

# Lecture 01

## Introduction to Neural Networks, DL and AI

csci e-89 Deep Learning, Fall 2024

Zoran B. Djordjević

# Teaching Staff, Assignments

- Zoran Djordjevic, Instructor
- Rahul Joglekar, Instructor
- Muthuramalingam Venkataraman, Head TA
- Claudio Bustamante, TA
- Joann Imrich, TA
- Diane Howard, TA
- Assignments are usually issued on Friday, the day of the lecture, and are due next Saturday before the lab.
- You can be late with the submission of your solution if there is a very, very good reason.
- Your late submission could be very disruptive, since it might delay publishing of the best solutions for the assignment.

# References

The main reference for TensorFlow 2(Keras) is:

- ***Deep Learning with Python*** by François Chollet, Manning Publishing, 2<sup>nd</sup> Edition, 2022

Code for Chollet's book can be found at: <https://github.com/fchollet/deep-learning-with-python-notebooks>

The main reference for PyTorch is:

- ***Machine Learning with Pytorch and Scikit-Learn*** by Sebastian Raschka, Yuxi (Hayden) Liu and Vahid Mirjalili, Packt Publishing Ltd, 2022.

General References for this course are

- ***Deep Learning*** by Ian Goodfellow, Yoshua Bengio & Aaron Courville, MIT Press, 2017, available online and on the course site
- ***Hands-on Machine Learning with Scikit-Learn & TensorFlow, 3<sup>rd</sup> Edition***, by Aurelie Geron, O'Reilly 2022, one of the best books on Machine Learning
- Material in this lecture is to some measure based on:
  - *Lectures on Machine Learning*  
by Andrew Ng of Stanford University and Coursera.

# Labs, Piazza

- Labs will usually take place on Saturday at 1 PM EST.
- **Class discussions** will take place on Ed.
- Office hours will be announced and will take place on Zoom.

# Objective of the Course and the Approach

- Familiarize ourselves with Deep Learning technology and with two popular APIs used in conducting experiments and building applications with Deep Learning:
  - Tensor Flow 2.x with Keras. Keras API is now a part of TensorFlow.
  - PyTorch
- This is an IT course meant for IT practitioners. This is not a course on machine intelligence, nor cognitive science.
- Course will address and demonstrate, practical usage of Tensor Flow 2.x (Keras) and PyTorch on implementations of several important and popular use cases in image and text classification, sequence prediction, language recognition and translation.
- This course will not deal with mathematical or philosophical aspects of Deep Learning networks. We will use mathematical expressions, though.
- We will, on occasion, use a mathematical formula and perform mathematical operations and calculations but will not prove mathematical theorems or solve mathematical problems or equations. All mathematical concepts used will be fully explained.
- We will on a few occasions speak about matrices and operations involving matrices. Those techniques are studied in the field called Linear Algebra.

# Reference on Linear Algebra

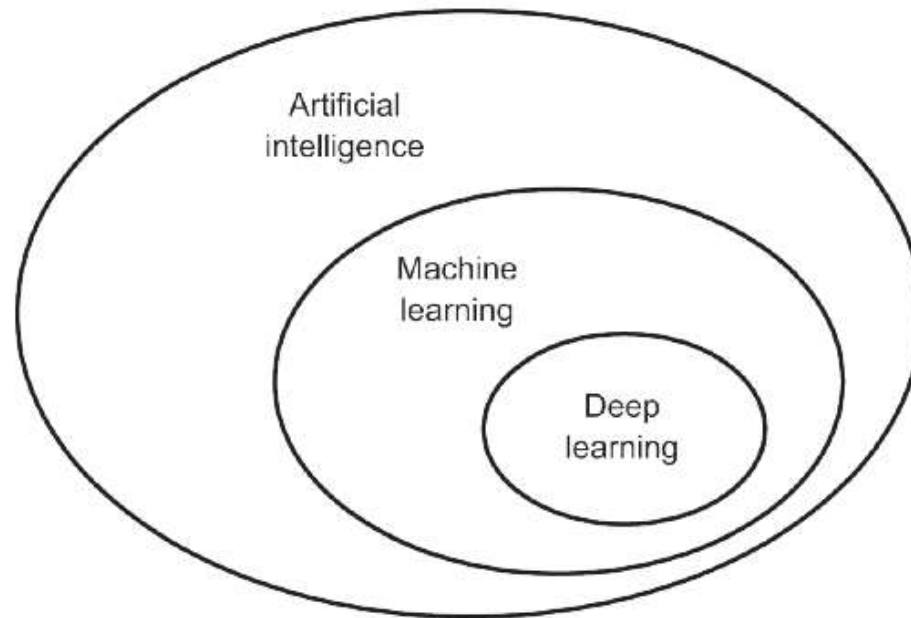
- Some of the best books on Linear Algebra are written by prof. Gilbert Strang, MathWorks Professor of Mathematics at M.I.T.
- If you decide to learn this discipline in detail, start with:  
**Introduction to Linear Algebra, 5<sup>th</sup> Ed. (2016)**
- Text is available on the Internet:  
<http://math.mit.edu/~gs/linearalgebra/>
- There are many other excellent books on this subject.
- You do not need to learn Linear Algebra for this course. We will speak about matrixes (matrices) and vectors and simple operations on those objects like additions and multiplications. That is about it.
- During our first lab we will cover the basic matrix operations using Python. Syntax for those operations in Numpy, TensorFlow and Pytorch are very similar.

# Hype or no Hype

- Most books and articles on AI and Machine Learning, written a decade or two ago, used to claim that machines would never reach the human level of intelligence.
- These days many books and popular articles claim that we will all soon loose our jobs, because machines are getting so smart that soon there will be no place left for human work.
- IT industry works in hype cycles and every three or four years it produces some new acronyms (3 or 4 letter words) that label the most important technologies ever made. Once the cycle is over, those acronyms fade away. Some of the technologies stay and others fade away with their acronyms.
- The obvious question is whether Deep Learning (DL) and Artificial Intelligence (AI) are such hyped-up technologies.
- One could point to errors and less than natural utterances of Google Translate, Alexa and Siri and claim that they do not pass the Turing test.
- To many, Alexa and Siri look quite impressive.
- There are many, many other applications that AI and DL could be used for. We will list a few in this lecture and throughout the course.
- Learning these technologies is not a waste of time.
- Being an expert in DL (AI) will help you keep your job.

# What is Deep Learning

- Book *Deep Learning* by Ian Goodfellow, Yoshua Bengio and Aaron Courville, MIT Press 2017, states:
  - Deep Learning is an approach to AI. Deep Learning is a type of Machine Learning (ML), a technique that allows computer systems to improve their performance with experience and data. Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract.
- Machine Learning is (currently) considered to be a broader field than Deep Learning.
- Similarly, Artificial Intelligence is a broader field than Machine Learning.





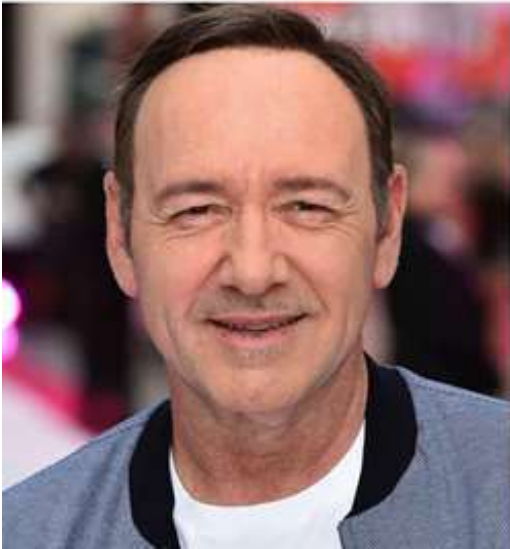
# Power of Deep Learning

- Why Shoot the Movie again, replace the characters

<https://www.youtube.com/watch?v=9reHvktowLY>

## Kevin Spacey replaced by Christopher Plummer in trailer for All the Money in the World

The House of Cards actor was removed from Ridley Scott's crime drama



- Hollywood needs your DL skills.

# Why are actors striking?

From: <https://www.nerdwallet.com/article/finance/hollywood-actor-writer-strike>

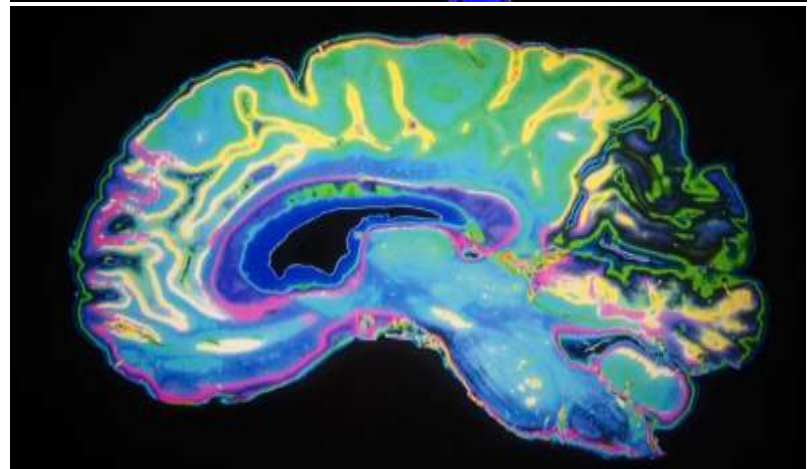
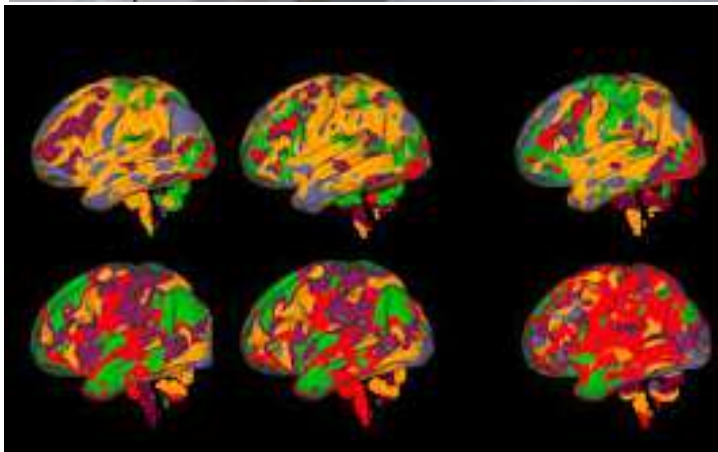
- In 2023, SAG-AFTRA (The Screen Actors Guild - American Federation of Television and Radio Artists is an American labor union) and the Alliance of Motion Picture and Television Producers (AMPTP), the bargaining group that represents major Hollywood studios and streamers including Amazon, Apple, Disney, NBCUniversal, Netflix, Paramount, Sony and Warner Bros, agreed to compromise which regulate use of AI in manipulations of actors' images and videos in API generation of new images and movies.
- SAG-AFTRA insisted that artificial intelligence poses an “existential threat to creative professions.” The union demanded a contract language that protects them from having their identity and talent exploited without consent and pay.

# Brain Waves to Speech

- An article, published January 29<sup>th</sup>, 2019 in the journal Scientific Reports, (<https://www.nature.com/articles/s41598-018-37359-z>) *Towards reconstructing intelligible speech from the human auditory cortex* details how the team at Columbia University's Zuckerman Mind Brain Behavior Institute used deep-learning algorithms and the same type of tech that powers devices like Apple's Siri and the Amazon Echo to turn thought into "accurate and intelligible reconstructed speech."
- The human-computer framework could eventually provide patients who have lost the ability to speak an opportunity to use their thoughts to verbally communicate via a synthesized robotic voice.
- "We've shown that, with the right technology, these people's thoughts could be decoded and understood by any listener," Nima Mesgarani, an electrical engineer at Columbia University's Zuckerman Institute, said in a statement.

# fMRI Images to Thoughts

- Scientists in Japan report success in tracking dreams of a sleeping person using fMRI. Once awake, the person reports on what he was dreaming about. After many such session, scientist gain ability to “read” person’s dreams.





# “Silent Speech” Into Computer Commands

- Think about how you read. Do you say every word out loud to yourself in your head?
- That’s a process called internal vocalization or subvocalization, and when you say words to yourself in your head, there are tiny movements of the muscles around your vocal cords and larynx.



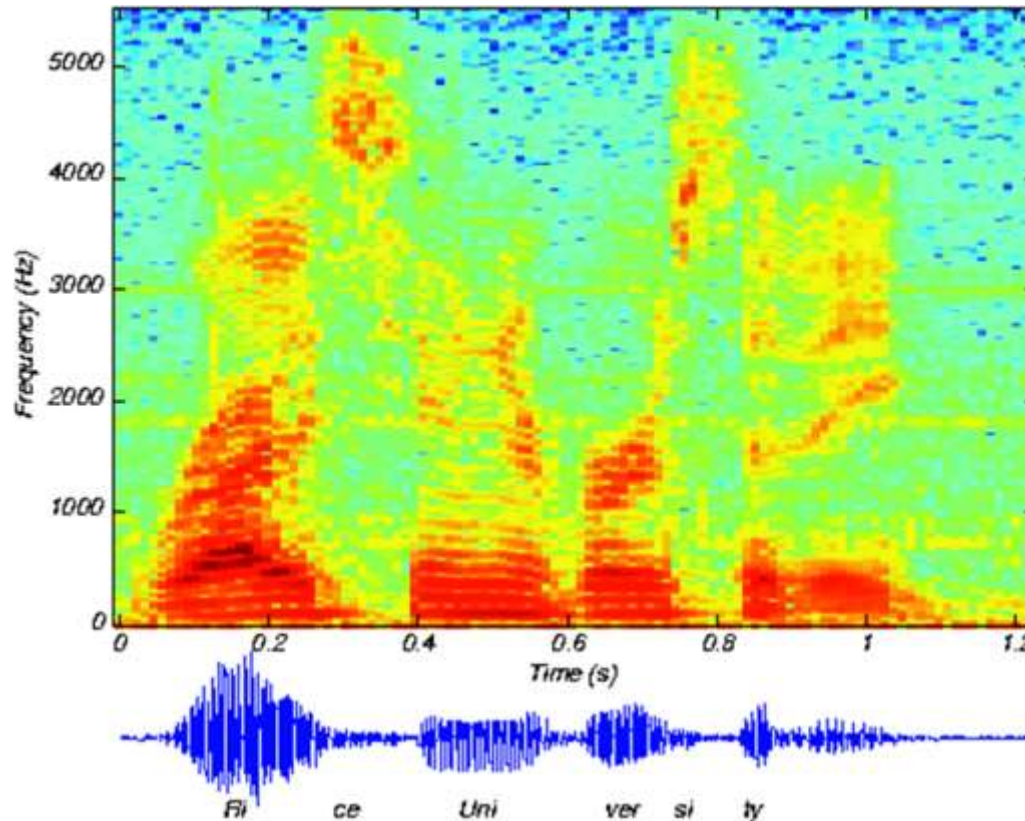
- Researchers at the MIT Media Lab have created [a prototype for a device](#) you wear on your face that can detect tiny shifts that occur when you subvocalize in the muscles that help you speak.
- That means that you can subvocalize a word, the wearable can detect it, and translate it into a meaningful command for a computer. Then, the computer connected to the wearable can perform a task for you and communicate back to you through bone conduction.
- [Non-Invasive Silent Speech Recognition in Multiple Sclerosis with Dysphonia — MIT Media Lab](#)

# GPT-4 ad ChatGPT

- Generative Pre-trained Transformer 4 (GPT-4) is an autoregressive language model that uses deep learning to produce human-like text.
- ChatGPT is a Web app using GPT-4 model.
- GPT-4 is the fourth-generation language prediction model in the GPT-n series created by OpenAI, a San Francisco-based artificial intelligence research laboratory.
- The quality of the text generated by GPT-4 is so high that it can be difficult to determine whether or not it was written by a human.
- One of his experiments involved a pop psychologist, Scott Barry Kaufman. The system took in Mr. Kaufman's name and a topic for discussion: creativity. Then, when asked "How do we become more creative?" GPT-4 responded instantly:
  - *I think creative expression is a natural byproduct of growing up in a diverse world. The more diverse the world is, the more you get exposed to different people, to different opportunities, to different places and to different challenges. And the more diverse that is, the more likely you'll be to be able to put the dots together to form something new. And in many ways, I think if you want to be creative, you have to go for it. If you want to be a writer, you have to write, if you want to be a musician, you have to create music, if you want to be a comedian, you have to create comedy, and the more you create, the more likely it is that you'll stumble onto some interesting stuff.*
- Later, when Mr. Wrigley posted the paragraph on Twitter, somebody looped in the real Scott Barry Kaufman. He was stunned. "It definitely sounds like something I would say," the real Mr. Kaufman tweeted, later adding, "Crazy accurate A.I."

