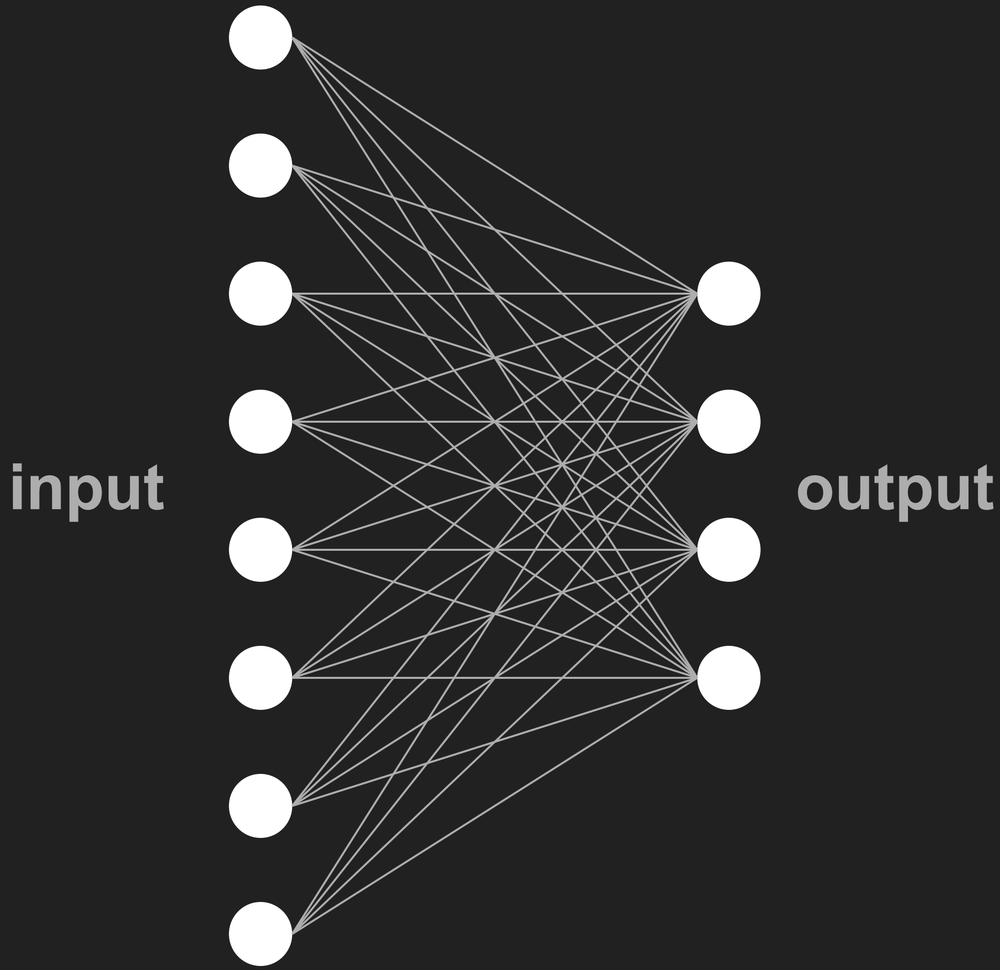
Neural networks

An introduction

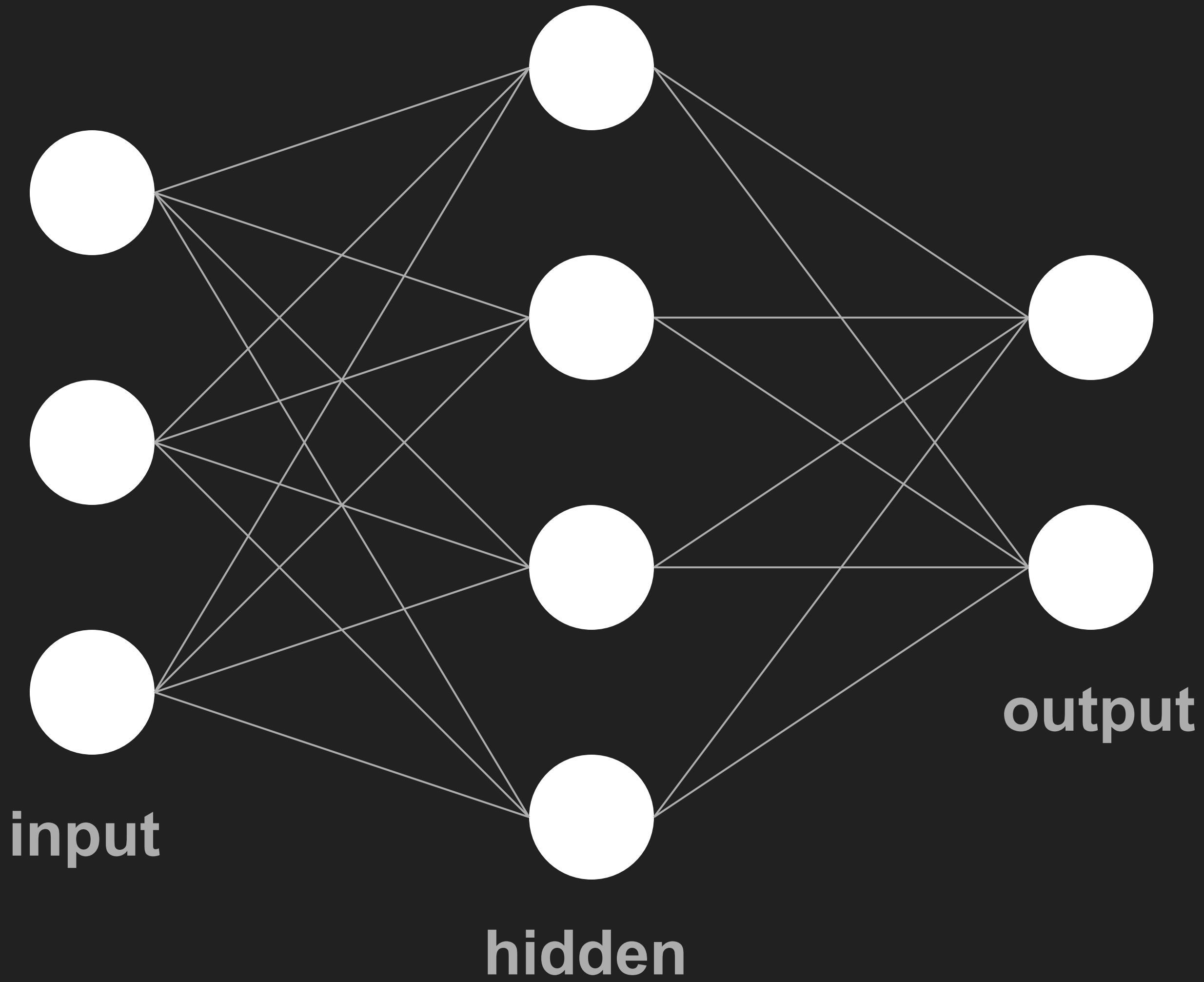
Carlos Martin and Lucas Schuermann

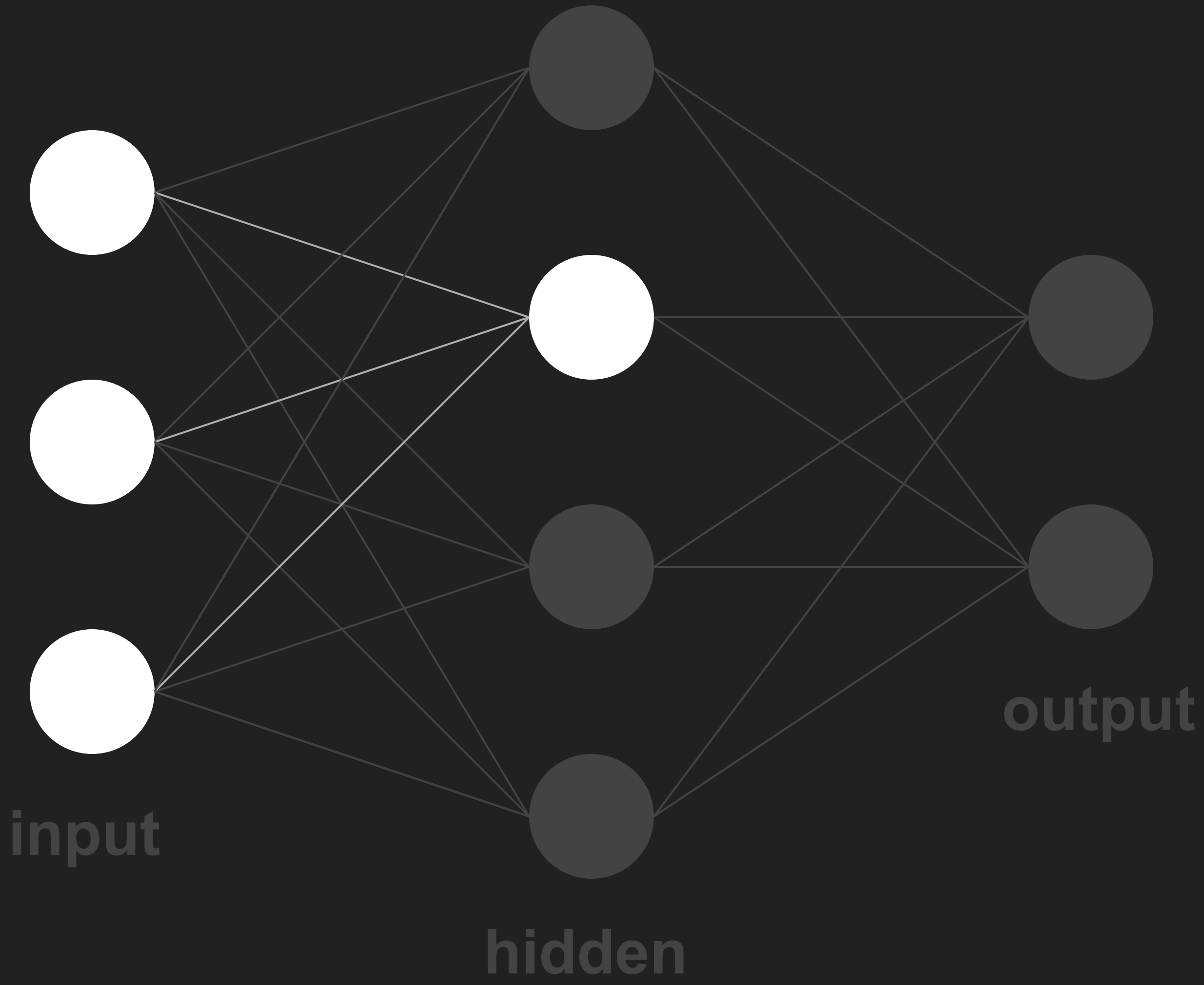
Single-layer perceptrons

