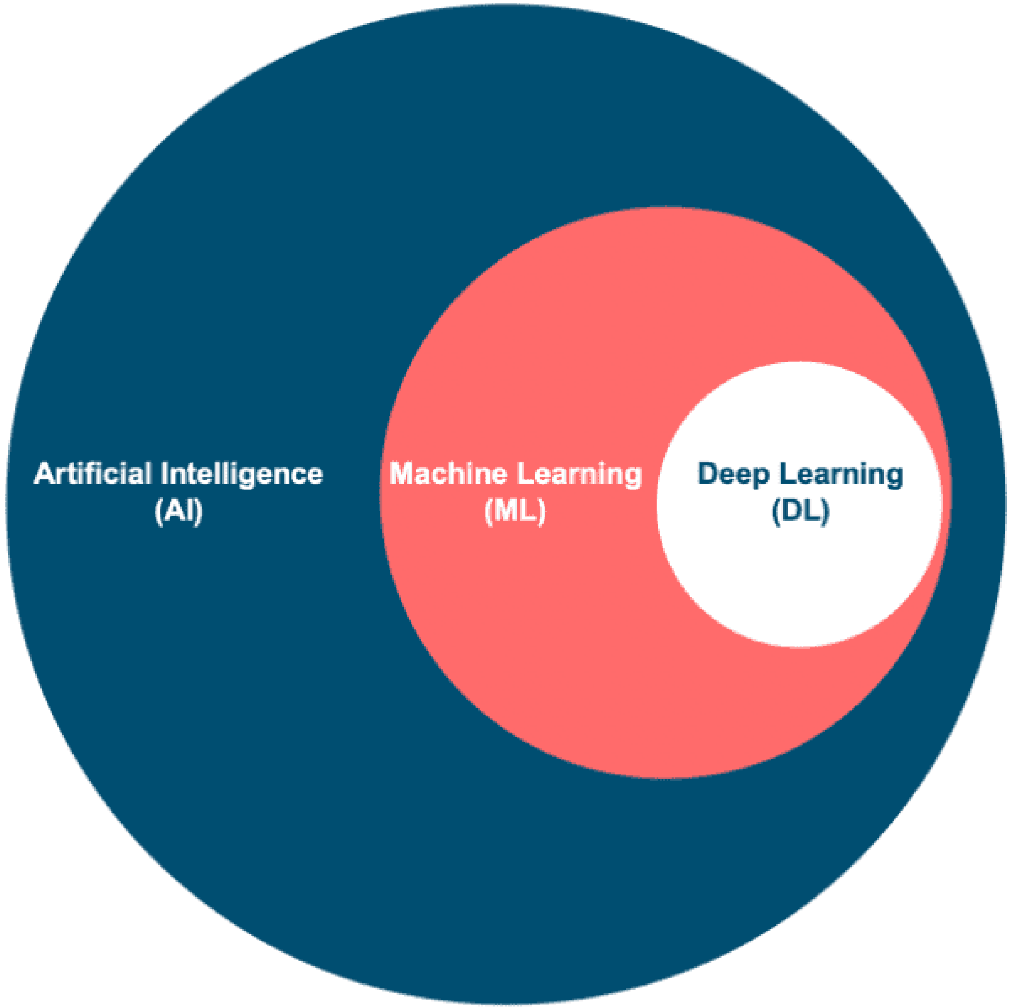
Mechatronics

Day9

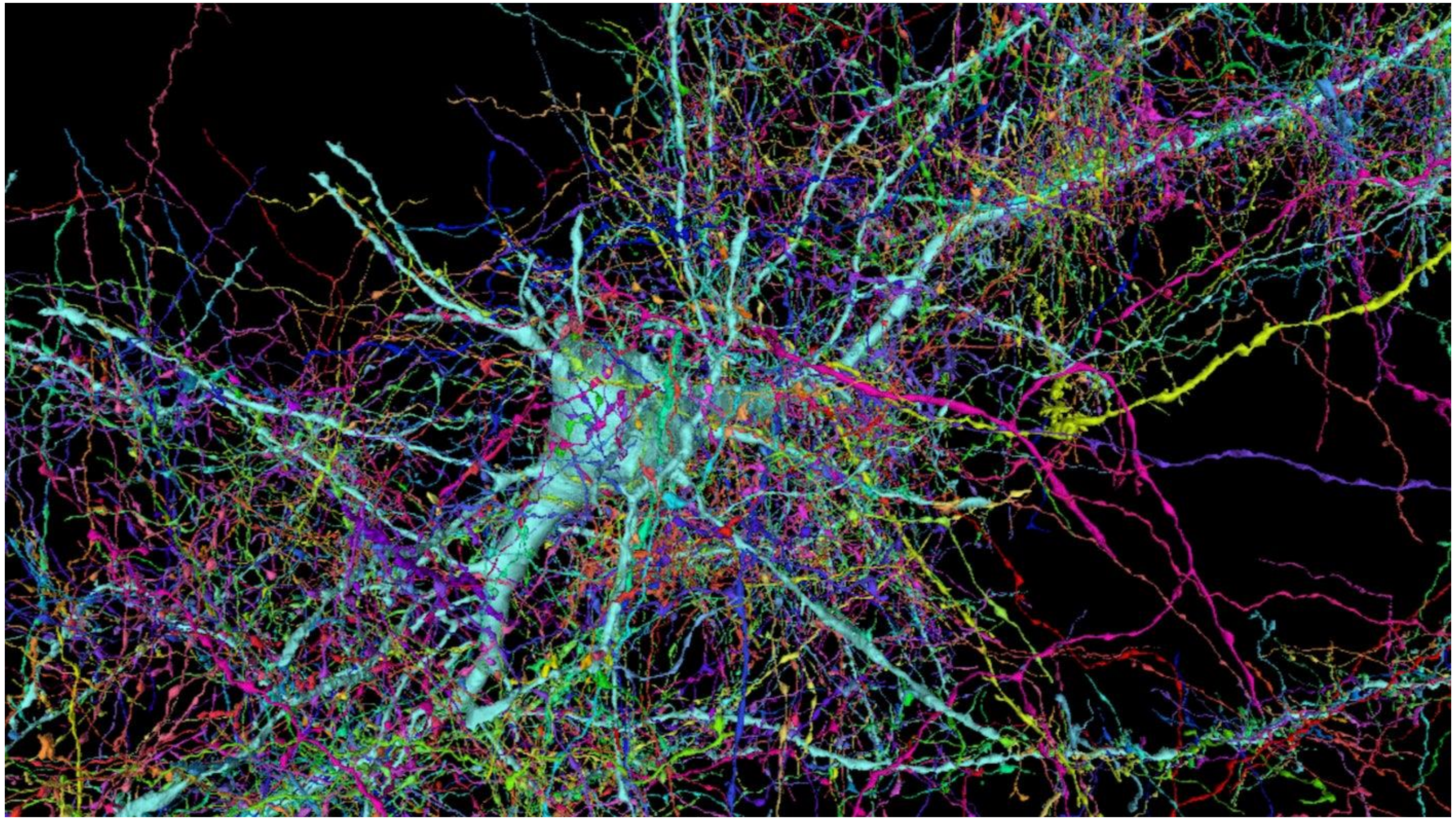
Deep Learning

AI ML and DL

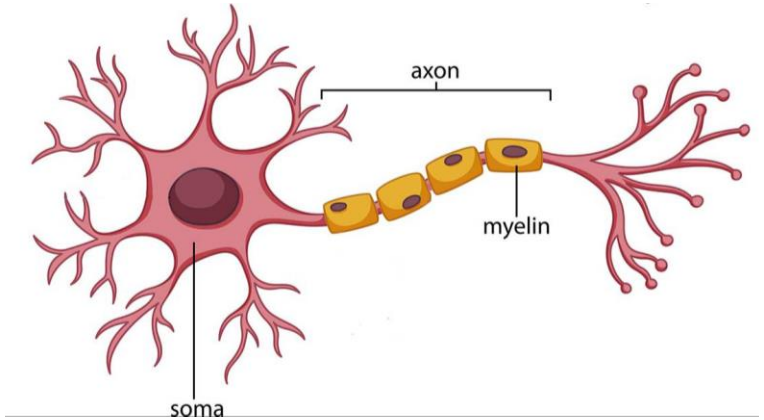


- Deep Learning is a specialized field of Machine Learning that relies on training of Deep Artificial Neural Networks (ANNs) using large dataset such as images.
- ANNs are information processing models inspired by **the human brain**.

Artificial Neural Networks (ANNs)



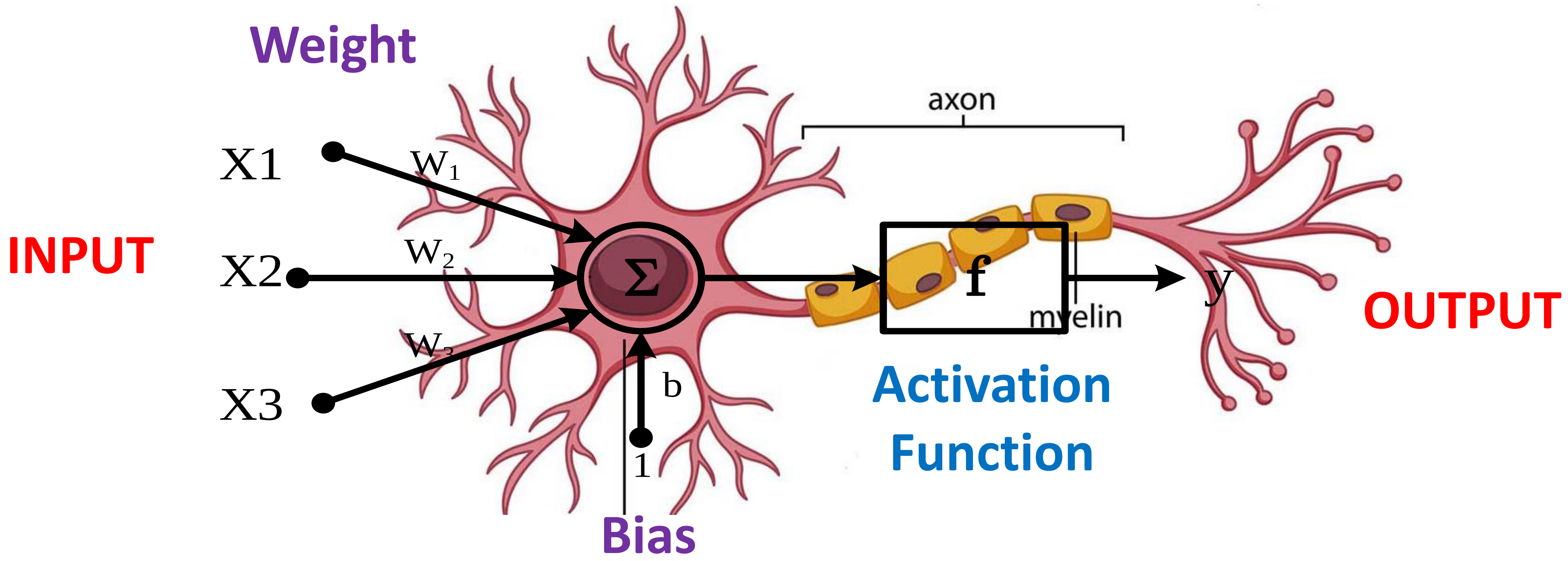
Human Neural Networks



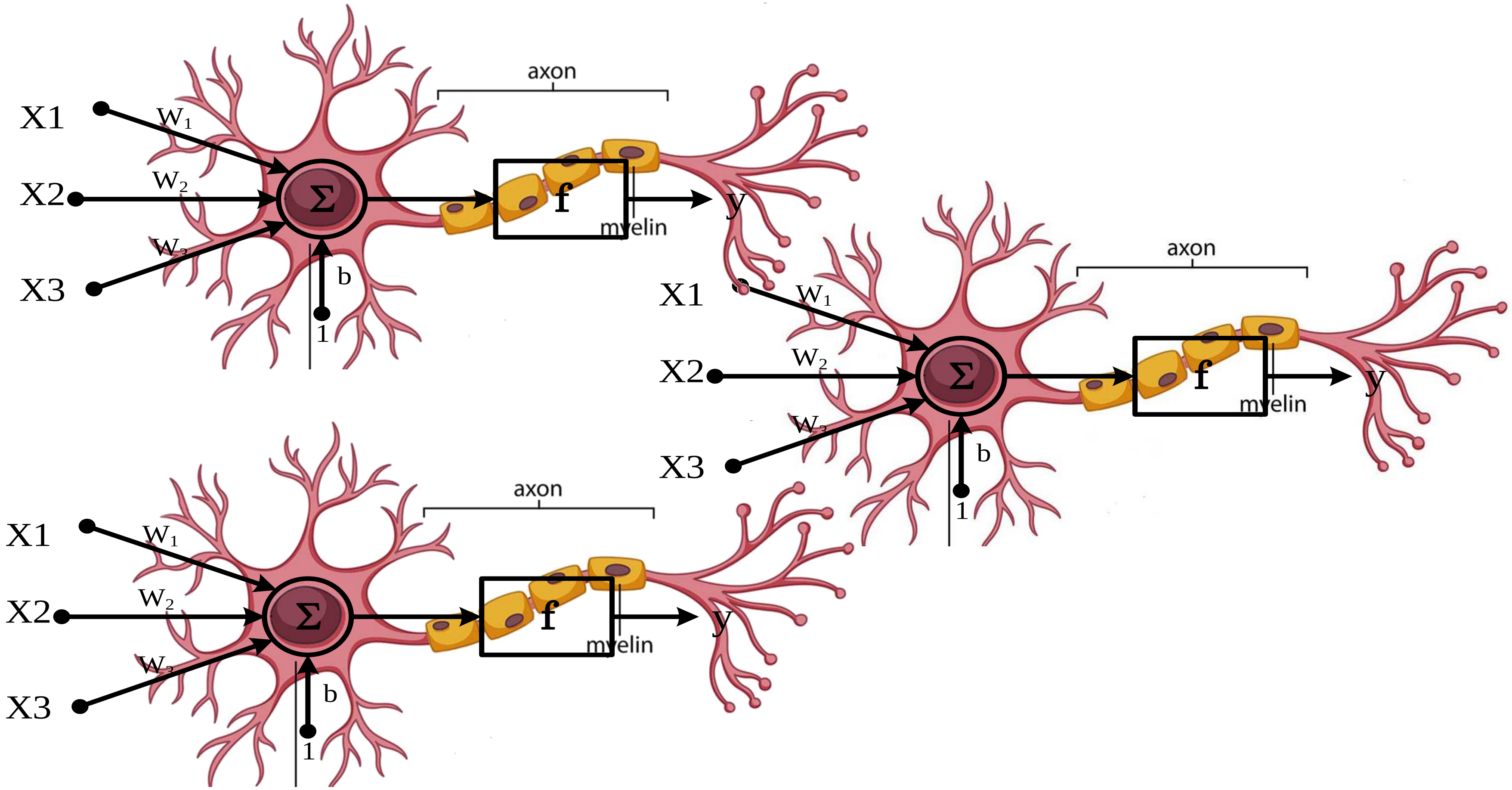
Neuron

https://www.hopkinsmedicine.org/health/conditions-and-diseases/anatomy-of-the-brain

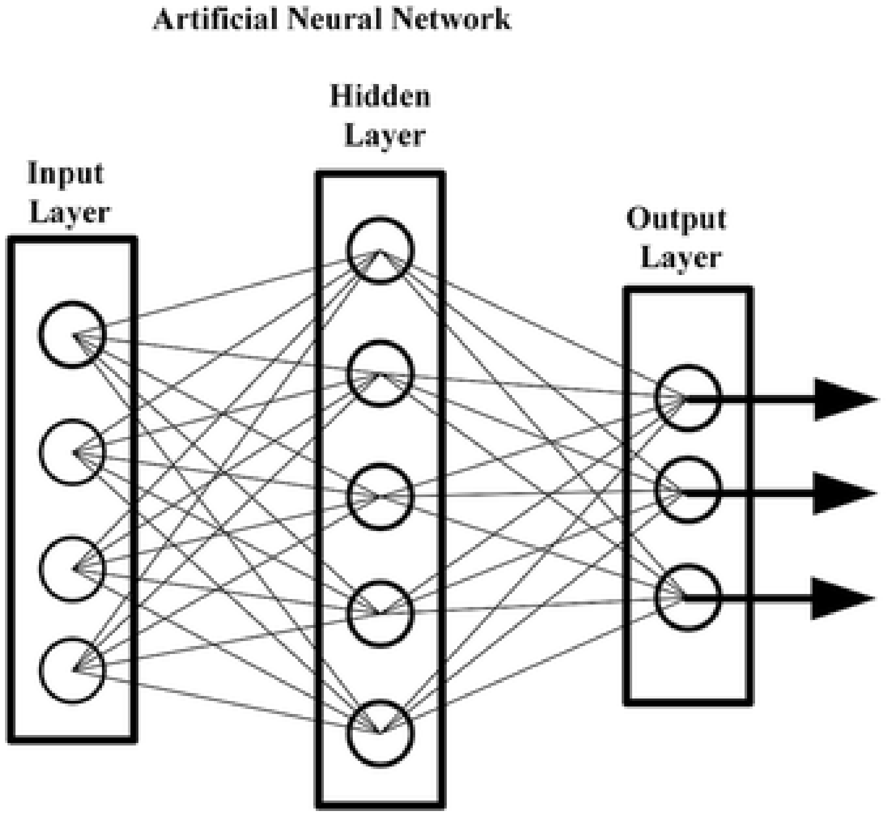
Artificial Neural Networks (ANNs)

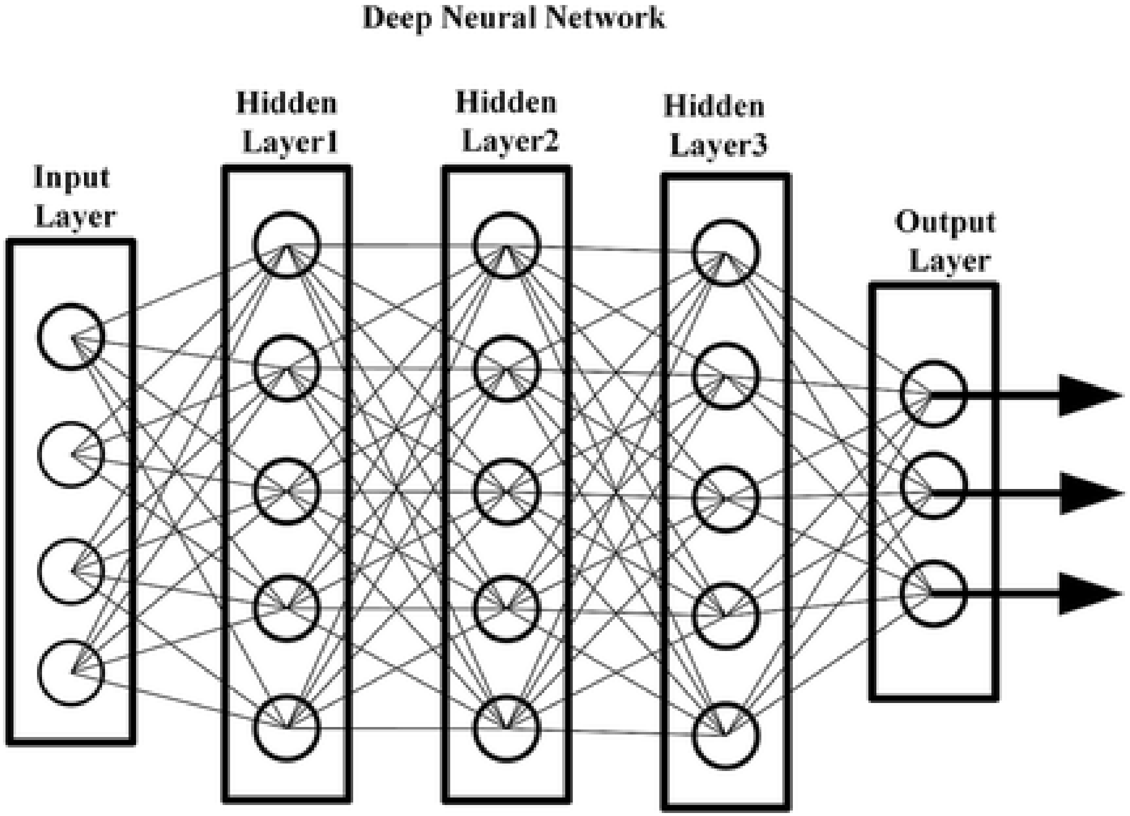


Artificial Neural Networks (ANNs)



