Computer Vision and Machine Learning

Haitham El-Hussieny, PhD
Associate Professor

Department of Mechatronics and Robotics Engineering
School of Innovative Design Engineering

الجامعة المصرية اليابانية للعلوم والتكنولوجيا
E-JUST
Egypt - Japan University of Science and Technology
エジプト日本科学技術大学

Fall 2022
March 7, 2023

Table of Contents

Table of Contents

Introduction to Computer Vision

What is Computer Vision?

Computer Vision

It is the field that enables computers and systems to **derive meaningful information** from digital images, videos and other visual inputs.

Every picture tells a story!

Computer Vision

It is the field that enables computers and systems to **derive meaningful information** from digital images, videos and other visual inputs.

Every picture tells a story!

Computer vs. Machine Vision

- **Computer vision:** uses a system with a PC-based processor to analyze the imaging data it collects.
- **Machine vision:** It needs embedded processors for image processing. It's built to quickly analyze image data and make simple, automated decisions.

Introduction to Computer Vision

Computer Vision vs. Image Processing.

Introduction to Computer Vision

Applications of Computer Vision.

1. Object Character Recognition (OCR).

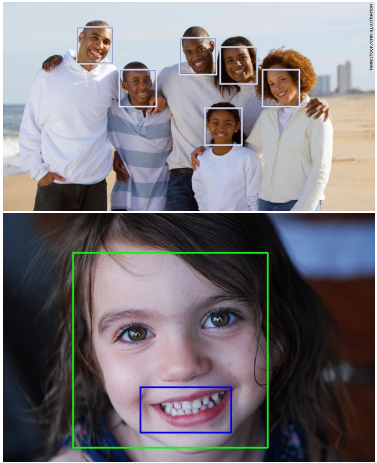
Object Character Recognition (OCR):

The electronic of images of typed, handwritten or printed text into machine-encoded text.

2. Face Recognition.

Face Detection/Recognition

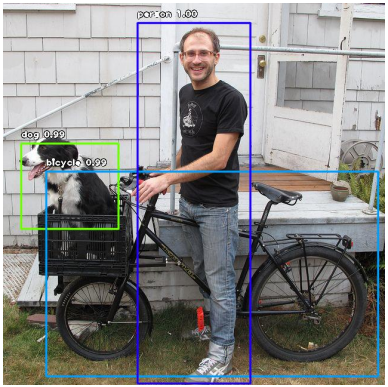
- Used in today's cameras to detect faces.
- Recently, blink and smile detection are used.



3. Object Recognition.

Object Recognition.

Object recognition allows robots and AI programs to pick out and identify objects from inputs like video and still camera images.

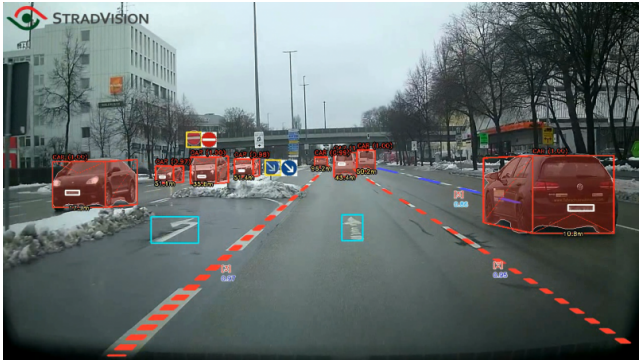


Yolo Object Recognition

4. Autonomous Vehicles

CV in Autonomous Vehicles

Computer vision is used in autonomous vehicles to detect and identify objects such as pedestrians, other vehicles, traffic lights, and road signs.

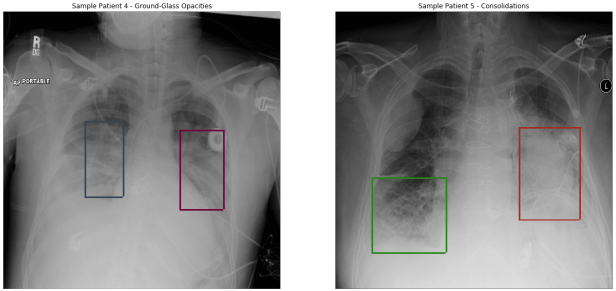


Object Recognition in Autonomous Vehicles

5. Medical image analysis

Medical image analysis

Computer vision is used to analyze medical images such as X-rays, CT scans, and MRI images to aid in the diagnosis and treatment of medical conditions.

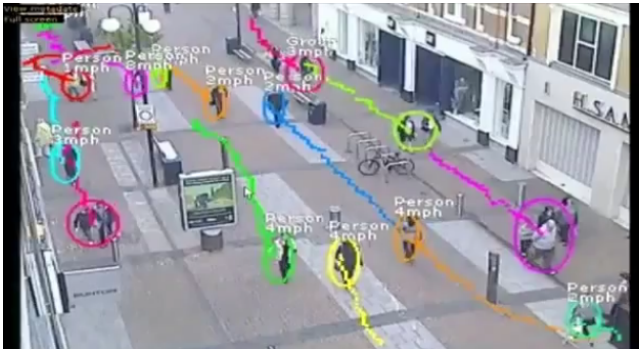


Pneumonia detection from chest radiograph using deep learning

6. Surveillance

Surveillance

Computer vision is used in surveillance systems to monitor and analyze live video feeds to detect and alert on suspicious activities.

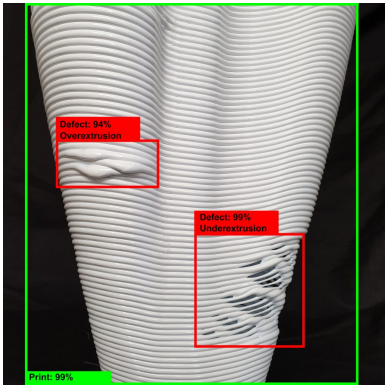


People tracking for suspicious detection

7. Quality Control

Quality Control

Computer vision is used in manufacturing to detect defects and ensure quality control.



Computer vision for 3D printing quality control

8. Agriculture

Agriculture

Computer vision is used in agriculture to monitor crop growth, detect diseases, and optimize yields.

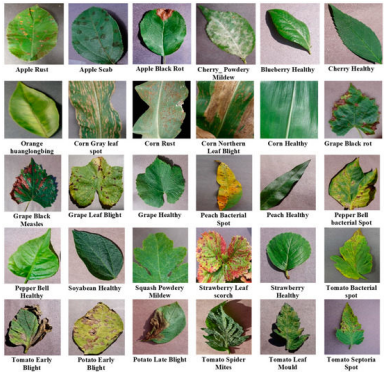


Image-Based Leaf Disease Detection

Table of Contents

Getting Started with OpenCV

OpenCV Overview.

The most common image processing library is OpencV.

- Open Source Computer Vision Library.
- Free for both academic and commercial use
- C++/Python/Java
- Windows, MacOS, Linux, iOS, Android
- Strong focus on real-time (written in C++ and optimized)

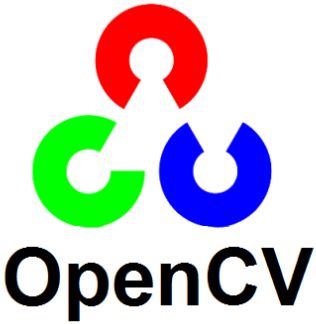


Image Segmentation

Image segmentation is the process of partitioning a digital image into more meaningful and easier to analyze multiple image segments.

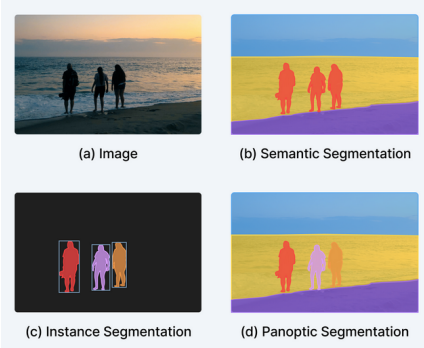


Image Thresholding

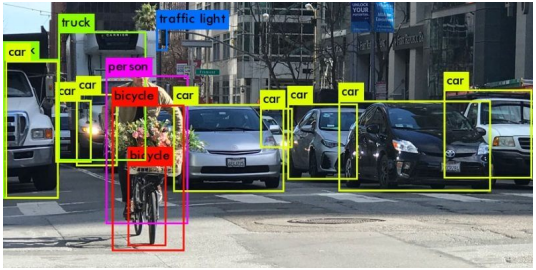
In digital image processing, thresholding is the simplest method of segmenting images. From a grayscale image, thresholding can be used to create binary images.

Original image.

The binary image resulting from a thresholding of the original image.

Object Detection and Recognition.

Detection of instances of semantic objects of a certain class (humans, buildings, or cars) in digital images.

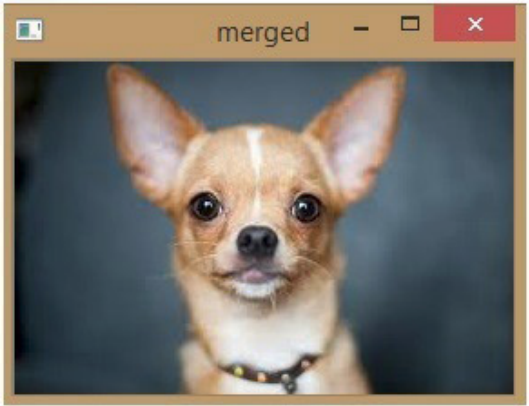


Edge Detection.

