



Lesson #23 - Deep Learning Fundamentals #1

- Representing neural network
- Nonlinear activation functions
- Hidden Layers
- Case study: build a handwritten digit classifier

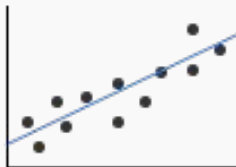
k-nearest neighbors



None
(nonexistent
training process)

3

linear regression



$$\hat{y} = 3x_1 + 10$$

logistic regression



$$\hat{p} = \frac{e^{3x}}{1 + e^{3x}}$$

decision trees &
random forests



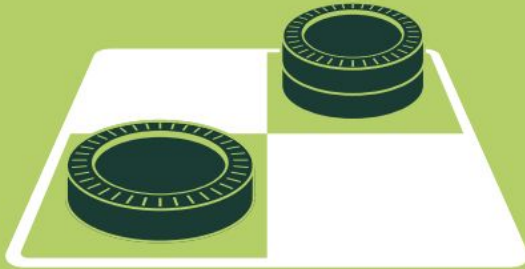
[ada,xg]boost



How do they related to each other?

ARTIFICIAL INTELLIGENCE

Early artificial intelligence stirs excitement.



MACHINE LEARNING

Machine learning begins to flourish.



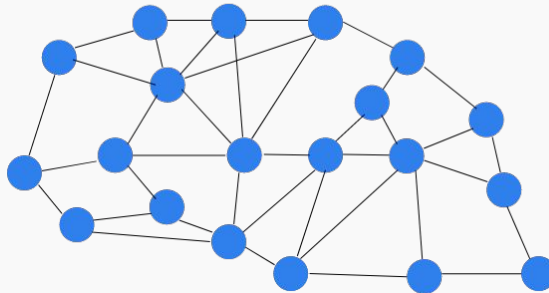
DEEP LEARNING

Deep learning breakthroughs drive AI boom.

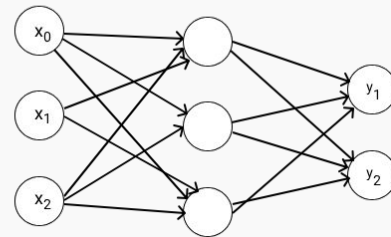


Neural Network

Biological Neural Network



Artificial Neural Network



Neuron

Neural network models were inspired by the structure of neurons in our brain and message passing between neurons



1943

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. McCULLOCH and WALTER H. PITTS

Because of the “all-or-none” character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

INTRODUCTION

THEORETICAL neurophysiology rests on certain cardinal assumptions. The nervous system is a net of neurons, each having a

THE PERCEPTRON: A PROBABILISTIC MODEL FOR
INFORMATION STORAGE AND ORGANIZATION
IN THE BRAIN¹

F. ROSENBLATT

Cornell Aeronautical Laboratory

If we are eventually to understand the capability of higher organisms for perceptual recognition, generalization, recall, and thinking, we must first have answers to three fundamental questions:

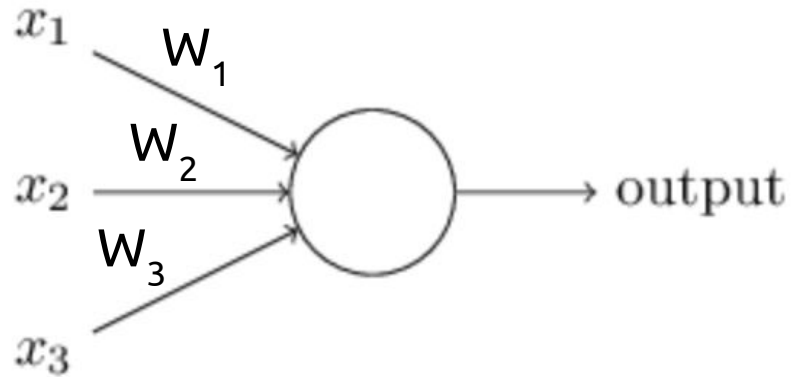
1. How is information about the physical world sensed, or detected, by the biological system?
2. In what form is information stored, or remembered?
3. How does information contained in storage, or in memory, influence recognition and behavior?

The first of these questions is in the province of sensory physiology, and is the only one for which appreciable understanding has been achieved. This article will be concerned primarily with the second and third questions, which are still subject to a vast amount of speculation, and where the few relevant facts currently supplied by neurophysiology have not yet been integrated into an acceptable theory.

With regard to the second question, two alternative positions have been maintained. The first suggests that

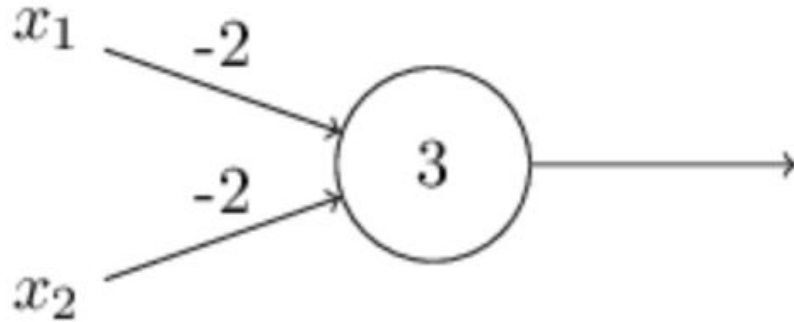
and the stored pattern. According to this hypothesis, if one understood the code or "wiring diagram" of the nervous system, one should, in principle, be able to discover exactly what an organism remembers by reconstructing the original sensory patterns from the "memory traces" which they have left, much as we might develop a photographic negative, or translate the pattern of electrical charges in the "memory" of a digital computer. This hypothesis is appealing in its simplicity and ready intelligibility, and a large family of theoretical brain models has been developed around the idea of a coded, representational memory (2, 3, 9, 14). The alternative approach, which stems from the tradition of British empiricism, hazards the guess that the images of stimuli may never really be recorded at all, and that the central nervous system simply acts as an intricate switching network, where retention takes the form of new connections, or pathways, between centers of activity. In many of the more recent developments of this position (Hebb's "cell assembly," and Hull's "cortical anticipatory goal response," for example) the "re-

How a perceptron work?

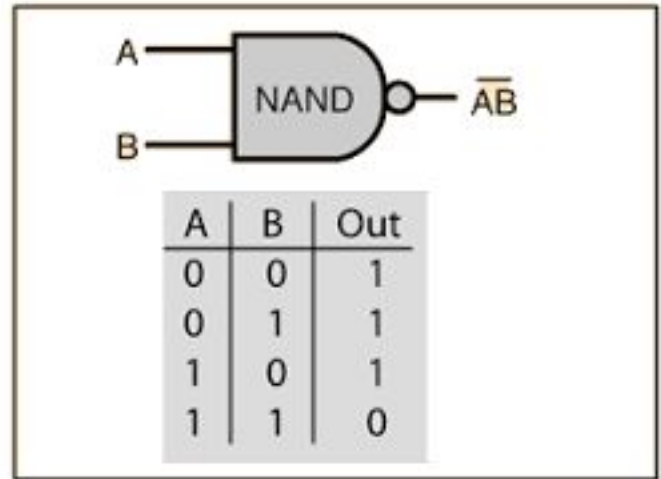


$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

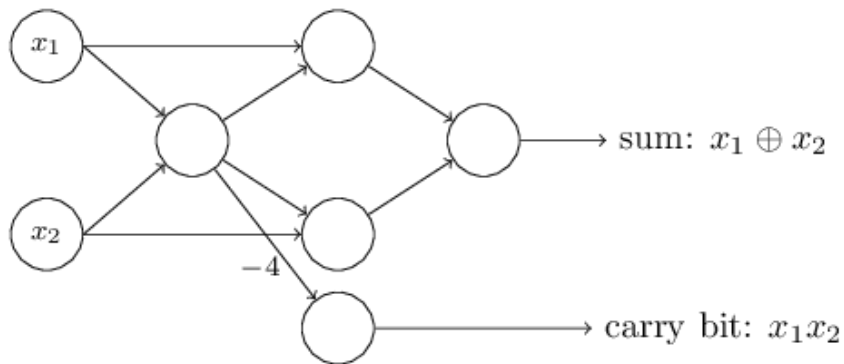
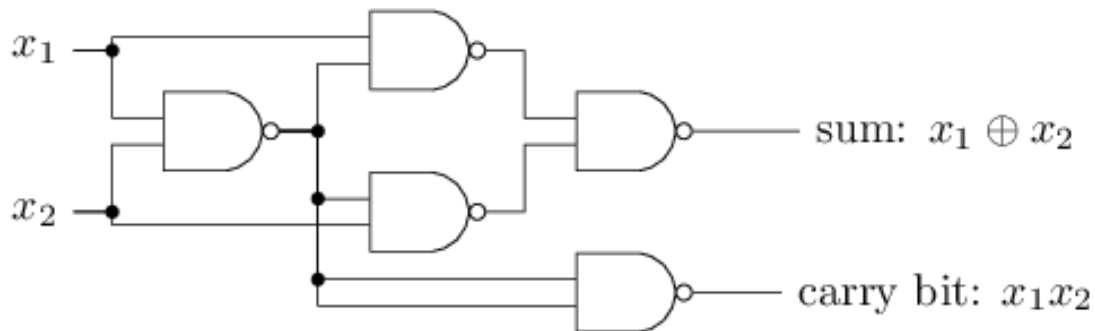
Perceptrons can be used to compute the elementary logical functions



