Neural Networks

Zhiyao Duan
Associate Professor of ECE and CS
University of Rochester

Some figures are copied from the following books

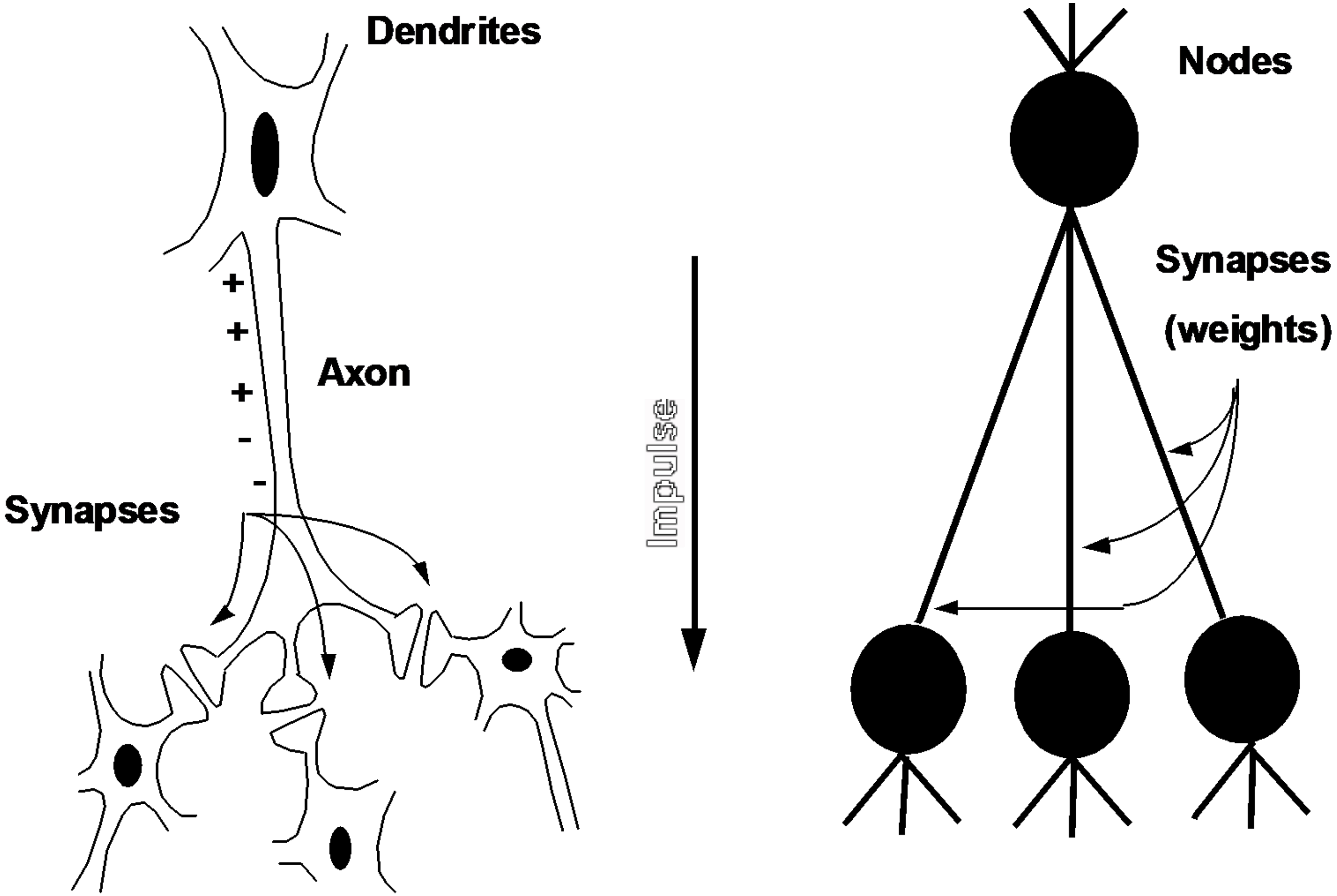
- **LWLS** - Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, Thomas B. Schön, *Machine Learning: A First Course for Engineers and Scientists*, Cambridge University Press, 2022.

Biological Motivation

- Human brain: a densely interconnected network
  - ~10^11 neurons
  - Each neuron connects to ~10^4 other neurons
  - Two states of neuron activity: excited vs. inhibited
  - Neuron switching speed: ~1kHz
    - CPU clock frequency: GHz
  - Yet many tasks (e.g., face recognition) can be completed within 0.1 s

