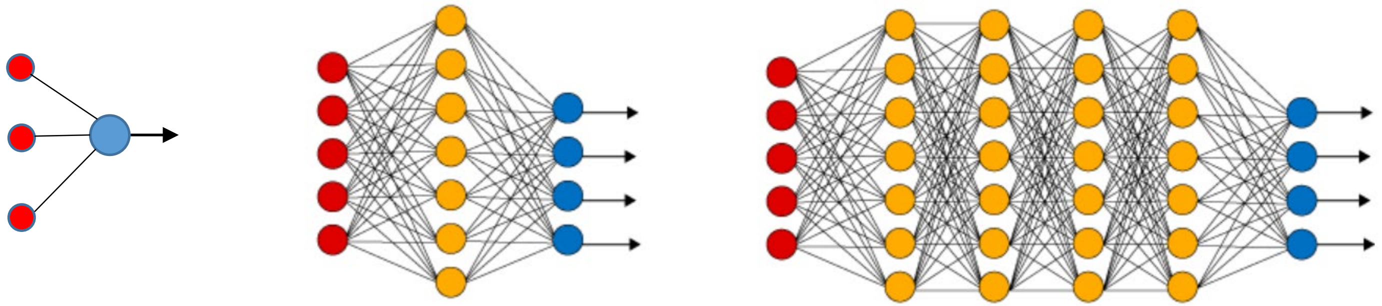
Intro to ML/DL, Neural Networks (NN), and Keras

CJ Chung

Keras



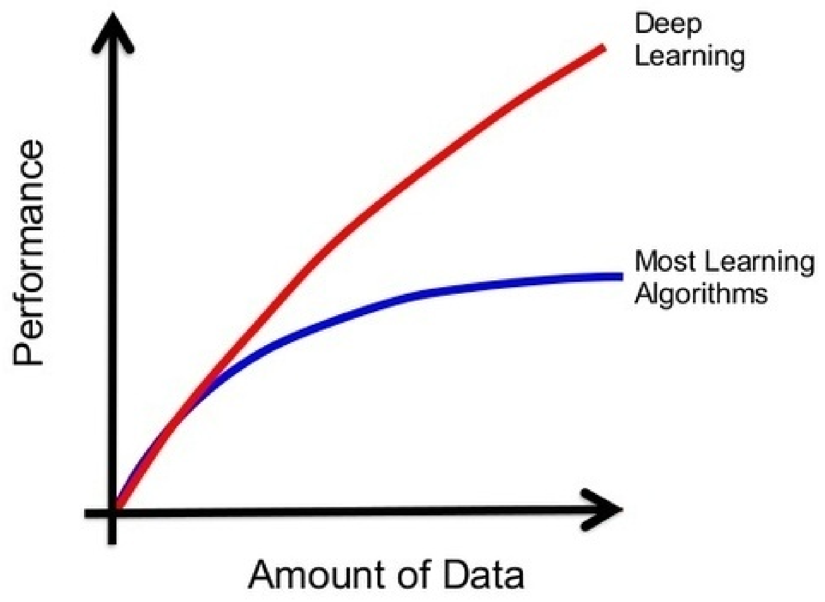
Lawrence Technological University Computer Science

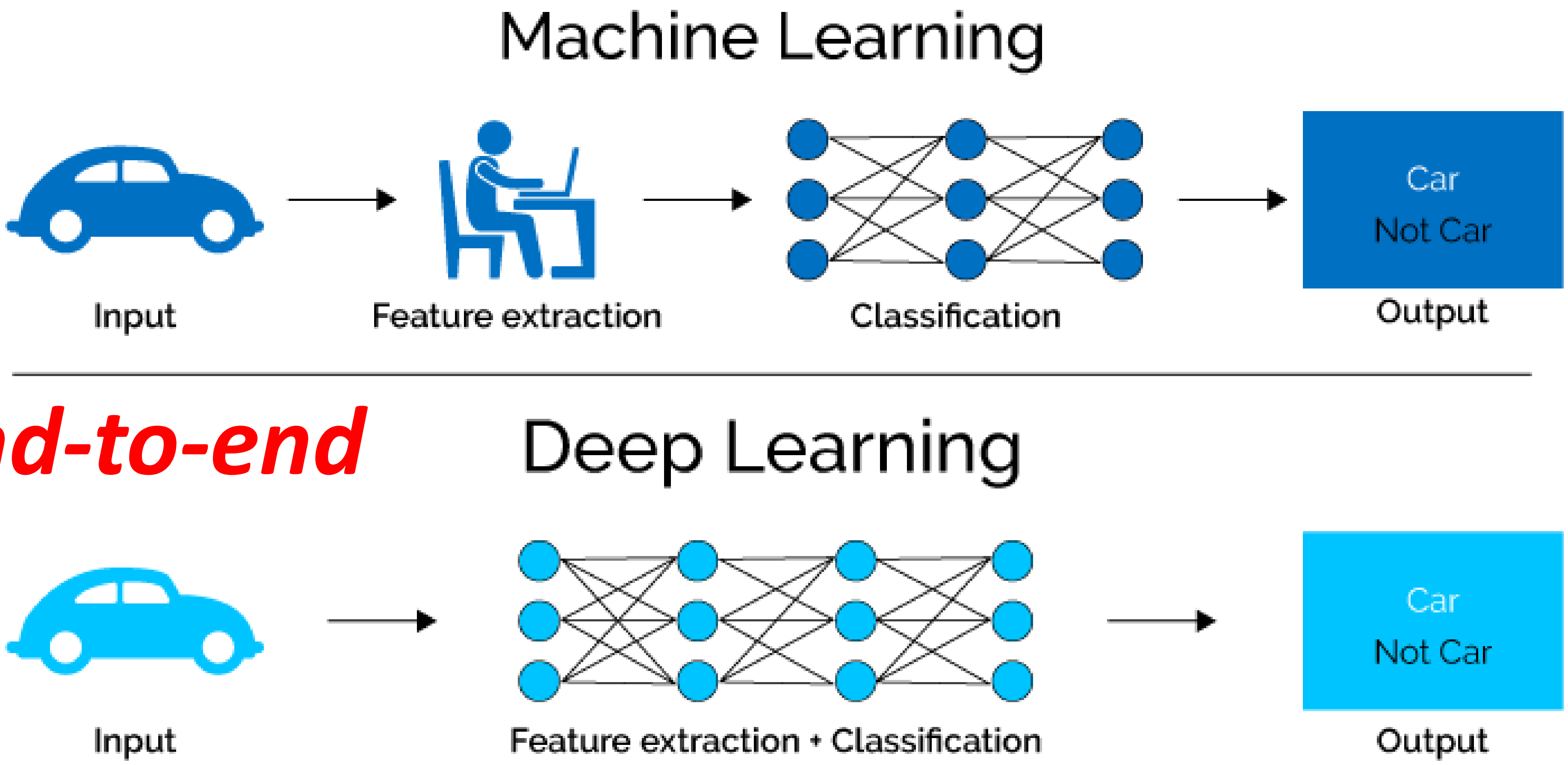
History of ML/DL algorithms

John Hopfield
Geoffrey Hinton
2024 Nobel Prize Winners

- Probabilistic algorithms using Bayes' theorem
- Early Neural Networks (Perceptrons, Hopfield Nets)
- SVM (Support Vector Machine)
- Random forests and Gradient boosting machines based on Decision Trees
- Deep NN Learning (Convolutional NN) became powerful & popular since 2011
- Google released TensorFlow, free & open source library for DL, November 9, 2015
- ChatGPT by OpenAI. Released on Nov 30, 2022

Power of DL


Performance
Deep Learning
Most Learning Algorithms
Amount of Data



End-to-end

As is without pre-processing

Lawrence Technological University Computer Science

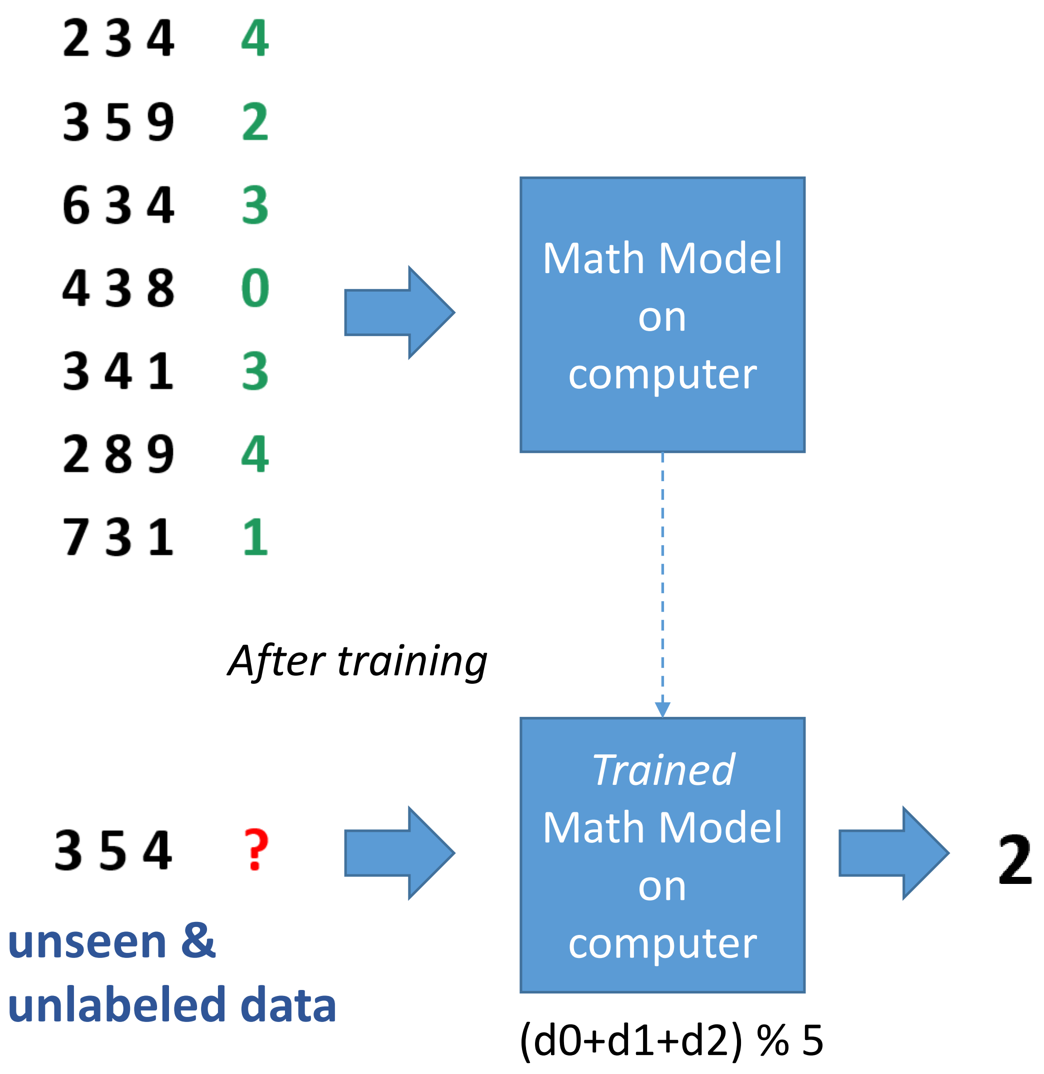
Learning Methods for ML/DL

ML/DL

- **Supervised learning**
  - Develop predictive model based on example input data and answers
  - **Classification**: output variable is categorical (or discrete).
  - **Regression**: output variable is numerical (or continuous)
- **Reinforcement Learning**
  - Policy optimization, control, path finding
- **Un-supervised learning**
  - Group & interpret data based on input data with **no answers** (labels).
  - Clustering, association, …