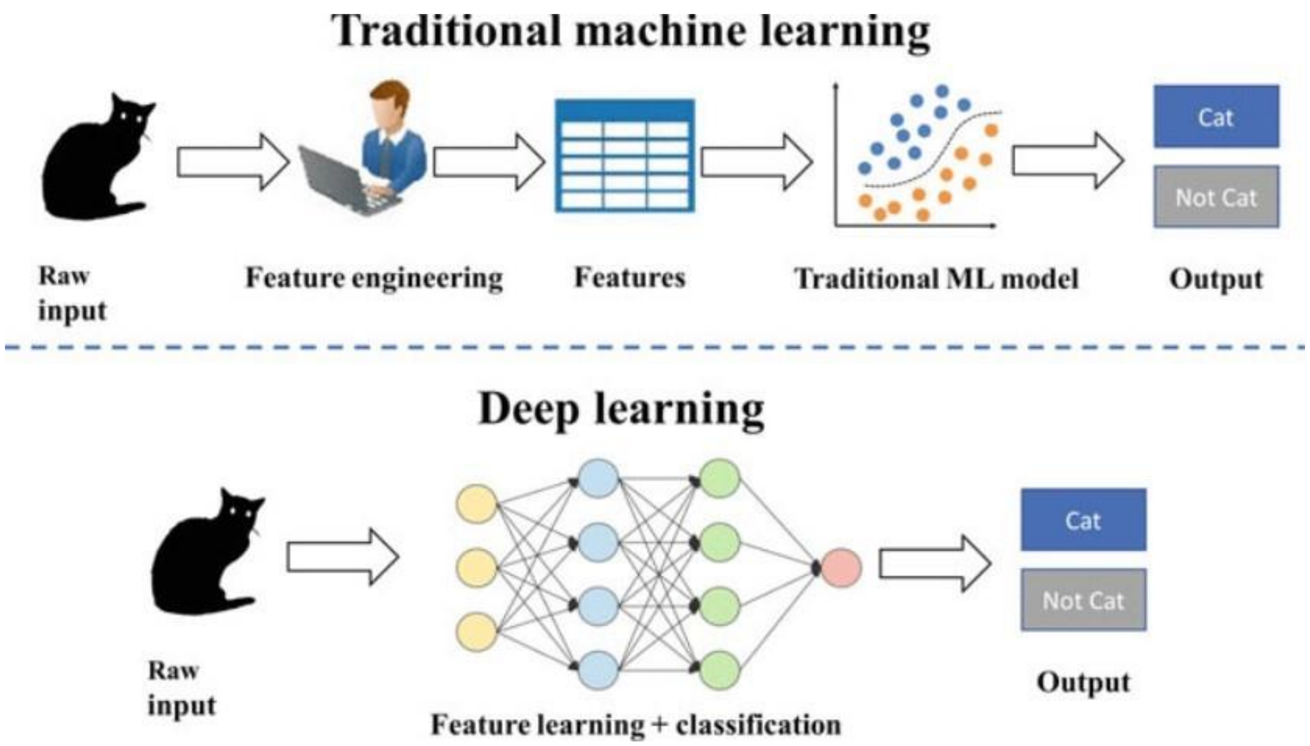
Deep Neural Networks





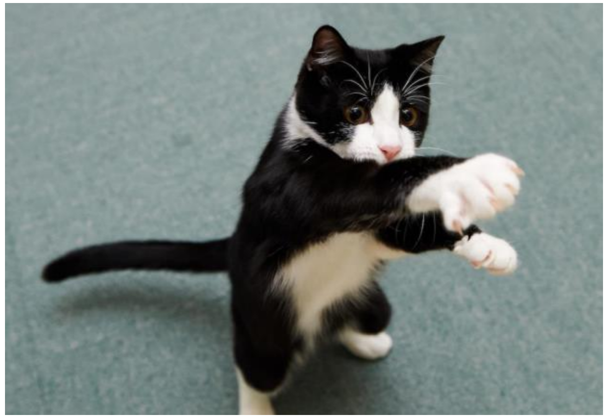
Machine Learning vs Deep Learning

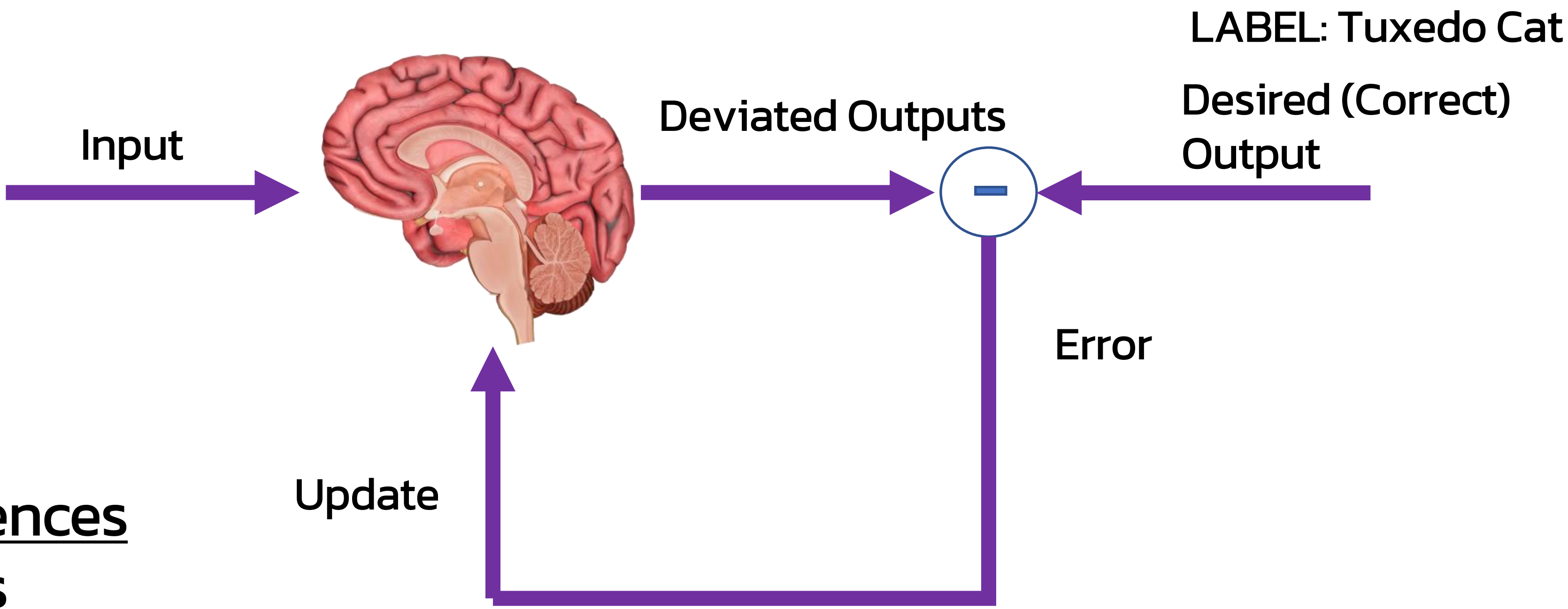
- Machine learning Process: (1) select the model to train, (2) manually perform feature extraction.
- Deep Learning Process: (1) Select the architecture of the network, (2) features are automatically extracted by feeding in the training data along with the target class (label).



https://www.linkedin.com/pulse/what-deep-learning-kognitiv-club/

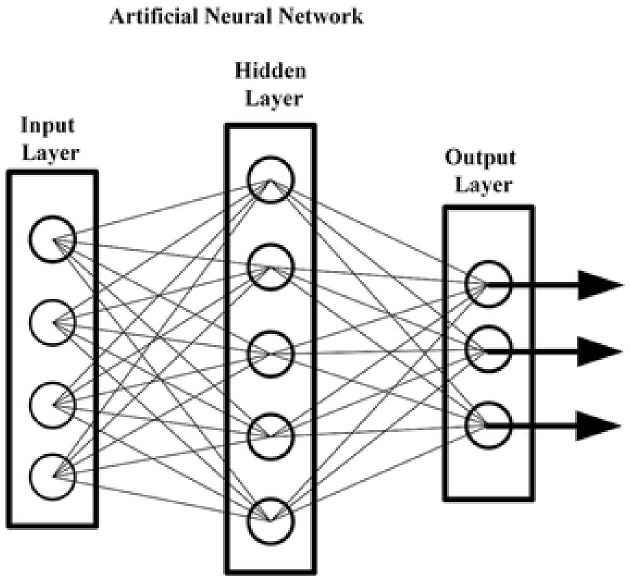
Human and Deep Learning Concept

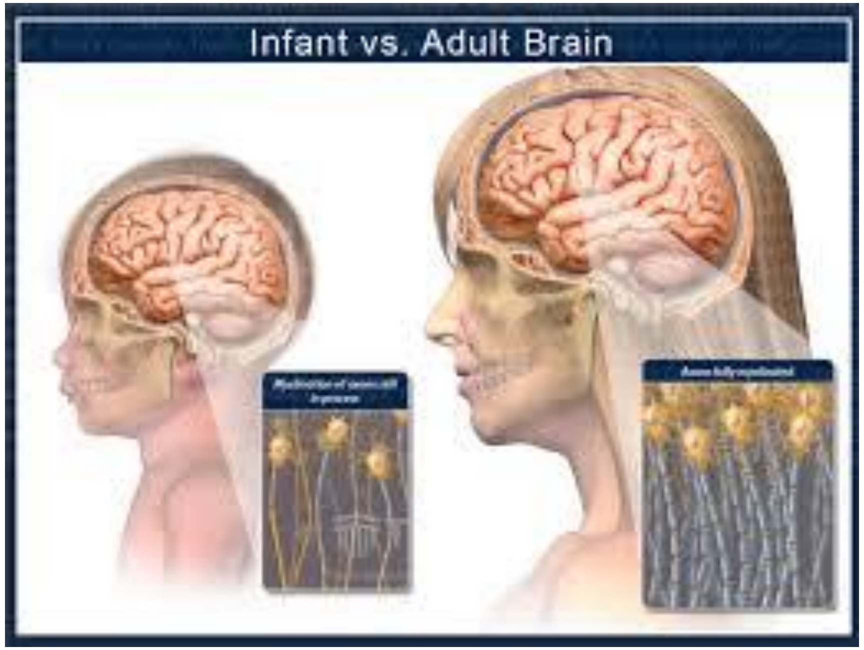


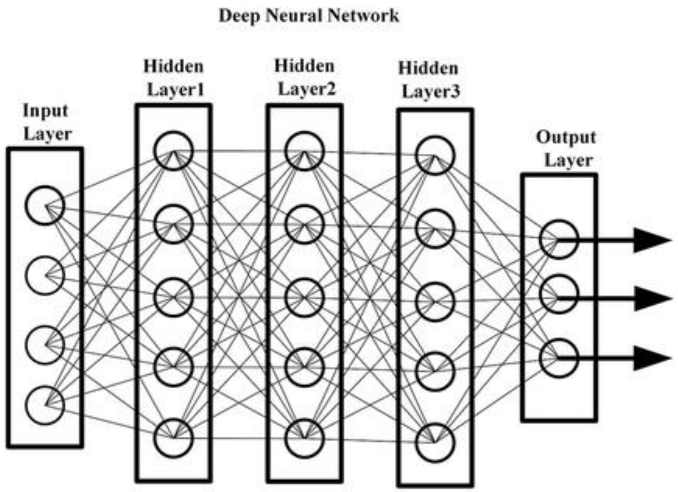


- Learn from <u>Experiences</u>
- Learn by <u>Examples</u>
- Learn <u>Over time</u>

Human and Deep Learning Concept



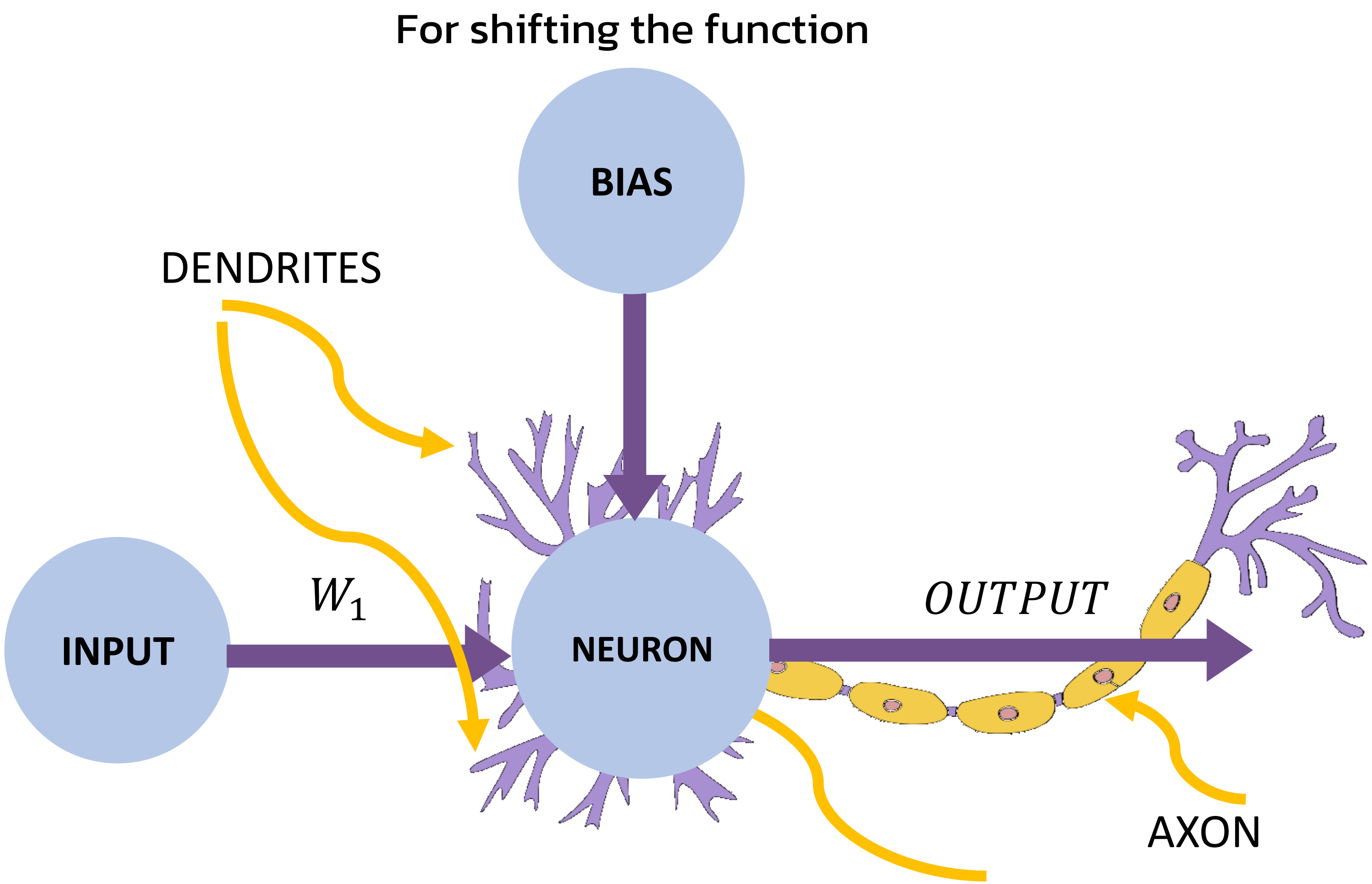




Smaller network → Smaller brain → Smaller capability

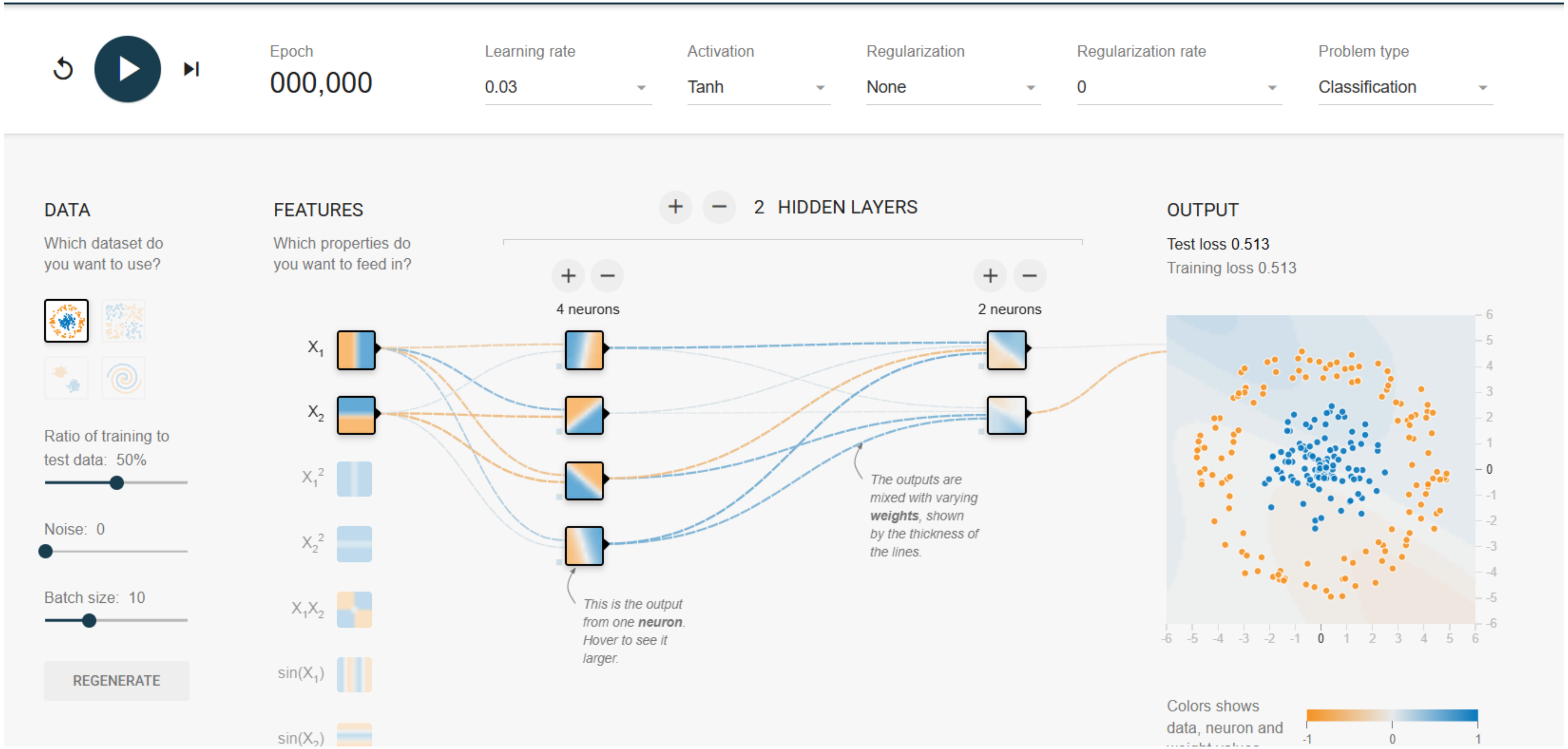
Simple Neuron

$$\boldsymbol{Output = Input \ * W_1 + Bias}$$



TensorFlow Playground

Let's play at tensorflow playground: https://playground.tensorflow.org/



DL Development Tools

TensorFlow

NumPy

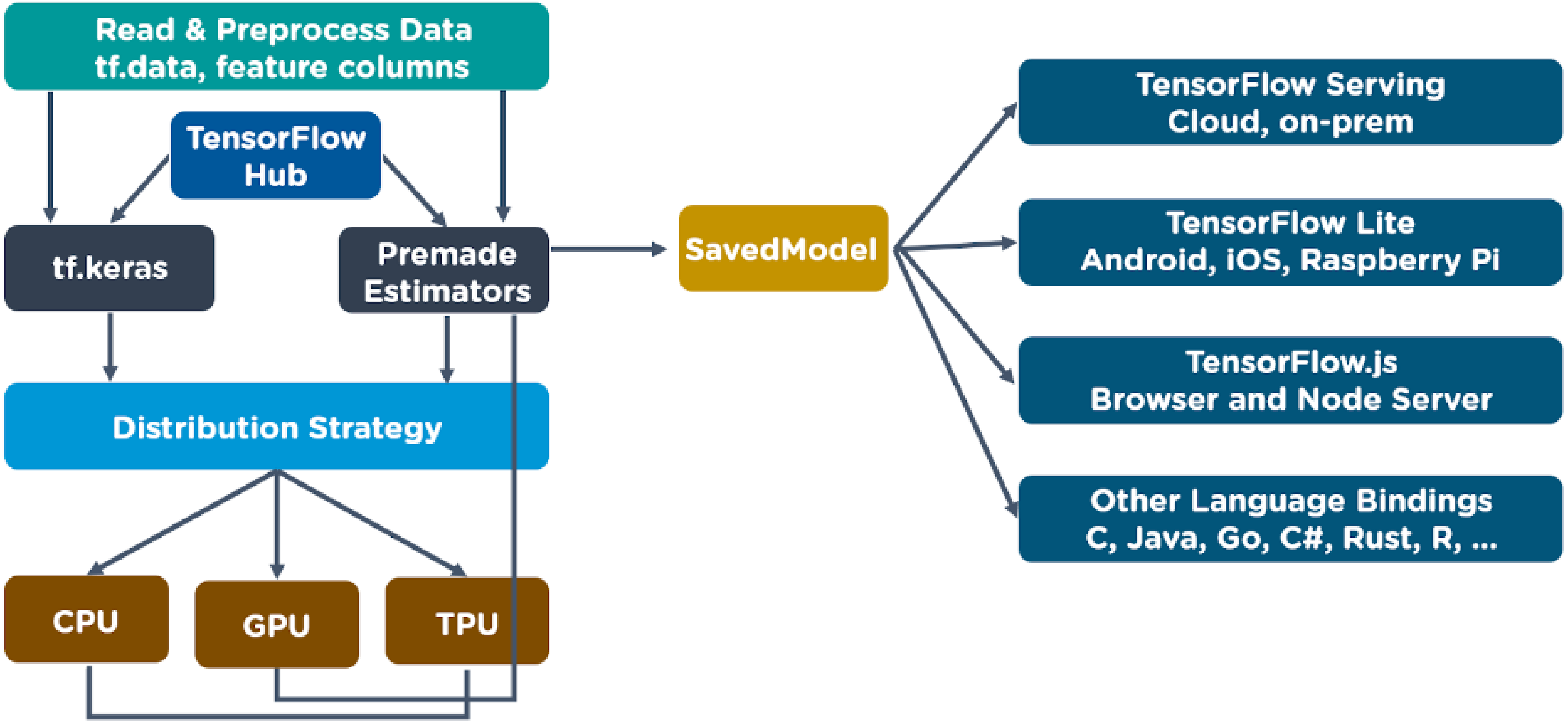
scikit learn

seaborn

pandas

matplotlib

TensorFlow


Read & Preprocess Data
tf.data, feature columns
TensorFlow Hub
tf.keras
Premade Estimators
Distribution Strategy
CPU
GPU
TPU
SavedModel
TensorFlow Serving
Cloud, on-prem
TensorFlow Lite
Android, iOS, Raspberry Pi
TensorFlow.js
Browser and Node Server
Other Language Bindings
C, Java, Go, C#, Rust, R, ...

