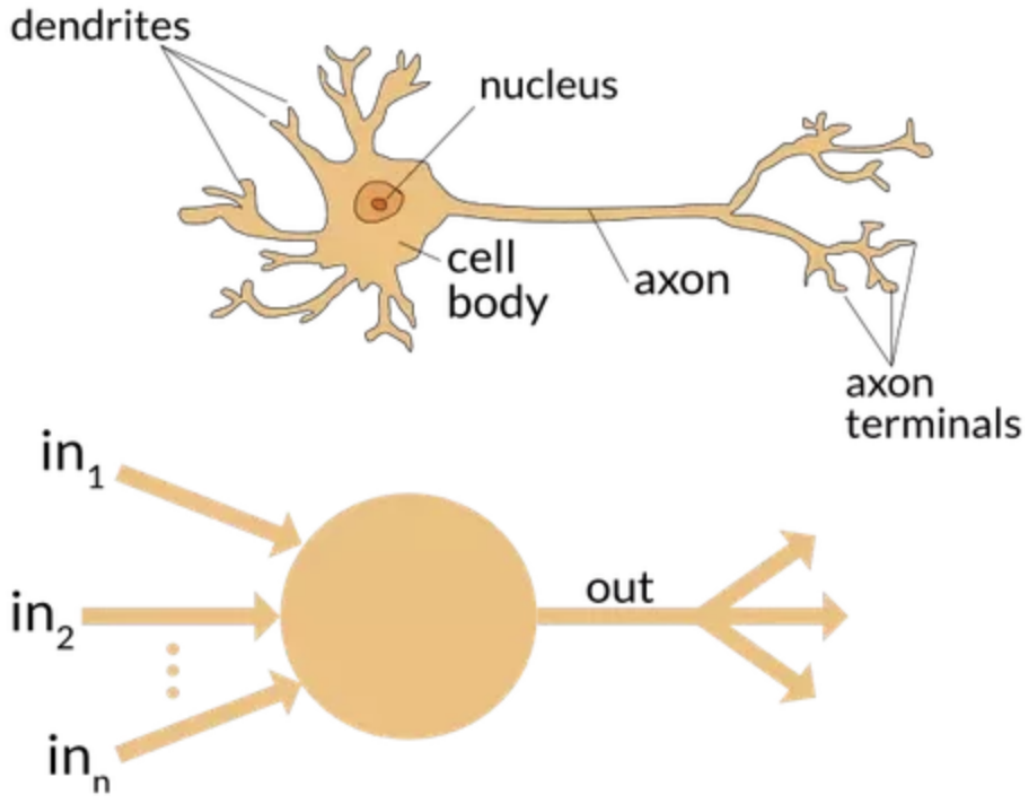
- Neural Networks
- Stochastic gradient descent

References

- Prof. Cho-Jui Hsieh's old lecture note on neuron networks

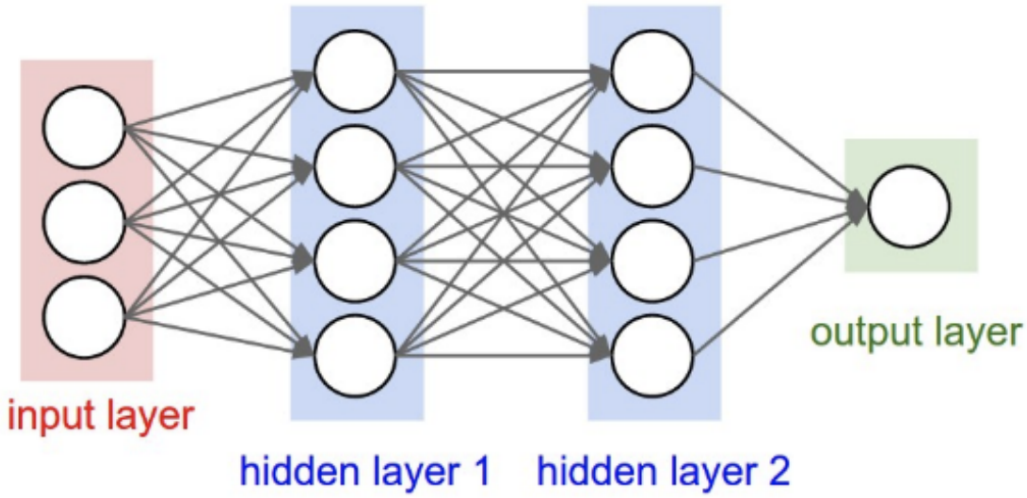
What is neural networks?

Artificial neural networks (ANNs), usually simply called neural networks (NNs), are computing systems inspired by the biological neural networks that constitute animal brains.



In mathematics, a neural network is a function which is generally comprised of

- Neurons: which pass input values through functions and output the result
- Weights: which carry values between neurons



Neurons are grouped into three layers:

- Input layer: d features $x_1, \ldots, x_d$
- Hidden layer(s): $L - 1$
- Output layer: $h(x)$

For a N-layer neural network, it consists $N - 1$ hidden layer and the output layer.

History of NNs

Table 1: Major milestones that will be covered in this paper

Year	Contributer	Contribution
300 BC	Aristotle	introduced Associationism, started the history of human's attempt to understand brain.
1873	Alexander Bain	introduced Neural Groupings as the earliest models of neural network, inspired Hebbian Learning Rule.
1943	McCulloch & Pitts	introduced MCP Model, which is considered as the ancestor of Artificial Neural Model.
1949	Donald Hebb	considered as the father of neural networks, introduced Hebbian Learning Rule, which lays the foundation of modern neural network.
1958	Frank Rosenblatt	introduced the first perceptron, which highly resembles modern perceptron.
1974	Paul Werbos	introduced Backpropagation
1980	Teuvo Kohonen	introduced Self Organizing Map
	Kunihiko Fukushima	introduced Neocogitron, which inspired Convolutional Neural Network
1982	John Hopfield	introduced Hopfield Network
1985	Hilton & Sejnowski	introduced Boltzmann Machine
1986	Paul Smolensky	introduced Harmonium, which is later known as Restricted Boltzmann Machine
	Michael I. Jordan	defined and introduced Recurrent Neural Network
1990	Yann LeCun	introduced LeNet, showed the possibility of deep neural networks in practice
1997	Schuster & Paliwal	introduced Bidirectional Recurrent Neural Network
	Hochreiter & Schmidhuber	introduced LSTM, solved the problem of vanishing gradient in recurrent neural networks
2006	Geoffrey Hinton	introduced Deep Belief Networks, also introduced layer-wise pretraining technique, opened current deep learning era.
2009	Salakhutdinov & Hinton	introduced Deep Boltzmann Machines
2012	Geoffrey Hinton	introduced Dropout, an efficient way of training neural networks

The current AI wave came in 2012 when AlexNet (60 million parameters) cuts the error rate of ImageNet competition (classify 1.2 million natural images) by half.

[Source: Wang and Raj, 2017. *"On the Origin of Deep Learning"*, arXiv:1702.07800.]

