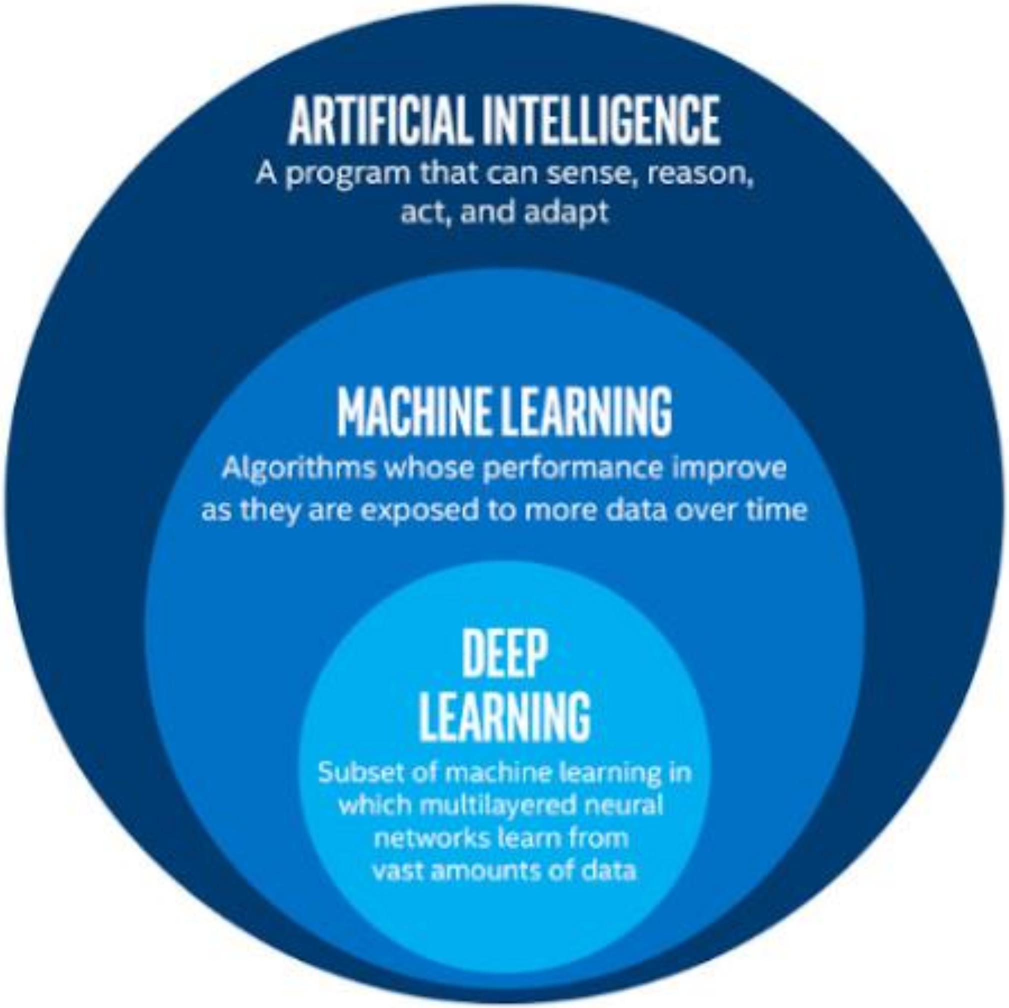
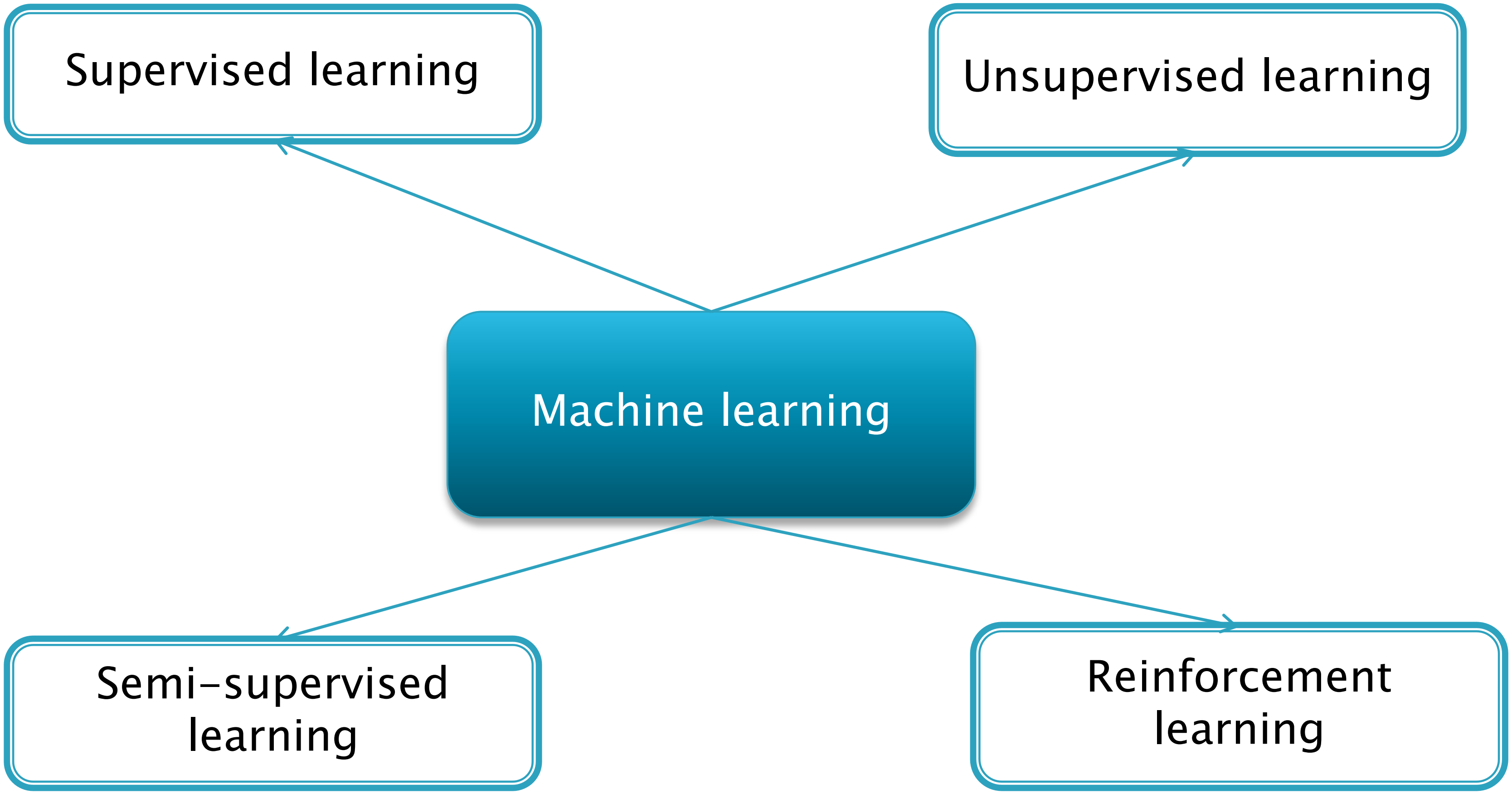
# Neural Networks




https://builtin.com/machine-learning/what-is-deep-learning


Supervised learning
Unsupervised learning
Machine learning
Semi-supervised learning
Reinforcement learning

# Supervised learning

Data: $(\mathbf{X}_i, \mathbf{Y}_i), i = 1, \ldots, N,$

N – number of examples (objects, samples) in the dataset;

$\mathbf{X}_i = \{x_{i,j} : j = 1, \ldots, N_I\}$ – feature vector;

$\mathbf{Y}_i = \{y_{i,j} : j = 1, \ldots, N_O\}$ – output vector.

$$\mathbf{X}_i = \{x_{i,j} : j = 1, \ldots, N_I\}$$

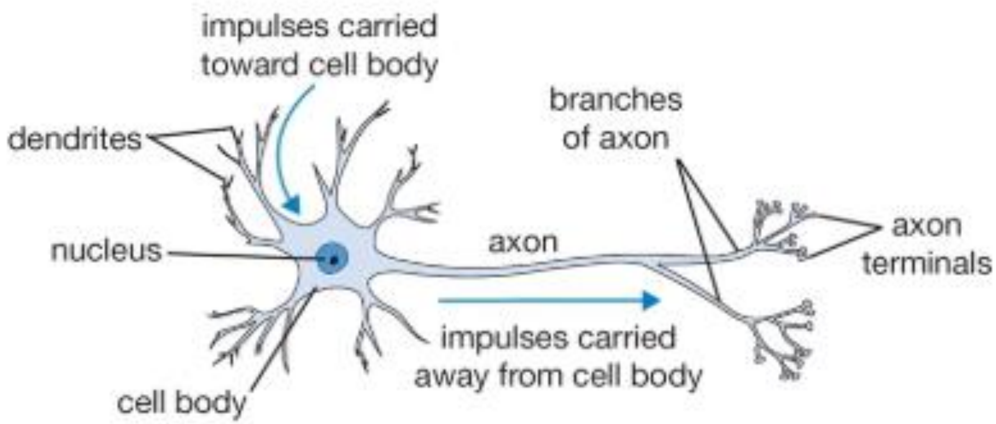
Model

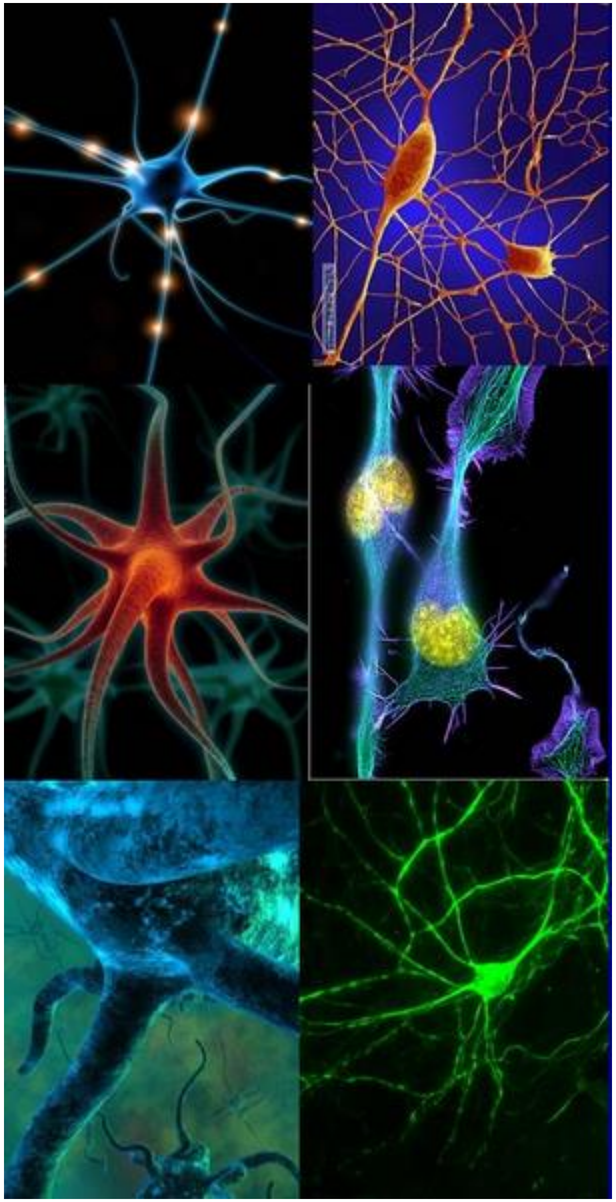
$$\hat{\mathbf{Y}}_i = \{\hat{y}_{i,j} : j = 1, \ldots, N_O\}$$

