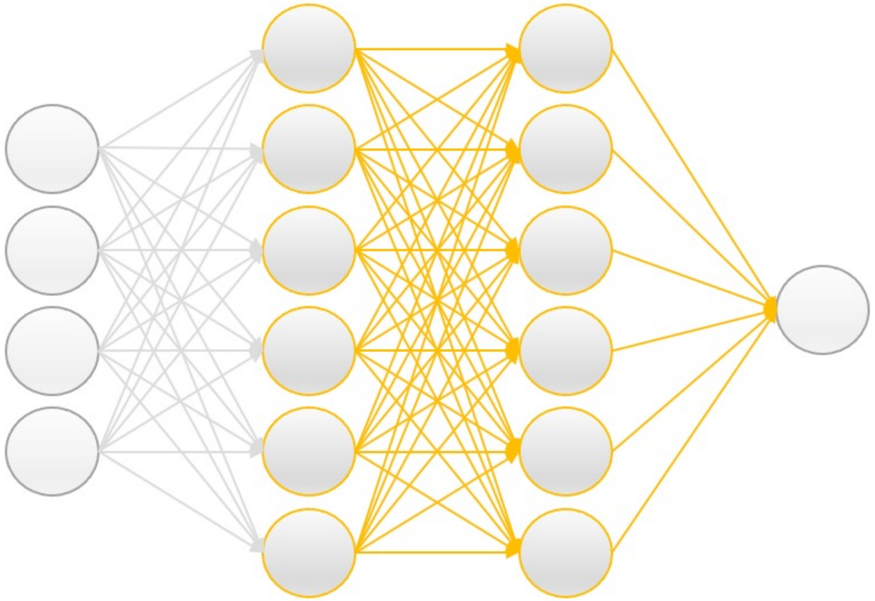
Artificial Neural Network



CS-EJ3311 - Deep Learning with Python
24.10.-11.12.2022
Aalto University & FiTech.io

31.10.2022 Shamsi Abdurakhmanova

First artificial neuron models (~1950)

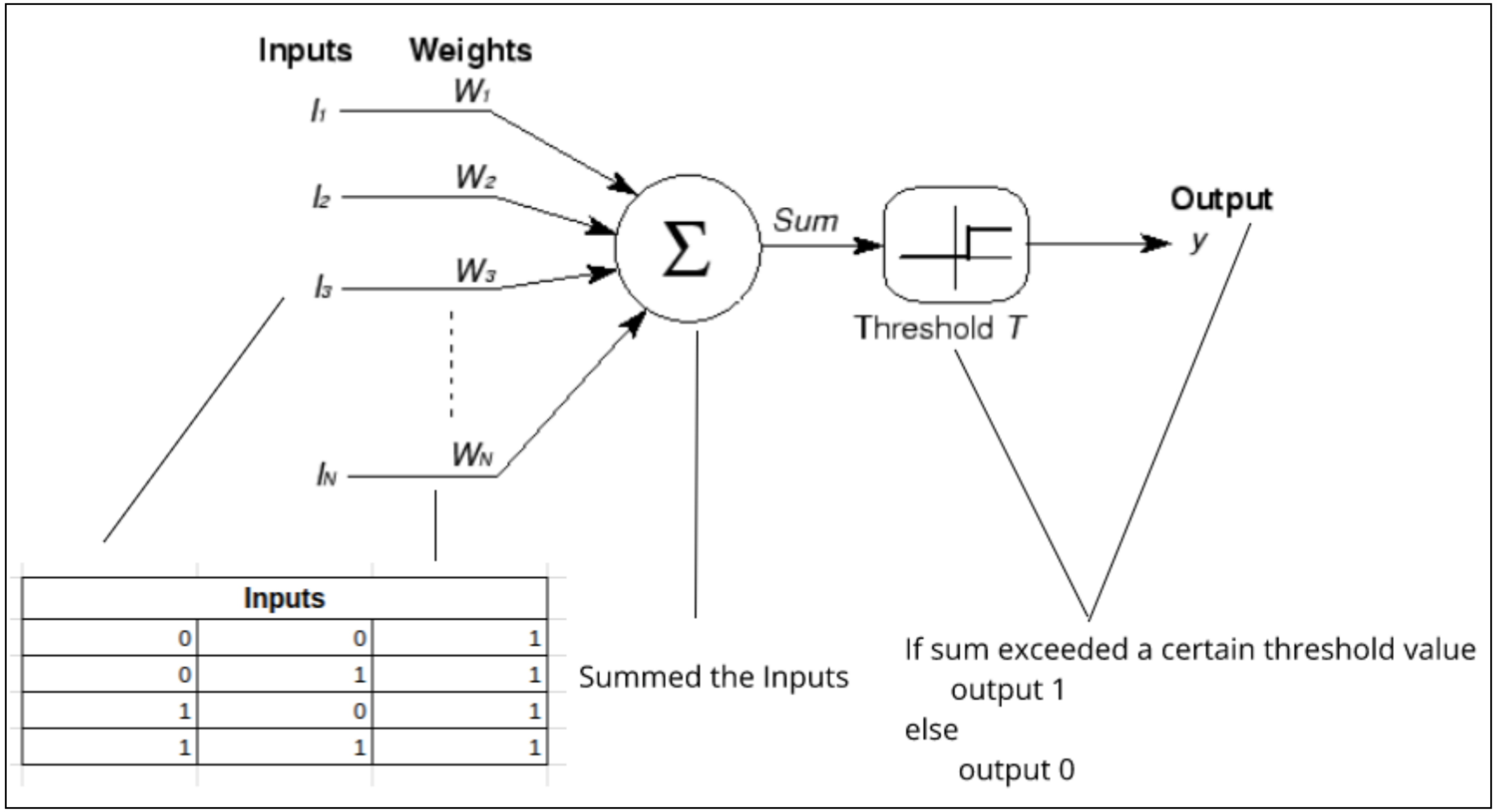


Figure 9.13: McCulloch-Pitts computation model of neuron (Image credit for NN: http://wwwold.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node12.html)

Deep Learning hype - Why now?

- Harware advances
- Optimization methods advances
- Big Data

House price prediction

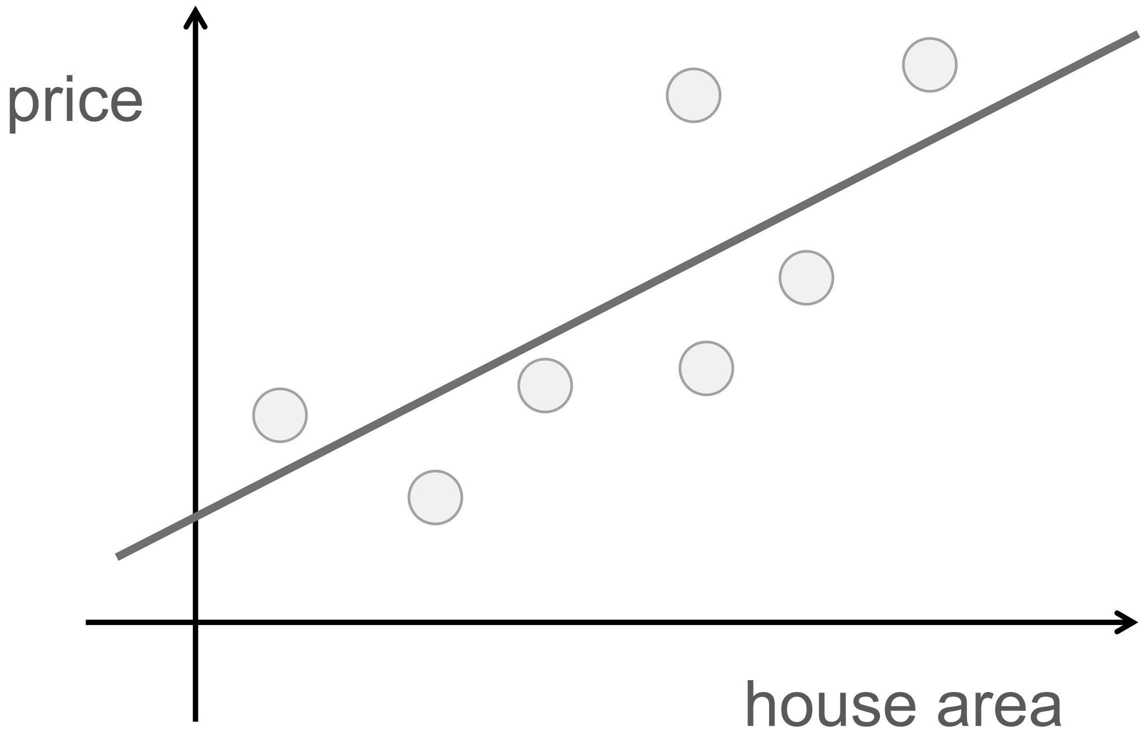
- house area,
- n.o. rooms/ bathrooms,
- neuborhood,
- building age,
- future rennovations,
- etc.

House price prediction

- house area $= x_1$
- n.o. rooms/ bathrooms $= x_2$
- neuborhood $= x_3$
- building age $= x_4$
- future rennovations $= x_5$

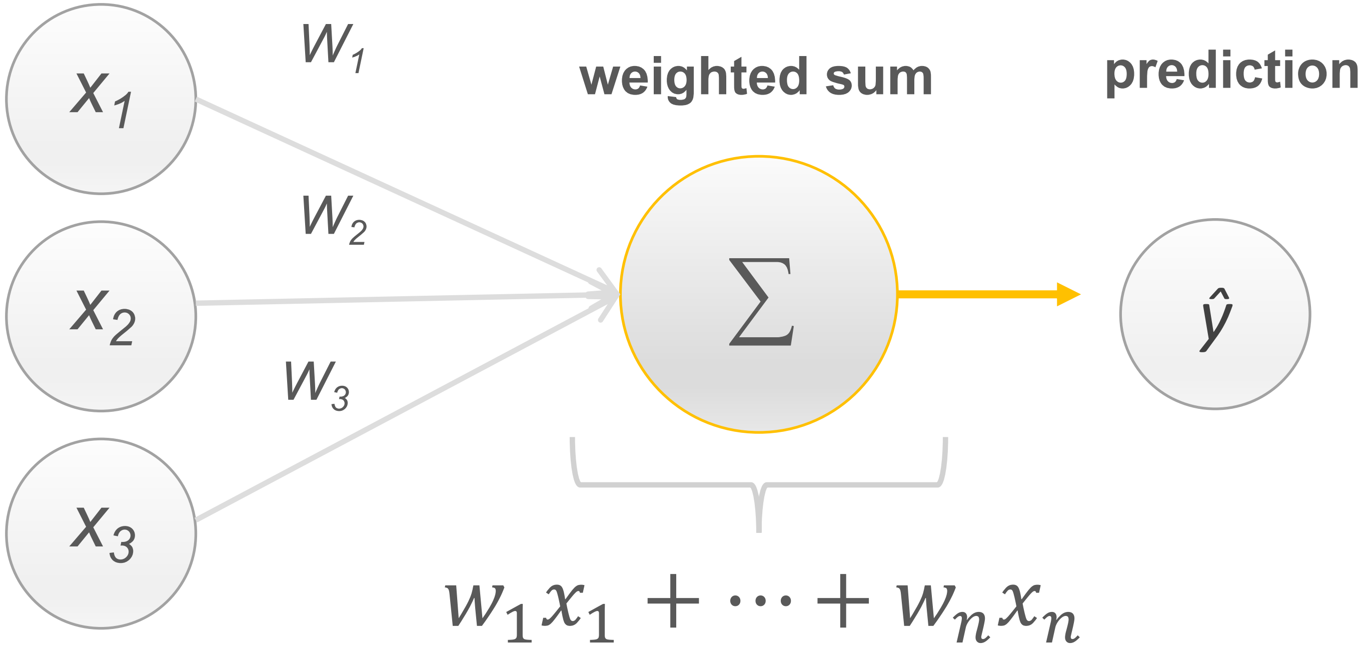
$$\longrightarrow \quad w_1 x_1 + w_2 x_2 + w_2 x_2 + w_3 x_3 + w_4 x_4 = \hat{y}$$