NNs: single layer

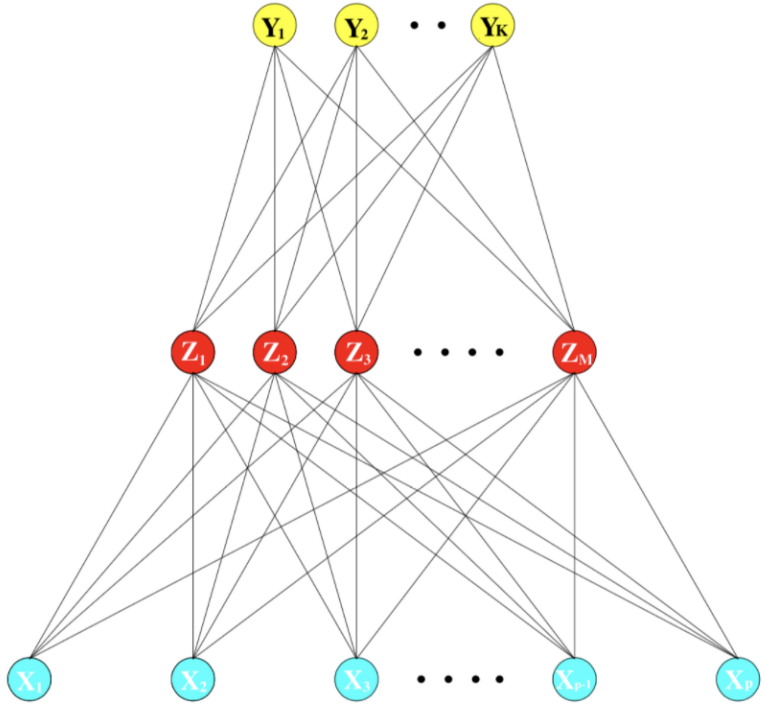


FIGURE 11.2. *Schematic of a single hidden layer, feed-forward neural network.*

- Output layer: $Y = (Y_1, \ldots, Y_k)$ are k-dimensional output; e.g., in classification, each corresponds to each class in the classification
- Input layer: $X = (X_1, \ldots, X_p)$ are p-dimensional input features
- Hidden layer: $Z = (Z_1, \ldots, Z_M)$ are the derived features created from linear combinations of inputs X.

Mathematical model for one hidden layer NNs

Weights:

$$w_{ij}^{(l)} : \begin{cases} 1 \leq l \leq 2 \text{ (layer)} \\ 0 \leq i \leq d^{(l-1)} \text{ (inputs)} \\ 1 \leq j \leq d^{(l)} \text{ (outputs)} \end{cases}$$

Then, j-th neuron in the l-th layer is

$$x_j^{(l)} = \theta\left(s_j^{(l)}\right) = \theta\left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}\right).$$

The output is

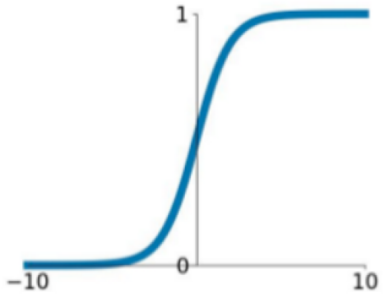
$$h(X) = X^{(L)}$$

$\theta(\cdot)$ is known as the activation function.

Activation function

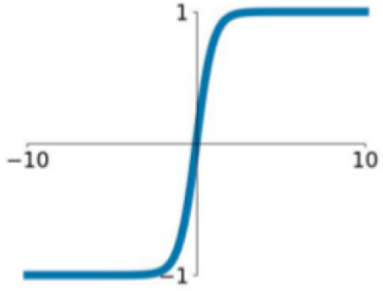
Sigmoid

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



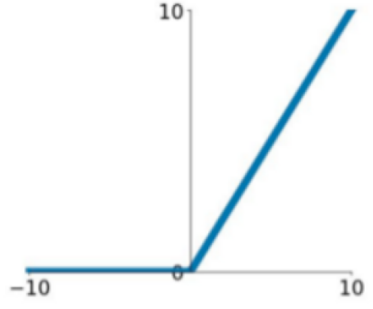
tanh

$$\tanh(x)$$

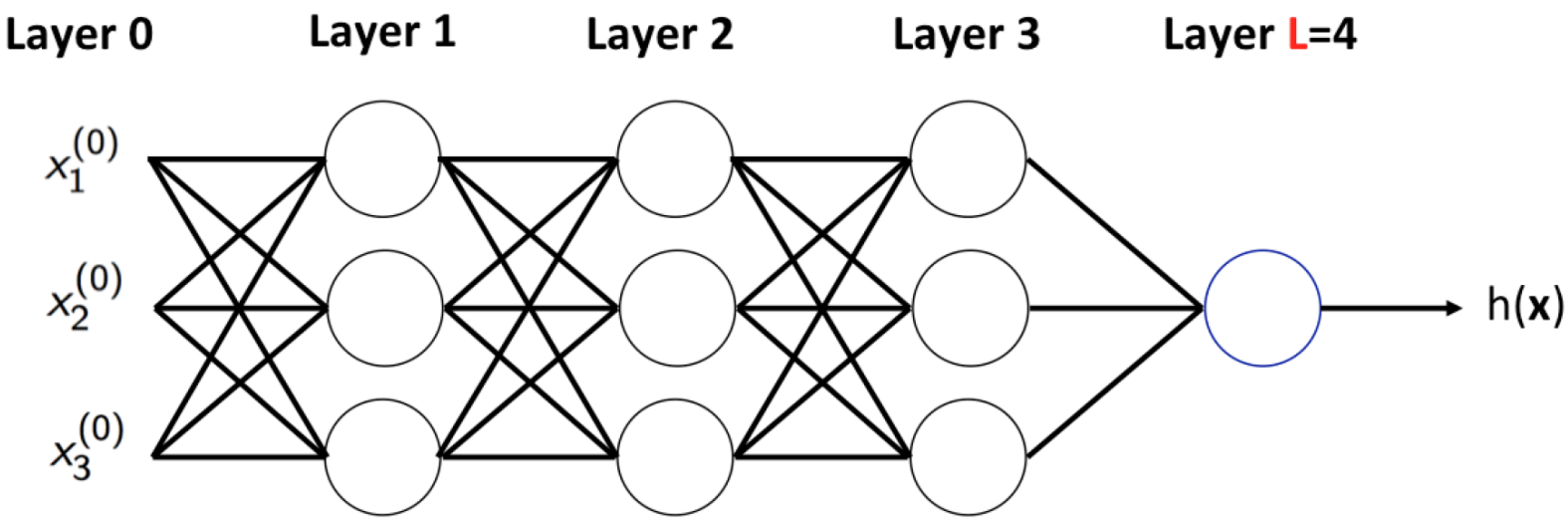


ReLU

$$\max(0, x)$$



L-layer NNs



Mathematical model for L-layer NNs

Weights:

$$w_{ij}^{(l)} : \begin{cases} 1 \leq l \leq L \text{ (layer)} \\ 0 \leq i \leq d^{(l-1)} \text{ (inputs)} \\ 1 \leq j \leq d^{(l)} \text{ (outputs)} \end{cases}$$