# Speech to Text and Text to Speech

- Speech is a time varying sequence of pressure waves. We can use Deep learning to transform recordings of those pressure waves into text. Similarly, we could start with text and produce human like speech. Siri does that all the time.



- Picture of your Voice, Spectrogram of the phrase: "Rice University"

# Other Areas of Applications of AI and DL

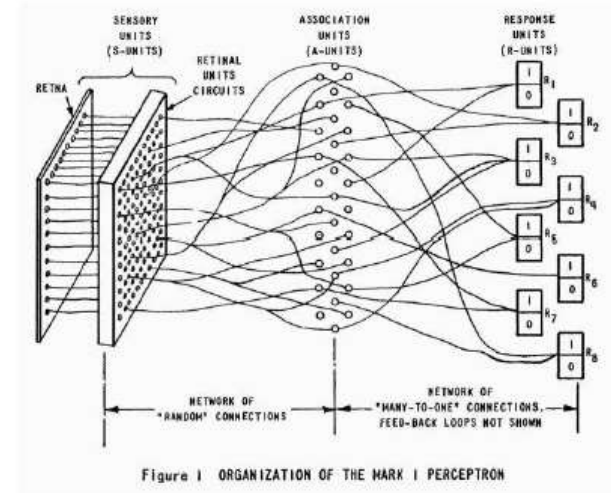
- Health and Medical Diagnostics, Medical image analysis
- Drug development
- Gaming and Video production
- Language understanding
- Robotics
- Self-driving cars
- Massive applications in military
- Monitoring and automatic control of IT systems
- Massive applications in agriculture
- Massive application opportunities in all manufacturing sectors
- Weather prediction
- Handwriting application
- Speech recognition
- Face recognition
- Object detection
- Biogenetics
- Extracting user' behaviors and user segmentation
  - Virtual behavior
  - Shopping behavior, ads, recommendations
  - Daily trip behavior



# History of Neural Networks

# History of Neural Networks

- 1943: Neural networks
- 1957-62: Rosenblatt Perceptron
- 1969: Marvin Minsky and Seymour Papert publish book “Perceptrons”
- 1970-86: Backpropagation, RNN
- 1979-98: CNN, MNIST, LSTM, Bidirectional RNN
- 2006: Deep Learning
- 2009: ImageNet + AlexNet
- 2014: GANs
- 2016-17: AlphaGo
- 2017-19: Transformers



Frank Rosenblatt, Perceptron (1957-1962)  
Early description and engineering of single and multi-layer perceptron

# Turing Award for Deep Learning

- ACM named Yoshua Bengio, Geoffrey Hinton, and Yann LeCun recipients of the 2018 ACM A.M. **Turing Award** for conceptual and engineering breakthroughs that have made **deep neural networks** a critical component of computing.



# What can you do with an NN and what not?

- In principle, NNs can compute any computable function, i.e., they can do everything a normal digital computer can do. Almost any mapping between vector spaces can be approximated to an arbitrary precision by feedforward NNs
- In practice, NNs are especially useful for **classification** and **interpolation** problems, usually when rules such as those that might be used in an expert system cannot easily be applied.
- There are no methods for training NNs that can magically create information that is not contained in the training data.
- Some AI scientists claim that their NNs are inspired by biological systems.
- Other AI scientists claim that theirs are not.

# Mechanisms in Living Brain

## Proof of Similar Nature of all Neurons

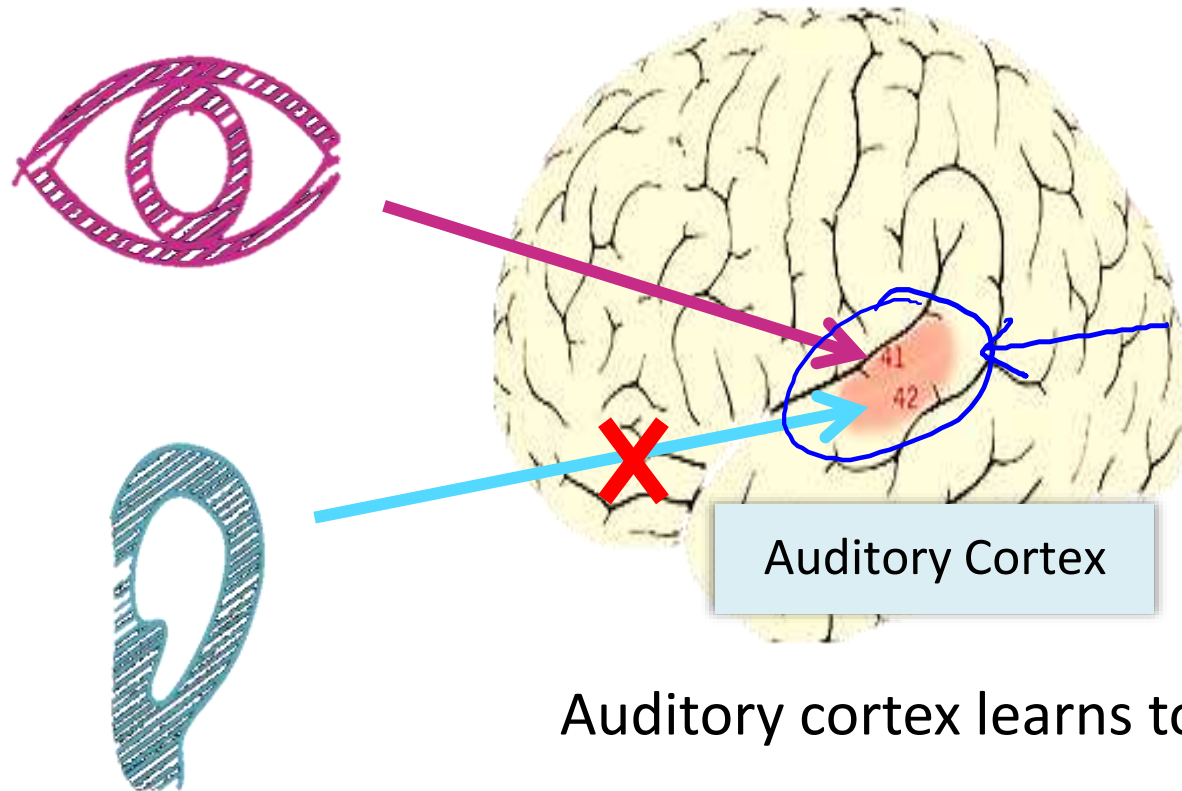
# All Brain Neurons have a Single Learning Algorithm

- Neuroscientists came up with a hypothesis that all neurons in the brain have a single learning algorithm, or that all neurons are practically the same
- Some scientists believe that this hypothesis justifies creation of the Artificial Neural Networks which are made of simple, identical, neurons. Rosenblatt's neurons were transistors. Our neurons are implemented in software.

## Evidence for the hypothesis:

- Auditory cortex --> takes sound signals  
If you cut the wiring from the ear to the auditory cortex and re-route optic nerve to the auditory cortex, auditory cortex learns to see
- Somatosensory context (touch processing)  
If you rewrite optic nerve to somatosensory cortex then it learns to see  
If different brain tissue could learn to see, they all learn in the same way.
- Seeing with your tongue  
Grayscale camera on head. Run wires to array of electrodes on person's tongue  
Pulses to tongue represent image signal. People could see with their tongue
- Human echolocation  
Blind people being trained in schools to interpret sound and echo
- Haptic belt direction sense  
Belt which buzzes towards north gives you a sense of direction.
- Any region in the brain can process and learn from data coming from any source.  
These ideas and experiments are illustrated on the next few slides.

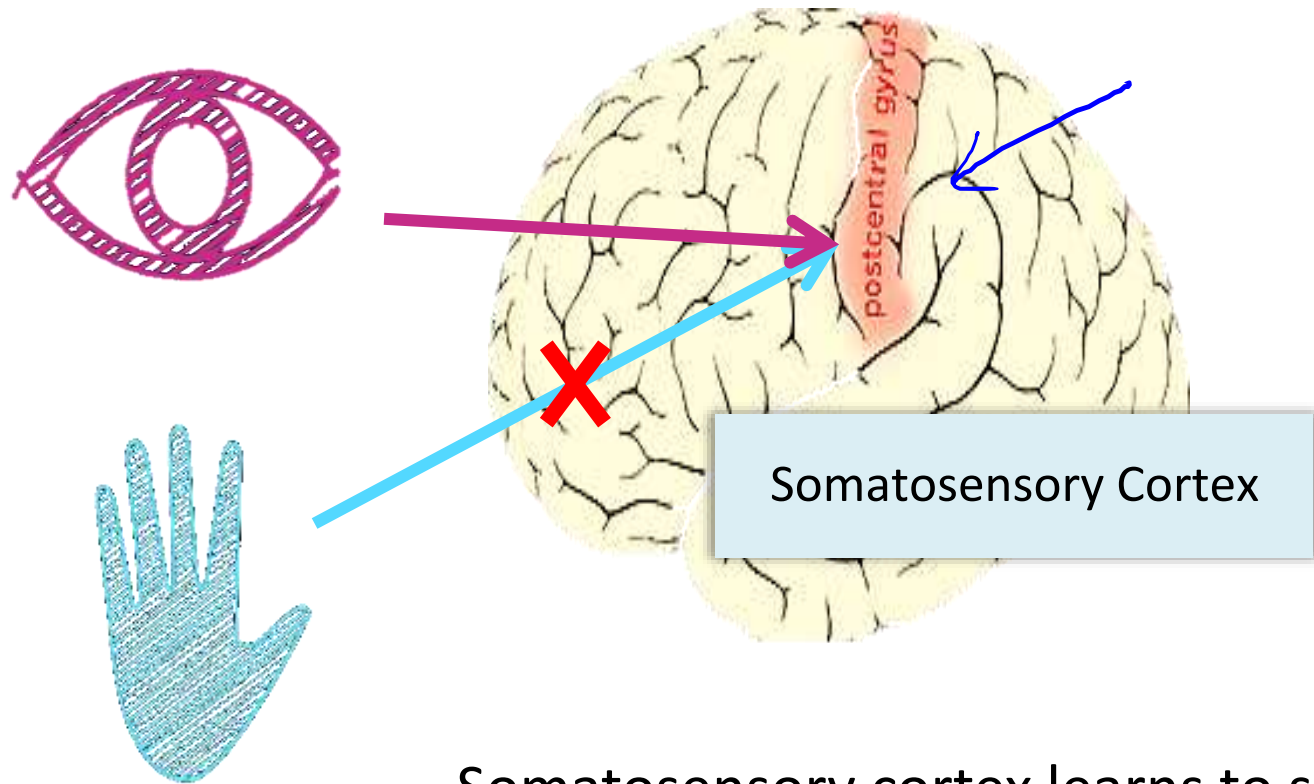
# The "one learning algorithm" hypothesis



Auditory cortex learns to see

[Professor Andrew Ng](#), originally posted on the [ml-class.org](#)

# The "one learning algorithm" hypothesis



Somatosensory cortex learns to see

[Professor Andrew Ng](#), originally posted on the [ml-class.org](#)



# Sensor representations in the brain



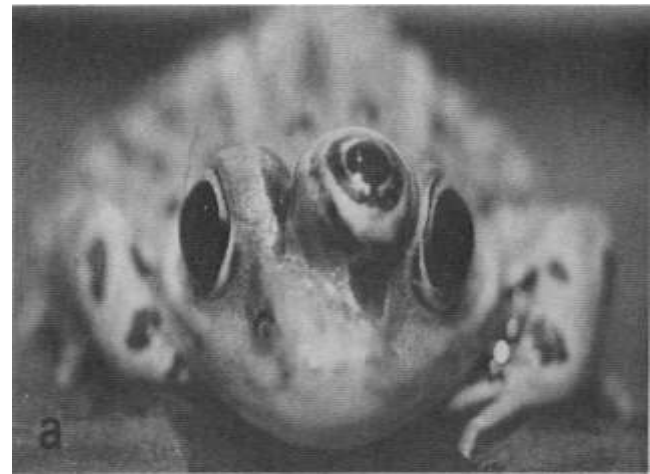
Seeing with your tongue



Human echolocation (sonar)



Haptic belt: Direction sense



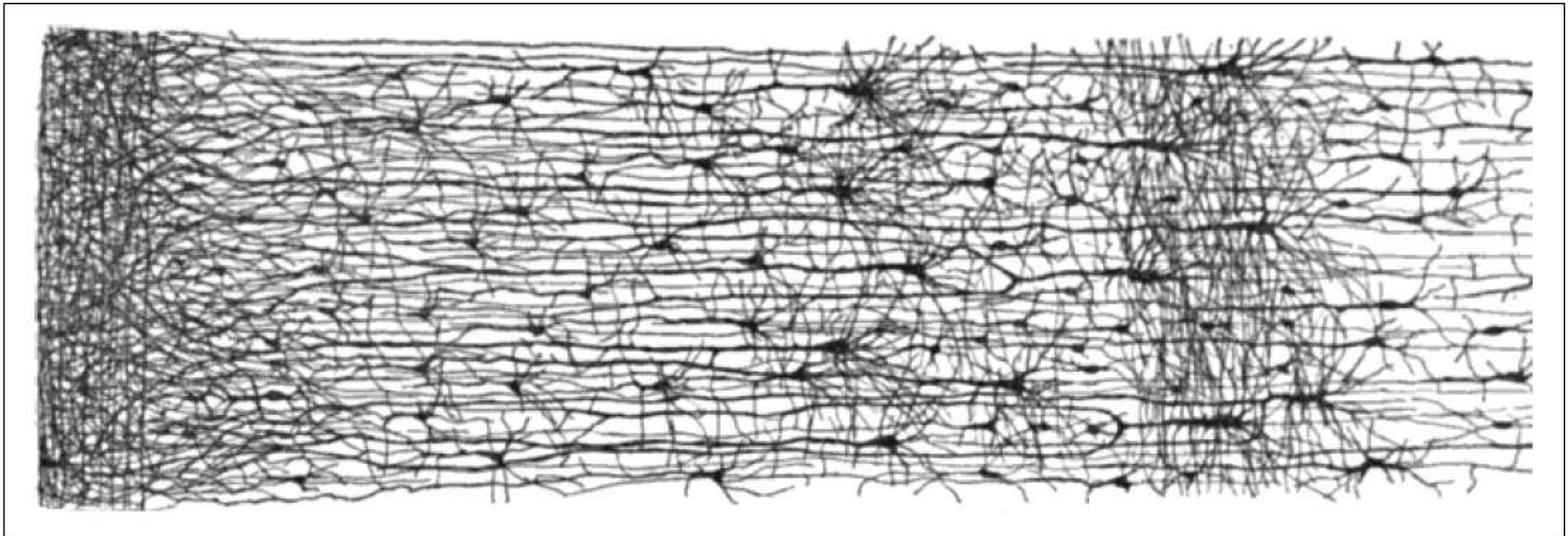
Implanting a 3<sup>rd</sup> eye

# One Learning Algorithm & Layers Hypothesis

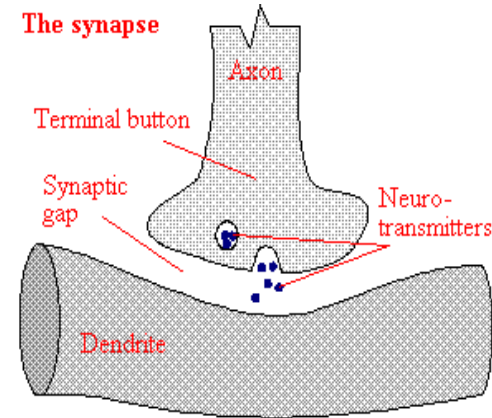
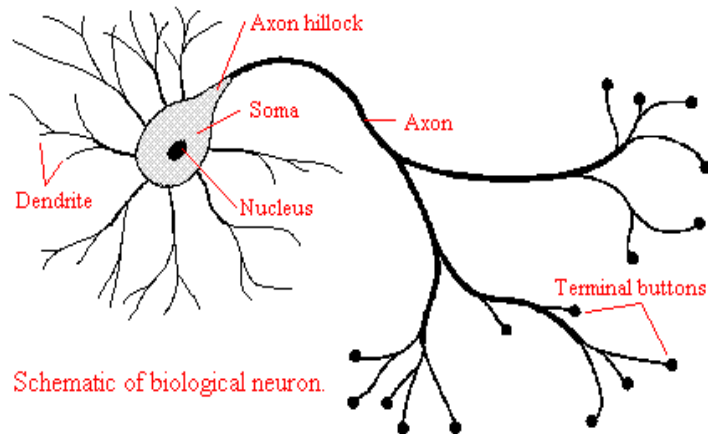
- What we have just described suggests that brain is not made of very different structures serving different purposes but is rather made of one and the same structure, the smallest unit being the neuron, which could adjust and perform different functions.

