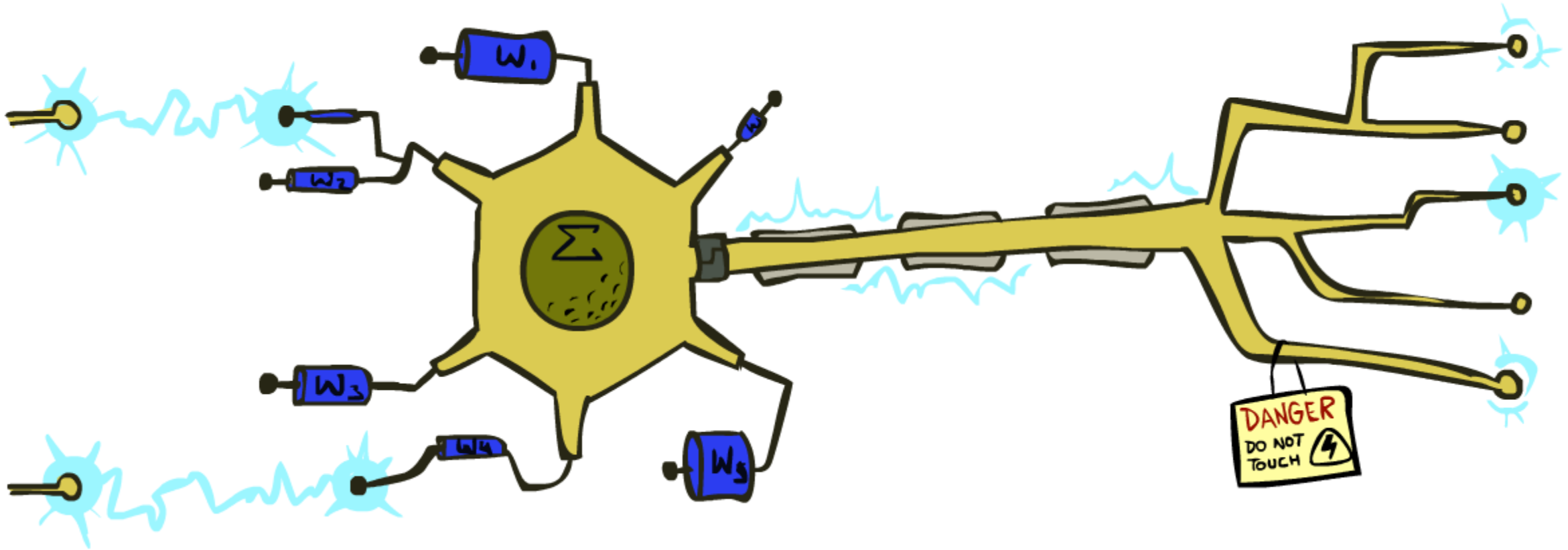
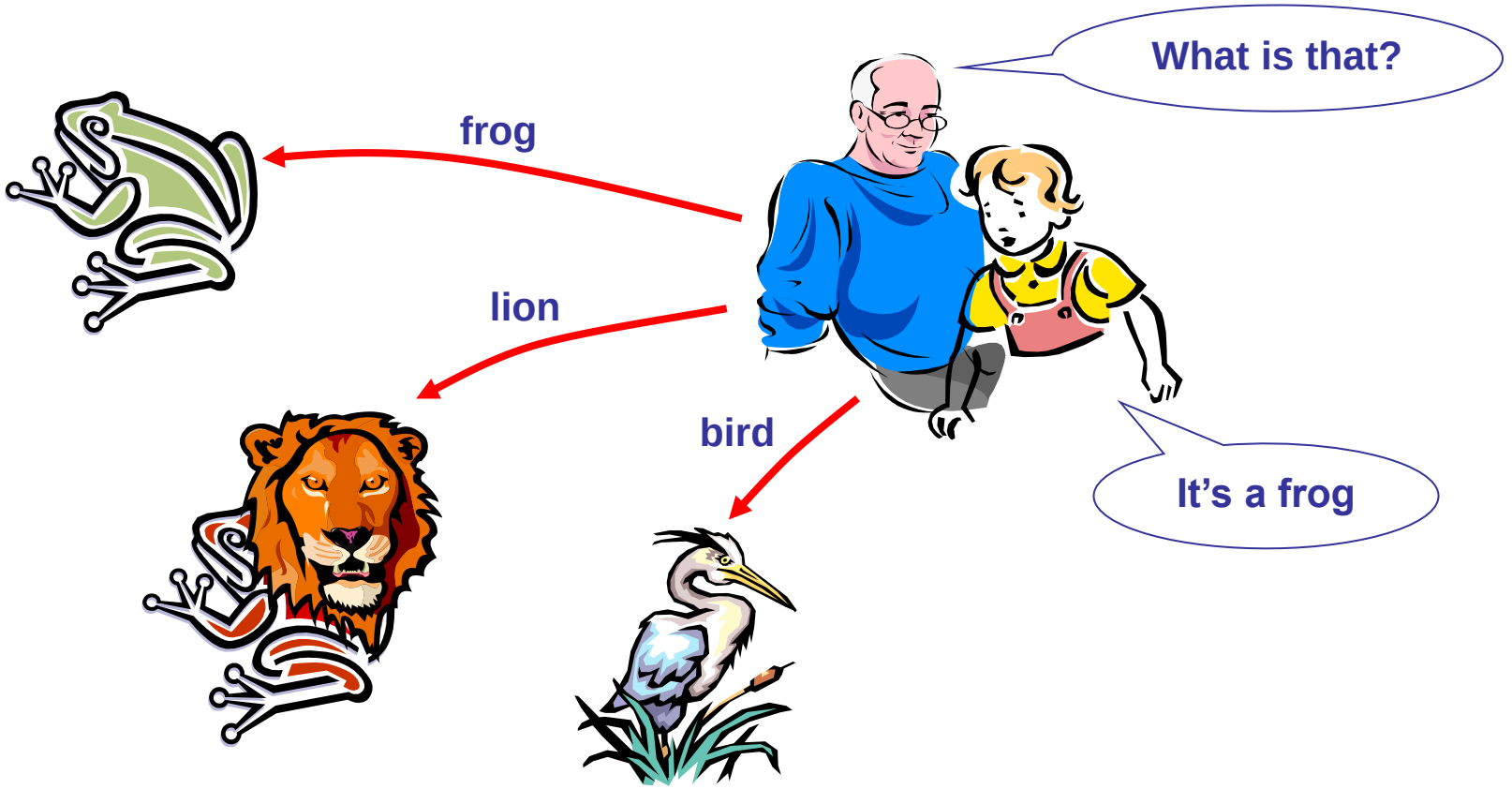
Artificial Intelligence

Neural Network



The idea of ANNs..?

- ANNs learn relationship between cause and effect or organize large volumes of data into orderly and informative patterns.



Artificial Neural Network

- **ANN Definition**:

"Data processing system consisting of a large number of simple, highly interconnected processing elements (artificial neurons) in an architecture inspired by the structure of the cerebral cortex of the brain"

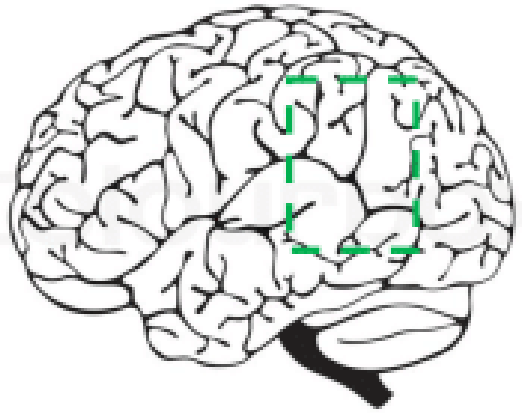
(Tsoukalas & Uhrig, 1997)

- **Neural network:** *information processing paradigm inspired by biological nervous systems, such as our brain*
- **Structure:** large number of highly interconnected processing elements (*neurons*) working together
- Like people, they learn *from experience* (by example)

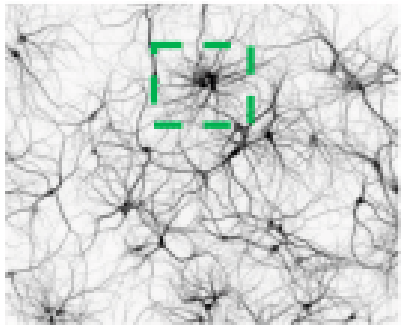
Artificial Neural Networks

- Computational models inspired by the human brain:
  - Algorithms that try to mimic the brain.
  - Massively parallel, distributed system, made up of simple processing units (neurons)
  - Synaptic connection strengths among neurons are used to store the acquired knowledge.
  - Knowledge is acquired by the network from its environment through a learning process

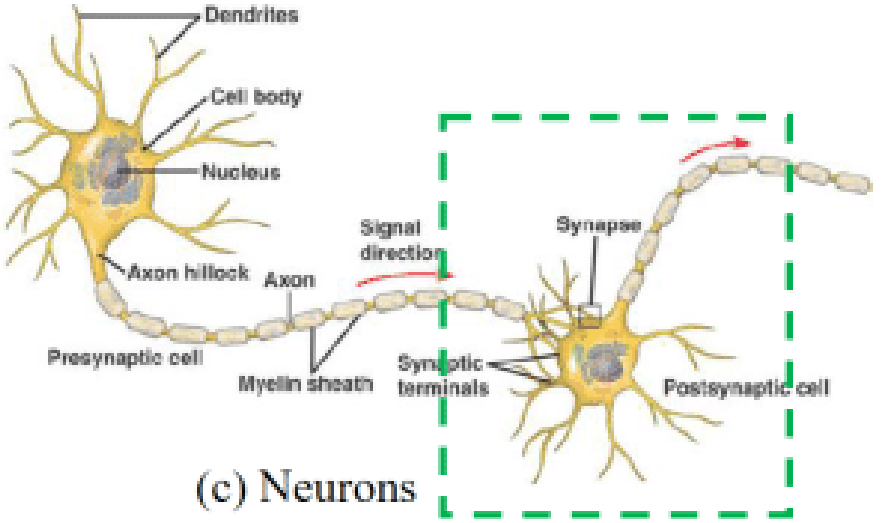
Biological Inspirations



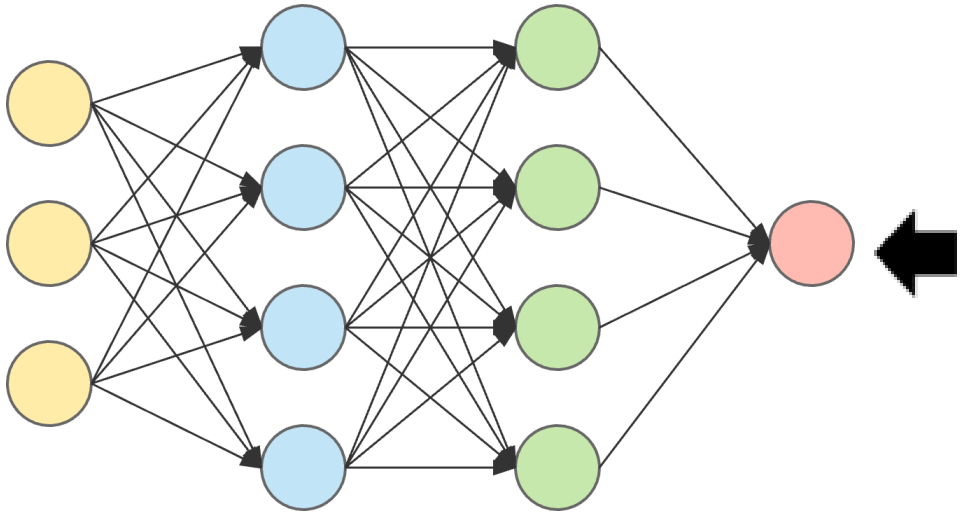
(a) Brain



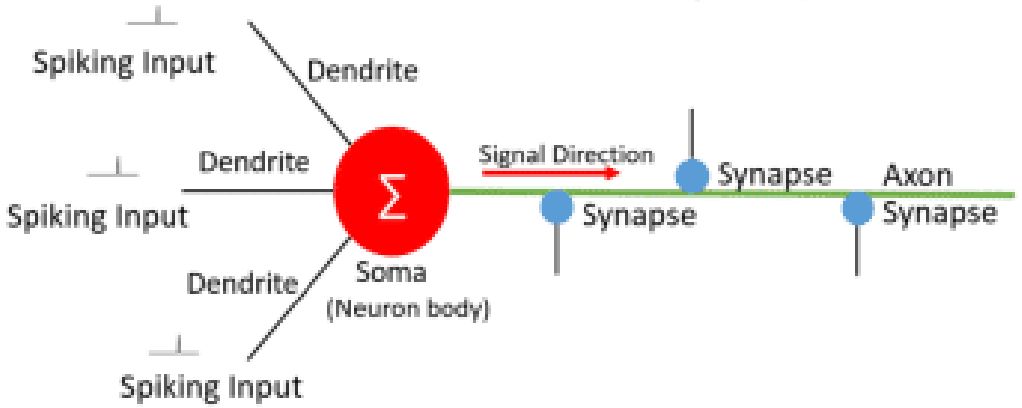
(b) Neuron network



(c) Neurons



(e) Neuron Network

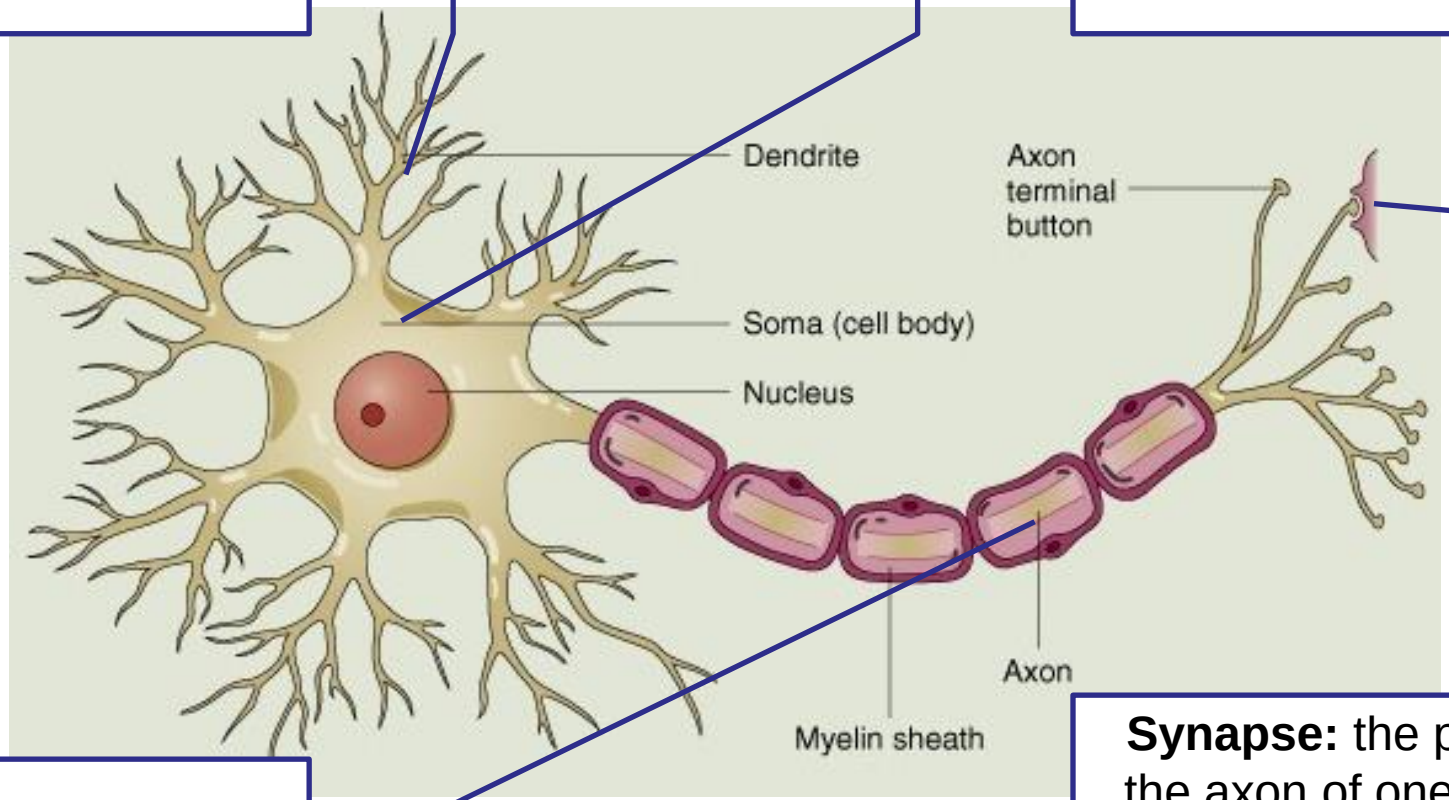


(d) Neuron Structure

Biological Neuron

Dendrites: nerve fibres carrying electrical signals to the cell

Soma (Cell body): computes a non-linear function of its inputs

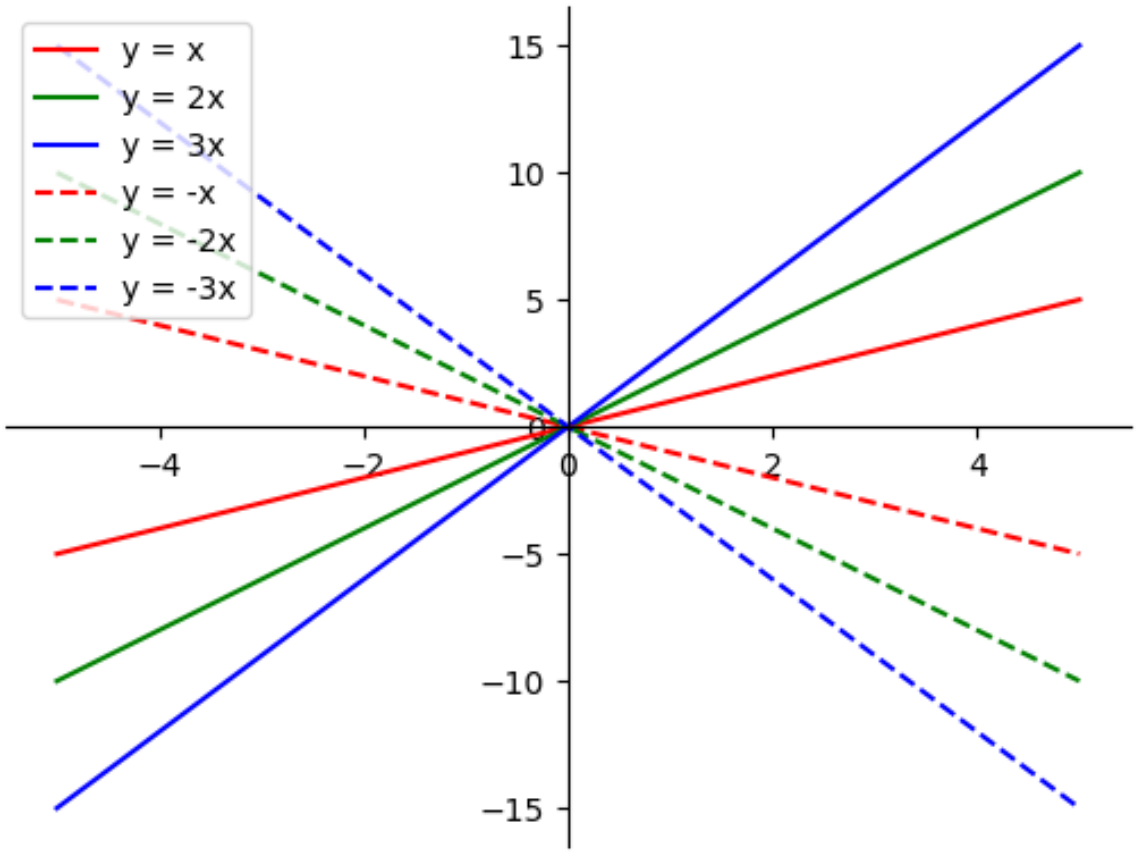


Axon: single long fiber that carries the electrical signal from the cell body to other neurons

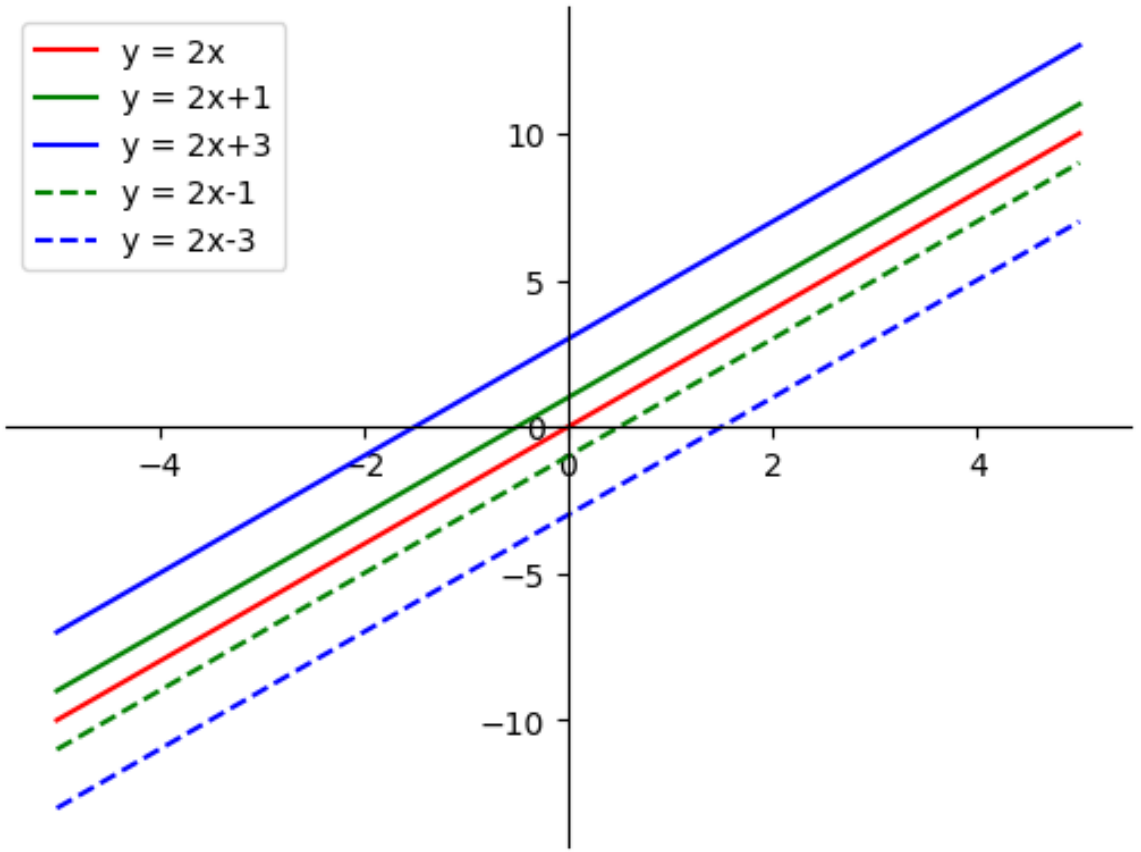
Synapse: the point of contact between the axon of one cell and the dendrite of another, regulating a chemical connection whose strength affects the input to the cell.

$y = wx + b$

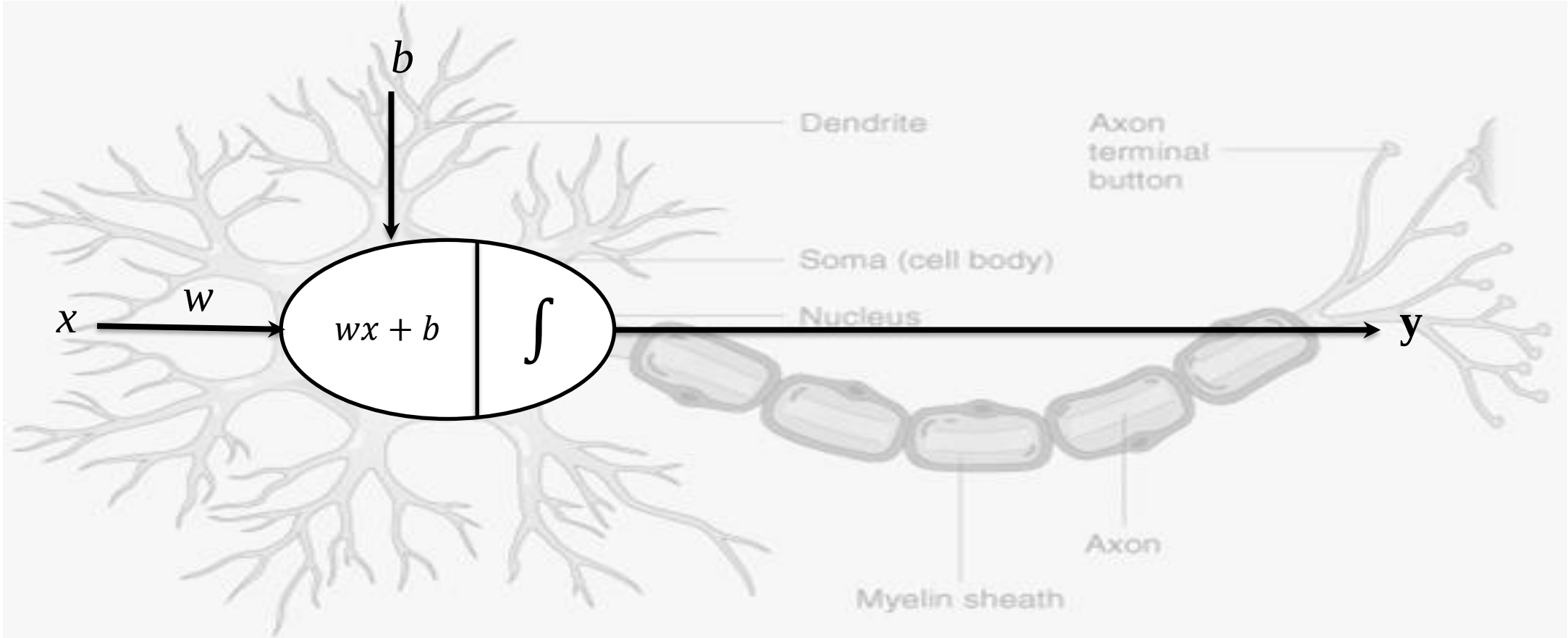
Effect of w



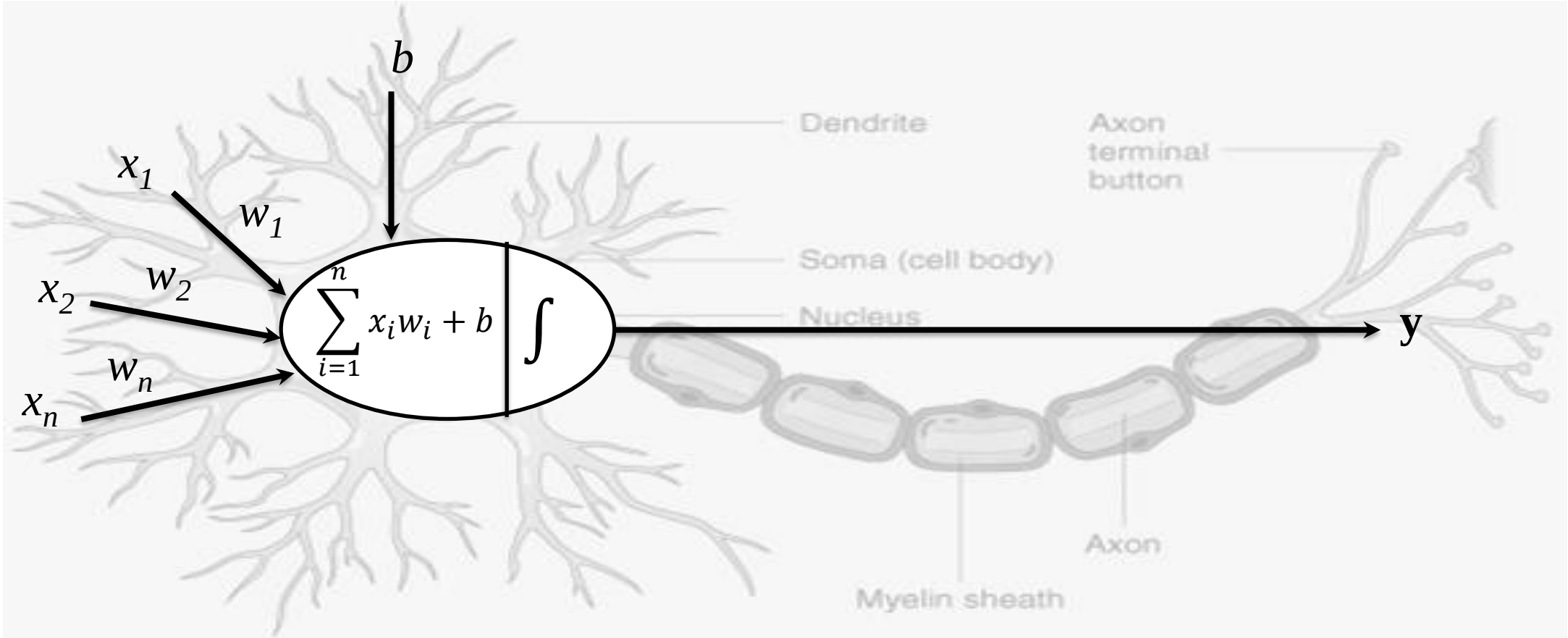
Effect of b



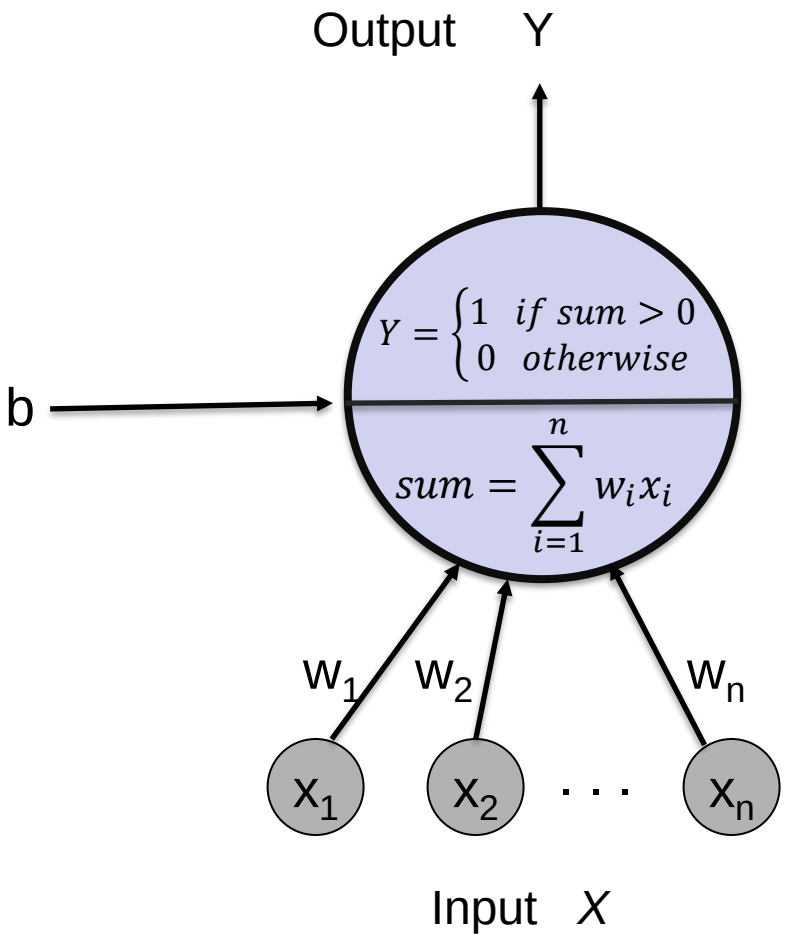
Computer neuron to mimic the properties of biological neurons:



Computer neuron to mimic the properties of biological neurons:



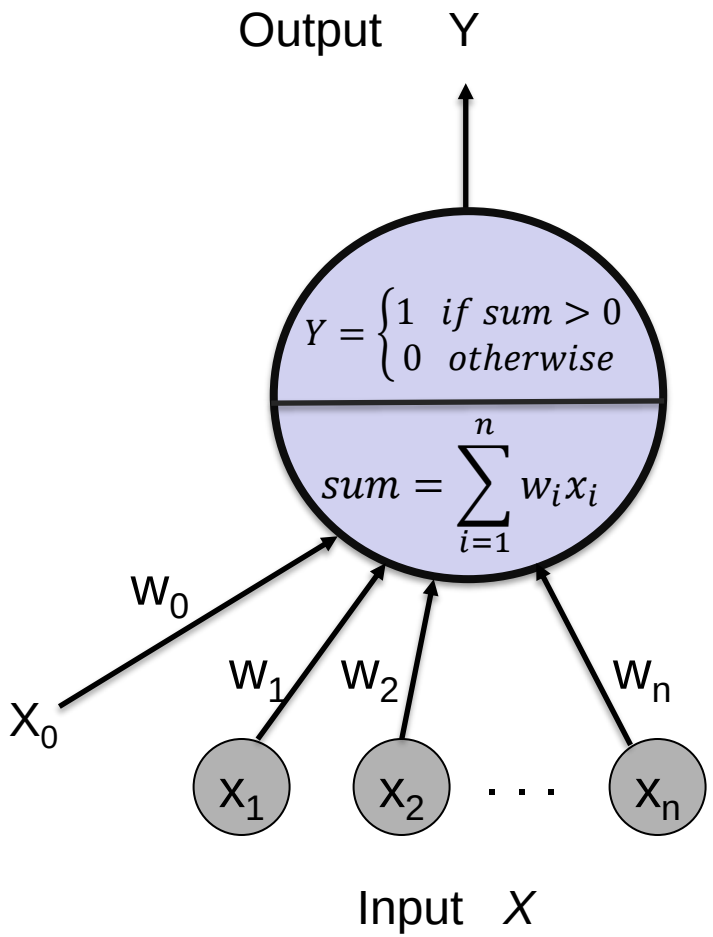
Perceptron



- Perceptron
  - computes the weighted sum of its input (called its net input)
  - adds its bias
  - passes this value through an activation function
- We say that the neuron "fires" (i.e. becomes active) if its output is above zero.
- Bias can be incorporated as another weight clamped to a fixed input of +1.0
  - This extra free variable (bias) makes the neuron more powerful.

$$net = \sum_{i=1}^{n} \omega_i x_i + b$$

Perceptron

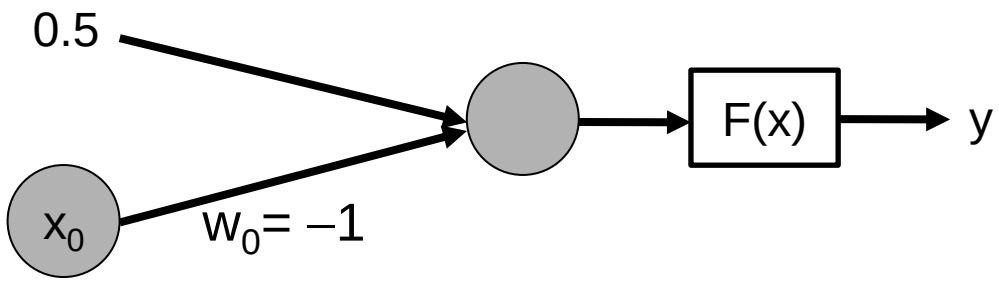


$$net = \sum_{i=0}^{n} \omega_i x_i$$

- Perceptron
  - computes the weighted sum of its input (called its net input)
  - adds its bias
  - passes this value through an activation function
- We say that the neuron "fires" (i.e. becomes active) if its output is above zero.
- Bias can be incorporated as another weight clamped to a fixed input of +1.0
  - This extra free variable (bias) makes the neuron more powerful.