Historical Data

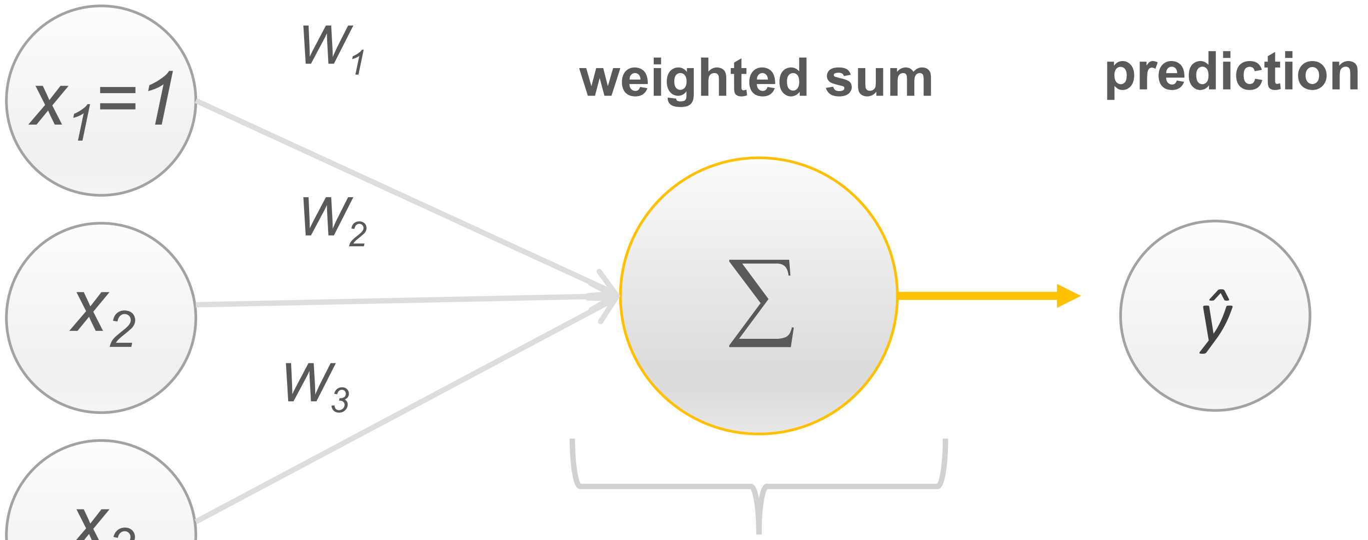


$$\hat{y} = wx + b$$

Linear Regression

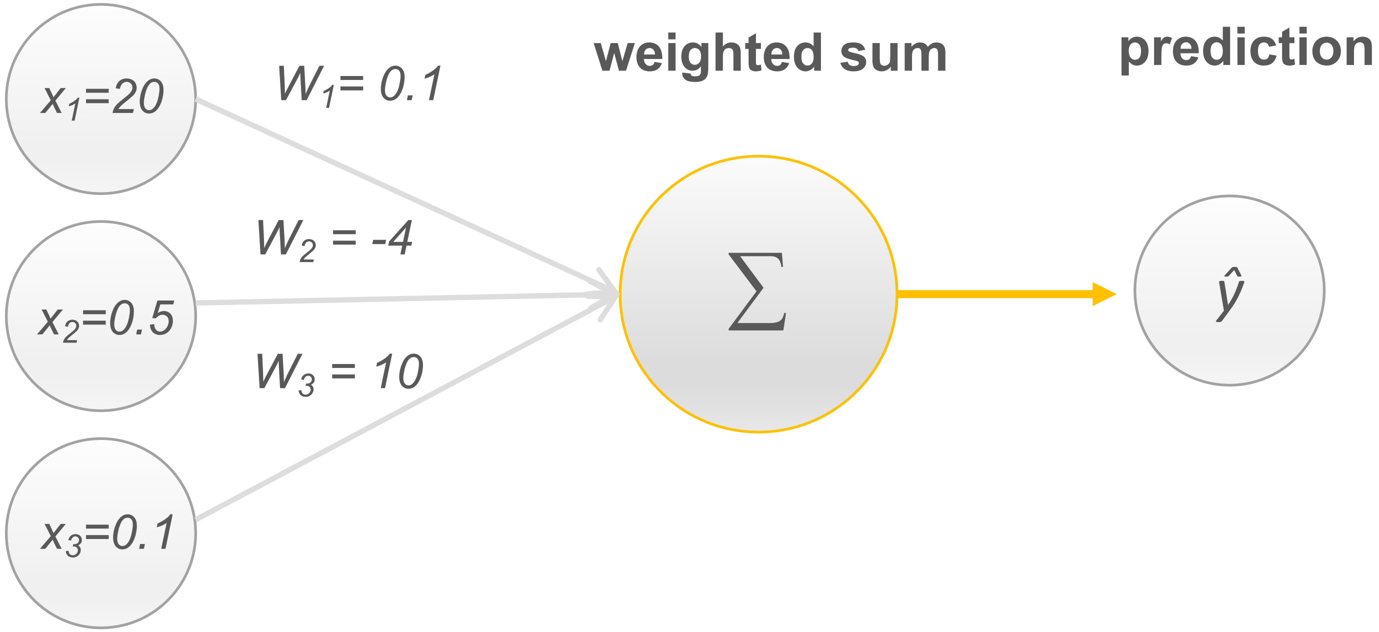


Linear Regression - Bias



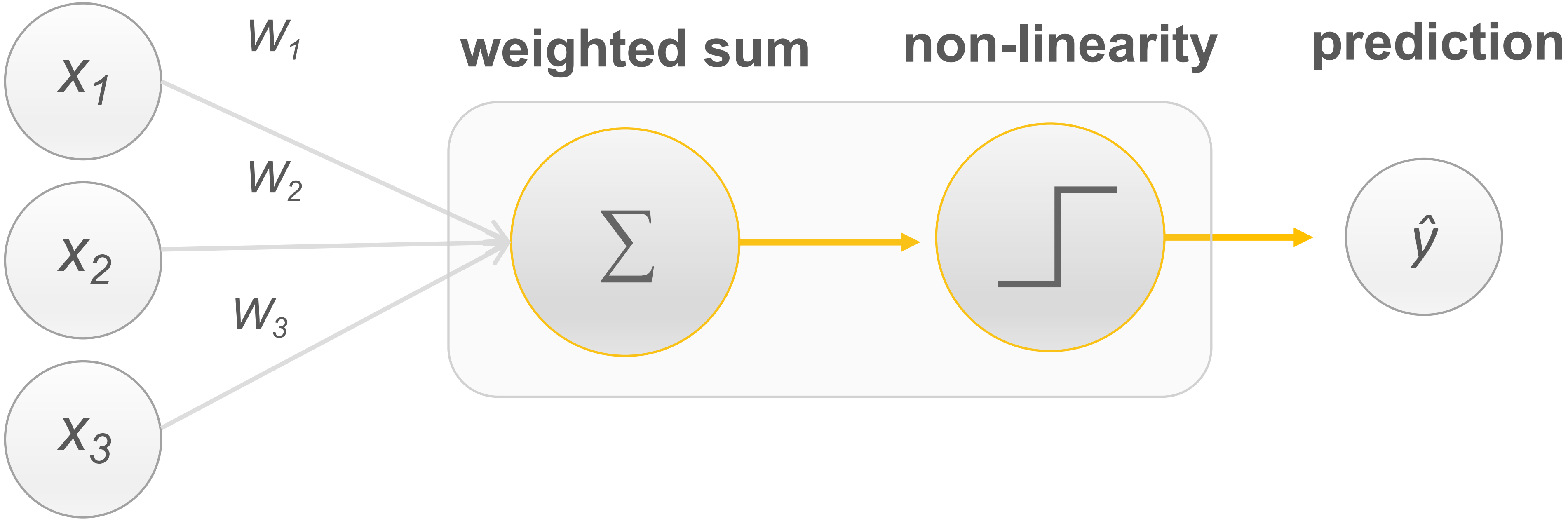
$$w_1x_1 + \cdots + w_nx_n =$$
$$w_1 \ + \cdots + w_nx_n$$

$$\hat{y} = 20 * 0.1 + 0.5 * (-4) + 0.1 * 10 = 1$$



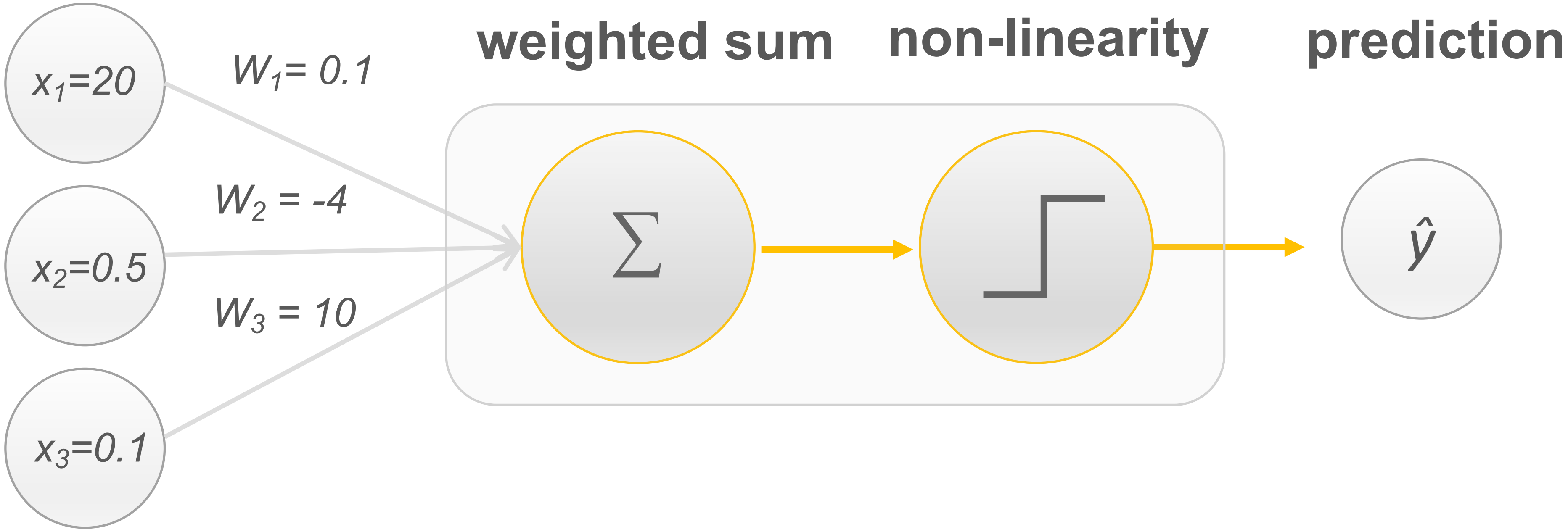
non-linear transformation

$$\boxed{\hat{y} = \sigma(w_1 x_1 + \cdots + w_n x_n)}$$

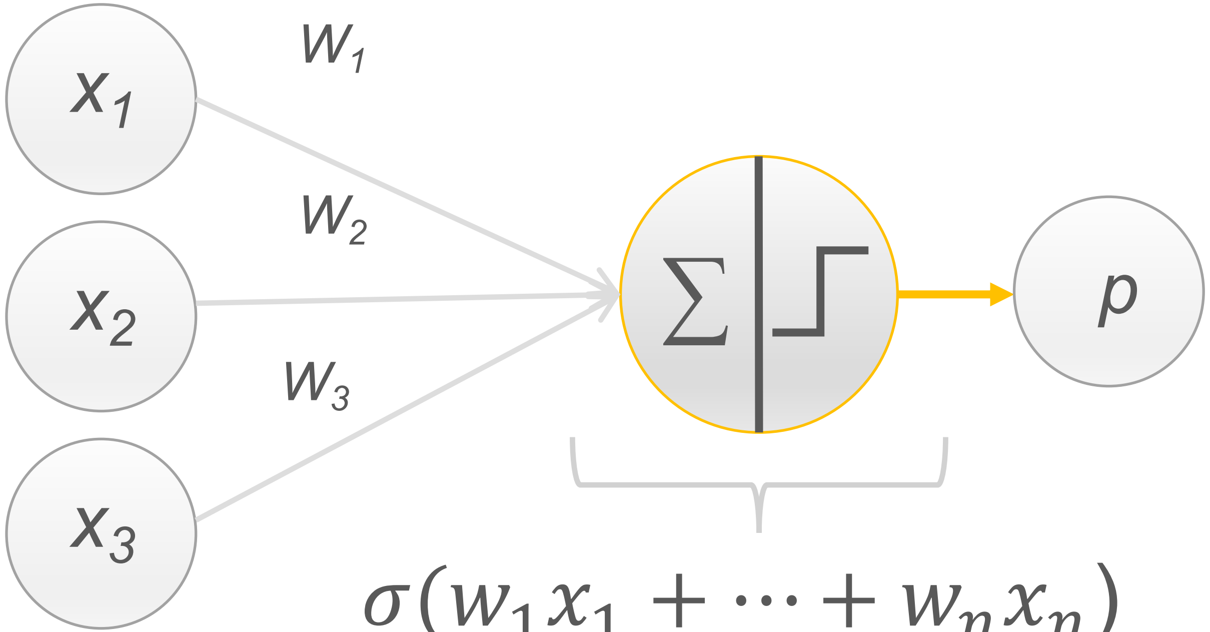


$$\hat{y} = \sigma(20 * 0.1 + 0.5 * (-4) + 0.1 * 10) \approx 0.73$$

$$\sigma = \frac{1}{1 + e^{-x}}$$

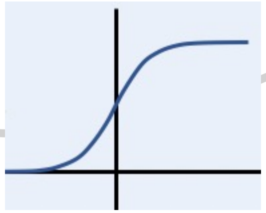


Logistic Regression



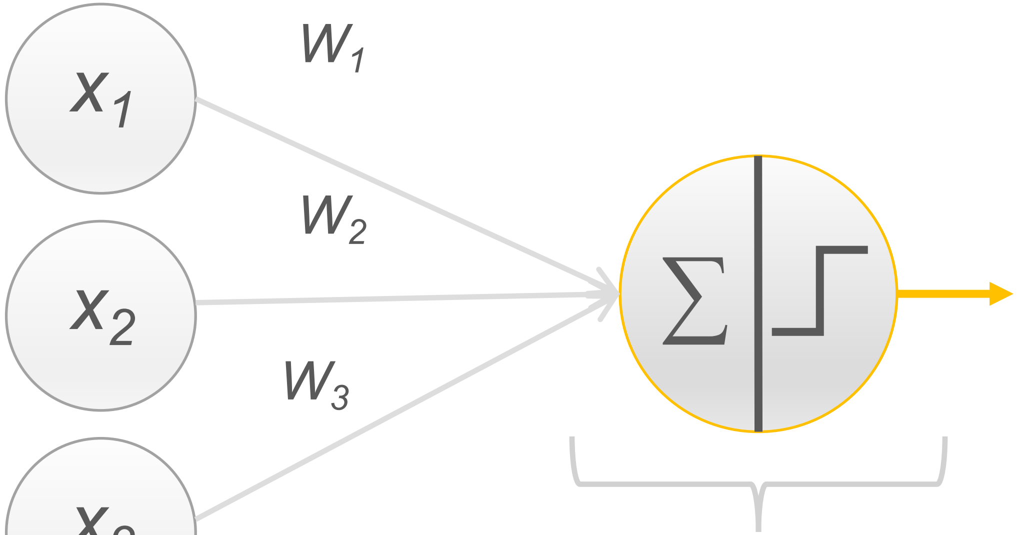
$$\sigma(w_1 x_1 + \cdots + w_n x_n)$$

$$p \in [0,1]$$



$$\sigma = \frac{1}{1 + e^{-x}}$$

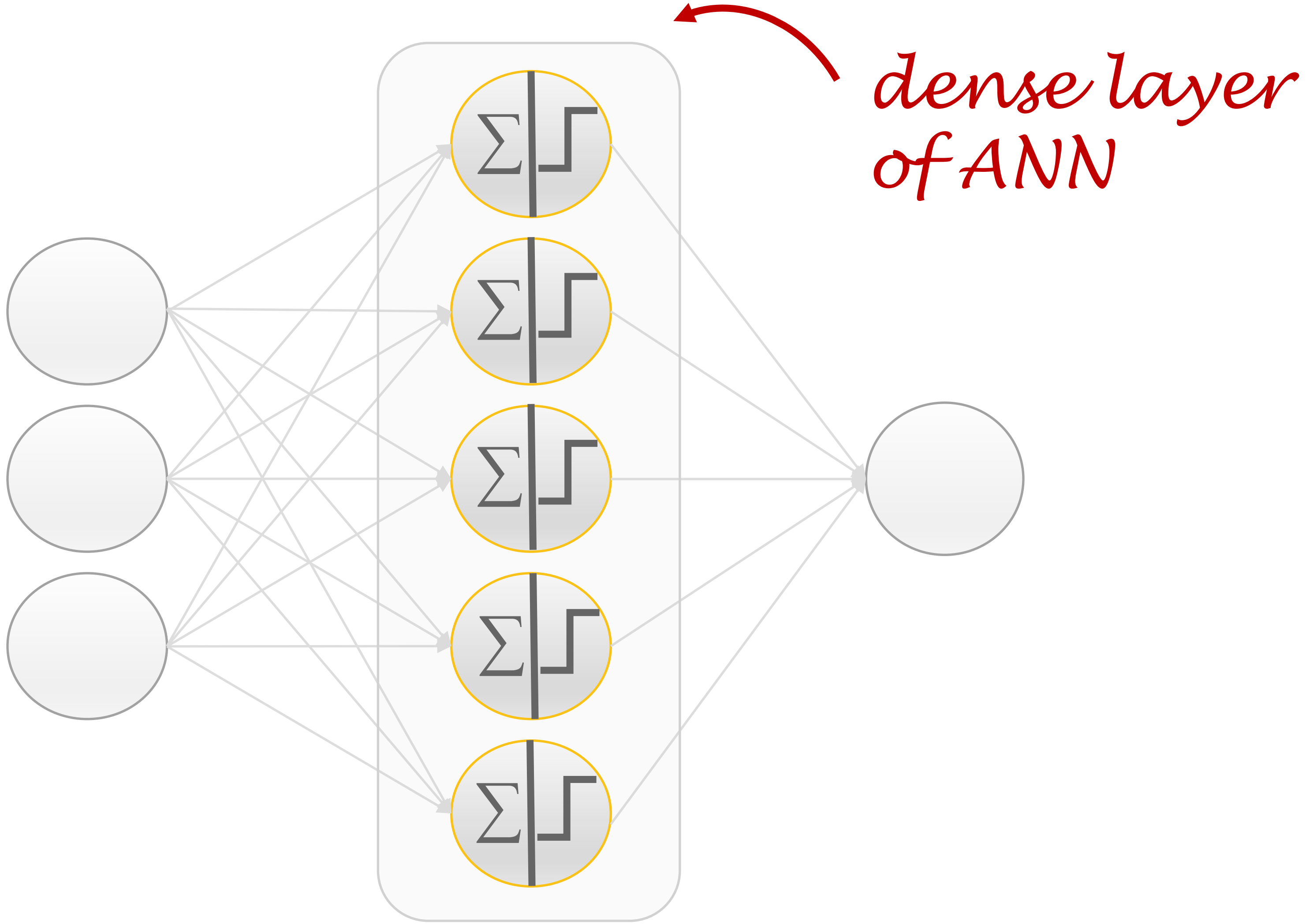
"Atom" of the ANN – artificial neuron or unit of ANN



