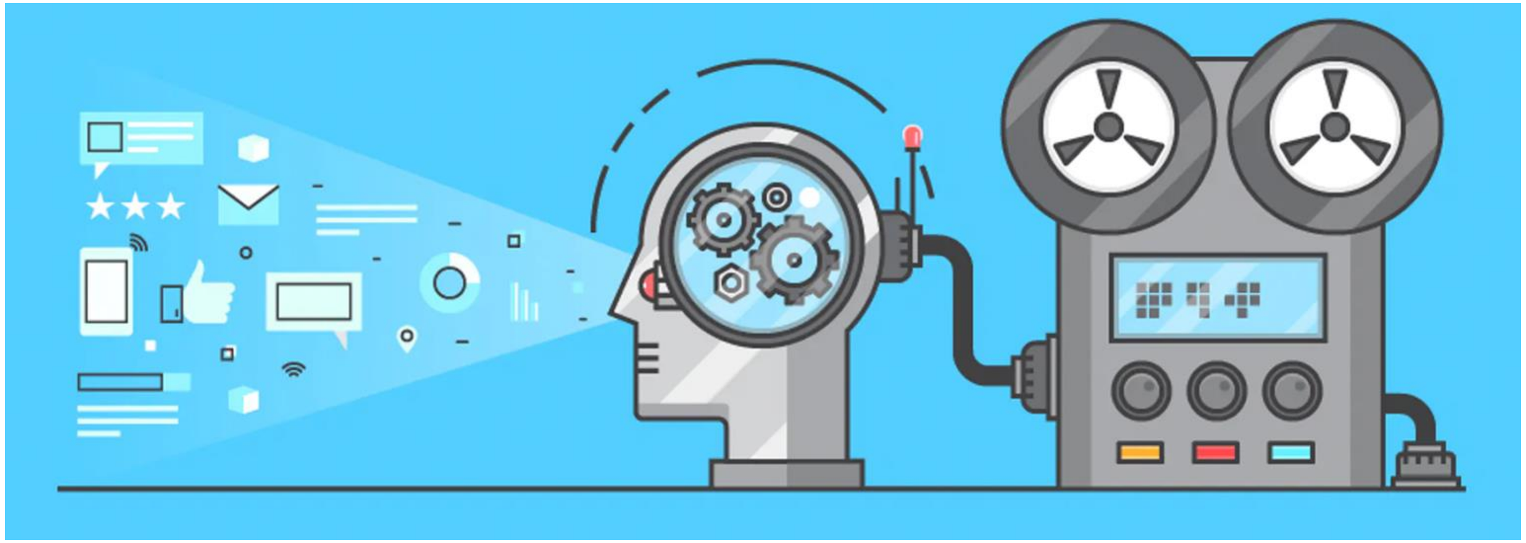
Deep Learning and Neural Networks

Topic 2:

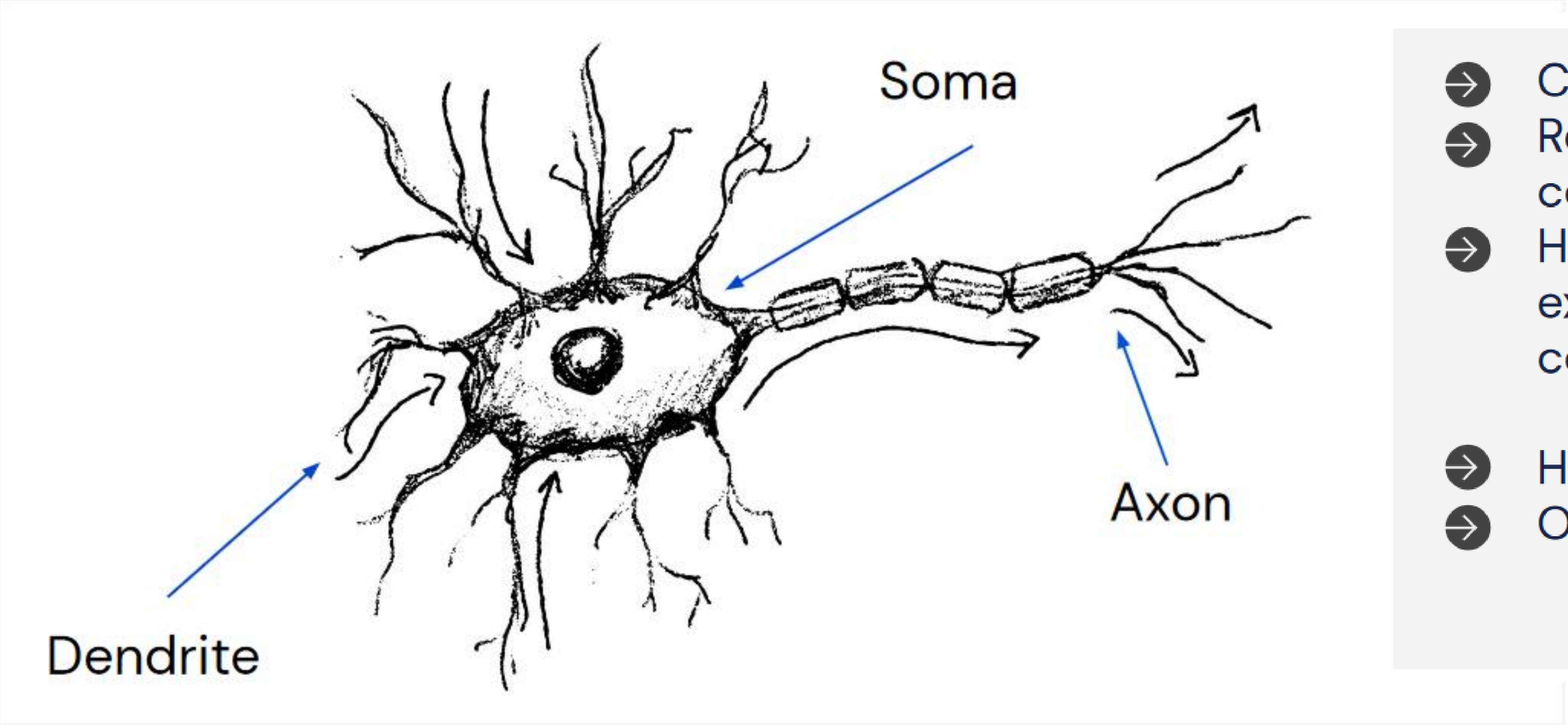
Shallow Networks



Ricardo Abel Espinosa Loera, McS

Researcher in DL & Computer Vision

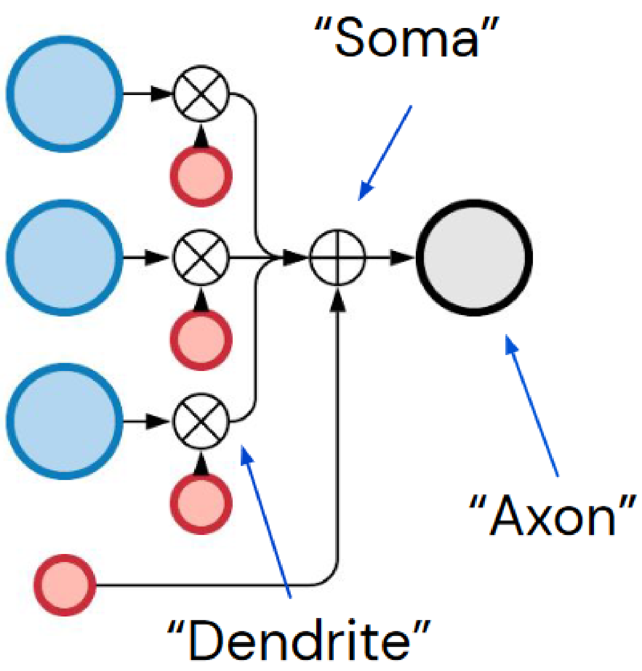
NEURAL NETWORKS
A model of a real neuorn



- Connected to others
- Represents simple computation
- Has inhibition and excitation connections
- Has a state
- Outputs spikes

NEURAL NETWORKS

A model of an artificial neuron



$$\sum_{i=1}^{d} \mathbf{w}_i \mathbf{x}_i + b$$

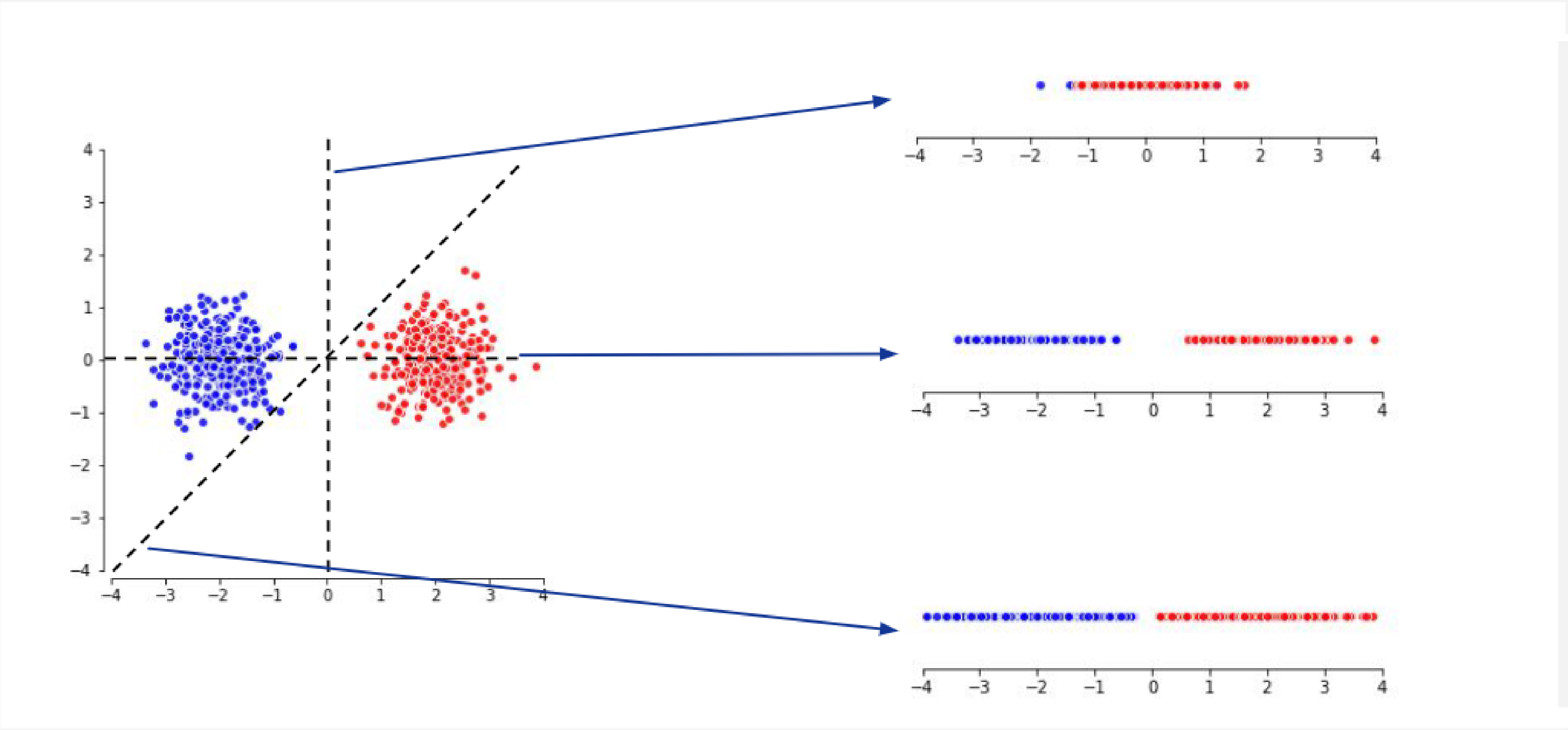
$$\sum_{i=0}^{d} \mathbf{w}_i \mathbf{x}_i \qquad \mathbf{x}_0 := 1$$

- Easy to compose
- Represents simple computation
- Has inhibition and excitation connections

- Is stateless wrt. time
- Outputs real values

NEURAL NETWORKS

A model of an artificial neuron

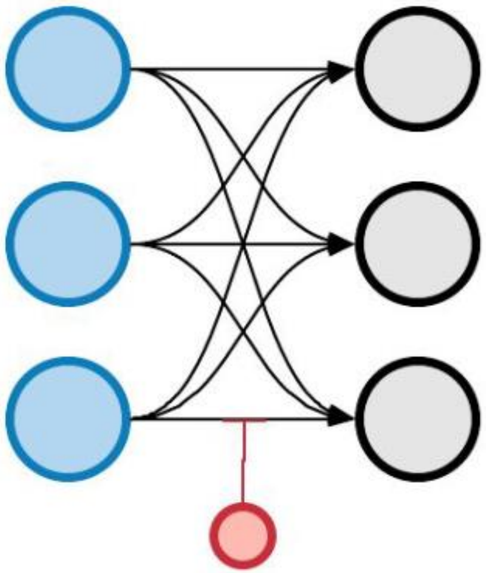


- Easy to compose
- Represents simple computation
- Has inhibition and excitation connections

- Is stateless wrt. time
- Outputs real values

NEURAL NETWORKS

A linear layer □ Single layer neural networks



$$h(\mathbf{x}, \mathbf{w}, b) = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