- This suggests
  - Highly parallel processing
  - Distributed representations

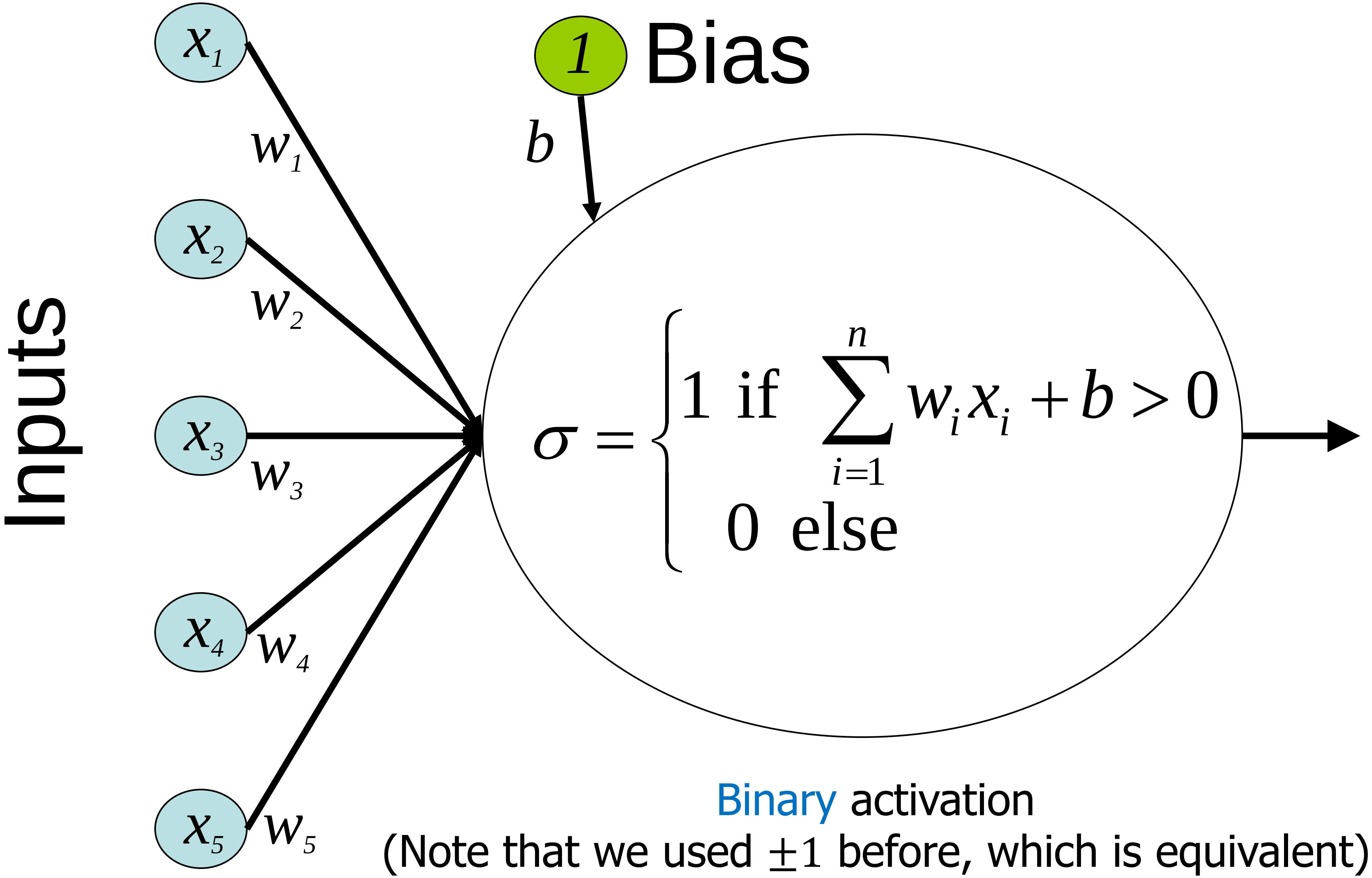
Biological Analogy



History of Neural Networks

- 1943 – first neural network computing model by McCulloch and Pitts
- 1958 – Perceptron by Rosenblatt
- 1960's – a big wave
- 1969 – Minsky & Papert's book "Perceptrons"
- 1970's – "winter" of neural networks
- 1975 – Backpropagation algorithm by Werbos
- 1980's – another big wave
- 1990's – overtaken by SVM proposed in 1993 by Vapnik
- 2006 – a fast learning algorithm for training deep belief networks by Hinton
- 2010's – another big wave
- 2018 – Turing Award to Hinton, Bengio & LeCun
- 2022 – ChatGPT!
- 2024 – Sora!
- Present – continue to transform various domains

Perceptron



$$\sigma = \begin{cases} 1 \text{ if } \sum_{i=1}^{n} w_i x_i + b > 0 \\ 0 \text{ else} \end{cases}$$

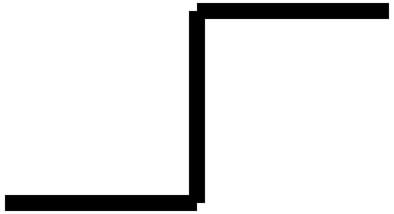
Binary activation
(Note that we used ±1 before, which is equivalent)

Nonlinear Activation Functions

- Step function

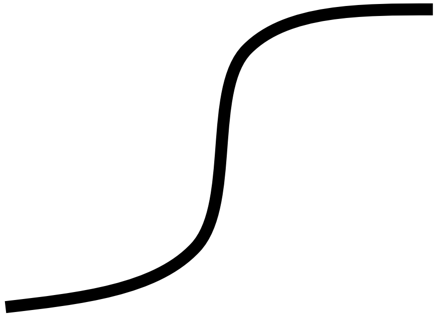
$$output = sign(\boldsymbol{w}^T\boldsymbol{x} + b)$$

  - Note: previously we used {-1,1} for sign function for perceptron, which is equivalent



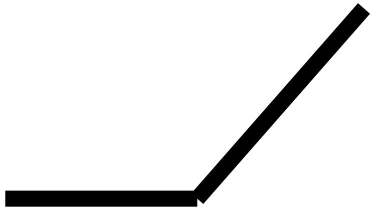
- Sigmoid function

$$output = \sigma(\boldsymbol{w}^T\boldsymbol{x} + b) = \frac{1}{1 + e^{-(\boldsymbol{w}^T\boldsymbol{x} + b)}}$$