Edge detection includes a variety of mathematical methods that aim at identifying edges, curves in a digital image at which the image brightness changes sharply or, more formally, has discontinuities.

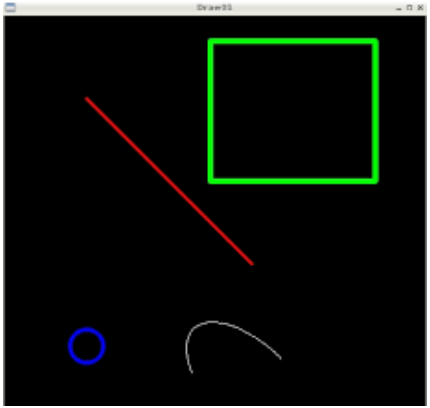
Video/Image Input Output.

OpenCV loads and saves images and videos from the disk or from a USB cams.



Shapes Drawing.

OpenCV can be used to draw basic shapes on images or videos.



Google Colab

Google Colaboratory (Colab for short) is a cloud-based development environment provided by Google for machine learning and data analysis that allows users to write and run Python code in their web browsers, with access to free GPU and TPU resources.

Note: In google colab only, we will use the function `cv2_imshow()` instead of `cv2.imshow()`

Table of Contents

Design a program that ***recognizes human activities.***

Walking

Running

Cycling

Design a program that ***recognizes human activities.***

Walking

Running

Cycling

```
if speed<4:
    activity = 'Walking'
```

Design a program that ***recognizes human activities.***

Walking

Running

Cycling

```
if speed<4:
    activity = 'Walking'
```

```
if speed<4:
    activity = 'Walking'
    else:
        activity = 'Running'
```

Design a program that ***recognizes human activities.***

Walking

Running

Cycling

```
if speed<4:
    activity = 'Walking'
```

```
if speed<4:
    activity = 'Walking'
    else:
        activity = 'Running'
```

```
if speed<4:
    activity = 'Walking'
    else if speed<12:
        activity = 'Running'
    else:
        activity = 'Cycling'
```

Design a program that ***recognizes human activities.***

Walking

Running

Cycling

```
if speed<4:
    activity = 'Walking'
```

```
if speed<4:
    activity = 'Walking'
    else:
        activity = 'Running'
```

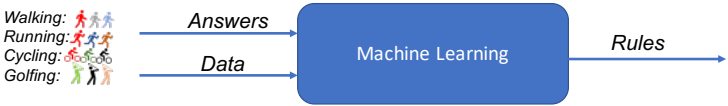
```
if speed<4:
    activity = 'Walking'
    else if speed<12:
        activity = 'Running'
    else:
        activity = 'Cycling'
```

Design a program that ***recognizes human activities.***

Traditional programming with logic rules could be useful to add some intelligence, but they are **limited** and in some applications could be **impossible**!

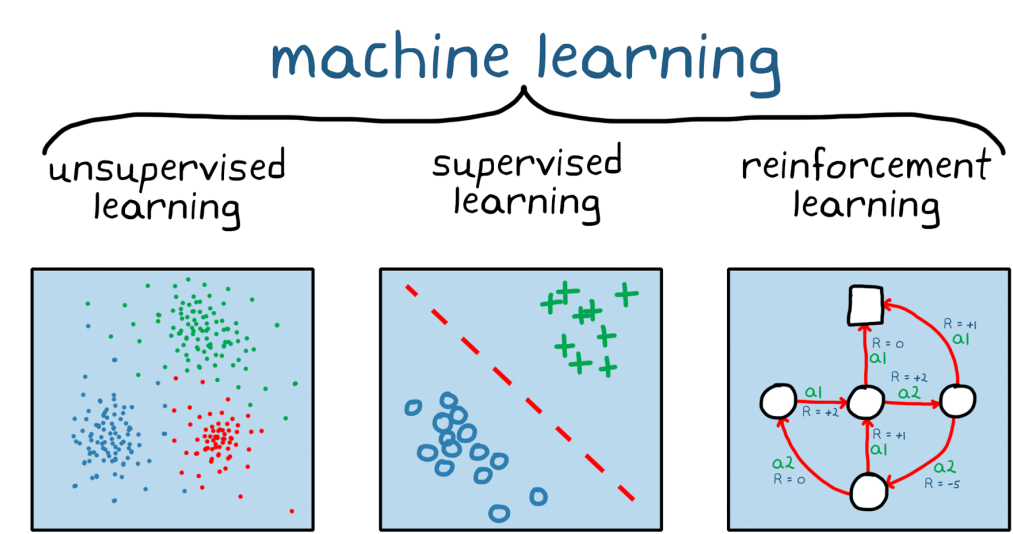
Design a program that ***recognizes human activities.***

Traditional programming with logic rules could be useful to add some intelligence, but they are **limited** and in some applications could be **impossible**!



In ML we have lots and lots of **labelled examples** and we need to use these data to say: this is what **walking** looks like, this is what **running** looks like, etc.

Types of Machine Learning Algorithms



Types of Machine Learning Algorithms

Supervised Learning

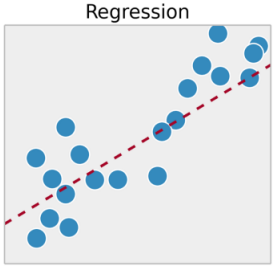
The goal of supervised learning is to learn a function that, given a sample of data and desired outputs, best approximates the relationship between input and output observable in the data.

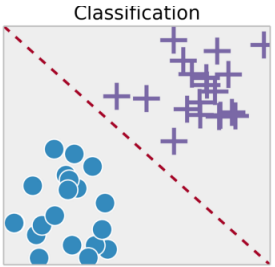
Types of Machine Learning Algorithms

Supervised Learning

The goal of supervised learning is to learn a function that, given a sample of data and desired outputs, best approximates the relationship between input and output observable in the data.

- **1. Regression:** The desire output is a continuous value.
- **2. Classification** The desired output is one of discrete classes.

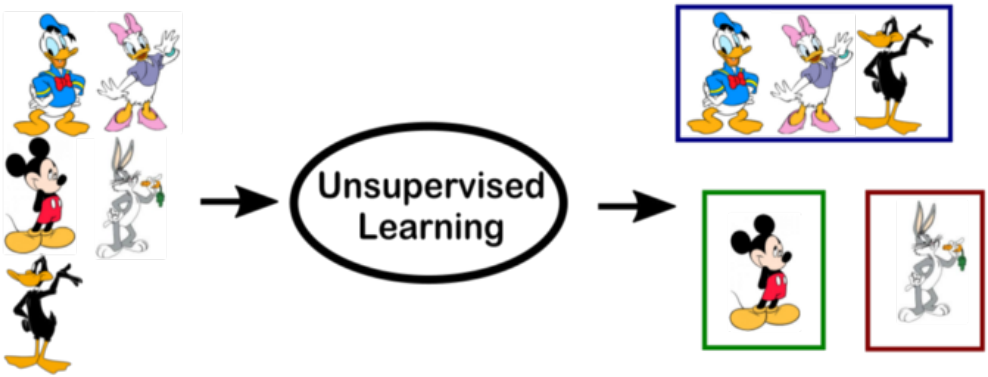




Types of Machine Learning Algorithms

Unsupervised Learning

The goal of unsupervised learning is to learn the inherent structure of our data without using explicitly-provided labels.



Introduction:

Inspiration from Biology:

- The human brain is an incredibly **impressive information processor**, even though it "works" quite a bit slower than an ordinary computer.

Introduction:

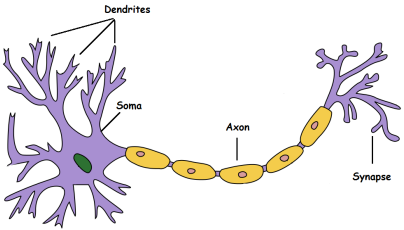
Inspiration from Biology:

- The human brain is an incredibly **impressive information processor**, even though it "works" quite a bit slower than an ordinary computer.
- Biological organisms are capable of **learning gradually over time**. They can learn through exposure to external stimuli and to generalize.

Introduction:

Inspiration from Biology:

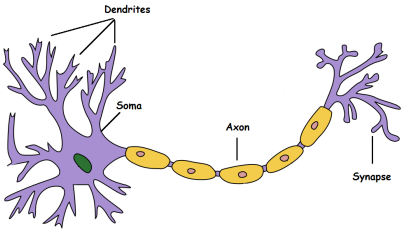
- The human brain is an incredibly **impressive information processor**, even though it "works" quite a bit slower than an ordinary computer.
- Biological organisms are capable of **learning gradually over time**. They can learn through exposure to external stimuli and to generalize.
- Human brain is made of billions of really simple cells called **neurons**.