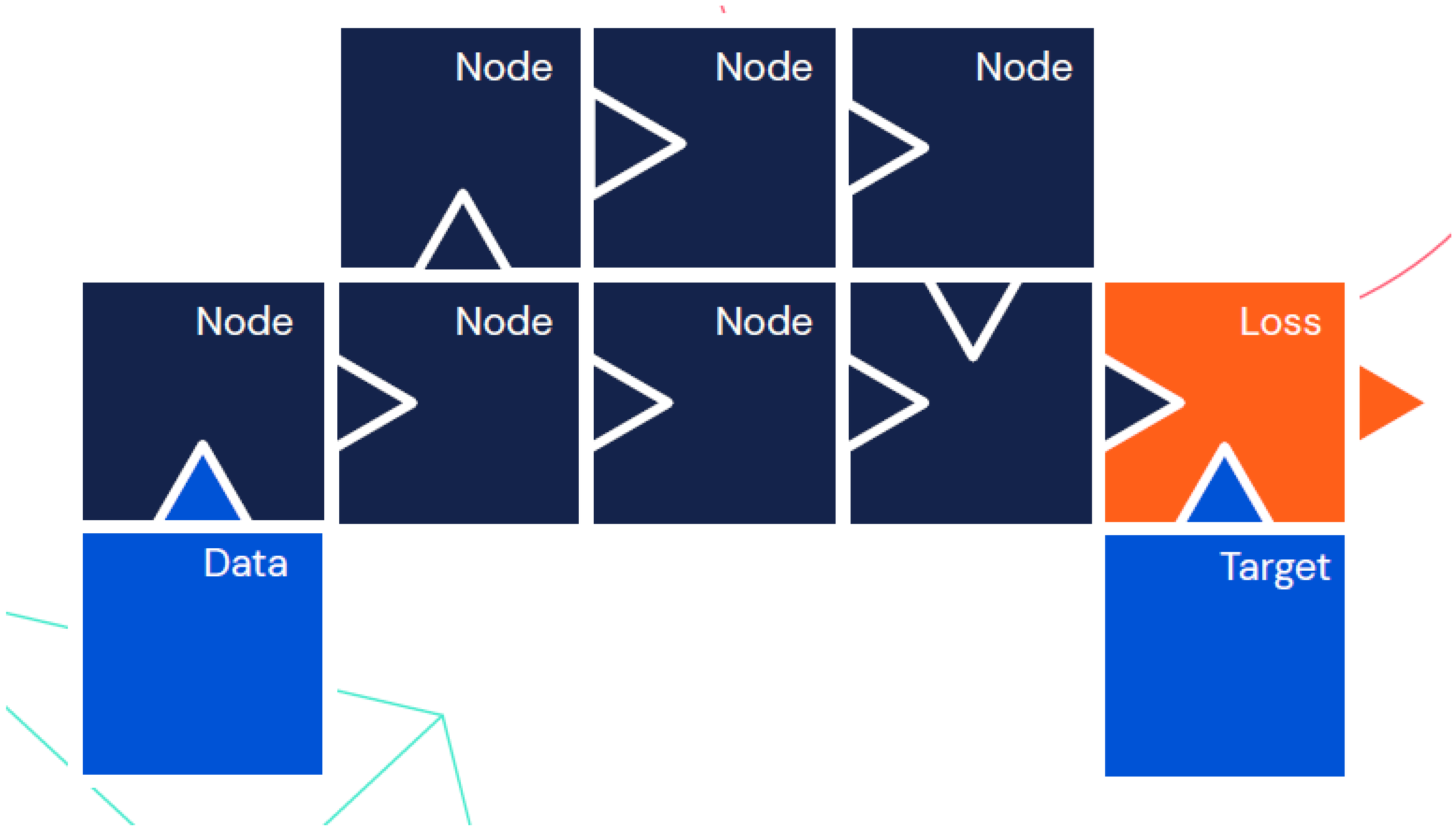
$$f_{\text{linear}}(\mathbf{x}, \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

- Easy to compose
- Collection of artificial neurons
- Can be efficiently vectorised
- Fits highly optimised hardware (GPU/TPU)

In Machine Learning linear really means affine.
Neurons in a layer are often called units.
Parameters are often called weights.

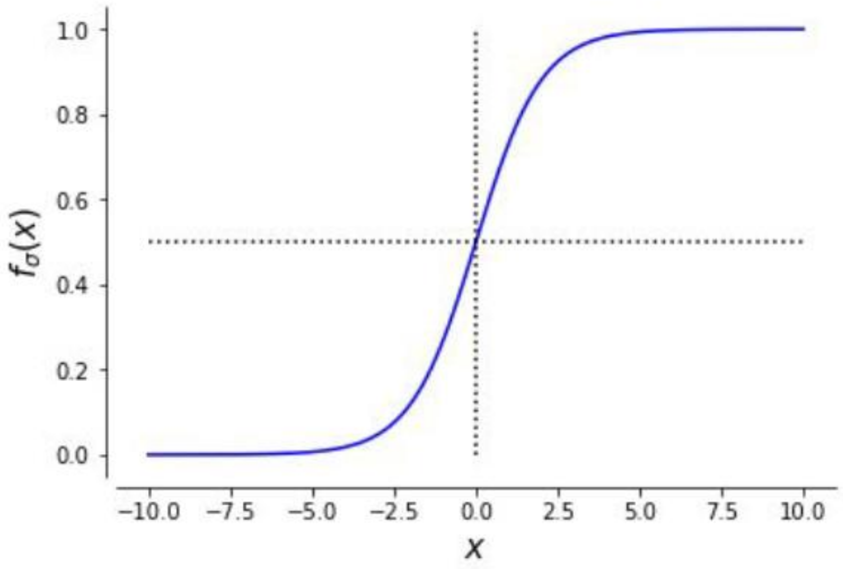
NEURAL NETWORKS

A linear layer Single layer neural networks



NEURAL NETWORKS

Sigmoid activation function



$$f_\sigma(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}}}$$

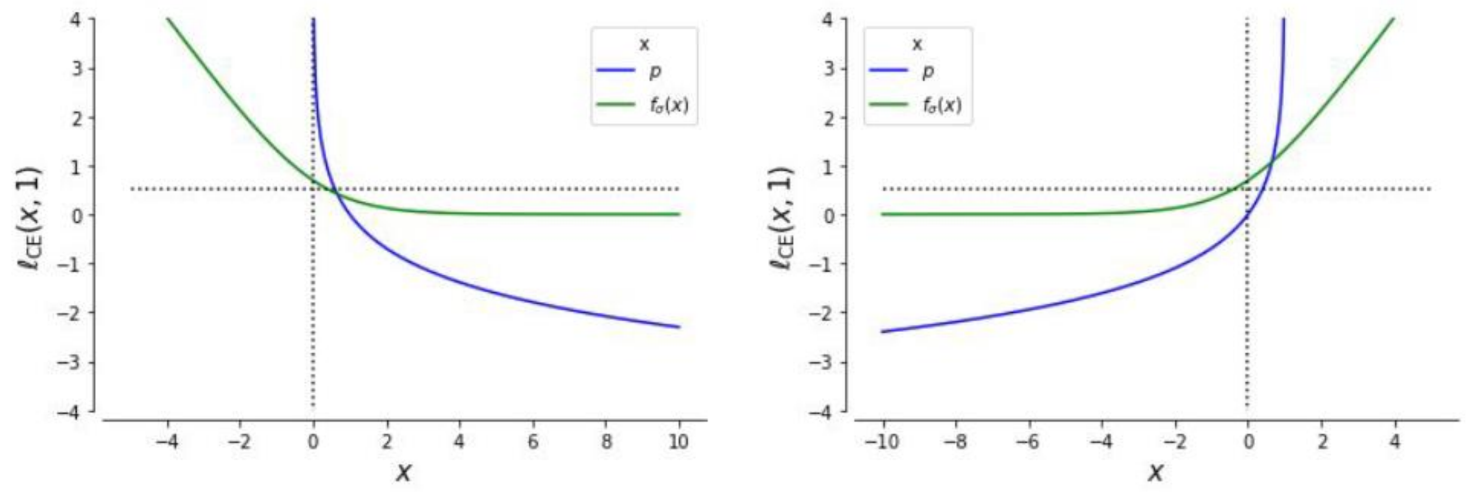
$$f_\sigma(\mathbf{x}) = \frac{e^{\mathbf{x}}}{e^{\mathbf{x}} + 1}$$

- Introduces non-linear behaviour
- Produces probability estimate
- Has simple derivatives

- Saturates
- Derivatives vanish

Activation functions are often called non-linearities.
Activation functions are applied point-wise

NEURAL NETWORKS
Cross Entropy Loss Function



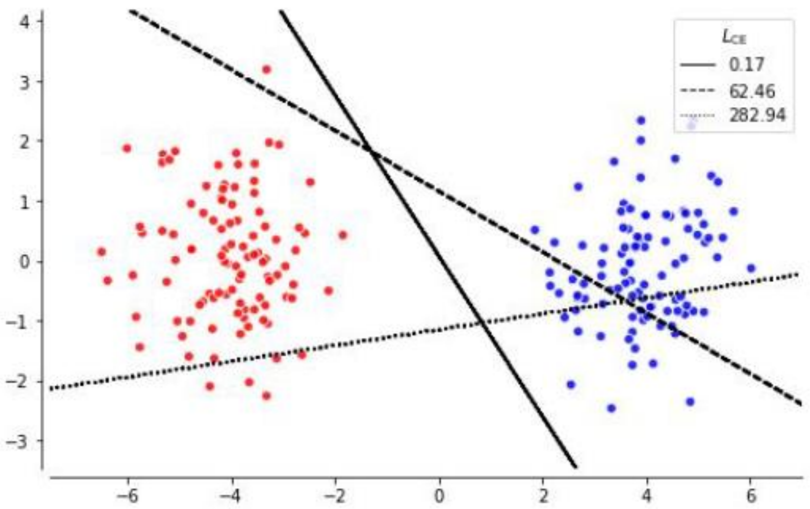
$$\ell_{CE}(\mathbf{p}, \mathbf{t}) = -[\mathbf{t} \log \mathbf{p} + (1 - \mathbf{t}) \log(1 - \mathbf{p})]$$

- Encodes negation of logarithm of probability of correct classification
- Composable with sigmoid
- Numerically unstable

NEURAL NETWORKS

The simplest “neural” network classifier

Cross entropy loss is also called
negative log likelihood or logistic loss.
Being additive over samples allows for efficient learning.



$$L_{\mathrm{CE}}(\mathbf{p}, \mathbf{t}) = -\sum_{i=1}^{n} [\mathbf{t}^{(i)} \log \mathbf{p}^{(i)} + (1 - \mathbf{t}^{(i)}) \log(1 - \mathbf{p}^{(i)})]$$

- Encodes negation of logarithm of probability of **entirely correct** classification
- Equivalent to logistic regression model
- Numerically unstable

NEURAL NETWORKS

Softmax

$$f_{\mathrm{sm}}(\mathbf{x}) = \frac{e^{\mathbf{x}}}{\sum_{j=1}^{k} e^{\mathbf{x}_j}}$$

$$f_{\mathrm{sm}}([x, 0]) = \left[\frac{e^x}{e^x + e^0}, \frac{e^0}{e^x + e^0}\right]$$

$$= [f_\sigma(x), 1 - f_\sigma(x)]$$

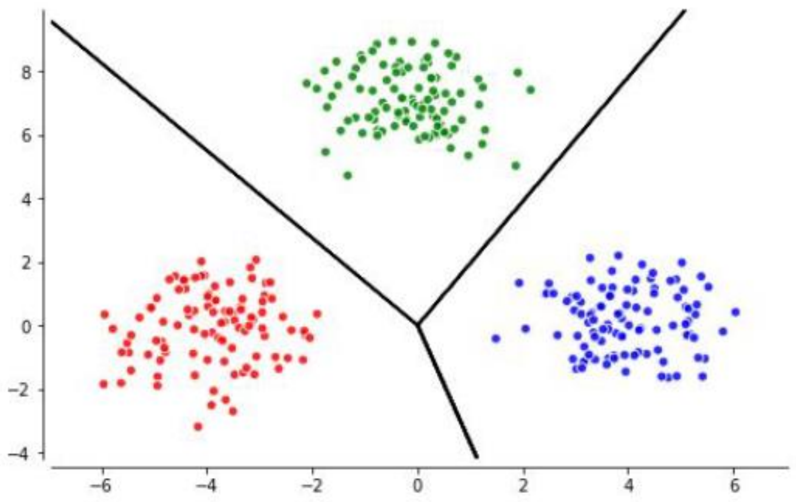
- Multi-dimensional generalisation of sigmoid
- Produces probability estimate
- Has simple derivatives

- Saturates
- Derivatives vanish

Softmax is the most commonly used final activation in classification.
It can also be used to have a smooth version of maximum.

NEURAL NETWORKS

Softmax + Cross-entropy



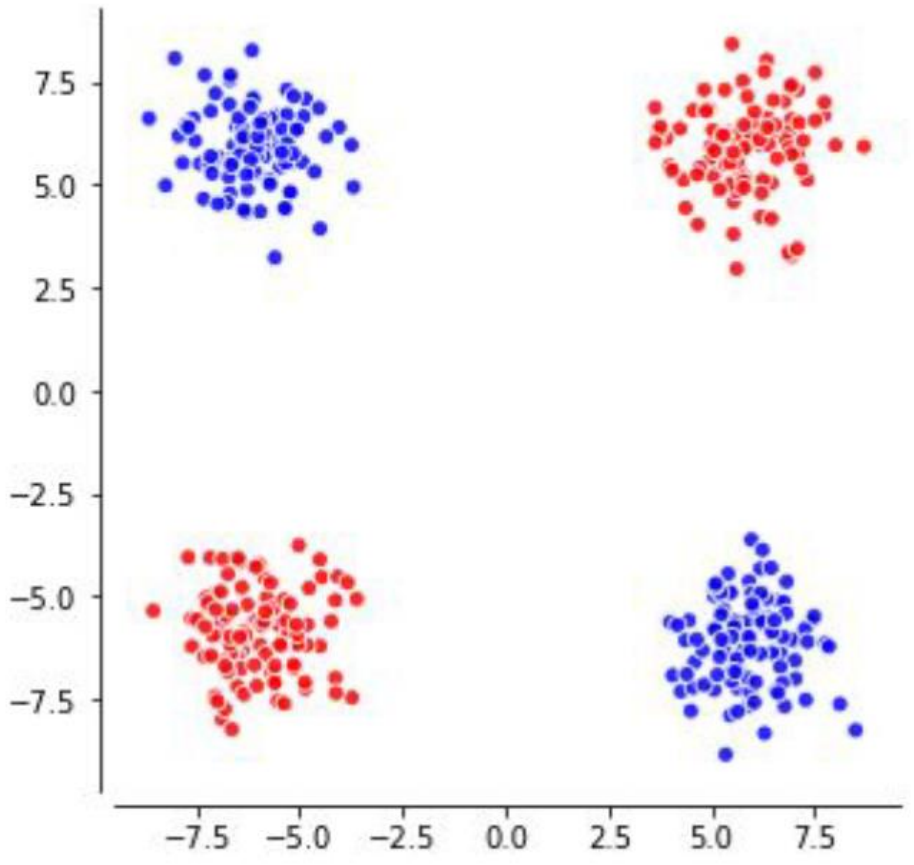
$$\ell_{\mathrm{CE}}(f_{\mathrm{sm}}(\mathbf{x}), \mathbf{t}) = -\sum_{j=1}^{k} \mathbf{t}_j \log[f_{\mathrm{sm}}(\mathbf{x}_j)] = -\sum_{j=1}^{k} \mathbf{t}_j [\mathbf{x}_j - \log \sum_{l=1}^{k} e^{\mathbf{x}_l}]$$