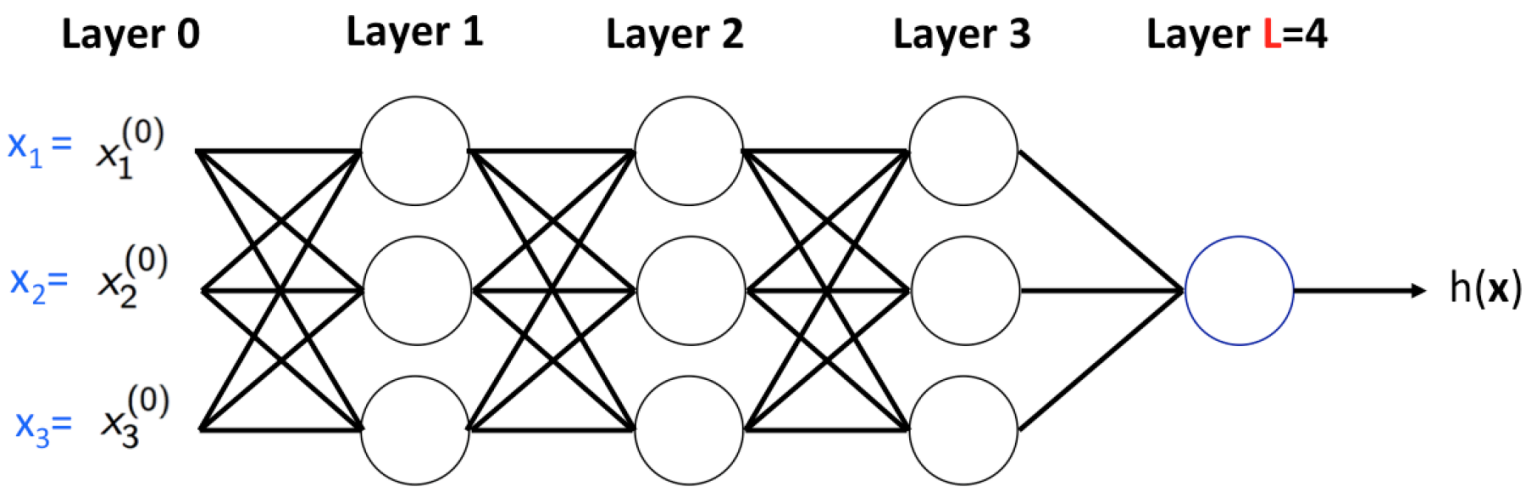
Then, j-th neuron in the l-th layer is

$$x_j^{(l)} = \theta\left(s_j^{(l)}\right) = \theta\left(\sum_{i=0}^{d^{l-1}} w_{ij}^{(l)} x_i^{(l-1)}\right).$$

The output is

$$h(x) = x_1^{(L)}$$

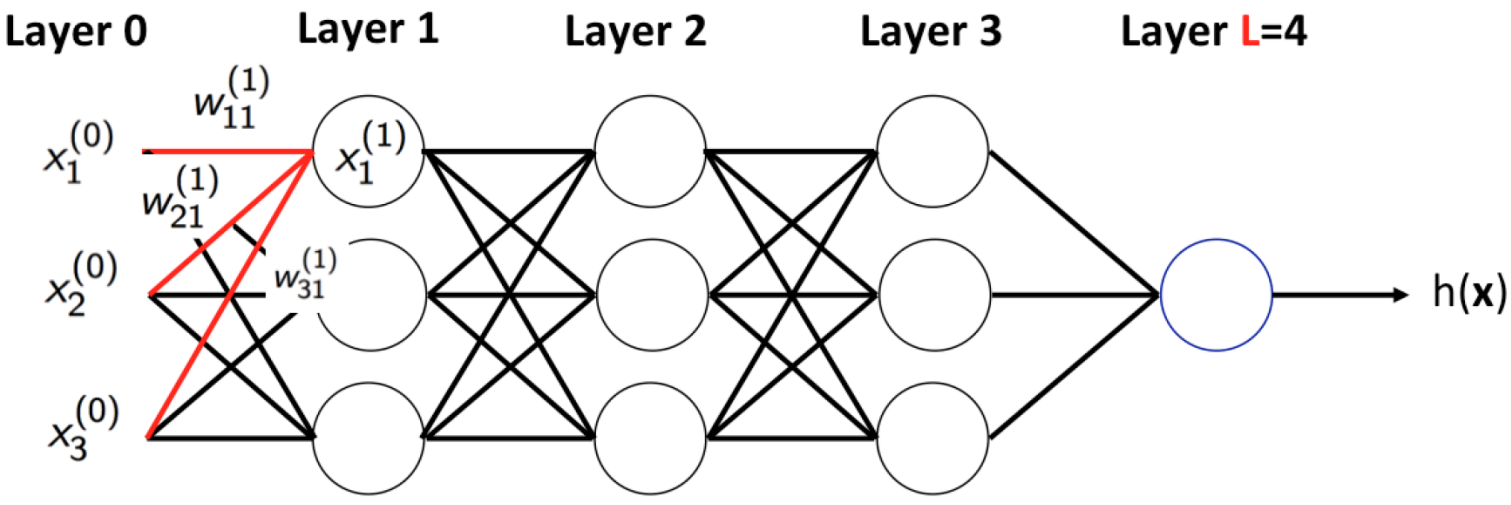
Forward propagation



features for one data point

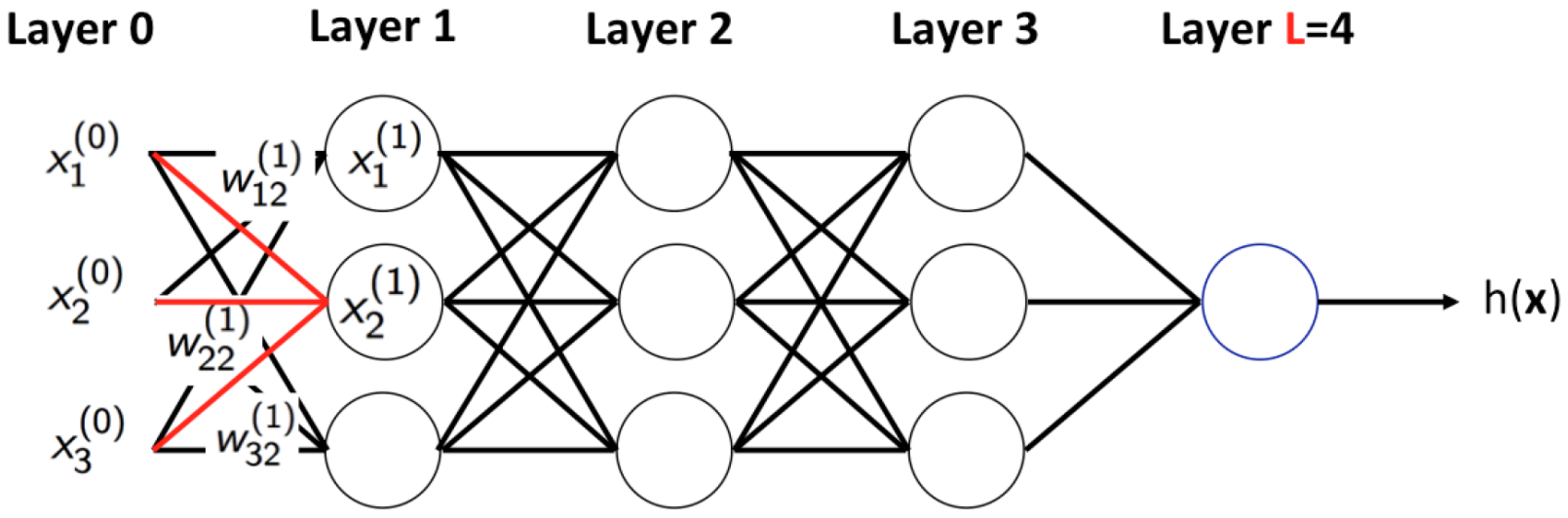
x = $[x_1, x_2, x_3]$

Forward propagation (cont'd)