$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$



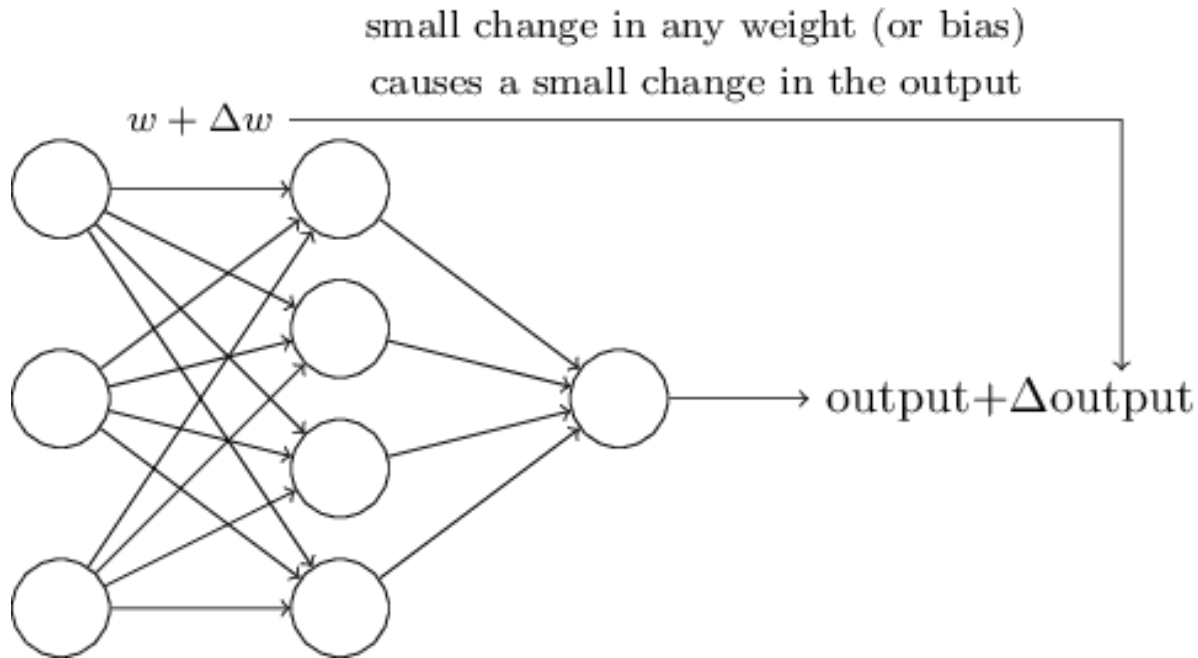
We can use networks of perceptrons to compute any logical function



Input		Output	
x_1	x_2	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

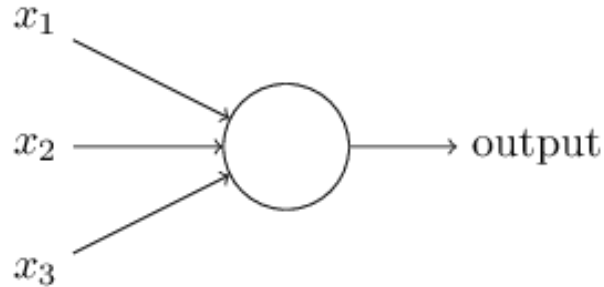
- It turns out that we can devise **learning algorithms** which can automatically tune the **weights** and **biases** of a network of artificial neurons.
- This tuning happens in response to external stimuli, **without direct intervention by a programmer.**

Learning Algorithms sound terrific. But how can we devise such algorithms?



Sigmoid Neurons

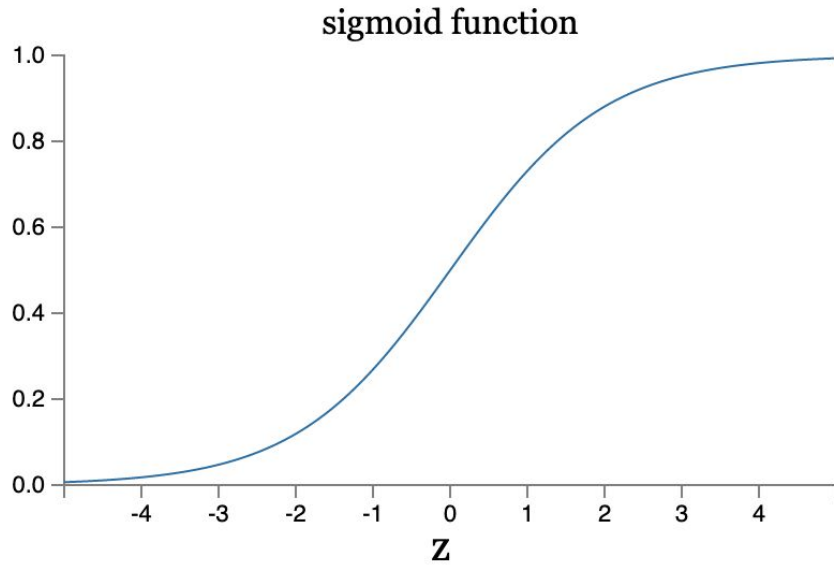
Logistic Neurons



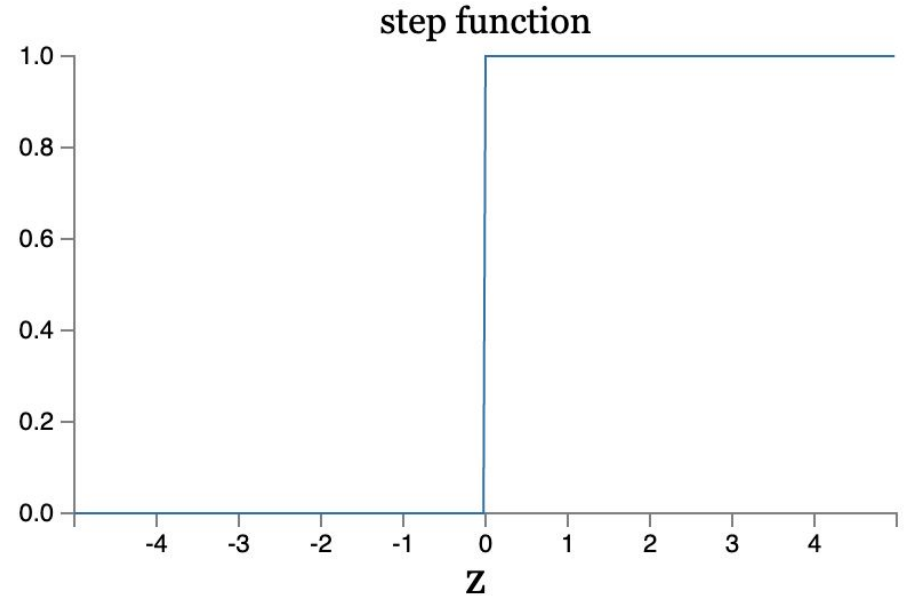
$$\sigma(w \cdot x + b)$$
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Instead of being just 0 or 1 , these inputs can also take on any values between 0 and 1

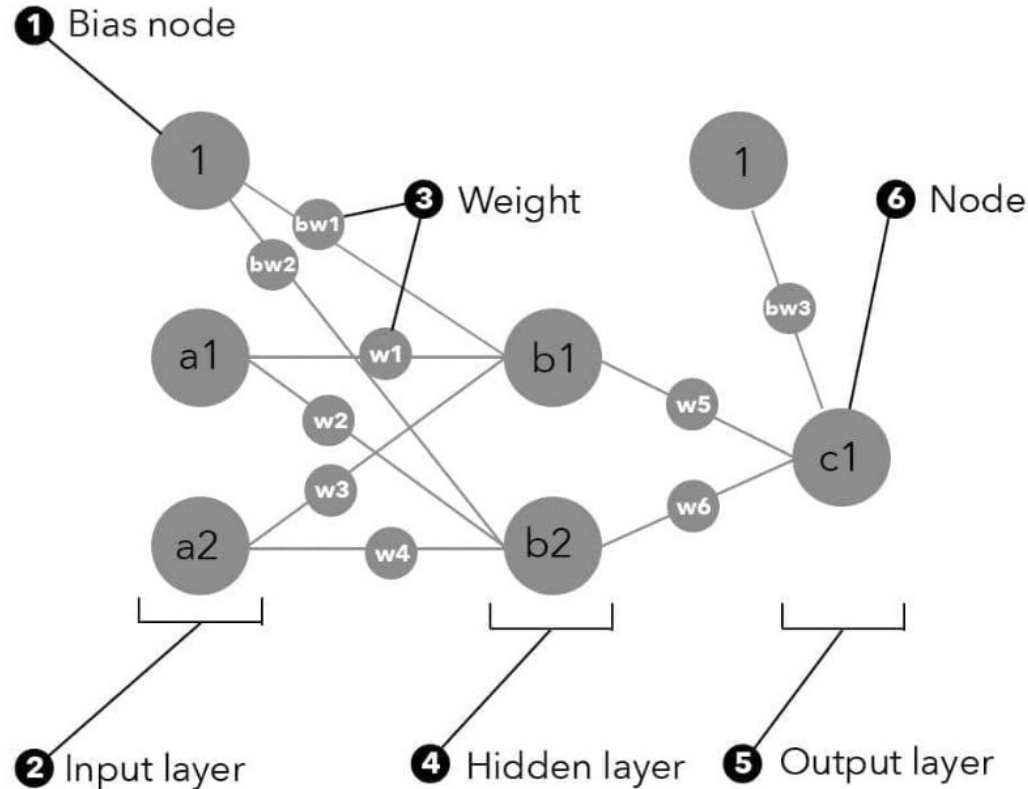
Sigmoid Neurons



Perceptrons



Representing a Neural Network Multilayer Perceptron (MLP)



Blog Home > Get started with machine learning on Arduino

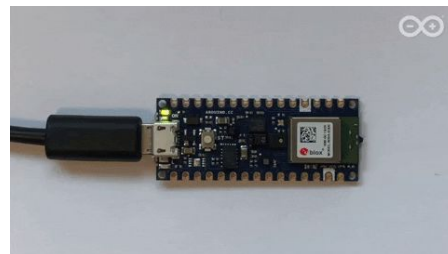
Get started with machine learning on Arduino

Posted by: ARDUINO TEAM — October 15th, 2019

This post was originally published by Sandeep Mistry and Dominic Pajak [on the TensorFlow blog](#).