$$\sigma(w_1 x_1 + \cdots + w_n x_n)$$

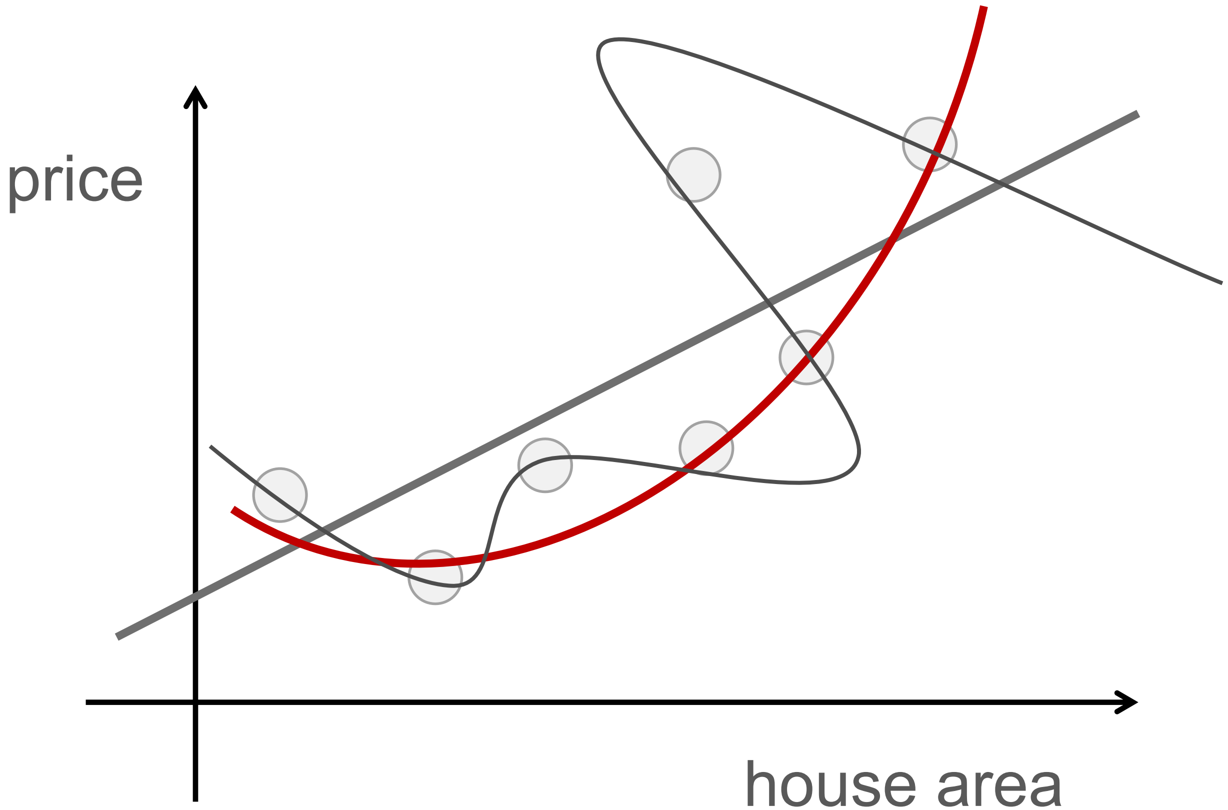
activation function

ANN – stacked elementary units



Why stacking neurons?

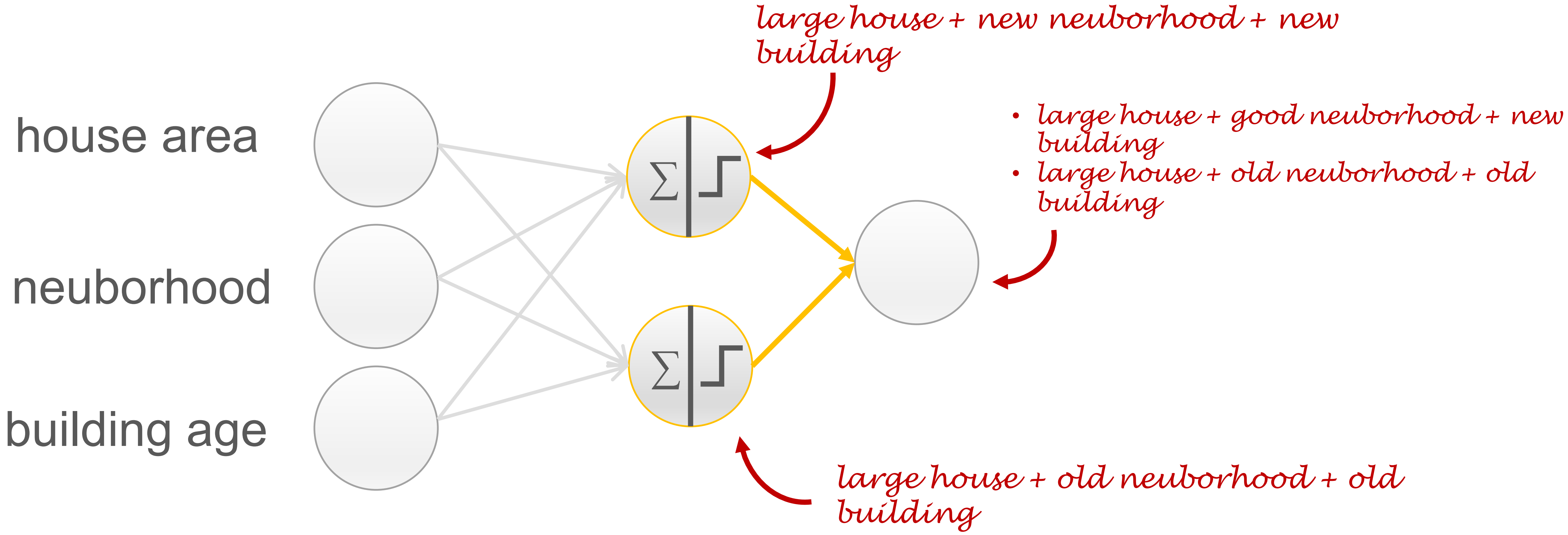
To build more complex & flexible model



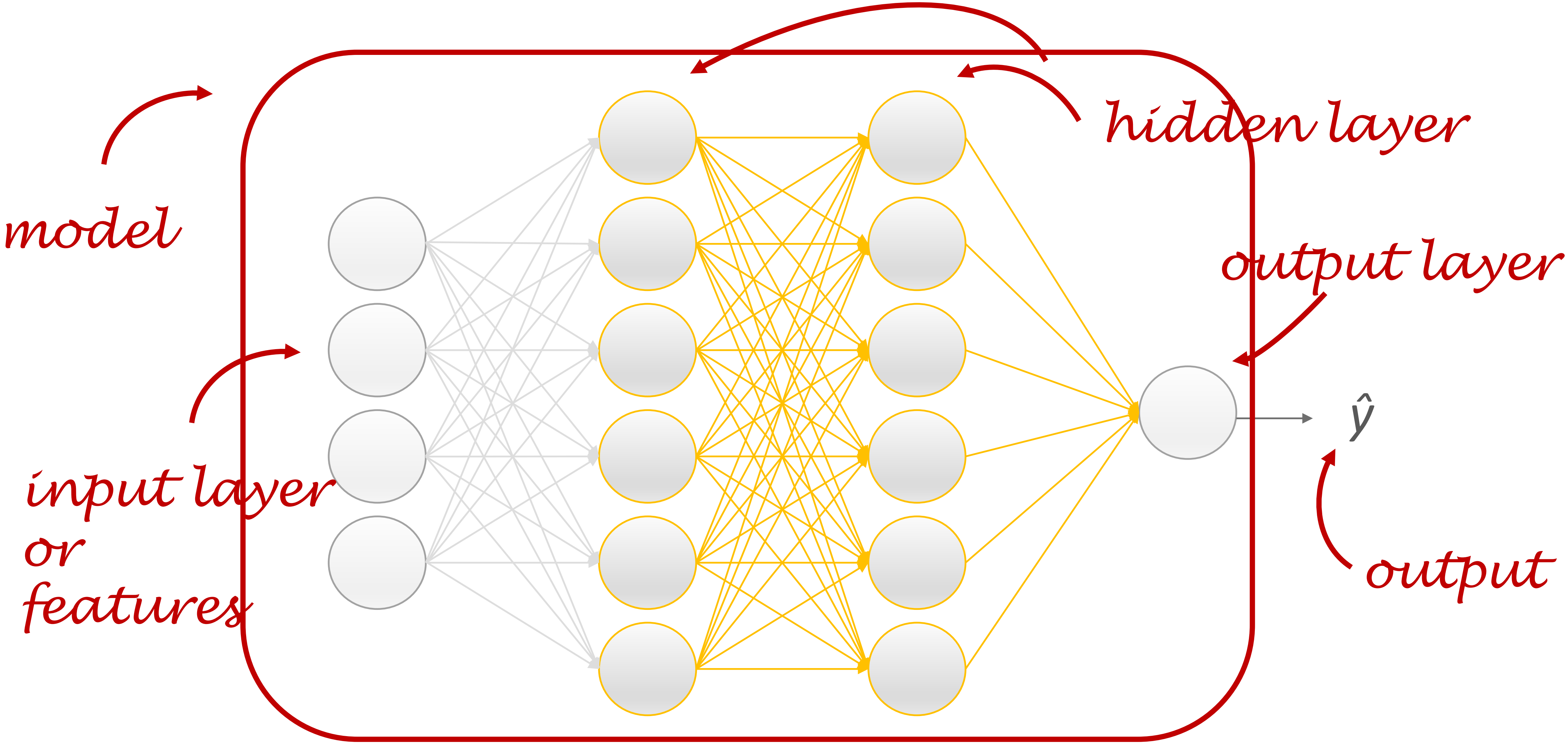
$$\hat{y} = wx$$

$$\hat{y} = w_1 x^2 + w_2 x + w_3$$

To "detect" various combinations of inputs



Feedforward ANN



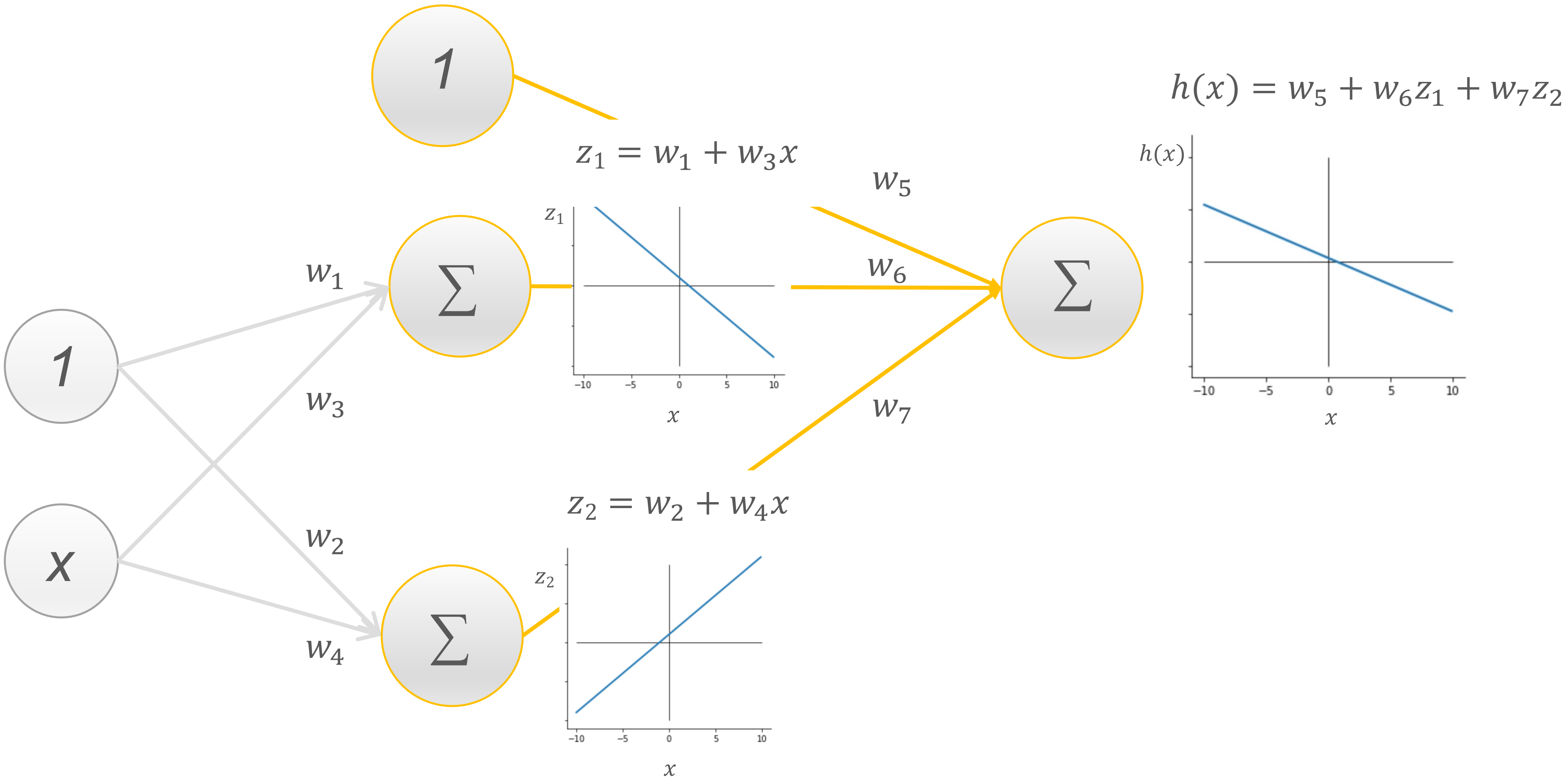
How many layers/ neurons?

https://www.deeplearningbook.org/contents/mlp.html

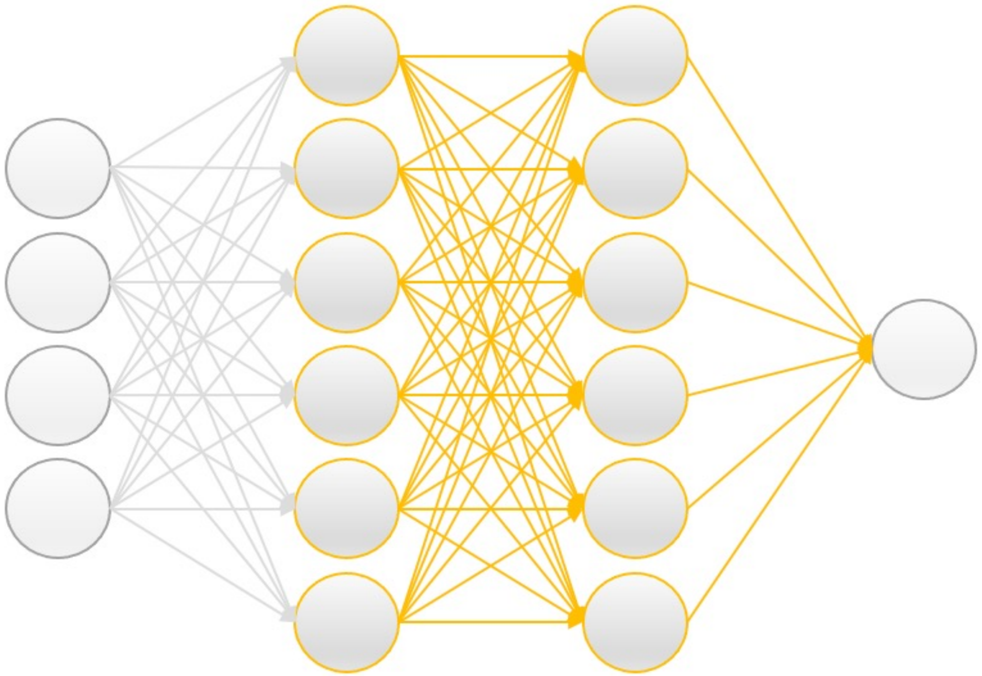
In summary, a feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly. In many circumstances, using deeper models can reduce the number of units required to represent the desired function and can reduce the amount of generalization error.