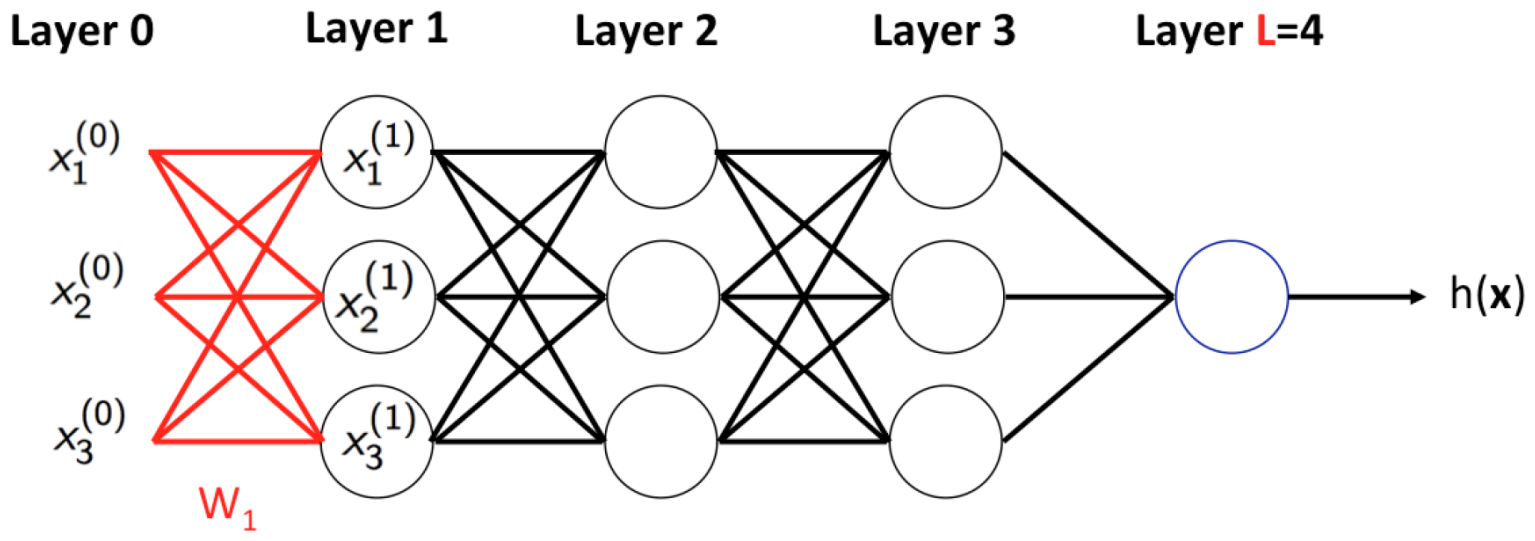
$$x_1^{(1)} = \theta\left(\sum_{i=1}^{3} w_{i1}^{(1)} x_i^{(0)}\right)$$

Forward propagation (cont'd)



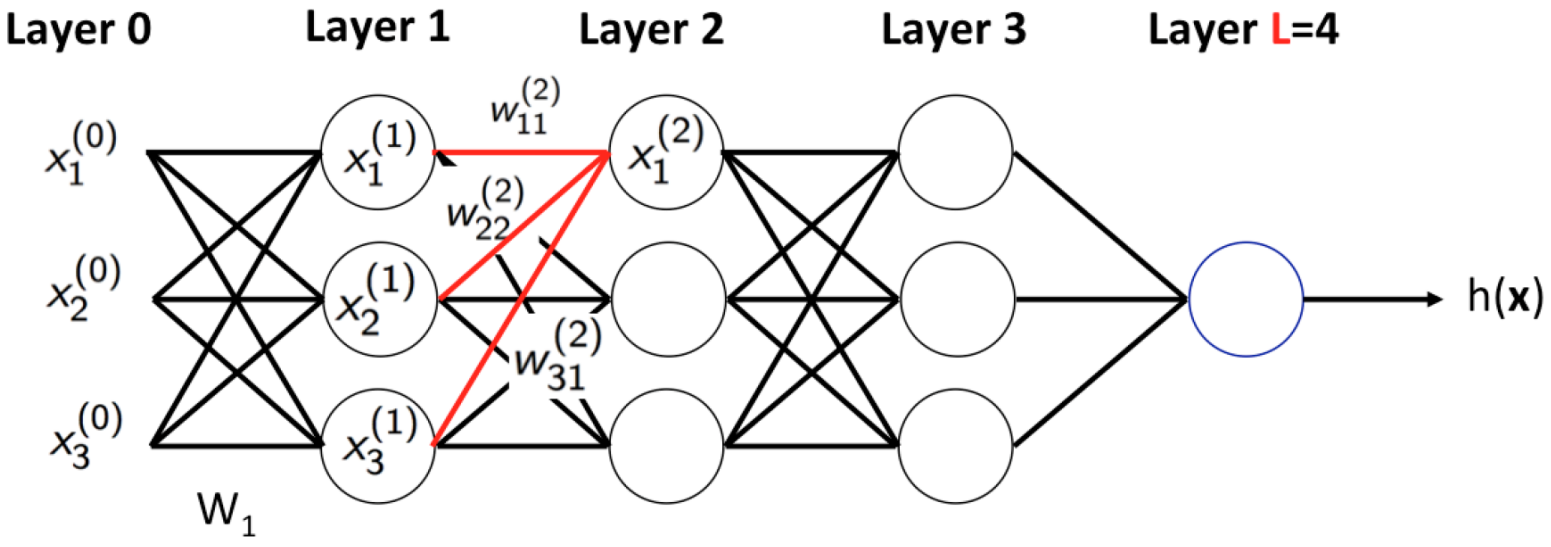
$$x_2^{(1)} = \theta\left(\sum_{i=1}^{3} w_{i2}^{(1)} x_i^{(0)}\right)$$

Forward propagation (cont'd)



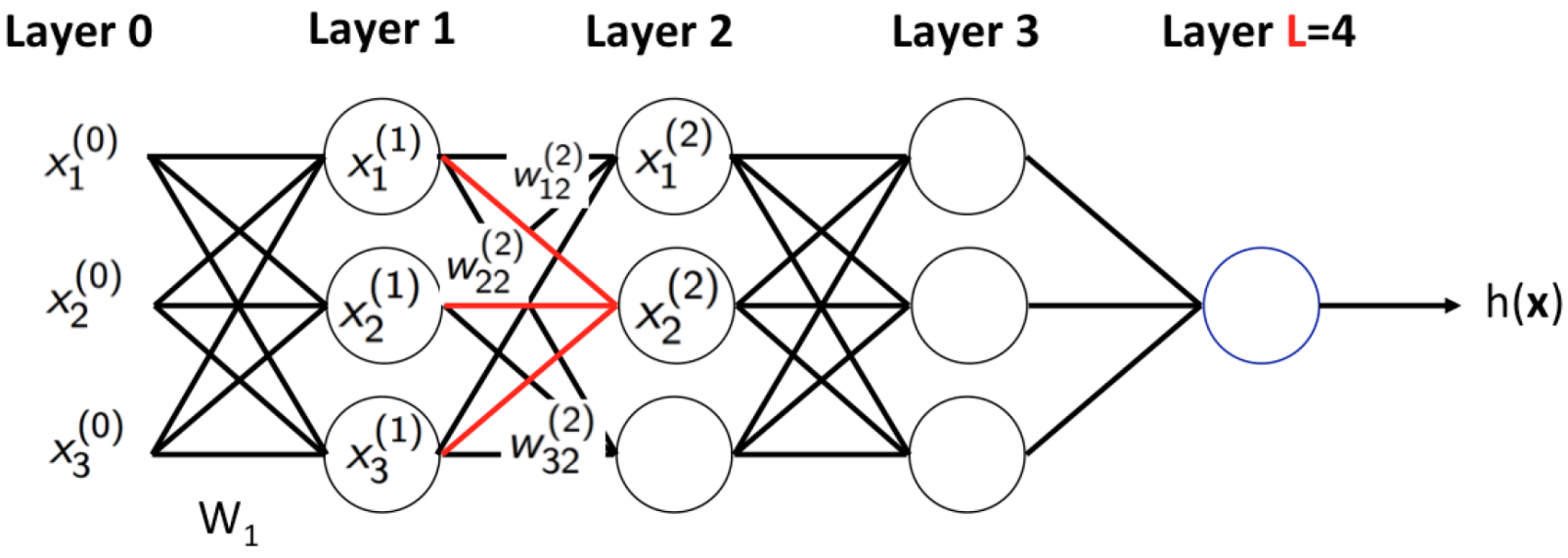
$$\boldsymbol{x}^{(1)} = \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} = \theta \left(\begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} \\ w_{13}^{(1)} & w_{23}^{(1)} & w_{33}^{(1)} \end{bmatrix} \times \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \\ x_3^{(0)} \end{bmatrix} \right) = \theta(W_1 \boldsymbol{x}^{(0)})$$

