Deep Learning: The Basics

Dripta Mj

Department of Mathematics

Ramakrishna Mission Vivekananda Educational and Research Institute

Belur Math, India

Machine Translation

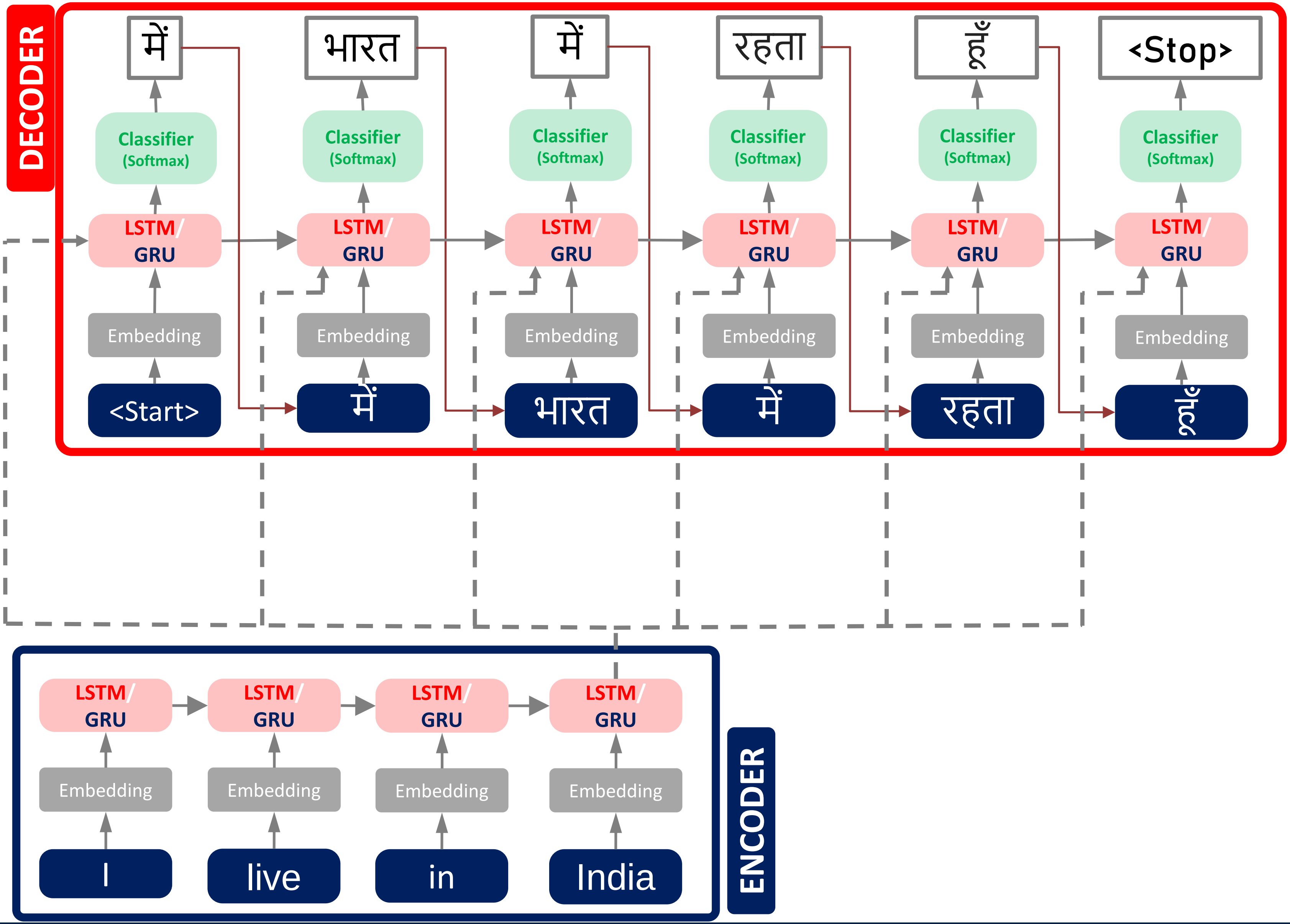
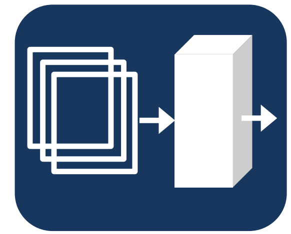
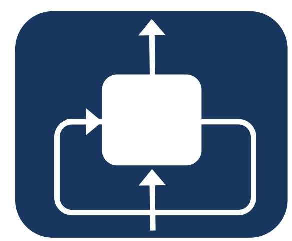


Image captioning



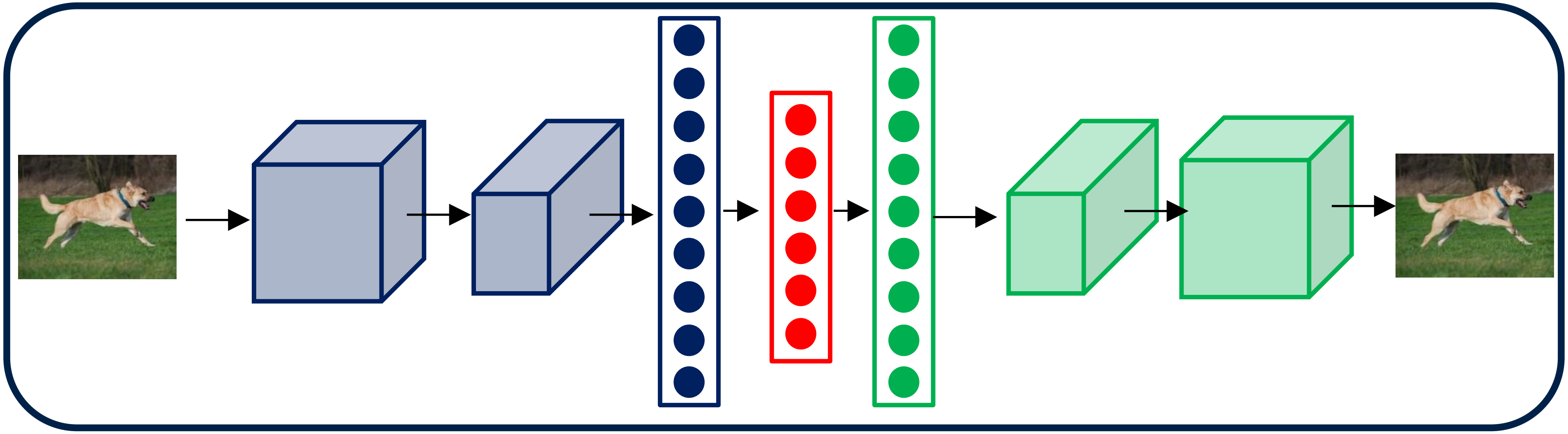
Deep CNN



Language generating RNN

A dog running in a field

Convolutional Autoencoder



The rise of Deep Learning ....

BIG DATA

- World is data rich!
- Deep learning needs big datasets.

HARDWARE

- Graphics Processing Units

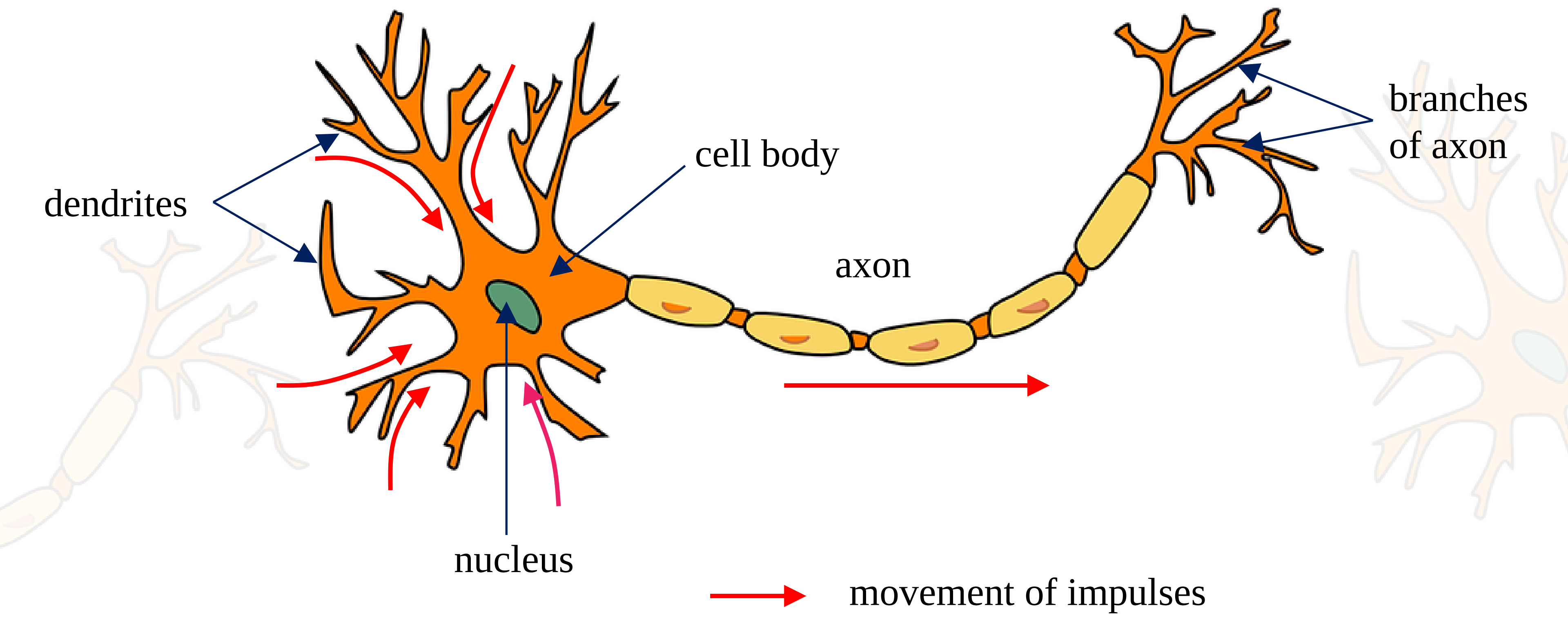
SOFTWARE

- Open source toolboxes
- Efficient implementations

TensorFlow PyTorch

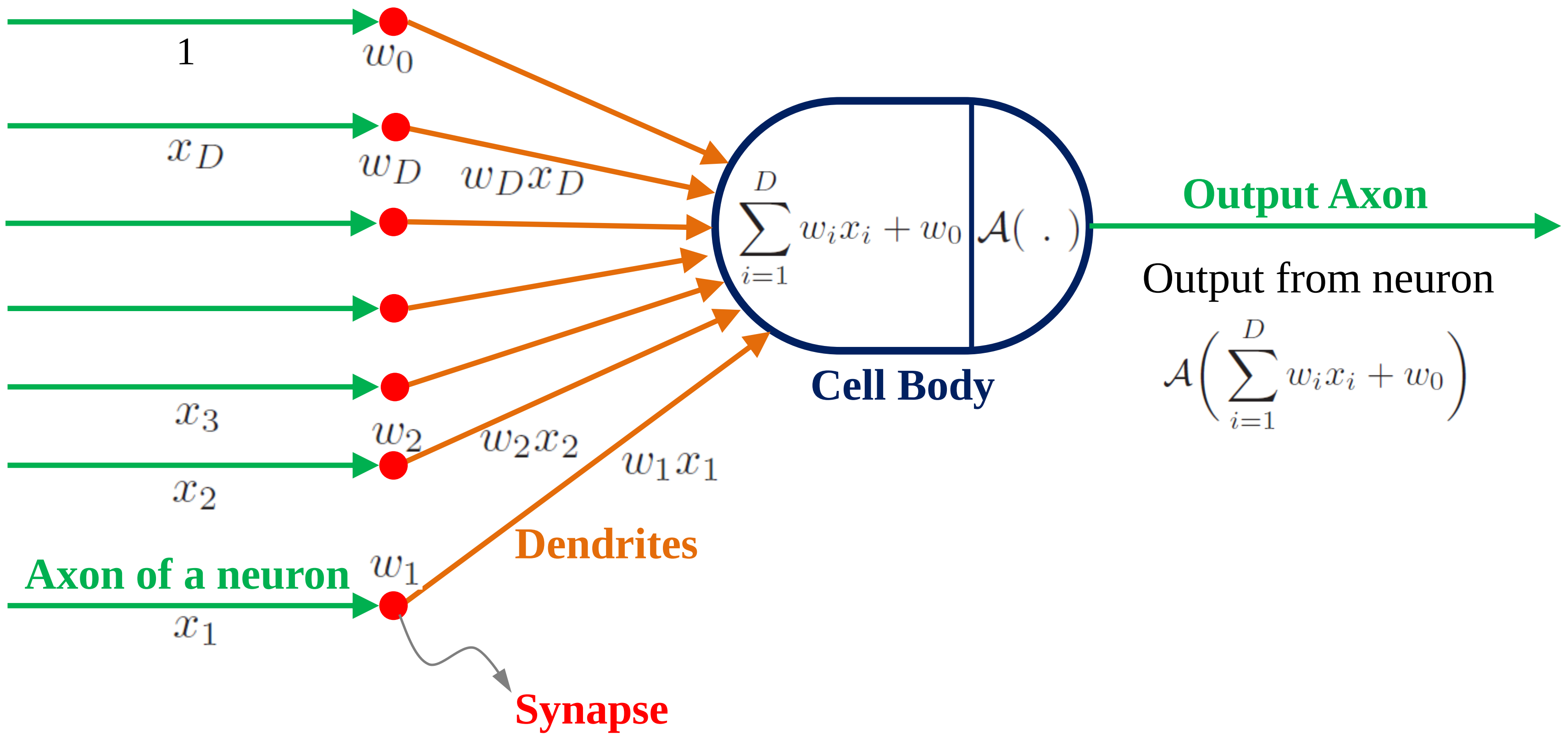
Neuron

- The brain is composed of densely interconnected network of neurons.
- Each neuron has a body, axon, synapses and dendrites.
- A neuron fires if the sum of the weighted signals is greater than a threshold.
- Signals propagate along neurons via axons, synapses and dendrites.

