Neural Computation

Week 4 - Perceptron and Neural Network

Yunwen Lei

School of Computer Science, University of Birmingham

Outline

Perceptron

Binary Classification

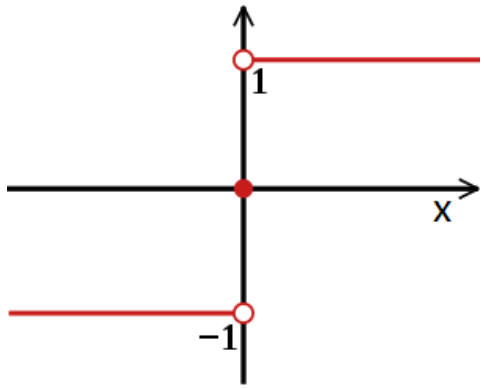
- Dataset: n input/output pairs

$$S = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\}, \quad y_i \in \{+1, -1\}$$

- Training examples with $y = +1$ are called positive examples
- Training examples with $y = -1$ are called negative examples

- **Aim**: find a classification rule $\mathbf{x} \mapsto \{-1, +1\}$
  - we always find first a function $f : \mathcal{X} \mapsto \mathbb{R}$ and do the prediction by $\operatorname{sgn}(f(x))$

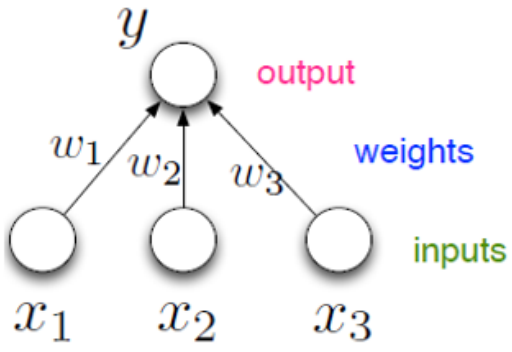
$$\operatorname{sgn}(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x = 0 \\ -1, & \text{if } x < 0. \end{cases}$$

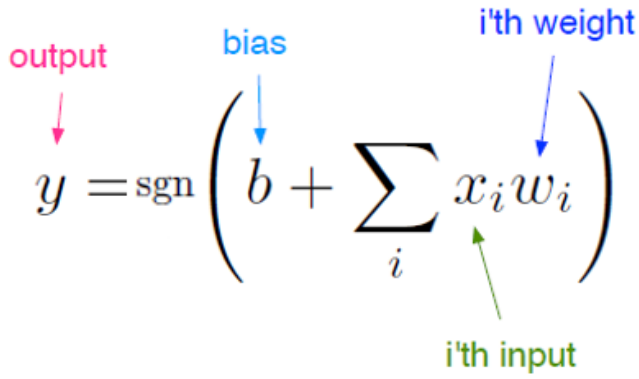


Classification by Perceptron

Perceptron

A perceptron outputs the sign of a linear function: $\mathrm{sgn}(\mathbf{w}^\top \mathbf{x} + b)$



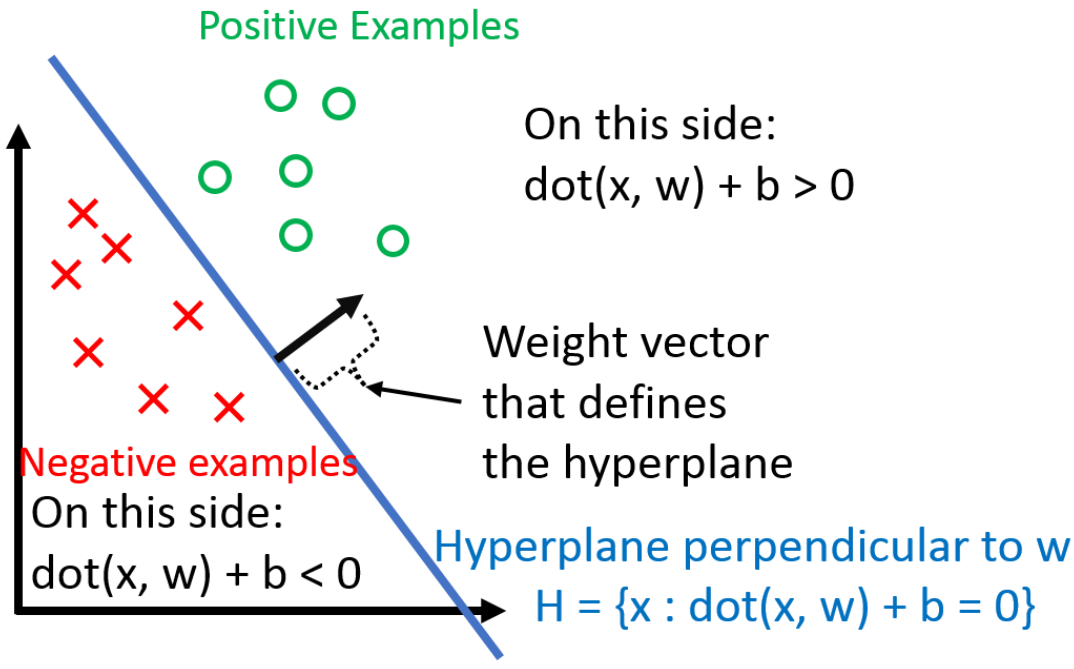


- Training: compute $(\mathbf{w}, b)$ such that $\mathrm{sgn}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \approx y^{(i)}$ for all i
- Prediction: given a testing example $\mathbf{x}$, predict the output as

$$\mathrm{sgn}(\mathbf{w}^\top \mathbf{x} + b)$$

Classification by Perceptron

The classifier built by a perceptron corresponds to finding a linear hyperplane to separate positive examples from negative examples



We can get the hyperplane by Perceptron Algorithm!

Perceptron Algorithm

Perceptron Algorithm

1: Initialize $\mathbf{w} = 0$ ▷ we assume no bias for simplicity
2: **while** All training examples are not correctly classified **do**
3: **for** $(\mathbf{x}, y) \in S$ **do** ▷ Loop over each (feature, label) pair in the dataset
4: **if** $y \cdot \mathbf{w}^\top \mathbf{x} \leq 0$ **then** ▷ If the pair $(\mathbf{x}, y)$ is misclassified
5: $\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}$ ▷ Update the weight vector $\mathbf{w}$

- It updates the model if encountering a misclassification on $(\mathbf{x}, y)$

$$y\mathbf{w}^\top \mathbf{x} \leq 0 \iff y \neq \operatorname{sgn}(\mathbf{w}^\top \mathbf{x})$$