TensorFlow Installation

CPU: pip install tensorflow

GPU: pip install tensorflow-gpu

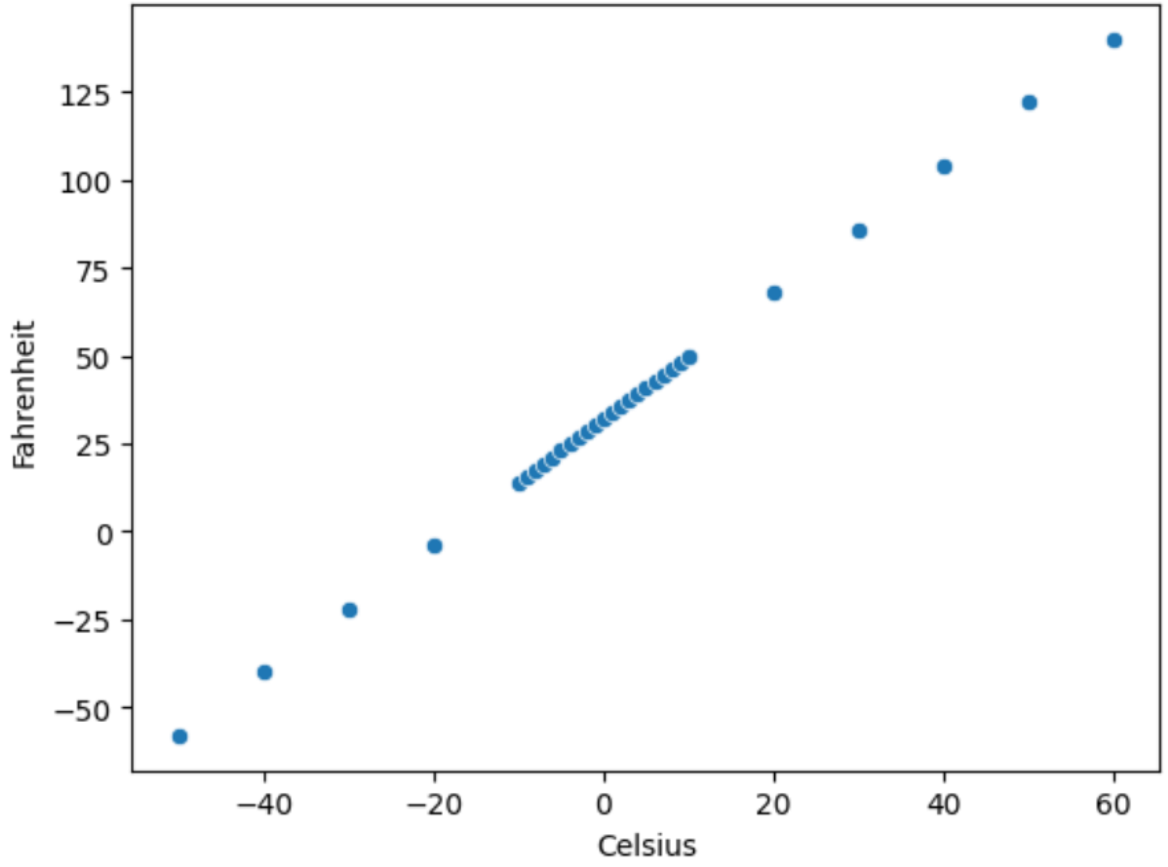
Install by specify the version

!pip install tensorflow-gpu==2.11.0

Example1: Simple Perceptron

From the CSV data, create the model to predict the value of output from desired input using TF.

	Celsius	Fahrenheit
0	-50	-58.0
1	-40	-40.0
2	-30	-22.0
3	-20	-4.0
4	-10	14.0
5	-9	15.8
6	-8	17.6
7	-7	19.4



Example1: Simple Perceptron

	Celsius	Fahrenheit
0	-50	-58.0
1	-40	-40.0
2	-30	-22.0
3	-20	-4.0
4	-10	14.0
5	-9	15.8

Step1: Import Library

```
import tensorflow as tf
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Step2: Read data from CSV

```
df = pd.read_csv("/content/drive/MyDrive/[redacted]ex1_simple_perceptron.csv")
```

Path of CSV File

Example1: Simple Perceptron

Step3: Show statistical parameters.

```
df.describe()
```

	Celsius	Fahrenheit
count	30.000000	30.000000
mean	2.000000	35.600000
std	22.780815	41.005466
min	-50.000000	-58.000000
25%	-6.750000	19.850000
50%	0.500000	32.900000
75%	7.750000	45.950000
max	60.000000	140.000000

Step4: Get data information

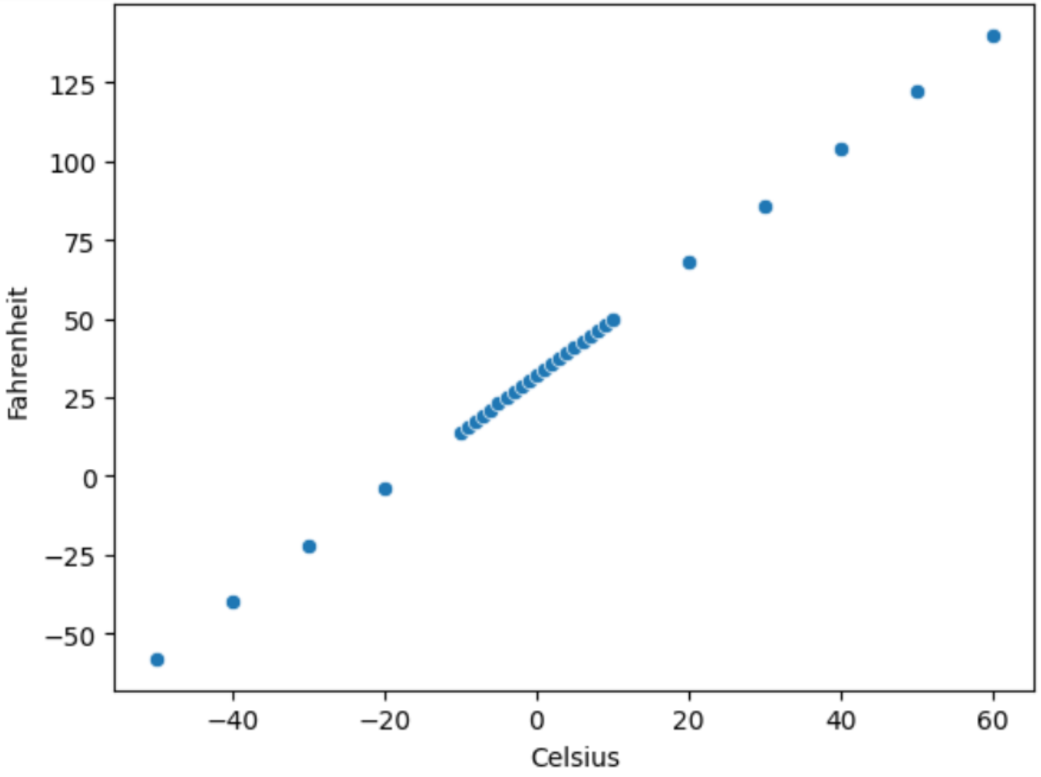
```
df.info()
```

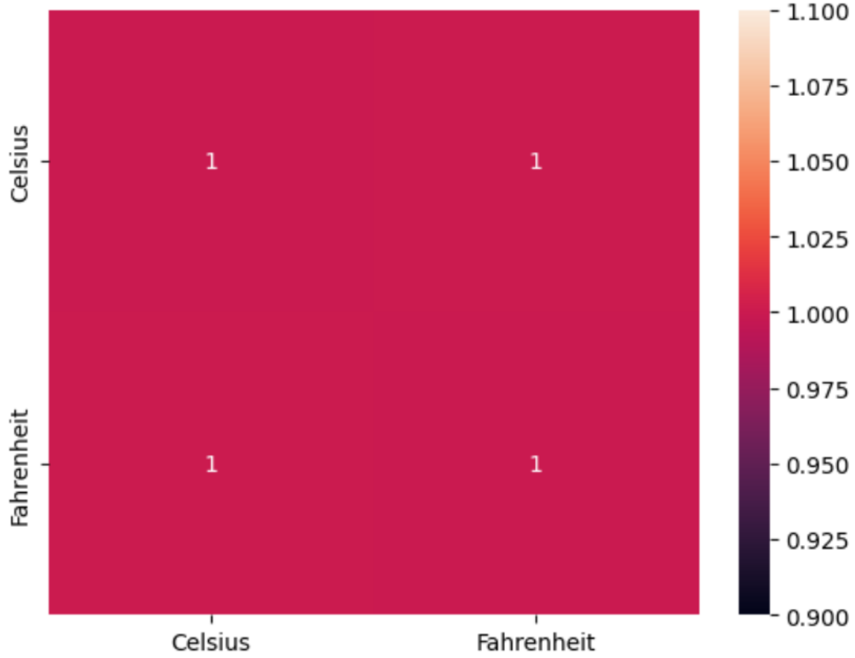
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Celsius     30 non-null     int64  
 1   Fahrenheit  30 non-null     float64
dtypes: float64(1), int64(1)
memory usage: 612.0 bytes
```

Example1: Simple Perceptron

Step5: Visualize dataset.

```
sns.scatterplot(data = df , x = 'Celsius', y = 'Fahrenheit')
```






```
sns.heatmap(df.corr() , annot = True)
```

Example1: Simple Perceptron

Step6: Create Datasets

```
x_train = df['Celsius']
y_train = df['Fahrenheit']
```

Step7: Create Model

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(1, input_shape=[1]))
```

```
model.summary()
```

Why it has 2 parameters?

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 1)	2

```
Total params: 2 (8.00 B)
Trainable params: 2 (8.00 B)
Non-trainable params: 0 (0.00 B)
```

Example1: Simple Perceptron

Step8: Train the model.

```
model.compile(optimizer=tf.keras.optimizers.Adam(0.5), loss='mean_squared_error')
```

Adam = Adaptive Moment Estimation

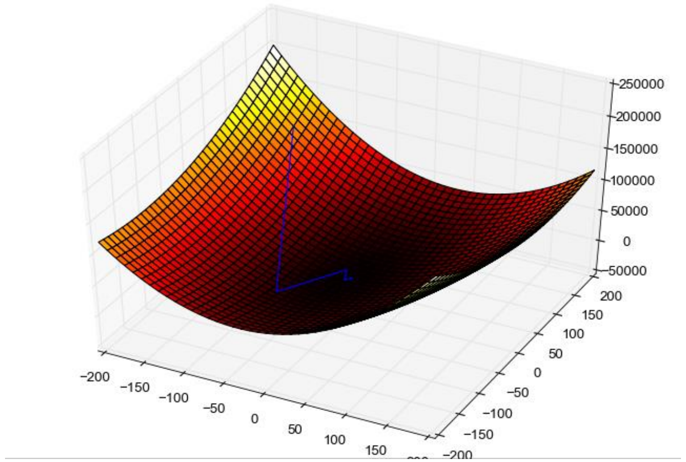
Learning rate

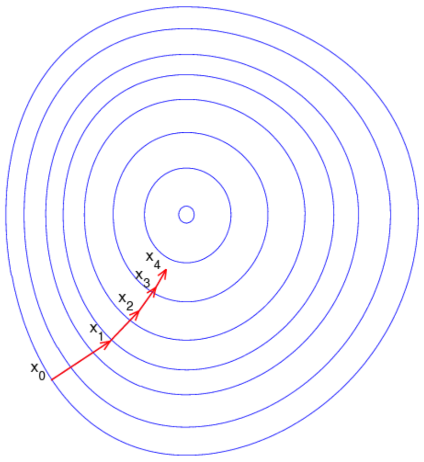
Loss Function

```
epochs_hist = model.fit(x_train, y_train, epochs = 300)
```

Gradient Descent

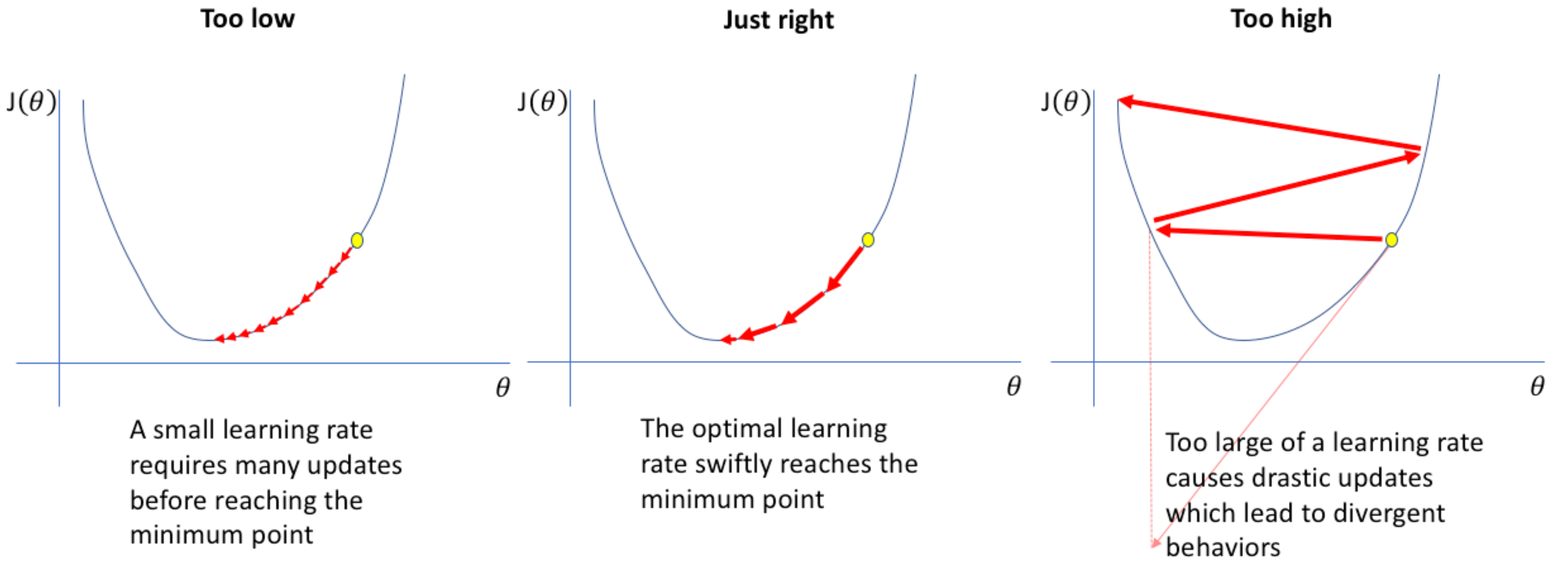
- Gradient descent is an optimization algorithm used to obtain the optimized network weight and bias values
- It works by iteratively trying to minimize the cost function
- It works by calculating the gradient of the cost function and moving in the negative direction until the local/global minimum is achieved
- If the positive of the gradient is taken, local/global maximum is achieved





Learning Rate

- The size of the steps taken are called the learning rate
- If learning rate increases, the area covered in the search space will increase so we might reach global minimum faster
- However, we can overshoot the target
- For small learning rates, training will take much longer to reach optimized weight values



https://www.jeremyjordan.me/nn-learning-rate/

Gradient Descent

Gradient descent works as follows:

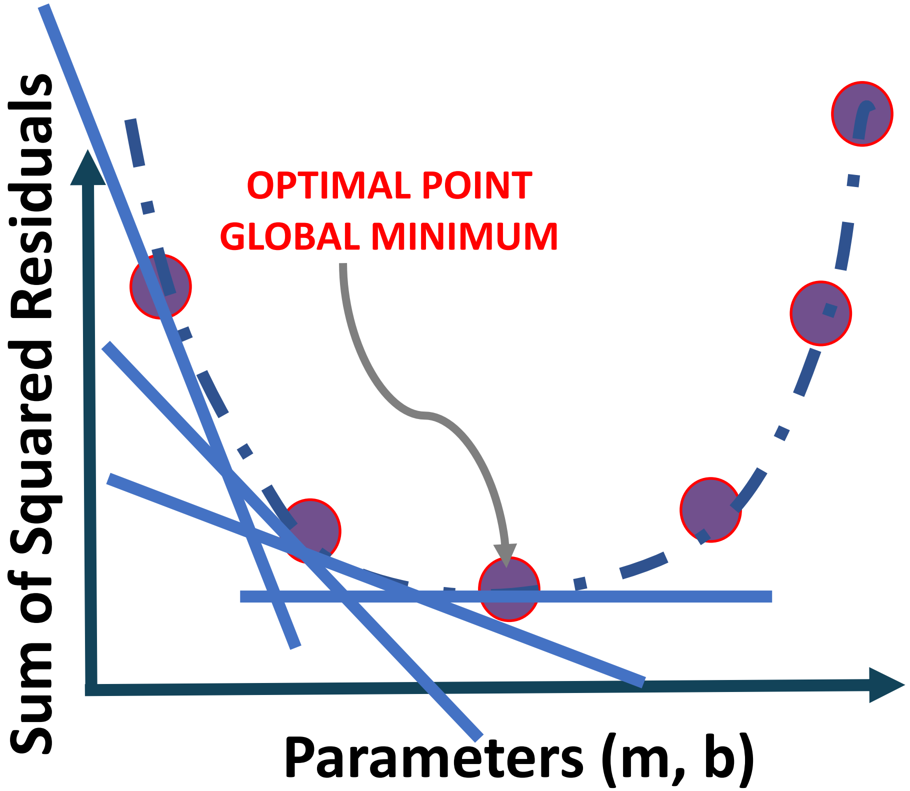
1. Calculate the derivative (gradient) of the Loss function
2. Pick random values for parameters m, b and substitute
3. Calculate the step size (how much are we going to update the parameters?)

$$Step\ size\ =\ Slope\ *\ learning\ rate$$

4. Update the parameters and repeat

$$y = \boxed{b} + \boxed{m} * x$$

GOAL IS TO FIND BEST PARAMETERS



**Note: in reality, this graph is 3D and has three axes, one for m, b and sum of squared residuals*

Gradient Descent

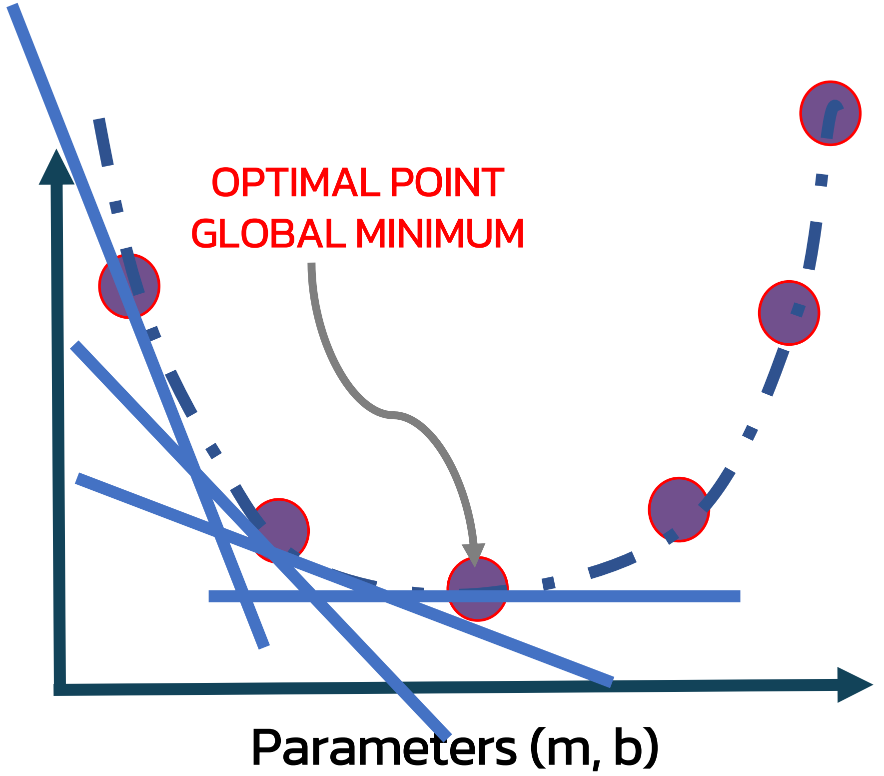
$$y = \boxed{b} + \boxed{m} * x$$

GOAL IS TO FIND
BEST PARAMETERS

$$Loss\ Function\ f(m,b) = \frac{1}{N}\sum_{i=1}^{n}\left(y_i - (b + m * x_i)\right)^2$$

$$gradient\ f'(m,b) = \begin{bmatrix} \frac{df}{dm} \\ \frac{df}{db} \end{bmatrix} = \begin{bmatrix} \frac{1}{N}\sum_{i=1}^{n} -2x_i\left(y_i - (b + m * x_i)\right)^2 \\ \frac{1}{N}\sum_{i=1}^{n} -2\left(y_i - (b + m * x_i)\right)^2 \end{bmatrix}$$

**Note: in reality, this graph is 3D and has three axes, one for m, b and sum of squared residuals*



Mean Square Error

- Mean Square Error (MSE) is very similar to the Mean Absolute Error (MAE) but instead of using absolute values, squares of the difference between the model predictions and the training dataset (true values) is being calculated.
- MSE values are generally **larger** compared to the MAE since the **residuals are being squared**.
- In case of data outliers, MSE will become much larger compared to MAE
- In MSE, error increases in a **quadratic fashion** while the error increases in **proportional fashion in MAE**
- In MSE, since the error is being squared, any predicting error is being heavily penalized
- The MSE is calculated as follows:

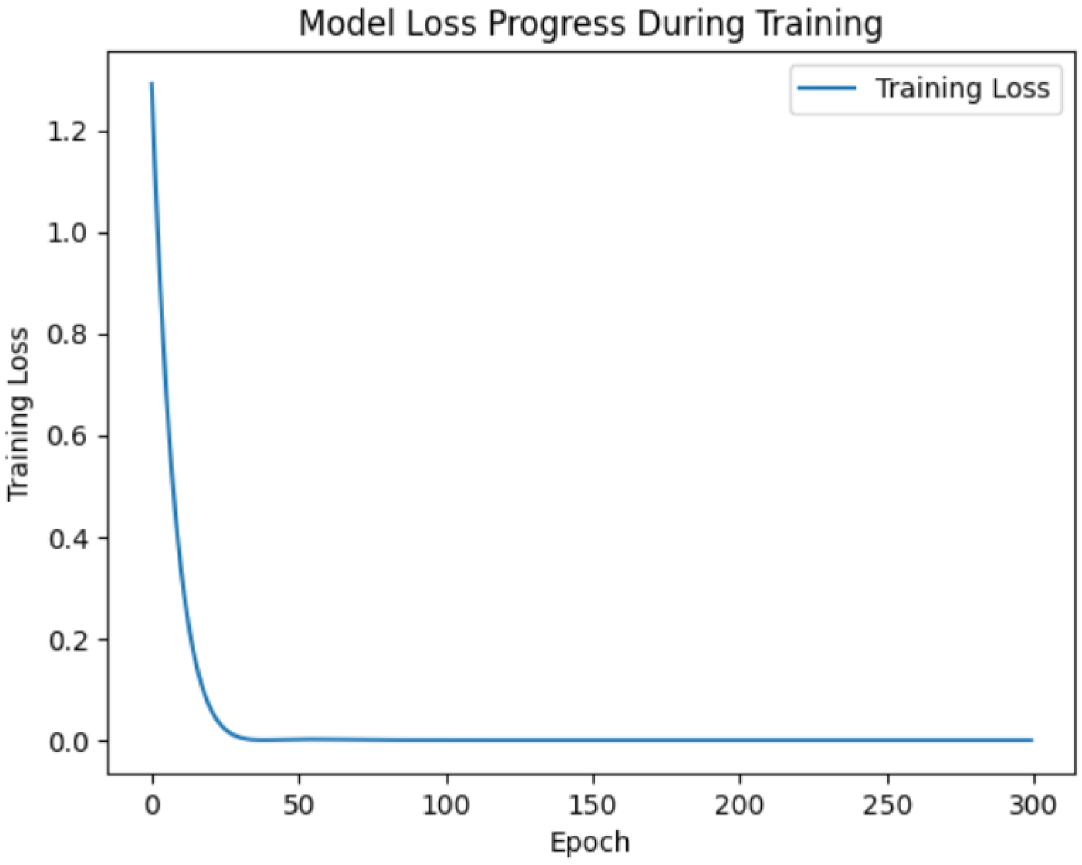
$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Example1: Simple Perceptron

Step9: Visualize training results.