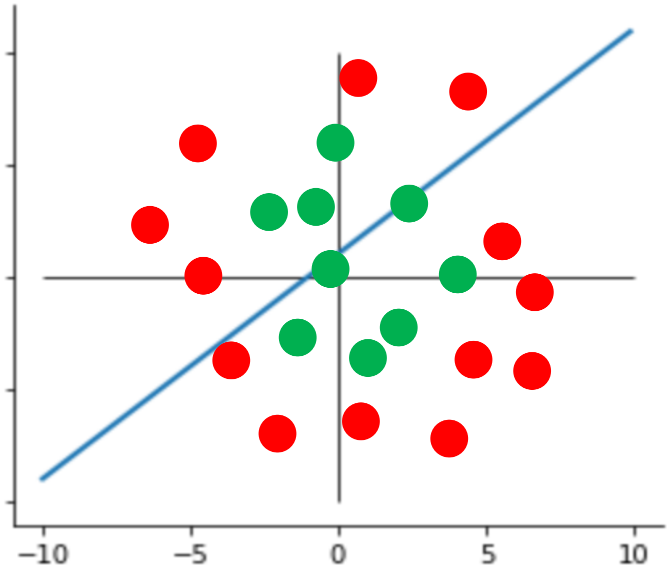
read more here

ANN without activation functions



ANN without activation functions – linear predictor

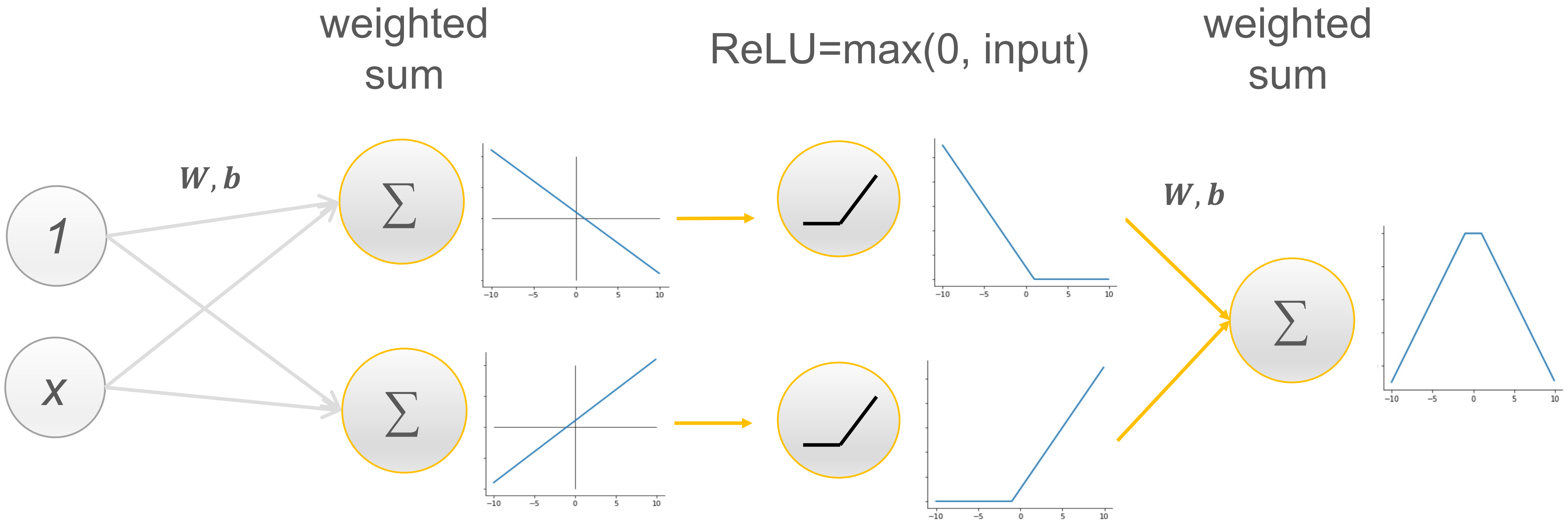




$$h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = \sum_{i=1}^{d} w_i x_i$$

linearly non-separable data

Introducing non-linearity with activation functions



Activation functions

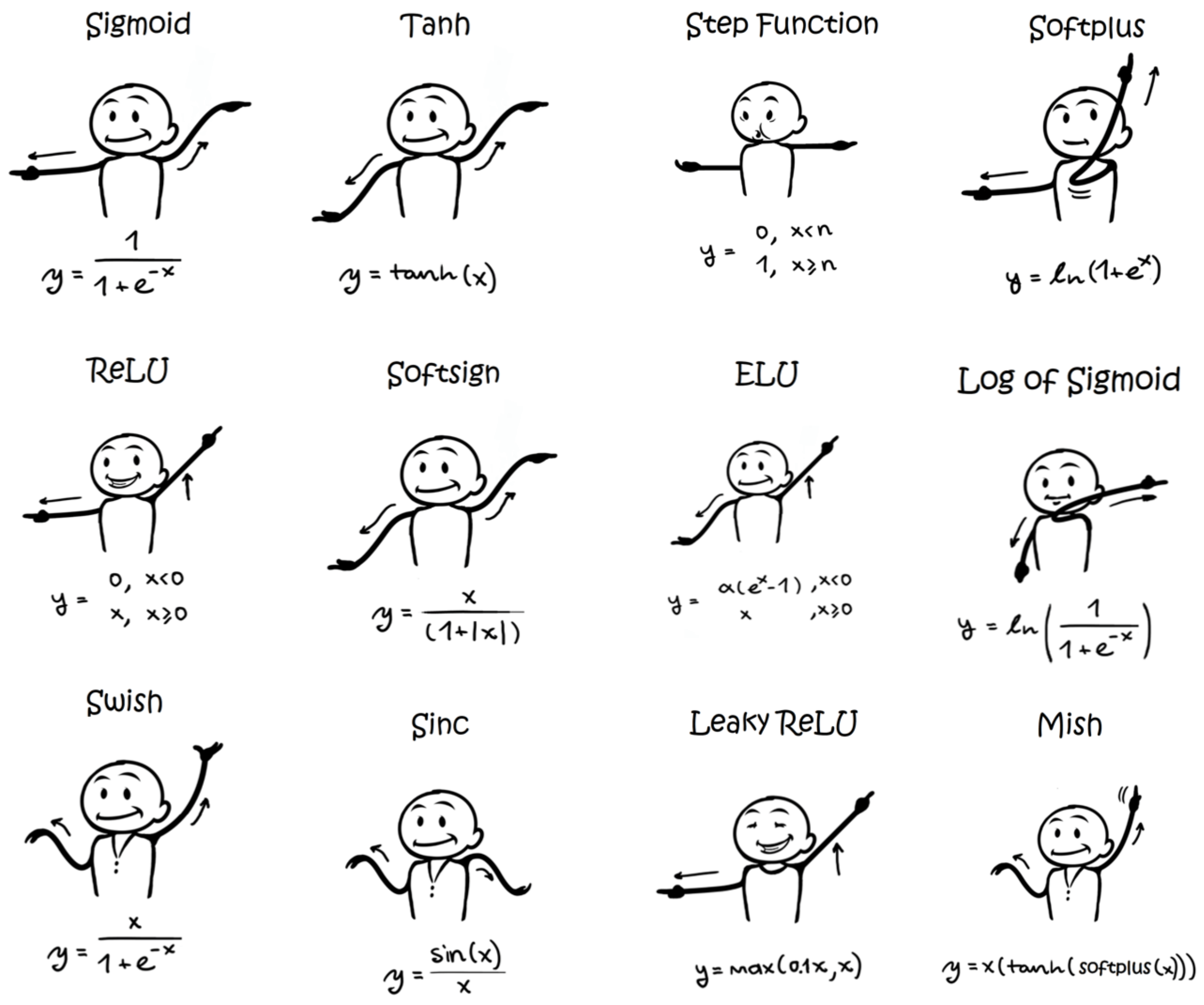
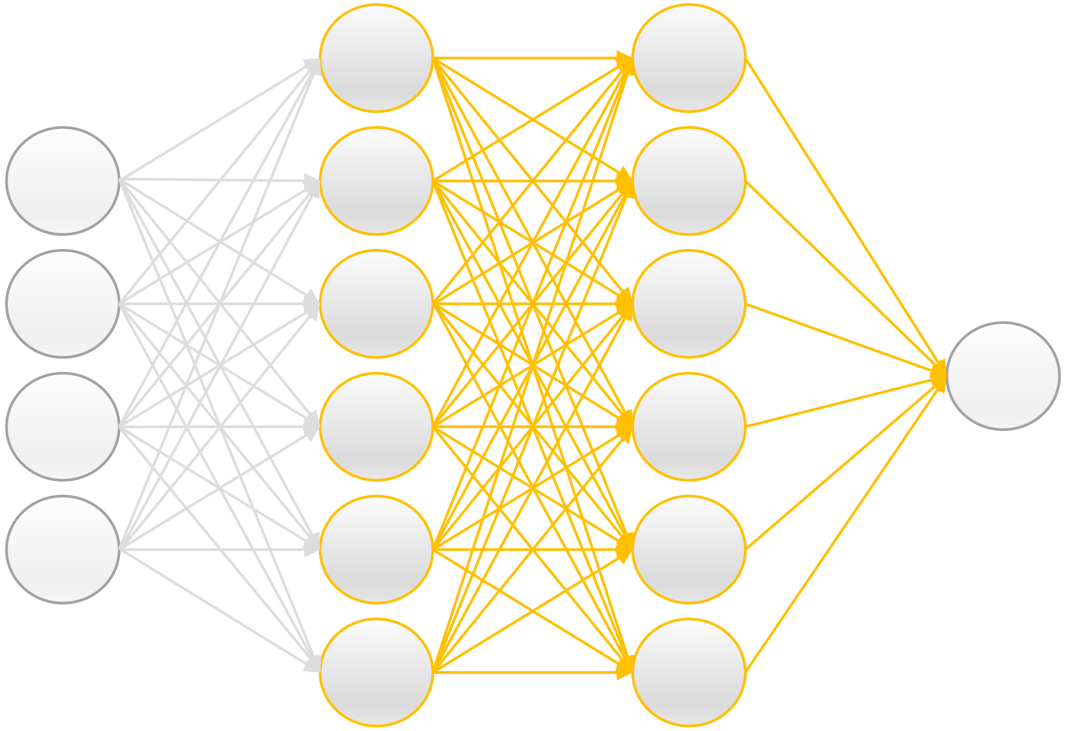


image source



input layer

hidden layer

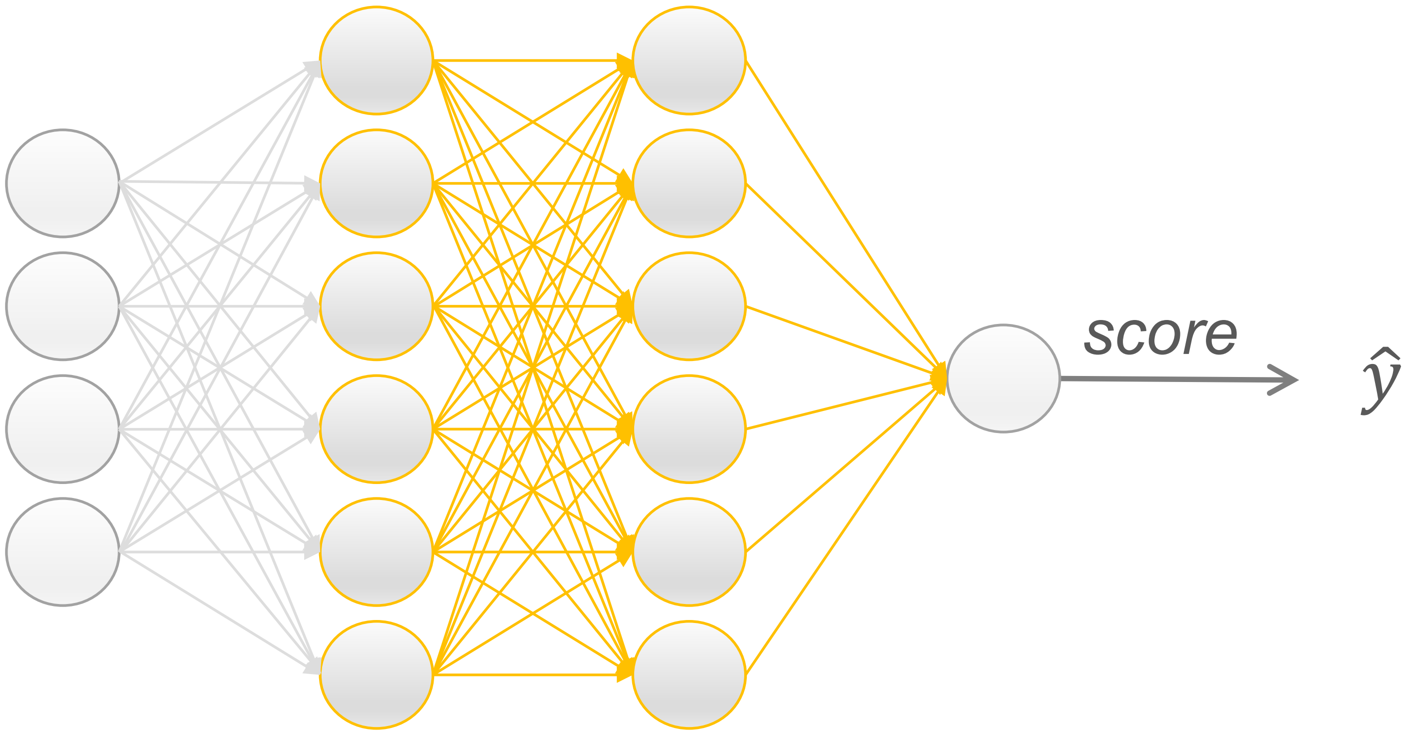
- ReLU
- Leaky ReLU
- ELU
- tanh

output layer

- sigmoid
- softmax
- None

ANN PLAYGROUND

Output layer for regression



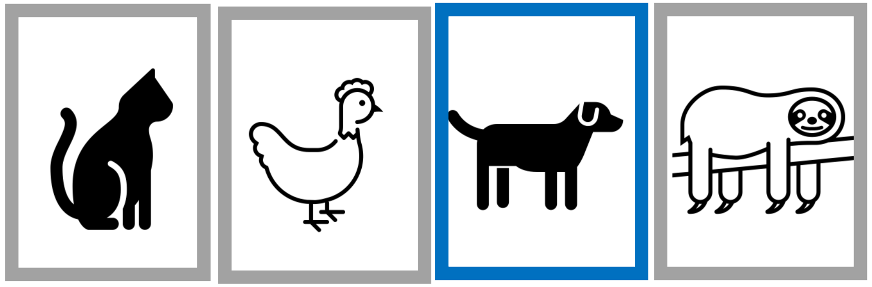
Score(s) – output of the last layer neuron(s) before applying activation function.