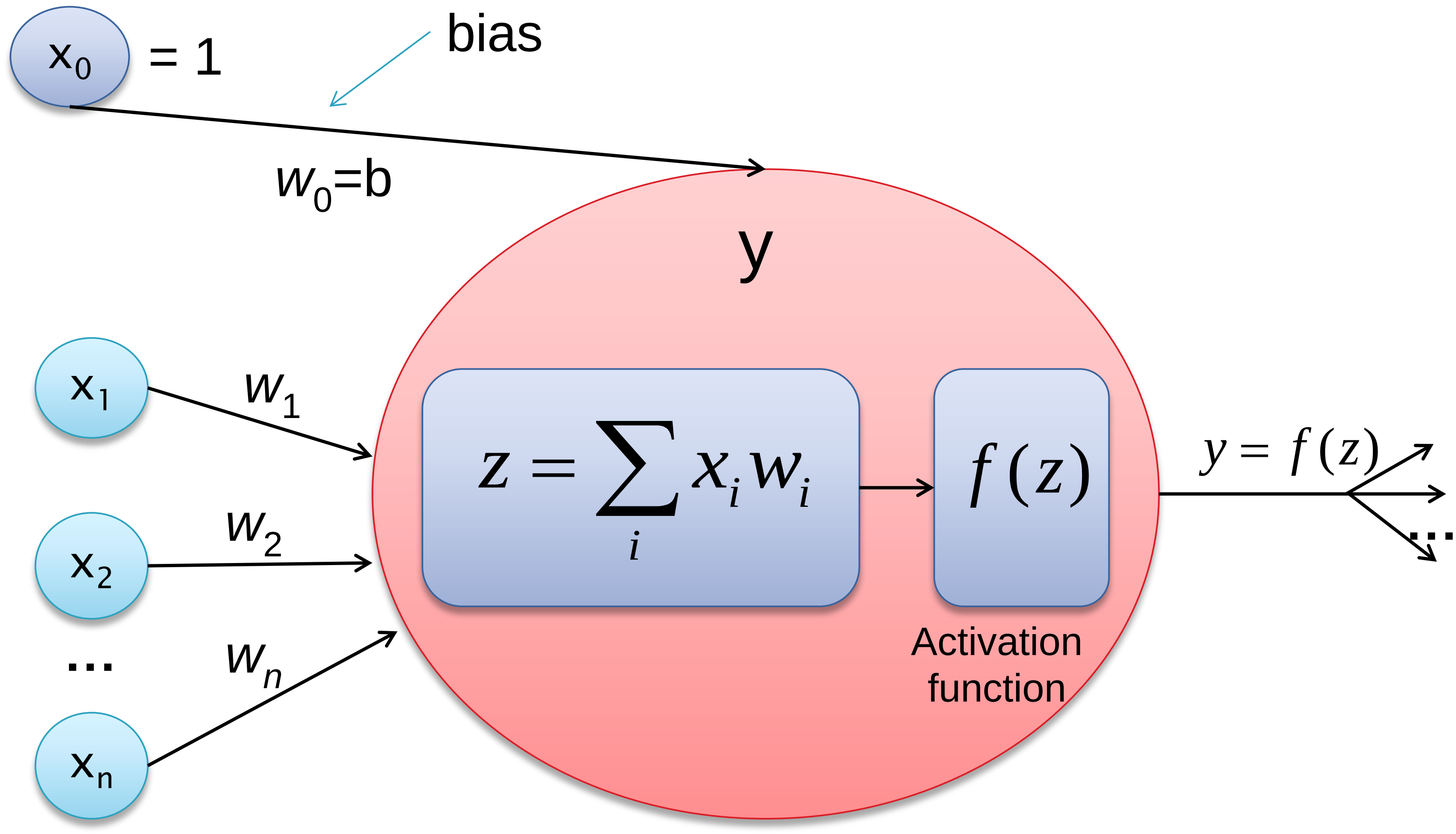
$$\approx$$

$$\mathbf{Y}_i = \{y_{i,j} : j = 1, \ldots, N_O\}$$

# Neuron structure



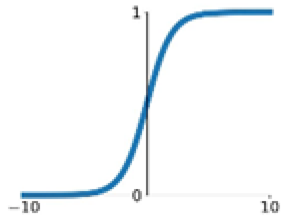



$$y = f(z), \quad z = \sum_{i=1}^{n} w_i x_i + b$$

# Commonly used activation functions

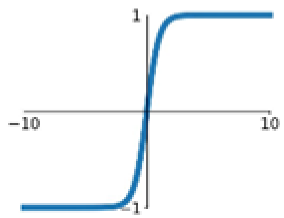
**Sigmoid**

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



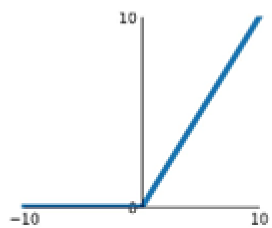

**tanh**

$\tanh(x)$



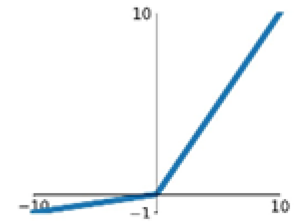

**ReLU**

$\max(0, x)$




**Leaky ReLU**

$\max(0.1x, x)$

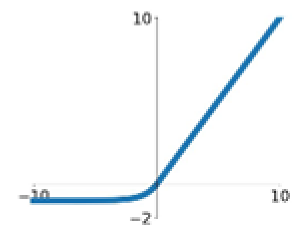



**Maxout**

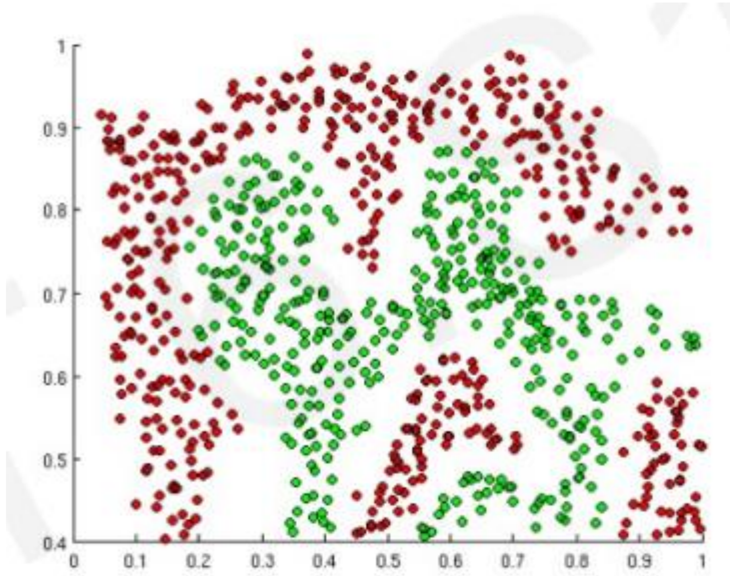
$\max(w_1^T x + b_1, w_2^T x + b_2)$

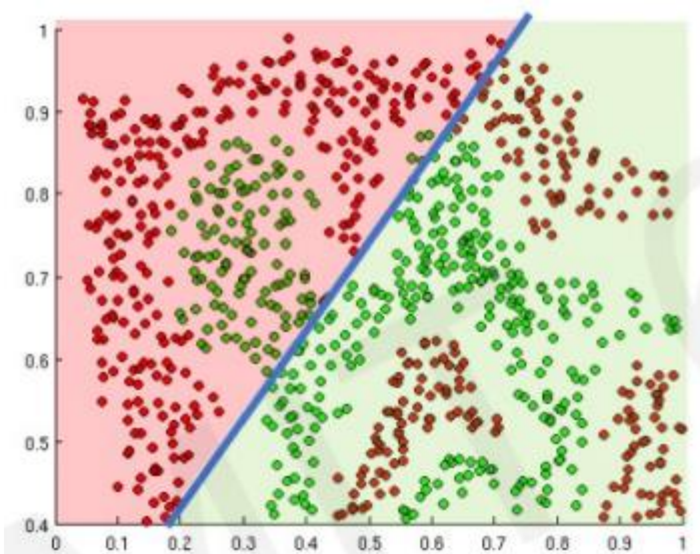
**ELU**

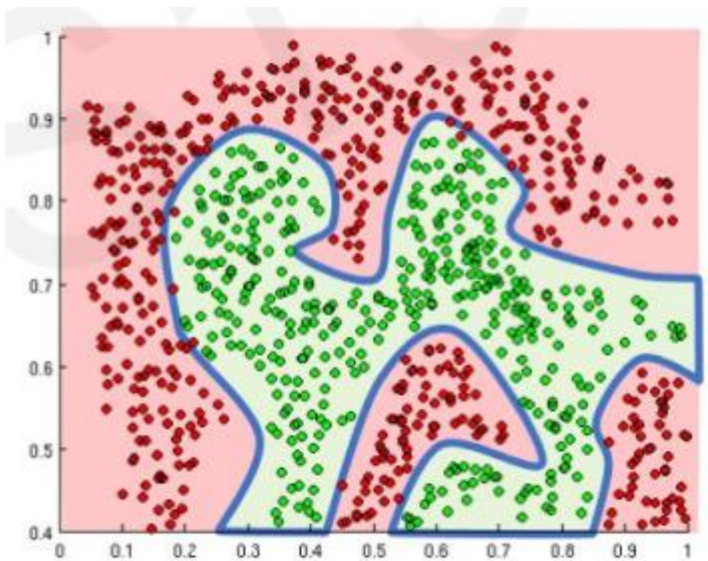
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$




http://cs231n.stanford.edu/slides/2021/lecture_7.pdf

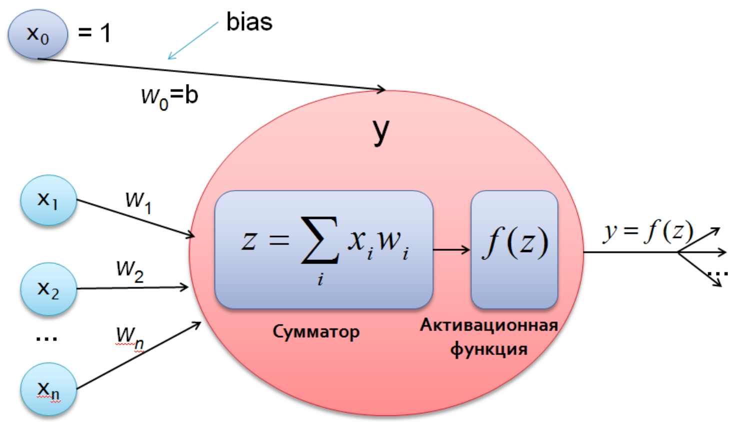






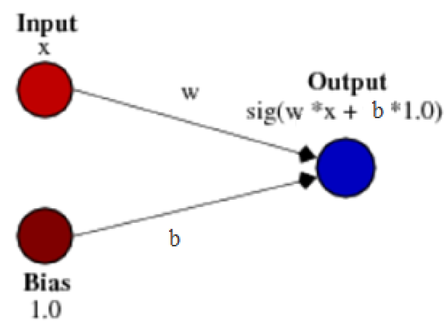
//introtodeeplearning.com/2021/slides/6S191_MIT_DeepLearning_L1.pdf