Neurons appear to be organized in layers.

- A Spanish scientist Santiago Ramón y Cajal made extensive work & discoveries on neural structures in late 1800-s. Image below is from "Texture of the Nervous System of Man and the Vertebrates" by Santiago Ramón y Cajal (1899–1904).  
<https://www.nytimes.com/2017/02/17/science/santiago-ramon-y-cajal-beautiful-brain.html>
- This image suggests that neurons do appear organized in loose layers.



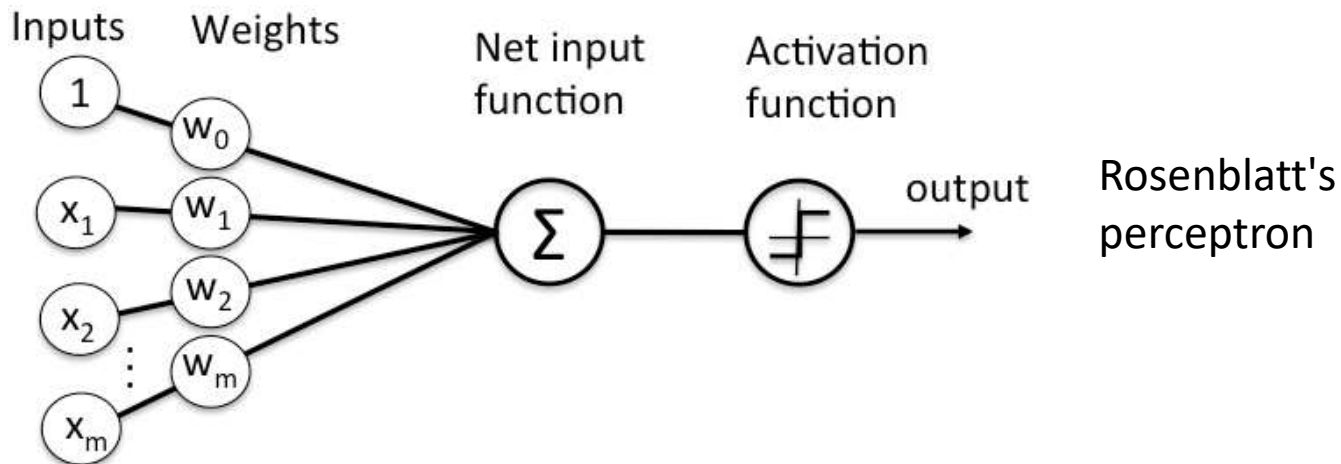
# The Biological Neuron



- Our brain is a collection of about 10 billion interconnected neurons. Each neuron is a cell that uses biochemical reactions to receive, process and transmit information.
- Each terminal button is connected to other neurons across a small gap called a synaptic gap.
- A neuron's dendritic tree is connected to hundreds of neighbouring neurons. When one of those neurons fires, a positive or negative charge is received by one of the dendrites. The strengths of all the received charges are added together through the processes of spatial and temporal summation.

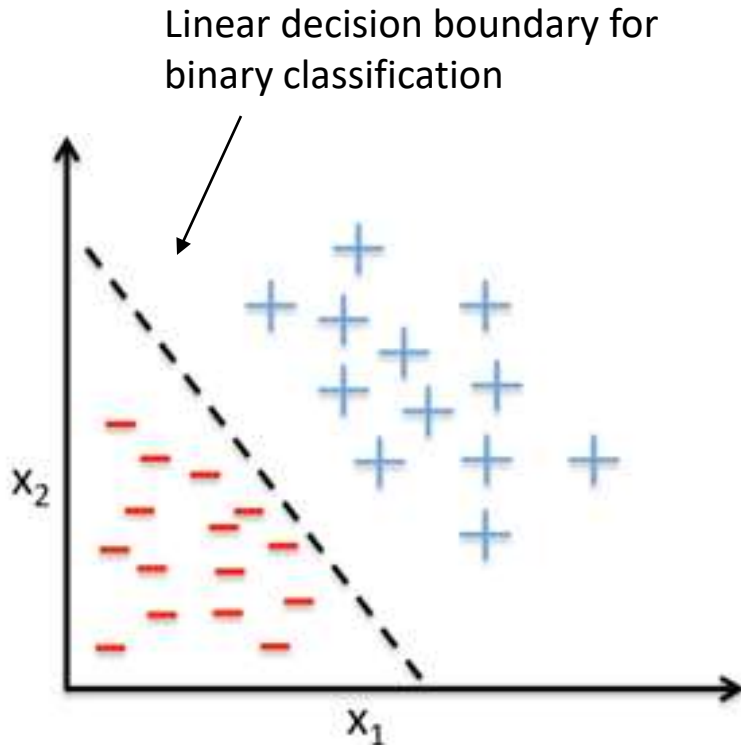
# Perceptron Learning Rule

- Neuro-scientists investigated the mechanisms used by neurons to make a decision. Eventually, it was understood that neurons accumulate inputs with different weights until the weighted sum of inputs reaches a threshold, when the neuron fires.
- One of the early mathematical models of neurons was Frank Rosenblatt's **Perceptron**.
- This perceptron mimics how a single neuron in the brain works: It either "fires" or not.
- A perceptron receives multiple input signals, applies different weights to them and if the sum of the input signals exceed a certain threshold it sends a signal. Otherwise, it remains "silent".
- This algorithm learns by selecting weights. [The perceptron algorithm is about learning the weights applied to the input signals in order to draw linear decision boundary that discriminates between the two linearly separable classes.](#)
- [Classes are labels with +1 and -1, 0 and 1, or in some other way.](#)

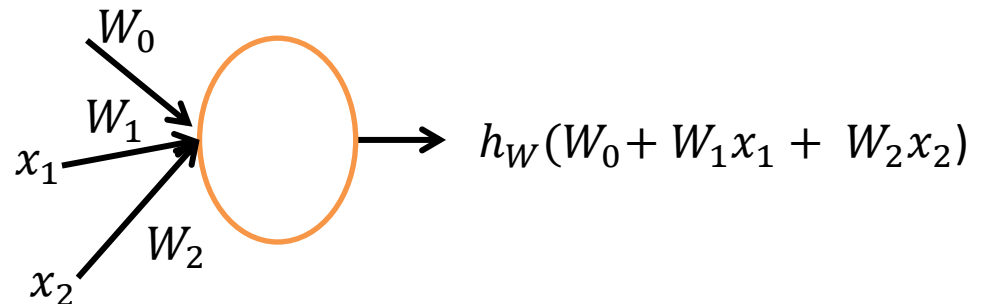


# Perceptron as Classifier

- Perceptron algorithm could be useful to determine if a sample belongs to one class or the other .



- The perceptron belongs to the category of supervised learning algorithms, so called single-layer binary linear classifiers.
- The task perceptron could solve is to predict to which of two possible categories a certain data point belongs based on a set of input variables ( $x_1$  and  $x_2$ ).
- $W_0$  is called the bias



We are looking for the best discrimination line described by 3 parameters:  $W_0$ ,  $W_1$  and  $W_2$   
Perceptron can perform linear classification.



# Unit Step Function

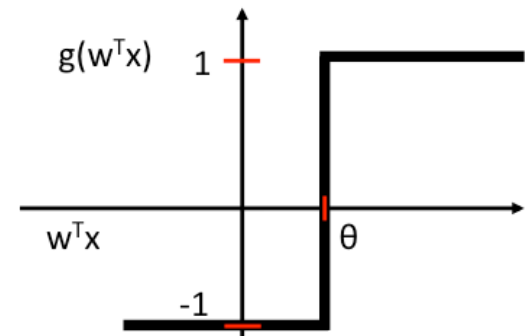
- If we are performing binary classification, we usually label the *positive* and *negative* class in our binary classification as "1" and "-1", respectively.
- We define an activation function  $g(z)$  that takes a linear combination of the input values  $x$  and weights  $w$  as input ( $z = W_0 + W_1x_1 + \dots + W_mx_m$ ).
- If  $g(z)$  is greater than a defined threshold  $\theta$ , we predict 1 and -1 otherwise.
- This activation function  $g$  is a simple "unit step function," which is sometimes also called "Heaviside step function."

$$g(z) = 1 \text{ if } z \geq \theta, \quad -1 \text{ if } z < \theta$$

- We can express  $z$  as a dot product of vectors  $w$  and  $x$ ,

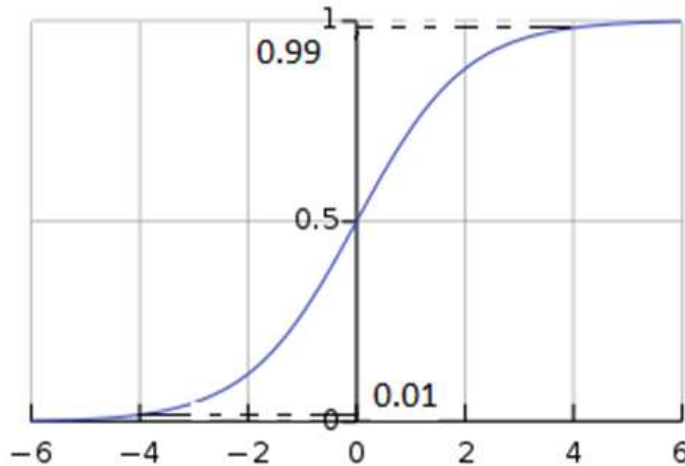
$$z = \sum_{j=1}^m x_j W_j = W^T x$$

- $W$  is the vector of weights, where  $W_0$  is the bias ( $b_0$ )
- $x$  is an  $m$ -dimensional sample from the training dataset.
- $x_0 = 1$  corresponds to the bias input.



# Sigmoid Nonlinear Activation Function

- Eventually, computer scientist learned that a differentiable Sigmoid (Logistic) function is much easier to work with, than the step function and it became the most frequently used neuron activation function.

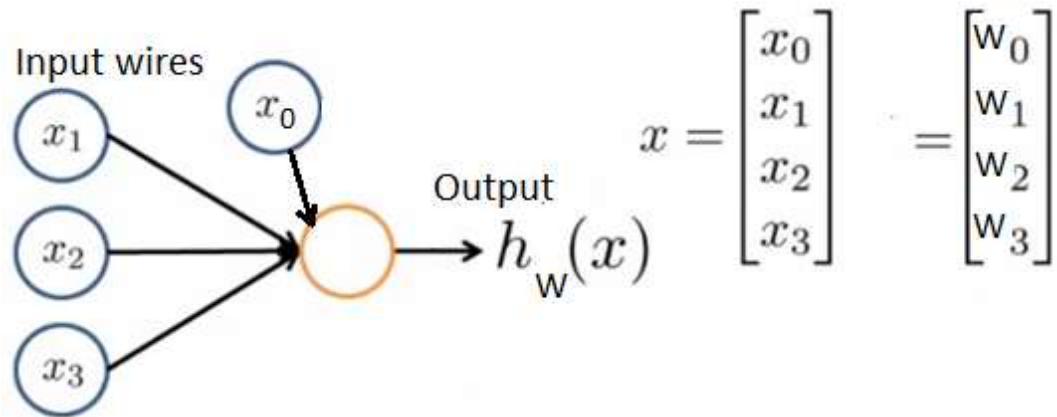


$$y_{hid}(u) = \frac{1}{1 + e^{-u}}$$

$$\frac{d}{dx} \sigma(x) = \sigma(x)(1 - \sigma(x))$$

- Where the argument  $u = z = \sum_{j=0}^m x_j W_j = W^T x$
- A single neuron with logistic activation functions can quite well identify classes in a binary problem where the boundary between two classes is close to a straight line or a multi-dimensional plane.
- For values of  $u$  above 4 logistic function is practically equal to 1. For values of  $u$  less than -4 it is practically equal to -1.

# Logistic unit represents a Neuron



In modern artificial neural network, a neuron is most frequently referred to as the logistic unit

- The input is fed via input wires plus bias input ( $x_0 = 1$ ). Bias weight  $w_0$  is adjustable.
- Logistic unit computes the logistic function:  $h_w(x) = \frac{1}{1 + e^{-w^T x}}$
- And sends output down the output wire

The name logistic comes from the “logistic regression” hypothesis calculation

- This is an artificial neuron with a sigmoid (logistic) activation function
  - Vector  $w$  represents the **weights** of the model
- The above diagram represents a single neuron.
- As we have seen, this single neuron could do some useful things.
- More complex classifications or problems might require models with several neurons.



# Need for Many Layers and Non-linear Activation Function

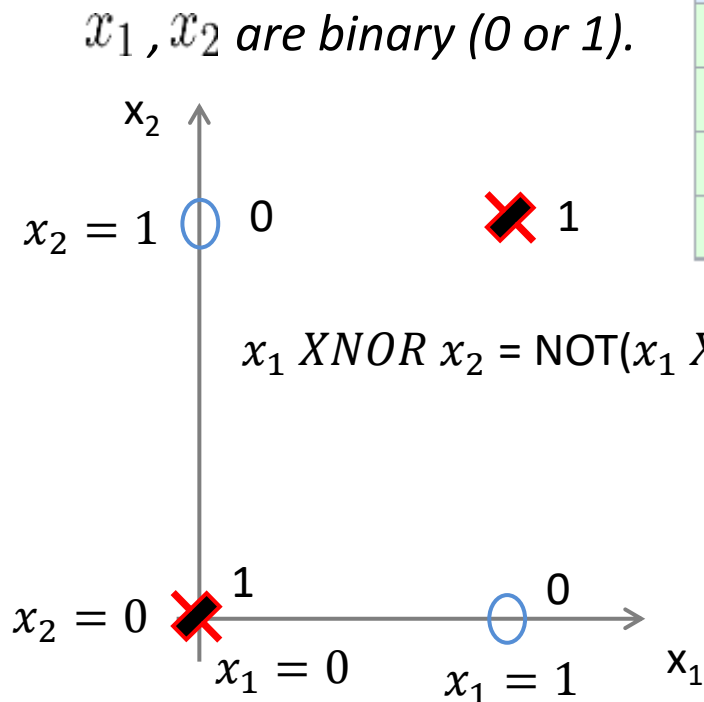
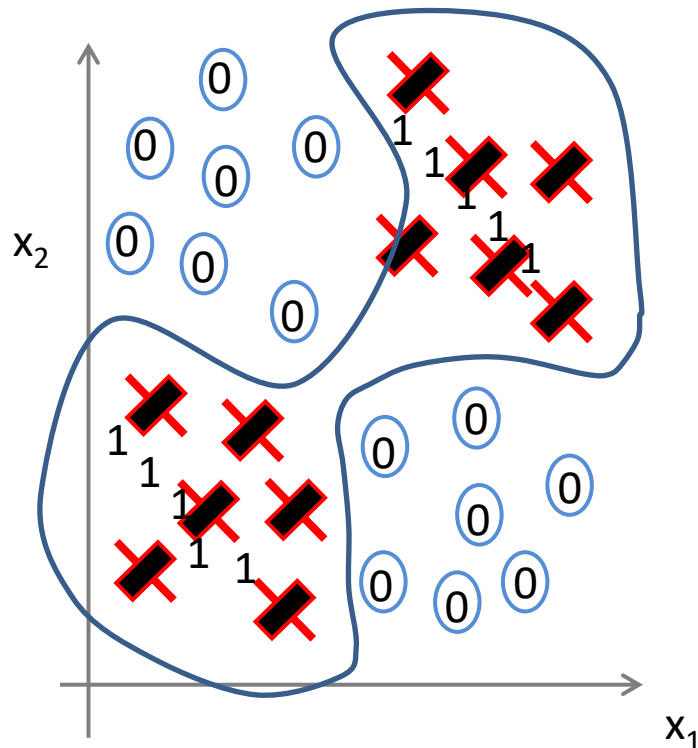
# Argument Against Perceptrons

- In a book published in 1969 by Minsky and Papert proved that single layer perceptrons are not feasible tool for general classification problem.
- On the next slide we present part of their argument and the resolution of the problem by the introduction of multi-layered.
- The argument basically states that perceptrons cannot classify many classes of binary events. The example presented on the next slide is of so called XNOR problem.

# Non-linear classification example: XOR/XNOR

Consider two classes indicated by 0-s and X-s respectively, as presented below. We would like to find the decision boundary between these two classes. A single straight line determined by a perceptron would not do it.