Structure of a typical neuron

Introduction:
Inspiration from Biology:

- The human brain is an incredibly **impressive information processor**, even though it "works" quite a bit slower than an ordinary computer.
- Biological organisms are capable of **learning gradually over time**. They can learn through exposure to external stimuli and to generalize.
- Human brain is made of billions of really simple cells called **neurons**.
- The neuron is a nerve cell and is the primary functional unit of the nervous system.

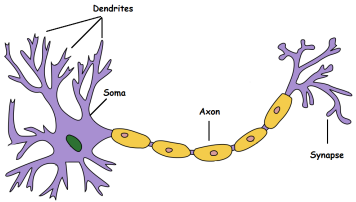


Structure of a typical neuron

Introduction:
Inspiration from Biology:

Basic structure of neuron:

Dendrites: Receives signals from other neurons.



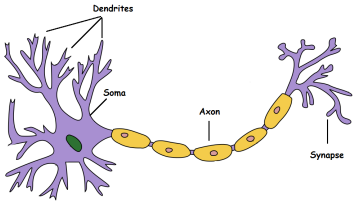
Structure of a typical neuron

Introduction:
Inspiration from Biology:

Basic structure of neuron:

Dendrites: Receives signals from other neurons.

Soma: Processes the information to trigger the output.



Structure of a typical neuron

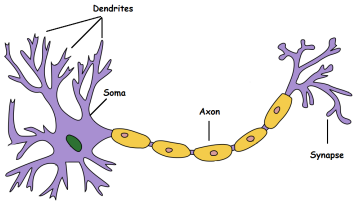
Introduction:
Inspiration from Biology:

Basic structure of neuron:

Dendrites: Receives signals from other neurons.

Soma: Processes the information to trigger the output.

Axon: Transmits the output of this neuron.



Structure of a typical neuron

Introduction:
Inspiration from Biology:

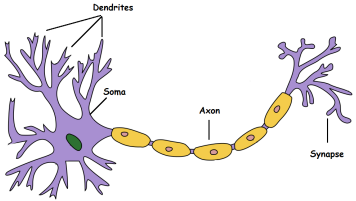
Basic structure of neuron:

Dendrites: Receives signals from other neurons.

Soma: Processes the information to trigger the output.

Axon: Transmits the output of this neuron.

Synapse: Point of connection to other neurons .

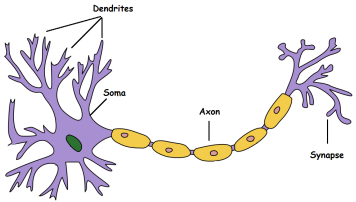


Structure of a typical neuron

Introduction:
Inspiration from Biology:

Basic structure of neuron:

Dendrites:	Receives signals from other neurons.
Soma:	Processes the information to trigger the output.
Axon:	Transmits the output of this neuron.
Synapse:	Point of connection to other neurons .



Structure of a typical neuron

Operation:

Basically, a neuron takes an input signal (dendrite), processes it like the CPU (soma),

Introduction:
Inspiration from Biology:

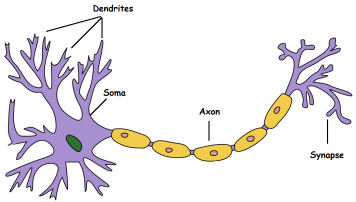
Basic structure of neuron:

Dendrites: Receives signals from other neurons.

Soma: Processes the information to trigger the output.

Axon: Transmits the output of this neuron.

Synapse: Point of connection to other neurons .



Structure of a typical neuron

Operation:

Basically, a neuron takes an input signal (dendrite), processes it like the CPU (soma), if the sum of these signals are more than a certain threshold, the neuron output is activated (fired)

Introduction:
Inspiration from Biology:

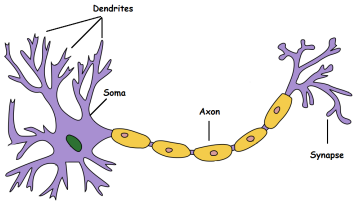
Basic structure of neuron:

Dendrites: Receives signals from other neurons.

Soma: Processes the information to trigger the output.

Axon: Transmits the output of this neuron.

Synapse: Point of connection to other neurons .



Structure of a typical neuron

Operation:

Basically, a neuron takes an input signal (dendrite), processes it like the CPU (soma), if the sum of these signals are more than a certain threshold, the neuron output is activated (fired)and passes the output through a cable-like structure to other connected neurons (axon to synapse to other neuron's dendrite).

Introduction:

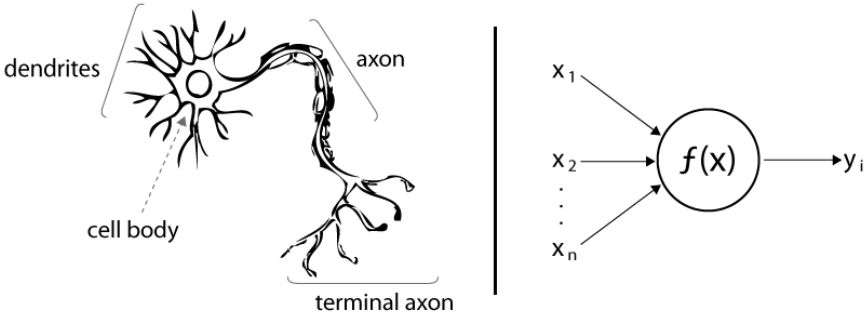
Artificial Neuron: (McCulloch-Pitts model)

The first computational model that mimics the functionality of a biological neuron was proposed by Warren McCulloch (neuroscientist) and Walter Pitts (logician) in 1943.

Introduction:

Artificial Neuron: (McCulloch-Pitts model)

The first computational model that mimics the functionality of a biological neuron was proposed by Warren McCulloch (neuroscientist) and Walter Pitts (logician) in 1943.



Biological Neuron **Artificial Neuron**

Introduction:

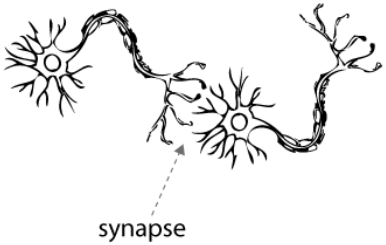
Artificial Neural Network (ANN):

More than one neuron are connected to construct a layer of neurons **Perceptron**. An ANN consists of Multi Layers of Perceptron (MLP).

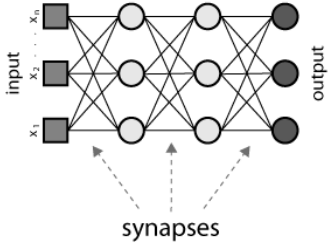
Introduction:

Artificial Neural Network (ANN):

More than one neuron are connected to construct a layer of neurons **Perceptron**. An ANN consists of Multi Layers of Perceptron (MLP).



Biological NN

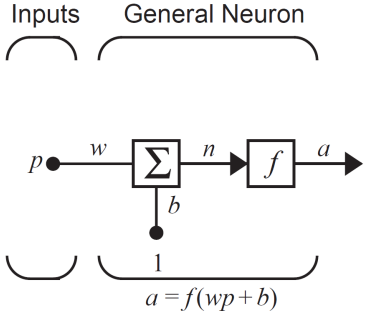


Artificial NN

Multilayer Perceptron Architecture.

Single-Input Neuron:

- The scalar input p is multiplied by the scalar weight w and summed with the input 1 that is multiplied by the bias (offest) b.

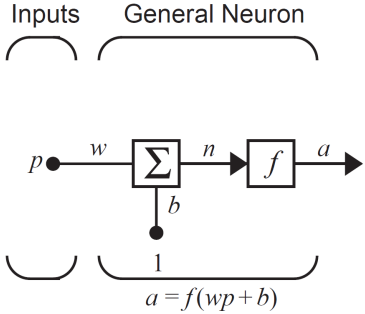


Single-Input Neuron

Multilayer Perceptron Architecture.

Single-Input Neuron:

- The scalar input p is multiplied by the scalar weight w and summed with the input 1 that is multiplied by the bias (offest) b.
- The summer output n, goes into a transfer function (activation function) f that produces the neuron output a.

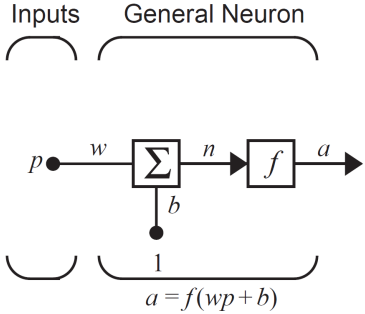


Single-Input Neuron

Multilayer Perceptron Architecture.

Single-Input Neuron:

- The scalar input p is multiplied by the scalar weight w and summed with the input 1 that is multiplied by the bias (offest) b.
- The summer output n, goes into a transfer function (activation function) f that produces the neuron output a.
- The transfer function is chosen by the designer and then the parameters w and b are adjusted by some **learning rule** so that the neuron input/output relationship meets some specific goal.



