



MACHINE LEARNING
University Master's Degree in Intelligent Systems

artificial neural networks

Ramón A. Mollineda Cárdenas

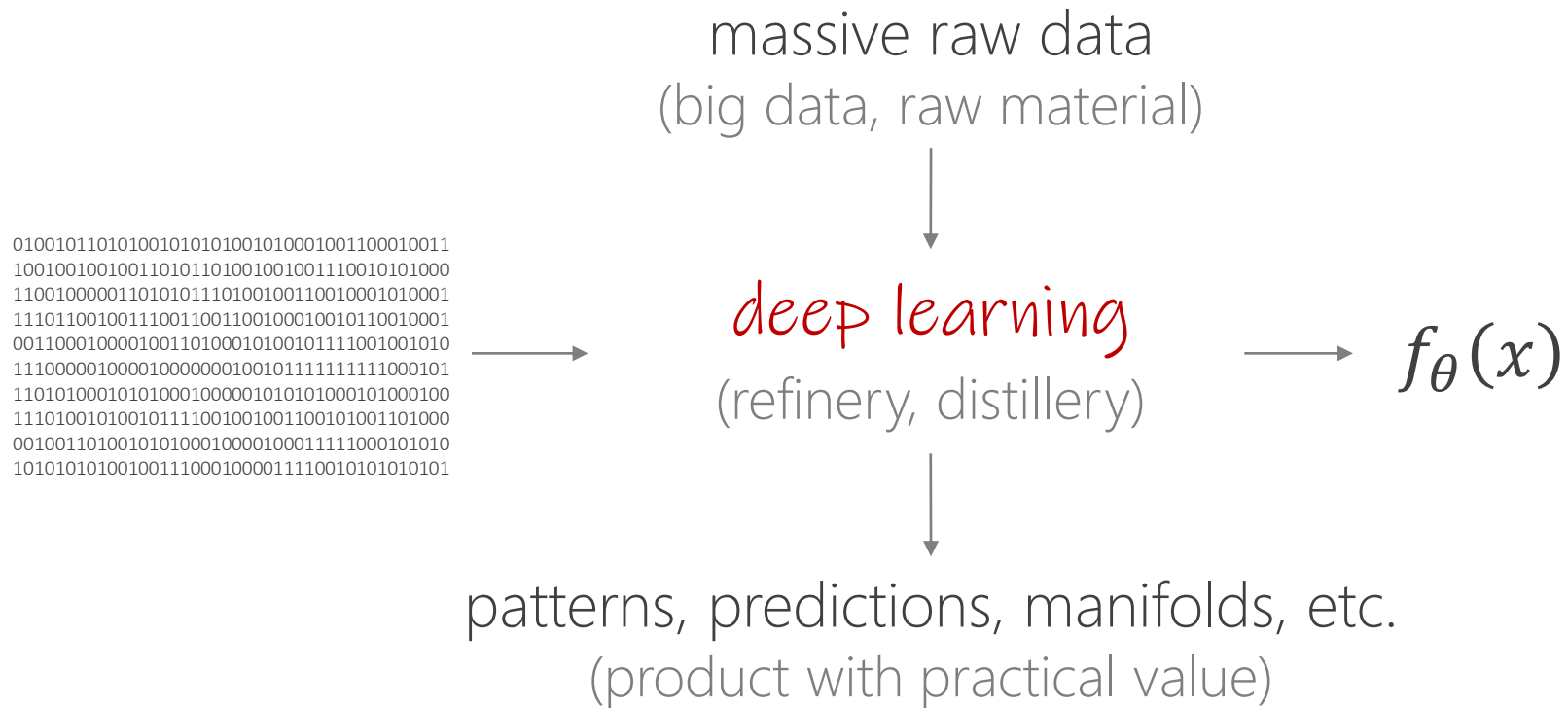
fuel of the future

The Economist, May 6th 2017

<https://www.economist.com/briefing/2017/05/06/data-is-giving-rise-to-a-new-economy>.

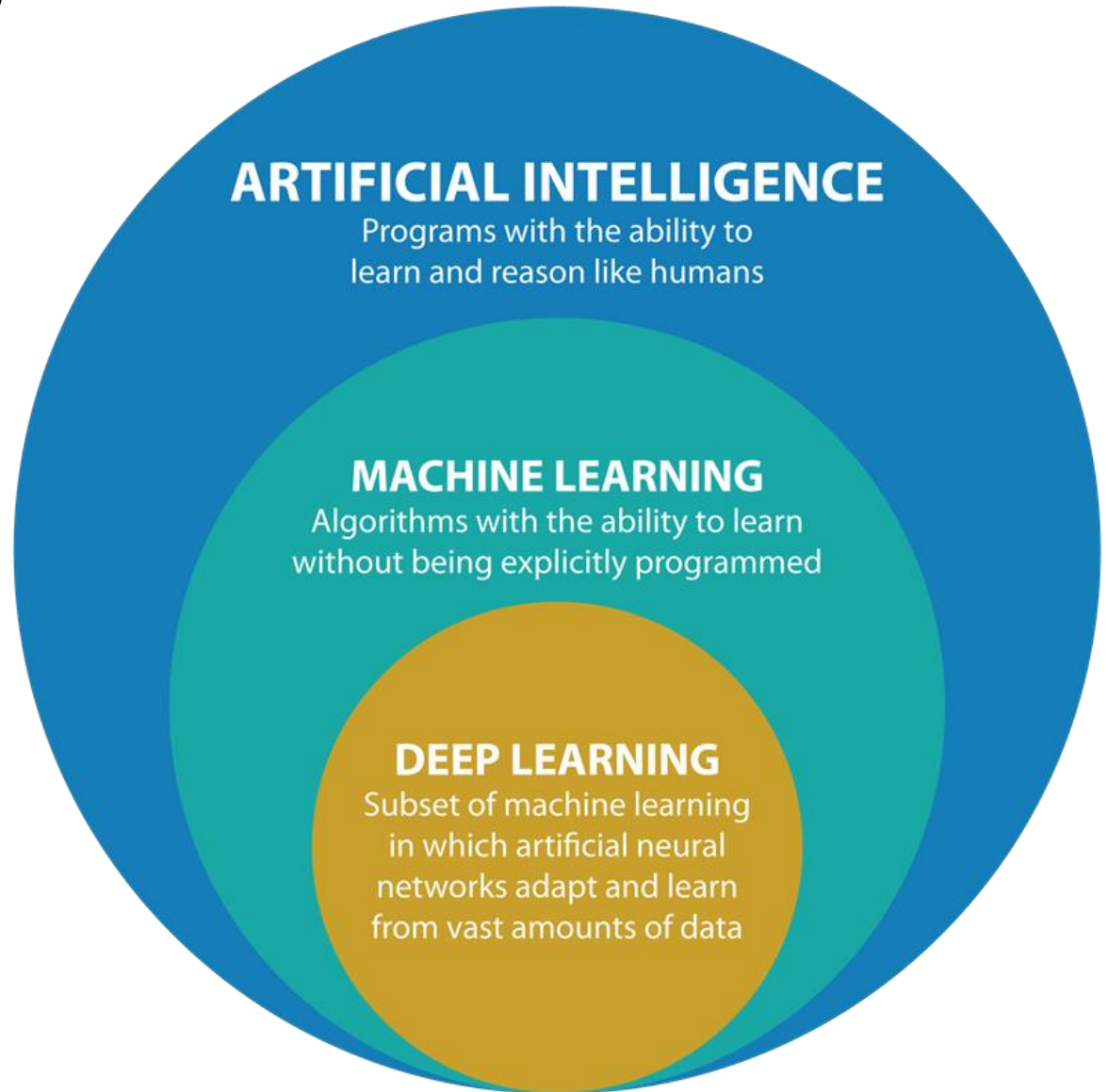
“AN OIL refinery... Data centres... the two have much in common. For one thing, both are stuffed with pipes. In refineries these collect petrol, propane and other components of crude oil ... In big data centres ... tens of thousands of computers ... extract value — patterns, predictions and other insights— from raw digital information.”

deep learning



deep learning

some context



deep learning

artificial neural networks outperform humans

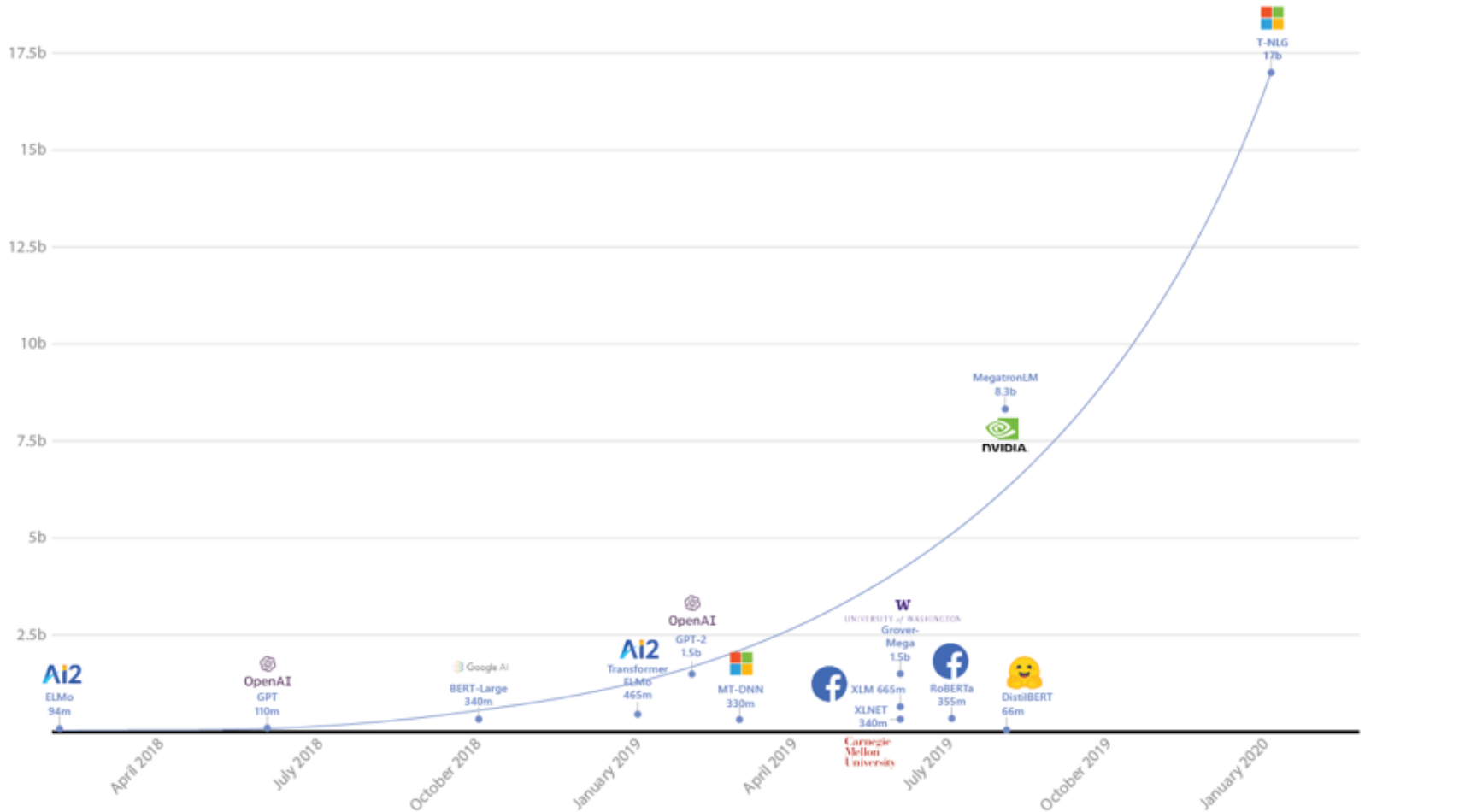
- object and image recognition ([ILSVRC](#), [¿aceptas el reto?](#))
- imitation of art styles ([DeepArt.io](#), [transferencia de estilo](#))
- medical diagnosis ([DL 95% - senior doctors 87%](#) in the diagnosis of malignant melanomas -distinguishing them from benign moles-)
- computer gaming ([modelos que aprenden](#), [AlphaGo vs. Lee Sedol](#))
- lipreading ([LipNet 95% - humanos 52%](#) according to grammar "command(4) + color(4) + preposition(4) + letter(25) + digit(10) + adverb(4)")
- big data analytics ([demografía a partir de Google Street View](#)).

"Using deep learning ..., we determined the make, model, and year of all motor vehicles encountered Data from this census of motor vehicles, which enumerated 22M automobiles in total (8% of all automobiles in the US), was used to accurately estimate income, race, education, and voting patterns ..." ([Gebre et al. 2017](#))

deep learning

natural language generation models

i billones (10^9) de parámetros!



applications to summarize texts, generate responses (chatbots), write stories, etc.

[news](#)

IRIS Data Set

classes

iris setosa



petal

sepal

iris versicolor



petal

sepal

iris virginica



petal

sepal

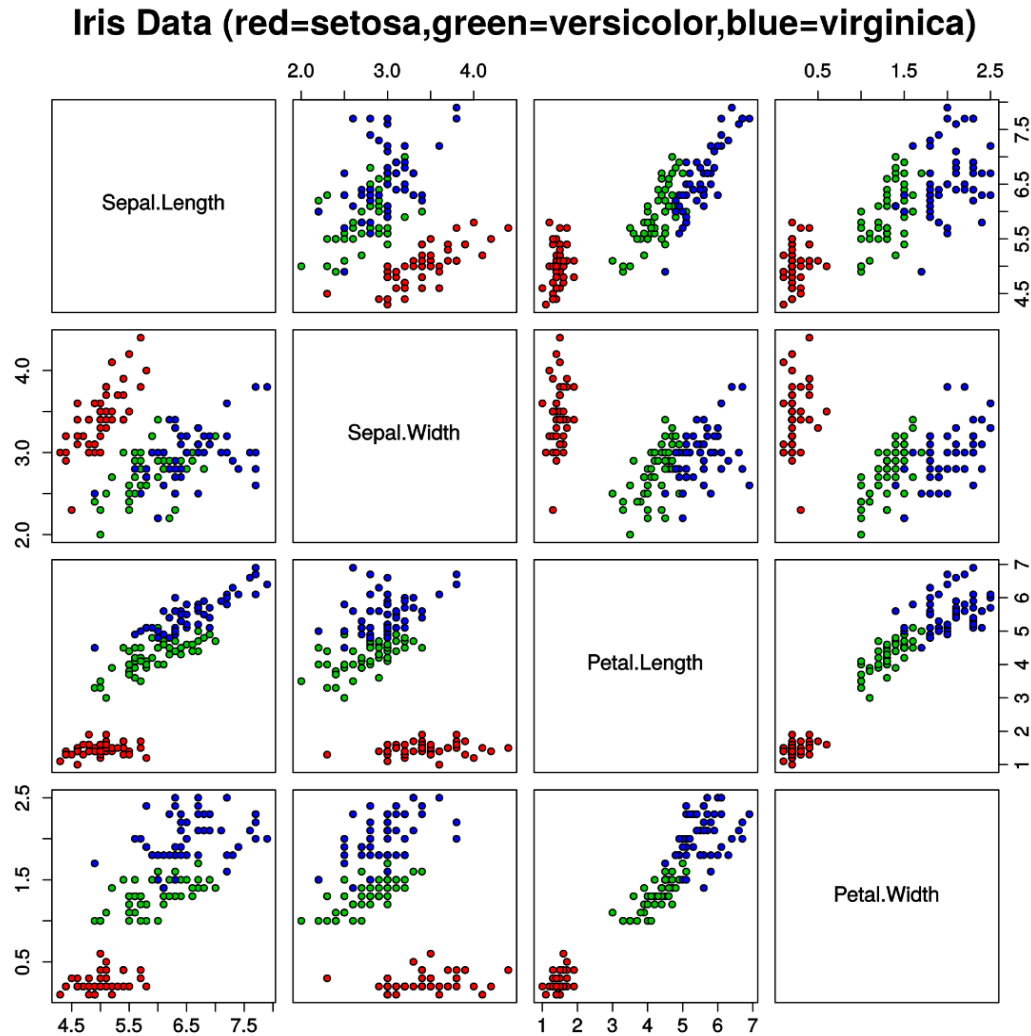
IRIS Data Set

description

- most popular database in machine learning
- 150 examples distributed among 3 classes :
 - Iris Versicolor (50)
 - Iris Setosa (50)
 - Iris Virginica (50)
- examples described by 4 measurements (in cm):
 - sepal length
 - sepal width
 - petal length
 - petal width

IRIS Data Set

distributions by classes in 2D subspaces



https://upload.wikimedia.org/wikipedia/commons/thumb/5/56/Iris_dataset_scatterplot.svg/2000px-Iris_dataset_scatterplot.svg.png

IRIS Data Set

original paper

Fisher, R.A. (1936). The use of multiple measurements in taxonomic problems. Annual Eugenics, 7, Part II, 179-188 ([acceder](#)).

linear model

a fully connected neural network f is a composition of nonlinear transformations of linear models (linear combinations of features)

$$f(x|w_k, \dots w_2, w_1) = g^* \left(w_k^t g_{k-1} \left(\dots g_2 \left(w_2^t g_1 (w_1^t x) \right) \right) \right)$$


being...