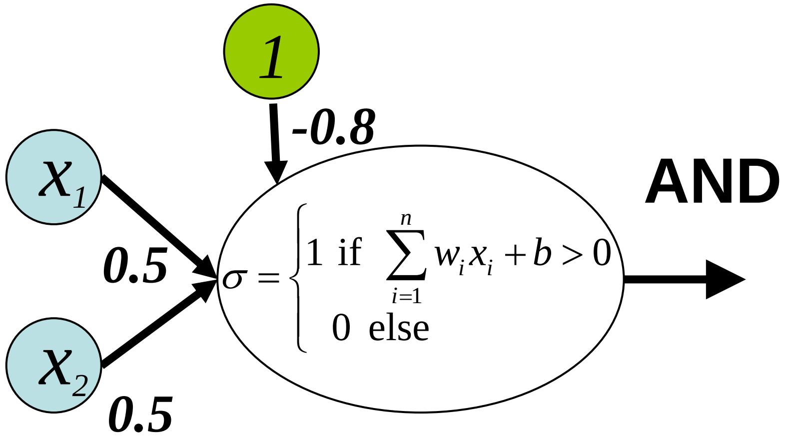
- Rectified Linear Unit (ReLU)

$$output = \max\{0, \boldsymbol{w}^T\boldsymbol{x} + b\}$$

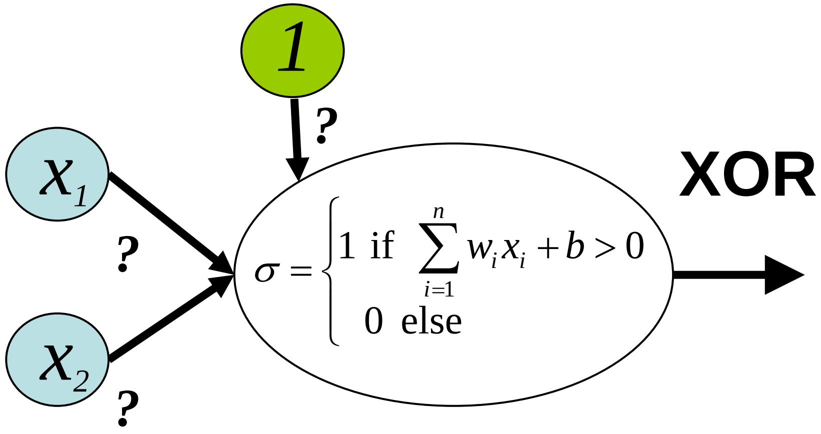


Limitations of 1-layer Nets

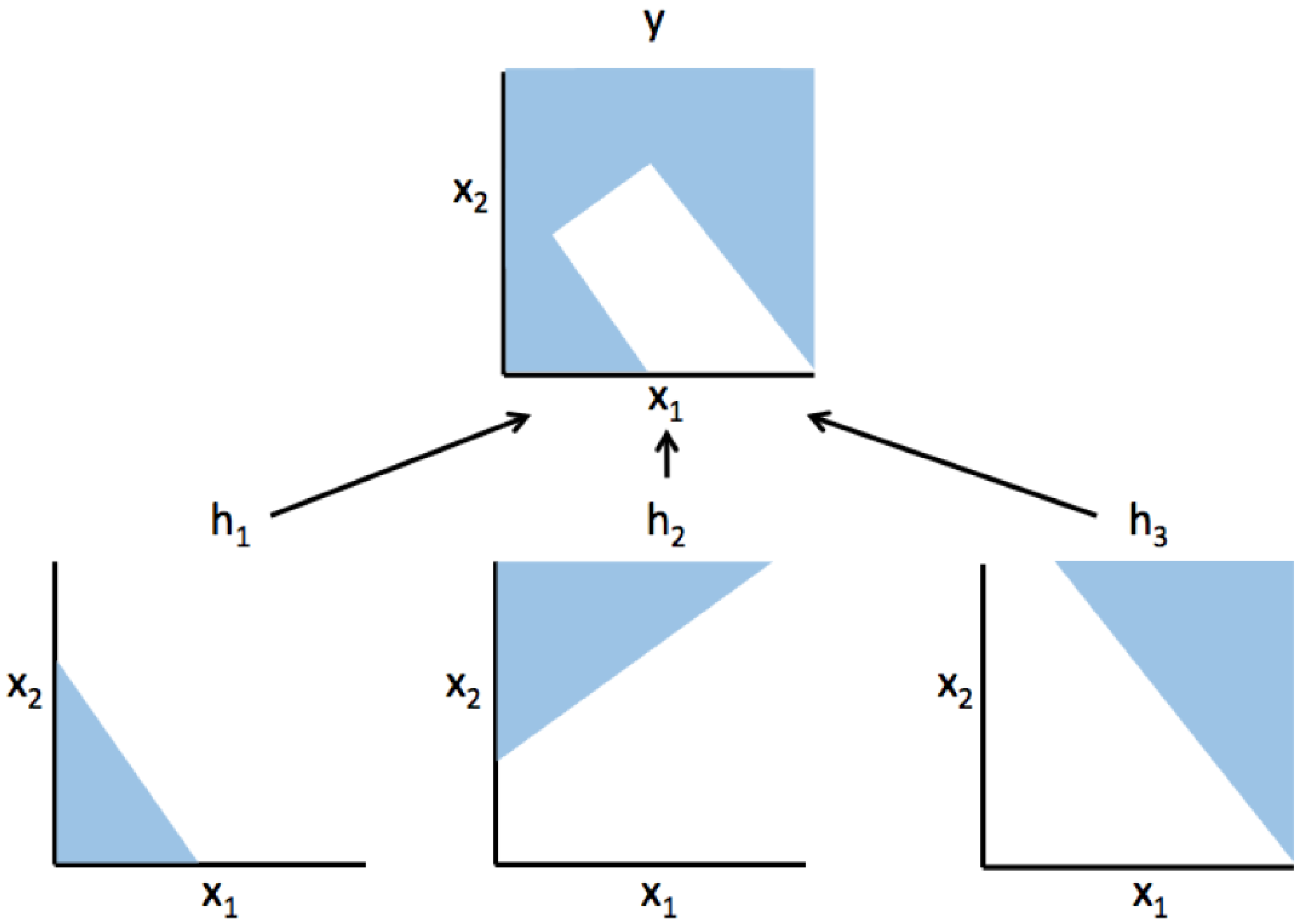
- Only express linearly separable cases
  - For example, they are good as logic operators "AND", "NOT", and "OR"