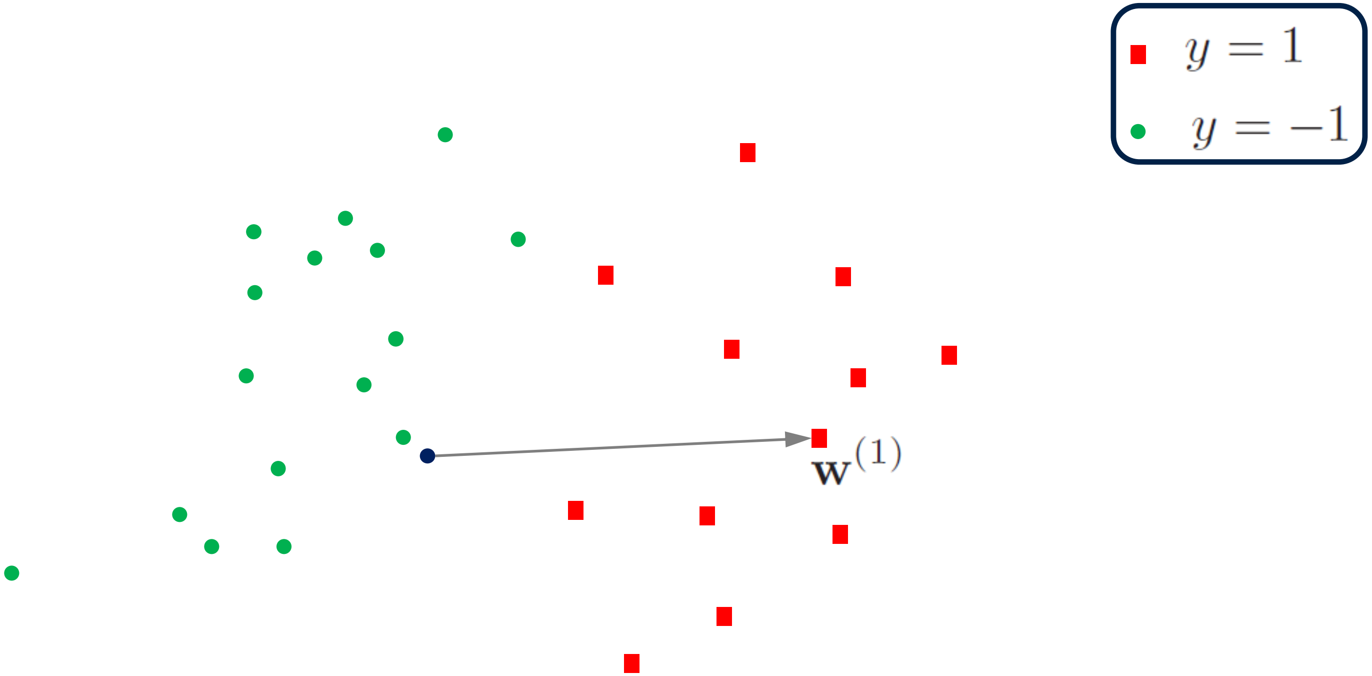


Mathematical model of a neuron

- The threshold activation function "fires" if the weighted sum of the inputs and bias exceeds a certain threshold.



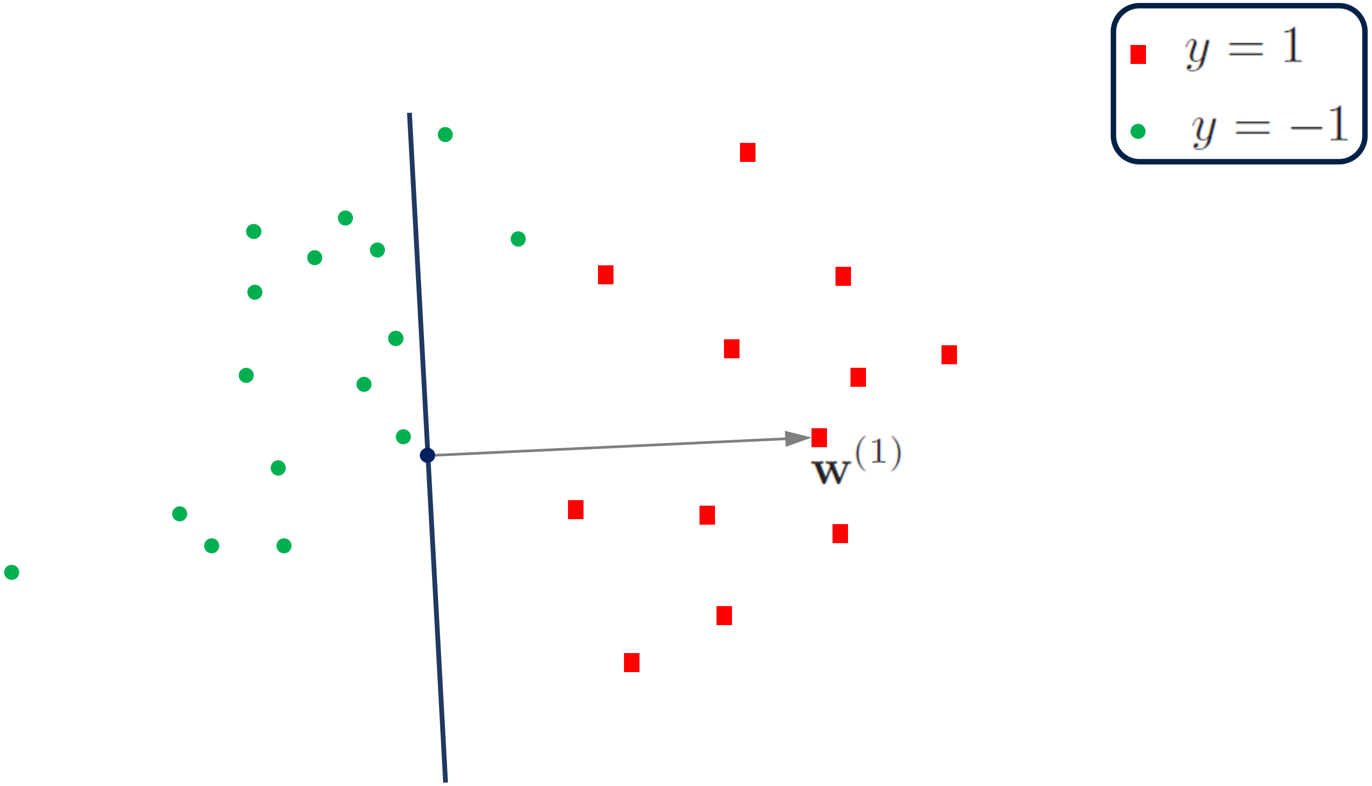
Visualization



Weight update rule:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + y^{(n)}\mathbf{x}^{(n)}$$

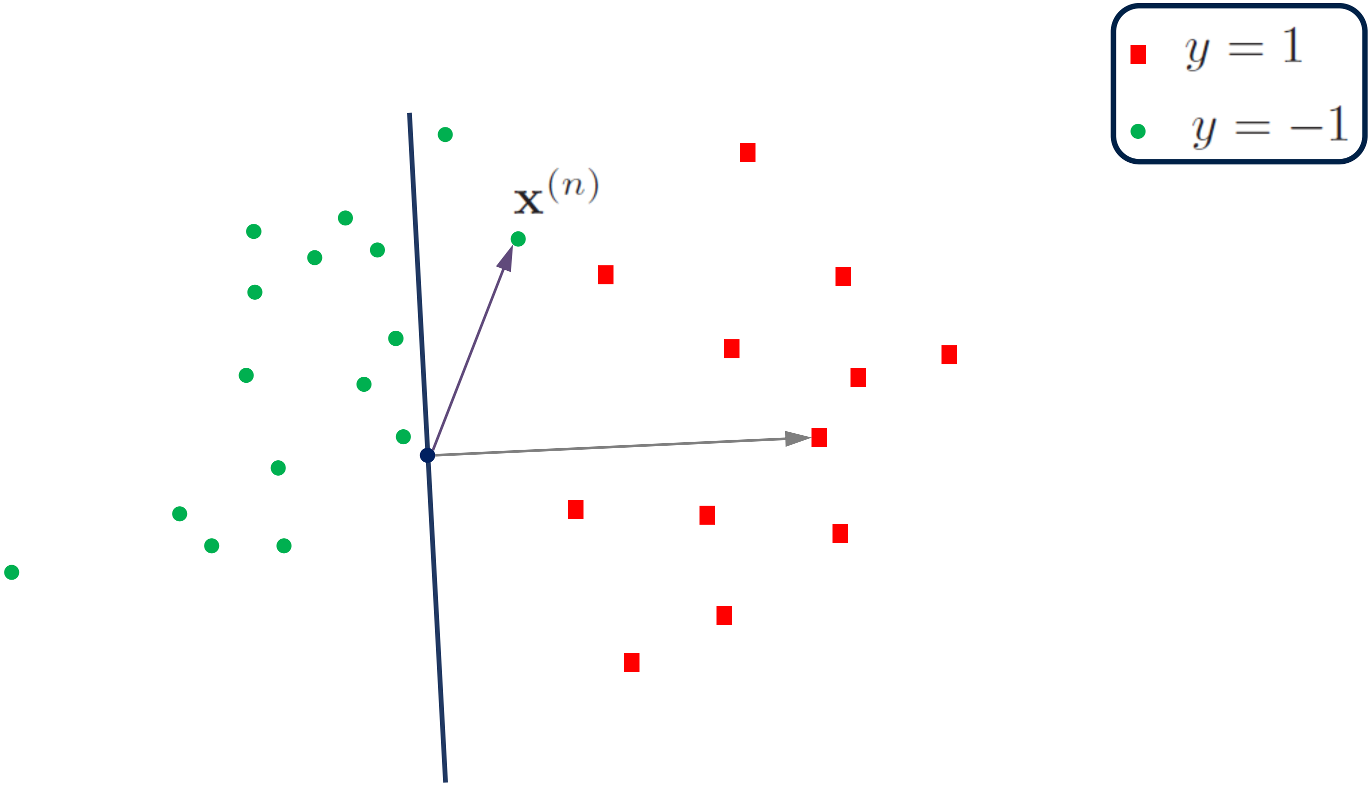
Visualization



Weight update rule:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + y^{(n)}\mathbf{x}^{(n)}$$

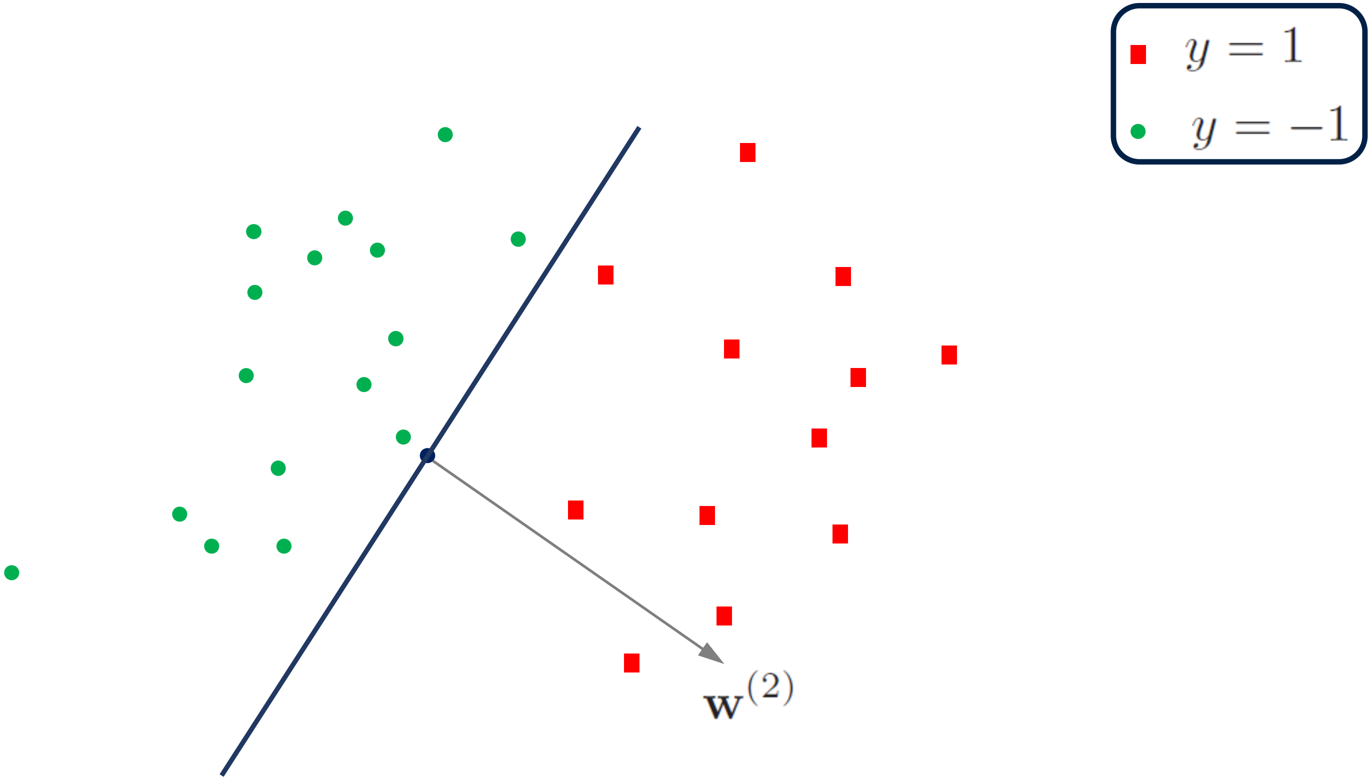
Visualization



Weight update rule:

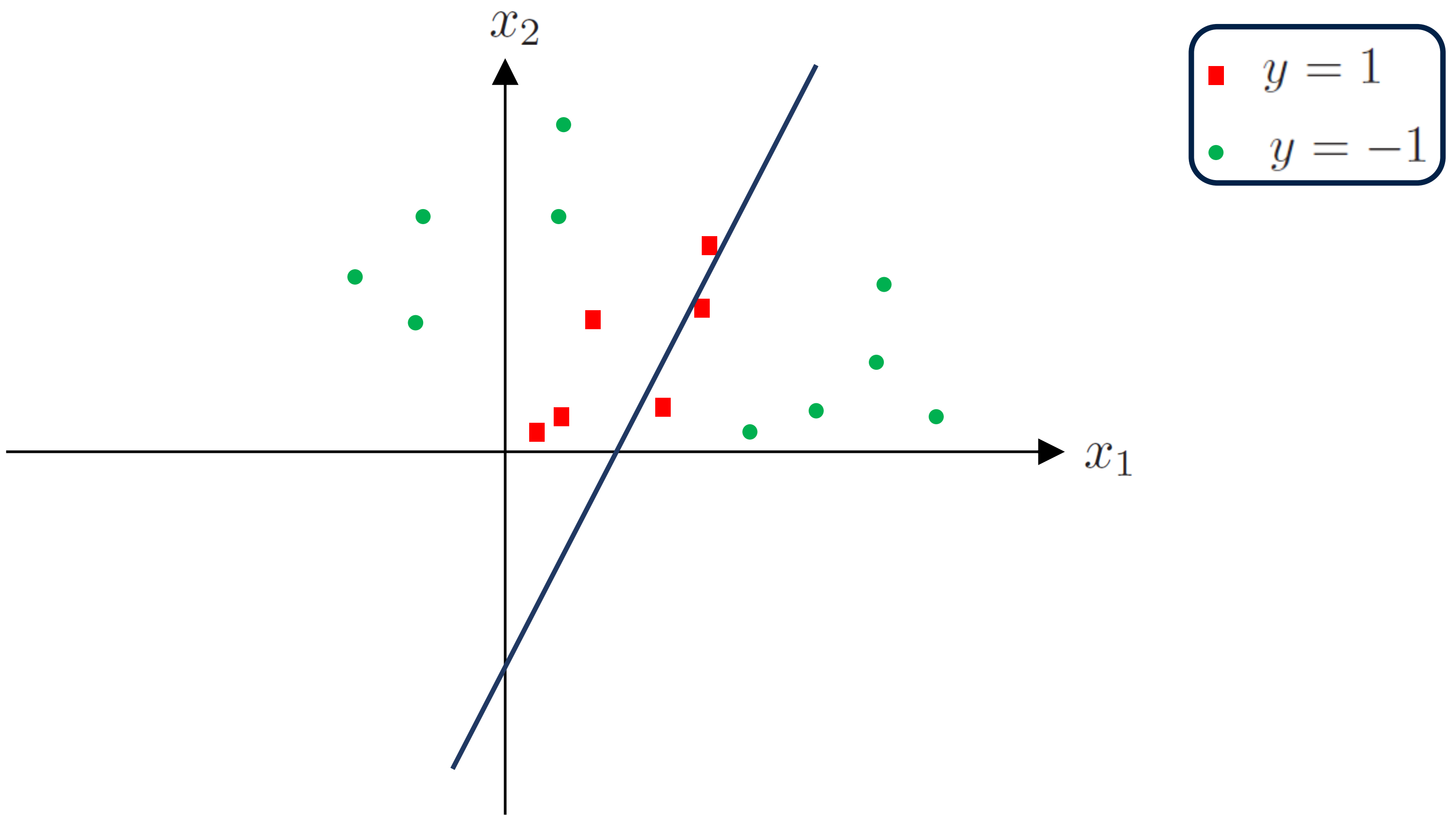
$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + y^{(n)}\mathbf{x}^{(n)}$$

Visualization

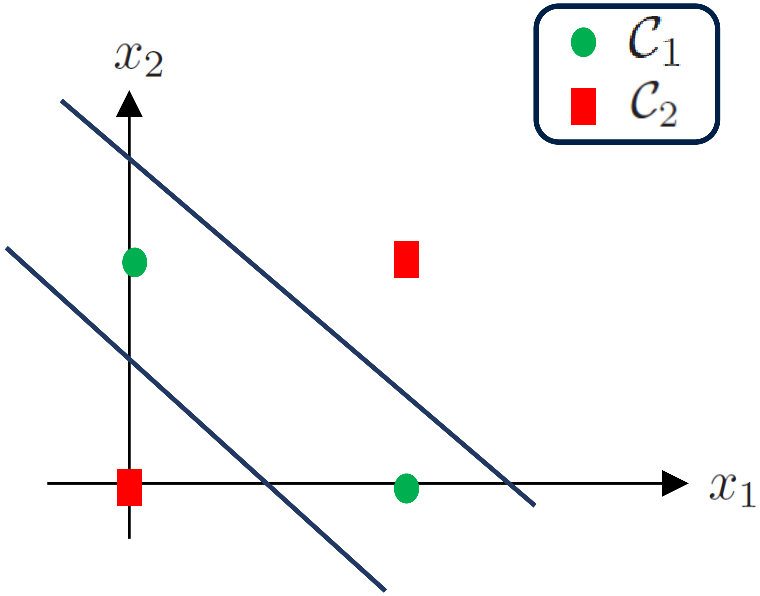


Weight update rule: $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + y^{(n)}\mathbf{x}^{(n)}$

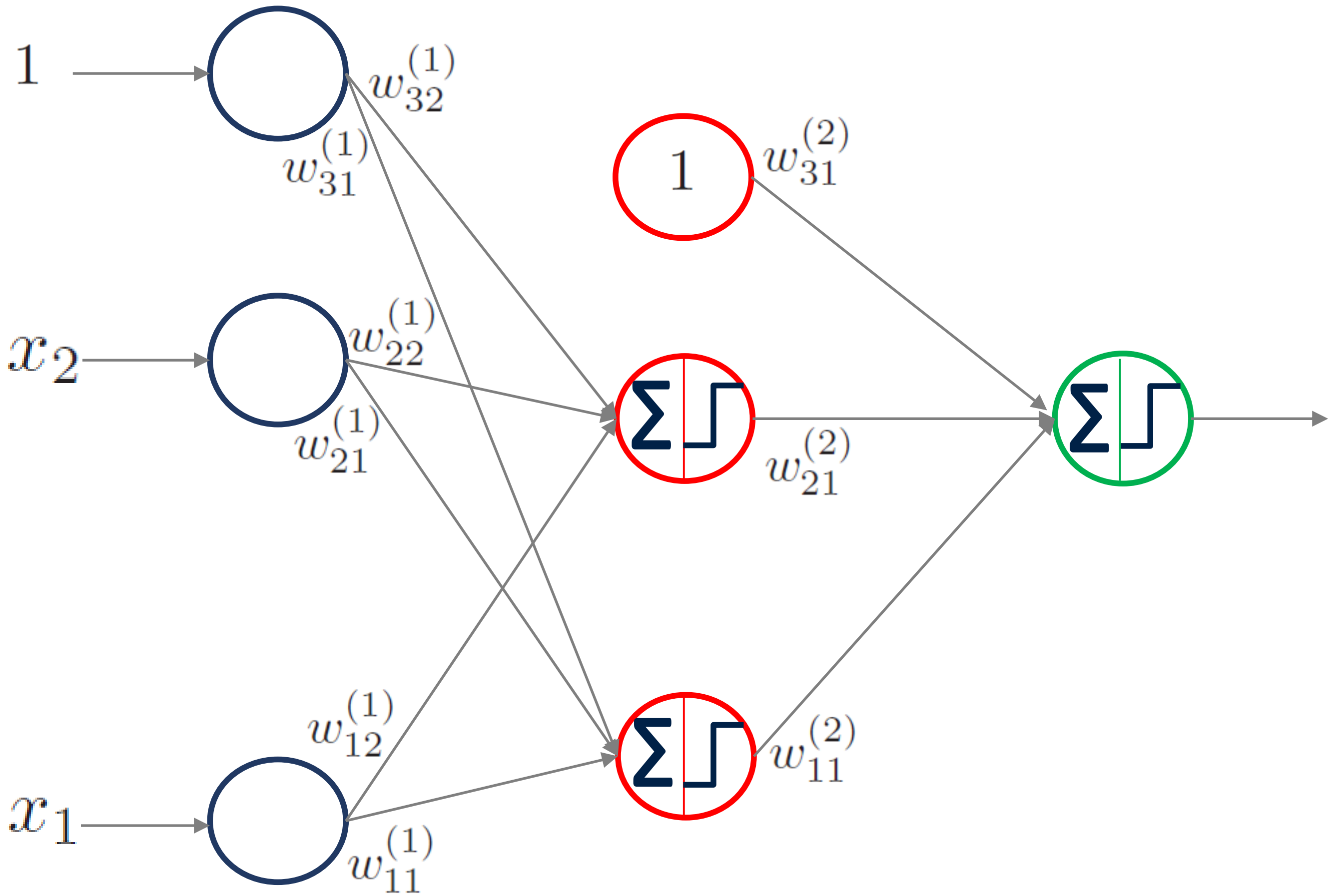
Shortcomings of single layer perceptron Learning



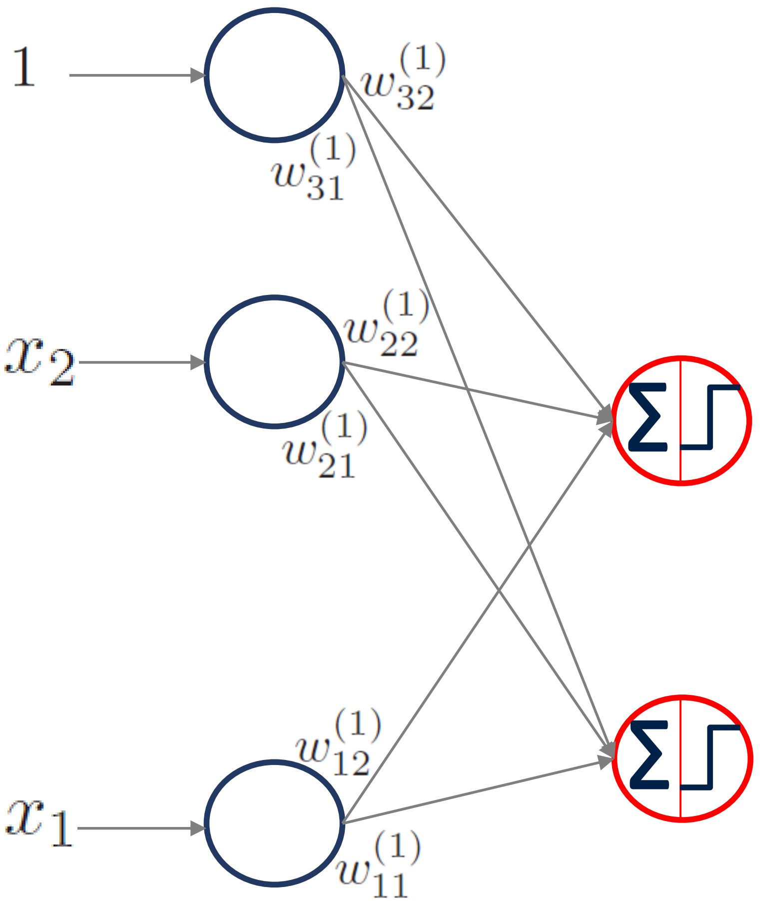
Combination of classifiers



Multi-layer perceptron



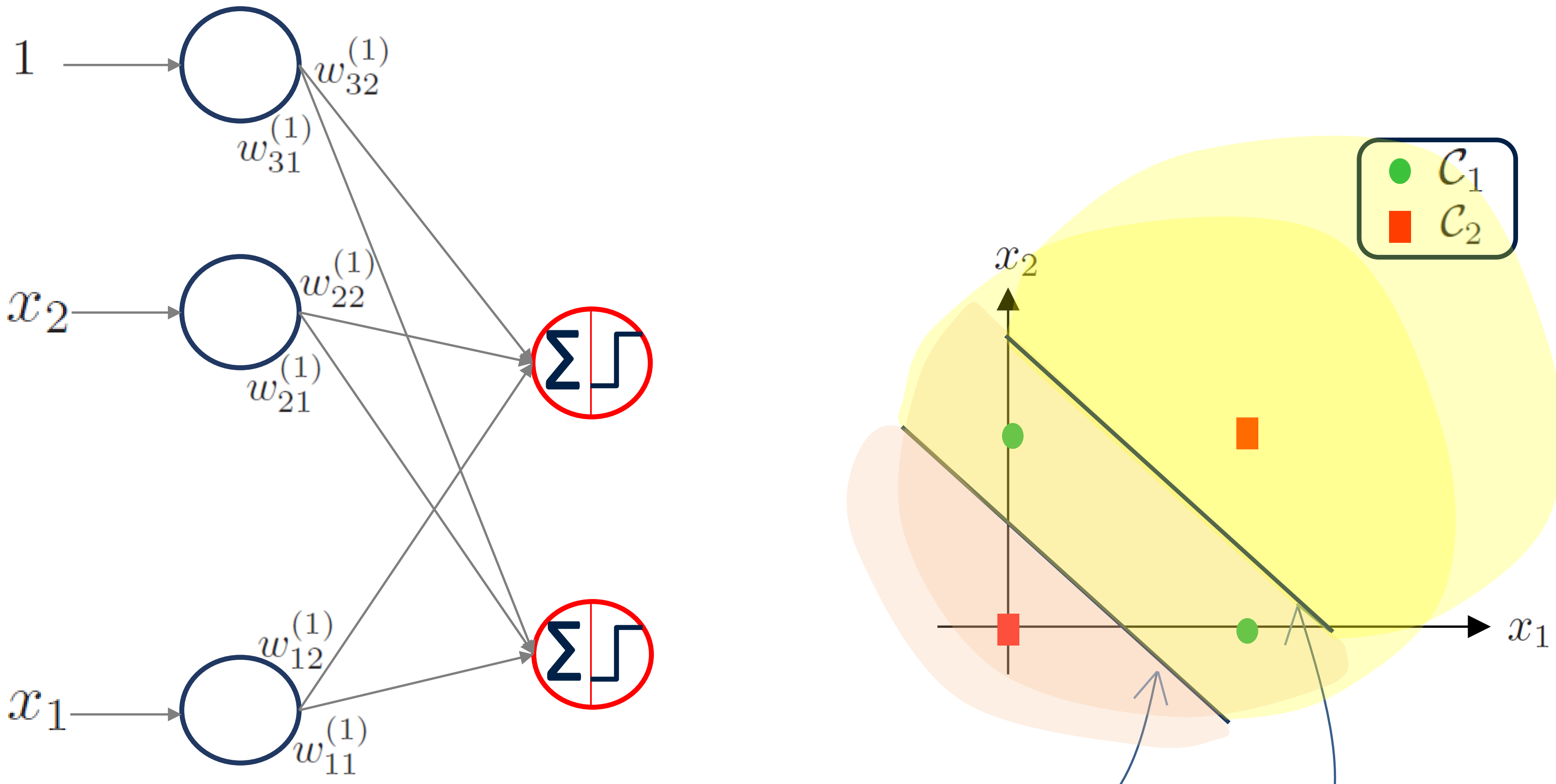
First layer



$$z_2 = \mathcal{A}(w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{32}^{(1)})$$