- x , the input vector to the network
- k , the number of layers (or transformations)
- w_i , weight matrix of the layer i , $1 \leq i \leq k$
- g_i , nonlinear activation function of the layer i (they are generally the same)
- g^* , output layer activation function (can be the identity function)
- f , network function composed from chains of linear and nonlinear functions

linear model

a fully connected neural network f is a composition of nonlinear transformations of linear models (linear combinations of features)

$$f(x|w_k, \dots, w_2, w_1) = g^* \left(w_k^t g_{k-1} \left(\dots g_2 \left(w_2^t g_1 (w_1^t x) \right) \right) \right)$$

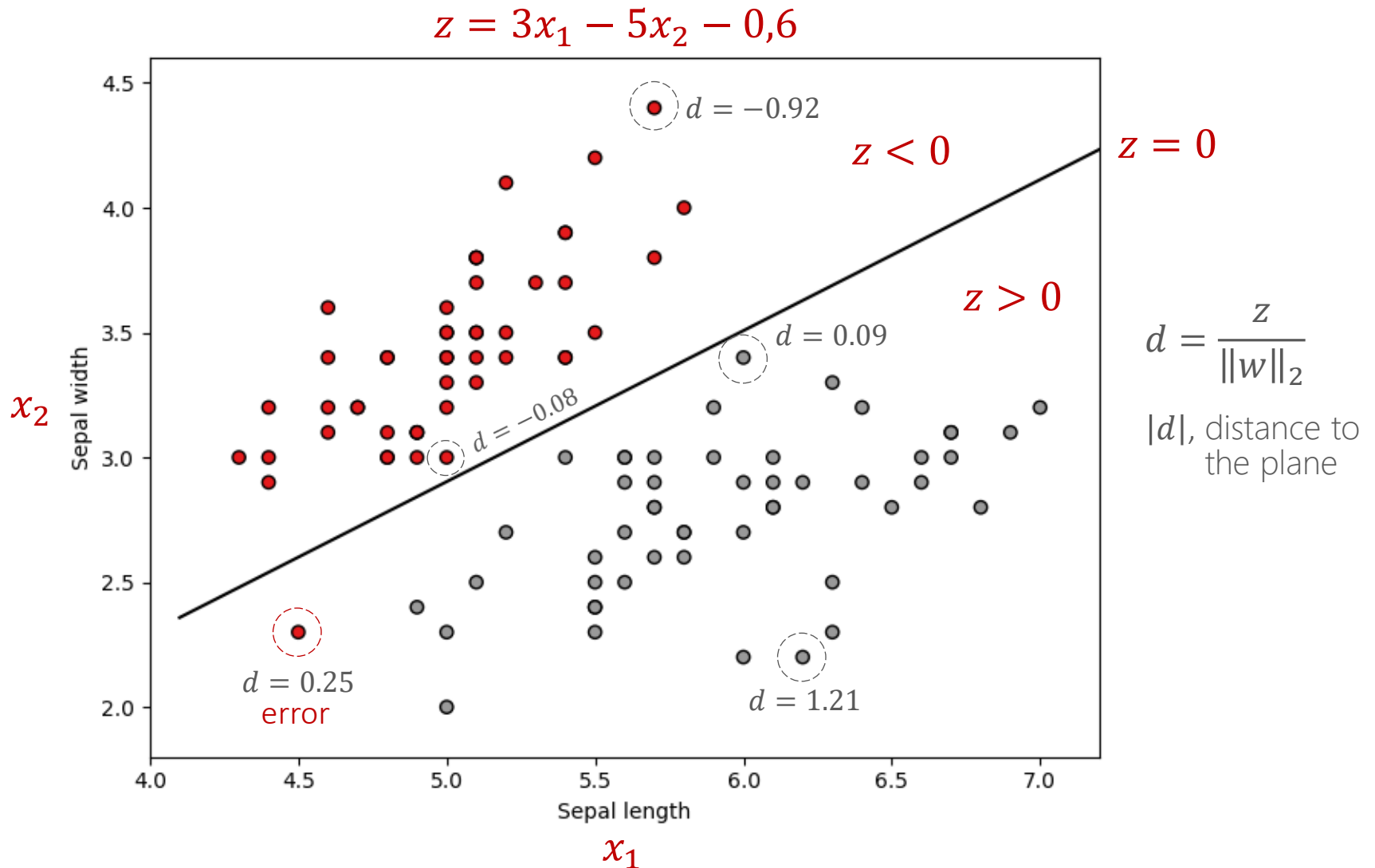
linear model 

being...

- x , the input vector to the network
- k , the number of layers (or transformations)
- w_i , weight matrix of the layer i , $1 \leq i \leq k$
- g_i , nonlinear activation function of the layer i (they are generally the same)
- g^* , output layer activation function (can be the identity function)
- f , network function composed from chains of linear and nonlinear functions

linear model

classes *Setosa* (red) and *Versicolor* (gray)



linear model

affine/linear transformation

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

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

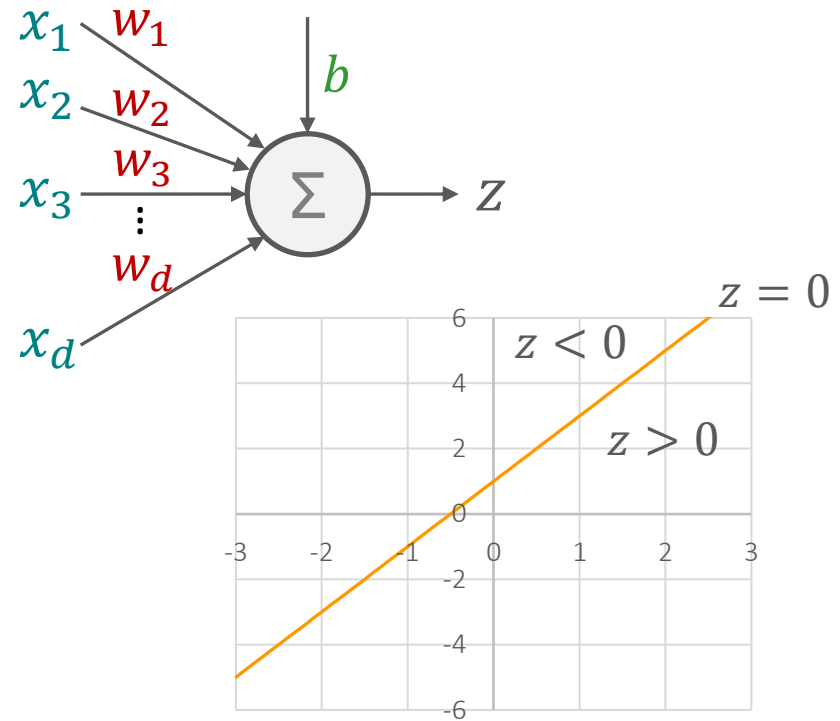
being:

\mathbf{w} , weight vector

\mathbf{x} , input data vector

b , scalar; bias, threshold

z , scalar; affine transformation output



$$z = 2x_1 - x_2 + 1$$

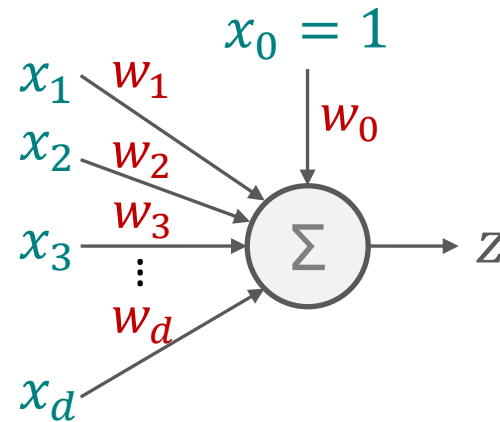
linear division of space

linear model

affine/linear transformation: compact notation

$$z = \mathbf{w}^T \mathbf{x}$$

$$z = \sum_{i=0}^d \mathbf{w}_i \mathbf{x}_i$$



being:

\mathbf{w} , weight vector (\mathbf{w}_0 , bias)

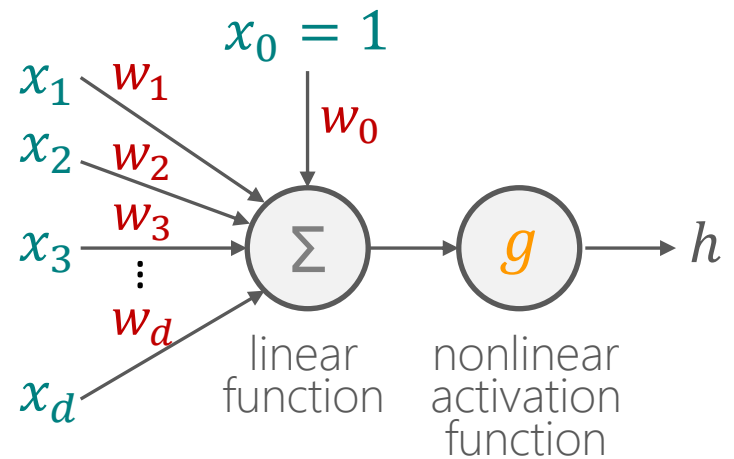
\mathbf{x} , input data vector

z , scalar; affine transformation output

linear model

activation function

$$h = g(z) = g(\mathbf{w}^T \mathbf{x})$$



being:

z , scalar; affine transformation output

g , nonlinear activation function

h , scalar; nonlinear function output

linear model

Perceptron (Frank Rosenblatt, 1958)

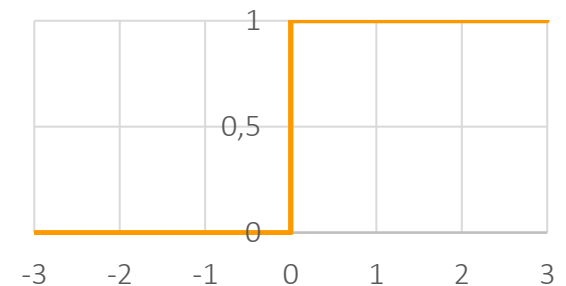
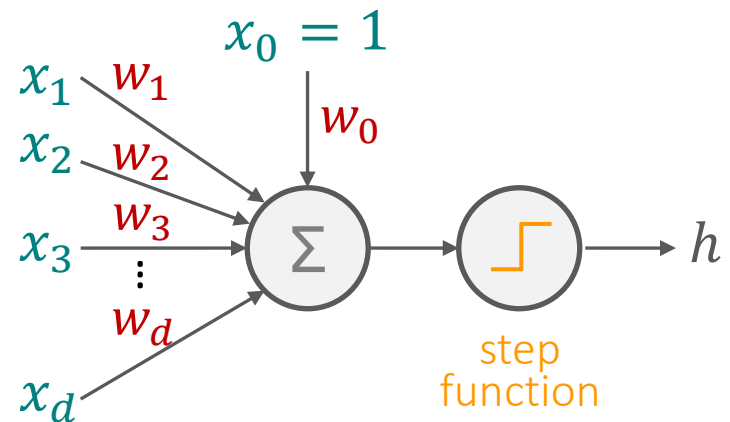
$$h = g(z) = g(\mathbf{w}^T \mathbf{x})$$

being:

z , affine transformation output

g , step function

$h \in \{0,1\}$

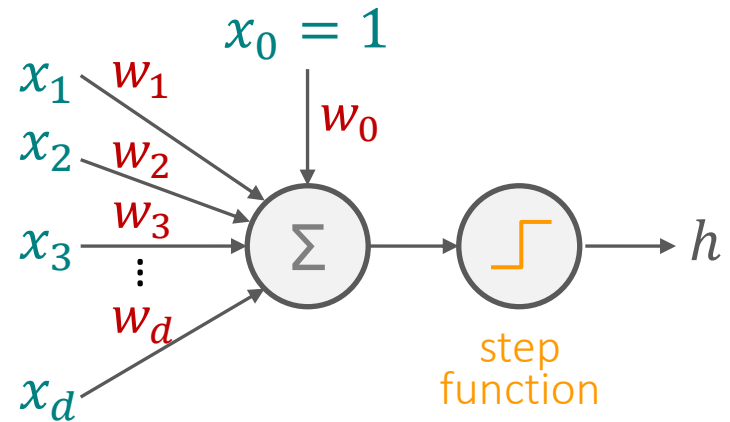


$$g(z) = \begin{cases} 1, & \text{si } z \geq 0 \\ 0, & \text{si } z < 0 \end{cases}$$

linear model

Perceptron (Frank Rosenblatt, 1958)

$$h = g(z) = g(\mathbf{w}^T \mathbf{x})$$

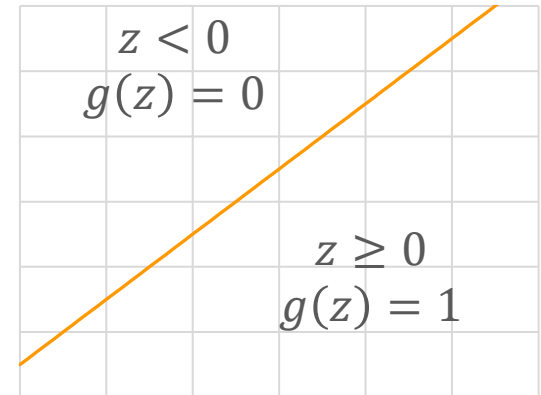


being:

z , affine transformation output

g , step function

$$h \in \{0,1\}$$

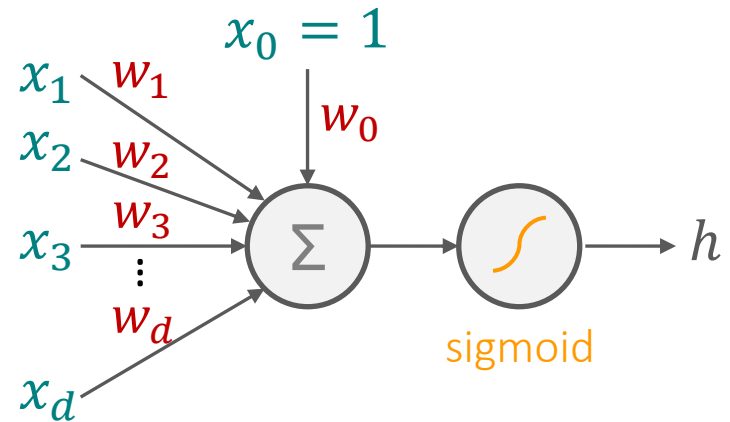


$$z = 2x_1 - x_2 + 1$$

linear model

logistic regression: a linear decision boundary

$$h = g(z) = g(w^T x)$$



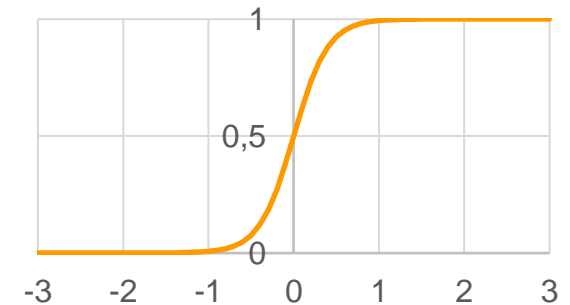
being:

z , affine transformation output

g , sigmoid function

$$h \in (0,1)$$

$$h = Pr(y = 1|x, w)$$



$$g(z) = \frac{1}{1 + e^{-z}}$$

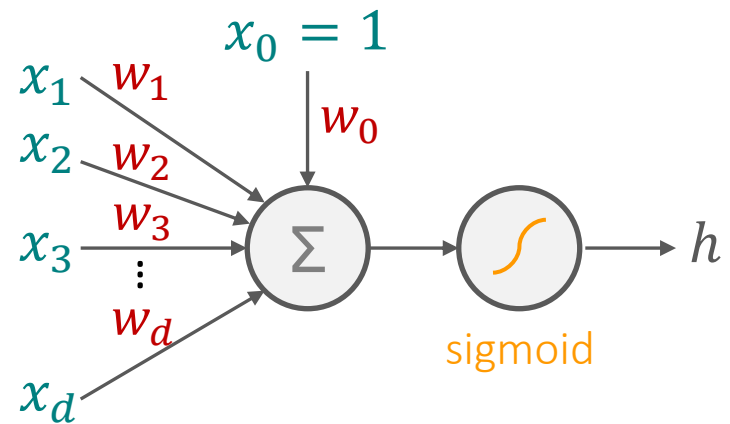
Why is logistic regression a linear classifier? ([link](#))

logistic regression

linear model

logistic regression: a linear decision boundary

$$h = g(z) = g(\mathbf{w}^T \mathbf{x})$$



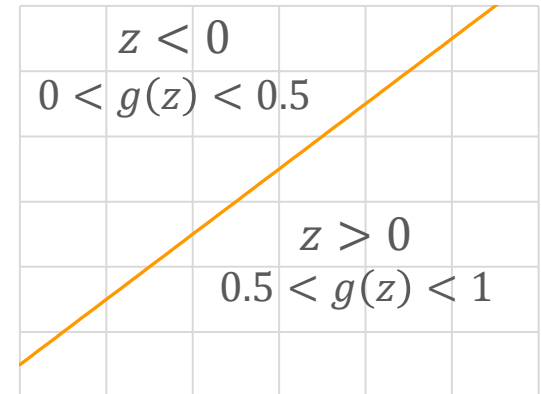
being:

z , affine transformation output

g , sigmoid function

$$h \in (0,1)$$

$$h = \Pr(y = 1 | \mathbf{x}, \mathbf{w})$$

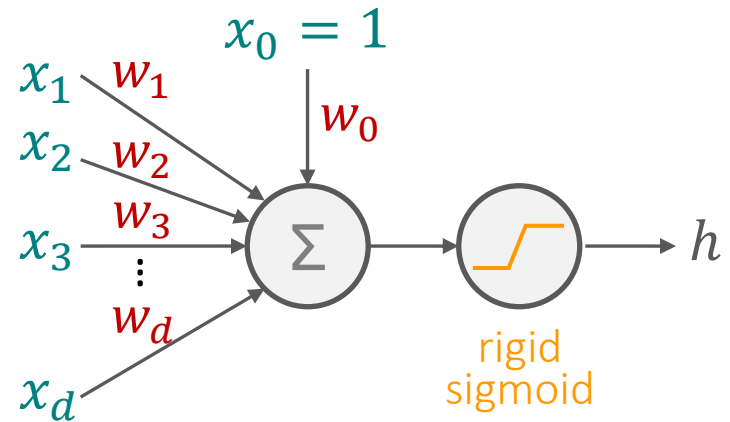


$$z = 2x_1 - x_2 + 1$$

linear model

approximate (or rigid) logistic regression

$$h = g(z) = g(\mathbf{w}^T \mathbf{x})$$



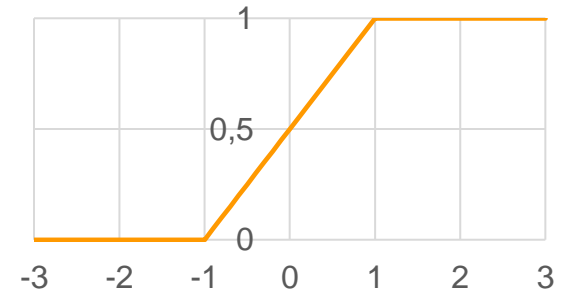
being:

z , affine transformation output

g , rigid sigmoid function

$$h \in (0,1)$$

$$h = Pr(y = 1 | \mathbf{x}, \mathbf{w})$$

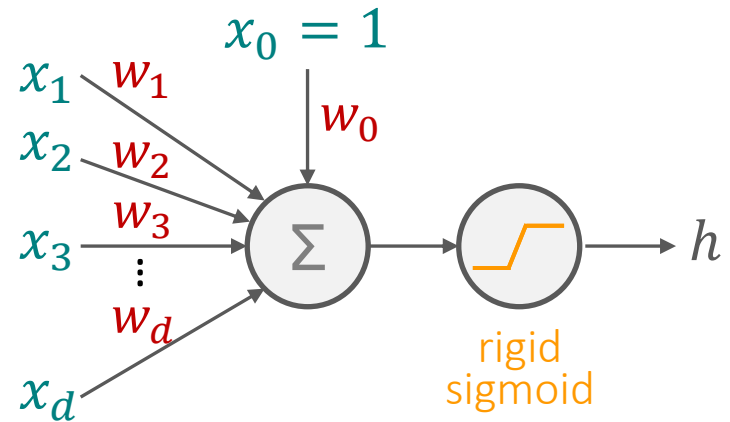


$$g(z) = \max \left(0, \min \left(1, \frac{z + 1}{2} \right) \right)$$

linear model

approximate (or rigid) logistic regression

$$h = g(z) = g(\mathbf{w}^T \mathbf{x})$$



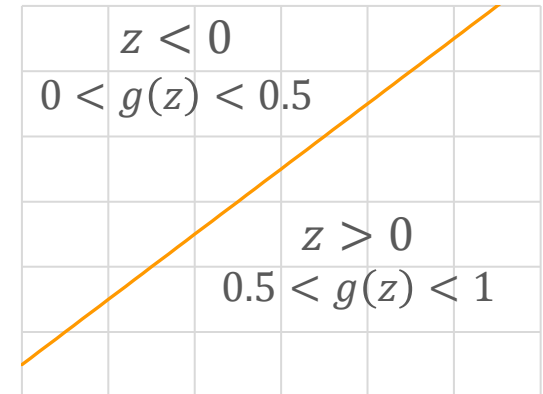
being:

z , affine transformation output

g , rigid sigmoid function

$$h \in (0,1)$$

$$h = \Pr(y = 1 | \mathbf{x}, \mathbf{w})$$



$$z = 2x_1 - x_2 + 1$$

linear model

approximate perceptron on Setosa and Versicolor classes

script simplificado usando numpy, sklearn y keras:

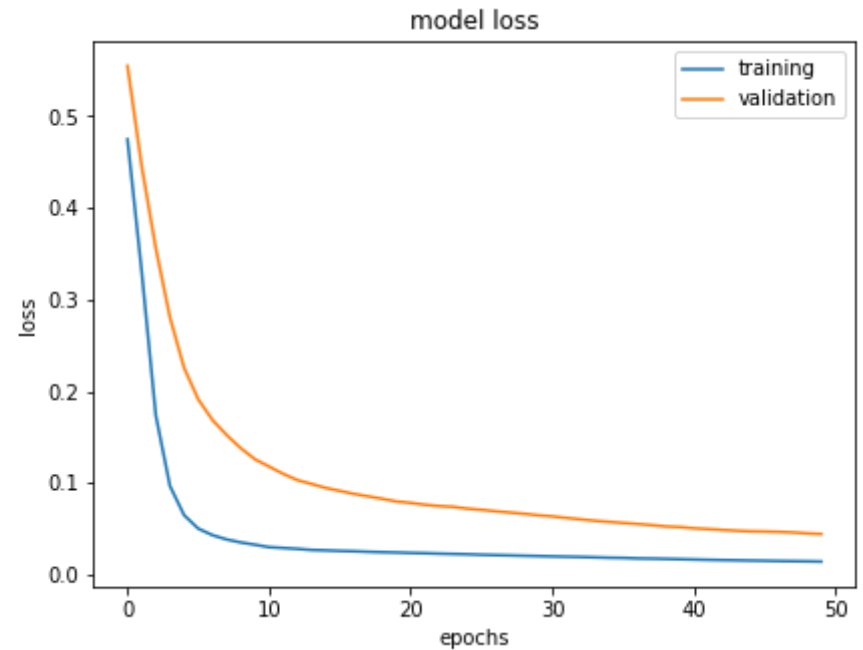
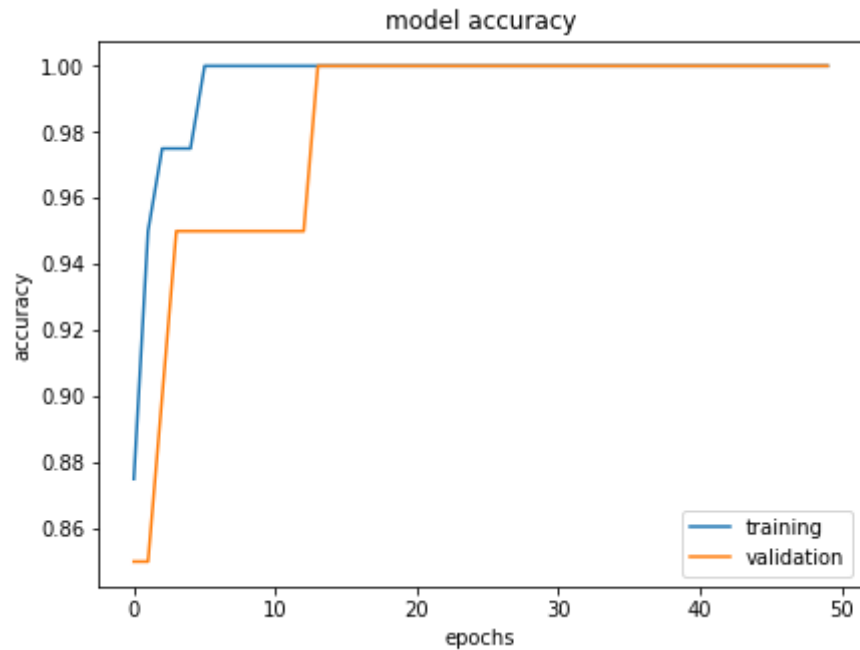
```
iris = sklearn.datasets.load_iris()
X = iris.data[:, :2] # first two features (sepal width, sepal length)
y = iris.target
X = (X-numpy.mean(X, axis=0))/numpy.std(X, axis=0) # data standardization

model = keras.models.Sequential()
model.add(keras.layers.Dense(1, input_shape=(2,),
                             activation=keras.activations.hard_sigmoid))

opt = keras.optimizers.SGD(learning_rate = 0.1, momentum=0.9)
model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
history = model.fit(X, y, epochs=50, batch_size=20, verbose=1,
                    validation_split=0.2)
```

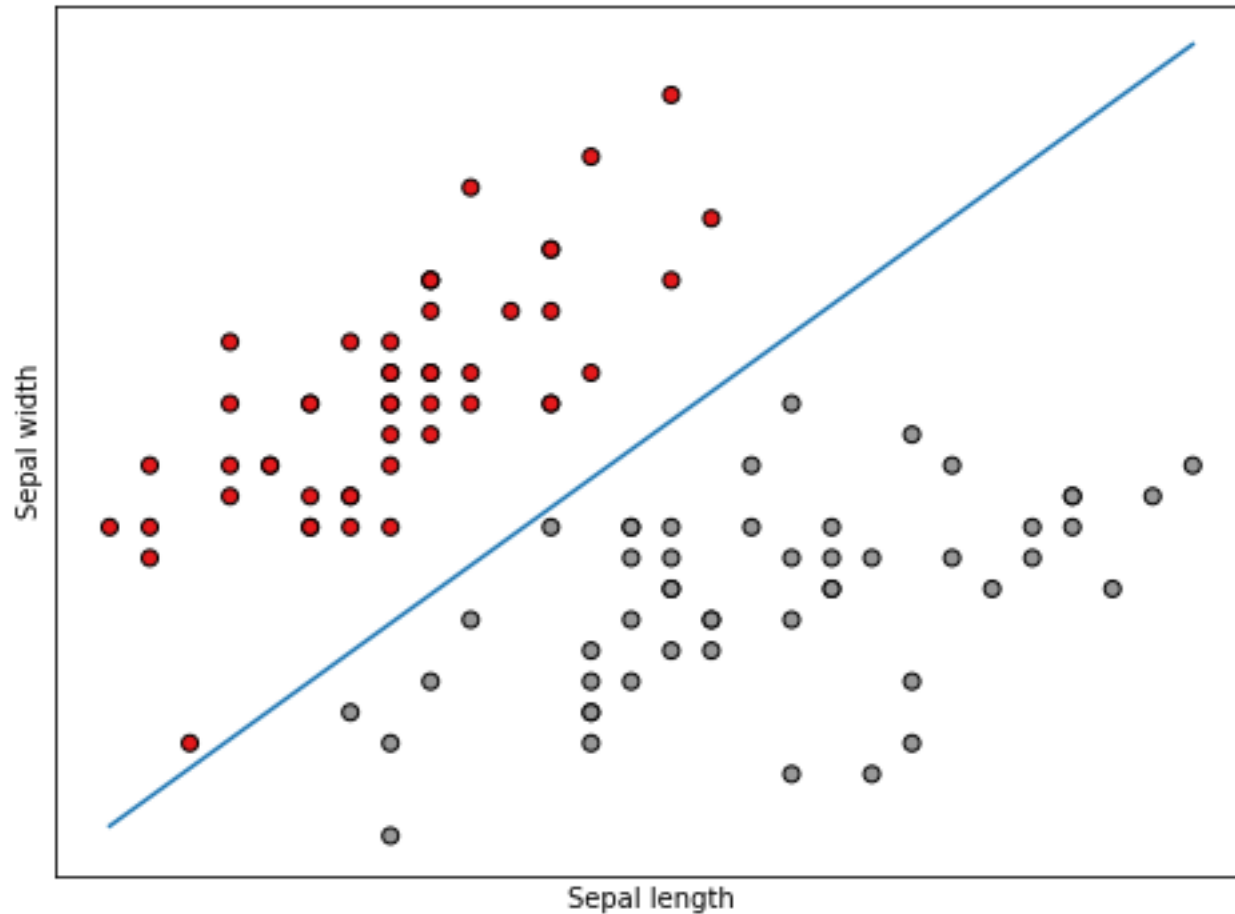
linear model

approximate perceptron on Setosa and Versicolor classes



linear model

approximate perceptron on Setosa and Versicolor classes

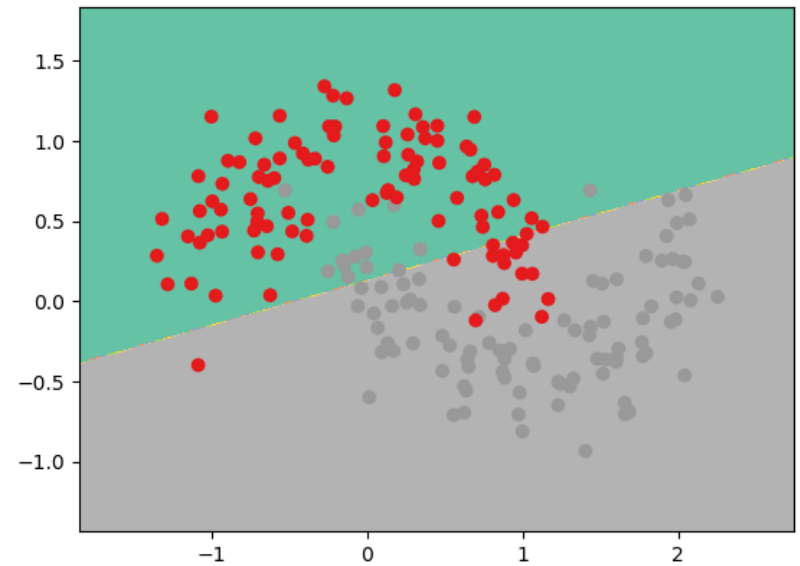
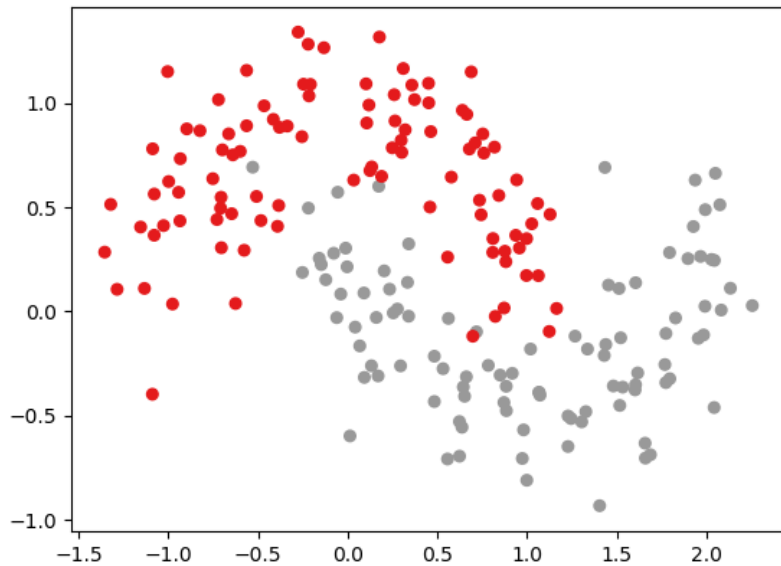