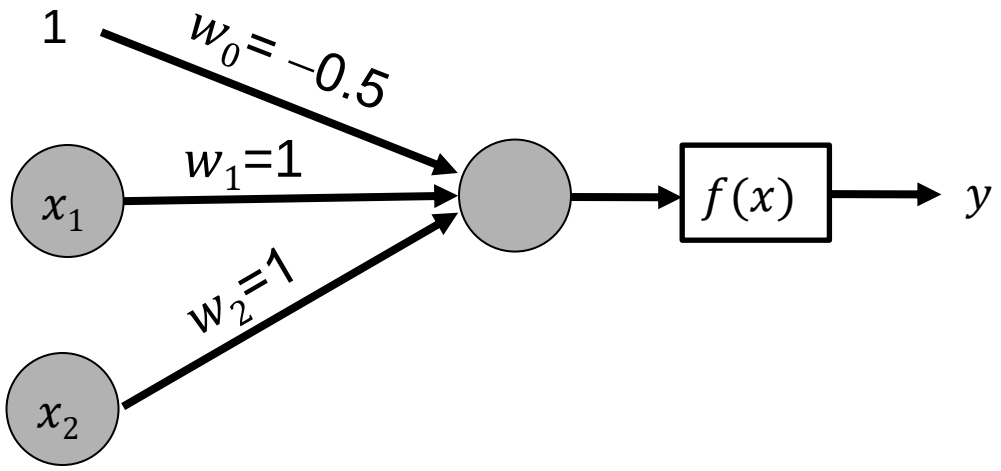
Inverter

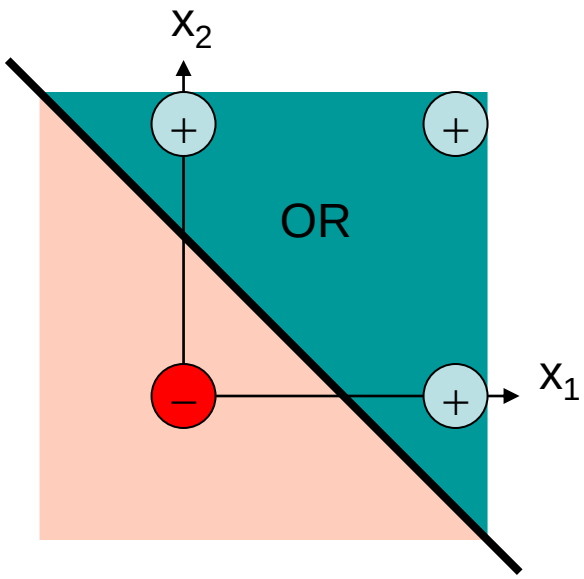
input X_1	output Y
0	1
1	0



Boolean OR

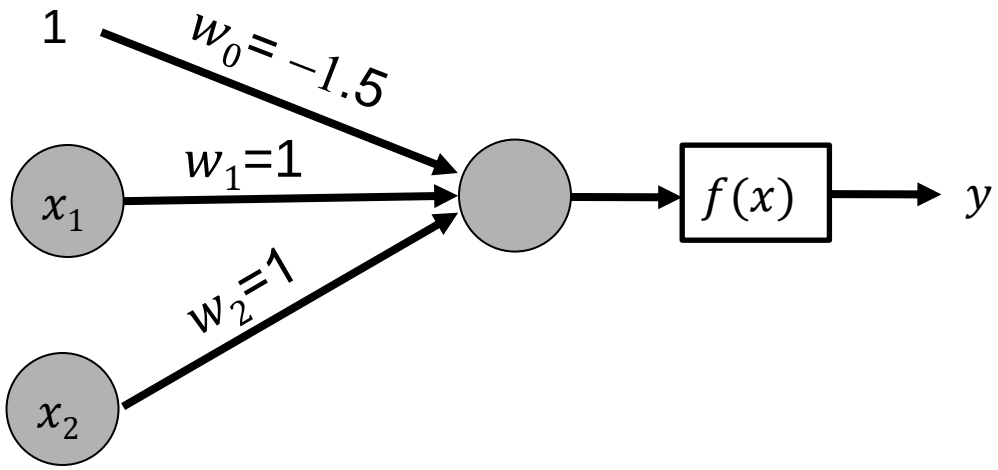
input x1	input x2	output Y
0	0	0
0	1	1
1	0	1
1	1	1

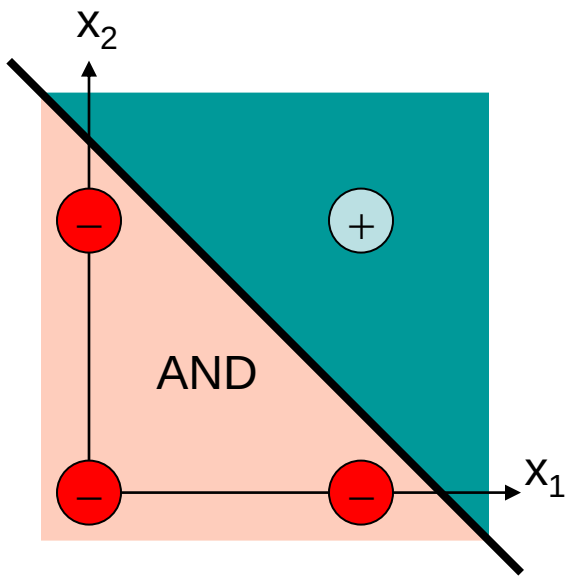




Boolean AND

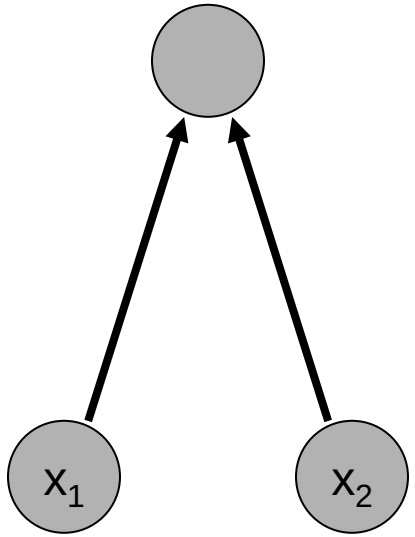
input x1	input x2	ouput
0	0	0
0	1	0
1	0	0
1	1	1

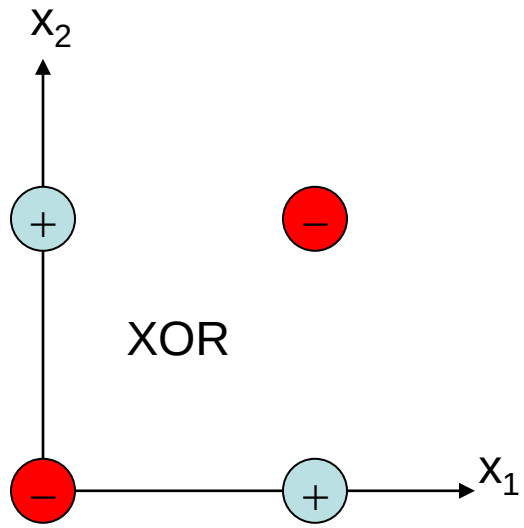




Boolean XOR

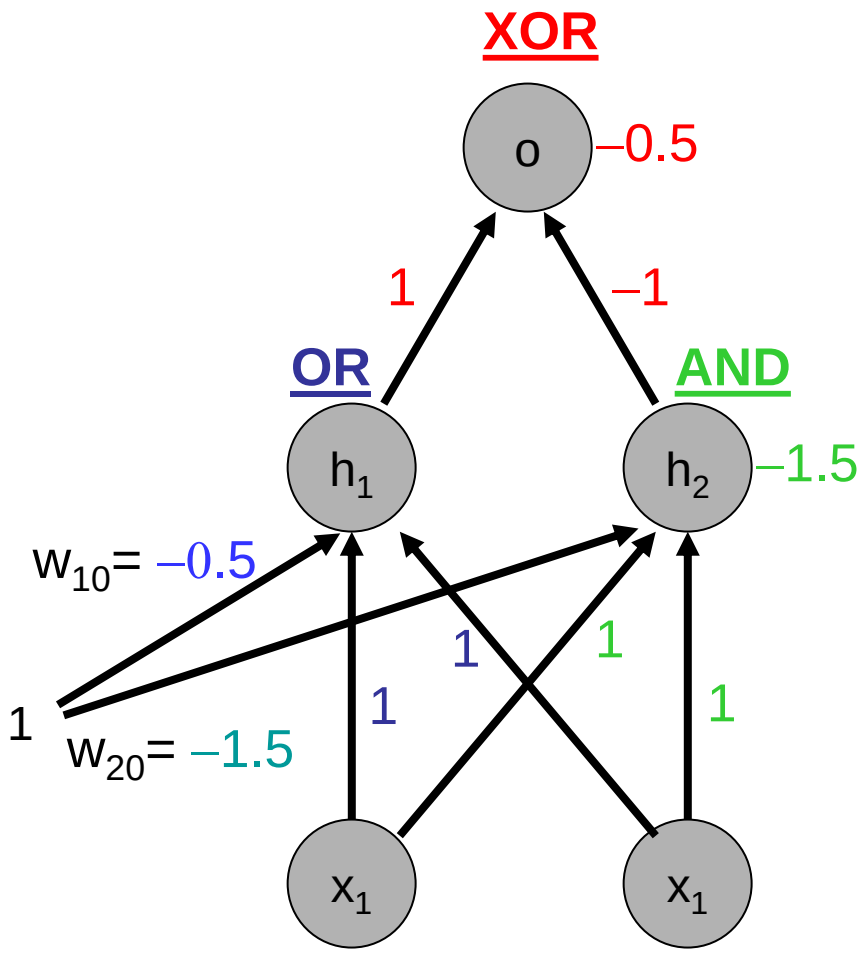
input x1	input x2	Output Y
0	0	0
0	1	1
1	0	1
1	1	0

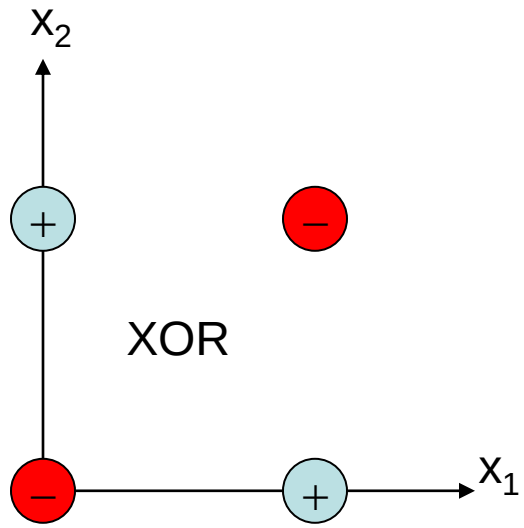




Boolean XOR

input x1	input x2	ouput
0	0	0
0	1	1
1	0	1
1	1	0

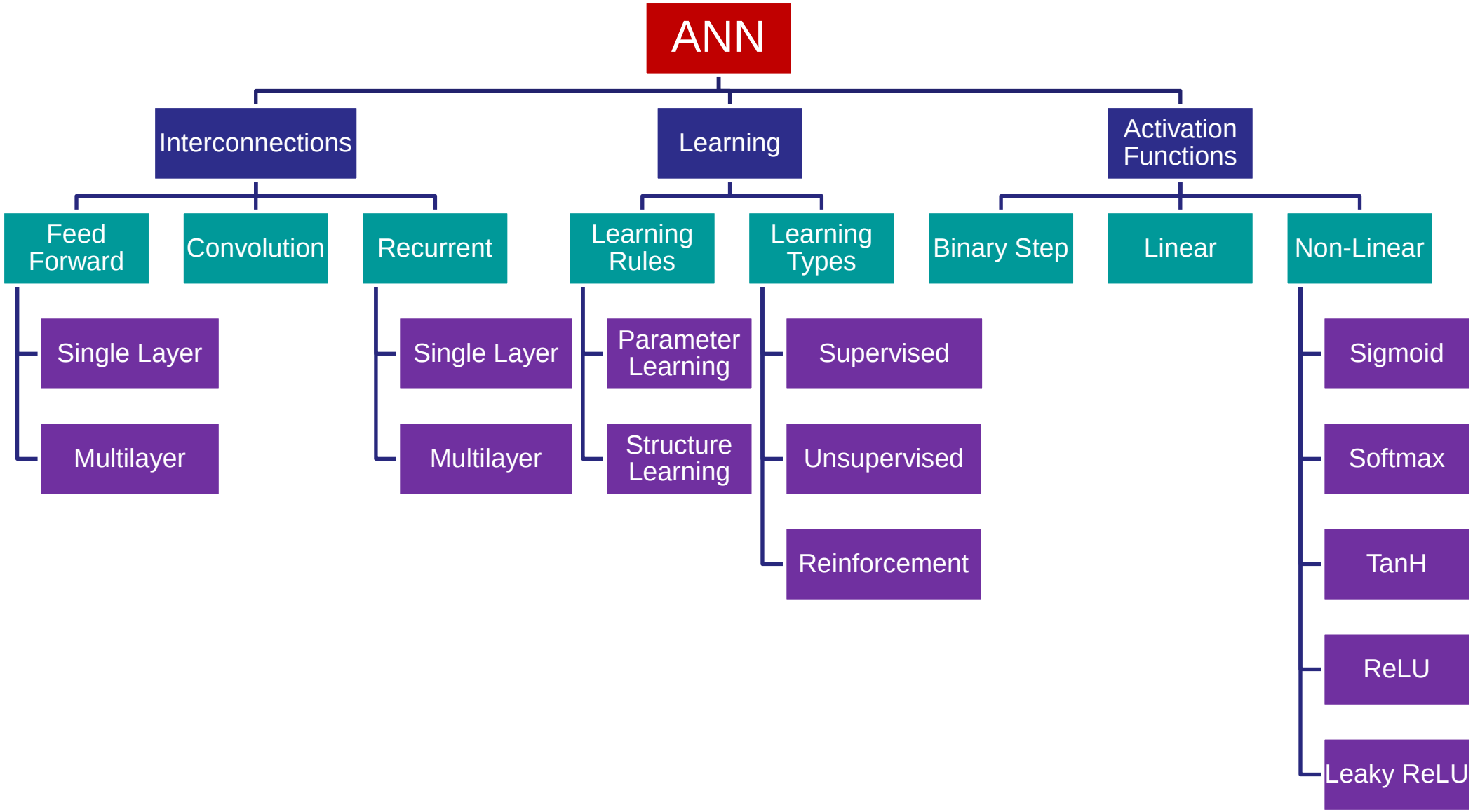




Exercise

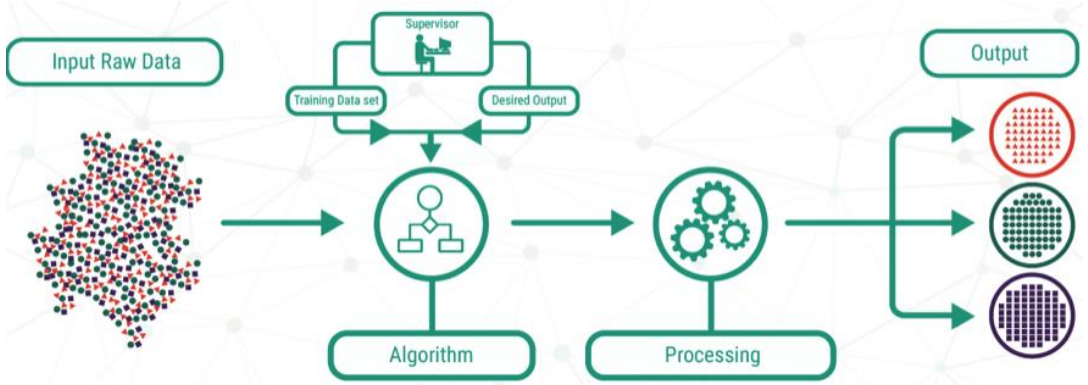
- Design neural network:
  - XNOR
  - NOR
  - NAND
  - XOR using OR, AND and NAND
  - XNOR using OR, AND and NOR

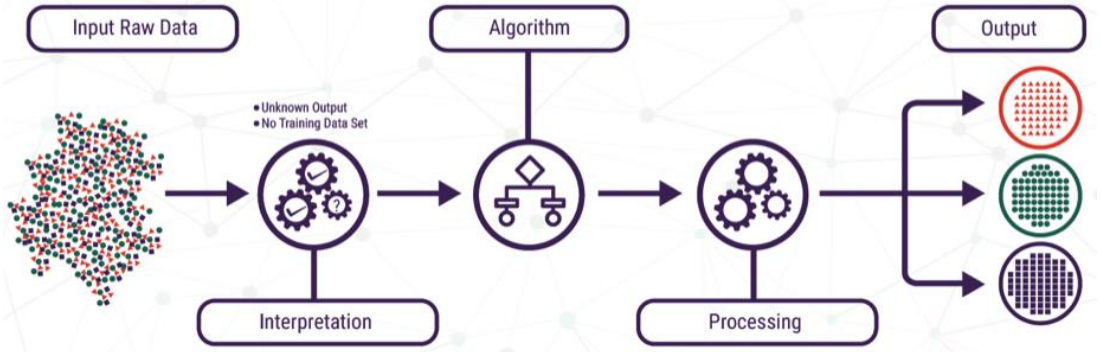
ANN Desing

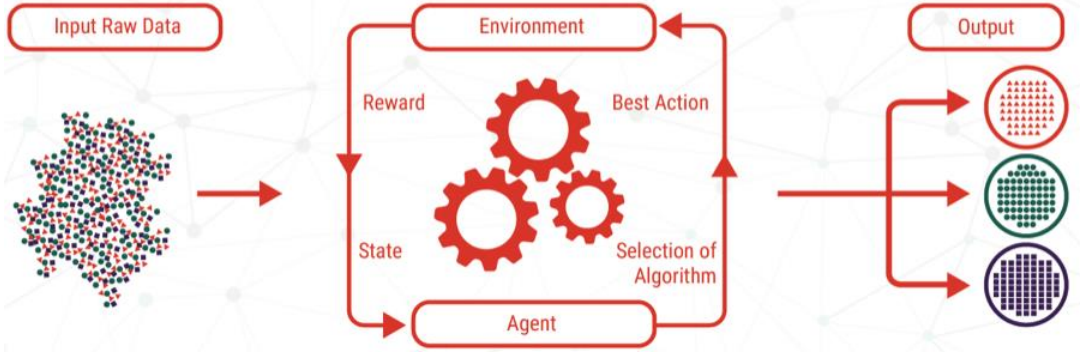

ANN
Interconnections
Learning
Activation Functions
Feed Forward
Convolution
Recurrent
Learning Rules
Learning Types
Binary Step
Linear
Non-Linear
Single Layer
Multilayer
Single Layer
Multilayer
Parameter Learning
Structure Learning
Supervised
Unsupervised
Reinforcement
Sigmoid
Softmax
TanH
ReLU
Leaky ReLU

Learning Types

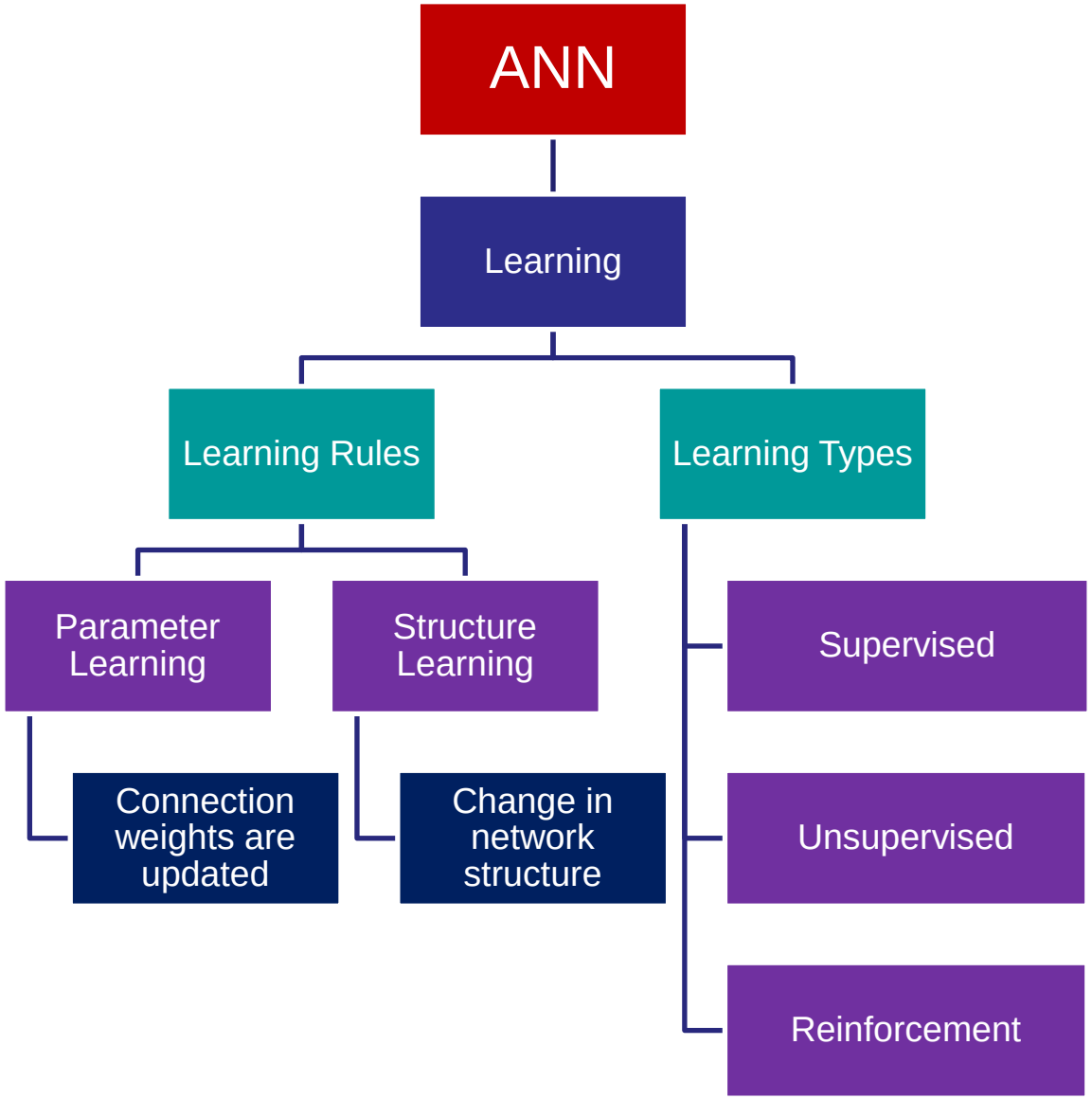
- Supervised learning: (Labeled examples)
  - Agent is given correct answers for each example
  - Agent is learning a function from examples of its inputs and outputs
- Unsupervised learning: (Unlabeled examples)
  - Agent must infer correct answers
  - Completely unsupervised learning is impractical, since agent has no context
- Reinforcement learning: (Rewards)
  - Agent is given occasional rewards for correct
  - Typically involves subproblem of learning “how the world works”

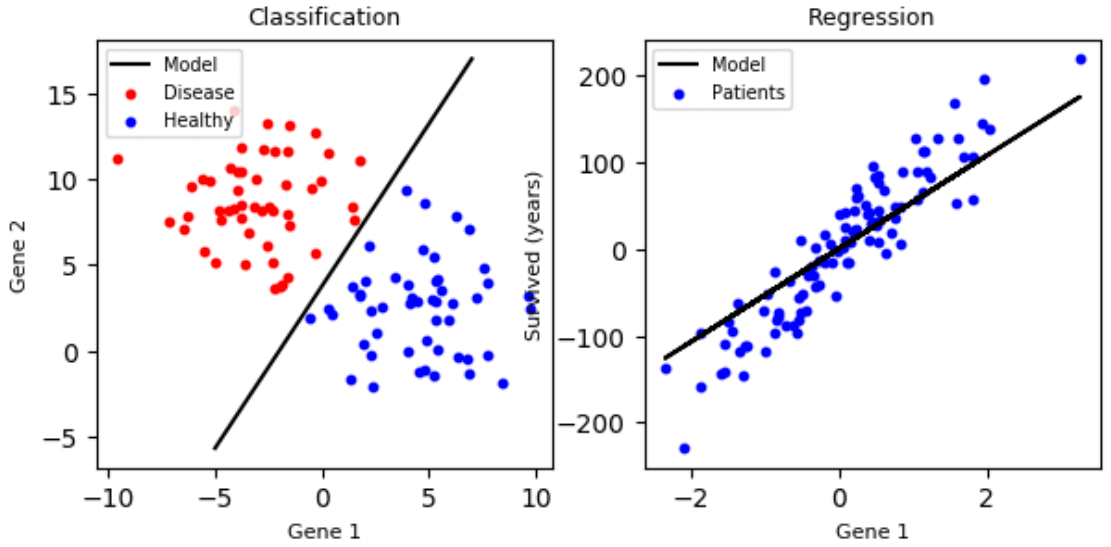


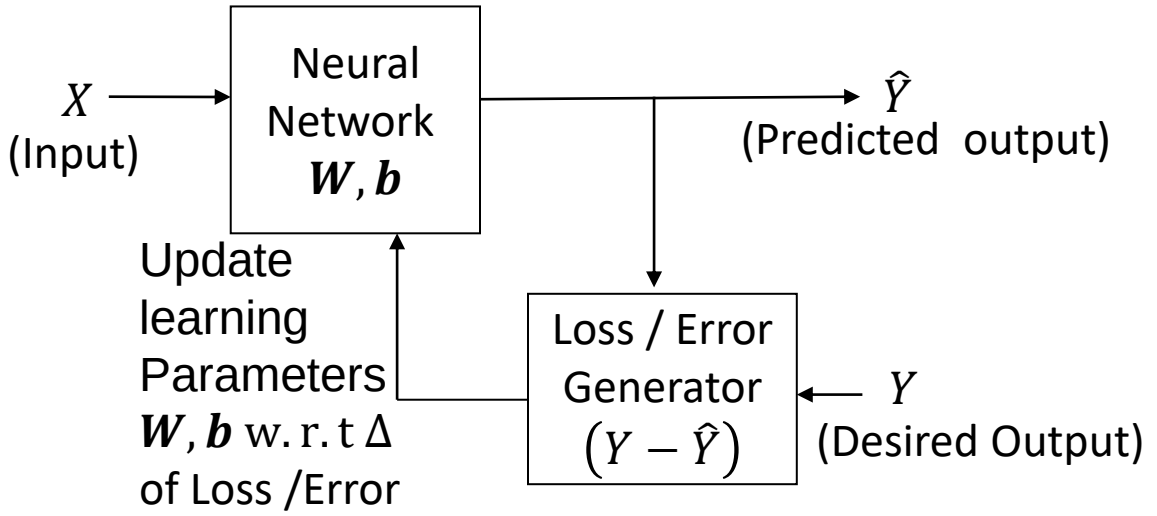




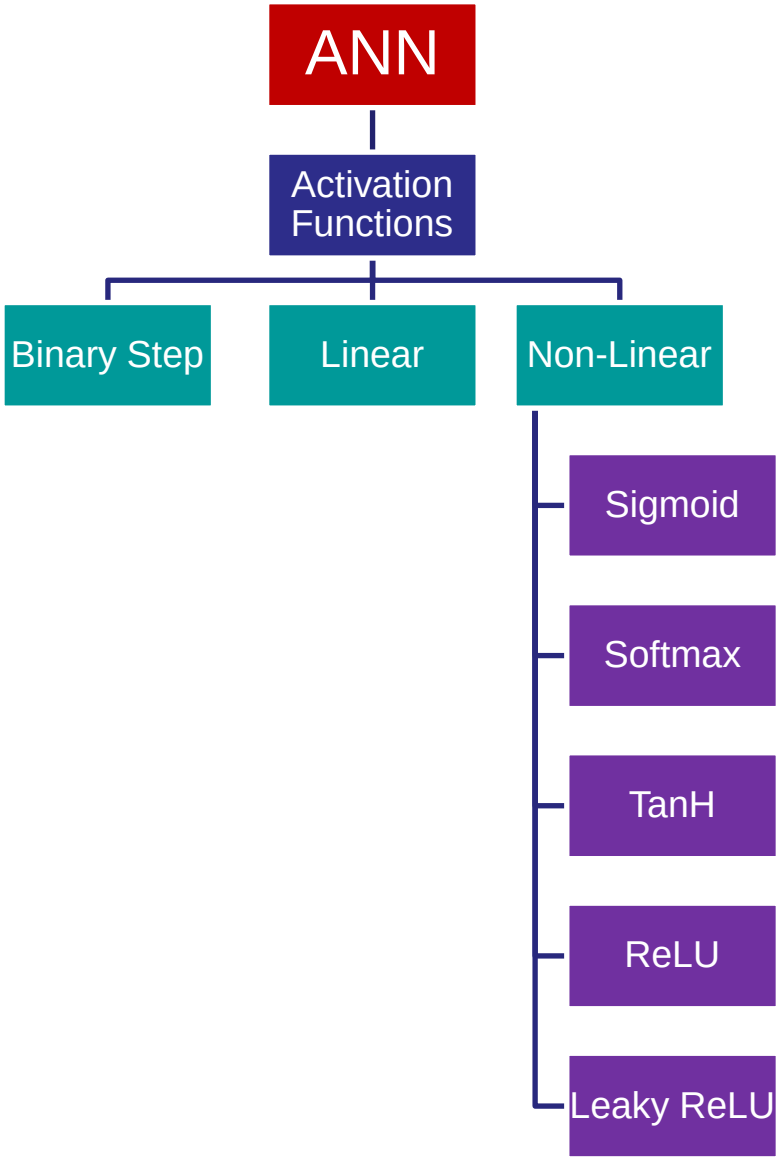
Learning in ANN

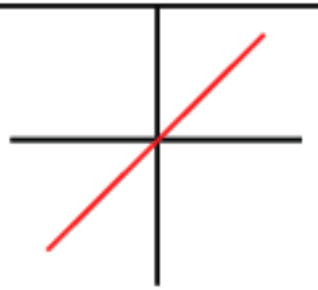
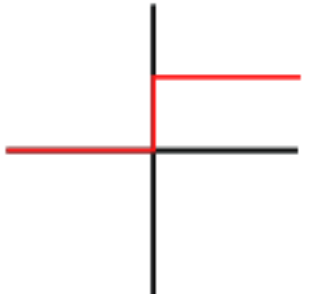