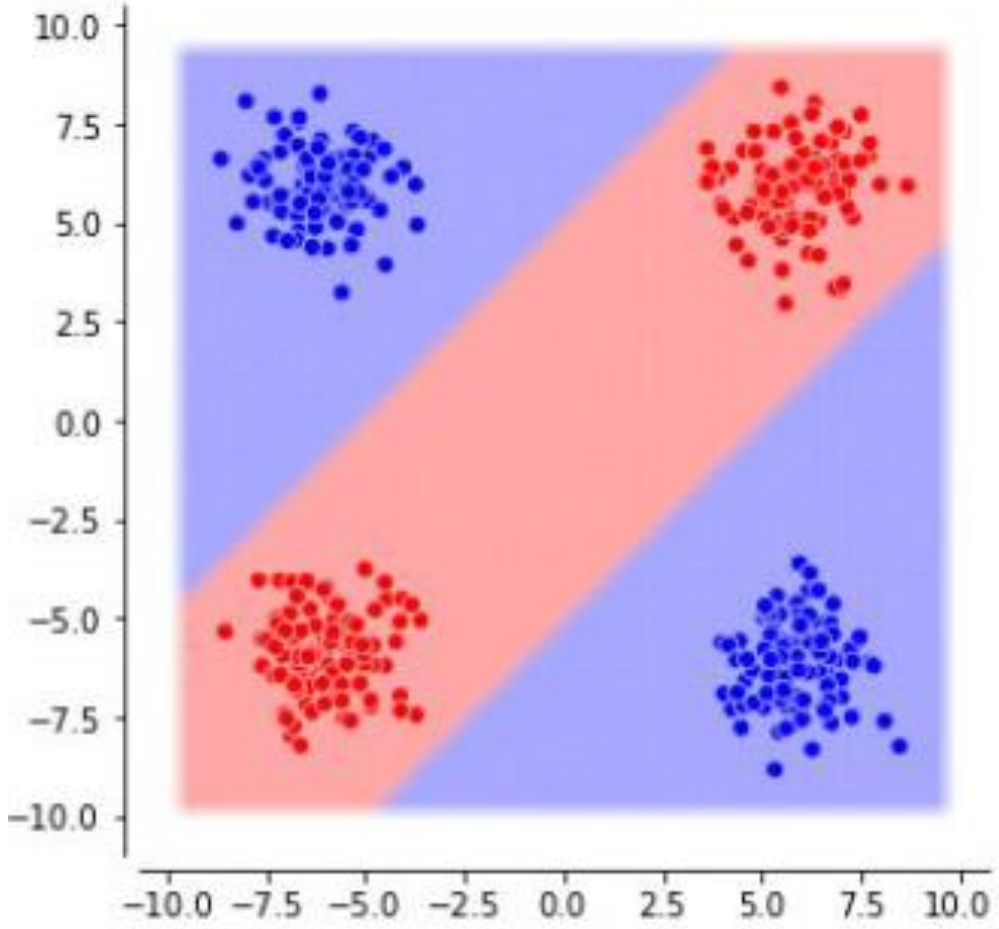
- Encodes negation of logarithm of probability of entirely correct classification
- Equivalent to multinomial logistic regression model
- Numerically stable combination

Widely used not only in classification but also in RL.
Cannot represent sparse outputs (sparsemax)

NEURAL NETWORKS

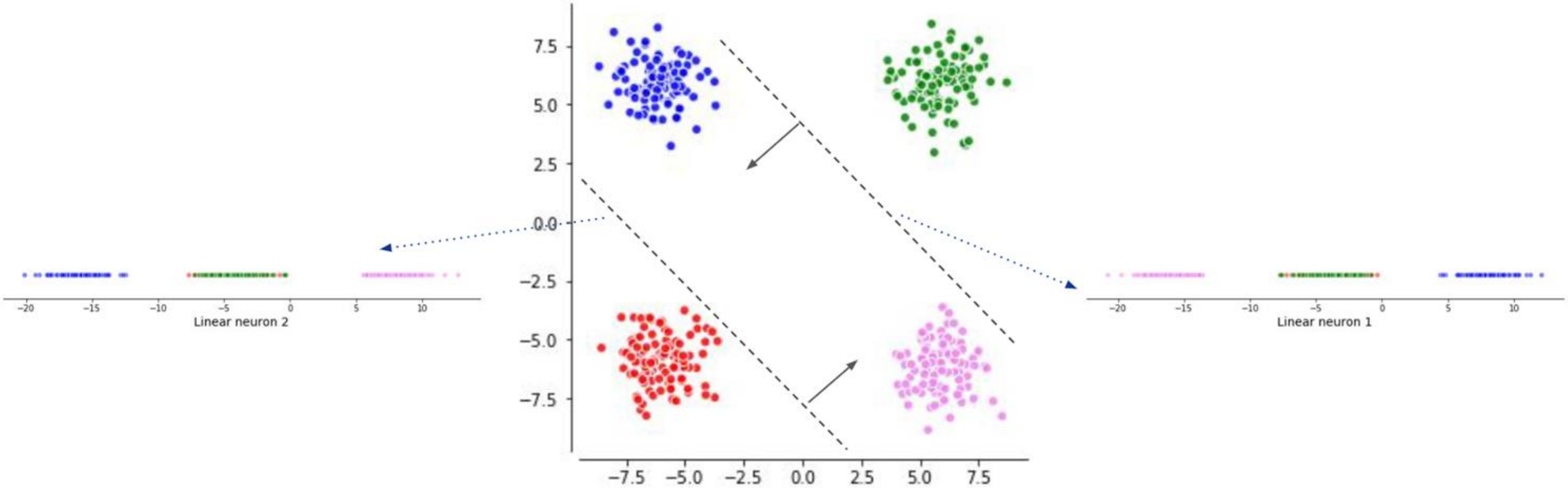
Limitations –> The XOR problem





NEURAL NETWORKS

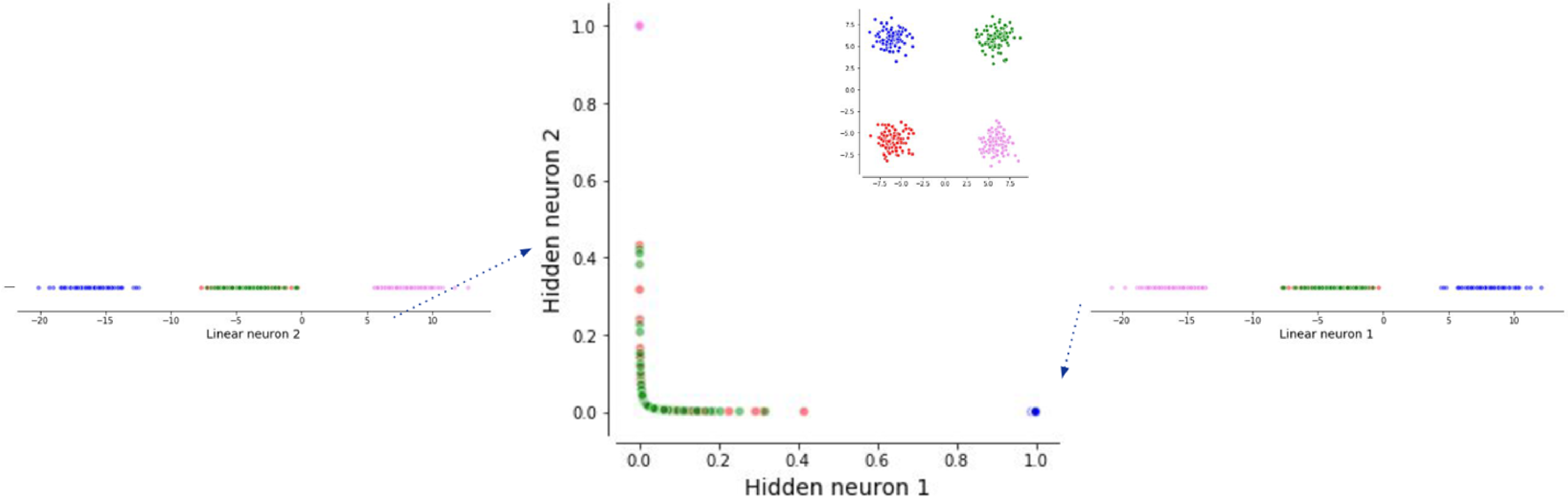
Limitations –> The XOR problem



$$\mathbf{W} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -4 \\ -4 \end{bmatrix}$$

NEURAL NETWORKS

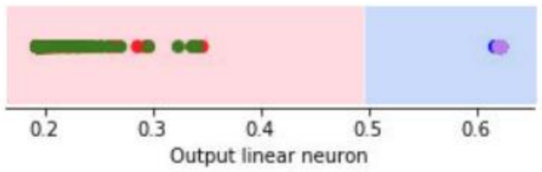
Limitations –> The XOR problem

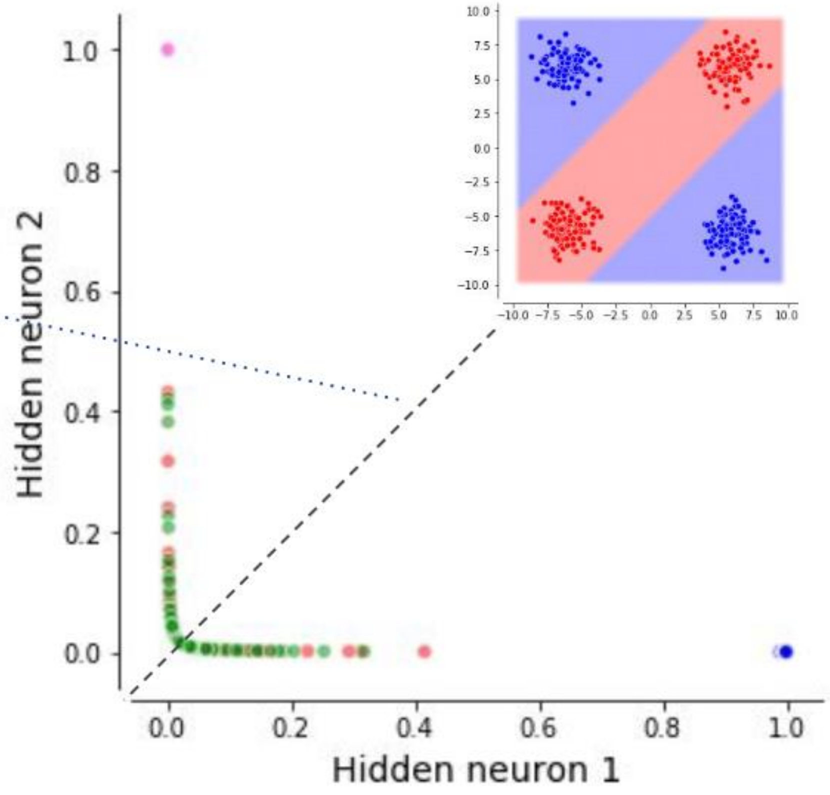


$$\mathbf{W} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -4 \\ -4 \end{bmatrix}$$

NEURAL NETWORKS

1 Hidden Layer Neural Network vs the XOR



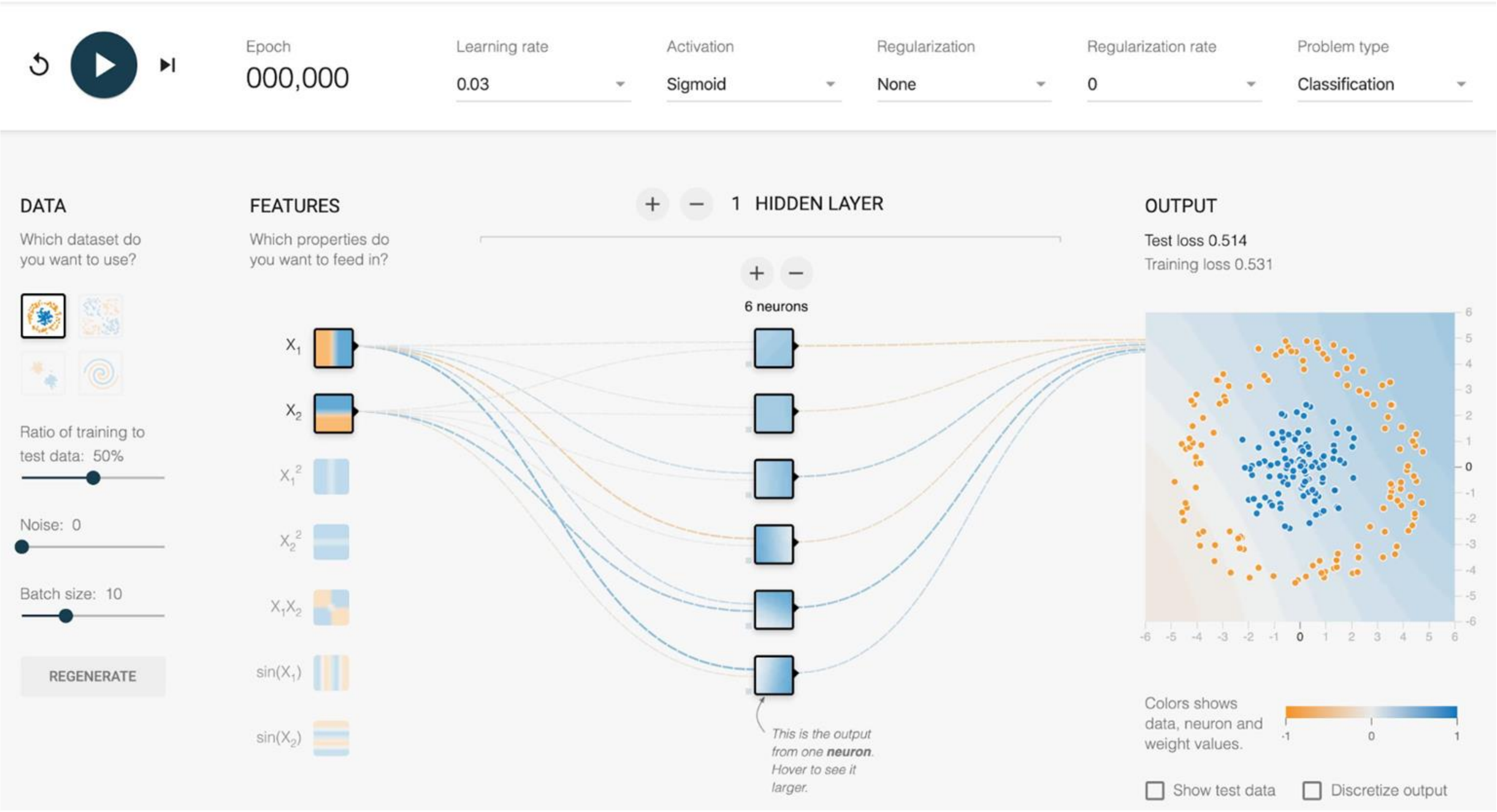


- With just 2 hidden neurons we solve XOR
- Hidden layer allows us to bent and twist input space
- We use linear model on top, to do the classification

$$\mathbf{W} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -4 \\ -4 \end{bmatrix}$$