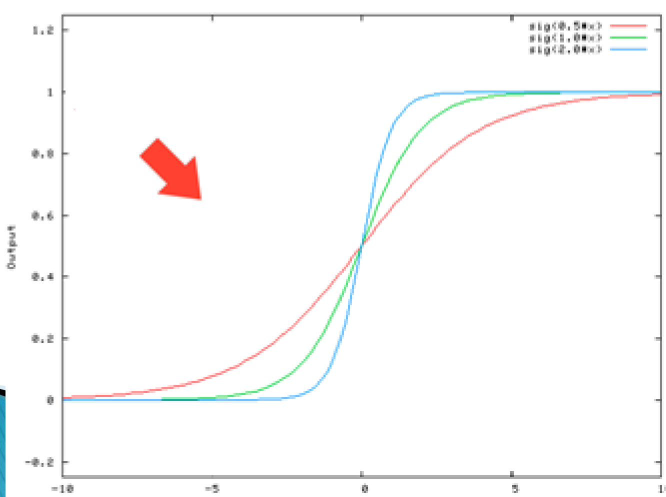
# Bias

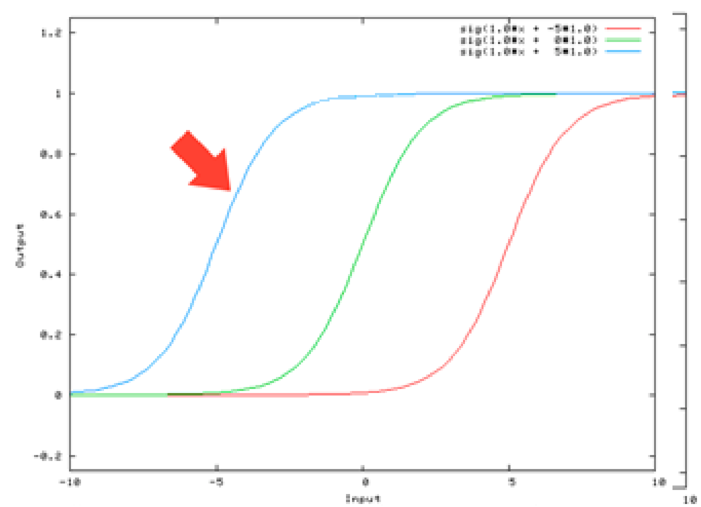


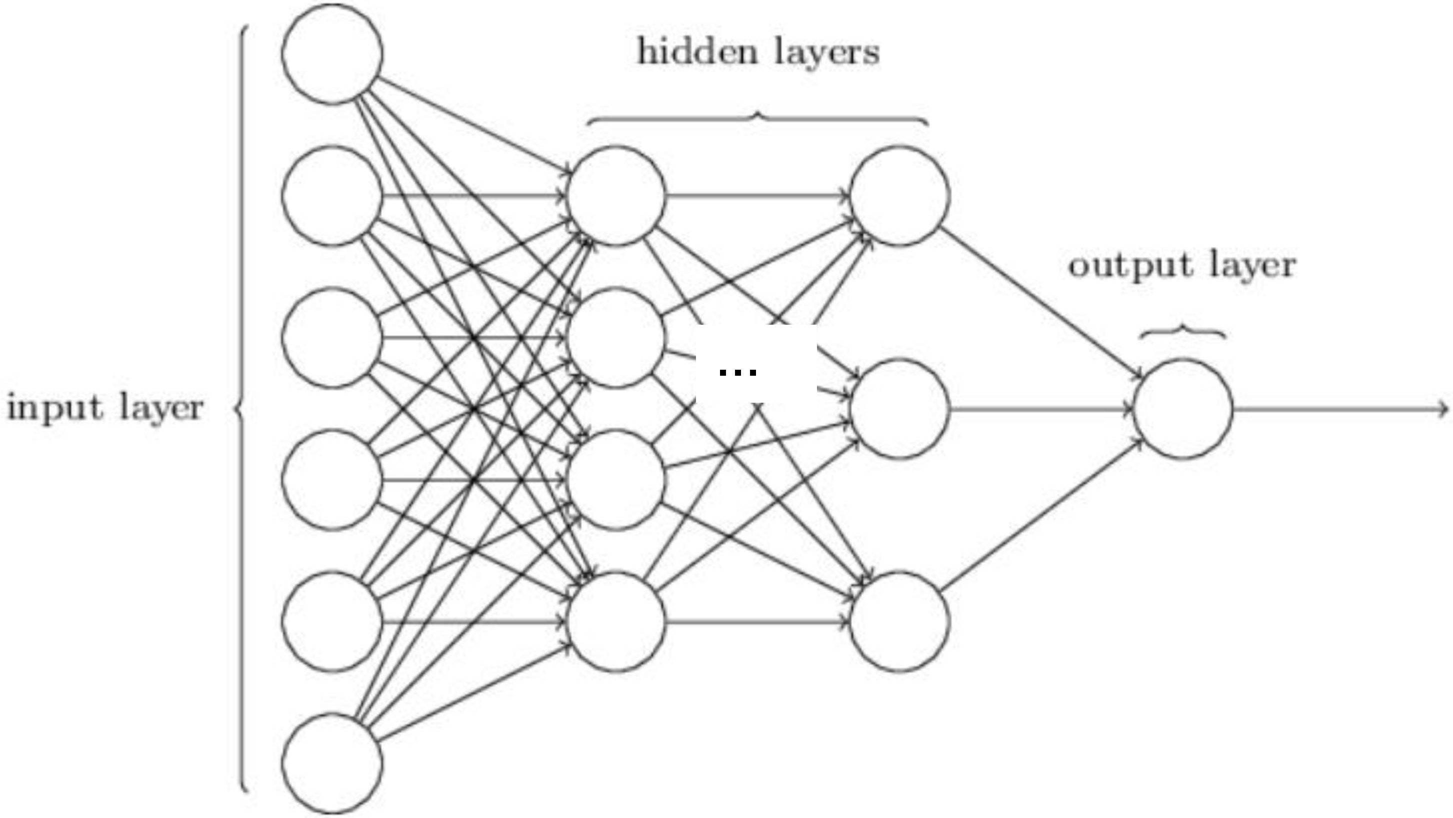

$$z = \sum_{i=1}^{n} w_i x_i + b$$

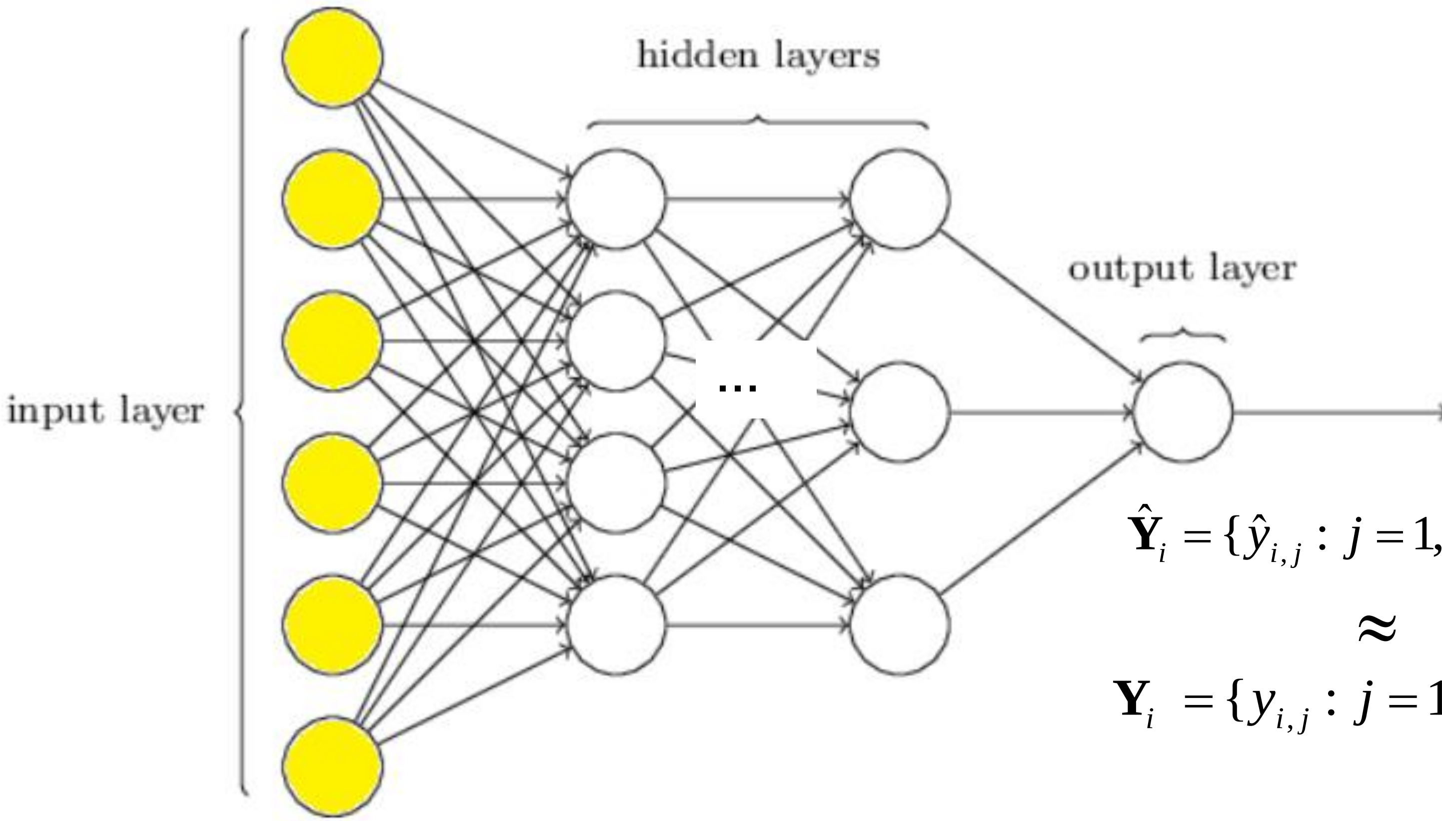
$$y = f(z) = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$


hidden layers
output layer
input layer
...




$$\hat{\mathbf{Y}}_i = \{\hat{y}_{i,j} : j = 1, \ldots, N_O\}$$

$$\approx$$

$$\mathbf{Y}_i = \{y_{i,j} : j = 1, \ldots, N_O\}$$

$$\mathbf{X}_i = \{x_{i,j} : j = 1, \ldots, N_I\}$$

The neurons of the input layer don’t perform any calculations, simply accept the input data

# Example 1

$\mathbf{X}_i$

SepalLength
SepalWidth
PetalLength
PetalWidth

$x_{1,1}\ x_{1,2} \dots x_{1,4}$

........................