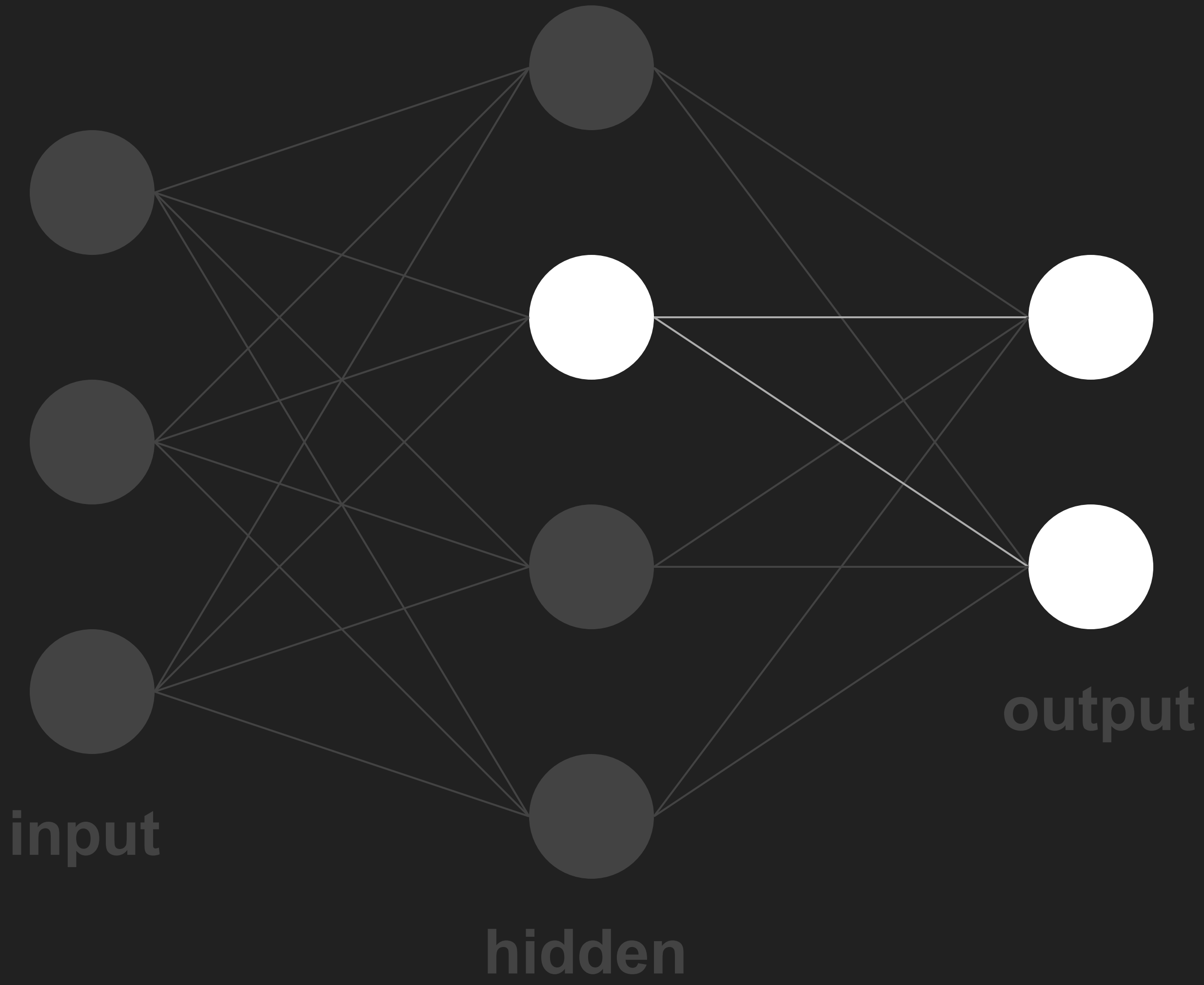

input
output

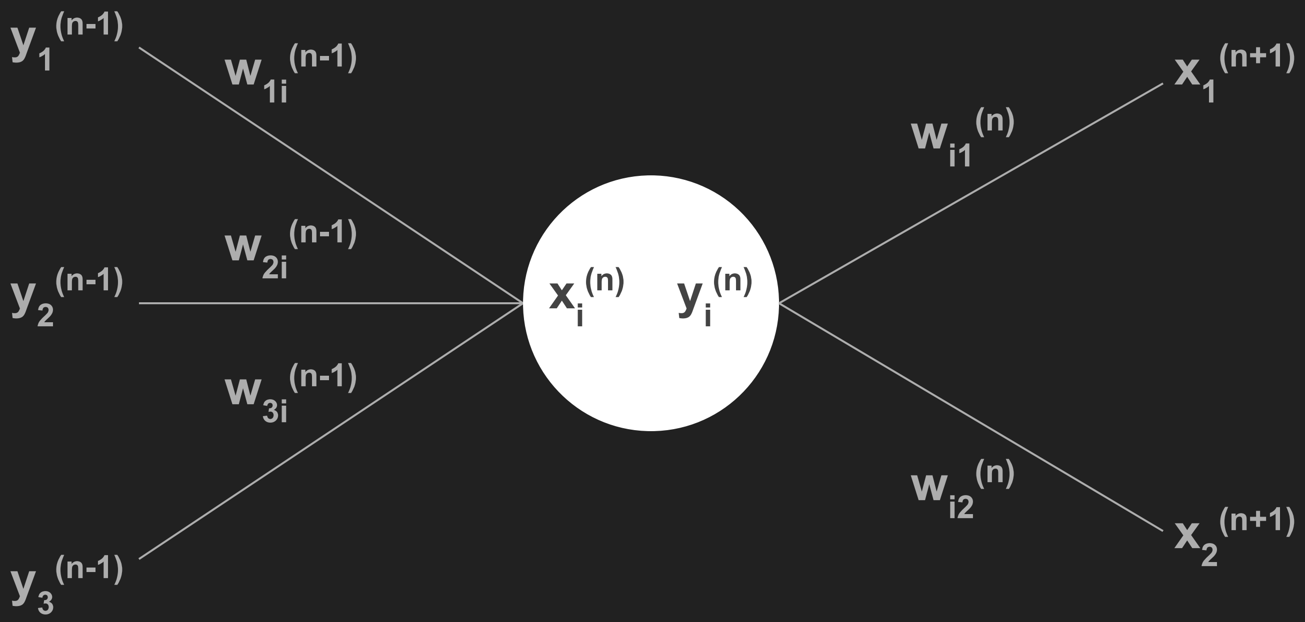
Example

Multiple layers


input
hidden
output


input
hidden
output

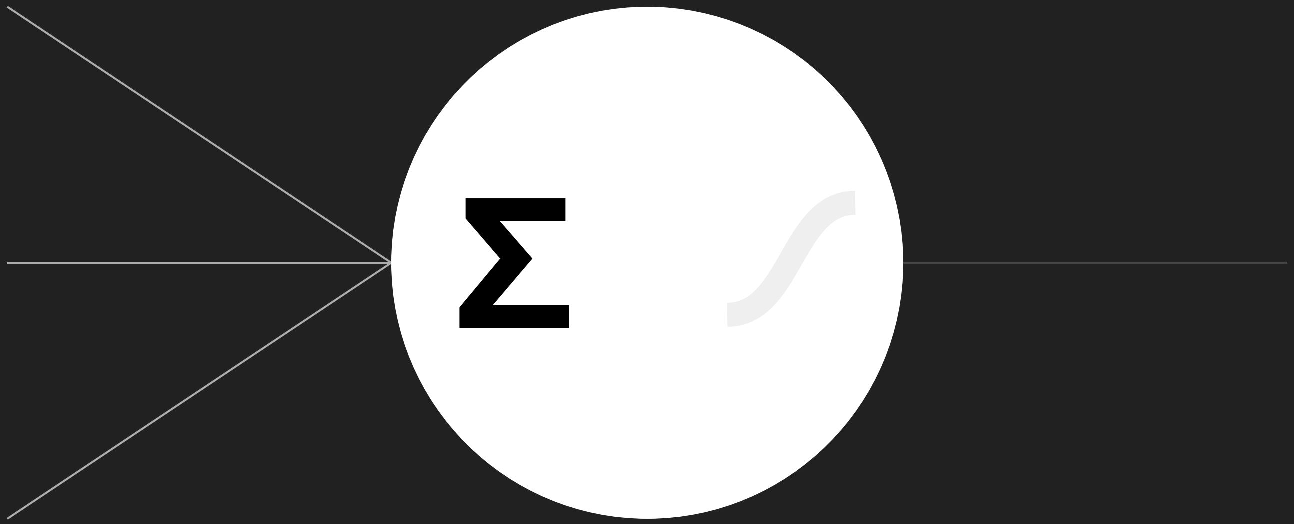

output
input