linear model

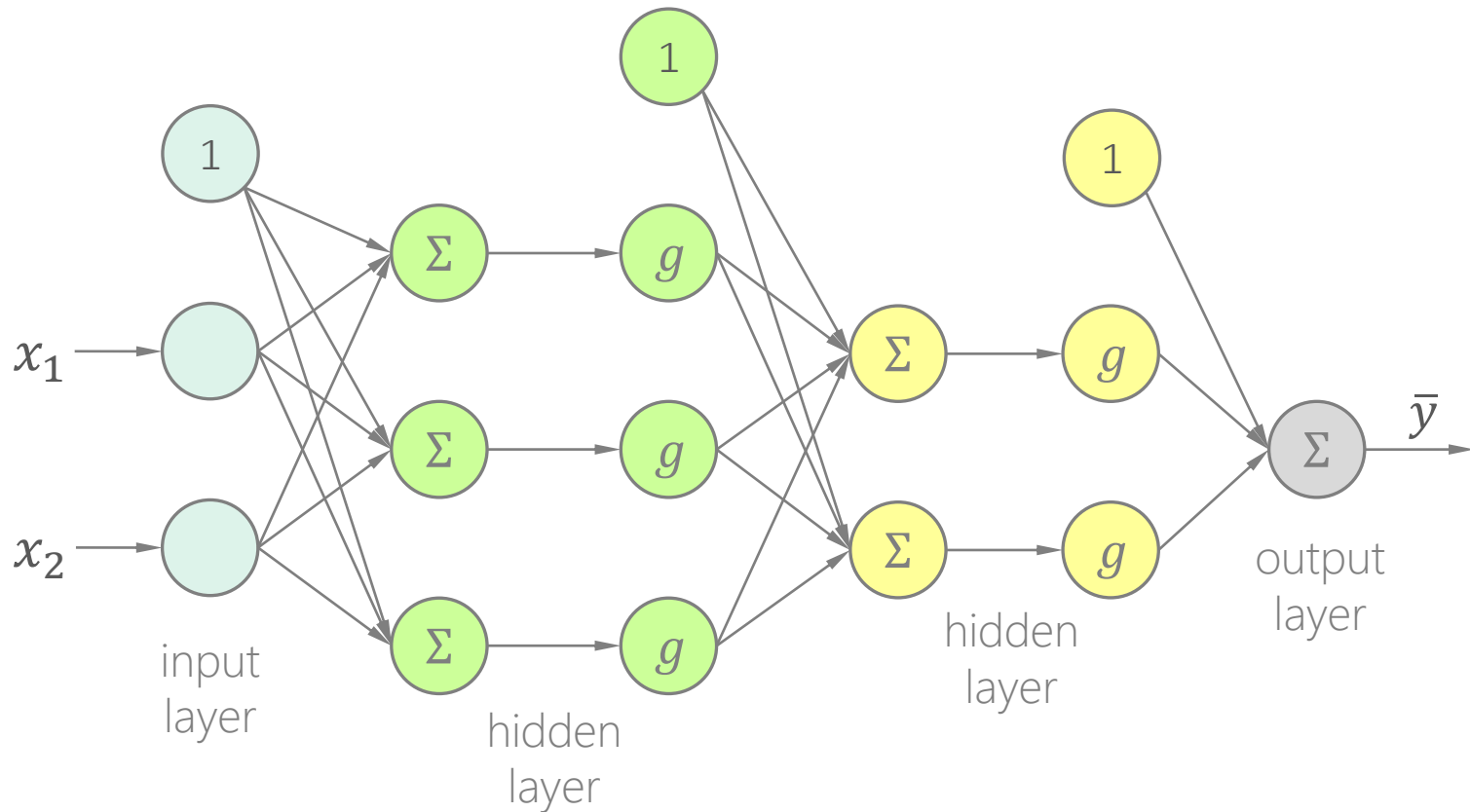
limitations

linear classifier

logistic regression => "sigmoid" activation function

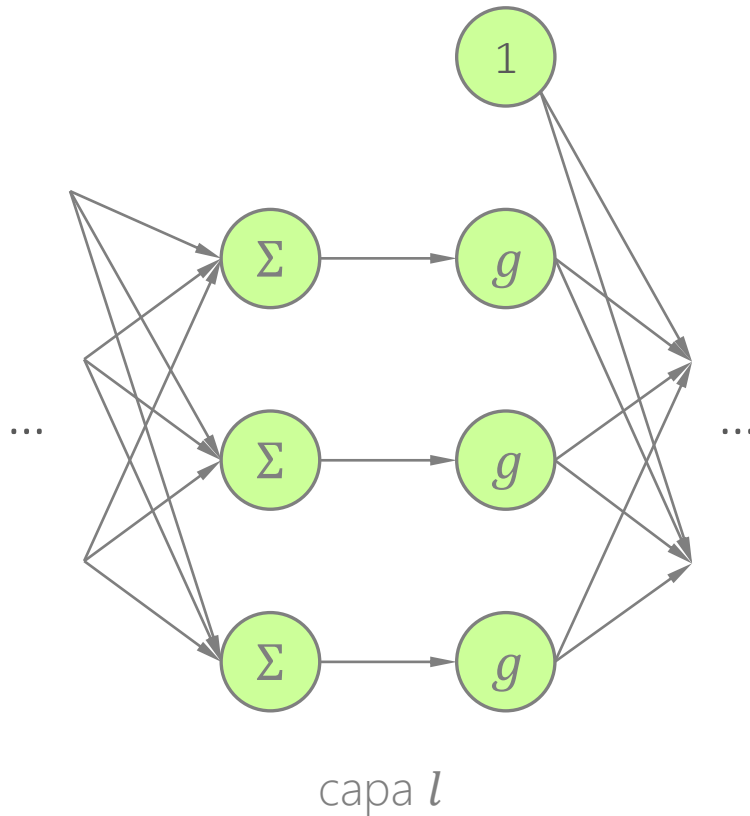


fully connected multilayer networks



g : función de activación (e.g. sigmoid, tanh, softmax, relu, ...)

fully connected multilayer networks

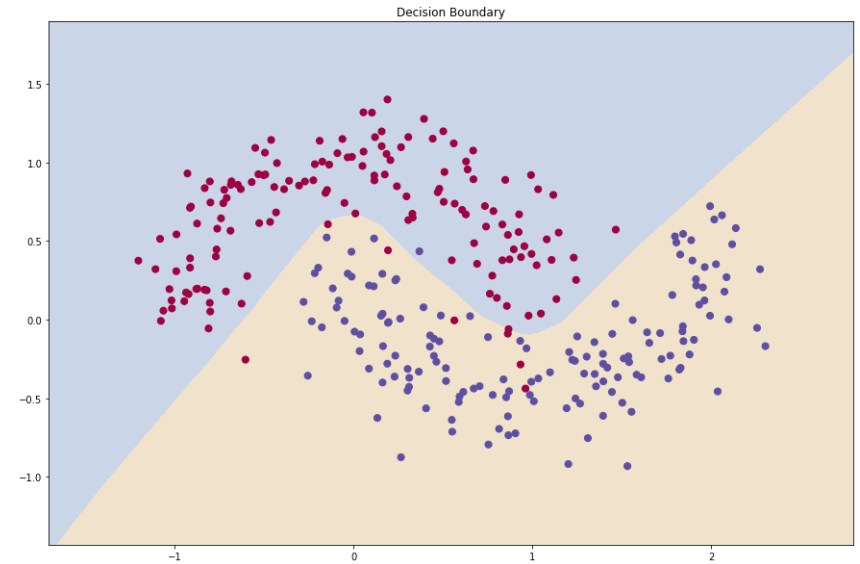
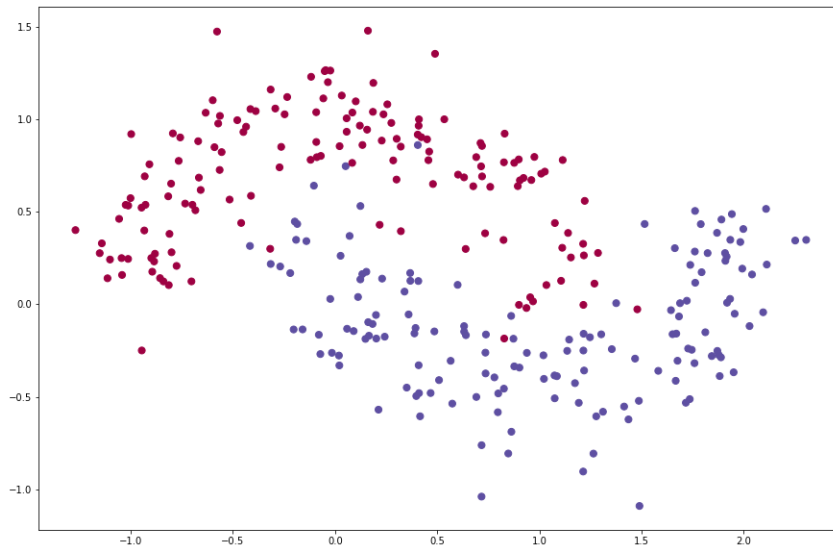


- cada capa consiste en:
función lineal + activación no lineal
- función de transformación en capa l :
$$z_l = W^l \cdot I_l$$
$$I_{l+1} = g(z_l)$$
- sea H_l el número de unidades de la capa l ; el tamaño de W^l sería ...
$$H_l \times H_{l-1}$$
- I_l es de tamaño $H_{l-1} \times 1$

fully connected multilayer networks

multilayer network

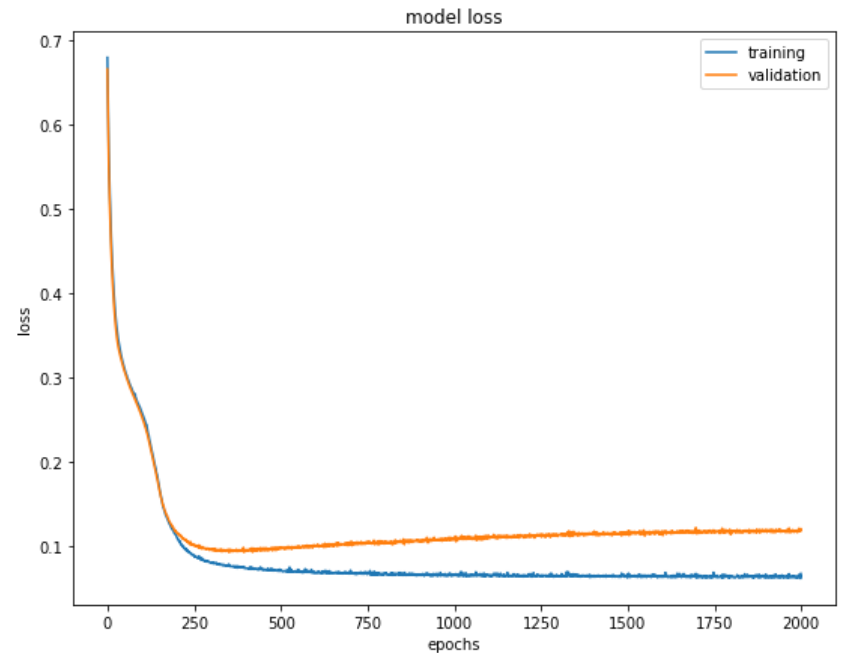
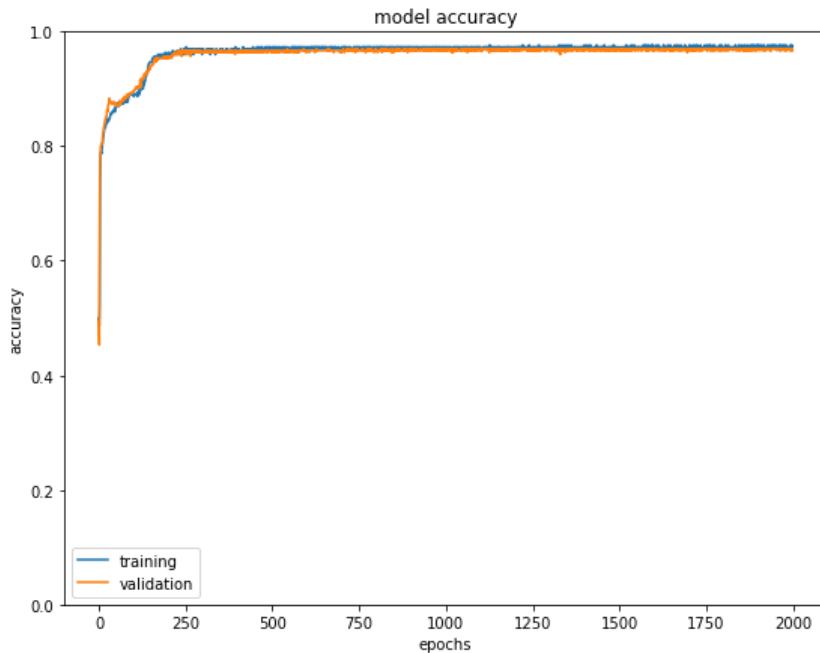
3 layers of 12+6+1 units, "relu" and "sigmoid" activation functions



fully connected multilayer networks

multilayer network

3 layers of 12+6+1 units, "relu" and "sigmoid" activation functions



backpropagation

learning goal

progressive adjustment (iterative optimization) of **weights that minimize the error/loss** of the network on a representative set of training samples.

potentially, a network with sufficient learning capacity could achieve zero error on the training set; however, it is not usually a desirable goal (overfitting risk).

backpropagation

training a neural network (mini-batches)

```
initialize network weights  $w(0)$  with small arbitrary values
for epoch = 1...K, do
  for batch = 1...N/batch_size, do
    batch <- randomly choose batch_size instances
    X, y <- preprocess(batch)
    z <- network(X) (forward execution)
     $\ell$  <- loss(z, y) backpropagation
    g <- gradients( $\ell$ , w) (backward execution)
     $w(t+1)$  <-  $w(t) - \gamma \cdot g$  (weight optimization/fitting)
  end for
end for
```


backpropagation

fundamentals: chain rule

some examples of composite of two (differentiable) functions $f(g(x))$:

- $f(g(x)) = (2x + 1)^3$
- $f(g(x)) = \sin(x^2)$
- $f(g(x)) = \sin(x)^2$

their derivative functions $f' = f'(g(x)) \cdot g'(x)$, or $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x}$

- $\frac{\partial f}{\partial x} = 3(2x + 1)^2 \cdot 2$
- $\frac{\partial f}{\partial x} = \cos(x^2) \cdot 2x$
- $\frac{\partial f}{\partial x} = 2 \cdot \sin(x) \cdot \cos(x)$

backpropagation

fundamentals: gradient descent

gradient descent (steepest descent) is an iterative optimization algorithm for finding a local minimum of a differentiable function.