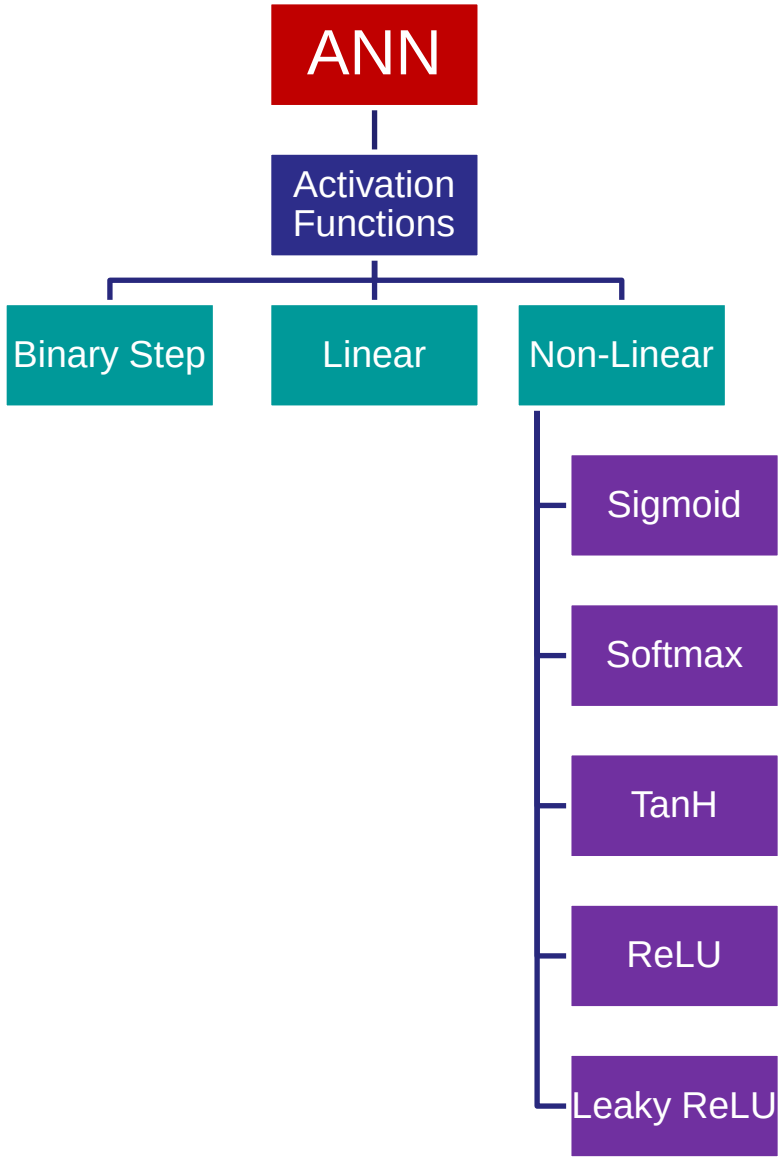


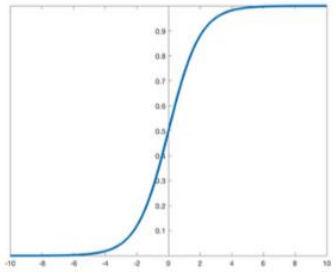
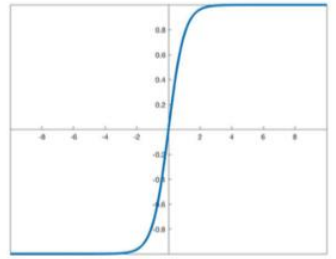
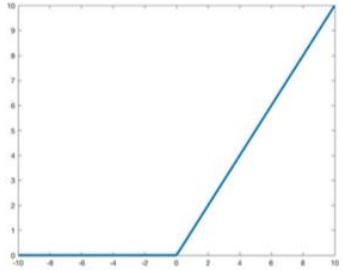
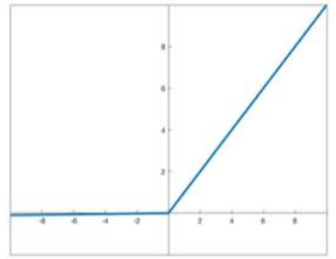
Activation Function



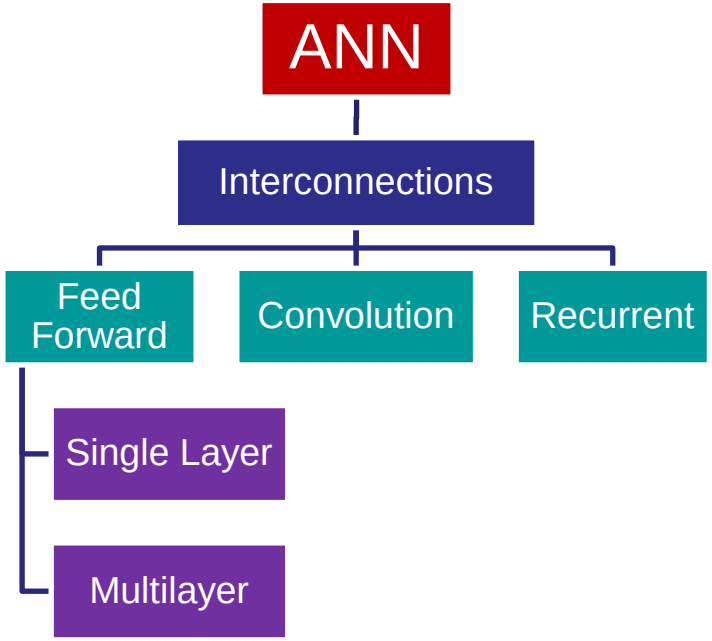
Activation function	Mathematical representation	Figure
Identity	$f(x) = x$	
Binary step	$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$	

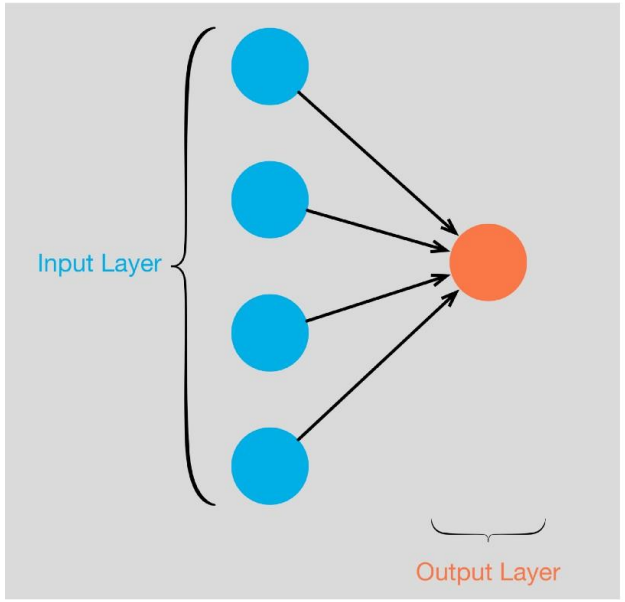
Activation Function

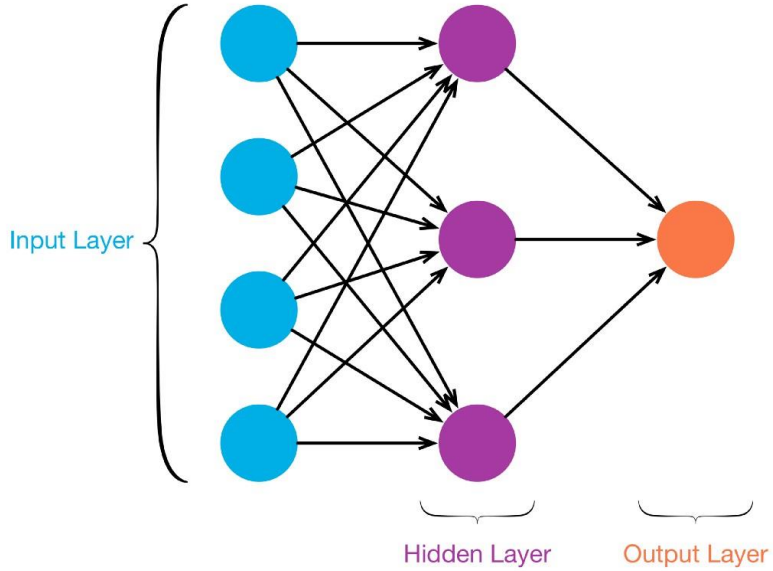


Activation function	Equation	Graph
Sigmoid	$S(x) = \frac{1}{1+e^{-x}}$	
Tanh	$\tanh x = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	
ReLU	$RELU(x) = \begin{cases} 0 \ if \ x<0 \\ x \ if \ x>=0 \end{cases}$	
Leaky ReLU	$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$	

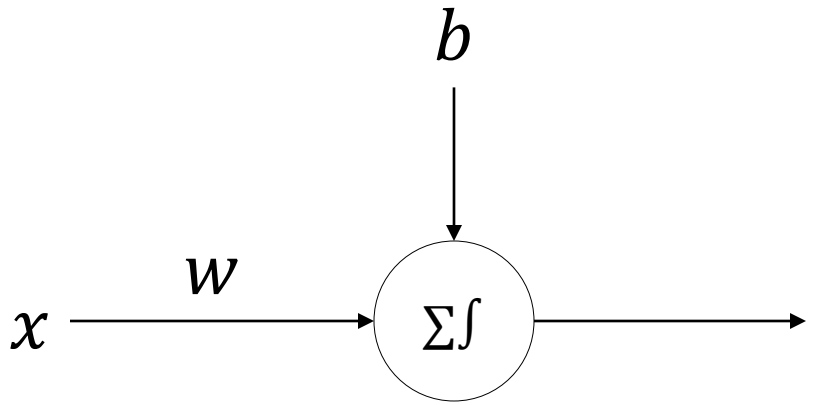
Characterization of ANN







Neuron Desing



$$y = \sigma(xw + b)$$

$x = 1.4,$
$w = 2.5,$
$b = 1$

$$z = 2.5 * 1.4 + 1$$
$$z = 4.5$$

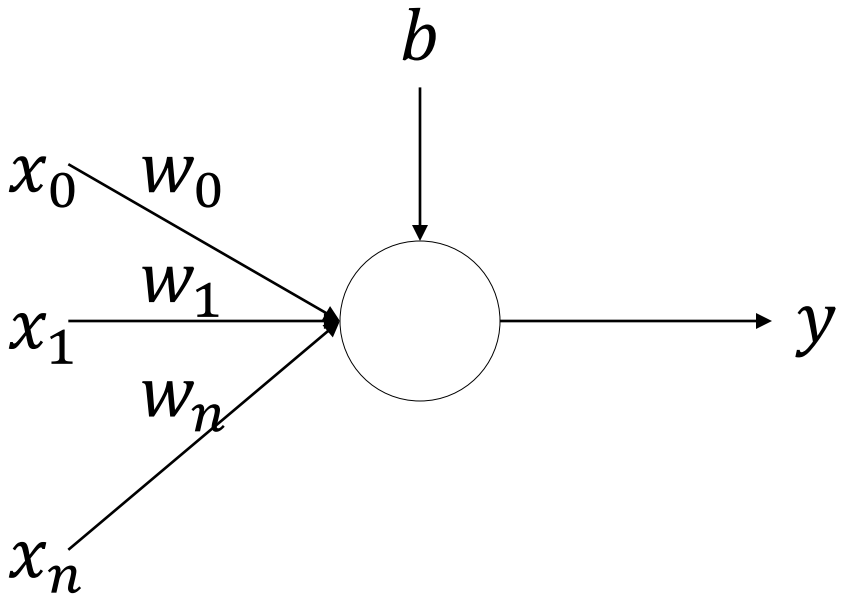
$$y = \frac{1}{1 + e^{-z}}$$

$$y = 0.989$$

$$y = \frac{1}{1 + e^{(2.5 * 1.4 + 1)}}$$

$$y = 0.989$$

Neuron Desing



$$y = \sigma\left(\sum_{i=0}^{n} x_i w_i + b\right)$$

$$Input = \boldsymbol{x}_{(1\times n)} = [x_0, x_1, \cdots, x_n]$$

$$Weights = \boldsymbol{w}_{(1\times n)} = [w_0, w_1, \cdots, w_n]$$

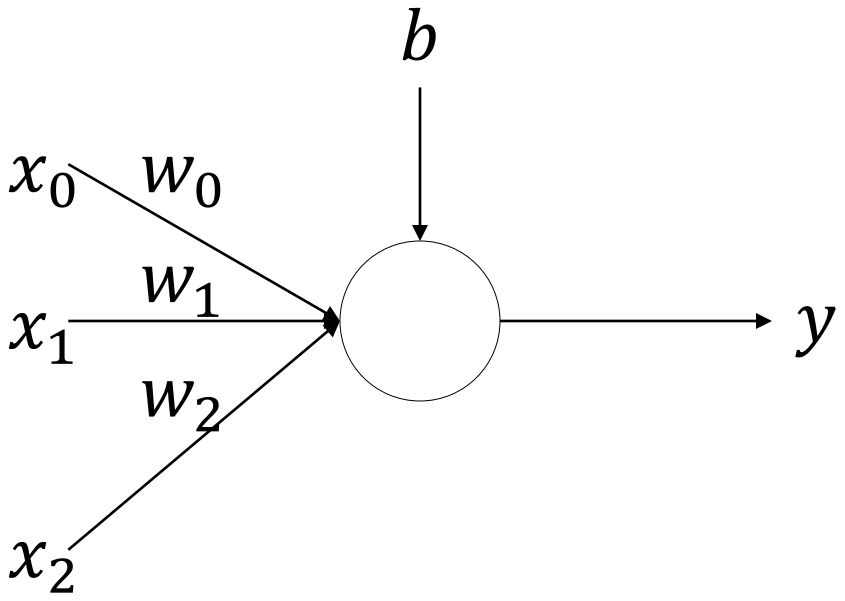
$$bias = \boldsymbol{b}_{(1\times 1)} = [b_0]$$

$$z = (x_0 w_0 + x_1 w_1 + \cdots + x_n w_n + b)$$

$$z = (\boldsymbol{x} \cdot \boldsymbol{w}^T) + b$$

$$y = \sigma(z)$$

Neuron Desing



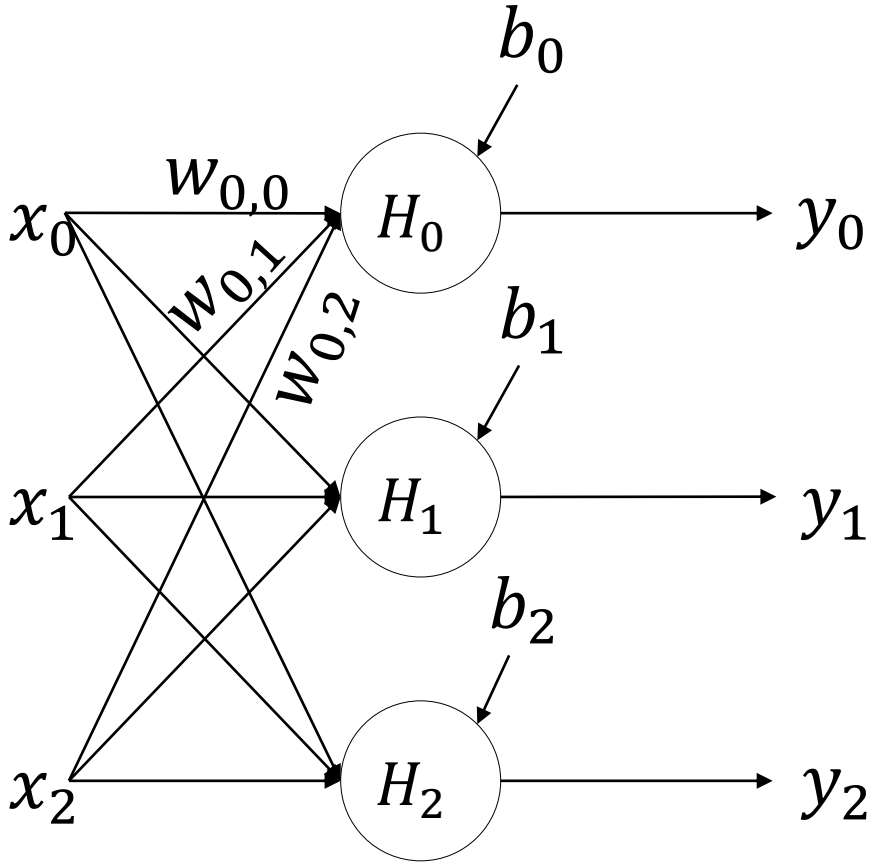
$$y = \sigma\left(\sum_{i=0}^{2} x_i w_i + b\right)$$

$\boldsymbol{x} = [1.4, 0.5, 2.2]$

$\boldsymbol{w} = [2.5, , 1.0, 1.1]$

$\boldsymbol{b} = [1]$

Single Layer Feedforward Neural Network Desing

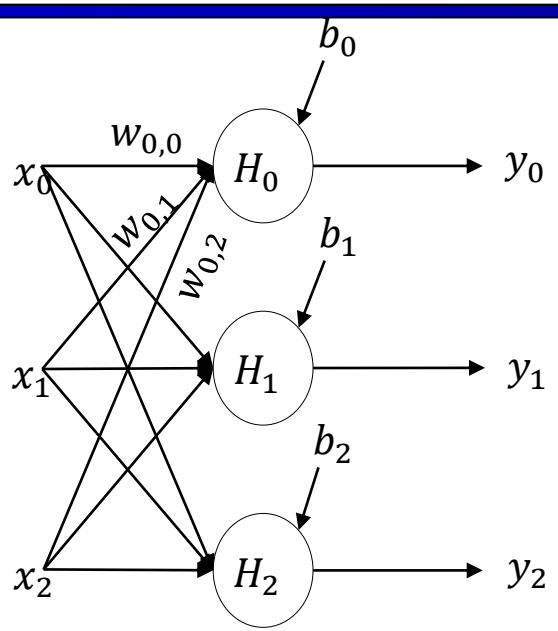


$$y_0 = \sigma\left(\sum_{i=0}^{2} x_i w_{0,i} + b_0\right)$$

$$y_1 = \sigma\left(\sum_{i=0}^{2} x_i w_{1,i} + b_1\right)$$

$$y_2 = \sigma\left(\sum_{i=0}^{2} x_i w_{2,i} + b_2\right)$$

Single Layer Feedforward Neural Network Desing



$$output = y_{(1\times3)}$$

$$Input = \boldsymbol{x}_{(1\times n)} = [x_0, x_1, x_2]$$

$$\boldsymbol{z} = \begin{bmatrix} x_0 \times w_{0,0} + x_1 \times w_{0,1} + x_2 \times w_{0,2} + b_0, \\ x_0 \times w_{1,0} + x_1 \times w_{1,1} + x_2 \times w_{1,2} + b_{1,} \\ x_0 \times w_{2,0} + x_1 \times w_{2,1} + x_2 \times w_{2,2} + b_2 \end{bmatrix}$$

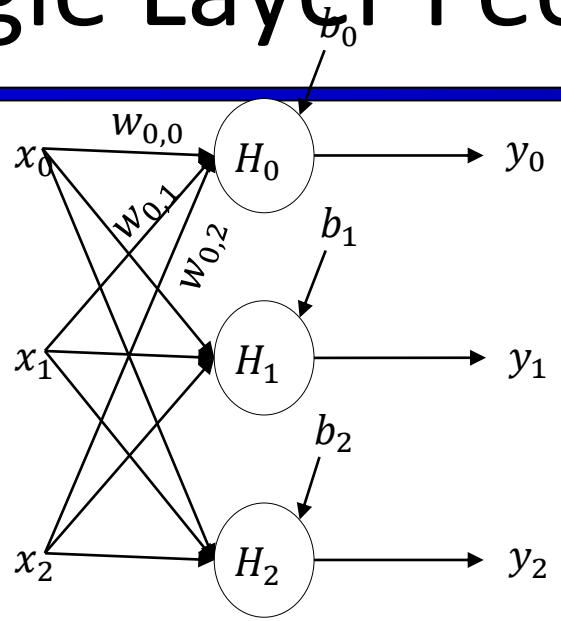
$$Weights = \boldsymbol{w}_{(3\times n)} = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{bmatrix}$$

$$\boldsymbol{z} = \boldsymbol{x} \cdot \boldsymbol{w}^T + \boldsymbol{b}$$

$$bias = \boldsymbol{b}_{(1\times3)} = [b_0, b_1, b_2]$$

$$y = \sigma(\boldsymbol{z})$$

Single Layer Feedforward Neuron Network Desing



$$output = Y_{(3\times3)}$$

$$Input = \boldsymbol{X}_{(3\times n)} = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} \\ x_{1,0} & x_{1,1} & x_{1,2} \\ x_{2,0} & x_{2,1} & x_{2,2} \end{bmatrix}$$

$$Weights = \boldsymbol{W}_{(3\times n)} = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{bmatrix}$$

$$bias = \boldsymbol{b}_{(1\times3)} = [b_0, b_1, b_2]$$

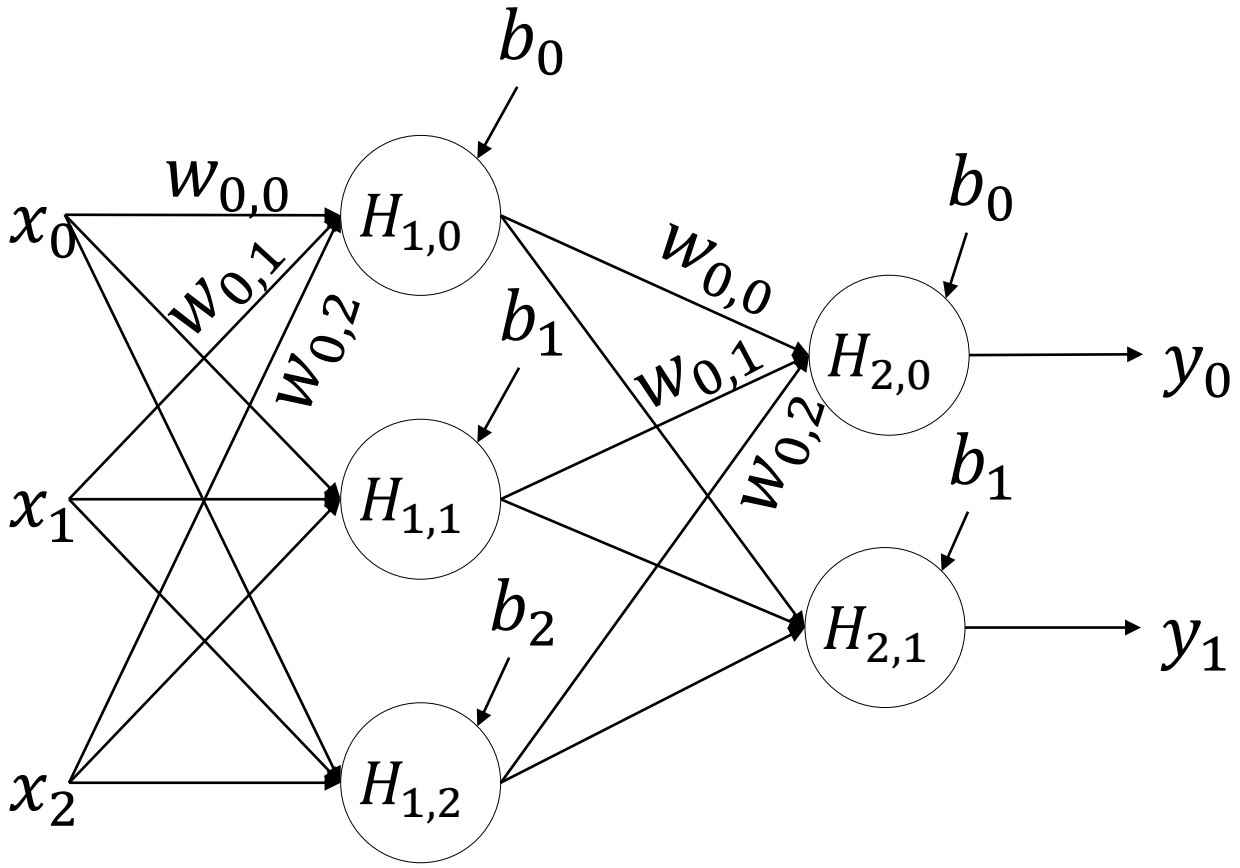
$$\boldsymbol{Z} = \boldsymbol{X} \cdot \boldsymbol{W}^T + \boldsymbol{b}$$

$$\boldsymbol{z}_0 = \begin{bmatrix} x_{0,0} \times w_{0,0} + x_{0,1} \times w_{0,1} + x_{0,2} \times w_{0,2} + b_0, \\ x_{0,0} \times w_{1,0} + x_{0,1} \times w_{1,1} + x_{0,2} \times w_{1,2} + b_1, \\ x_{0,0} \times w_{2,0} + x_{0,1} \times w_{2,1} + x_{0,2} \times w_{2,2} + b_2 \end{bmatrix}_{1\times3}$$

$$\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{z}_0 \\ \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{bmatrix}_{3\times3}$$

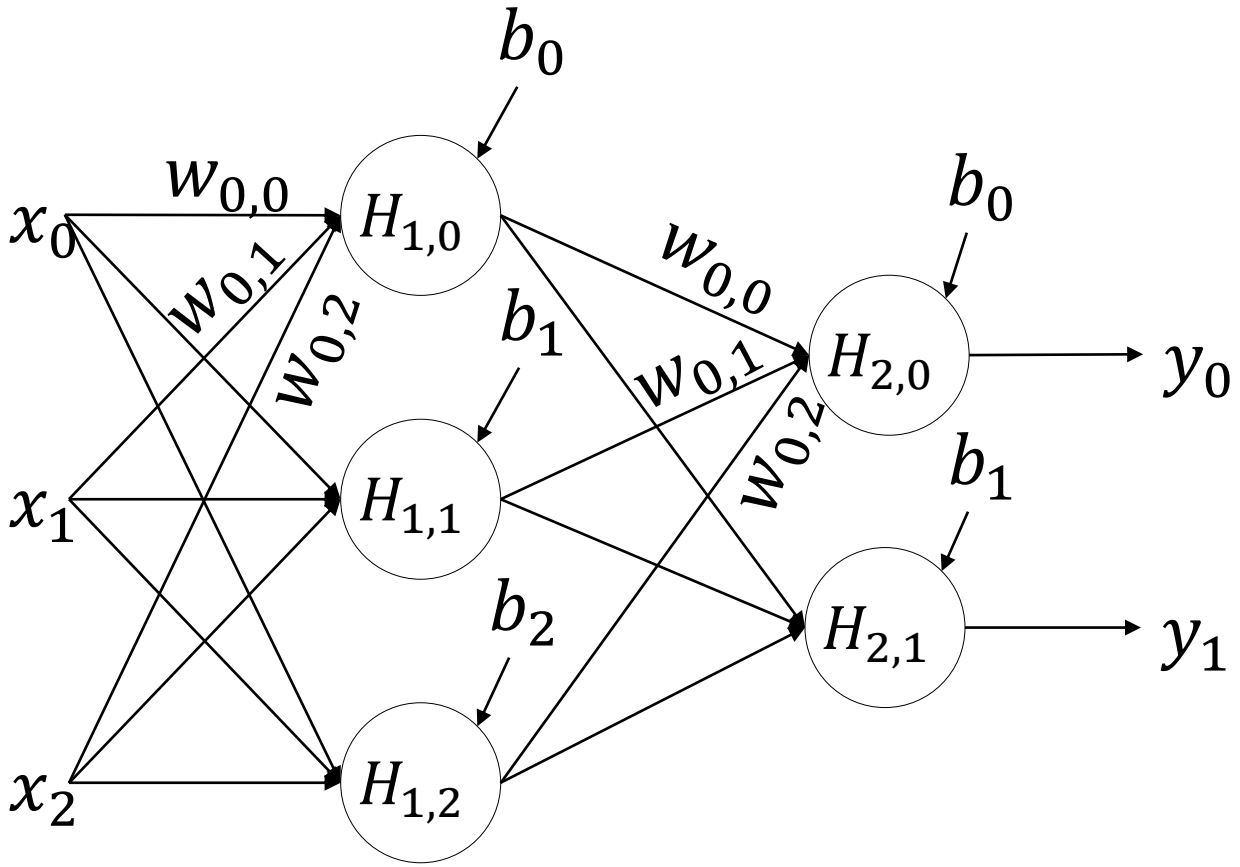
$$y = \sigma(\boldsymbol{Z})$$

Multi-Layer Feedforward Neural Network Desing



$[\boldsymbol{X}]_{(10\times3)}$ $[\boldsymbol{W}_1]_{(3\times3)}$ $[\boldsymbol{b}_1]_{(1\times3)}$ $[\boldsymbol{W}_2]_{(2\times3)}$ $[\boldsymbol{b}_2]_{(1\times2)}$ $\boldsymbol{Y}_{(10\times2)}$

Multi-Layer Neural Network Feedforward Desing



$[\boldsymbol{X}]_{(10\times3)}$

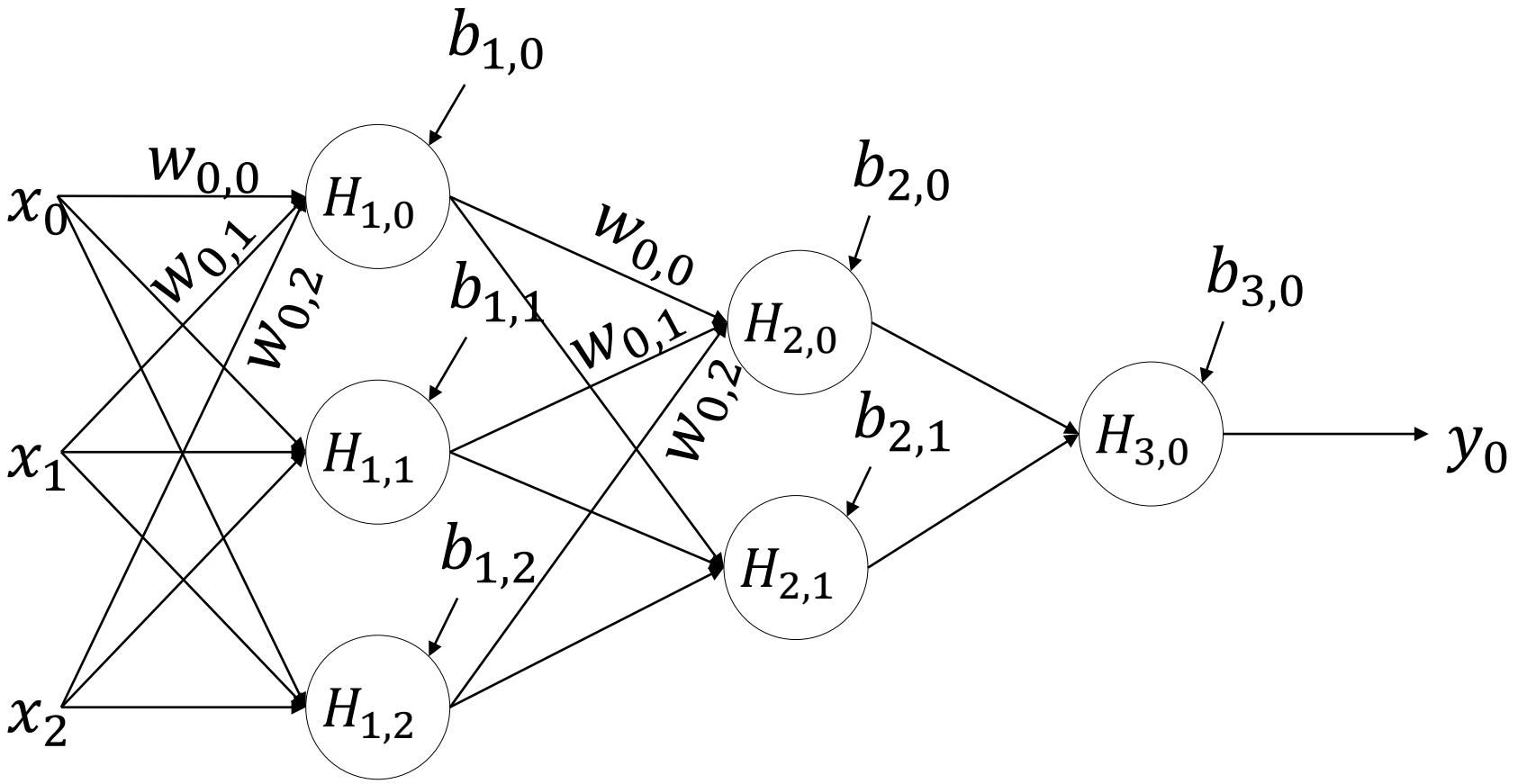
$[\boldsymbol{W}_1]_{(3\times3)}$
$[\boldsymbol{b}_1]_{(1\times3)}$

$[\boldsymbol{W}_2]_{(2\times3)}$
$[\boldsymbol{b}_2]_{(1\times2)}$

$\boldsymbol{Y}_{(10\times2)}$

$$\hat{Y} = \sigma(\sigma(X \cdot W_1 + b_1) \cdot W_2 + b_2)$$

Multi-Layer Feedforward Neural Network Desing

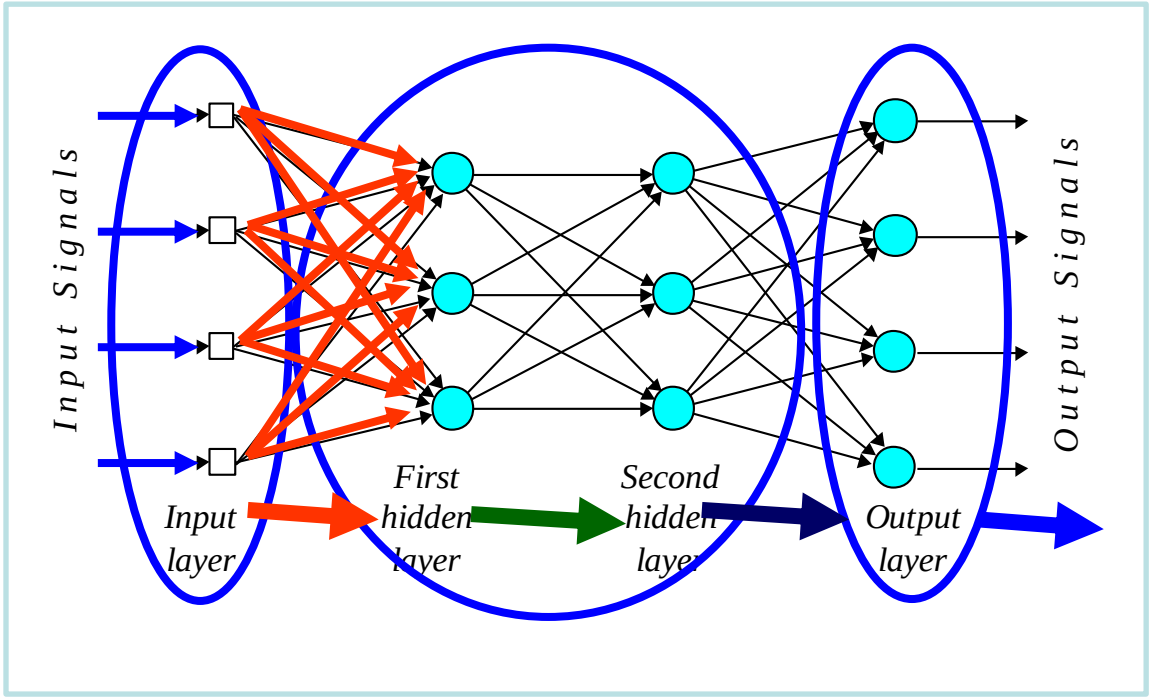


$[\boldsymbol{X}]_{(10\times3)}$ $[\boldsymbol{W}_1]_{(3\times3)}$ $[\boldsymbol{b}_1]_{(1\times3)}$ $[\boldsymbol{W}_2]_{(2\times3)}$ $[\boldsymbol{b}_2]_{(1\times2)}$ $[\boldsymbol{W}_3]_{(1\times2)}$ $[\boldsymbol{b}_3]_{(1\times1)}$ $\boldsymbol{Y}_{(10\times1)}$

$$\hat{Y} = \sigma(\sigma(\sigma(X \cdot W_1 + b_1) \cdot W_2 + b_2) \cdot W_3 + b_3)$$

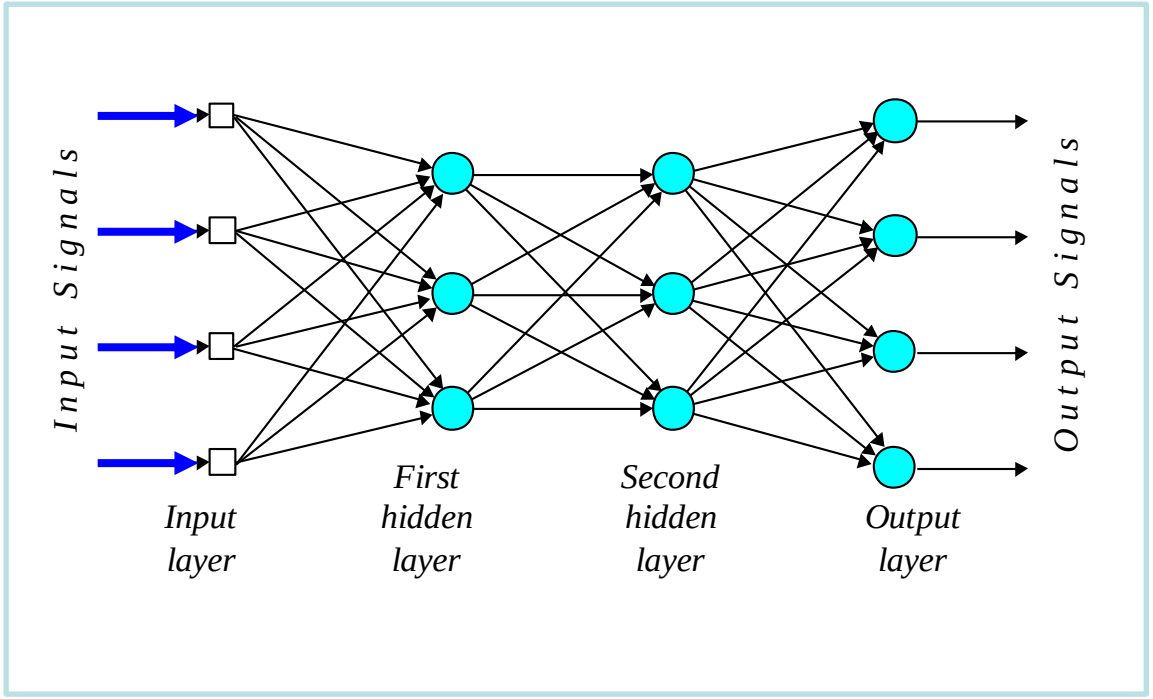
Multilayer Neural Networks

- A multi-layer feedforward neural network with ≥ 1 hidden layers.