Forward propagation (cont'd)



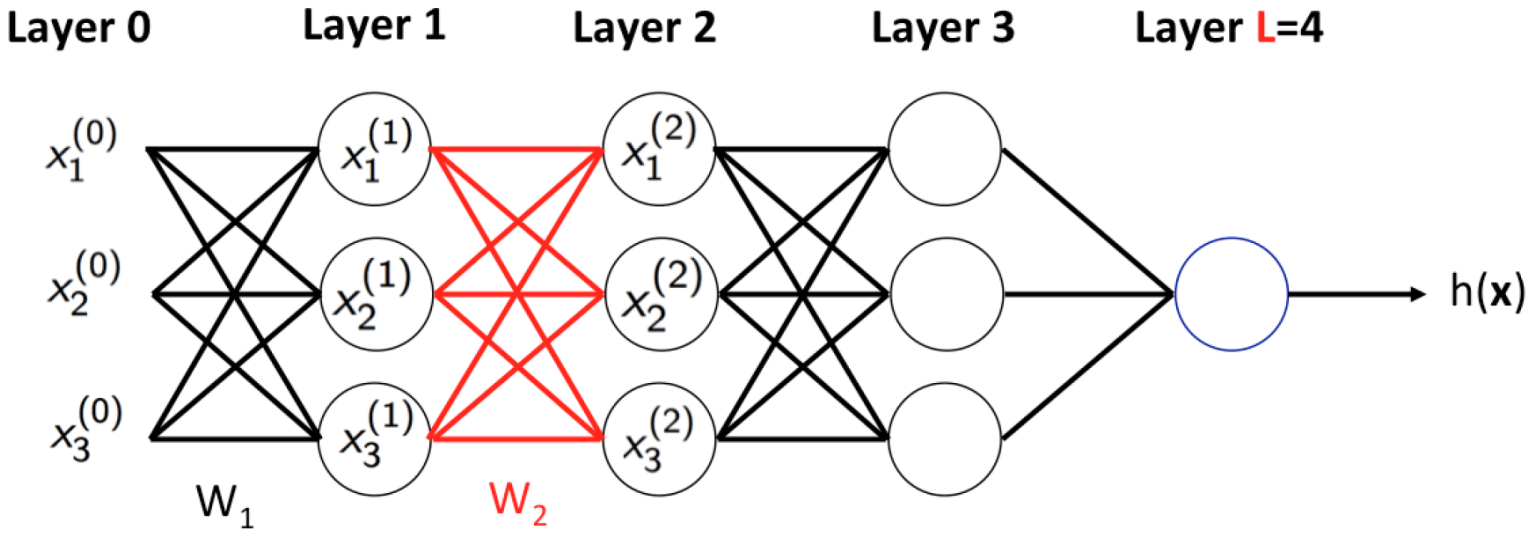
$$x_1^{(2)} = \theta\left(\sum_{i=1}^{3} w_{i1}^{(2)} x_i^{(1)}\right)$$

Forward propagation (cont'd)



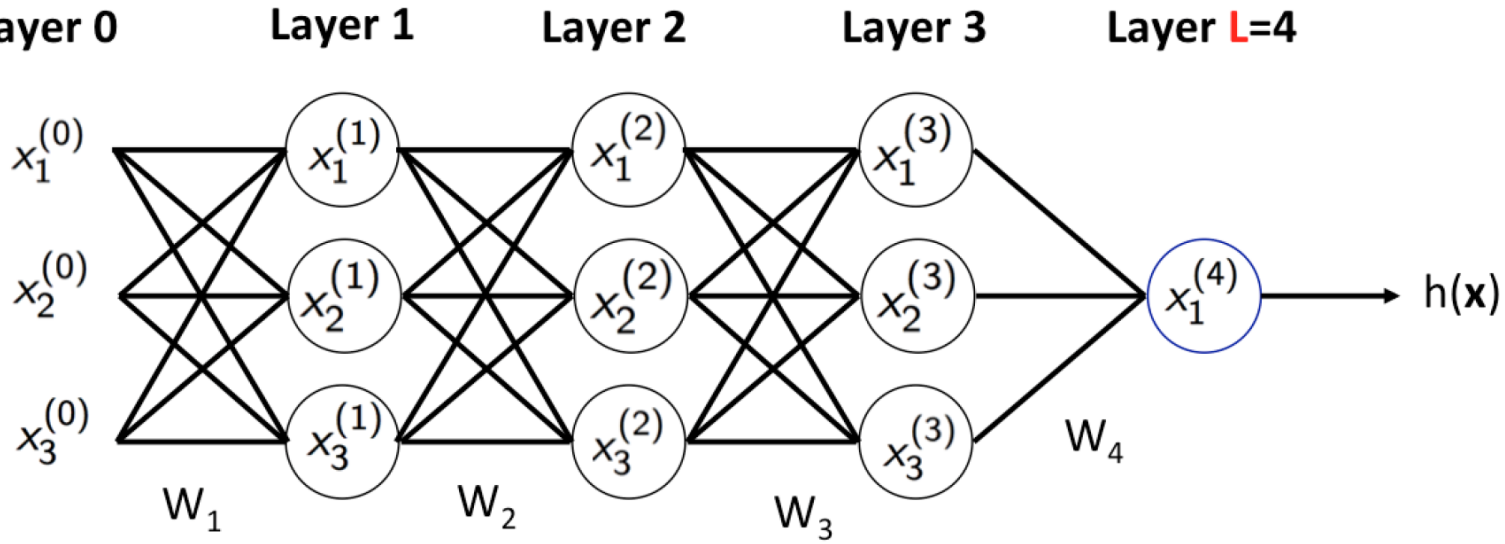
$$x_2^{(2)} = \theta\left(\sum_{i=1}^{3} w_{i2}^{(2)} x_i^{(1)}\right)$$

Forward propagation (cont'd)



$$\boldsymbol{x}^{(2)} = \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \end{bmatrix} = \theta\left(\begin{bmatrix} w_{11}^{(2)} & w_{21}^{(2)} & w_{31}^{(2)} \\ w_{12}^{(2)} & w_{22}^{(2)} & w_{32}^{(2)} \\ w_{13}^{(2)} & w_{23}^{(2)} & w_{33}^{(2)} \end{bmatrix} \times \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} \right) = \theta(W_2 \boldsymbol{x}^{(1)})$$

Forward propagation (cont'd)



$$h(\boldsymbol{x}) = x_1^{(4)} = \theta(W_4 \boldsymbol{x}^{(3)}) = \theta(W_4 \theta(W_3 \boldsymbol{x}^{(2)}))$$
$$= \cdots = \theta(W_4 \theta(W_3 \theta(W_2 \theta(W_1 \boldsymbol{x}))))$$

Unknown parameter: $W = (W_1, \ldots, W_L)$

Goal:

$$\min_W \frac{1}{N} \sum_{i=1}^{N} e(h(x_i), y_i),$$

where $e(\cdot, \cdot)$ is the loss function

How to solve W?