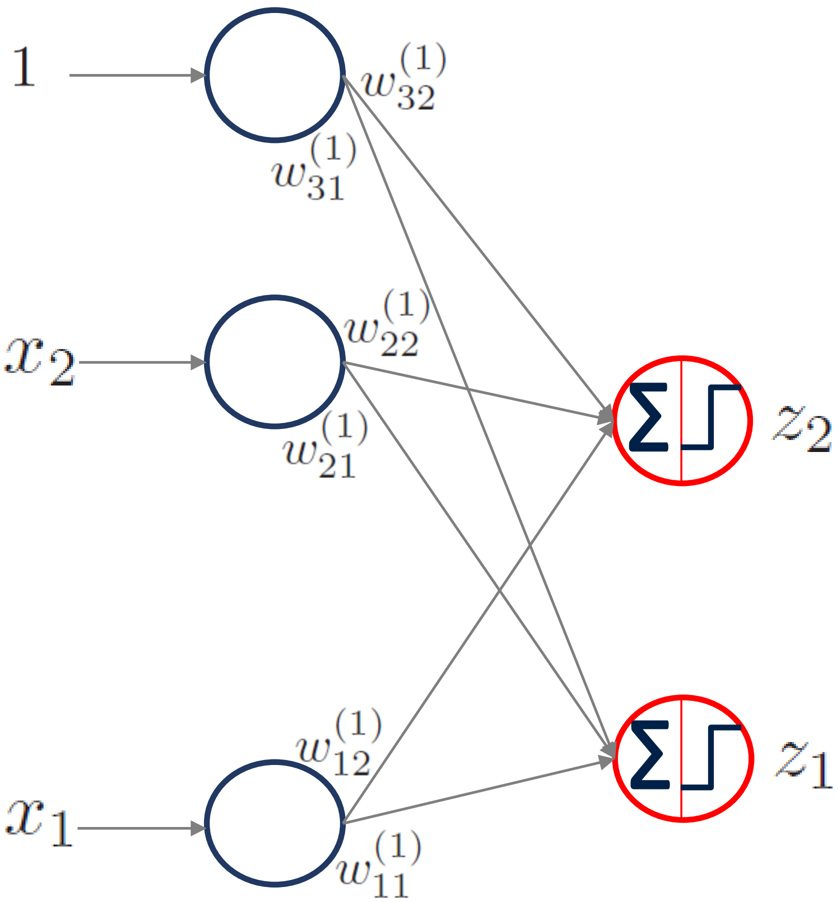
$$z_1 = \mathcal{A}(w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + w_{31}^{(1)})$$

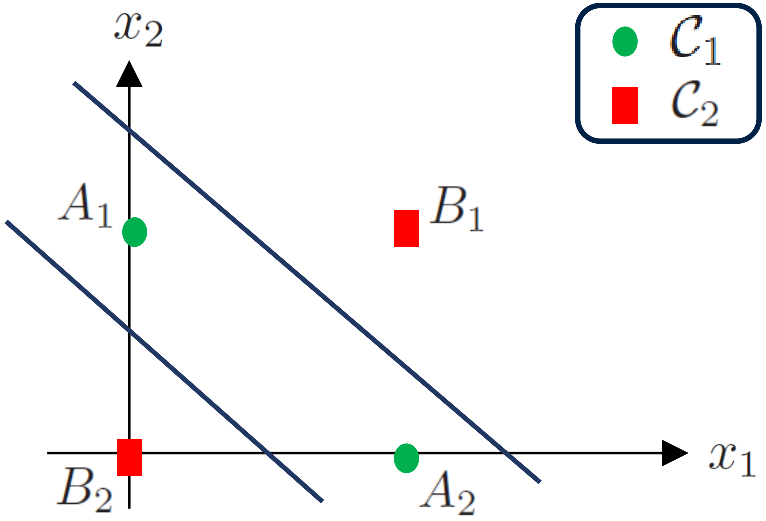
First layer: geometry

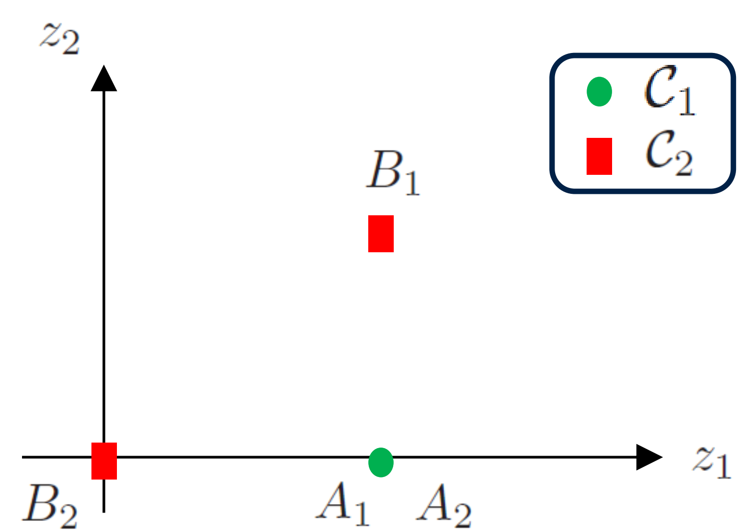


- Take $w_{11}^{(1)} = 1$, $w_{21}^{(1)} = 1$, and $w_{31}^{(1)} = -0.5$.
- Take $w_{12}^{(1)} = 1$, $w_{22}^{(1)} = 1$, and $w_{32}^{(1)} = -1.5$.

First layer: geometry

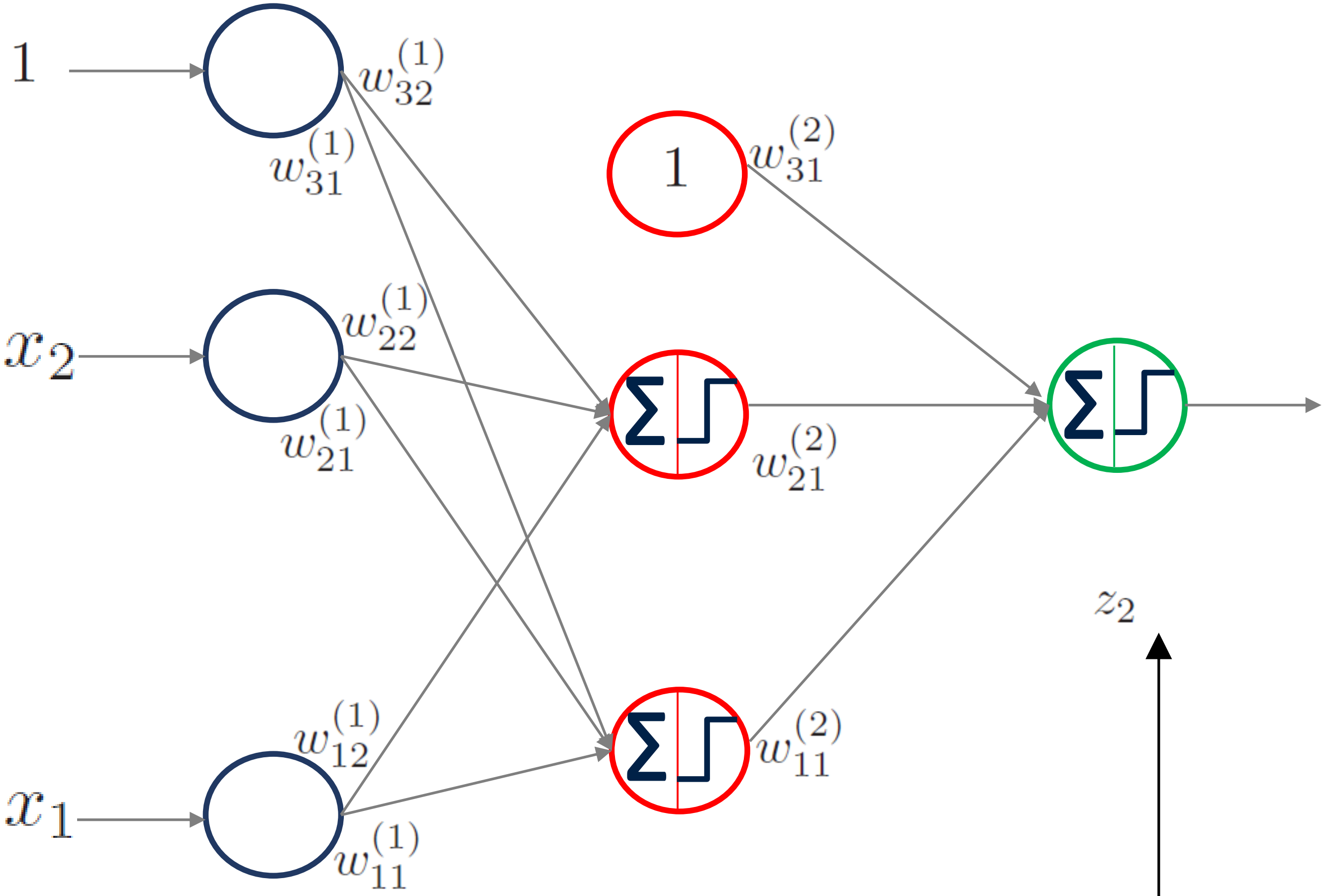


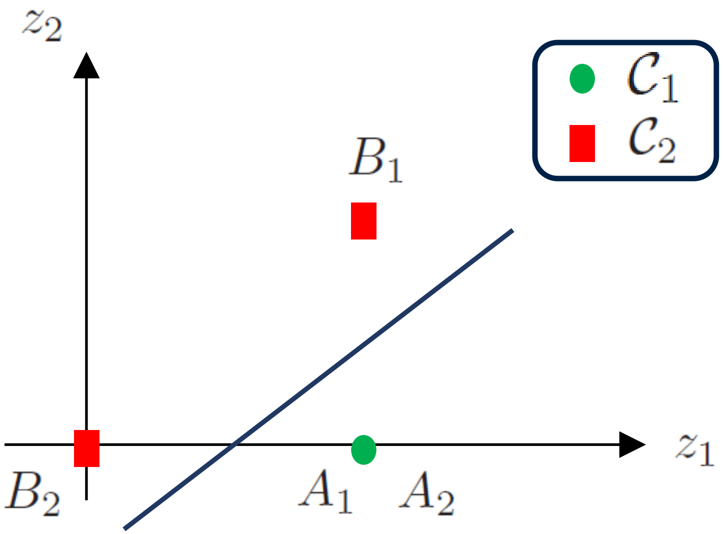




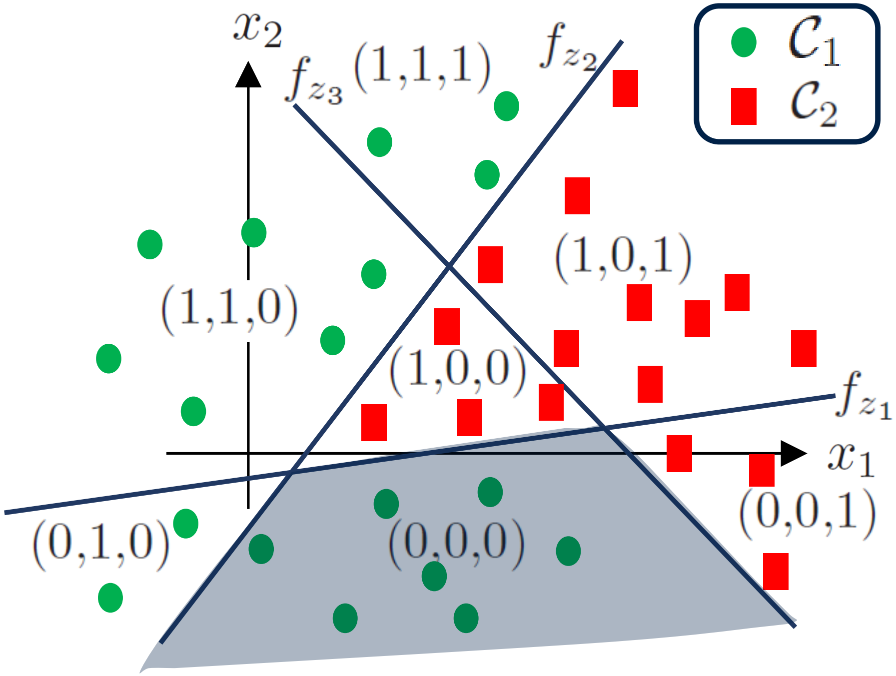
- Now what do you think you need to do?