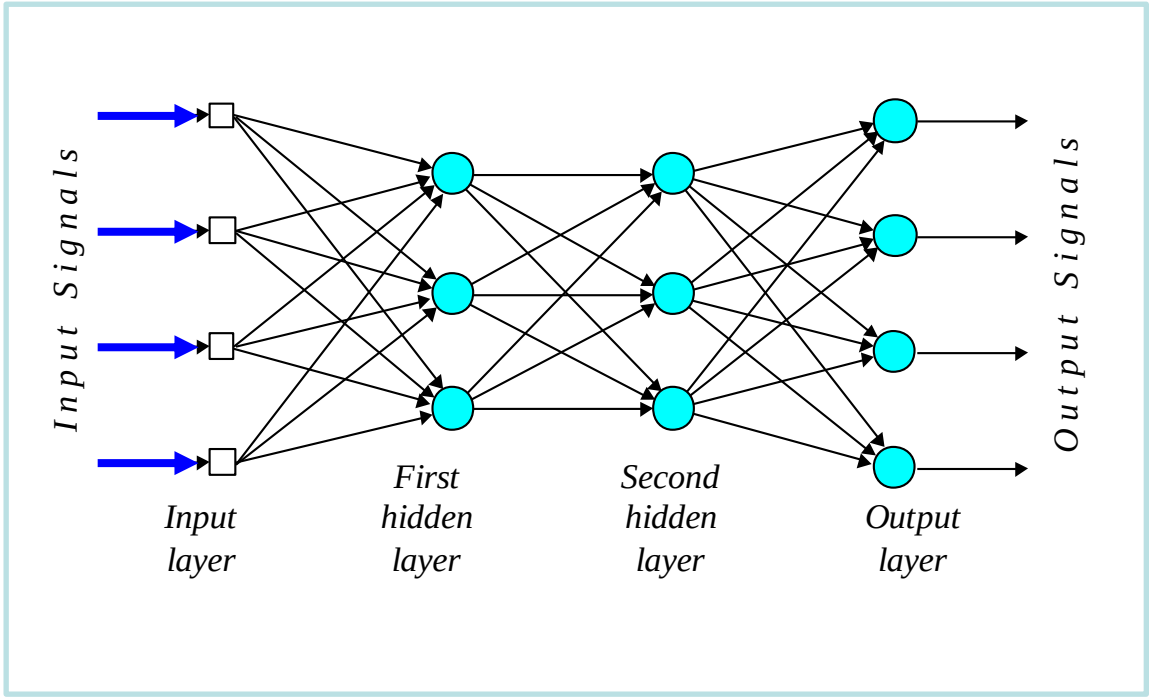
Roles of Layers

- Input Layer
  - Accepts input signals from outside world
  - Distributes the signals to neurons in hidden layer
  - Usually does not do any computation
- Output Layer (computational neurons)
  - Accepts output signals from the previous hidden layer
  - Outputs to the world
  - Knows the desired outputs
- Hidden Layer (computational neurons)
  - Determines its own desired outputs

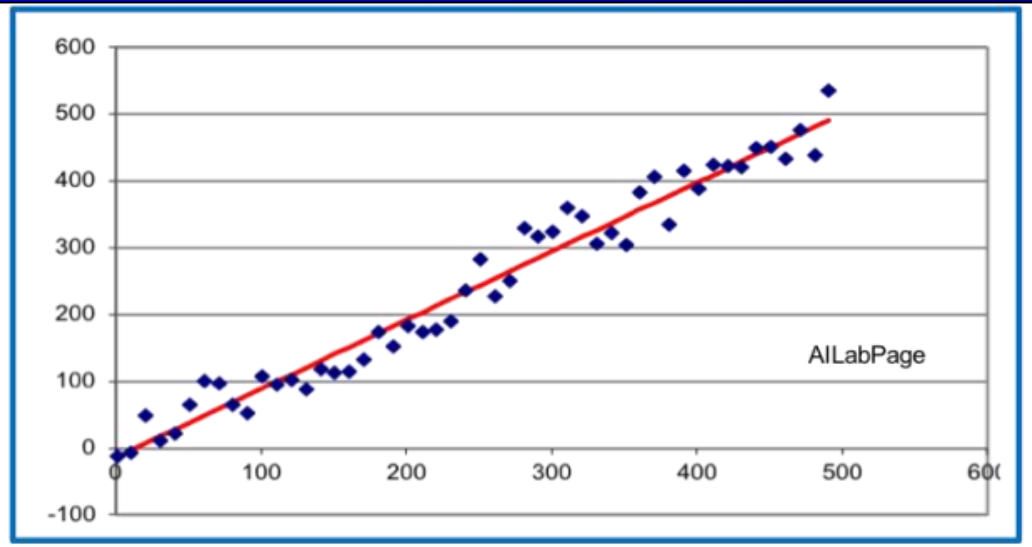


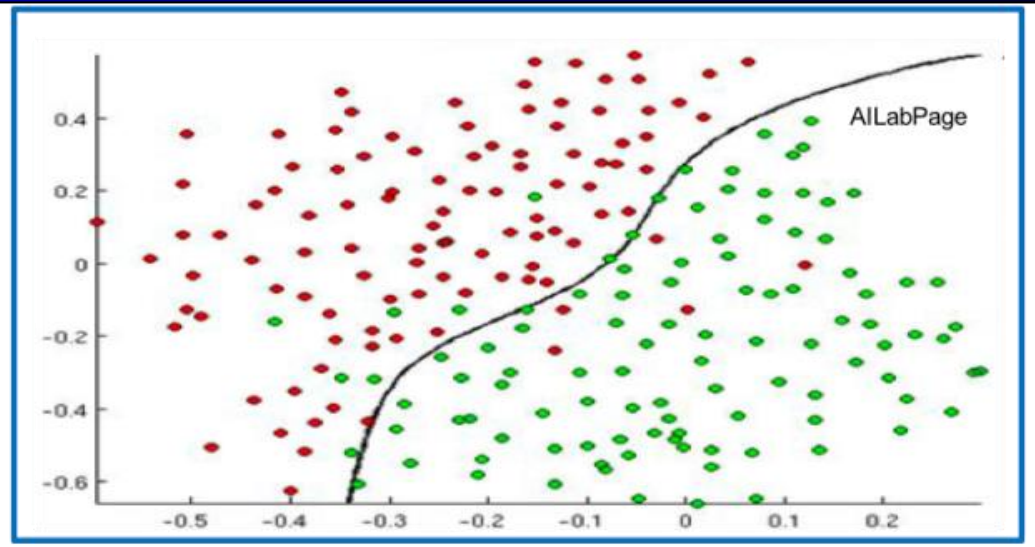
Roles of Layers

- Neurons in hidden layers unobservable through input and output of the networks.
- Desired output unknown (hidden) from the outside and determined by the layer itself
- 1 hidden layer for continuous functions
- 2 hidden layers for discontinuous functions
- Practical applications mostly use 3 layers
- More layers are possible, but each additional layer exponentially increases computing load



Supervise Learning





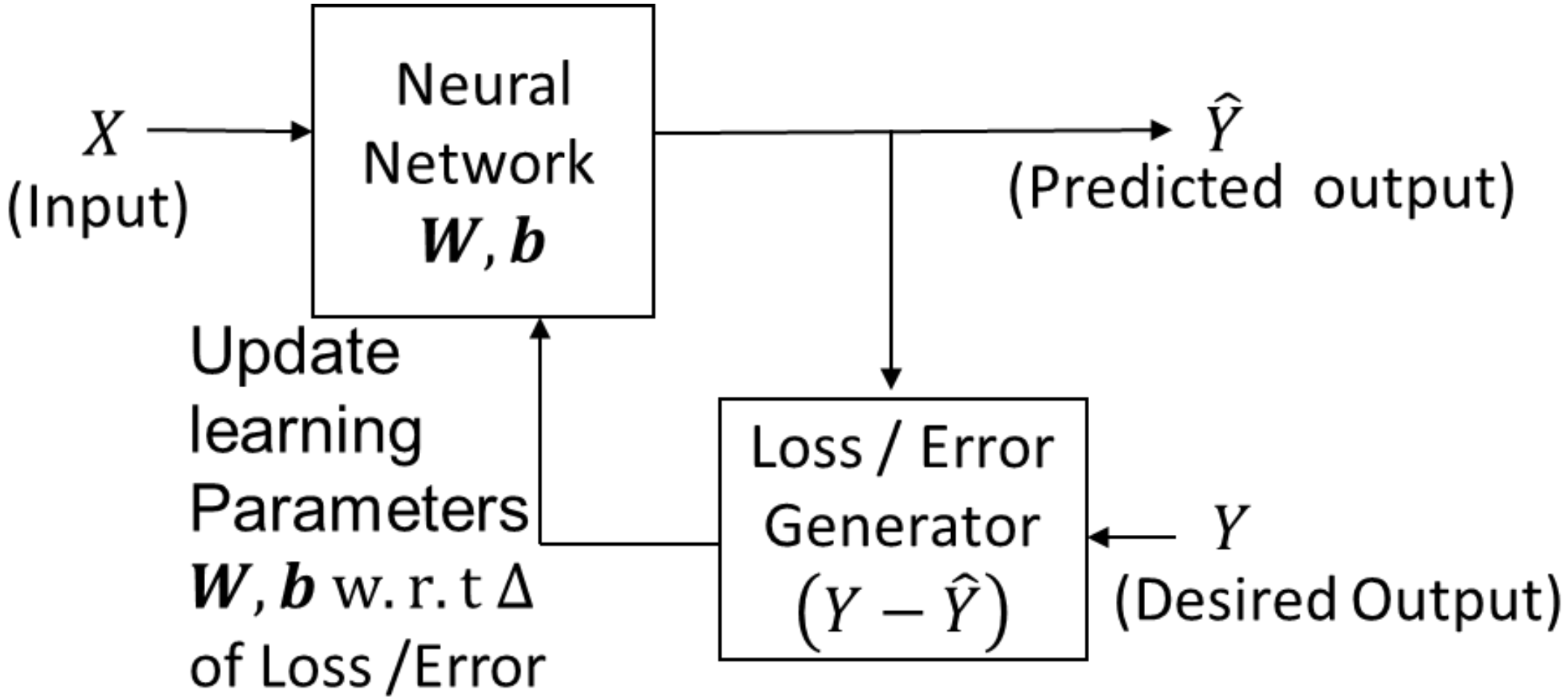
Regression

1. The system attempts to predict a value for an input based on past data.
2. Real number / Continuous numbers – Regression problem
3. Example – 1. Temperature for tomorrow

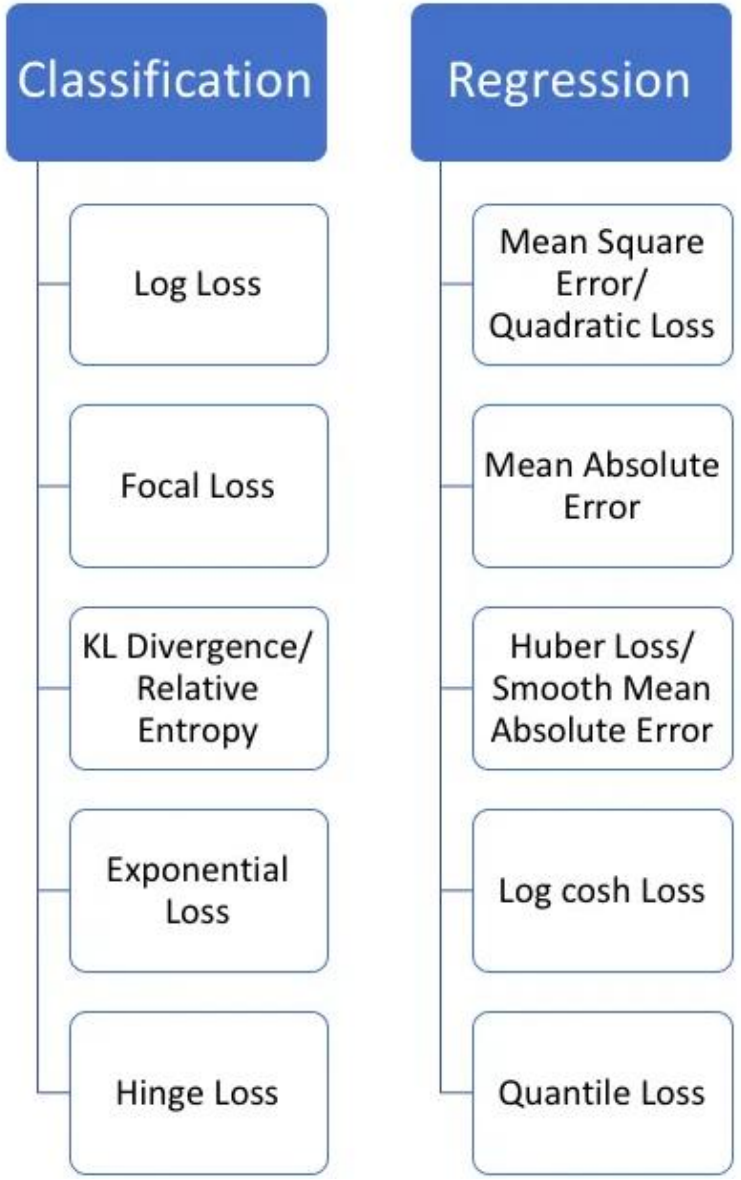
Classification

1. In classification, predictions are made by classifying them into different categories.
2. Discreate / categorical variable – Classification problem
3. Example – 1. Type of cancer 2. Cancer Y/N

Supervised Learning

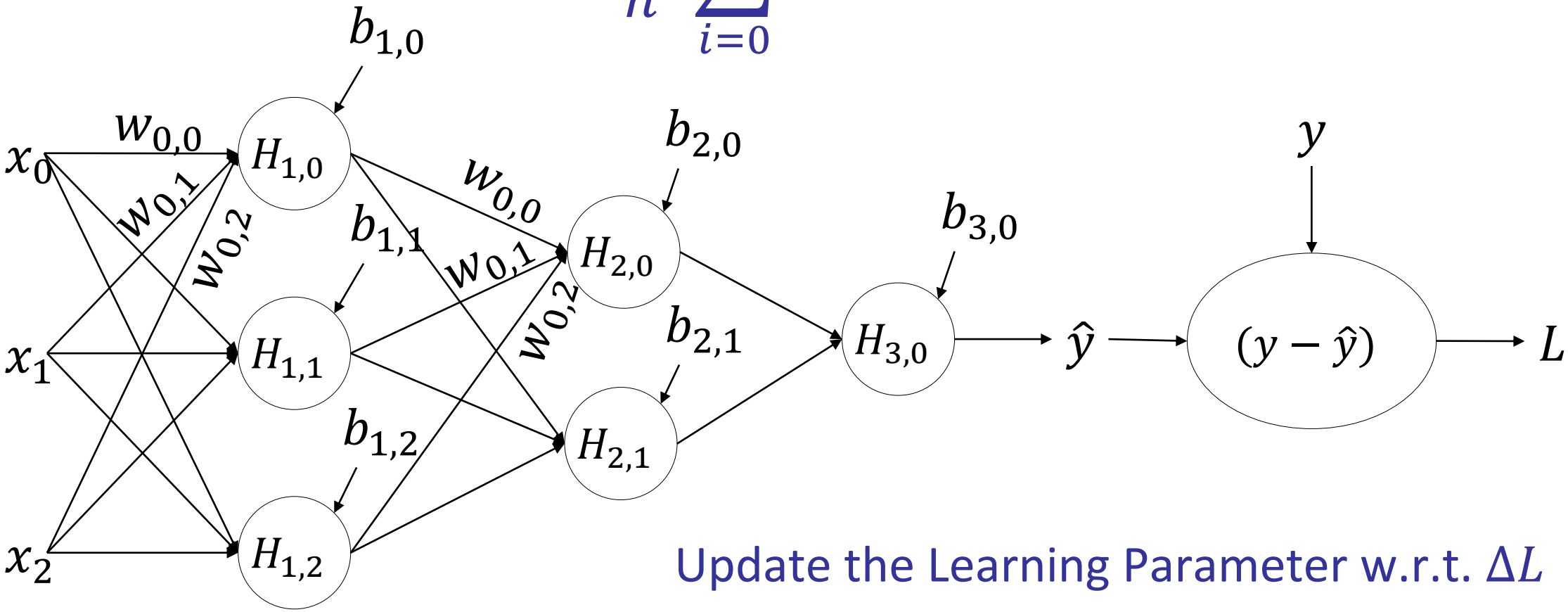


Error / Loss Function



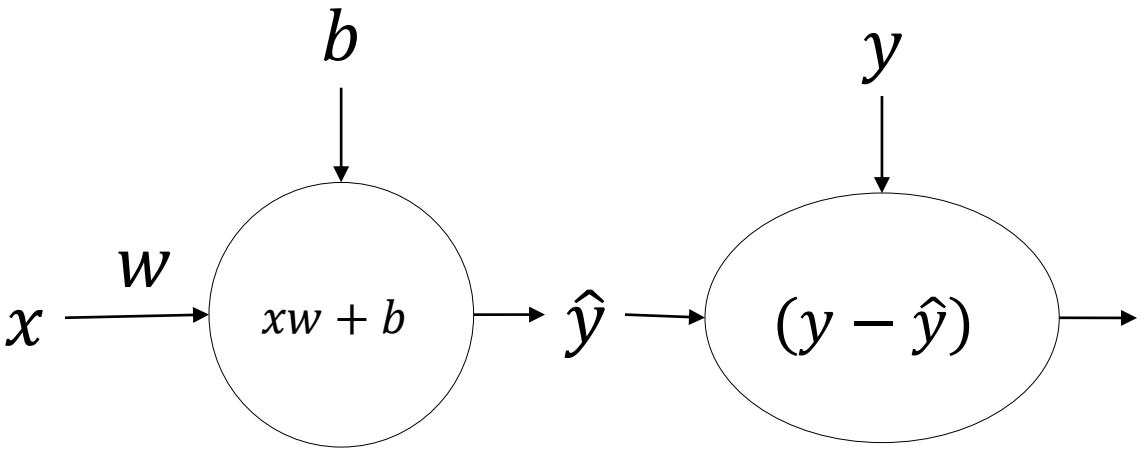
Mean Square Error / Loss (MSE)

$$L_i = \frac{1}{n}\sum_{i=0}^{n}(\boldsymbol{y}_i - \hat{y}_i)^2$$

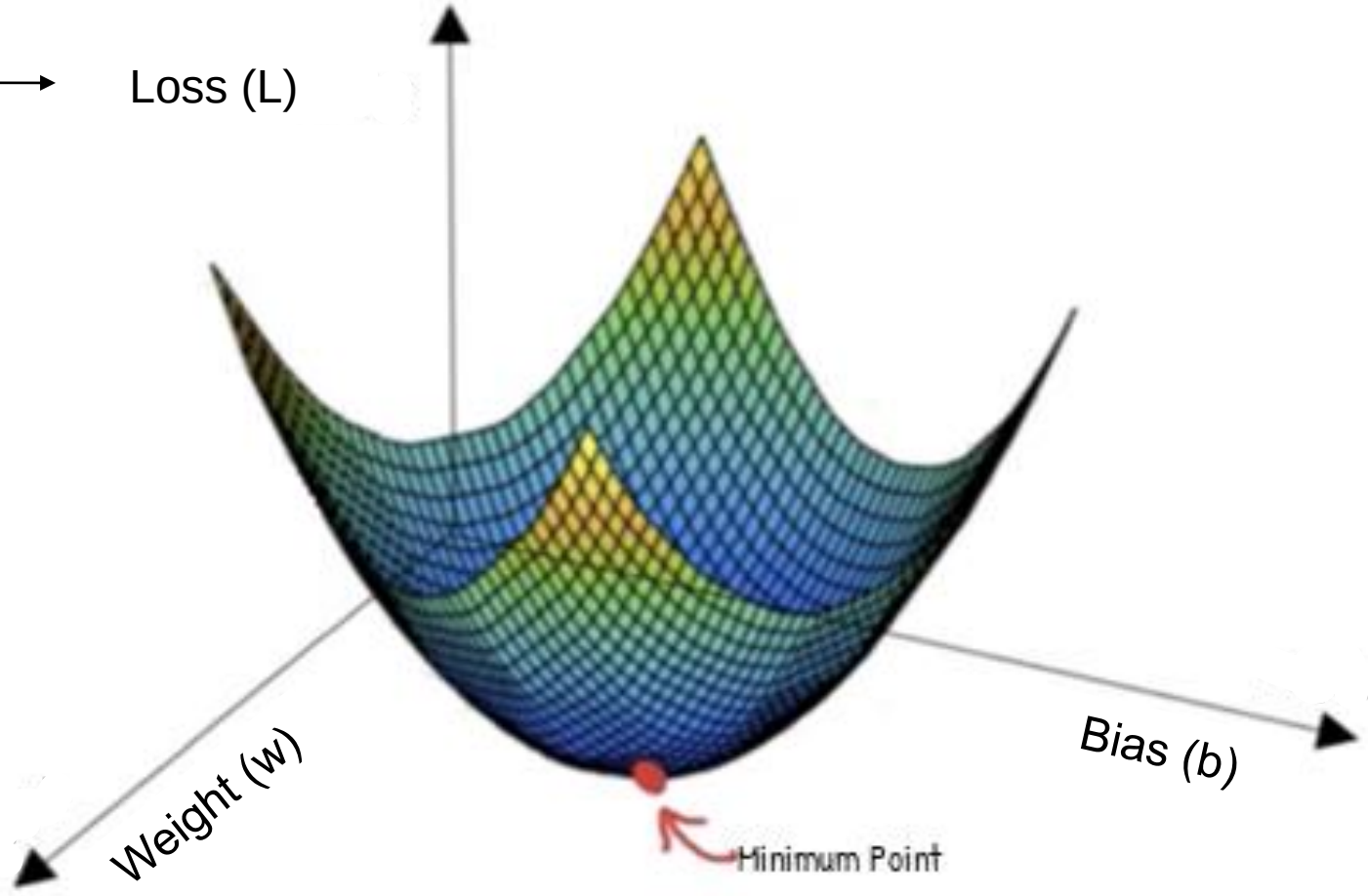


Update the Learning Parameter w.r.t. ΔL

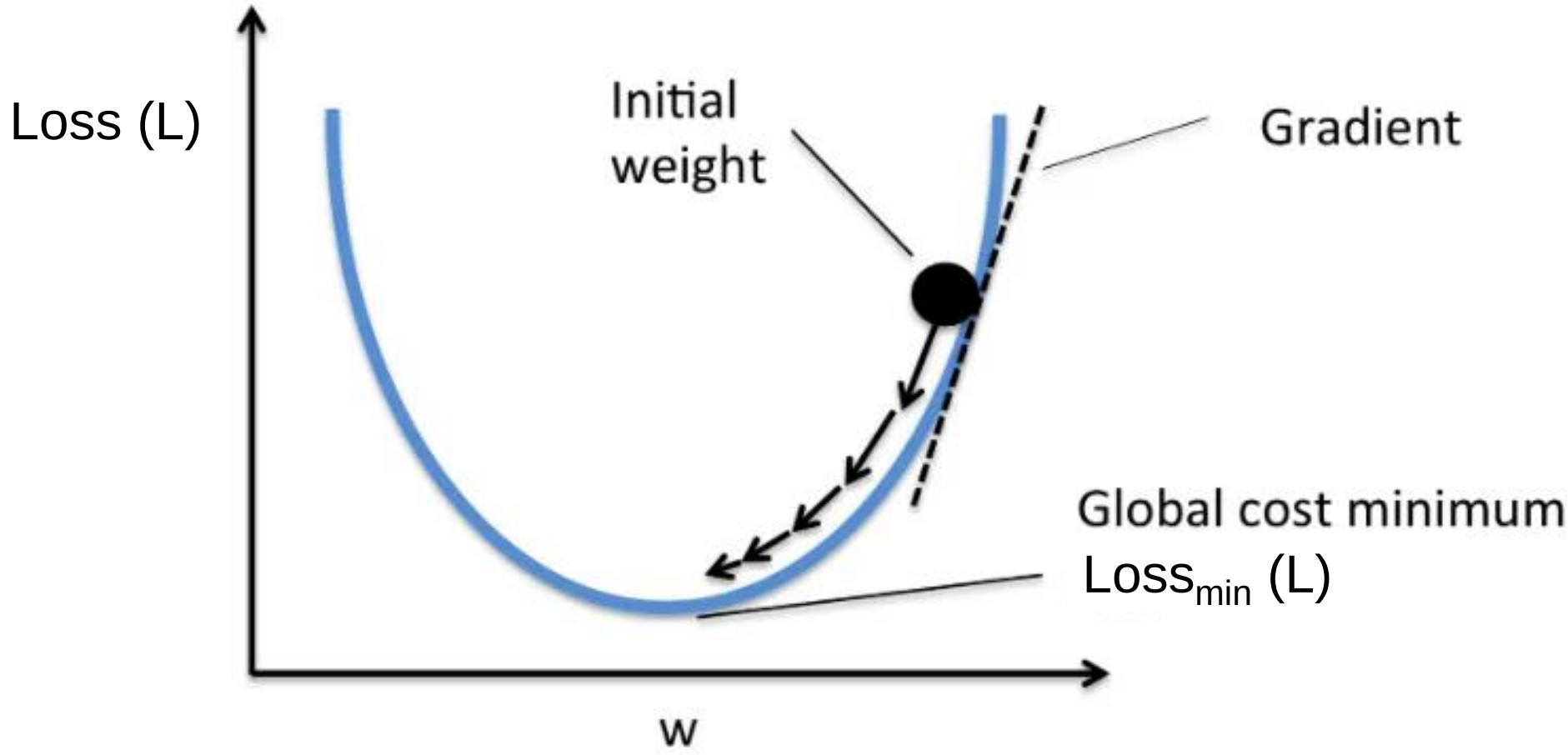
Single Neuron



- Find the derivative of L w.r.t. w i.e. $\frac{\partial L}{\partial w}$
- If w changes by a small amount, how much will L change?
- Update the w to reduce the Loss L
- $w = w - \alpha \cdot \Delta w$

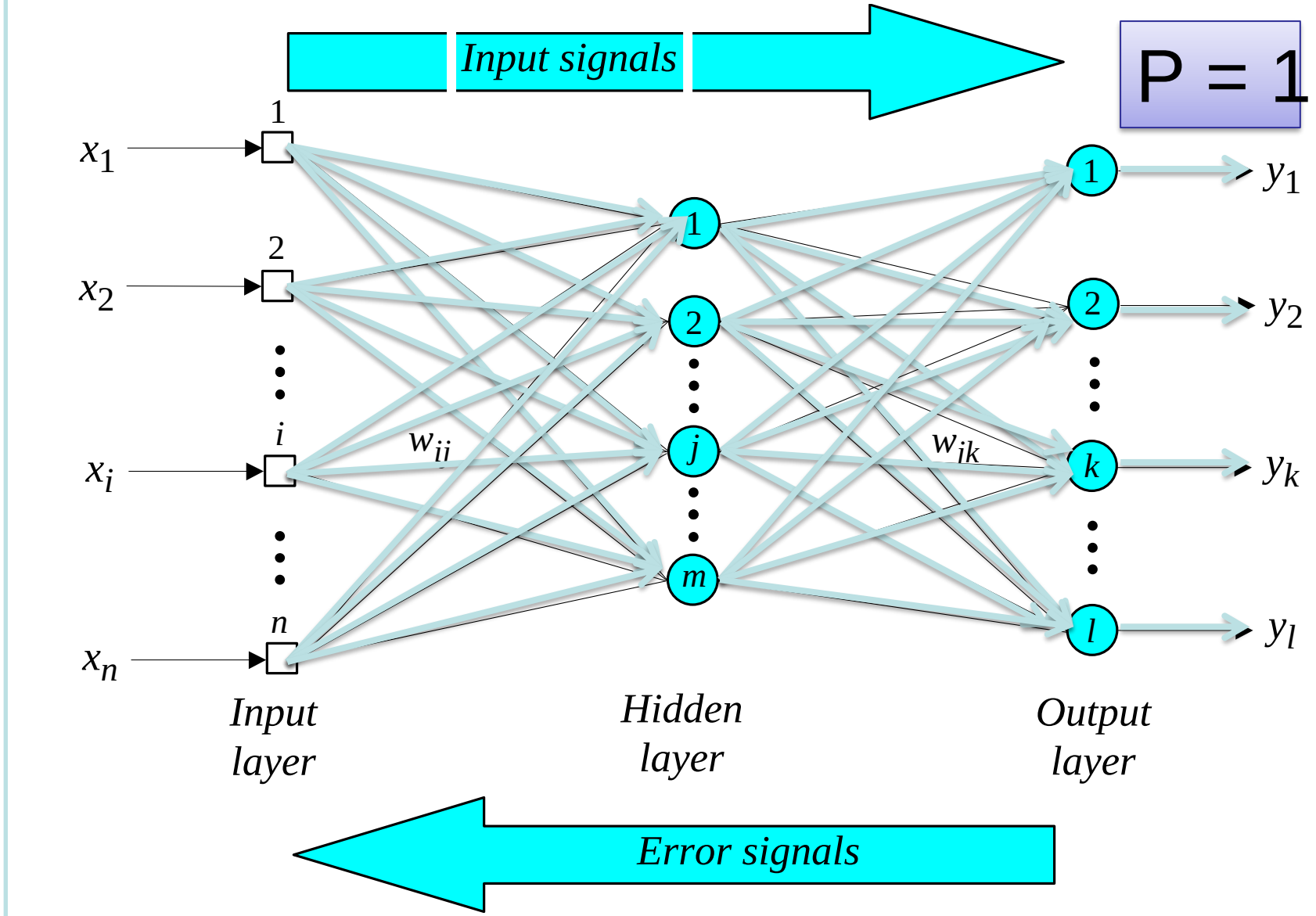


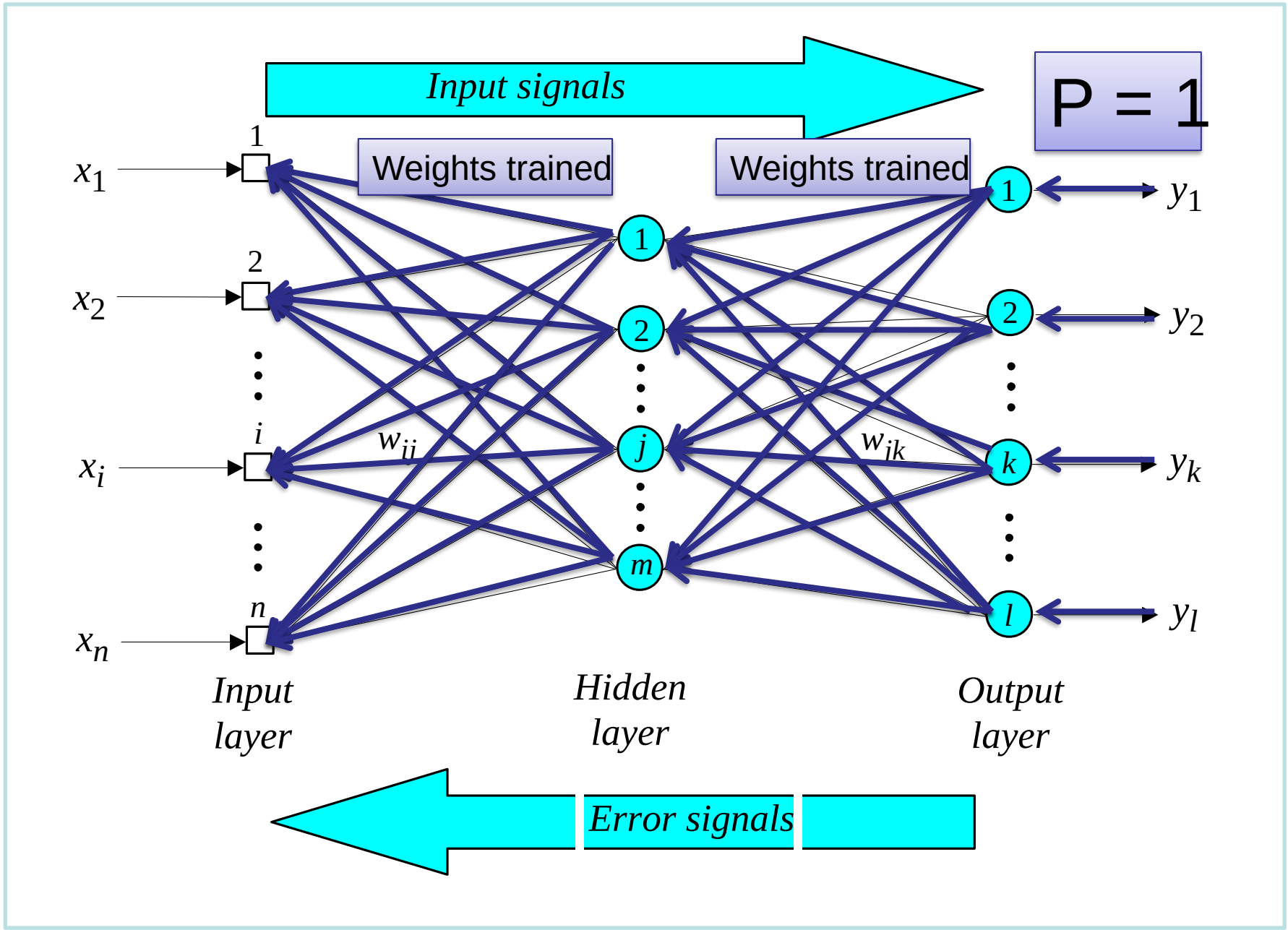
Optimization



Backpropagation

- "backward propagation of errors"
- Method used in ANN to calculate a gradient of a loss L w.r.t. weights w that is needed in the calculation of updated weights.
- Computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule.
- Used to train ANN by minimizing the difference between predicted $\hat{Y}$ and actual outputs Y.


