CSCI4622 Machine Learning

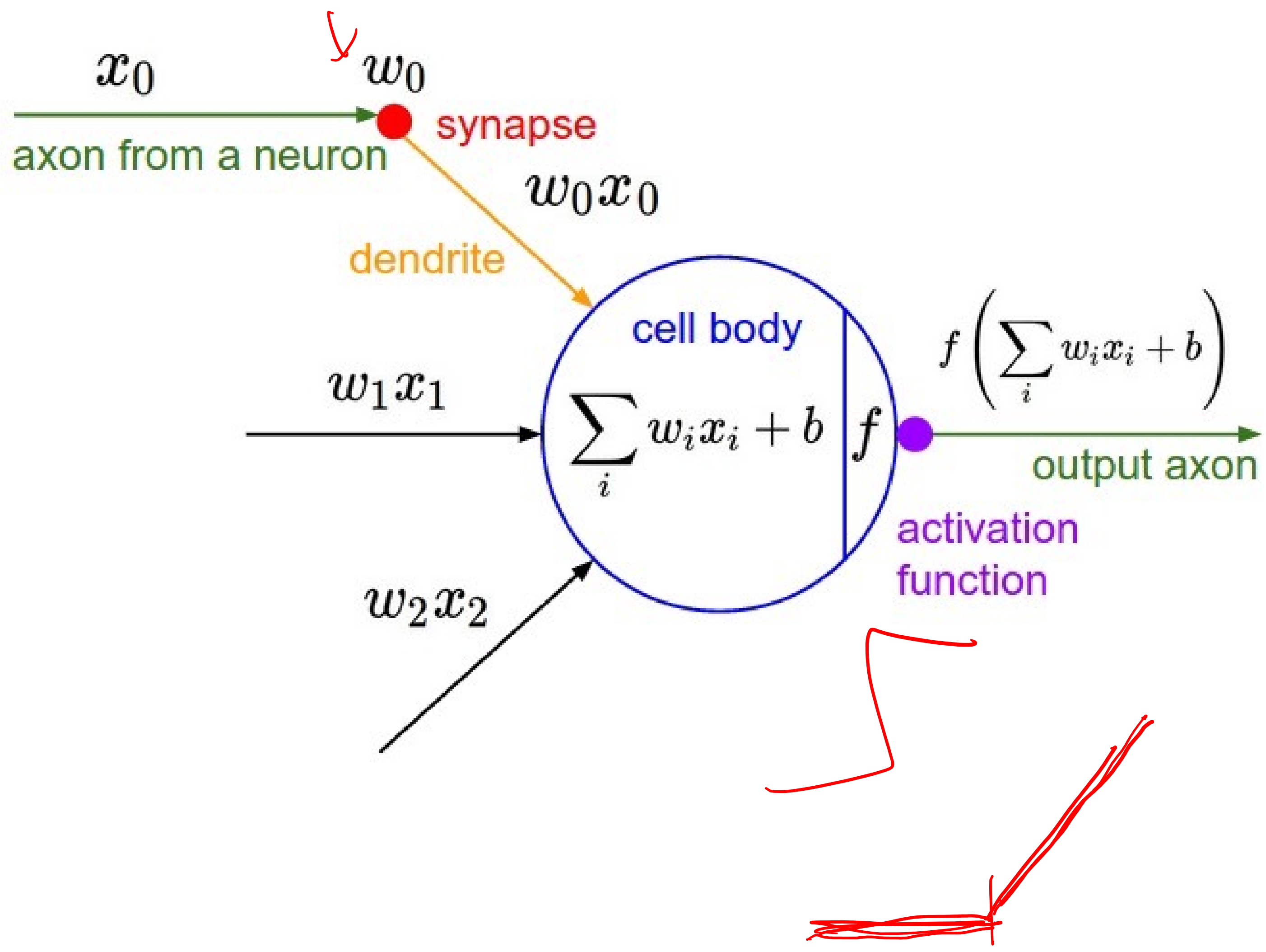
2020 Spring

Neural Networks (2)

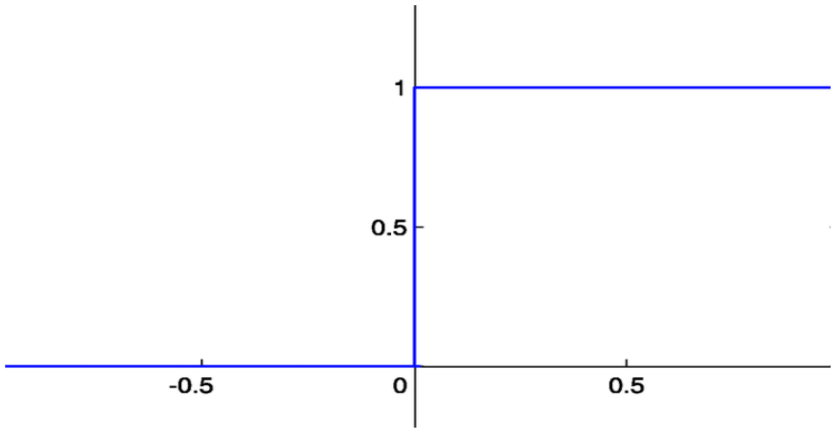
Geena Kim

*Some of the slide/diagram adopted from CMU deep learning course

Perceptron

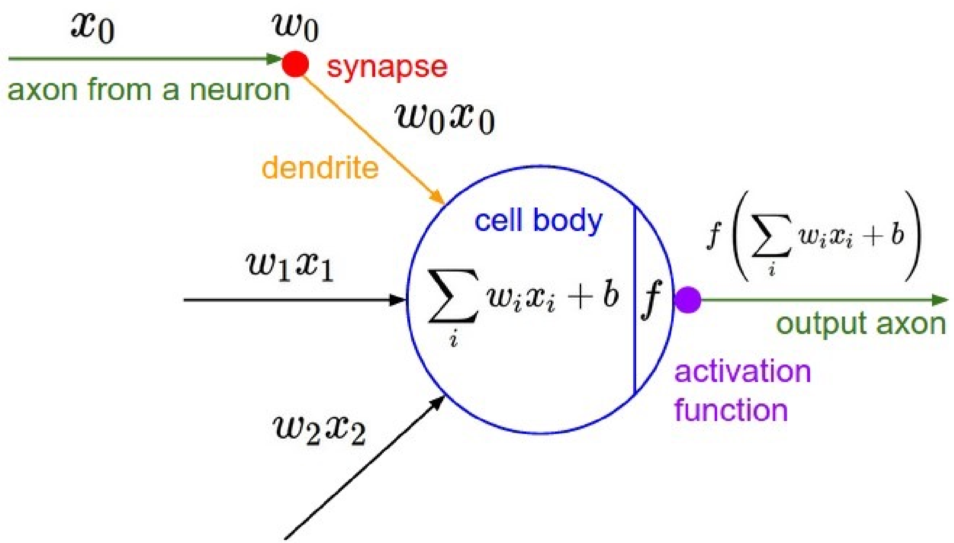


- Binary Threshold (Step function)

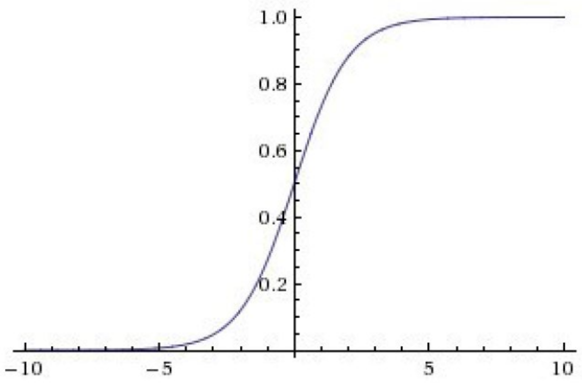


$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

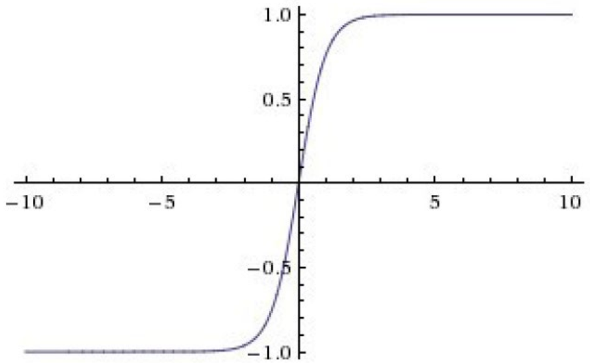
Activation functions



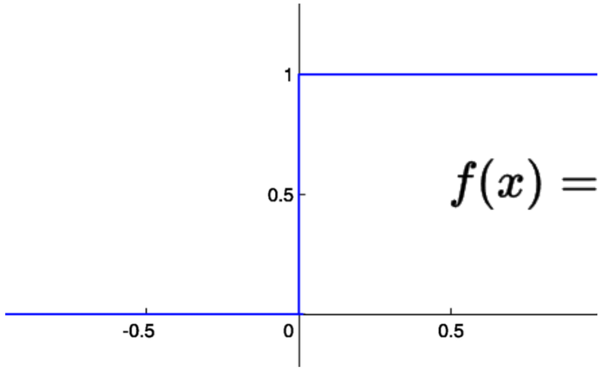
- Sigmoid



- Tanh



- Binary Threshold (Step function)



$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

Perceptron

What can a perceptron do?