strategy

- starting from an initial value of the parameter
- move “downhill” along the surface of the function, in the direction of the negative gradient, looking for a parameter value that minimizes it
- stop when the minimum of the function is reached

backpropagation

fundamentals: gradient descent

goal

to find $\theta = \theta^*$ such that $L(\theta^*)$ is the minimum value of L

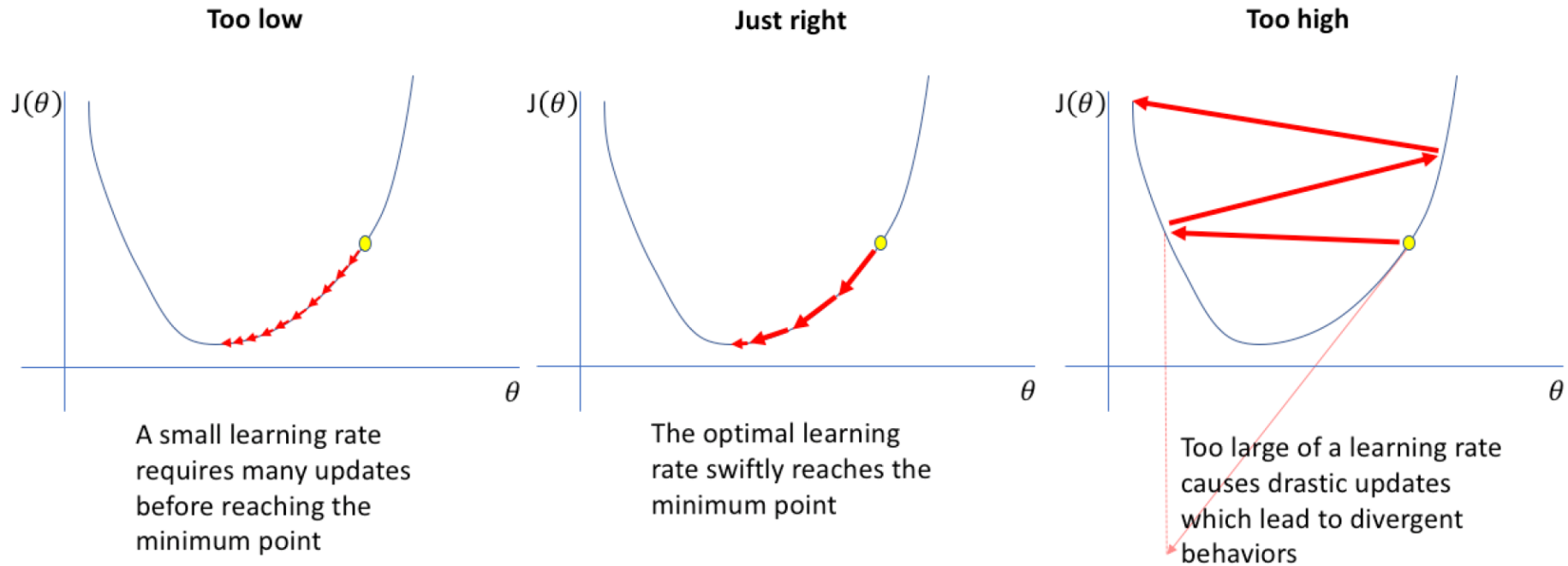
method

- let $L(\theta)$ be a function defined by the parameter θ
- initialization : $\theta_0 = \theta_{inicial}$
- updating: $\theta_{i+1} = \theta_i - \gamma \frac{\partial L}{\partial \theta}(\theta_i),$
 - $\gamma \ll 1$ is the learning rate; it determines the magnitude of change
- repeat as long as $L(\theta_{i+1}) < L(\theta_i)$
- solution: $\theta^* = \theta_i$

backpropagation

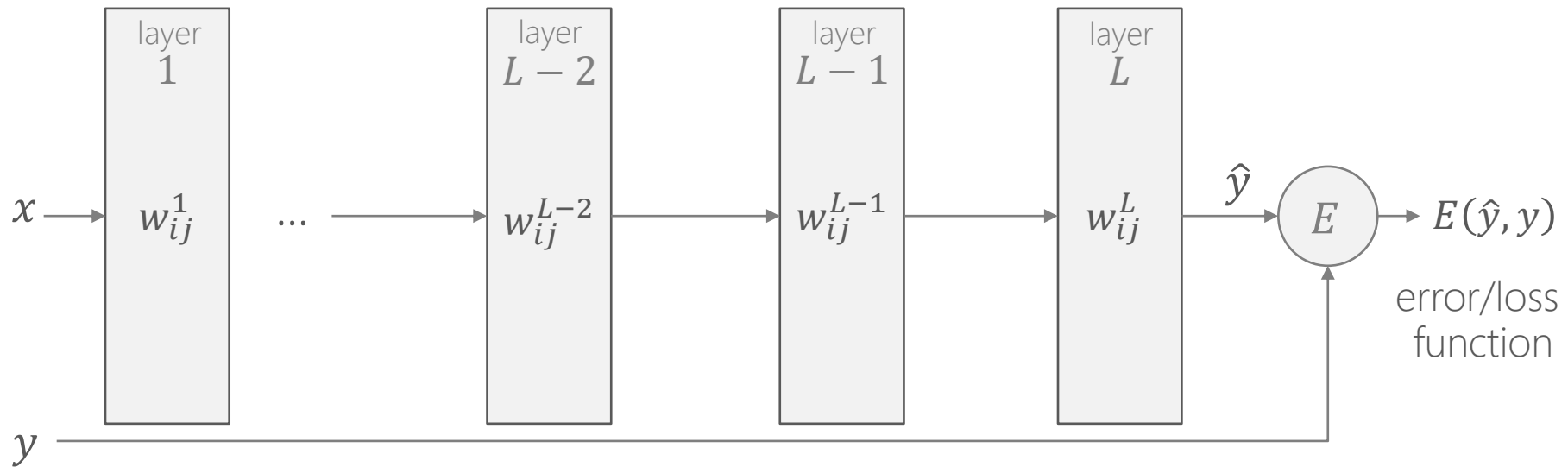
fundamentals: gradient descent

effect of the learning coefficient γ



backpropagation

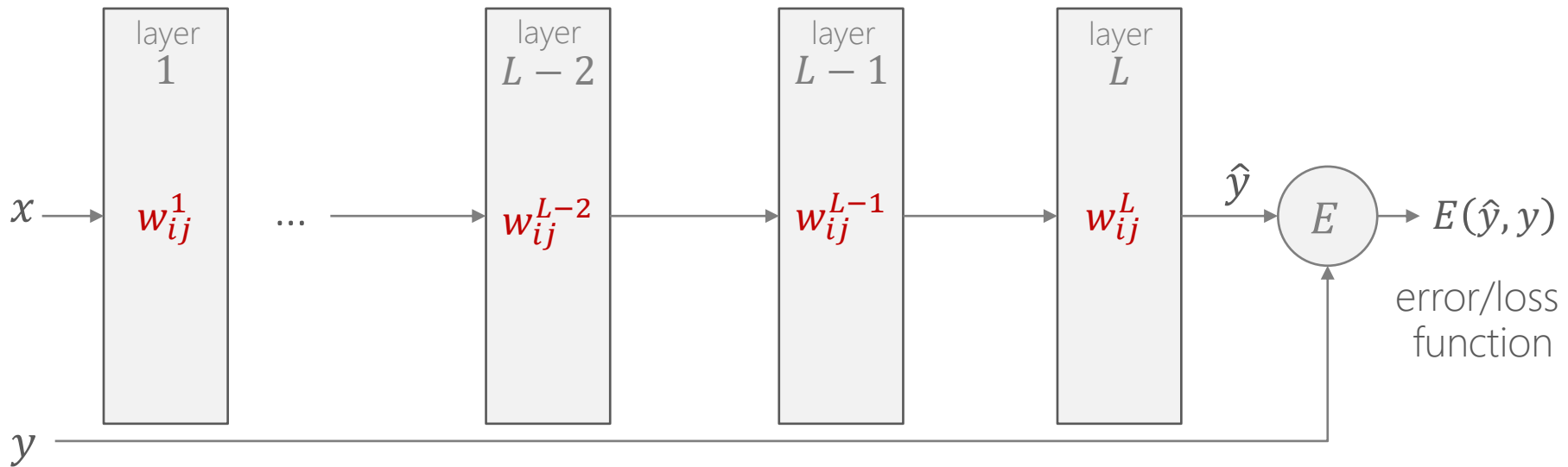
solution scheme



backpropagation

solution scheme

goal: $\frac{\partial E}{\partial w_{ij}^l}$ for all w_{ij}^l



backpropagation

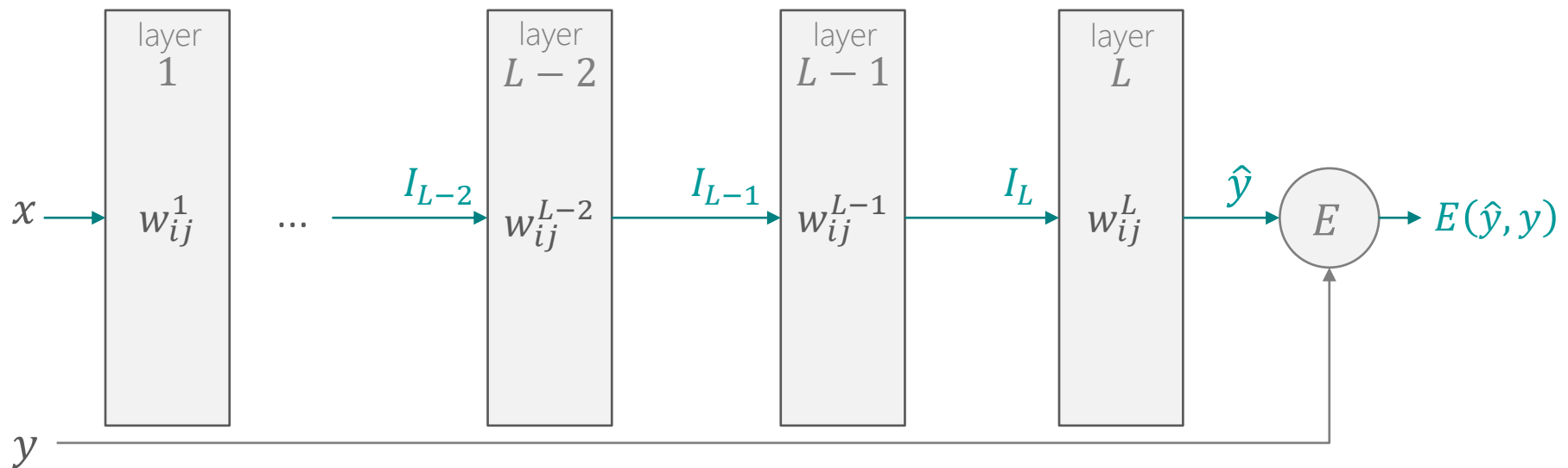
solution scheme

1. initialize weights, choose learning coefficient, choose stop criterion
2. create an arbitrary batch (X, y) --> (subset of instances, expected output).
3. forward execution: walk X through the network and get output z .
4. compute $\text{loss}(y, z)$
5. backward execution (**backpropagation**)
 - compute sensitivity coefficient $\delta_L = f(\delta_{L+1})$ from input to layer L .
 - compute gradients $\frac{\partial E}{\partial w_{ij}^{L-1}} = f(\delta_L)$
6. weight optimization
7. check stopping criteria; if it is fulfilled, then finish; otherwise, go to step 2.

backpropagation

solution scheme

forward execution



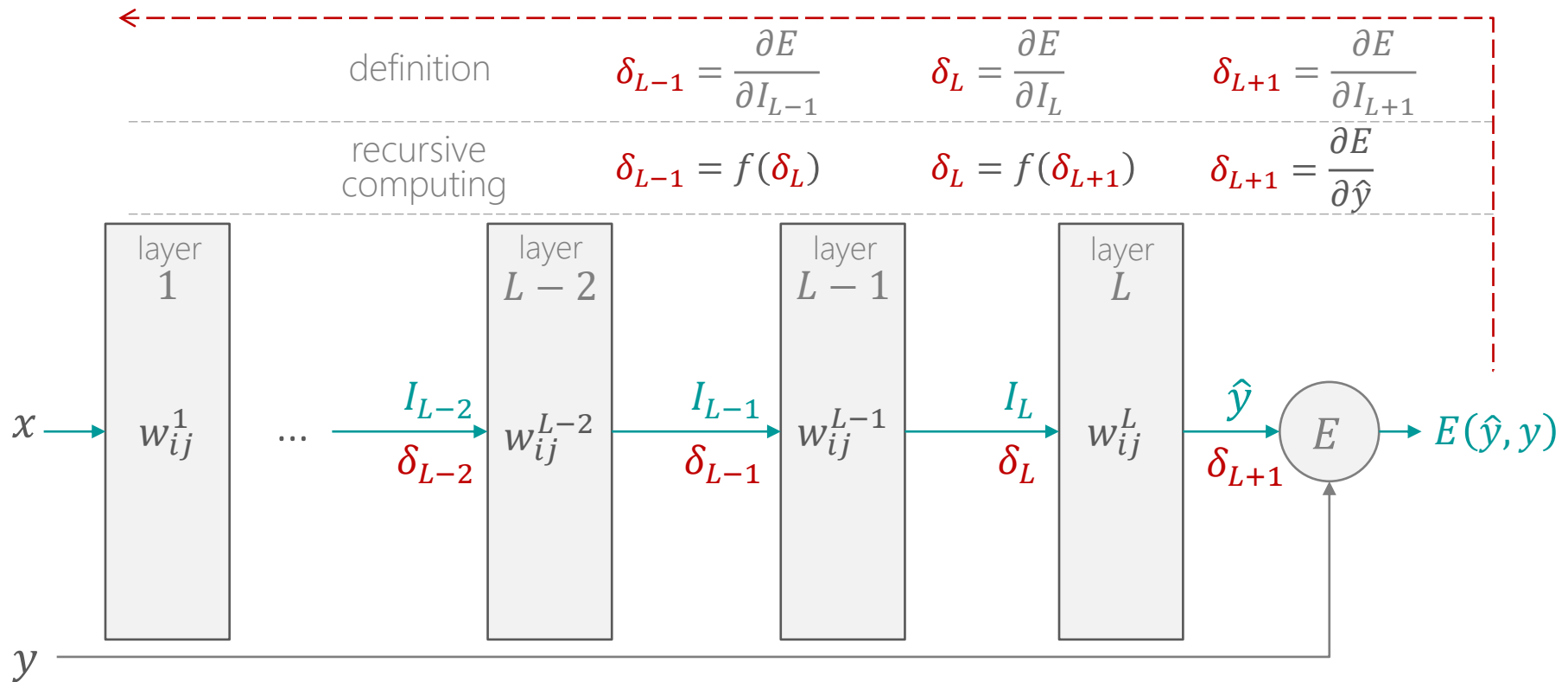
the following pieces of information are calculated:

- input I_l to each layer l (matches the output of the layer $l-1$),
- actual network output \hat{y}
- value of the error or loss function $E(\hat{y}, y)$

backpropagation

solution scheme

backward execution (calculation of sensitivities)



δ_l measures the sensitivity of E to changes in I_l

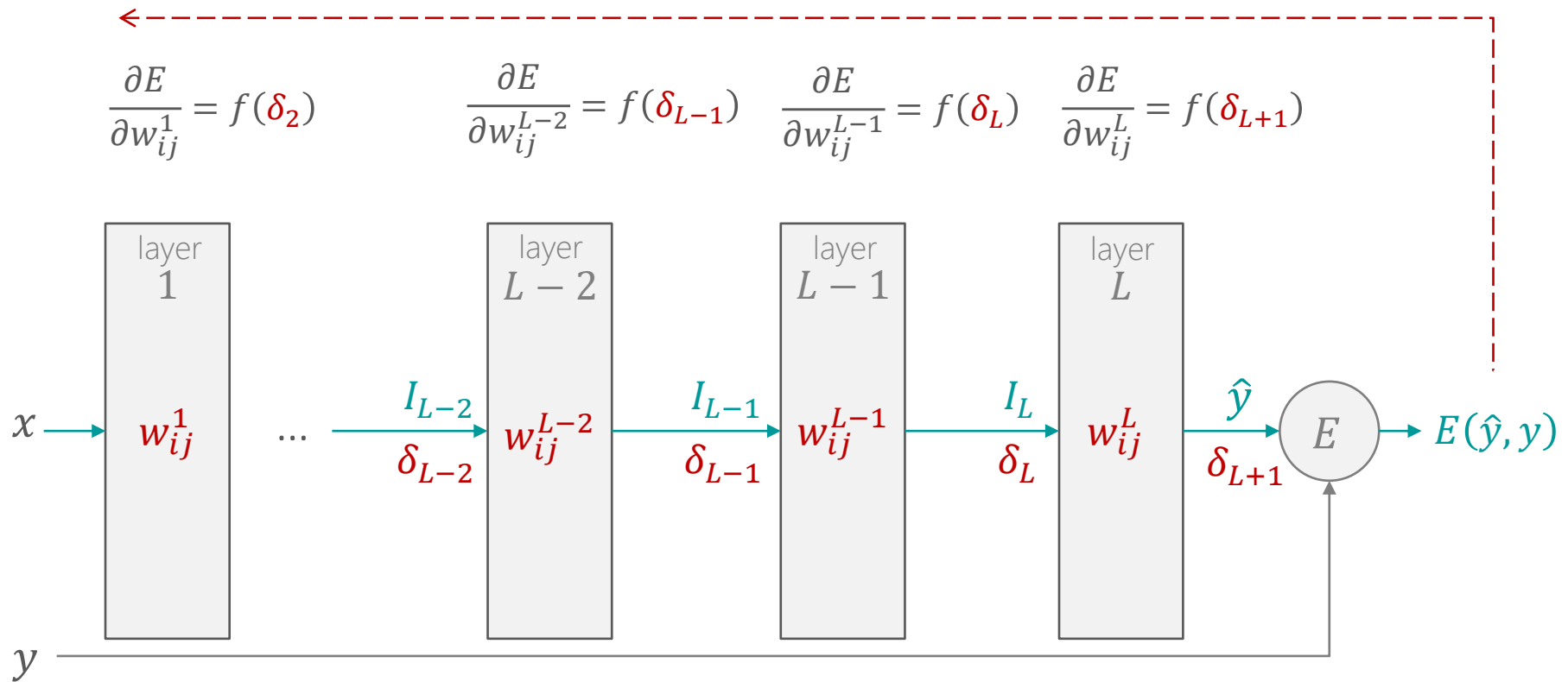
δ_l is computed recursively from δ_{l+1}