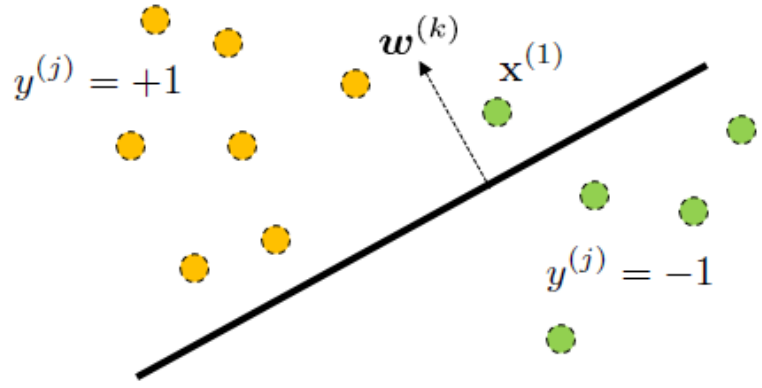
Update $\mathbf{w}$ so that after update it is more likely to predict correctly on $(\mathbf{x}, y)$

$$y\big(\underbrace{\mathbf{w} + y\mathbf{x}}_{\text{after update}}\big)^\top \mathbf{x} = y\mathbf{w}^\top \mathbf{x} + \underbrace{y^2 \mathbf{x}^\top \mathbf{x}}_{>0} > y \underbrace{\mathbf{w}^\top}_{\text{before update}} \mathbf{x}.$$

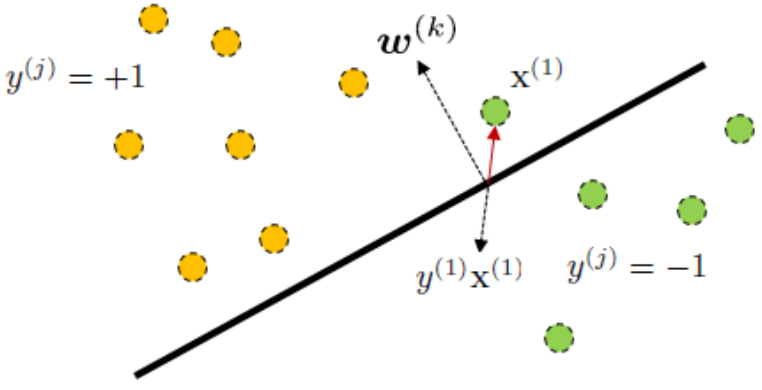
Guaranteed to stop if dataset is linearly separable!

We will prove this result as an exercise

Updating One Example

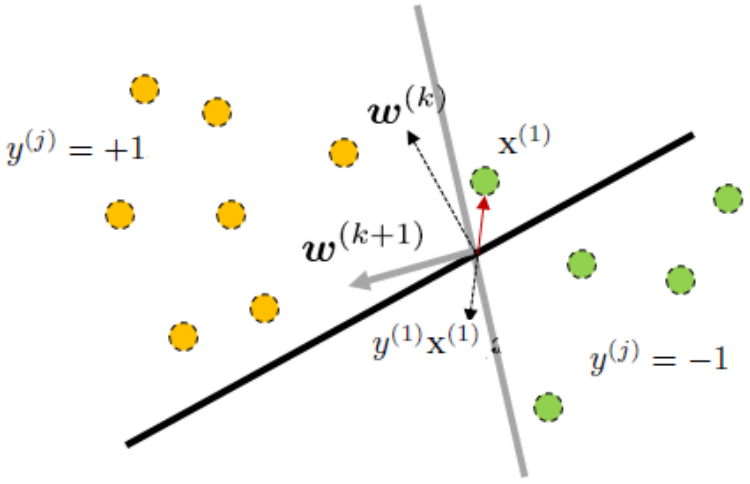


- Initially there is a $\mathbf{w}^{(k)}$
- There is a misclassified training example $\mathbf{x}^{(1)}$

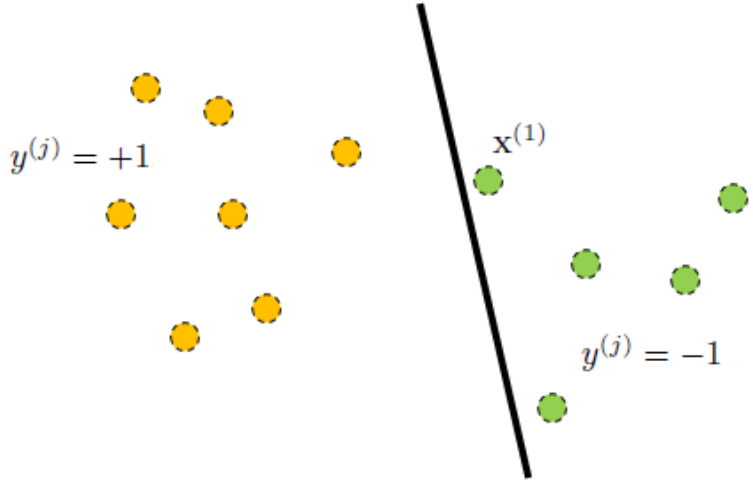


- Perceptron algorithm finds $y^{(1)}\mathbf{x}^{(1)}$
- $y^{(1)}\mathbf{x}^{(1)}$ is in the opposite direction of $\mathbf{x}^{(1)}$

Updating One Example



- $\mathbf{w}^{(k+1)}$ is a linear combination of $\mathbf{w}^{(k)}$ and $y^{(1)}\mathbf{x}^{(1)}$
- $\mathbf{w}^{(k+1)}$ gives a new separating hyperplane



- Now you are happy!

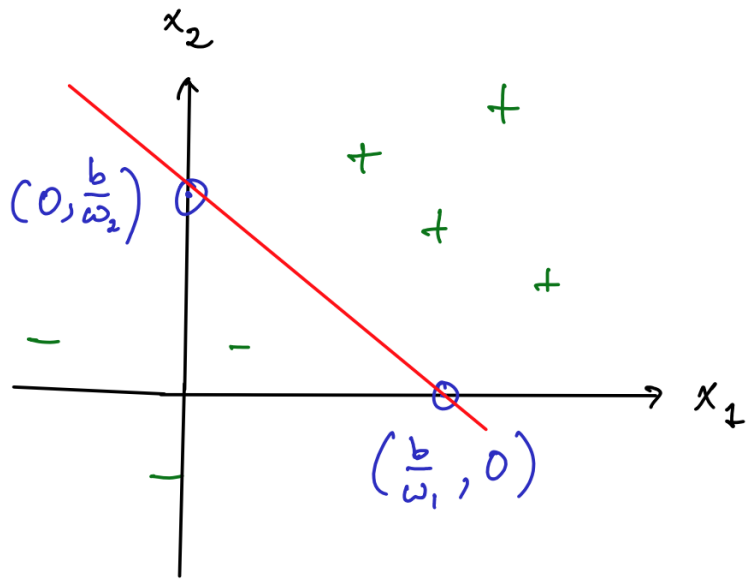
Single-Layer Neural Network

Limitations of Perceptron

- Perceptron is just a linear classifier (Marvin Minsky and Seymour Papert, 1969)

