Today

Perceptron (including implementation)

→ Extension from one to multiple units

→ Recitation (scikit learn part)

→ Neural Network Training (including loss + back propagation)

→ Neural Network from scratch in numpy

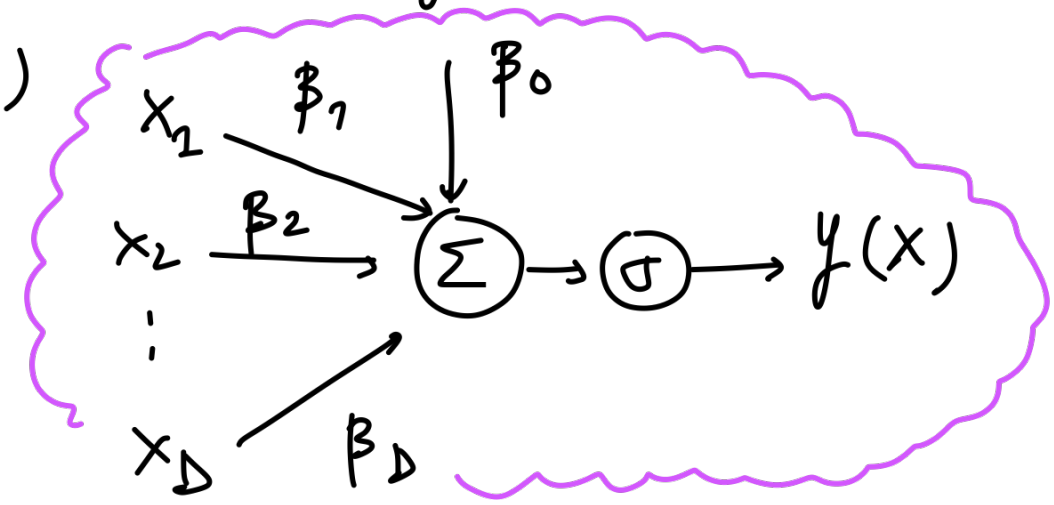
So far we have covered a collection of classifiers which can all read as the general form "activation + linear model"

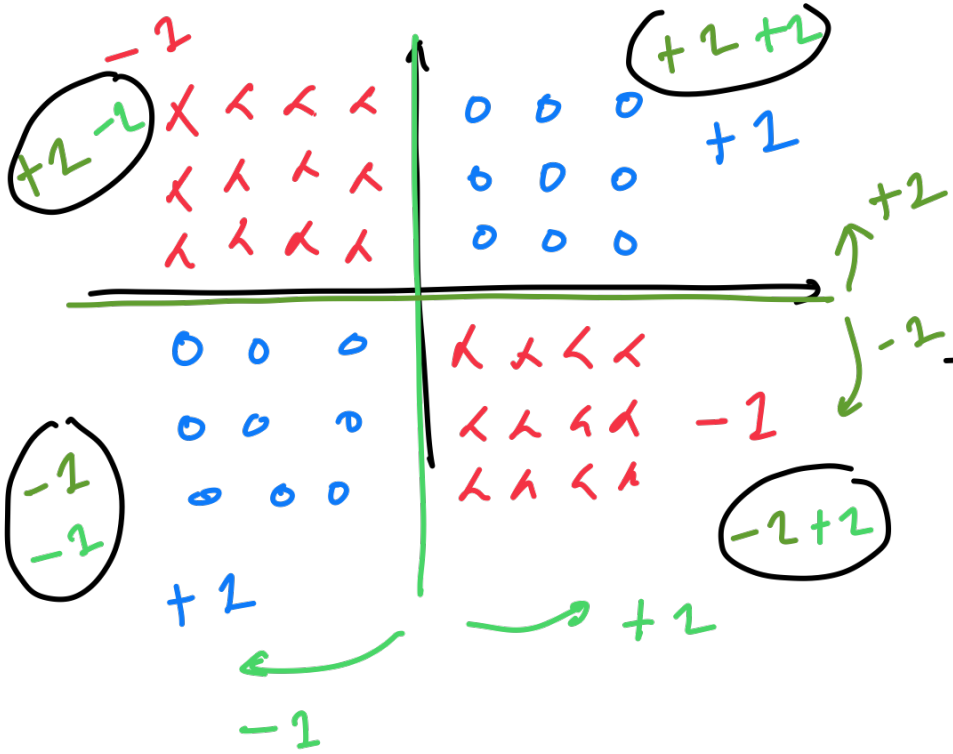
→ linear model : $\sigma(\beta_0 + \beta_1 x_1 + \cdots + \beta_D x_D)$ w/ $\sigma(x) = x$

