STOR566: Introduction to Deep Learning

Lecture 6: Neural Networks

Yao Li
UNC Chapel Hill

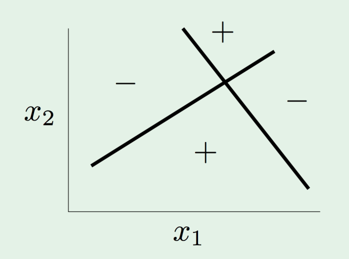
Sep 1, 2022

Materials are from *Learning from data (Caltech)* and *Deep Learning (UCLA)*

Neural Networks

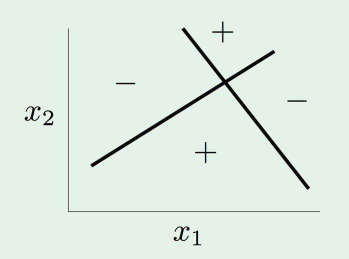
Another way to introduce nonlinearity

- How to generate this nonlinear hypothesis?

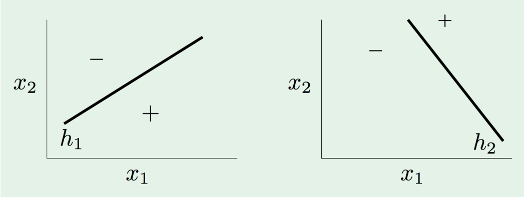


Another way to introduce nonlinearity

- How to generate this nonlinear hypothesis?



- Combining multiple linear hyperplanes to construct nonlinear hypothesis

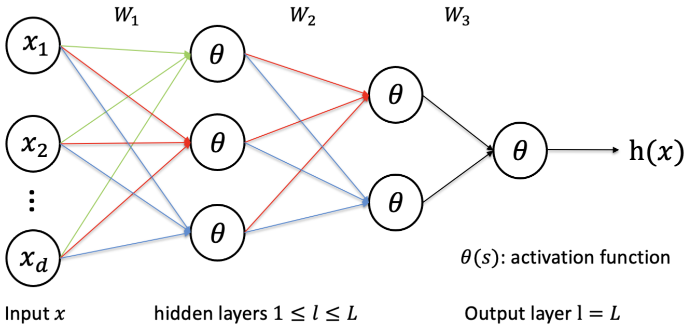


Neural Network

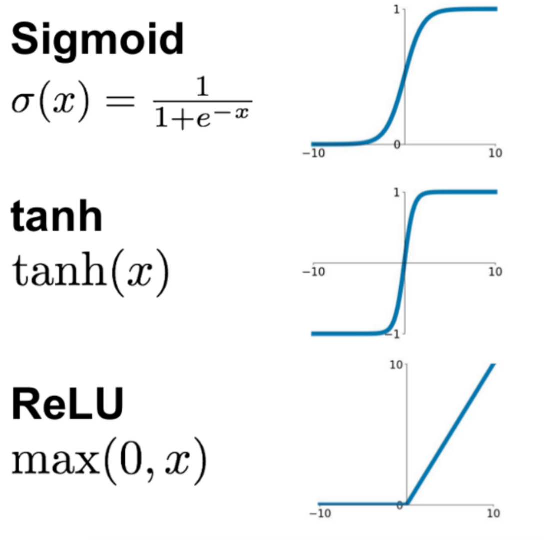
- Input layer: d neurons (input features)
- Neurons from layer 1 to L: Linear combination of previous layers + activation function

$$\theta(\boldsymbol{w}^T \boldsymbol{x}), \quad \theta: \text{ activation function}$$

- Final layer: one neuron $\Rightarrow$ prediction by $\text{sign}(h(\boldsymbol{x}))$



Activation Function



Formal Definitions

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

bias: $b_j^{(l)}$: added to the j-th neuron in the l-th layer

Formal Definitions

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

bias: $b_j^{(l)}$: added to the j-th neuron in the l-th layer

j-th neuron in the l-the layer:

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

Formal Definitions

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

bias: $b_j^{(l)}$: added to the j-th neuron in the l-th layer

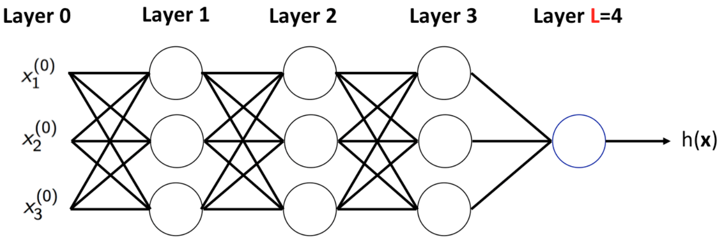
j-th neuron in the l-the layer:

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

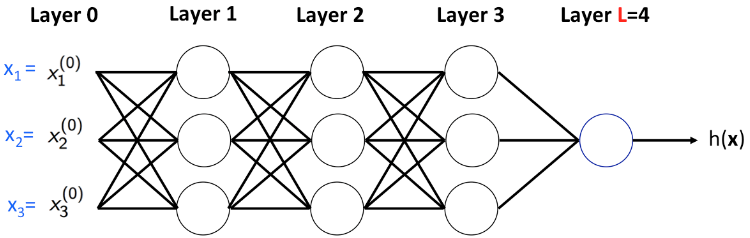
Output:

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

Forward propagation



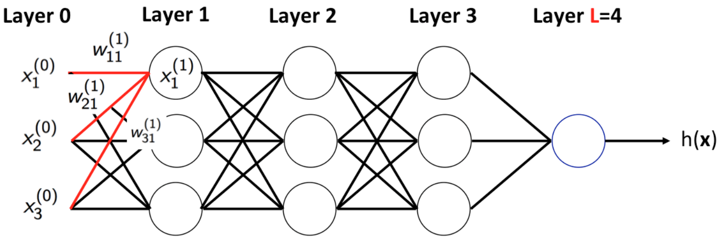
Forward propagation



features for one data point

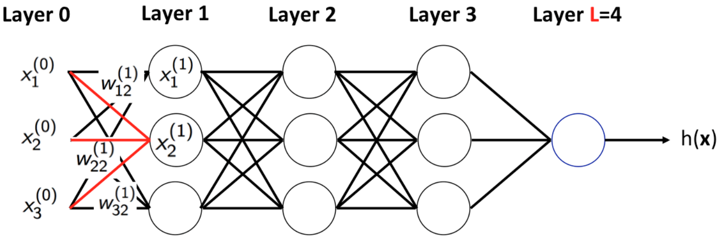
$\mathbf{x} = [x_1, x_2, x_3]$

Forward propagation



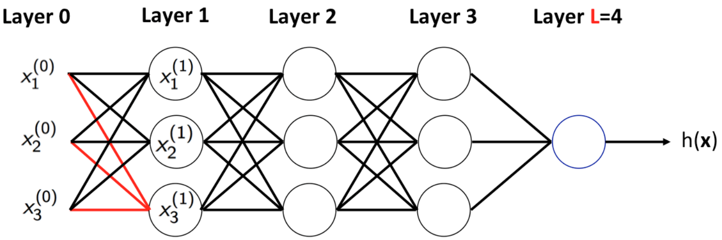
$$x_1^{(1)} = \theta(\textstyle\sum_{i=1}^{3} w_{i1}^{(1)} x_i^{(0)})$$