  Example : In two dimensions, we have

$$f(x_1, x_2) = \mathrm{sgn}(\omega_1 x_1 + \omega_2 x_2 - b)$$

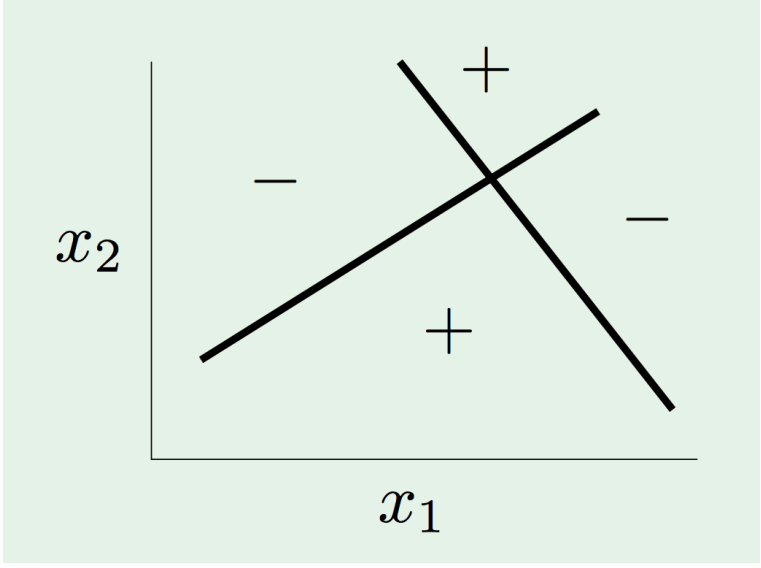


$$\omega_1 x_1 + \omega_2 x_2 - b > 0 \iff x_2 \begin{cases} > \frac{b-\omega_1 x_1}{\omega_2}, & \text{if } w_2 > 0 \\ \leq \frac{b-\omega_1 x_1}{\omega_2}, & \text{otherwise.} \end{cases}$$

- Training does not stop if data are linearly non-separable

Limitations of Perceptron

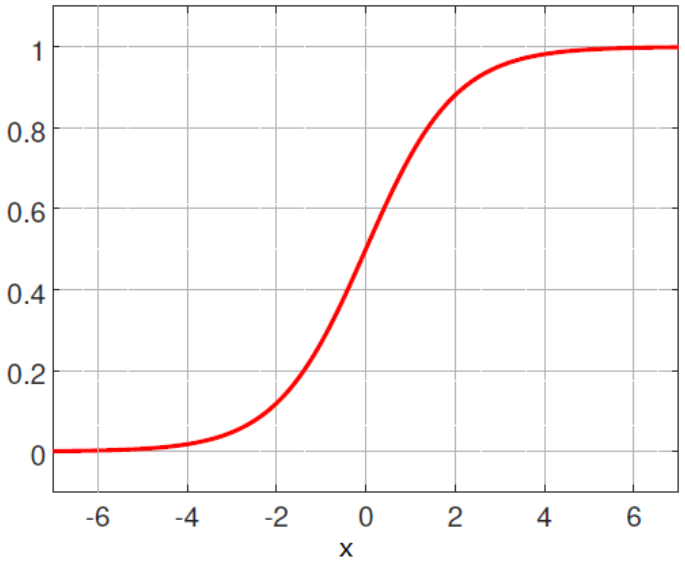
- Weights **w** are adjusted for misclassified data only (correctly classified data are not considered at all)
- Multiple perceptron can form complex decision boundaries (piecewise-linear), but it is hard to train



How can we resolve the problem of training?

Single-Layer Neural Network

- The idea is to replace the sgn function with a differentiable non-linear function

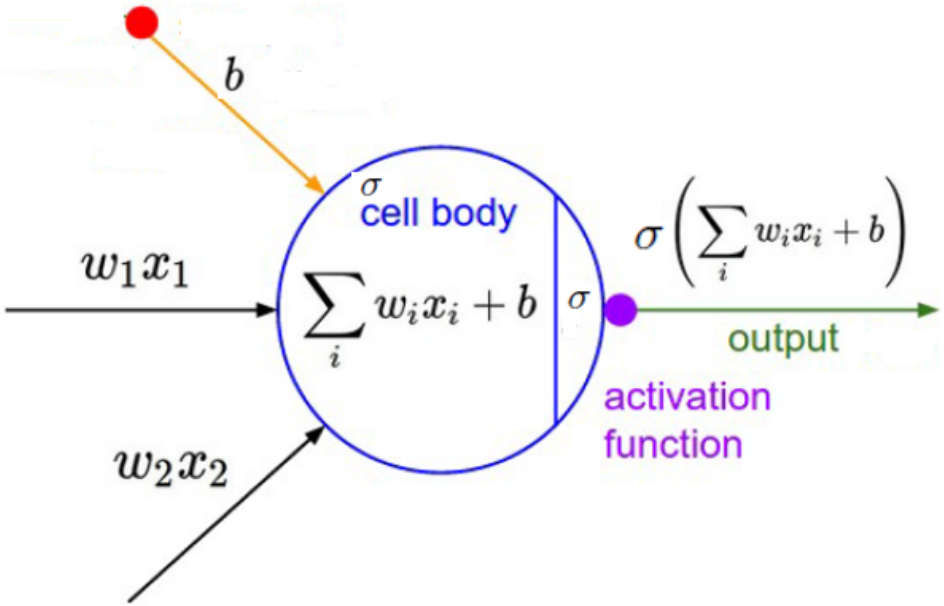


- e.g., sigmoid function

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

- Mapping: $(-\infty, +\infty) \mapsto (0, 1)$
- $\sigma'(x) = \sigma(x)(1 - \sigma(x))$ (exercise)

- This leads to a single-layer neural network



- prediction rule

$$f(\mathbf{x}) = \sigma\Big(\sum_i w_i x_i + b\Big)$$

- real-valued output

Training Single-Layer Neural Network

- Dataset: n input/output pairs $S = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\}$
- Mean-Square Error

$$C(\mathbf{w}) = \frac{1}{2n} \sum_{i=1}^{n} \Big(\underbrace{\sigma(\mathbf{w}^\top \mathbf{x}^{(i)} + b)}_{\text{predicted output}} - \underbrace{y^{(i)}}_{\text{output}} \Big)^2.$$

- No closed form solution for the minimizer of $C(\mathbf{w})$
- Use Gradient Descent to train a $\mathbf{w} \in \mathbb{R}^d$ with a small $C(\mathbf{w})$

$$\mathbf{w}^{\text{new}} = \mathbf{w} - \eta \nabla C(\mathbf{w}) \quad \Longleftrightarrow w_i^{\text{new}} = w_i - \eta \frac{\partial C(\mathbf{w})}{\partial w_i}.$$

- We need to find a way to compute gradient and more specifically $\frac{\partial(\sigma(\mathbf{w}^\top \mathbf{x}+b)-y)^2}{\partial w_i}$
- It suffices to compute $\frac{d(\sigma(wx+b)-y)^2}{dw}$ by noting (univariate)

$$(\sigma(\mathbf{w}^\top \mathbf{x} + b) - y)^2 = \Big(\sigma\big(w_i x_i + \underbrace{\sum_{j:j\neq i} w_j x_j + b}_{:=\bar{b} \text{ is a const w.r.t } w_i} \big) - y \Big)^2$$

Chain Rule and Gradient Descent for Single-Layer Neural Network

Univariate Chain Rule

The composition of two functions f and g is defined by

$$(f \circ g)(x) = f(g(x)).$$

For example, if $f(x) = x^3$ and $g(x) = \sin x$, then

$$f \circ g(x) = f(g(x)) = f(\sin x) = \sin^3(x)$$
$$g \circ f(x) = g(f(x)) = g(x^3) = \sin(x^3).$$

Chain Rule: One Dimensional Case

For one dimensional differentiable functions f and g, we have

$$\frac{d}{dx} f(g(x)) = f'(g(x)) \cdot g'(x)$$

In words, the derivative of $f(g(x))$ is the derivative of the outside function f evaluated at the inside function g, times the derivative of the inside function