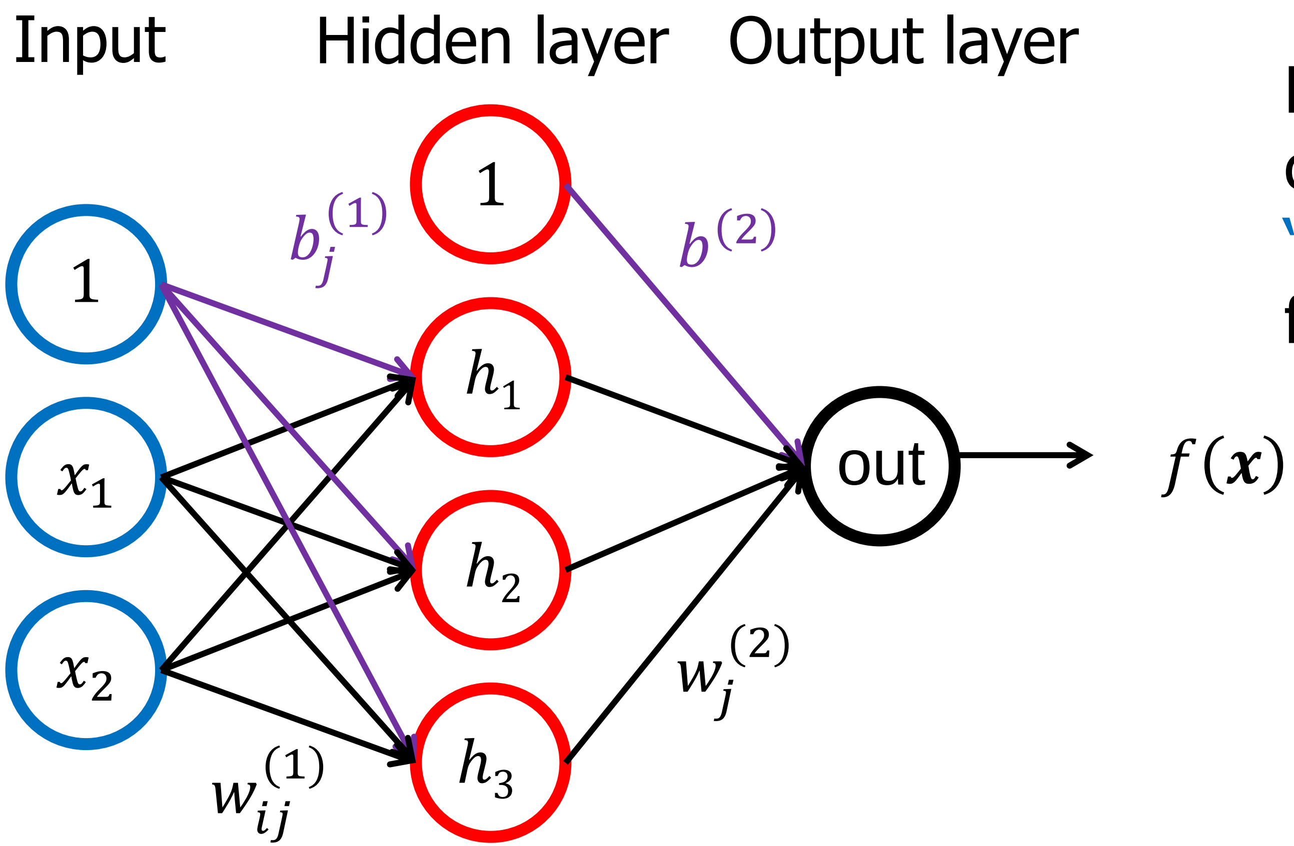
- Cannot represent "XOR", which is not linearly separable



But, we can combine them!