Basic Supervised ML/DL Terminology

- A **feature** (data) is an input variable
- A **label** (answer) is what are predicting
- A **model** (set of rules) defines the relationship between features and label
- **Training** means creating the model (lets the machine learn.)
- **Inference** means applying the trained model to unlabeled data

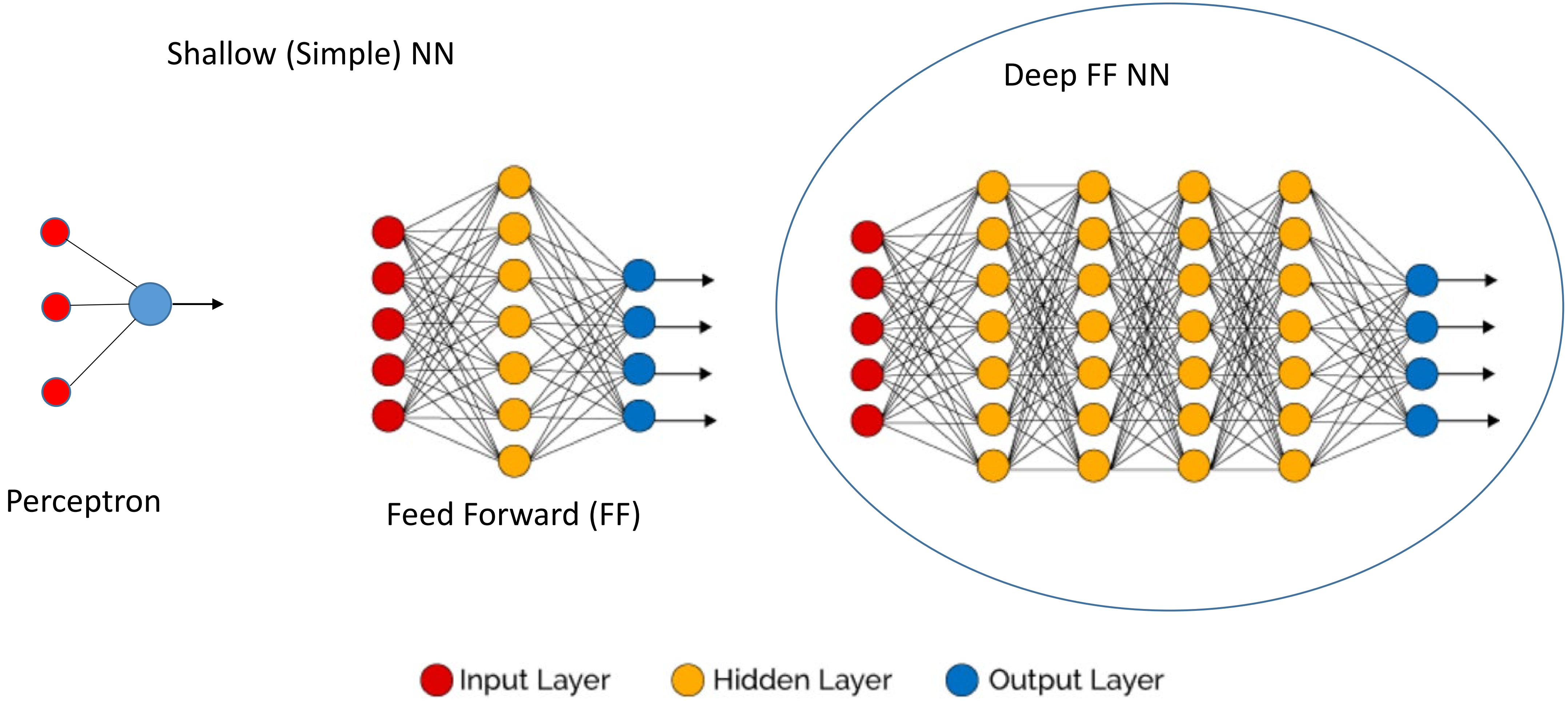


Lawrence Technological University Computer Science

Major Types of Data for DS, ML/DL (excluding images, audio, …)

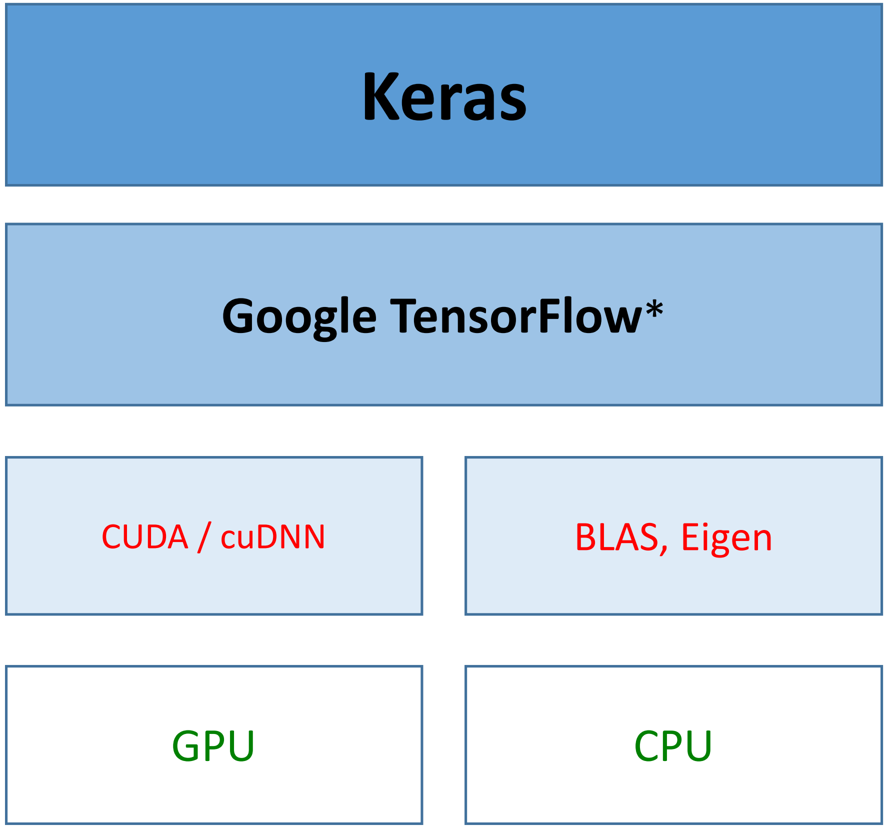
- **Qualitative**: in narrative form, usually from open survey
- **Quantitative** - Numerical
  - (integer based) Discrete
    - 5 kids
    - 34 workers
    - 3 purchases in 2022
  - Continuous (can take any value between two numbers)
    - 3.25kg
    - 17.576534 miles
- **Categorical**: numbers can be used, but no mathematical meaning
  - Gender
  - US State
- **Ordinal**: a mixture of numerical and categorical. Mathematical meaning
  - Movie ratings. 1 is worse than 2.

Type of Brain-inspired NNs



Tools for DL

- Anaconda Python 3
- Jupyter notebook
- TensorFlow*
- Keras: model-level *interface* library providing high-level building blocks for developing DL models to use primary DL platforms such as TensorFlow



PyTorch
(for Yolo)

*This class will mainly use Keras with TensorFlow

Python, TensorFlow, Keras Versions in Colab

Sep. 2024

```
[1] !python --version
```

```
Python 3.10.12
```

```
[2] import tensorflow as tf
    print(tf.__version__)
```

```
2.17.0
```

```
import keras
print(keras.__version__)
```

```
3.4.1
```

Sep. 2025

```
[79] !python --version
```

```
Python 3.12.11
```

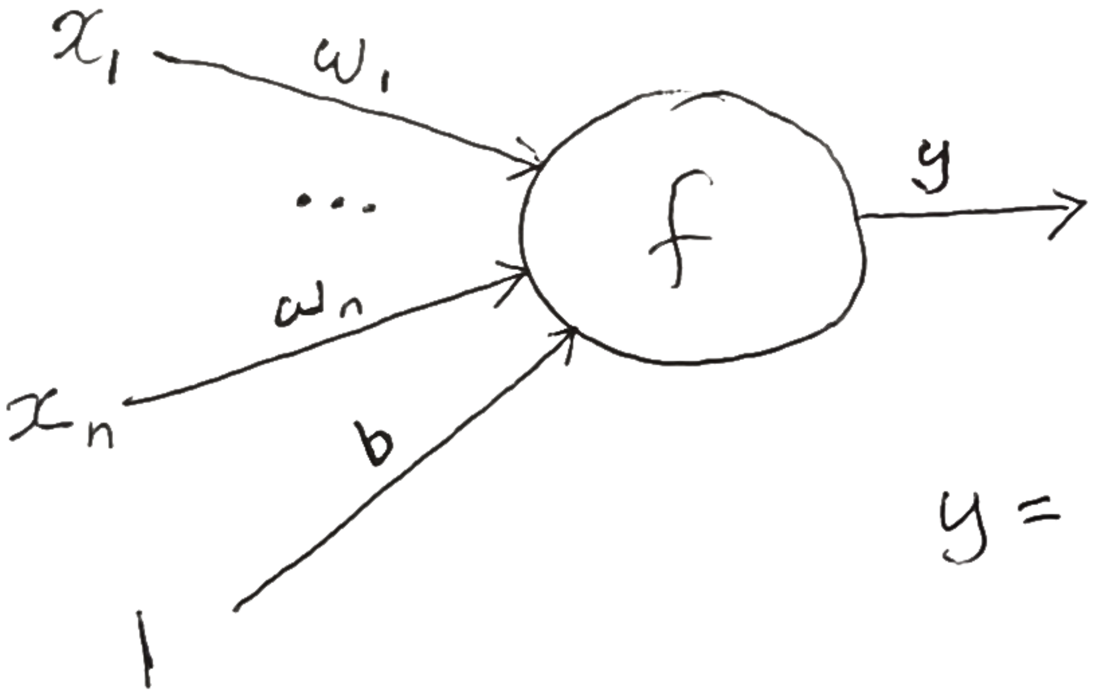
```
[81] import tensorflow as tf
     print(tf.__version__)
```

```
2.19.0
```

```
[82] import keras
     print(keras.__version__)
```

```
3.10.0
```