We could roughly approximate the problem on the left with a simple logical: XNOR function.



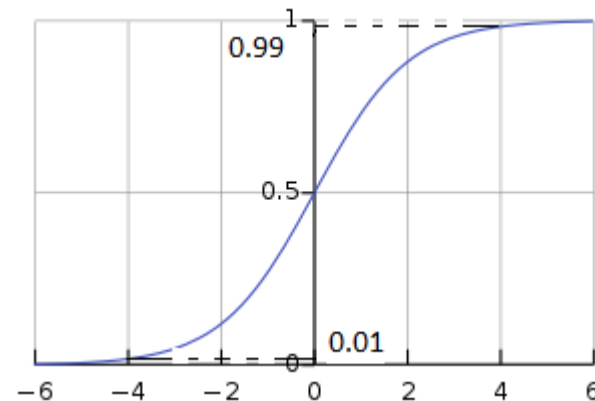
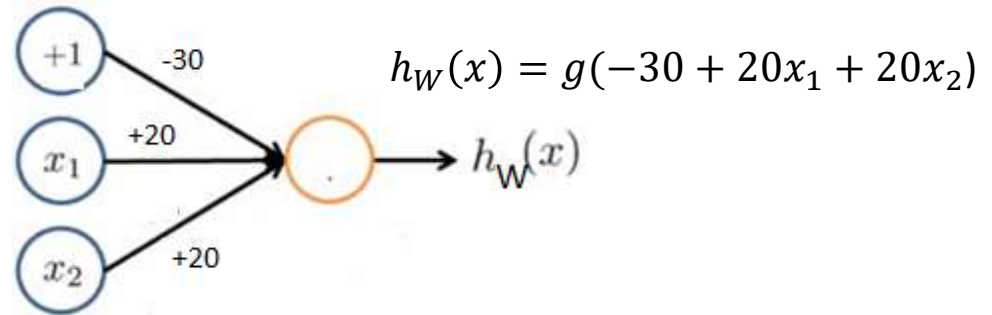
Input		Output
A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

[Andrew Ng](#)

Perceptrons can generate only linear decision boundaries. We will try to implement a logical network for the “computation” on the right.

# AND Logical Function as Neuron

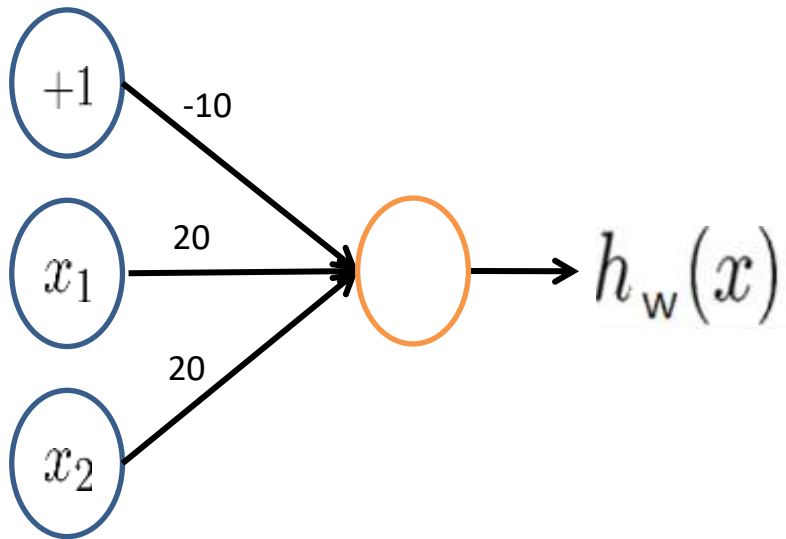
- Could we get a one-unit neural network to compute logical AND function? Add a bias unit
  - Add some weights for the networks
- What are weights?
  - Weights are the parameter values which multiply into the input nodes (i.e.  $W$ )
- Sometimes it's convenient to add the weights into the diagram
  - These values are just the  $W$  parameters so
    - $W_{10}^1 = -30$
    - $W_{11}^1 = 20$
    - $W_{12}^1 = 20$
- Look at the four input values.
- The table on the right is called “logical table”
- Sigmoid function at  $\pm 4.6$  is  $0.01 = 1\%$  away from its saturation value.



$x_1$	$x_2$	$h_W(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(+10) \approx 1$

$$h_W(x) \approx x_1 \text{ AND } x_2$$

# OR function as a Neuron

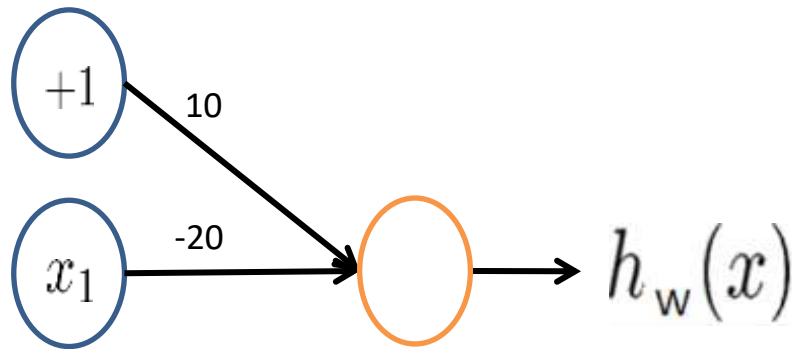


$x_1$	$x_2$	$h_w(x)$
0	0	$g(-10) \sim 0$
0	1	$g(10) \sim 1$
1	0	$\sim 1$
1	1	$\sim 1$

$$g(-10 + 20x_1 + 20x_2)$$

# NOT function as a neuron

- Negation is achieved by putting a large negative weight in front of the variable you want to negate



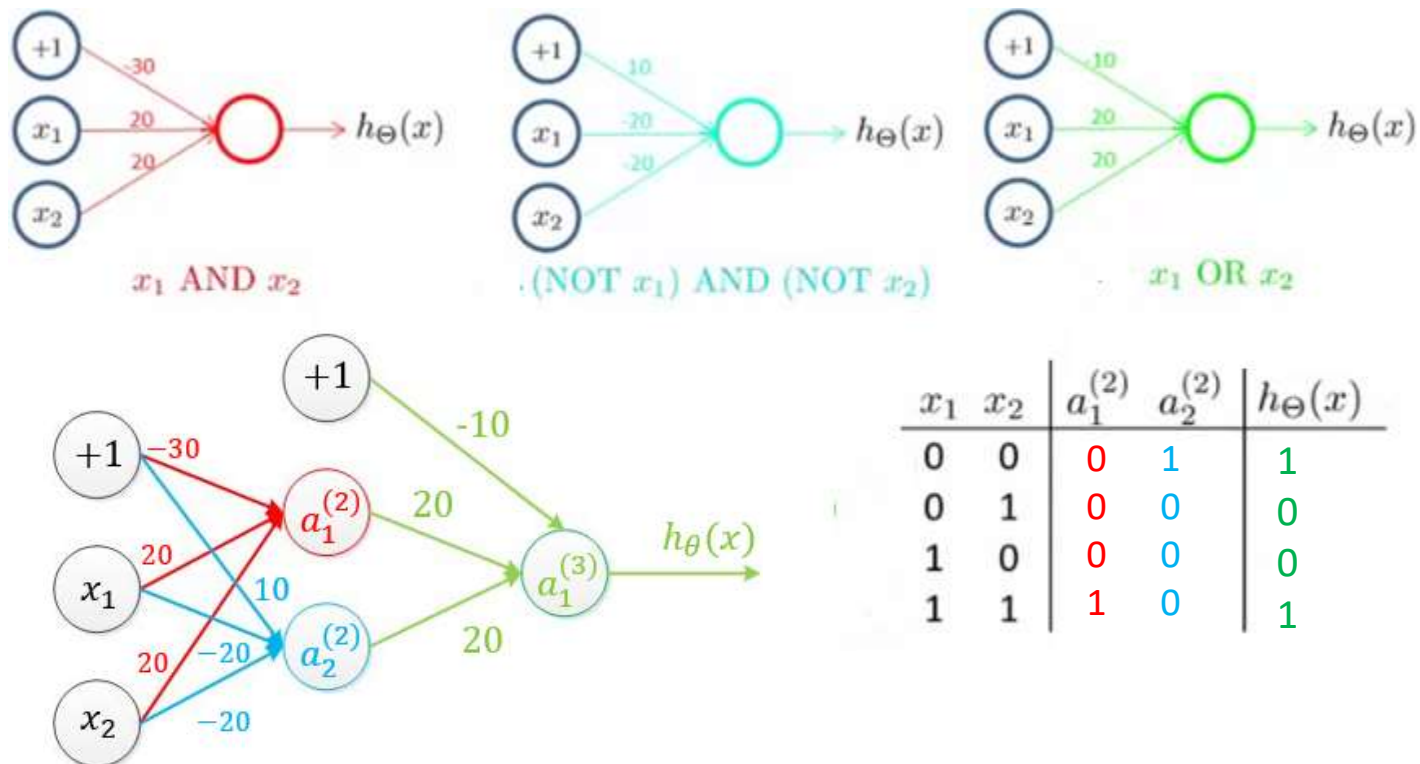
$$g(10 - 20x_1)$$

$x_1$	$h_w(x)$
0	$g(10) \sim 1$
1	$g(-10) \sim 0$

- If you recall from your Computer Science courses, by using AND, OR and NOT functions, one could build any logical function. At the micro level, all our computers are collections of AND, OR and NOT gates.
- This is where our claim that Neural Networks could calculate anything is coming from.

# XNOR as a combination of AND, OR and NOT neurons

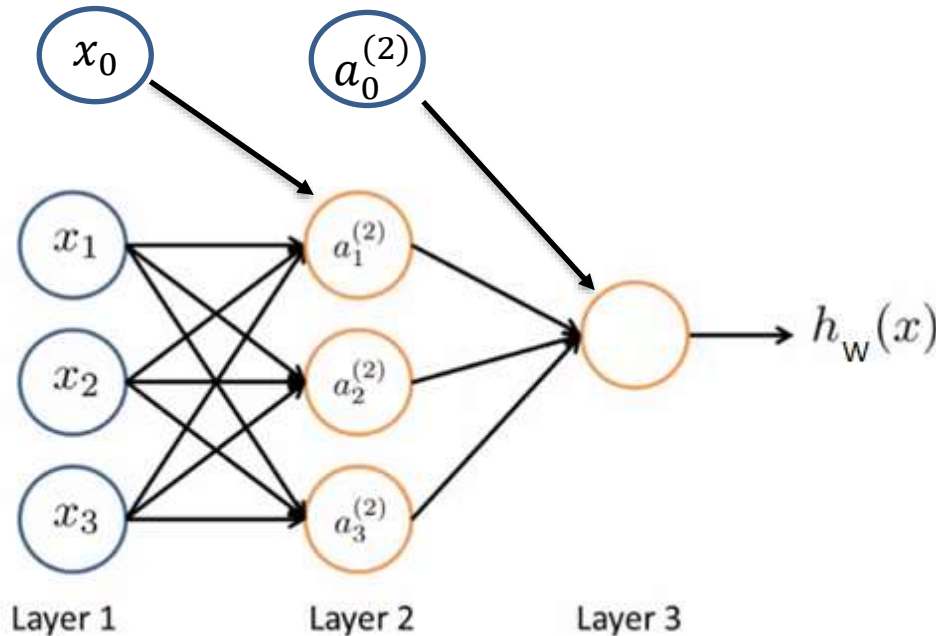
- XNOR is short for NOT XOR
  - i.e. NOT an exclusive OR, so either (1,1) or (0,0) input will give 1 as output
- We can construct a logical structure which produces a positive output (1) when both inputs have the same value and a negative output when they are different.
- We combine NN representations of Logical OR and AND gates into a neural network as shown below;



$x_1$	$x_2$	$a_1^{(2)}$	$a_2^{(2)}$	$h_{\Theta}(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

# Multi-layer Neural Networks

- The key take-away from previous analysis is that we could simulate complex functions by organizing groups of neurons in several layers and not only one layer.



Here, inputs are  $x_1$ ,  $x_2$  and  $x_3$ .

- Usually, we call the inputs “the first layer”, Layer 1.  $x_0=1$
- There are three neurons in layer 2 ( $a_1^{(2)}$ ,  $a_2^{(2)}$  and  $a_3^{(2)}$ )
- The final fourth neuron in Layer 3 produces the output of the network.
- First layer is called the **input layer**
- Final layer is called the **output layer**. It produces value computed by the hypothesis
- Middle layer(s) are called the **hidden layers**
  - Typically, you do not observe the values processed in the hidden layers
  - We can have many hidden layers

$a_i^{(j)}$  - **activation of unit  $i$  in layer  $j$**

So,  $a_i^{(2)}$  - is the **activation** of the 1<sup>st</sup> unit in the second layer

By activation, we mean the value which is computed and set at the output by that node

$a_0^{(j)}$  is the bias term of layer  $j$ .

[Andrew Ng](#)



# Why do we need non-linear Activation Functions

- If we would build a multilayer neural network and use only linear activation functions, any multi-layer network would “collapse” to a single layer.
- Multi-layer networks with all linear activation functions are equivalent to matrix multiplications. Products of many matrixes is still a single matrix.

# Typical multi-output network: Classification.



Pedestrian



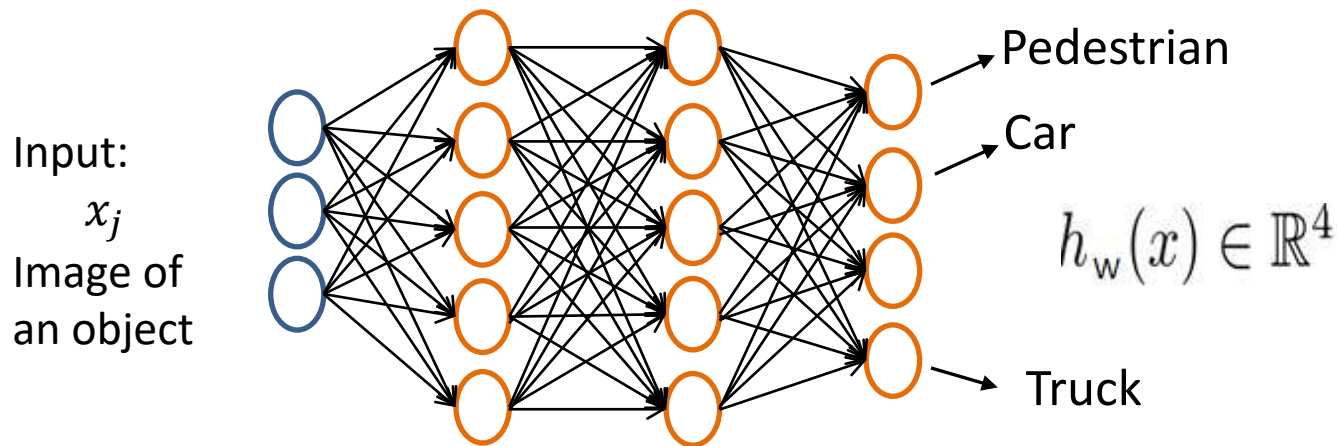
Car



Motorcycle



Truck



Want  $h_w(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ ,  $h_w(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ ,  $h_w(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ , etc.