hidden



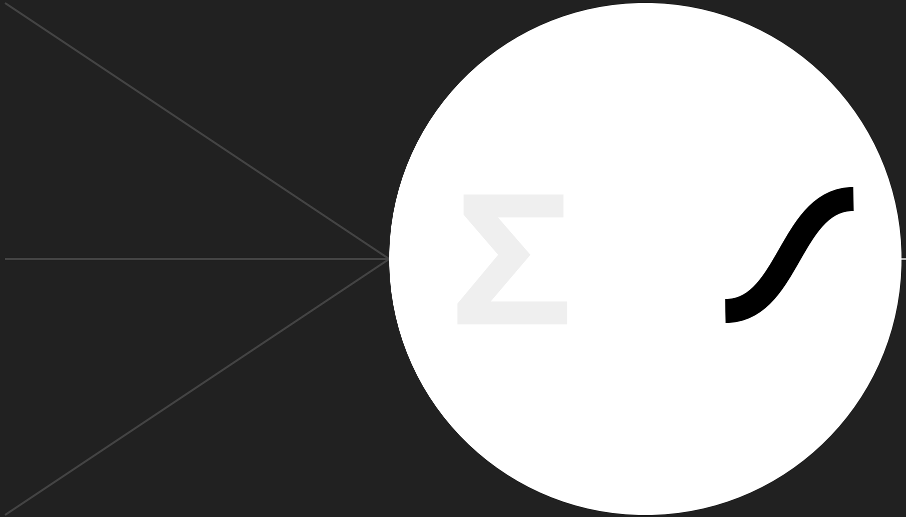
$y_1^{(n-1)}$
$w_{1i}^{(n-1)}$
$x_1^{(n+1)}$
$w_{i1}^{(n)}$
$w_{2i}^{(n-1)}$
$y_2^{(n-1)}$
$x_i^{(n)}$
$y_i^{(n)}$
$w_{3i}^{(n-1)}$
$w_{i2}^{(n)}$
$x_2^{(n+1)}$
$y_3^{(n-1)}$