[Arduino](#) is on a mission to make machine learning simple enough for anyone to use. We've been working with the TensorFlow Lite team over the past few months and are excited to show you what we've been up to together: bringing TensorFlow Lite Micro to the [Arduino Nano 33 BLE Sense](#). In this article, we'll show you how to install and run several new TensorFlow Lite Micro examples that are now available in the [Arduino Library Manager](#).

The first tutorial below shows you how to install a neural network on your Arduino board to recognize simple voice commands.



How **neural networks** are represented and how to represent **linear regression** and **logistic regression** models in that representation

A neural network that performs a linear regression

$$\hat{y} = a_0 + a_1x_1 + a_2x_2 + \dots + a_nx_n$$

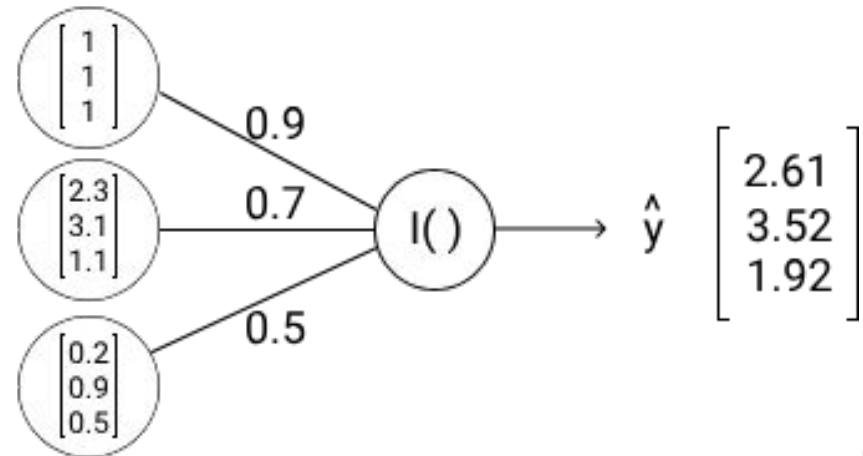
$$Xa^T = \hat{y}$$

Linear Algebra

$$I\left(\begin{bmatrix} 1 & 2.3 & 0.2 \\ 1 & 3.1 & 0.9 \\ 1 & 1.1 & 0.5 \end{bmatrix} \begin{bmatrix} 0.9 \\ 0.7 \\ 0.5 \end{bmatrix}\right) = \begin{bmatrix} 2.61 \\ 3.52 \\ 1.92 \end{bmatrix}$$

$X \qquad a^T \qquad \hat{y}$

Neural Network



Generate yourself dataset

Scikit-learn contains the following convenience functions for generating data:

- [sklearn.datasets.make_regression\(\)](#)
- [sklearn.datasets.make_classification\(\)](#)
- [sklearn.datasets.make_moons\(\)](#)

Generating regression data

```
from sklearn.datasets import make_regression
import pandas as pd
```

```
# make_regression return a tuple
# data[0].shape (1000,3) -> features
# data[1].shape (1000,) -> target
```

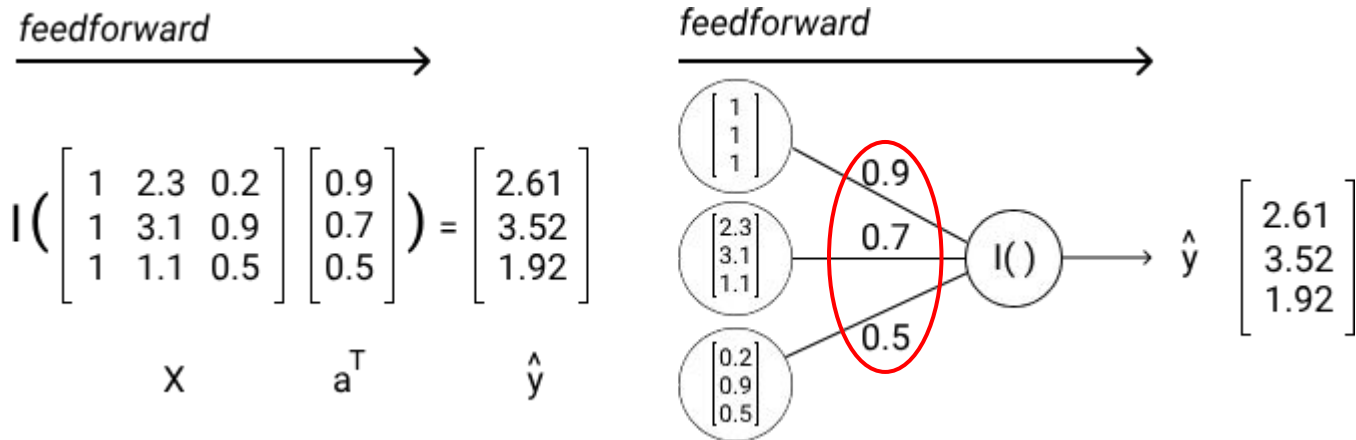
```
data = make_regression(n_samples=1000,
                      n_features=3,
                      random_state=1)
```

```
features = pd.DataFrame(data[0])
labels = pd.Series(data[1])
```

```
0    -10.378660
1     25.512450
2     19.677056
3    149.502054
4   -121.652109
dtype: float64
```

	0	1	2
0	1.293226	-0.617362	-0.110447
1	-2.793085	0.366332	1.937529
2	0.801861	-0.186570	0.046567
3	0.129102	0.502741	1.616950
4	-0.691661	-0.687173	-0.396754

Fitting a linear regression neural network



```
from sklearn.linear_model import SGDRegressor
lr = linear_model.SGDRegressor()
lr.fit(X,y)
```

Stochastic gradient descent

Randomly shuffle (reorder)
training examples

Repeat {

 for $i := 1, \dots, m$ {

$$\theta_j := \theta_j - \alpha(h_{\theta}(x^{(i)}) - y^{(i)})x_j^{(i)}$$

 (for every $j = 0, \dots, n$)

 }

}

Mini-Batch gradient descent

Say $b = 10, m = 1000$.

Repeat {

for $i = 1, 11, 21, 31, \dots, 991$ {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_{\theta}(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

(for every $j = 0, \dots, n$)

}

}

Fitting a linear regression neural network

```
# generate the dataset
```

```
data = make_regression(n_samples=100,  
                      n_features=3,  
                      random_state=1)
```

```
features = pd.DataFrame(data[0])
```

```
labels = pd.Series(data[1])
```

```
# configure the bias
```

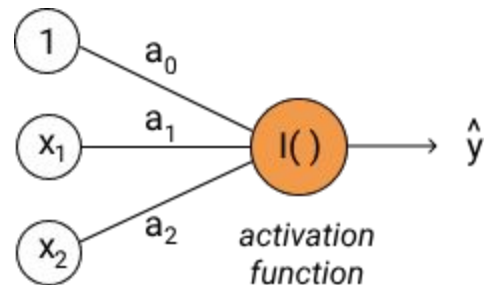
```
features["bias"] = 1
```

```
train_weights = train(features, labels)
```

```
linear_predictions = feedforward(features,  
                                  train_weights)
```

```
def train(features, labels):  
    lr = SGDRegressor()  
    lr.fit(features, labels)  
    # Returns a nested NumPy array of weights.  
    weights = lr.coef_  
    return weights
```

```
def feedforward(features, weights):  
    predictions = np.dot(features, weights.T)  
    return predictions
```




Generating a classification data


```
from sklearn.datasets import make_classification
class_data = make_classification(n_samples=100,
                                n_features=4,
                                random_state=1)
```

```
class_features = class_data[0]
```

```
class_labels = class_data[1]
```