δ_l allows efficient computation of $\frac{\partial E}{\partial w_{ij}^{l-1}}$

backpropagation

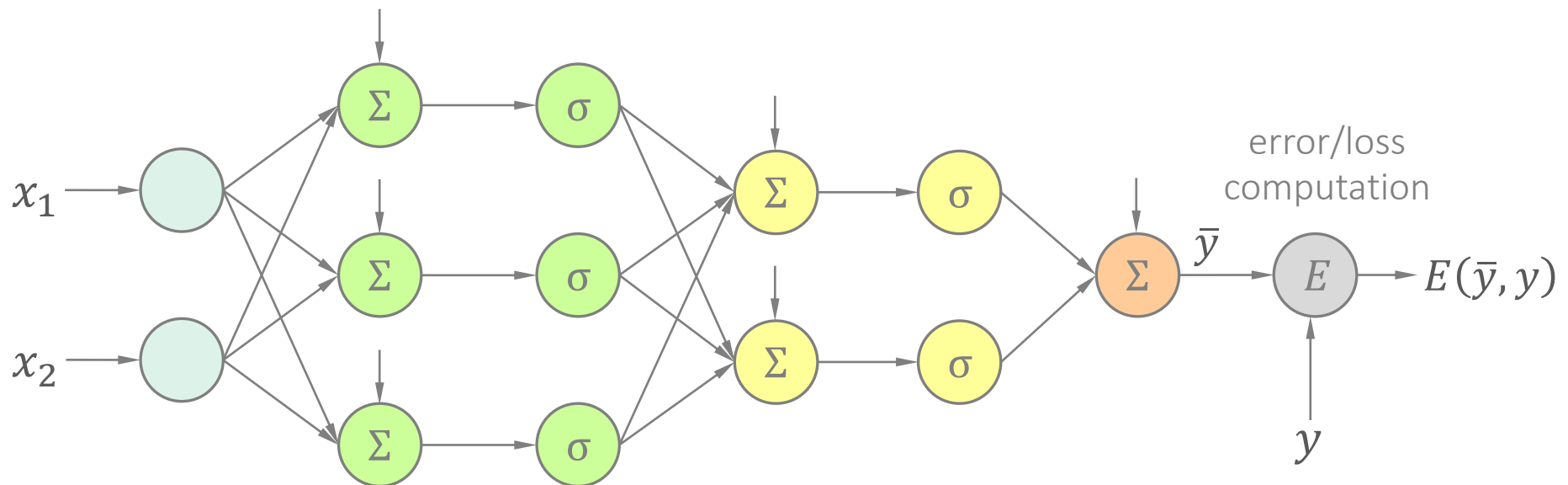
solution scheme

backward execution (calculation of sensitivities)



backpropagation

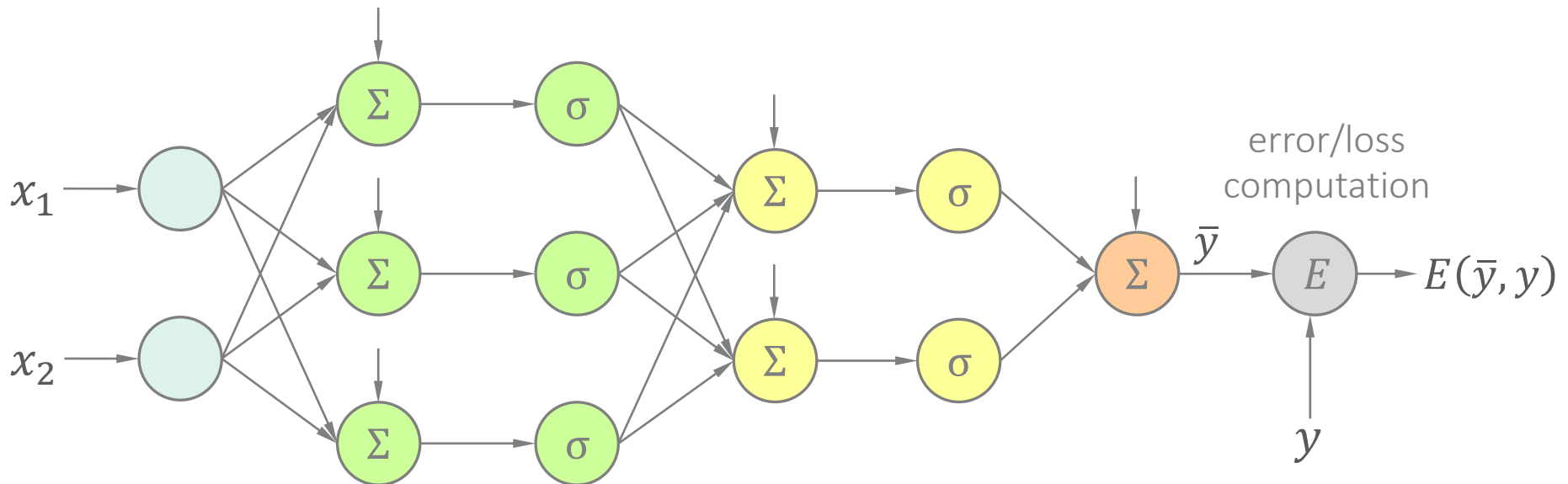
case study: fully connected network with 3 layers



σ : activation function (e.g. sigmoid, tanh, relu, etc.)

backpropagation

case study: fully connected network with 3 layers

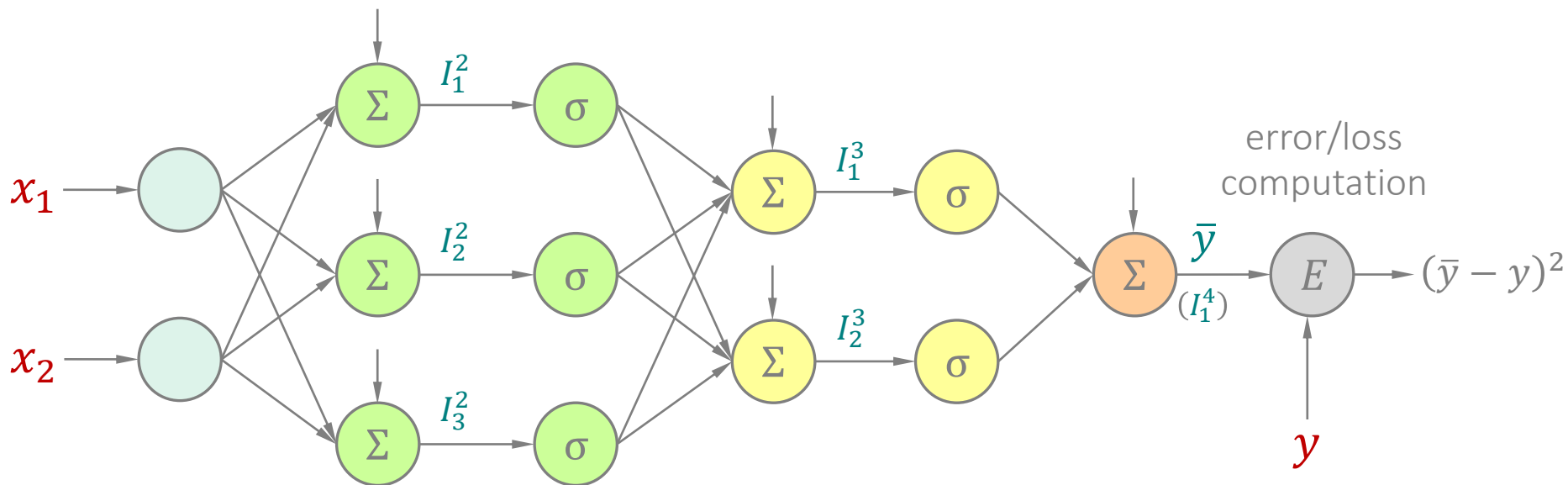


$2 \times 3 + 3$	+	$3 \times 2 + 2$	+	$2 \times 1 + 1$	=
9	+	8	+	3	=
20 trainable parameters/weights					

backpropagation

case study: fully connected network with 3 layers

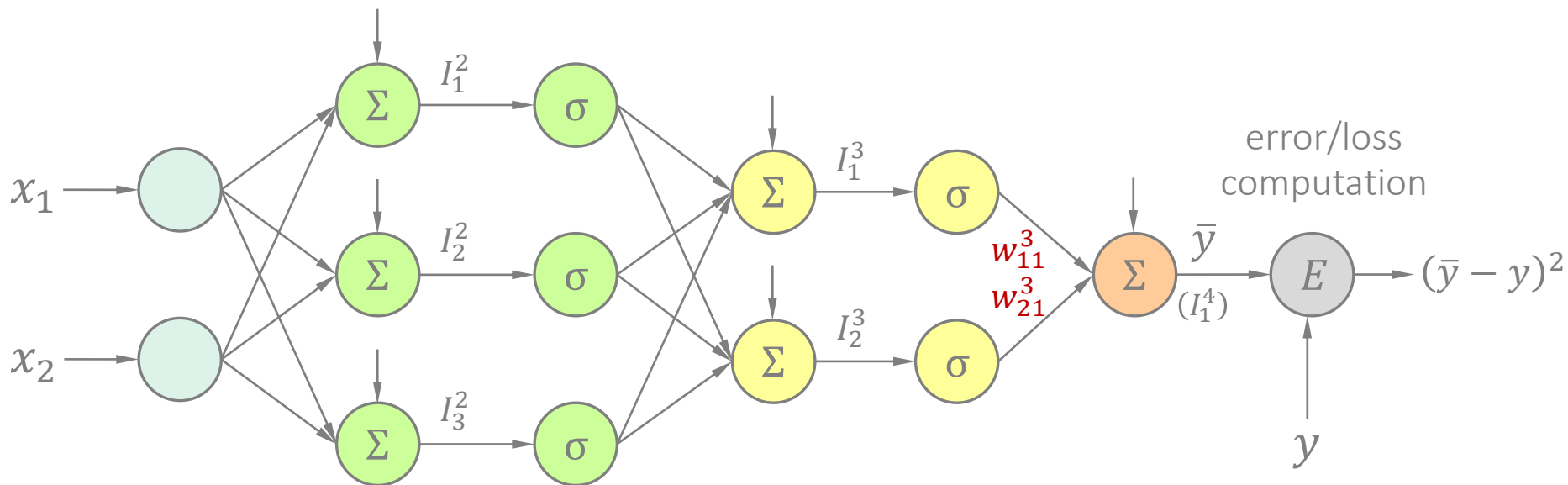
forward execution: introduce (x, y) and compute I_i^l and \bar{y}



backpropagation

case study: fully connected network with 3 layers

backward execution: compute gradients $\frac{\partial E}{\partial w_{ij}^l}$

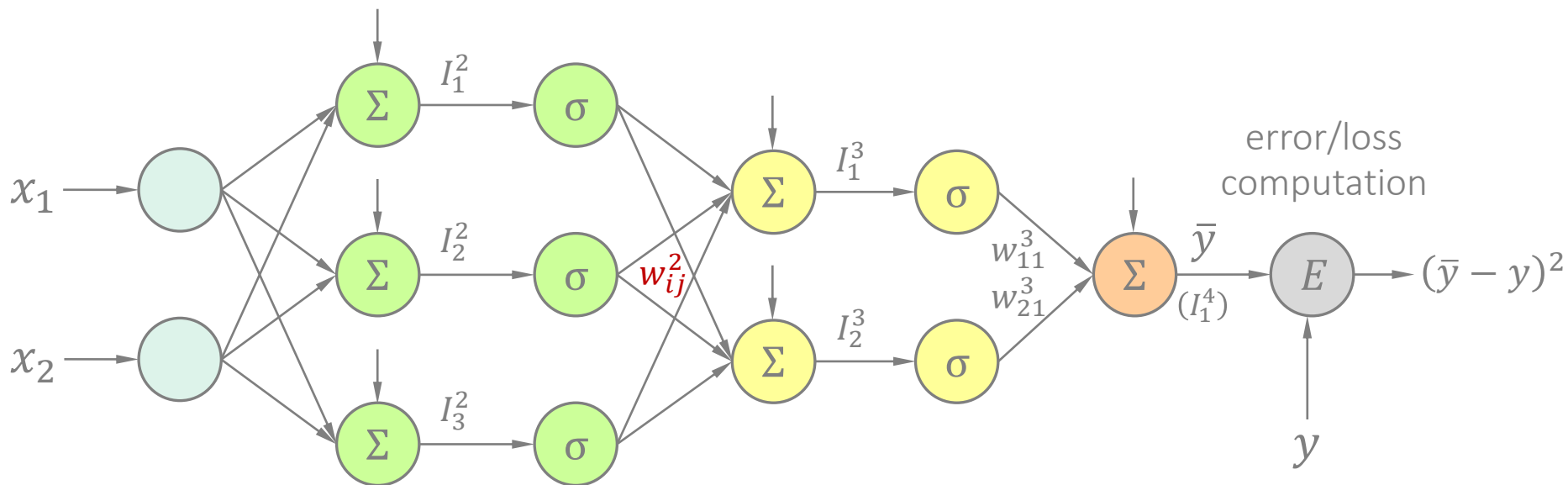


$$\frac{\partial E}{\partial w_{ij}^3} = \frac{\partial I_1^4}{\partial w_{ij}^3} \cdot \delta_1^4 = \sigma(I_i^3) \cdot 2(\bar{y} - y)$$

backpropagation

case study: fully connected network with 3 layers

backward execution: compute gradients $\frac{\partial E}{\partial w_{ij}^l}$

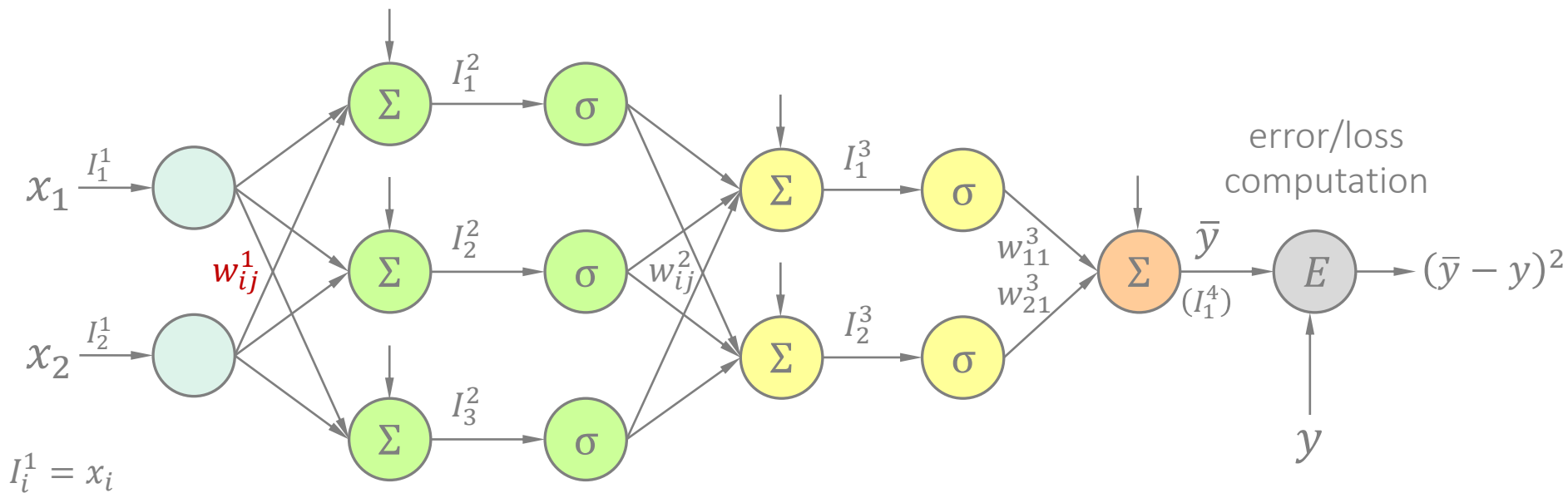


$$\frac{\partial E}{\partial w_{ij}^2} = \frac{\partial I_j^3}{\partial w_{ij}^2} \cdot \delta_j^3 = \sigma(I_i^2) \cdot \sigma'(I_j^3) \cdot w_{j1}^3 \cdot \delta_1^4$$