$x_{150,1}\ x_{150,2} \dots x_{150,4}$

$Y_i$

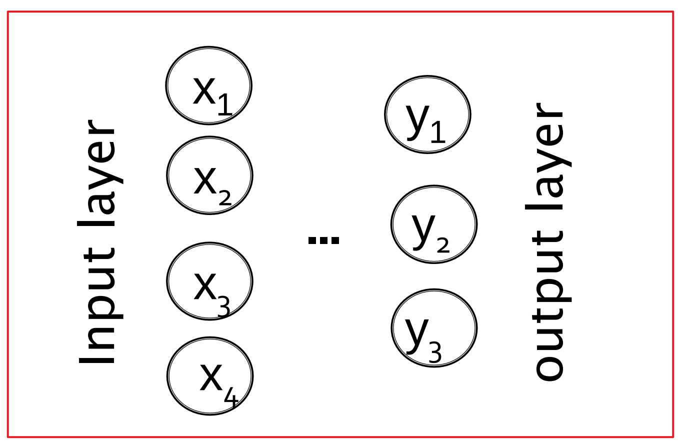
(*Iris setosa*) (*Iris versicolor*) (*Iris virginica*)

$y_{1,1}\ y_{1,2}\ y_{1,3}$

......................

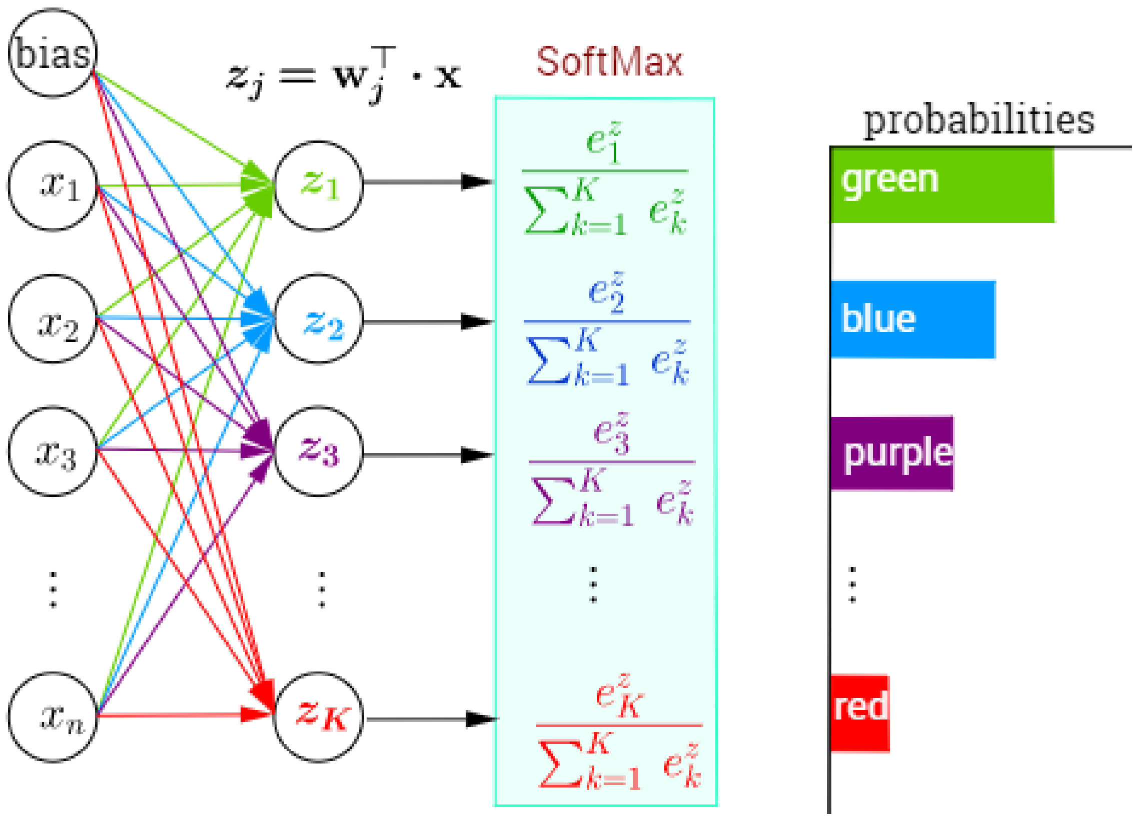
$y_{150,1}\ y_{150,2}\ y_{150,3}$

$$y_{i,j} = \begin{cases} 1, & x_i \in классу\ \ j \\ 0, & x_i \notin классу\ \ j \end{cases}$$



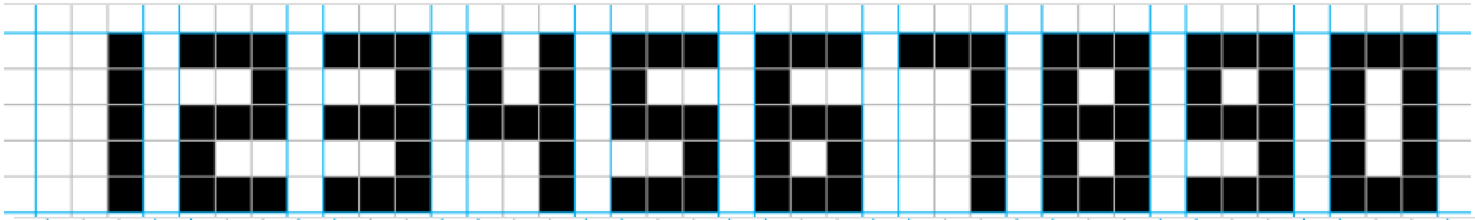
# SoftMax function

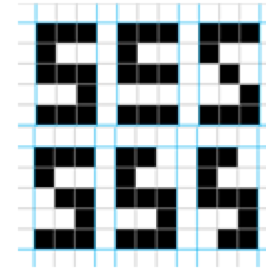
$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}$$




https://deepnotes.io/category/cnn-series

# Example 2





| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 0 1 | 1 1 1 | 1 1 1 | 1 0 1 | 1 1 1 | 1 1 1 | 1 1 1 | 1 1 1 | 1 1 1 | 1 1 1 |
| 0 0 1 | 0 0 1 | 0 0 1 | 1 0 1 | 1 0 0 | 1 0 0 | 0 0 1 | 1 0 1 | 1 0 1 | 1 0 1 |
| 0 0 1 | 1 1 1 | 1 1 1 | 1 1 1 | 1 1 1 | 1 1 1 | 0 0 1 | 1 1 1 | 1 1 1 | 1 0 1 |
| 0 0 1 | 1 0 0 | 0 0 1 | 0 0 1 | 0 0 1 | 1 0 1 | 0 0 1 | 1 0 1 | 0 0 1 | 1 0 1 |
| 0 0 1 | 1 1 1 | 1 1 1 | 0 0 1 | 1 1 1 | 1 1 1 | 0 0 1 | 1 1 1 | 1 1 1 | 1 1 1 |

$\mathbf{X}_i$ $Y_i$

1 – 001001001001001

.......................................

9 – 111101111001111

0 – 111101101101111

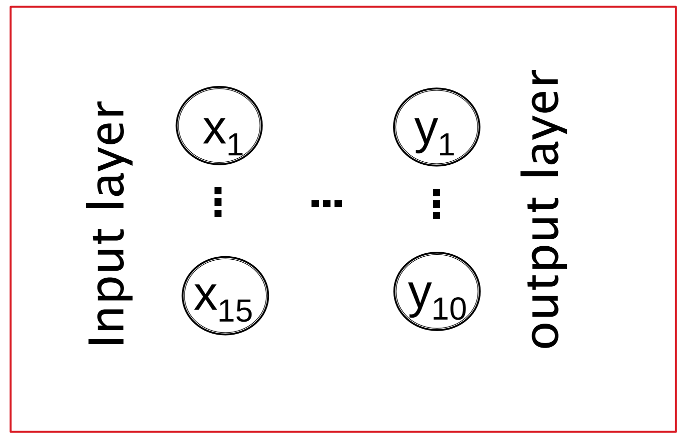
1000000000

0100000000

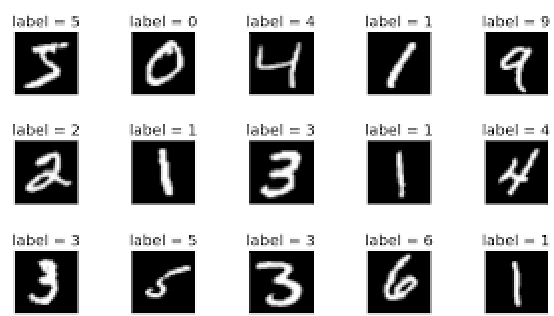
0010000000

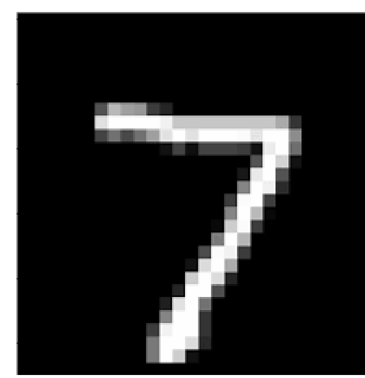
..................