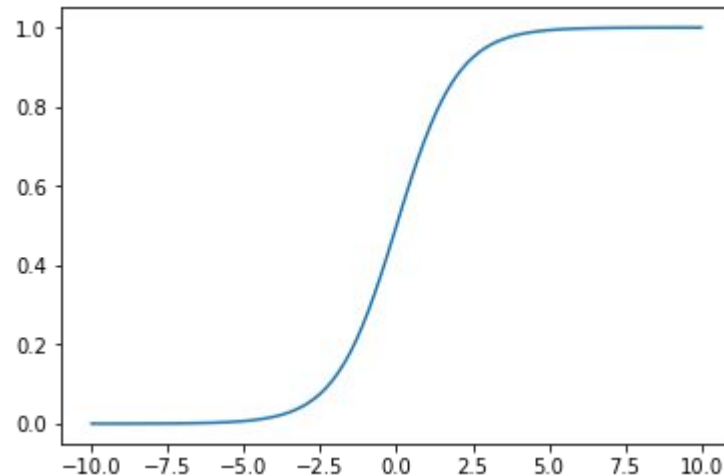
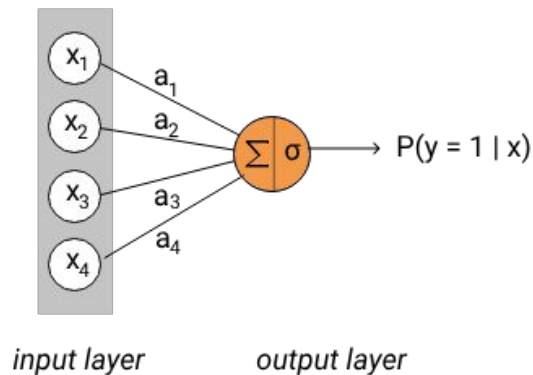
```
array([[ 1.91518414,  1.14995454, -1.52847073,  0.79430654],
       [ 1.4685668 ,  0.80644722, -1.04912964,  0.74652026],
       [ 1.47102089,  0.90060386, -1.20228498,  0.57845433],
       [ 1.07642824, -0.1813636 ,  0.49116807,  1.95642108],
       [-5.34139911, -2.29763222,  2.77907005, -3.87463248]])
```



```
array([1, 1, 1, 1, 0])
```

Implementing a neural network that performs classification

Binary Classification



$$\begin{bmatrix} 100 \times 4 \\ X \end{bmatrix} \begin{bmatrix} 4 \times 1 \\ a^T \end{bmatrix} = \begin{bmatrix} 100 \times 1 \\ P(y = 1 | x) \end{bmatrix}$$

$$\hat{y} = \sigma(Xa^T)$$

$$P(y = 1 | x) > 0.5$$

$$P(y = 0 | x) < 0.5$$


```
class_data = make_classification(n_samples=100,
                                n_features=4,
                                random_state=1)
```


```
class_features = class_data[0]
```

```
class_labels = class_data[1]
```

```
def log_train(class_features, class_labels):
    sg = SGDClassifier()
    sg.fit(class_features, class_labels)
    return sg.coef_
```

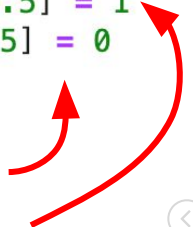
```
def sigmoid(linear_combination):
    return 1/(1+np.exp(-linear_combination))
```

```
def log_feedforward(class_features, log_train_weights):
    linear_combination = np.dot(class_features,
                                log_train_weights.T)
    log_predictions = sigmoid(linear_combination)
    log_predictions[log_predictions >= 0.5] = 1
    log_predictions[log_predictions < 0.5] = 0
    return log_predictions
```

$$\hat{y} = \sigma(Xa^T)$$


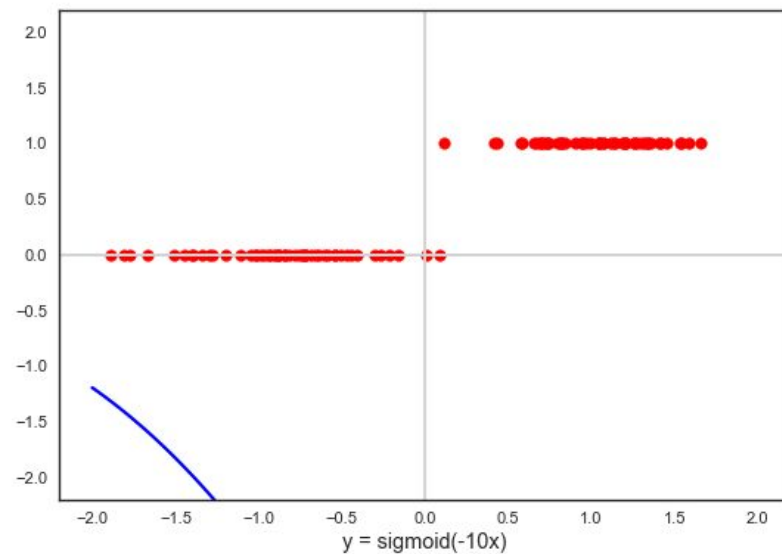
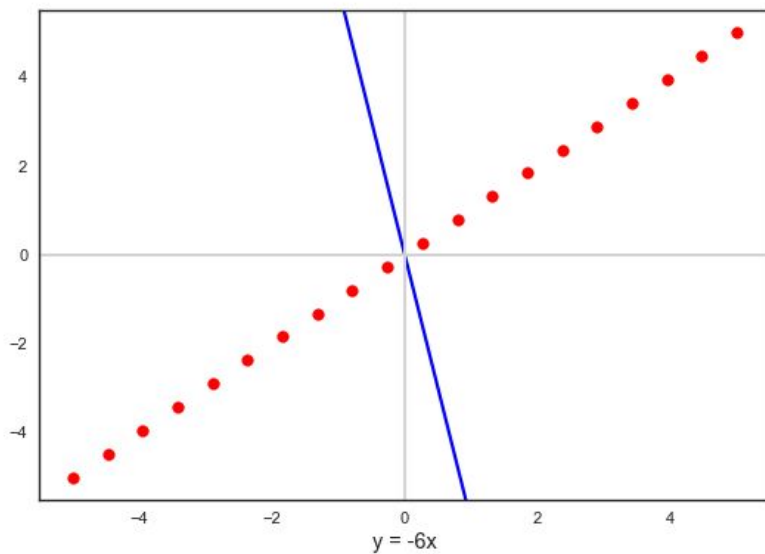
```
log_train_weights = log_train(class_features,
                                class_labels)
log_predictions = log_feedforward(class_features,
                                log_train_weights)
```

$$P(y = 0|x) < 0.5$$

$$P(y = 1|x) > 0.5$$


Activation functions

Nonlinear Activation Functions



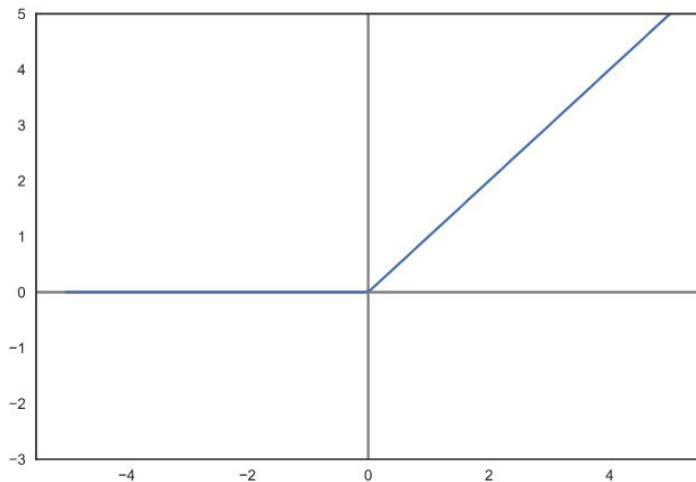
Neural Networks - Activation Functions

The three most commonly used activation functions in neural networks are:

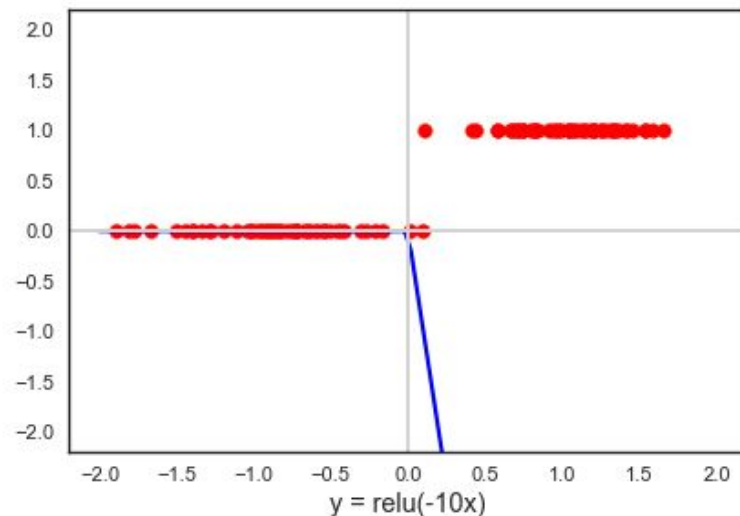
- the sigmoid function
- the ReLU function
- the tanh function

ReLU Function - Rectifier Linear Unit

```
relu = lambda x: np.maximum(0,x)  
x=np.linspace(-10,10,100)  
plt.plot(x,relu(x))
```