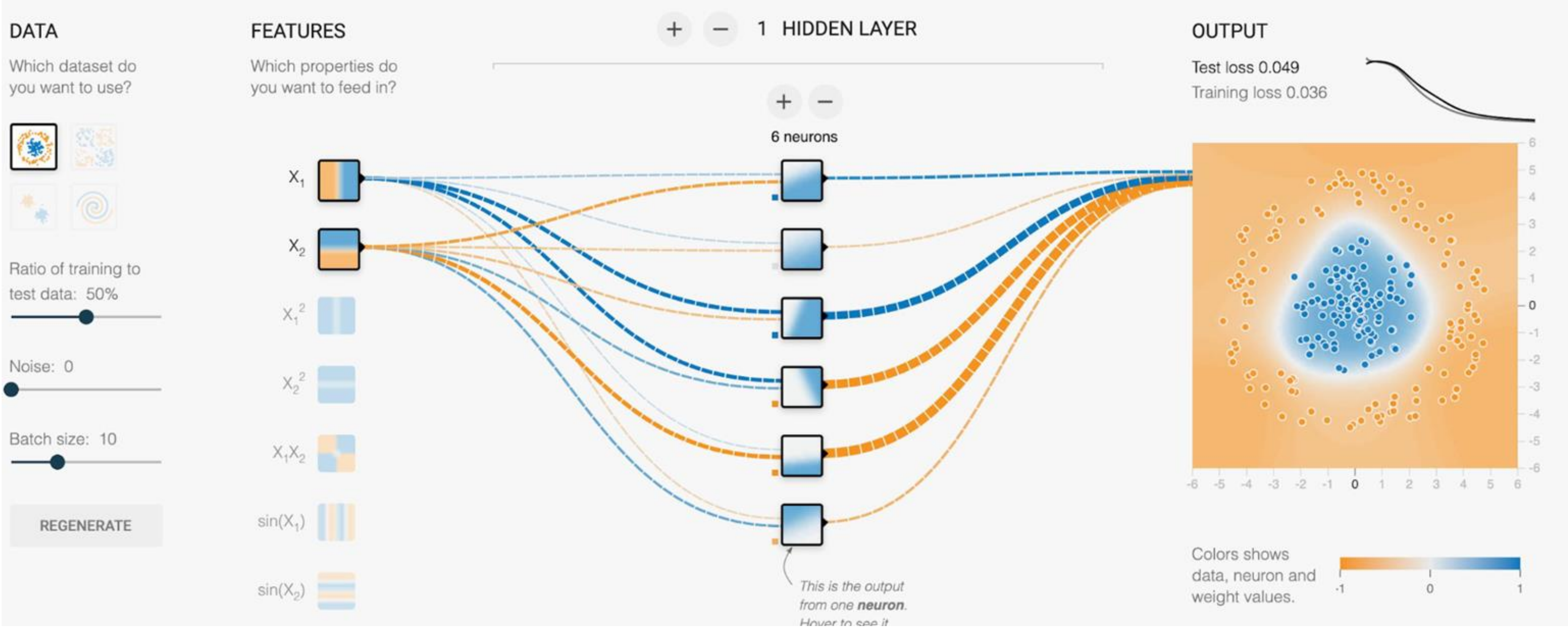
NEURAL NETWORKS
Neural Network Playground

http://playground.tensorflow.org/ by **Daniel Smilkov** and **Shan Carter**



NEURAL NETWORKS

Neural Network Playground



NEURAL NETWORKS
Representation Learning

- Methods that allow a machine to be fed with raw data to automatically discover representations needed for detection or classification

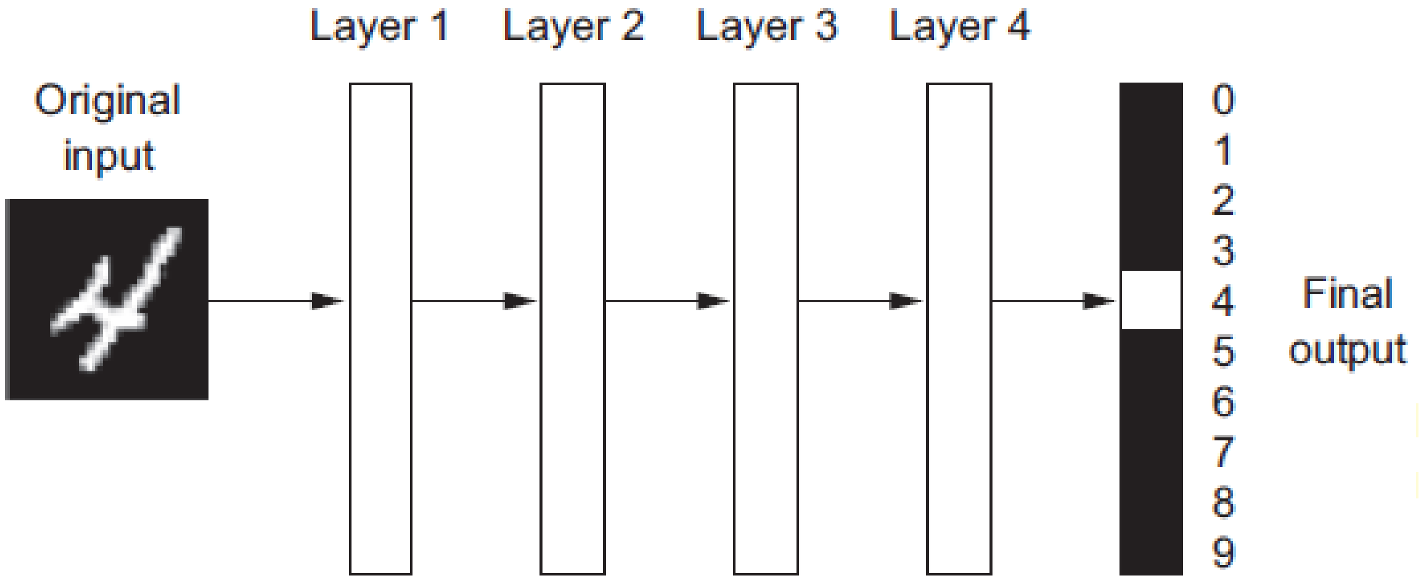
- Deep Learning methods are Representation Learning Methods

- Use multiple levels of representation

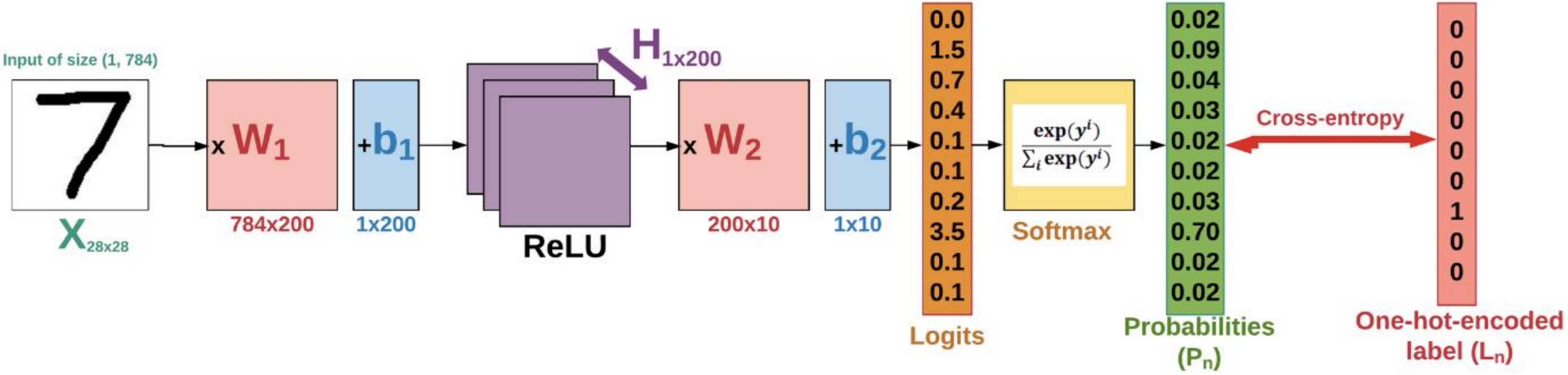
- Composing simple but non-linear modules that transform representation at one level (starting with raw input) into a representation at a higher slightly more abstract level

- Complex functions can be learned

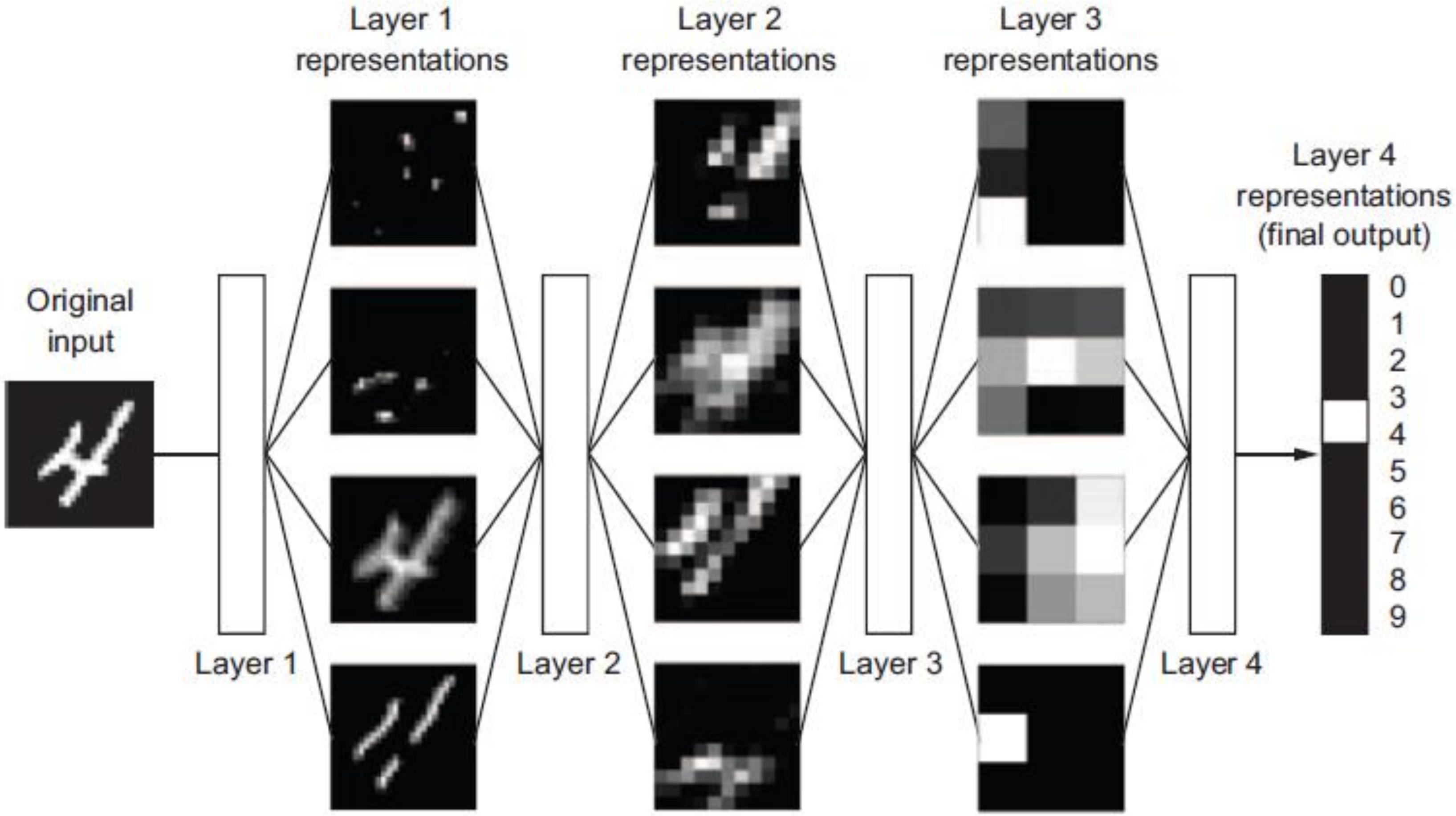
NEURAL NETWORKS
what are they?



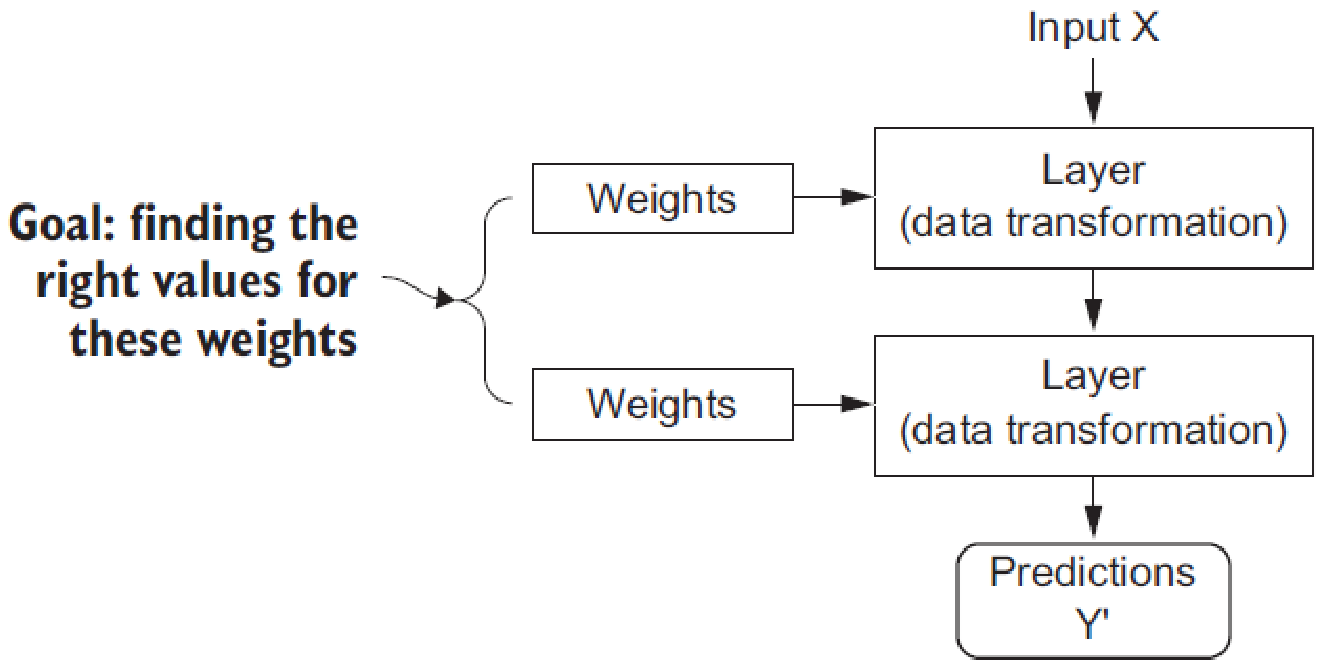
NEURAL NETWORKS
what are they?