```
epochs_hist.history.keys()
```

```
plt.plot(epochs_hist.history['loss'])
plt.title('Model Loss Progress During Training')
plt.xlabel('Epoch')
plt.ylabel('Training Loss')
plt.legend(['Training Loss'])
```



Example1: Simple Perceptron

Step10: Show model weight

```
model.get_weights()
```

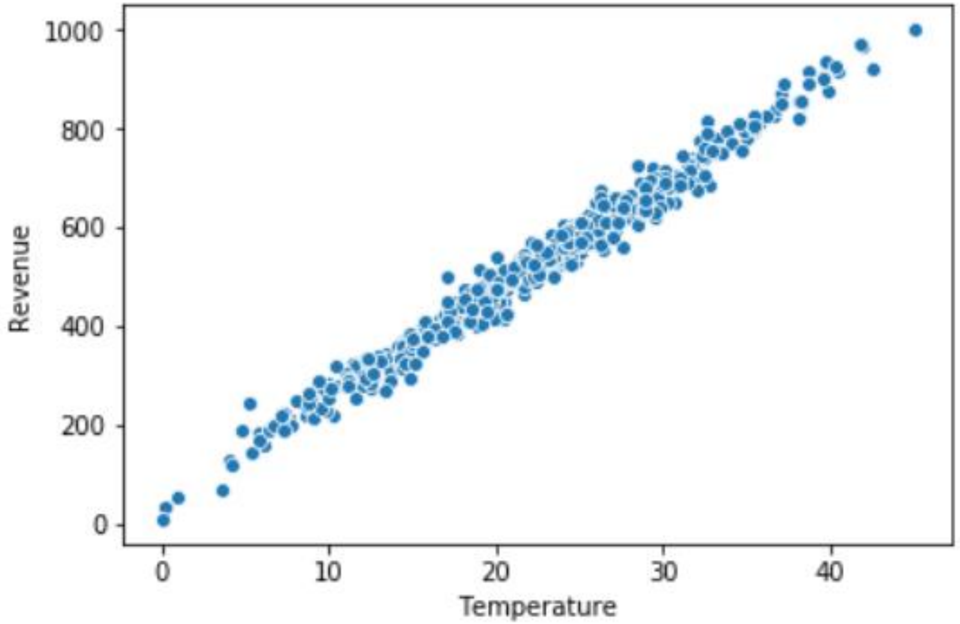
Step11: Test the model