Classification task

Binary

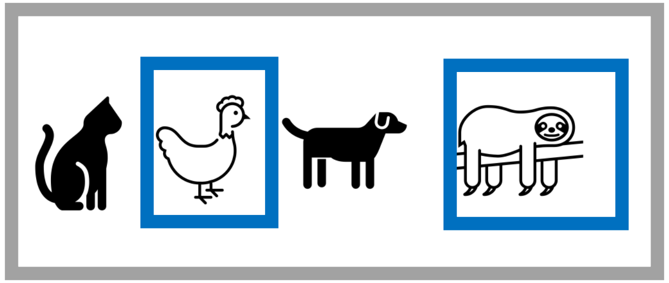
- spam
  y=1
- not spam
  y=0

Multiclass

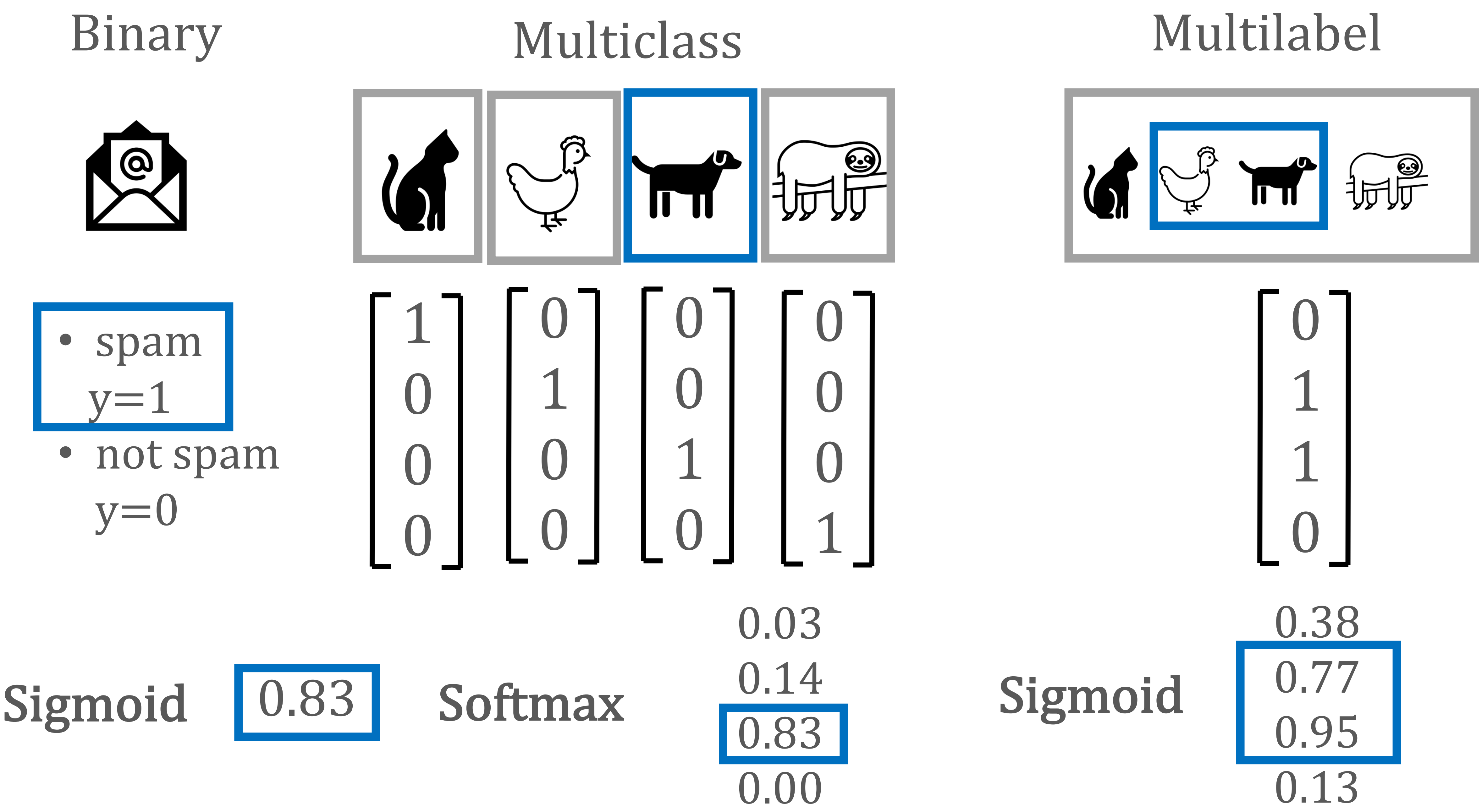


$$\begin{bmatrix}1\\0\\0\\0\end{bmatrix} \quad \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \quad \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \quad \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

Multilabel

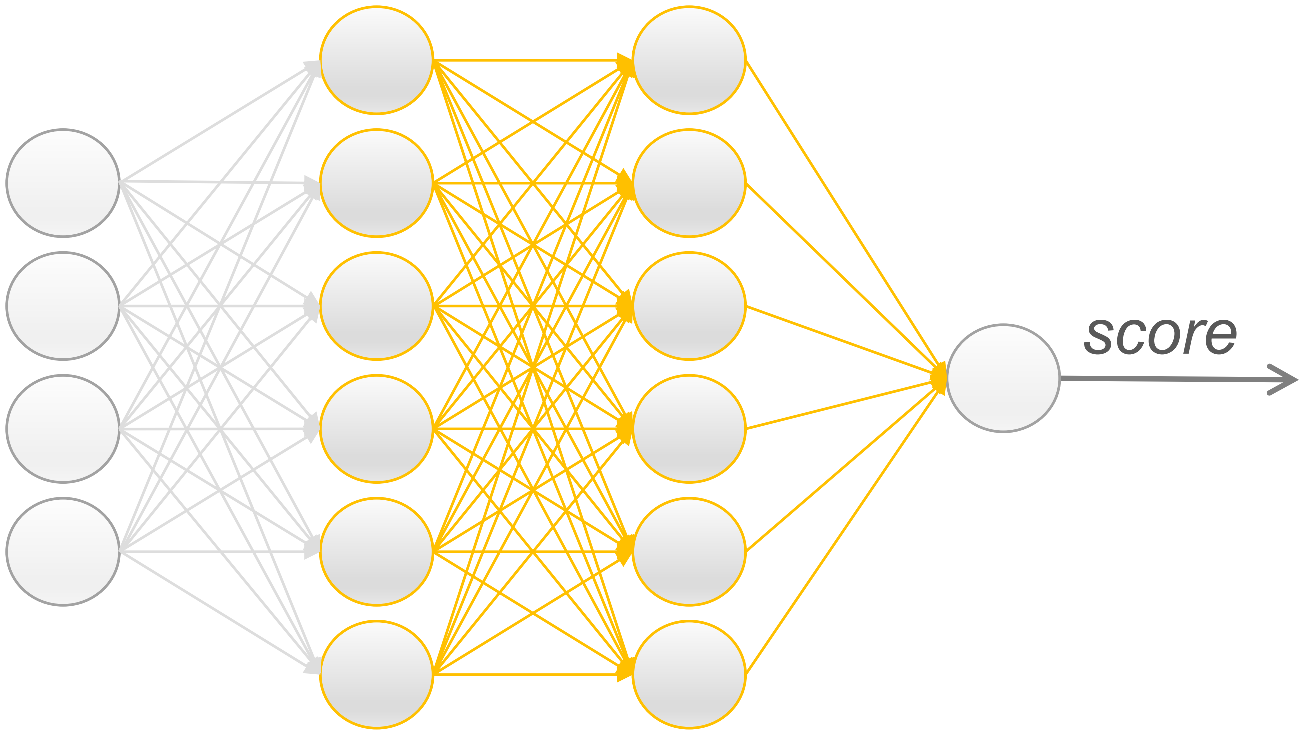


$$\begin{bmatrix}0\\1\\0\\1\end{bmatrix}$$



Binary
Multiclass
Multilabel
• spam
y=1
• not spam
y=0
$\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$ $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$ $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$ $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$
$\begin{bmatrix}0\\1\\1\\0\end{bmatrix}$
Sigmoid
0.83
Softmax
0.03
0.14
0.83
0.00
Sigmoid
0.38
0.77
0.95
0.13

Output layer for Binary clf

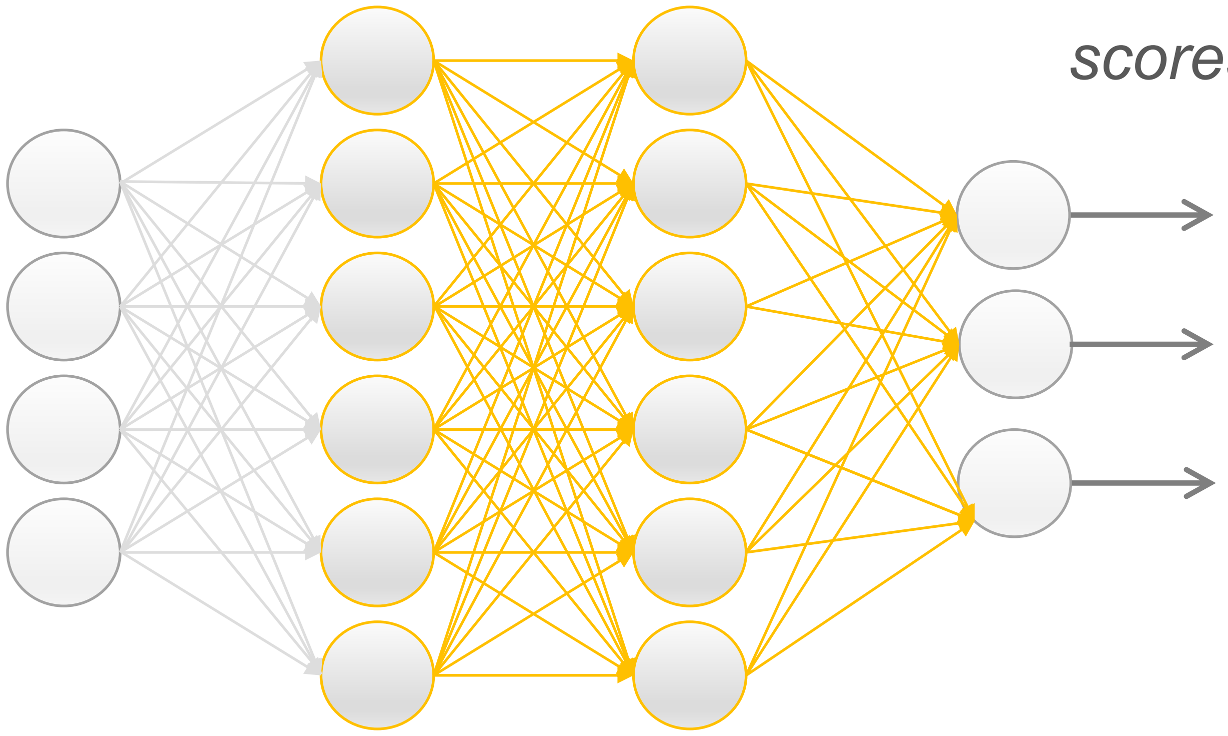


$$\sigma(score) \longrightarrow \hat{y}$$

$$\sigma = \frac{1}{1+e^{-x}}$$

Output layer for Multiclass clf

scores



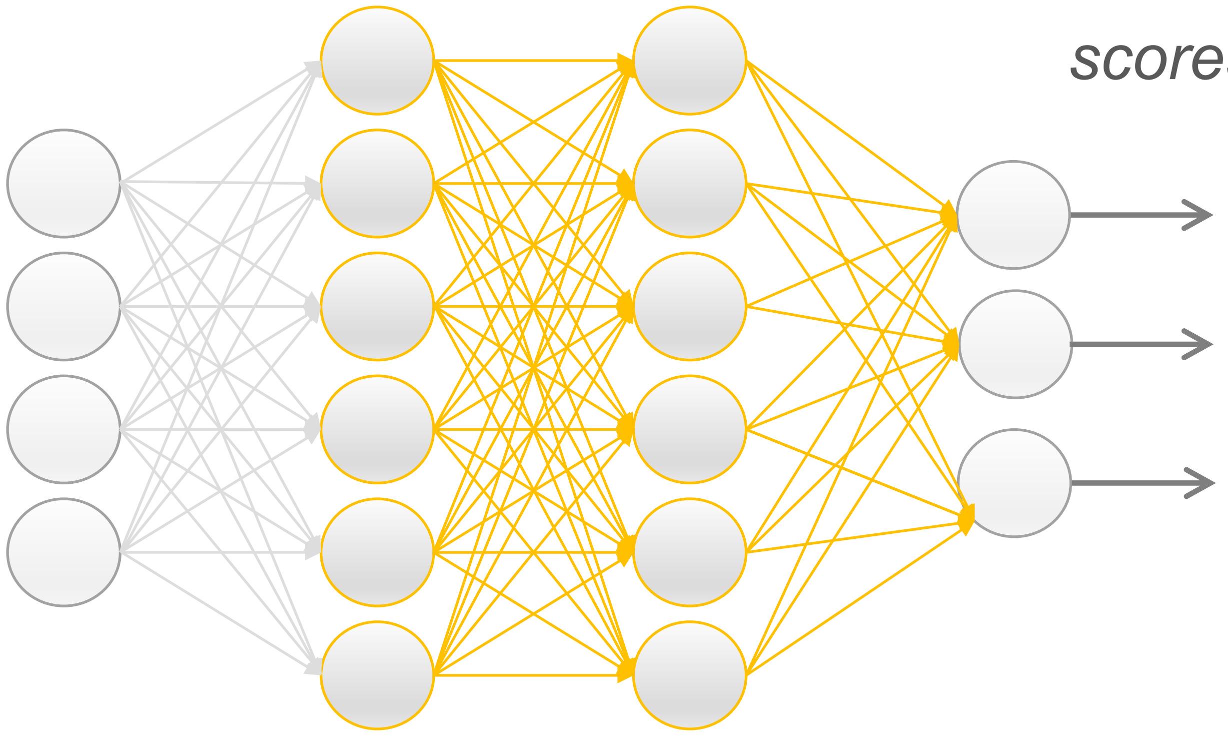
$\sigma(score) \longrightarrow \hat{y}$

$\sigma(score) \longrightarrow \hat{y}$

$\sigma(score) \longrightarrow \hat{y}$

$$\sigma = \frac{e^{s_i}}{\sum_{j=1}^{C} e^{s_j}}$$

Output layer for Multilabel clf



scores

$\sigma(score) \longrightarrow \hat{y}$

$\sigma(score) \longrightarrow \hat{y}$

$\sigma(score) \longrightarrow \hat{y}$

$$\sigma = \frac{1}{1 + e^{-x}}$$

applied independently to each element

Output (predictions)