when pedestrian

when car

when motorcycle

How do we determine network parameters

# Cost or Loss Function, Optimization Problem

- Initially, we do not know proper values of parameters ( $w_{ij}^{(k)}$  and  $b_j^{(k)}$ ) of any neuron in any layer. The training set of experiments  $\{X^i\}$  contains proper (correct) labels  $\{Y^i\}$  for all its elements. Superscript  $i$  is sample/experiment index.
- Forward calculation for any one experiment (sample)  $X^i$  produces an output vector  $Y_{predicted}^i$ . The value of that vector does not have to match the known label  $Y^i$ .
- The sum over all  $N$  samples of the difference between known labels  $Y^i$  and the forward calculated values  $Y_{predicted}^i$  is a measure of the quality of the current network.
- Usually, we express that quality as the **cost** or **loss function** of the form:

$$L_2(\{W\}, \{b\}) = \sum_{i=1}^N (Y^i - Y_{forward\ pass\ calculated}^i)^2$$

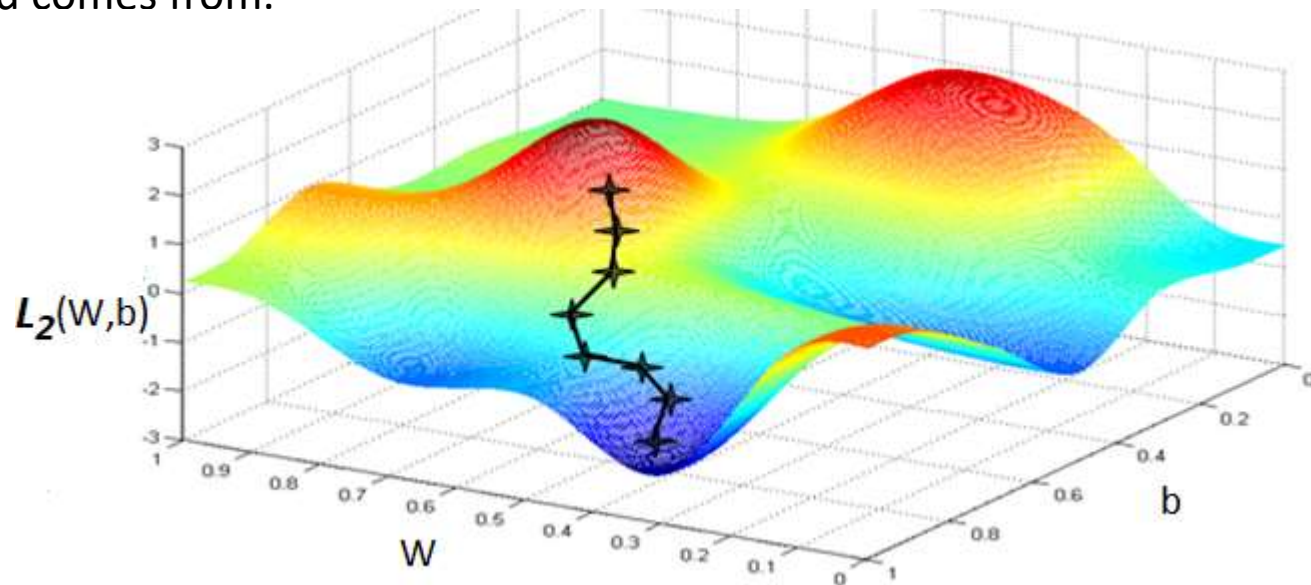
- which is the sum of squares of the errors the model makes at forward pass for all the samples. Notice that the loss function depends on values of all weights and biases of all neurons in all hidden layers.
- Finding the best collection of weights and biases which minimizes the loss function is an **optimization problem**.
- The input vectors could have very large dimensionalities, 10s, 100s, 1000s, even more. We could have very large number of weights just in the first layer. Since we could have many hidden layers, the number of weights that we must find could be truly large (hundreds of thousands, millions, ...)
- We are dealing with an optimization in a space of a (very) large dimensionality.

# Optimization Method, Gradient Descent

- Input variables of the loss or cost function  $L_2$  are the model weight(s)  $W$  and the biases  $b$ , not the actual dataset features inputs  $\{X^i\}$ . Values of  $X^i$ -s are fixed by the dataset and cannot be optimized.
- The slope of  $L_2$  at any point in  $(W, b)$  space is a vector, called the gradient.

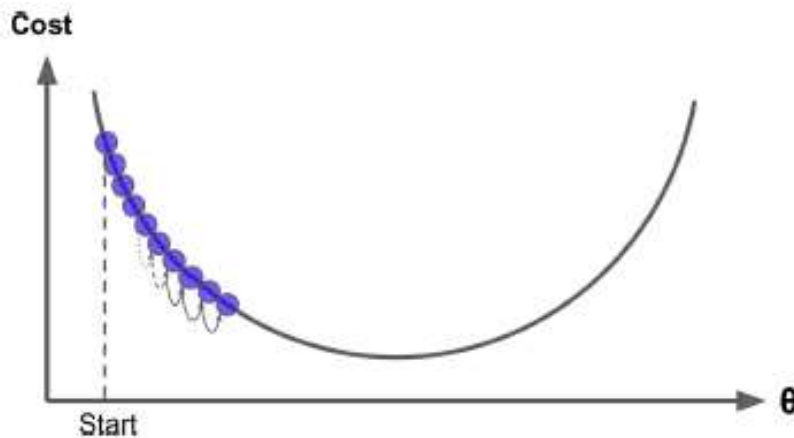
$$\nabla L_2 = \left( \frac{\partial L_2}{\partial W}, \frac{\partial L_2}{\partial b} \right)$$

- The gradient indicates the direction of maximum growth for the loss function. To find the minimum of  $L_2$  we want to move in the direction opposite to the gradient.
- That is where the name "gradient descent" of an iterative numerical optimization method comes from.

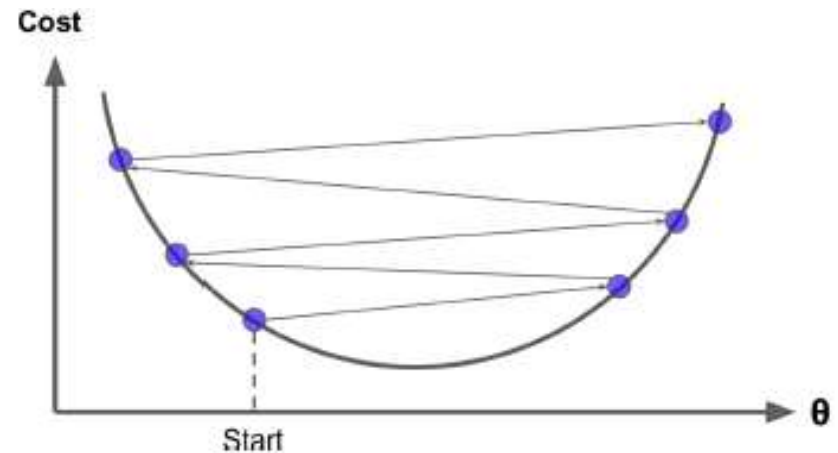


# Importance of Learning Rate

- Choosing the step size, the learning rate, is one of the most important hyper-parameter settings in training of a neural network. In our blindfolded hill-descent analogy, we feel the hill below our feet sloping in some direction, but the step length we should take is uncertain.
- If we choose our steps as very small, we can expect to make consistent but very slow progress.
- Conversely, we can choose to make a large, confident step in an attempt to descend faster, but this may not pay off. At some point taking a bigger step gives a higher loss as we "overstep".



Learning rate too small



Learning rate too large

# Compute Gradients Analytically

- One way to compute the gradients is analytically using Calculus, by deriving a direct formula for the gradient without any approximation.
- Analytic gradients are very fast to compute. For small networks analytic gradients are tractable.
- However, unlike the numerical gradient, implementations of analytical gradients could be more error prone. In practice, it is very common to compute the analytic gradient and compare it to the numerical gradient to verify the correctness of your implementation.
- If you are interested in this technique, you should look at Autograd API:
- <https://github.com/HIPS/autograd>
- Usually, we do not perform optimizations ourselves. We use software tools, APIs, such as TensorFlow, Keras, Pytorch or others.



# TensorFlow & PyTorch

# Competition and Convergence



TensorFlow™ 2.13

- Eager execution by default (imperative programming)
- Keras integration & Promotion
- Cleanup of API
- TensorFlow.js
- TensorFlow Lite
- TensorFlow Serving
- Tensorflow-Hub



PyTorch™ 1.12

- TorchScript (graph representation)
- Quantization
- PyTorch Mobile
- TPU Support

# TensorFlow

- TensorFlow™ is an open-source software library for numerical computation using data flow graphs.
- Nodes in a graph represent mathematical operations.
- Graph edges represent the multidimensional data arrays (tensors) communicated between nodes.
- The architecture allows TF to deploy computations to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API.
- TensorFlow calculations could scale and run on clusters of many machines  
TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research.
- The system is general enough to be applicable in a wide variety of other domains as well.
- TF has APIs in Python, Java, JavaScript, C++, Swift. They also mention Go and C.

# TensorFlow Eco System

- TensorFlow includes several platforms and libraries



## TensorFlow.js

A WebGL accelerated, browser based JavaScript library for training and deploying ML models.



## TensorFlow Lite

TensorFlow's lightweight solution for mobile and embedded devices.



## Swift for TensorFlow

Swift for TensorFlow is based on the belief that machine learning is important enough to deserve first-class language and compiler support.



## TensorFlow Extended (TFX)

TensorFlow Extended (TFX) is an end-to-end ML platform, spanning from data ingestion, training, to production-grade serving.



## TensorFlow Hub

TensorFlow Hub is a library to foster the publication, discovery, and consumption of reusable parts of machine learning models.



## TensorFlow Probability

TensorFlow Probability is a library for probabilistic reasoning and statistical analysis.



## XLA

XLA (Accelerated Linear Algebra) is a domain-specific compiler for linear algebra that optimizes TensorFlow computations.

# TensorFlow Eco System

- TensorFlow includes several platforms and libraries



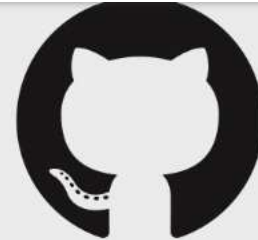
## TensorFlow Research Cloud

Accelerating open machine learning research with Cloud TPUs.



## Tensor2Tensor

Library of deep learning models and datasets designed to make deep learning more accessible and accelerate ML research.



## Models

Official models and examples built with TensorFlow.



## Magenta

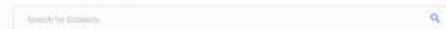
Magenta is a research project exploring the role of machine learning in the process of creating art and music.



## Google Colaboratory

A hosted Jupyter notebook environment that is free to use and requires no setup. Used to demonstrate the [TensorFlow tutorials](#).

## Google Dataset Search



## Google Dataset Search

Dataset Search enables users to find datasets stored across the Web through a simple keyword search. [Learn more](#)

# What TensorFlow Offers

- TF supports distributed computing (across multiple devices and servers).
- TF includes a just-in-time (JIT) compiler that allows it to optimize computations for speed and memory usage.
- JIT compiler works by:
  - extracting the *computation graph* from a Python function,
  - optimizing it (e.g., by pruning unused nodes) and finally
  - running it efficiently (e.g., by automatically running independent operations in parallel).
- Computation graphs can be exported to a portable format, so you can train a TensorFlow model in one environment (e.g., using Python on Linux), and run it in another (e.g., using Java on an Android device).
- TF implements *autodiff* , and provides many excellent optimizers, such as RMSProp, Nadam and FTRL, so you can easily minimize all sorts of loss functions.
- TensorFlow offers many more features, built on top of these core features:
  - a high level easy to use API `tf.keras`,
  - data loading & preprocessing ops (`tf.data`, `tf.io`, etc.),
  - image processing ops (`tf.image`),
  - signal processing ops (`tf.signal`), and more

# TensorFlow Installation

# On Windows, Install TF== 2.10.0

- Caution: TensorFlow 2.10 is the last TensorFlow release that supported GPU on native-Windows. Starting with TensorFlow 2.11, you need to install TensorFlow in WSL2, or install `tensorflow` or `tensorflow-cpu` and, optionally, try the TensorFlow-DirectML-Plugin.
- Unless you know how to connect your GPU card to your WSL2 on Windows, please install TF 2.10.0 without WLS2.
- The first step is to create a virtual environment with Python 3.10.0

```
conda create -name tf-gpu python==3.10.0
```

- If you need to work with a specific version of CUDA and CUDNN packages install them by typing a command similar to the one below, with your versions of packages:

```
conda install -c conda-forge cudatoolkit=11.2 cudnn=8.1.0
```

- Anything above 2.10 is not supported on the GPU on Windows Native

```
conda install tensorflow==2.10.0
```

- To use pip, from <https://pypi.org/project/tensorflow-gpu/2.10.0/#files> download file: `tensorflow_gpu-2.10.0-cp310-cp310-win_amd64.whl`

- Then use `pip` to install this whl file:

```
pip install tensorflow_gpu-2.10.0-cp310-cp310-win_amd64.whl
```

- Verify installation:

```
>>> import tensorflow as tf
```

```
>>> tf.__version__  
'2.10.0'
```

```
>>> print(tf.config.list_physical_devices('GPU'))
```

```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```



# tensorflow\_docs & tensorflow\_datasets

- From the page: <https://pypi.org/project/tensorflow-docs/#files> download tensorflow\_docs wheel and install:

```
pip install tensorflow_docs-2024.5.24.56664-py3-none-any.whl
```

- From the page <https://pypi.org/project/tensorflow-datasets/#files> download and install wheel: tensorflow\_datasets-4.9.3-py3-none-any.whl
- I have an issue with version of protobuf. Tensorboard 2.10.0 requires protobuf-3.19.6. Tensorflow\_datasets require protobuf-3.20.3

# Installation of Windows System Linux

```
(base) PS C:\Users\Zoran> wsl --list --online
The following is a list of valid distributions that can be installed.
Install using 'wsl.exe --install <Distro>'.
```

NAME	FRIENDLY NAME
Ubuntu	Ubuntu
Debian	Debian GNU/Linux
kali-linux	Kali Linux Rolling
Ubuntu-18.04	Ubuntu 18.04 LTS
Ubuntu-20.04	Ubuntu 20.04 LTS
Ubuntu-22.04	Ubuntu 22.04 LTS
OracleLinux_7_9	Oracle Linux 7.9
OracleLinux_8_7	Oracle Linux 8.7
OracleLinux_9_1	Oracle Linux 9.1
openSUSE-Leap-15.5	openSUSE Leap 15.5
SUSE-Linux-Enterprise-Server-15-SP4	SUSE Linux Enterprise Server 15 SP4
SUSE-Linux-Enterprise-15-SP5	SUSE Linux Enterprise 15 SP5
openSUSE-Tumbleweed	openSUSE Tumbleweed

```
(base) PS C:\Users\Zoran> wsl --install --distribution Ubuntu-22.04
Installing: Ubuntu 22.04 LTS
Ubuntu 22.04 LTS has been installed.
Installing, this may take a few minutes...
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: ubuntu
New password:
Retype new password:
passwd: password updated successfully
Installation successful!
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
```

```
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.15.90.1-microsoft-standard-WSL2 x86_64)
```

```
* Documentation: https://help.ubuntu.com
* Management:   https://landscape.canonical.com
* Support:      https://ubuntu.com/advantage
```

```
This message is shown once a day. To disable it please create the
/home/ubuntu/.hushlogin file.
ubuntu@DESKTOP-U3NH4I2:~$
```

# Install Miniconda

- Download the latest shell script

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

- Make the miniconda installation script executable

```
chmod +x Miniconda3-latest-Linux-x86_64.sh
```

- Run miniconda installation script

```
./Miniconda3-latest-Linux-x86_64.sh
```

- Create and activate a conda environment

To create a conda environment, run `conda create -n newenv`

- You can also create the environment from a file like `environment.yml`, you can use the `conda env create -f` command: `conda env create -f environment.yml`. The environment name will be the directory name.

# TF installation on Windows System Linux (WSL2)

- Note: Starting with TensorFlow 2.10, Linux CPU-builds for Aarch64/ARM64 processors are built, maintained, tested and released by a third party: AWS.
- Installing the tensorflow package on an ARM machine installs AWS's `tensorflow-cpu-aws` package. They are provided as-is. Tensorflow will use reasonable efforts to maintain the availability and integrity of this pip package. There may be delays if the third party fails to release the pip package.