```
f = model.predict(np.array([20]))
print(f)
```

Example2: Ice cream sales prediction

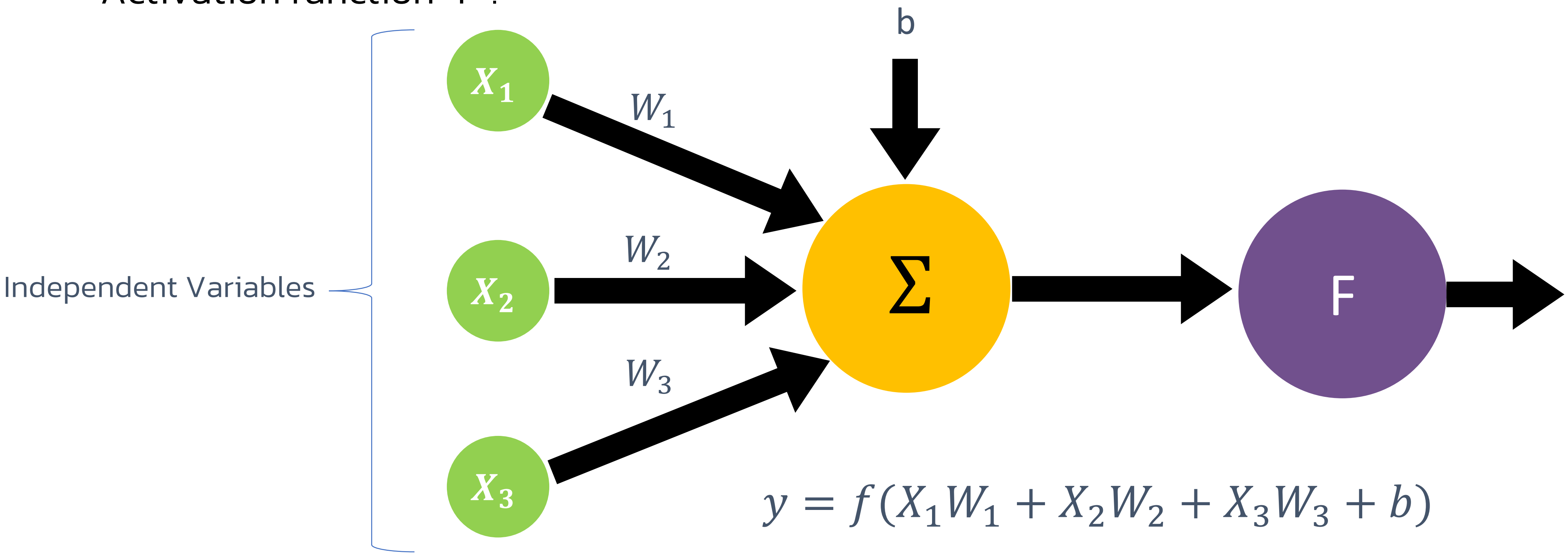
From the CSV data, create the model to predict the sales of ice cream by TF.

	Temperature	Revenue
0	24.566884	534.799028
1	26.005191	625.190122
2	27.790554	660.632289
3	20.595335	487.706960
4	11.503498	316.240194
5	14.352514	367.940744
6	13.707780	308.894518
7	30.833985	696.716640
8	0.976870	55.390338



Multi-Input Neuron Model

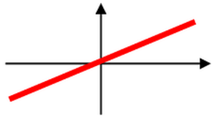
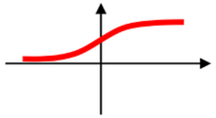
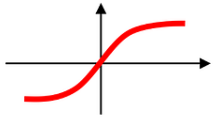
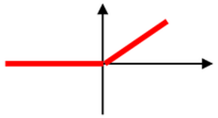
- Bias allows to shift the activation function curve up or down.
- Number of adjustable parameters = 4 (3 weights and 1 bias).
- Activation function “F”.



$$y = f(X_1W_1 + X_2W_2 + X_3W_3 + b)$$

Activation Function

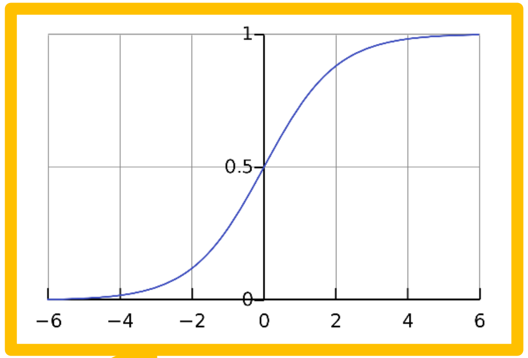
- An activation function in a neural network is a mathematical function that determines whether a neuron should be activated or not based on the weighted sum of its inputs. It introduces non-linearity, allowing the network to learn and model complex patterns beyond simple linear relationships.

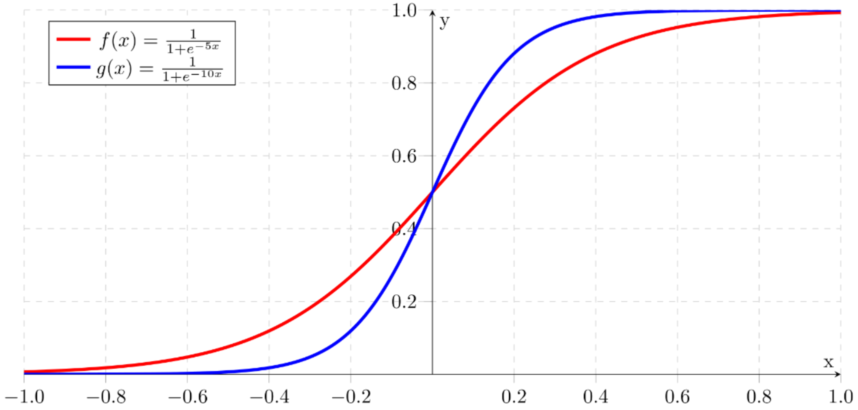
Activation function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent (Tanh)	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	

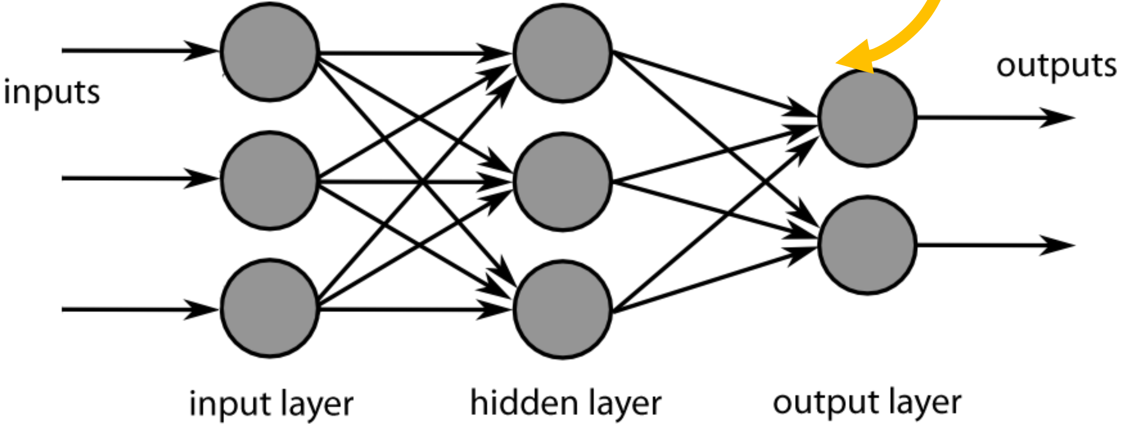
https://medium.com/@BenDosch/ml-activation-functions-f851fd6334d2

SIGMOID Function

- Takes a number and sets it between 0 and 1
- Converts large negative numbers to 0 and large positive numbers to 1.
- Generally used in output layer.



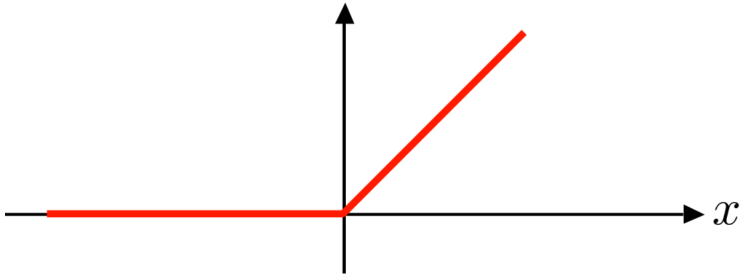


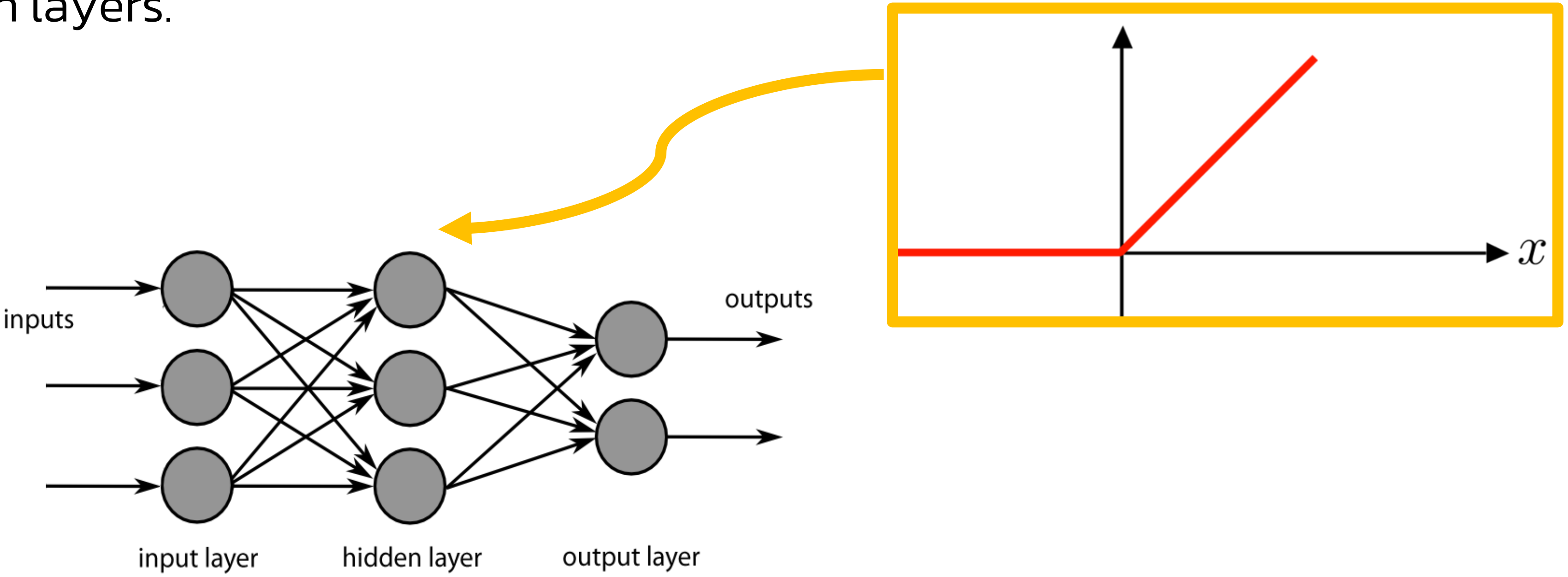


RELU (RECTIFIED LINEAR UNITS) ACTIVATION FUNCTION

- if input x < 0, output is 0 and if x > 0 the output is x.
- RELU does not saturate so it avoids vanishing gradient problem.
- It uses simple thresholding so it is computationally efficient.
- Generally used in hidden layers.

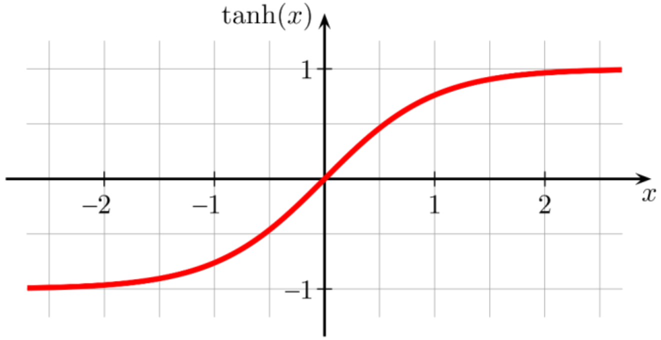
$$\mathrm{ReLU}(x) \triangleq \max(0, x)$$

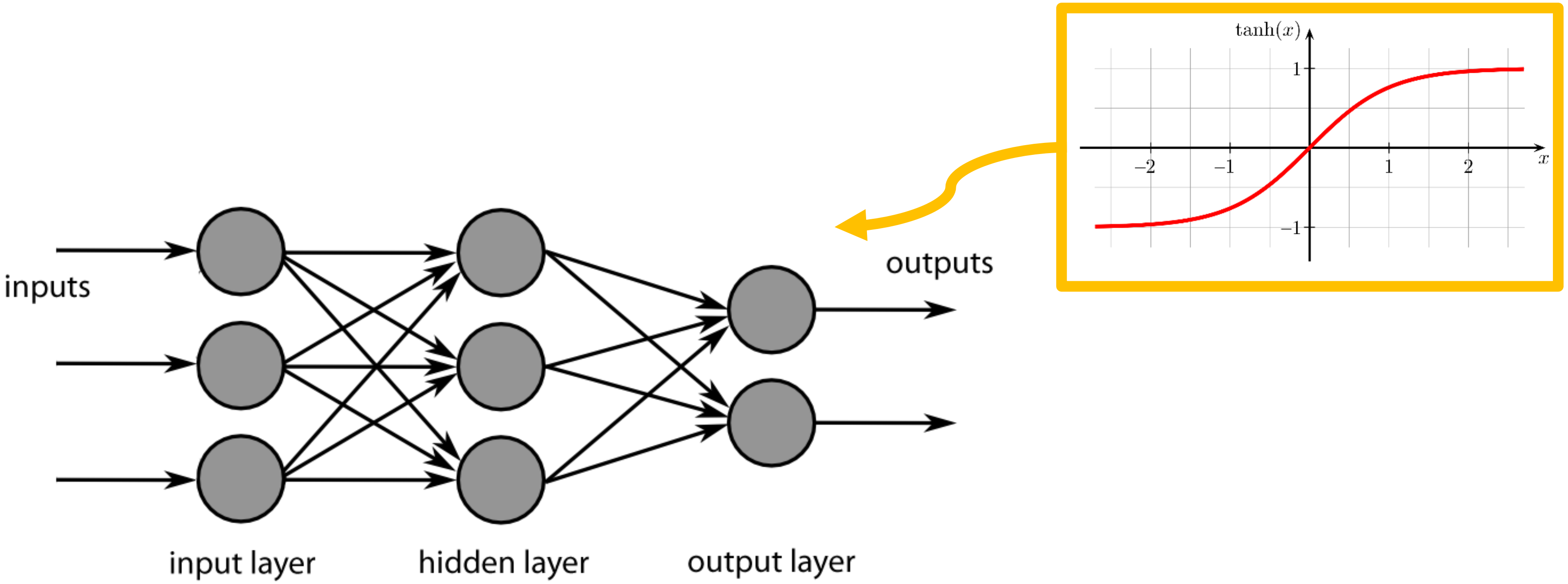




HYPERBOLIC TANGENT ACTIVATION FUNCTION

- "Tanh" is similar to sigmoid, converts number between -1 and 1.
- Unlike sigmoid, tanh outputs are zero-centered (range: -1 and 1).
- Tanh suffers from vanishing gradient problem so it kills gradients when saturated.
- In practice, tanh is preferable over sigmoid.



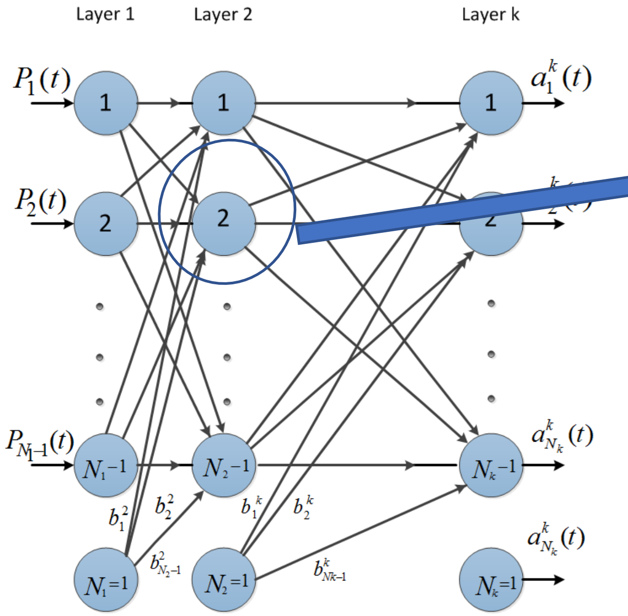


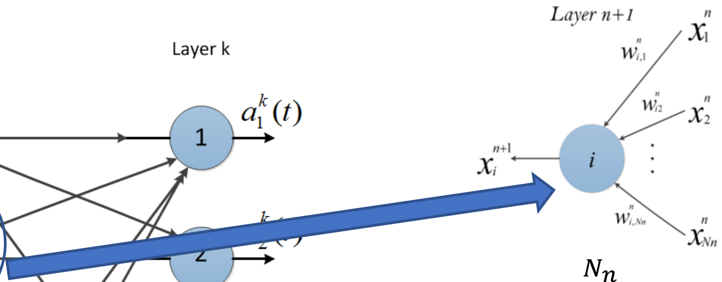
ARTIFICIAL NEURAL NETWORK REGRESSION

- The more hidden layers, the more "deep" the network will get.

$$P = \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_{N_1} \end{bmatrix}$$

$$\begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1,N_1} \\ W_{21} & W_{22} & & W_{2,N_1} \\ \vdots & & \ddots & \vdots \\ W_{m-1,1} & W_{m-1,2} & \cdots & W_{m-1,N_1} \\ W_{m,1} & W_{m,2} & & W_{m,N_1} \end{bmatrix}$$





$$x_i^{n+1}(t) = \varphi\left(\sum_{j=1}^{N_n} w_{i,j}^n \, x_j^n(t)\right)$$

Node *(n+1, i)* representation

Non-Linear Sigmoid Activation function

$$\varphi(w) = \frac{1}{1 + e^{-w}}$$

m: number of neurons in the hidden layer

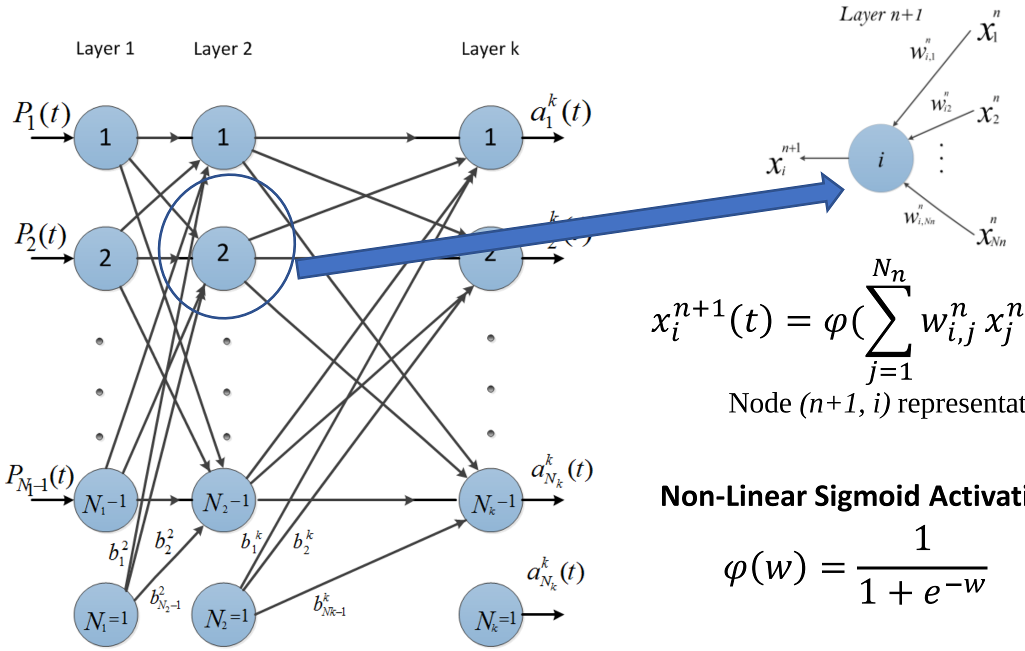
N_1*: number of inputs*

ARTIFICIAL NEURAL NETWORK REGRESSION

- The more hidden layers, the more “deep” the network will get.

$$P = \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_{N_1} \end{bmatrix}$$

$$\begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1,N_1} \\ W_{21} & W_{22} & & W_{2,N_1} \\ \vdots & & \ddots & \vdots \\ W_{m-1,1} & W_{m-1,2} & \cdots & W_{m-1,N_1} \\ W_{m,1} & W_{m,2} & & W_{m,N_1} \end{bmatrix}$$



$$x_i^{n+1}(t) = \varphi\left(\sum_{j=1}^{N_n} w_{i,j}^n \, x_j^n(t)\right)$$

Node *(n+1, i)* representation

Non-Linear Sigmoid Activation function

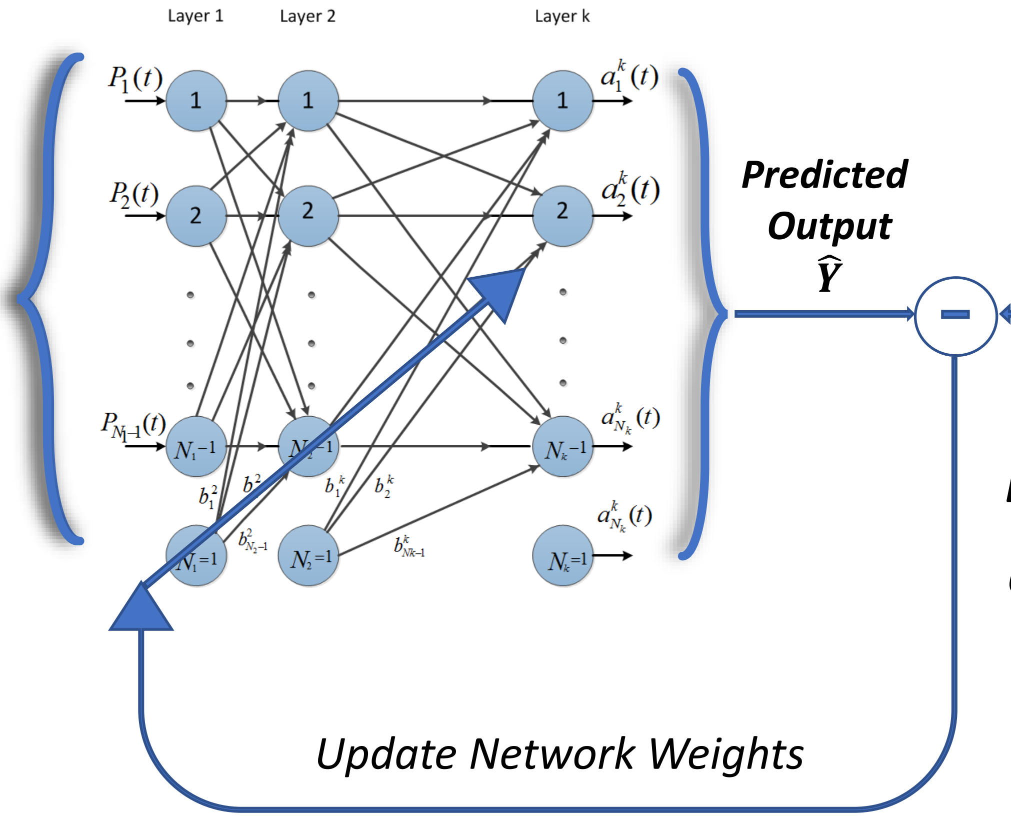
$$\varphi(w) = \frac{1}{1 + e^{-w}}$$

m: number of neurons in the hidden layer

N_1: number of inputs

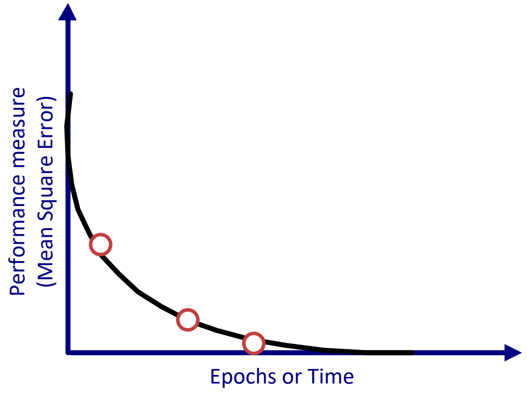
ARTIFICIAL NEURAL NETWORK REGRESSION

Training inputs X



Predicted Output $\hat{Y}$

Desired (True) Output Y

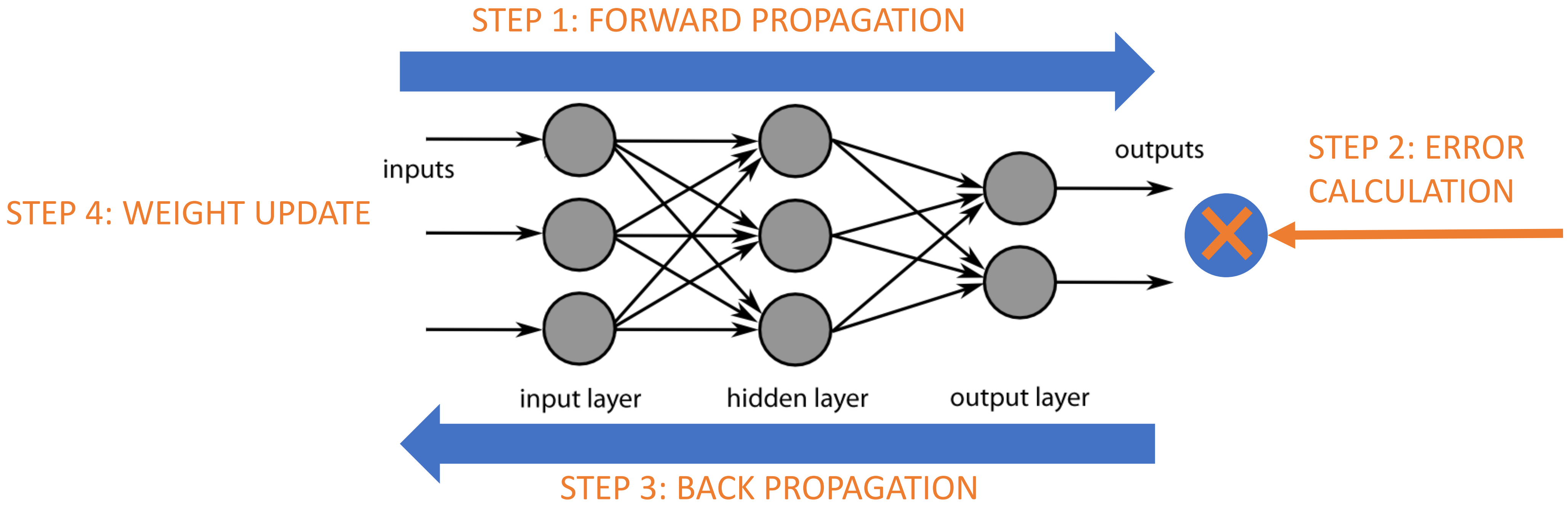


Error = $\hat{Y} - Y$

$$C = ½(\hat{Y} - Y)^2$$

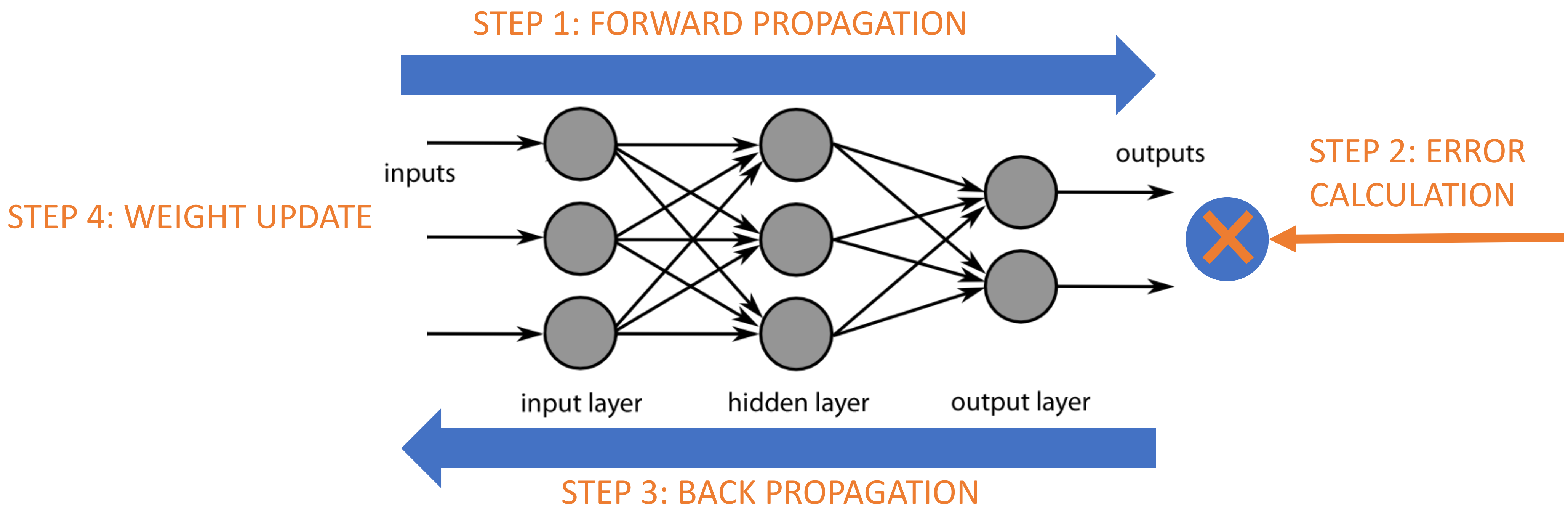
BACK PROPAGATION

- Backpropagation is a method used to train ANNs by calculating gradient needed to update network weights.
- It is commonly used by the gradient descent optimization algorithm to adjust the weight of neurons by calculating the gradient of the loss function.



BACK PROPAGATION

- Backpropagation Phase 1: propagation
  - Propagation forward through the network to generate the output value(s)
  - Calculation of the cost (error term)
  - Propagation of output activations back through network using training pattern target in order to generate the deltas (difference between targeted and actual output values)

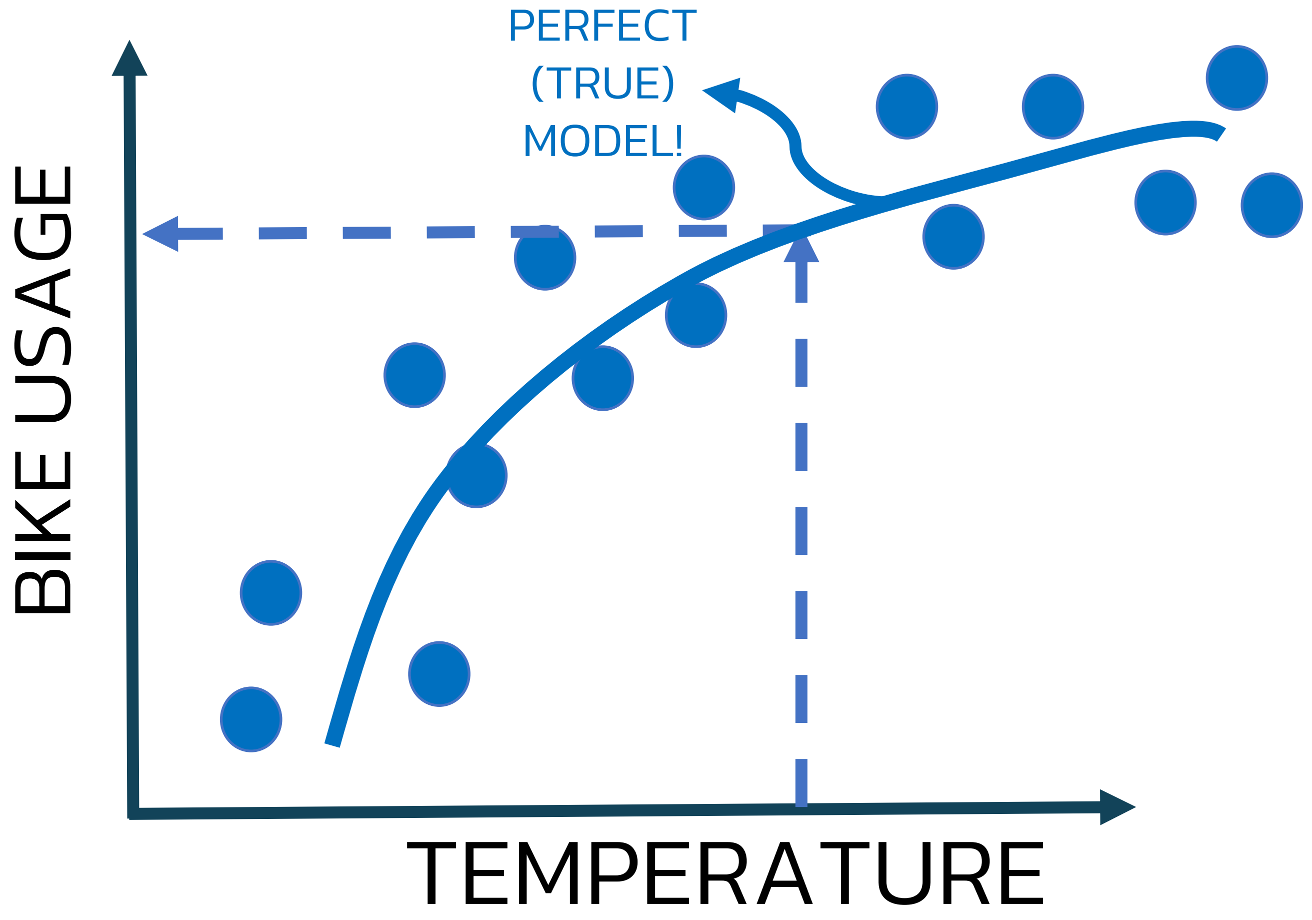


BACK PROPAGATION

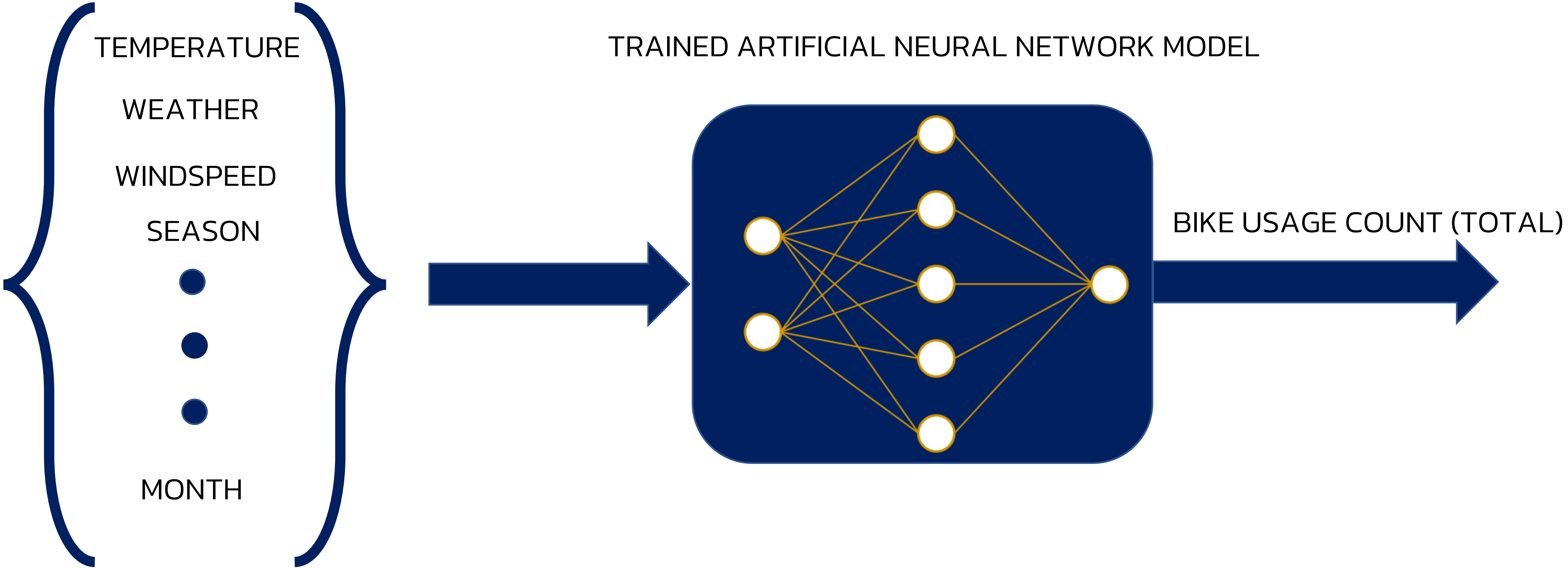
- Phase 2: weight update
  - Calculate weight gradient.
  - A ratio (percentage) of the weight's gradient is subtracted from the weight.