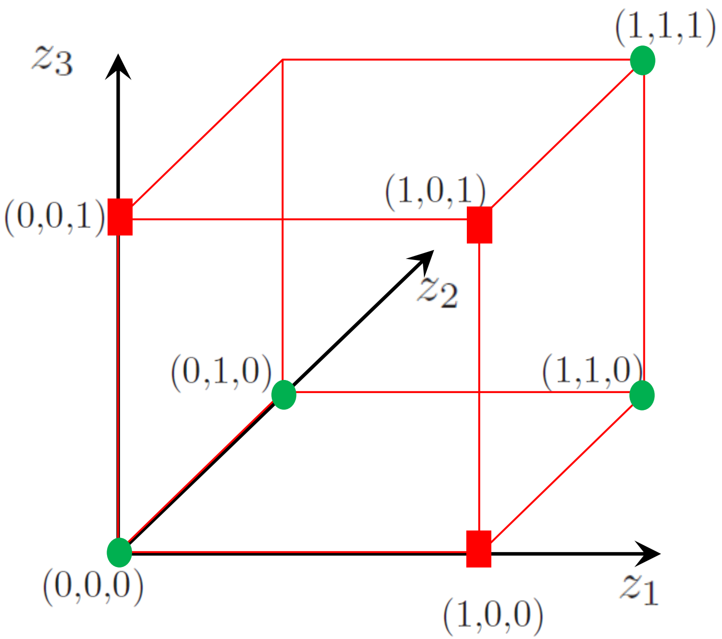
Second layer: geometry

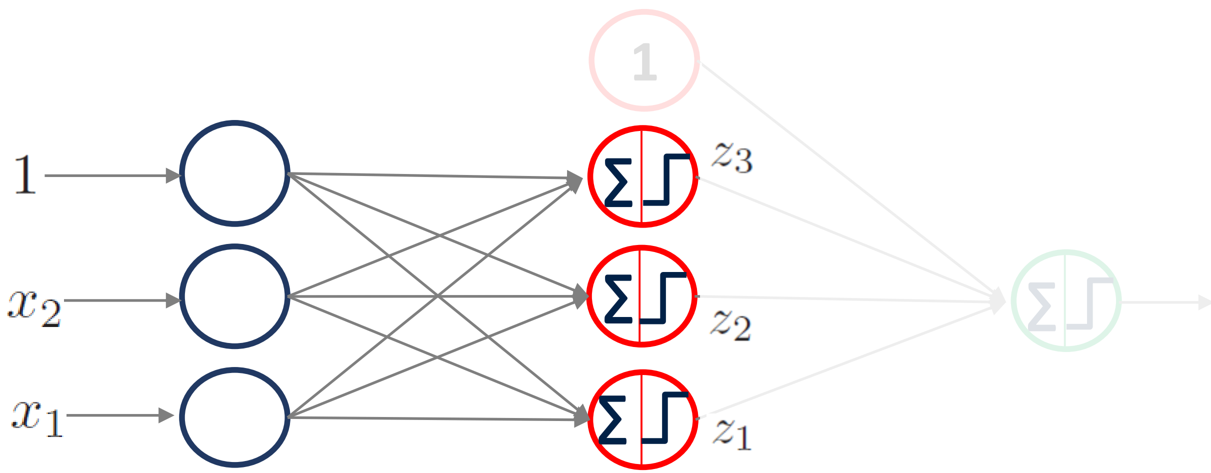




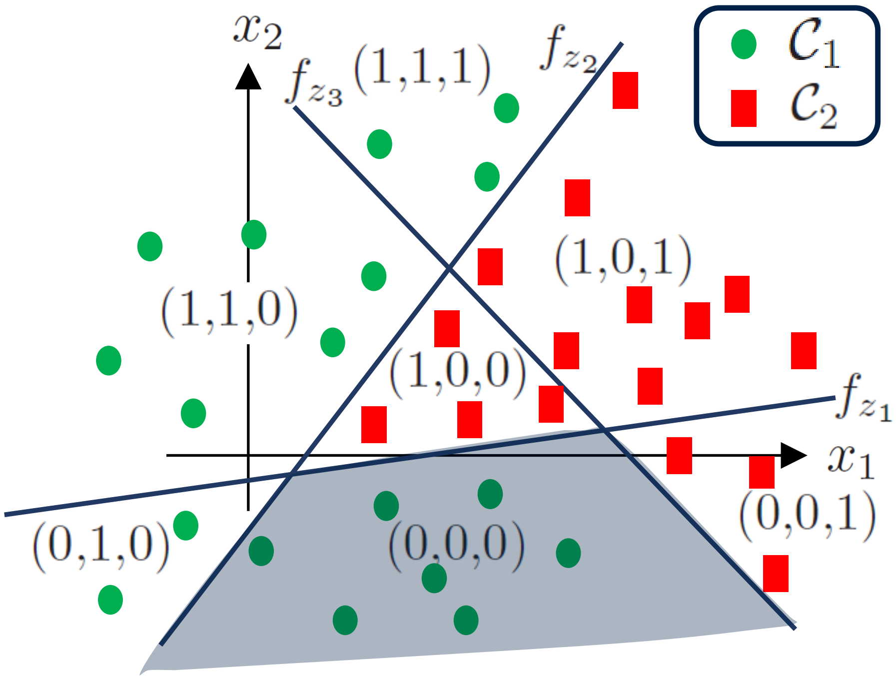
Single hidden layer

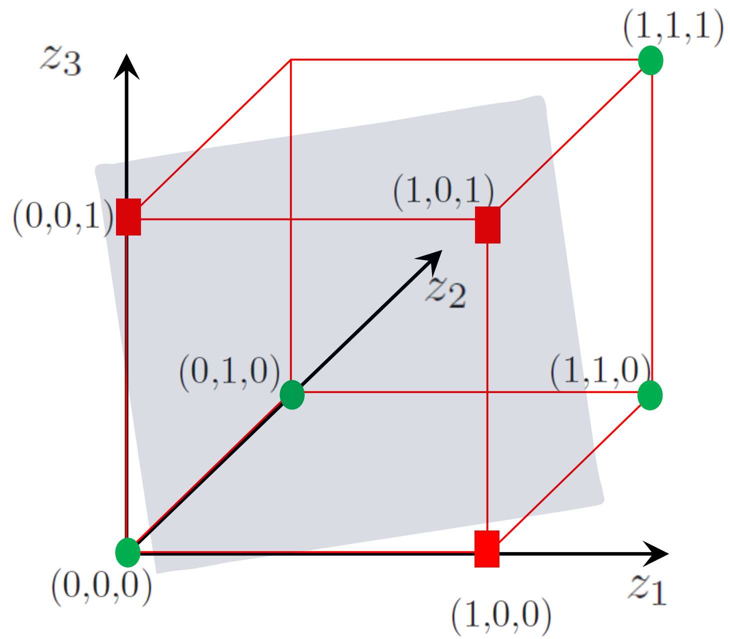


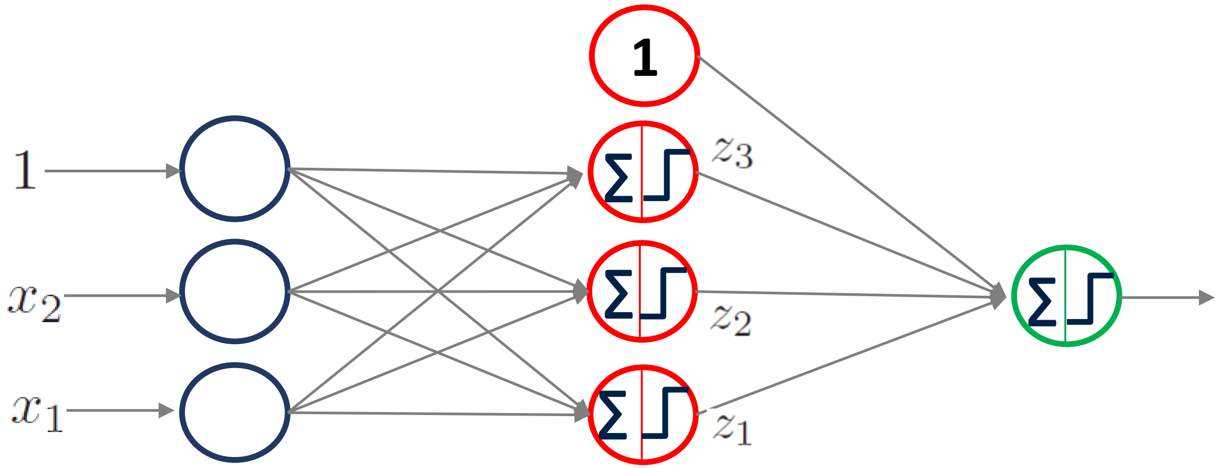




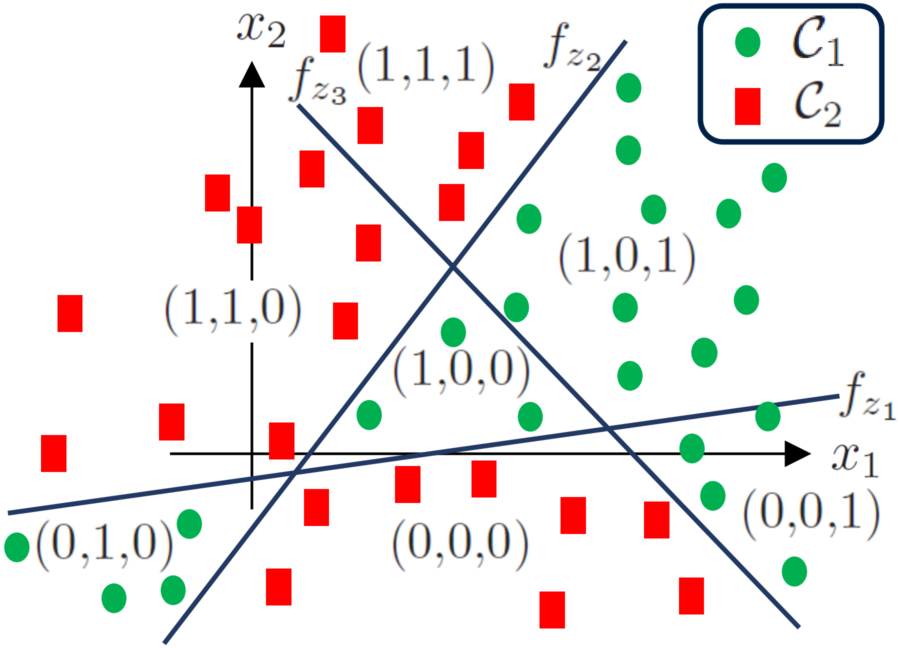
Single hidden layer

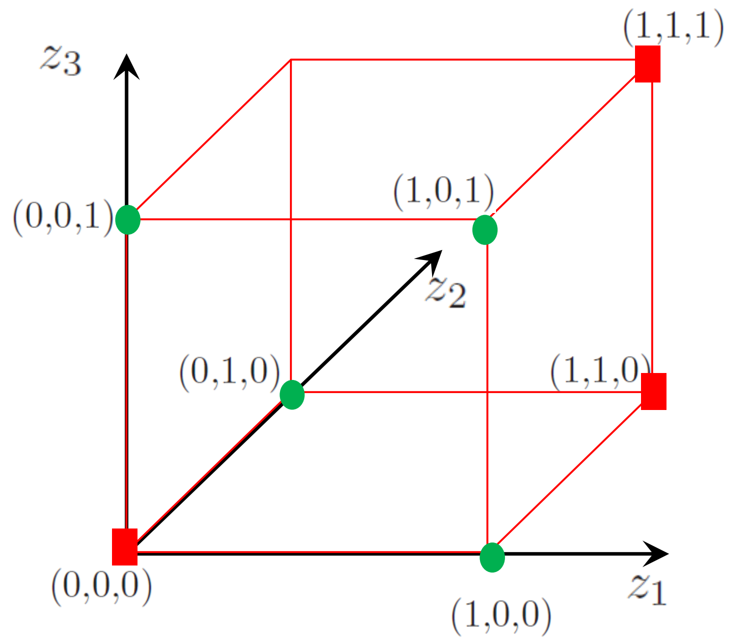


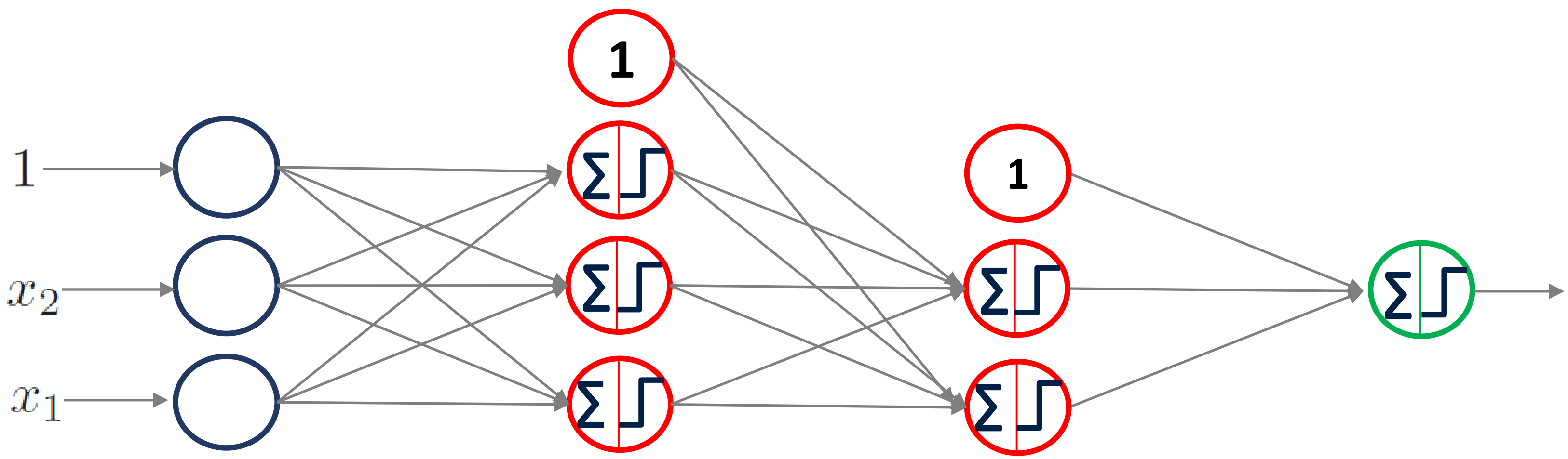




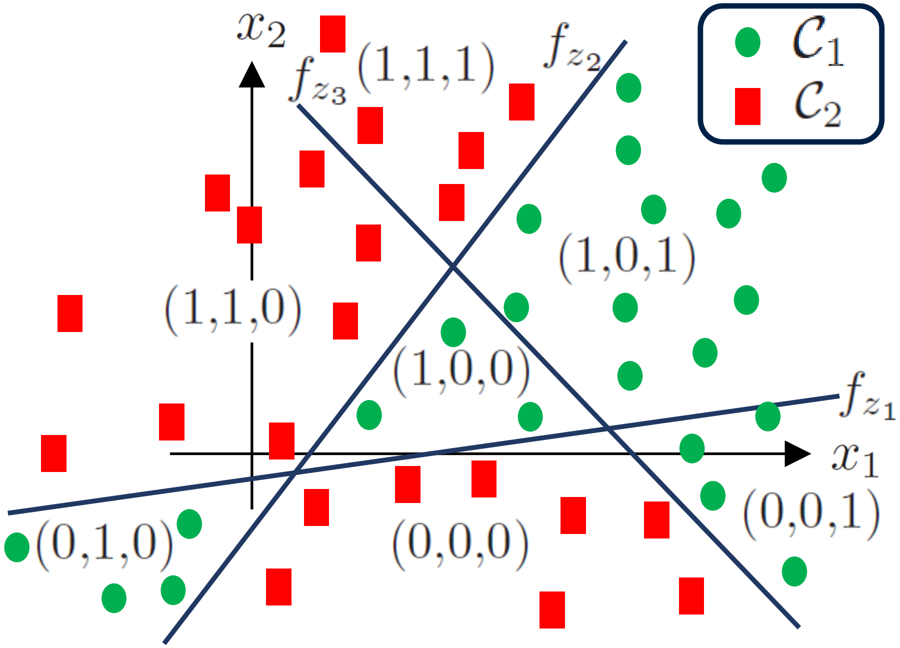
Two hidden layers

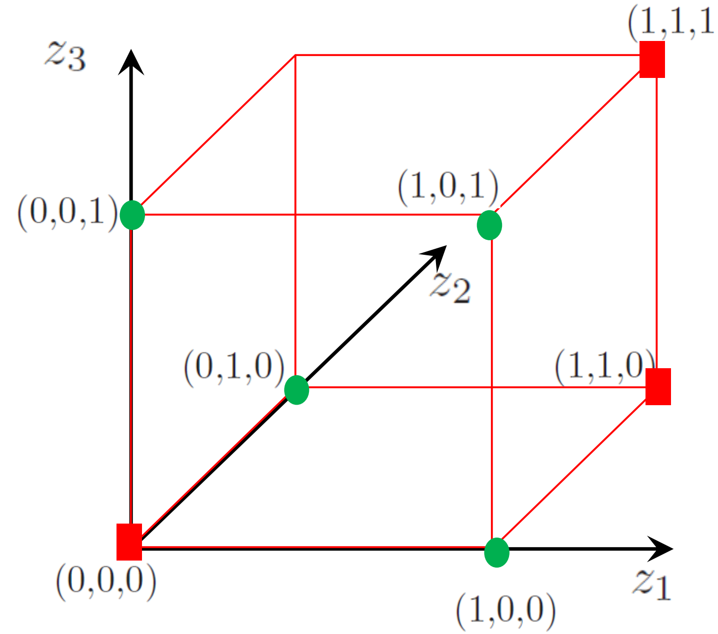


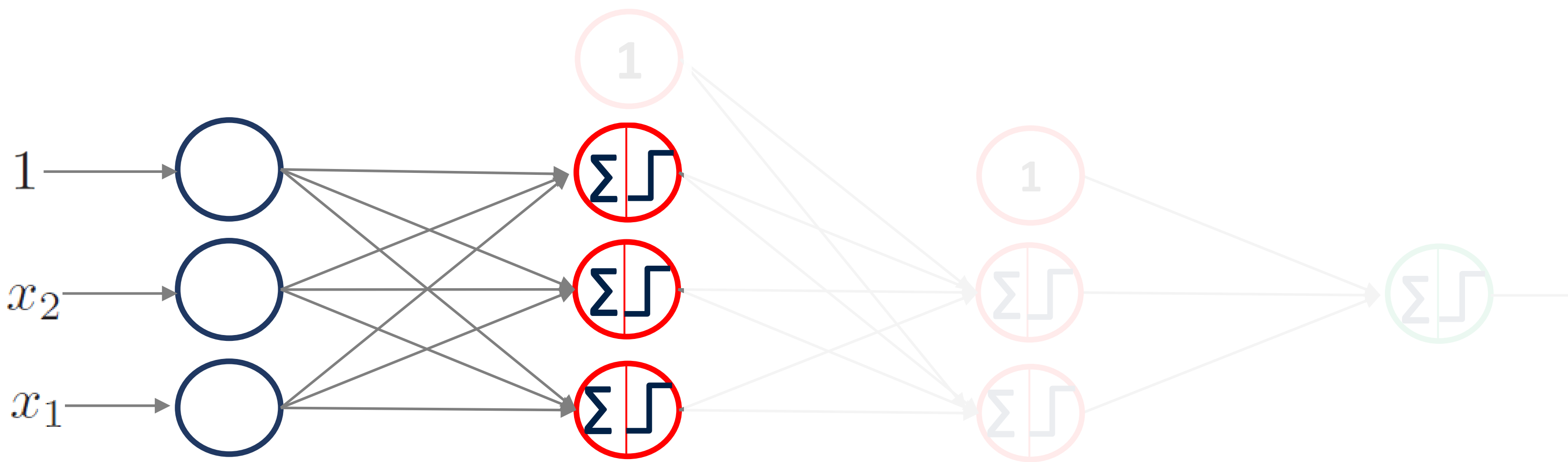




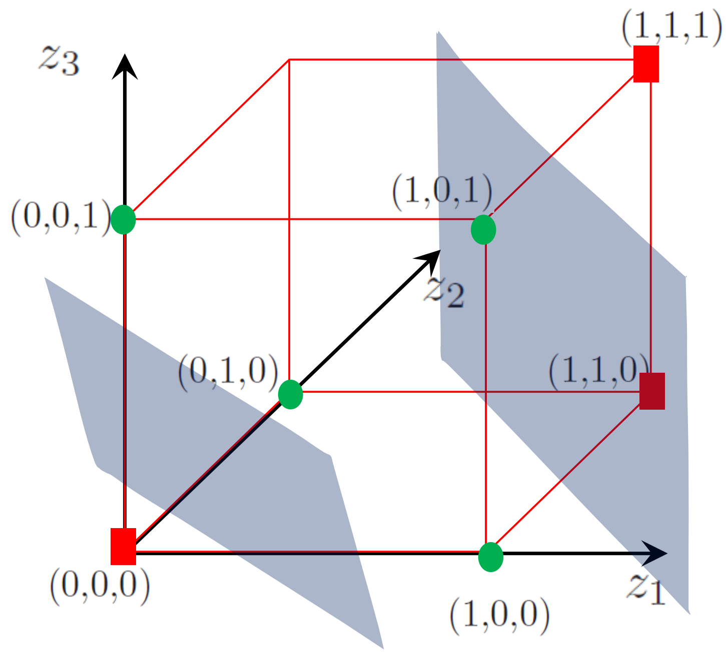
Two hidden layers

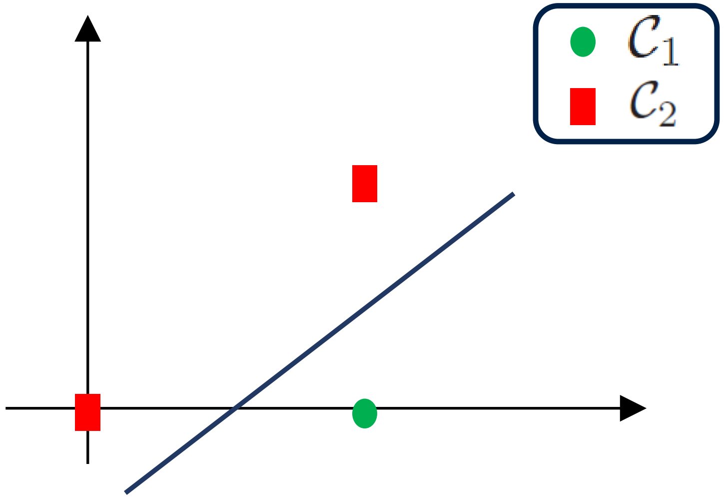


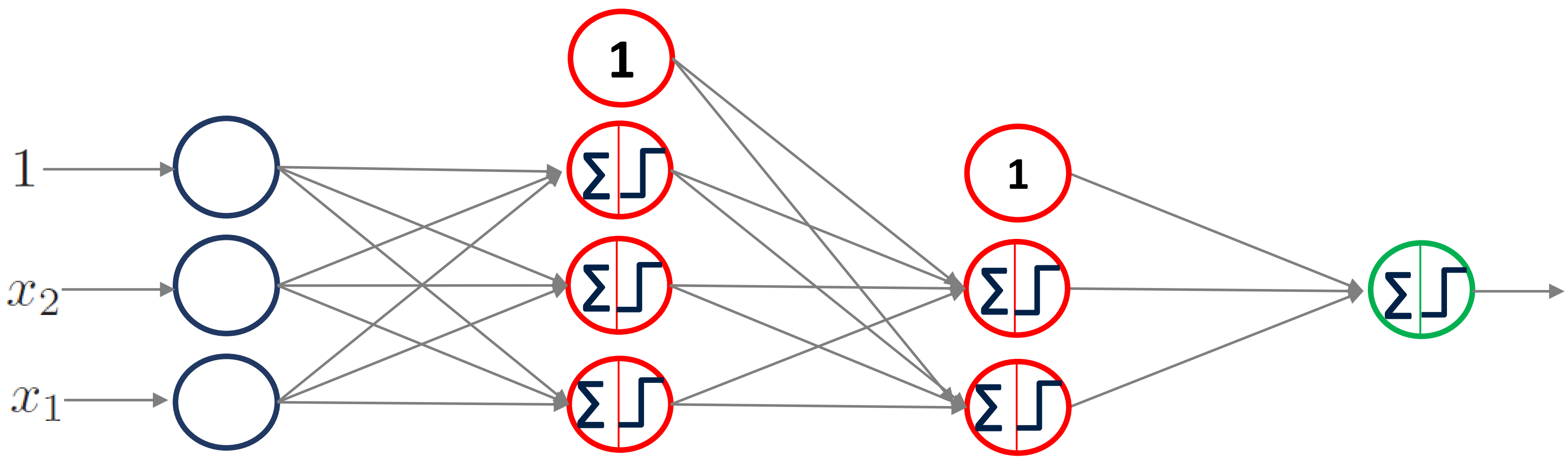




Two hidden layers



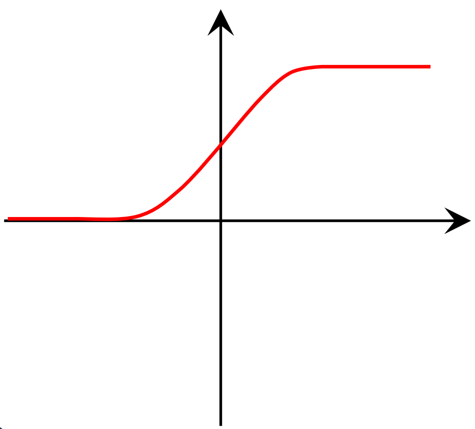




Activation functions

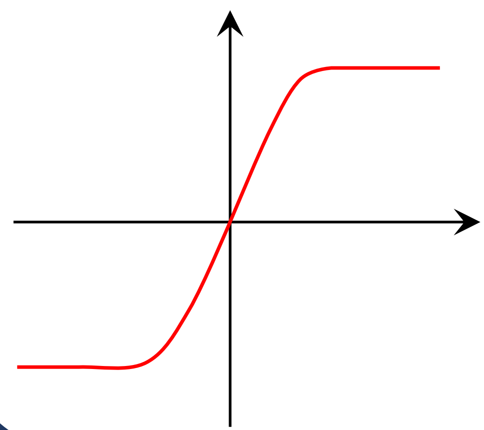
Sigmoid

$$\mathcal{A}(z) = \frac{1}{1 + \exp(-z)}$$



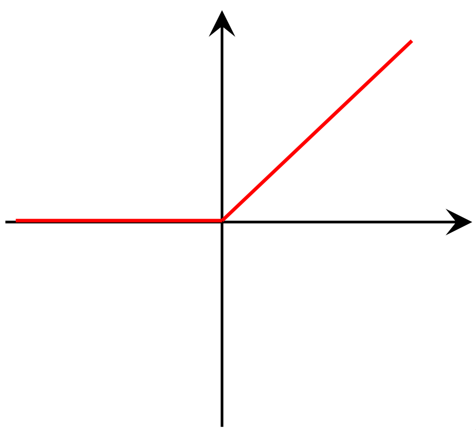
tanh

$$\mathcal{A}(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$

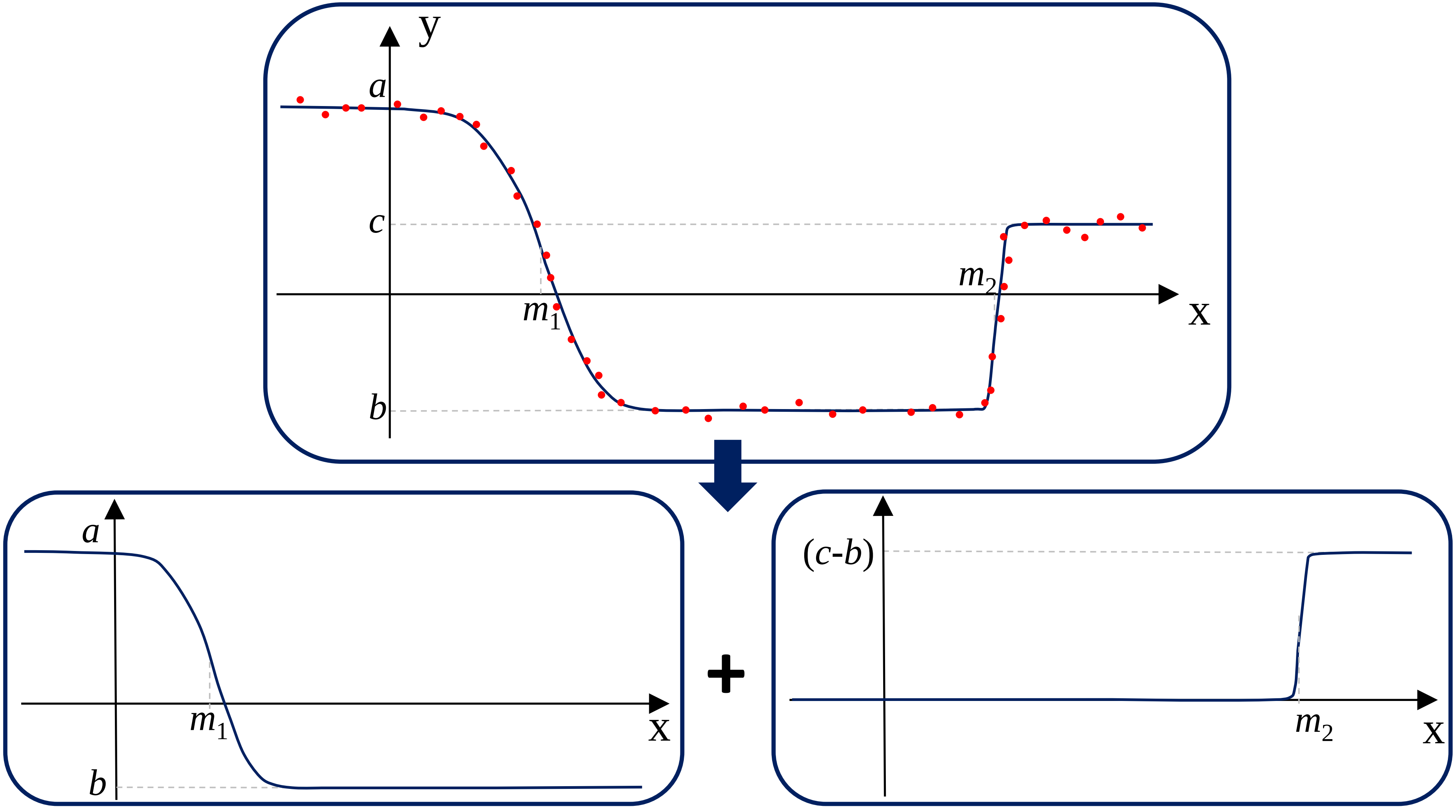


ReLU

$$\mathcal{A}(z) = \max(z, 0)$$

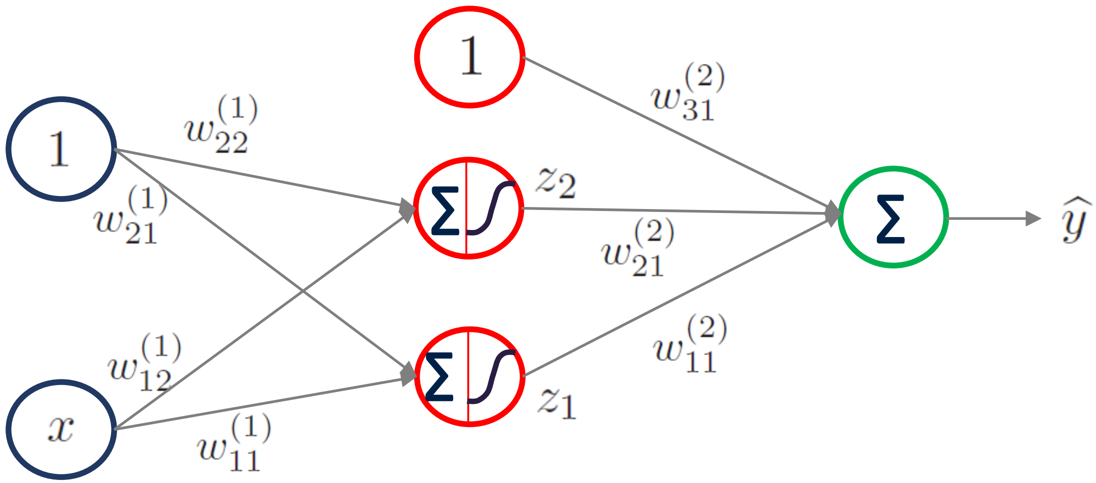


Regression problem



$$\overline{y} = b + \frac{a-b}{1+\exp\big[-k_1(x-m_1)\big]} + \frac{c-b}{1+\exp\big[-k_2(x-m_2)\big]}$$

Neural network structure



- Final output:

$$\begin{aligned}\widehat{y} &= w_{11}^{(2)} z_1 + w_{21}^{(2)} z_2 + w_{31}^{(2)} \\ &= \frac{w_{11}^{(2)}}{1+\exp\left[-w_{11}^{(1)}x - w_{21}^{(1)}\right]} + \frac{w_{21}^{(2)}}{1+\exp\left[-w_{12}^{(1)}x - w_{22}^{(1)}\right]} + w_{31}^{(2)}\end{aligned}$$

- Consider the following values of the weights:

$$\text{First layer}: w_{11}^{(1)} = k_1; \quad w_{21}^{(1)} = -k_1 m_1; \quad w_{12}^{(1)} = k_2; \quad w_{22}^{(1)} = -k_2 m_2$$