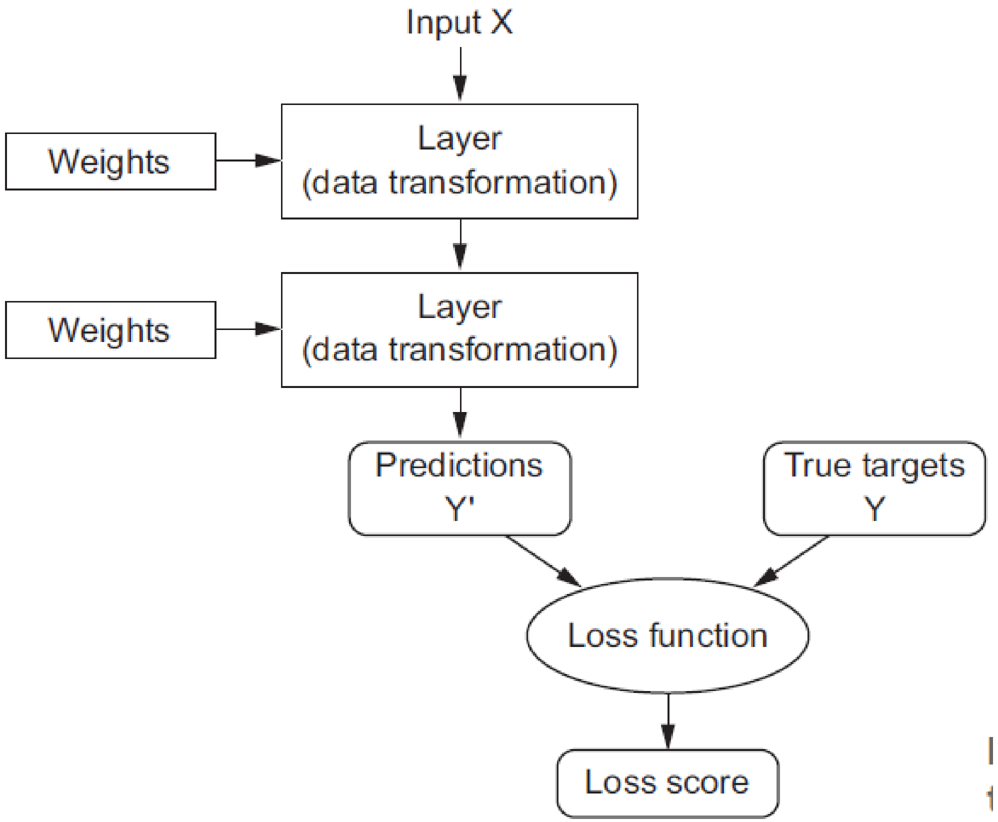


NEURAL NETWORKS
what are they?



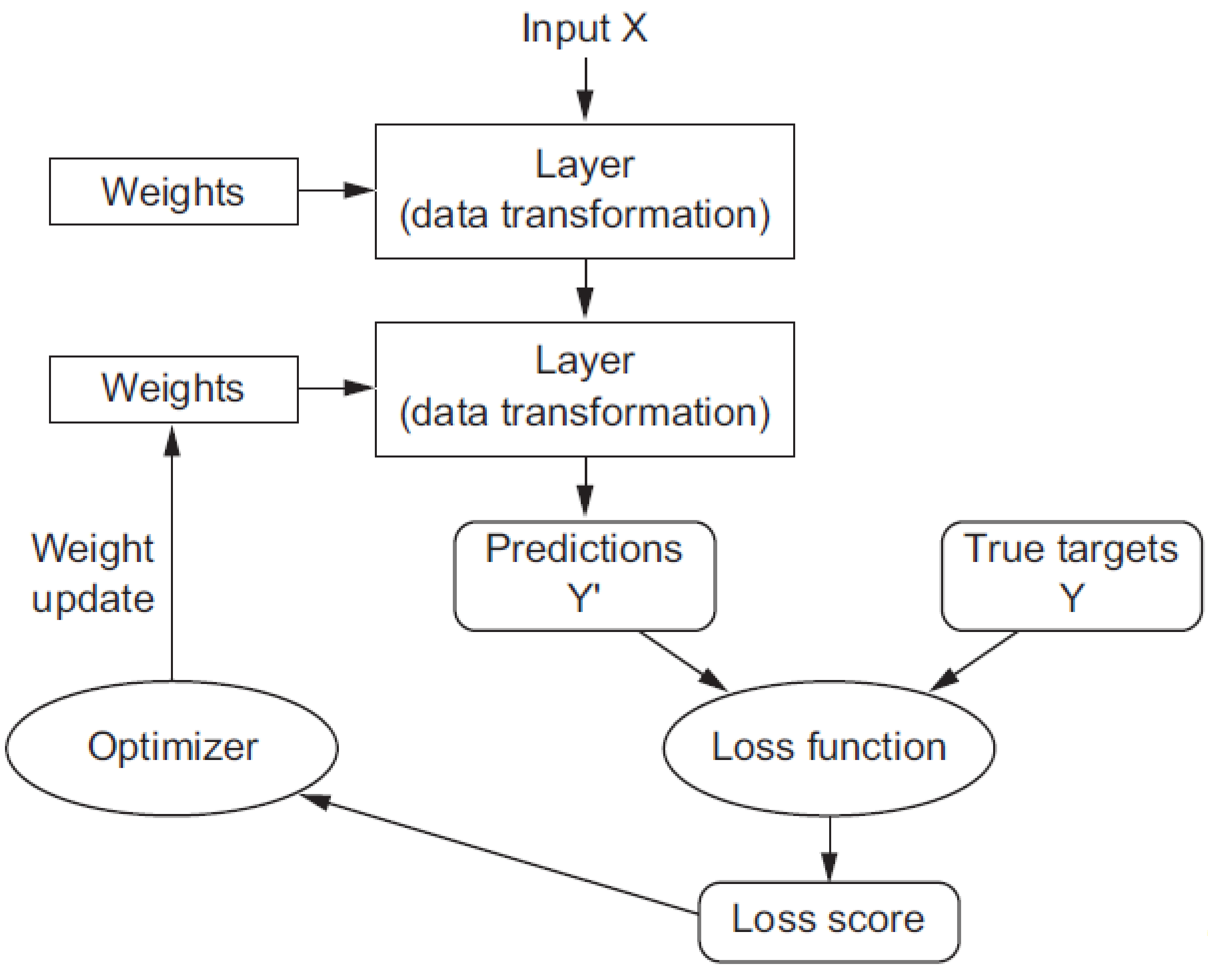
NEURAL NETWORKS
How are they trained?





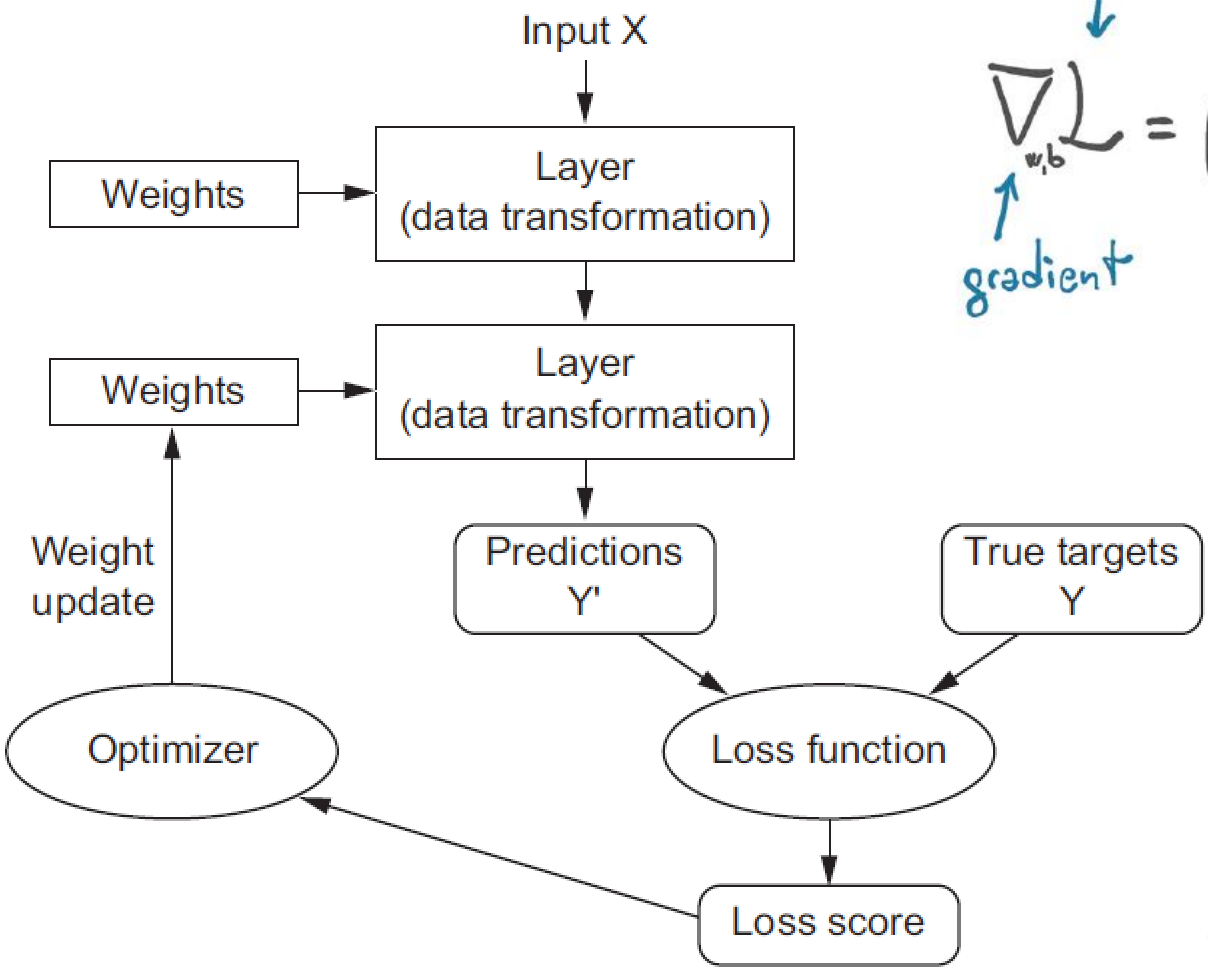
NEURAL NETWORKS

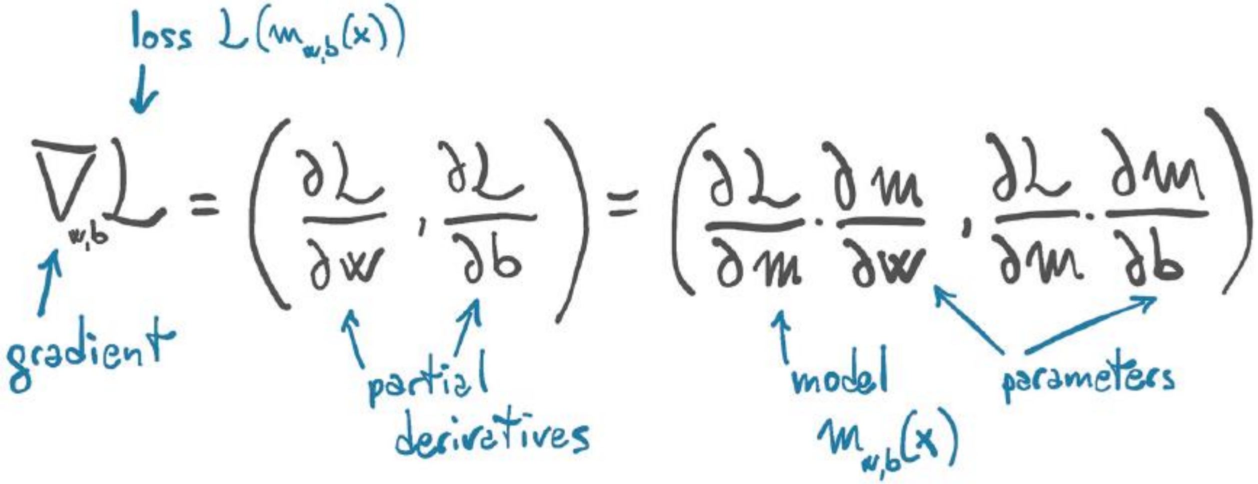
How are they trained?

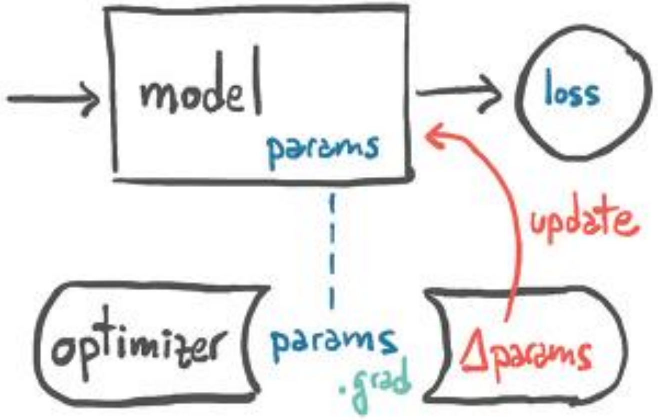


NEURAL NETWORKS

How are they trained?