2-layer Nets

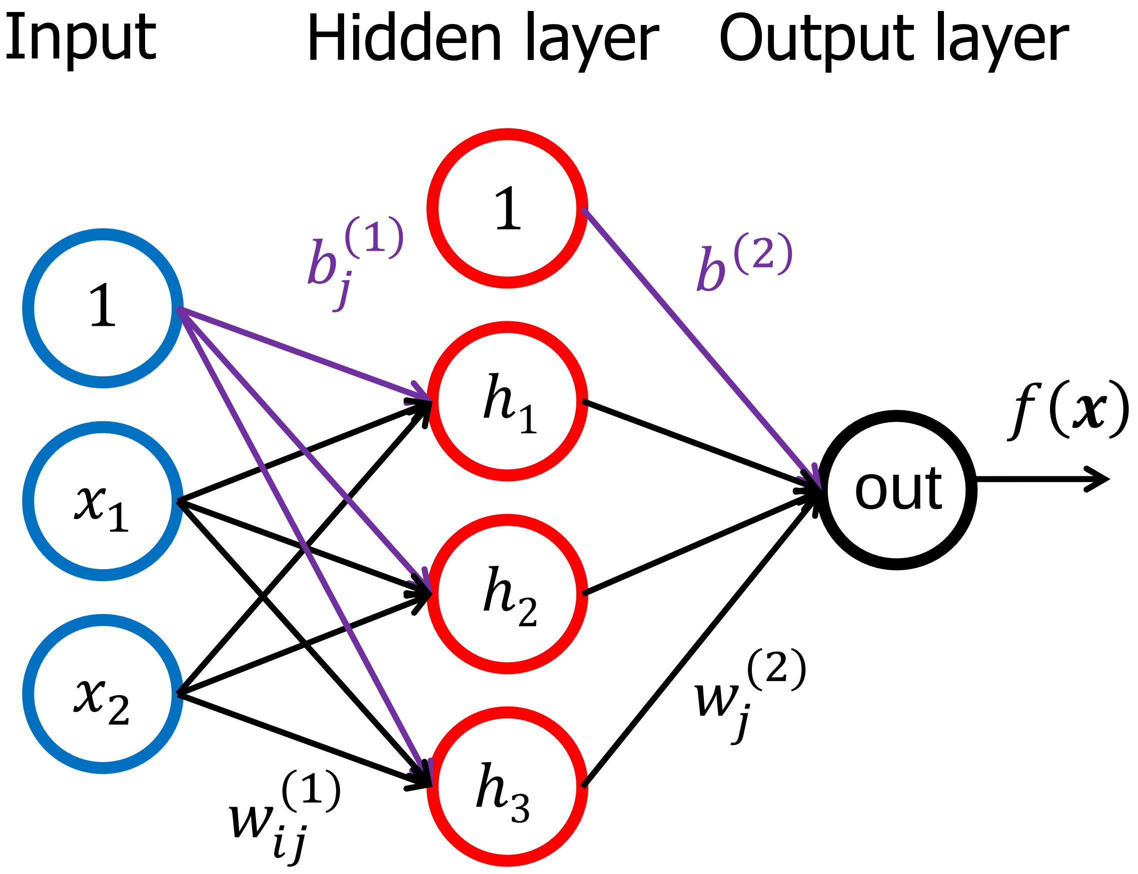


Hidden layer output can be viewed as “features” calculated from the raw input

$$f(\boldsymbol{x}) = \sigma\left(\sum_j w_j^{(2)} h_j + b^{(2)}\right) = \sigma\left(\sum_j w_j^{(2)} \sigma\left(\sum_i w_{ij}^{(1)} x_i + b_j^{(1)}\right) + b^{(2)}\right)$$

Matrix Notation

$$f(\boldsymbol{x}) = \sigma\left(\sum_j w_j^{(2)}\,\sigma\left(\sum_i w_{ij}^{(1)}\,x_i + b_j^{(1)}\right) + b^{(2)}\right)$$



$$f(\boldsymbol{x}) = \sigma\left(\boldsymbol{W}_2^T\boldsymbol{\sigma}\left(\boldsymbol{W}_1^T\boldsymbol{x} + \boldsymbol{b}_1\right) + b_2\right)$$