Σ

$$x_i^{(n)} = \sum_j w_{ij}^{(n-1)} y_j^{(n-1)}$$



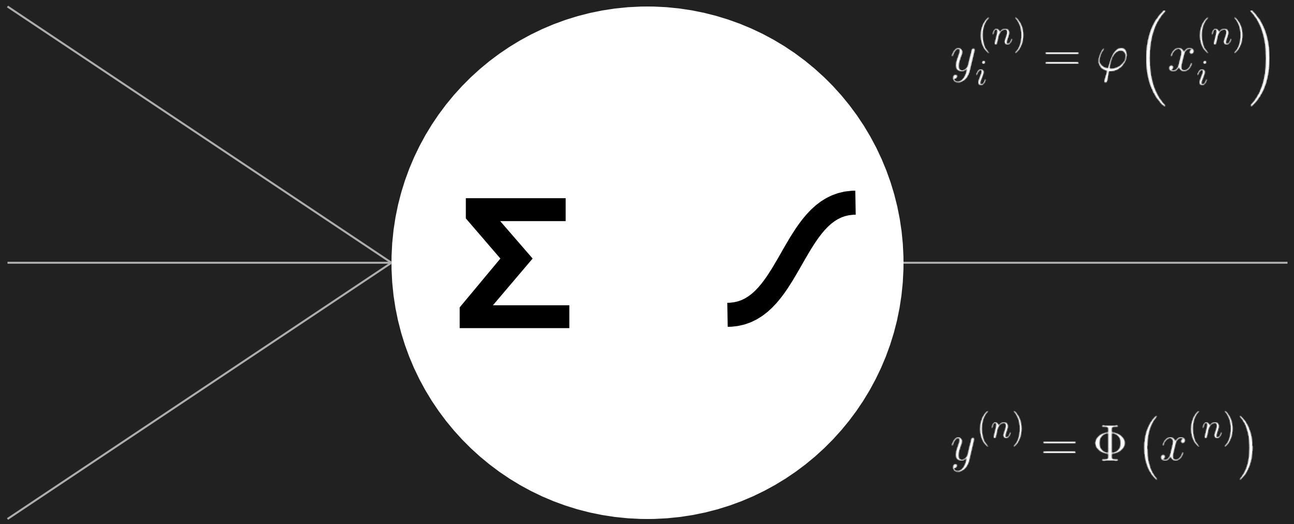
$$x^{(n)} = w^{(n-1)} \cdot y^{(n-1)}$$



$$y_i^{(n)} = \varphi\left(x_i^{(n)}\right)$$

$$y^{(n)} = \Phi\left(x^{(n)}\right)$$

$$x_i^{(n)} = \sum_j w_{ij}^{(n-1)} y_j^{(n-1)}$$

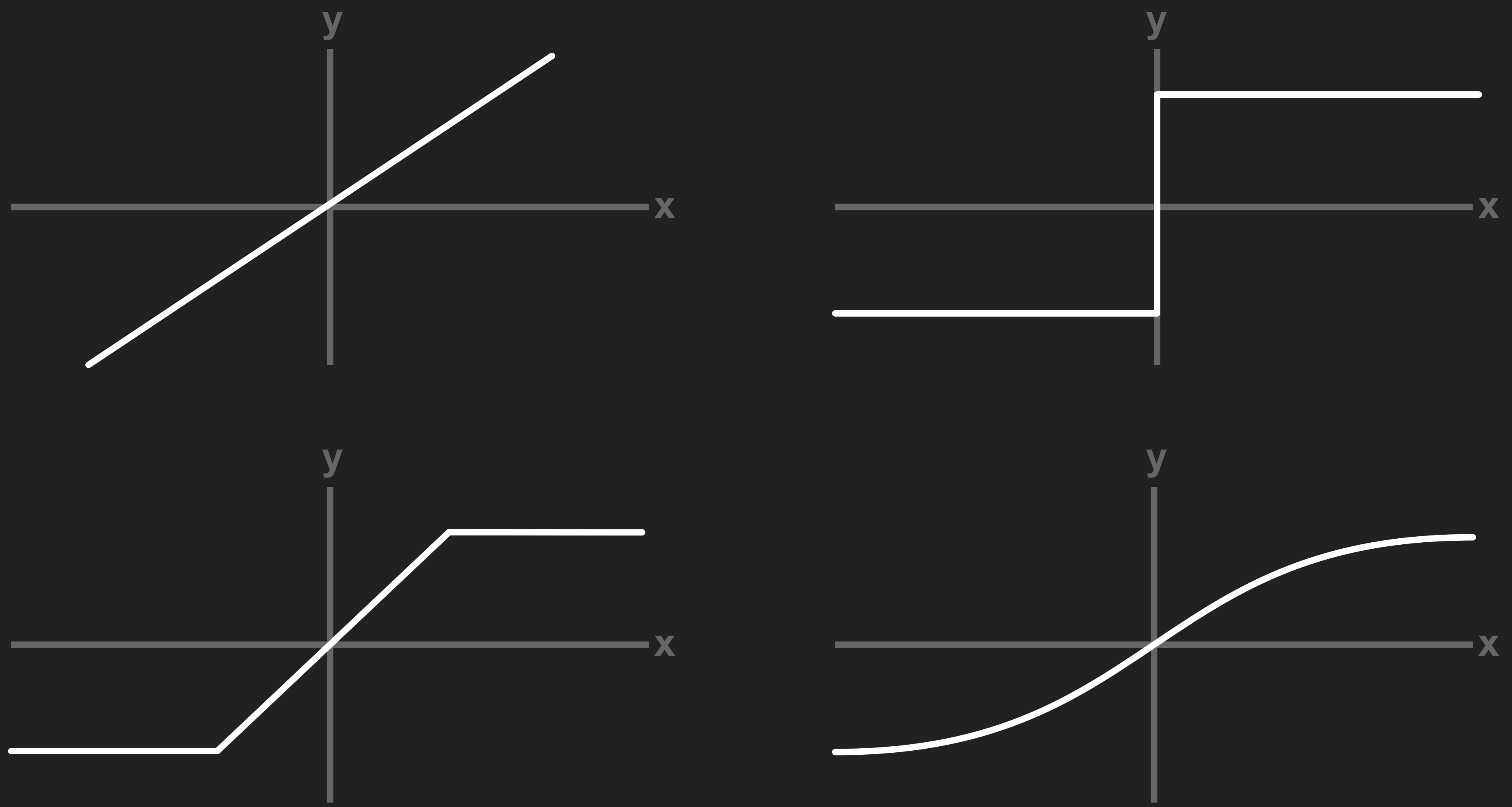