Pick a loss function $e(W) = e(\cdot, \cdot)$:

Quadratic loss (sum of square errors):

$$e(W) = \sum_{i}^{N} (h(x_i) - y_i)^2$$

Cross-entropy (deviance):

$$e(W) = \sum_{i}^{N} y_i \log h(x_i)$$

- Method 1: gradient descent (GD); how?
- Method 2: stochastic gradient descent (SGD)

Stochastic gradient descent (SGD)

- SGD is similar to GD.
- For each iteration, pick a subset of $(x_1, y_1), \ldots, (x_N, y_N)$ randomly, get $(x_1, y_1), \ldots, (x_n, y_n)$
- Update

$$W \longleftarrow W - s \times \bigtriangledown e(h(x_n), y_n)$$

- We need to calculate the gradient $\bigtriangledown e(h(x_n), y_n)$:

  For each i, j, l, the gradient is

$$\frac{\partial e(h(x_n), y_n)}{\partial w_{ij}^{(l)}}.$$

Computing the gradient

Apply chain rule

$$\frac{\partial e(W)}{\partial W_{ij}^{(l)}} = \frac{\partial e(W)}{\partial s_j^{(l)}} \times \frac{\partial s_j^{(l)}}{\partial W_{ij}^{(l)}}$$

$$s_j^{(l)} = \sum_{i=1}^{d} x_i^{(l-1)} w_{ij}^{(l)}$$

Thus, we have

$$\frac{\partial s_j^{(l)}}{\partial W_{ij}^{(l)}} = x_i^{(l-1)}.$$

Computing the gradient (cont'd)

We now compute $\dfrac{\partial e(W)}{\partial s_j^{(l)}}$.

- Define $\delta_j^{(l)} = \frac{\partial e(W)}{\partial s_j^{(l)}}$
- Compute it layer-by-layer:

$$\delta_j^{(l-1)} = \frac{\partial e(W)}{\partial s_j^{(l-1)}} = \sum_{j=1}^{d} \frac{\partial e(W)}{\partial s_j^{(l)}} \times \frac{\partial s_j^{(l)}}{\partial x_i^{(l-1)}} \times \frac{\partial x_i^{(l-1)}}{\partial s_i^{(l-1)}}$$

$$= \sum_{i=1}^{d} \delta_j^{(l)} \times w_{ij}^{(l)} \times \theta'\left(s_i^{(l-1)}\right),$$

  where $\theta(s)$ is the activation function. For example $\theta'(s) = 1 - \theta^2(s)$ if θ is tanh.

- For example $\theta'(s) = 1 - \theta^2(s)$ if θ is tanh. Then

$$\delta_j^{(l-1)} = \left(1 - (x_i^{(l-1)})^2\right) \sum_{j=1}^{d} w_{ij}^{(l)} \delta_j^{(l)}.$$

Final layer

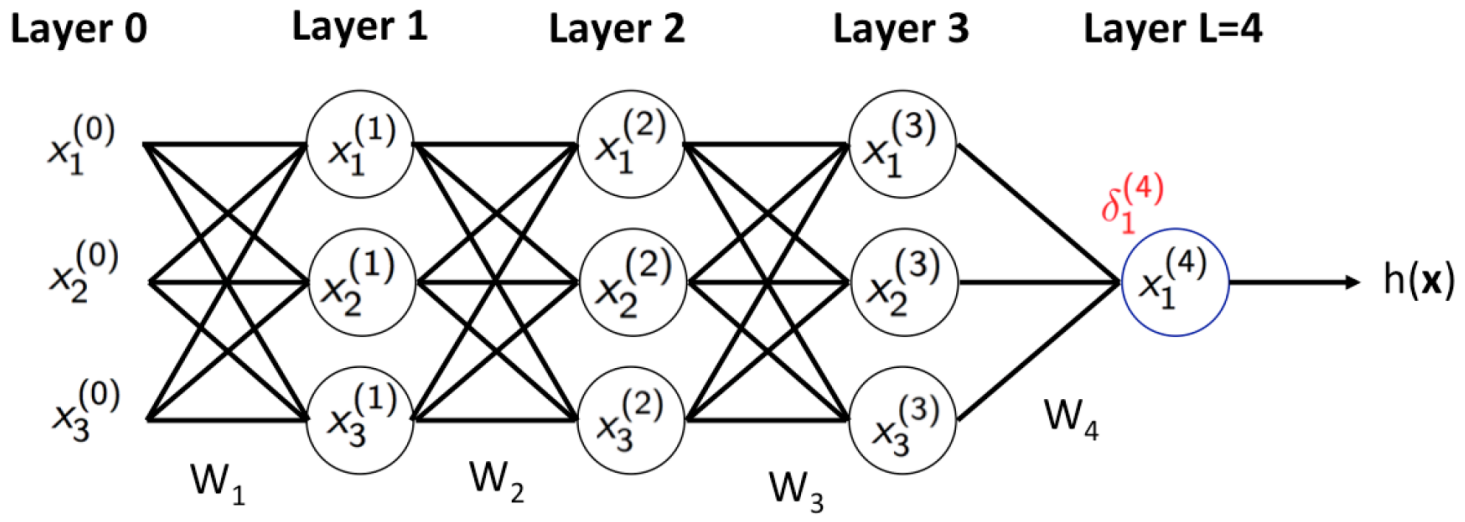
In the final layer, $x_1^{(L)} = \theta(s_1^L)$, therefore,

$$\begin{aligned}\delta_1^{(L)} &= \frac{\partial e(W)}{\partial s_1^{(L)}} \\ &= \frac{\partial e(W)}{\partial x_1^{(L)}} \times \frac{\partial x_1^{(L)}}{\partial s_1^{(L)}} \\ &= \frac{\partial e(W)}{\partial x_1^{(L)}} \times \theta'(s_1^{(L)}).\end{aligned}$$

Assume $e(W) = (x_1^{(L)} - y_n)^2$, then

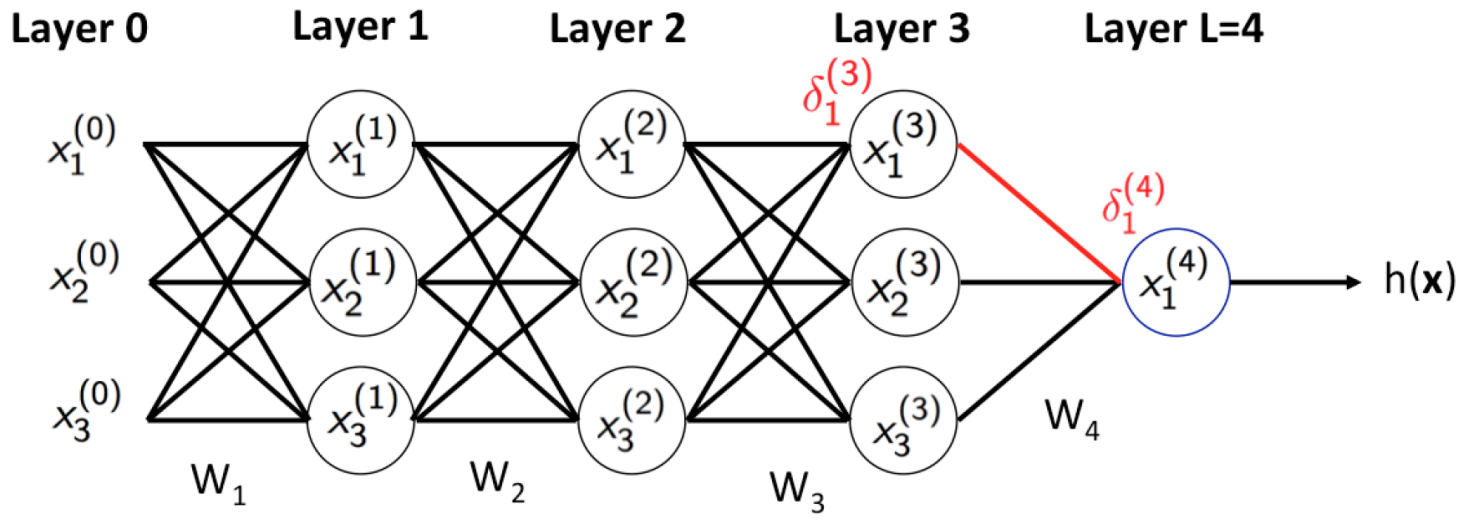
$$\frac{\partial e(W)}{\partial x_1^{(L)}} = 2(x_1^{(L)} - y_n).$$