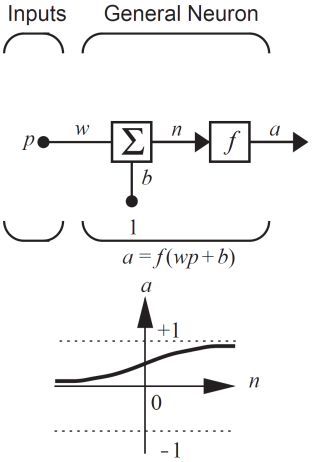
Single-Input Neuron

Multilayer Perceptron Architecture.

Single-Input Neuron:

Activation functions:

It is used to make the output of NN like **yes** or **no**. It maps the input n to a value in between 0 to 1 or -1 to 1 (depending upon the function).



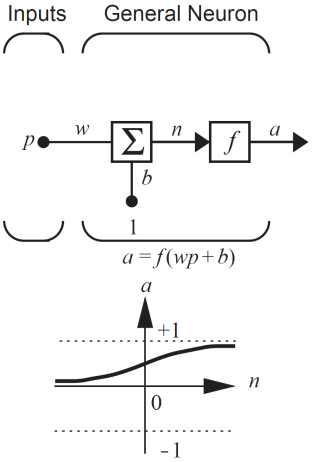
Multilayer Perceptron Architecture.

Single-Input Neuron:

Activation functions:

It is used to make the output of NN like **yes** or **no**. It maps the input n to a value in between 0 to 1 or -1 to 1 (depending upon the function).

- One commonly used functions is the **sigmoid** activation function that adds non-linearity to the neuron.



Multilayer Perceptron Architecture.

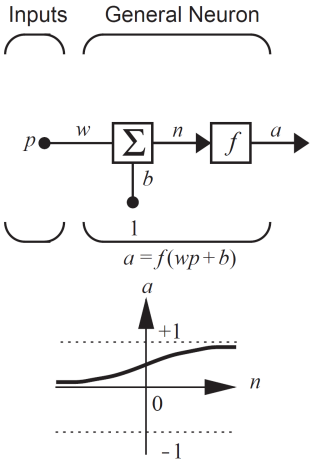
Single-Input Neuron:

Activation functions:

It is used to make the output of NN like **yes** or **no**. It maps the input n to a value in between 0 to 1 or -1 to 1 (depending upon the function).

- One commonly used functions is the **sigmoid** activation function that adds non-linearity to the neuron.
- The sigmoid function takes the input n and squashes the output into the range 0 to 1 according to the expression:

$$a = \frac{1}{1 + e^{-n}} \qquad -\infty < n < +\infty$$



Multilayer Perceptron Architecture.

Single-Input Neuron:

Commonly Used Activation Functions:

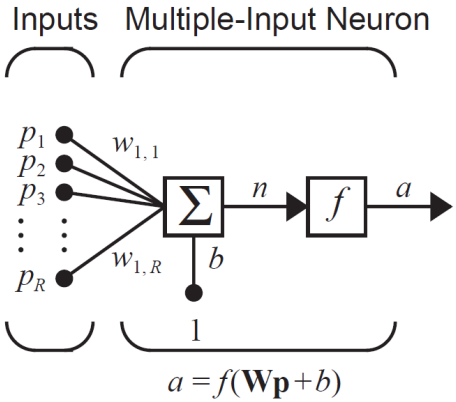
Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	

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

Multilayer Perceptron Architecture.

Multiple-Input Neuron.

- Typically, a neuron has more than one input.



Multiple-Input Neuron

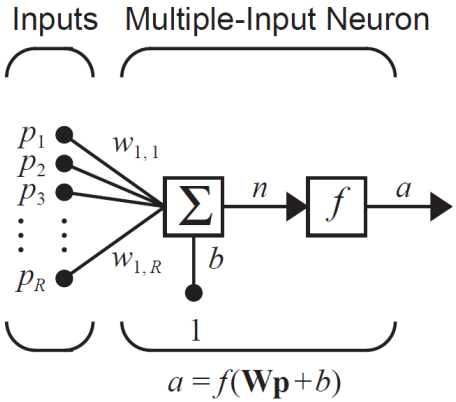
Multilayer Perceptron Architecture.

Multiple-Input Neuron.

- Typically, a neuron has more than one input.
- A neuron with R inputs, $p_1, p_2, \ldots, p_R$ are weighted by $w_{1,1}, w_{1,2}, \ldots, w_{1,R}$.

$$n = w_{1,1}.p_1 + w_{1,2}.p_2 + \ldots + w_{1,R}.p_R + b$$

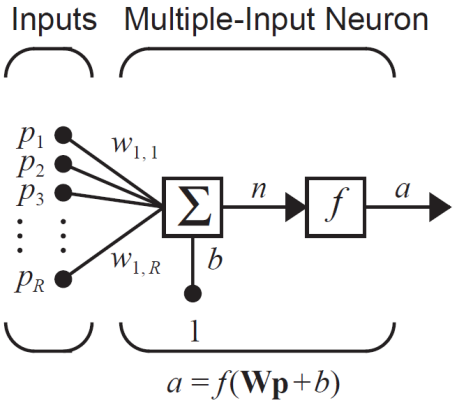
$w_{i,j}$ means that the weight represents the connection to the i neuron from the j source.



Multiple-Input Neuron

Multilayer Perceptron Architecture.

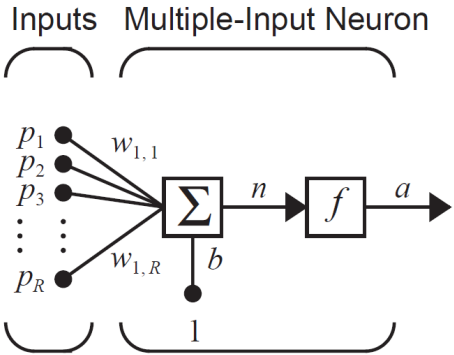
Multiple-Input Neuron.



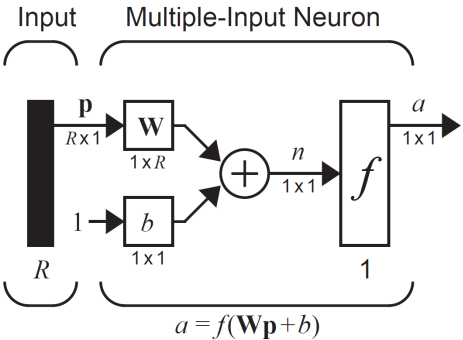
Multiple-Input Neuron

Multilayer Perceptron Architecture.

Multiple-Input Neuron.



Multiple-Input Neuron

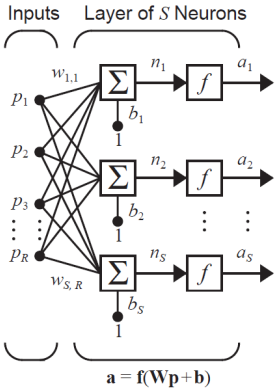


Abbreviated Notation

Multilayer Perceptron Architecture.

A Layer of Neurons.

- A layer of S neurons are connected to the R inputs.

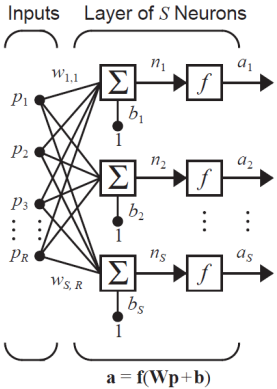


Multiple-Input Neuron

Multilayer Perceptron Architecture.

A Layer of Neurons.

- A layer of S neurons are connected to the R inputs.
- Weight matrix: $\mathbf{W} \in \mathbb{R}^{S \times R}$.
- Bias vector: $\mathbf{b} \in \mathbb{R}^{S}$

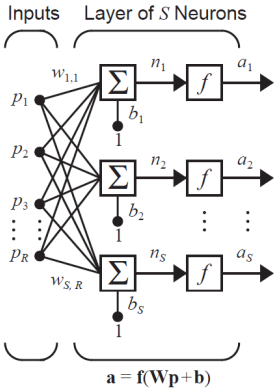


Multiple-Input Neuron

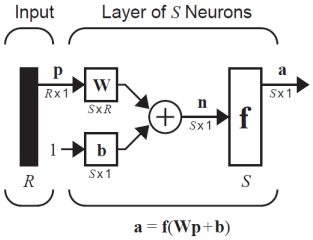
Multilayer Perceptron Architecture.

A Layer of Neurons.

- A layer of S neurons are connected to the R inputs.
- Weight matrix: $\mathbf{W} \in \mathbb{R}^{S \times R}$.
- Bias vector: $\mathbf{b} \in \mathbb{R}^{S}$



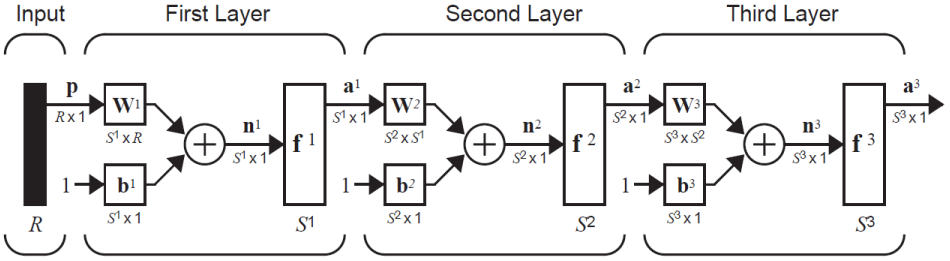
Multiple-Input Neuron



$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

Multilayer Perceptron Architecture.