Univariate Chain Rule

$$C = \frac{1}{2}\big(\sigma(wx + b) - y\big)^2, \quad w, b \in \mathbb{R}.$$

We can write C by composite functions

$$C = \frac{1}{2}p^2$$
$$p = \sigma(q) - y$$
$$q = wx + b$$

$$\begin{aligned}\frac{\partial C}{\partial w} &= \frac{\partial C}{\partial p}\frac{\partial p}{\partial w} = p\frac{\partial p}{\partial q}\frac{\partial q}{\partial w} \\ &= p\sigma'(q)x = \big(\sigma(q) - y\big)\sigma'(q)x \\ &= \big(\sigma(wx + b) - y\big)\sigma'(wx + b)x\end{aligned}$$

Similarly we can show

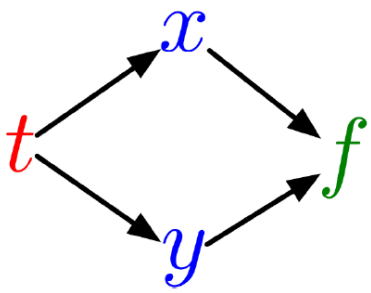
$$\frac{\partial C}{\partial b} = \big(\sigma(wx + b) - y\big)\sigma'(wx + b)$$

$\sigma'(x) = \sigma(x)(1 - \sigma(x))$ for sigmoid activation function

Multivariate Chain Rule

- Suppose we have a function $f(x, y)$ and functions $x(t)$ and $y(t)$. Then

$$\frac{d}{dt} f(x(t), y(t)) = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$



- Example

$$f(x, y) = y + e^{xy}$$
$$x(t) = \cos t$$
$$y(t) = t^2$$

$$\begin{aligned}\frac{df}{dt} &= \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt} \\ &= (ye^{xy}) \cdot (-\sin t) + (1 + xe^{xy}) \cdot 2t.\end{aligned}$$

Chain Rule: Multivariate Case

For $f(u_1, \ldots, u_m)$ with $u_i = g_i(x_1, \ldots, x_n), i = 1, \ldots, m$, then

$$\frac{\partial f}{\partial x_i} = \sum_{j=1}^{m} \frac{\partial f}{\partial u_j} \cdot \frac{\partial u_j}{\partial x_i}.$$

Gradient Descent for Single-Layer Neural Network

- Let $C_i(\mathbf{w}) = \frac{1}{2}(\sigma(\mathbf{w}^\top \mathbf{x}^{(i)} + b) - y^{(i)})^2$. Then $C(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} C_i(\mathbf{w})$
- In the previous slide, we derive

$$\frac{\partial C_i}{\partial w_j} = \big(\sigma(\mathbf{w}^\top \mathbf{x}^{(i)} + b) - y^{(i)}\big)\sigma'(\mathbf{w}^\top \mathbf{x}^{(i)} + b)x_j^{(i)} \tag{1}$$

$$\frac{\partial C_i}{\partial b} = \big(\sigma(\mathbf{w}^\top \mathbf{x}^{(i)} + b) - y^{(i)}\big)\sigma'(\mathbf{w}^\top \mathbf{x}^{(i)} + b). \tag{2}$$

Vectorization of (1): $$\frac{\partial C_i}{\partial \mathbf{w}} = \big(\sigma(\mathbf{w}^\top \mathbf{x}^{(i)} + b) - y^{(i)}\big)\sigma'(\mathbf{w}^\top \mathbf{x}^{(i)} + b)\mathbf{x}^{(i)}. \tag{3}$$

Gradient Descent for One-Layer NN

1: Initialize $\mathbf{w}^{(1)} = 0, b^{(1)} = 0$
2: **for** $t = 1, 2, \ldots, T$ **do** ▷ T is the number of iterations
3: Use (3), (2) to compute gradients

$$\nabla_{\mathbf{w}} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial C_i(\mathbf{w}^{(t)})}{\partial \mathbf{w}^{(t)}}, \qquad \nabla_b = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial C_i(\mathbf{w}^{(t)})}{\partial b^{(t)}}$$

4: Update the model

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta_t \nabla_{\mathbf{w}}, \qquad b^{(t+1)} = b^{(t)} - \eta_t \nabla_b.$$

Single-Layer Neural Network

Single-layer neural network still leads to linear classifiers!

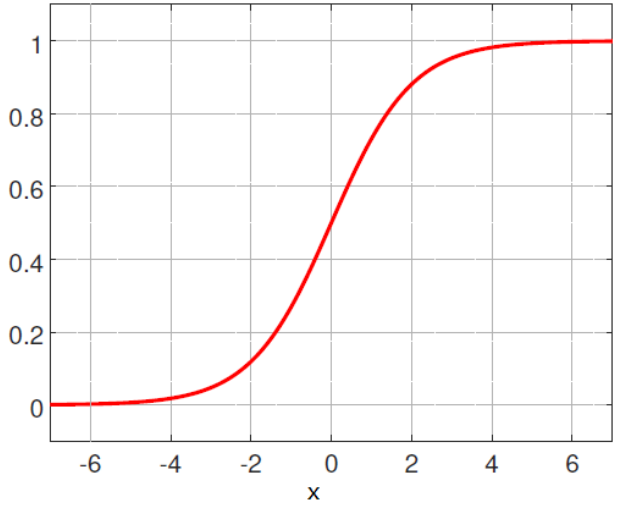
$$f(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$$

We do classification as follows

$$\hat{y} = 1 \text{ iff } f(\mathbf{x}) \geq 1/2$$

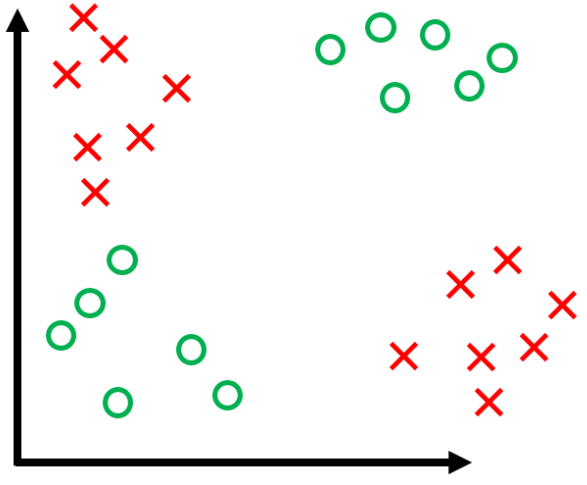
$$f(\mathbf{x}) \geq 1/2 \iff \mathbf{w}^\top \mathbf{x} + b \geq 0$$

$$\hat{y} = 1 \text{ iff } \mathbf{w}^\top \mathbf{x} + b \geq 0$$



How to handle nonlinear data?