To calculate output y of a neuron (Perceptron) with activation function f and threshold bias b

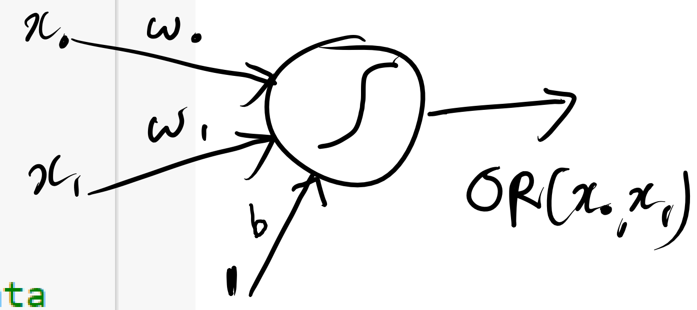


$$y = f\left(\sum_{i=1}^{n} x_i \cdot w_i + b\right)$$

Lawrence Technological University Computer Science

OR_function.ipynb

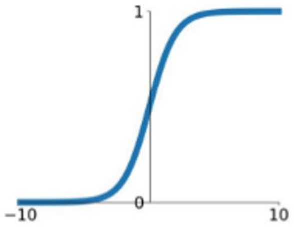
```python
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
import numpy as np

# training data. Bias "1" is not needed in Keras
X = np.array([[0,0], [0,1], [1,0], [1,1]]) #training data
y = np.array([[0],   [1],   [1],   [1]  ]) #target labels
X1 = np.array([[0.1,0], [0,0.9], [0.9,0], [1,0.9]]) # Unseen & unlabeled data
```




```python
model = Sequential([
    Dense(1, input_dim=2, activation="sigmoid")
])

model.compile(
    loss='mean_squared_error', # https://keras.io/losses/
    optimizer=optimizers.RMSprop(learning_rate=0.1), # https://keras.io/optimizers/
    metrics=["binary_accuracy"] # for binary classification problem
)

# batch_size: num of samples per weight update
# An epoch is an iteration over the entire X and y data provided
model.fit(X, y, batch_size=1, epochs=15)
```

Sigmoid

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



Lawrence Technological University Computer Science

```
Epoch 1/15
4/4 [==============================] - 0s 5ms/step - loss: 0.1626 - binary_accuracy: 0.7500
Epoch 2/15
4/4 [==============================] - 0s 4ms/step - loss: 0.1207 - binary_accuracy: 0.7500
Epoch 3/15
4/4 [==============================] - 0s 4ms/step - loss: 0.1085 - binary_accuracy: 0.7500
Epoch 4/15
4/4 [==============================] - 0s 3ms/step - loss: 0.0995 - binary_accuracy: 0.7500
Epoch 5/15
4/4 [==============================] - 0s 3ms/step - loss: 0.0912 - binary_accuracy: 0.7500
Epoch 6/15
4/4 [==============================] - 0s 4ms/step - loss: 0.0829 - binary_accuracy: 0.7500
Epoch 7/15
4/4 [==============================] - 0s 3ms/step - loss: 0.0754 - binary_accuracy: 1.0000
Epoch 8/15
4/4 [==============================] - 0s 4ms/step - loss: 0.0676 - binary_accuracy: 1.0000
Epoch 9/15
4/4 [==============================] - 0s 4ms/step - loss: 0.0611 - binary_accuracy: 1.0000
Epoch 10/15
4/4 [==============================] - 0s 4ms/step - loss: 0.0548 - binary_accuracy: 1.0000
```

Note: your run results will be different, due to the randomly generated initial weight values.

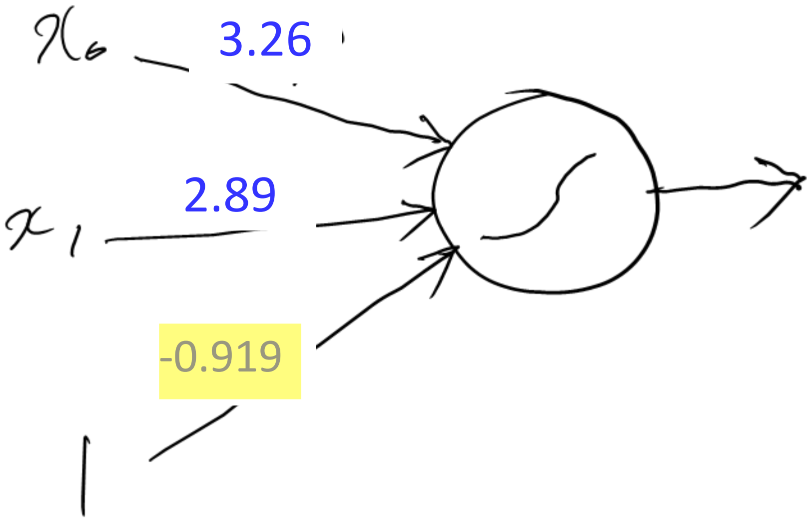
```
# print learned weight values
for layer in model.layers:
  print(layer.get_weights())
```

```
[array([[3.2605395],
       [2.8943603]], dtype=float32), array([-0.91959494], dtype=float32)]
```

Weight for the bias

```
# Test, Inference
print(model.predict(X))
print(model.predict(X1)) # Unseen dataset
print(model.predict(X1).round())
```

```
1/1 [=============
[[0.28504044]
 [0.878122  ]
 [0.9122118 ]
 [0.994703  ]]
1/1 [=============
[[0.3558228 ]
 [0.84360886]
 [0.88235164]
 [0.9929375 ]]
1/1 [=============
[[0.]
 [1.]
 [1.]
 [1.]]
```



```
Epoch 1/200
4/4 [==============================] - 0s 25ms/step - loss: 0.5235
Epoch 2/200
4/4 [==============================] - 0s 3ms/step - loss: 0.4857
Epoch 3/200
4/4 [==============================] - 0s 2ms/step - loss: 0.4425
Epoch 4/200
4/4 [==============================] - 0s 2ms/step - loss: 0.3954
Epoch 5/200
4/4 [==============================] - 0s 3ms/step - loss: 0.3475
Epoch 6/200
4/4 [==============================] - 0s 2ms/step - loss: 0.3054
Epoch 7/200
```

...

```
Epoch 197/200
4/4 [==============================] - 0s 2ms/step - loss: 0.0276
Epoch 198/200
4/4 [==============================] - 0s 2ms/step - loss: 0.0275
Epoch 199/200
4/4 [==============================] - 0s 1ms/step - loss: 0.0273
Epoch 200/200
4/4 [==============================] - 0s 1ms/step - loss: 0.0271
```

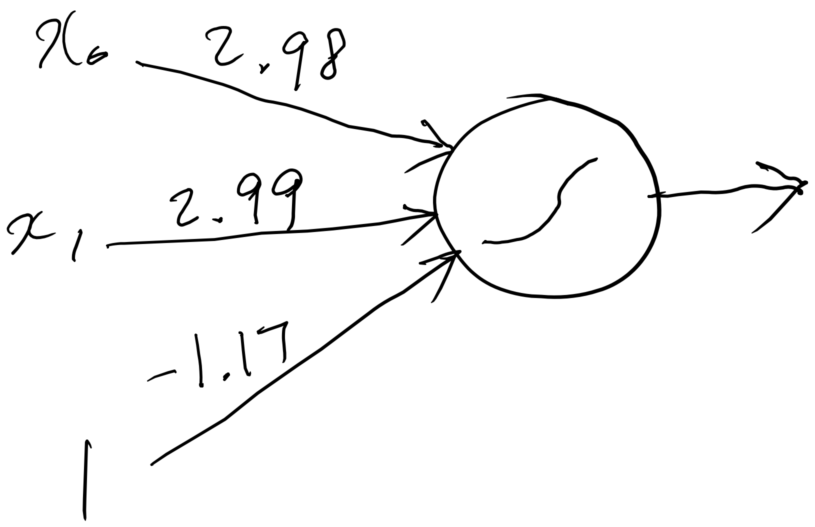
binary accuracy not printed here

Another result: your run results will be different, due to the randomly generated initial weight values.

```
for layer in model.layers:
  print(layer.get_weights())
```

```
[array([[2.985022 ],
       [2.9994013]], dtype=float32), array([-1.1718323], dtype=float32)]
[]
```

Weight for the bias




```
print(model.predict(X))
print(model.predict(X1))
print(model.predict(X1).round())
```

```
[[0.22538908]
 [0.86369103]
 [0.8702809 ]
 [0.9932018 ]]
[[0.35275817]
 [0.86369103]
 [0.9017872 ]
 [0.9963174 ]]
[[0.]
 [1.]
 [1.]
 [1.]]
```

Q: Does this training with 4 examples work all the time using Keras?

- No
- GD based search is heavily dependent on the initial search point randomly generated
- Too few examples
- Evolutionary search algorithms will be better, when the training datasize is small (EC is better for Few-shot learning)

Not trained...

```python
X1 = np.array([[0.1,0], [0,0.9], [0.9,0], [1,0.9]])
```

```python
# print learned weight values
for layer in model.layers:
  print(layer.get_weights())
```

```
[array([[2.7077217],
       [1.6644008]], dtype=float32), array([-0.26141205], dtype=float32)]
```

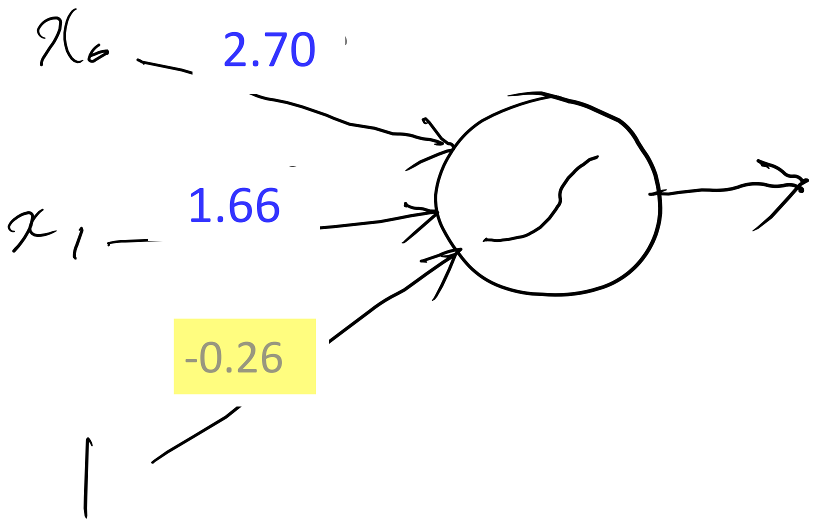
Weight for the bias

$$0.1 \times 2.70 + 0 \times 1.66 + 1 \times -0.26 = 0.01$$

$$\frac{1}{1+e^{-0.01}} = 0.502$$

```python
print(model.predict(X))
print(model.predict(X1))
print(model.predict(X1).round())
```

```
[[0.4350166 ]
 [0.8026577 ]
 [0.9202912 ]
 [0.98386836]]

[[0.50234   ]
 [0.77496266]
 [0.8980312 ]
 [0.98100257]]

[[1.]
 [1.]
 [1.]
 [1.]]
```