- Boolean tasks
- Update the weights whenever the perceptron output is wrong
- Proved convergence for linearly separable classes

Perceptron

Learning in perceptron

- Perceptron rule
- Delta rule (Gradient Descent)

Perceptron

Perceptron rule

$$\omega_j \leftarrow \omega_j - \overset{1}{\alpha}(\hat{y}_i - y_i)X_{ij}$$

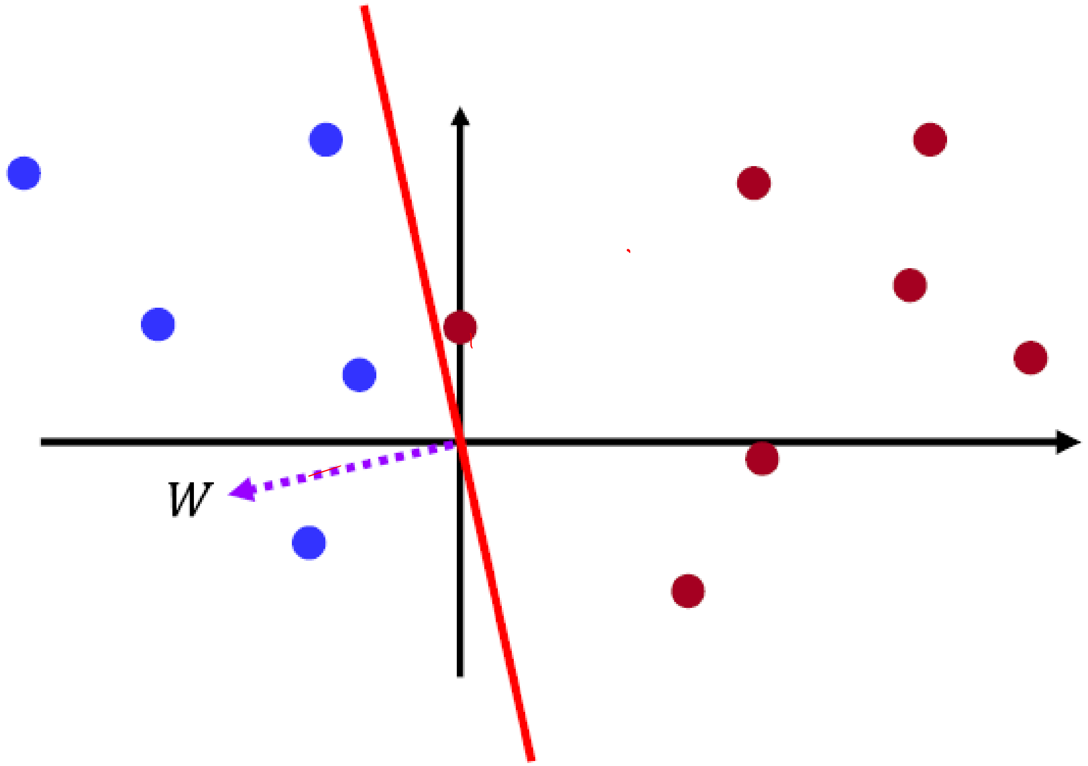
feature column

sample

-2

$+2$

Key: Red -1, Blue = +1



Perceptron

Perceptron algorithm

- Cycle through the training instances
- Only update W on misclassified instances
- If instance misclassified:
  - If instance is positive class (positive misclassified as negative)

$$W = W + X_i$$

  - If instance is negative class (negative misclassified as positive)

$$W = W - X_i$$

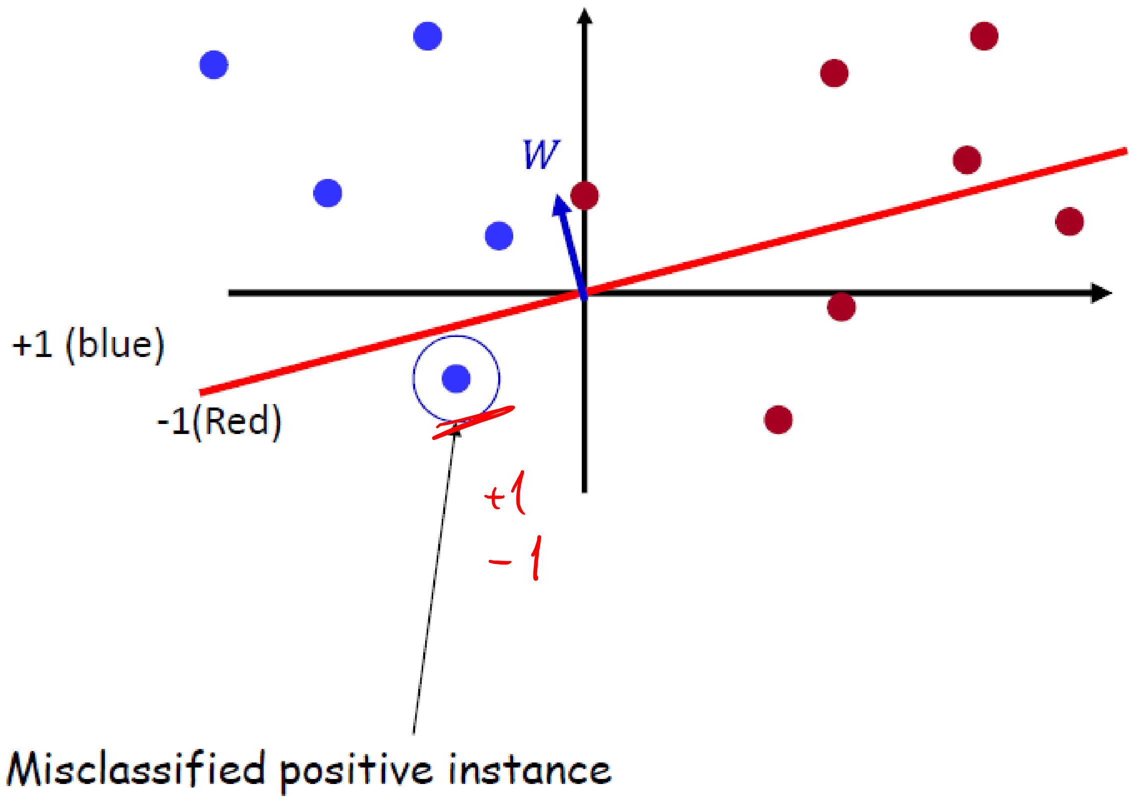
$$w_j' \leftarrow w_j - 1(\hat{y}_i - y_i)X_{ij}$$