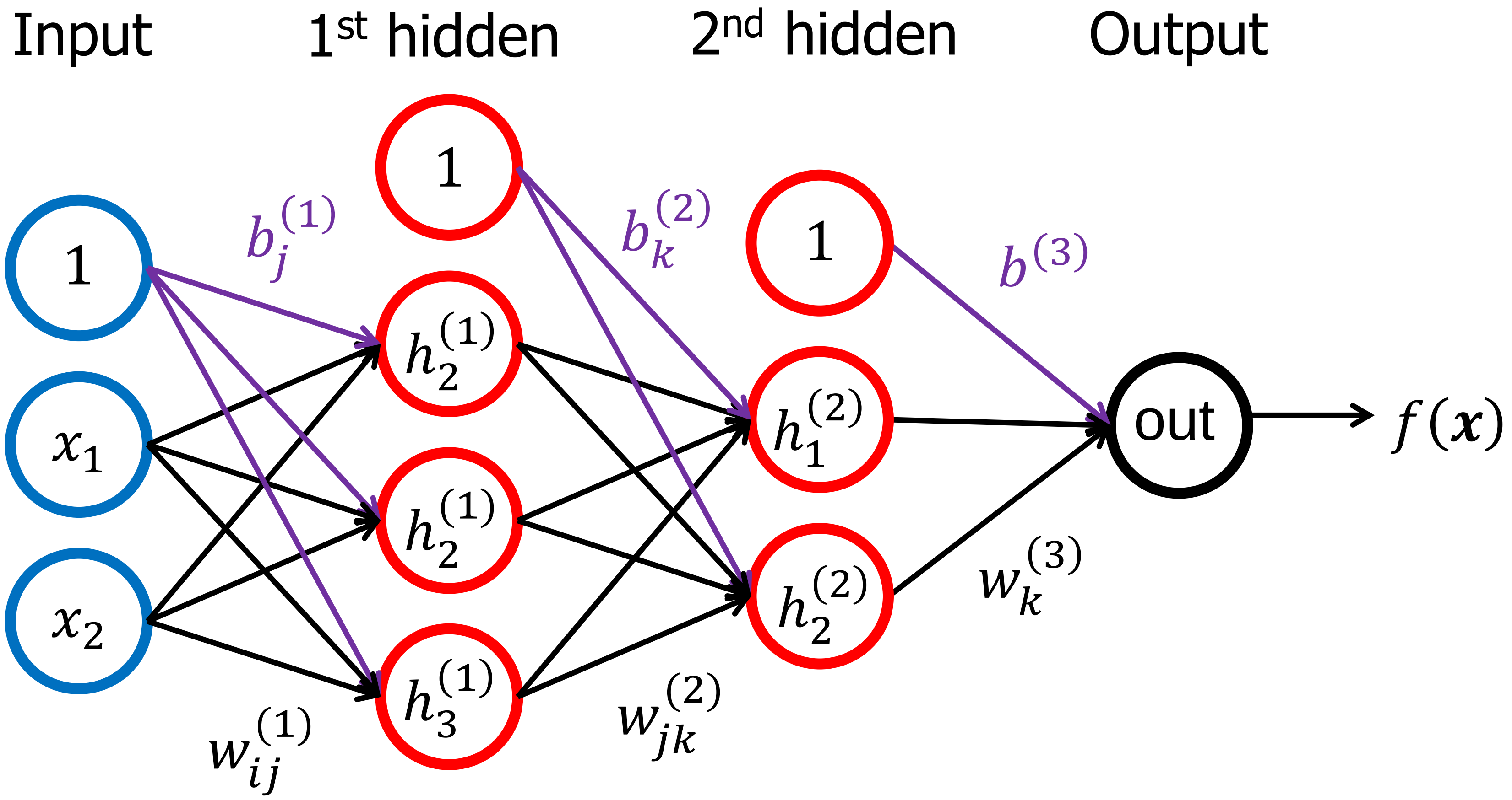
where

$$\boldsymbol{W}_1 = \left[w_{ij}^{(1)}\right]_{d\times l_1}, \boldsymbol{b}_1 = \left[b_j^{(1)}\right]_{l_1\times 1}$$

$$\boldsymbol{W}_2 = \left[w_{jk}^{(2)}\right]_{l_1\times l_2}, b_2 = b^{(2)}$$

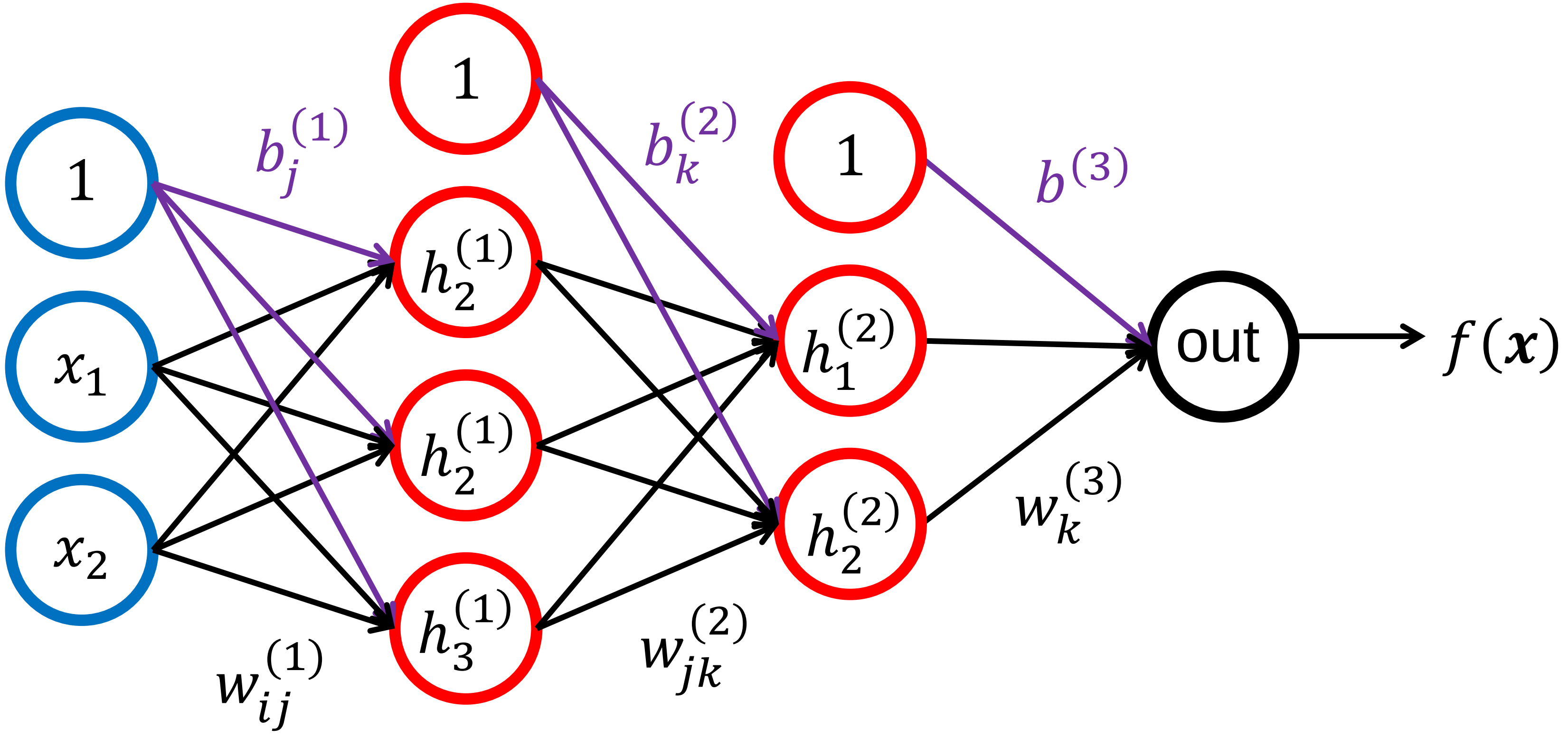
- What does $\boldsymbol{W}_1^T\boldsymbol{x}$ compute?
  - Inner products between columns of $\boldsymbol{W}_1$ and $\boldsymbol{x}$
  - Columns of $\boldsymbol{W}_1$ are “receptors” or “filters”
  - $\boldsymbol{W}_1^T\boldsymbol{x}$ are their responses to input