Input signals
P = 1
1
2
i
n
x_1
x_2
x_i
x_n
w_{ii}
w_{ik}
1
2
j
m
1
2
k
l
y_1
y_2
y_k
y_l
Input layer
Hidden layer
Output layer
Error signals


Input signals
P = 1
Weights trained
Weights trained
1
2
i
n
x_1
x_2
x_i
x_n
1
2
j
m
1
2
k
l
y_1
y_2
y_k
y_l
w_{ij}
w_{jk}
Input layer
Hidden layer
Output layer
Error signals

Derivatives Rule

$f(x) = k \quad \Rightarrow \quad f'(x) = \frac{dk}{dx} = 0$, where k is any constant

$f(x) = x \quad \Rightarrow \quad f'(x) = \frac{dx}{dx} = 1$

$f(x) = x^n \quad \Rightarrow \quad f'(x) = \frac{d(x^n)}{dx} = nx^{n-1}$

$f(x) = kx \quad \Rightarrow \quad f'(x) = \frac{d(kx)}{dx} = k$, where k is any constant

$f(x) = \sqrt{x} \quad \Rightarrow \quad f'(x) = \frac{d(\sqrt{x})}{dx} = \frac{1}{2}\sqrt{x}$

$f(x) = \frac{1}{x} \quad \Rightarrow \quad f'(x) = \frac{d(\frac{1}{x})}{dx} = \frac{-1}{x^2}$

$f(x) = \log x \quad \Rightarrow \quad f'(x) = \frac{d(\log x)}{dx} = \frac{1}{x}, x > 0$

$f(x) = e^x \quad \Rightarrow \quad f'(x) = \frac{d(e^x)}{dx} = e^x$

$f(x) = a^x \quad \Rightarrow \quad f'(x) = \frac{d(a^x)}{dx} = a^x \log a$

Derivative Rule

$$f(x,y) = x + y \quad \rightarrow \quad \frac{\partial}{\partial x} f(x,y) = \frac{\partial}{\partial x}[x+y] = \frac{\partial}{\partial x}x + \frac{\partial}{\partial x}y = 1 + 0 = 1$$

$$\frac{\partial}{\partial y} f(x,y) = \frac{\partial}{\partial y}[x+y] = \frac{\partial}{\partial y}x + \frac{\partial}{\partial y}y = 0 + 1 = 1$$

$$f(x,y) = x \cdot y \quad \rightarrow \quad \frac{\partial}{\partial x} f(x,y) = \frac{\partial}{\partial x}[x \cdot y] = y\frac{\partial}{\partial x}x = y \cdot 1 = y$$

$$\frac{\partial}{\partial y} f(x,y) = \frac{\partial}{\partial y}[x \cdot y] = x\frac{\partial}{\partial y}y = x \cdot 1 = x$$

$$f(x,y) = max(x,y) \quad \rightarrow \quad \frac{\partial}{\partial x} f(x,y) = \frac{\partial}{\partial x} max(x,y) = 1(x > y)$$

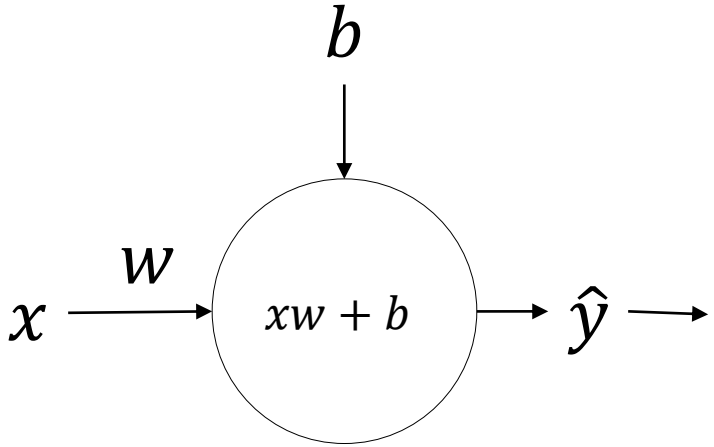
Chain Rule

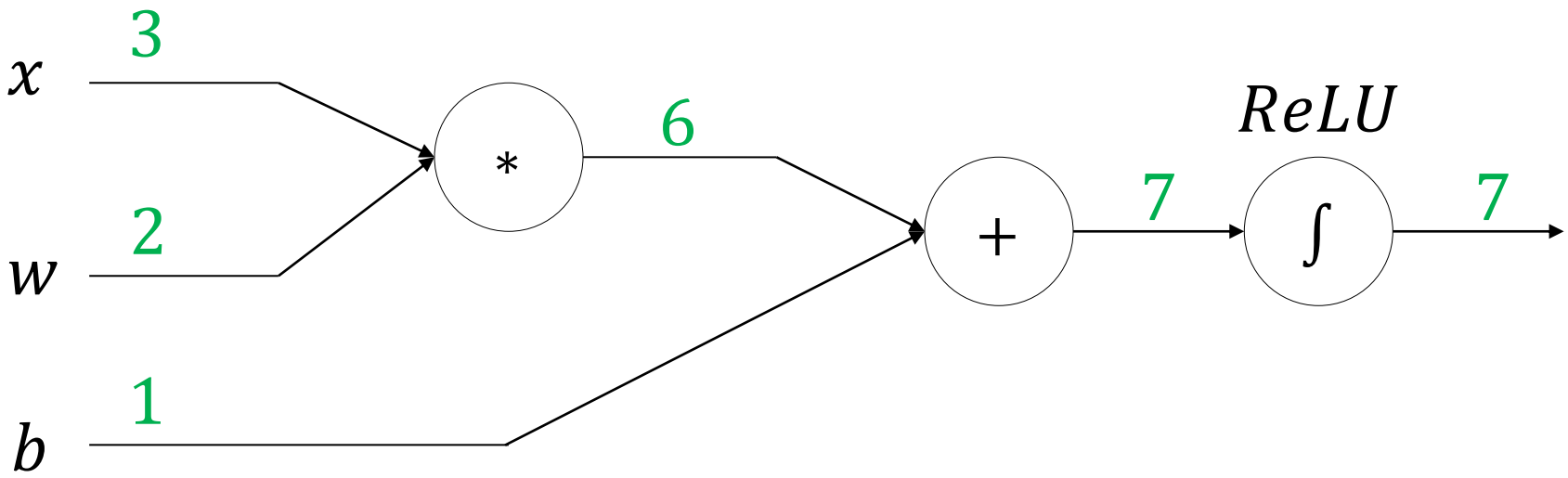
$$y = f(g(x))$$

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

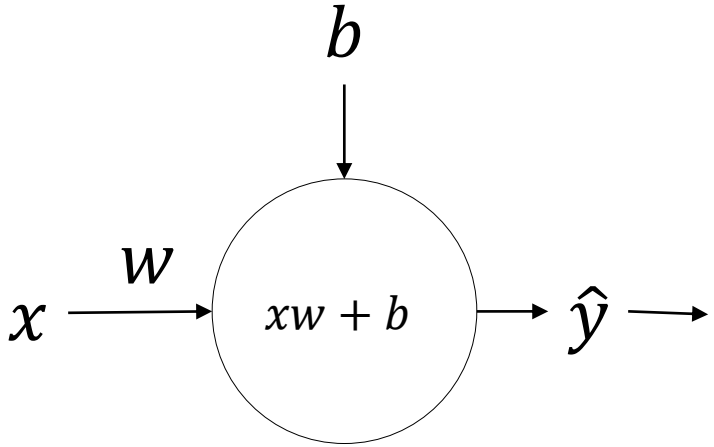
$$\frac{\partial}{\partial x} f(g(y, h(x, z))) = \frac{\partial f(g(y, h(x, z)))}{\partial g(y, h(x, z))} \cdot \frac{\partial g(y, h(x, z))}{\partial h(x, z)} \cdot \frac{\partial h(x, z)}{\partial x}$$

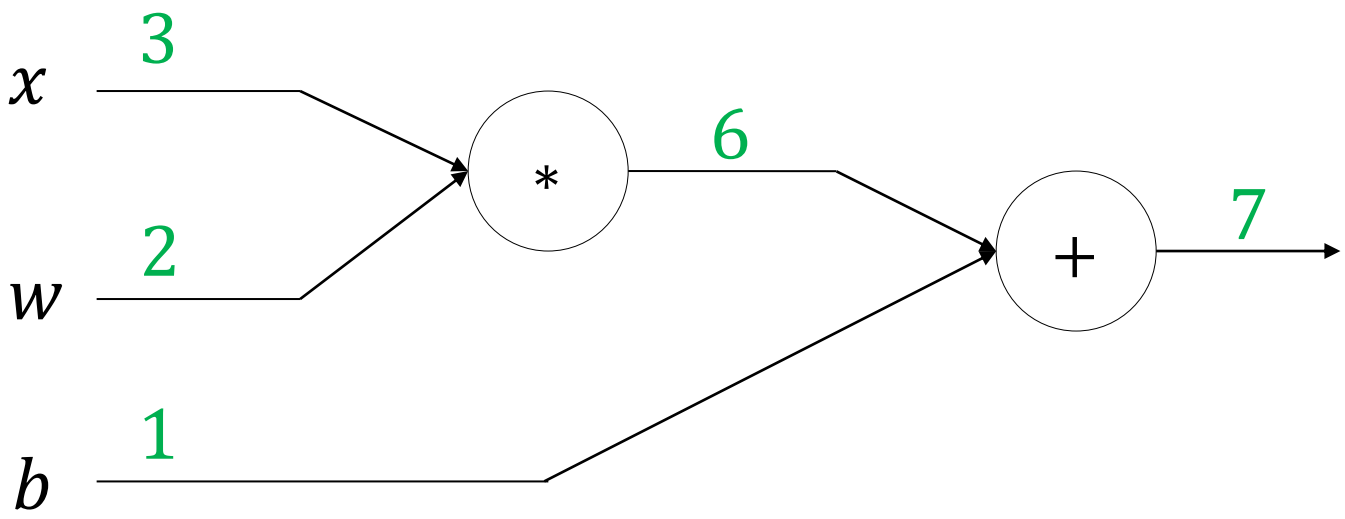
Single Neuron





Single Neuron

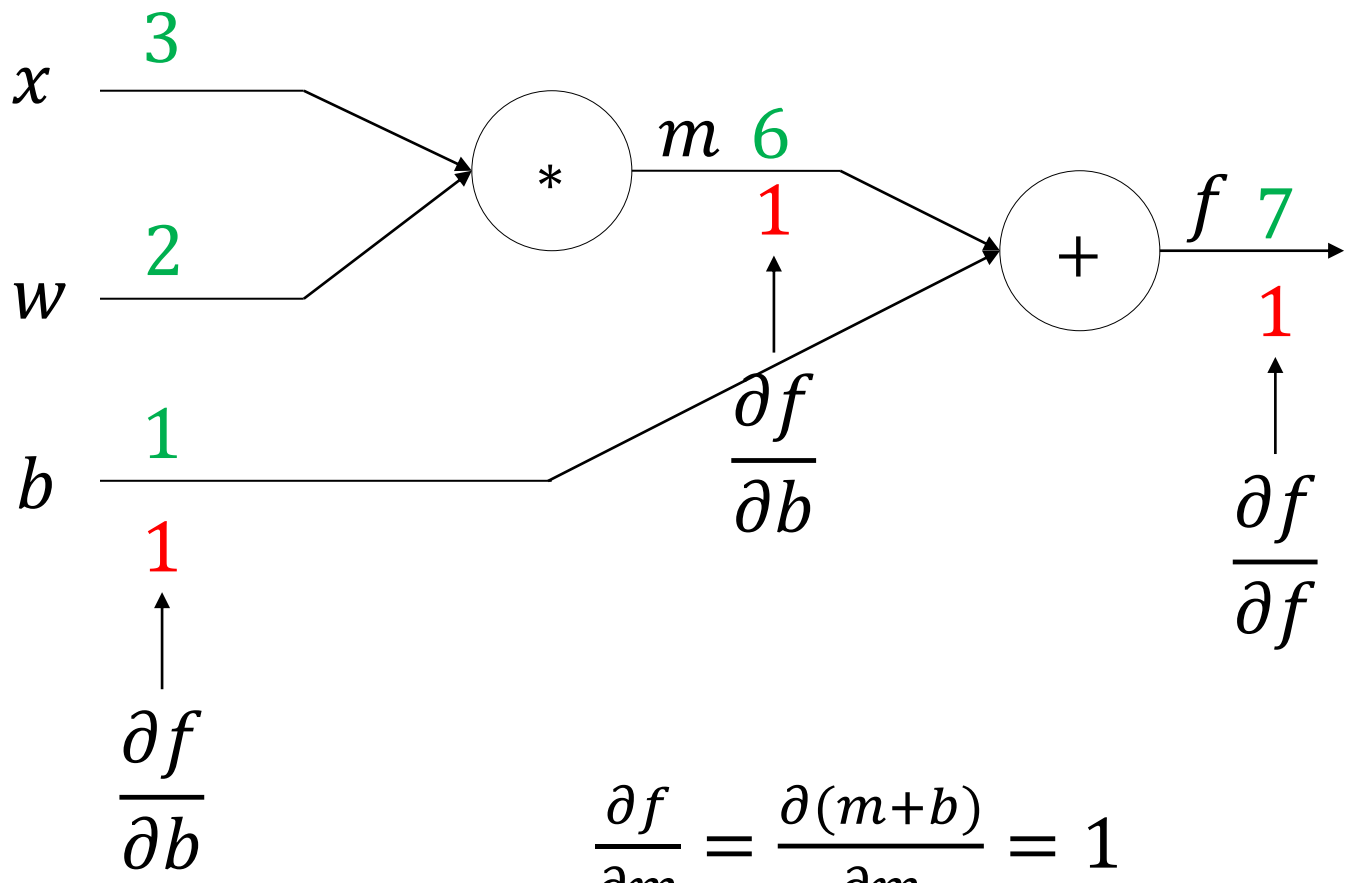




Single Neuron

$f(x, w, b) = x \cdot w + b$

1. Forward Pass:
   - $m = x \cdot w$
   - $f = m + b$
2. Backward Pass:
   - $\frac{\partial f}{\partial x}$
   - $\frac{\partial f}{\partial w}$
   - $\frac{\partial f}{\partial b}$



$$\frac{\partial f}{\partial m} = \frac{\partial (m+b)}{\partial m} = 1$$

$$\frac{\partial f}{\partial b} = \frac{\partial (m+b)}{\partial b} = 1$$

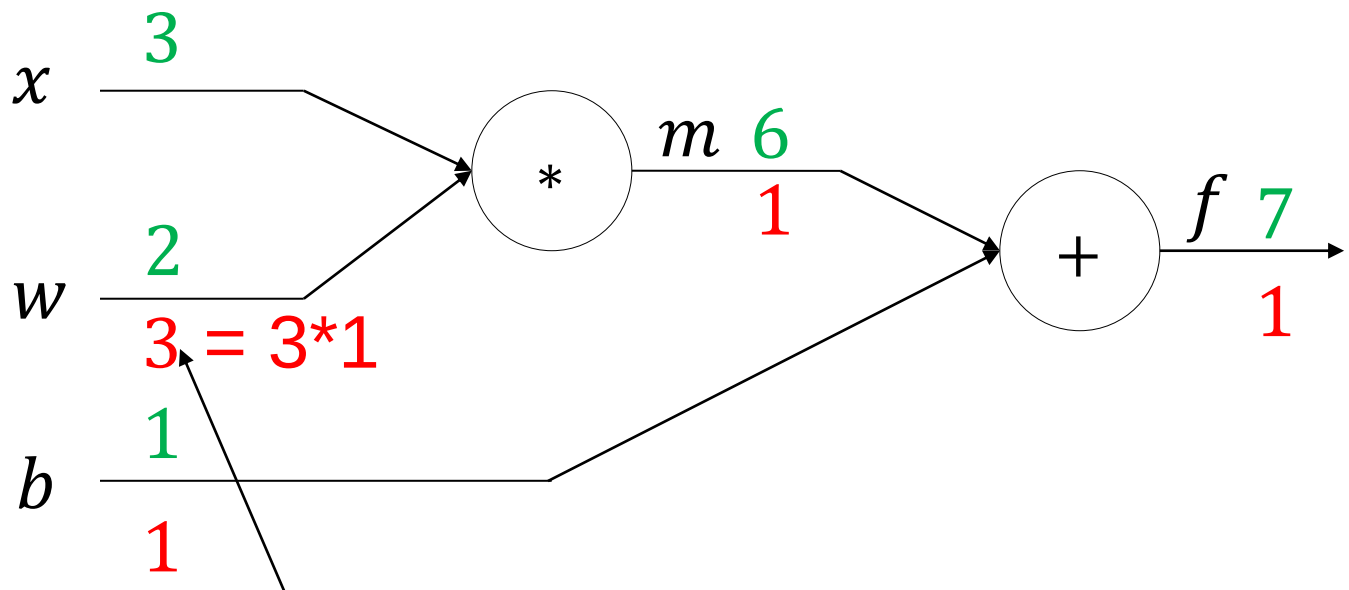
Single Neuron

$f(x, w, b) = x \cdot w + b$

$f(x, w, b) = f(sum(mul(x, w), b))$

1. Forward Pass:
   - $m = x \cdot w$
   - $f = m + b$

2. Backward Pass:
   - $\frac{\partial f}{\partial x}$
   - $\frac{\partial f}{\partial w}$
   - $\frac{\partial f}{\partial b}$



$$\frac{\partial f}{\partial w} = \frac{\partial m}{\partial w}\frac{\partial f}{\partial m}$$

$$\frac{\partial m}{\partial w} = \frac{\partial (x \cdot w)}{\partial w} = x = 3$$

Downstream Gradient

Local Gradient

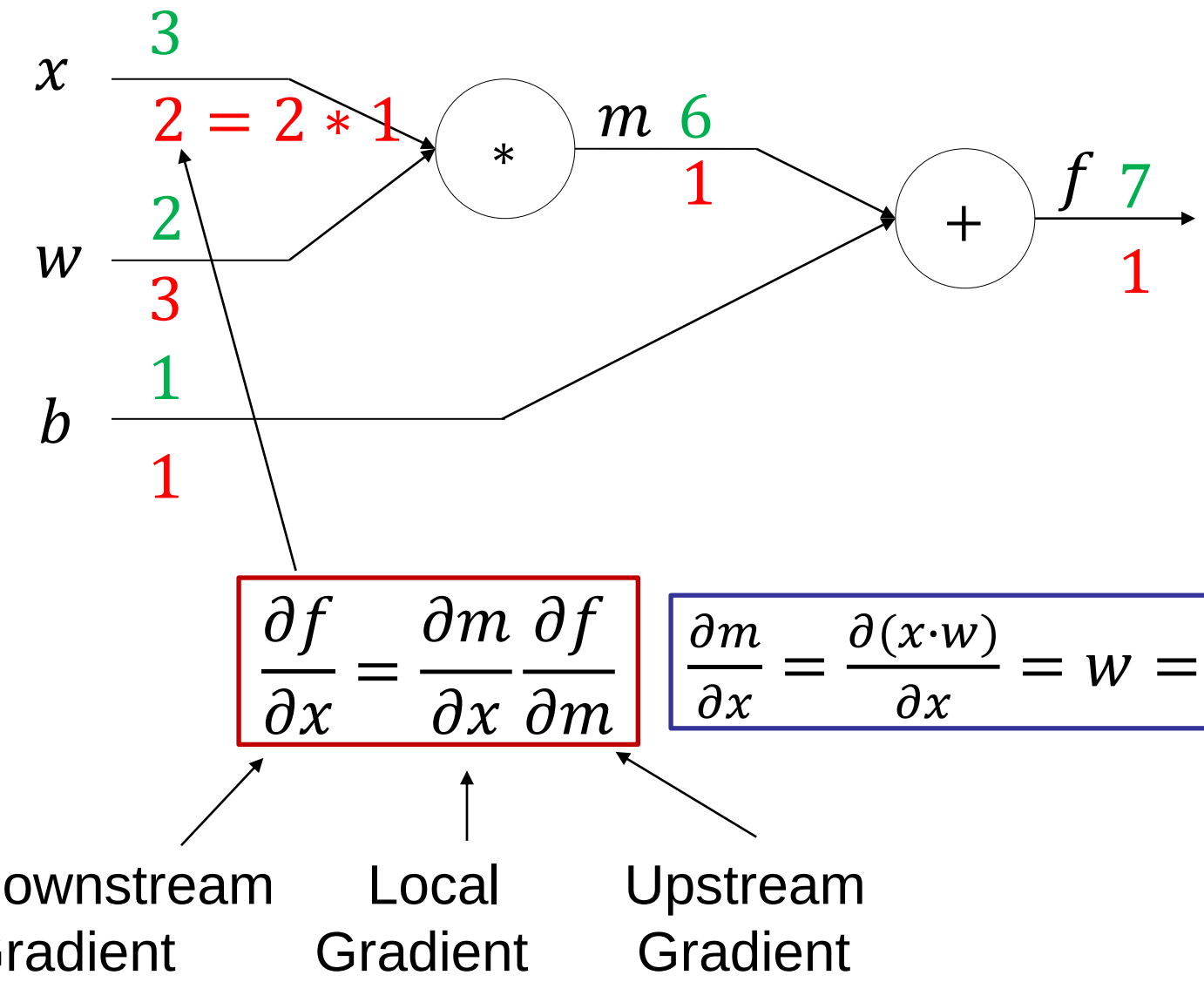
Upstream Gradient

Single Neuron

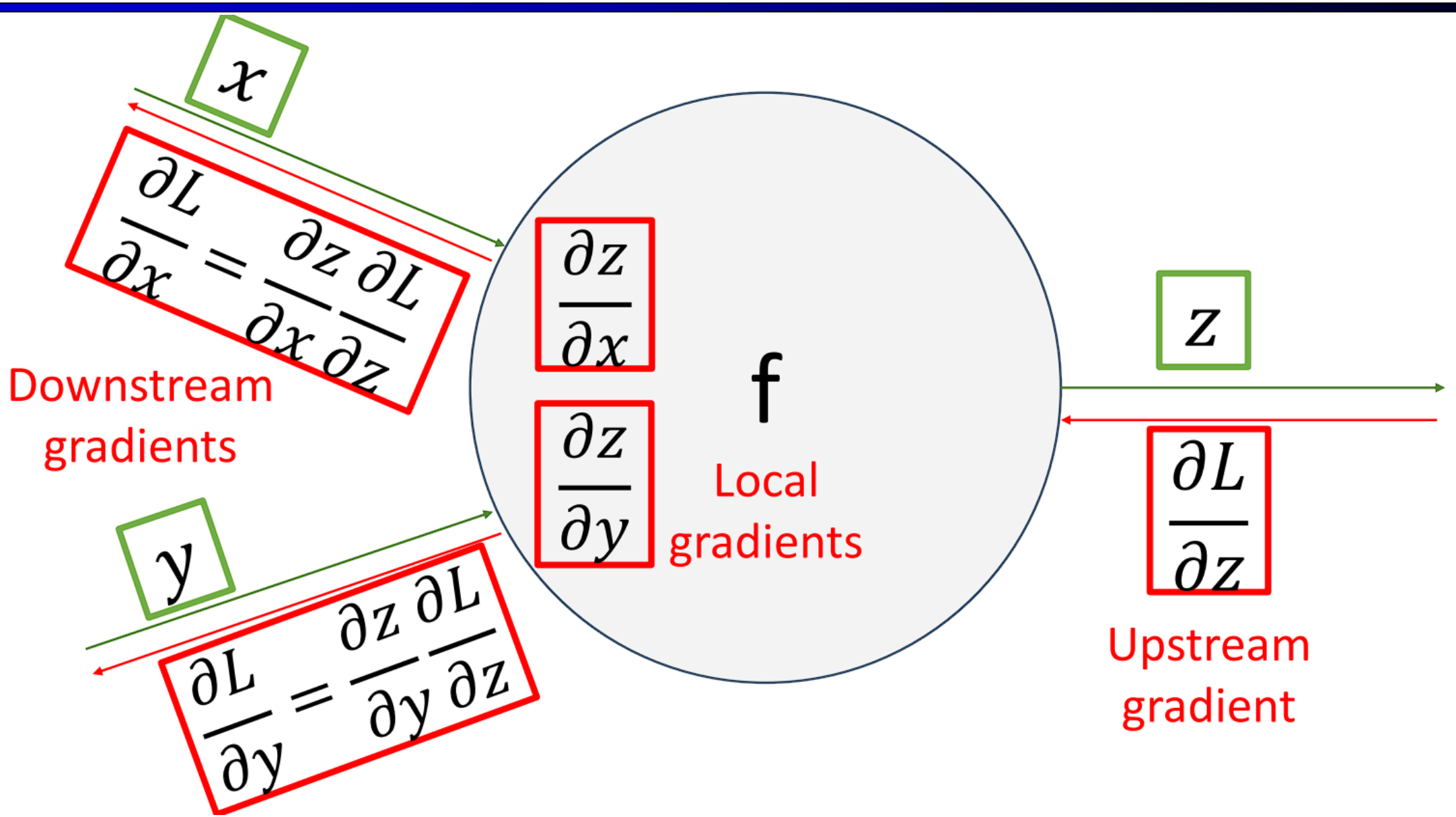
$f(x, w, b) = x \cdot w + b$

$f(x, w, b) = f(sum(mul(x, w), b))$

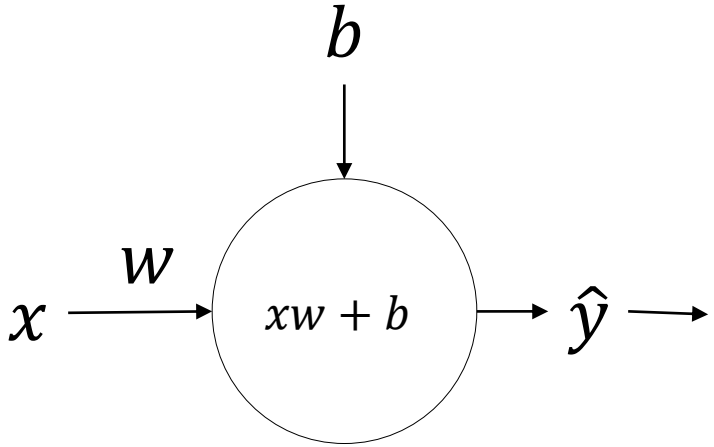
1. Forward Pass:
   - $m = x \cdot w$
   - $f = m + b$
2. Backward Pass:
   - $\frac{\partial f}{\partial x}$
   - $\frac{\partial f}{\partial w}$
   - $\frac{\partial f}{\partial b}$