$$\hat{y}_i = 1 \qquad w_j =$$
$$y_i = -1 \qquad w_j - 2$$

$$\hat{y}_i = -1 \qquad w_j$$
$$y_i = +1 \qquad w_j + 2$$

Perceptron

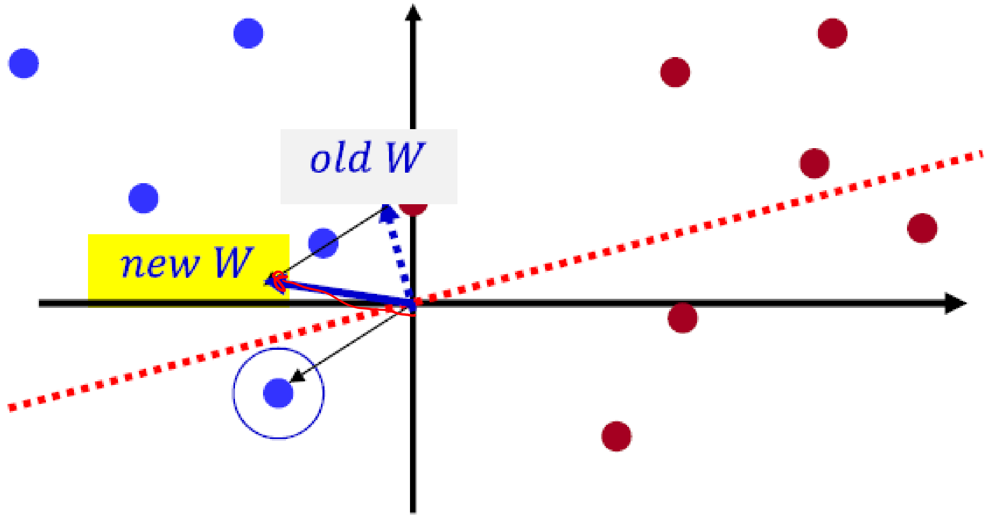
Perceptron algorithm



Perceptron

Perceptron algorithm

ΔW

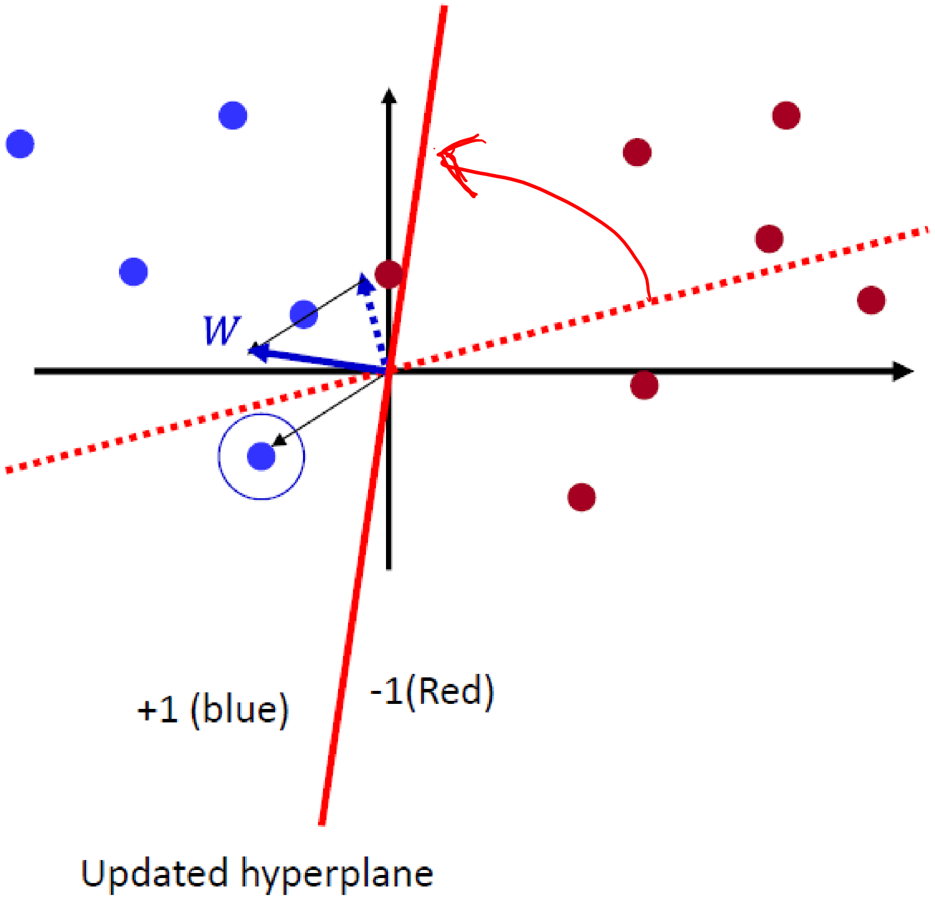


Updated weight vector

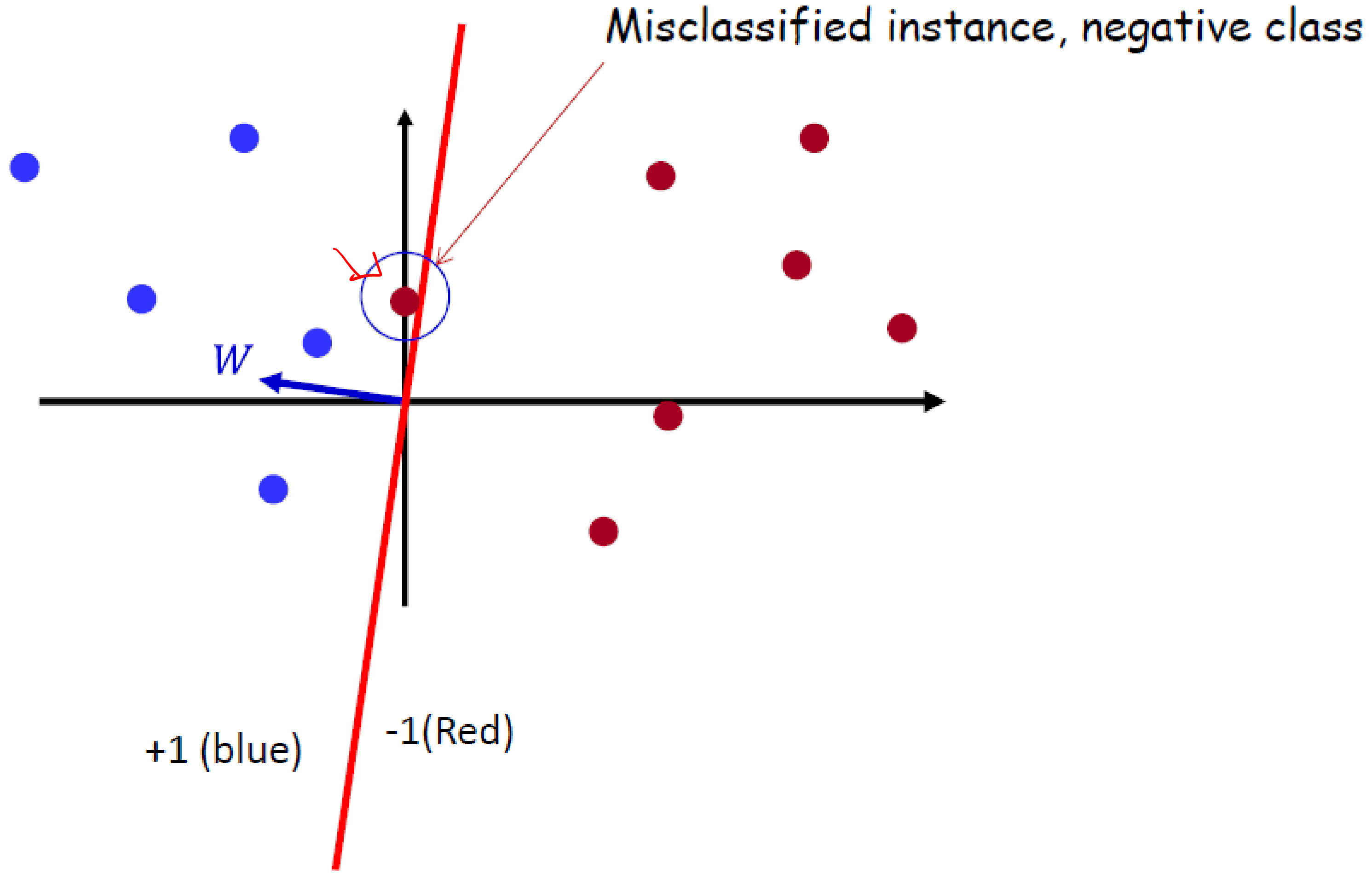
Misclassified *positive* instance, *add* it to W