backpropagation

case study: fully connected network with 3 layers

backward execution: compute gradients $\frac{\partial E}{\partial w_{ij}^l}$



$$\frac{\partial E}{\partial w_{ij}^1} = \frac{\partial I_j^2}{\partial w_{ij}^1} \cdot \delta_j^2 = \sigma(I_i^1) \cdot \sigma'(I_j^2) \sum_{k=1}^2 w_{jk}^2 \cdot \delta_k^3$$

case study

fully connected neural network for the MNIST digits task

implementation in Keras

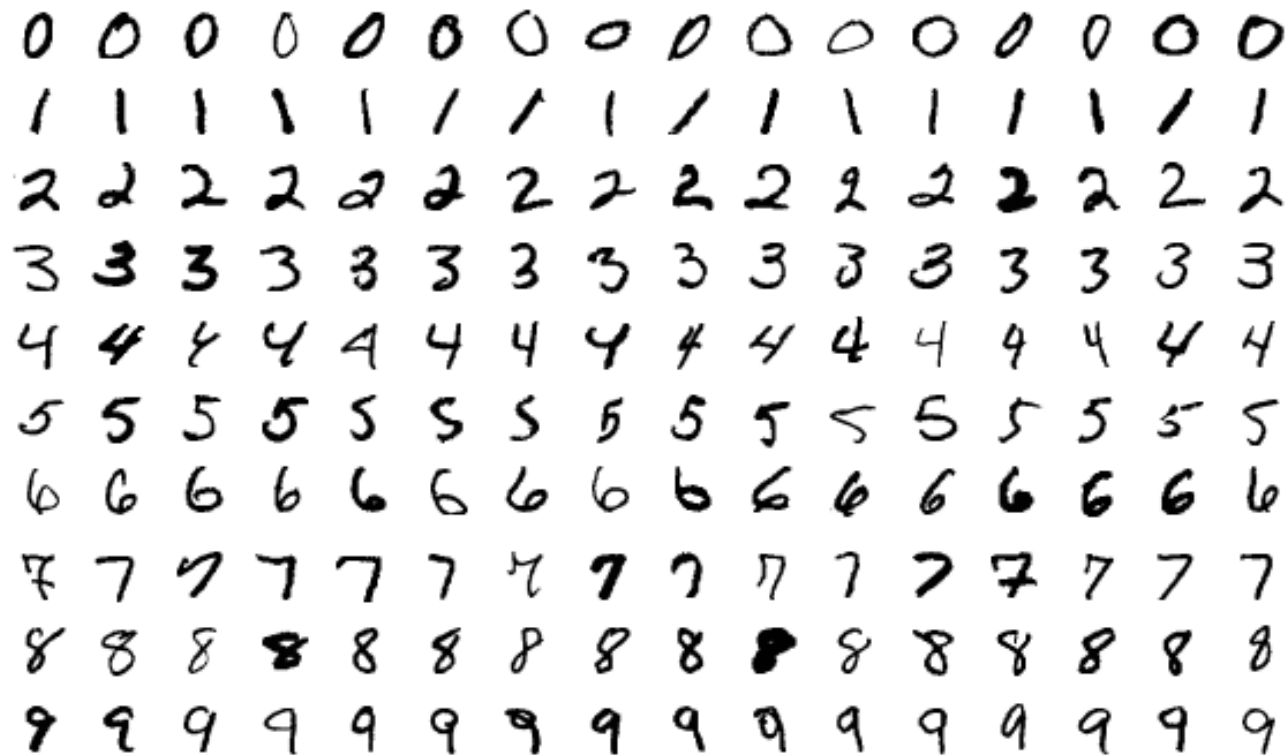
- deep learning framework: Python library for creating neural networks
- high-level interface based on Tensorflow, Theano, or the Microsoft Cognitive Toolkit
- it allows defining and assembling pieces in neural networks such as layers, objective functions, activation, optimizers, etc.

case study

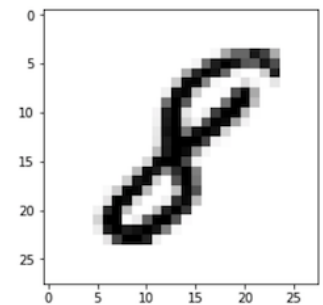
fully connected neural network for the MNIST digits task

MNIST (Modified National Institute of Standards and Technology database)

handwritten digits image collection ([official website](#))



- 60,000 training images
- 10,000 test images
- 10 classes
- image size: 28x28
- grey images



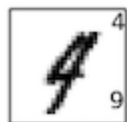
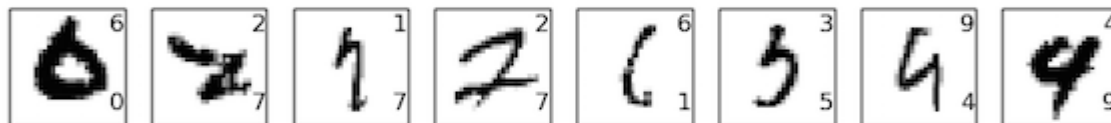
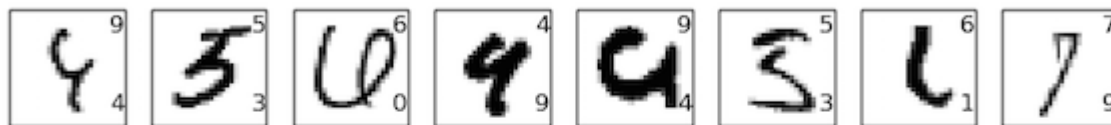
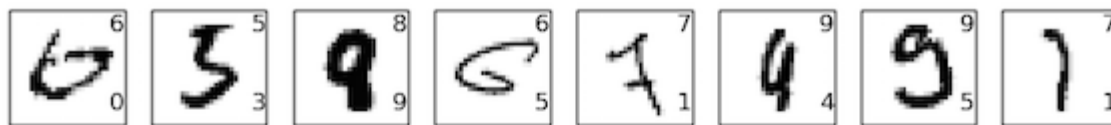
Source: Medium ([+](#))

case study

fully connected neural network for the MNIST digits task

MNIST (Modified National Institute of Standards and Technology database)

- relatively simple task
- although it includes complex cases that are difficult to read (noise)



33 test errors with CNN

- 33 errors + 9,967 hits
- true class at top right
- estimated class at bottom right
- error rate = 0.33%

Source: Michael Nielsen (2019). *Neural Networks and Deep Learning*, online book ([+](#)).

case study

fully connected neural network for the MNIST digits task

simplified script

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(60000, 784).astype('float32')/255
x_test = x_test.reshape(10000, 784).astype('float32')/255
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Dense(1024, activation='relu', input_shape=(784,)))
model.add(Dense(1024, activation='relu'))
model.add(Dense(1024, activation='relu'))
model.add(Dense(10, activation='softmax'))

sgd=SGD(lr=0.01, decay=1e-6, momentum=0.9)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=100, epochs=5,
                    validation_data=(x_test, y_test))
```

case study

fully connected neural network for the MNIST task

How many parameters
does this fully connected
network have?

case study

fully connected neural network for the MNIST task

layers	input	hidden 1	hidden 2	hidden 3	output	TOTAL
units	784	1.024	1.024	1.024	10	3.082
weights		802.816	1.048.576	1.048.576	10.240	2.910.208
bias		1.024	1.024	1.024	10	3.082
TOTAL		803.840	1.049.600	1.049.600	10.250	2.913.290

case study

fully connected neural network for the MNIST task

hyperparameters

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(60000, 784).astype('float32')/255
x_test = x_test.reshape(10000, 784).astype('float32')/255
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Dense(1024, activation='relu', input_shape=(784,)))
model.add(Dense(1024, activation='relu'))
model.add(Dense(1024, activation='relu'))
model.add(Dense(10, activation='softmax'))

sgd=SGD(lr=0.01, momentum=0.9)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=100, epochs=5,
                    validation_data=(x_test,y_test))
```

case study

fully connected neural network for the MNIST task

activation 'relu'

- Rectified Linear Unit (ReLU)
- a piecewise linear activation function (hidden u)

activation 'softmax'

- output layer activation function
- probability distribution over K outputs

loss 'categorical_crossentropy'

- one-hot vector + softmax + cross entropy
- loss function
- measures discrepancy between two distributions

optimizer 'sgd'

- stochastic gradient descent
- basic optimization method

batch_size 100

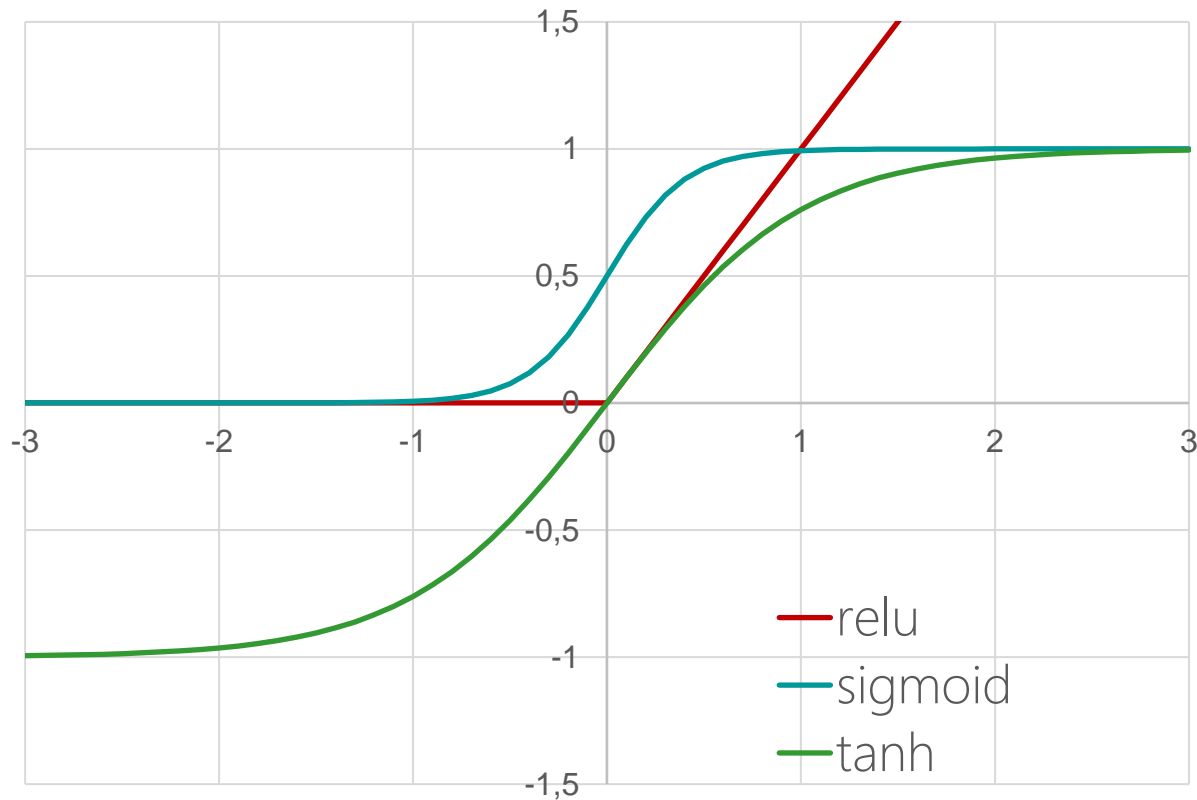
- mini-batch based training
- stochastic method (random subsets)
- weights update after each batch

epochs 5

- learning loop over all training samples

activation function

relu, *sigmoid*, *tanh*



relu

$$g(z) = \max\{0, z\}$$

sigmoid

$$g(z) = \frac{1}{1 + e^{-z}}$$

tanh

$$g(z) = 2\sigma(2z) - 1$$

activation function

softmax (normalized exponential function)

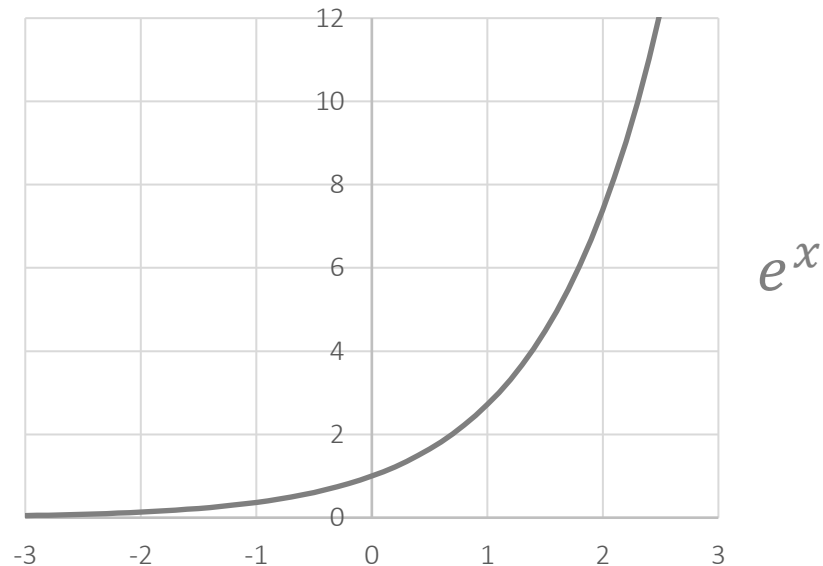
given a vector $\mathbf{z} \in \mathbb{R}^K$

softmax projects a vector of real data onto a “probability distribution”

$$s: \mathbb{R}^K \rightarrow [0,1]^K, \quad \sum_i s(\mathbf{z})_i = 1$$

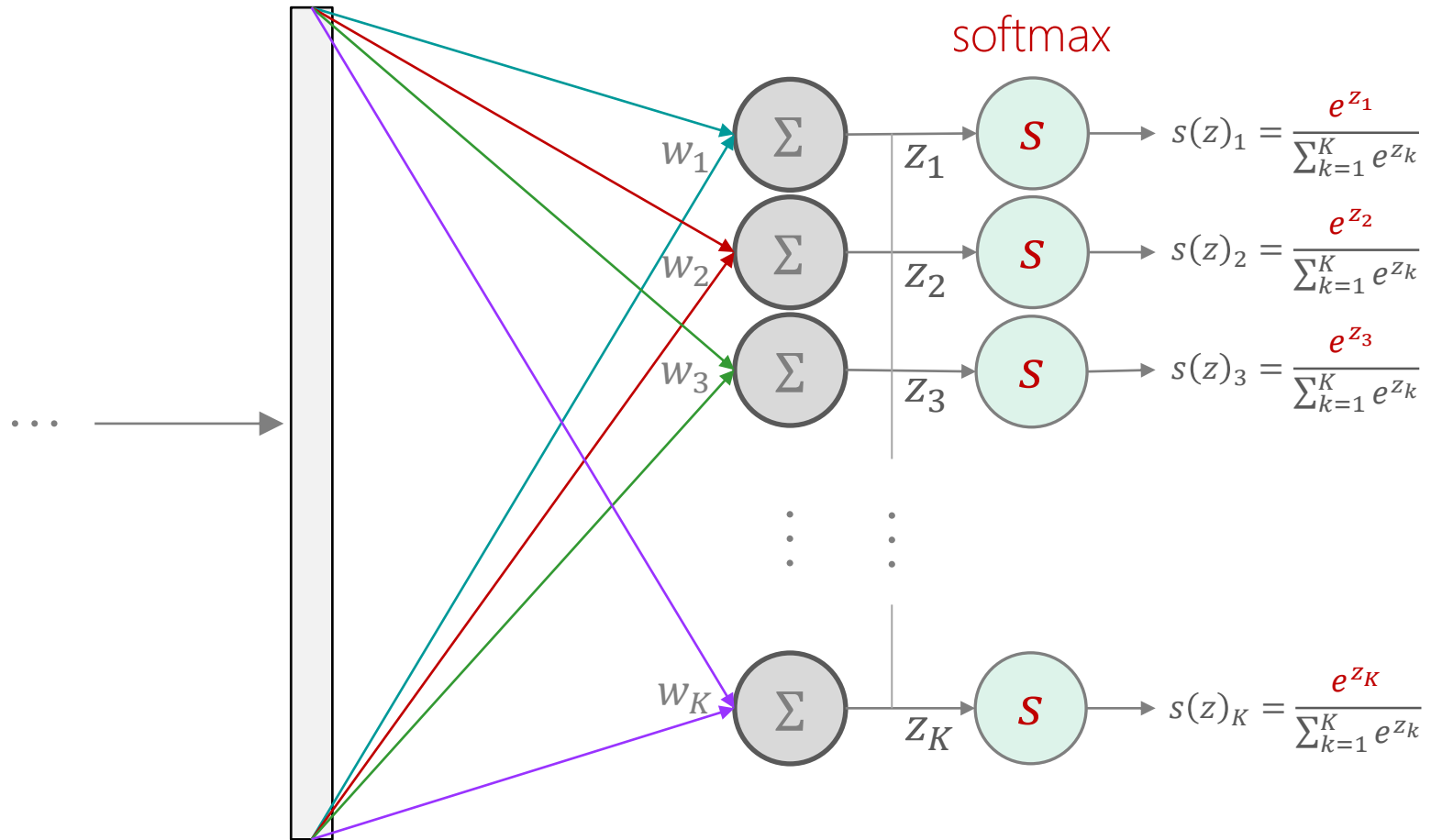
formulation:

$$s(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{k=1}^K e^{z_k}}$$



activation function

softmax (normalized exponential function)



activation function

softmax (normalized exponential function)

softmax is increasing: if $z_i < z_j$, then $s(z_i) < s(z_j)$

example

z_i	e^{z_i}	$s(z)_i$
-1	0,37	0,01
0	1,00	0,02
1	2,72	0,04
2	7,39	0,11
4	54,60	0,83
	66,07	1,00

penalizes non-maximum
activation values

reinforces the highest
activation value

activation function

softmax (normalized exponential function)

softmax saturates if $\min_i z_i \ll \max_i z_i$

ejemplo

z_i	e^{z_i}	$s(z)_i$
-1	0,37	0,00
0	1,00	0,00
1	2,72	0,00
2	7,39	0,00
8	2.980,96	1,00
	2.992,43	1,00

← the winner
takes it all

loss function

operating principle

given

- y , expectation (ground truth)
- \hat{y} , prediction, estimation

a los/cost/error function $L(y, \hat{y})$

- measures the distance, difference, or discrepancy between y e \hat{y}
- when y e \hat{y} are very different, then L is large (high loss)
- when y e \hat{y} are very similar, then L is small (low loss)
- when y e \hat{y} are equal, then $L = 0$

learning objective

- find parameters of the model that minimize L over the validation set (part of the training data, not the test data!)

loss function

one-hot encoding (output encoding)

categorical/nominal variable: takes symbolic values, not numeric ones.

examples:

- **pet**: "dog", "cat", "bird"
- **model**: "sedan", "minivan", "bus", "truck"
- **dígitos**: '0', '1', '2', ...'9'

limitation: do not support numerical comparisons/operations

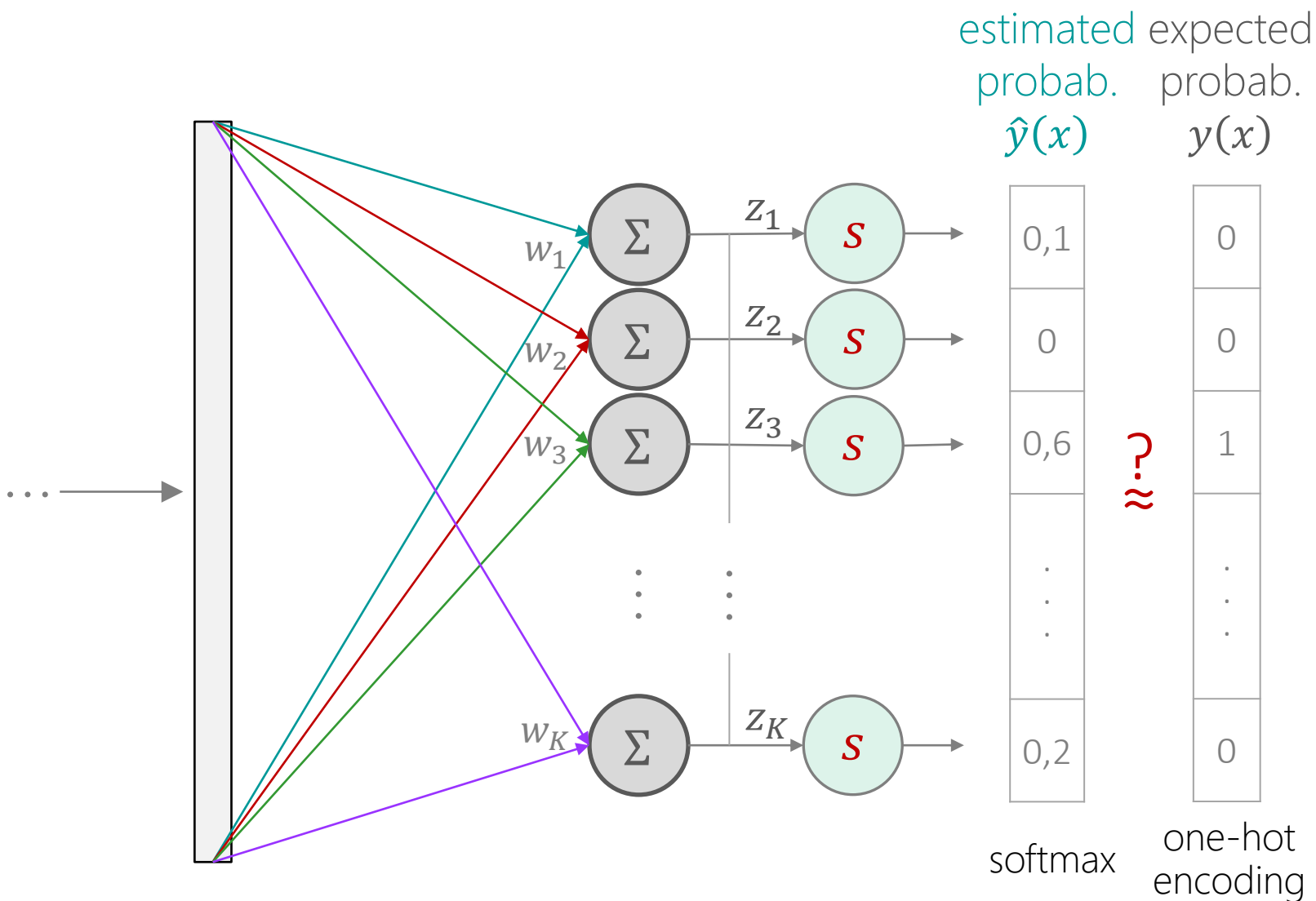
one-hot encoding:

	sedan	minivan	bus	truck
x_{sed}	1	0	0	0
x_{min}	0	1	0	0
x_{bus}	0	0	1	0
x_{tru}	0	0	0	1

loss function

how to measure the difference between two distributions?

MNIST
 $K = 10$



loss function

cross entropy

given two probability distributions:

- \mathbf{y} (e.g. expected, ground truth)
- $\hat{\mathbf{y}}$ (e.g. predicted, observed)

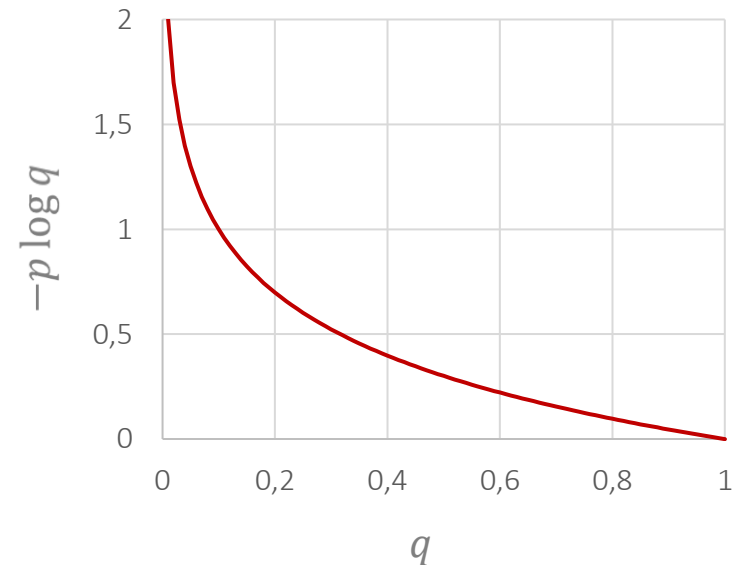
cross entropy (between two distributions):

$$E(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_i y_i \log \hat{y}_i$$

effect:

- if \mathbf{y} and $\hat{\mathbf{y}}$ are similar, then $E(\mathbf{y}, \hat{\mathbf{y}})$ is small
- if \mathbf{y} and $\hat{\mathbf{y}}$ are different, then $E(\mathbf{y}, \hat{\mathbf{y}})$ is large

cross entropy for $p = 1$



loss function

cross entropy

two opposite scenarios

favorable scenario

y	\hat{y}	$-y_i \log \hat{y}_i$
0	0,05	0,00
0	0,05	0,00
1	0,80	0,10
0	0,05	0,00
0	0,05	0,00
loss		0,10

unfavorable scenario

y	\hat{y}	$-y_i \log \hat{y}_i$
0	0,20	0,00
0	0,20	0,00
1	0,20	0,70
0	0,20	0,00
0	0,20	0,00
loss		0,70

only the \hat{y}_i associated with $y_i = 1$ contributes (**log 1** is the smallest loss)

optimization problem

MNIST dataset

given 60.000 training digit images, with their classes annotated as one-hot vectors...

$$T = \{(x_i, y_i)\}_{i=1,60.000} \text{ such that } x_i \in [0,1]^{784}, y_i \in \{0,1\}^{10}$$

goal: to find optimal values for the 2.913.920 model parameters

$$W^* = \arg \min_W L(W), \quad W \in \mathbb{R}^{2.913.920}$$

with $L(W)$ being the loss function defined as follows:

$$L(W) = \frac{1}{60.000} \sum_{i=1}^{60.000} E(y_i, \hat{y}_i) = -\frac{1}{60.000} \sum_{i=1}^{60.000} \sum_{j=1}^{10} y_{ij} \log \hat{y}_{ij}$$

optimizer

sgd + momentum + weight decay

stochastic gradient descent (sgd):

$$\theta_{t+1} = \theta_t - \gamma \frac{\partial L}{\partial \theta}(\theta_t)$$

sgd + momentum + weight decay (γ denotes the learning rate):

$$\theta_{t+1} = \theta_t + v_t$$

$$v_t = \text{momentum} \cdot v_{t-1} - \gamma \frac{\partial L}{\partial \theta}(\theta_t)$$

$$\gamma = \gamma \cdot \frac{1}{1 + \text{decay} \cdot \text{iteración}}$$

hyperparameters

overview

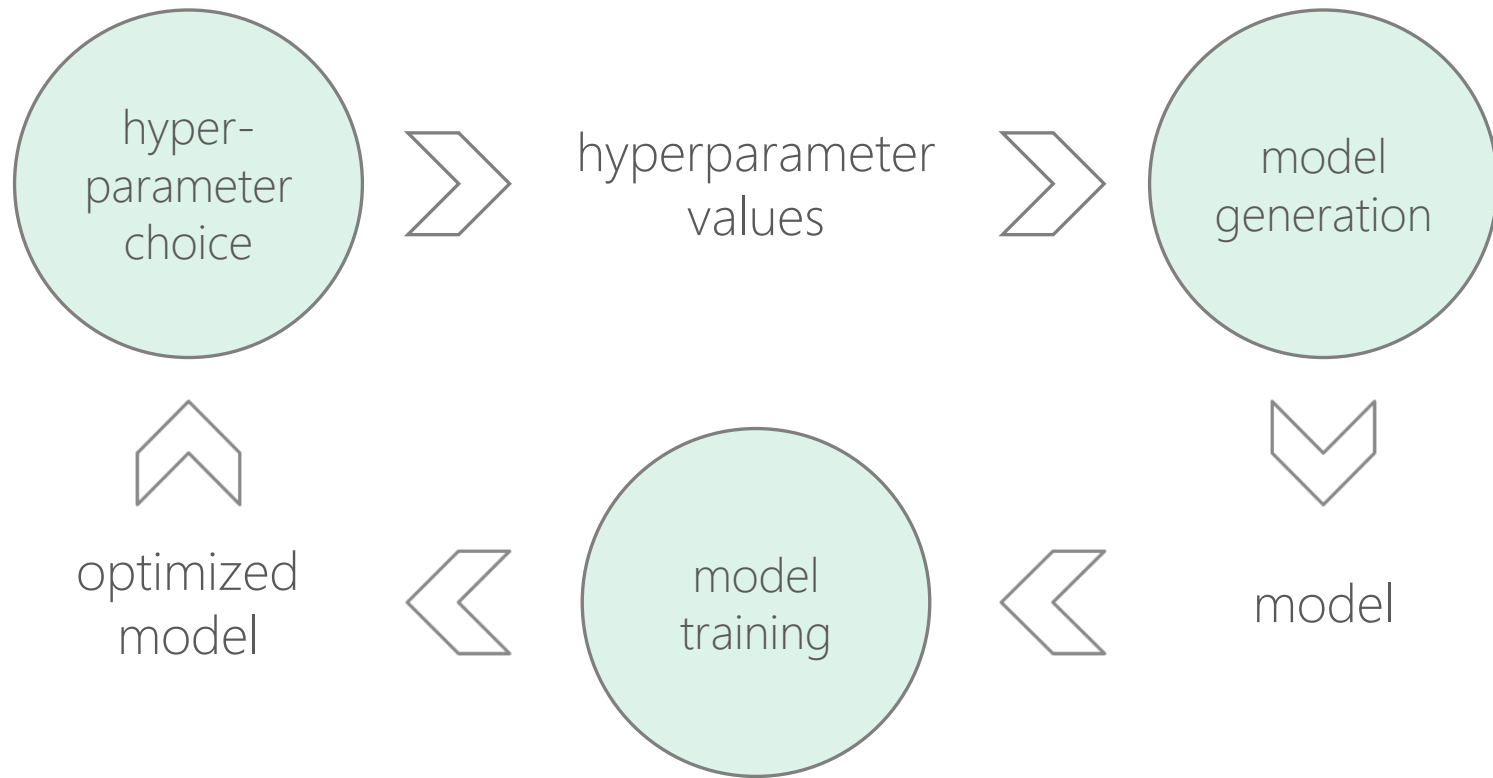
hyperparameters are parameters needed to generate the model or to define the training process

hyperparameters determine the structure or configuration of the model or characteristics of the learning process; their values are chosen before each training session.

their **optimal values** depend on the complexity of the task, the nature of the data (dimensionality, distribution, quantity, etc.), and the interdependence with other hyperparameters.

hyperparameters

overview



hyperparameters

fully connected neural network for the MNIST task

hyperparameters of the network architecture defined for MNIST:

- 4 trainable layers
- layer 1: 1024 units, ReLU activation
- layer 1: 1024 units, ReLU activation
- layer 1: 1024 units, ReLU activation
- layer 4: softmax activation

hyperparameters

fully connected neural network for the MNIST task

hyperparameters of the learning process defined for MNIST:

- optimizer: SGD(lr=0.01, decay=1e-6, momentum=0.9)
- loss = 'categorical_crossentropy'
- batch_size = 100
- epochs = 5

summary

- a machine learning algorithm optimizes models from data
- a linear model solves only tasks with linear boundaries
- Perceptron is a linear model of binary classification
- a fully connected multilayer network is composed of a sequence of fully connected layers.
- each unit consists of a linear function + nonlinear activation
- a multilayer network could learn any decision boundary
- backpropagation is an algorithm for training feedforward networks in supervised learning; includes calculation of gradients, not how to use them.