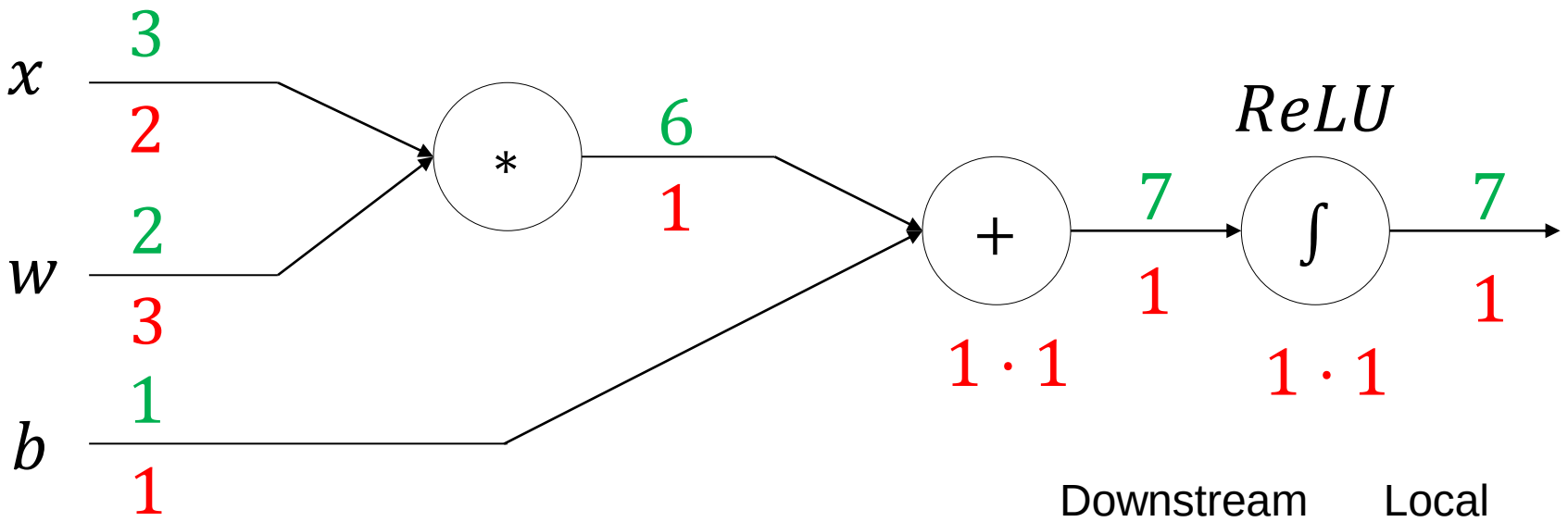
Single Neuron Gradient



Single Neuron

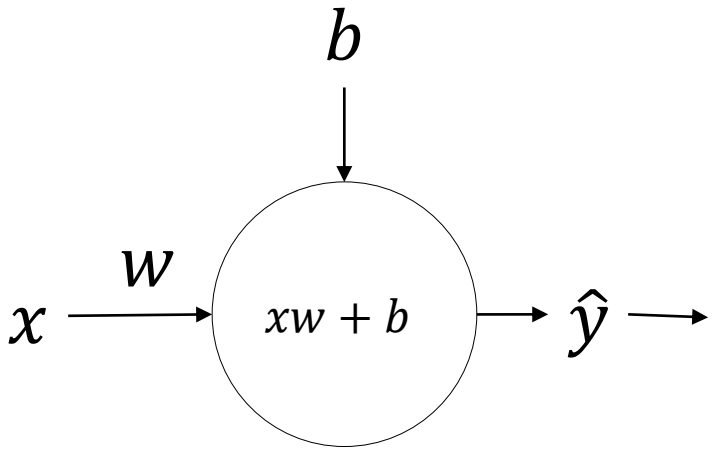


$$\frac{\partial \max(x, 0)}{\partial x} = 1 \; if (x > 0)$$



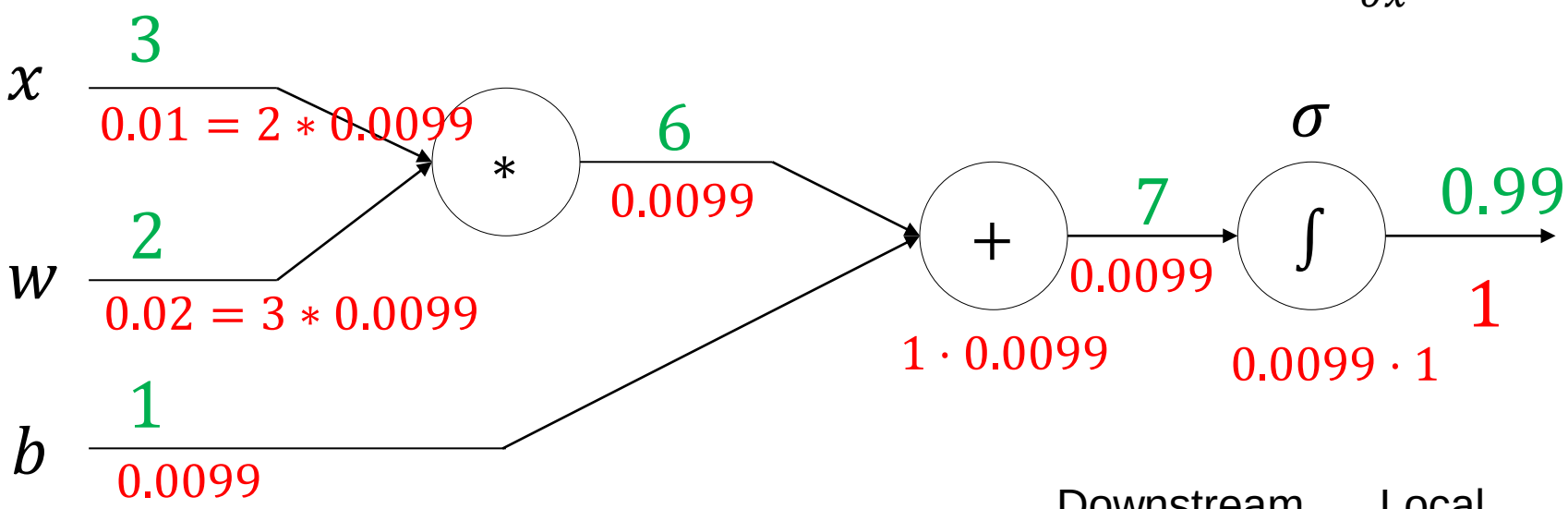
Downstream Gradient | Local Gradient | Upstream Gradient

Single Neuron



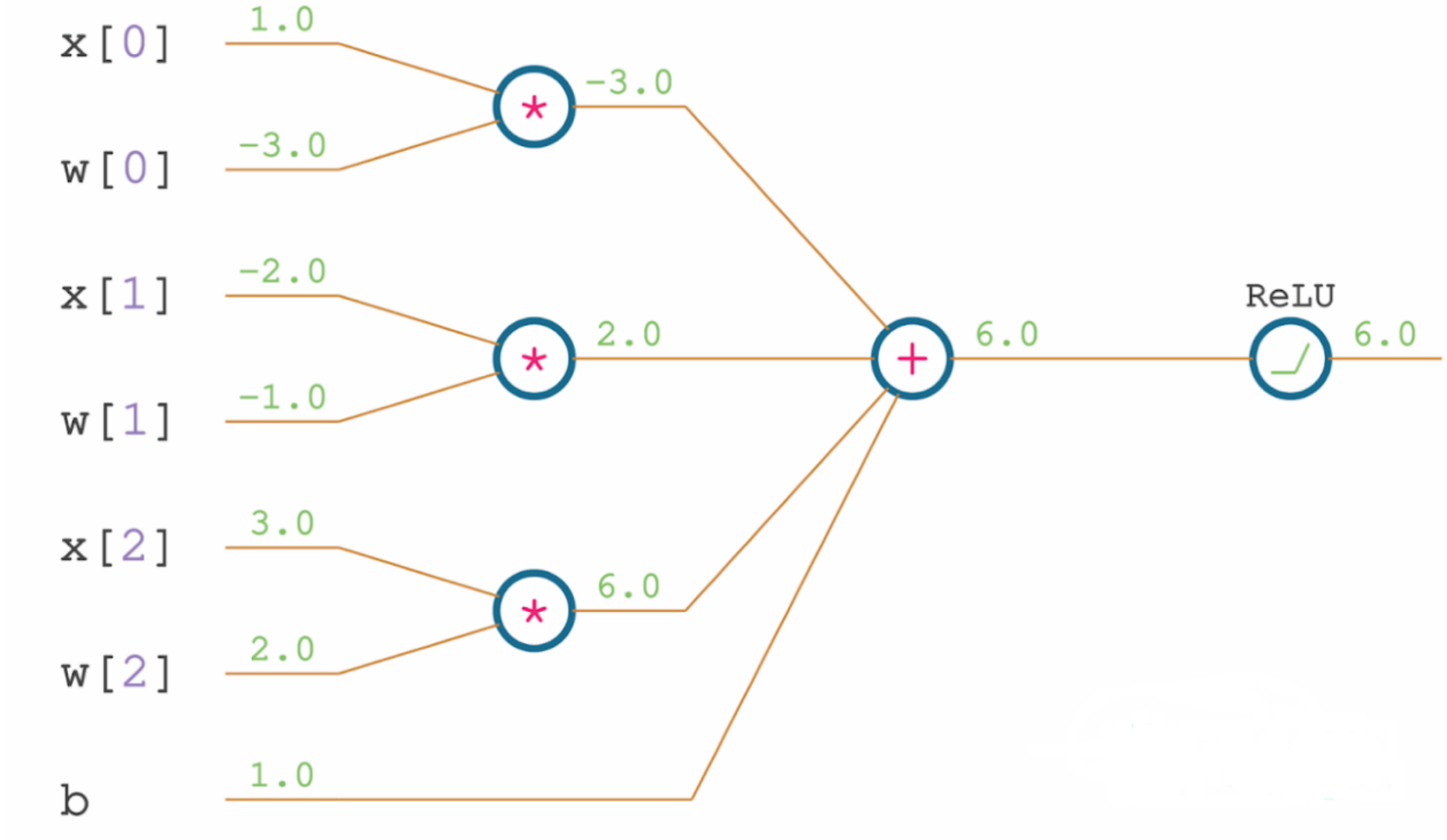
$$\frac{\partial\, \sigma\,(x)}{\partial x} = (1 - \sigma\,(x))\,\sigma\,(x)$$

$$\frac{\partial\, \sigma\,(x)}{\partial x} = (1 - 0.99)0.99$$

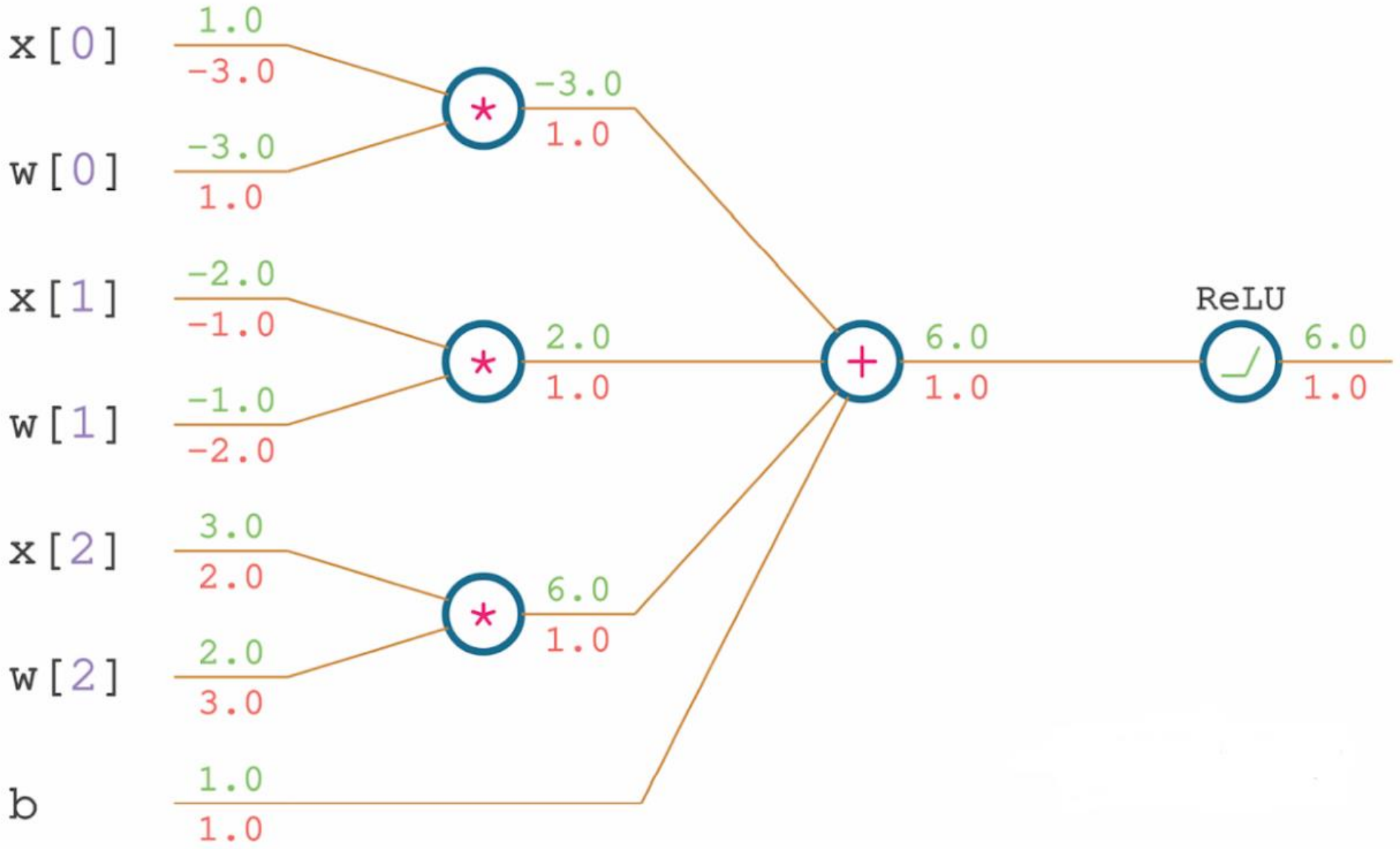


Downstream Gradient | Local Gradient | Upstream Gradient

Example

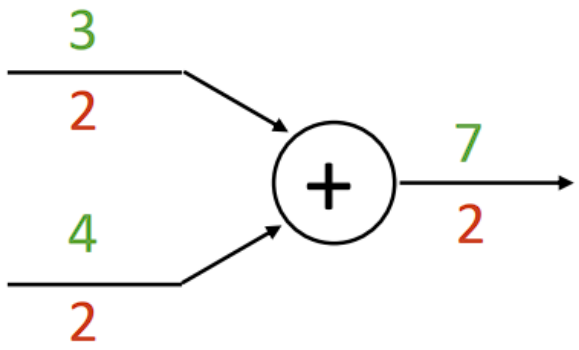


Example

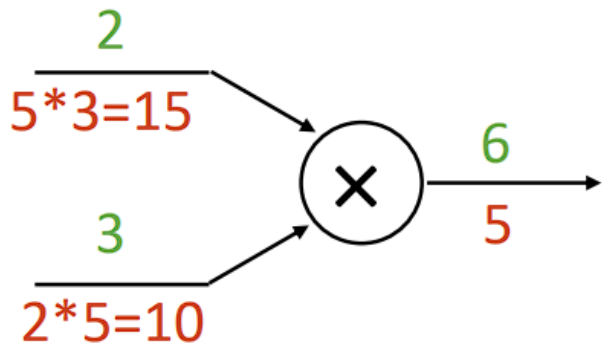

x[0]
w[0]
x[1]
w[1]
x[2]
w[2]
b
1.0
-3.0
-3.0
1.0
-2.0
-1.0
-1.0
-2.0
3.0
2.0
2.0
3.0
1.0
1.0
*
*
*
-3.0
1.0
2.0
1.0
6.0
1.0
+
6.0
1.0
ReLU
6.0
1.0

Pattern in Gradient Flow

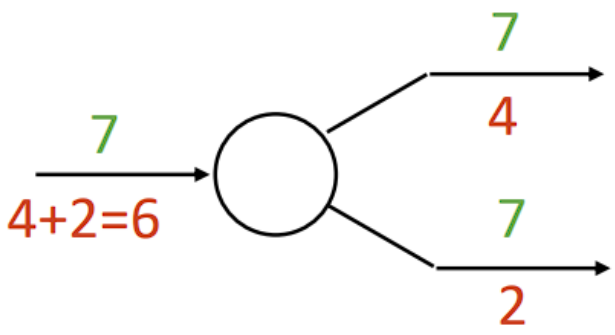
add gate: gradient distributor



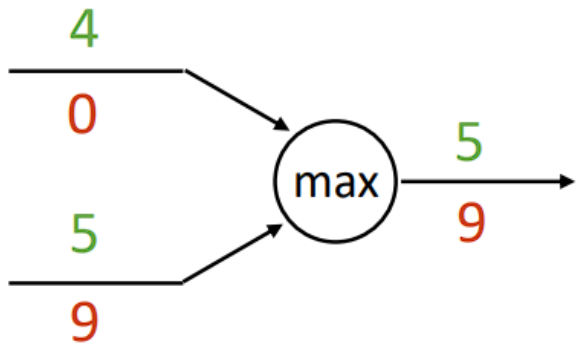
mul gate: "swap multiplier"



copy gate: gradient adder



max gate: gradient router



Vector Derivatives

$x \in \mathbb{R}, y \in \mathbb{R}$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

$x \in \mathbb{R}^N, y \in \mathbb{R}$

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N,$$

$$\left(\frac{\partial y}{\partial x}\right)_i = \frac{\partial y}{\partial x_i}$$

For each element of x, if it changes by a small amount then how much will y change?

$x \in \mathbb{R}^N, y \in \mathbb{R}^M$

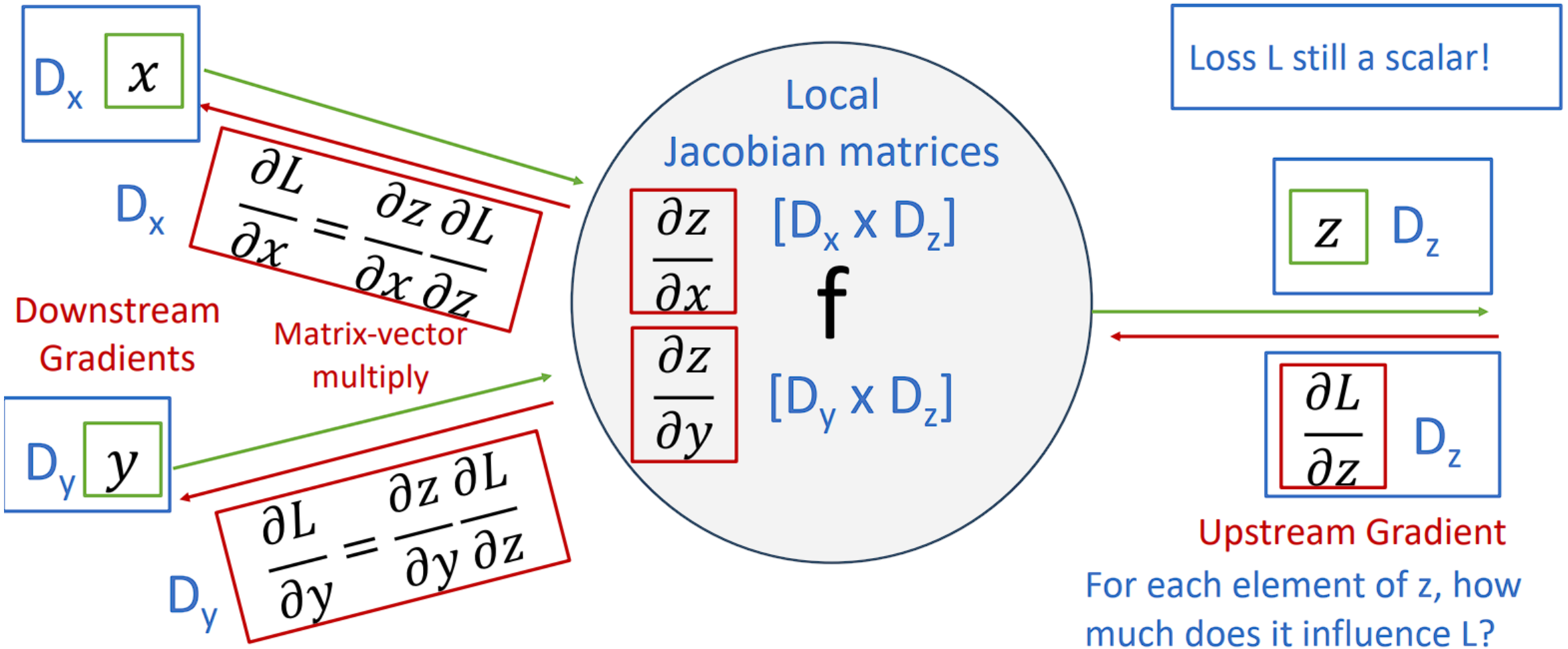
Derivative is **Jacobian**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M}$$

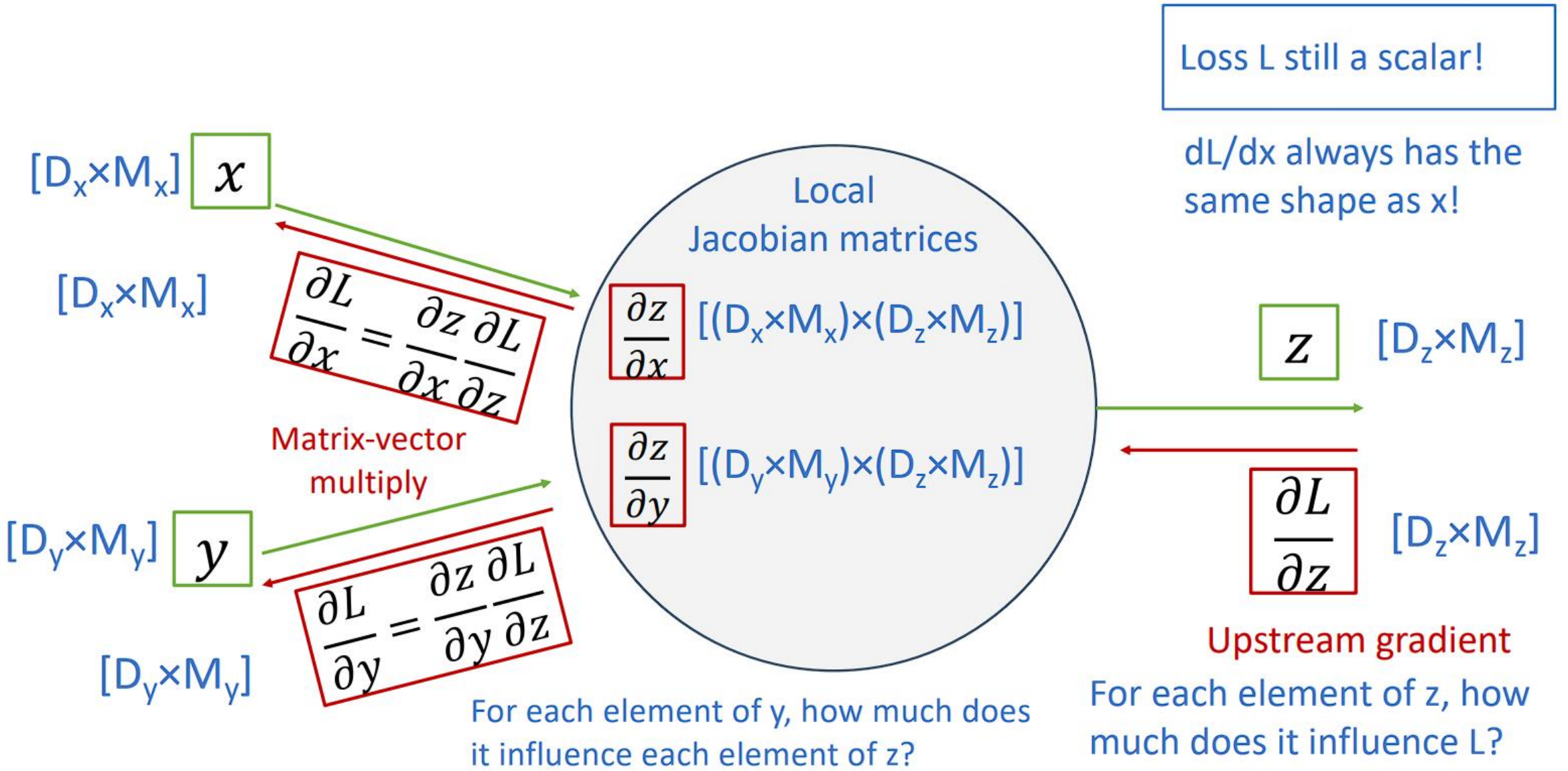
$$\left(\frac{\partial y}{\partial x}\right)_{i,j} = \frac{\partial y_j}{\partial x_i}$$

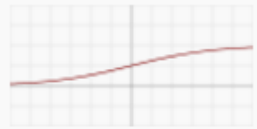
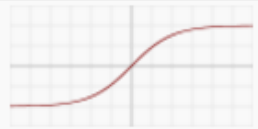
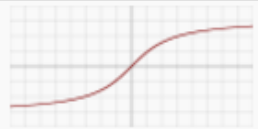
For each element of x, if it changes by a small amount then how much will each element of y change?

Backpropagation with Vectors



Backpropagation with Metrics



Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Loss Function and Their Local Gradient

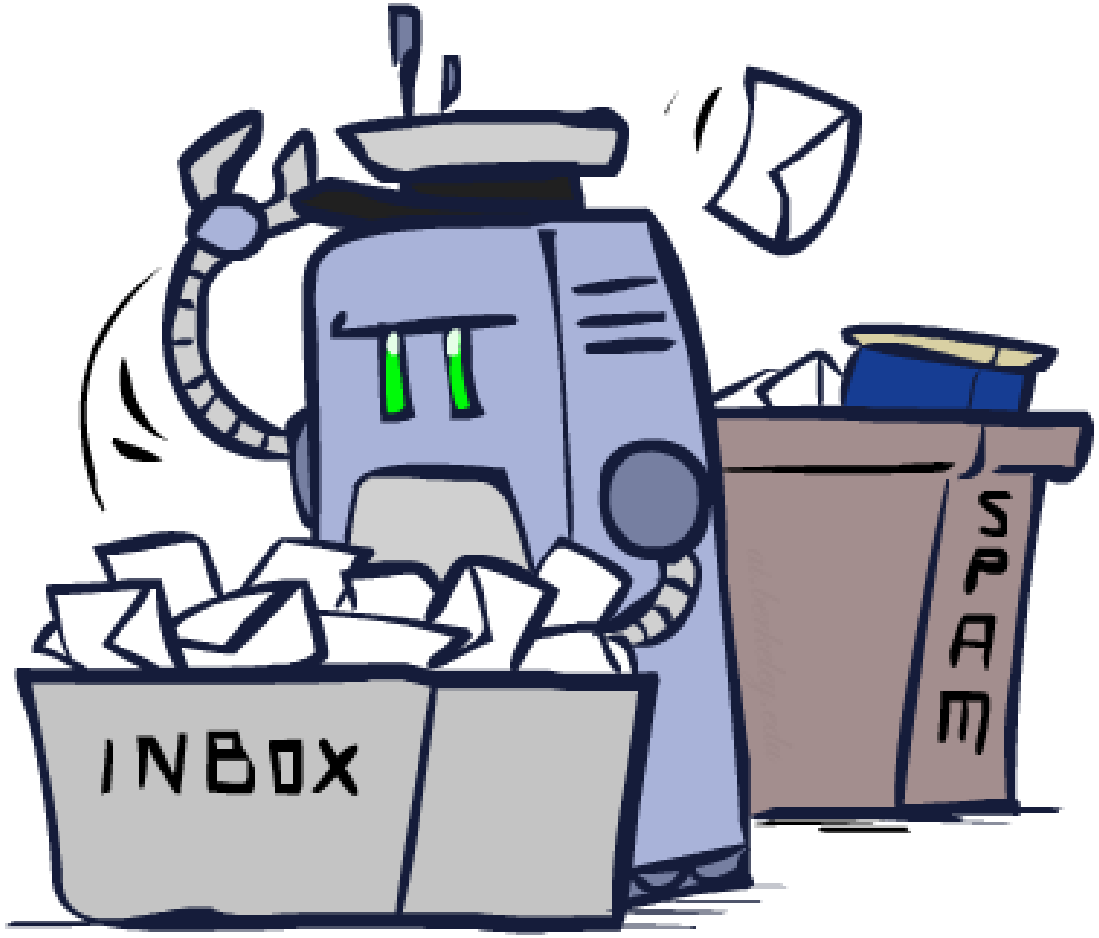
Mean Square Error	$L_i = \frac{1}{n}\sum_{i=0}^{n}(\boldsymbol{y}_i - \hat{y}_i)^2$	$\frac{\partial L_i}{\partial \hat{y}} = -\frac{2}{n}(\boldsymbol{y}_i - \hat{y}_i)$
Binary Cross Entropy	$L_i = -y_i \cdot \log(\widehat{y_i}) - (1-y) \cdot \log(1-\widehat{y_i})$	$\frac{\partial L_i}{\partial \hat{y}} = -\frac{y_i}{\hat{y}_i} - \frac{1-y_i}{1-\hat{y}_i}$
Categorical Cross Entropy	$L_i = -\sum_{i} y_i \cdot \log(\widehat{y_i})$	$\frac{\partial L_i}{\partial \hat{y}} = -\frac{y_i}{\hat{y}_i}$

Back-propagation Algorithm

- In a back-propagation neural network, the learning algorithm has 2 phases.
  1. Forward propagation of inputs
  2. Backward propagation of errors
- The algorithm loops over the 2 phases until the errors obtained are lower than a certain threshold.
- Learning is done in a similar manner as in a perceptron
  - A set of training inputs is presented to the network.
  - The network computes the outputs.
  - The weights are adjusted to reduce errors.
- The activation function used is a sigmoid function.

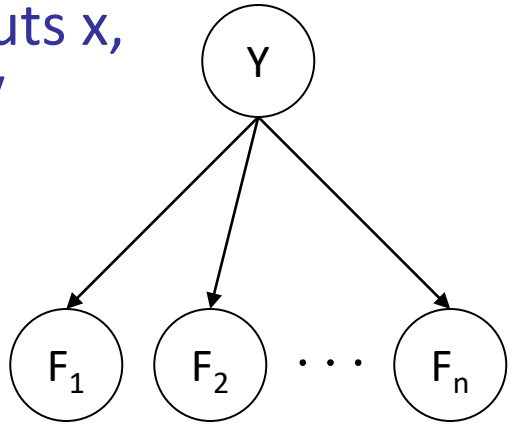
$$Y^{sigmoid} = \frac{1}{1+e^{-X}}$$

Classification



Last Time

- Classification: given inputs x, predict labels (classes) y



- Naïve Bayes

$$P(Y|F_{0,0} \dots F_{15,15}) \propto P(Y) \prod_{i,j} P(F_{i,j}|Y)$$

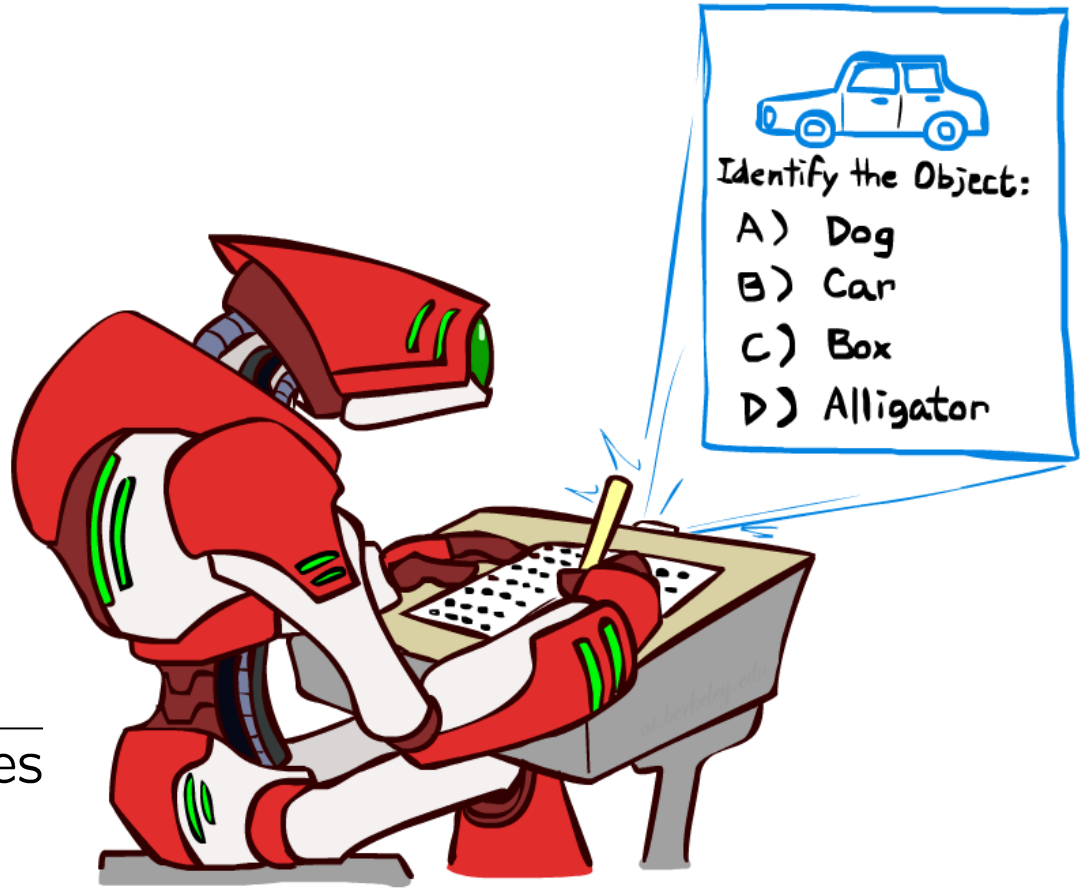
- Parameter estimation:
  - MLE, MAP, priors

$$P_{\text{ML}}(x) = \frac{\text{count}(x)}{\text{total samples}}$$

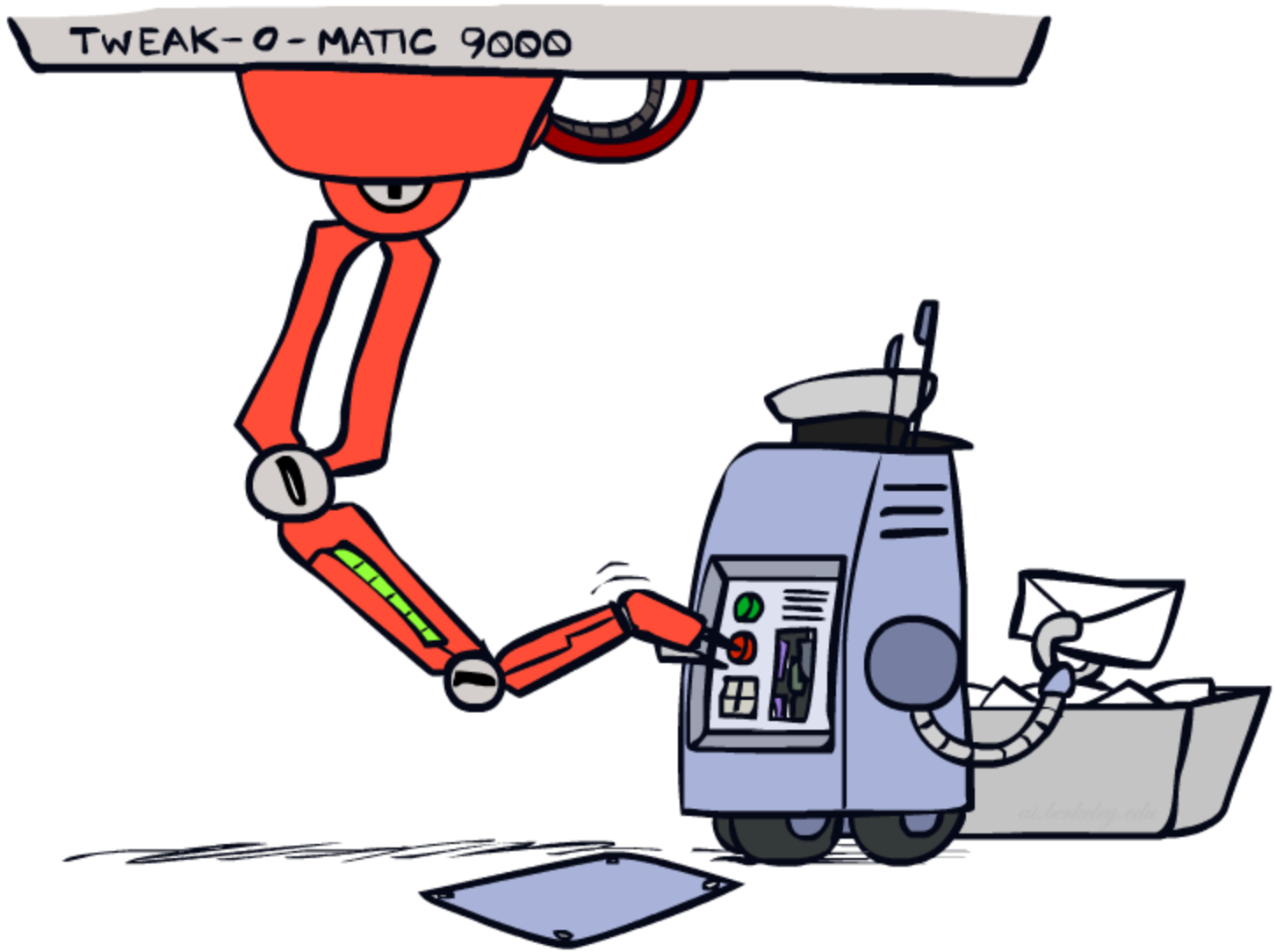
- Laplace smoothing

$$P_{LAP,k}(x) = \frac{c(x) + k}{N + k|X|}$$

- Training set, held-out set, test set

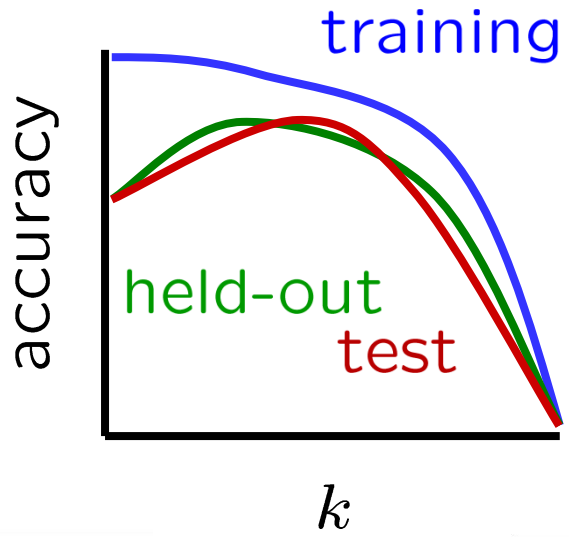


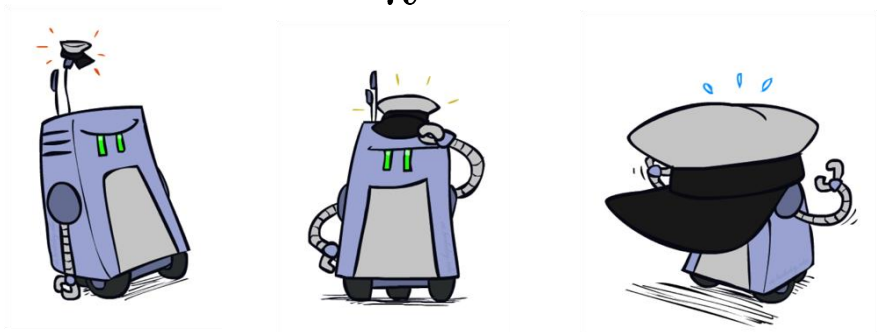
Tuning



Tuning on Held-Out Data

- Now we've got two kinds of unknowns
  - Parameters: the probabilities P(X|Y), P(Y)
  - Hyperparameters: e.g. the amount / type of smoothing to do, k, α
- What should we learn where?
  - Learn parameters from training data
  - Tune hyperparameters on different data
    - Why?
  - For each value of the hyperparameters, train and test on the held-out data
  - Choose the best value and do a final test on the test data