Perceptron

Perceptron algorithm

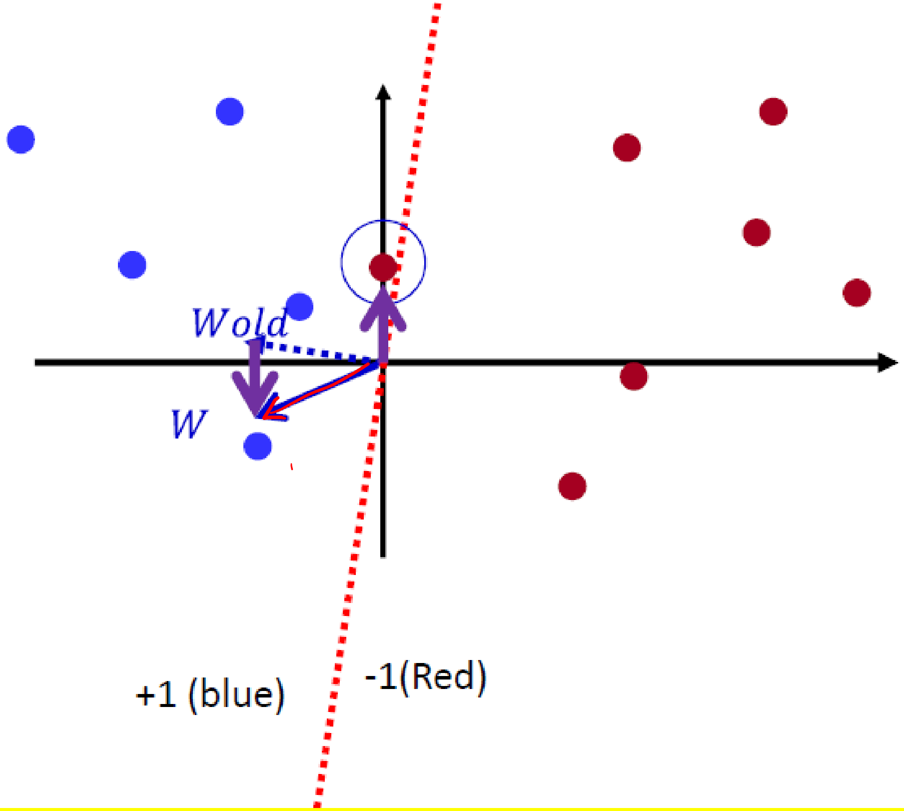


Perceptron algorithm



Perceptron

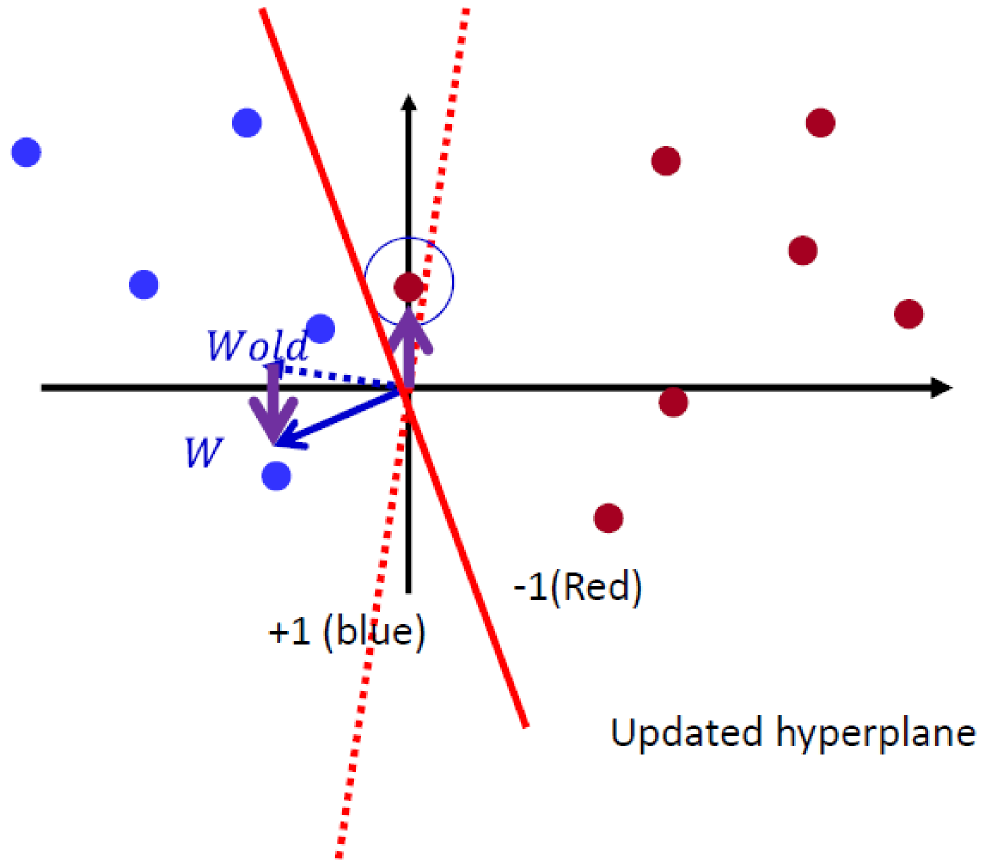
Perceptron algorithm



Misclassified *negative* instance, *subtract* it from W

Perceptron

Perceptron algorithm



Perceptron

Delta rule (Gradient Descent)

$$\omega_j \leftarrow \omega_j - \alpha \frac{\partial \mathcal{L}}{\partial \omega_j}$$

$\mathcal{L}(\omega_j, x_{ji})$

$MSE = \sum_i \frac{1}{2} (y_i - \hat{y}_i)^2$

$\frac{\partial}{\partial w} (y - (wx+b))^2$

$(y - \underbrace{(wx+b)}_{\hat{y}}) \cdot (-x)$

$$\mathcal{L} = \frac{1}{2}(\hat{y}_i - y_i)^2$$

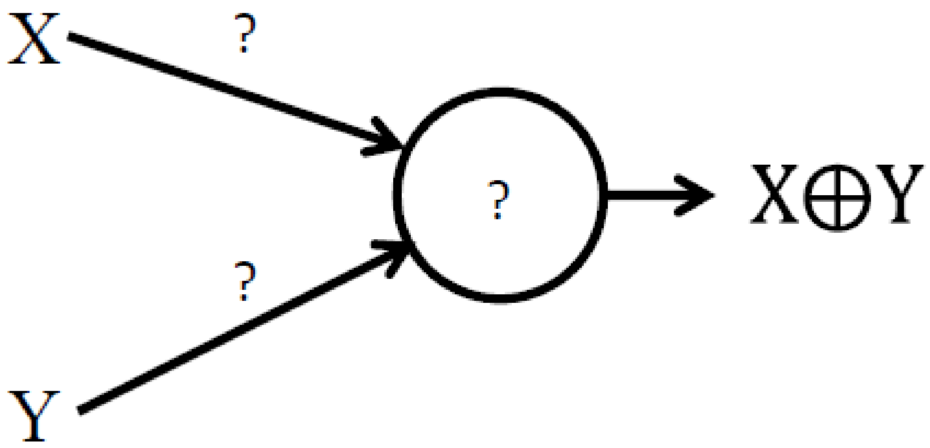
$-\alpha(\hat{y} - y)\, x$

$$\hat{y}_i = \sum_j \omega_j X_{ij}$$

$$\omega_j \leftarrow \omega_j - \alpha(\hat{y}_i - y_i) X_{ij}$$

Perceptron

No solution for XOR!
Not universal!

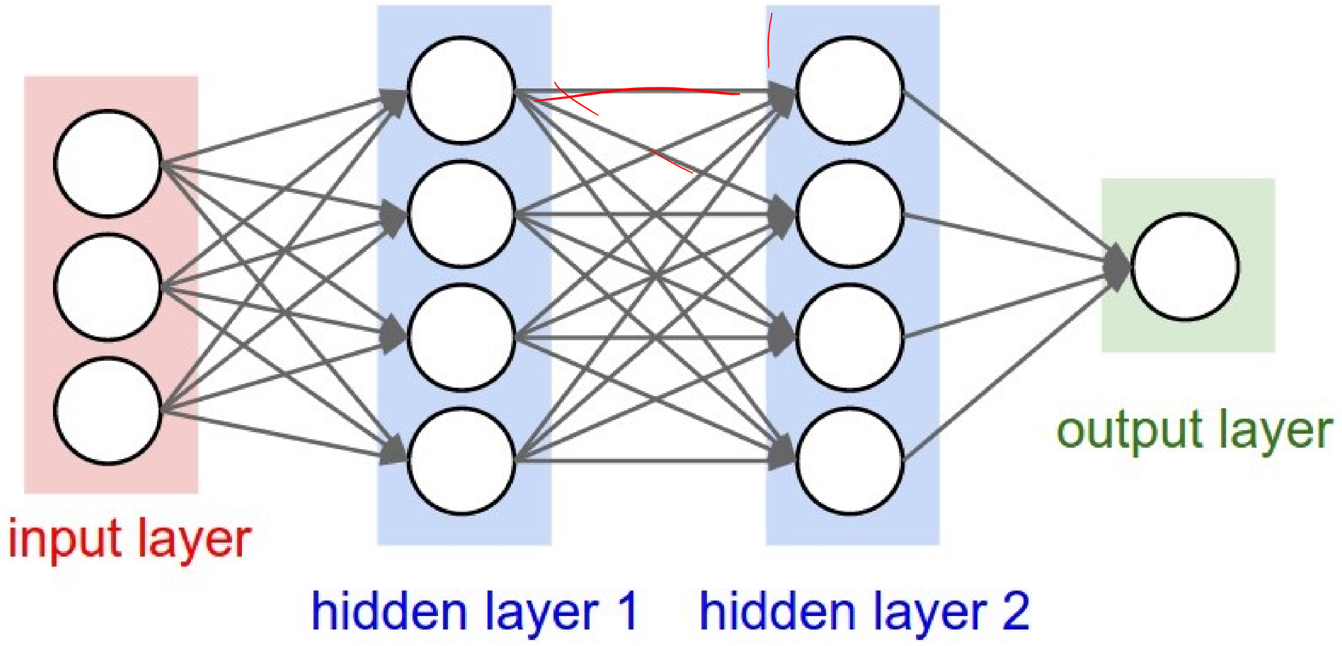


- Minsky and Papert, 1968

MLP

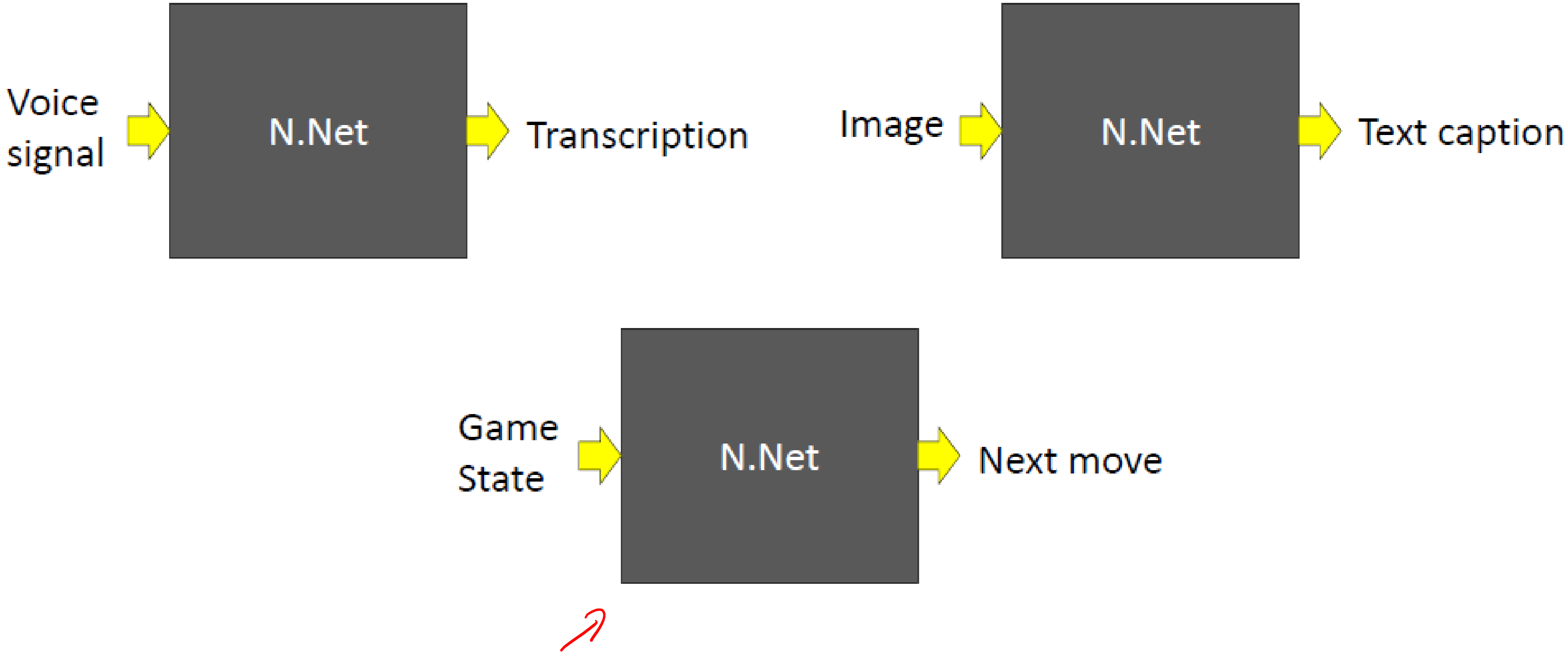
How do we handle linearly inseparable cases?