0000000001



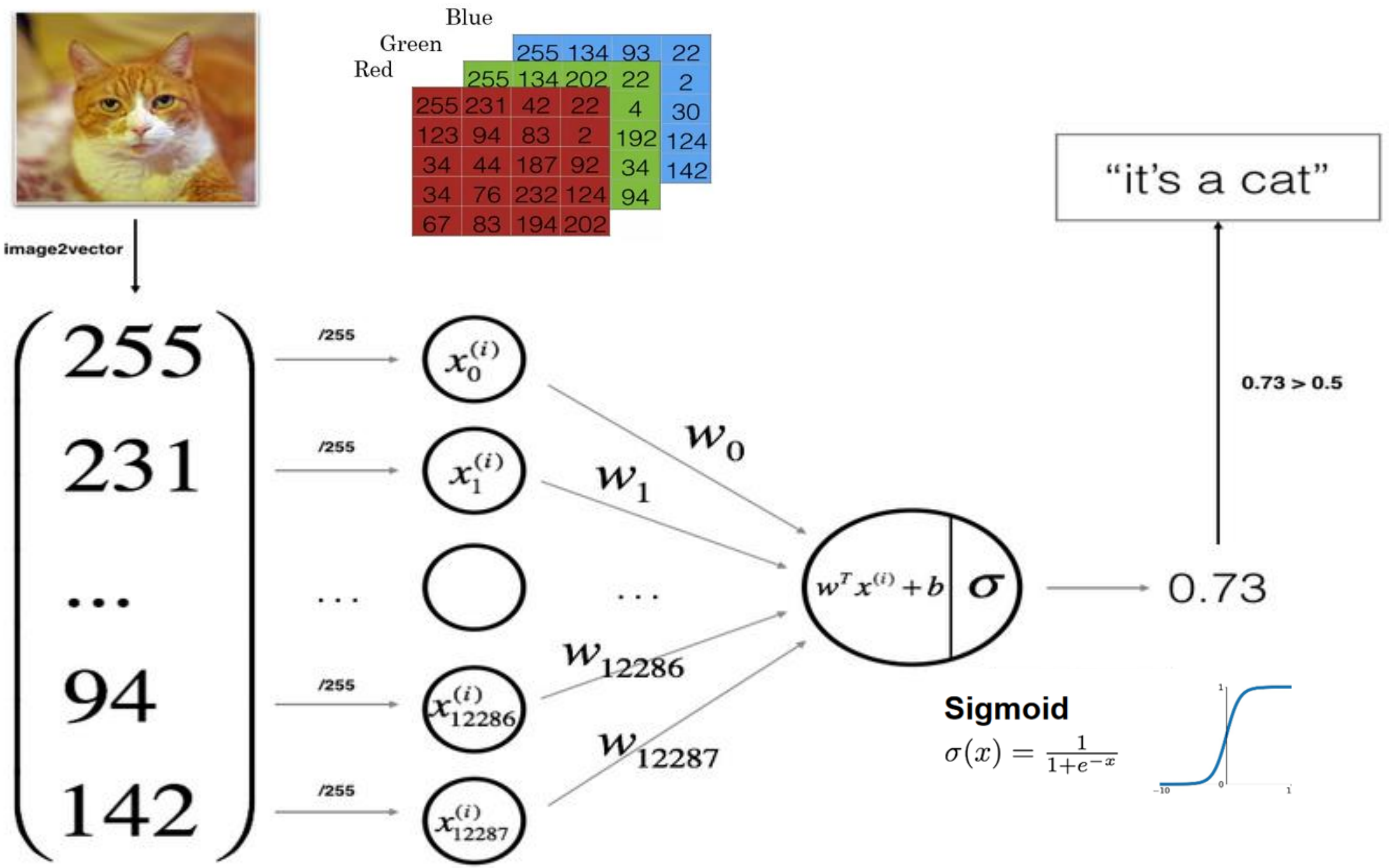

**MNIST Dataset**



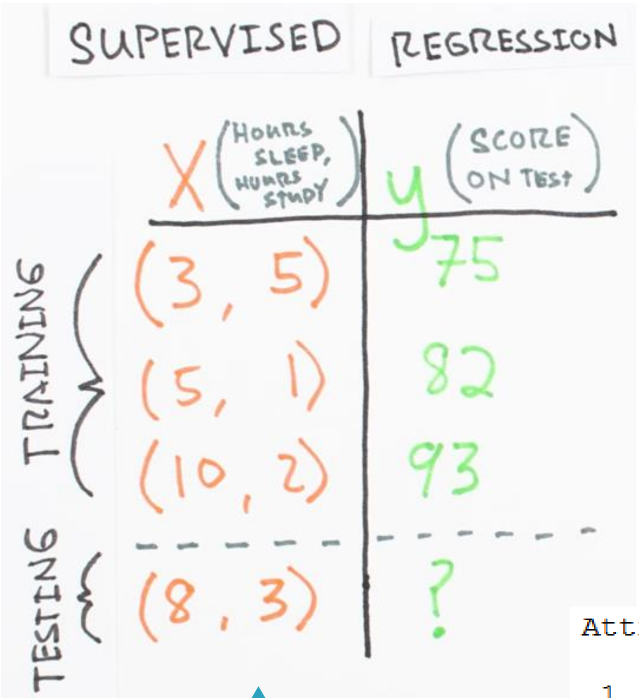
28x28 pixels

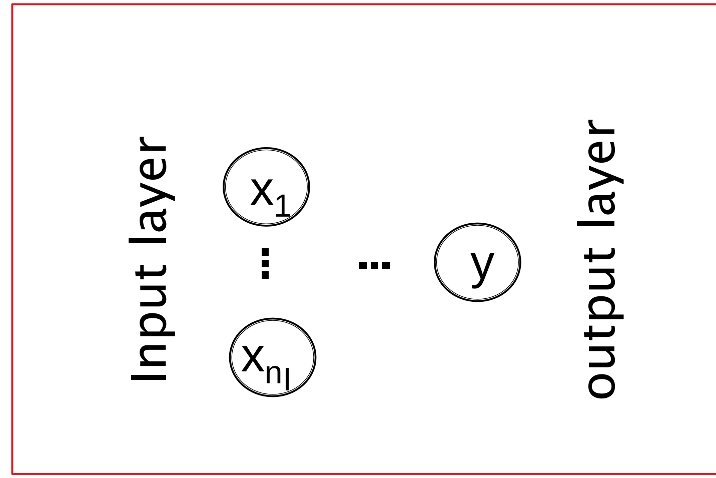
# Example 3




https://www.coursera.org/learn/neural-networks-deep-learning/lecture/praGm/why-is-deep-learning-taking-off

# Example 3. Regression task



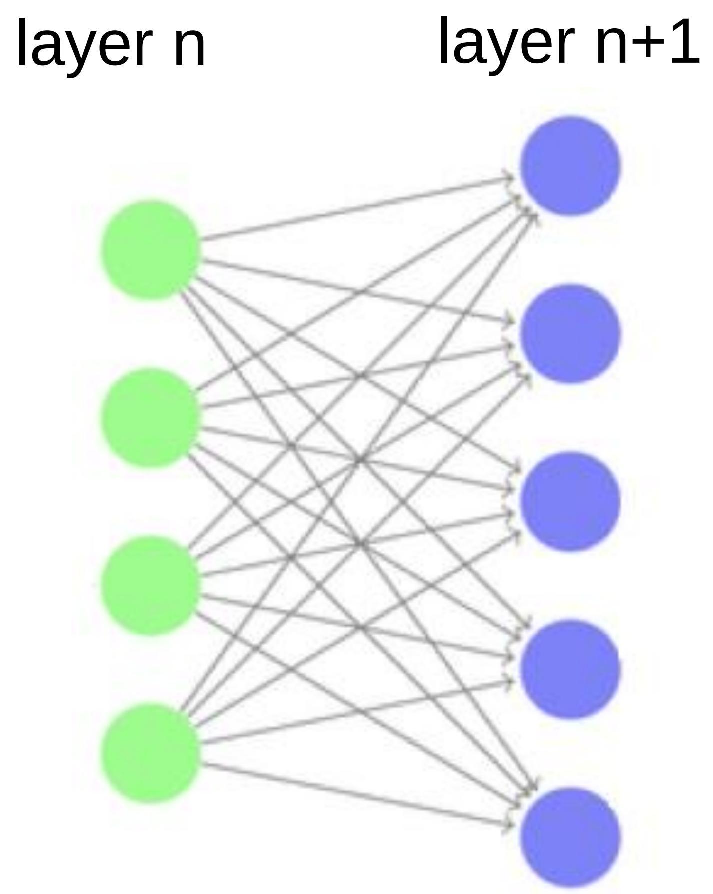



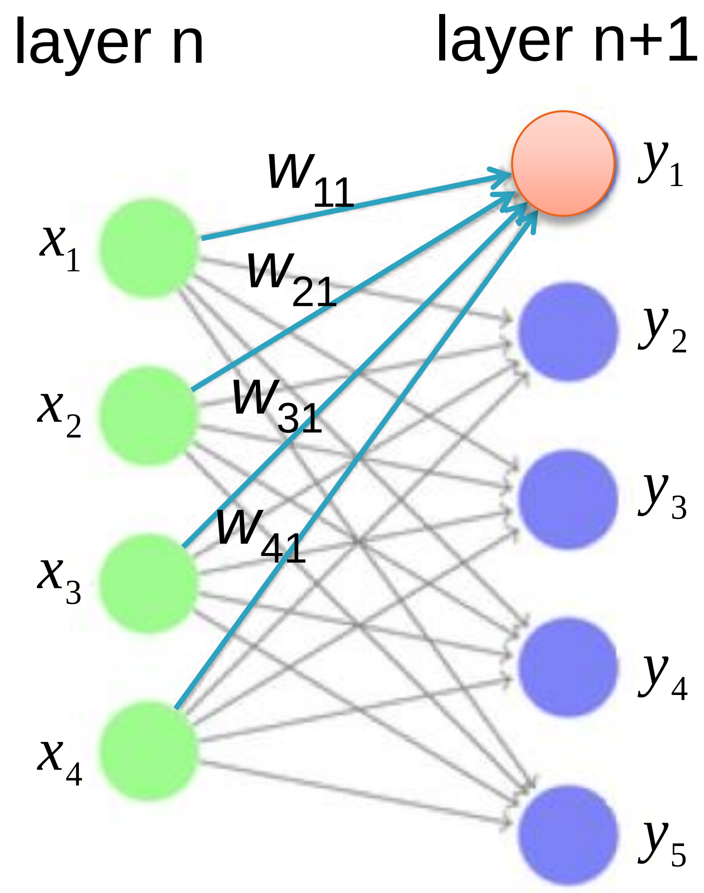

Task 1

Task 2

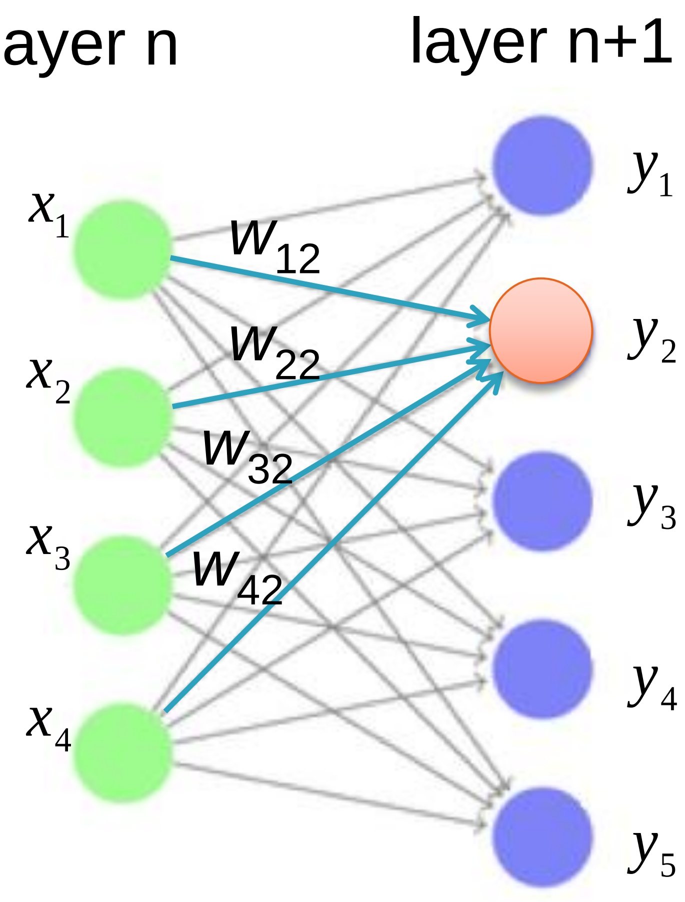
Attribute Information:

| | | |
|---|---|---|
| 1. | CRIM | per capita crime rate by town |
| 2. | ZN | proportion of residential land zoned for lots over 25,000 sq.ft. |
| 3. | INDUS | proportion of non-retail business acres per town |
| 4. | CHAS | Charles River dummy variable (= 1 if tract bounds river; 0 otherwise) |
| 5. | NOX | nitric oxides concentration (parts per 10 million) |
| 6. | RM | average number of rooms per dwelling |
| 7. | AGE | proportion of owner-occupied units built prior to 1940 |
| 8. | DIS | weighted distances to five Boston employment centres |
| 9. | RAD | index of accessibility to radial highways |
| 10. | TAX | full-value property-tax rate per $10,000 |
| 11. | PTRATIO | pupil-teacher ratio by town |
| 12. | B | 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town |
| 13. | LSTAT | % lower status of the population |
| 14. | MEDV | Median value of owner-occupied homes in $1000's |