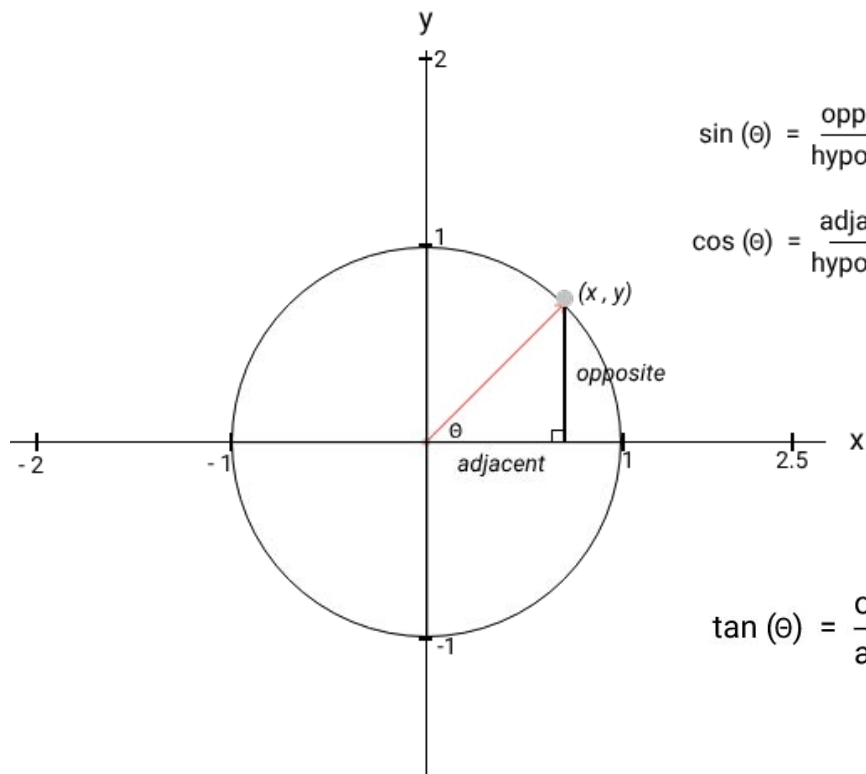


$$\text{ReLU} = \max(0, x)$$



ReLU is a commonly used activation function in neural networks for solving regression problems.

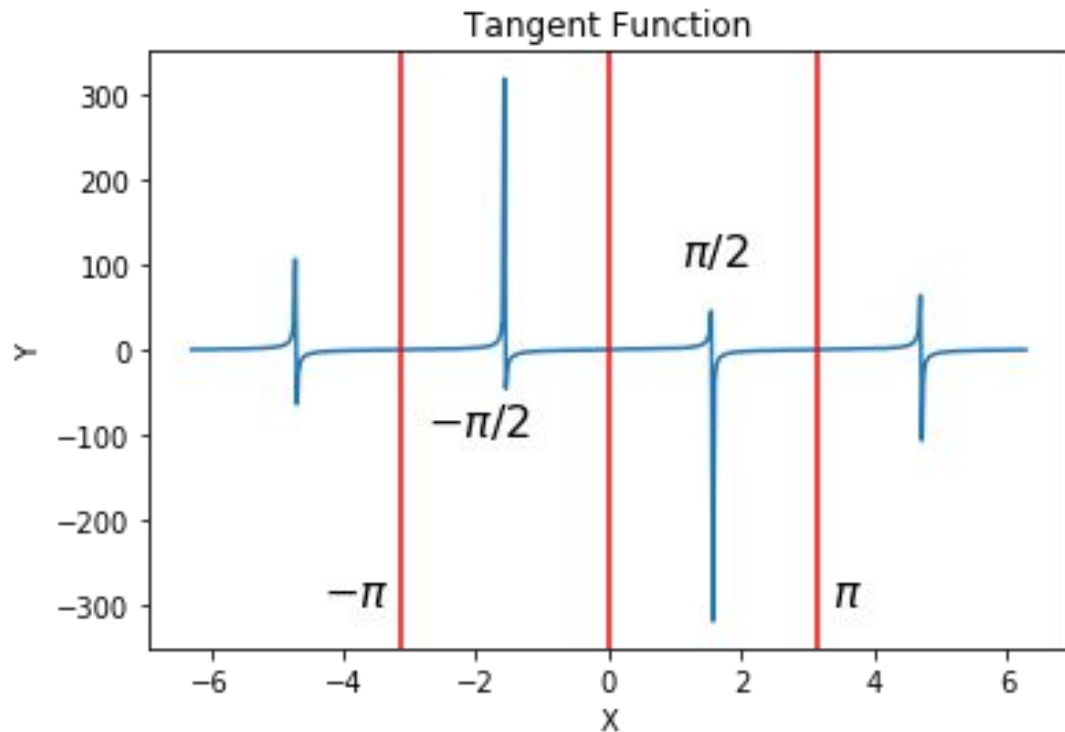
Trigonometric Functions



$$\sin(\theta) = \frac{\text{opposite}}{\text{hypotenuse}} = \text{opposite} = y$$

$$\cos(\theta) = \frac{\text{adjacent}}{\text{hypotenuse}} = \text{adjacent} = x$$

$$\tan(\theta) = \frac{\text{opposite}}{\text{adjacent}}$$

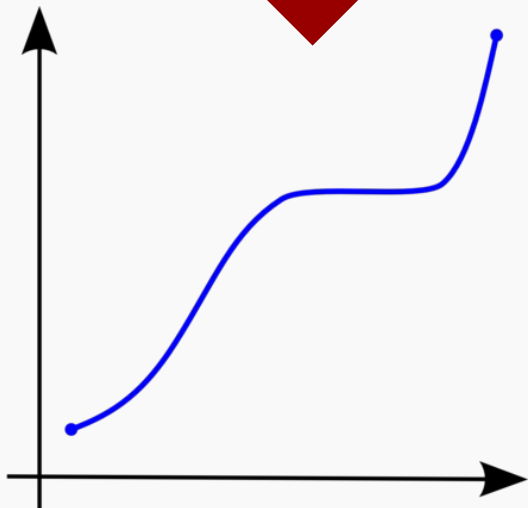


x	tan(x)
$-\pi$	0
0	0
π	0

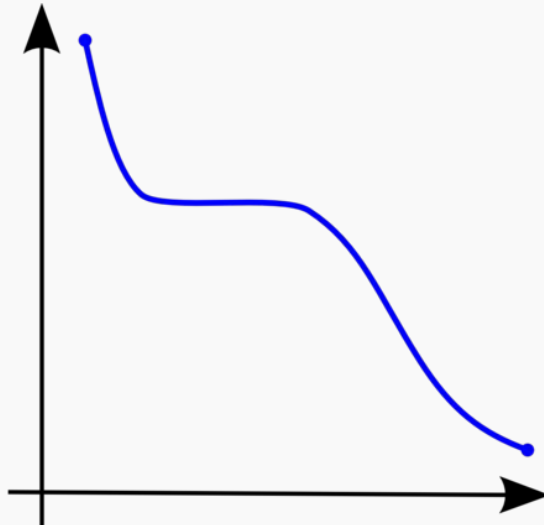
The tangent function repeats itself every π , which is known as the period
The periodic nature isn't a pattern that's found in real datasets