Dense layer, Fully connected layer

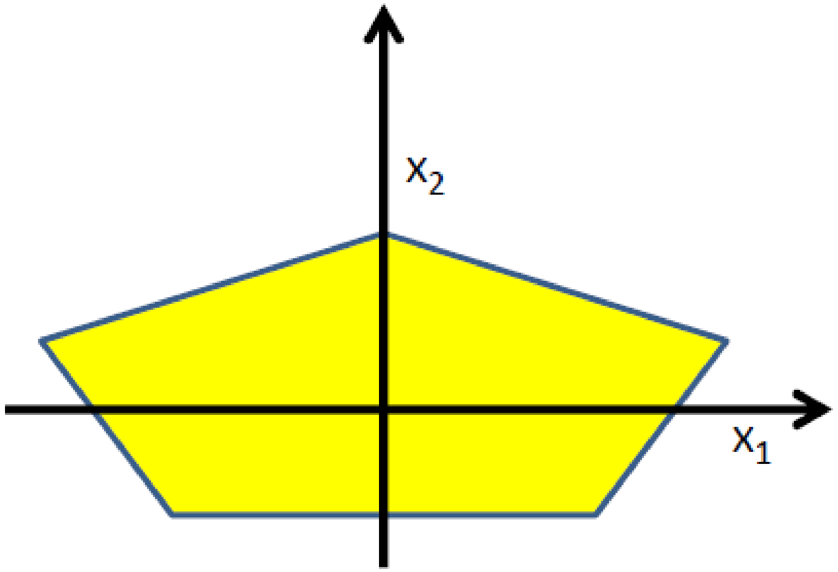


MLP

Neural networks are universal function approximator

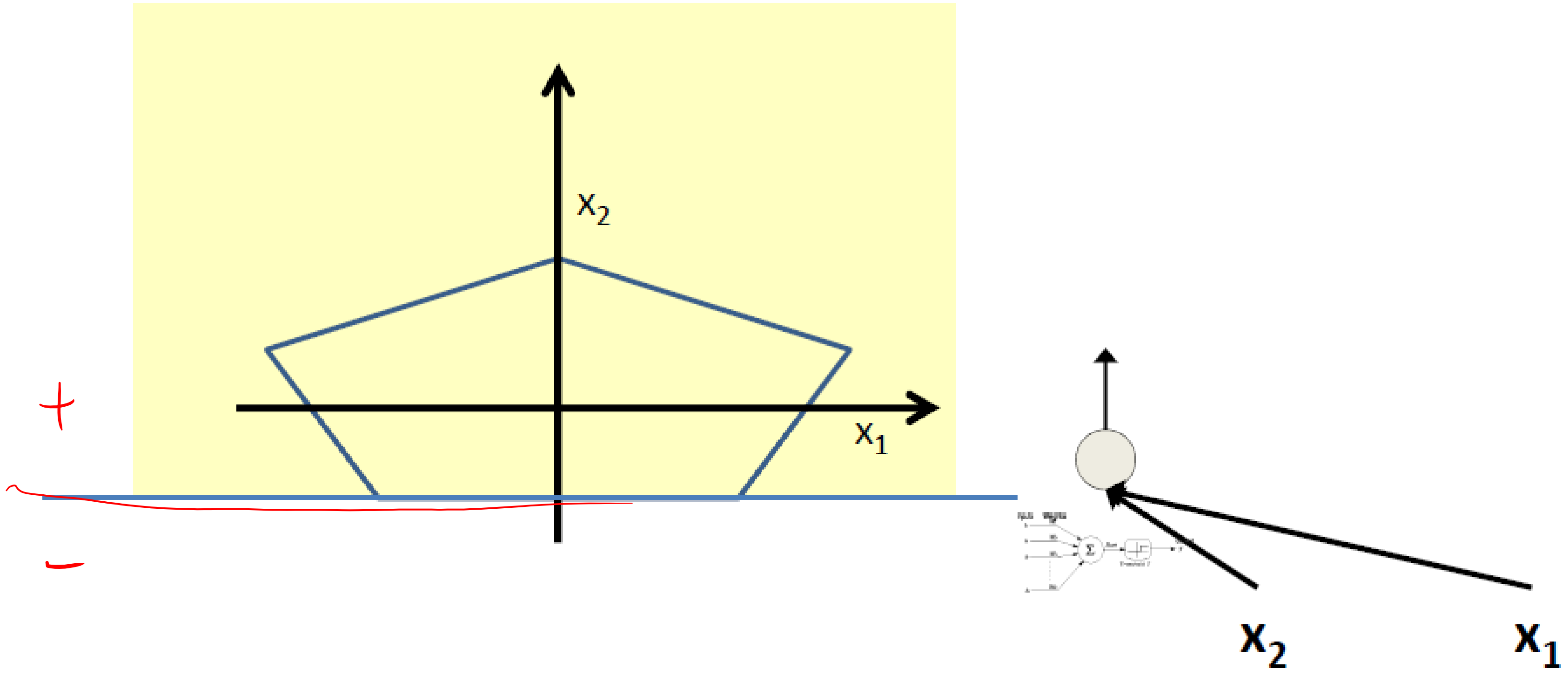


We can make an arbitrary shape decision boundary



Can now be composed into "networks" to compute arbitrary classification "boundaries"