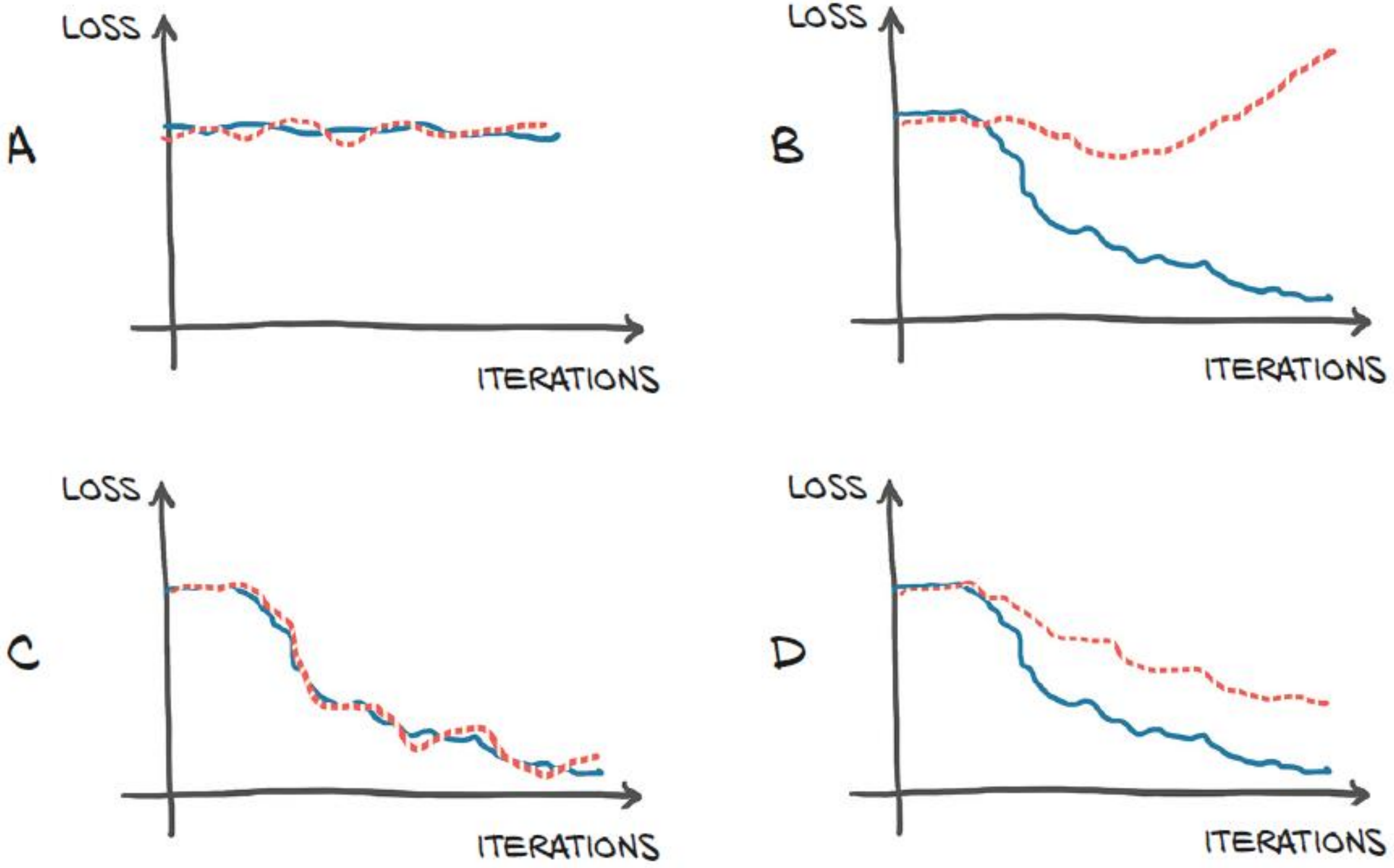




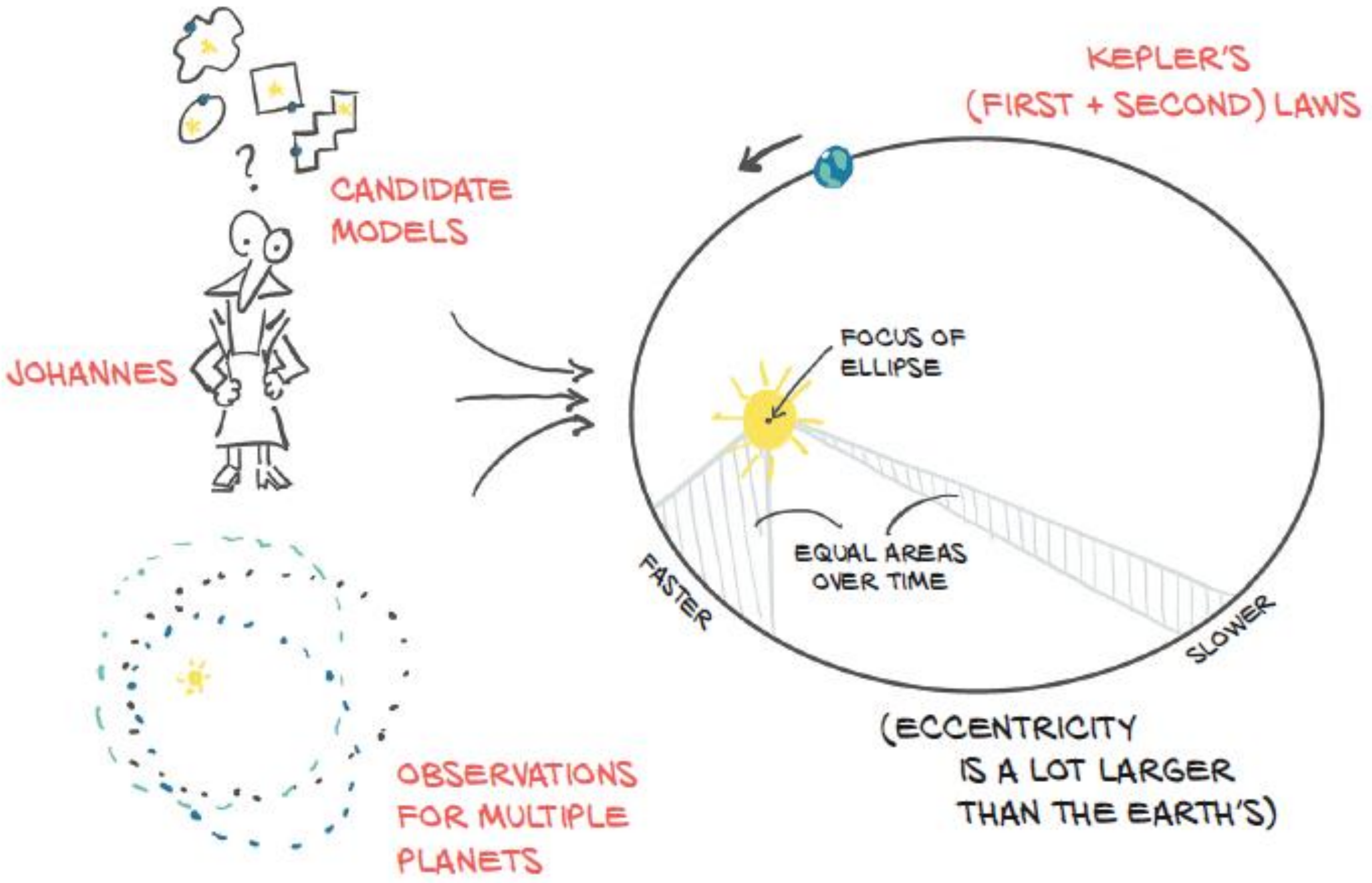


NEURAL NETWORKS

How are they trained?

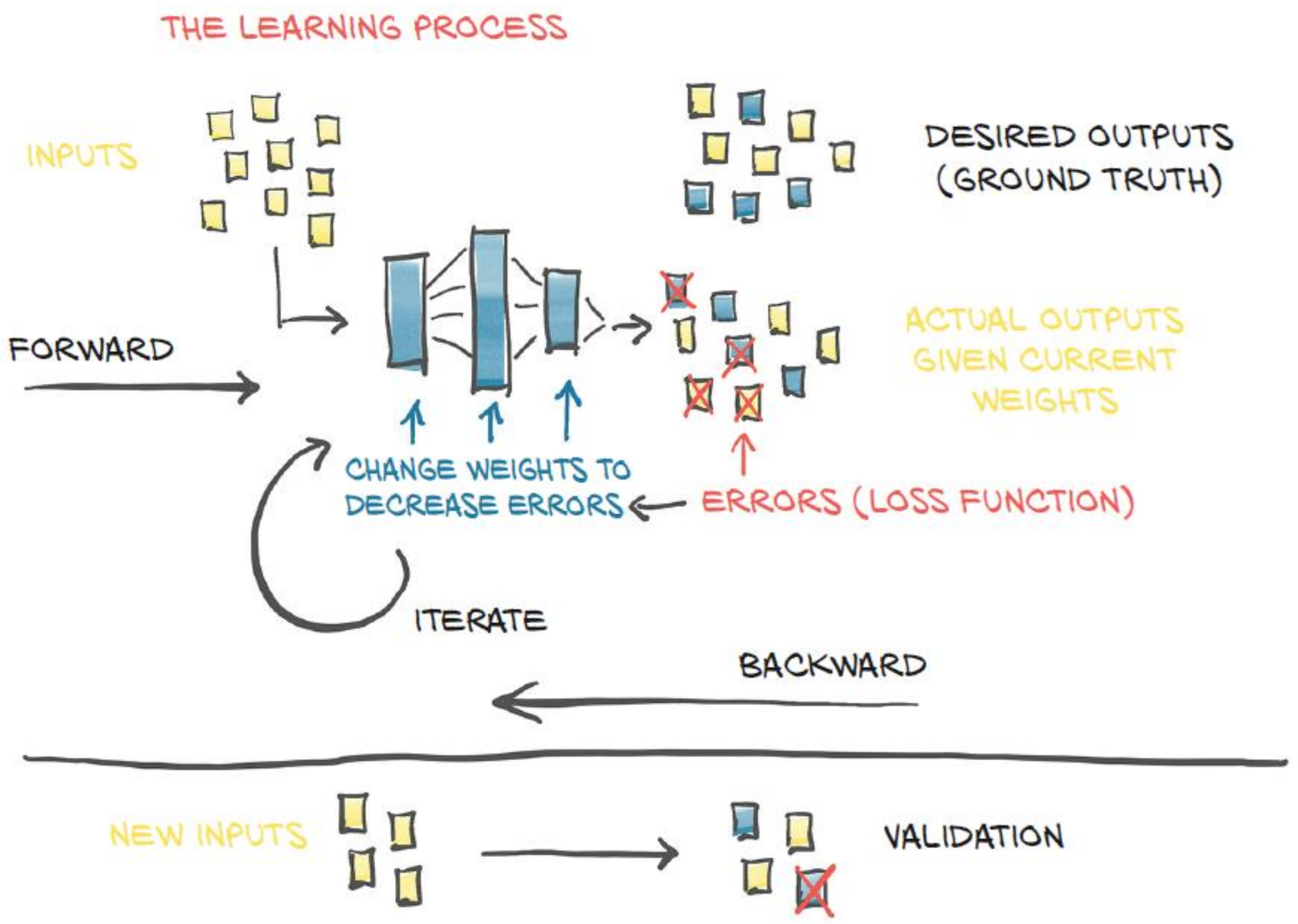


NEURAL NETWORKS
How are they trained?



NEURAL NETWORKS

How are they trained?



NEURAL N

Mathematical Intuitions

THE "NEURON"

LEARNED PARAMETERS

$$o = \tanh(\underbrace{wx + b}_{\text{LINEAR TRANSFORMATION}})$$

OUTPUT

INPUT

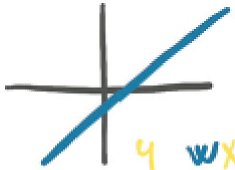
NONLINEAR FUNCTION (ACTIVATION)

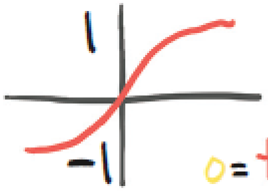
LEARNED

↓

$w = 2$

$b = 6$





$o = \tanh(y)$

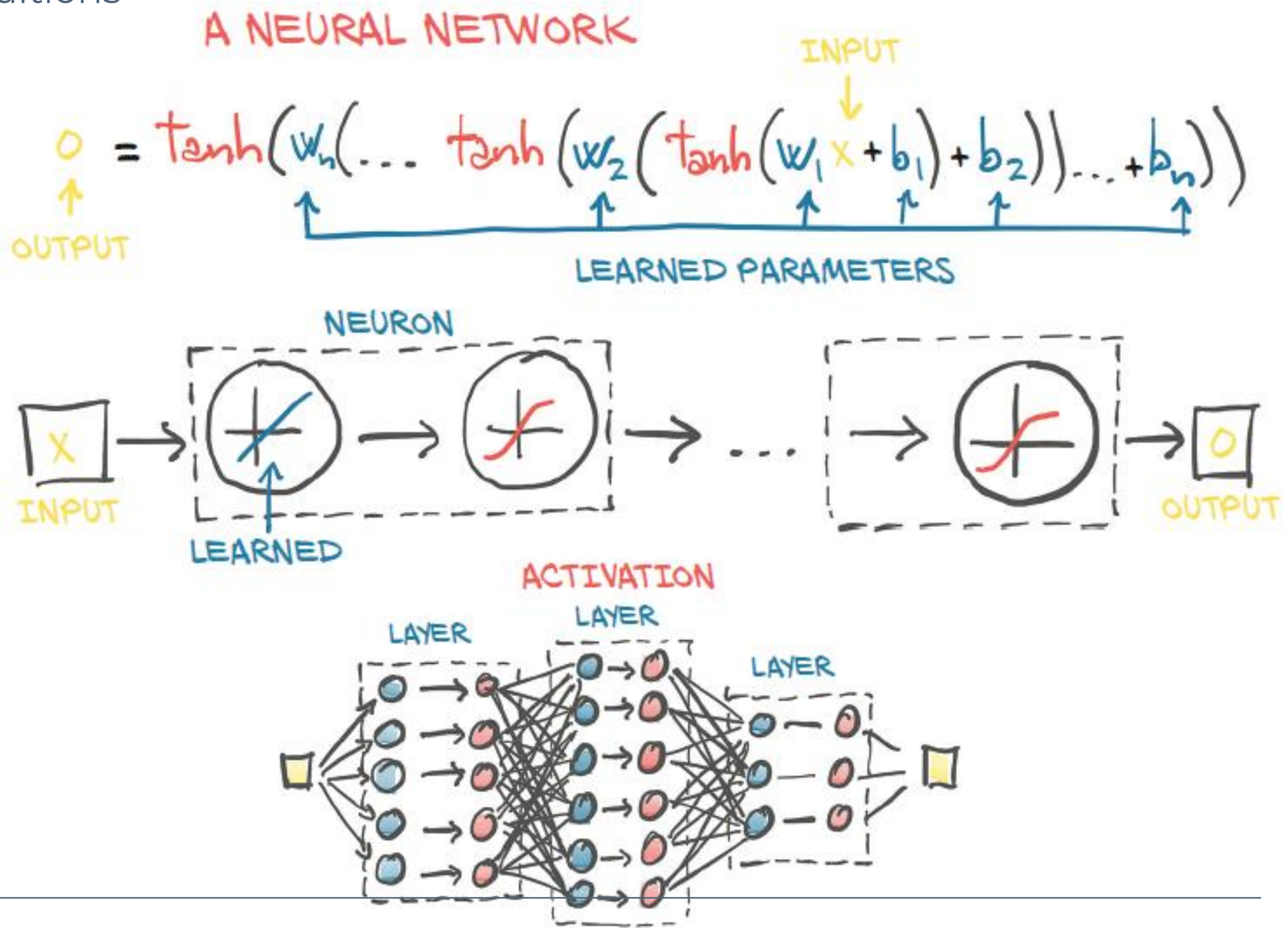
$18 \longrightarrow 2 \times 18 + 6 = 42 \rightarrow \tanh(42) = 1$

$-2.79 \longrightarrow 2 \times (-2.79) + 6 = .042 \rightarrow \tanh(.042) = 0.3969$

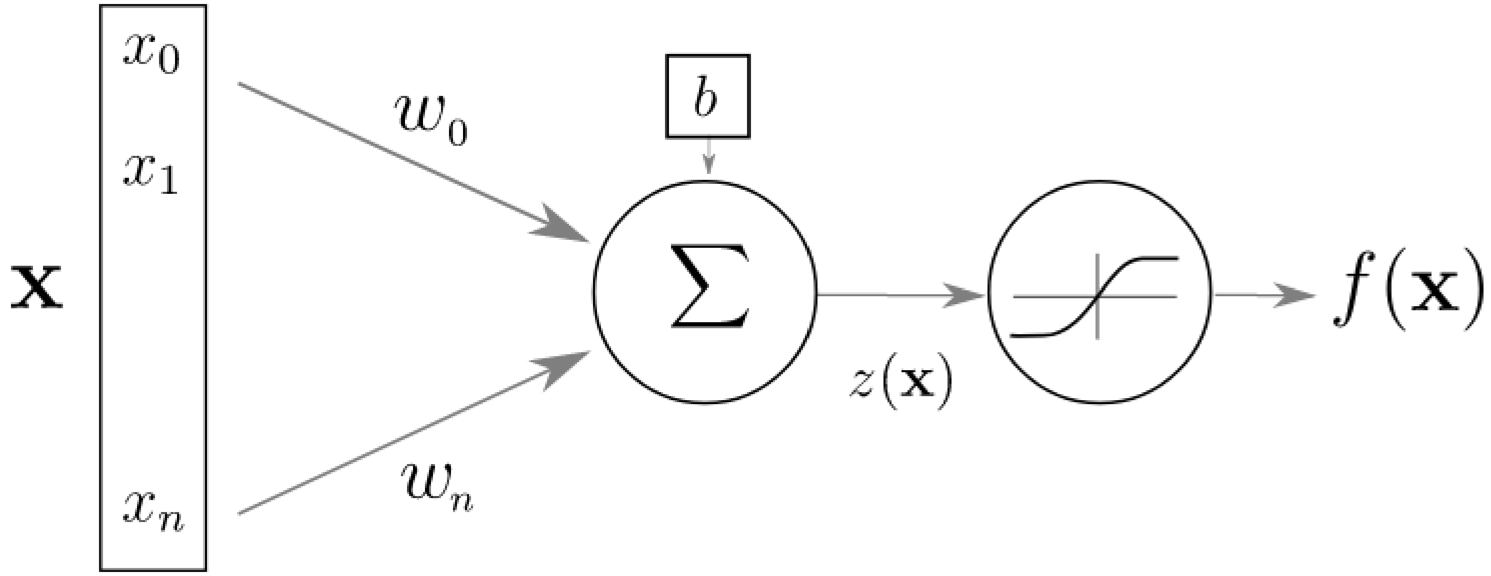
$-10 \longrightarrow 2 \times (-10) + 6 = -14 \rightarrow \tanh(-14) = -1$