the activation functions that
are used in neural networks

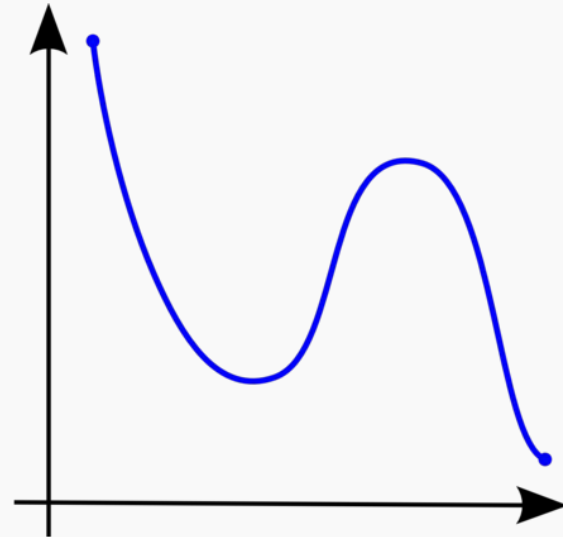
34



A monotonically
increasing function

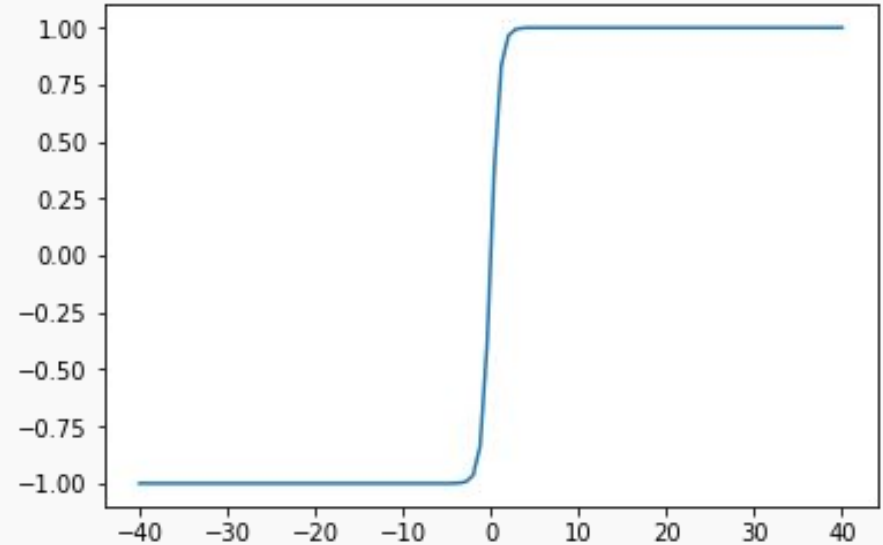
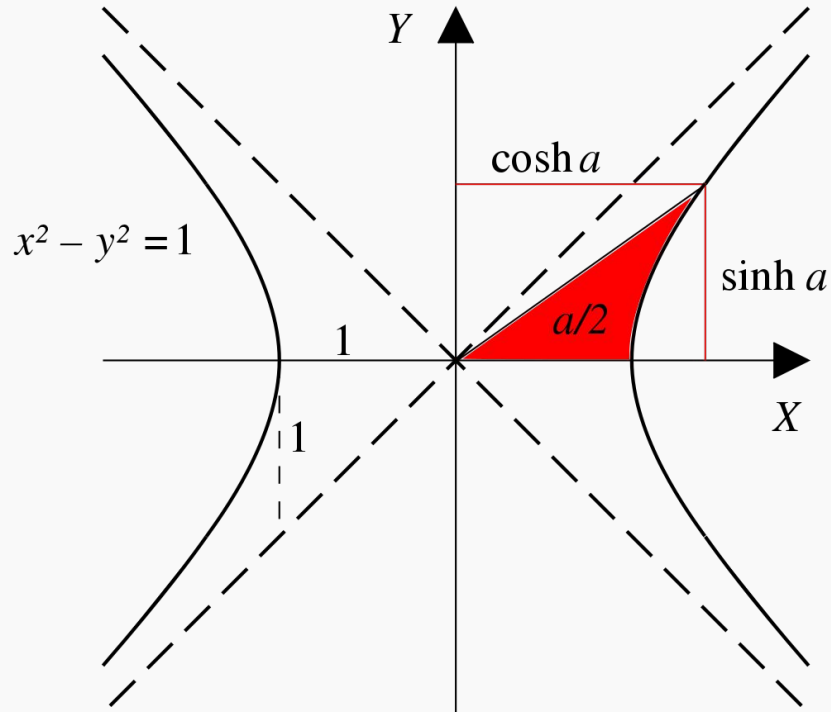


A monotonically
decreasing function



A function that is not
monotonic

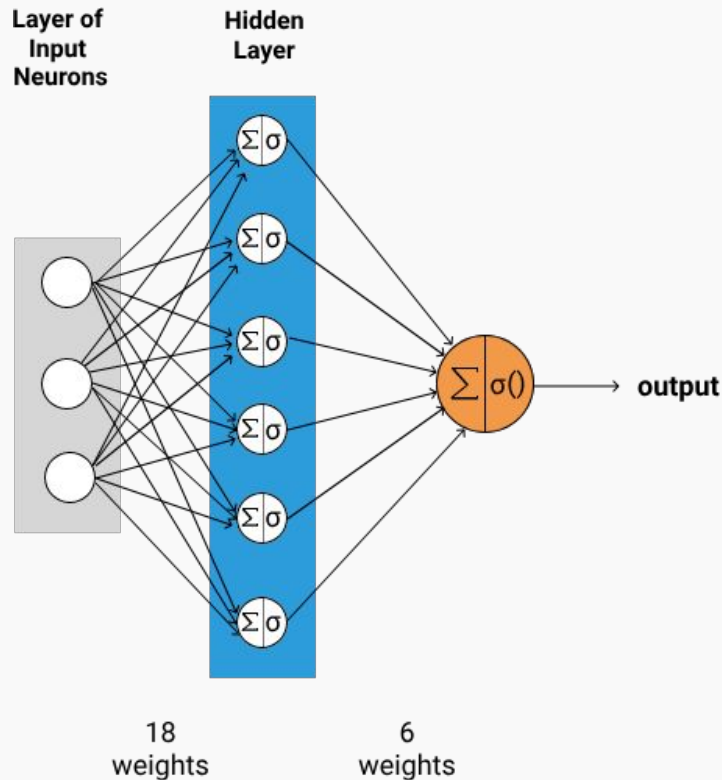
Hyperbolic Tangent Function (tanh)



It is commonly used in neural networks for classification tasks.

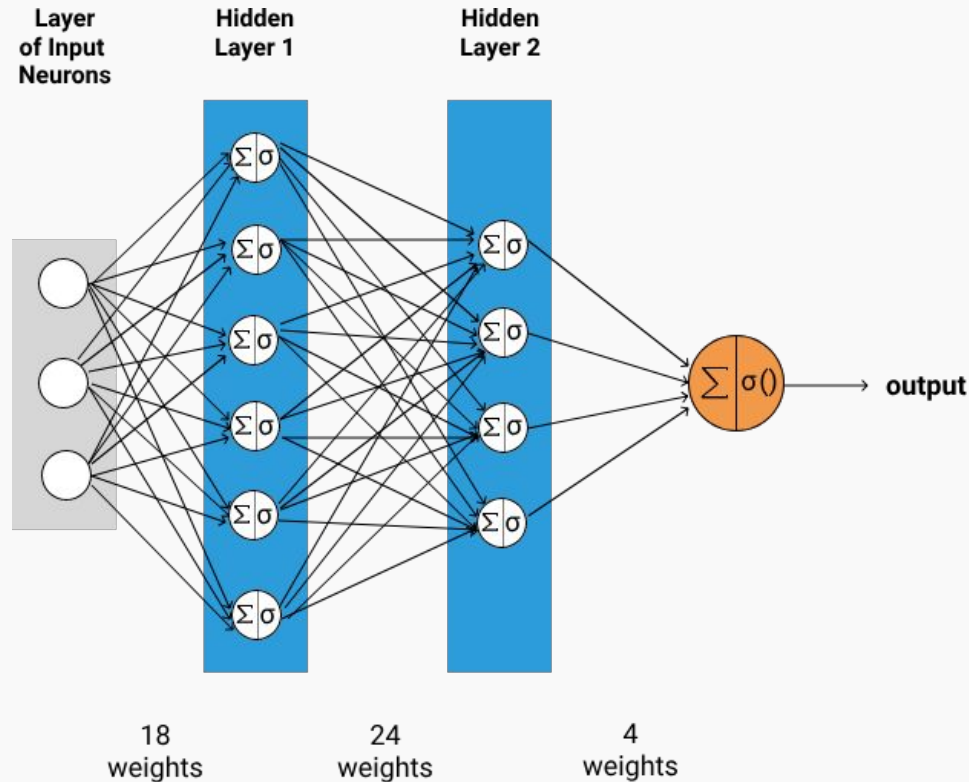
Hidden Layers

Multi-layers networks (deep neural networks)



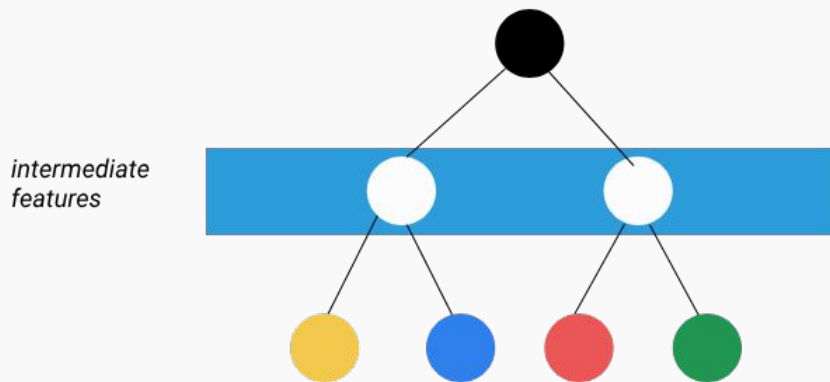
- This kind of models are able to better capture nonlinearity in the data
- Choosing the number of neurons in this layer is a bit of an art
- We can think of each hidden layer as intermediate features that are learned during the training process.

Multi-layers networks (deep neural networks)