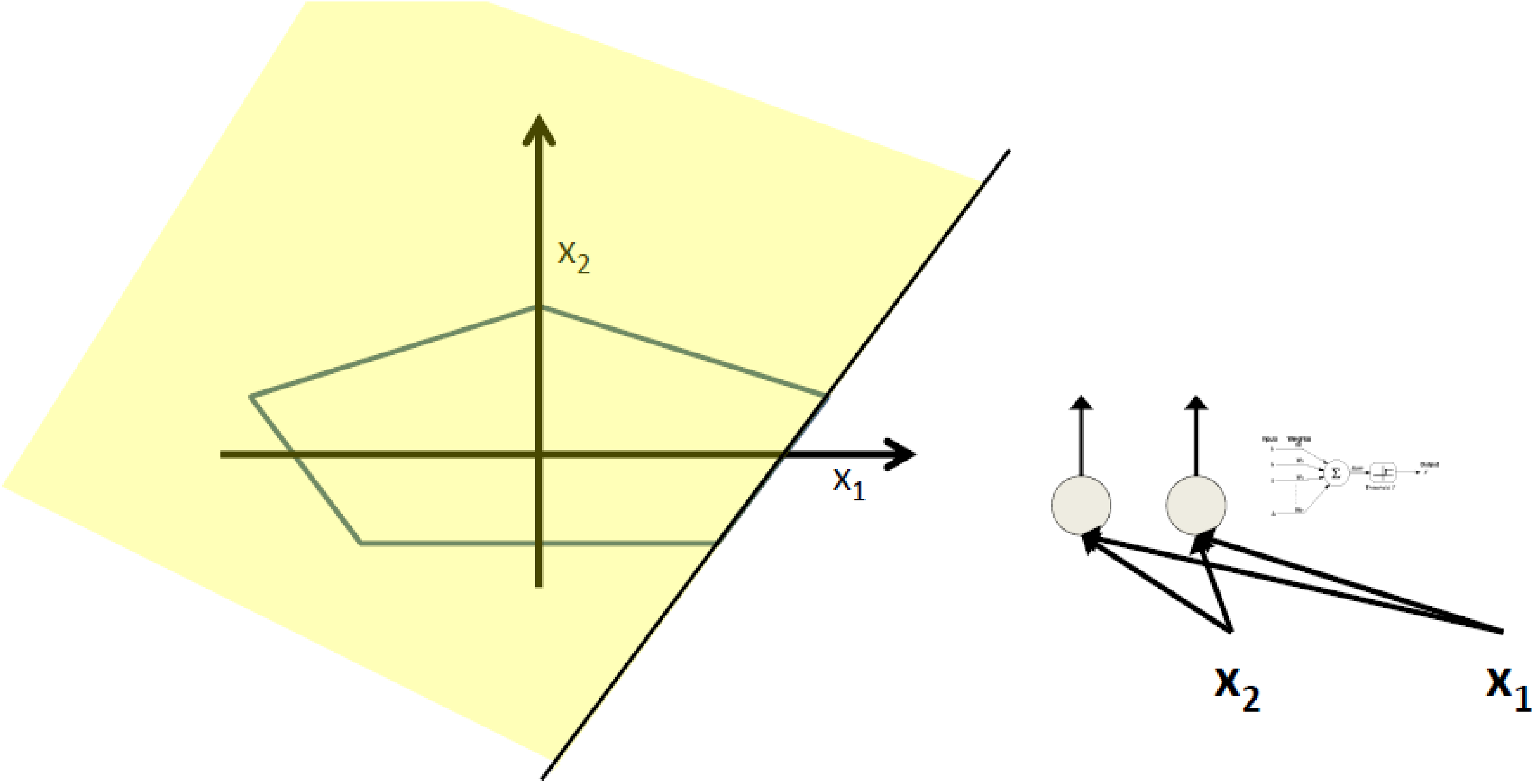
Boolean over real numbers



The network must fire if the input is in the coloured area

MLP

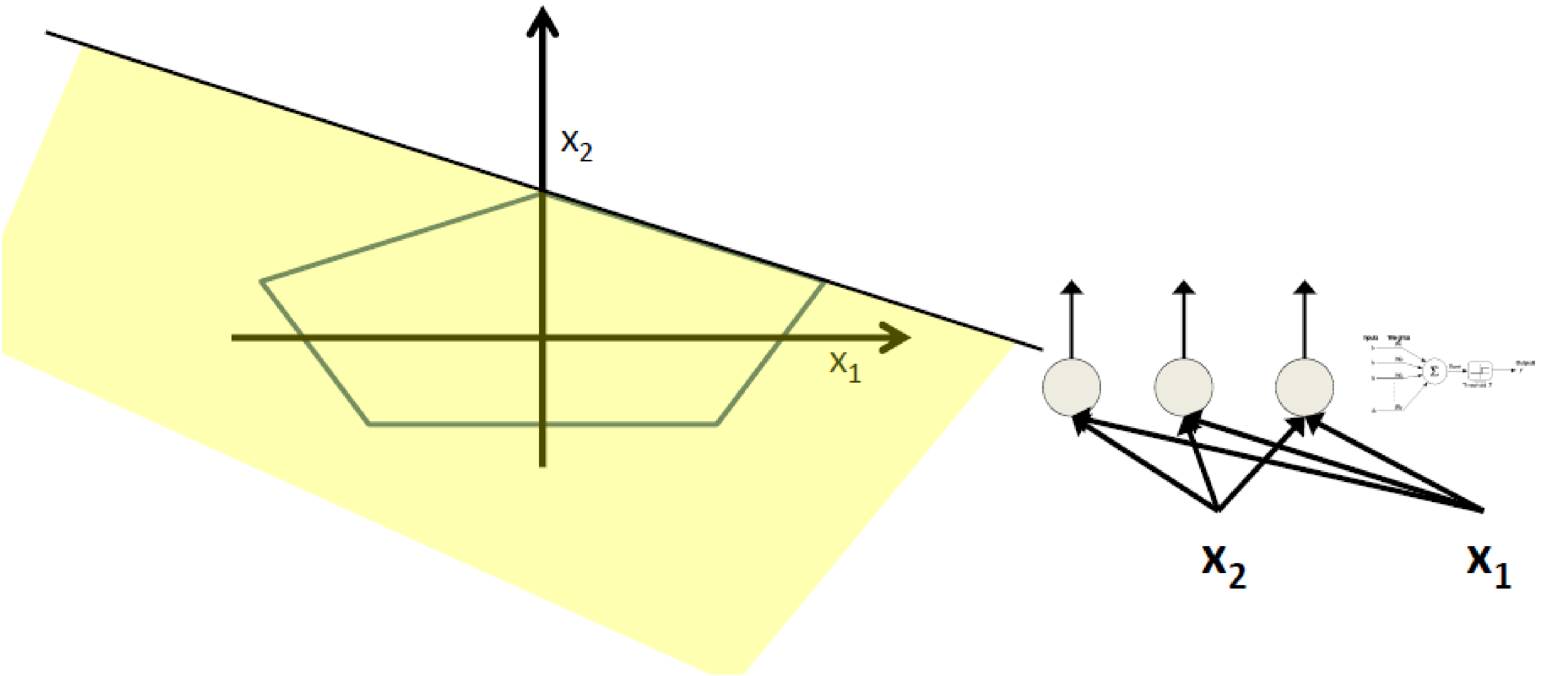
Boolean over real numbers



The network must fire if the input is in the coloured area

MLP

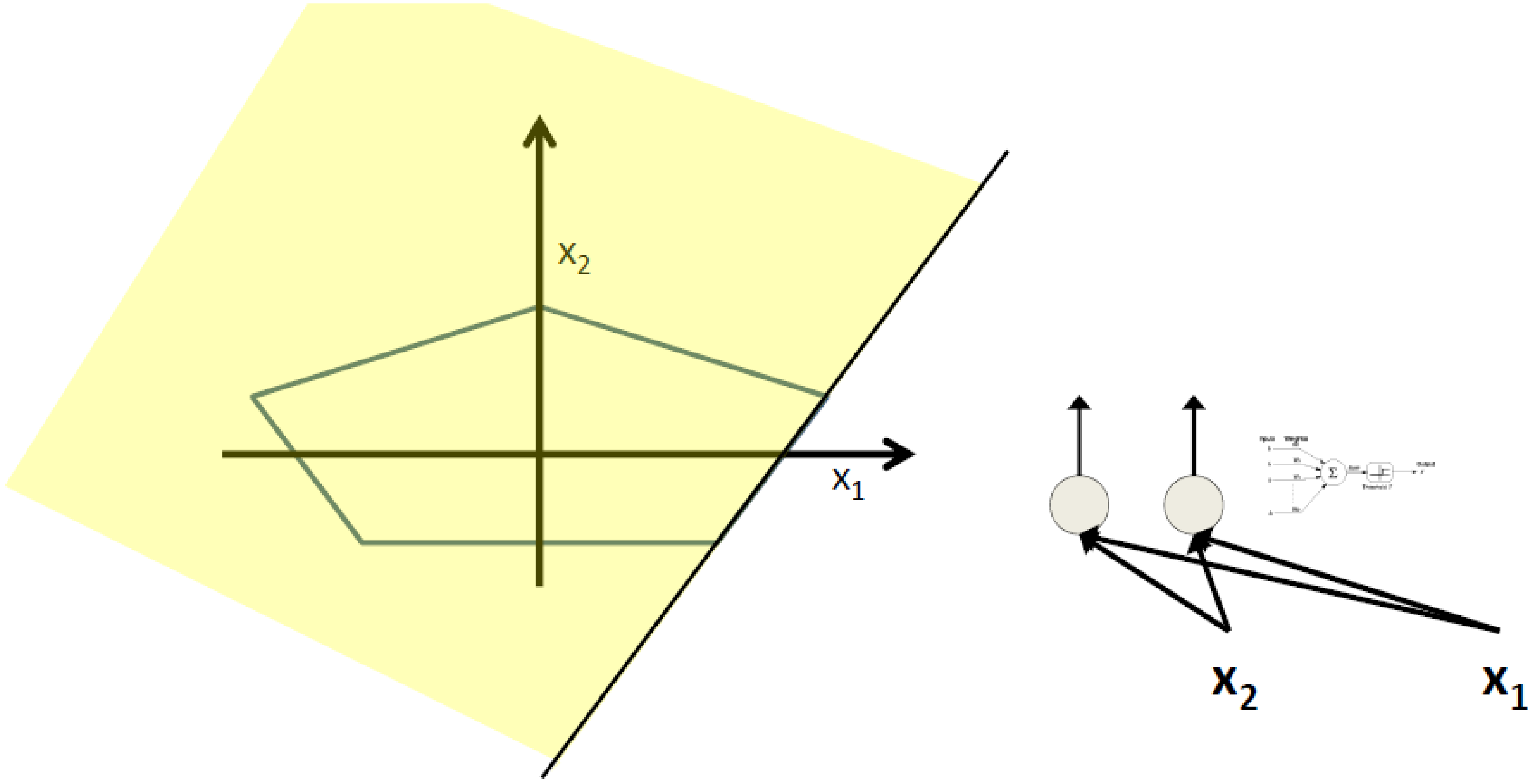
Boolean over real numbers



The network must fire if the input is in the coloured area

MLP

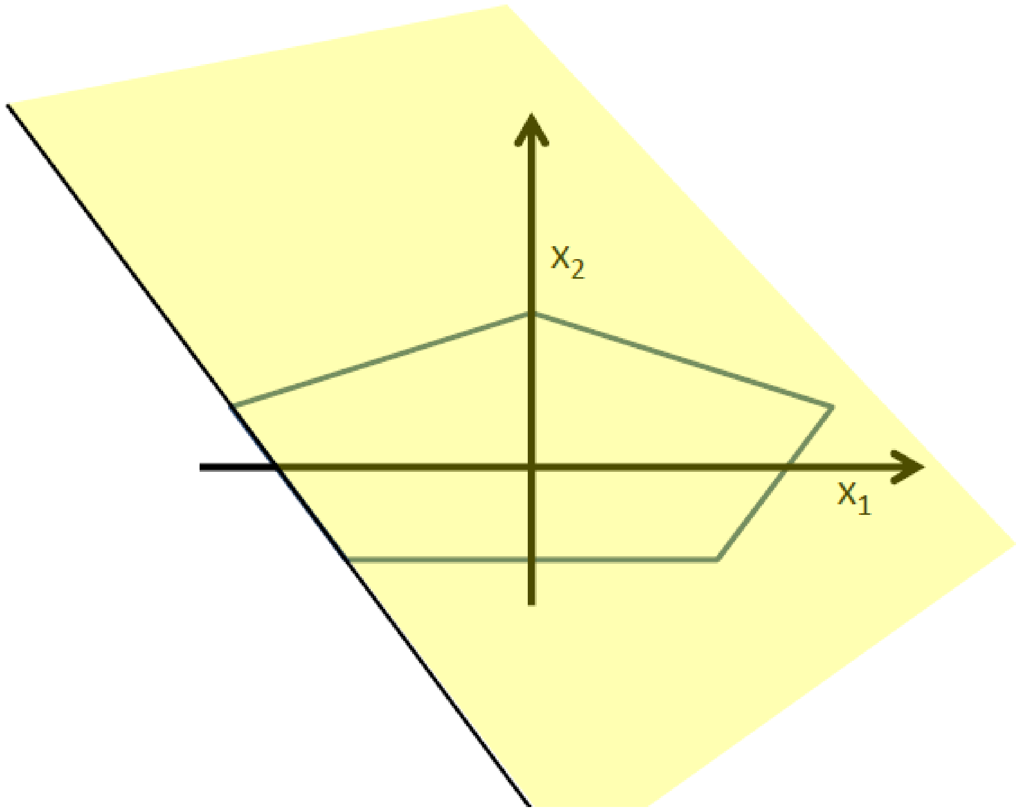
Boolean over real numbers

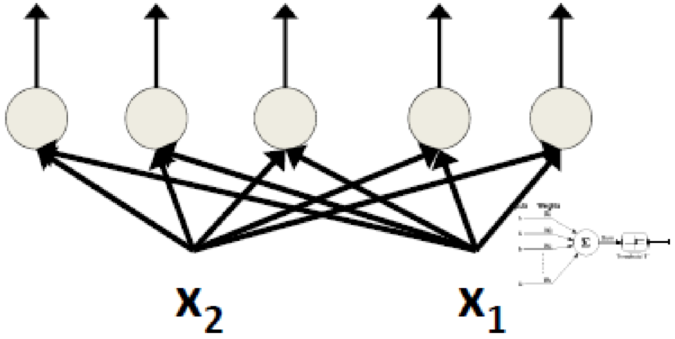


The network must fire if the input is in the coloured area

MLP

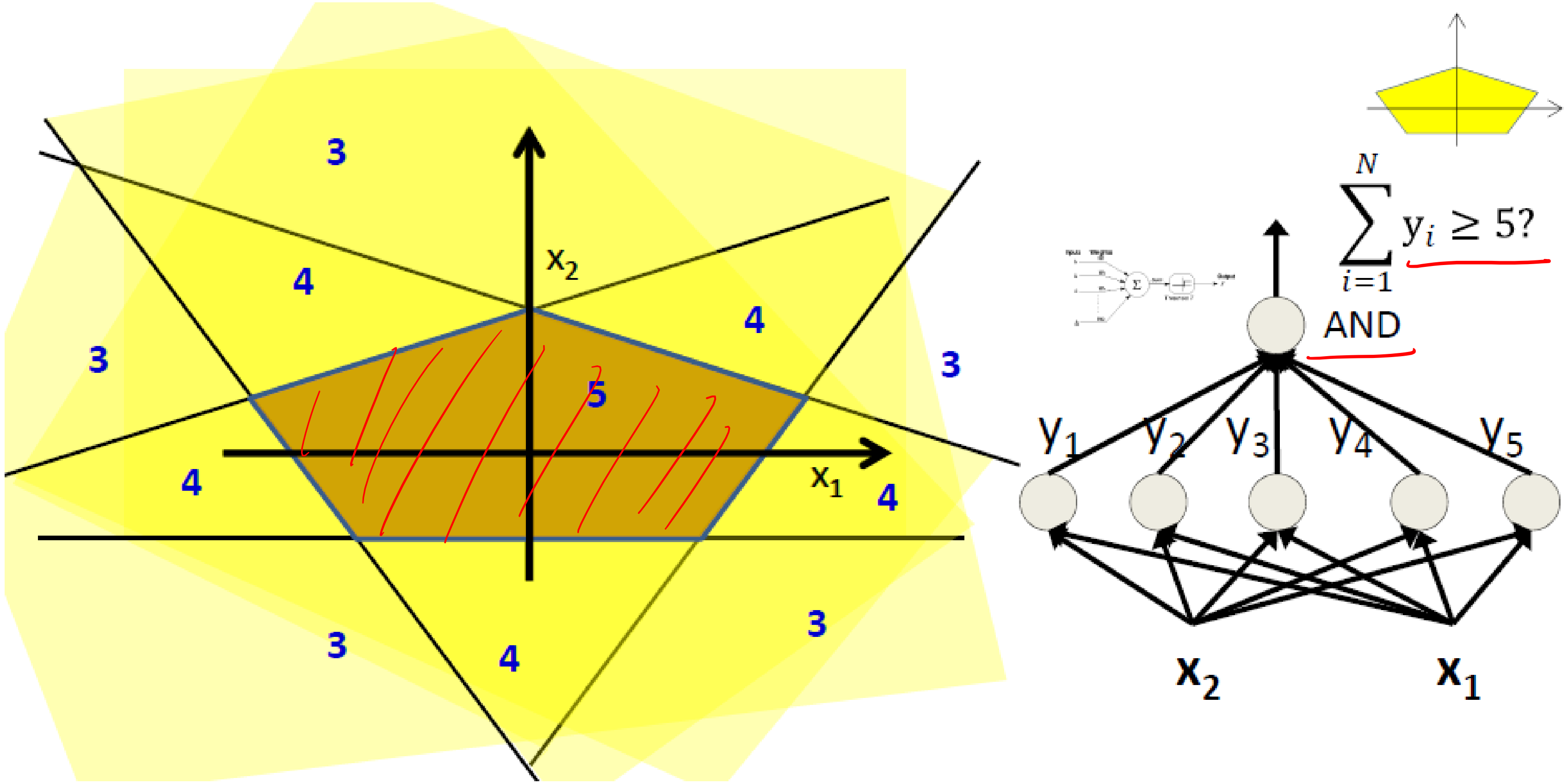
Boolean over real numbers





The network must fire if the input is in the coloured area

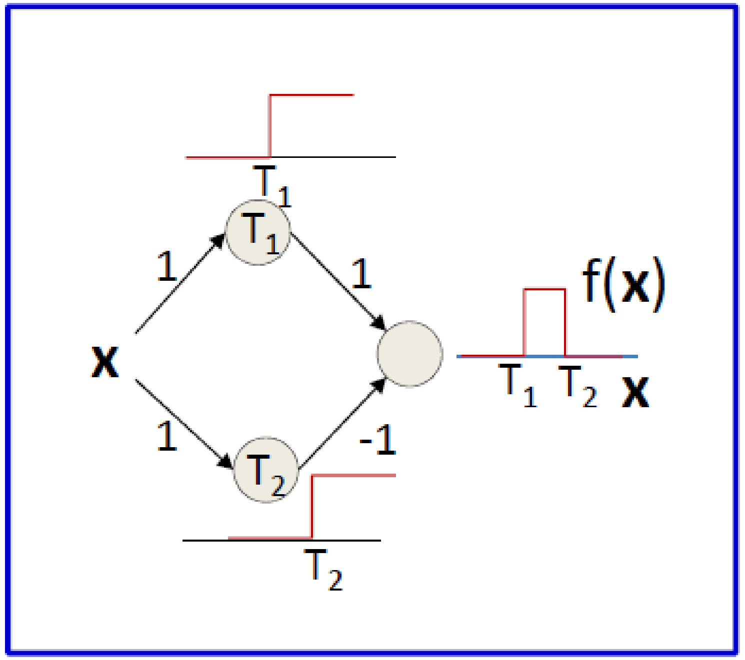
Boolean over real numbers

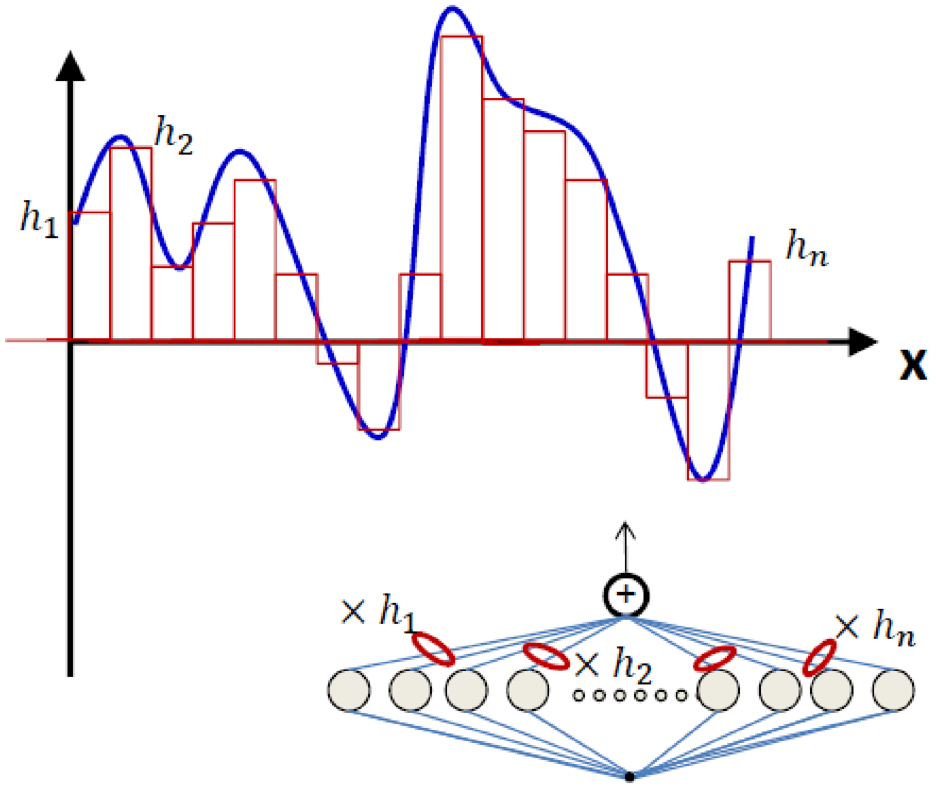