Backward propagation



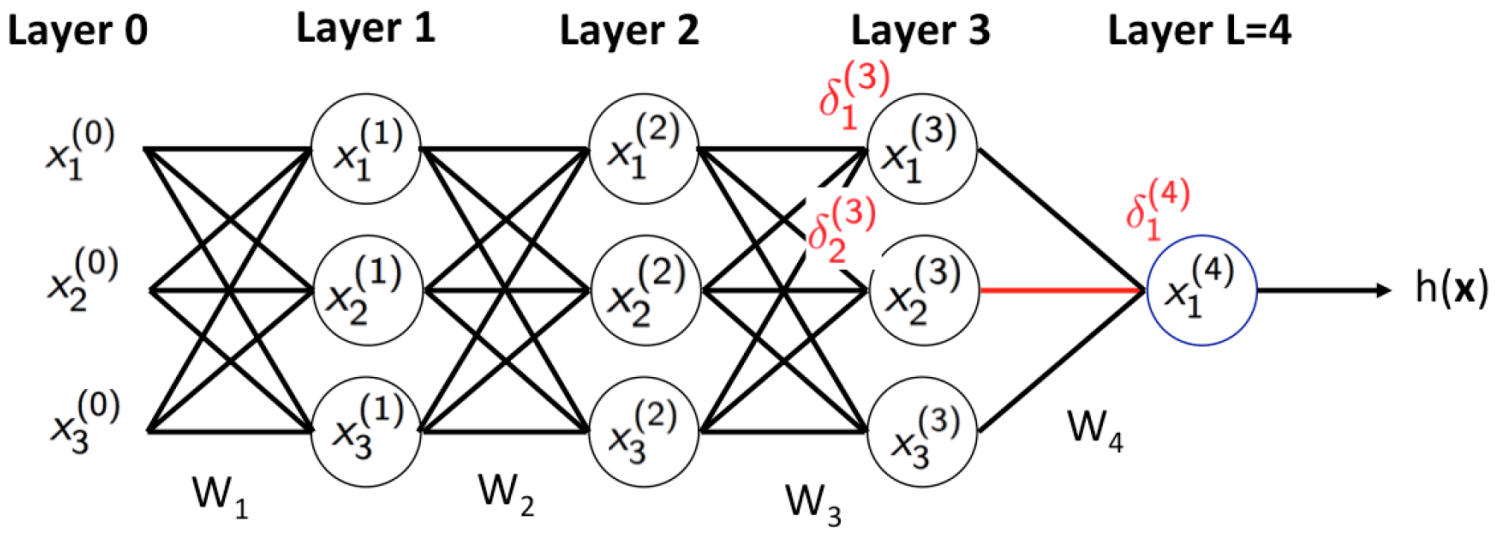
$$\delta_1^{(4)} = 2(x_1^{(4)} - y_n) \times (1 - (x_1^{(4)})^2)$$

Backward propagation (cont'd)



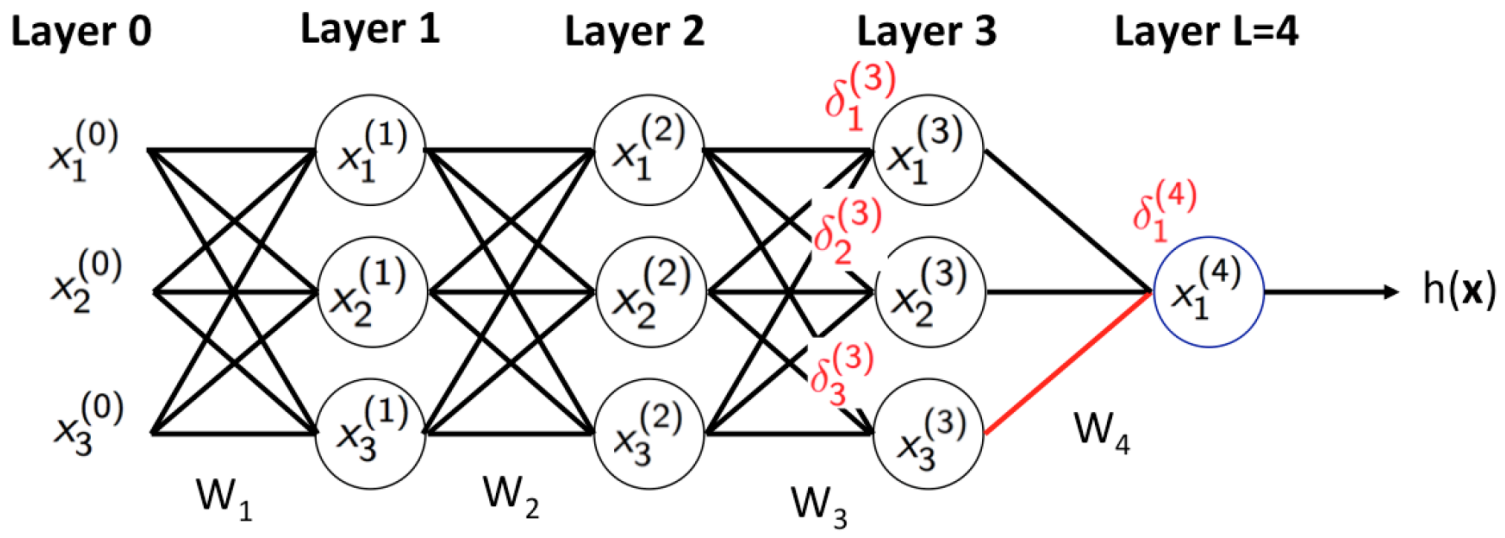
$$\delta_1^{(3)} = (1 - (x_1^{(3)})^2) \times \delta_1^{(4)} \times w_{11}^{(4)}$$

Backward propagation (cont'd)



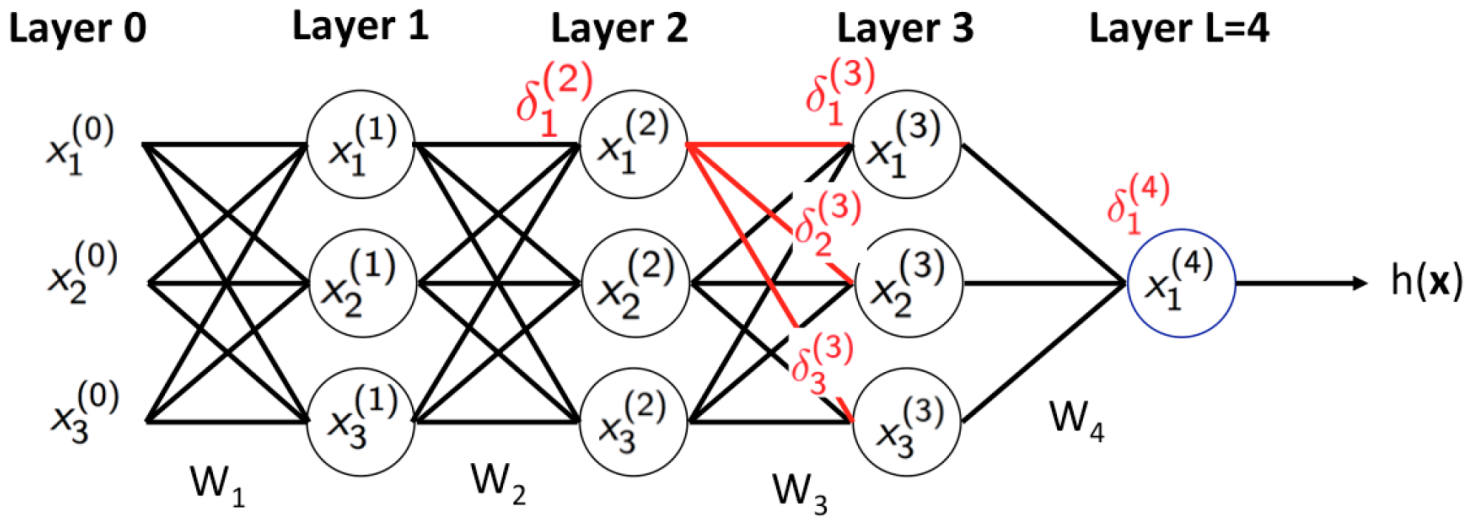
$$\delta_2^{(3)} = (1 - (x_2^{(3)})^2) \times \delta_1^{(4)} \times w_{21}^{(4)}$$