3-layer Nets



$$f(\boldsymbol{x}) = \sigma\left(\sum_k w_k^{(3)} h_k^{(2)} + b^{(3)}\right) = \sigma\left(\sum_k w_k^{(3)} \sigma\left(\sum_j w_{jk}^{(2)} h_j^{(1)} + b_k^{(2)}\right) + b^{(3)}\right) = \sigma\left(\sum_k w_k^{(3)} \sigma\left(\sum_j w_{jk}^{(2)} \sigma\left(\sum_i w_{ij}^{(1)} x_i + b_j^{(1)}\right) + b_k^{(2)}\right) + b^{(3)}\right)$$

Matrix Notation



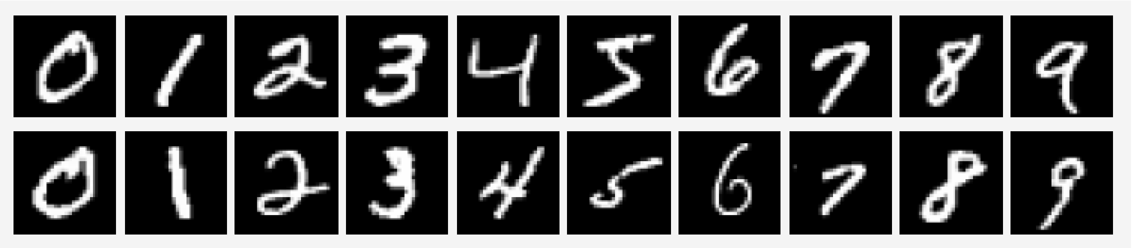
$$f(\boldsymbol{x}) = \sigma\left(\sum_k w_k^{(3)} \sigma\left(\sum_j w_{jk}^{(2)} \sigma\left(\sum_i w_{ij}^{(1)} x_i + b_j^{(1)}\right) + b_k^{(2)}\right) + b^{(3)}\right)$$

$$f(\boldsymbol{x}) = \sigma\left(\boldsymbol{W}_3^T \boldsymbol{\sigma}\left(\boldsymbol{W}_2^T \boldsymbol{\sigma}\left(\boldsymbol{W}_1^T \boldsymbol{x} + \boldsymbol{b}_1\right) + \boldsymbol{b}_2\right) + b_3\right)$$

Richer Representations with More Layers

- 1-layer nets (e.g., perceptron) only model linear hyperplanes
- 2-layer nets can approximate any continuous function, given enough hidden nodes
- >=3-layer nets can do so with fewer nodes and weights
- Nonlinear activation is key!
  - Multiple layers of linear activations is still linear!

Example Application



(Fig. 6.5 in LWLS, from MNIST dataset)
70,000 grayscale images (28*28) from 10 classes

- One-layer MLP (i.e., logistic regression)
  - Input: 28*28=784-d vectors
  - Output layer size: 10 nodes
  - #parameters: 784*10+10 = 7,850
- Two-layer MLP
  - Input: 28*28=784-d vectors
  - Hidden layer size: 200 nodes
  - Output layer size: 10 nodes
  - #parameters for hidden layer: 784*200+200
  - #parameters for output layer: 200*10+10
  - #Total parameters = 159,010

Properties of NNs

- Large capacity: able to learn complex relations between input and output
- Support various data formats: continuous, discrete, categorical (needs to be encoded into numeric)
- Robust to some level of noise in training data
- Inference (i.e., making predictions on test examples) is fast

- Data hungry
- Training is slow
- Lack of mathematical analysis and difficult to interpret

How to learn the weights?

- Given training data - input and label pairs $\{\boldsymbol{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$
- Update network weights to minimize the difference (error) between $f\left(\boldsymbol{x}^{(i)}\right)$ and $y^{(i)}$
  - Calculate derivative of error w.r.t. weights
  - Gradient descent to update weights
  - Backpropagation algorithm: recursive computation of these gradients

- See derivation on white board

Backpropagation Recap

- Assume we use sigmoid activation and the squared error loss
  - We can also use other activations, e.g., ReLU
  - We can also use other losses, e.g., cross entropy
- Then the loss on the entire training set is

$$E(\boldsymbol{\theta}) = \frac{1}{2N}\sum_{i=1}^{N}\left(y^{(i)} - \hat{y}^{(i)}\right)^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(y^{(i)} - f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)\right)^2$$

where $\boldsymbol{\theta}$ denotes network parameters, i.e., network weights

- We compute gradient $\nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta})$ (called the true gradient, versus stochastic gradient computed on a subset of data), and then update $\boldsymbol{\theta}$ along the negative gradient direction iteratively
- The computation of $\nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta})$ is recursive, backward from the last layer to the first layer, leveraging the layer-wise structure of the network
- The computation also requires node outputs at each layer, which are computed in a forward pass