More examples of using model.predict()

```
[29] print(model.predict( np.array([X[0]]), verbose=0 )) # to get a label of the first sample
```

```
[[0.2688555]]
```

```
[30] print(model.predict( np.array([ [0.1, 0.05]]), verbose=0 )) # to get a lable of a test sample
```

```
[[0.37908965]]
```

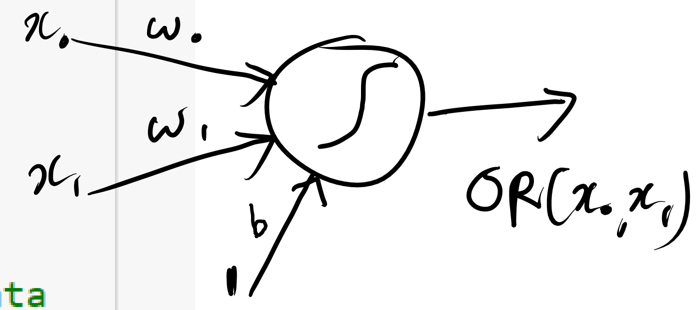
Model Evaluate with Unseen Test Dataset

```
X1 = np.array([[0.1,0], [0,0.9], [0.9,0], [1,0.9]])
```

```
# Evaluate the model with X1
loss, accuracy = model.evaluate(X1, y, verbose=0)
print(f"Loss on X1: {loss:.4f}")
print(f"Accuracy on X1: {accuracy:.4f}")
```

OR_function.ipynb

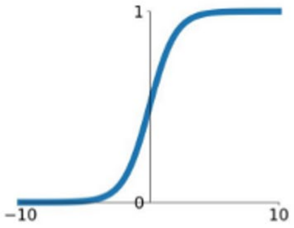
```
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
import numpy as np

# training data. Bias "1" is not needed in Keras
X = np.array([[0,0], [0,1], [1,0], [1,1]]) #training data
y = np.array([[0],   [1],   [1],   [1]  ]) #target labels
X1 = np.array([[0.1,0], [0,0.9], [0.9,0], [1,0.9]]) # Unseen & unlabeled data
```




```
model = Sequential([
    Dense(1, input_dim=2, activation="sigmoid")
])

model.compile(
    loss='mean_squared_error', # https://keras.io/losses/
    optimizer=optimizers.RMSprop(learning_rate=0.1), # https://keras.io/optimizers/
    metrics=["binary_accuracy"] # for binary classification problem
)

# batch_size: num of samples per weight update
# An epoch is an iteration over the entire X and y data provided
model.fit(X, y, batch_size=1, epochs=15)
```

Sigmoid

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



Loss functions: SSE, MSE, and MAE

```
y_desired = np.array([3,   4,   5,   6  ])
y =         np.array([5,   4,   5.1, 5.5])

def sumSqErr(y_d, y):
  sum_sq_err = 0.0
  for i in range (len(y_d)):
    sum_sq_err += (y_d[i]-y[i])**2
  return sum_sq_err

def meanSqErr(y_d, y):
  return sumSqErr(y_d, y) / len(y_d)
```

$$SS = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

$$\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$$

MSE = mean squared error

n = number of data points

Y_i = observed values

$\hat{Y}_i$ = predicted values

Please test "**SSE_MSE_MAE.ipynb**"

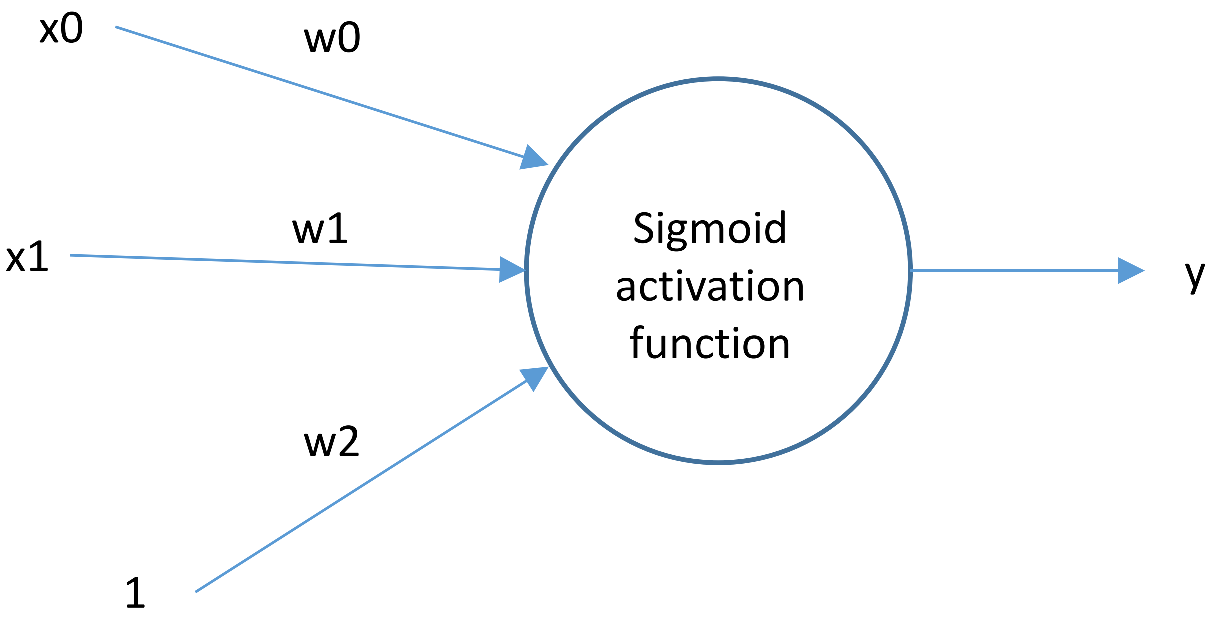
```
sse = ((y - y_desired)**2).sum()
mse = ((y - y_desired)**2).mean()
```

$$MAE = \frac{\sum_{i=1}^{n} |\, y_i - \hat{y}_i \,|}{n}$$

Self-Exercise using Keras: AND function

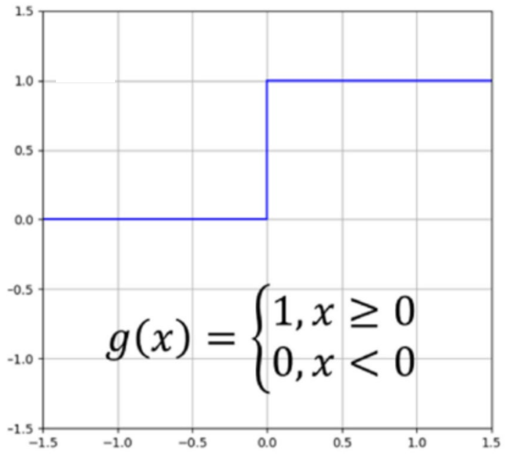
- Test “AND” function based on OR_function.ipynb