→ logistic regression $\sigma(\beta_0 + \beta_1 x_1 + \cdots + \beta_D x_D)$ w/ $\sigma(x) = \frac{1}{1+e^{-x}}$

→ perceptron $\sigma(\beta_0 + \beta_1 x_1 + \cdots + \beta_D x_D)$ w/ $\sigma(x) = \begin{cases} +1 & x > 0 \\ -1 & x < 0 \end{cases}$

→ general structure can be captured by a simple diagram (a.k.a neuron)





XOR dataset

→ first neuron $y_1(x)$

$$y_1(x) = \sigma(\beta_0 + \beta_1 x_1 + \beta_2 x_2)$$

$$= \sigma(x_2) = \begin{cases} +2 & \text{if } x_1 > 0 \\ -2 & x_1 < 0 \end{cases}$$

→ second neuron $y_2(x)$

$$y_2(x) = \sigma(\beta_0 + \beta_1 x_1 + \beta_2 x_2)$$

$$= \sigma(x_2) = \begin{cases} +1 & x_2 > 0 \\ -1 & x_2 < 0 \end{cases}$$

→ third neuron $y_3(x)$

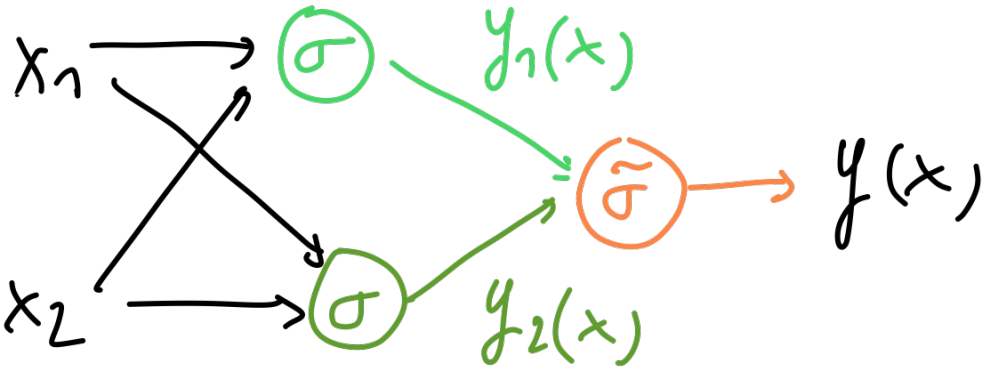
$$y_3(x) = \tilde{\sigma}(\beta_0 + \beta_1 y_1 + \beta_2 y_2)$$

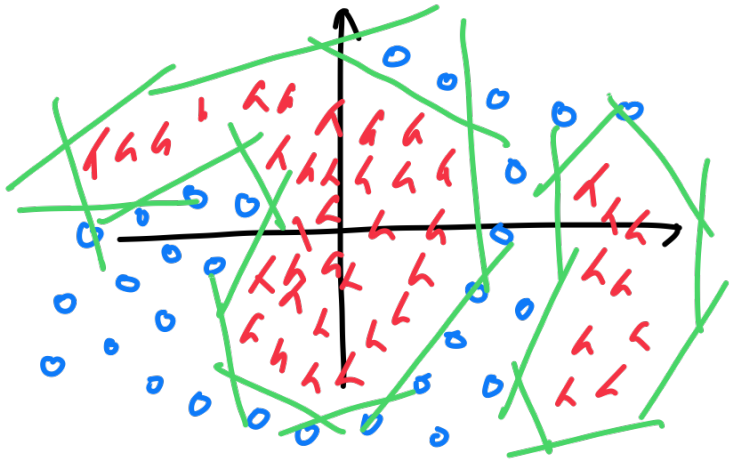
$$= \tilde{\sigma}(y_1(x) + y_2(x))$$

with for example $\tilde{\sigma}(x) = x^2$

→ Full network defined as

$$y(x) = \tilde{\sigma}\left(\sigma(x_2) + \sigma(x_1)\right) = \begin{cases} 0 & \text{for } x_1 > 0 \; x_2 > 0 \\ & x_2 < 0 \; x_1 < 0 \\ 4 & \text{for } x_1 > 0 \; x_1 < 0 \\ & x_2 > 0 \; x_2 < 0 \end{cases}$$