Forward Pass In Matrix Notation

- Start from input $\boldsymbol{X}_{N\times d} = \left[\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(N)}\right]^T$ corresponding to all N points

- Compute first hidden layer net input $\boldsymbol{Z}_1$

$$[\boldsymbol{Z}_1]_{N\times l_1} = [\boldsymbol{X}\boldsymbol{W}_1]_{N\times l_1} + [repmat(\boldsymbol{b}_1^T)]_{N\times l_1}$$

- Compute first hidden layer output $\boldsymbol{H}_1$

$$[\boldsymbol{H}_1]_{N\times l_1} = \boldsymbol{\sigma}(\boldsymbol{Z}_1)$$

- Compute second hidden layer net input $\boldsymbol{Z}_2$

$$[\boldsymbol{Z}_2]_{N\times l_2} = [\boldsymbol{H}_1\boldsymbol{W}_2]_{N\times l_2} + [repmat(\boldsymbol{b}_2^T)]_{N\times l_2}$$

- Compute second hidden layer output $\boldsymbol{H}_2$

$$[\boldsymbol{H}_2]_{N\times l_2} = \boldsymbol{\sigma}(\boldsymbol{Z}_2)$$

- ……

- Compute final output $\hat{\boldsymbol{y}}$, a vector corresponding to all N points

Backward Pass in Matrix Notation

- Mean squared error computed on all data: $E(\boldsymbol{\theta}) = \frac{1}{2N}\sum_{i=1}^{N}\left(y^{(i)} - \hat{y}^{(i)}\right)^2 = \frac{1}{2N}(\boldsymbol{y} - \widehat{\boldsymbol{y}})^T(\boldsymbol{y} - \widehat{\boldsymbol{y}})$
- Compute gradients w.r.t. weights in the output layer (the M-th layer)

$$\left[\frac{\partial E}{\partial \widehat{\boldsymbol{y}}}\right]_{N\times 1} = \frac{1}{N}(\widehat{\boldsymbol{y}} - \boldsymbol{y})$$

$$[\sigma'(\boldsymbol{z}_M)]_{N\times 1} = \widehat{\boldsymbol{y}} \odot (1 - \widehat{\boldsymbol{y}})$$

$$\left[\frac{\partial E}{\partial \boldsymbol{w}_M}\right]_{l_{M-1}\times 1} = \underbrace{\left[\frac{\partial \boldsymbol{z}_M}{\partial \boldsymbol{w}_M}\right]_{l_{M-1}\times N}}_{\boldsymbol{H}_{M-1}^T} \cdot \left[\left[\frac{\partial E}{\partial \widehat{\boldsymbol{y}}}\right]_{N\times 1} \odot [\sigma'(\boldsymbol{z}_M)]_{N\times 1}\right]$$

$$\frac{\partial E}{\partial b_M} = \underbrace{\left[\frac{\partial \boldsymbol{z}_M}{\partial b_M}\right]_{1\times N}}_{\boldsymbol{1}^T} \cdot \left[\left[\frac{\partial E}{\partial \widehat{\boldsymbol{y}}}\right]_{N\times 1} \odot [\sigma'(\boldsymbol{z}_M)]_{N\times 1}\right]$$

Backward Pass in Matrix Notation

- Compute gradients w.r.t. weights in the $(m-1)$-th layer recursively

$$\left[\frac{\partial E}{\partial \boldsymbol{H}_{m-1}}\right]_{N\times l_{m-1}} = \left[\frac{\partial E}{\partial \boldsymbol{H}_{m}}\right]_{N\times l_{m}} \odot [\sigma'(\boldsymbol{Z}_m)]_{N\times l_m} \cdot [\boldsymbol{W}_m^T]_{l_m\times l_{m-1}}$$

$$\left[\frac{\partial E}{\partial \boldsymbol{W}_{m-1}}\right]_{l_{m-2}\times l_{m-1}} = [\boldsymbol{H}_{m-2}^T]_{l_{m-2}\times N} \cdot \left[\left[\frac{\partial E}{\partial \boldsymbol{H}_{m-1}}\right]_{N\times l_{m-1}} \odot [\sigma'(\boldsymbol{Z}_{m-1})]_{N\times l_{m-1}}\right]$$

$$\left[\frac{\partial E}{\partial \boldsymbol{b}_{m-1}}\right]_{l_{m-1}\times 1} = \left[\left[\frac{\partial E}{\partial \boldsymbol{H}_{m-1}}\right]_{N\times l_{m-1}} \odot [\sigma'(\boldsymbol{Z}_{m-1})]_{N\times l_{m-1}}\right]^T \cdot \boldsymbol{1}_{N\times 1}$$

Problems of BP for Deep Networks

- Vanishing gradient problem
  - Gradients vanish when they are propagated back to early layers, hence their weights are hard to adjust
  - Sigmoid activation → ReLU activation

- Many local minima
  - Which will trap gradient decent methods
  - In practice, local minima are pretty good

Summary

- (Artificial) neural networks are inspired by biological neural networks
  - Parallel processing + distributed representation
- Feedforward neural networks use a layer-wise structure
  - Full connection between adjacent layers
  - Linear mapping + nonlinear activation
- Representation power
  - 1-layer NNs are just perceptron or logistic regression
  - 2-layer NNs can represent (almost) any continuous function, with sufficient hidden nodes
  - >=3-layer NNs can do so with much fewer nodes
- Gradient descent to update network weights using training data
- Backpropagation algorithm to recursively compute gradients
  - Vanishing gradient issues for sigmoid activation