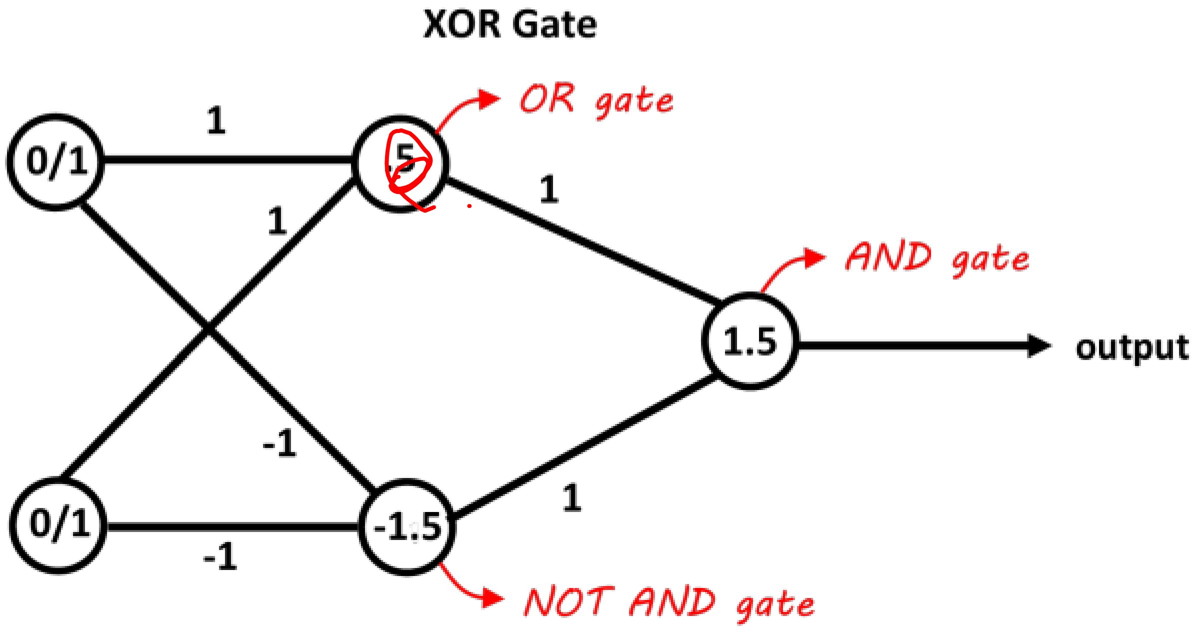
x0	x1	y
0	0	0
0	1	0
1	0	0
1	1	1



A simple XOR using StairStep activation function

x0	x1	y
0	0	0
0	1	1
1	0	1
1	1	0

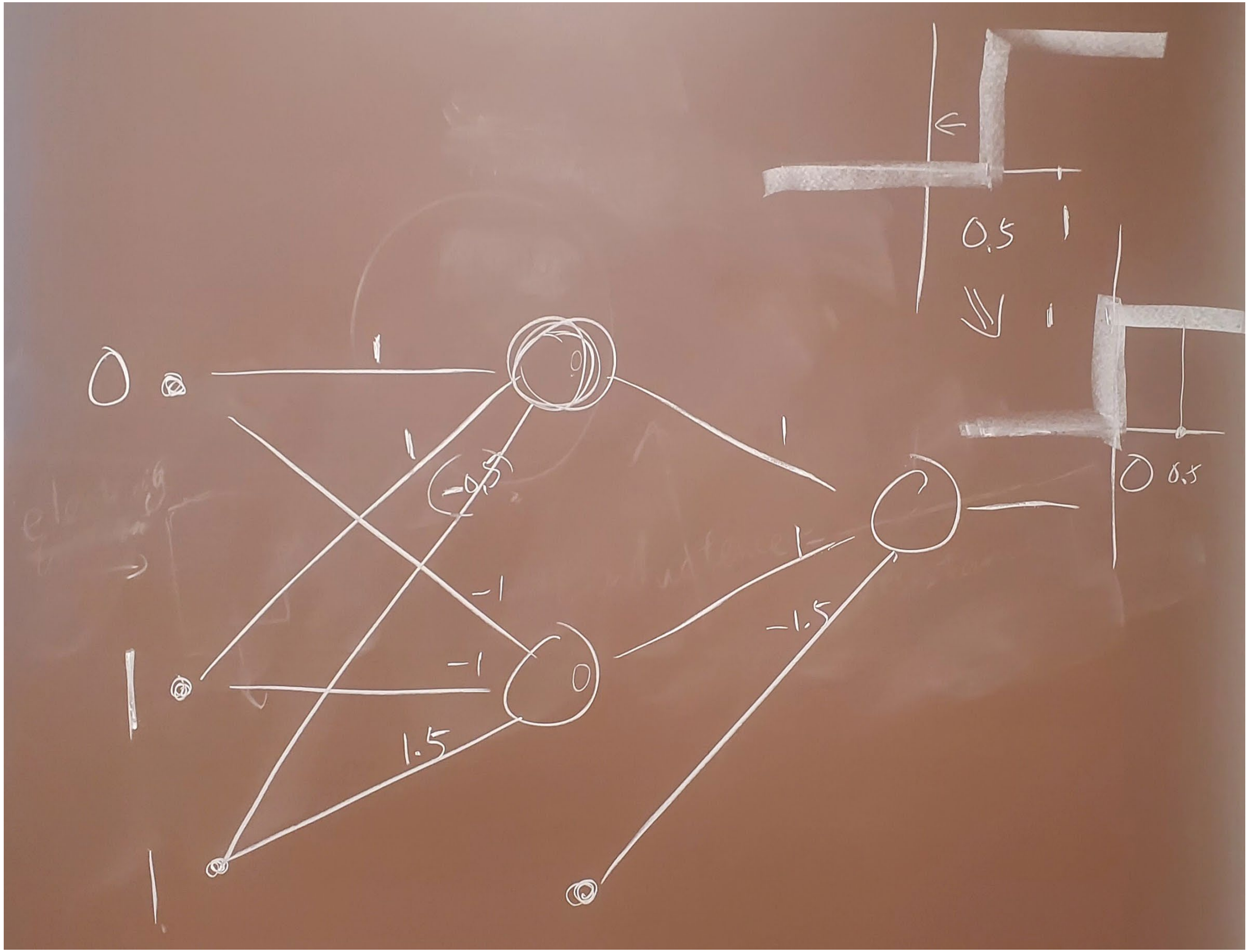




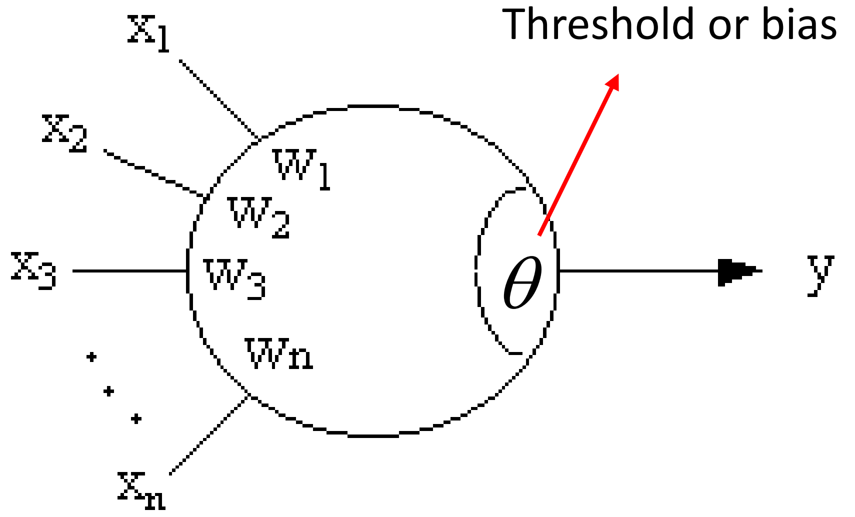
0

1

$g(0 \cdot 1 + 1 \cdot 1 - 0.5) = 1$


0
1
1
-0.5
-1
-1
1.5
1
1
-1.5
0.5
1
1
0
0.5

Review: Perceptron using SS funtcion



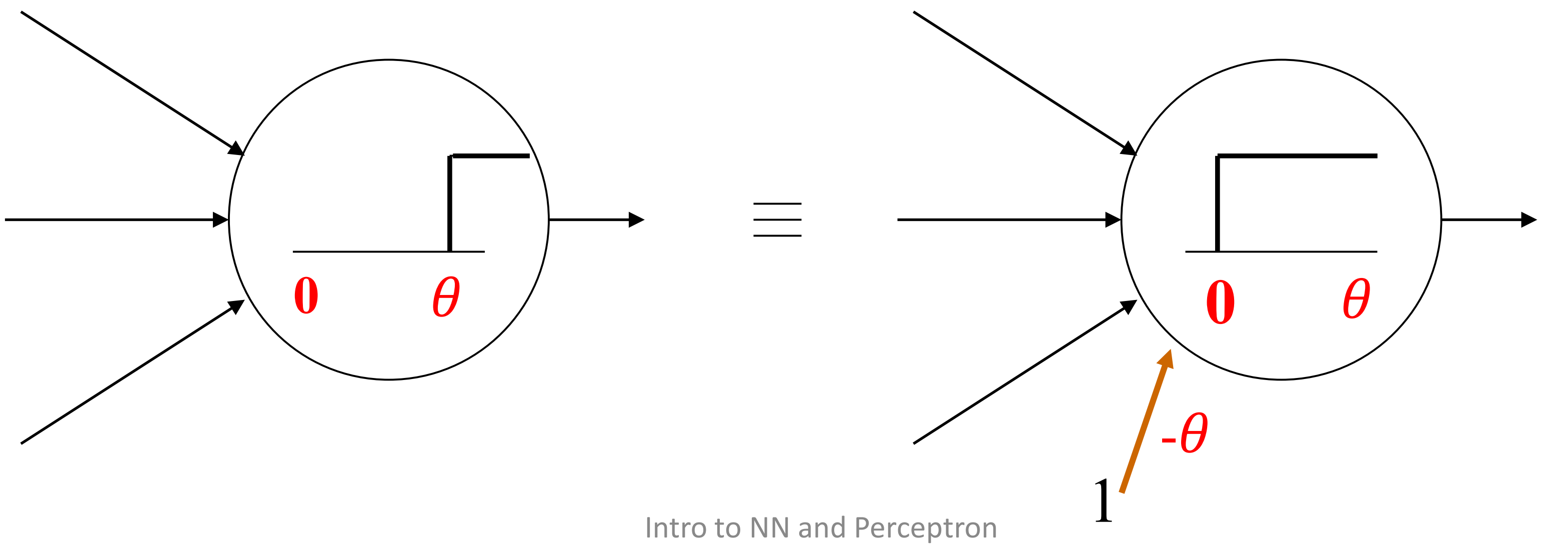
f: activation function

$$wsum = \sum_{i=1}^{n} w_i x_i$$

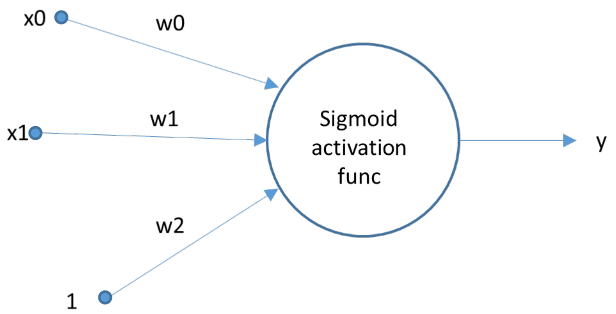
$$y = \begin{cases} 1 & if \;\; wsum > \theta \\ 0 & if \;\; wsum \le \theta \end{cases}$$

Removing non-zero threshold value

- Non-zero threshold can be eliminated.
- A nonzero-threshold neuron is computationally equivalent to a **zero-threshold** neuron with an extra link held at 1 (not –1, in case of basic perceptron). The -θ value becomes the connecting weight's value.

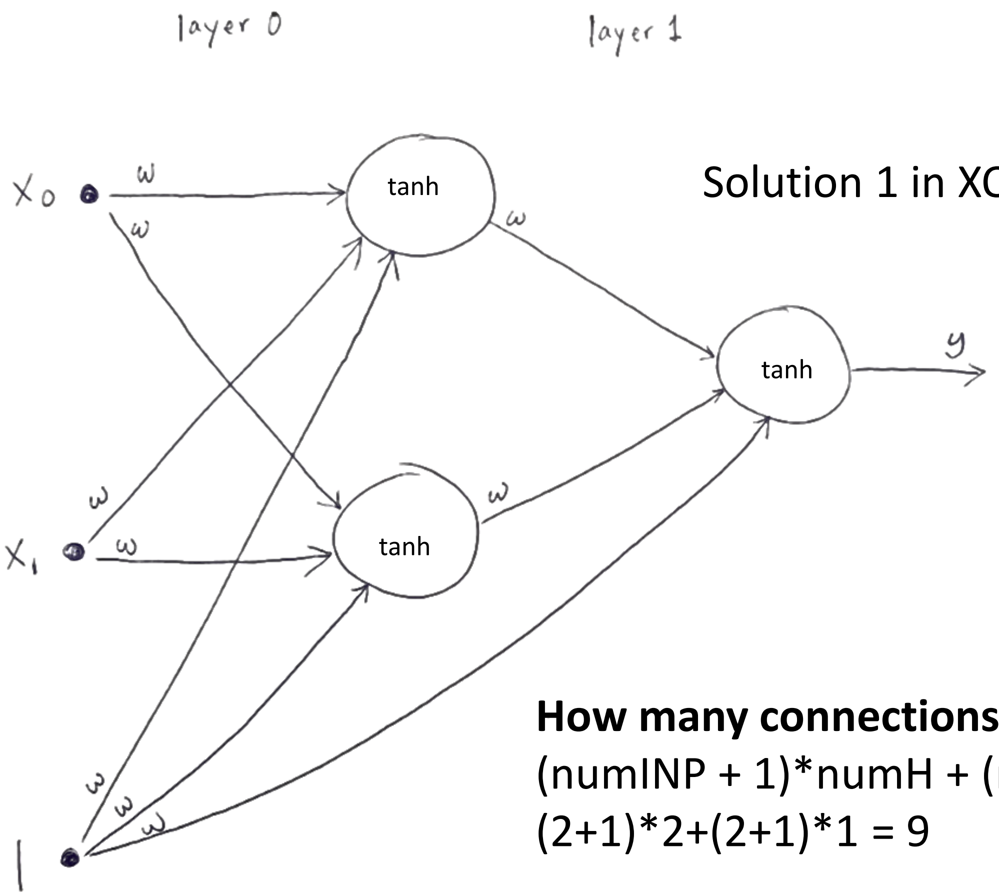


Keras Example: XOR.ipynb (on Canvas, sol. 1)