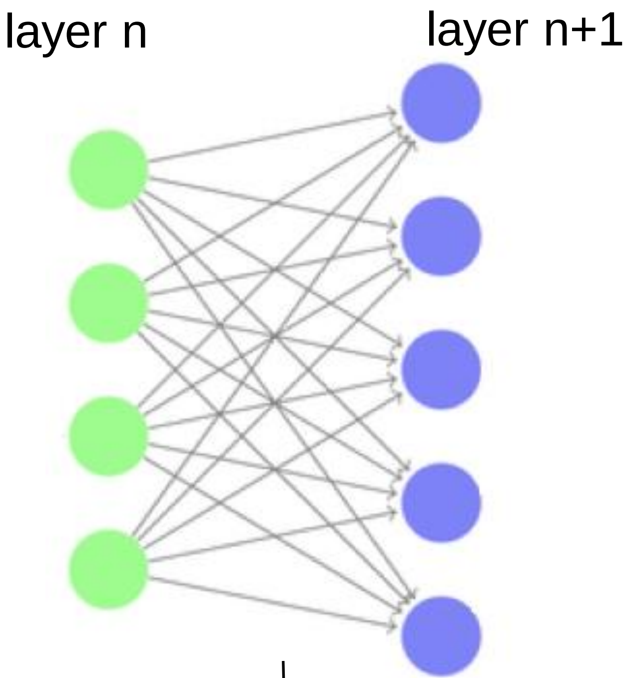

layer n
layer n+1




$$y_1 = f(x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41} + b)$$




$$y_2 = f(x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42} + b)$$




$$
W = \begin{array}{cc}
N_{\text{layer n+1}} & \\
\begin{array}{ccccc}
w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \\
w_{21} & w_{22} & w_{23} & w_{24} & w_{25} \\
w_{31} & w_{32} & w_{33} & w_{34} & w_{35} \\
w_{41} & w_{42} & w_{43} & w_{44} & w_{45}
\end{array} & N_{\text{layer n}}
\end{array}
$$

$$X = \{x_1\ x_2\ x_3\ x_4\}$$

$$Y = X \cdot W$$

$$W = \begin{matrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} \\ w_{41} & w_{42} & w_{43} & w_{44} & w_{45} \end{matrix}$$

$$y_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41}$$

$$y_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42}$$

$$y_3 = x_1 w_{13} + x_2 w_{23} + x_3 w_{33} + x_4 w_{43}$$

$$y_4 = x_1 w_{14} + x_2 w_{24} + x_3 w_{34} + x_4 w_{44}$$

$$y_5 = x_1 w_{15} + x_2 w_{25} + x_3 w_{35} + x_4 w_{45}$$

$$Y = \{y_1\ y_2\ y_3\ y_4\ y_5\}$$

```
a = np.random.rand(1000000)
b = np.random.rand(1000000)

tic = time.time()
c = np.dot(a,b)
toc = time.time()

print(c)
print("Vectorized version:" + str(1000*(toc-tic)) +"ms")

c = 0
tic = time.time()
for i in range(1000000):
    c += a[i]*b[i]
toc = time.time()

print(c)
print("For loop:" + str(1000*(toc-tic)) + "ms")
```

```
249946.964024
Vectorized version:1.505136489868164ms
249946.964024
For loop:481.3110828399658ms
```

!

# Bias trick

$$X = \{x_1\ x_2\ x_3\ x_4\ 1\}$$

$$Y = X \cdot W$$

$$W = \begin{matrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} \\ w_{41} & w_{42} & w_{43} & w_{44} & w_{45} \\ b_1 & b_2 & b_3 & b_4 & b_5 \end{matrix}$$

$$y_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41} + b_1$$

$$y_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42} + b_2$$

$$y_3 = x_1 w_{13} + x_2 w_{23} + x_3 w_{33} + x_4 w_{43} + b_3$$

$$y_4 = x_1 w_{14} + x_2 w_{24} + x_3 w_{34} + x_4 w_{44} + b_4$$

$$y_5 = x_1 w_{15} + x_2 w_{25} + x_3 w_{35} + x_4 w_{45} + b_5$$

$$Y = \{y_1\ y_2\ y_3\ y_4\ y_5\}$$

$$X = \begin{matrix} x_{11} \; x_{12} \; x_{13} \; x_{14} \\ x_{21} \; x_{22} \; x_{23} \; x_{24} \\ \ldots \\ x_{N1} \; x_{N2} \; x_{N3} \; x_{N4} \end{matrix}$$

$$Y = \begin{matrix} y_{11} \; y_{12} \; y_{13} \; y_{14} \; y_{15} \\ y_{21} \; y_{22} \; y_{23} \; y_{24} \; y_{25} \\ \ldots \\ y_{N1} \; y_{N2} \; y_{N3} \; y_{N4} \; y_{N5} \end{matrix}$$

$$W = \begin{matrix} w_{11} \; w_{12} \; w_{13} \; w_{14} \; w_{15} \\ w_{21} \; w_{22} \; w_{23} \; w_{24} \; w_{25} \\ w_{31} \; w_{32} \; w_{33} \; w_{34} \; w_{35} \\ w_{41} \; w_{42} \; w_{43} \; w_{44} \; w_{45} \end{matrix}$$