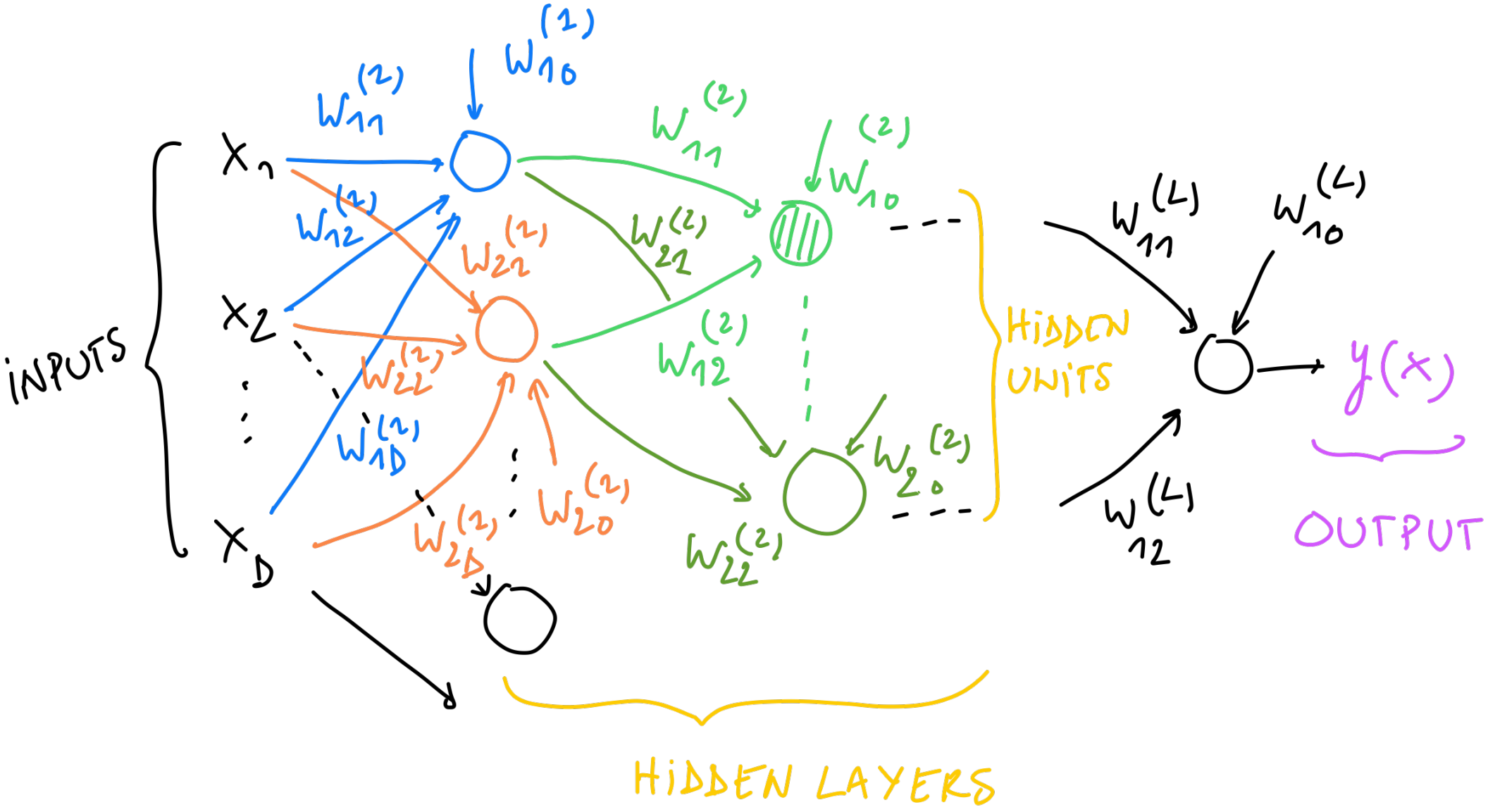
Backpropagation

→ We consider a simple binary classification setting (just like for logistic regression except that this time our class probability is defined by the network itself and not the simpler logistic regression model.)

$$p(t^{(i)}=1 \mid x^{(i)}) = y(x^{(i)})$$

$$p(t^{(i)}=0 \mid x^{(i)}) = 1 - y(x^{(i)})$$

with $y(x)$ encoding the network.



FEED FORWARD FULLY CONNECTED NEURAL NETWORK

k^{th} unit from layer l can be encoded by a combination "linear combination + activation function" as

$$a_k^{(l)} = \sum_{j=1}^{N_{l-1}} w_{kj}^{(l)} z_j^{(l-1)} + w_{k0}^{(l)} \rightarrow \text{"pre-activation"}$$

$$z_k^{(l)} = \sigma(a_k^{(l)}) \rightarrow \text{"post-activation"}$$

→ the network's weights can be learned (just as we did for logistic regression) by maximizing the likelihood defined on the training set $\{x^{(i)}, t^{(i)}\}_{i=1}^{N}$