Forward propagation

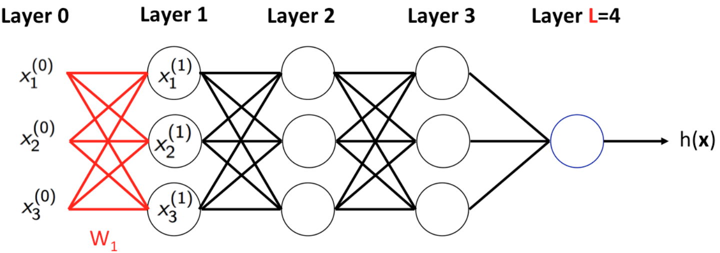


$$x_2^{(1)} = \theta(\sum_{i=1}^{3} w_{i2}^{(1)} x_i^{(0)})$$

Forward propagation

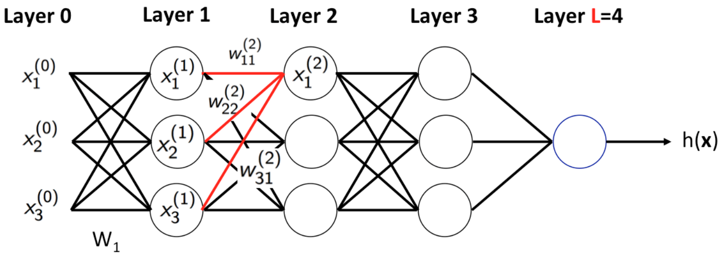


Forward propagation



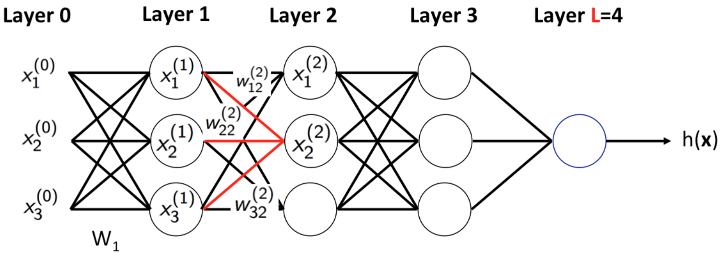
$$\mathbf{x}^{(1)} = \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} = \theta\left(\begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} \\ w_{13}^{(1)} & w_{23}^{(1)} & w_{33}^{(1)} \end{bmatrix} \times \begin{bmatrix} x_1^{(0)} \\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

Forward propagation



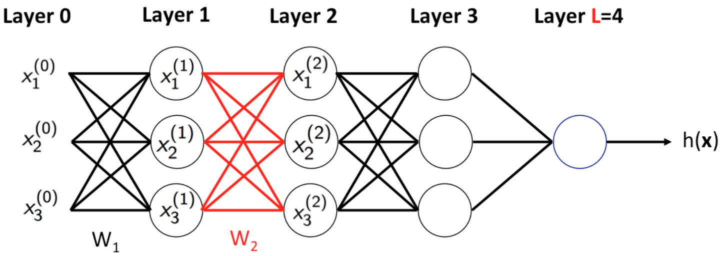
$$x_1^{(2)} = \theta(\sum_{i=1}^{3} w_{i1}^{(2)} x_i^{(1)})$$

Forward propagation



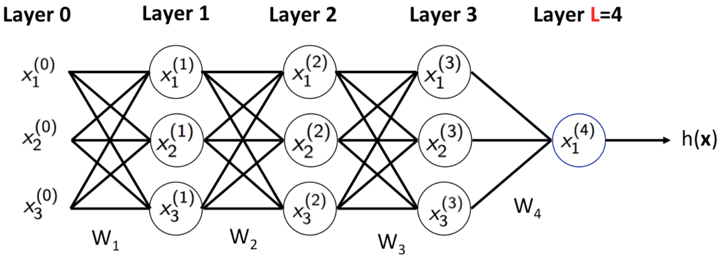
$$x_2^{(2)} = \theta(\sum_{i=1}^{3} w_{i2}^{(2)} x_i^{(1)})$$

Forward propagation



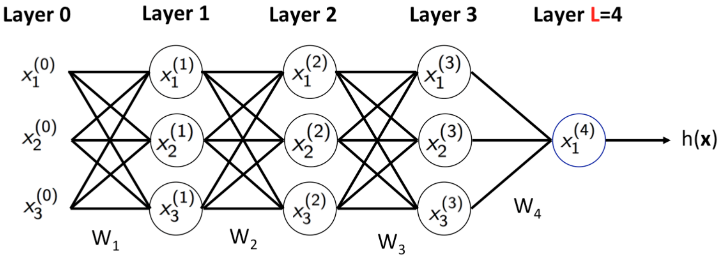
$$\mathbf{x}^{(2)} = \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \end{bmatrix} = \theta\left(\begin{bmatrix} w_{11}^{(2)} & w_{21}^{(2)} & w_{31}^{(2)} \\ w_{12}^{(2)} & w_{22}^{(2)} & w_{32}^{(2)} \\ w_{13}^{(2)} & w_{23}^{(2)} & w_{33}^{(2)} \end{bmatrix} \times \begin{bmatrix} x_1^{(1)} \\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