NEURAL NETWORKS

Mathematical Intuitions



NEURAL NETWORKS
Mathematical Intuitions



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

$$f(\mathbf{x}) = g(\mathbf{w}^T\mathbf{x} + b)$$

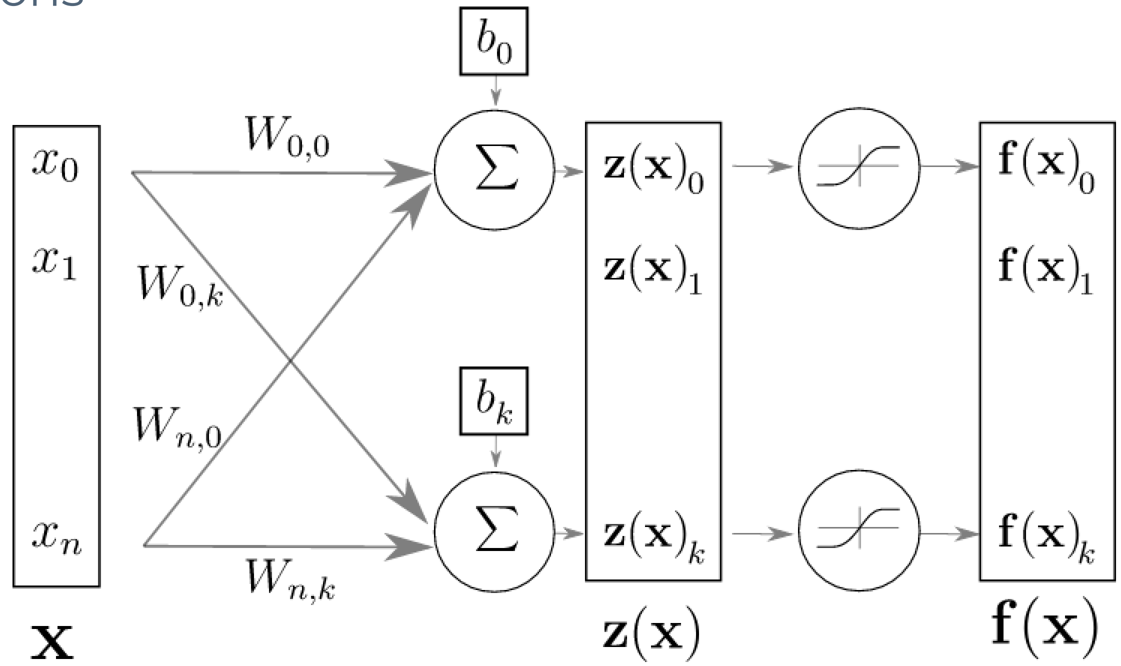
$\mathbf{x}$,$f(\mathbf{x})$ input and output

$z(\mathbf{x})$ pre-activation

$\mathbf{w}$,b weights and bias

g activation function

NEURAL NETWORKS
Mathematical Intuitions

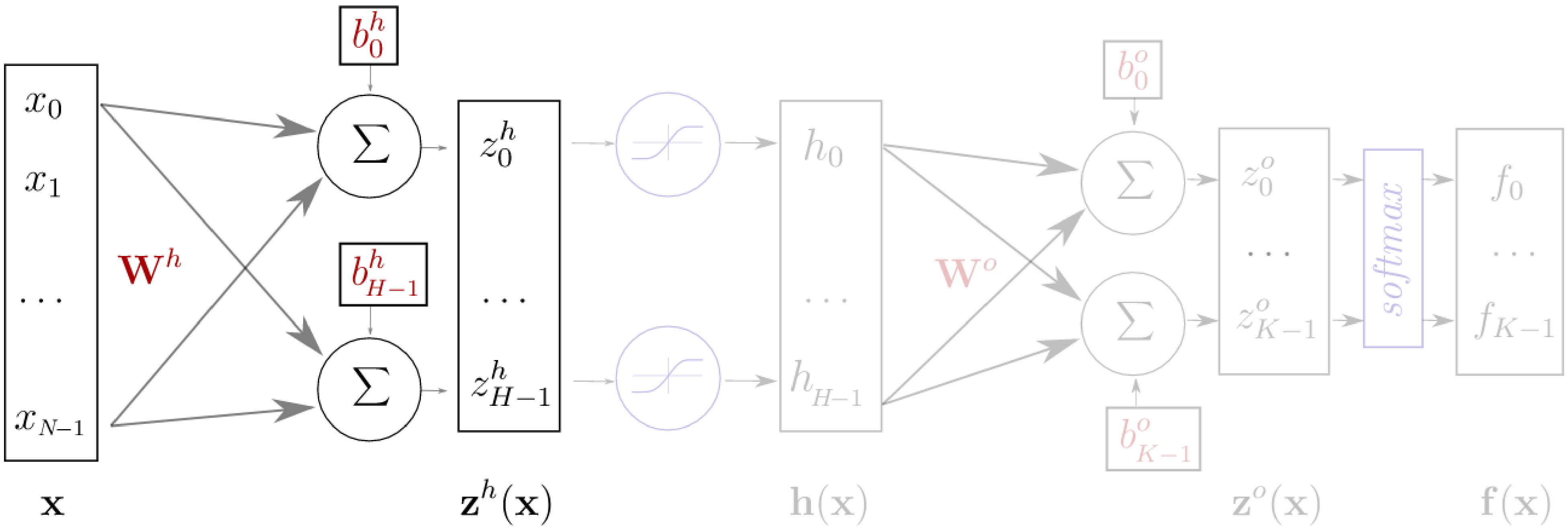


$$\mathbf{f}(\mathbf{x}) = g(\mathbf{z}(\mathbf{x})) = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