$$y_i^{(n)} = \varphi\left(x_i^{(n)}\right)$$

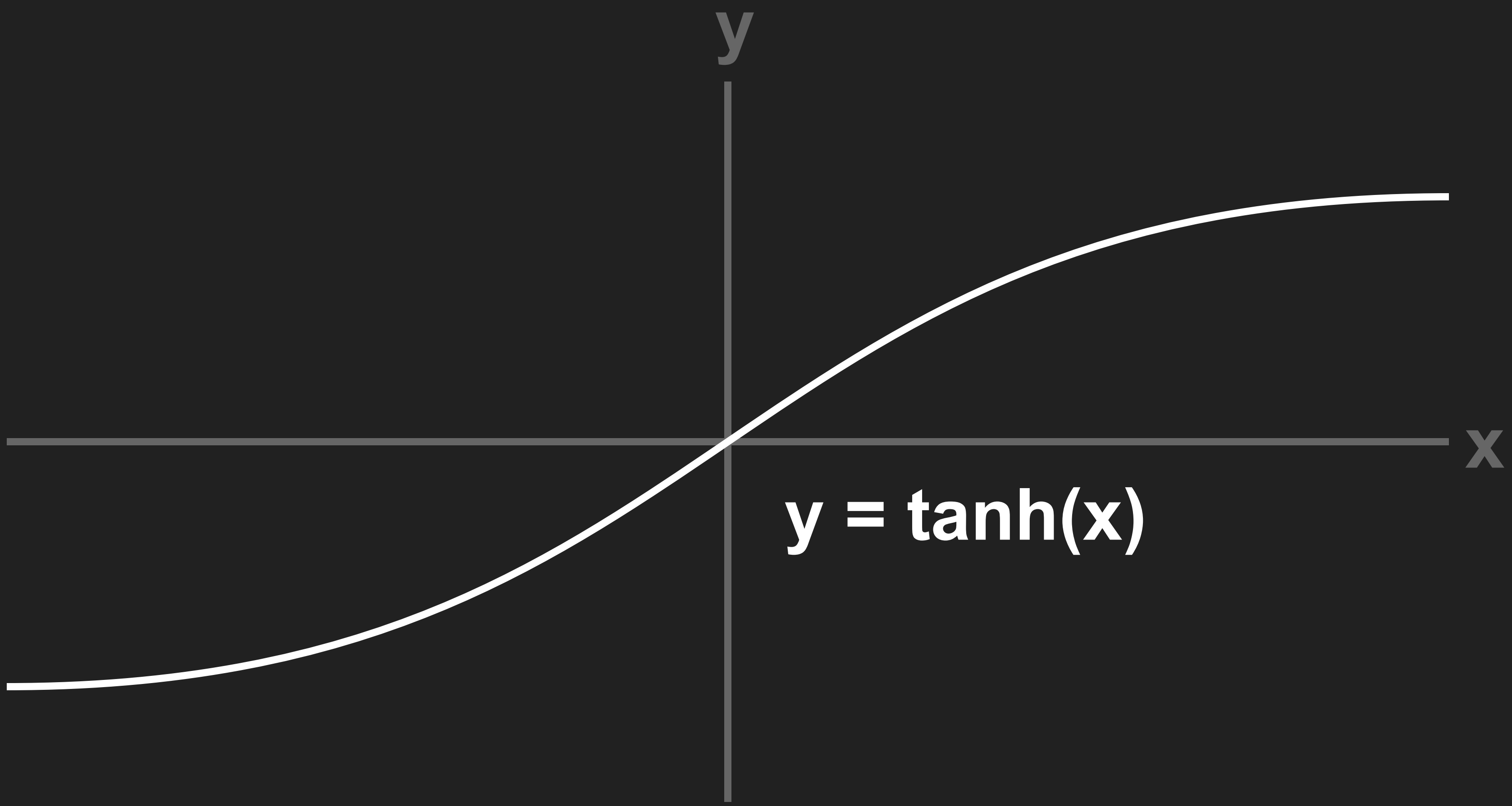
$$y^{(n)} = \Phi\left(x^{(n)}\right)$$

$$x^{(n)} = w^{(n-1)} \cdot y^{(n-1)}$$

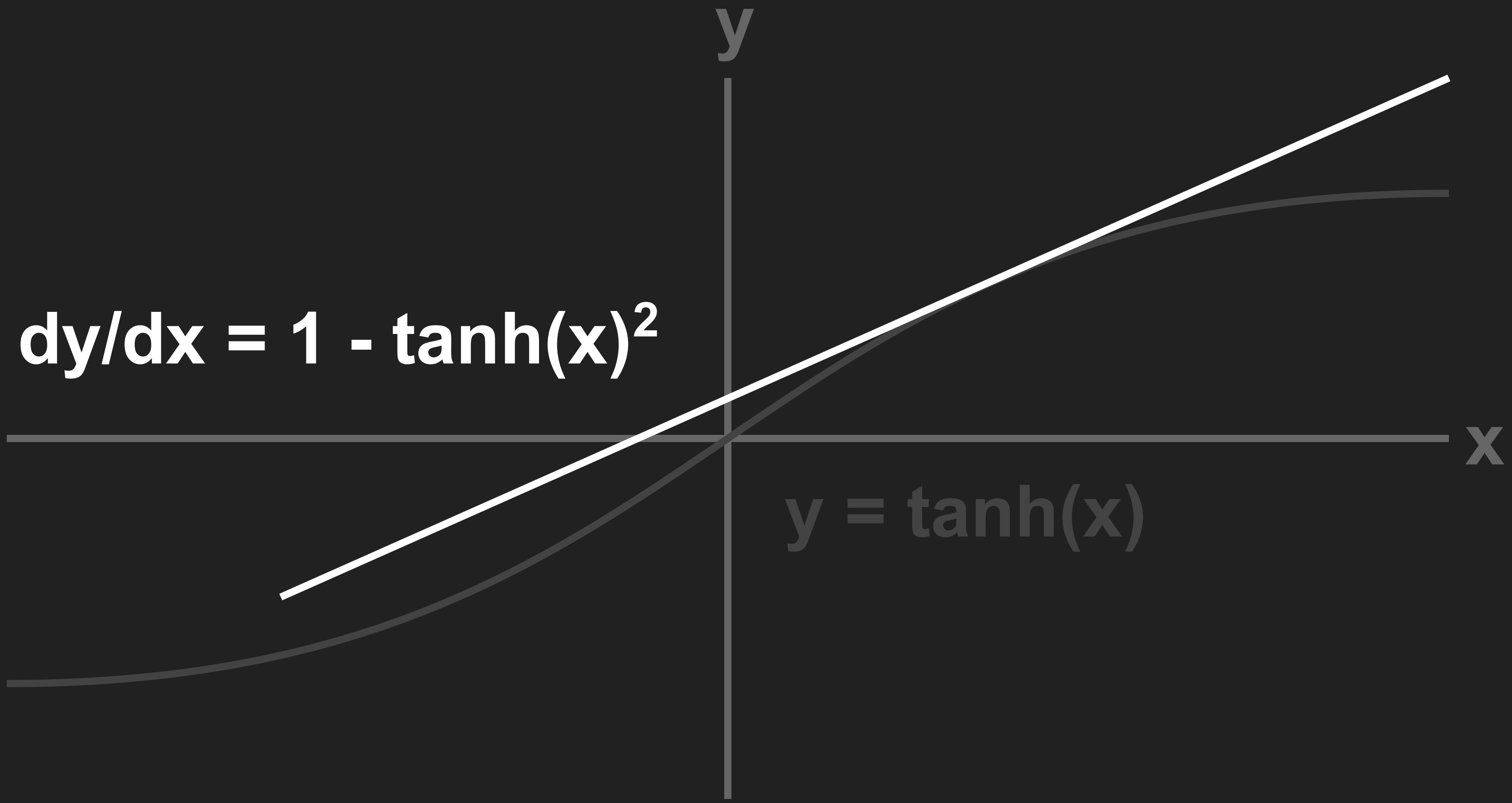
Activation functions



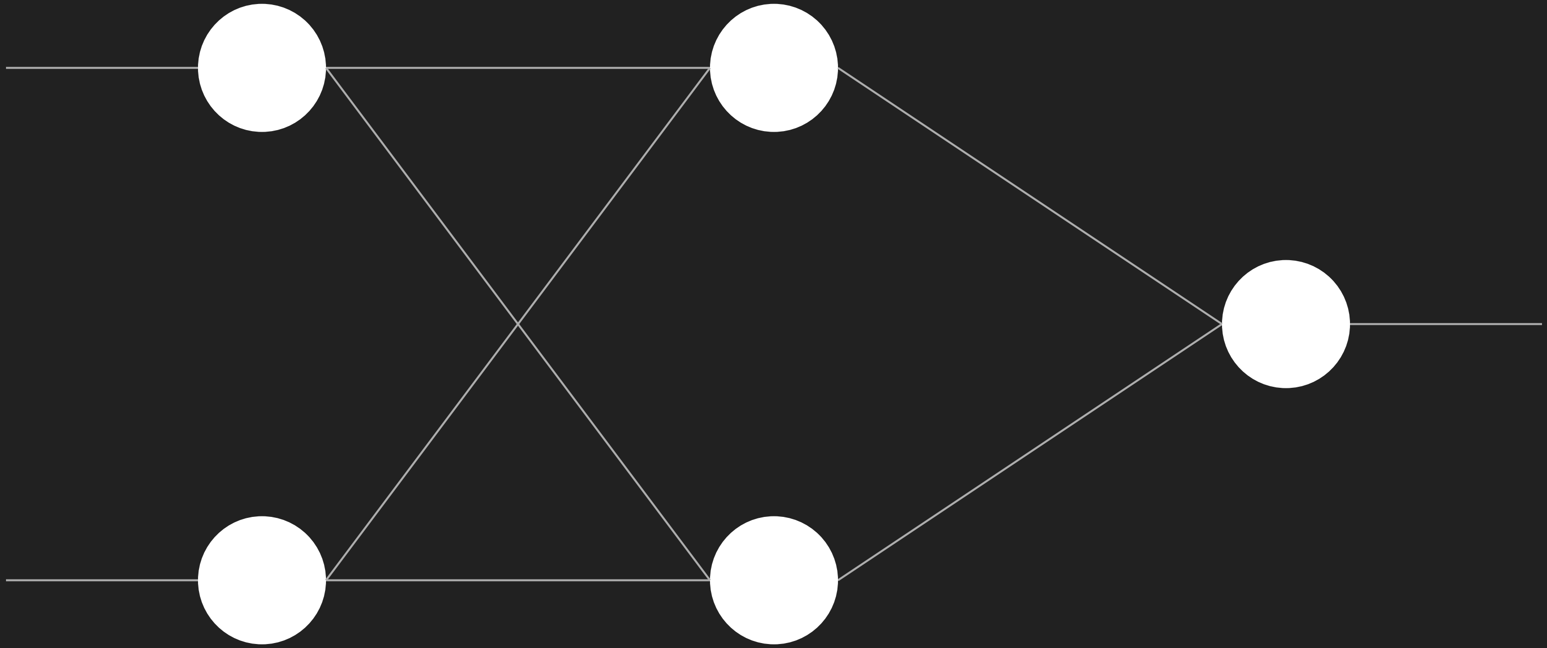
y
x
y
x
y
x
y
x



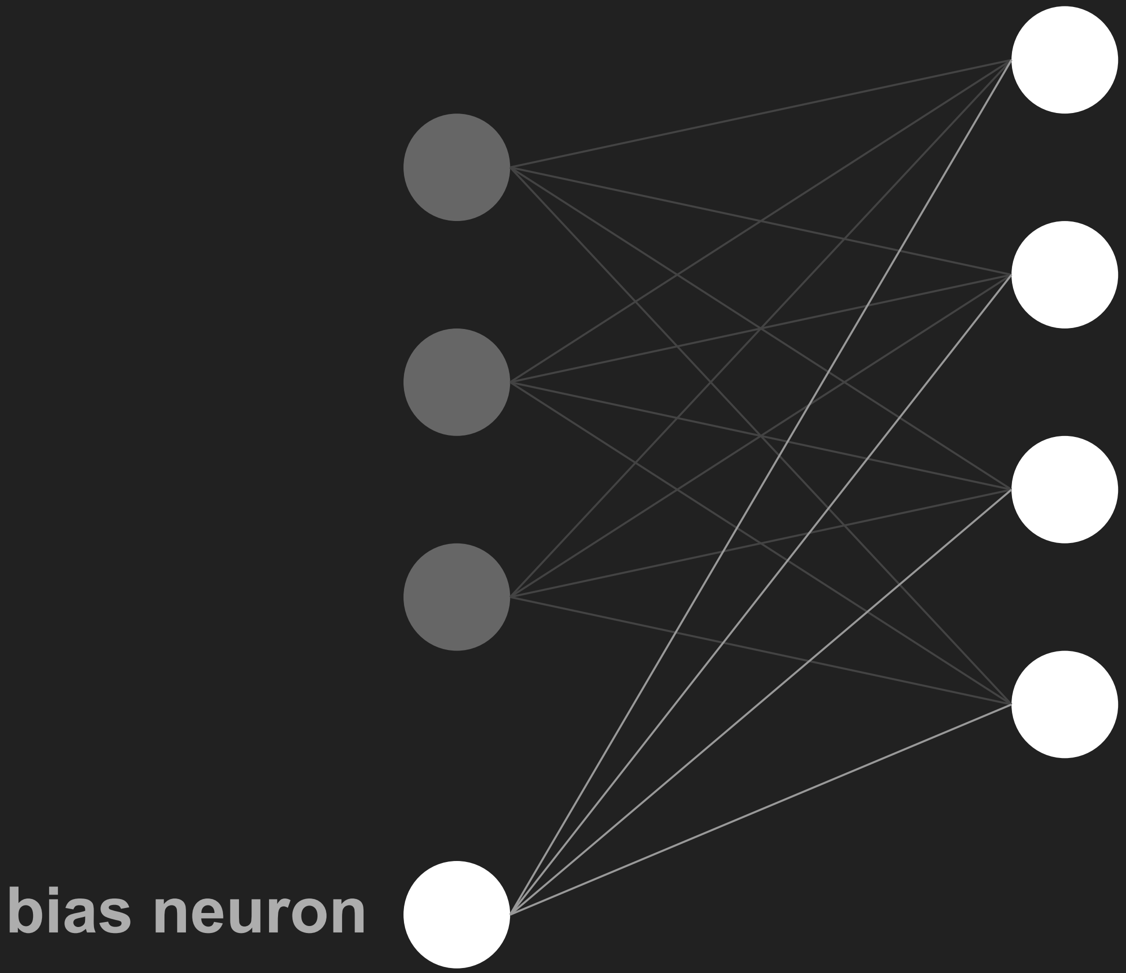
y
x
y = tanh(x)


y