Forward propagation



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

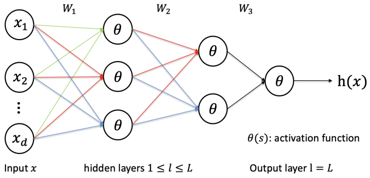
Forward propagation



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

With the bias term: $h(\boldsymbol{x}) = \theta(W_4\theta(W_3\theta(W_2\theta(W_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2) + \boldsymbol{b}_3) + \boldsymbol{b}_4)$

Example: Forward Pass Computation



- Input data: $\boldsymbol{x} = (1.5, -1.0, 1.3)^T$
- Activation: ReLU $(\theta(x) = \max(0, x))$
- Weights:

$$W_1 = \begin{pmatrix} 0.3 & 0.4 & 0.2 \\ 0.3 & 0.5 & 0.2 \\ 0.8 & 1.0 & -1.0 \end{pmatrix}, W_2 = \begin{pmatrix} 0 & -1.2 & 0.5 \\ 0.9 & 1.0 & 0 \end{pmatrix}$$

$$W_3 = (-1.0, 1.0)$$

- Please compute $h(\boldsymbol{x})$.
- Reminder: $h(\boldsymbol{x}) = \theta(W_3\theta(W_2\theta(W_1\boldsymbol{x})))$

Training

- Weights $W = \{W_1, \cdots, W_L\}$ and bias $\{\boldsymbol{b}_1, \cdots, \boldsymbol{b}_L\}$ determine $h(\boldsymbol{x})$
- Learning the weights: solve ERM with SGD.
- Loss on example $(\boldsymbol{x}_n, y_n)$ is

$$e(h(\boldsymbol{x}_n), y_n) = e(W)$$

Training

- Weights $W = \{W_1, \cdots, W_L\}$ and bias $\{\boldsymbol{b}_1, \cdots, \boldsymbol{b}_L\}$ determine $h(\boldsymbol{x})$
- Learning the weights: solve ERM with SGD.
- Loss on example $(\boldsymbol{x}_n, y_n)$ is

$$e(h(\boldsymbol{x}_n), y_n) = e(W)$$

- To implement SGD, we need the gradient

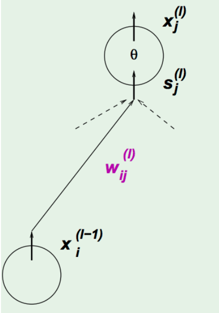
$$\nabla e(W) : \quad \{\frac{\partial e(W)}{\partial w_{ij}^{(l)}}\} \text{ for all } i, j, l$$

(for simplicity we ignore bias in the derivations)

Computing Gradient $\frac{\partial e(W)}{\partial w_{ij}^{(l)}}$

- Use chain rule:

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



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

- We have $\frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}} = x_i^{(l-1)}$

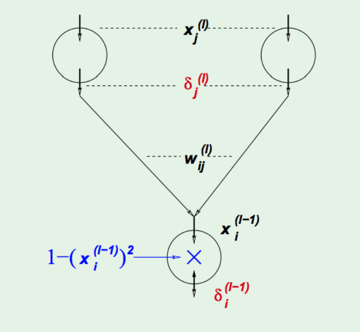
Computing Gradient $\frac{\partial e(W)}{\partial w_{ij}^{(l)}}$

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

$$\delta_i^{(l-1)} = \frac{\partial e(W)}{\partial s_i^{(l-1)}}$$
$$= \sum_{j=1}^{d} \frac{\partial e(W)}{\partial s_j^{(l)}} \times \frac{\partial s_j^{(l)}}{\partial x_i^{(l-1)}} \times \frac{\partial x_i^{(l-1)}}{\partial s_i^{l-1}}$$
$$= \sum_{j=1}^{d} \delta_j^{(l)} \times w_{ij}^{(l)} \times \theta'(s_i^{(l-1)}),$$

where $\theta'(s) = 1 - \theta^2(s)$ for tanh

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



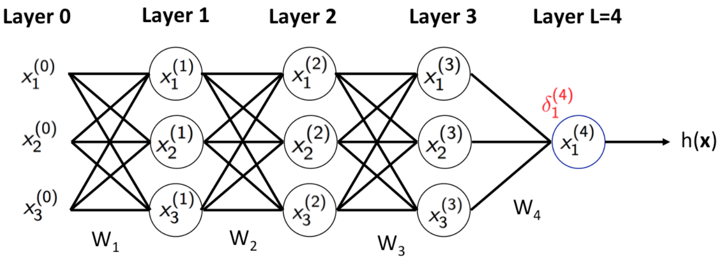
Final layer

(Assume square loss)