$\hat{y}_i$

scores s_i

$$\begin{bmatrix} -0.5 \\ 1.2 \\ 3 \\ -2 \end{bmatrix}$$

Sigmoid $\dfrac{1}{1+e^{-s_i}}$ → $\begin{bmatrix} 0.38 \\ 0.77 \\ 0.95 \\ 0.13 \end{bmatrix}$

cat
not cat
dog
not dog

Softmax $\dfrac{e^{s_i}}{\sum_{j=1}^{C} e^{s_j}}$ → $\begin{bmatrix} 0.03 \\ 0.14 \\ 0.83 \\ 0.00 \end{bmatrix}$

cat
dog

outputs sums up to 1, depends on all elements

Cross-entropy loss

n.o. classes

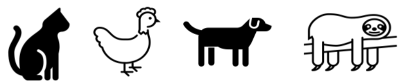
$$\boxed{CE = -\sum_{i}^{C} y_i \, log(\hat{y}_i)}$$

+ Sigmoid

Binary

spam not spam

Multilabel



$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

cat
not cat

Binary CE

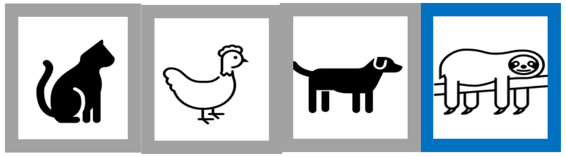
$$y_2 = 1 - y_1$$
$$\hat{y}_2 = 1 - \hat{y}_1$$

$$CE = -\sum_{i}^{C=2} y_i \, log(\hat{y}_i) = -\, y_1 \, log(\hat{y}_1) - (1 - y_1) \, log(1 - \hat{y}_1)$$

Cross-entropy loss

$$CE = -\sum_{i}^{C} y_i \, log(\hat{y}_i)$$

Multiclass



$$\begin{bmatrix}1\\0\\0\\0\end{bmatrix} \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

y_i for non-positive class is zero

y_i for positive class is one

+ Softmax

Categorical CE

$$CE = -\sum_{i}^{C} y_i \, log(\hat{y}_i) = -\log\left(\frac{e^{s_p}}{\sum_{j=1}^{C} e^{s_j}}\right)$$

score for positive class

Last-layer activation function + loss

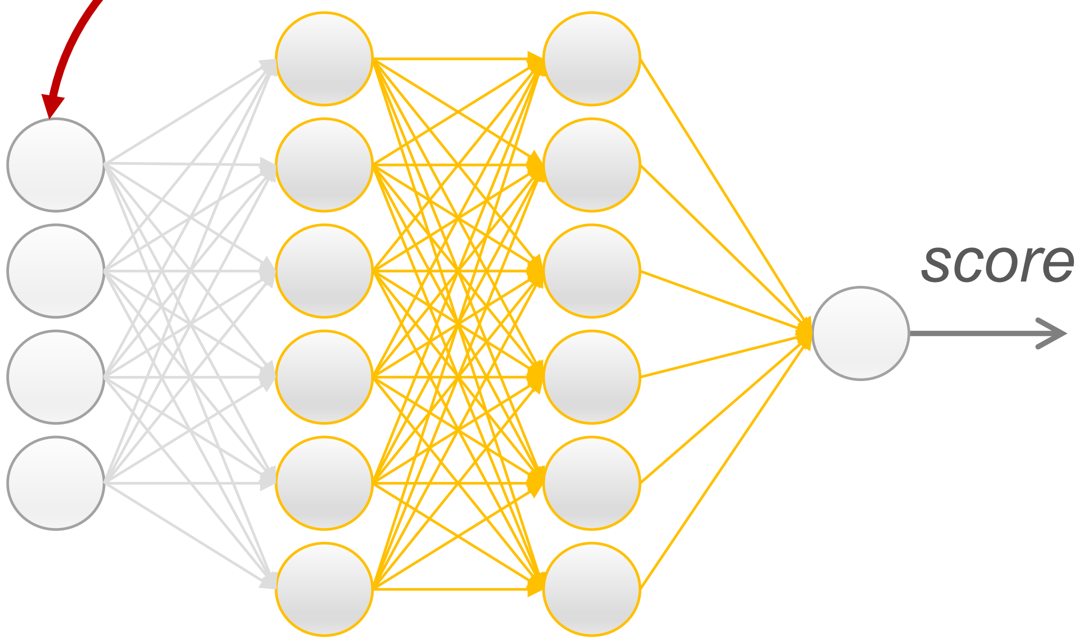
Table 4.1 Choosing the right last-layer activation and loss function for your model

Problem type	Last-layer activation	Loss function
Binary classification	sigmoid	binary_crossentropy
Multiclass, single-label classification	softmax	categorical_crossentropy
Multiclass, multilabel classification	sigmoid	binary_crossentropy
Regression to arbitrary values	None	mse
Regression to values between 0 and 1	sigmoid	mse or binary_crossentropy

DL with Python, Chollet

ANN – vectors and matrices

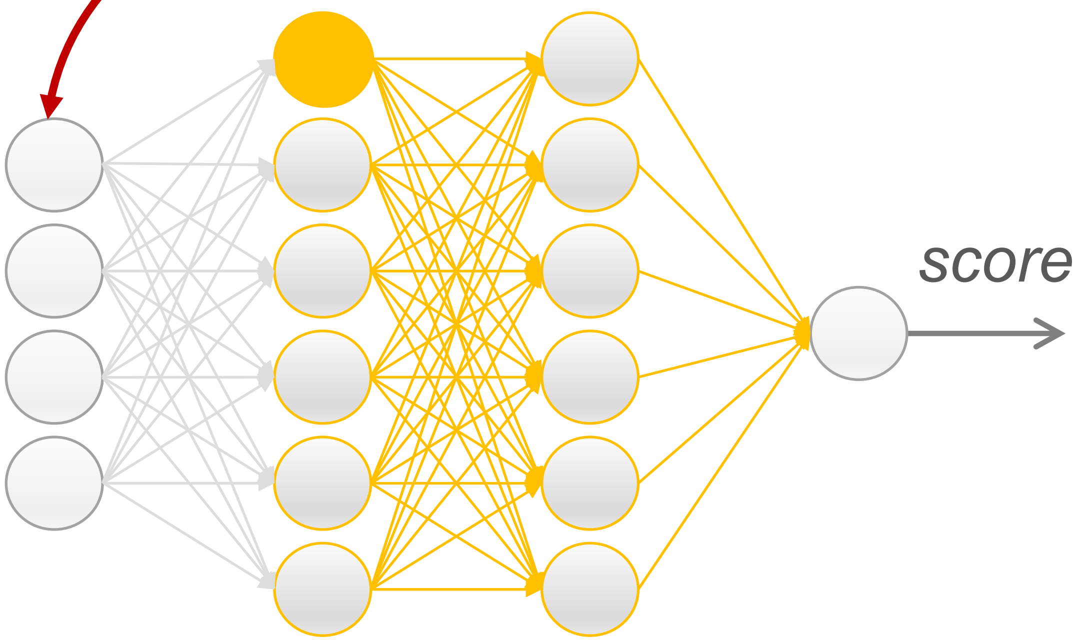
$$\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, x_4^{(i)})$$ feature vector of ith datapoint



$$\mathbf{w}_1^{(1)} = (w_{1,1}^{(1)}, w_{1,2}^{(1)}, w_{1,3}^{(1)}, w_{1,4}^{(1)})$$ weight vector of 1st hidden neuron of 1st hidden layer

ANN – vectors and matrices

$$\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, x_4^{(i)})$$



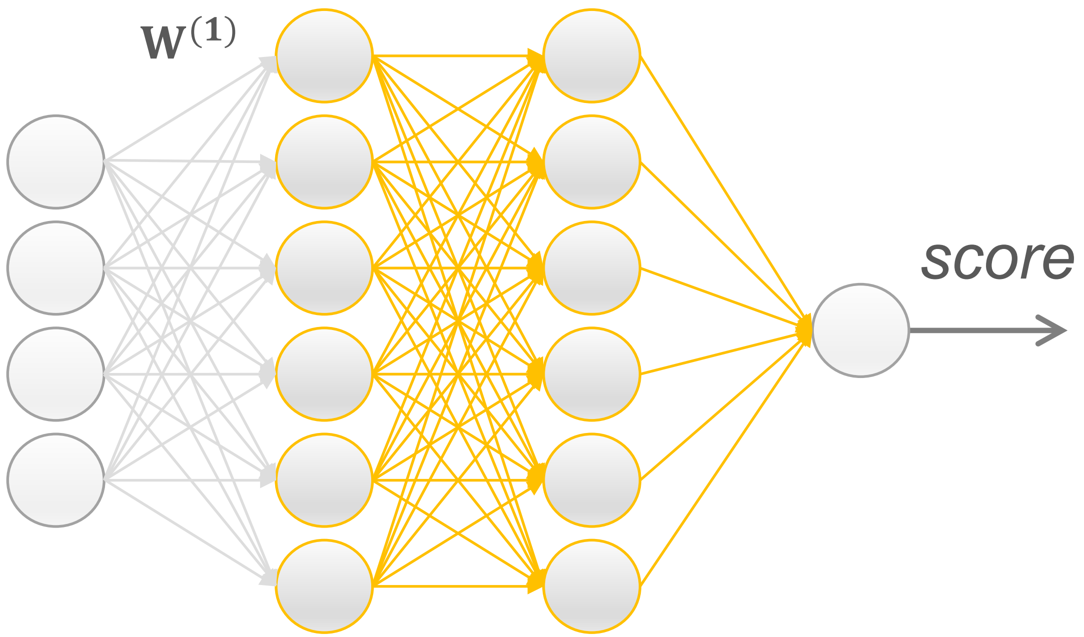
$$\mathbf{w}_1^{(1)} = (w_{1,1}^{(1)}, w_{1,2}^{(1)}, w_{1,3}^{(1)}, w_{1,4}^{(1)})$$

output of 1st hidden neuron
of 1st hidden layer for ith datapoint

$$\sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}\right) = \mathbf{out}$$

(1,4) (4,1) (1,1)

ANN – vectors and matrices



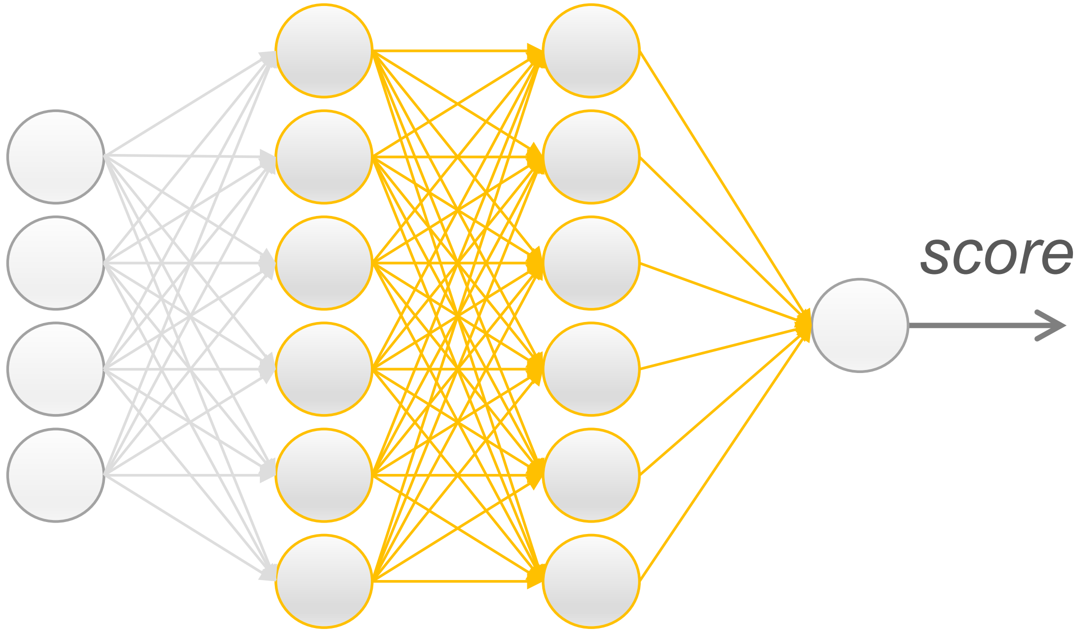
Feature matrix; shape (m,4)

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & \dots & x_4^{(1)} \\ \dots & \dots & \dots \\ x_1^{(m)} & \dots & x_4^{(m)} \end{bmatrix}$$

Weight matrix of the 1st hidden layer; shape (4,6)

$$\mathbf{W}^{(1)} = \begin{bmatrix} w_{1,1}^{(1)} & \dots & w_{6,1}^{(1)} \\ \dots & \dots & \dots \\ w_{1,4}^{(1)} & \dots & w_{6,4}^{(1)} \end{bmatrix}$$

ANN – vectors and matrices

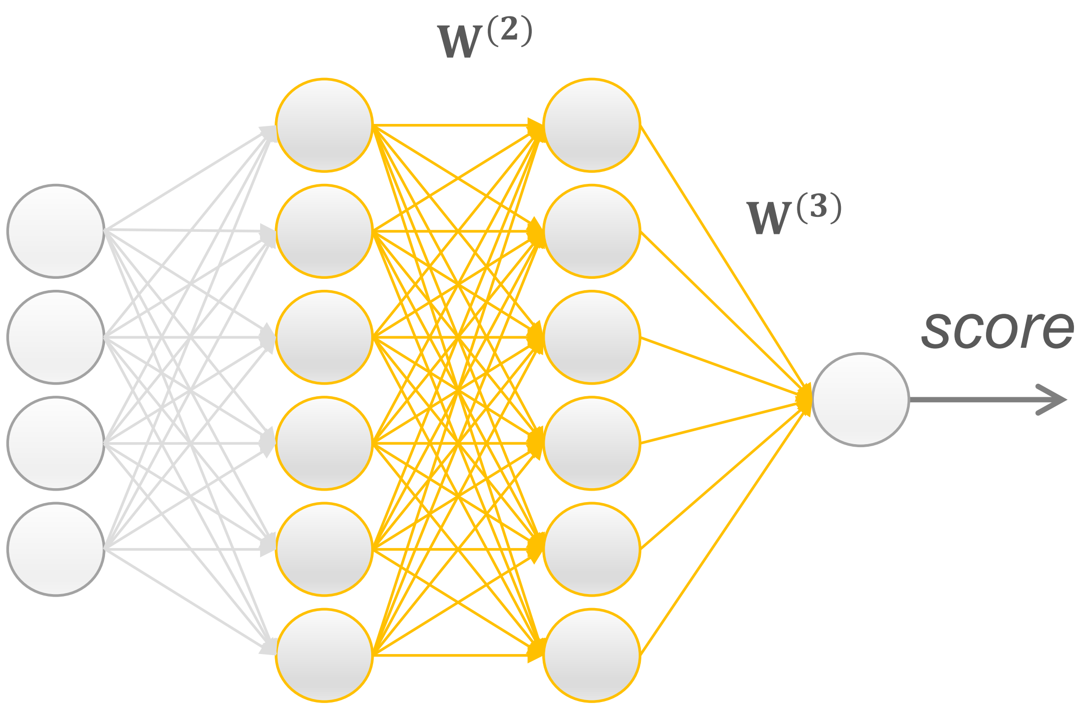


output of 1st hidden layer for m datapoints

$$\sigma\left(\mathbf{X}\mathbf{W}^{(1)}\right) = \boldsymbol{h}^{(1)}$$

(m,4) (4,6) (m,6)