The network must fire if the input is in the coloured area

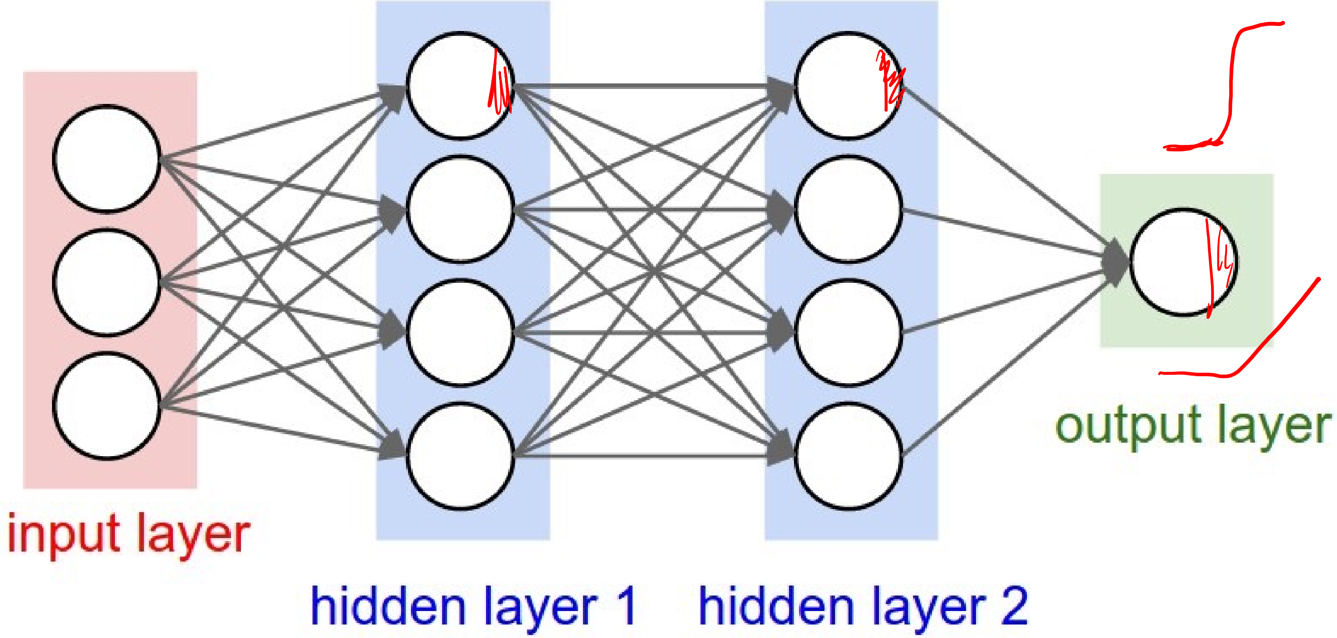
MLP

Neural networks can be used for regression tasks





Multi-layer Perceptron (Neural network)



Design Parameters

- Architecture
- Number of layers
- Number of neurons in a layer
- Activation functions

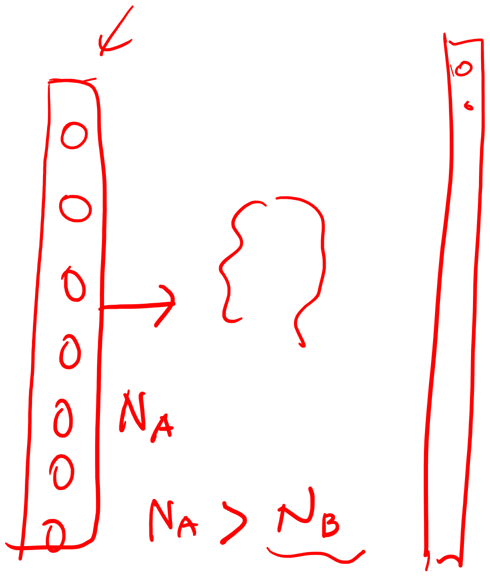
Effect of depth in network architecture

Theorem 5

A certain class of functions $\mathcal{F}$ of n inputs can be represented using a deep network with $\mathcal{O}(n)$ units, whereas it would require $\mathcal{O}(2^{\sqrt{n}})$ units for a shallow network.

Theorem 6

For a certain class of functions $\mathcal{G}$ of n inputs, the deep sum-product network with depth k can be represented with $\mathcal{O}(nk)$ units, whereas it would require $\mathcal{O}((n-1)^k)$ units for a shallow network.