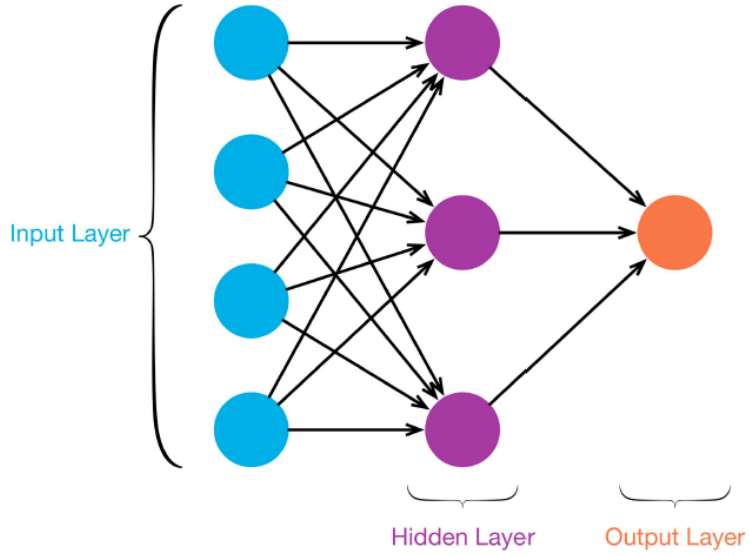
Use multiple neurons!



Feed-Forward Neural Network

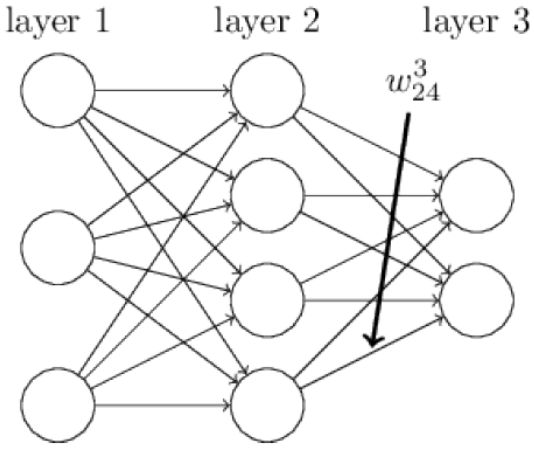
Feed-Forward Neural Network

- We can connect lots of units/neurons together into a directed acyclic graph
- Typically, units are grouped together into layers
- Each unit (simplest case) in a layer is connected to each unit in the next layer (fully connected)
- We call it Feed-Forward neural network or multi-layer perceptron (MLP)
- The number of units in the input layer is equal to the number d of features
- An example of MLP with 3 layers



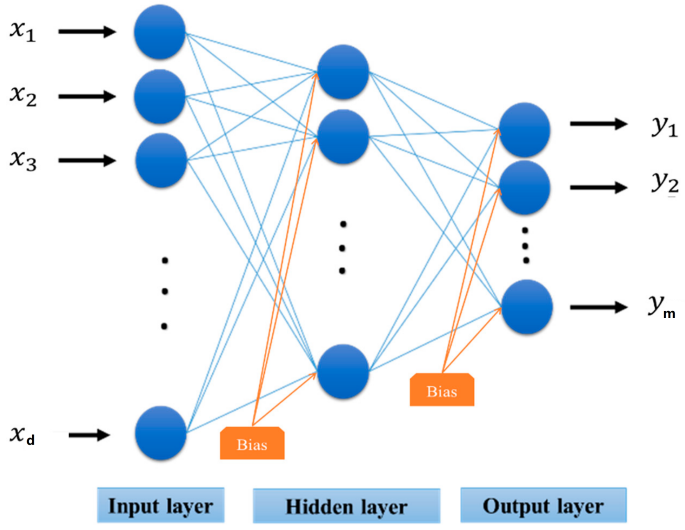
By introducing more layers we improve the expression power of MLP!