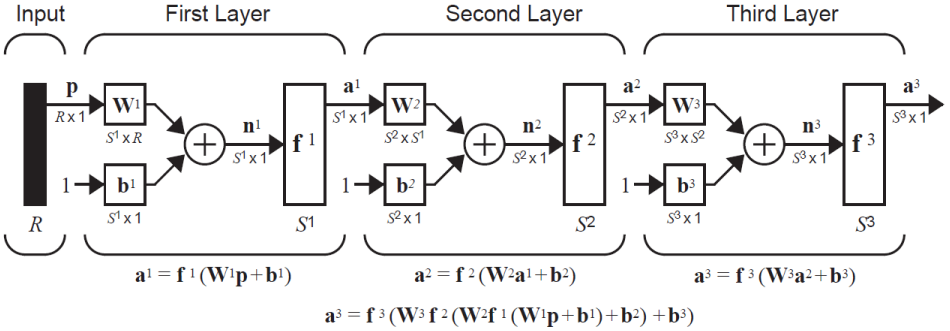
Multiple Layers of Neurons: (Feed-Forward NN)



Three Layers Feed-Forward ANN

Multilayer Perceptron Architecture.

Multiple Layers of Neurons: (Feed-Forward NN)



Three Layers Feed-Forward ANN

The superscript is used to identify the layer. $w^2_{1,2}$ is the weight of the second input in the first neuron of the second layer. Any other layer than the output layer is a **hidden layer**.

Table of Contents

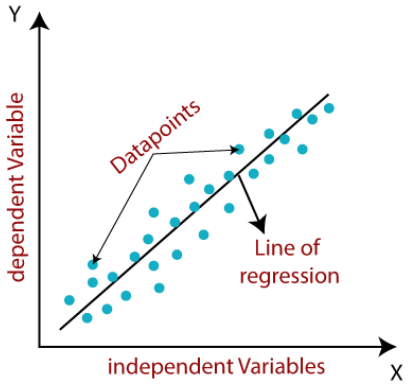
What is Machine Learning?

Machine learning as the field of study that gives computers the ability to learn without being explicitly programmed. Arthur Samuel, 1959

Linear Regression

Introduction

- **Linear regression:** a linear approach for modeling the relationship between a scalar dependent and one or more independent variables.



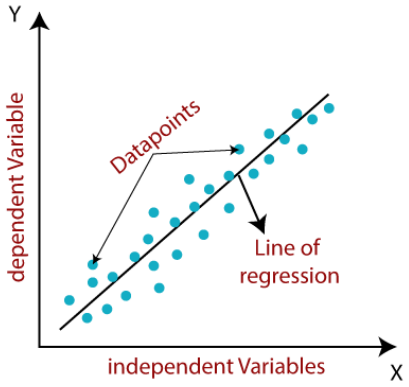
Linear Regression

Introduction

- **Linear regression:** a linear approach for modeling the relationship between a scalar dependent and one or more independent variables.
- Examples:

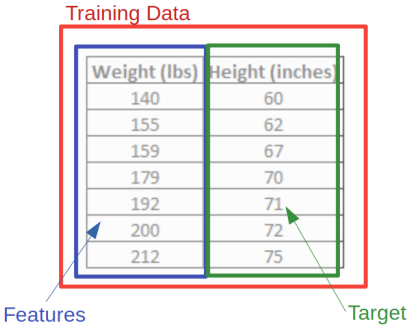
Weight (lbs)	Height (inches)
140	60
155	62
159	67
179	70
192	71
200	72
212	75

Student Name	Hours Studied	Grade
Jack	6	53%
Anne	7	60%
Harry	6,5	56%
Sharon	8	79%
John	6,6	58%
James	8,1	85%
Jill	6,8	70%



Some terminologies

- **Training data**: The dataset used to train a ML model.
- **Features:** The input variable x.
- **Target:** The output to predict y.
- m: Number of training examples.
- (x, y): Single training example.
- $\left(x^{(i)}, y^{(i)}\right)$: ith training example.

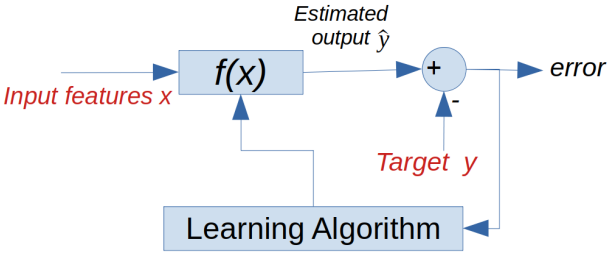


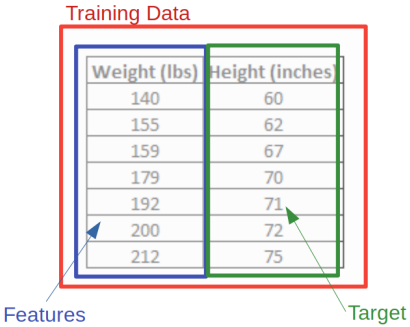
Linear Regression

Training ANN

Training ANN: (Supervised Learning)

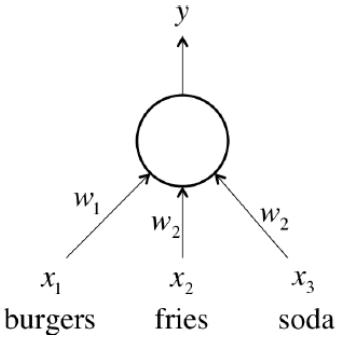
Training an ANN is the process of finding the set of weights and biases of the network that will best approximate a certain function that maps **given samples of inputs and outputs**.





Example: The Fast-Food Problem

Every single day, we purchase a restaurant meal consisting of burgers, fries, and sodas. We buy some number of servings for each item. We want to be able to predict how much a meal is going to cost us, but the items don't have price tags. The only thing the cashier will tell us is the total price of the meal. We want to train a single linear neuron to solve this problem. How do we do it?



$$y = x_1 w_1 + x_2 w_2 + x_3 w_3$$

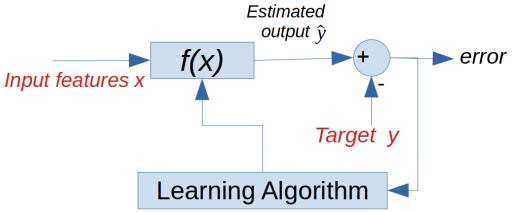
Linear Regression

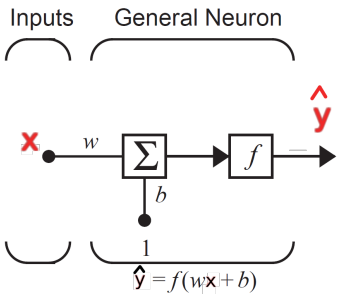
Cost function

- The **cost function** will tell us how well our model is doing.
- A sum of squared errors SSE cost function:

$$\mathbf{F}(w, b) = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right)^2$$

- m is the number of training examples.
- The goal of training a neural network is to minimize **F**.





Linear Regression

Cost function: Example

- Imagine we have the following training data:

x	1	2	3
y	1	2	3

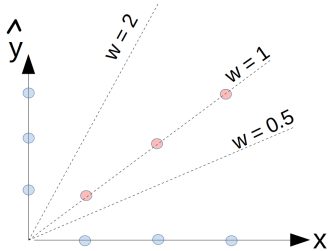
Linear Regression

Cost function: Example

- Imagine we have the following training data:

x	1	2	3
y	1	2	3

- If we have a single neuron $y = wx + b$ and $b = 0$, let's see the effect of the value of w on the estimation $\hat{y}$ and the cost function **F**:



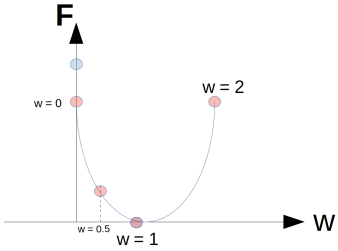
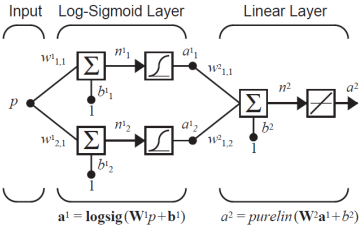


Table of Contents

Back Propagation Algorithm

Gradient Descent

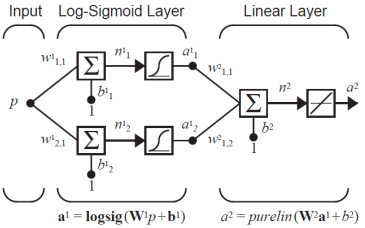
- Back-propagation is an algorithm used for training a multi-layer neural network.



Back Propagation Algorithm

Gradient Descent

- Back-propagation is an algorithm used for training a multi-layer neural network.
- Given a batch of input data $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ and a set of target (output) data $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m$, the goal is to find the network parameters $\mathbf{W}$ and $\mathbf{b}$ that minimizes the cost function $\mathbf{F}$.