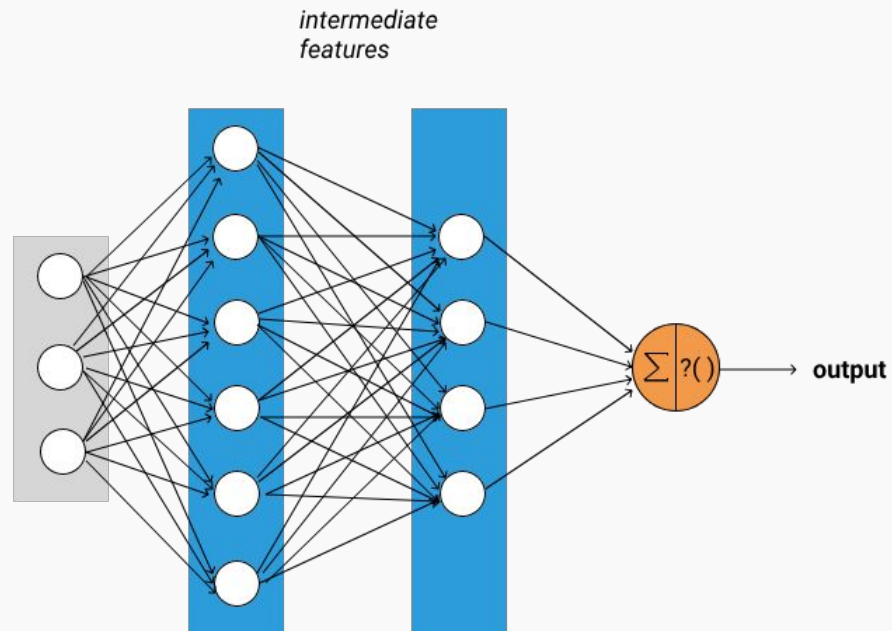


Comparison with Decision Tree Models

Decision Tree



Deep Neural Network

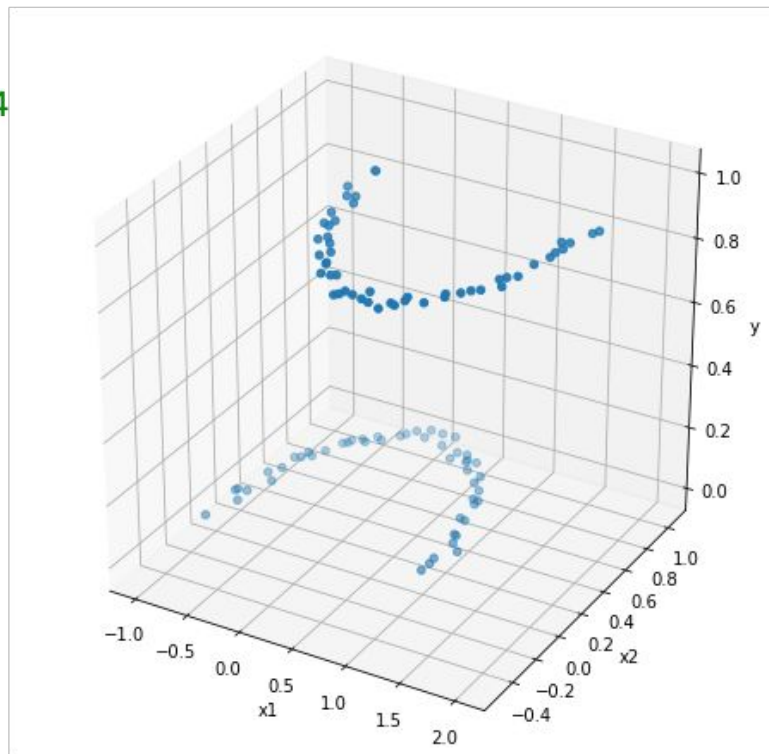


Generating data that contains nonlinearity

```
data = make_moons(100, random_state=3, noise=0.04)
features = pd.DataFrame(data[0])
labels = pd.Series(data[1])
```

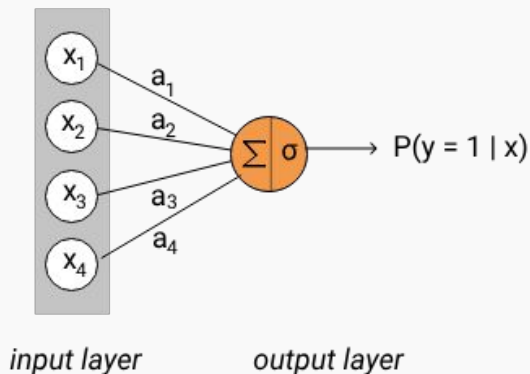
```
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')
```

```
ax.scatter(features[0], features[1], labels)
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y')
```

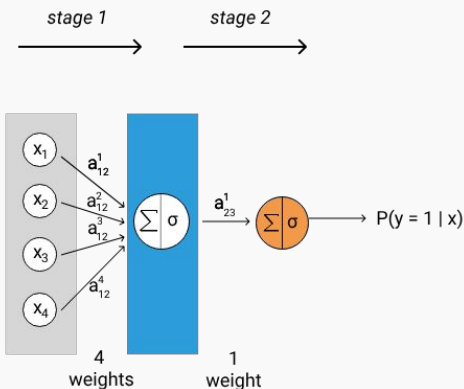


Hidden Layer with a single neuron

Binary Classification



$$\begin{bmatrix} 100 \times 4 \\ X \end{bmatrix} \begin{bmatrix} 4 \times 1 \\ a^T \end{bmatrix} = \begin{bmatrix} 100 \times 1 \\ P(y = 1 | x) \end{bmatrix}$$



$$\text{stage 1} \quad \sigma \left(\begin{bmatrix} 100 \times 4 \\ X \end{bmatrix} \begin{bmatrix} 4 \times 1 \\ a_1^T \end{bmatrix} \right) = \begin{bmatrix} 100 \times 1 \\ L_1 \end{bmatrix}$$

$$\text{stage 2} \quad \sigma \left(\begin{bmatrix} 100 \times 1 \\ L_1 \end{bmatrix} \begin{bmatrix} 1 \times 1 \\ a_2^T \end{bmatrix} \right) = \begin{bmatrix} 100 \times 1 \\ L_2 \end{bmatrix}$$

Training a neural network using scikit-learn

Scikit-learn contains two classes for working with neural networks:

- [MLPClassifier](#)
- [MLPRegressor](#)

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier()
mlp.fit(X_train, y_train)
predictions = mlp.predict(X_test)
```

```
mlp = MLPClassifier(hidden_layer_sizes=(6,), activation='logistic')
```

```
data = make_moons(1000, random_state=3, noise=0.04)
features = pd.DataFrame(data[0])
labels = pd.Series(data[1])
features["bias"] = 1

train_x, test_x, train_y, test_y = train_test_split(features,
                                                    labels,
                                                    test_size=0.30,
                                                    random_state=42)
```

```
mlp = MLPClassifier(hidden_layer_sizes=(4,),
                    activation='logistic', max_iter=10000)
mlp.fit(train_x, train_y)
nn_predictions = mlp.predict(test_x)
```

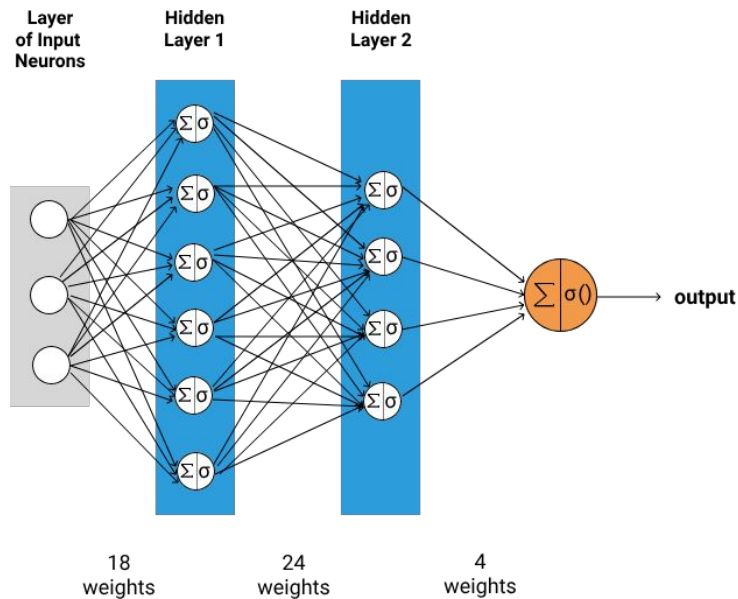
```
lr = LogisticRegression(solver='lbfgs')
lr.fit(train_x, train_y)
log_predictions = lr.predict(test_x)
```

0.88

0.88

```
nn_accuracy = accuracy_score(test_y, nn_predictions)
log_accuracy = accuracy_score(test_y, log_predictions)
```

Multiple Hidden Layers

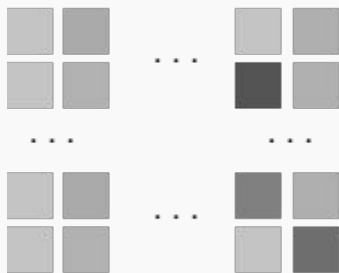