MLP: Weights and Bias



w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

weights



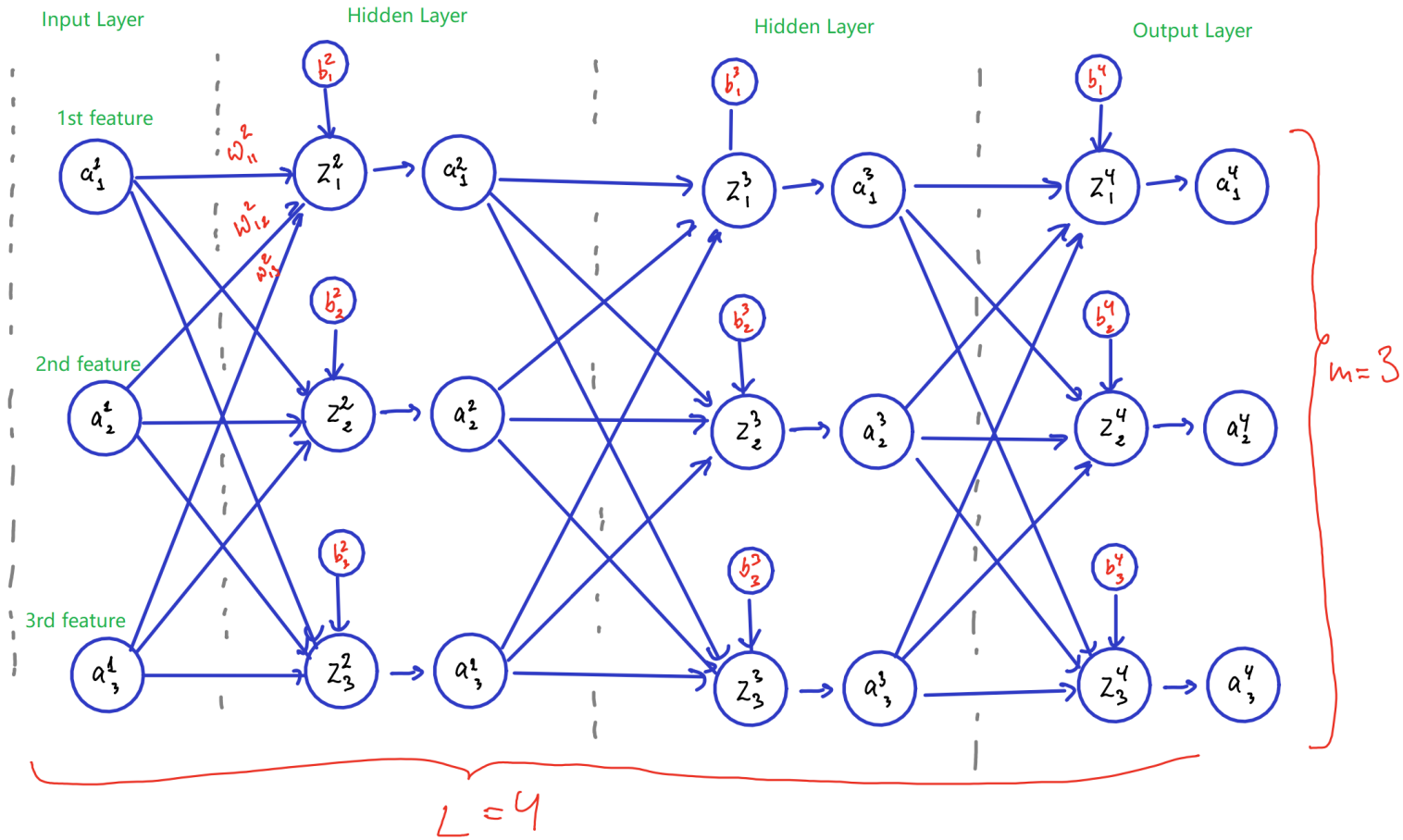
bias

Each edge in MLP is associated with a number called weight

- ω^3_{24}: from 4-th neuron in 2nd layer to 2-nd neuron in 3-rd layer

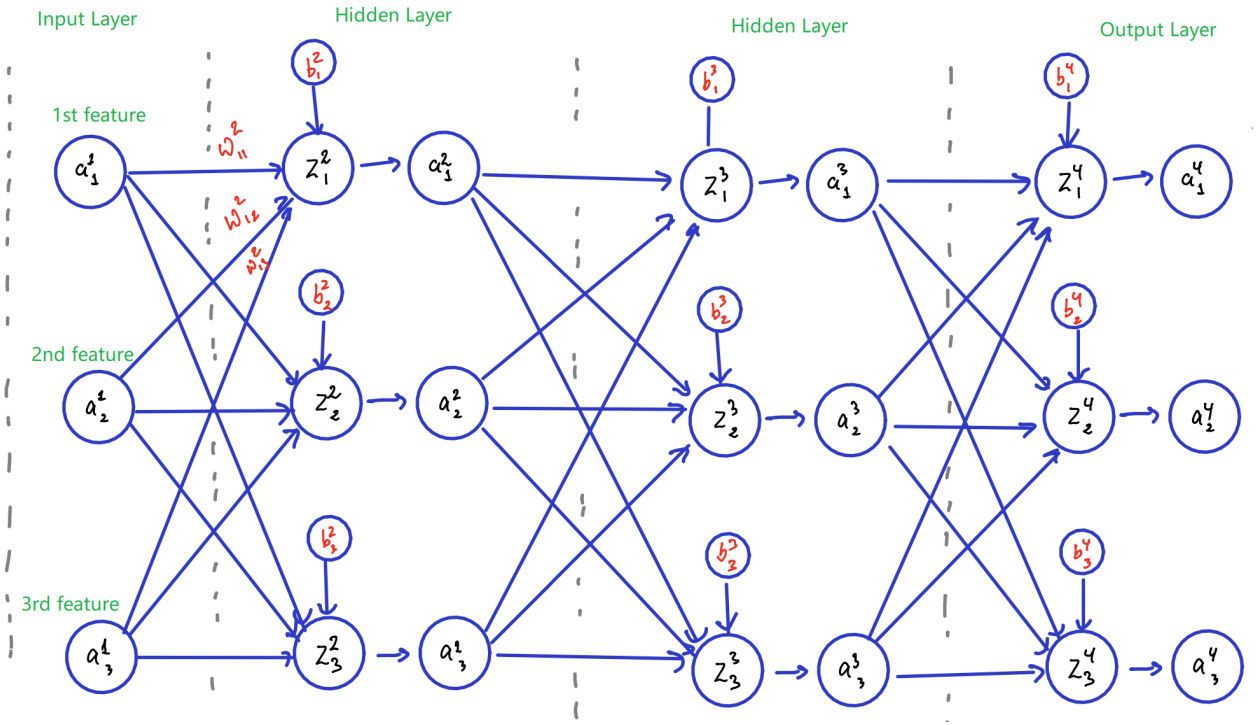
Each node is associated with a number called bias

- b^2_3: the 3-rd node in the 2-nd layer



- L: number of layers (1-st layer is "input layer", L-th layer is "output layer")
- m: "width" of network (can vary between layers)
- ω_{jk}^{ℓ}: "weight" of connection between k-th unit in layer $\ell-1$, to j-th unit in layer ℓ
- b_j^{ℓ}: "bias" of j-th unit in layer ℓ
- $z_j^{\ell} = \sum_k \omega_{jk}^{\ell} a_k^{\ell-1} + b_j^{\ell}$: weighted input to unit j in layer ℓ
- $a_j^{\ell} = \sigma(z_j^{\ell})$: "activation" of unit j in layer ℓ, where σ is an "activation function"

Trainable Parameters



We collect weights into several matrices, and bias into several vectors (trainable parameters)

$$\mathbf{W}^{\ell} = \begin{pmatrix} \omega_{11}^{\ell} & \omega_{12}^{\ell} & \cdots & \omega_{1m}^{\ell} \\ \omega_{21}^{\ell} & \omega_{22}^{\ell} & \cdots & \omega_{2m}^{\ell} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_{m1}^{\ell} & \omega_{m2}^{\ell} & \cdots & \omega_{mm}^{\ell} \end{pmatrix}, \quad \mathbf{b}^{\ell} = \begin{pmatrix} b_1^{\ell} \\ b_2^{\ell} \\ \vdots \\ b_m^{\ell} \end{pmatrix}, \qquad \ell = 2, \ldots, L.$$

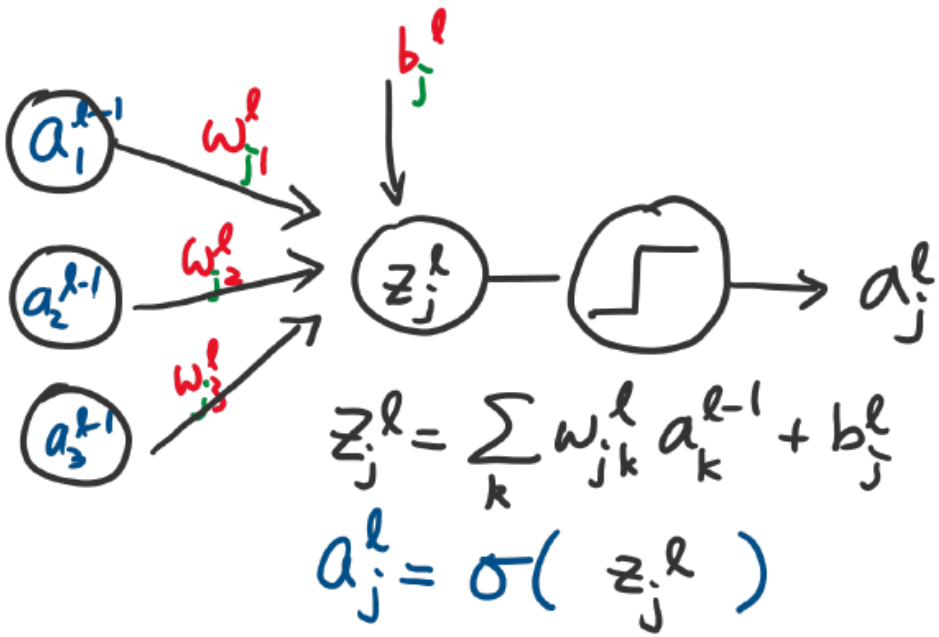
The number of trainable parameters is $(m^2 + m) * (L - 1)$

MLP: Main Equation

- This is the main equation

$$a_j^\ell = \sigma\Big(\underbrace{\sum_k \omega_{jk}^\ell a_k^{\ell-1} + b_j^\ell}_{z_j^\ell}\Big) \quad \text{or} \quad a_j^\ell = \sigma(z_j^\ell)$$