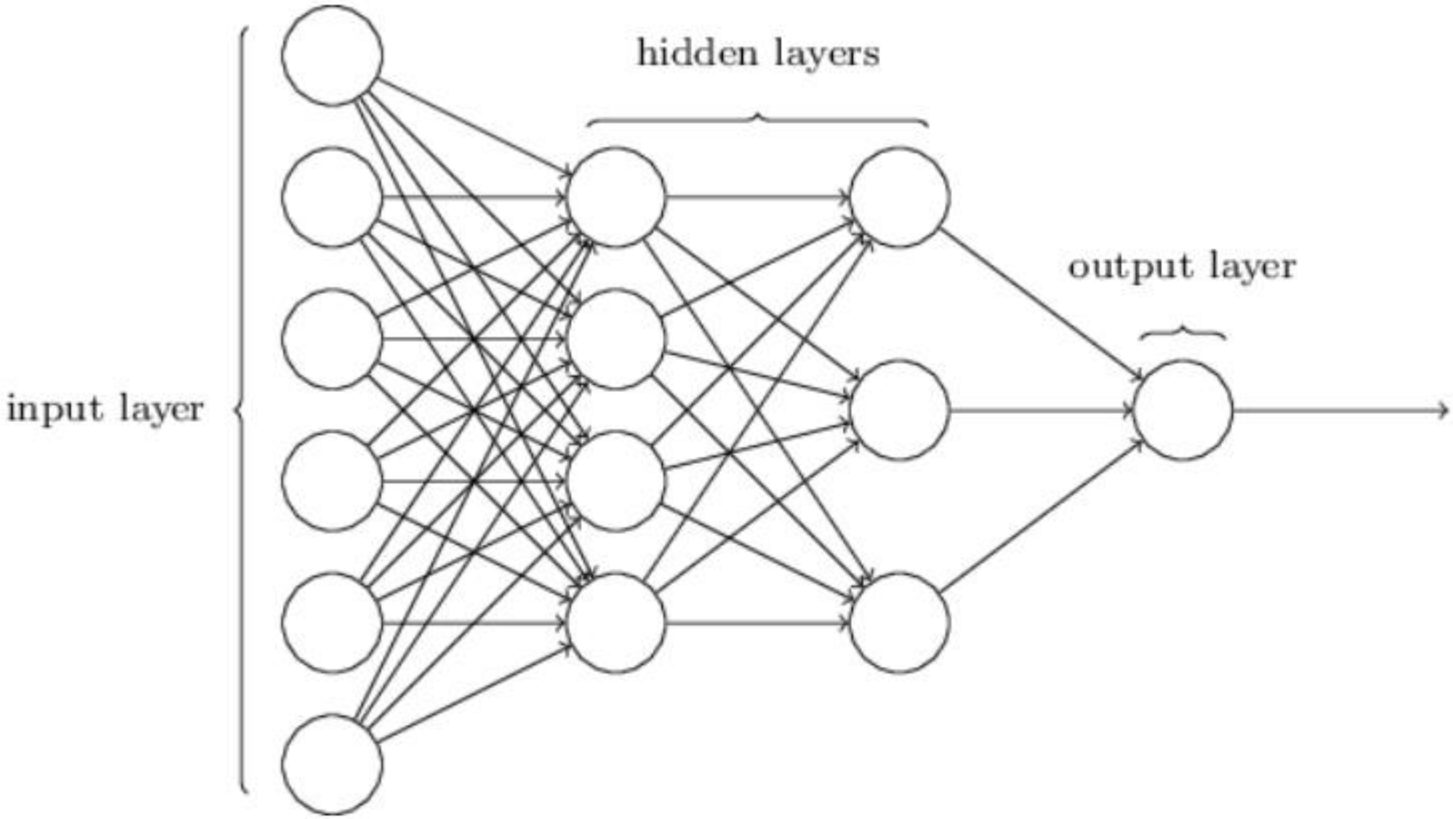
$$Y = X \cdot W$$

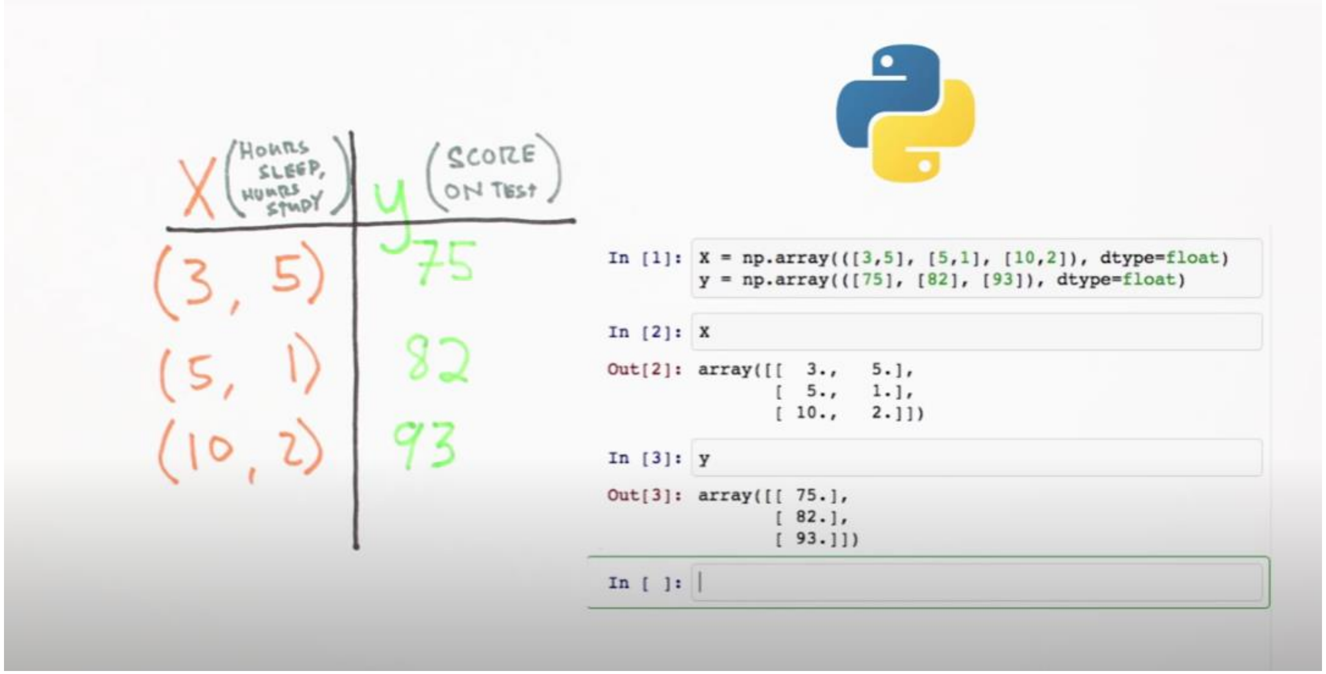
$$X = \begin{matrix} x_{11}\ x_{12}\ x_{13}\ x_{1\mathrm{N}_{\text{layer n}}} \\ x_{21}\ x_{22}\ x_{23}\ x_{2\mathrm{N}_{\text{layer n}}} \\ \ldots \\ x_{N1}\ x_{N2}\ x_{N3}\ x_{N\mathrm{N}_{\text{layer n}}} \end{matrix}$$

$$Y = \begin{matrix} y_{11}\ y_{12}\ y_{13}\ y_{14}\ y_{1\mathrm{N}_{\text{layer n+1}}} \\ y_{21}\ y_{22}\ y_{23}\ y_{24}\ y_{2\mathrm{N}_{\text{layer n+1}}} \\ \ldots \\ y_{N1}\ y_{N2}\ y_{N3}\ y_{N4}\ y_{N\mathrm{N}_{\text{layer n+1}}} \end{matrix}$$

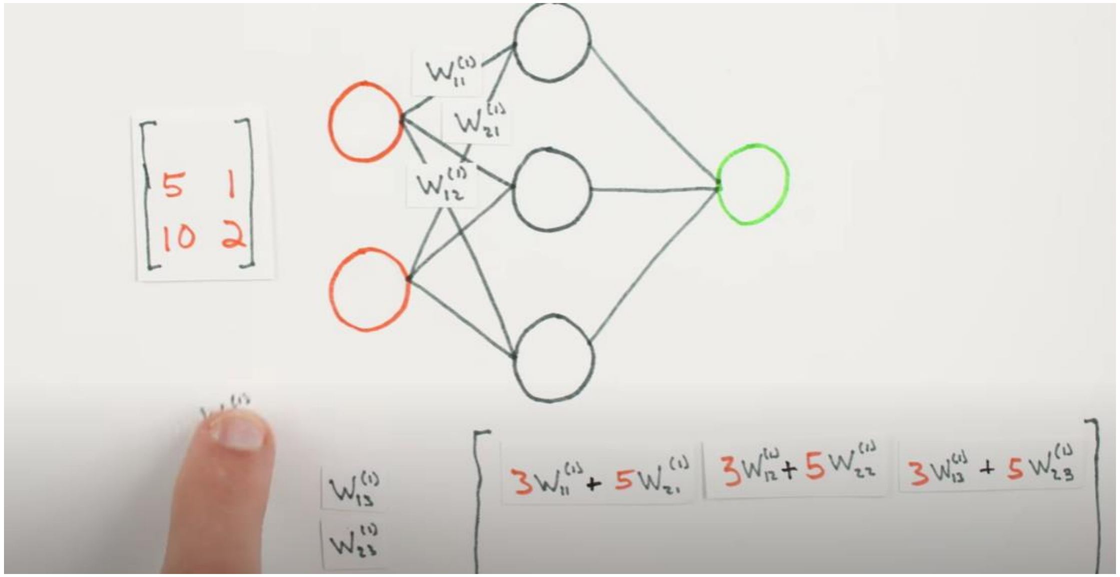
$$W = \begin{matrix} w_{11}\ w_{12}\ \ldots w_{1\mathrm{N}_{\text{layer n+1}}} \\ w_{21}\ w_{22}\ \ldots w_{2\mathrm{N}_{\text{layer n+1}}} \\ \ldots \\ w_{\mathrm{N}_{\text{layer n}}1}\ w_{\mathrm{N}_{\text{layer n}}2}\ \ldots w_{\mathrm{N}_{\text{layer n}}\mathrm{N}_{\text{layer n+1}}} \end{matrix}$$

$$\underset{\left[N \times N_{\text{layer n+1}}\right]}{Y} = \overset{\left[N \times N_{\text{layer n}}\right]}{X} \cdot \underset{\left[N_{\text{layer n}} \times N_{\text{layer n+1}}\right]}{W}$$


hidden layers
output layer
input layer




https://www.youtube.com/watch?v=bxe2T-V8XRs




https://www.youtube.com/watch?v=UJwK6jAStmg

# Neural Networks with TensorFlow

# Module: tf.keras.layers

See Stable See Nightly

Keras layers API.

## Modules

`experimental` module: Public API for tf.keras.layers.experimental namespace.

## Classes

`class AbstractRNNCell`: Abstract object representing an RNN cell.

`class Activation`: Applies an activation function to an output.

`class ActivityRegularization`: Layer that applies an update to the cost function based input activity.

`class Add`: Layer that adds a list of inputs.



https://www.tensorflow.org/api_docs/python/tf/keras/layers tensorflow.org/api_docs/python/tf/keras/layers/Dense

TensorFlow Install Learn API Resources Community Why TensorFlow Search

TensorFlow Core v2.6.0

Overview Python C++ Java

initializers
layers
Overview
AbstractRNNCell
Activation
ActivityRegularization
Add
AdditiveAttention
AlphaDropout
Attention
Average
AveragePooling1D
AveragePooling2D
AveragePooling3D
BatchNormalization
Bidirectional
CategoryEncoding
CenterCrop
Concatenate
Conv1D
Conv1DTranspose
Conv2D
Conv2DTranspose
Conv3D
Conv3DTranspose
ConvLSTM1D
ConvLSTM2D
ConvLSTM3D
Cropping1D
Cropping2D
Cropping3D
Dense
DenseFeatures

TensorFlow > API > TensorFlow Core v2.6.0 > Python

Was this helpful?

# tf.keras.layers.Dense

TensorFlow 1 version | View source on GitHub

Just your regular densely-connected NN layer.

Inherits From: `Layer`, `Module`

View aliases

```
tf.keras.layers.Dense(
    units, activation=None, use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros', kernel_regularizer=None,
    bias_regularizer=None, activity_regularizer=None, kernel_constraint=None,
    bias_constraint=None, **kwargs
)
```

## Used in the notebooks

| Used in the guide | Used in the tutorials |
|---|---|
| • Save and load Keras models<br>• The Sequential model | • Overfit and underfit<br>• Time series forecasting |

Table of contents
Used in the notebooks
Args

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

```
tf.keras.layers.Dense(
    units, activation=None, use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros', kernel_regularizer=None,
    bias_regularizer=None, activity_regularizer=None, kernel_constraint=None,
    bias_constraint=None, **kwargs
)
```

# Вбудовані активаційні функції

## Module: tf.keras.activations

TensorFlow 1 version

Public API for tf.keras.activations namespace.

### Functions

`deserialize(...)` : Returns activation function given a string identifier.

`elu(...)` : Exponential Linear Unit.

`exponential(...)` : Exponential activation function.

`gelu(...)` : Applies the Gaussian error linear unit (GELU) activation function.

`get(...)` : Returns function.

`hard_sigmoid(...)` : Hard sigmoid activation function.

`linear(...)` : Linear activation function (pass-through).

`relu(...)` : Applies the rectified linear unit activation function.

`selu(...)` : Scaled Exponential Linear Unit (SELU).

`serialize(...)` : Returns the string identifier of an activation function.

`sigmoid(...)` : Sigmoid activation function, `sigmoid(x) = 1 / (1 + exp(-x))`.

`softmax(...)` : Softmax converts a vector of values to a probability distribution.

`softplus(...)` : Softplus activation function, `softplus(x) = log(exp(x) + 1)`.

`softsign(...)` : Softsign activation function, `softsign(x) = x / (abs(x) + 1)`.

`swish(...)` : Swish activation function, `swish(x) = x * sigmoid(x)`.