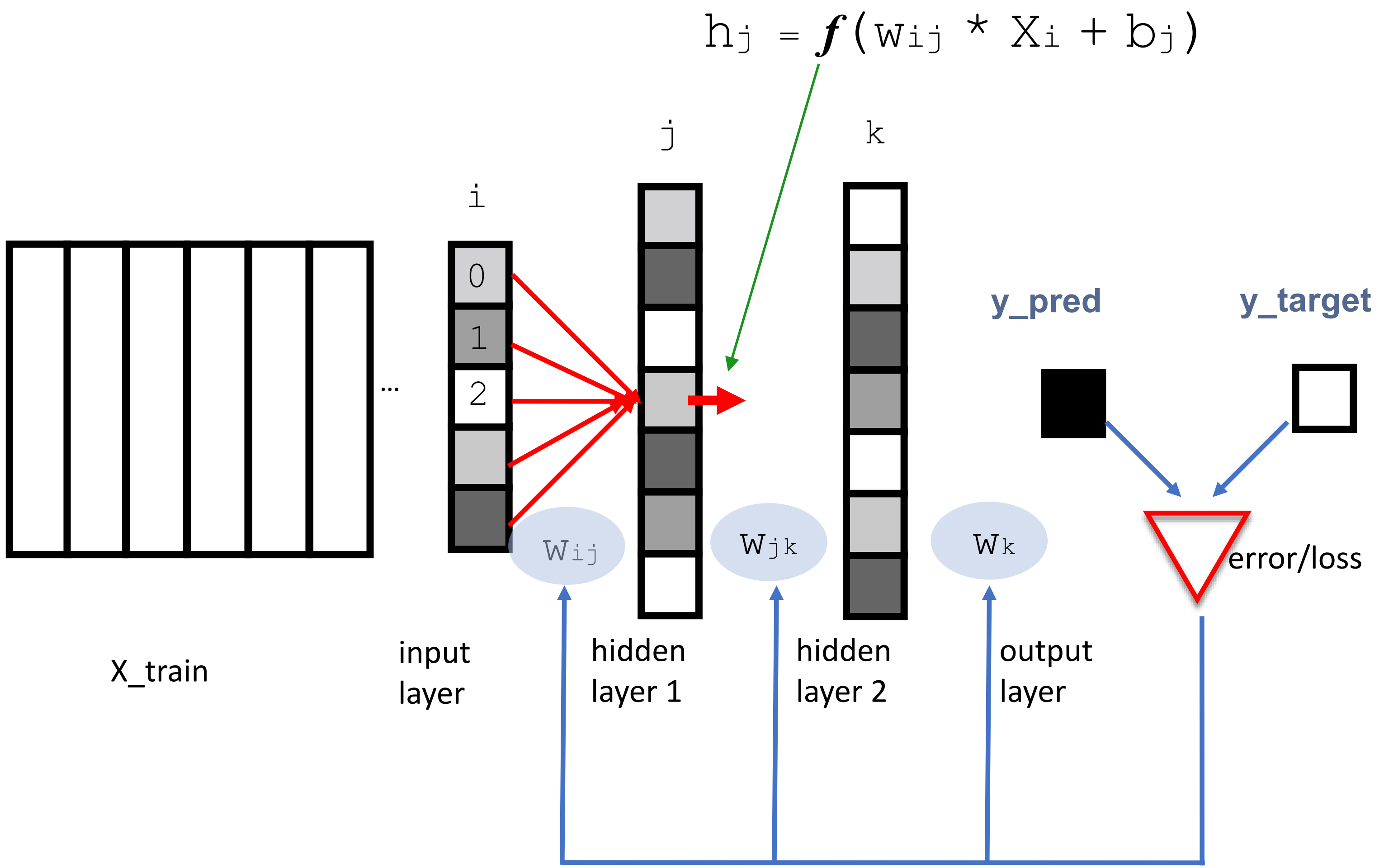


Shallow vs. Deep Sum-Product Networks

Olivier Delalleau
Department of Computer Science and Operation Research
Université de Montréal
delallea@iro.umontreal.ca

Yoshua Bengio
Department of Computer Science and Operation Research
Université de Montréal
yoshua.bengio@umontreal.ca

Next time: How Neural Network Training Works



Next time: More on Weight Update in deep neural nets

$$W_{ij} \leftarrow W_{ij} - \alpha \frac{\partial \mathcal{L}}{\partial W_{ij}}$$

Weight Update Rule

Loss Function

Gradient (Chain Rule)

Back Propagation

Play with Neural network training!

You can change the model architecture, model hyper parameters, train hyperparameters.
See what happens when you make each change.

https://colab.research.google.com/drive/1IctSzyugG9YLJKhg7WAlRtGi0nEIxy9y