```
conda install -c conda-forge cudatoolkit=11.8.0
python -m pip install nvidia-cudnn-cu11==8.6.0.163 tensorflow==2.13.*
mkdir -p $CONDA_PREFIX/etc/conda/activate.d
echo 'CUDNN_PATH=$(dirname $(python -c "import
        nvidia.cudnn;print(nvidia.cudnn.__file__)"))' >>
        $CONDA_PREFIX/etc/conda/activate.d/env_vars.sh
echo
'export LD_LIBRARY_PATH=$CUDNN_PATH/lib:$CONDA_PREFIX/lib/:$LD_LIBRARY_PATH'
>> $CONDA_PREFIX/etc/conda/activate.d/env_vars.sh
source $CONDA_PREFIX/etc/conda/activate.d/env_vars.sh
# Verify install:
python3 -c "import tensorflow as tf;
print(tf.config.list_physical_devices('GPU'))"
```

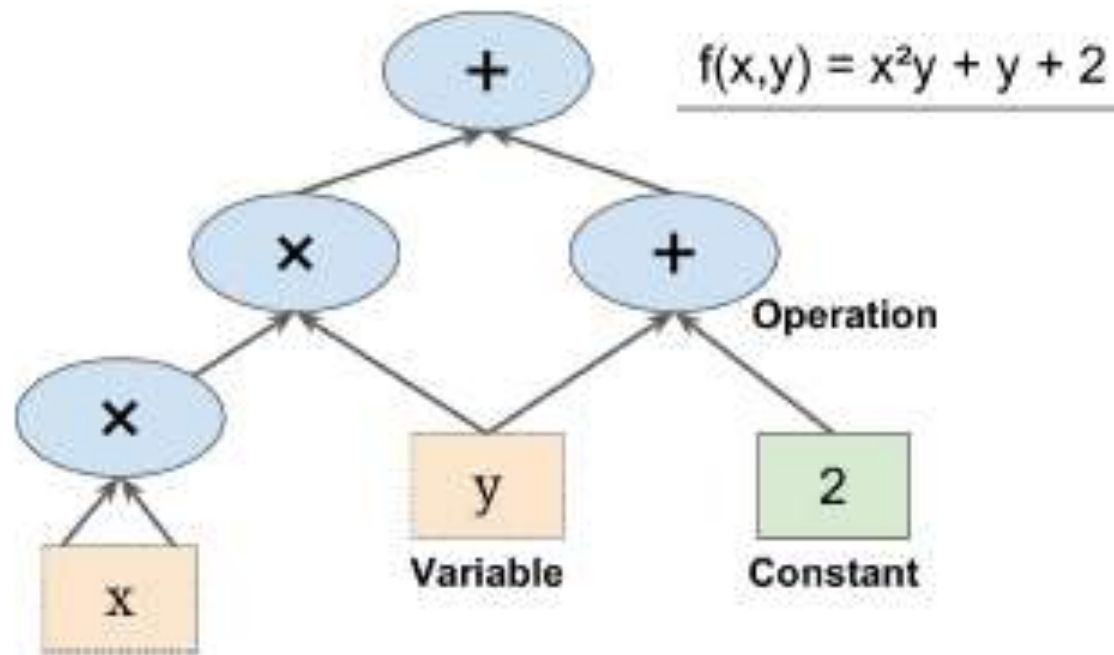
# Hardware requirements

- Note: TensorFlow binaries use AVX (Advanced Vector Extensions) instructions which may not run on older CPUs.
- The following GPU-enabled devices are supported:
  - NVIDIA® GPU card with CUDA® architectures 3.5, 5.0, 6.0, 7.0, 7.5, 8.0 and higher. See the list of CUDA®-enabled GPU cards.
  - For GPUs with unsupported CUDA® architectures, or to avoid JIT compilation from PTX, or to use different versions of the NVIDIA® libraries, see the Linux build from source guide.
  - Packages do not contain PTX code except for the latest supported CUDA® architecture; therefore, TensorFlow fails to load on older GPUs when CUDA\_FORCE\_PTX\_JIT=1 is set. (See Application Compatibility for details.)
- Note: The error message "Status: device kernel image is invalid" indicates that the TensorFlow package does not contain PTX for your architecture. You can enable compute capabilities by building TensorFlow from source.
- Parallel Thread Execution (PTX or NVPTX) is a low-level parallel thread execution virtual machine and instruction set architecture used in Nvidia's Compute Unified Device Architecture (CUDA) programming environment.
- System requirements
  - Ubuntu 16.04 or higher (64-bit)
  - macOS 10.12.6 (Sierra) or higher (64-bit) (no GPU support)
  - Windows Native - Windows 7 or higher (64-bit) (no GPU support after TF 2.10)
  - Windows WSL2 - Windows 10 19044 or higher (64-bit)
  - Note: GPU support is available for Ubuntu and Windows with CUDA®-enabled cards.

# TensorFlow Tensors and Basic Operations

# Computational Graphs

- The basic principle of TensorFlow is simple:
- We first define in a graph of computations you want to perform, then TensorFlow takes that graph and runs it efficiently using optimized C++ code.
- Nodes of the graph are both data (variables, constants) and operations.
- All variables and constants are Tensors.



# Tensors and Constants

- TensorFlow's API revolves around *tensors*, hence the name Tensor-Flow. A tensor is usually a multidimensional array (exactly like a NumPy `ndarray`). Tensor can also hold a scalar (a simple value, 42).
- You can create a tensor, using `tf.constant()`. For example, here is a tensor representing a matrix with two rows and three columns of floats:

```
>>> tf.constant([[1., 2., 3.], [4., 5., 6.]]) # matrix
<tf.Tensor: id=0, shape=(2, 3), dtype=float32, numpy=
array([[1., 2., 3.],
       [4., 5., 6.]], dtype=float32)>
```

- The following is a tensor representing a scalar

```
>>> tf.constant(42) # scalar
<tf.Tensor: id=1, shape=(), dtype=int32, numpy=42>
```

- Just like a `numpy.ndarray`, a `tf.Tensor` has a shape and a data type (`dtype`):

```
>>> t = tf.constant([[1., 2., 3.], [4., 5., 6.]])
>>> t.shape
TensorShape([2, 3])
>>> print(t.shape[0], t.shape[1])
2 3
>>> t.dtype
tf.float32
```



# Variables

- We have seen constant tensors: as their name suggests, you cannot modify them. However, the weights in a neural network need to be tweaked by backpropagation, and other parameters may also need to change over time (e.g., a momentum optimizer keeps track of past gradients).
- What we need is a `tf.Variable`:

```
>>> v = tf.Variable([[1., 2., 3.], [4., 5., 6.]])
>>> v
<tf.Variable 'Variable:0' shape=(2, 3) dtype=float32, numpy=
array([[1., 2., 3.],
       [4., 5., 6.]], dtype=float32)>
```

- A `tf.Variable` acts much like a constant tensor: you can perform the same operations with it. Variables convert back and forth to NumPy as well. Variables use all the listed types.
- In difference to constants, variables can be modified in place using the `assign()` method (or `assign_add()` or `assign_sub()` which increment or decrement the variable by the given value).
- You can also modify individual cells (or slices), using the cell's (or slice's) `assign()` method (direct item assignment will not work), or using the `scatter_update()` or `scatter_nd_update()` methods:

```
>>> v.assign(2 * v)
[[2., 4., 6.], [8., 10., 12.]]
>>> v[0, 1].assign(42)
[[2., 42., 6.], [8., 10., 12.]]
>>> v[:, 2].assign([0., 1.]) # => [[2., 42., 0.], [8., 10., 1.]]
>>> v.scatter_nd_update(indices=[[0, 0], [1, 2]], updates=[100., 200.])
[[100., 42., 0.], [8., 10., 200.]]
```

# Variables

```
>>> v = tf.Variable([[1., 2., 3.], [4., 5., 6.]])
>>> v.assign(2 * v)
<tf.Variable 'UnreadVariable' shape=(2, 3) dtype=float32, numpy=
array([[ 2.,  4.,  6.],
       [ 8., 10., 12.]], dtype=float32)>

>>> v[0, 1].assign(42)
<tf.Variable 'UnreadVariable' shape=(2, 3) dtype=float32, numpy=
array([[ 2., 42.,  6.],
       [ 8., 10., 12.]], dtype=float32)>

>>> v[:, 2].assign([0., 1.])
<tf.Variable 'UnreadVariable' shape=(2, 3) dtype=float32, numpy=
array([[ 2., 42.,  0.],
       [ 8., 10.,  1.]], dtype=float32)>
```

# Variables

```
>>> try:
    v[1] = [7., 8., 9.]
except TypeError as ex:
    print(ex)
'ResourceVariable' object does not support item assignment

>>> v.scatter_nd_update(indices=[[0, 0], [1, 2]],
                        updates=[100., 200.])
<tf.Variable 'UnreadVariable' shape=(2, 3) dtype=float32, numpy=
array([[100.,  42.,   0.],
       [  8.,  10., 200.]], dtype=float32)>

>>> sparse_delta = tf.IndexedSlices(values=[[1., 2., 3.], [4.,
5., 6.]],
                                    indices=[1, 0])

>>> v.scatter_update(sparse_delta)
<tf.Variable 'UnreadVariable' shape=(2, 3) dtype=float32, numpy=
array([[4., 5., 6.],
       [1., 2., 3.]], dtype=float32)>
```

# Indexing and Operations

- Indexing and slicing works much like in NumPy:

```
>>> t[1,1]
<tf.Tensor: id=46, shape=(), dtype=float32, numpy=5.0>
>>> t[:, 1:]
<tf.Tensor: id=5, shape=(2, 2), dtype=float32, numpy=
array([[2., 3.],
       [5., 6.]], dtype=float32)>
```

- All standard and expected tensor and mathematical operations are available:

```
>>> t + 10
<tf.Tensor: id=18, shape=(2, 3), dtype=float32,
numpy=array([[11., 12., 13.],
            [14., 15., 16.]], dtype=float32)>
```

- Note that writing `t + 10` is equivalent to calling `tf.add(t, 10)` (indeed, Python calls the magic method `t.__add__(10)`**

```
>>> tf.add(t,10)
<tf.Tensor: id=51, shape=(2, 3), dtype=float32, numpy=
array([[11., 12., 13.],
       [14., 15., 16.]], dtype=float32)>
```

```
>>> t*10 # equivalent to tf.multiply(t,10)
<tf.Tensor: id=56, shape=(2, 3), dtype=float32, numpy=
array([[10., 20., 30.],
       [40., 50., 60.]], dtype=float32)>
```

- As you see, these simple operations act elementwise. Scalar 10 in both add and multiply operations was applied to the every element of the tensor.

# Operations

- Other operators (like -, \*, etc.) are also supported.
- Standard mathematical functions are supported

```
>>> tf.square(t)
<tf.Tensor: id=20, shape=(2, 3), dtype=float32, numpy=
array([[ 1.,  4.,  9.],
       [16., 25., 36.]], dtype=float32)>
```

```
>>> tf.sqrt(t)
<tf.Tensor: id=20, shape=(2, 3), dtype=float32, numpy=
array([[1.          ,  1.4142135,  1.7320508],
       [2.          ,  2.236068 ,  2.4494898]], dtype=float32)>
```

- To multiply 2 matrices, we could use new Python matrix-multiplication operator @, like:

```
>>> t @ tf.transpose(t)
<tf.Tensor: id=24, shape=(2, 2), dtype=float32, numpy=
array([[14., 32.],
       [32., 77.]], dtype=float32)>
```

- or

```
>>> tf.linalg.matmul(t, tf.transpose(t))
```

- Many functions and classes have aliases. For example, `tf.add()` and `tf.math.add()` are the same function. This allows TensorFlow to have concise names for the most common operations, while preserving well organized packages. `tf.math.log()` has no `tf.log()` alias

# Comparison with NumPy operations

- All the basic math operations, e.g., `tf.add()`, `tf.multiply()`, `tf.square()`, `tf.exp()`, `tf.sqrt()`, etc., and most other tensor manipulation operations that you can find in NumPy (e.g., `tf.reshape()`, `tf.squeeze()`, `tf.tile()`) are present in TensorFlow.
- Sometimes TF operations have a different name (e.g., `tf.reduce_mean()`, `tf.reduce_sum()`, `tf.reduce_max()`, `tf.math.log()` are the equivalent of `np.mean()`, `np.sum()`, `np.max()` and `np.log()`).
- When the name differs, there is often a good reason for it. For example, in TensorFlow you must write `tf.transpose(t)`, you cannot just write `t.T` like in NumPy. The reason is that it does not do exactly the same thing.
- In TensorFlow, a new tensor is created with its own copy of the transposed data, while in NumPy, `t.T` is just a transposed view on the same data.
- Similarly, the `tf.reduce_sum()` operation is named this way because its GPU kernel (i.e., GPU implementation) uses a reduce algorithm that does not guarantee the order in which the elements are added: because 32-bit floats have limited precision, this means that the result may change ever so slightly every time you call this operation. The same is true of `tf.reduce_mean()`.
- Of course `tf.reduce_max()` is deterministic.

# Conversion between TF and NumPy types

- We can create a tensor from a NumPy array, and vice versa.
- We can even apply TensorFlow operations to NumPy arrays and NumPy operations to tensors.

- Convert numpy array into a tensor using `tf.constant()`

```
>>> a = np.array([2., 4., 5.])
>>> tf.constant(a)
<tf.Tensor: id=111, shape=(3,), dtype=float64, numpy=array([2., 4., 5.])>
```

- Convert tensor into a numpy array using `.numpy()`

```
>>> t.numpy() # or np.array(t)
array([[1., 2., 3.],
       [4., 5., 6.]], dtype=float32)
```

- Conversion can be implicit. TF operation converts numpy types into tensors and vice-versa

```
>>> tf.square(a)
<tf.Tensor: id=116, shape=(3,), dtype=float64, numpy=array([4., 16., 25.])>
>>> np.square(t)
array([[ 1.,  4.,  9.],
       [16., 25., 36.]], dtype=float32)
```

- Notice: NumPy uses 64-bit precision by default, while Tensor-Flow uses 32-bit. This is because 32-bit precision is generally more than enough for neural networks, plus it runs faster and uses less RAM.
- So when you create a tensor from a NumPy array, make sure to set `dtype=tf.float32`.

# Data Types in TensorFlow

- TensorFlow, in difference to regular Python, comes with a modest set of basic types.
- `class DType`: Represents the type of the elements in a Tensor and has 4 basic functions:
  - `as_dtype(...)`: Converts the given `type_value` to a `DType`.
  - `cast(...)`: Casts a tensor to a new type.
  - `complex(...)`: Converts two real numbers to a complex number.
  - `saturate_cast(...)`: Performs a safe saturating cast of value to dtype.
- Various types are members of class `Dtype`. Those are:
  - `tf.float16`: 16-bit half-precision floating-point.
  - `tf.float32`: 32-bit single-precision floating-point.
  - `tf.float64`: 64-bit double-precision floating-point.
  - `tf.bfloat16`: 16-bit truncated floating-point.
  - `tf.complex64`: 64-bit single-precision complex.
  - `tf.complex128`: 128-bit double-precision complex.
  - `tf.int8`: 8-bit signed integer.
  - `tf.uint8`: 8-bit unsigned integer.
  - `tf.uint16`: 16-bit unsigned integer.
  - `tf.uint32`: 32-bit unsigned integer.
  - `tf.uint64`: 64-bit unsigned integer.
  - `tf.int16`: 16-bit signed integer.
  - `tf.int32`: 32-bit signed integer.
  - `tf.int64`: 64-bit signed integer.
  - `tf.bool`: Boolean.
  - `tf.string`: String.
  - `tf.qint8`: Quantized 8-bit signed integer.
  - `tf.quint8`: Quantized 8-bit unsigned integer.
  - `tf.qint16`: Quantized 16-bit signed integer.
  - `tf.quint16`: Quantized 16-bit unsigned integer.
  - `tf.qint32`: Quantized 32-bit signed integer.
  - `tf.resource`: Handle to a mutable resource.
  - `tf.variant`: Values of arbitrary types.



# Type Conversion

- Type conversions can significantly hurt performance. Conversions can easily go unnoticed when they are done automatically. To avoid this, TensorFlow does not perform any type conversions automatically: it just raises an exception if you try to execute an operation on tensors with incompatible types.
- For example, you cannot add a float tensor and an integer tensor, and you cannot even add a 32-bit float and a 64-bit float:

```
>>> tf.constant(2.) + tf.constant(40)
Traceback[...]InvalidArgumentError[...]expected to be a float[...]
>>> tf.constant(2.) + tf.constant(40., dtype=tf.float64)
Traceback[...]InvalidArgumentError[...]expected to be a double[...]
```

- This may be a bit annoying. It is done for a good cause!
- You can use `tf.cast()` when you really need to convert types:

```
>>> t2 = tf.constant(40., dtype=tf.float64)
>>> tf.constant(2.0) + tf.cast(t2, tf.float32)
<tf.Tensor: id=136, shape=(), dtype=float32, numpy=42.0>
```

# Other Data Structures

- TensorFlow supports several other data structures, like:
- *Sparse tensors* (`tf.SparseTensor`) efficiently represent tensors containing mostly 0s. The `tf.sparse` package contains operations for sparse tensors.
- *Tensor arrays* (`tf.TensorArray`) are lists of tensors. They have a fixed size by default, but can optionally be made dynamic. All tensors they contain must have the same shape and data type.
- *Ragged tensors* (`tf.RaggedTensor`) represent static lists of lists of tensors, where every tensor has the same shape and data type. The `tf.ragged` package contains operations for ragged tensors.
- *String tensors* are regular tensors of type `tf.string`. These represent byte strings, not Unicode strings, so if you create a string tensor using a Unicode string (e.g., a regular Python 3 string like `"café"`), then it will get encoded to UTF-8 automatically (e.g., `b"caf\xc3\xa9"`). Alternatively, you can represent Unicode strings using tensors of type `tf.int32`, where each item represents a Unicode codepoint (e.g., `[99, 97, 102, 233]`). The `tf.strings` package (with an `s`) contains ops for byte strings and Unicode strings (and to convert one into the other).
- *Sets* are just represented as regular tensors (or sparse tensors) containing one or more sets, and you can manipulate them using operations from the `tf.sets` package.
- *Queues*, including First In, First Out (FIFO) queues (`FIFOQueue`), queues that can prioritize some items (`PriorityQueue`), queues that shuffle their items (`RandomShuffleQueue`), and queues that can batch items of different shapes by padding (`PaddingFIFOQueue`). These classes are all in the `tf.queue` package.

# Strings