Baselines

- First step: get a baseline
  - Baselines are very simple “straw man” procedures
  - Help determine how hard the task is
  - Help know what a “good” accuracy is

- Weak baseline: most frequent label classifier
  - Gives all test instances whatever label was most common in the training set
  - E.g. for spam filtering, might label everything as ham
  - Accuracy might be very high if the problem is skewed
  - E.g. calling everything “ham” gets 66%, so a classifier that gets 70% isn’t very good…

- For real research, usually use previous work as a (strong) baseline

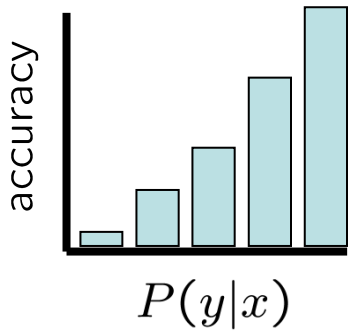
Confidences from a Classifier

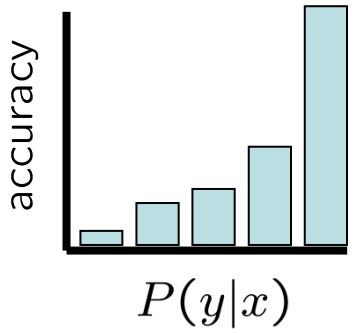
- The confidence of a probabilistic classifier:
  - Posterior over the top label

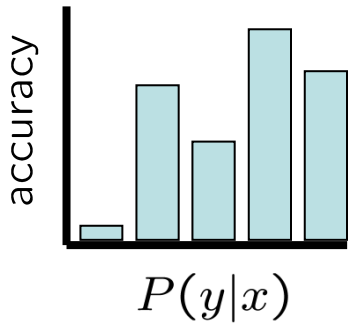
$$\text{confidence}(x) = \max_y P(y|x)$$

  - Represents how sure the classifier is of the classification
  - Any probabilistic model will have confidences
  - No guarantee confidence is correct

- Calibration
  - Weak calibration: higher confidences mean higher accuracy
  - Strong calibration: confidence predicts accuracy rate
  - What's the value of calibration?







Errors, and What to Do

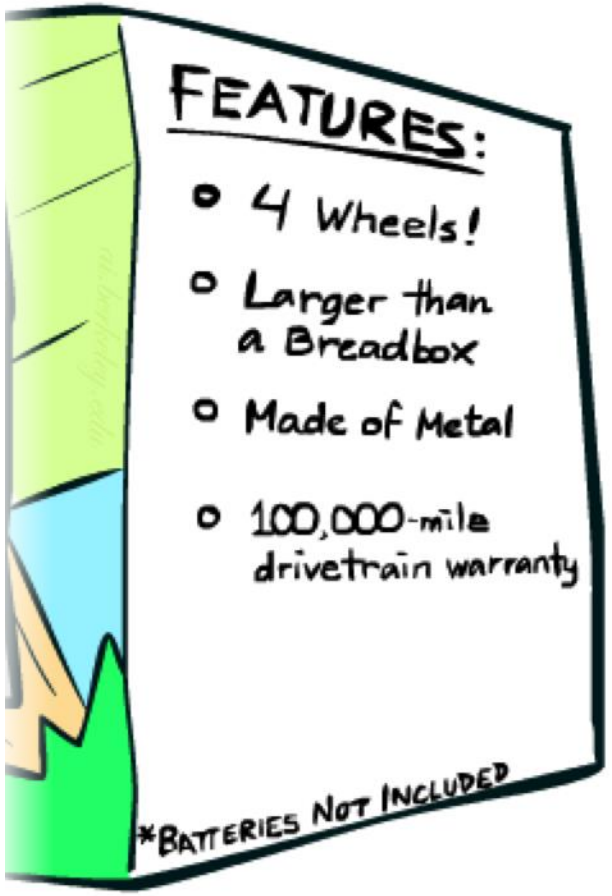
- Examples of errors

> Dear GlobalSCAPE Customer,
>
> GlobalSCAPE has partnered with ScanSoft to offer you the latest version of OmniPage Pro, for just $99.99* - the regular list price is $499! The most common question we've received about this offer is - Is this genuine? We would like to assure you that this offer is authorized by ScanSoft, is genuine and valid. You can get the . . .

> . . . To receive your $30 Amazon.com promotional certificate, click through to
>
> http://www.amazon.com/apparel
>
> and see the prominent link for the $30 offer. All details are there. We hope you enjoyed receiving this message. However, if you'd rather not receive future e-mails announcing new store launches, please click . . .

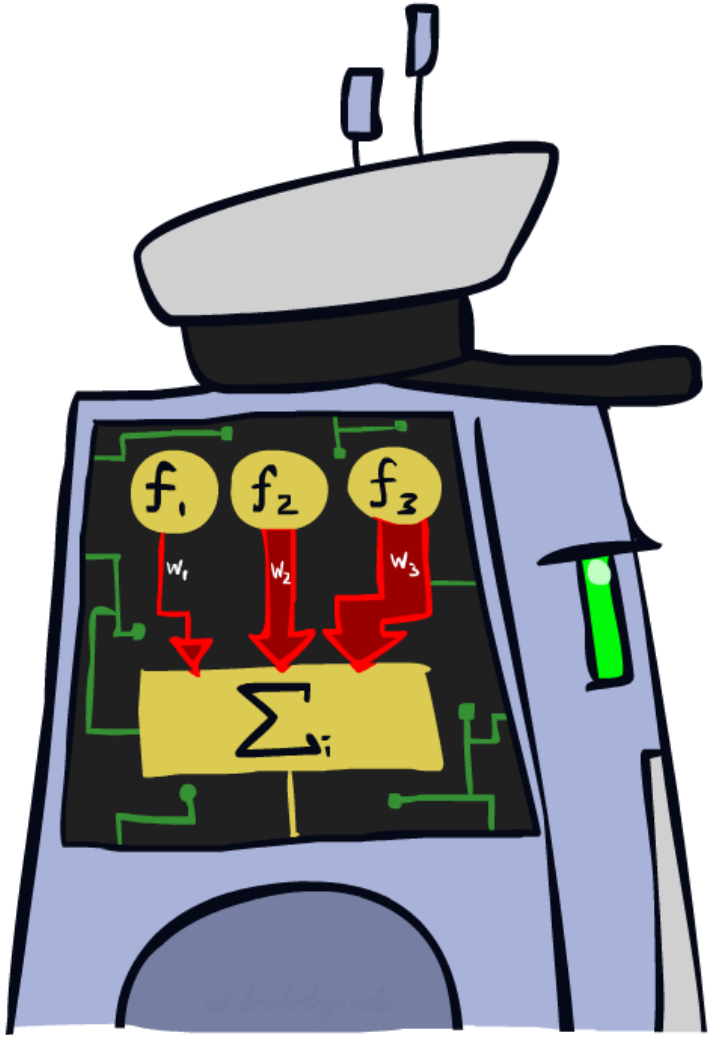
What to Do About Errors?

- Need more features– words aren't enough!
  - Have you emailed the sender before?
  - Have 1K other people just gotten the same email?
  - Is the sending information consistent?
  - Is the email in ALL CAPS?
  - Do inline URLs point where they say they point?
  - Does the email address you by (your) name?
- Can add these information sources as new variables in the NB model
- …but NB must *model* all of the features
- Features often not independent, NB is not a good model in this case



Error-Driven Classification

Linear Classifiers

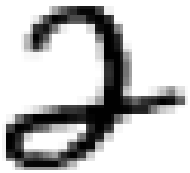


Feature Vectors

x $f(x)$ y

```
Hello,

Do you want free printr
cartriges?  Why pay more
when you can get them
ABSOLUTELY FREE!  Just
```



```
# free       : 2
YOUR_NAME    : 0
MISSPELLED   : 2
FROM_FRIEND  : 0
...
```

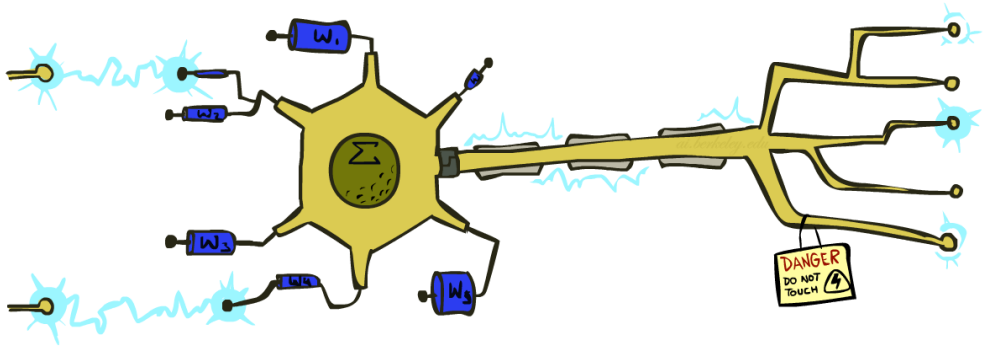
```
PIXEL-7,12   : 1
PIXEL-7,13   : 0
...
NUM_LOOPS    : 1
...
```

SPAM
or
+

“2”

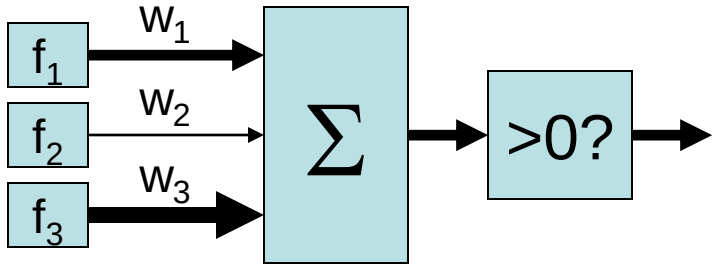
Linear Classifiers

- Inputs are feature values
- Each feature has a weight
- Sum is the activation



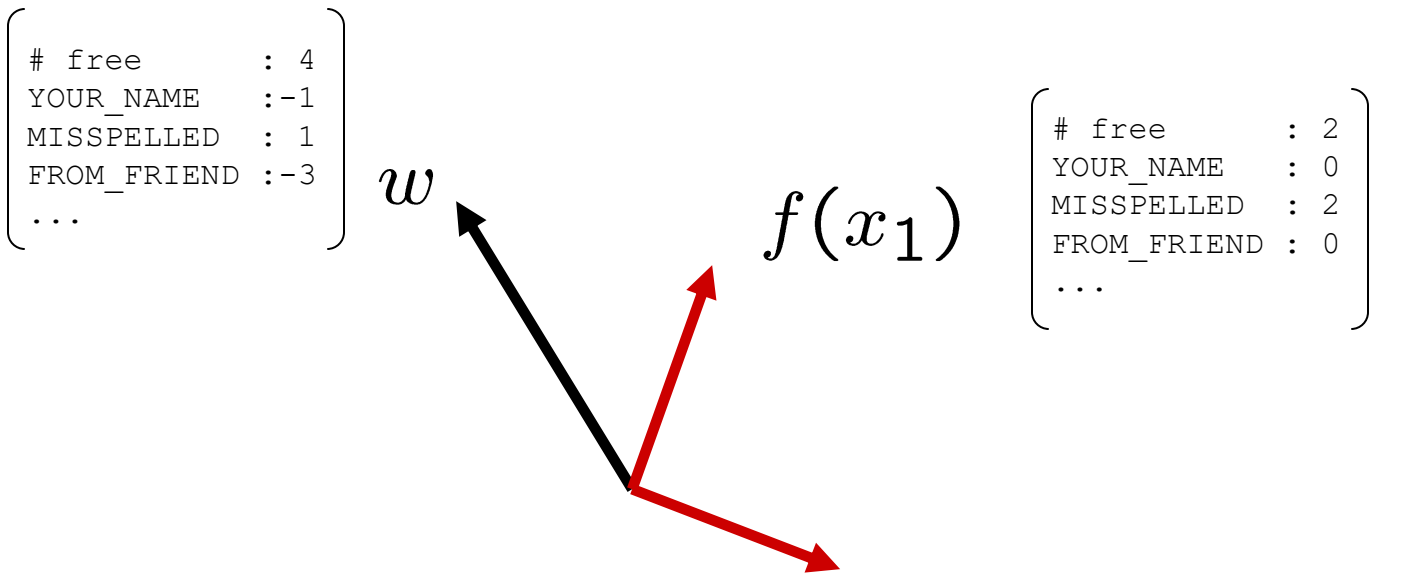
$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- If the activation is:
  - Positive, output +1
  - Negative, output -1



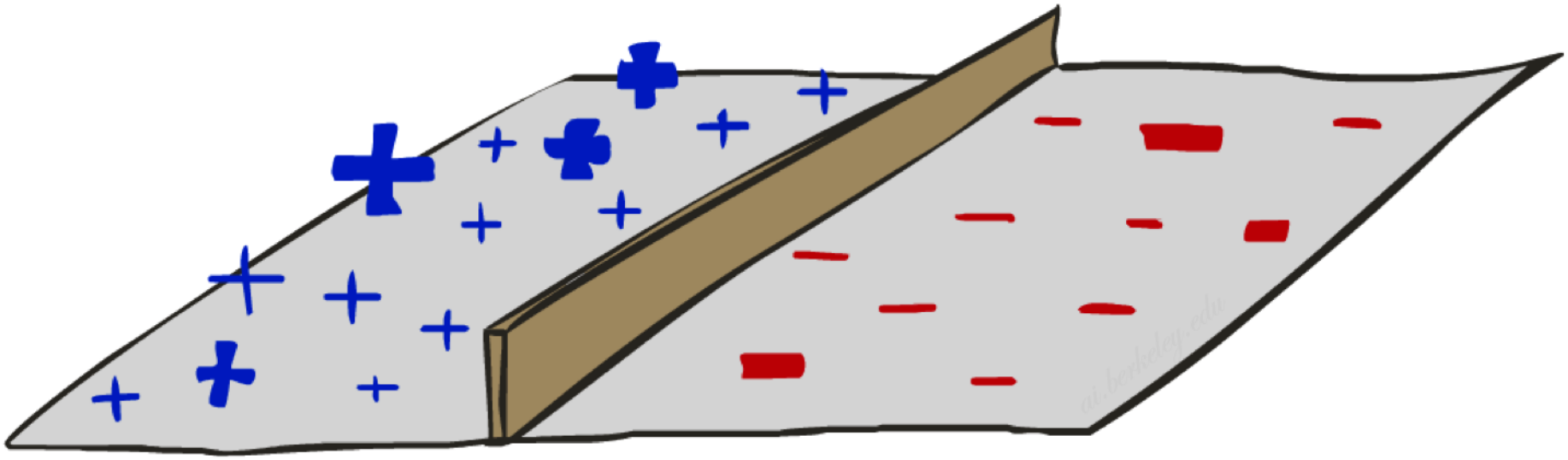
Weights

- Binary case: compare features to a weight vector
- Learning: figure out the weight vector from examples



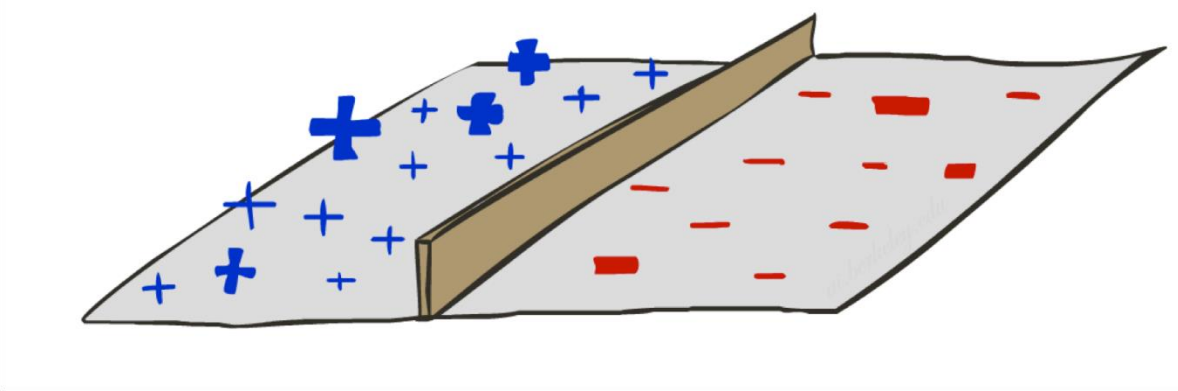
Dot product $w \cdot f$ positive means the positive class

Decision Rules



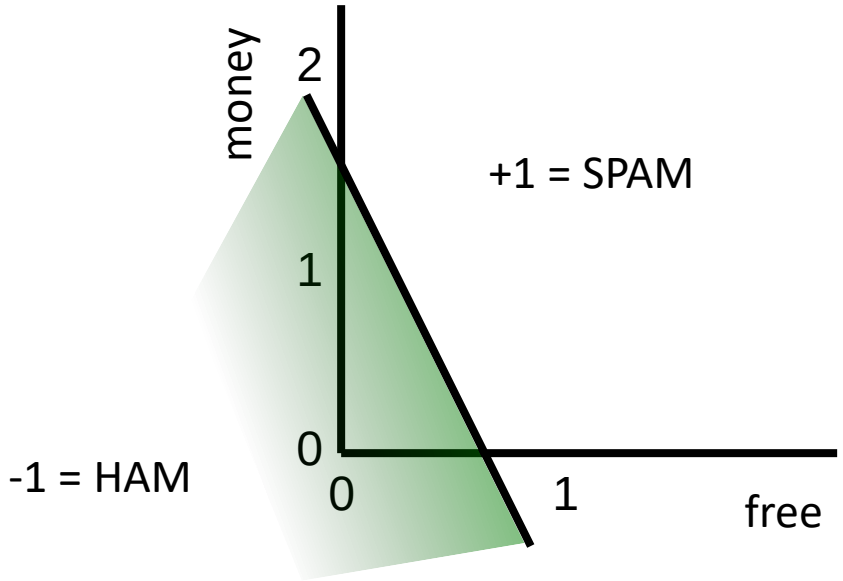
Binary Decision Rule

- In the space of feature vectors
  - Examples are points
  - Any weight vector is a hyperplane
  - One side corresponds to Y=+1
  - Other corresponds to Y=-1



w

```
BIAS   : -3
free   :  4
money  :  2
...
```

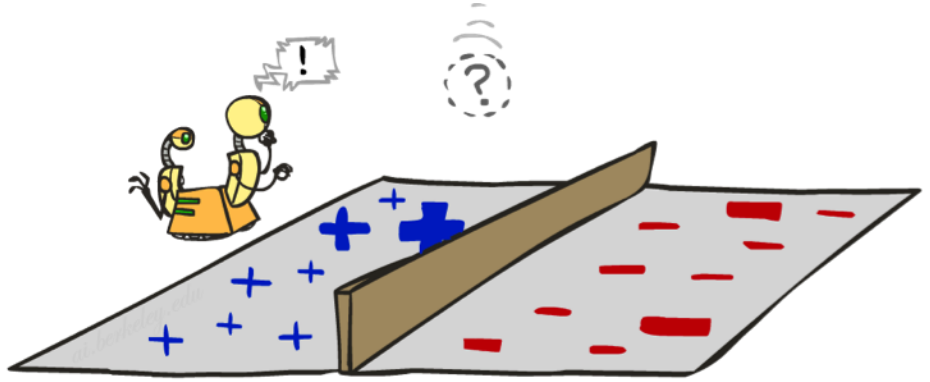


$f \cdot w = 0$

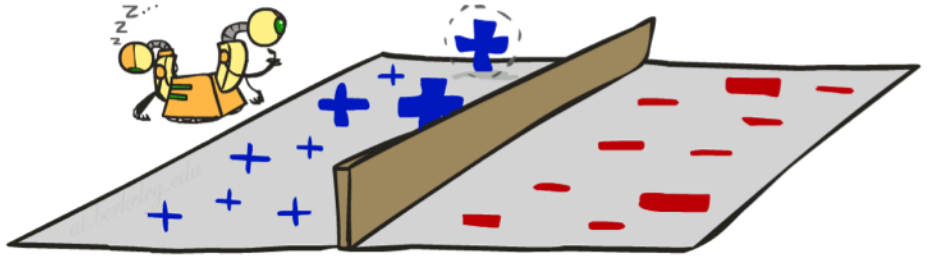
Weight Updates

Learning: Binary Perceptron

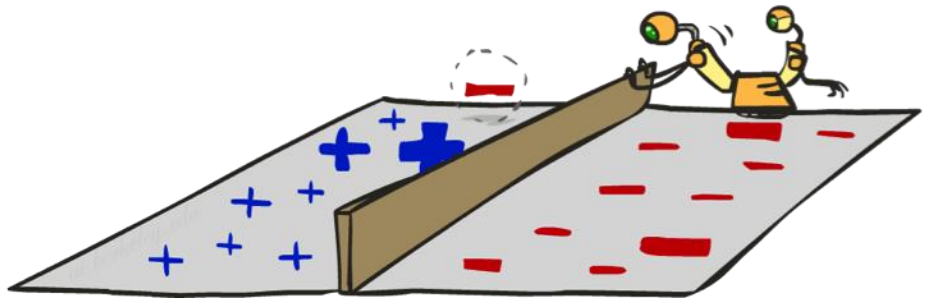
- Start with weights = 0
- For each training instance:
  - Classify with current weights



  - If correct (i.e., y=y*), no change!



  - If wrong: adjust the weight vector



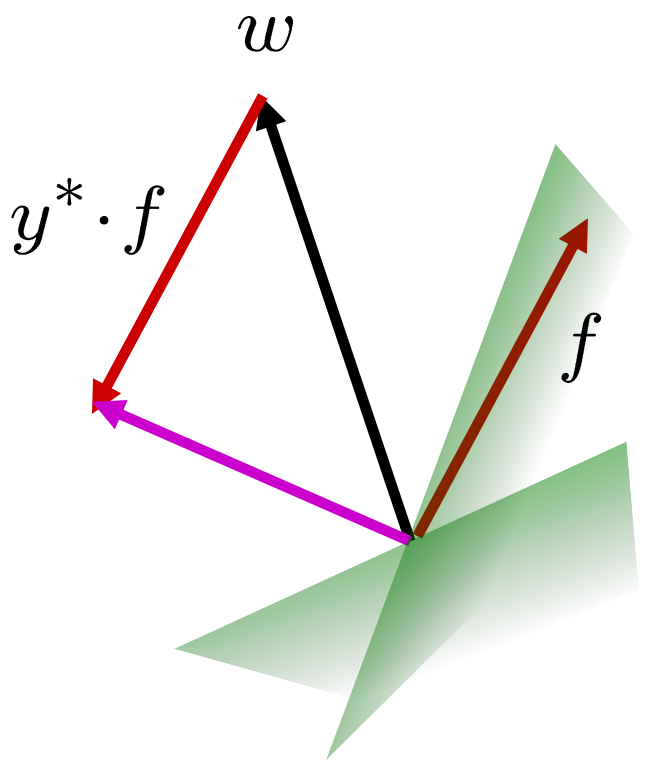
Learning: Binary Perceptron

- Start with weights = 0
- For each training instance:
  - Classify with current weights

$$y = \begin{cases} +1 & \text{if } w \cdot f(x) \geq 0 \\ -1 & \text{if } w \cdot f(x) < 0 \end{cases}$$

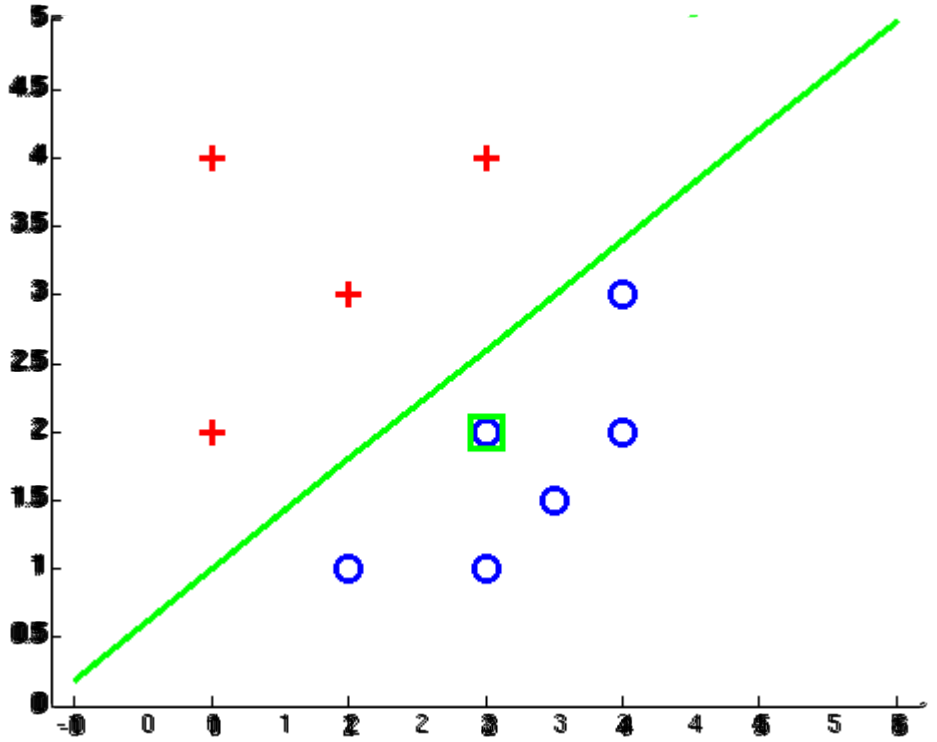
  - If correct (i.e., y=y*), no change!
  - If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if y* is -1.

$$w = w + y^* \cdot f$$



Examples: Perceptron

- Separable Case



Multiclass Decision Rule

- If we have multiple classes:
  - A weight vector for each class:

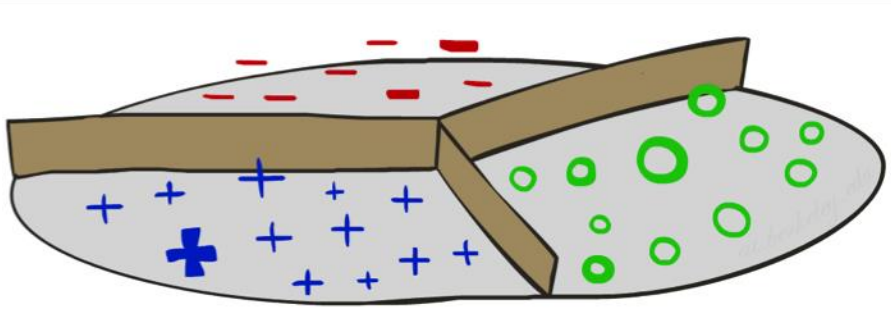
$$w_y$$

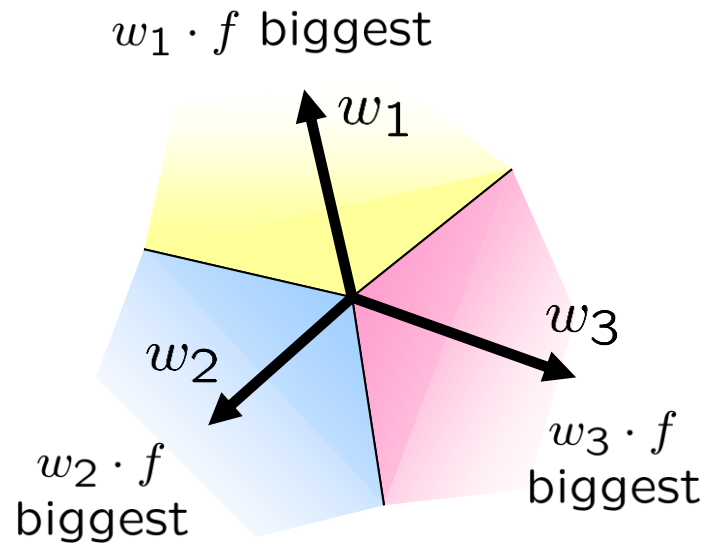
  - Score (activation) of a class y:

$$w_y \cdot f(x)$$

  - Prediction highest score wins

$$y = \arg\max_y \ w_y \cdot f(x)$$





Binary = multiclass where the negative class has weight zero

Learning: Multiclass Perceptron

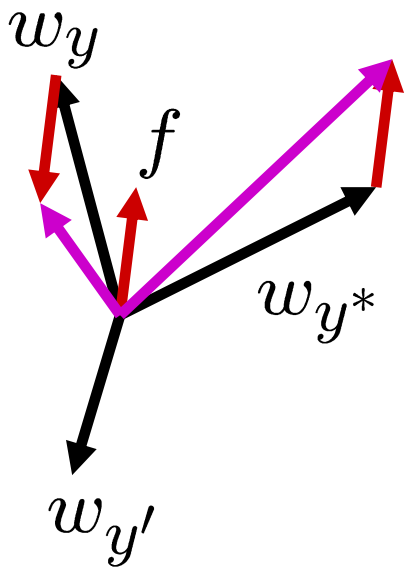
- Start with all weights = 0
- Pick up training examples one by one
- Predict with current weights

$$y = \arg\max_y \; w_y \cdot f(x)$$

- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer

$$w_y = w_y - f(x)$$

$$w_{y^*} = w_{y^*} + f(x)$$



Example: Multiclass Perceptron

"win the vote"