- a_j^ℓ: activation, z_j^ℓ: intermediate



Vectorization

We can relate $(a_1^{\ell-1}, \ldots, a_m^{\ell-1})$ to $(z_1^\ell, \ldots, z_m^\ell)$ in terms of matrix

$$\begin{pmatrix} z_1^\ell = \sum_k \omega_{1k}^\ell a_k^{\ell-1} + b_1^\ell \\ z_2^\ell = \sum_k \omega_{2k}^\ell a_k^{\ell-1} + b_2^\ell \\ \vdots \\ z_m^\ell = \sum_k \omega_{mk}^\ell a_k^{\ell-1} + b_m^\ell \end{pmatrix} = \begin{pmatrix} \omega_{11}^\ell & \omega_{12}^\ell & \cdots & \omega_{1m}^\ell \\ \omega_{21}^\ell & \omega_{22}^\ell & \cdots & \omega_{2m}^\ell \\ \vdots & \vdots & \vdots & \vdots \\ \omega_{m1}^\ell & \omega_{m2}^\ell & \cdots & \omega_{mm}^\ell \end{pmatrix} \begin{pmatrix} a_1^{\ell-1} \\ a_2^{\ell-1} \\ \vdots \\ a_m^{\ell-1} \end{pmatrix} + \begin{pmatrix} b_1^\ell \\ b_2^\ell \\ \vdots \\ b_m^\ell \end{pmatrix}$$

This amounts to saying

$$\mathbf{z}^\ell = \mathbf{W}^\ell \mathbf{a}^{\ell-1} + \mathbf{b}^\ell$$
$$\mathbf{a}^\ell = \sigma(\mathbf{z}^\ell)$$
$$\ell = 2, \ldots, L.$$

$$\mathbf{W}^\ell := \begin{pmatrix} \omega_{11}^\ell & \omega_{12}^\ell & \cdots & \omega_{1m}^\ell \\ \omega_{21}^\ell & \omega_{22}^\ell & \cdots & \omega_{2m}^\ell \\ \vdots & \vdots & \vdots & \vdots \\ \omega_{m1}^\ell & \omega_{m2}^\ell & \cdots & \omega_{mm}^\ell \end{pmatrix}$$

$$\mathbf{z}^\ell := \begin{pmatrix} z_1^\ell \\ z_2^\ell \\ \vdots \\ z_m^\ell \end{pmatrix}, \quad \mathbf{a}^\ell := \begin{pmatrix} a_1^\ell \\ a_2^\ell \\ \vdots \\ a_m^\ell \end{pmatrix}, \quad \mathbf{b}^\ell := \begin{pmatrix} b_1^\ell \\ b_2^\ell \\ \vdots \\ b_m^\ell \end{pmatrix}$$

$$\mathbf{a}^\ell = \sigma(\mathbf{z}^\ell) = \sigma\left(\mathbf{W}^\ell \mathbf{a}^{\ell-1} + \mathbf{b}^\ell\right)$$

Forward Propagation to Compute Prediction

$$\mathbf{W}^\ell = \begin{pmatrix} \omega_{11}^\ell & \omega_{12}^\ell & \cdots & \omega_{1m}^\ell \\ \omega_{21}^\ell & \omega_{22}^\ell & \cdots & \omega_{2m}^\ell \\ \vdots & \vdots & \vdots & \vdots \\ \omega_{m1}^\ell & \omega_{m2}^\ell & \cdots & \omega_{mm}^\ell \end{pmatrix}, \mathbf{z}^\ell = \begin{pmatrix} z_1^\ell \\ z_2^\ell \\ \vdots \\ z_m^\ell \end{pmatrix}, \quad \mathbf{a}^\ell = \begin{pmatrix} a_1^\ell \\ a_2^\ell \\ \vdots \\ a_m^\ell \end{pmatrix}, \quad \mathbf{b}^\ell = \begin{pmatrix} b_1^\ell \\ b_2^\ell \\ \vdots \\ b_m^\ell \end{pmatrix}$$

Forward Propagation

Input: $\mathbf{x} \in \mathbb{R}^d, \mathbf{W}^\ell, \mathbf{b}^\ell, \ell = 2, \ldots, L$

Output: $\mathbf{a}^L$

1: Set $\mathbf{a}^1 = \mathbf{x}$ ▷ We assign the original feature to the first layer

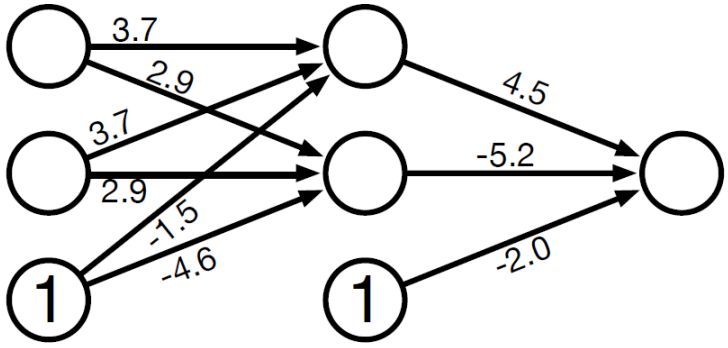
2: **for** $\ell = 2, \ldots, L$ **do** ▷ from bottom to top

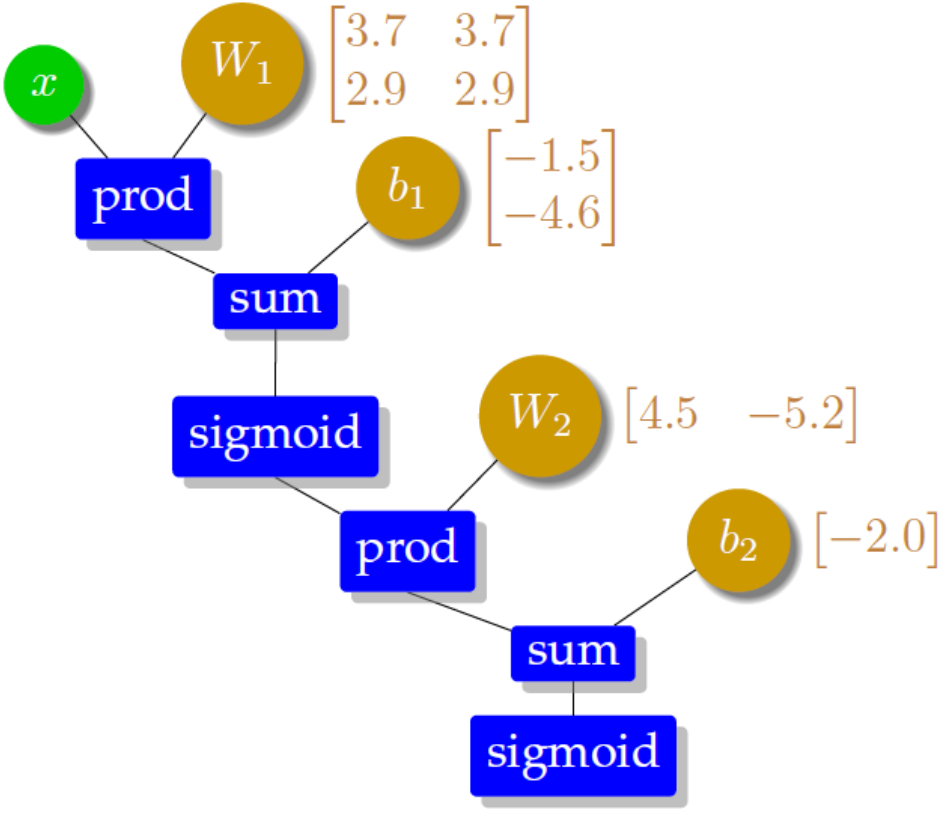
3: Compute the activations in ℓ-th layer via