W,**b** now matrix and vector

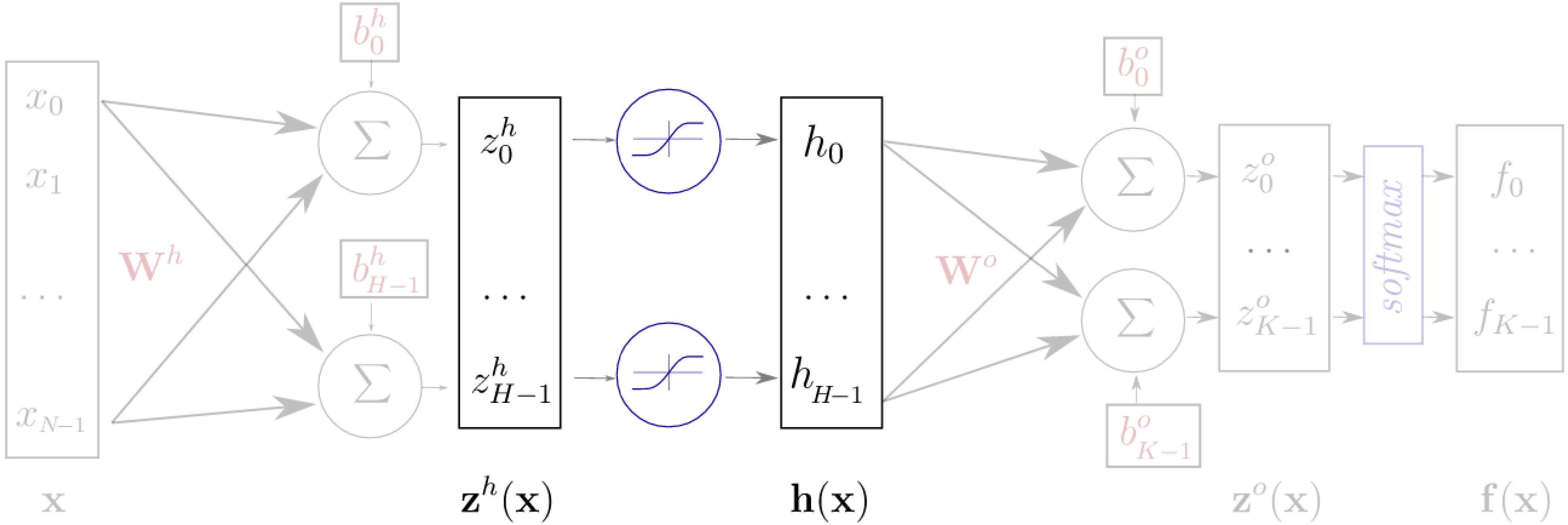
NEURAL NETWORKS

Mathematical Intuitions



- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

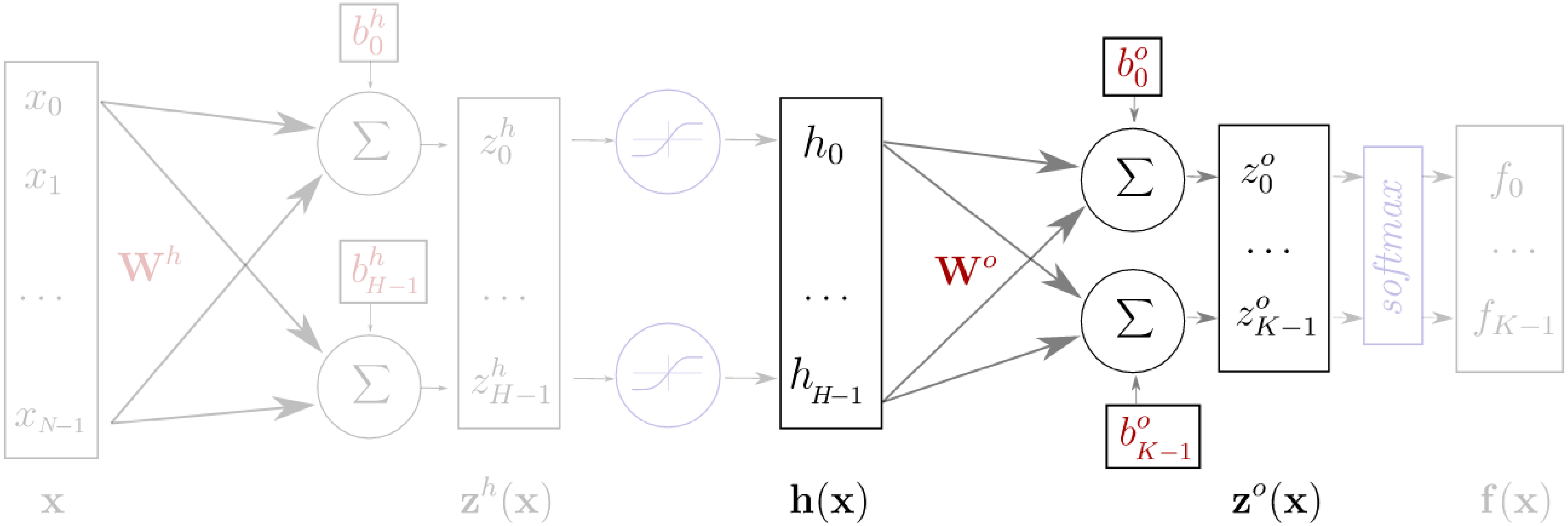
NEURAL NETWORKS
Mathematical Intuitions



- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

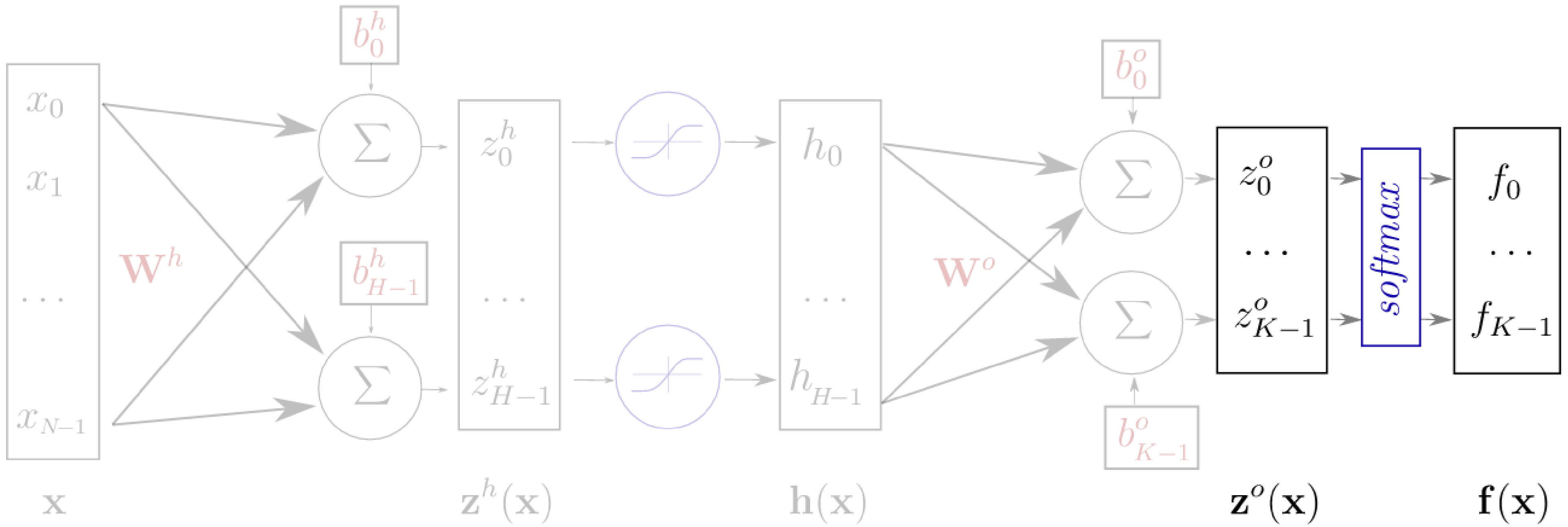
NEURAL NETWORKS

Mathematical Intuitions



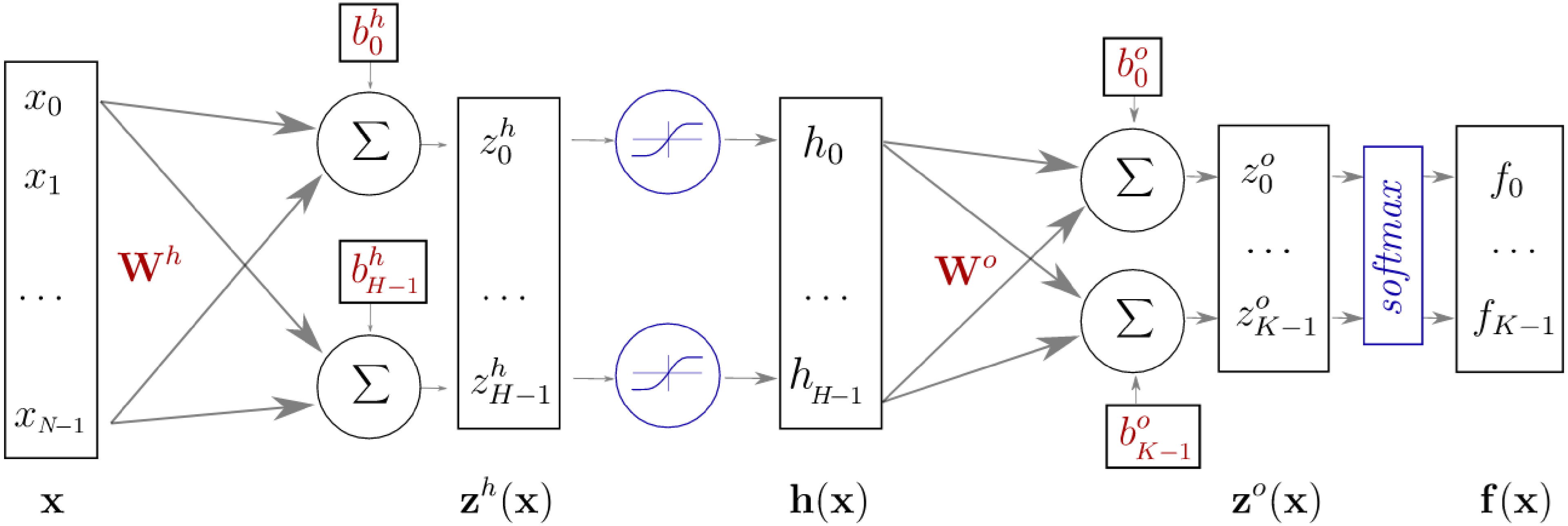
- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

NEURAL NETWORKS
Mathematical Intuitions



- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

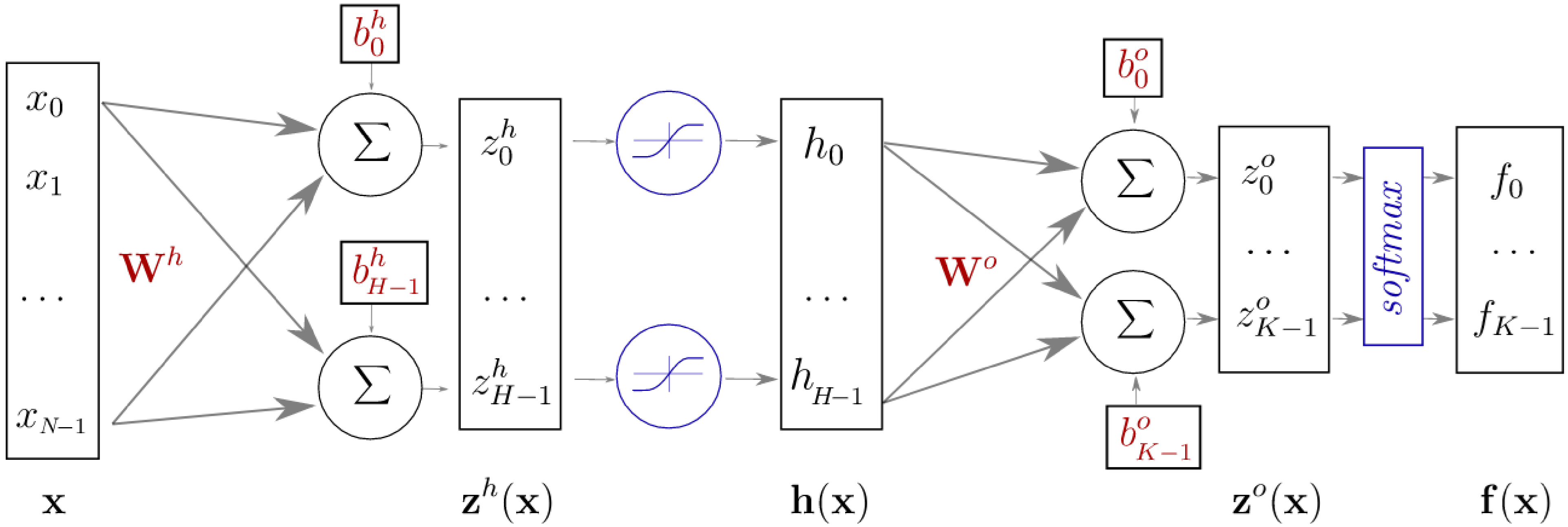
NEURAL NETWORKS
Mathematical Intuitions



- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

NEURAL NETWORKS

Mathematical Intuitions

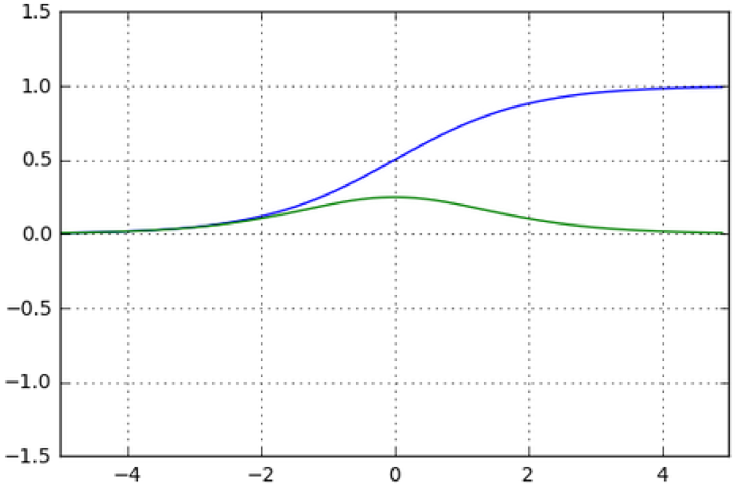


Keras Implementation

```
model = Sequential()
model.add(Dense(H, input_dim=N))  # weight matrix dim [N * H]
model.add(Activation("tanh"))
model.add(Dense(K))               # weight matrix dim [H x K]
model.add(Activation("softmax"))
```

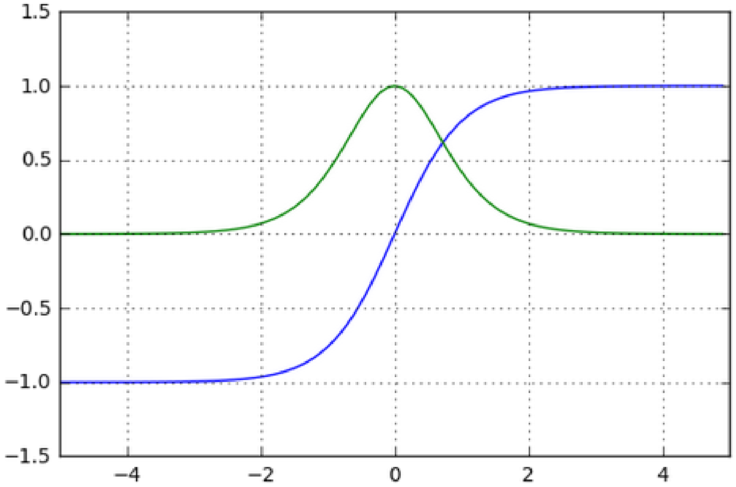
NEURAL NETWORKS

Activation Functions



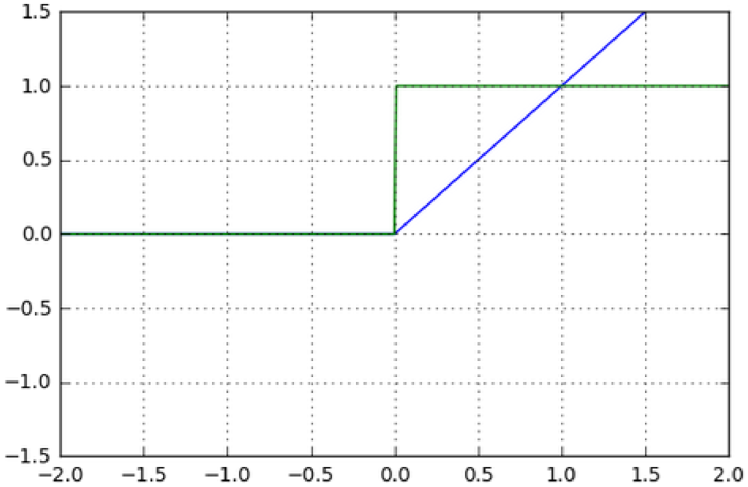
$$\mathrm{sigm}(x) = \frac{1}{1+e^{-x}}$$

$$\mathrm{sigm}'(x) = \mathrm{sigm}(x)(1 - \mathrm{sigm}(x))$$



$$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

$$\tanh'(x) = 1 - \tanh(x)^2$$

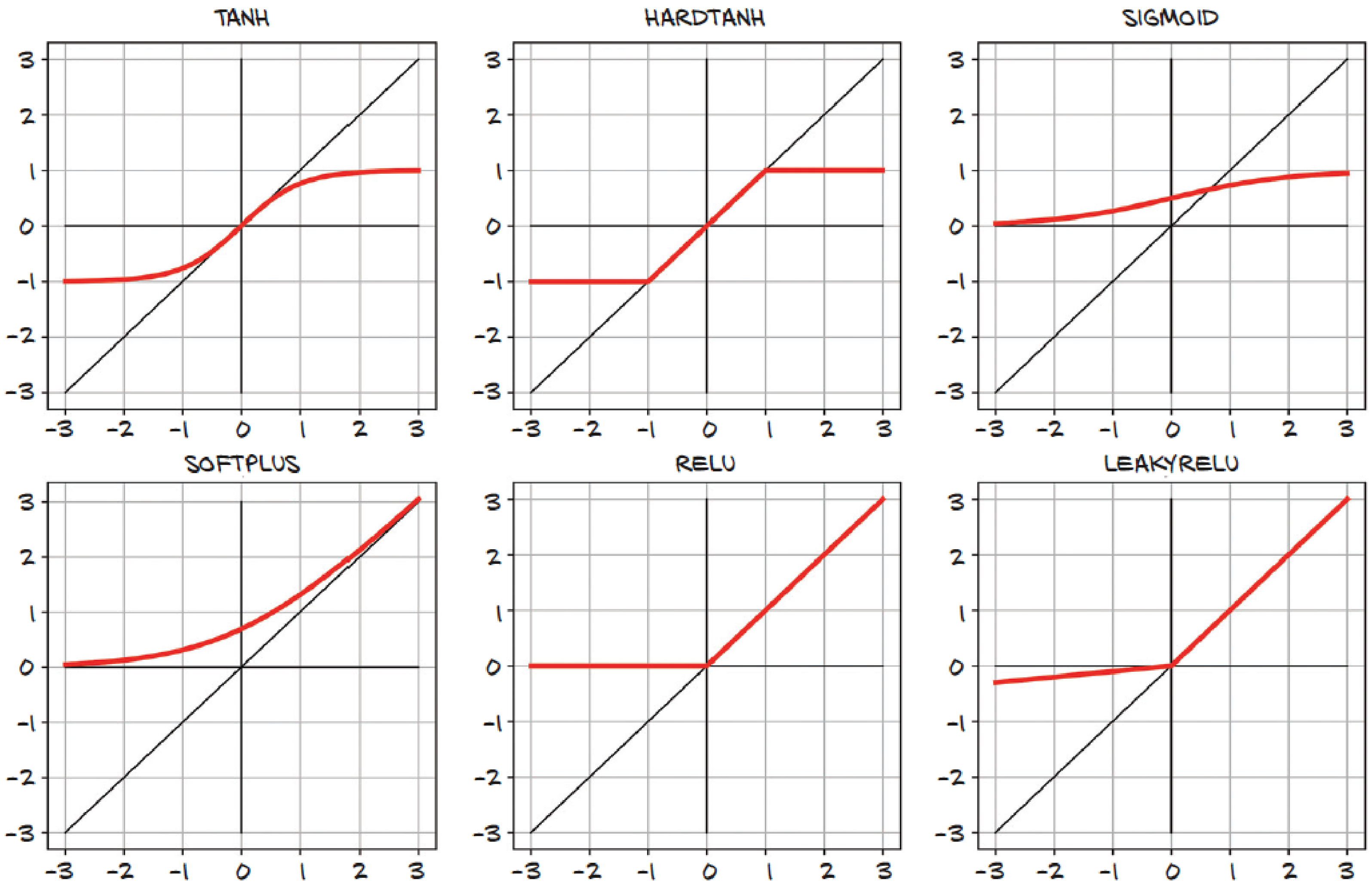


$$\mathrm{relu}(x) = \max(0, x)$$

$$\mathrm{relu}'(x) = 1_{x>0}$$

NEURAL NETWORKS

Activation Functions



NEURAL NETWORKS
Activation Functions

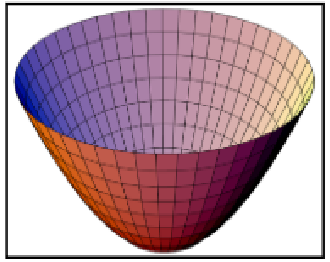
Largest difference between simple ML models and neural networks is:

– Nonlinearity of neural network causes interesting loss functions to be non-convex

Logistic Regression Loss:

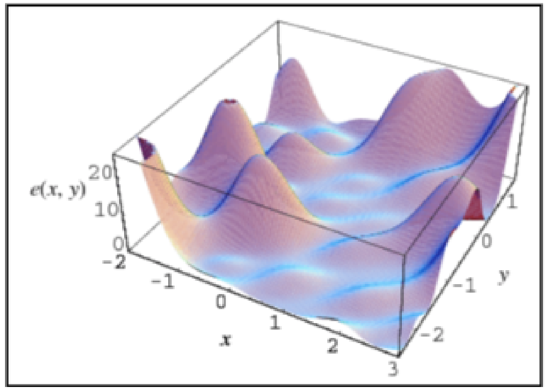
Linear Regression with Basis Functions:

$$E_D(\mathrm{w}) = \frac{1}{2}\sum_{n=1}^{N}\{ t_n - \mathrm{w}^T \varphi(x_n) \}^2$$



Neural Network Loss:

$$J(\boldsymbol{\theta}) = -E_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{data}} \log p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x})$$



Use iterative gradient-based optimizers that merely drives cost to low value, rather than

- Exact linear equation solvers used for linear regression or
- convex optimization algorithms used for logistic regression or SVMs