```
>>> tf.constant(b"hello world")
<tf.Tensor: id=33, shape=(), dtype=string, numpy=b'hello world'>

>>> tf.constant("café")
<tf.Tensor: id=34, shape=(), dtype=string, numpy=b'caf\xc3\xa9'>

>>> u = tf.constant([ord(c) for c in "café"])
>>> u
<tf.Tensor: id=35, shape=(4,), dtype=int32, numpy=array([ 99,
 97, 102, 233], dtype=int32)>

>>> b = tf.strings.unicode_encode(u, "UTF-8")
>>> tf.strings.length(b, unit="UTF8_CHAR")
<tf.Tensor: id=46, shape=(), dtype=int32, numpy=4>

>>> tf.strings.unicode_decode(b, "UTF-8")
<tf.Tensor: id=50, shape=(4,), dtype=int32, numpy=array([ 99,
 97, 102, 233], dtype=int32)>
```

# String Arrays

```
>>> p = tf.constant(["Café", "Coffee", "caffè", "咖啡"])
>>> p
<tf.Tensor: id=13, shape=(4,), dtype=string, numpy=
array([b'Caf\xc3\xa9', b'Coffee', b'caff\xc3\xa8',
      b'\xe5\x92\x96\xe5\x95\xa1'], dtype=object)>

>>> tf.strings.length(p, unit="UTF8_CHAR")
<tf.Tensor: id=14, shape=(4,), dtype=int32, numpy=array([4, 6, 5,
2])>

>>> r = tf.strings.unicode_decode(p, "UTF8")
>>> r
<tf.RaggedTensor [[67, 97, 102, 233], [67, 111, 102, 102, 101,
101], [99, 97, 102, 102, 232], [21654, 21857]]>

>>> print(r)
<tf.RaggedTensor [[67, 97, 102, 233], [67, 111, 102, 102, 101,
101], [99, 97, 102, 102, 232], [21654, 21857]]>
```

# Ragged Tensors

```
>>> print(r[1])
tf.Tensor([ 67 111 102 102 101 101], shape=(6,), dtype=int32)

>>> print(r[1:3])
<tf.RaggedTensor [[67, 111, 102, 102, 101, 101], [99, 97, 102, 102, 232]]>

>>> r2 = tf.ragged.constant([[65, 66], [], [67]])
>>> print(tf.concat([r, r2], axis=0))
<tf.RaggedTensor [[67, 97, 102, 233], [67, 111, 102, 102, 101, 101], [99,
97, 102, 102, 232], [21654, 21857], [65, 66], [], [67]]>

>>> r3 = tf.ragged.constant([[68, 69, 70], [71], [], [72, 73]])
>>> print(tf.concat([r, r3], axis=1))
<tf.RaggedTensor [[67, 97, 102, 233, 68, 69, 70], [67, 111, 102, 102, 101,
101, 71], [99, 97, 102, 102, 232], [21654, 21857, 72, 73]]>

>>> tf.strings.unicode_encode(r3, "UTF-8")
<tf.Tensor: id=202, shape=(4,), dtype=string, numpy=array([b'DEF', b'G',
b'', b'HI'], dtype=object)>

>>> r.to_tensor()
<tf.Tensor: id=267, shape=(4, 6), dtype=int32, numpy=
array([[ 67,    97,   102,   233,     0,     0],
       [ 67,   111,   102,   102,   101,   101],
       [ 99,    97,   102,   102,   232,     0],
       [21654, 21857,     0,     0,     0,     0]], dtype=int32)>
```

# Sparse Tensors

```
>>> s = tf.SparseTensor(indices=[[0, 1], [1, 0], [2, 3]],
                        values=[1., 2., 3.],
                        dense_shape=[3, 4])

>>> print(s)
SparseTensor(indices=tf.Tensor(
[[0 1]
 [1 0]
 [2 3]], shape=(3, 2), dtype=int64), values=tf.Tensor([1. 2. 3.], shape=(3,), dtype=float32),
dense_shape=tf.Tensor([3 4], shape=(2,), dtype=int64))

>>> tf.sparse.to_dense(s)
<tf.Tensor: id=272, shape=(3, 4), dtype=float32, numpy=
array([[0., 1., 0., 0.],
       [2., 0., 0., 0.],
       [0., 0., 0., 3.]], dtype=float32)>

>>> s2 = s * 2.0
>>> try:
    s3 = s + 1.
except TypeError as ex:
    print(ex)
unsupported operand type(s) for +: 'SparseTensor' and 'float'

>>> s4 = tf.constant([[10., 20.], [30., 40.], [50., 60.], [70., 80.]])
tf.sparse.sparse_dense_matmul(s, s4)
<tf.Tensor: id=276, shape=(3, 2), dtype=float32, numpy=
array([[ 30.,  40.],
       [ 20.,  40.],
       [210., 240.]], dtype=float32)>
```

# Sparse Tensors

```
>>> s5 = tf.SparseTensor(indices=[[0, 2], [0, 1]],
                        values=[1., 2.],
                        dense_shape=[3, 4])

>>> print(s5)
SparseTensor(indices=tf.Tensor(
[[0 2]
 [0 1]], shape=(2, 2), dtype=int64), values=tf.Tensor([1. 2.], shape=(2,),
dtype=float32), dense_shape=tf.Tensor([3 4], shape=(2,), dtype=int64))

>>> try:
    tf.sparse.to_dense(s5)
except tf.errors.InvalidArgumentError as ex:
    print(ex)

indices[1] = [0,1] is out of order [Op:SparseToDense]

>>> s6 = tf.sparse.reorder(s5)
tf.sparse.to_dense(s6)
<tf.Tensor: id=285, shape=(3, 4), dtype=float32, numpy=
array([[0., 2., 1., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]], dtype=float32)>
```

# Sets

```
>>> set1 = tf.constant([[2, 3, 5, 7], [7, 9, 0, 0]])
>>> set2 = tf.constant([[4, 5, 6], [9, 10, 0]])
>>> tf.sparse.to_dense(tf.sets.union(set1, set2))
<tf.Tensor: id=292, shape=(2, 6), dtype=int32, numpy=
array([[ 2,  3,  4,  5,  6,  7],
       [ 0,  7,  9, 10,  0,  0]], dtype=int32)>
```

```
>>> tf.sparse.to_dense(tf.sets.difference(set1, set2))
<tf.Tensor: id=297, shape=(2, 3), dtype=int32, numpy=
array([[2, 3, 7],
       [7, 0, 0]], dtype=int32)>
```

```
>>> tf.sparse.to_dense(tf.sets.intersection(set1, set2))
<tf.Tensor: id=302, shape=(2, 2), dtype=int32, numpy=
array([[5, 0],
       [0, 9]], dtype=int32)>
```



# Tensor Arrays

```
>>> array = tf.TensorArray(dtype=tf.float32, size=3)
>>> array = array.write(0, tf.constant([1., 2.]))
>>> array = array.write(1, tf.constant([3., 10.]))
>>> array = array.write(2, tf.constant([5., 7.]))

>>> array.read(1)
<tf.Tensor: id=337, shape=(2,), dtype=float32, numpy=array([ 3., 10.],
dtype=float32)>

>>> array.stack()
<tf.Tensor: id=342, shape=(3, 2), dtype=float32, numpy=
array([[1., 2.],
       [0., 0.],
       [5., 7.]], dtype=float32)>

>>> mean, variance = tf.nn.moments(array.stack(), axes=0)
>>> mean
<tf.Tensor: id=350, shape=(2,), dtype=float32, numpy=array([2., 3.],
dtype=float32)>
>>> variance
<tf.Tensor: id=351, shape=(2,), dtype=float32, numpy=array([4.6666665,
8.666667 ], dtype=float32)>
```

# A Simple Example

# A Simple Example

- To give you a feeling about operations with TensorFlow, in what follows, we will present the most basic of example. Full details of this and similar examples will be presented in the following lecture.

- We start all our work, with importing TensorFlow:

```
import tensorflow as tf
print(tf.__version__)
```

- If you have not done it already, install `matplotlib` package. We will use `matplotlib` to produce most of our pretty images

```
!pip install matplotlib
import numpy as np
import matplotlib.pyplot as plt
```

- Load MNIST dataset of images of handwritten digits.

```
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
```

- The `load_data()` operation splits the MNIST dataset into a larger portion of training images and labels and a smaller portion of test images and labels.

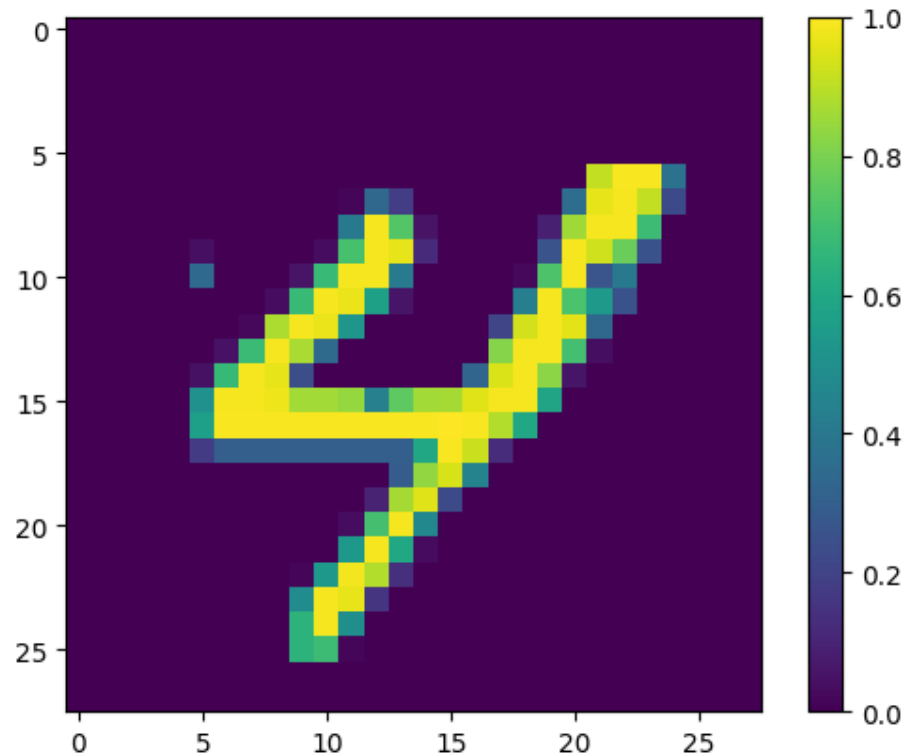
# MNIST Images

- Images are of 28X28 pixels.
- Labels are integers: 0, 1, 2,..., 9

```
print(x_train.shape, y_train.shape)
(60000, 28, 28) (60000,)
y_train
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

- Inspect one of the images:

```
index = 89
print(y_train[index])
plt.figure()
plt.imshow(x_train[index])
plt.colorbar()
plt.grid(False)
plt.show()
4
```



# Build a Machine Learning Model

- We will use a `Sequential` model which is useful for stacking layers where each layer has one input tensor and one output tensor. Layers are functions with a known mathematical structure that can be reused and have trainable variables. Most TensorFlow models are composed of layers. This model uses the `Flatten`, `Dense`, and `Dropout` layers.

```
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)
])
```

- For each example image, the model returns a vector of ten scores (we have 10 labels, 0,1,2,...,9) one for each class.

```
predictions = model(x_train[:1]).numpy()
predictions
array([[ -0.06450526, -0.3642863 , -0.02172264, -0.69311845,  0.68280196,
         0.67123336,  0.434183  , -1.1686009 , -0.0384794 , -0.1947177 ]],
      dtype=float32)
```

- The `tf.nn.softmax` function could convert these scores to probabilities for each class:

```
tf.nn.softmax(predictions).numpy()
array([[0.08772931, 0.0650057 , 0.09156404, 0.04678876, 0.18522352,
        0.18309309, 0.14445157, 0.02908319, 0.09004252, 0.07701835]] ,
      dtype=float32)
```

# Loss Function and Optimizer

- Two most essential components of Deep Learning models or processes are a loss function and an optimizer.
- The loss function is used to assess the quality of fit between the scores produced by the network and the expected labels.
- The optimizer is a class which performs the search, i.e., the optimization process.
- As the loss function we will use `losses.SparseCategoricalCrossentropy`:  
`loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)`
- The loss function takes a vector of ground truth values and a vector of logits and returns a scalar loss for each example. This loss is equal to the negative log probability of the true class: The loss is zero if the model is sure of the correct class.
- This untrained model gives probabilities close to random (1/10 for each class), so the initial loss should be close to `-tf.math.log(1/10) ~= 2.3`.

```
loss_fn(y_train[:1], predictions).numpy()  
1.6977606
```

- Before you start training, configure and compile the model using `Keras Model.compile`. Set the optimizer class to `adam`, set the loss to the `loss_fn` function you defined earlier, and specify a `metric` to be evaluated for the model by setting the `metrics` parameter to `accuracy`.

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

# Train and Evaluate

- We use the `Model.fit` method to train the model, i.e., to adjust our model parameters and minimize the loss function. `Epochs=5` specifies that we will make 5 passes through the entire dataset:

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

```
Epoch 1/5
```

```
1875/1875 [=====] - 3s 1ms/step - loss: 0.2934 - accuracy: 0.9151
```

```
Epoch 2/5
```

```
1875/1875 [=====] - 3s 1ms/step - loss: 0.1411 - accuracy: 0.9583
```

```
Epoch 3/5
```

```
1875/1875 [=====] - 2s 1ms/step - loss: 0.1073 - accuracy: 0.9675
```

```
Epoch 4/5
```

```
1875/1875 [=====] - 2s 1ms/step - loss: 0.0873 - accuracy: 0.9730
```

```
Epoch 5/5
```

```
1875/1875 [=====] - 2s 1ms/step - loss: 0.0752 - accuracy: 0.9764
```

```
<keras.src.callbacks.History at 0x2a1e6af0850>
```

- The `Model.evaluate` method checks the model's performance, usually on a validation set or test set.

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

```
313/313 - 0s - loss: 0.0754 - accuracy: 0.9763 - 376ms/epoch - 1ms/step
```

```
[0.07537086308002472, 0.9763000011444092]
```

- The image classifier is now trained to ~98% accuracy on this dataset.