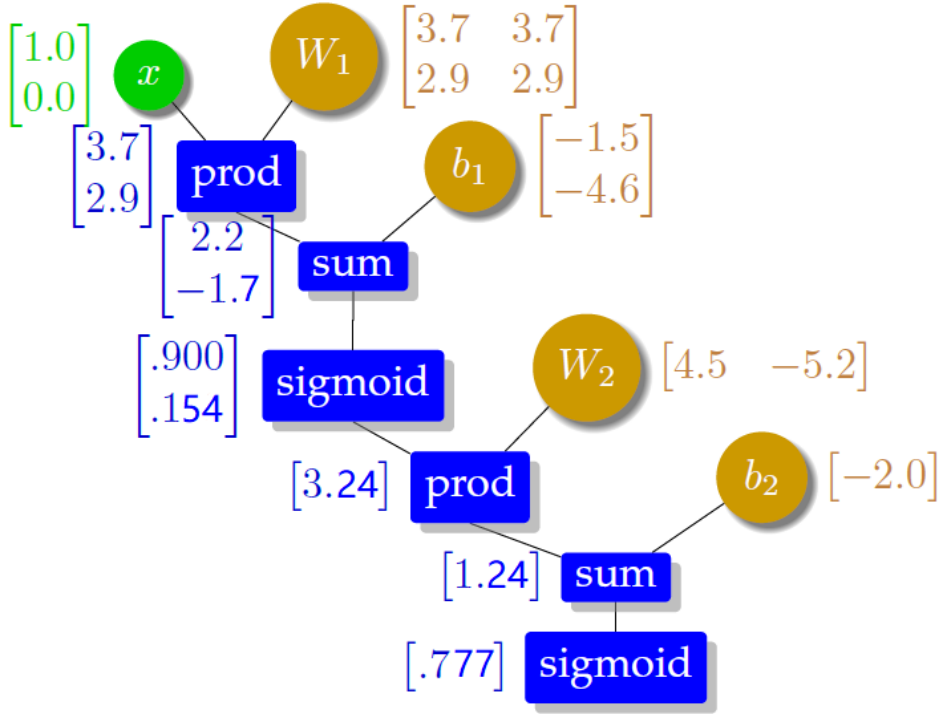
$$\mathbf{a}^\ell = \sigma\left(\mathbf{W}^\ell \mathbf{a}^{\ell-1} + \mathbf{b}^\ell\right)$$

$\mathbf{a}^\ell$ can be considered as new feature learned from data!

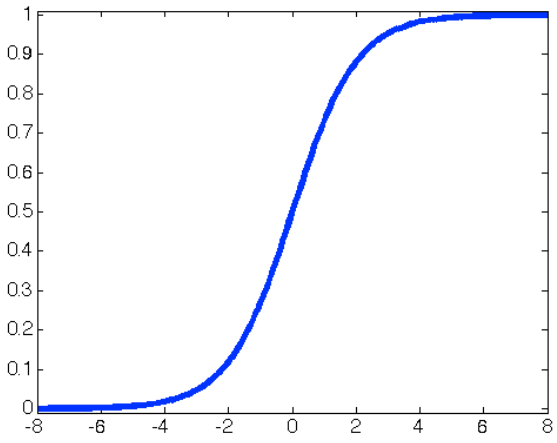
Forward Propagation: Example

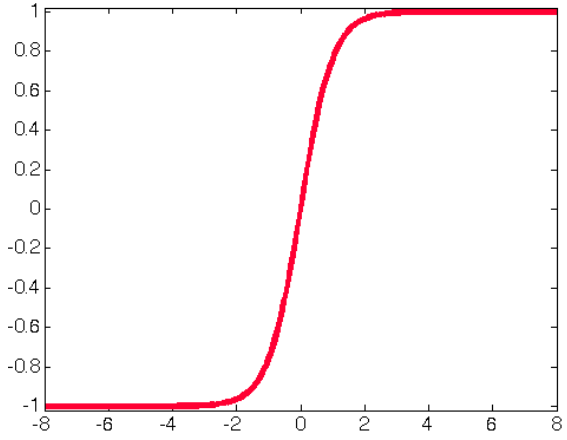


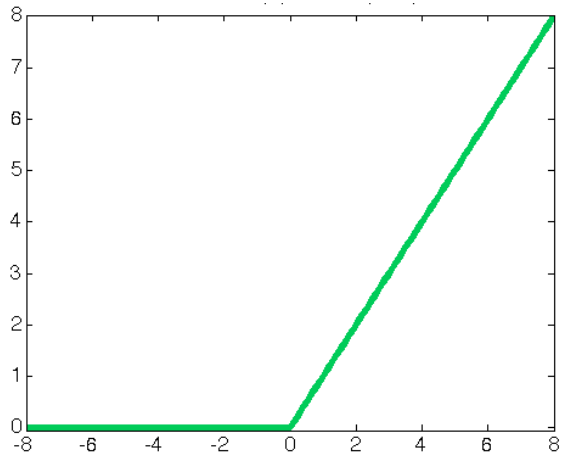




Continuous Activation Functions







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

$$(-\infty, \infty) \mapsto (0, 1)$$

$$\mathrm{Tanh}(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

$$(-\infty, \infty) \mapsto (-1, 1)$$

$$\mathrm{ReLU}(x) = \max(0, x)$$

$$(-\infty, \infty) \mapsto (0, \infty)$$

- ReLU is simple to compute but not differentiable
- Sigmoid and Tanh are differentiable but require computation cost (exponential function)

Training MLPs

- Here is a loss function we want to minimize (at a point $(\mathbf{x}, y)$)

$$C_{(\mathbf{x},y)}(\mathbf{W}^2, \ldots, \mathbf{W}^L, \mathbf{b}^2, \ldots, \mathbf{b}^L) = \Big(\underbrace{y - \sigma(\mathbf{W}^L(\sigma \cdots \sigma(\underbrace{\mathbf{W}^2\mathbf{x} + \mathbf{b}^2}_{\mathbf{z}^2})) + \mathbf{b}^L)}_{\mathbf{z}^L}\Big)^2$$

- To optimize, we need gradient descent. For example, for $\mathbf{W}^2$

$$\mathbf{W}^2 \leftarrow \mathbf{W}^2 - \frac{\eta}{n} \sum_{(\mathbf{x},y)\in S} \nabla C_{(\mathbf{x},y)}(\mathbf{W}^2, \ldots, \mathbf{W}^L, \mathbf{b}^2, \ldots, \mathbf{b}^L)$$

- But we need to compute gradients for for all $\mathbf{W}^\ell, \mathbf{b}^\ell$ and there are a lot of sigmoid functions
- Brute force is impossible if there are lots of parameters

We need backpropagation, which is chain rule + very careful book keeping!

Summary

Summary

- Perceptron
  - corresponds to linear separator
  - perceptron algorithm stops for linearly separable data
  - hard to train in general
- Single-layer NN (no hidden layer)
  - use sigmoid as activation function
  - can be trained by gradient descent
  - chain rule
- Multi-layer NN
  - forward propagation to compute function
  - can be considered as a feature learning algorithm
  - activation function

Next Lecture

Computation Graph and Back Propagation!