  - This ratio influences the speed and quality of learning and called learning rate. The greater the ratio, the faster neuron train, but lower ratio, more accurate the training is.

Example3: Bike Sharing Daily

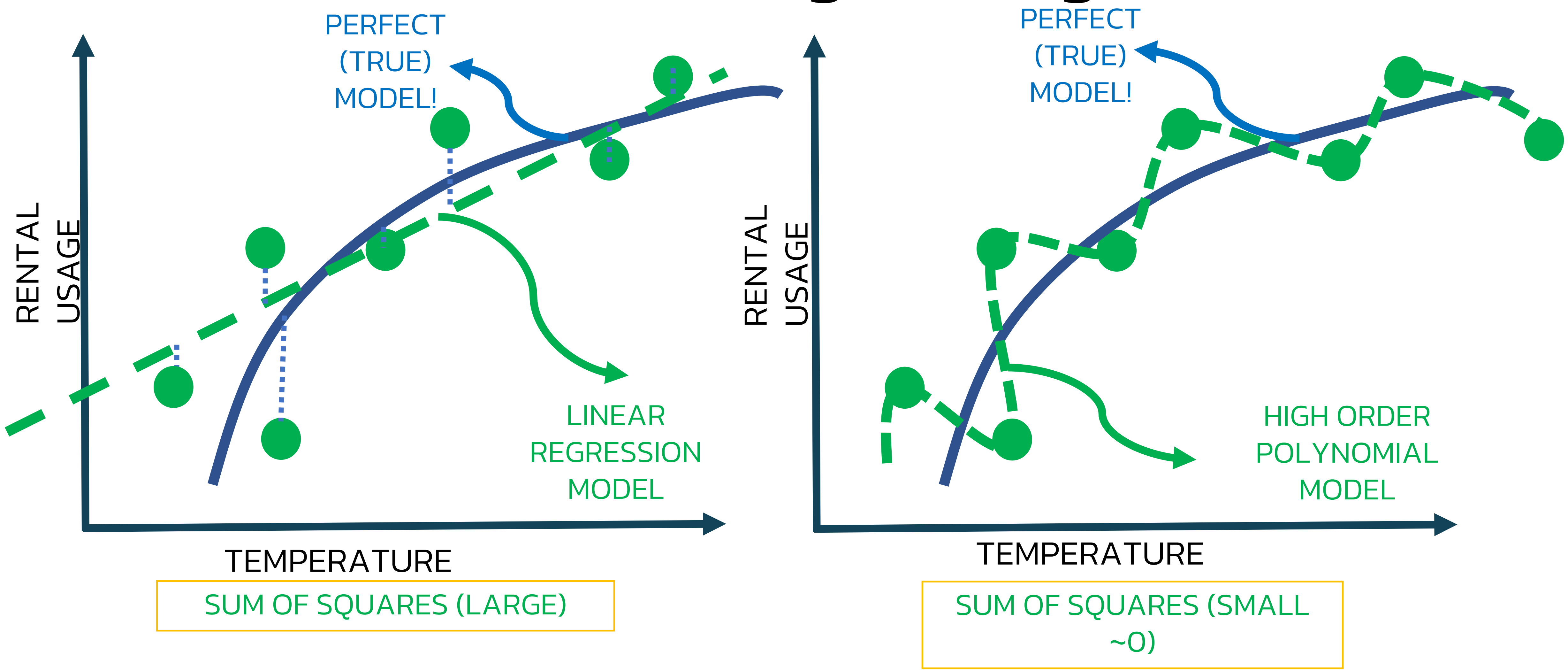
- Read data from provided csv and train the model to predict the bike sharing daily.
- As temperature experience increase, the bike rental usage tend to increase as well.
- As temperature goes beyond a certain limit, usage tend to plateau and it does not increase anymore



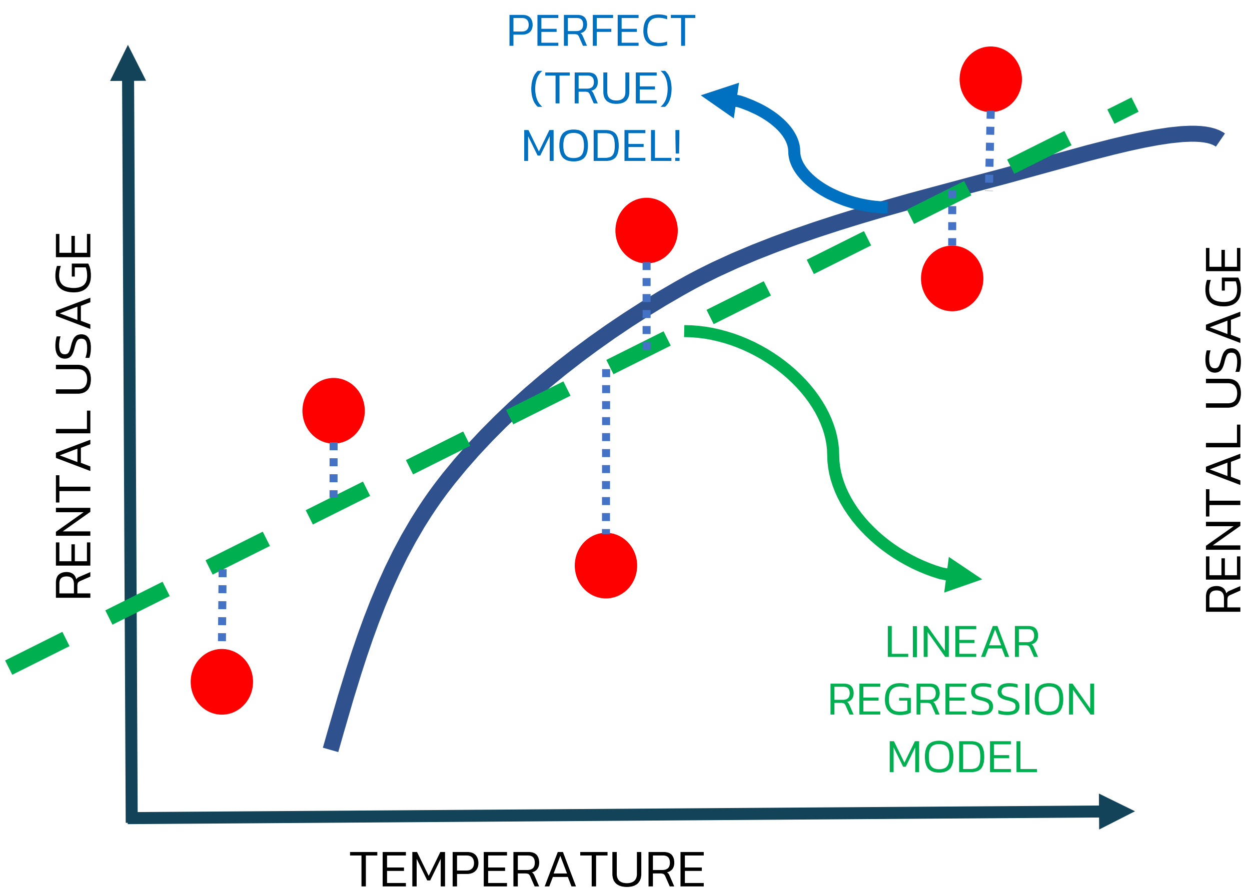
Example3: Bike Sharing Daily



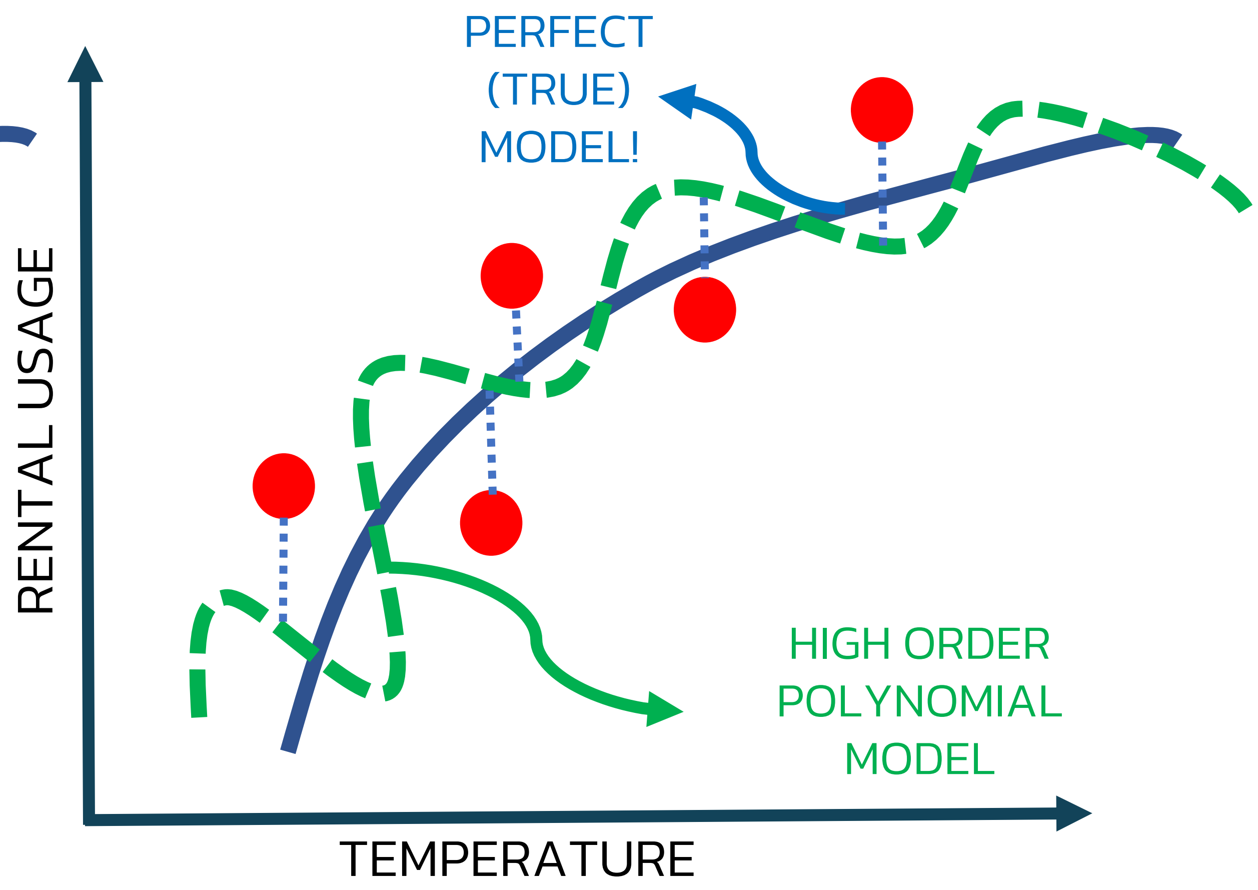
Bias and Variance during training



Bias and Variance during testing



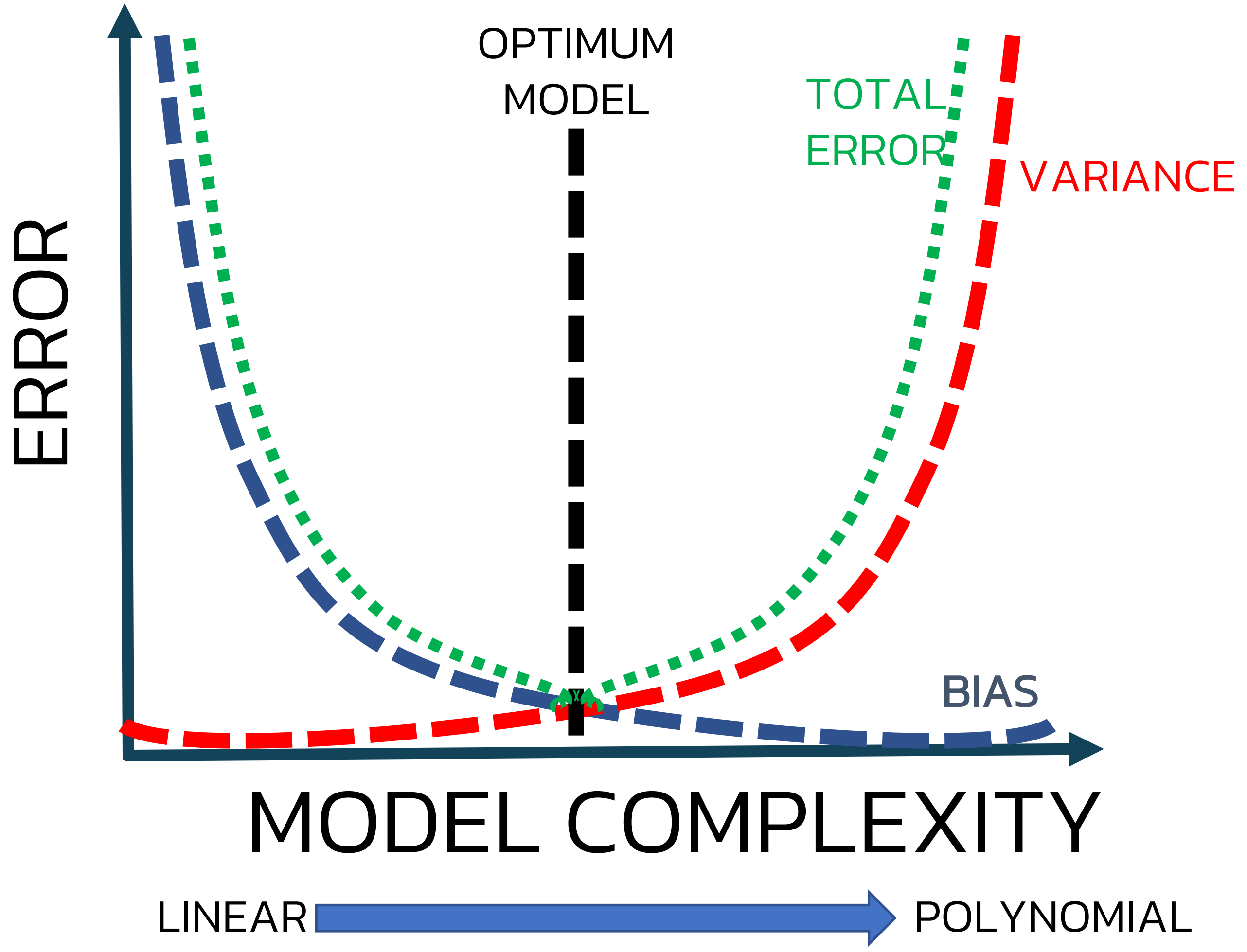
SUM OF SQUARES (SMALL)



SUM OF SQUARES (LARGE)

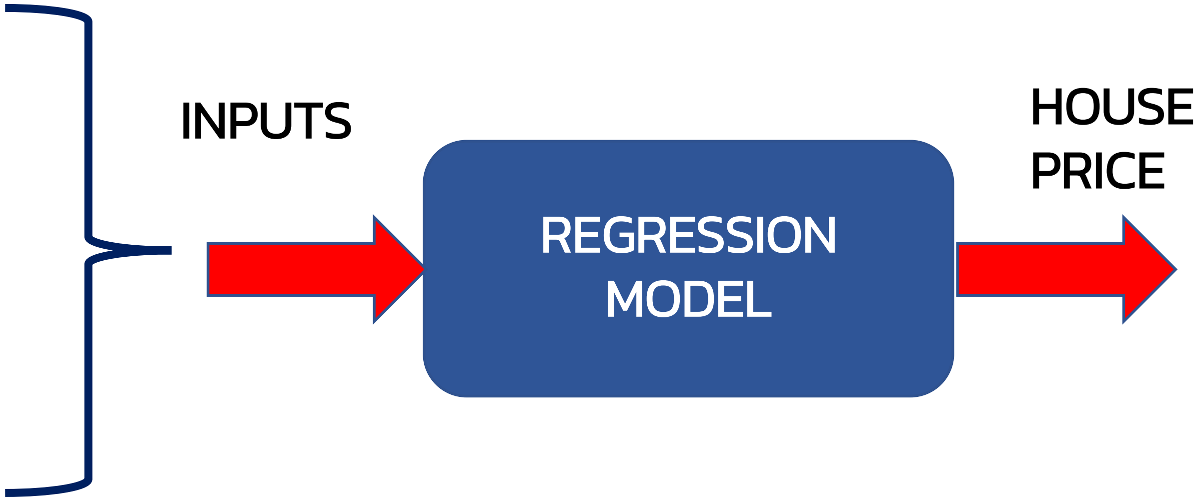
Complexity vs. Error

- Regularization works by reducing the variance at the cost of adding some bias to the model.
- A trade-off between variance and bias occurs



Example4: House prices prediction

- Read data from provided csv file and create the model to predict the house prices using TF.
- Dataset includes house sale prices for King County in Washington, USA.
- Homes that are sold in the time period: May, 2014 and May, 2015.
- Data Source: https://www.kaggle.com/harlfoxem/housesalesprediction
- Model inputs:
  - ida: notation for a house
  - date: Date house was sold
  - bedrooms: Number of Bedrooms
  - bathrooms: Number of bathrooms
  - sqft_living: home square footage
  - sqft_lot: square footage of the lot
  - floors: Total floors (levels) in house
  - waterfront: waterfront property



Example4: House prices prediction

- Model inputs:
  - condition: How good the condition is (Overall)
  - grade: overall grade given to housing unit, based on King County grading system
  - sqft_abovesquare: footage of house apart from basement
  - sqft_basement: square footage of the basement
  - yr_built: Built Year
  - yr_renovated: Year when house was renovated
  - zipcode: zip
  - lat: Latitude coordinate
  - long: Longitude coordinate
  - sqft_living15: Living room area in 2015
  - sqft_lot15: lotSize area in 2015(implies-- some renovations)
- The model should predict:
  - House Price

Example4: House prices prediction

BEDROOMS

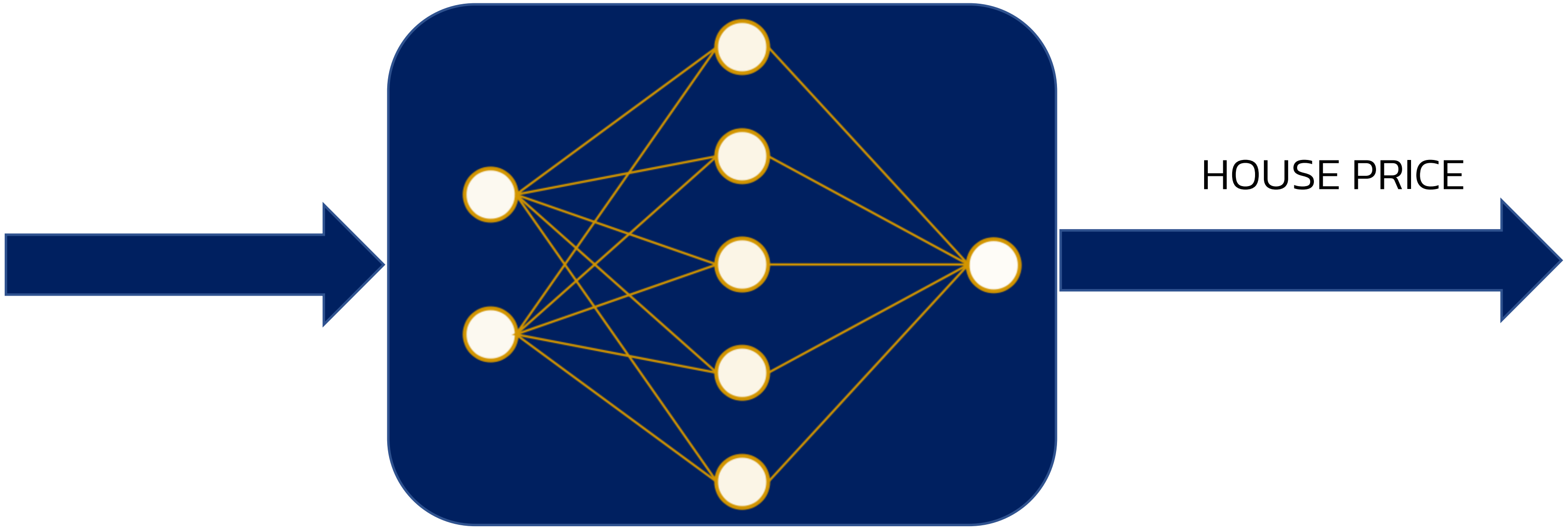
BATHROOMS

FLOORS

WATERFRONT

SQFT LIVING

TRAINED ARTIFICIAL NEURAL NETWORK MODEL

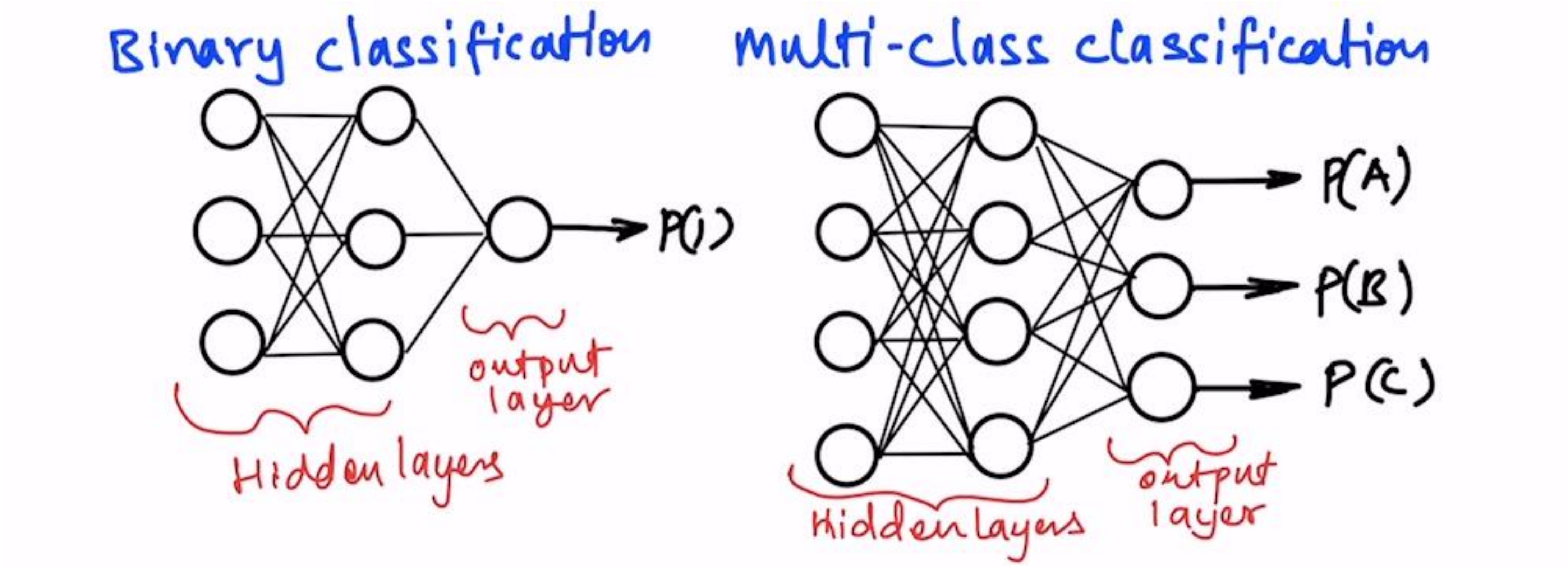


HOUSE PRICE

Practice

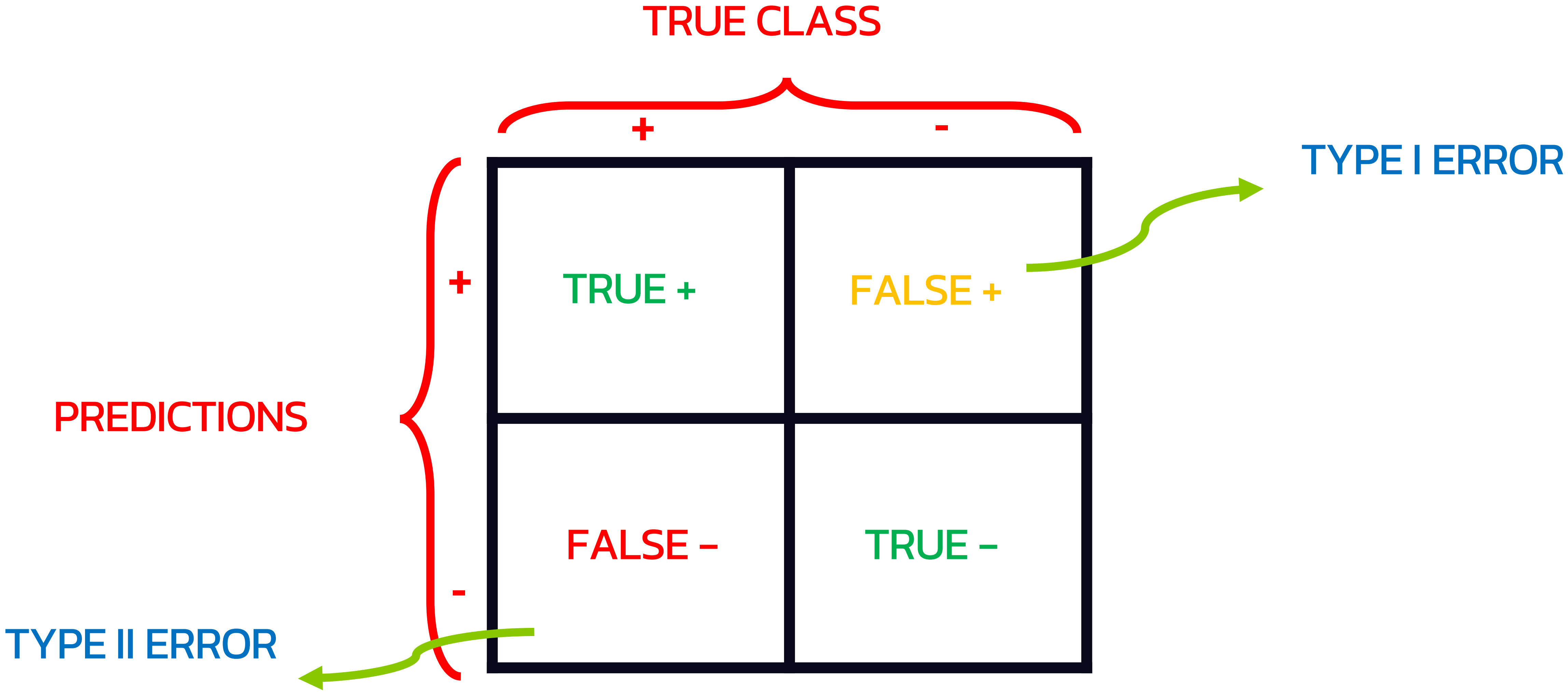
- From provided dataset of material properties and quality rating in manufacturing process, create the model to predict the quality rating using the ANNs.

ARTIFICIAL NEURAL NETWORK CLASSIFICATION



https://thinkingneuron.com/how-to-use-artificial-neural-networks-for-classification-in-python/

Confusion Matrix



Confusion Matrix

A confusion matrix is used to describe the performance of a classification model:

- True positives (TP): cases when classifier predicted TRUE (they have the disease), and correct class was TRUE (patient has disease).
- True negatives (TN): cases when model predicted FALSE (no disease), and correct class was FALSE (patient do not have disease).
- False positives (FP) (Type I error): classifier predicted TRUE, but correct class was FALSE (patient did not have disease).
- False negatives (FN) (Type II error): classifier predicted FALSE (patient do not have disease), but they actually do have the disease

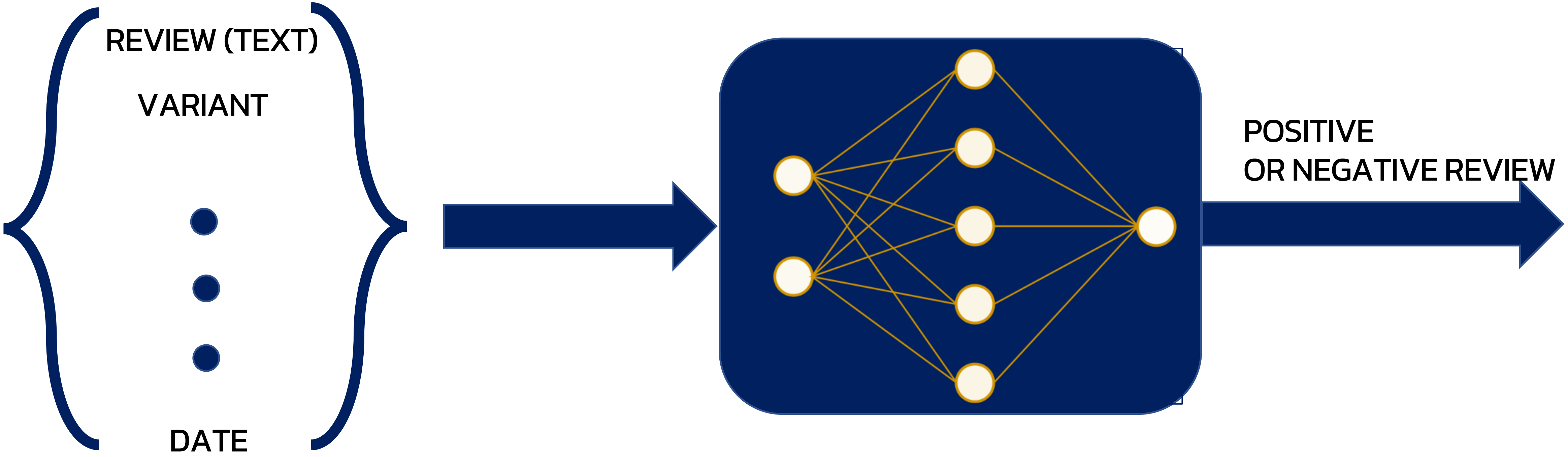
Confusion Matrix

- Classification Accuracy = (TP+TN) / (TP + TN + FP + FN)
- Misclassification rate (Error Rate) = (FP + FN) / (TP + TN + FP + FN)
- Precision = TP/Total TRUE Predictions = TP/ (TP+FP) (When model predicted TRUE class, how often was it right?)
- Recall = TP/ Actual TRUE = TP/ (TP+FN) (when the class was actually TRUE, how often did the classifier get it right?)

Example5: Text Classification

- Read data from provided csv file and create the model to classify type of customer reviews using TF.
- Dataset consists of 3000 Amazon customer reviews, star ratings, date of review, variant and feedback of various amazon Alexa products like Alexa Echo, Echo dots.
- The objective is to discover insights into consumer reviews and perform sentiment analysis on the data.
- Dataset: www.kaggle.com/sid321axn/amazon-alexa-reviews

Example5: Text Classification



TOKENIZATION (COUNT VECTORIZER)

Tokenization is a fundamental process in Natural Language Processing (NLP) that involves breaking down a stream of text into smaller units called tokens. These tokens can range from individual characters to full words or phrases, depending on the level of granularity required. By converting text into these manageable chunks, machines can more effectively analyze and understand human language.

This is the first document.
This document is the second document.
And this is the third one.
Is this the first document?

→

[[0 1 1 1 0 0 1 0 1]
[0 2 0 1 0 1 1 0 1]
[1 0 0 1 1 0 1 1 1]
[0 1 1 1 0 0 1 0 1]]

	'and'	'document'	'first'	'is'	'one'	'second'	'the'	'third'	'this'
Training Sample #1	0	1	1	1	0	0	1	0	1
Training Sample #2	0	2	0	1	0	1	1	0	1
Training Sample #3	1	0	0	1	1	0	1	1	1
Training Sample #4	0	1	1	1	0	0	1	0	1

TOKENIZATION (COUNT VECTORIZER)

Count Vectorization