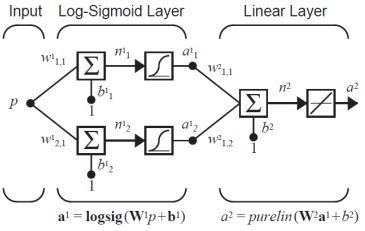
Back Propagation Algorithm

Gradient Descent

- Back-propagation is an algorithm used for training a multi-layer neural network.
- Given a batch of input data $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ and a set of target (output) data $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m$, the goal is to find the network parameters **W** and **b** that minimizes the cost function **F**.

$$\min_{\mathbf{W},\,\mathbf{b}} \mathbf{F}(w, b) = \min_{\mathbf{W},\,\mathbf{b}} \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right)^2$$

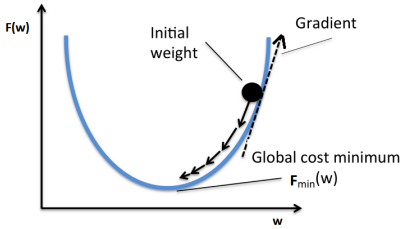
where $(\mathbf{W}, \mathbf{b})$ are the network parameters and $\hat{y}$ is the predicted network output.



Back Propagation Algorithm

Gradient Descent

- The NN parameters **W**, **b** are updated using the **Gradient-Descent** optimization technique to minimize the index **F**.



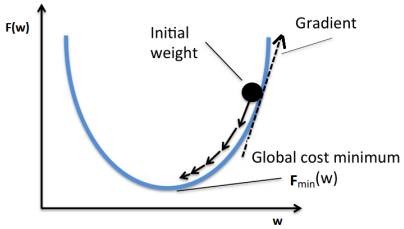
$\alpha = [0, 1]$ is the learning rate

- If α is too **big**, gradient decent may **overshoot**.
- If α is too **small**, gradient decent may **be slow**.

Back Propagation Algorithm

Gradient Descent

- The NN parameters **W**, **b** are updated using the **Gradient-Descent** optimization technique to minimize the index **F**.
- the gradient of **F** w.r.t **W**, **b** points in the direction of the **greatest rate of increase of the function**.



$\alpha = [0, 1]$ is the learning rate

- If α is too **big**, gradient decent may **overshoot**.
- If α is too **small**, gradient decent may **be slow**.

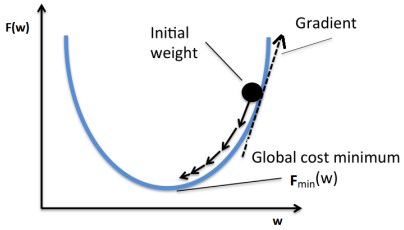
Back Propagation Algorithm

Gradient Descent

- The NN parameters **W**, **b** are updated using the **Gradient-Descent** optimization technique to minimize the index **F**.
- the gradient of **F** w.r.t **W**, **b** points in the direction of the **greatest rate of increase of the function**.
- Thus, the direction of updating **W**, **b** in iteration k is the negative of the gradient direction:

$$w_{i,j}^m(k+1) = w_{i,j}^m(k) - \alpha \frac{\partial \mathbf{F}}{\partial w_{i,j}^m}$$

$$b_{i,j}^m(k+1) = b_{i,j}^m(k) - \alpha \frac{\partial \mathbf{F}}{\partial b_{i,j}^m}$$



$\alpha = [0, 1]$ is the learning rate

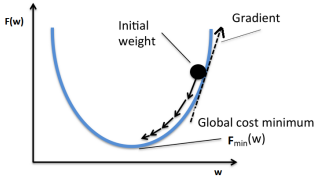
- If α is too **big**, gradient decent may **overshoot**.
- If α is too **small**, gradient decent may **be slow**.

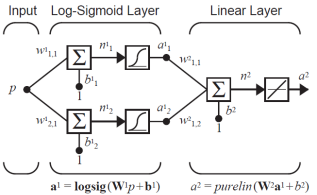
Back Propagation Algorithm

Gradient Descent

$$w_{i,j}^m(k+1) = w_{i,j}^m(k) - \alpha \frac{\partial \mathbf{F}}{\partial w_{i,j}^m}$$

$$b_{i,j}^m(k+1) = b_{i,j}^m(k) - \alpha \frac{\partial \mathbf{F}}{\partial b_{i,j}^m}$$





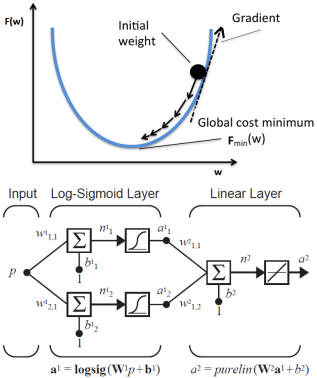
Back Propagation Algorithm

Gradient Descent

$$w^m_{i,j}(k+1) = w^m_{i,j}(k) - \alpha \frac{\partial \mathbf{F}}{\partial w^m_{i,j}}$$

$$b^m_{i,j}(k+1) = b^m_{i,j}(k) - \alpha \frac{\partial \mathbf{F}}{\partial b^m_{i,j}}$$

- **Chain rule** is used to calculate $\frac{\partial \mathbf{F}}{\partial w^m_{i,j}}$ and $\frac{\partial \mathbf{F}}{\partial b^m_{i,j}}$



Back Propagation Algorithm

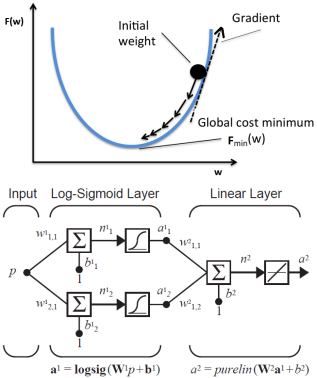
Gradient Descent

$$w^m_{i,j}(k+1) = w^m_{i,j}(k) - \alpha \frac{\partial \mathbf{F}}{\partial w^m_{i,j}}$$

$$b^m_{i,j}(k+1) = b^m_{i,j}(k) - \alpha \frac{\partial \mathbf{F}}{\partial b^m_{i,j}}$$

- **Chain rule** is used to calculate $\frac{\partial \mathbf{F}}{\partial w^m_{i,j}}$ and $\frac{\partial \mathbf{F}}{\partial b^m_{i,j}}$
- Example:

$$\frac{\partial \mathbf{F}}{\partial w^1_{1,1}} = \frac{\partial \mathbf{F}}{\partial a^2} \cdot \frac{\partial a^2}{\partial n^2} \cdot \frac{\partial n^2}{\partial a^1_1} \cdot \frac{\partial a^1_1}{\partial n^1_1} \cdot \frac{\partial n^1_1}{\partial w^1_{1,1}}$$



Training Multilayer Networks

Generalization of ANN

- In supervised learning, we want to build a model on the training data and then be able to **make accurate predictions on new, unseen** data (generalization).

Training Multilayer Networks

Generalization of ANN

- In supervised learning, we want to build a model on the training data and then be able to **make accurate predictions on new, unseen** data (generalization).
- The only measure of whether an algorithm will perform well on new data is the evaluation on the **test set**.

Training Data

Testing Data

Training Multilayer Networks

Generalization of ANN

- In supervised learning, we want to build a model on the training data and then be able to **make accurate predictions on new, unseen** data (generalization).
- The only measure of whether an algorithm will perform well on new data is the evaluation on the **test set**.
- Use 70% of the data for training and 30% for testing.

Training Data

Testing Data

Training Multilayer Networks:
Generalization capability of ANN:

Overfitting problem

Overfitting is defined as fitting a model to a particular training data set with no guarantee it will provide good prediction performance on **unseen test data**, even if the model predicts the targets on the training data perfectly.

Rule of thumb

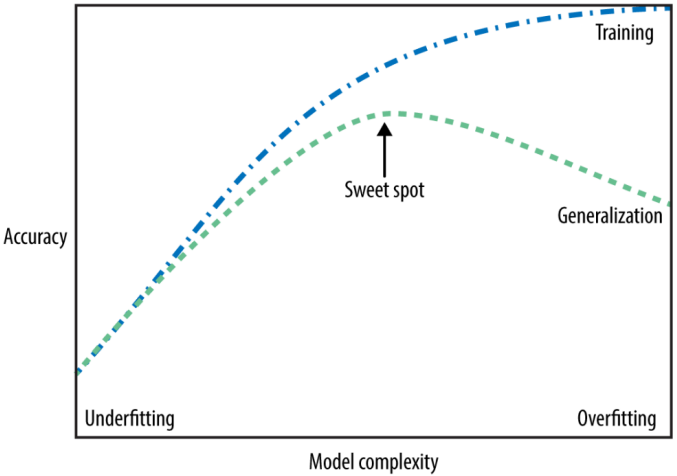
- Enough neurons should be used to capture the complexity of the underlying function without having the network **overfit** the training data, in which case it will not generalize to new situations.