"win the election"

"win the game"

w_{SPORTS}

```
BIAS  : 1
win   : 0
game  : 0
vote  : 0
the   : 0
...
```

$w_{POLITICS}$

```
BIAS  : 0
win   : 0
game  : 0
vote  : 0
the   : 0
...
```

w_{TECH}

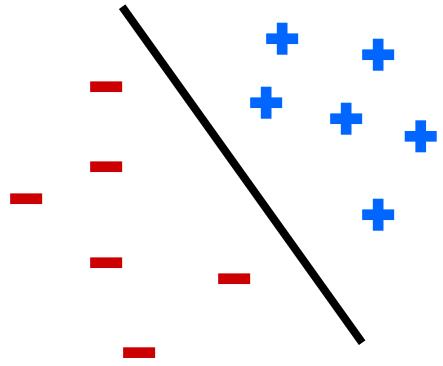
```
BIAS  : 0
win   : 0
game  : 0
vote  : 0
the   : 0
...
```

Properties of Perceptrons

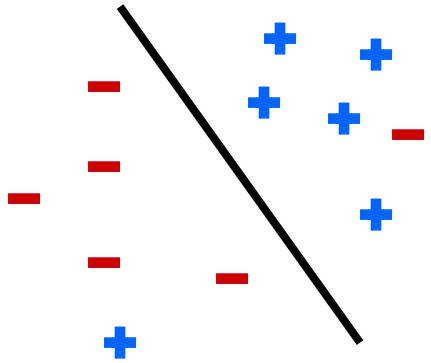
- Separability: true if some parameters get the training set perfectly correct
- Convergence: if the training is separable, perceptron will eventually converge (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

$$\text{mistakes} < \frac{k}{\delta^2}$$

Separable

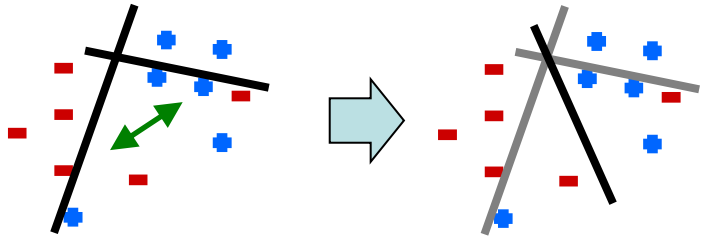


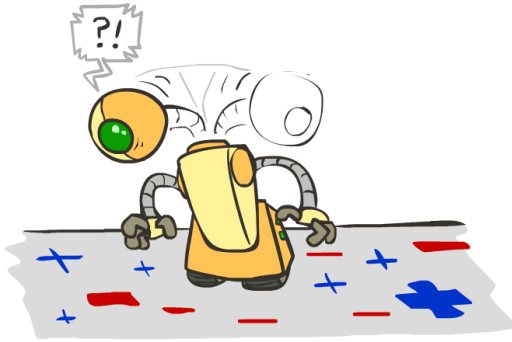
Non-Separable



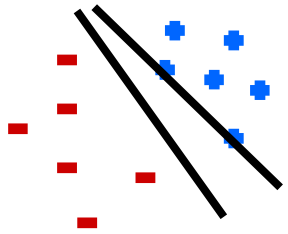
Problems with the Perceptron

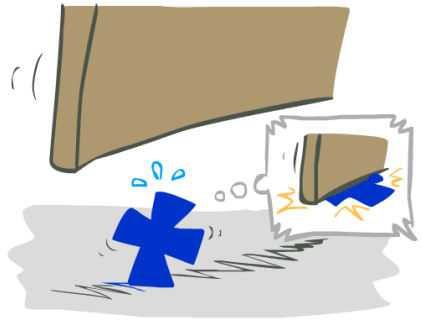
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)



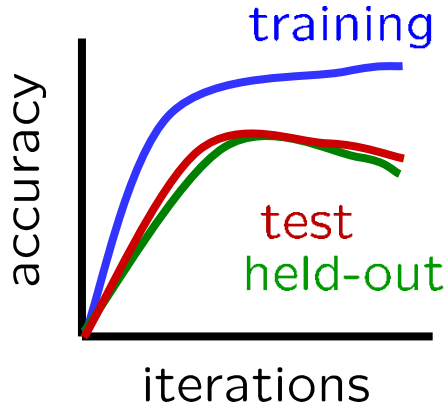


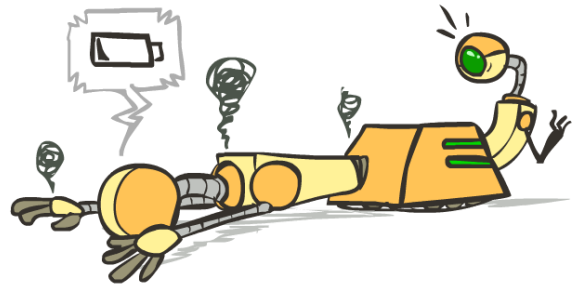
- Mediocre generalization: finds a "barely" separating solution



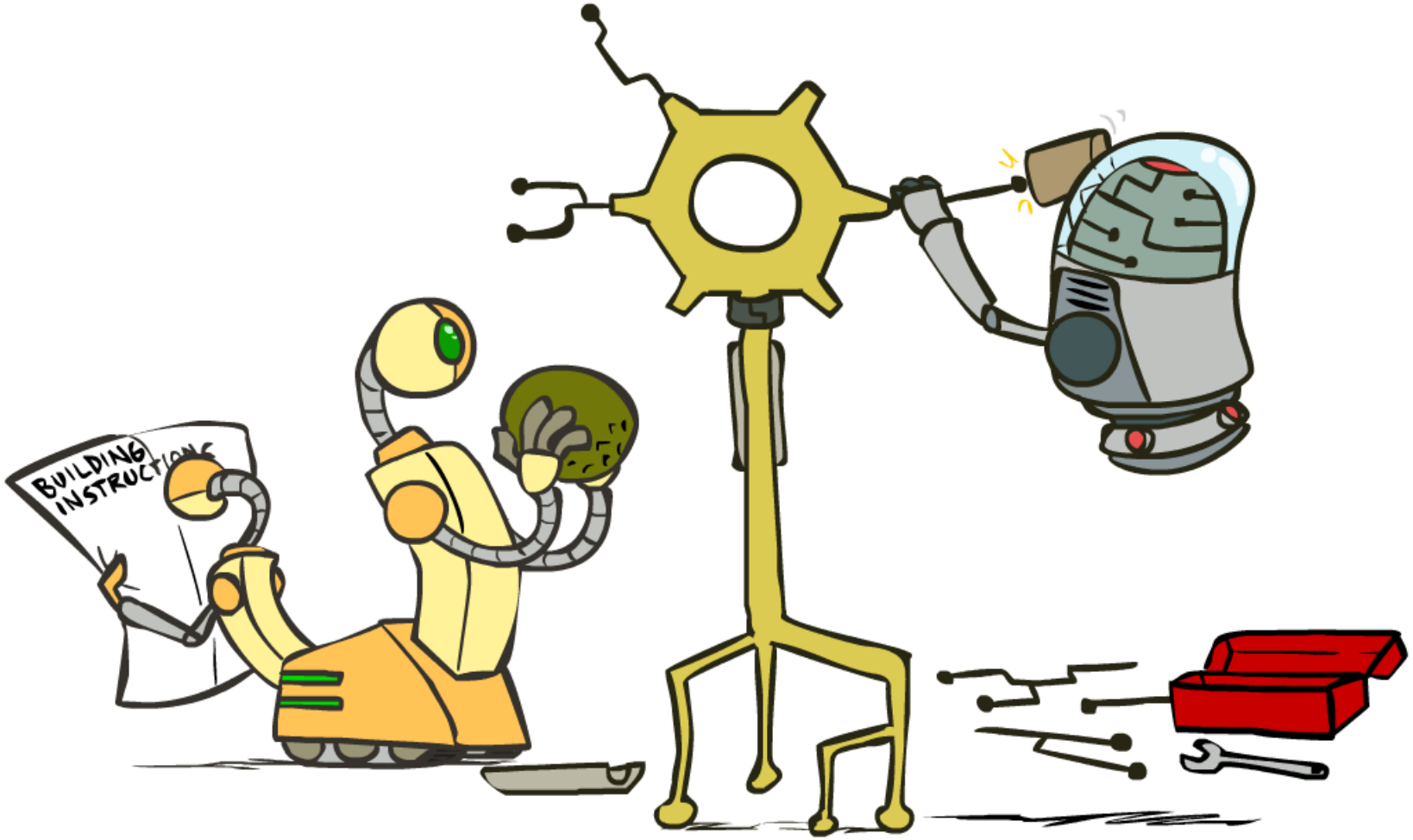


- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting





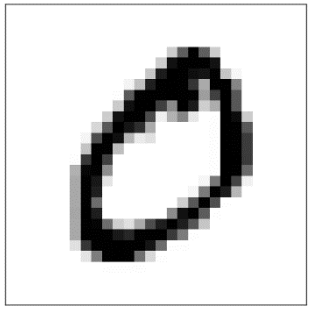
Improving the Perceptron



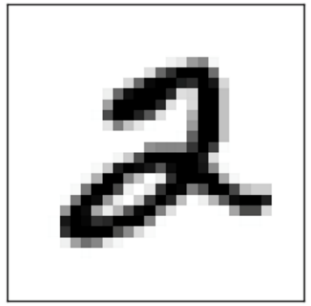
Probabilistic Classification

- Naïve Bayes provides *probabilistic* classification

Answers the query: $P(Y = y_i | x_1, \ldots, x_n)$



1: 0.001
2: 0.001
…
0: 0.991

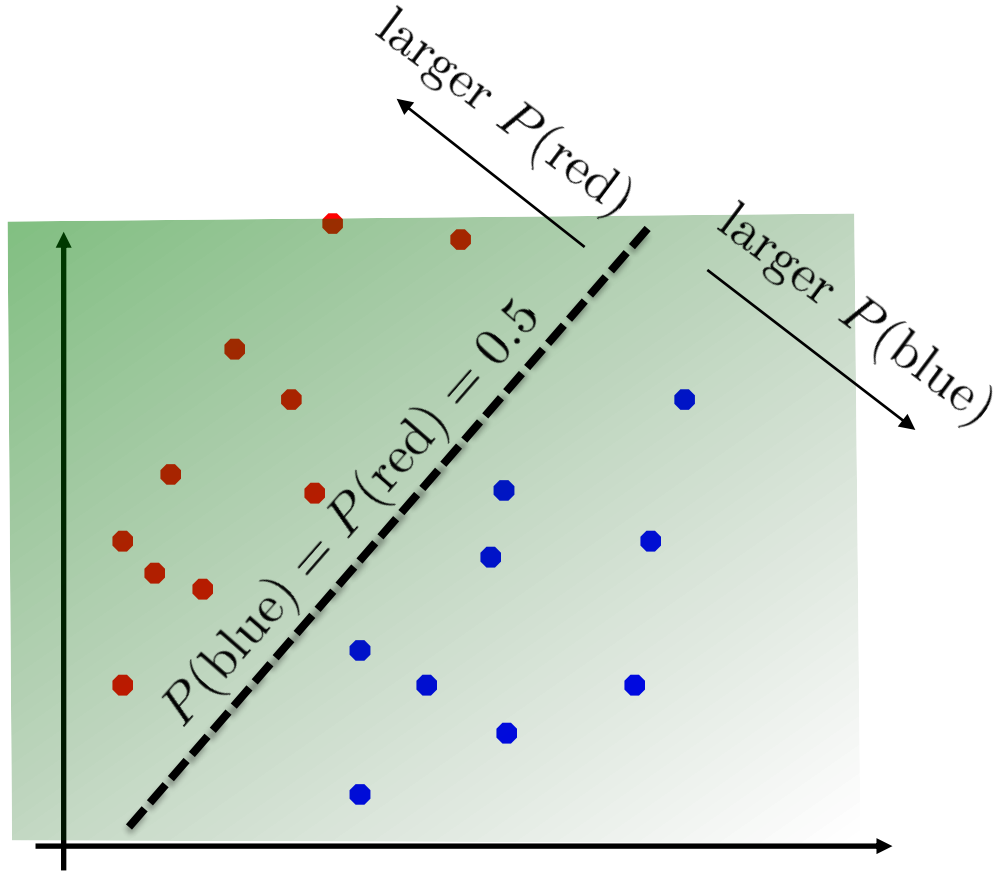


1: 0.001
2: 0.703
…
6: 0.264
…
0: 0.001

- Perceptron just gives us a class prediction
  - Can we get it to give us probabilities?
  - Turns out it also makes it easier to train!

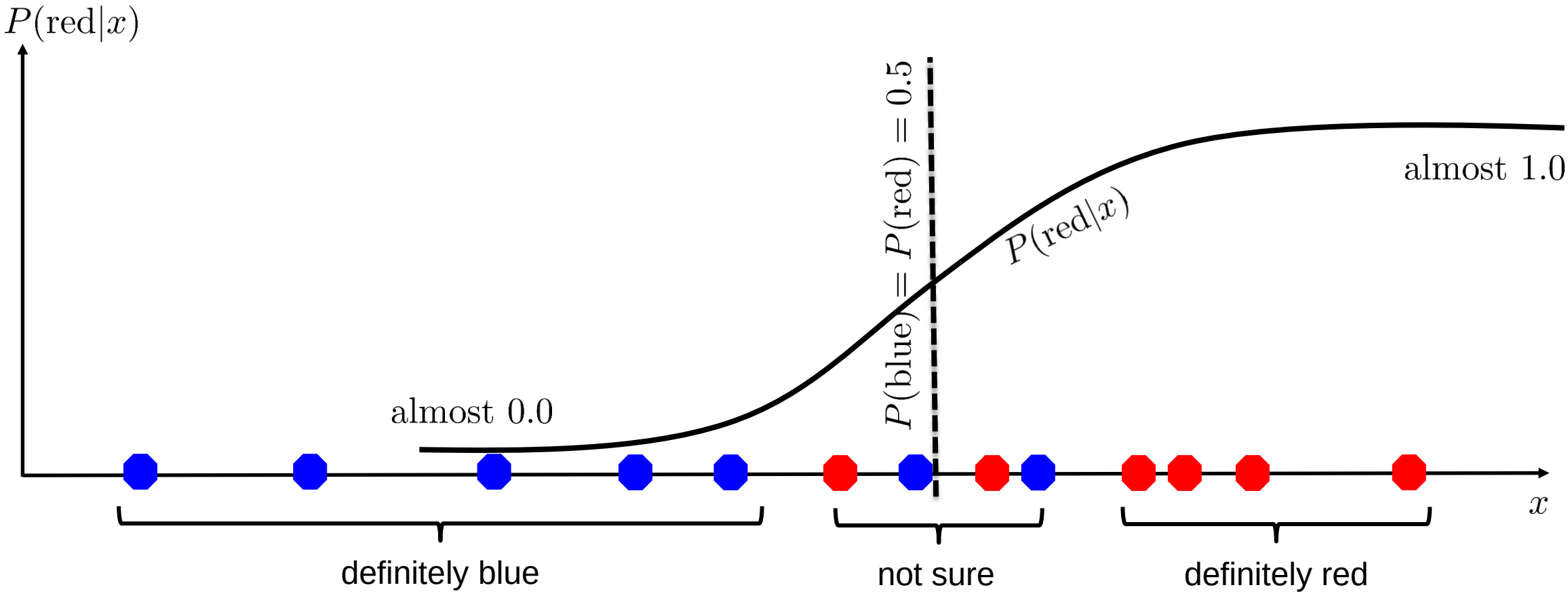
Note: I'm going to be lazy and use "x" in place of "f(x)" here – this is just for notational convenience!

A *Probabilistic* Perceptron



As $w_y \cdot x$ gets bigger, $P(y|x)$ gets bigger

A 1D Example

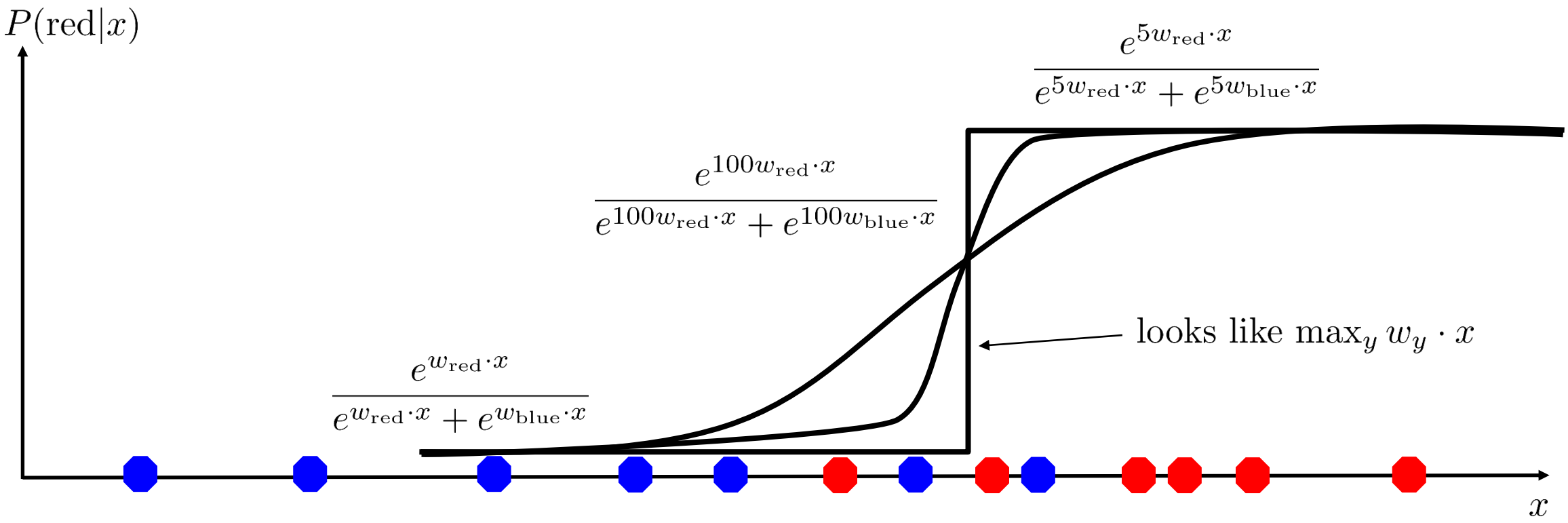


$$P(\text{red}|x) = \frac{e^{w_{\text{red}} \cdot x}}{e^{w_{\text{red}} \cdot x} + e^{w_{\text{blue}} \cdot x}}$$

probability increases exponentially as we move away from boundary

normalizer

The *Soft* Max

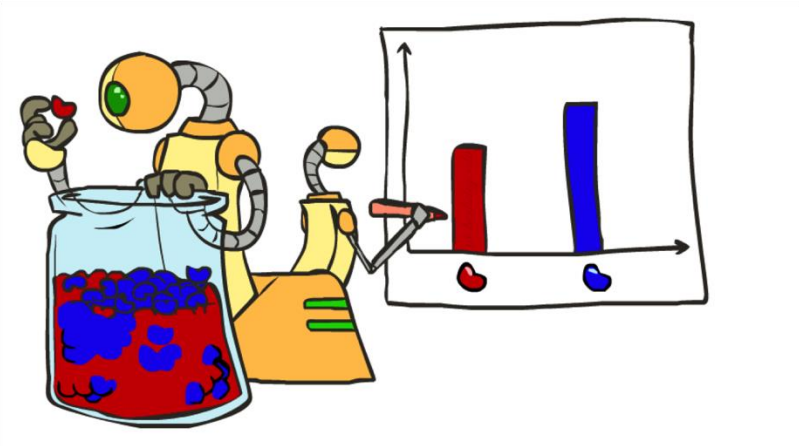


$$P(\text{red}|x) = \frac{e^{w_{\text{red}}\cdot x}}{e^{w_{\text{red}}\cdot x} + e^{w_{\text{blue}}\cdot x}}$$

How to Learn?

- Maximum likelihood estimation

$$\theta_{ML} = \arg\max_{\theta} P(\mathbf{X}|\theta)$$
$$= \arg\max_{\theta} \prod_i P_\theta(X_i)$$



- Maximum *conditional* likelihood estimation

$$\theta^\star = \arg\max_{\theta} P(\mathbf{Y}|\mathbf{X},\theta)$$
$$= \arg\max_{\theta} \prod_i \underbrace{P_\theta(y_i|x_i)}$$
$$\ell(w) = \prod_i \frac{e^{w_{y_i}\cdot x_i}}{\sum_y e^{w_y \cdot x_i}}$$

$$\ell\ell(w) = \sum_i \log P_w(y_i|x_i)$$
$$= \sum_i w_{y_i}\cdot x_i - \log \sum_y e^{w_y \cdot x_i}$$

Local Search

- Simple, general idea:
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit
  - Neighbors = small perturbations of w

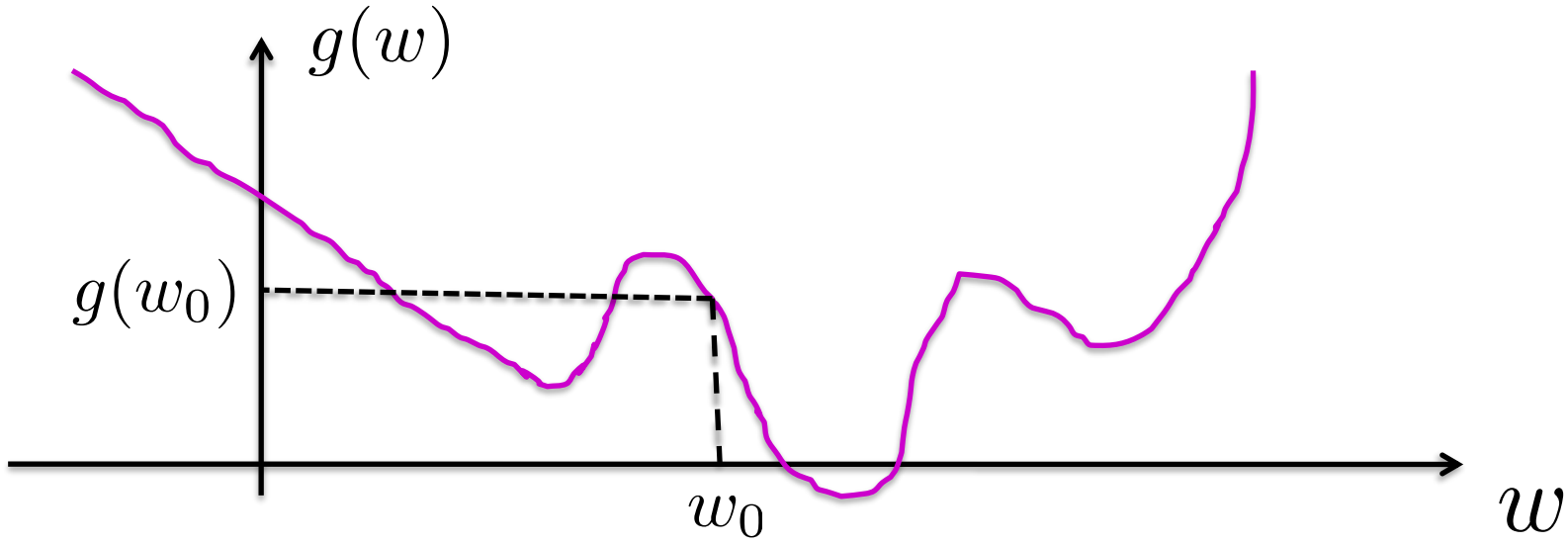
Our Status

- Our objective $ll(w)$
- Challenge: how to find a good w ?
- Equivalently:

$$\max_w ll(w)$$

$$\min_w -ll(w)$$

1D optimization

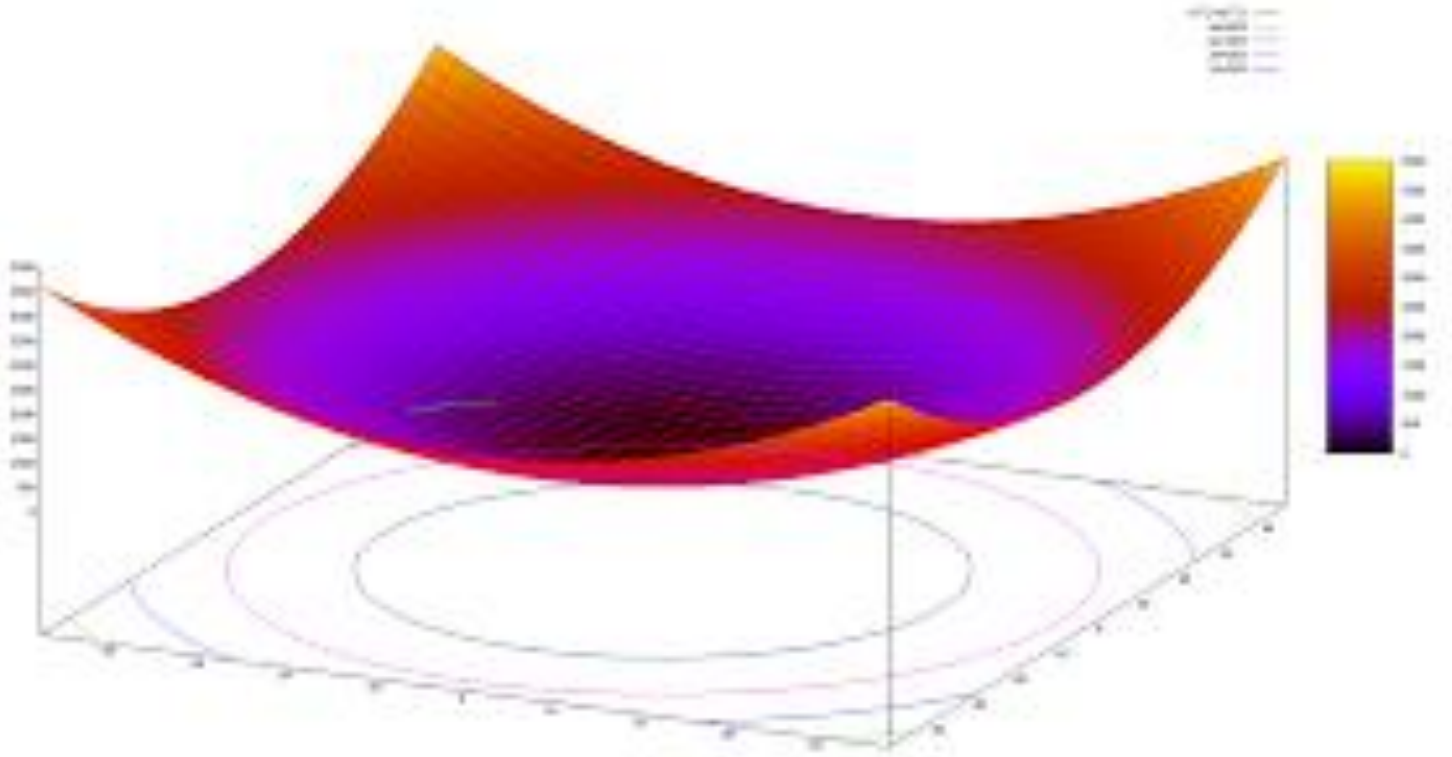


- Could evaluate $g(w_0 + h)$ and $g(w_0 - h)$
  - Then step in best direction

- Or, evaluate derivative: $\frac{\partial g(w_0)}{\partial w} = \lim_{h \to 0} \frac{g(w_0 + h) - g(w_0 - h)}{2h}$

  - Which tells which direction to step into

2-D Optimization



Source: Thomas Jungblut's Blog

Steepest Descent

- Idea:
  - Start somewhere
  - Repeat: Take a step in the steepest descent direction

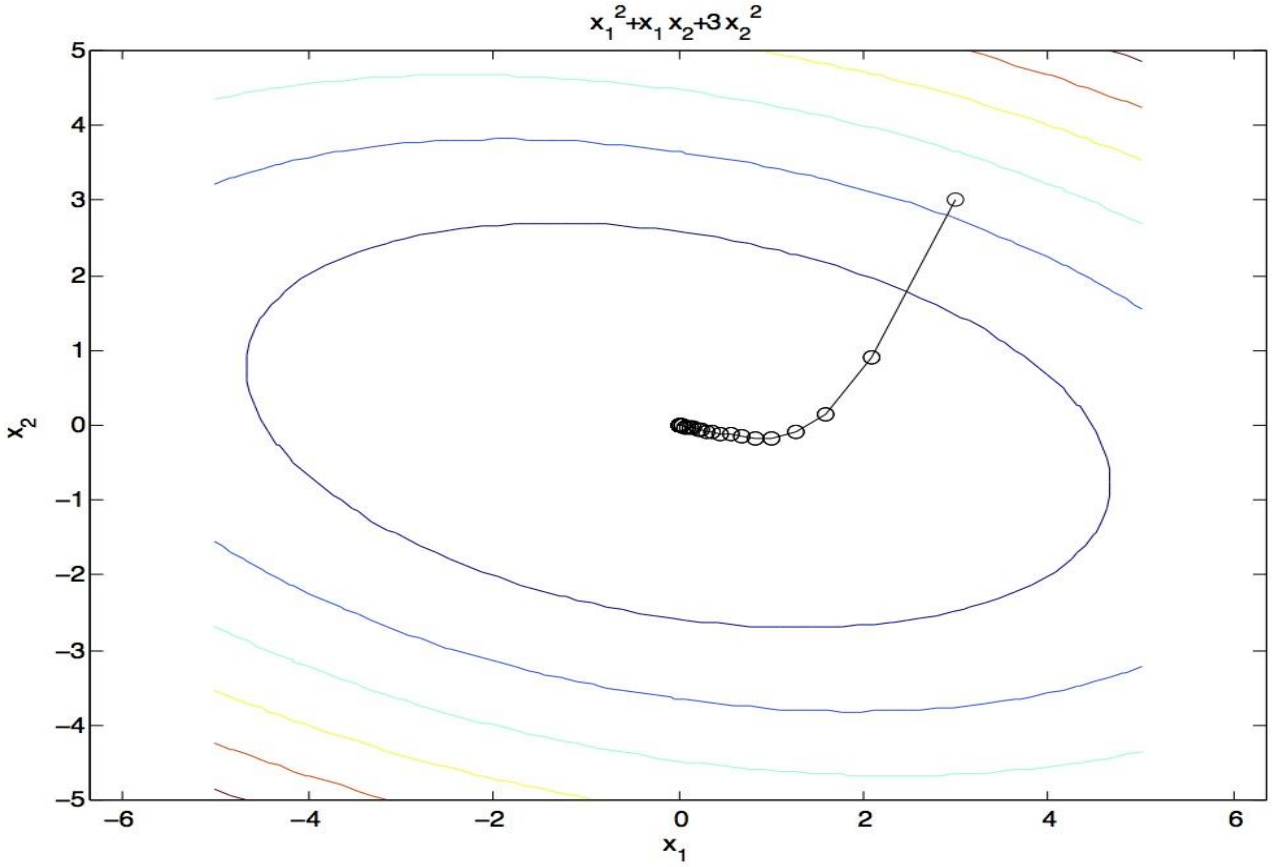


Figure source: Mathworks

Steepest Direction

- Steepest Direction = direction of the gradient

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \\ \cdots \\ \frac{\partial g}{\partial w_n} \end{bmatrix}$$

How to Learn?

$$\ell\ell(w) = \sum_i \log P_w(y_i|x_i)$$

$$= \sum_i w_{y_i} \cdot x_i - \log \sum_y e^{w_y \cdot x_i}$$

$$\frac{d}{dw_y} \log P_w(y_i|x_i) = \begin{cases} x_i - x_i \frac{e^{w_y \cdot x_i}}{\sum_{y'} e^{w_{y'} \cdot x_i}} & \text{if } y = y_i \\ -x_i \frac{e^{w_y \cdot x_i}}{\sum_{y'} e^{w_{y'} \cdot x_i}} & \text{otherwise} \end{cases}$$

$$= x_i(I(y = y_i) - P(y|x_i))$$

Optimization Procedure: Gradient Descent

initialize w (e.g., randomly)

repeat for K iterations:

  for each example (x_i, y_i):

    compute gradient $\Delta_i = -\nabla_w \log P_w(y_i|x_i)$

  compute gradient $\nabla_w \mathcal{L} = \sum_i \Delta_i$

  $w \leftarrow w - \alpha \nabla_w \mathcal{L}$

$$\frac{d}{dw_y} \log P_w(y_i|x_i) = x_i(I(y = y_i) - P(y|x_i))$$

- α: learning rate --- tweaking parameter that needs to be chosen carefully
- How? Try multiple choices
  - Crude rule of thumb: update should change w by about 0.1 – 1 %

Stochastic Gradient Descent

initialize w (e.g., randomly)

repeat for K iterations:

  for each example (x_i, y_i):

    compute gradient $\Delta_i = -\nabla_w \log P_w(y_i|x_i)$

    $w \leftarrow w - \alpha\Delta_i$

$$\frac{d}{dw_y} \log P_w(y_i|x_i) = x_i(I(y = y_i) - P(y|x_i))$$

if $y_i = y$, move w_y toward x_i with weight $1 - P(y_i|x_i)$

probability of *incorrect* answer

if $y_i \neq y$, move w_y away from x_i with weight $P(y|x_i)$

probability of *incorrect* answer

compare this to the multiclass perceptron: probabilistic weighting!

Logistic Regression Demo!

https://playground.tensorflow.org/