$dy/dx = 1 - \tanh(x)^2$
x
$y = \tanh(x)$

XOR network



Bias neurons


bias neuron

Do we need more than 1 hidden layer?

Universal approximation theorem

Any continuous function on a compact interval can be approximated by a feed-forward neural network with a single hidden layer

For any *f* and ε there exist *a*, *b*, and *c* such that

$$g(x) = \sum_i c_i \varphi(a_i \cdot x + b_i)$$

$$|g(x) - f(x)| < \varepsilon$$

for all *x* in the interval

Training neural networks:

Backpropagation

Initializing weights

Draw from normal distribution

$$w_{ij}^{(n)} \sim \mathcal{N}(0, \sigma)$$

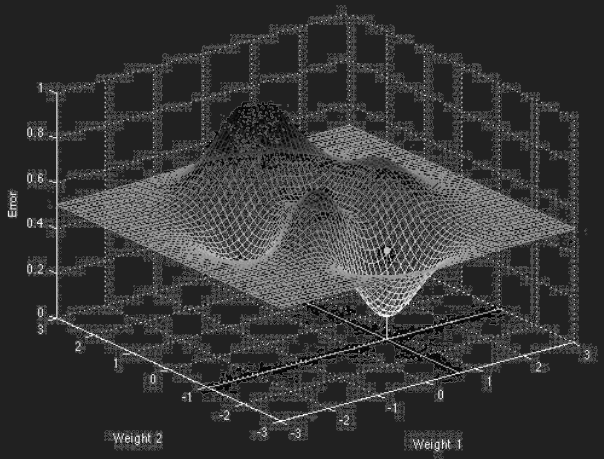
More inputs $\longrightarrow$ Less variance