PROBABILITY of observing the training set (provided training examples are i.i.d)

$$p(\{x^{(i)}, t^{(i)}\}_{i=1}^{N}) = \prod_{i=1} y(x^{(i)})^{t^{(i)}} (1 - y(x^{(i)}))^{1-t^{(i)}}$$

$$\log(p(\{x^{(i)}, t^{(i)}\}_{i=1}^{N})) = \sum_{i=1}^{N} t^{(i)} \log(y(x^{(i)})) + (1-t^{(i)}) \times \log(1-y(x^{(i)}))$$

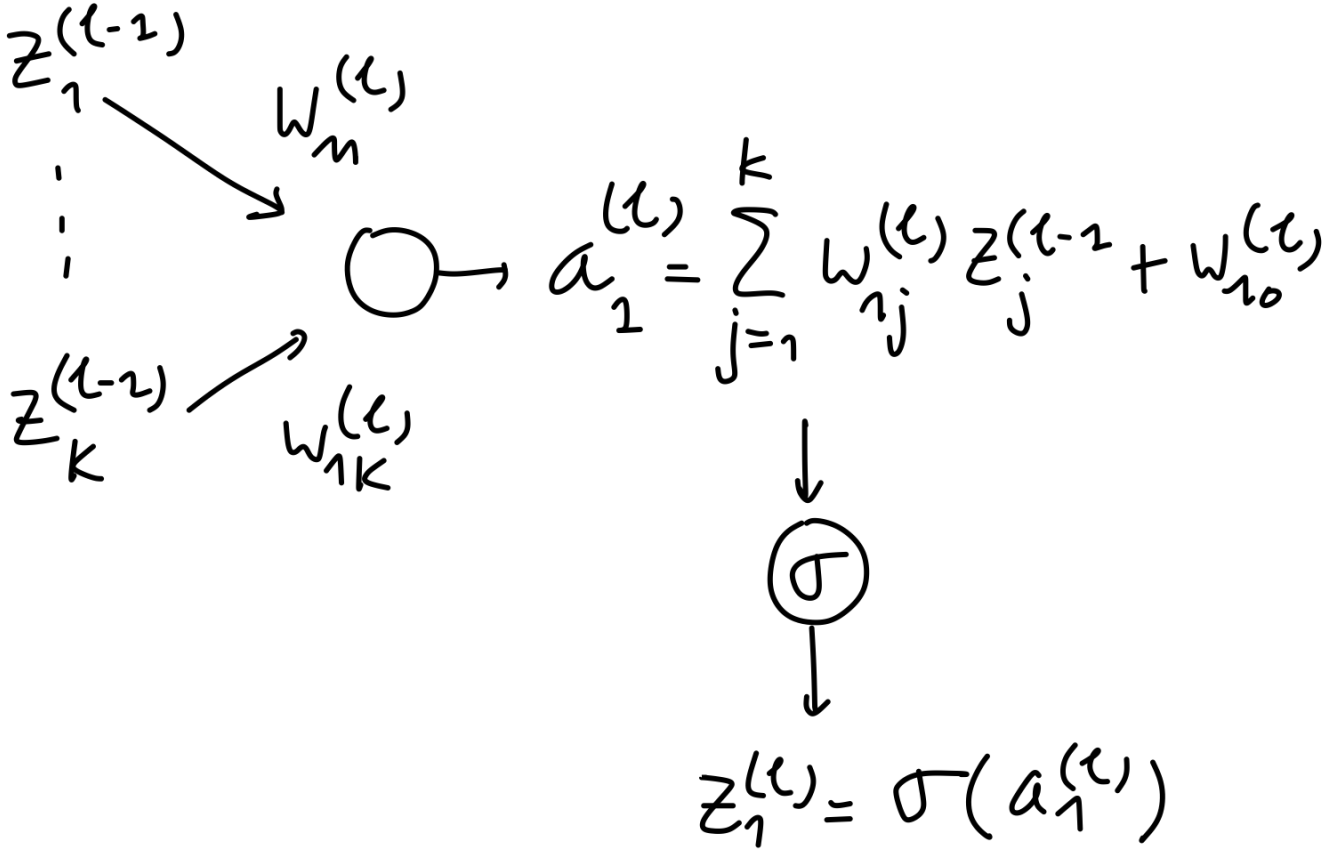
BINARY CROSS ENTROPY

LOG LOSS

→ in order to train the network through gradient descent, we then need to compute the gradient of the loss with respect to the network's parameters (i.e the weights w_{ij}

which are hidden in the expression of $y(x^{(i)})$

For any particular unit



Optimization : → Start with random $W_{ij}^{(\ell)}$

→ sup date $W_{ij}^{(\ell)}$ through

$$W_{ij}^{(\ell)} \leftarrow W_{ij}^{(\ell)} + \eta \, \text{grad loss}_{W_{ij}^{(\ell)}}$$