Over-fitting Problem

Training Multilayer Networks:

Generalization capability of ANN:

Trade-off of model complexity against training and test accuracy

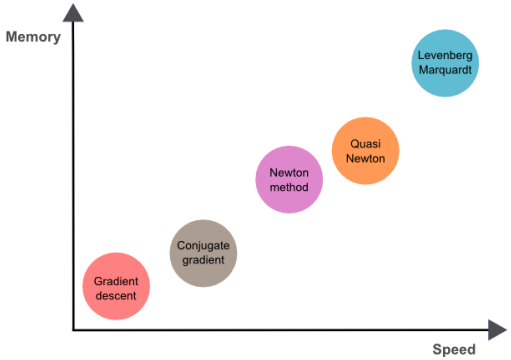


Training Multilayer Networks:
Other Learning Algorithms:

- There are many different training algorithms for updating the NN parameters other than **Gradient-descent**.

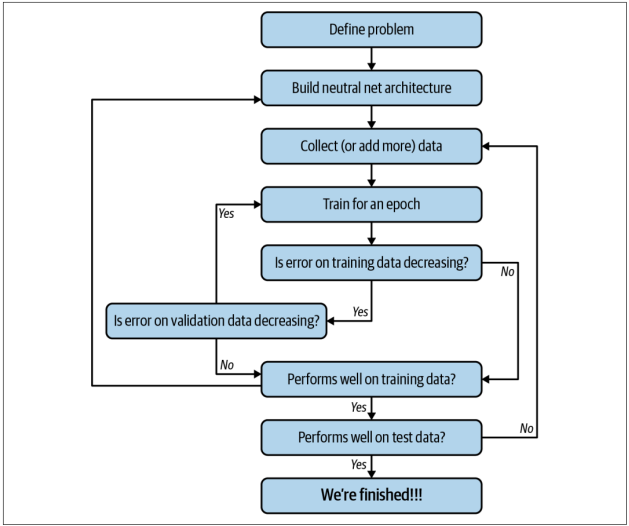
Training Multilayer Networks:
Other Learning Algorithms:

- There are many different training algorithms for updating the NN parameters other than **Gradient-descent**.
- **Levenberg-Marquardt** is often the fastest backpropagation algorithm. It is highly recommended as a first-choice supervised algorithm, although it does require more memory than other algorithms.



Training Multilayer Networks:

Detailed workflow for training and evaluating an ANN model



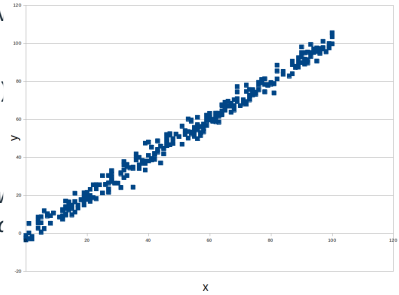
Training Multilayer Networks:

Simple Linear Regression Examples in MATLAB

```
clear all; clc; close all;

%% Load datasets:
train_data = readtable('simple_train.csv
features = [train_data.x'];
targets = [train_data.y'];
test_data = readtable('simple_test.csv');
test_features = [test_data.x'];
test_targets = [test_data.y'];
%% Build the fitnet:
net = fitnet([4]); % One hidden layer
% net = fitnet([10 8]); % Two hidden l
view(net); % Preview the network
%% Train the network:
[net, tr] = train(net, features, targets);

%% Test the network:
prediction = net(test_features);
error = gsubtract(test_targets, prediction); % element-wise substraction
performance = perform(net, test_targets, prediction)
```



$$y = x + \eta$$

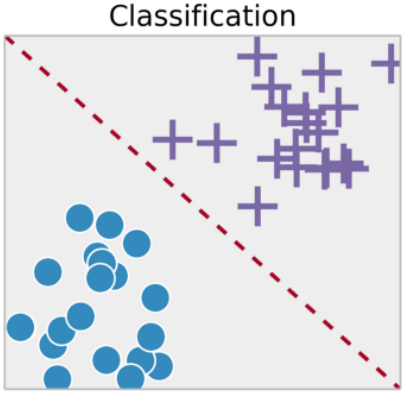
Table of Contents

Classification with Logistic Regression

Classification

In classification, the output (target) is one of two or more possible (discrete) values.

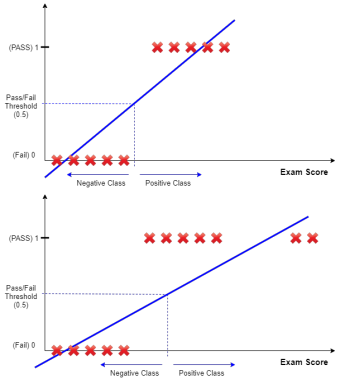
Binomial	Multinomial	Ordered
Only two possible values for the response variable.	Three or more values for the response variable.	Response values have an inherent order.



Classification with Logistic Regression

Logistic Regression

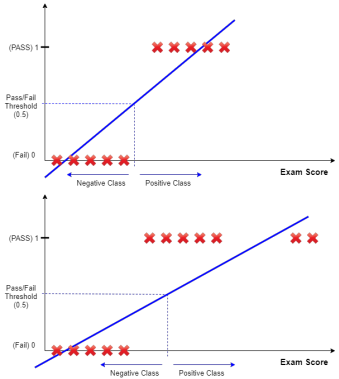
- We can used linear regression in classification and put a threshold to decide if yes or no.



Classification with Logistic Regression

Logistic Regression

- We can used linear regression in classification and put a threshold to decide if yes or no.
- But this adjustment also do not work if there are outliers in dataset.

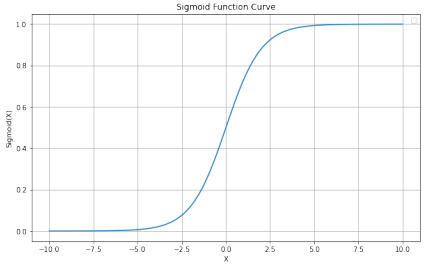


Classification with Logistic Regression

Sigmoid Function

Sigmoid activation function

- Since our objective is to get discrete value (0 or 1) we will create a function that will return values between 0 and 1.
- **Sigmoid** function do exactly that, it maps the whole real number range between 0 and 1. It is also called as **Logistic function.**



$$g(z) = \frac{1}{1 + e^{-z}} \quad 0 < g(z) < 1$$

$$z = xw + b$$

Classification with Logistic Regression

Cost Function

- The squared error cost function used in linear regression is not ideal in logistic regression.

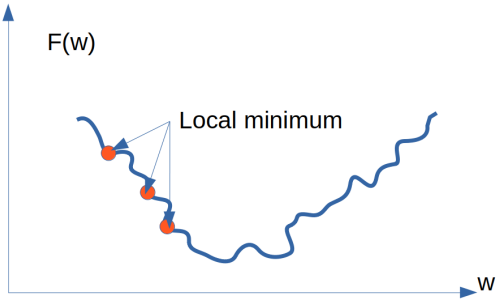
Object	A1	A2 (Female, Male)	A3	Target Class
1	F	F	1.0	F
2	T	M	6.0	T
3	T	F	5.0	F
4	F	F	4.0	F
5	T	F	5.0	T
6	F	F	7.0	T
7	F	M	3.0	T
8	T	F	8.0	F
9	T	M	7.0	T

Example of binary classification

Classification with Logistic Regression

Cost Function

- The squared error cost function used in linear regression is not ideal in logistic regression.
- If we used the squared error F, the cost will change with the weights like this in logistic regression:



Object	A1	A2 (Female, Male)	A3	Target Class
1	F	F	1.0	F
2	T	M	6.0	T
3	T	F	5.0	F
4	F	F	4.0	F
5	T	F	5.0	T
6	F	F	7.0	T
7	F	M	3.0	T
8	T	F	8.0	F
9	T	M	7.0	T

Example of binary classification

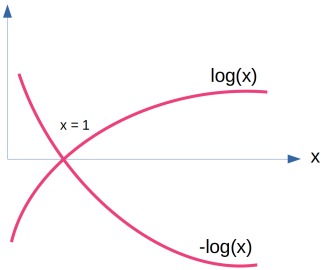
Classification with Logistic Regression

Cost Function

- Logistic loss function (Cross entropy)

$$\boldsymbol{F} = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

$$L(\hat{y}^{(i)}, y^{(i)}) = \begin{cases} -\log(\hat{y}^{(i)}), & \text{if } y^{(i)} = 1. \\ -\log(1-\hat{y}^{(i)}), & \text{if } y^{(i)} = 0. \end{cases}$$



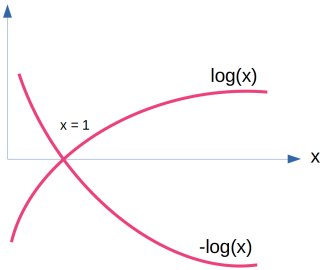
Classification with Logistic Regression

Cost Function

- Logistic loss function (Cross entropy)

$$\boldsymbol{F} = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

$$L(\hat{y}^{(i)}, y^{(i)}) = \begin{cases} -\log(\hat{y}^{(i)}), & \text{if } y^{(i)} = 1. \\ -\log(1-\hat{y}^{(i)}), & \text{if } y^{(i)} = 0. \end{cases}$$



- Equivalent version

$$\boxed{L(\hat{y}^{(i)}, y^{(i)}) = -y^{(i)} \log \hat{y}^{(i)} - (1-y^{(i)}) \log (1-\hat{y}^{(i)})}$$

Table of Contents

What is TensorFlow 2 ?