Backward propagation (cont'd)



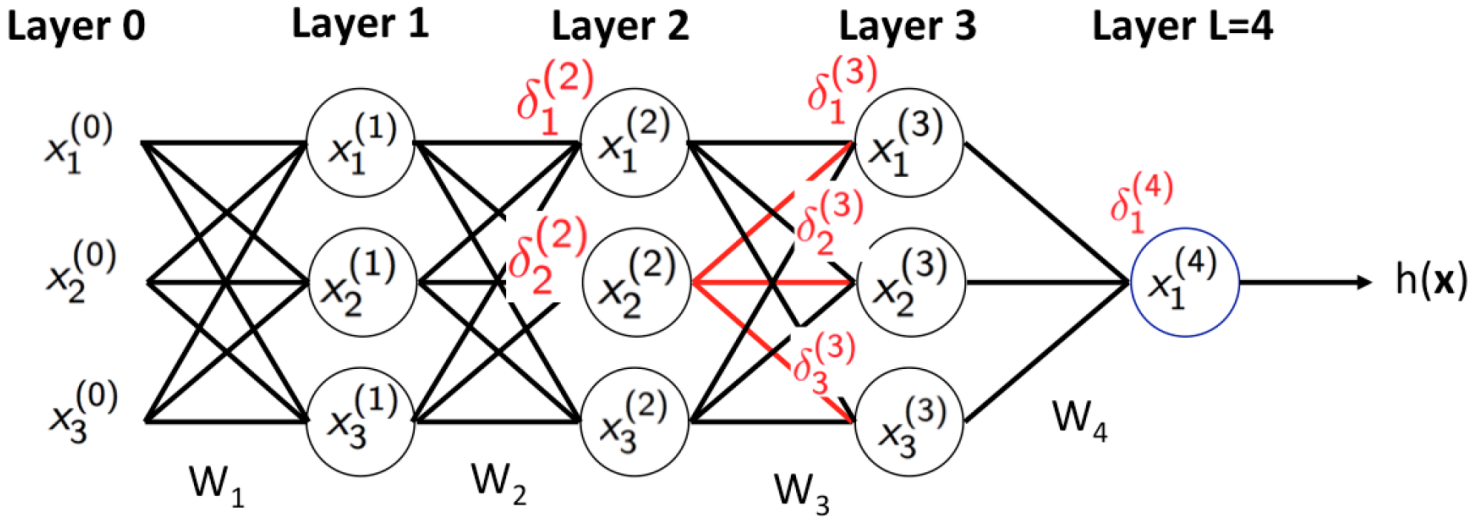
$$\delta_3^{(3)} = (1 - (x_3^{(3)})^2) \times \delta_1^{(4)} \times w_{31}^{(4)}$$

Backward propagation (cont'd)



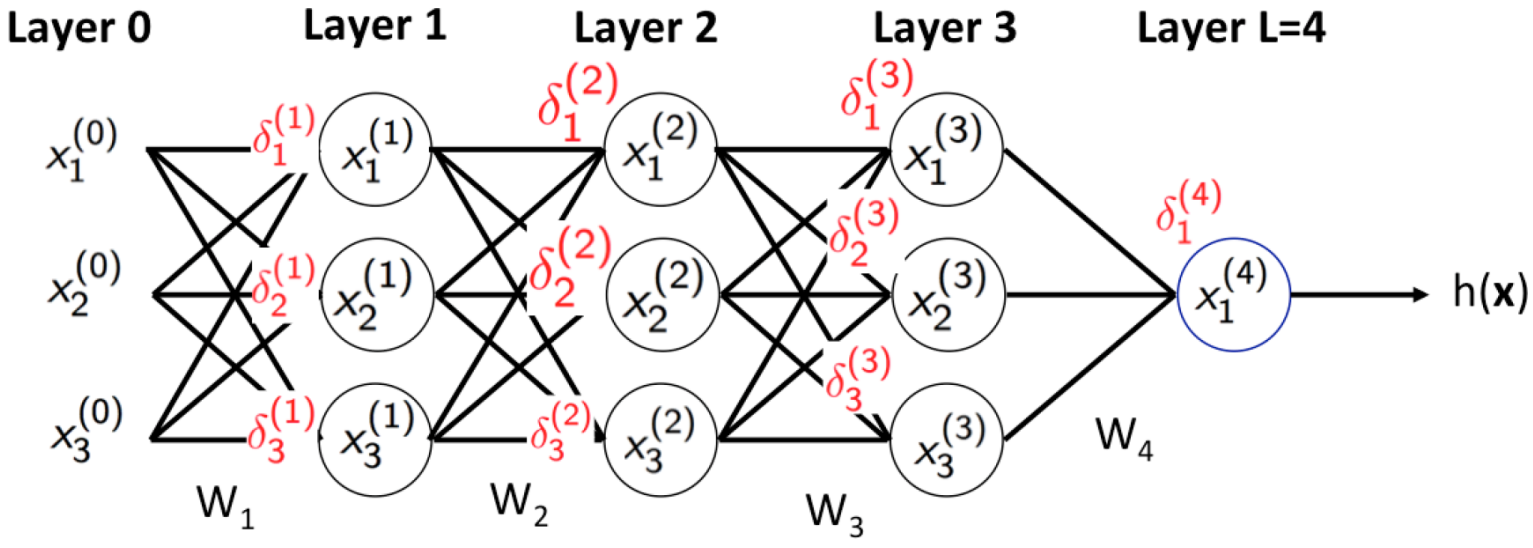
$$\delta_1^{(2)} = (1 - (x_1^{(2)})^2) \sum_{j=1}^{3} \delta_j^{(3)} w_{1j}^{(3)}$$

Backward propagation (cont'd)



$$\delta_2^{(2)} = (1 - (x_2^{(2)})^2) \sum_{j=1}^{3} \delta_j^{(3)} w_{2j}^{(3)}$$

Backward propagation (cont'd)



Computing W for NNs

- Initial $w_{ij}^{(l)}$ at random for all i, j, l
- For iteration $t = 1, 2, \cdots, T$, do:
  - Forward propagation: Compute all $x_j^{(l)}$ from input to output
  - Backward propagation: Compute all $\delta_j^{(l)}$ from output to input
  - Update all the weights:

$$w_{ij}^{(l)} \longleftarrow w_{ij}^{(l)} - \eta x_i^{(l-1)} \delta_j^{(l)}.$$

The algorithm is implemented in most deep learning packages (e.g., pytorch, tensorflow).

Practical issues

- Starting values: usually starting values for weights are chosen to be random values near zero; hence the model starts out nearly linear (for sigmoid), and becomes nonlinear as the weights increase.

  Hanin and Rolnick gives suggestions for choosing the initial value: `https://proceedings.neurips.cc/paper/2018/file/d81f9c1be2e08964bf9f24b15f0e4900-Paper.pdf`

- Overfitting (too many parameters)
- How many hidden units and how many hidden layers: guided by domain knowledge and experimentation
- Multiple minimum

NNs for regression models:

$$y = f(x) + \epsilon,$$

estimating $f(x)$ using NNs. Recent work by Johannes Schmidt-Hieber (`https://arxiv.org/pdf/1708.06633.pdf`) showed that NNs are particular attractive for estimating the multivariate nonparametric regression model.

Application? (Watch this video: `https://www.youtube.com/watch?v=aircAruvnKk&t=989s`)