Pappart proved that this is not working

x0	x1	y
0	0	0
0	1	1
1	0	1
1	1	0



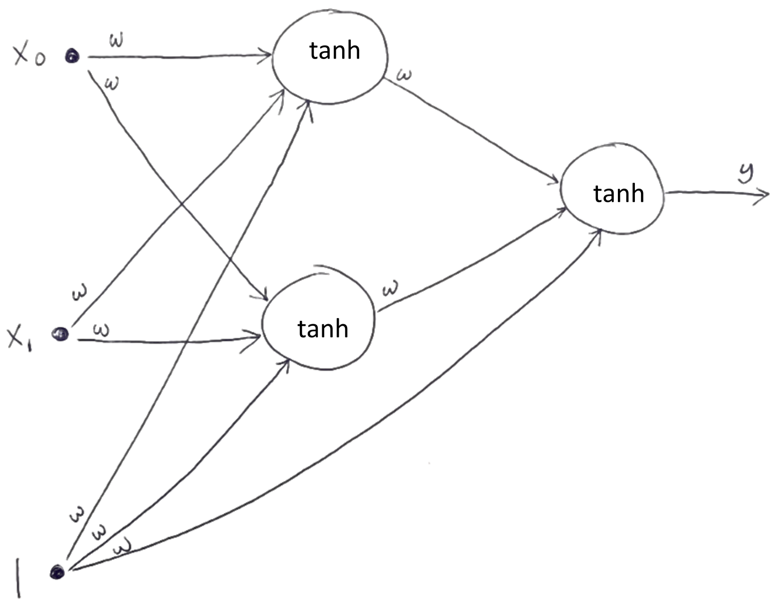
Solution 1 in XOR.ipynb

How many connections (weights)?

(numINP + 1)*numH + (numH + 1) * numO

(2+1)*2+(2+1)*1 = 9

Keras Example: XOR.ipynb (on Canvas, sol. 1)



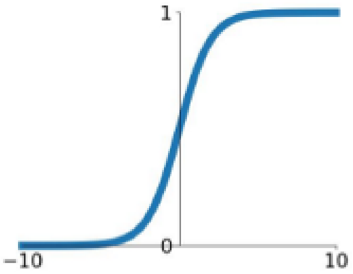

```python
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
import numpy as np

X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[0],[1],[1],[0]])

model = Sequential([
    Dense(2, input_dim=2, activation='tanh'),
    Dense(1, activation='tanh')
])

model.compile(
    loss='mean_squared_error',
    optimizer=optimizers.SGD(learning_rate=0.1), # stochastic gradient decent
    #optimizer=optimizers.RMSprop(learning_rate=0.01),
    metrics=['binary_accuracy']
)
model.fit(X, y, batch_size=1, epochs=200)
```

Lawrence Technological University Computer Science

Activation Functions

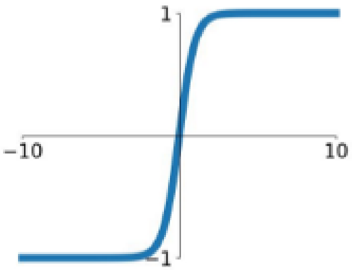
Sigmoid

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



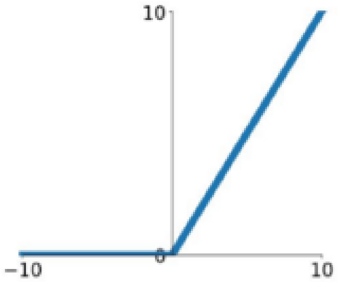
tanh

$\tanh(x)$



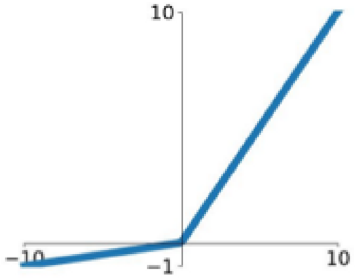
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

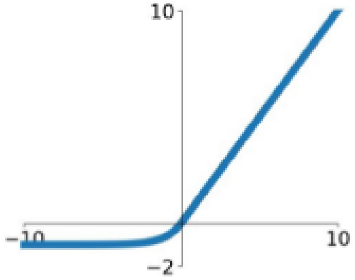


Maxout

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

ELU

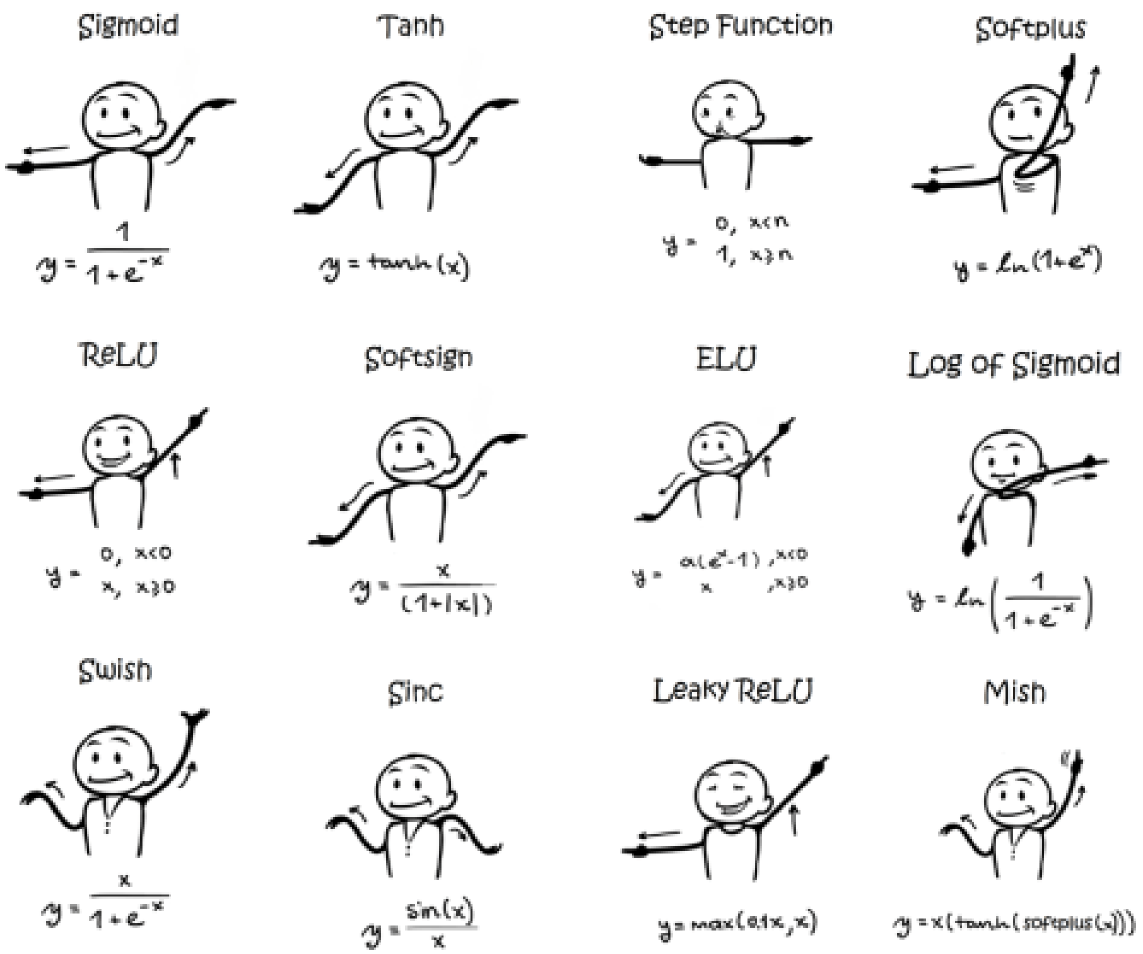
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



https://medium.com/@shrutijadon10104776/survey-on-activation-functions-for-deep-learning-9689331ba092

Dance Moves of 12 DL Activation Functions

https://youtu.be/1Du1XScHCww

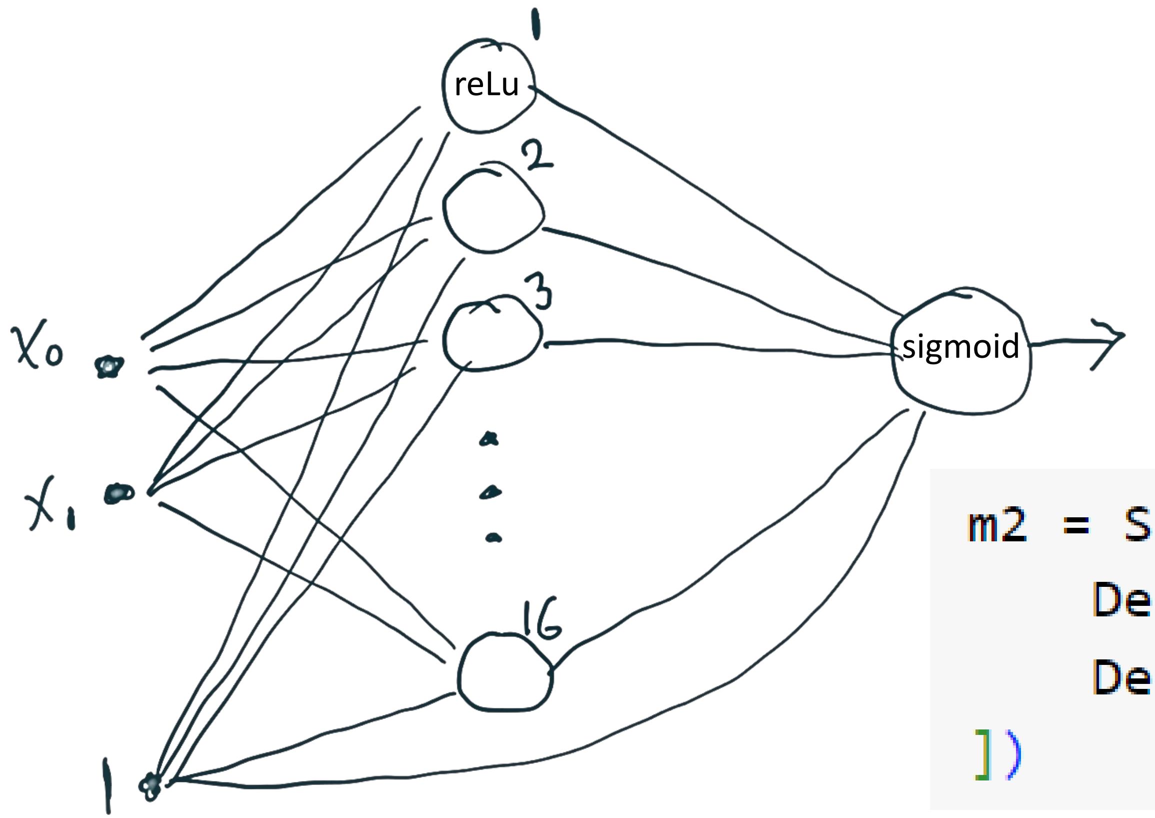


Lawrence Technological University
Computer Science

Plotting Activation Functions using "matplotlib"