ANN – vectors and matrices



output of 2nd hidden layer for m datapoints

$$\sigma\left(\boldsymbol{h}^{(1)}\mathbf{W}^{(2)}\right) = \boldsymbol{h}^{(2)}$$

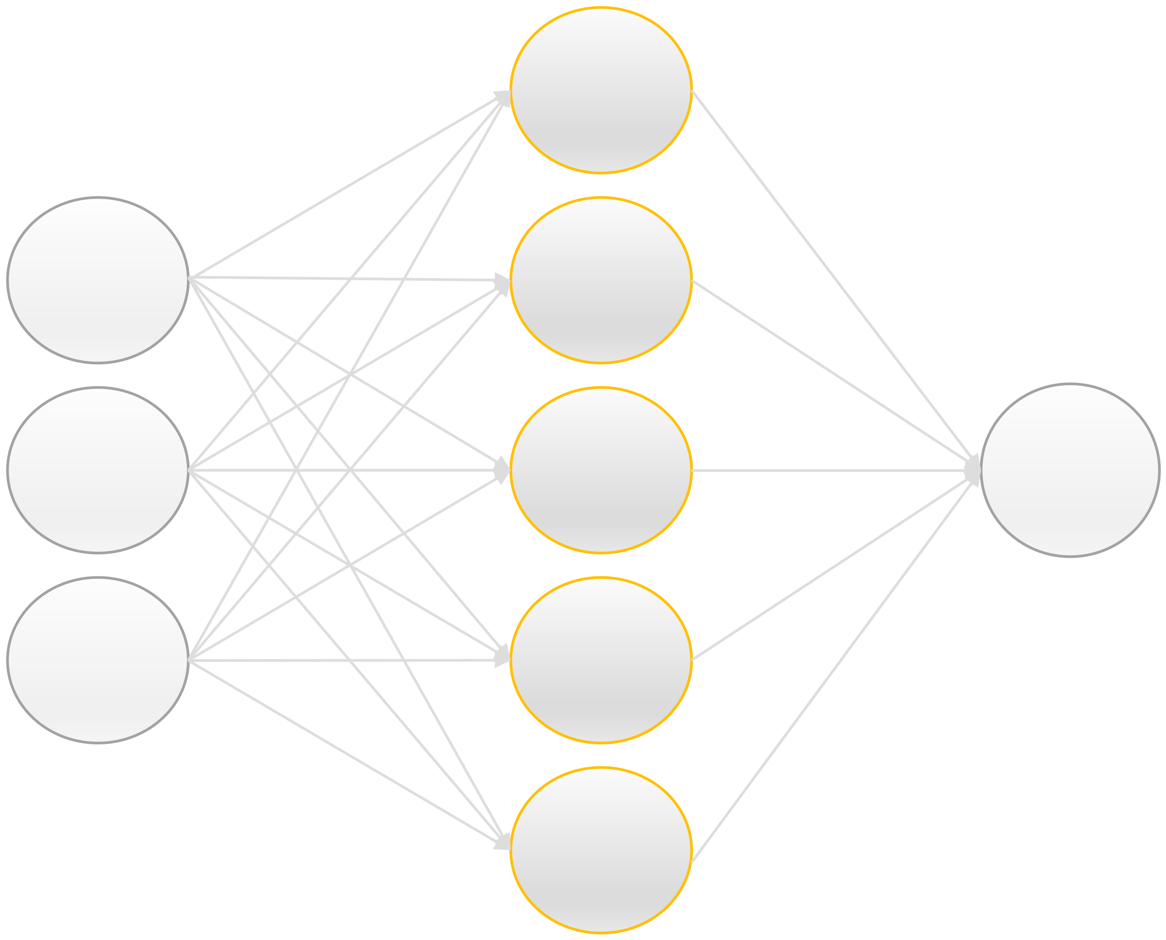
(m,6) (6,6) (m,6)

output score for m datapoints

$$\boldsymbol{h}^{(2)}\mathbf{W}^{(3)} = score$$

(m,6) (6,1) (m,1)

How to find good model parameters?

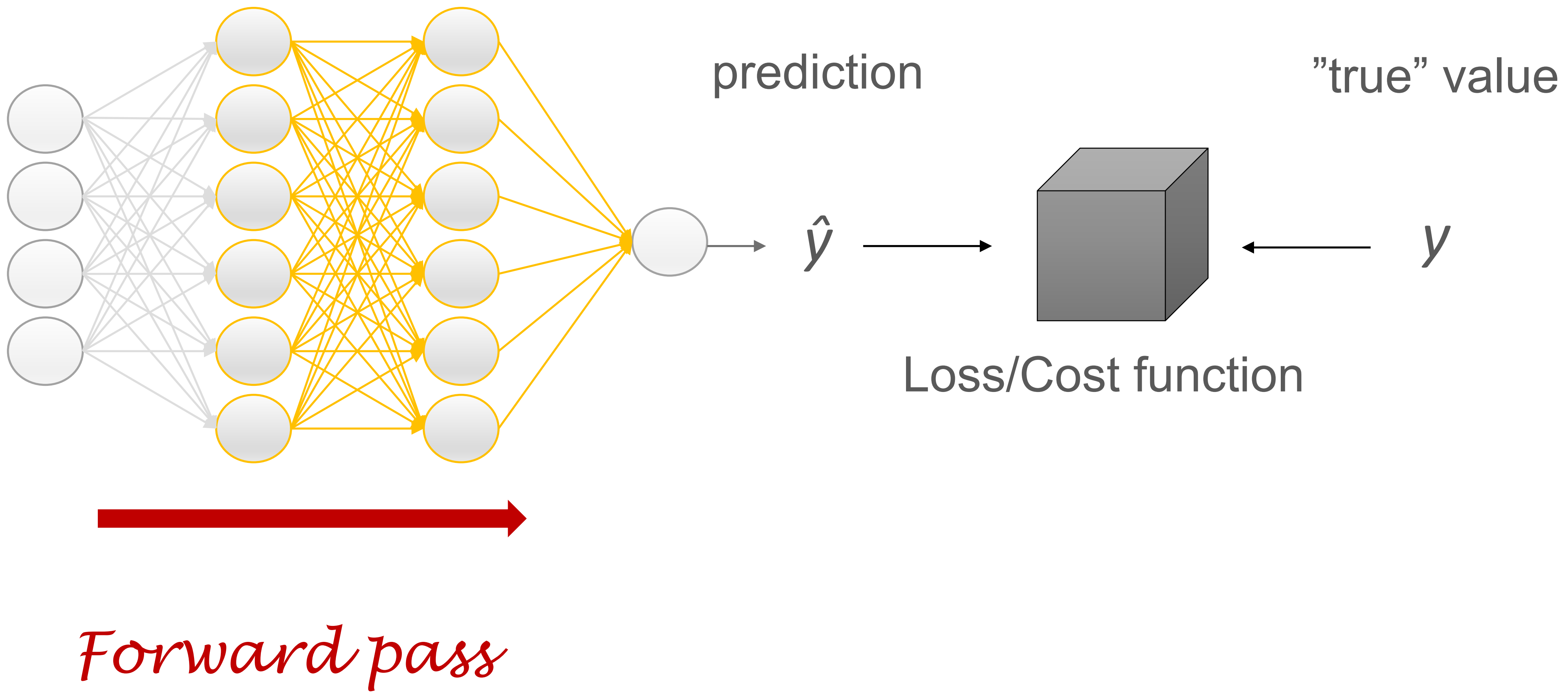


n.o. parameters = 3*5 + 5*1

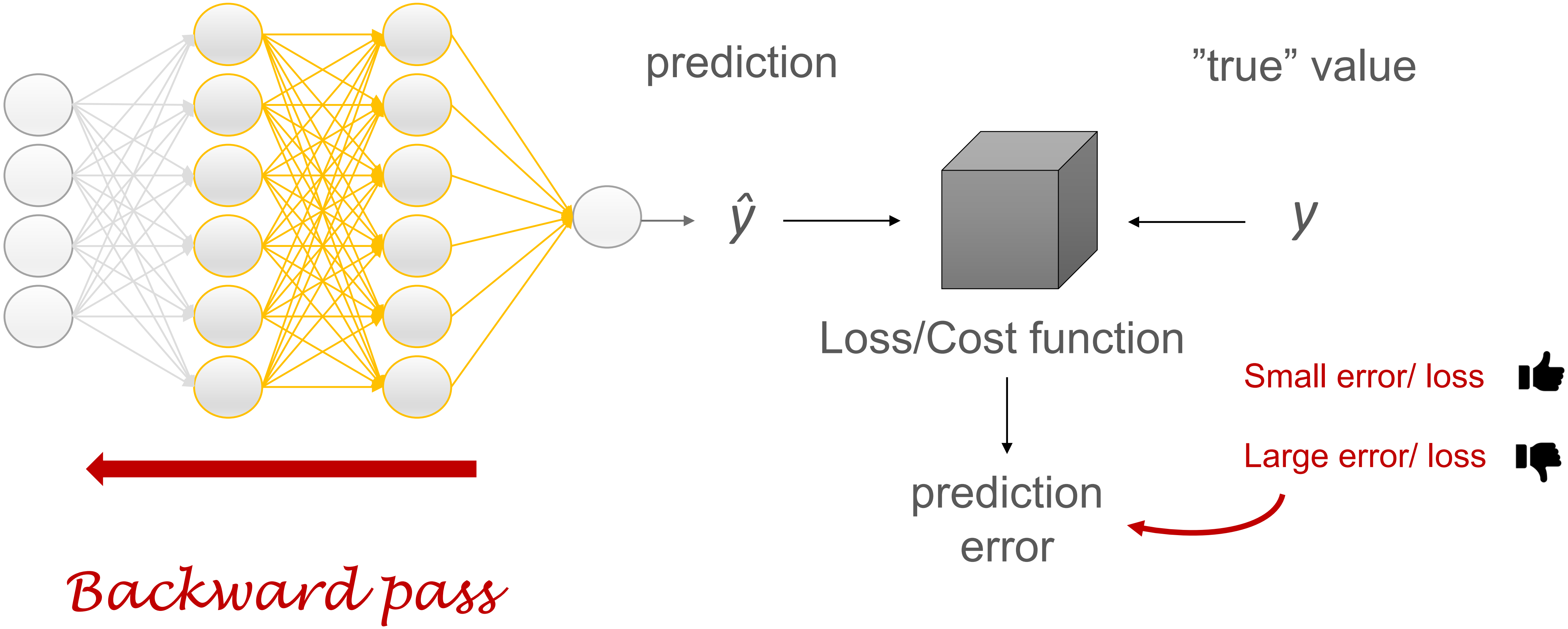
$w_1, w_2, w_3, w_4, w_5, \ldots$???