- $e(W) = (x_1^{(L)} - y_n)^2$
  $x_1^{(L)} = \theta(s_1^{(L)})$
- So,

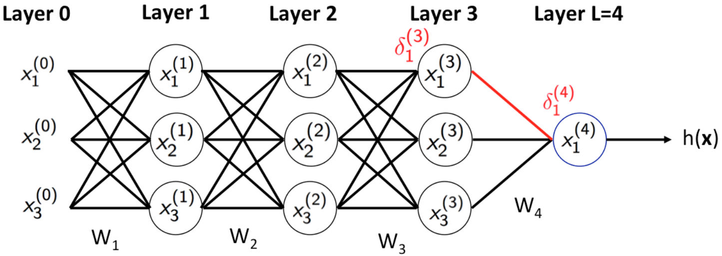
$$\begin{aligned}
\color{red}{\delta_1^{(L)}} &= \frac{\partial e(W)}{\partial s_1^{(L)}} \\
&= \frac{\partial e(W)}{\partial x_1^{(L)}} \times \frac{\partial x_1^{(L)}}{\partial s_1^{(L)}} \\
&= 2(x_1^{(L)} - y_n) \times \theta'(s_1^{(L)})
\end{aligned}$$

Backward propagation



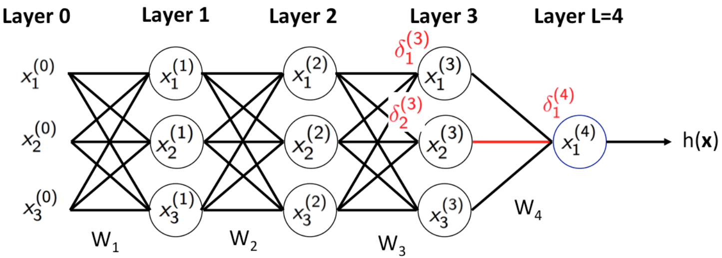
$$\delta_1^{(4)} = 2(x_1^{(4)} - y_n) \times (1 - (x_1^{(4)})^2)$$

Backward propagation



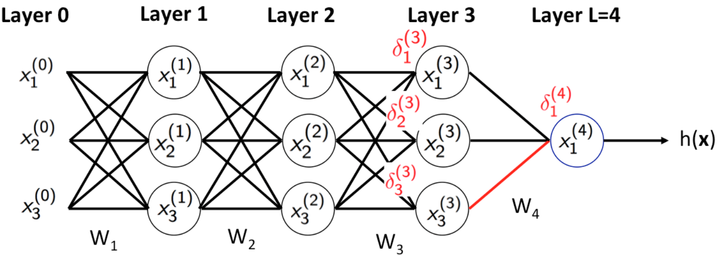
$$\delta_1^{(3)} = (1 - (x_1^{(3)})^2) \times \delta_1^{(4)} \times w_{11}^{(4)}$$

Backward propagation



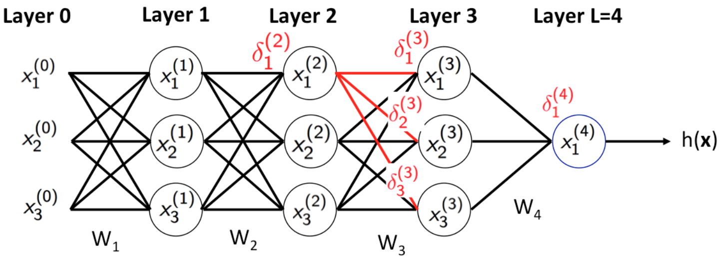
$$\delta_2^{(3)} = (1 - (x_2^{(3)})^2) \times \delta_1^{(4)} \times w_{21}^{(4)}$$