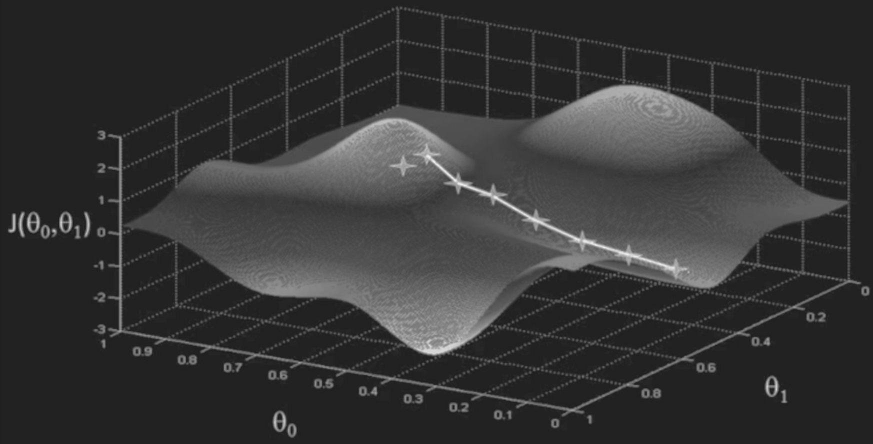
$$\sigma^2 = \frac{1}{N^{(n)}}$$

Prevents saturation

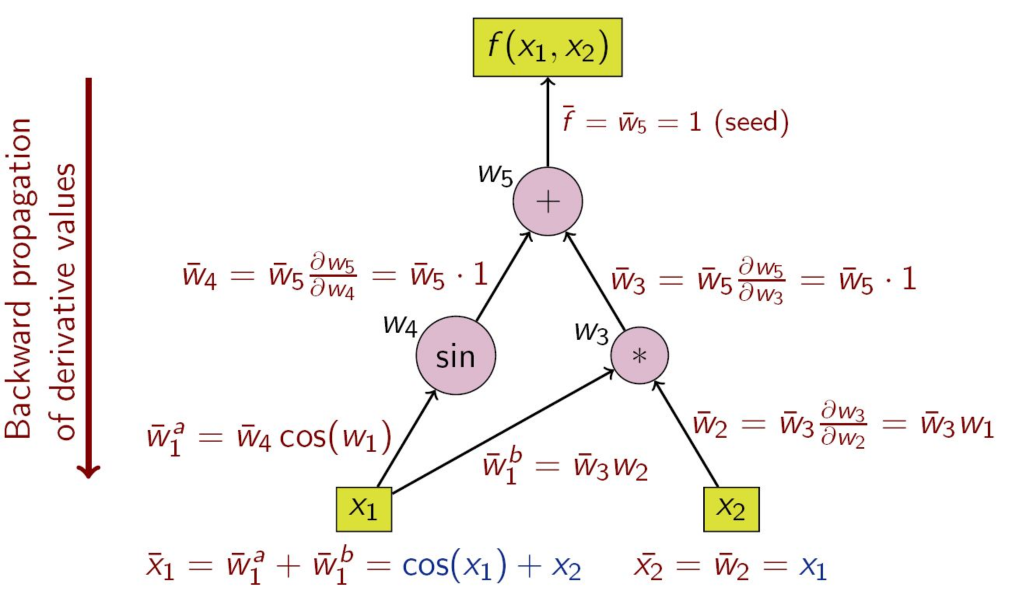
Backpropagation

Gradient descent





Automatic differentiation (chain rule)



Overall error

$$c = \frac{1}{2}(y_N - t)^2$$

Error gradient of inputs

$$\delta_n = \frac{\partial c}{\partial x_n}$$

Error gradient at last layer

$$\delta_N = (y_N - t)f'(x_N)$$

Error gradient at inner layer

$$\delta_n = \delta_{n+1} w_n f'(x_n)$$

Error gradient

$$\delta_n = f'(x_n) \begin{cases} (y_N - t) & \text{if } n = N \\ \delta_{n+1} w_n & \text{if } n < N \end{cases}$$

Error gradient of weights

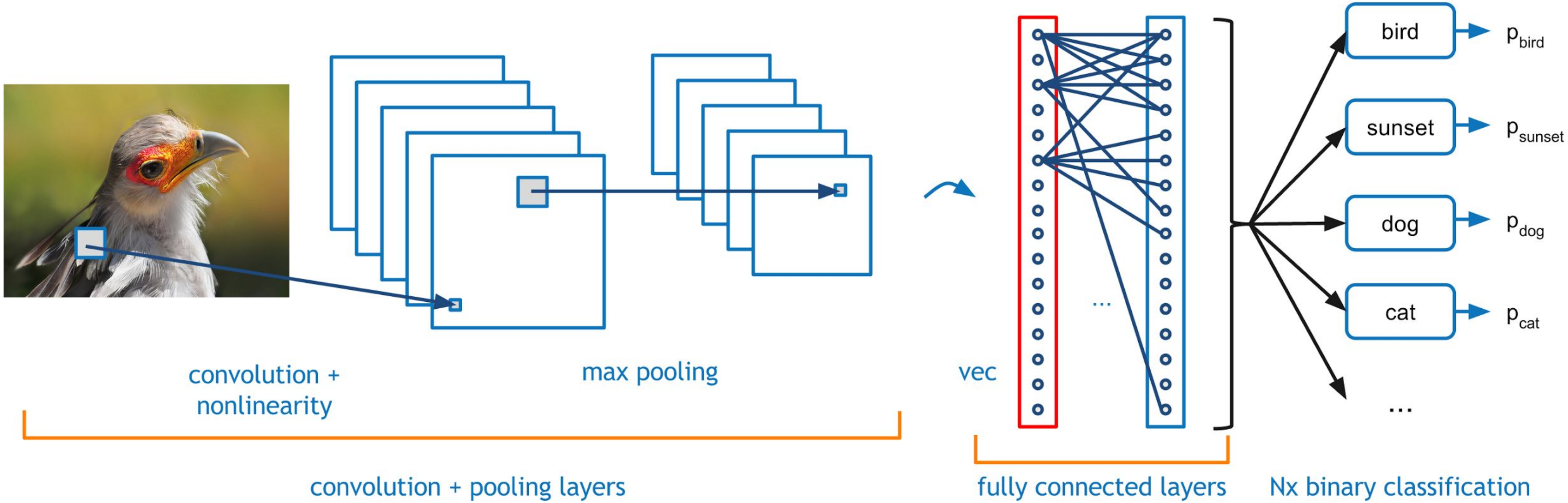
$$\frac{\partial c}{\partial w_n} = \delta_{n+1} y_n$$

Adjusting the weights (gradient descent)

$$\Delta w_n = -\alpha \delta_{n+1} y_n$$

Data sets

Convolutional networks and deep learning


convolution +
nonlinearity
max pooling
convolution + pooling layers
vec
...
fully connected layers
bird
sunset
dog
cat
...
p_{bird}
p_{sunset}
p_{dog}
p_{cat}
Nx binary classification