$$\text{Second layer}: w_{11}^{(2)} = (a-b); \quad w_{21}^{(2)} = (c-b); \quad w_{31}^{(2)} = b$$

- On substitution we get $\widehat{y} = \overline{y}$

Training a neural network

- Goal – Optimize for weights:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{n=1}^{N} L\big(\mathbf{y}^{(n)}, \mathbf{y}^{*(n)}\big)$$

  where $\mathbf{y}^{*(n)}$ is the prediction of the neural network.

- Select an appropriate loss function:
  - Squared loss:

$$L\big(\mathbf{y}^{(n)}, \mathbf{y}^{*(n)}\big) = \frac{1}{2} \sum_{j=1}^{J} \big(y_j^{(n)} - y_j^{*(n)}\big)^2$$

  - Binary cross-entropy loss:

$$L\big(y^{(n)}, y^{*(n)}\big) = -y^{(n)} \log(y^{*(n)}) - (1 - y^{(n)}) \log(1 - y^{*(n)})$$

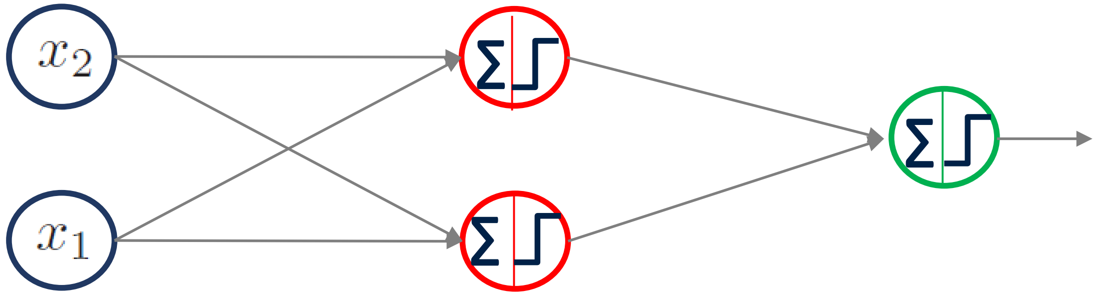
  - Cross-entropy loss:

$$L\big(\mathbf{y}^{(n)}, \mathbf{y}^{*(n)}\big) = -\sum_{j=1}^{J} y_j^{(n)} \log y_j^{*(n)}$$

- Gradient descent:

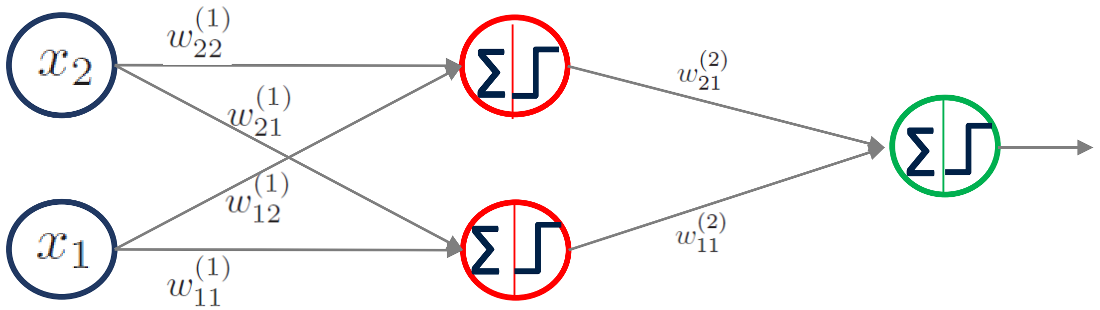
$$\mathbf{w}^{t+1} = \mathbf{w}^t - \xi \frac{\partial L}{\partial \mathbf{w}^t}$$

Backpropagation: Procedure



- Training is achieved in two steps:
  - Step 1: Forward pass the inputs through the network.
  - Step 2: In order to adjust the parameters we go backwards. Parameters are updated using gradients.

Binary classification

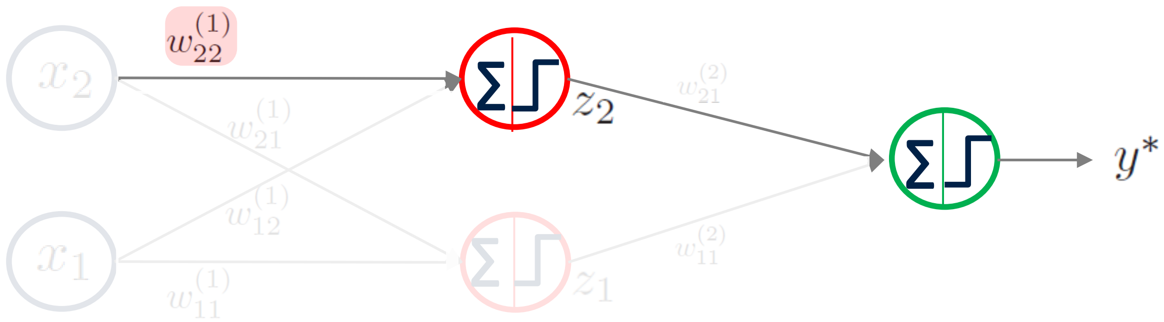


- Hidden layer outputs
  - $z_1 = \mathcal{A}\big(w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2\big)$
  - $z_2 = \mathcal{A}\big(w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2\big)$
- Final output: $y^* = \mathcal{A}\big(w_{11}^{(2)} z_1 + w_{21}^{(2)} z_2\big)$
- Activation function – Sigmoid

$$\mathcal{A}(z) = \frac{1}{1+\exp(-z)}$$

- Loss function: $L = -y \log(y^*) - (1-y)\log(1-y^*)$

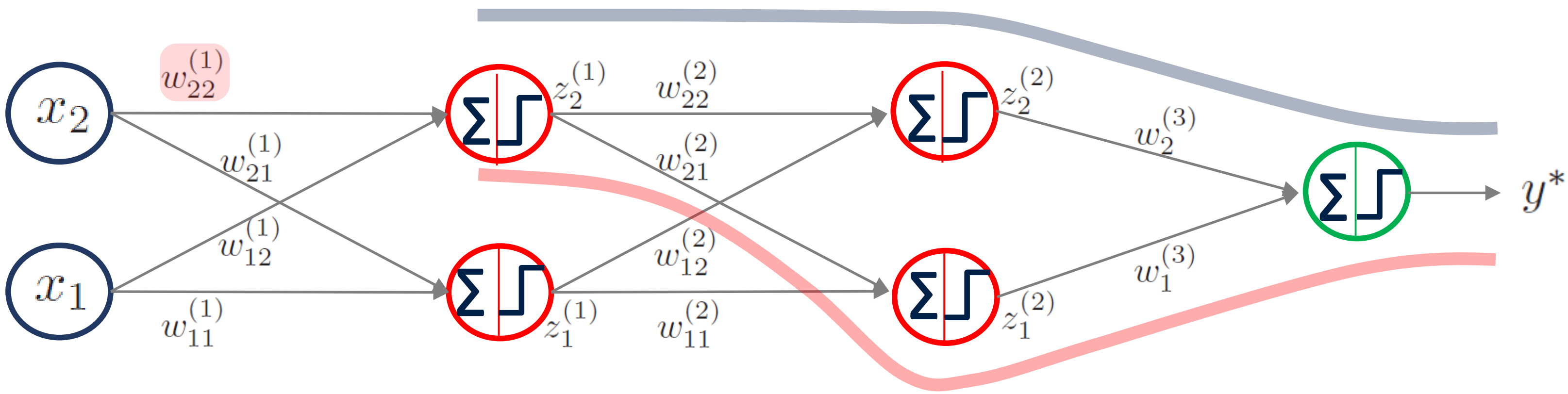
Backpropagation: 2 layer network



- Gradient of the loss function $L(y, y^*)$ w.r.t. $w_{22}^{(1)}$:

$$\frac{\partial L(y^*, y)}{\partial w_{22}^{(1)}} = \frac{\partial L}{\partial y^*} \frac{\partial y^*}{\partial z_2} \frac{\partial z_2}{\partial w_{22}^{(1)}}$$

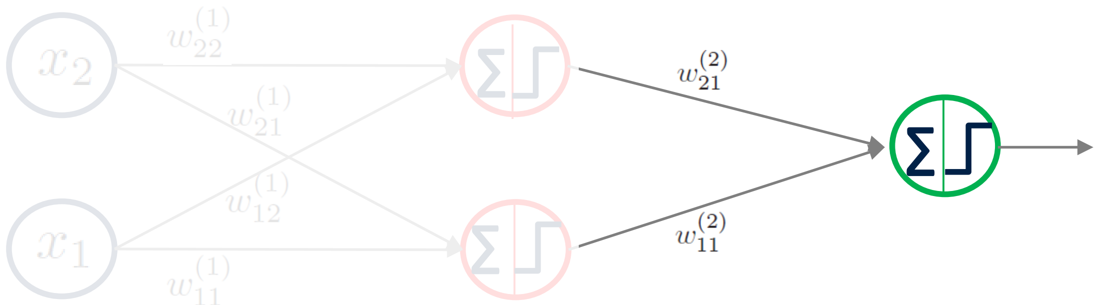
Backpropagation: 3 layer network



- Gradient of the loss function $L(y, y^*)$ w.r.t. $w_{22}^{(1)}$:

$$\frac{\partial L(y^*, y)}{\partial w_{22}^{(1)}} = \underbrace{\frac{\partial L}{\partial y^*}\frac{\partial y^*}{\partial z_2^{(2)}}\frac{\partial z_2^{(2)}}{\partial z_2^{(1)}}\frac{\partial z_2^{(1)}}{\partial w_{22}^{(1)}}} + \underbrace{\frac{\partial L}{\partial y^*}\frac{\partial y^*}{\partial z_1^{(2)}}\frac{\partial z_1^{(2)}}{\partial z_2^{(1)}}\frac{\partial z_2^{(1)}}{\partial w_{22}^{(1)}}}$$

Looking backwards....



- Derivative of the loss function with respect to weights:

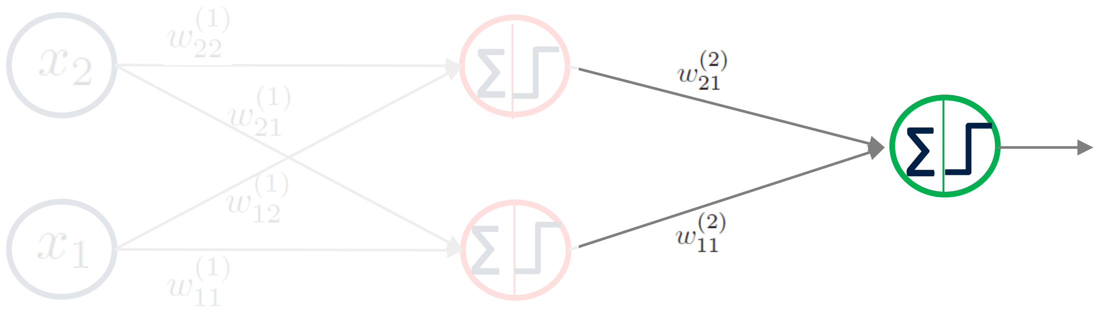
$$\frac{\partial L}{\partial w_{j1}^{(2)}} = \frac{\partial\big(-y\log(y^*) - (1-y)\log(1-y^*)\big)}{\partial w_{j1}^{(2)}}$$

$$= \left(-\frac{y}{y^*} + \frac{(1-y)}{(1-y^*)}\right)\frac{\partial \mathcal{A}\big(\sum_{j=1}^{2} w_{j1}^{(2)} z_j\big)}{\partial w_{j1}^{(2)}}$$

$$= \left(\frac{y^*-y}{y^*(1-y^*)}\right)\mathcal{A}'\big(\mathbf{v}_1^{(2)}\big)z_j \qquad \text{where } \mathbf{v}_1^{(2)} = \big(\mathbf{w}^{(2)}\big)^{\mathrm{T}}\mathbf{z}$$

$$= (y^* - y)z_j$$

Looking backwards: second layer



- Weight update at the second layer:

$$w_{j1}^{(2)} := w_{j1}^{(2)} - \xi \frac{\partial L}{\partial w_{j1}^{(2)}}$$

$$:= w_{j1}^{(2)} - \xi \Big(y^* - y \Big) z_j$$

Gradient based optimization

- Gradient descent:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \xi \sum_{n=1}^{N} \nabla_{\mathbf{w}^{(t)}} L\big(y^{(n)}, y^{*(n)}(\mathbf{w}^{(t)})\big)$$

  where ξ is the learning rate.