$\sigma(w_1 x_1 + \cdots + w_n x_n)$

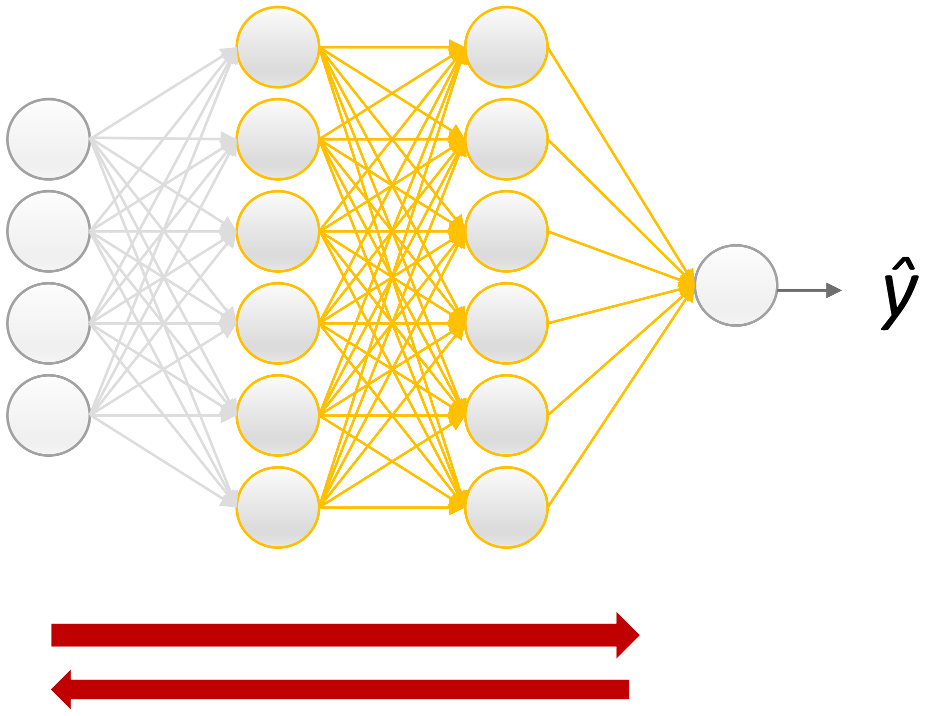
Gradient Descent Algorithm

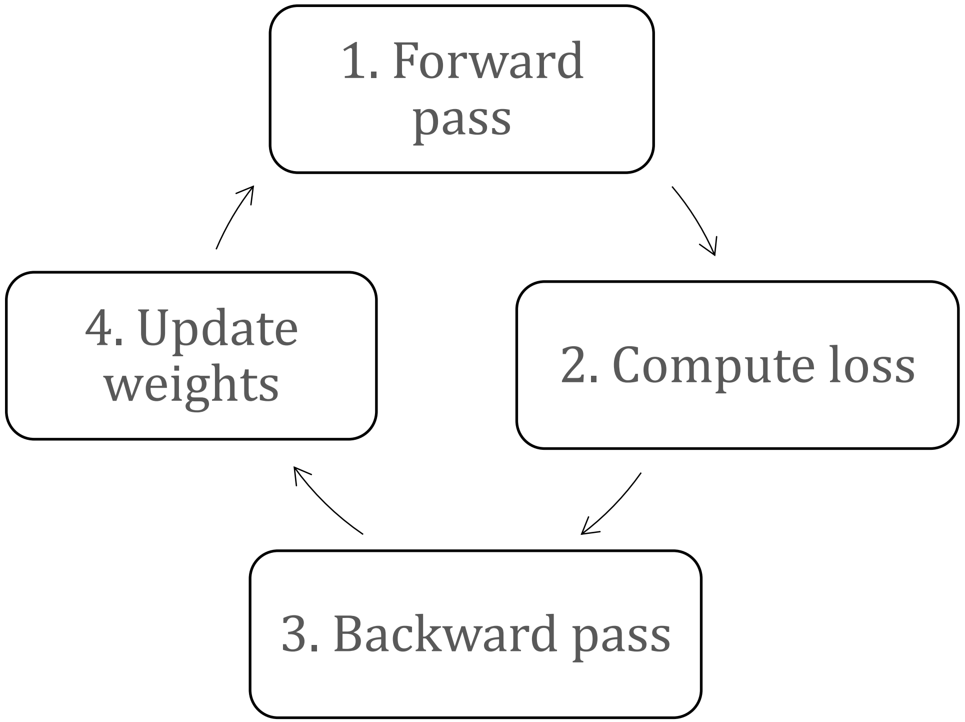


Gradient Descent Algorithm



Gradient Descent Algorithm



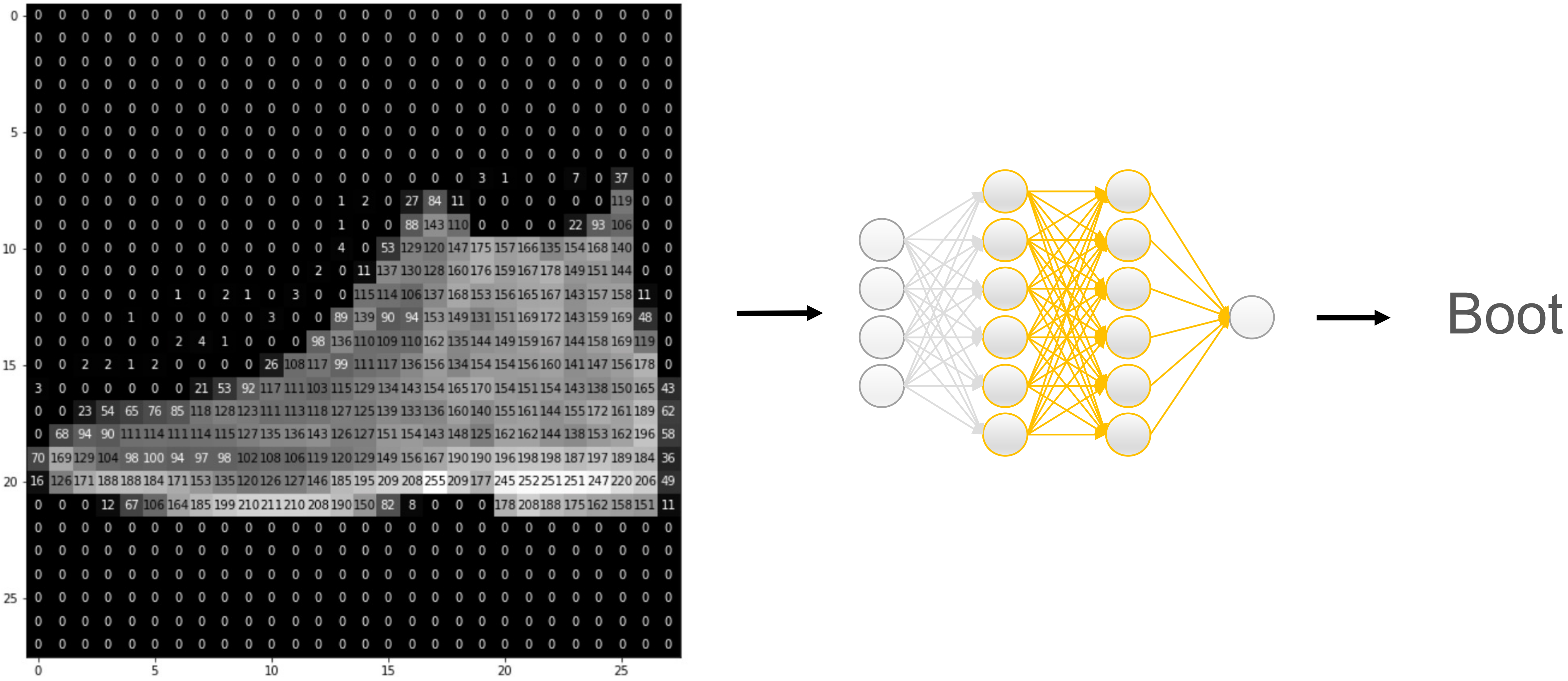


Hyperparameters

- Batch size
- Learning rate
- Epochs
- Optimizer

- n.o. neurons, layers
- activation functions

Hands on: shallow ANN for classification



Tools

Here we keep our code

Python libraries:

operations with arrays/ matrices

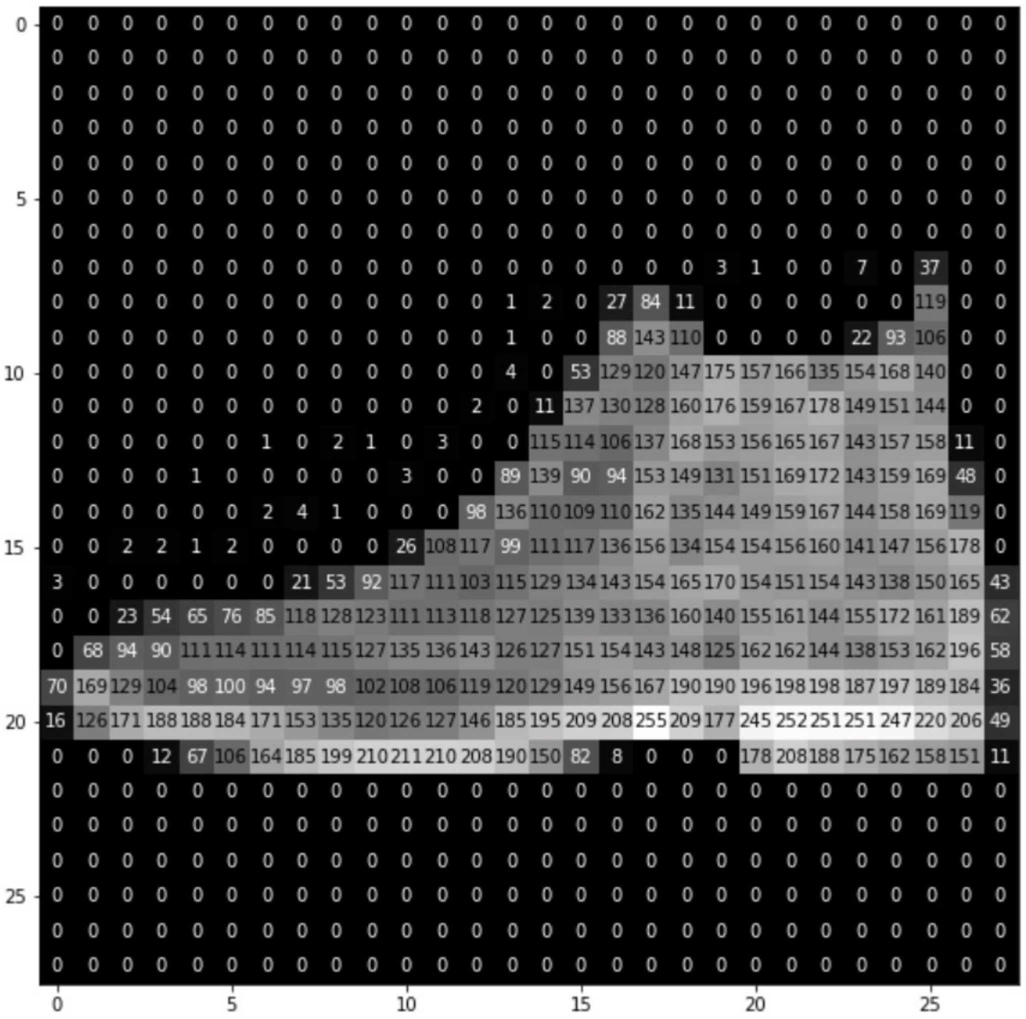
graphs

Here we run a code

deep learning

Data

- Data point: 28x28 grayscale image of shopping item
- Features: 28x28=784 pixel values (range 0-255)
- Labels: Product category (boot, shirt, hat, …). 10 classes in total.



Typical Workflow

1. Data preprossing (scale 0-1), splitting (training/validation/test).

2. Define ANN model

3. Choose Loss function, Optimizer

4. Train ANN

5. Evaluate on unseen (test) dataset

Data preprossing

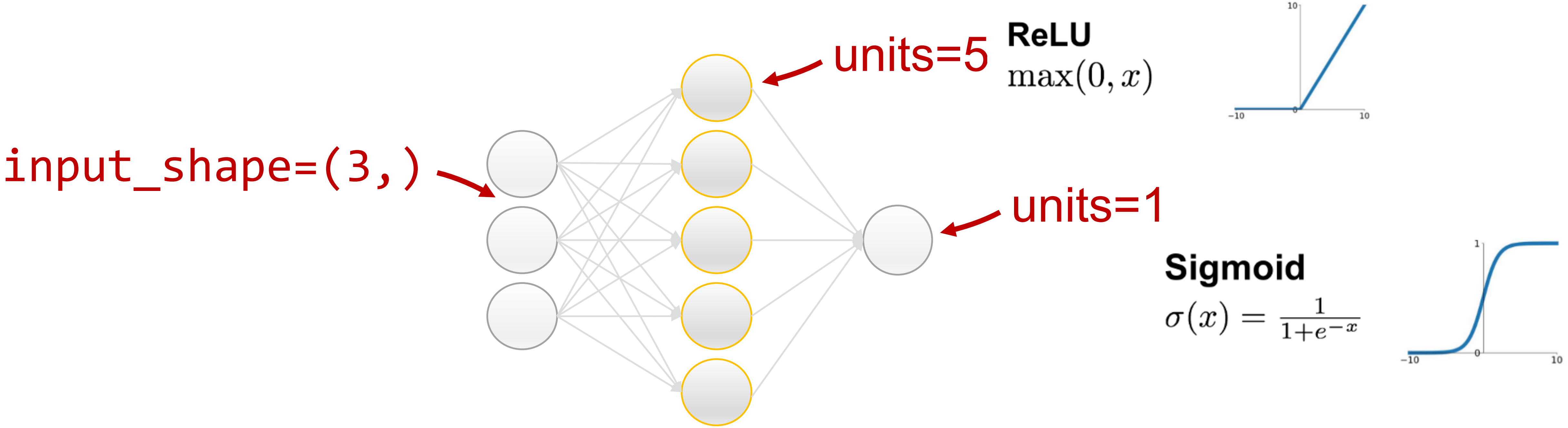
1. Scaling features to 0-1: makes it easier for algoritm to find “good” model parameters

2. Splitting (training/validation/test datasets):

    - Training – to find “good” model parameters
    - Validation – to choose model hyper-parameters (learning rate, n.o. epochs)
    - Test – final performance evaluation

Define ANN model

```
model = keras.Sequential([
        layers.Dense(units=5, activation='relu',input_shape=(3,)),
        layers.Dense(units= 1, activation='sigmoid')
        ])
```



Choose Loss function, Optimizer

Variation of basic Gradient Descent Algorithm

```
model.compile(optimizer='RMSprop',
              loss='sparse_categorical_crossentropy',
              metrics=['sparse_categorical_accuracy'])
```

Compare predictions and labels

Easy to understand performance measure

Train ANN

Features and labels

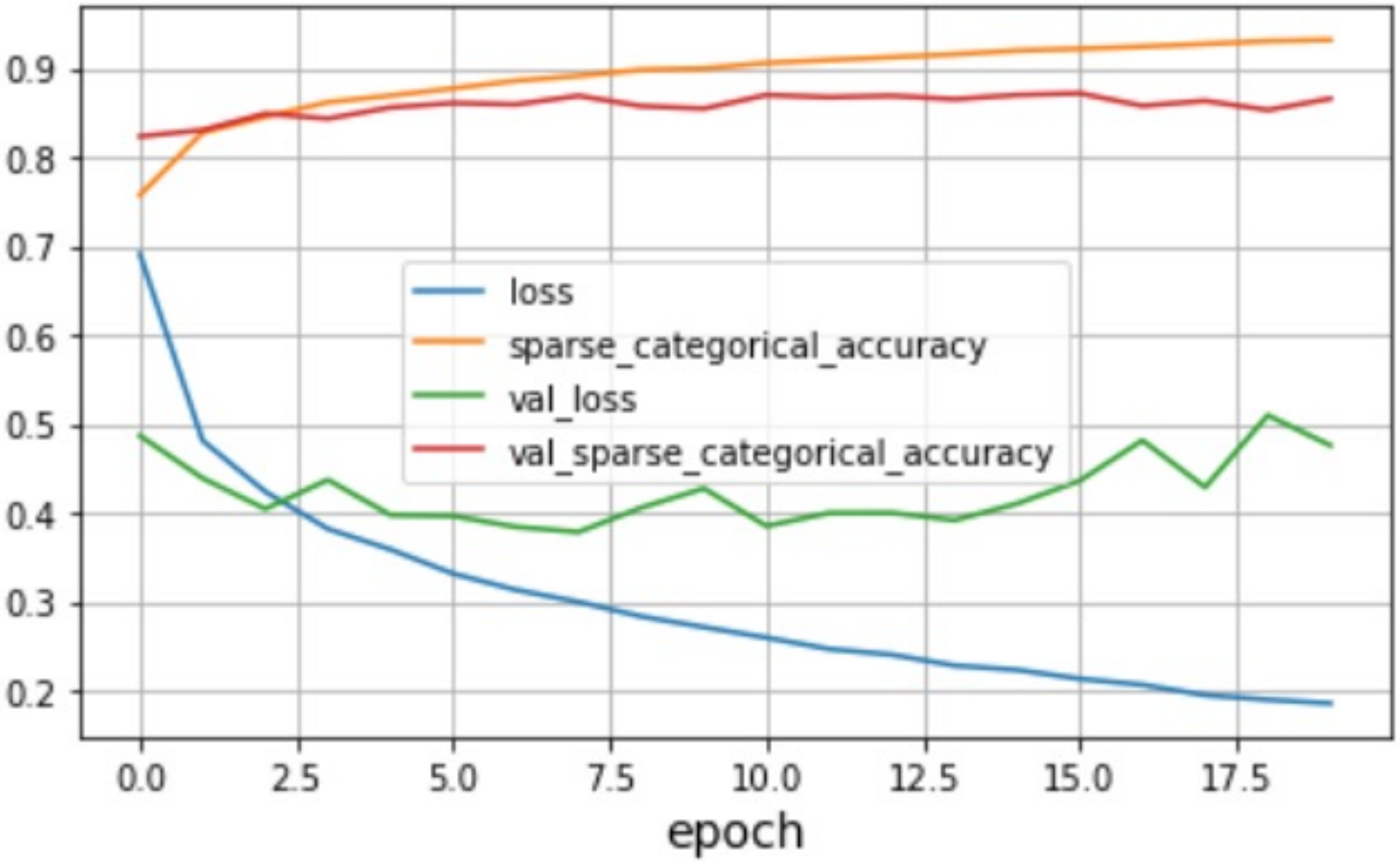
```
history = model.fit(X_trainval,
                    y_trainval,
                    validation_split=0.2,
                    batch_size=32,
                    epochs=20,
                    verbose=1)
```

Training data = 80%
Validation = 20%

Feed data by batches of 32

Printing info during training

Train ANN



0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
loss
sparse_categorical_accuracy
val_loss
val_sparse_categorical_accuracy
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
epoch

Evaluate on test dataset

```
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print('Accuracy on test dataset:', test_accuracy)
```