```
mlp = MLPClassifier(hidden_layer_sizes=(n,k))
```

Case Study: building a handwritten digits classifier

Why is image classification a hard task?

Single Image in the Dataset



rendered image

50	90	...	70	90
50	82	...	180	70
...
50	120	50
50	50	120

pixel values

thousands or millions of columns

50	90	...	70	90	50	82	...	50	120
----	----	-----	----	----	----	----	-----	----	-----

single observation in the data

A 128 x 128 image has 16384 features

The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches

Md Zahangir Alom¹, Tarek M. Taha¹, Chris Yakopcic¹, Stefan Westberg¹, Paheding Sidike², Mst Shamima Nasrin¹, Brian C Van Essen³, Abdul A S. Awwal³, and Vijayan K. Asari¹

Abstract—In recent years, deep learning has garnered tremendous success in a variety of application domains. This new field of machine learning has been growing rapidly, and has been applied to most traditional application domains, as well as some new areas that present more opportunities. Different methods have been proposed based on different categories of learning, including supervised, semi-supervised, and un-supervised learning. Experimental results show state-of-the-art performance using deep learning when compared to traditional machine learning approaches in the fields of image processing, computer vision, speech recognition, machine translation, art, medical imaging, medical information processing, robotics and control, bio-informatics, natural language processing (NLP), cybersecurity, and many others.

This report presents a brief survey on the advances that have occurred in the area of DL, starting with the Deep Neural Network (DNN). The survey goes on to cover the Convolutional Neural Network (CNN), the Recurrent Neural Network (RNN) including Long Short Term Memory (LSTM) and Gated Recurrent Units (GRU), the Auto-Encoder (AE), the Deep Belief Network (DBN), the Generative Adversarial Network (GAN), and Deep Reinforcement Learning (DRL). Additionally, we have included recent developments such as advanced variant DL techniques based on these DL approaches. This work considers most of the papers published after 2012 from when the history of deep learning began. Furthermore, DL approaches that have been

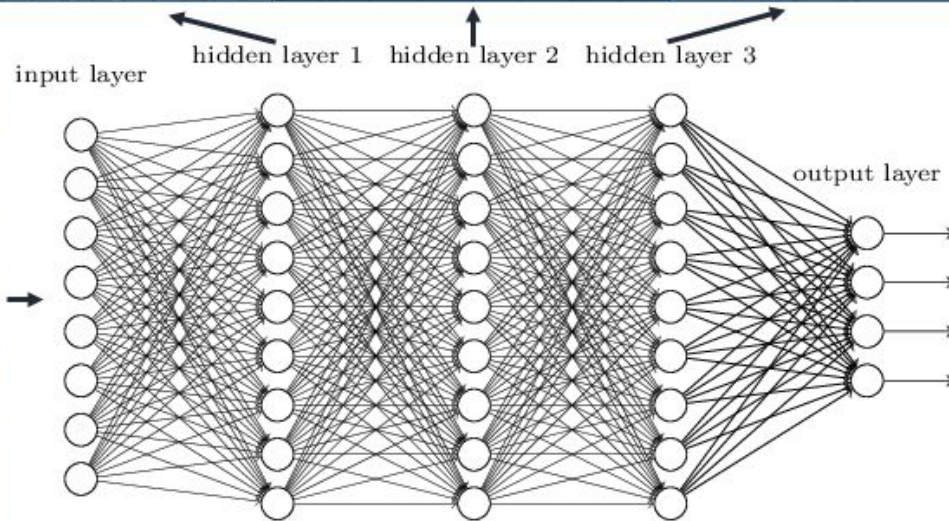
I. INTRODUCTION

Since the 1950s, a small subset of Artificial Intelligence (AI), often called Machine Learning (ML), has revolutionized several fields in the last few decades. Neural Networks (NN) are a subfield of ML, and it was this subfield that spawned Deep Learning (DL). Since its inception DL has been creating ever larger disruptions, showing outstanding success in almost every application domain. Fig. 1 shows, the taxonomy of AI. DL (using either deep architecture of learning or hierarchical learning approaches) is a class of ML developed largely from 2006 onward. Learning is a procedure consisting of estimating the model parameters so that the learned model (algorithm) can perform a specific task. For example, in Artificial Neural Networks (ANN), the parameters are the weight matrices ($w_{i,j}$'s). DL on the other hand consists of several layers in between the input and output layer which allows for many stages of non-linear information processing units with hierarchical architectures to be present that are exploited for feature learning and pattern classification [1, 2]. Learning methods based on representations of data can also be defined as representation learning [3]. Recent literature states that DL based representation learning involves a hierarchy of features or concepts, where the high level concepts can be defined from

March 2018

Why is deep learning effective in image classification?

Deep neural networks learn hierarchical feature representations

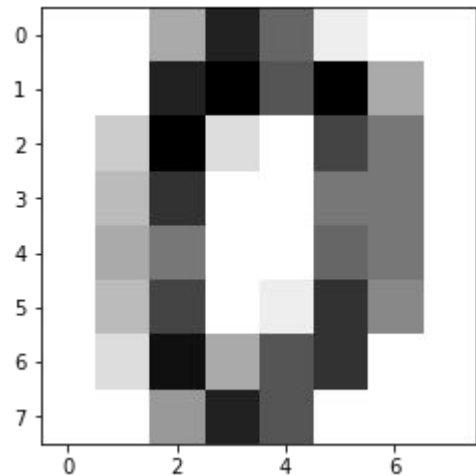


Working with image data

```
from sklearn.datasets import load_digits
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
digits = load_digits()
df_digits = pd.DataFrame(digits["data"])
labels = pd.Series(digits["target"])
```

```
first_image = df_digits.iloc[0].values.reshape(8,8)
plt.imshow(first_image, cmap='gray_r')
```

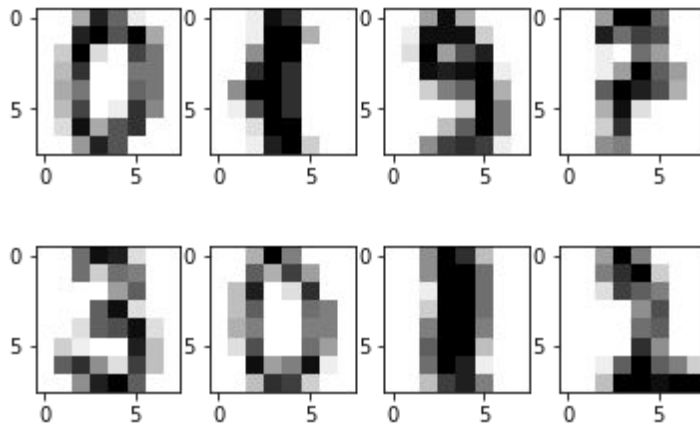


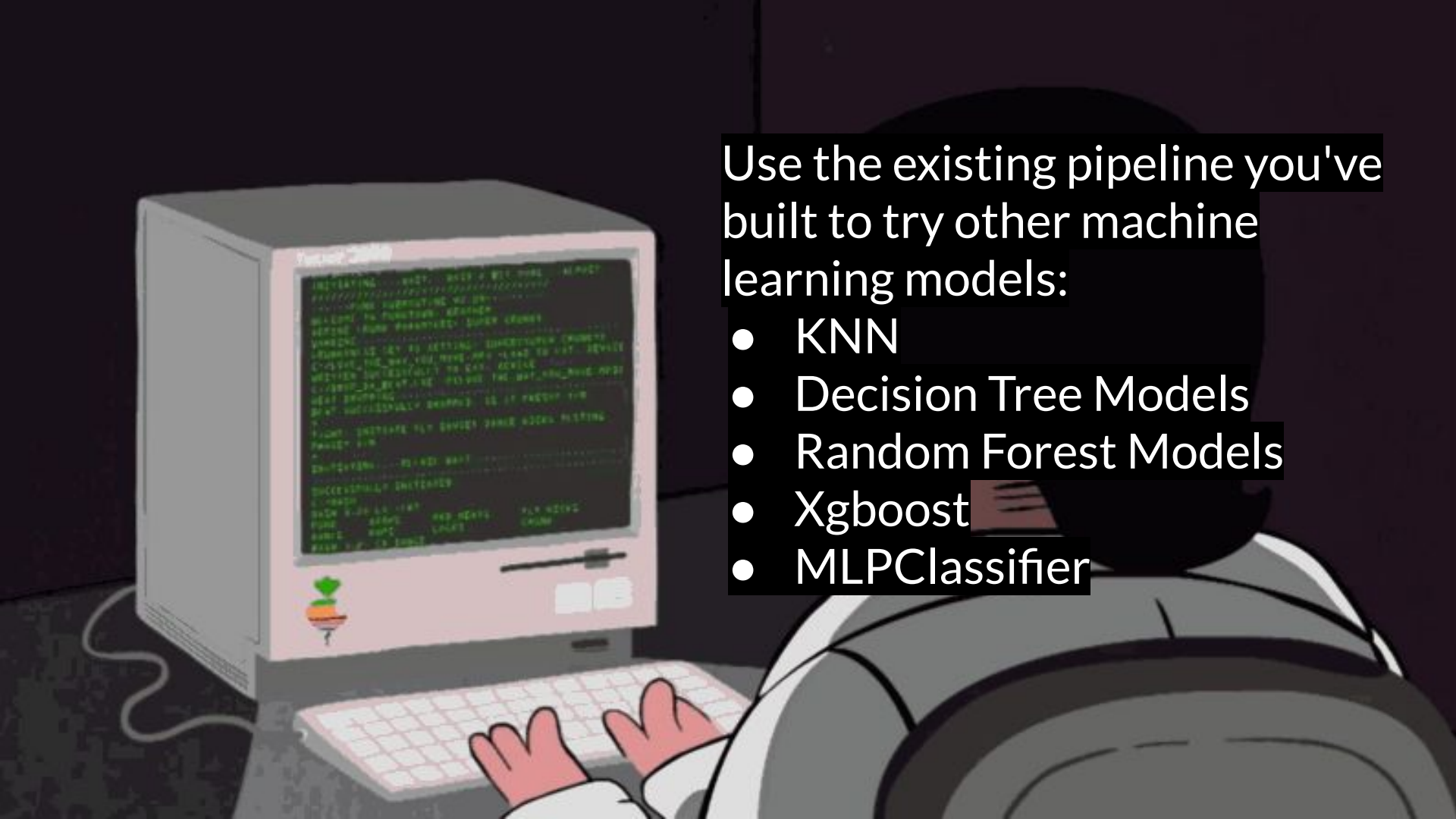
Working with image data

```
fig, ax = plt.subplots(2, 4)
```

```
selected_rows = [0, 99, 199, 299, 999, 1099, 1199, 1299]
```

```
for i, index in enumerate(selected_rows):  
    ax[i//4, i%4].imshow(df_digits.iloc[index].values.reshape(8,8), cmap='gray_r')
```





Use the existing pipeline you've built to try other machine learning models:

- KNN
- Decision Tree Models
- Random Forest Models
- Xgboost
- MLPClassifier