```
from sklearn.feature_extraction.text import CountVectorizer
sample_data = ['This is the first document.',
               'This document is the second document.',
               'And this is the third one.',
               'Is this the first document?']

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(sample_data)
```

Get Key

```
print(vectorizer.get_feature_names())
```

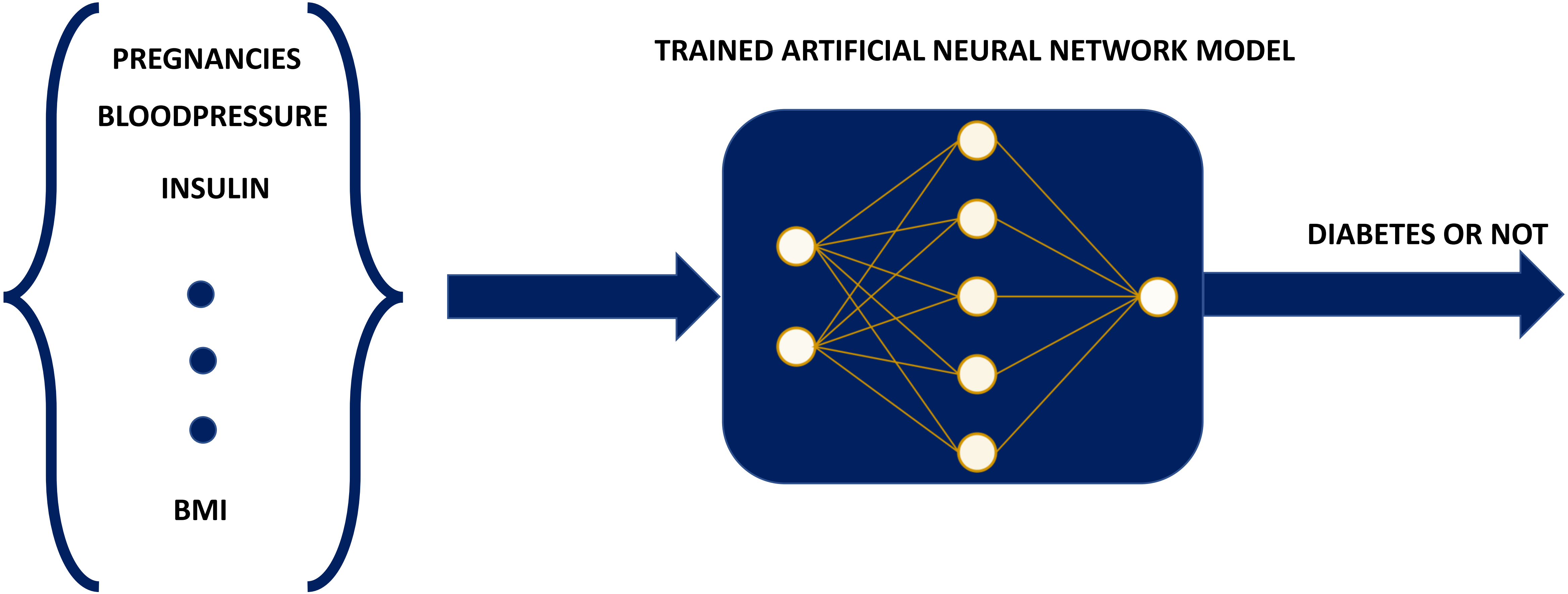
Get Array