- Frequency of updates:
  - Batch gradient descent: Updates after evaluating the loss gradient w.r.t. all training examples.
  - Stochastic gradient descent: Updates after evaluating the loss gradient w.r.t. every training example.
  - Mini-batch gradient descent: Updates after evaluating the loss gradient w.r.t. a subset of the training dataset.

Gradient based optimization

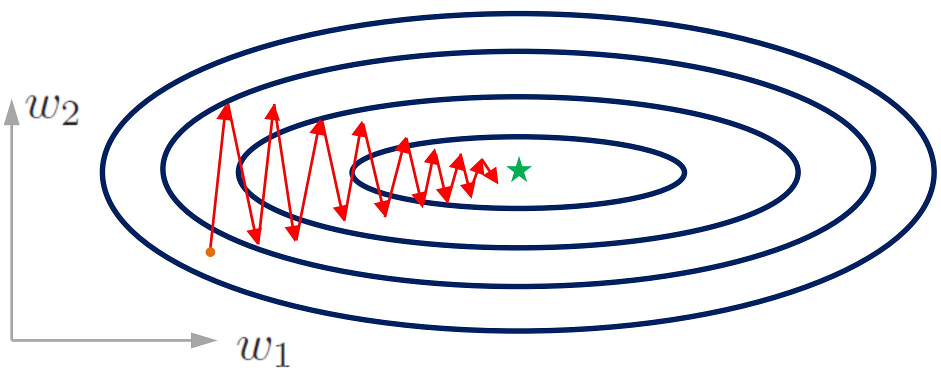
- Gradient descent:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \xi \sum_{n=1}^{N} \nabla_{\mathbf{w}^{(t)}} L\big(y^{(n)}, y^{*(n)}(\mathbf{w}^{(t)})\big)$$

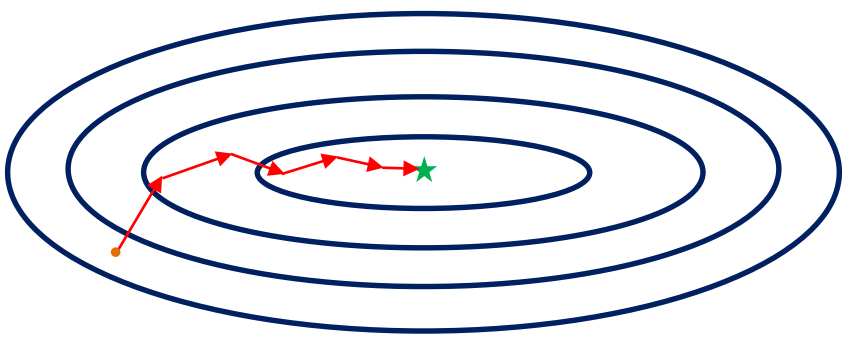
  where ξ is the learning rate.

- Type of updates:
  - Fixed learning rate
  - With momentum
  - Adaptive learning rate
  - Adaptive learning rate + Momentum

Gradient descent



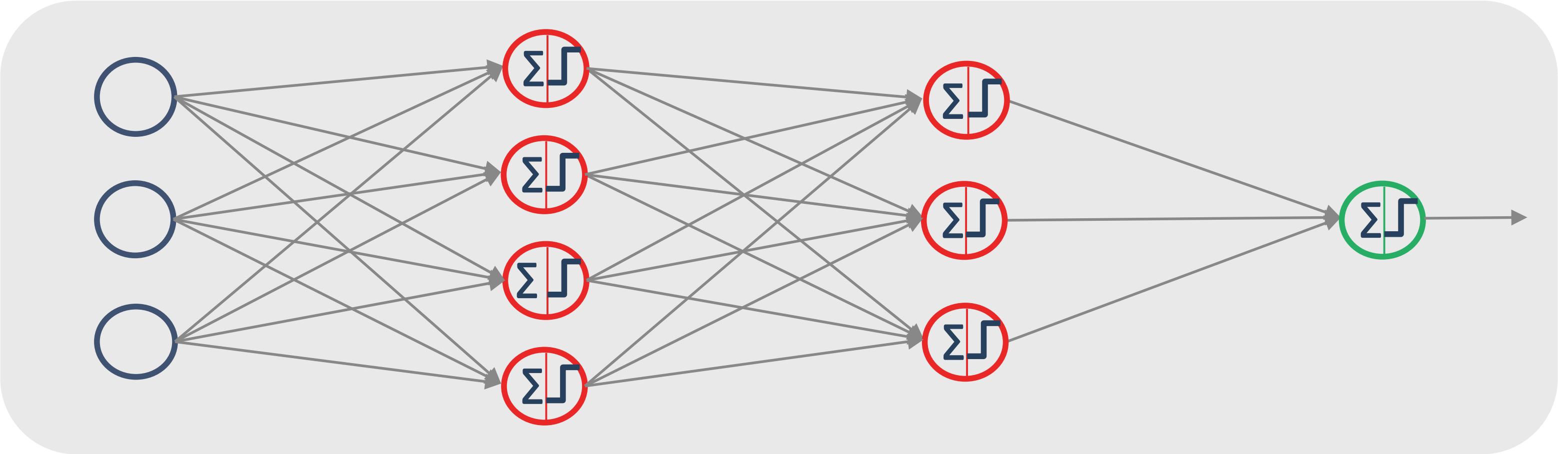
Gradient descent
with momentum



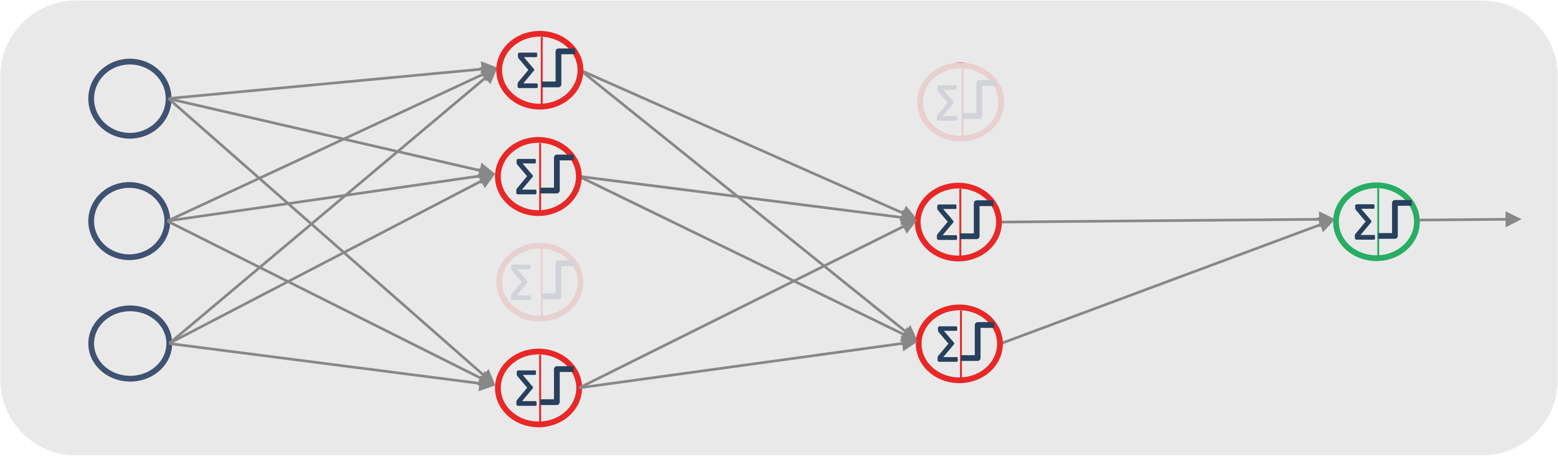
Figures for illustration only

Dropout: Network architecture

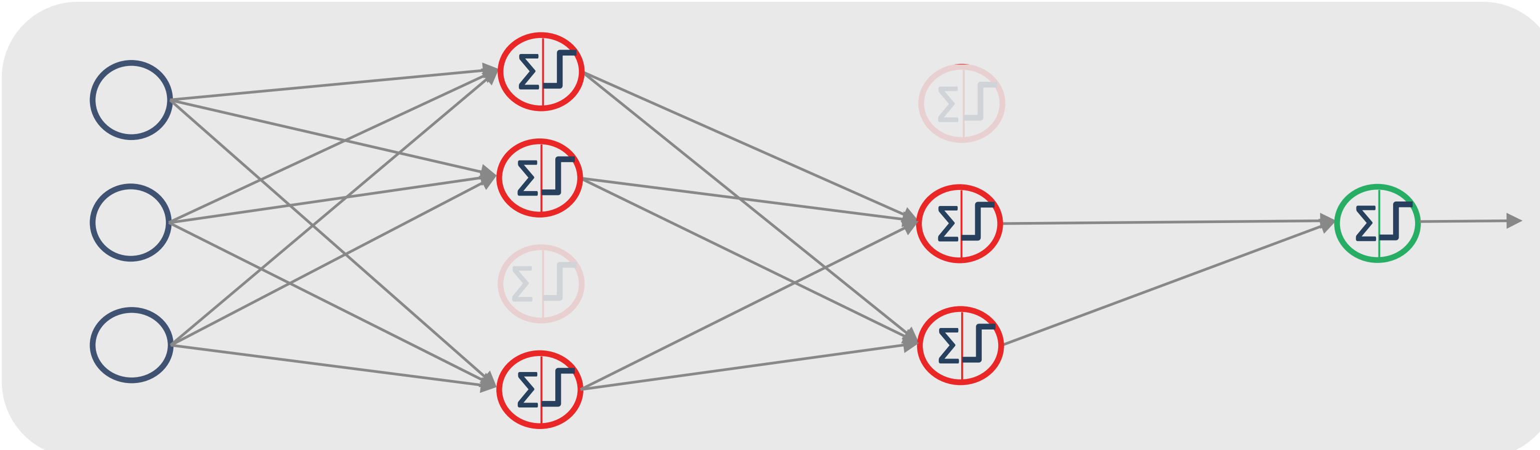
- Standard network



- Dropout network

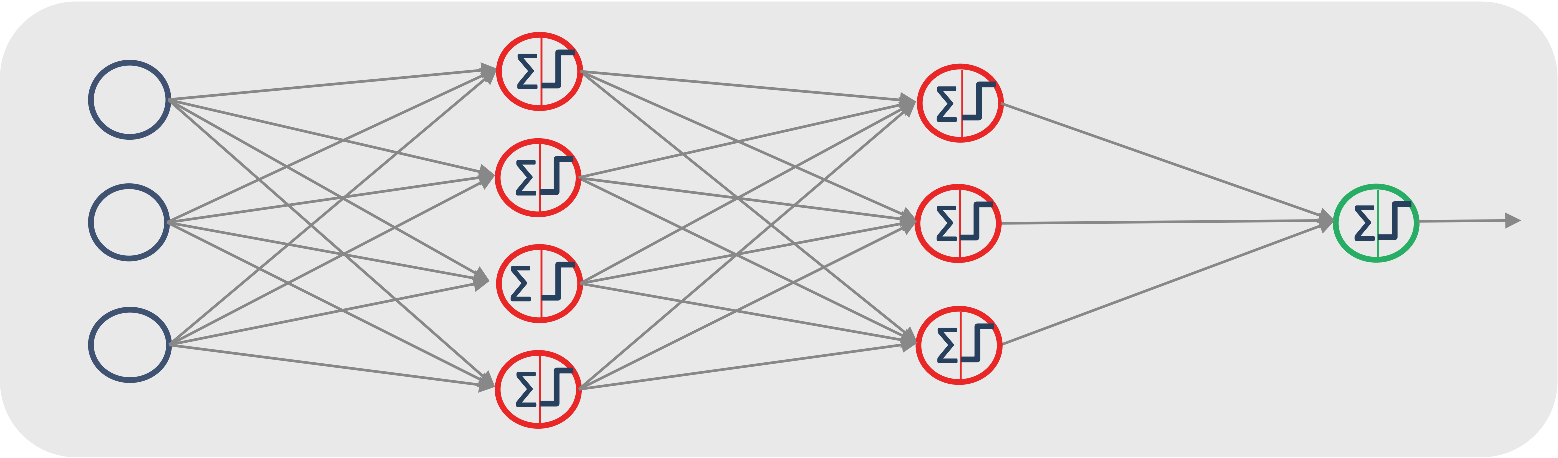


Dropout: Training using SGD



- Training dropout networks using stochastic gradient descent.
- For a particular mini-batch
  - a thinned network is sampled for each training example.
  - the gradient of a parameter is obtained by averaging over its gradients from all the training cases in the mini-batch.
  - parameters which are absent in a training case (due to dropout) contribute a value of zero in the average gradient computations.

Dropout: Test



- Not convenient to make predictions with all the (exponentially many) dropout networks, and then compute the average prediction.
- Idea: Use a single network without dropout.
  - The weights of this network are reduced by some factor.
  - During training, if a unit is kept with probability p, then during test the outgoing weights of that unit are multiplied by p.
- The approach ensures that for test runs the expected output of any unit is the same as the actual output.

References

- I. Goodfellow, Y. Bengio, A. Courville, Y. Bengio, "Deep Learning," *MIT Press*, 2016.
- Y. LeCun, Y. Bengio, G, Hinton, "Deep Learning," *Nature*, 521(7553): 436–444, 2015.