Backward propagation



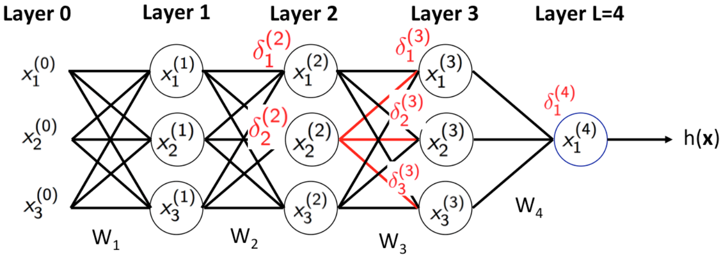
$$\delta_3^{(3)} = (1 - (x_3^{(3)})^2) \times \delta_1^{(4)} \times w_{31}^{(4)}$$

Backward propagation



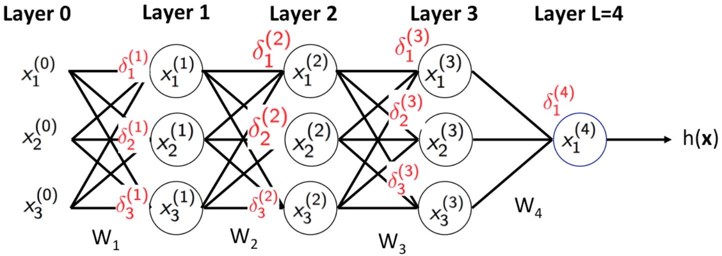
$$\delta_1^{(2)} = (1 - (x_1^{(2)})^2) \sum_{j=1}^{3} \delta_j^{(3)} w_{1j}^{(3)}$$

Backward propagation



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

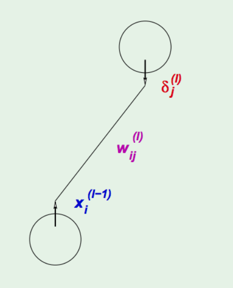
Backward propagation



Backpropagation

SGD for neural networks

- Initialize all weights $w_{ij}^{(l)}$ **at random**
- For iter $= 0, 1, 2, \cdots$
  - Forward: Compute all $x_j^{(l)}$ from input to output
  - Backward: Compute all $\delta_j^{(l)}$ from output to input
  - Update all the weights $w_{ij}^{l} \leftarrow w_{ij}^{(l)} - \eta x_i^{(l-1)} \delta_j^{(l)}$



Backpropagation

- Just an automatic way to apply chain rule to compute gradient
- Auto-differentiation (AD) — as long as we define derivative for each basic function, we can use AD to compute any of their compositions
- Implemented in most deep learning packages

  (e.g., pytorch, tensorflow)

Backpropagation

- Just an automatic way to apply chain rule to compute gradient
- Auto-differentiation (AD) — as long as we define derivative for each basic function, we can use AD to compute any of their compositions
- Implemented in most deep learning packages

  (e.g., pytorch, tensorflow)
- Auto-differentiation needs to store all the intermediate nodes of each sample

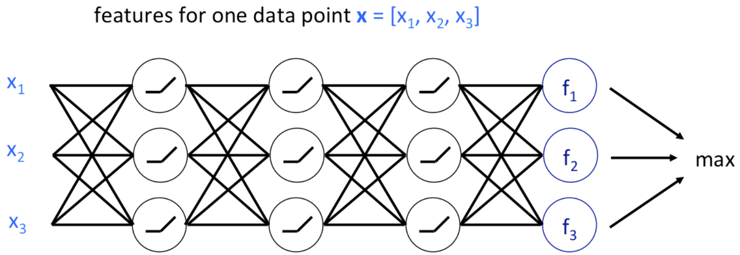
  $\Rightarrow$ Memory cost

  $\Rightarrow$ This poses a constraint on the batch size

Multiclass Classification

- K classes: K neurons in the final layer
- Output of each f_i is the score of class i

  Taking $\arg\max_i f_i(x)$ as the prediction



Multiclass loss

- Softmax function: transform output to probability:

$$[f_1, \cdots, f_K] \rightarrow [p_i, \cdots, p_K]$$

  where $p_i = \frac{e^{f_i}}{\sum_{j=1}^{K} e^{f_j}}$

- Cross-entropy loss:

$$L = -\sum_{i=1}^{K} y_i \log(p_i)$$

  where y_i is the i-th label

Conclusions

- Neural network
- Forward propagation
- Back-propagation for computing gradient

Questions?