`tanh(...)` : Hyperbolic tangent activation function.

https://www.tensorflow.org/api_docs/python/tf/keras/activations

# Model

Sequential API

Functional API

# Sequential model

```python
tf.keras.Sequential(
    layers=None, name=None
)
```

```python
import tensorflow as tf
```

```python
model = tf.keras.Sequential([
  tf.keras.layers.Dense(10, activation='relu'),
  tf.keras.layers.Dense(10, activation='relu'),
  tf.keras.layers.Dense(3)
])
```

```python
model = keras.Sequential()
model.add(layers.Dense(2, activation="relu"))
model.add(layers.Dense(3, activation="relu"))
model.add(layers.Dense(4))
```

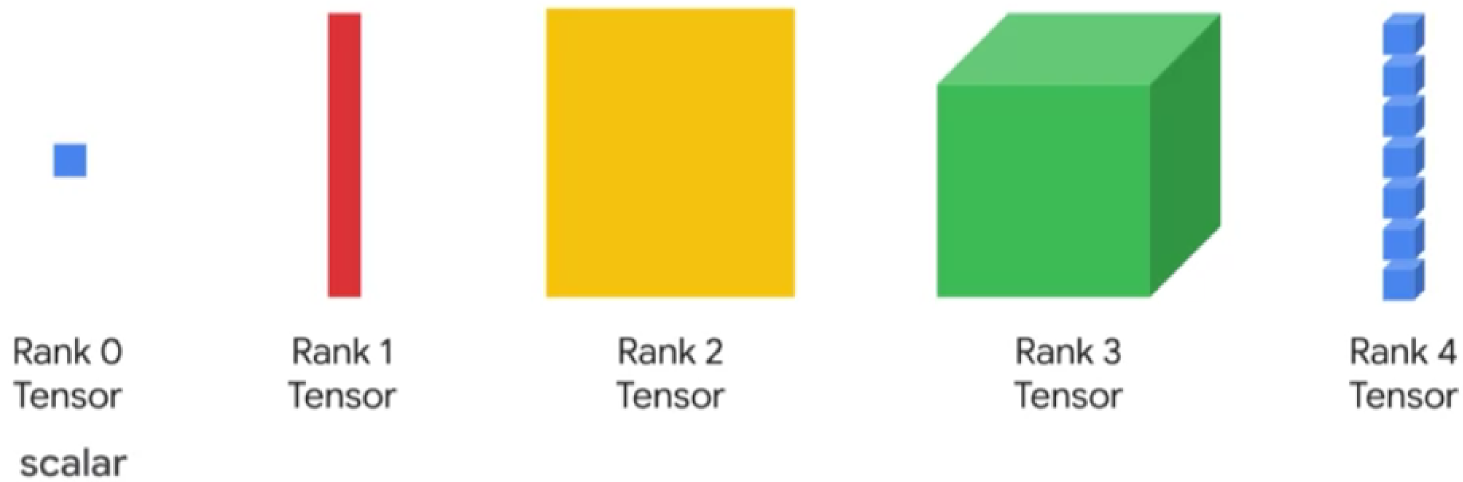
```python
from tensorflow import keras
from tensorflow.keras import layers
```

https://www.tensorflow.org/guide/keras/sequential_model
https://www.tensorflow.org/api_docs/python/tf/keras/Sequential

```python
model = keras.Sequential(
    [
        layers.Dense(2, activation="relu", name="layer1"),
        layers.Dense(3, activation="relu", name="layer2"),
        layers.Dense(4, name="layer3"),
    ]
)
# Call model on a test input
x = tf.ones((3, 3))
y = model(x)

print(x)
print(y)
```

```
tf.Tensor(
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]], shape=(3, 3), dtype=float32)
tf.Tensor(
[[-0.51288277  0.4053292   0.34165433 -0.2714369 ]
 [-0.51288277  0.4053292   0.34165433 -0.2714369 ]
 [-0.51288277  0.4053292   0.34165433 -0.2714369 ]], shape=(3, 4), dtype=float32)
```

# A tensor is an N-dimensional array of data




https://w ursera.org/learn/intro-tensorflow/lecture/Spag9/introduction-to-tensorflow

```python
model = tf.keras.Sequential([
  tf.keras.layers.Dense(10, activation='relu'),
  tf.keras.layers.Dense(10, activation='relu'),
  tf.keras.layers.Dense(3)
])
```

```python
model = keras.Sequential(
    [
        keras.Input(shape=(4,)),
        layers.Dense(2, activation="relu"),
        layers.Dense(3, activation="relu"),
        layers.Dense(3),
    ]
)
```

```python
model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10)
])
```

```python
model = keras.Sequential(
    [
        layers.Dense(5, activation="relu", name="l1"),
        layers.Dense(3, activation="relu", name="l2"),
        layers.Dense(1, name="y"),
    ]
)
model.build((None, 4))
```

https://www.tensorflow.org/guide/keras/sequential_model
https://www.tensorflow.org/api_docs/python/tf/keras/Sequential

# model.summary

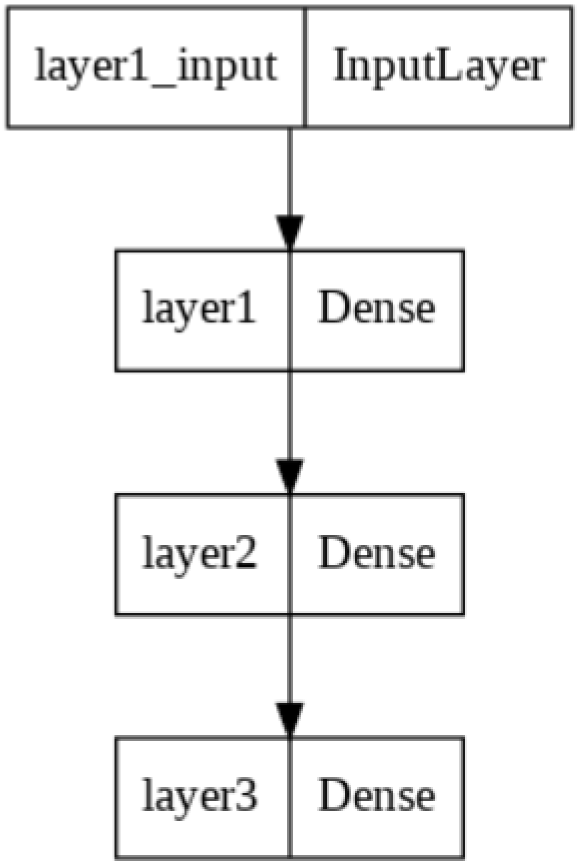
```
model = keras.Sequential(
    [
        layers.Dense(2, activation="relu", name="l1"),
        layers.Dense(3, activation="relu"),
        layers.Dense(3),
    ]
)

# Call the model on a test input
x = tf.ones((1, 4))
y = model(x)

model.summary()
```

```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 l1 (Dense)                  (1, 2)                    10

 dense_35 (Dense)            (1, 3)                    9

 dense_36 (Dense)            (1, 3)                    12

=================================================================
Total params: 31
Trainable params: 31
Non-trainable params: 0
```

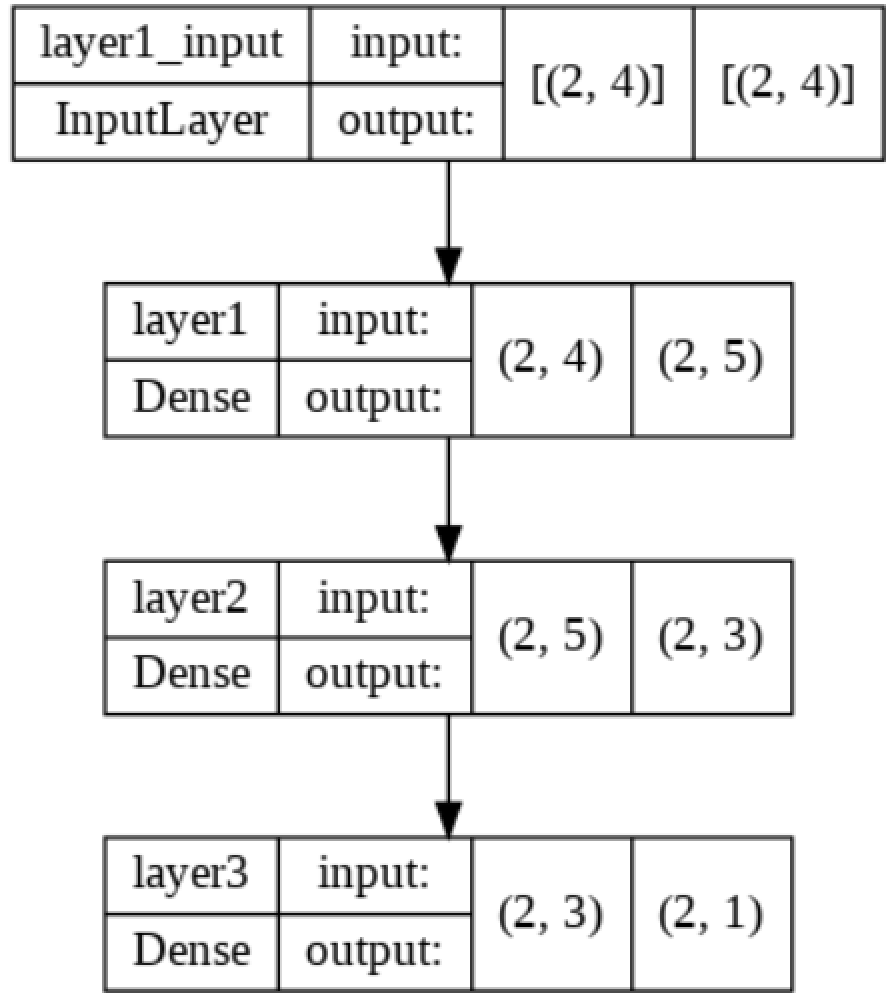
```python
my_model = keras.Sequential(
    [
        layers.Dense(5, activation="relu", name="l1"),
        layers.Dense(3, activation="relu", name="l2"),
        layers.Dense(1, name="y"),
    ],
    name="my_model"
)

# Call the model on a test input
x = tf.ones((2, 4))
y = model(x)


model.summary()
```

```
Model: "my_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 l1 (Dense)                  (2, 5)                    25

 l2 (Dense)                  (2, 3)                    18

 y (Dense)                   (2, 1)                    4

=================================================================
Total params: 47
Trainable params: 47
Non-trainable params: 0
```

```
keras.utils.plot_model(model)
```




```
keras.utils.plot_model(model, show_shapes=True)
```



# get_layer

```
my_model = keras.Sequential(
    [
        keras.Input(shape=(4,)),
        layers.Dense(2, activation="relu", name="l1"),
        layers.Dense(3, activation="relu", name="l2"),
        layers.Dense(3, name="y"),
    ]
)

feature_extractor = keras.Model(
    inputs=my_model.inputs,
    outputs=my_model.get_layer(name="y").output,
)
# Call feature extractor on test input.
x = tf.ones((1, 4))
features = feature_extractor(x)

print(features)
```

```
tf.Tensor([[0. 0. 0.]], shape=(1, 3), dtype=float32)
```

```python
my_model = keras.Sequential(
    [
        keras.Input(shape=(4,)),
        layers.Dense(2, activation="relu", name="l1"),
        layers.Dense(3, activation="relu", name="l2"),
        layers.Dense(3, name="y"),
    ]
)

feature_extractor = keras.Model(
    inputs=my_model.inputs,
    outputs=[layer.output for layer in my_model.layers],
)
# Call feature extractor on test input.
x = tf.ones((1, 4))
features = feature_extractor(x)

print(features)
```

```
[<tf.Tensor: shape=(1, 2), dtype=float32, numpy=array([[1.808532  , 0.15643644]], dtype=float32)>,
 <tf.Tensor: shape=(1, 3), dtype=float32, numpy=array([[0.       , 1.659713, 0.       ]], dtype=float32)>,
 <tf.Tensor: shape=(1, 3), dtype=float32, numpy=array([[ 0.30876577,  0.63318014, -1.5857009 ]], dtype=float32)>
```

# Розробка власної активаційної функції

```
tf.keras.layers.Lambda(lambda x: tf.abs(x))
```

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

```
if(x>0):
    return x
else:
    return 0
```

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128),
  tf.keras.layers.Lambda(lambda x: tf.abs(x)),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

```
def my_relu(x):
  return K.maximum(0.0, x)

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128),
  tf.keras.layers.Lambda(my_relu),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

https://www.coursera.org/learn/custom-models-layers-loss-functions-with-tensorflow/home/info