TensorFlow

TensorFlow is an open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers and developer push the state-of-the-art in ML applications.

What is TensorFlow 2 ?

TensorFlow

TensorFlow is an open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers and developer push the state-of-the-art in ML applications.

Easy model building

Build and train ML models easily using intuitive high-level APIs like Keras with eager execution, which makes for immediate model iteration and easy debugging.

Robust ML production anywhere

Easily train and deploy models in the cloud, on-prem, in the browser, or on-device no matter what language you use.

Powerful experimentation for research

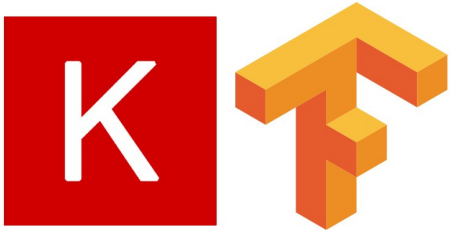
A simple and flexible architecture to take new ideas from concept to code, to state-of-the-art models, and to publication faster.

Source:

www.tensorflow.org

What is TensorFlow 2 ?

Recently **Keras** is the high-level API of TensorFlow 2.



Companies use TensorFlow.

Source:

www.tensorflow.org

Table of Contents

TensorFlow Quickstart.

Installation of TensorFlow: Two options:

- Install TensorFlow with pip (local installation).
- Using Google Colab: An easy way to learn and use TensorFlow

TensorFlow Quickstart.

Installation of TensorFlow: Two options:

- Install TensorFlow with pip (local installation).
- Using Google Colab: An easy way to learn and use TensorFlow

Powered by TensorFlow

Helping doctors detect respiratory diseases using machine learning.

TensorFlow Quickstart: Celsius to Fahrenheit

- The problem we will solve is to convert from Celsius c to Fahrenheit f.
- We will give TensorFlow some sample Celsius values (0, 8, 15, 22, 38) and their corresponding Fahrenheit values (32, 46, 59, 72, 100).
- Then, we will train a model that figures out the above formula through the training process.

$$f = c \times 1.8 + 32$$

open in colab

TensorFlow Quickstart: Celsius to Fahrenheit

Import dependencies:

```
import tensorflow as tf
import numpy as np
```

TensorFlow Quickstart: Celsius to Fahrenheit

Import dependencies:

```
import tensorflow as tf
import numpy as np
```

Check and confirm TF version:

```
print("TensorFlow version:", tf.__version__)
```

TensorFlow Quickstart: Celsius to Fahrenheit

Import dependencies:

```
import tensorflow as tf
import numpy as np
```

Check and confirm TF version:

```
print("TensorFlow version:", tf.__version__)
```

Set up training data:

```
celsius_q = np.array([-40, -10, 0, 8, 15, 22, 38], dtype=float)
fahrenheit_a = np.array([-40, 14, 32, 46, 59, 72, 100], dtype=float)
for i,c in enumerate(celsius_q): print(" degrees Celsius =  degrees Fahrenheit".format(c, fahrenheit_a[i]))
```

TensorFlow Quickstart: Celsius to Fahrenheit

Create the machine learning model:

```
layer_0 = tf.keras.layers.Dense(units=1, input_shape=[1])      # one layer, one neuron NN
```

TensorFlow Quickstart: Celsius to Fahrenheit

Create the machine learning model:

```
layer_0 = tf.keras.layers.Dense(units=1, input_shape=[1])      # one layer, one neuron NN
```

Assemble layers into the model:

```
model = tf.keras.Sequential([layer_0])
```

Note: layers could be defined inside Sequential() function.

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

TensorFlow Quickstart: Celsius to Fahrenheit

Compile the model:

model.**compile**(loss='mean_squared_error',
optimizer=tf.keras.optimizers.Adam(0.1))

The loss function ([mean squared error]) and the optimizer is 'Adam' with learning rate = 0.1

TensorFlow Quickstart: Celsius to Fahrenheit

Compile the model:

model.**compile**(loss='mean_squared_error',
optimizer=tf.keras.optimizers.Adam(0.1))

The loss function ([mean squared error]) and the optimizer is 'Adam' with learning rate = 0.1

Train the model:

history = model.**fit**(celsius_q, fahrenheit_a, epochs=500, verbose=False)
print("Finished training the model")

TensorFlow Quickstart: Celsius to Fahrenheit

Display training statistics:

```
import matplotlib.pyplot as plt
plt.xlabel('Epoch Number')
plt.ylabel("Loss Magnitude")
plt.plot(history.history['loss'])
```

TensorFlow Quickstart: Celsius to Fahrenheit

Display training statistics:

```
import matplotlib.pyplot as plt
plt.xlabel('Epoch Number')
plt.ylabel("Loss Magnitude")
plt.plot(history.history['loss'])
```

Use the model for prediciton:

```
print(model.predict([100.0]))
```

Answer: 211.28163

TensorFlow Quickstart: Celsius to Fahrenheit

Look into the layer's weights:

```
print("These are the layer variables: ".format(layer_0.get_weights()))
```

TensorFlow Quickstart: Celsius to Fahrenheit

Look into the layer's weights:

```
print("These are the layer variables: ".format(layer_0.get_weights()))
```

Change the model's number of layers:

```
model = tf.keras.Sequential([
tf.keras.layers.Dense(units=4, input_shape=[1])
tf.keras.layers.Dense(units=4)
tf.keras.layers.Dense(units=1)
])
model.compile(loss='mean_squared_error', optimizer=tf.keras.optimizers.Adam(0.1))
model.fit(celsius_q, fahrenheit_a, epochs=500, verbose=False)
print(model.predict([100.0]))
```

Answer: 211.74745

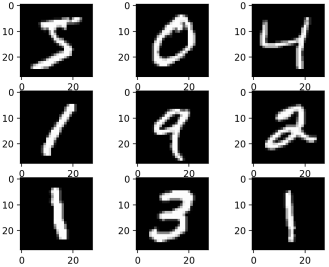
Table of Contents

MNIST Classification with TensorFlow.

We will discuss how to:

- Load a pre-built dataset.
- Build a neural network machine learning model that classifies MNIST images.
- Train this neural network.
- Evaluate the accuracy of the model.

open in colab



MNIST Dataset

MNIST Classification with TensorFlow.

Import dependencies:

```
import tensorflow as tf
import numpy as np
```

MNIST Classification with TensorFlow.

Import dependencies:

```
import tensorflow as tf
import numpy as np
```

Load a dataset.

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

MNIST Classification with TensorFlow.

Build a machine learning model:

```
model = tf.keras.models.Sequential([
tf.keras.layers.Flatten(input_shape=(28, 28)),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(10)
])
```

MNIST Classification with TensorFlow.

Build a machine learning model:

```
model = tf.keras.models.Sequential([
tf.keras.layers.Flatten(input_shape=(28, 28)),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(10)
])
```

Define a loss function .

```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

MNIST Classification with TensorFlow.

Binary cross-entropy:

It is intended to use with binary classification where the target value is 0 or 1. It will calculate a difference between the actual and predicted probability distributions for predicting class 1.

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N} y_i \ \log \hat{y}_i + (1 - y_i) \ \log (1 - \hat{y}_i)$$

MNIST Classification with TensorFlow.

Categorical cross-entropy:

- It is the default loss function to use for multi-class classification problems.
- It will calculate the average difference between the actual and predicted probability distributions.

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N} y_i \ \log \hat{y}_i$$

- The target need to be **one-hot encoded**.

MNIST Classification with TensorFlow.

One-hot Encoding:

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

MNIST Classification with TensorFlow.

Sparse Categorical cross-entropy:

- One hot encoding is frustrating when using cross-entropy with classification problems with **a large number of labels like the 1000 classes.**

if you use categorical-cross-entropy you need one-hot encoding, and if you use sparse-categorical-cross-entropy you encode as normal integers.

MNIST Classification with TensorFlow.

Compile the model:

```
model.compile(optimizer='adam',
loss=loss_fn,
metrics=['accuracy'])
```

MNIST Classification with TensorFlow.

Compile the model:

```
model.compile(optimizer='adam',
loss=loss_fn,
metrics=['accuracy'])
```

Train your model:

```
model.fit(x_train, y_train, epochs=5)
```

MNIST Classification with TensorFlow.

Compile the model:

```
model.compile(optimizer='adam',
loss=loss_fn,
metrics=['accuracy'])
```

Train your model:

```
model.fit(x_train, y_train, epochs=5)
```

Evaluate your model:

```
model.evaluate(x_test, y_test, verbose=2)
```

MNIST Classification with TensorFlow.

Add a softmax layer to return the probability:

```
probability_model = tf.keras.Sequential([
model,
tf.keras.layers.Softmax()
])
```

Check the new model:

```
probability_model(x_test[:5])
```

End of Lecture

haitham.elhussieny@ejust.edu.eg