```
print(X.toarray())
```

Example6: Diabetes Classification

- This dataset is used to predict whether or not a patient has diabetes, based on given features/diagnostic measurements.
- Only female patients are considered with at least 21 years old of Pima Indian heritage.
- INPUTS:
    - Pregnancies: Number of times pregnant
    - GlucosePlasma: glucose concentration 2 hours in an oral glucose tolerance test
    - BloodPressure: Diastolic blood pressure (mm Hg)
    - Skin: ThicknessTriceps skin fold thickness (mm)
    - Insulin: 2-Hour serum insulin (mu U/ml)
    - BMI: Body mass index (weight in kg/(height in m)^2)
    - DiabetesPedigreeFunction: Diabetes pedigree function
    - Age: Age (years)

Example6: Diabetes Classification



Practice

- From provided Wafer manufacturing data, create ANNs model to detect anomaly in the process by TF.

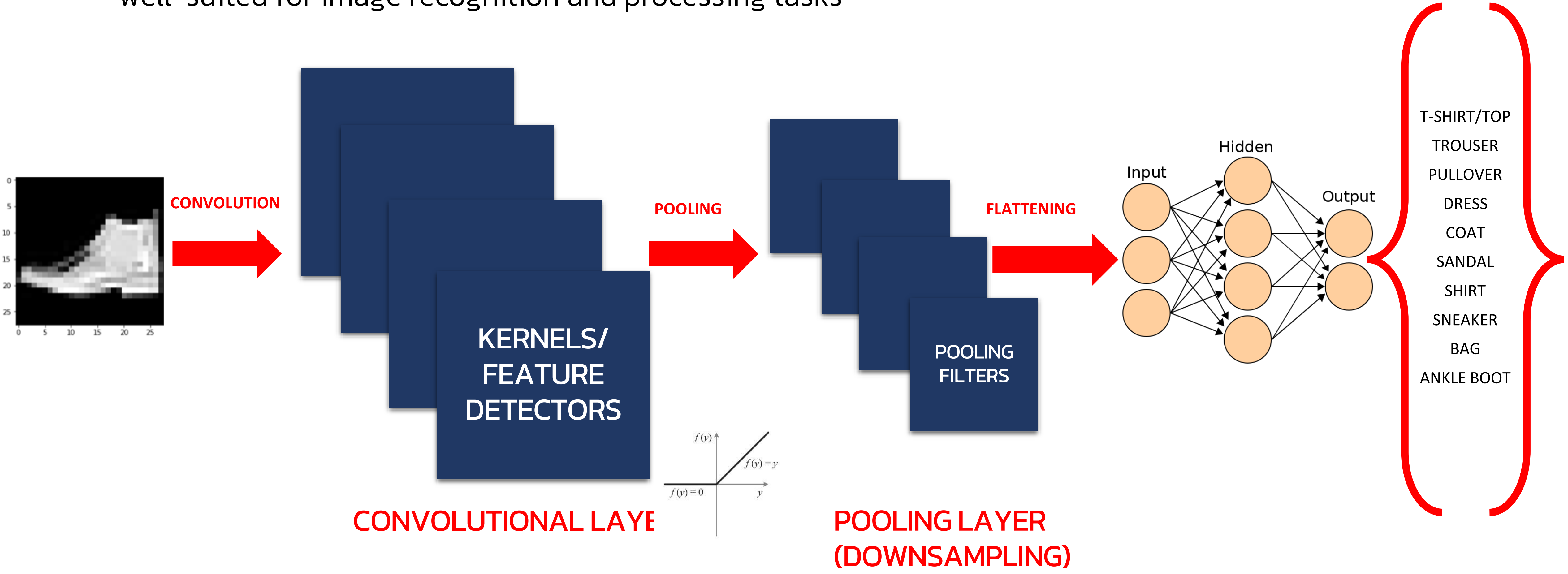
Mechatronics

Day10

Convolutional Neural Networks (CNNs)

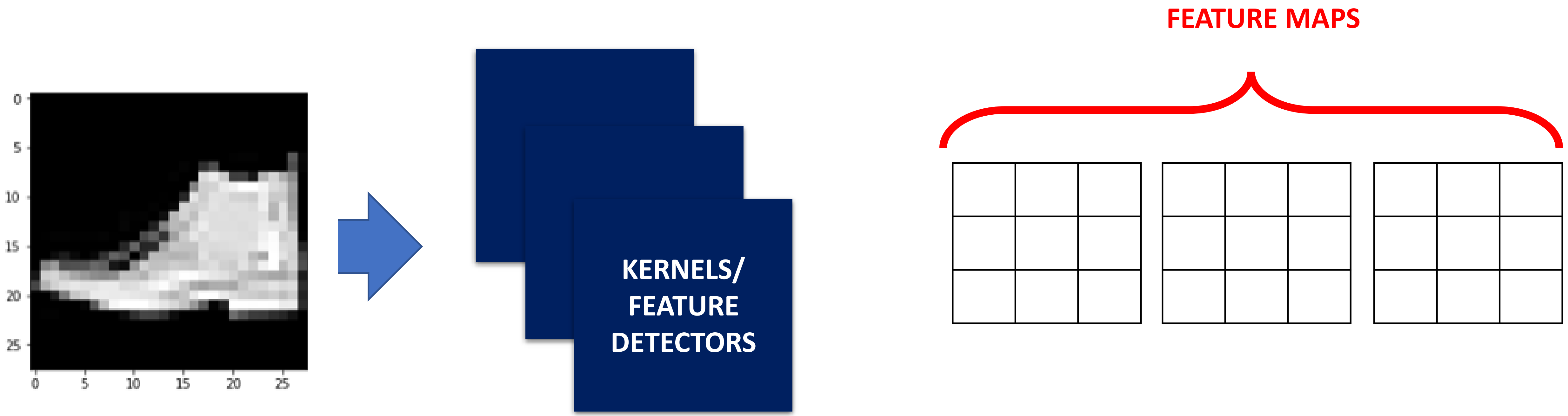
Overview

A Convolutional Neural Network (CNN) is a type of deep learning algorithm that is particularly well-suited for image recognition and processing tasks

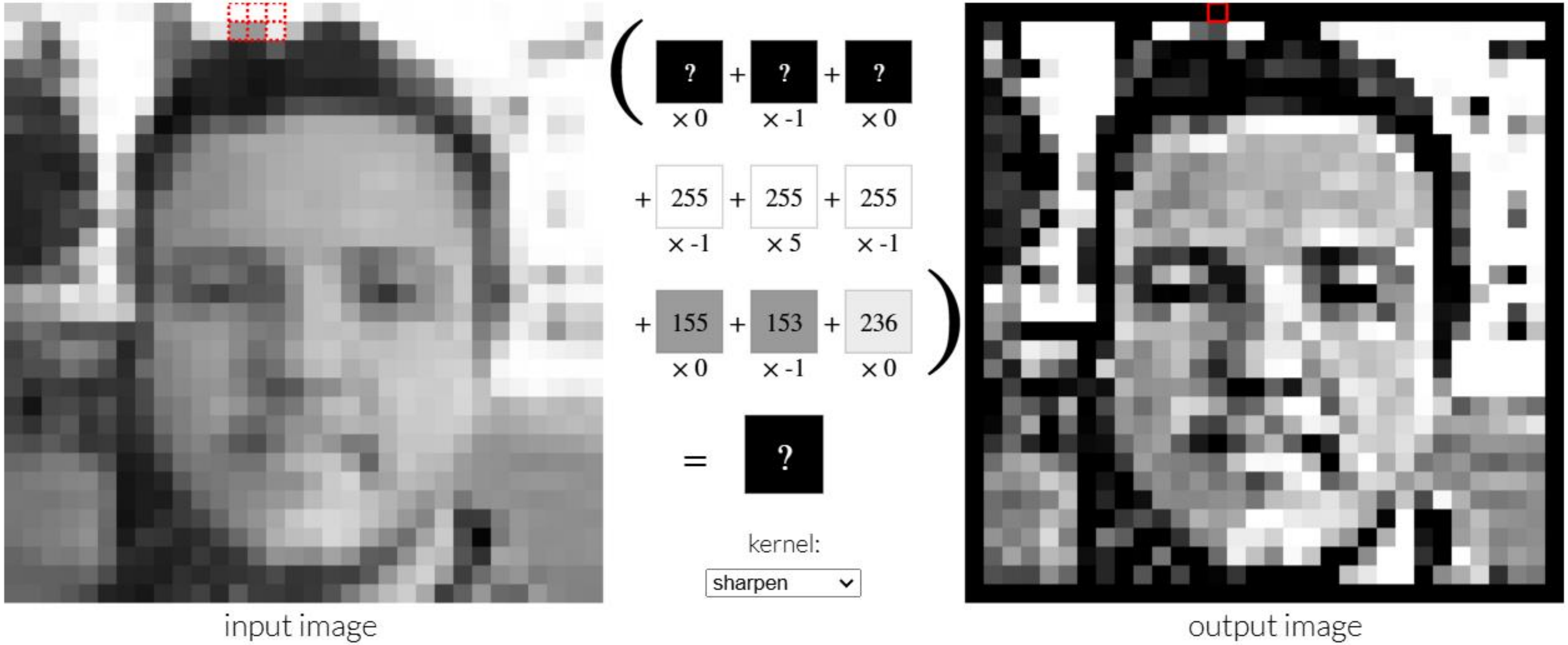


Feature Detectors

- Convolutions use a kernel matrix to scan a given image and apply a filter to obtain a certain effect.
- An image Kernel is a matrix used to apply effects such as blurring and sharpening.
- Kernels are used in machine learning for feature extraction to select most important pixels of an image.
- Convolution preserves the spatial relationship between pixels.



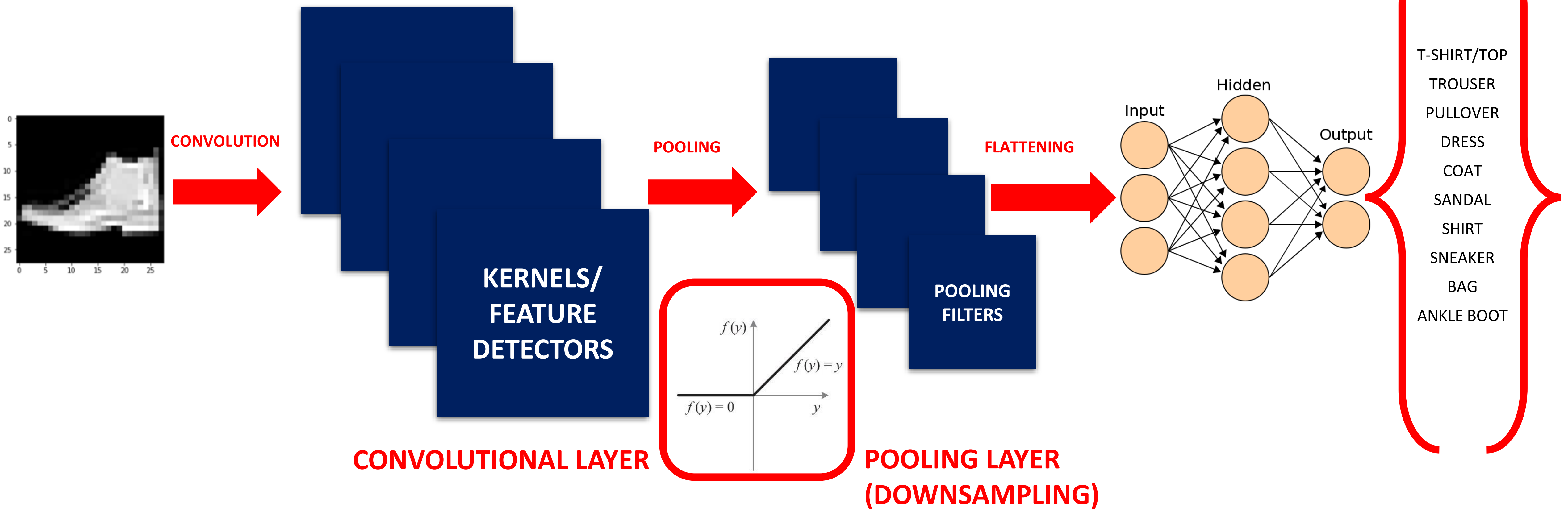
Feature Detectors

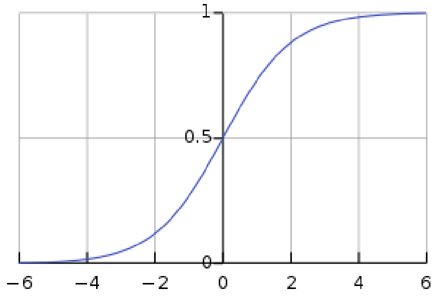


https://setosa.io/ev/image-kernels/

ReLU in CNNs

- ReLU Layers are used to add non-linearity in the feature map.
- It also enhances the sparsity or how scattered the feature map is.

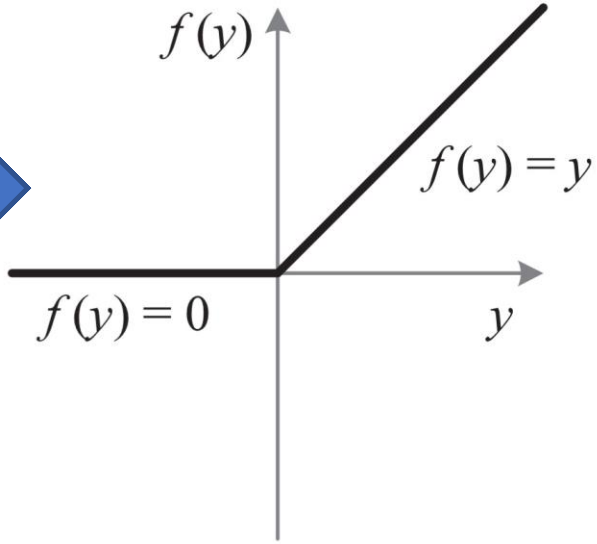




ReLU in CNNs

- RELU Layers are used to add non-linearity in the feature map.
- It also enhances the sparsity or how scattered the feature map is.
- The gradient of the RELU does not vanish as we increase x compared to the sigmoid function

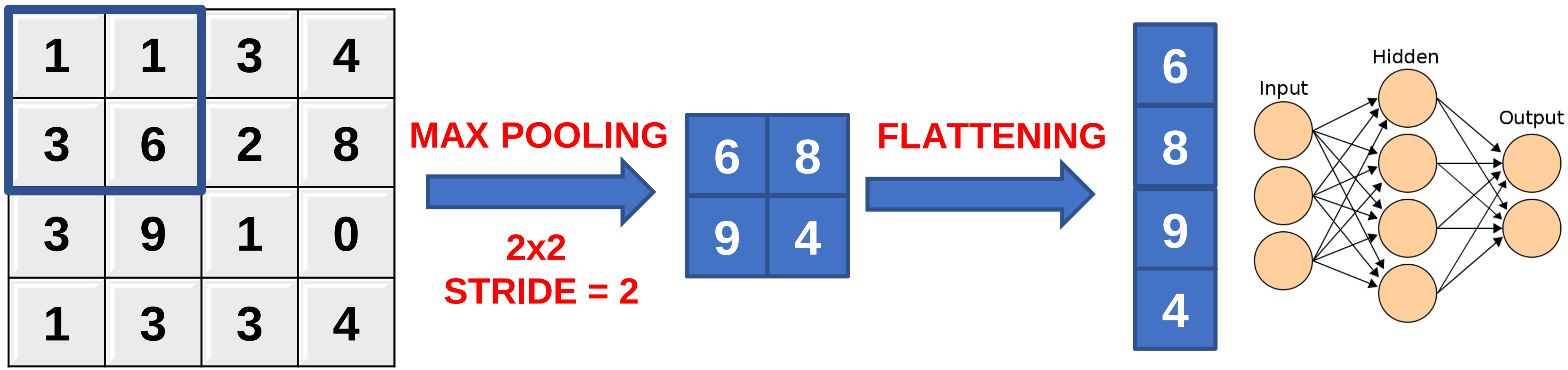
7	10	-5	2	1
1	0	2	3	-6
1	17	-5	0	0
0	1	1	1	0
0	0	-8	12	1



7	10	0	2	1
1	0	2	3	0
1	17	0	0	0
0	1	1	1	0
0	0	0	12	1

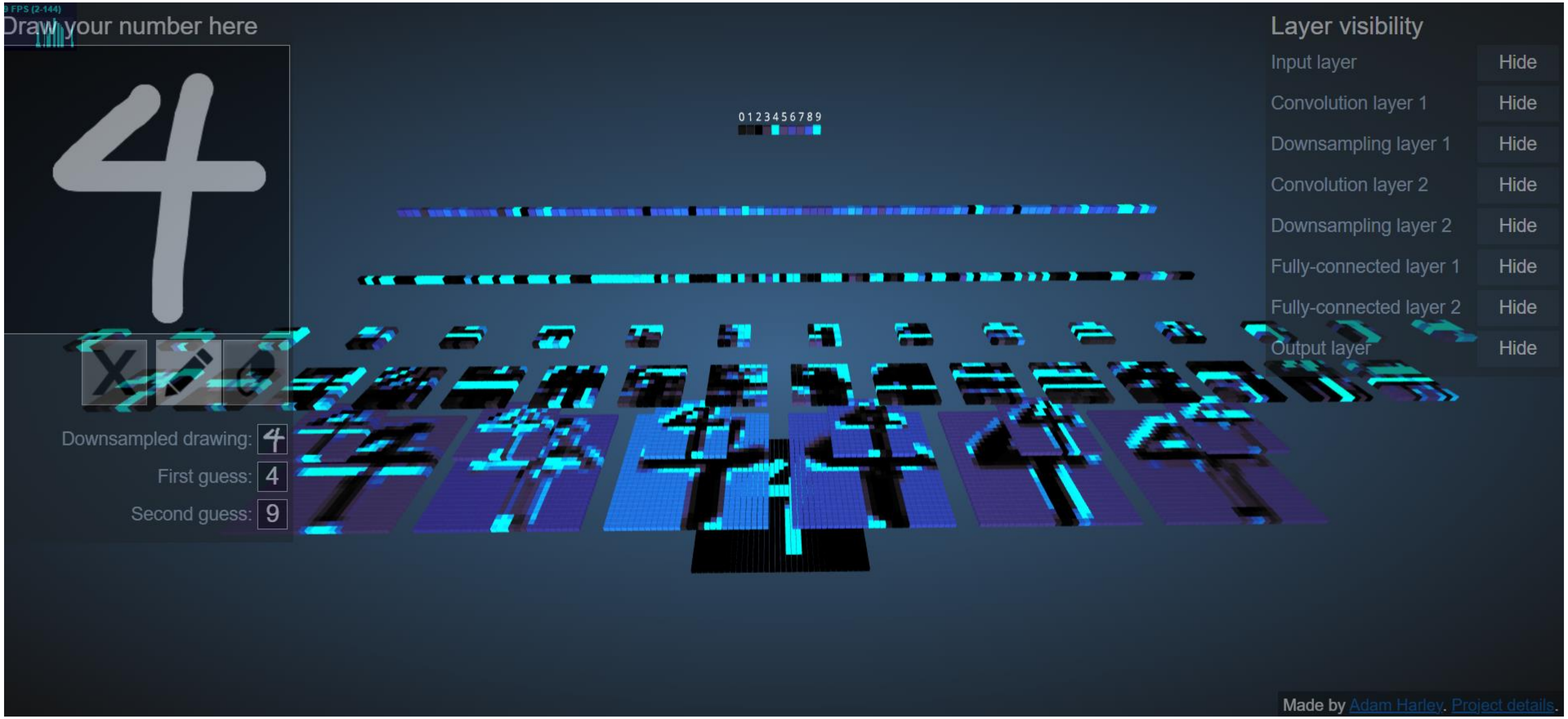
Pooling (Down Sampling)

- Pooling or down sampling layers are placed after convolutional layers to reduce feature map dimensionality.
- This improves the computational efficiency while preserving the features.
- Pooling helps the model to generalize by avoiding overfitting.
- If one of the pixel is shifted, the pooled feature map will still be the same.
- Max pooling works by retaining the maximum feature response within a given sample size in a feature map.



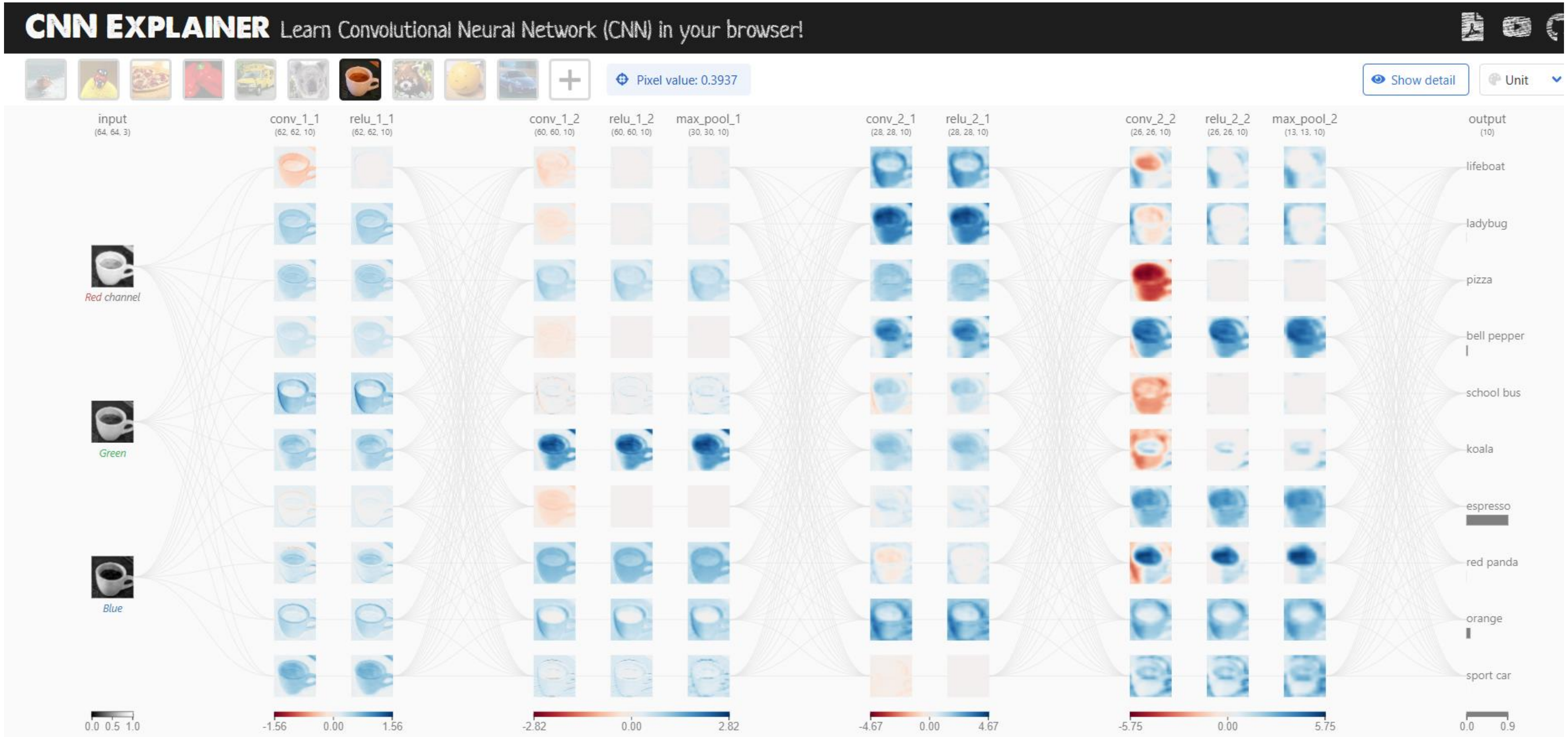
CNC Explainer

- https://adamharley.com/nn_vis/cnn/3d.html



CNC Explainer

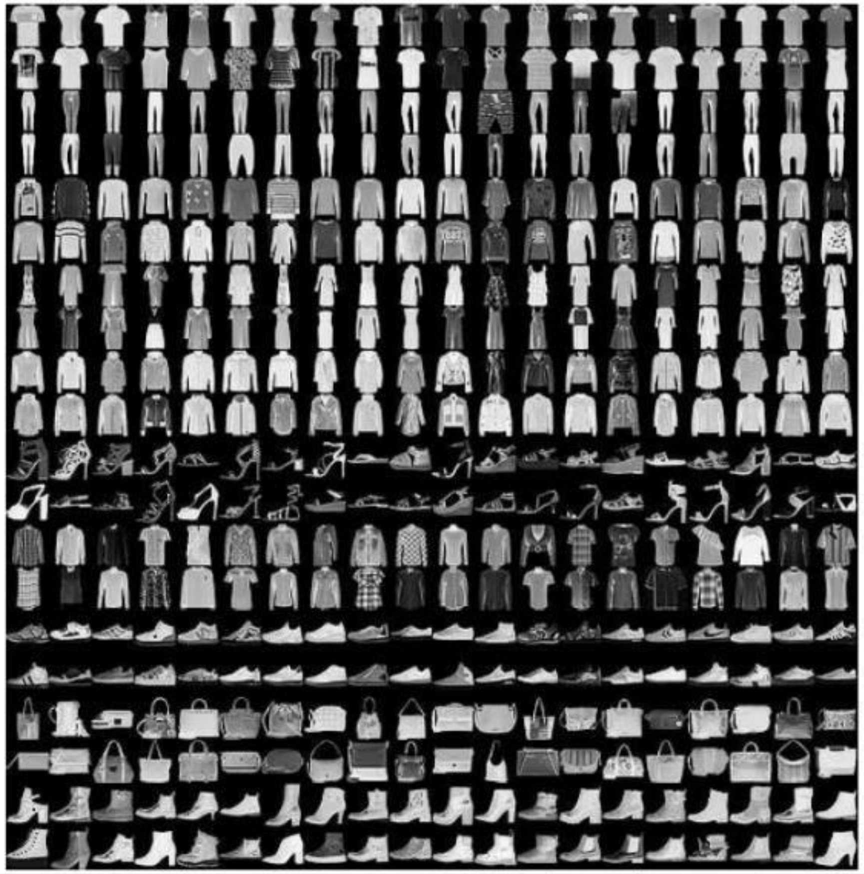
- https://poloclub.github.io/cnn-explainer/



Example7: Fashion Classification

- Create the CNNs model for fashion MNIST datasets classification using TF.

Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	



Example7: Fashion Classification

Step1: Download and split dataset.

```
fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

Step2: Create model.

```
model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Conv2D(32, (3,3), activation = 'relu', input_shape = (28,28,1)))
model.add(tf.keras.layers.MaxPooling2D(2,2))

model.add(tf.keras.layers.Conv2D(64, (3,3), activation = 'relu'))
model.add(tf.keras.layers.MaxPooling2D(2,2))

model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))

model.add(tf.keras.layers.Flatten())

model.add(tf.keras.layers.Dense(64, activation = 'relu'))

model.add(tf.keras.layers.Dense(10, activation = 'softmax'))
```

Example7: Fashion Classification

Step3: Train model.

```
model.compile(loss ='sparse_categorical_crossentropy', optimizer = 'Adam', metrics =['accuracy'])
```

```
hist = model.fit(train_images,
                 train_labels,
                 batch_size = 32,
                 epochs = 10)
```

Step4: Evaluate model.

```
evaluation = model.evaluate(test_images, test_labels)
print('Test Accuracy : {:.3f}'.format(evaluation[1]))
```

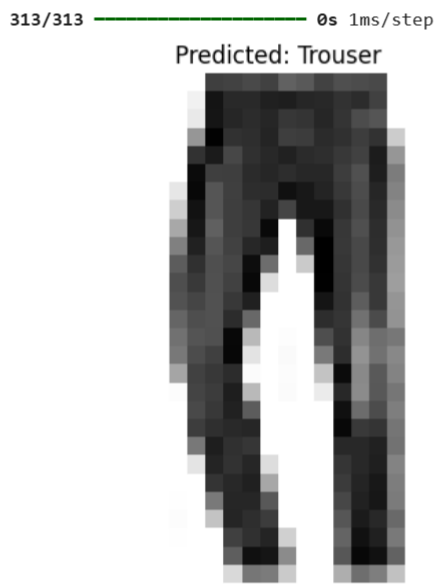
Example7: Fashion Classification

Step5: Test model.

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

predictions = model.predict(test_images)
index = 5
predicted_class = np.argmax(predictions[index])

# Display the image
plt.imshow(test_images[index].reshape(28, 28), cmap=plt.cm.binary)
plt.title(f"Predicted: {class_names[predicted_class]}")
plt.axis("off")  # Hide axis
plt.show()
```



Example8: Custom Dataset Classification

- From provided dataset, create the model to classify the image between steak and pizza using TF.

Step1: Upload provided zip file to the google drive.

Step2: Extract file.

```
import zipfile

zip_ref = zipfile.ZipFile("/content/drive/MyDrive/                    /EX8/pizza_steak.zip", "r")
zip_ref.extractall()
zip_ref.close()
```

Example8: Custom Dataset Classification

Step3: Generate dataset from image.

```
tf.random.set_seed(42)
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale = 1./255)
valid_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale = 1./255)
train_dir = "pizza_steak/train/"
test_dir = "pizza_steak/test/"
```

```
train_data = train_datagen.flow_from_directory(train_dir, batch_size = 32, target_size = (224,224),
                                               class_mode= "binary")
valid_data = valid_datagen.flow_from_directory(test_dir, batch_size = 32, target_size = (224,224),
                                               class_mode= "binary")
```

Example8: Custom Dataset Classification

Step4: Create model.

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(10, 3,activation="relu", input_shape=(224, 224, 3)))
model.add(tf.keras.layers.Conv2D(10, 3, activation = "relu"))
model.add(tf.keras.layers.MaxPool2D() )
model.add(tf.keras.layers.Conv2D(10, 3, activation = "relu"))
model.add(tf.keras.layers.Conv2D(10, 3, activation = "relu"))
model.add(tf.keras.layers.MaxPool2D())

model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(1, activation="sigmoid"))
```

Example8: Custom Dataset Classification

Step5: Train model.

```
model.compile(loss="binary_crossentropy",optimizer=tf.keras.optimizers.Adam(),
              metrics=["accuracy"])
```

```
hist = model.fit(train_data,
                    epochs=15,
                    steps_per_epoch=len(train_data),
                    validation_data=valid_data,
                    validation_steps=len(valid_data))
```

Example8: Custom Dataset Classification

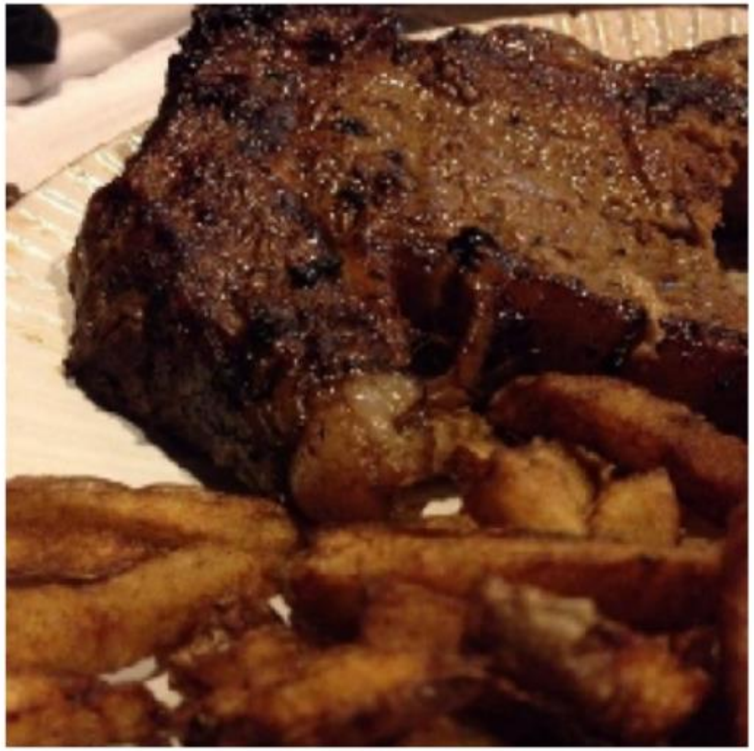
Step6: Test the result.

```python
from tensorflow.keras.preprocessing import image
```

```python
img_path = "/content/pizza_steak/test/steak/1012080.jpg"
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0

prediction = model.predict(img_array).flatten()
class_names = ['pizza', 'steak']
plt.imshow(img)
plt.title(f"Predicted: { [prediction[0]] }")
plt.axis("off")
plt.show()
```

Predicted: [0.98620504]



Practice

- From pill QC datasets, create CNNs model to classify OK pill and NG pill by TF. [chip, dirt, normal]

SUT

SURANAREE UNIVERSITY OF TECHNOLOGY

INSTITUTE OF ENGINEERING

Mechatronics

Modern Automotive