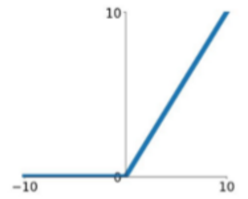
- https://youtu.be/D1twox0mcrM
- https://github.com/yacineMahdid/artificial-intelligence-and-machine-learning/blob/master/deep-learning-from-scratch-python/activation_functions.ipynb (code)

XOR Solution 2 in XOR.ipynb with 16 hidden neurons



ReLU

$\max(0, x)$

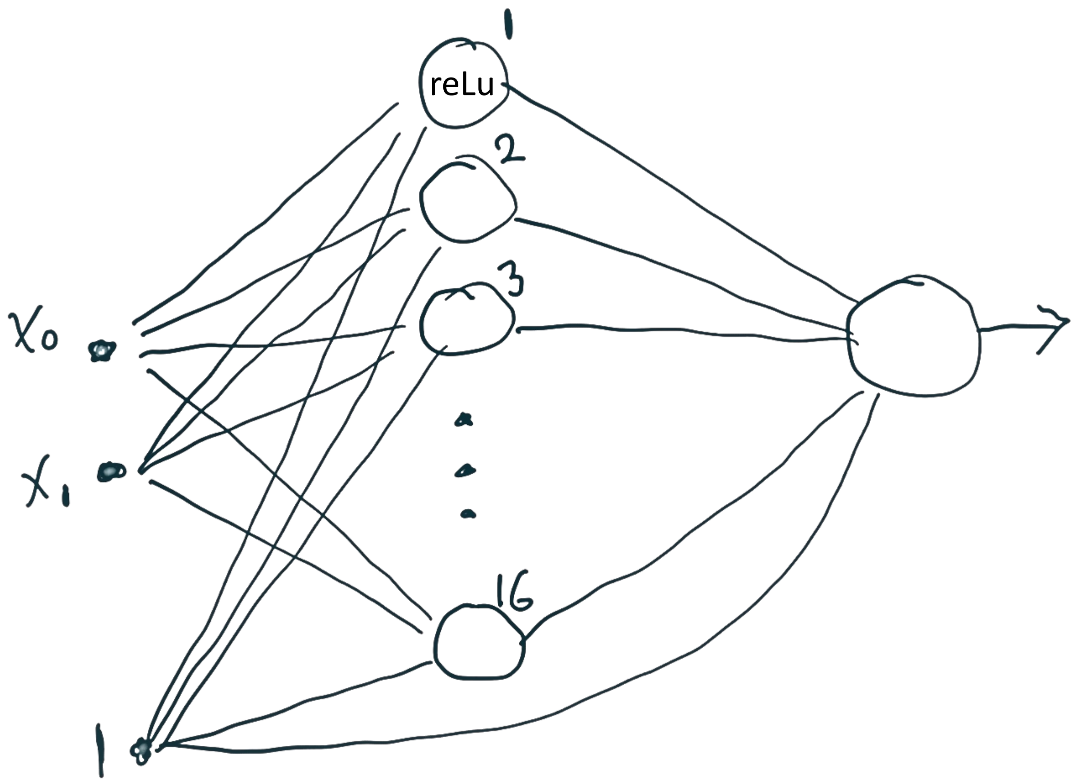


relu was the best. Tanh or sigmoid seems not working

```
m2 = Sequential([
    Dense(16, input_dim=2, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

***Alternative optimizer you can try: <u>adam</u>* - a SGD method that computes individual adaptive learning rates for different parameters from estimates of first- and second-order moments of the gradients. Its is faster than SGD.**

XOR Solution 2 in XOR.ipynb with 16 hidden neurons



How many weight values to train?

3x16 + 17x1 = 65

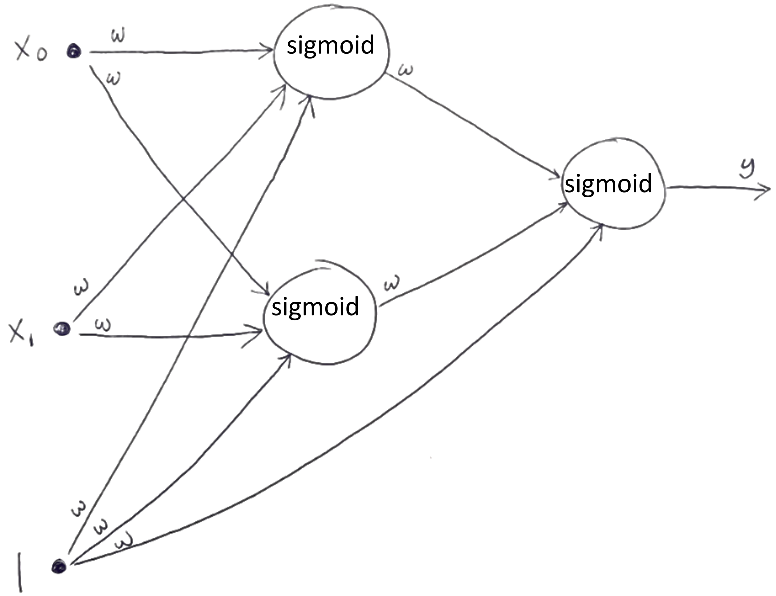
```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 16)                48
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 17
=================================================================
Total params: 65
```

Self-Ex: training XOR with more samples (Solution 1a) – Hard to train if sigmoid funcs are used for all the neurons

Introduce more samples to train a model for XOR with sigmoid activation function, for example

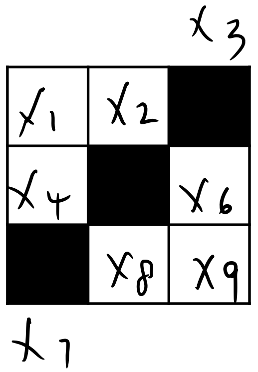
```
X = np.array([[0,0],[0.1,0.1],[0,1],[0.1,0.9],[1,0],[0.9,0.1],[0.9, 0.9],[1,1]])
y = np.array([[0],  [0],      [1],  [1],      [1],  [1],      [0],       [0]])
```

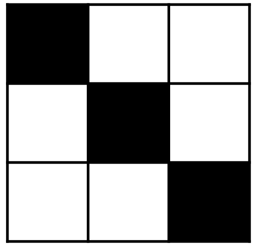


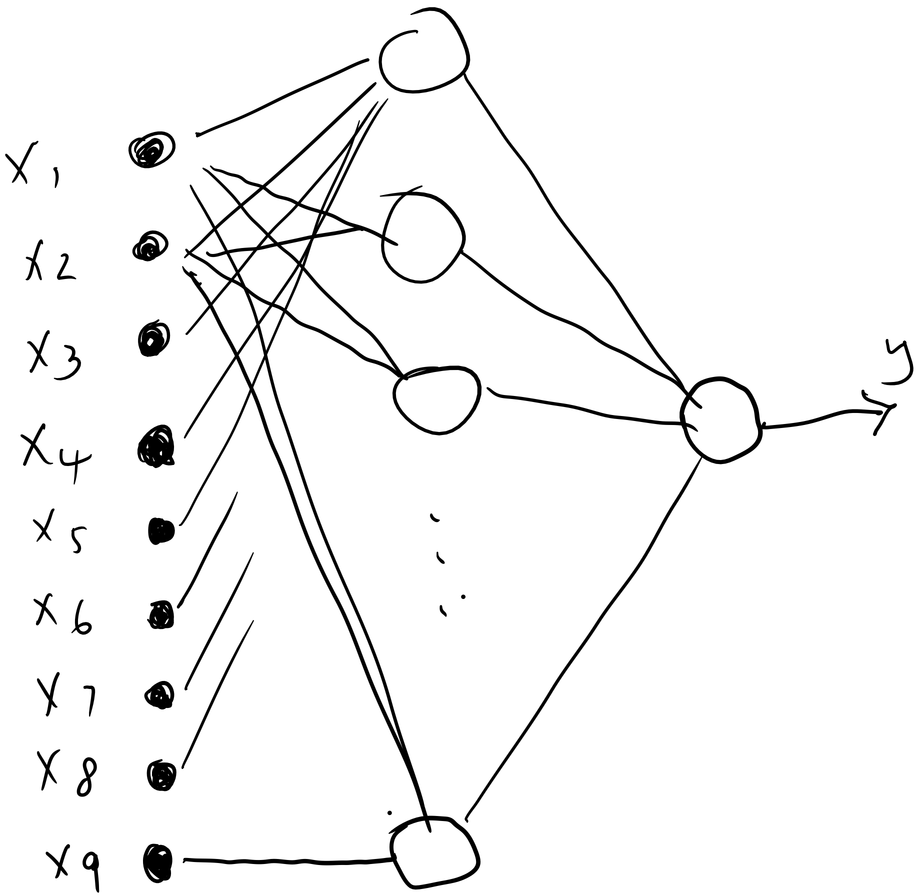
Train models for XOR is not easy when we have only a few training data…

- We will transform XOR training problem as an minimization function
- Then we will use ES(1+1) with 1/5 optimizer to find weight values

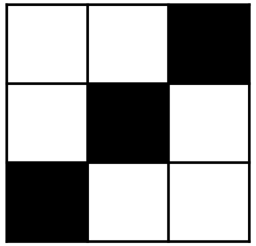
Self-Exercise: Test S_BS0.ipynb – Classifier for Slashes and Backslashes

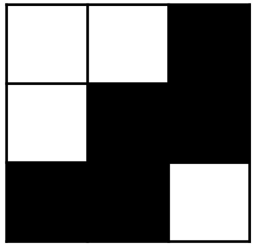


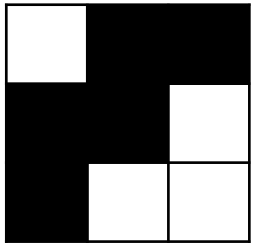




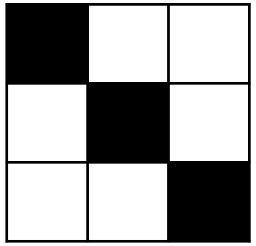
How to represent inputs & desired labels?







…

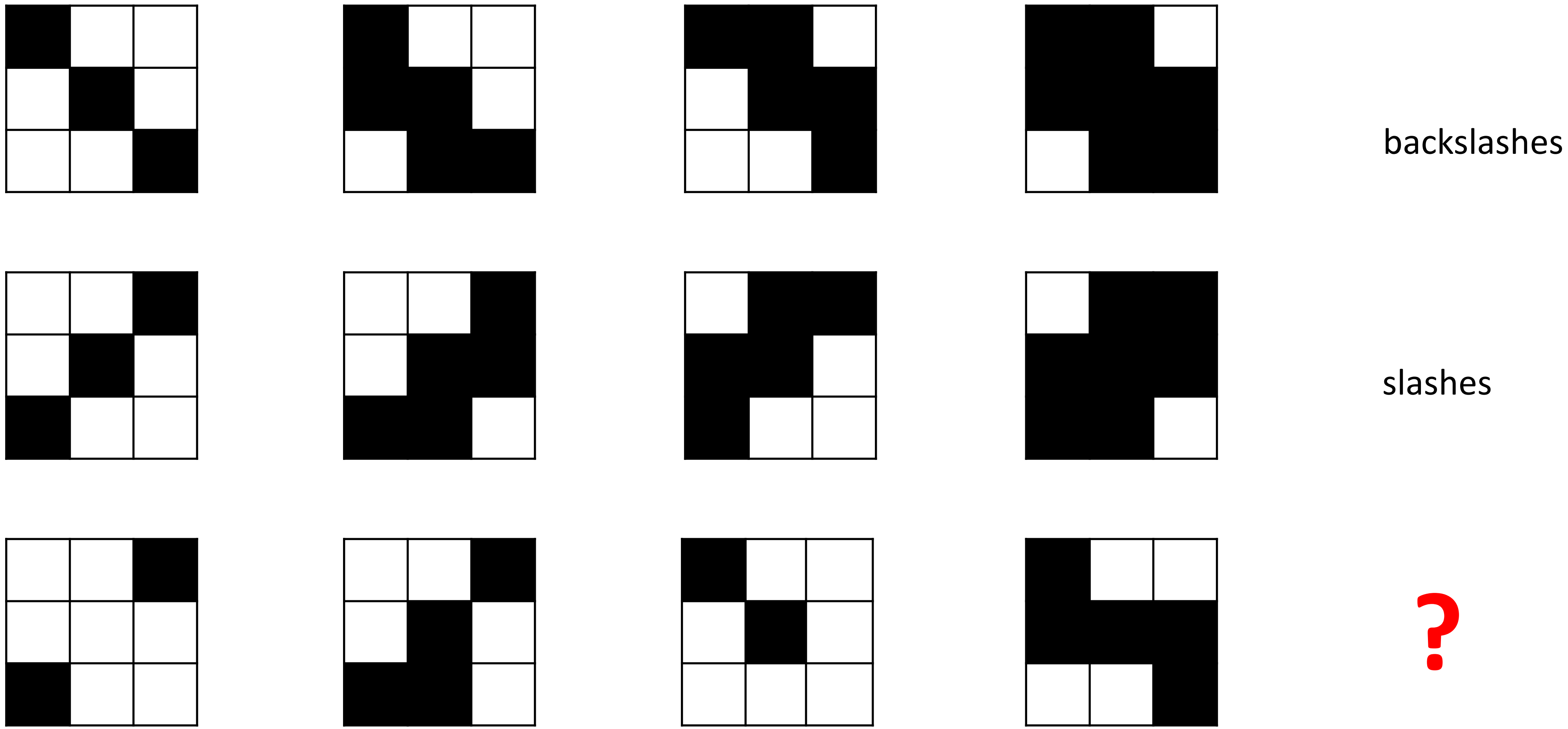


…

```
training_data = np.array([
                         [0,0,1,
                          0,1,0,
                          1,0,0],
                         [0,0,1,
                          0,1,1,
                          1,1,0],
                         [0,1,1,
                          1,1,0,
                          1,0,0],
                           ...
                         [1,0,0,
                          0,1,0,
                          0,0,1],
                          ...
                        ],
                        “float32”)
```

```
train_labels = np.array([[0],[0],[0], … # Slash
                         [1], …         # Backslash
                        ],
                        "float32")
```

Self-Ex: Add more examples and test the following unseen data

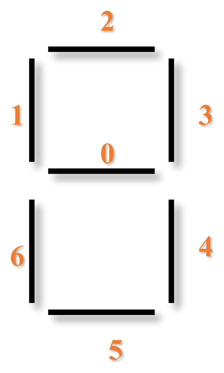


Another sample program on Canvas

- S_BS3.ipynb
- using (4,3,3,1) gray image shape to input
- Assume each cell has [0 - 9], where 9 is the darkest

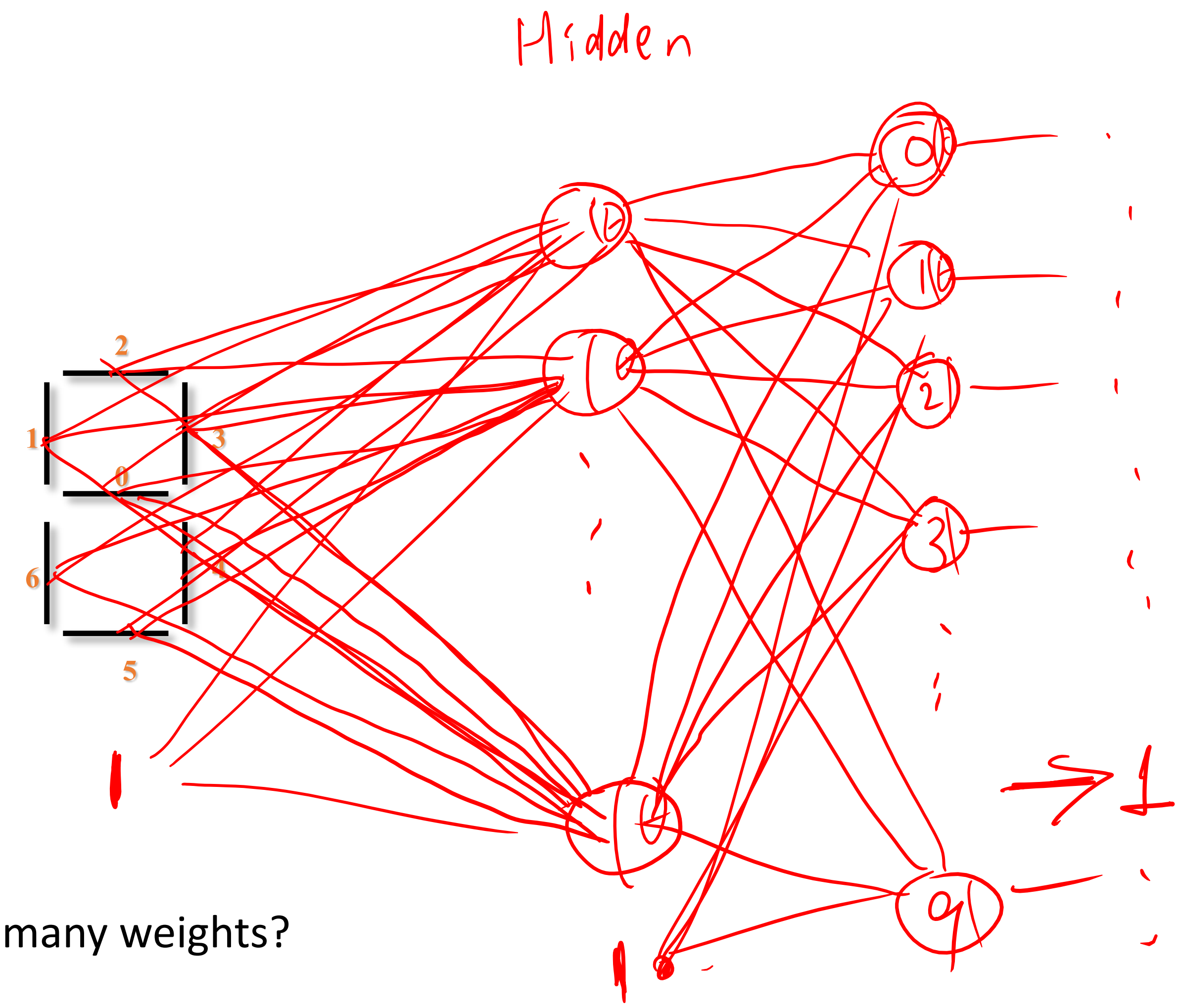
```
training_data = np.array([# Slash /
                          [[[0],[0],[9]],
                           [[0],[9],[0]],
                           [[9],[0],[0]]],

                          [[[0],[0],[8]],
                           [[0],[8],[0]],
                           [[8],[0],[0]]],

                          # Backslash \
                          [[[9],[0],[0]],
                           [[0],[9],[0]],
                           [[0],[0],[9]]],

                          [[[8],[0],[0]],
                           [[0],[8],[0]],
                           [[0],[0],[8]]]
                         ],
                         "float32")
```

Shallow NN with Input, Hidden, and Output layers for the 7 segment digit recognition task



- How many input neurons?
- How many output neurons?
- How many hidden neurons?

Shallow NN with Input, Hidden, and Output layers for the 7 segment digit recognition task



$$(7+1)\cdot H + (H+1)\cdot 10$$

How many weights?

Plotting with matplotlib (PyPlot, PyLab)

- https://queirozf.com/entries/matplotlib-pylab-pyplot-etc-what-s-the-different-between-these
- Test and study: “matplotlib_0.ipynb” was posted

Q: Learning Methods for ML/DL

ML/DL

- **Supervised learning**
  - Develop predictive model based on example input data and answers
  - ? : output variable is categorical (or discrete).
  - ? : output variable is numerical (or continuous)
- **Reinforcement Learning**
  - Policy optimization, control, path finding
- **?**
  - Group & interpret data based on input data with **no answers** (labels).
  - Clustering, association, …

Q. Major Types of Data for DS, ML/DL (excluding images, audio, …)

- Qualitative: in narrative form, usually from open survey
- ? - Numerical
  - (integer based) Discrete
    - 5 kids
    - 34 workers
    - 3 purchases in 2022
  - Continuous (can take any value between two numbers)
    - 3.25kg
    - 17.576534 miles
- ? : numbers can be used, but no mathematical meaning
  - Gender
  - US State
- ? : a mixture of numerical and categorical. Mathematical meaning
  - Movie ratings. 1 is worse than 2.

Important

- Please test sample programs using Keras
  - OR_function.ipynb
  - XOR.ipynb
  - Others
- Modify OR_function.ipynb to solve AND function
- Keras will be used for Hyperparameter Optimization