# Appendix: TensorFlow Implementation

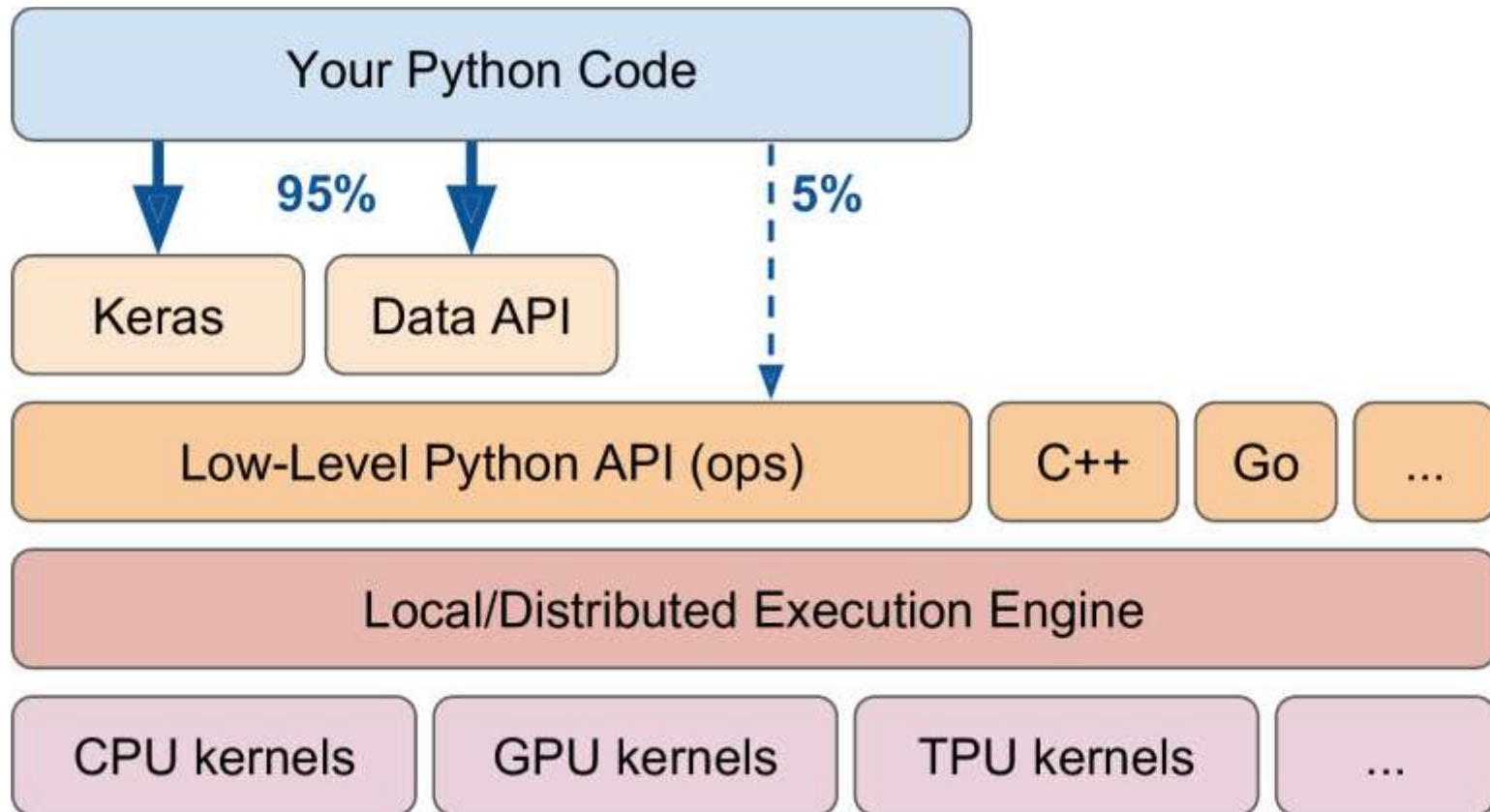


# Implementation

- At the lowest level, each TensorFlow operation is implemented using highly efficient C++ code.
- Many operations have multiple implementations, called *kernels*. Each kernel is dedicated to a specific device type, such as CPUs, GPUs, or TPUs (*Tensor Processing Units*).
- GPUs can dramatically speed up computations by splitting computations into many smaller chunks and running them in parallel across many GPU threads.
- TPUs are even faster. You can purchase your own GPU devices ( TensorFlow readily supports Nvidia cards with CUDA Compute Capability 3.5+).
- TPUs are only available on *Google Cloud Machine Learning Engine* or *Google Cloud Platform*. **Colaboratory** offers CPU, GPU and TPU processors.
- TensorFlow runs on Windows, Linux, and MacOS, and mobile devices (using *TensorFlow Lite*), including both iOS and Android.
- TF works well on Linux, Windows and Mac OS machines.
- TF is being developed on Linux machines. For professional work, please use a local Linux box or a Cloud machine. All major Cloud providers, AWS, GCP, Azure and others, offer Linux boxes with preinstalled TF and associated software packages.

# TensorFlow Architecture

- Most of our code will use the high-level APIs ( tf.keras and tf.data). When we need more flexibility, we use the lower-level Python API, handling tensors directly.
- TensorFlow's execution engine takes care of running the operations efficiently, even across many devices and machines. TensorFlow is a distributed computational engine, just like Hadoop or Spark.



# TensorFlow Python APIs

- This table presents most important TensorFlow's APIs in Python

<code>tf.estimator</code> <code>tf.feature_column</code>	High-level Deep Learning APIs	<code>tf.saved_model</code> <code>tf.autograph</code> <code>tf.lite</code> <code>tf.distribute</code> <code>tf.quantization</code> <code>tf.graph_util</code>	Deployment & optimization
<code>tf.keras</code>			
<code>tf.data</code> <code>tf.io</code> <code>tf.queue</code> <code>tf.image</code> <code>tf.strings</code>	I/O and preprocessing	<code>tf.summary</code>	Visualization with TensorBoard
<code>tf.nn</code> <code>tf.rnn</code> <code>tf.train</code>	Low-level Deep Learning API	<code>tf.sparse</code> <code>tf.ragged</code> <code>tf.sets</code>	Special data structures
<code>tf.GradientTape</code> <code>tf.gradients()</code>	Autodiff	<code>tf.math</code> <code>tf.linalg</code> <code>tf.signal</code> <code>tf.random</code> <code>tf.bitwise</code>	Mathematics, including linear algebra, signal processing, and more

# GPUs

- Graphical Processing Units (GPUs) were originally developed for games. GPUs have hundreds and even thousands of small, simple, processing units and offer massive parallelization.
- It turned out that for Deep Learning tasks we need massive parallelization and do not need long word precision. GPU card proved almost ideal environment for DL tasks.
- Major vendors are Nvidia and ADM. There are a few others.
- Prices of decent NVIDIA cards range from \$100s to \$1000s. GPU cards could be parallelized and ran 2, 3, 4 or many more in the same machine giving you supercomputer performance. To run more than 2 GPU cards in a single machine you need special skills. Learn those skills before you buy too many carts.
- Some APIs work with one vendor only. Until recently TensorFlow would work with Nvidia cards and not with ADM cards. Please check, before you buy an ADM card.



A promotional banner for the NVIDIA GeForce RTX 2070. The banner has a dark background. At the top, the "GEFORCE" logo is on the left, and navigation links for "PRODUCTS", "GEFORCE EXPERIENCE", "DRIVERS", "GAMES", "NEWS", and "COMMUNITY" are on the right. Below this, "GEFORCE RTX 2070" is written in white. To the right of this, "TURING" and "SEE RTX" are visible. The main text "RTX. IT'S ON." is in large white letters, with "GEFORCE RTX 2070" in green below it. To the right of the text is a small image of a game character. Further right, text says "Choose *Battlefield V* or *Anthem* game code with purchase, limited offer". At the bottom, a paragraph describes the RTX 2070's advantage of NVIDIA Turing architecture.

**GEFORCE** PRODUCTS ▾ GEFORCE EXPERIENCE DRIVERS GAMES ▾ NEWS COMMUNITY

**GEFORCE RTX 2070** TURING SEE RTX

**RTX. IT'S ON.**  
**GEFORCE RTX 2070**

Choose *Battlefield V* or *Anthem* game code with purchase, limited offer

The powerful new GeForce RTX<sup>™</sup> 2070 takes advantage of the cutting-edge **NVIDIA Turing<sup>™</sup>** architecture to immerse you in incredible realism and performance in the latest games. The future of gaming starts here.

# TPUs

- A **Tensor Processing Unit (TPU)** is an [AI accelerator application-specific integrated circuit](#) (ASIC) developed by [Google](#) specifically for [neural network machine learning](#).
- Google's TPUs are proprietary and are not commercially available. Google allows other companies to access to those chips through its cloud-computing service.
- Google has stated that TPUs were used in the [AlphaGo versus Lee Sedol](#) series of man-machine [Go](#) games, as well as in the [AlphaZero](#) system which produced [Chess](#), [Shogi](#) and Go playing programs from the game rules alone and went on to beat the leading programs in those games.
- Google has also uses TPUs for [Google Street View](#) text processing, and is able to find all the text in the Street View database in less than five days. In [Google Photos](#), an individual TPU can process over 100 million photos a day. It is also used in [RankBrain](#) which Google uses to provide search results.
- There are rumors that Tesla is introducing its own DL Accelerator Chip through project DOJO (<https://www.youtube.com/watch?v=keWEE9FwS9o&t=585s>) . Those chips might not be available for public use for a while.

# FPGA

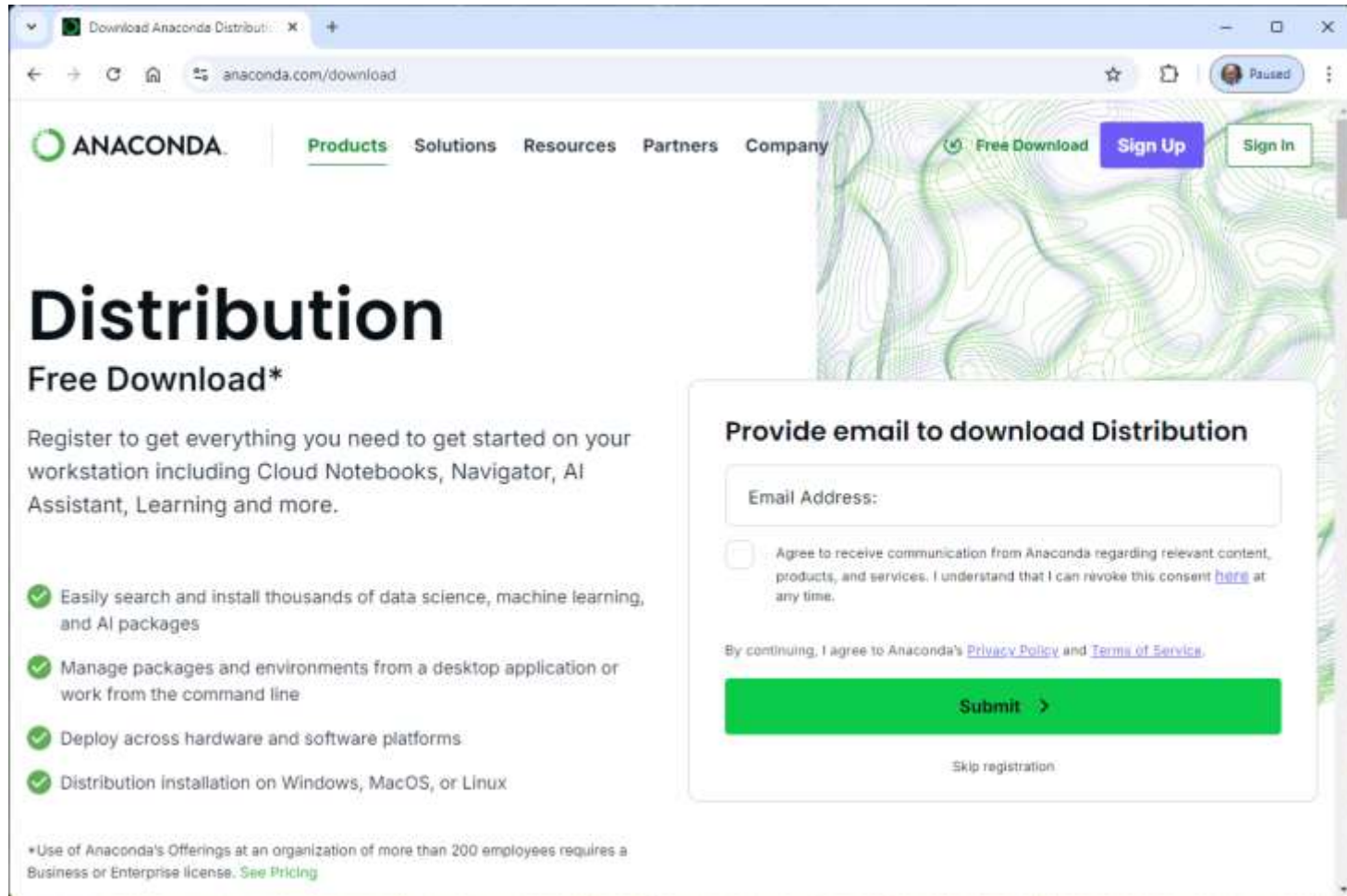
- A **field-programmable gate array (FPGA)** is an [integrated circuit](#) designed to be configured by a customer or a designer after manufacturing – hence the term "[field-programmable](#)". The FPGA configuration is generally specified using a [hardware description language](#) (HDL), similar to that used for an [application-specific integrated circuit](#) (ASIC).
- **Amazon AWS** offers FPGA Developer AMIs. The AMIs have supported and maintained Linux images provided by Amazon Web Services. Those AMIs are pre-built with FPGA development tools.
- **Intel** FPGA-based acceleration platforms include PCIe\*-based programmable acceleration cards, socket-based server platforms with integrated FPGAs, and others that are supported by the Acceleration Stack for Intel Xeon CPU with FPGAs. Intel platforms are qualified, validated, and deployed through several leading OEM server providers such as Dell, Fujitsu, and Hewlett Packard Enterprise (HPE).
- FPGA machines offer performance orders of magnitude better than CPU or GPU machines. However, vendors believe that you should program in Java or C. Lack of high level, modern, programming environments and somewhat higher prices of FPGA machines keeps them out of the main-stream, for now. Military is interested, though.

# Appendix: Anaconda and TensorFlow Installation



# Anaconda

- If you are using some other distribution of Python, like [Python.org](https://python.org) and you know what you are doing, do not bother. Otherwise, install Anaconda.
- Anaconda is a collection of data science-oriented Python packages.



The screenshot shows the Anaconda website's download page. The browser address bar shows 'anaconda.com/download'. The page features the Anaconda logo and a navigation menu with links to Products, Solutions, Resources, Partners, and Company. On the right, there are buttons for 'Free Download', 'Sign Up', and 'Sign In'. The main heading is 'Distribution' with a subheading 'Free Download\*'. Below this, a paragraph explains that registration provides access to Cloud Notebooks, Navigator, AI Assistant, and more. A list of four benefits is shown with green checkmarks: easily searching and installing thousands of data science packages, managing packages from a desktop application or command line, deploying across hardware and software platforms, and distribution installation on Windows, MacOS, or Linux. A small footnote at the bottom left states that use at organizations with more than 200 employees requires a Business or Enterprise license. A registration form on the right asks for an email address, includes a checkbox for agreeing to communication, and a 'Submit' button. A 'Skip registration' link is also present.

Download Anaconda Distribution

anaconda.com/download

ANACONDA

Products Solutions Resources Partners Company

Free Download Sign Up Sign In

## Distribution

### Free Download\*

Register to get everything you need to get started on your workstation including Cloud Notebooks, Navigator, AI Assistant, Learning and more.

- ✓ Easily search and install thousands of data science, machine learning, and AI packages
- ✓ Manage packages and environments from a desktop application or work from the command line
- ✓ Deploy across hardware and software platforms
- ✓ Distribution installation on Windows, MacOS, or Linux

\*Use of Anaconda's Offerings at an organization of more than 200 employees requires a Business or Enterprise license. [See Pricing](#)

### Provide email to download Distribution

Email Address:

☐ Agree to receive communication from Anaconda regarding relevant content, products, and services. I understand that I can revoke this consent [here](#) at any time.

By continuing, I agree to Anaconda's [Privacy Policy](#) and [Terms of Service](#).

Submit >

[Skip registration](#)



# Conda, Miniconda, Anaconda, Virtual Environments

- **Conda** is a language-agnostic tool for **package management** and **environment management**. As a package manager, Conda can install, update and remove packages. As an environment manager, it can manage virtual environments.
- **Anaconda** is the most popular Python distribution (i.e., a way of distributing Python to end users). By installing Anaconda, you get Miniconda, Anaconda Navigator (i.e. a graphical user interface) and curated selection of Python packages.
- **Miniconda** is a mini-scale version of Anaconda. It is also a Python distribution. By installing Miniconda, you get Conda, Python and a smaller number of packages.
- A virtual environment is a Python tool for **dependency management** and **project isolation**. They allow Python **site packages** (third party libraries) to be installed locally in an isolated directory for a particular project, as opposed to being installed globally (i.e., as part of a system-wide Python).
- **Virtual environments resolve dependency issues** by allowing you to use different versions of a package for different projects. For example, you could use Package A v2.7 for Project X and Package A v1.3 for Project Y.
- Make your project **self-contained** and **reproducible** by capturing all package dependencies in a requirements file.

# Conda vs Pip vs Venv

**Pip** is a package manager for Python.

- This means both Conda and Pip can be used to install, update and remove packages in Python. While Pip gives access to a wider range of packages available on [the Python Package Index \(PyPI\)](#), Conda gives access to a smaller range of packages available on [its channels](#).
- As a result, there are times when a certain package will only be available for installation via Pip. On the other hand, Conda installs some dependences that Pip ignores.

**Venv** is an environment manager for Python.

- Both Conda and Venv are great at managing virtual environments (i.e. isolated and independent environments) with different versions of packages.
- One obvious benefit of Conda over Venv is its ease of managing multiple Python versions.
- Conda can create virtual environments using different versions of Python.
- With Venv, we need to use an additional tool to manage Python versions or install multiple Python versions before creating virtual environments.

# Anaconda Prompt on Windows

## Anaconda command PROMPT

- When creating and using virtual environments, use Anaconda command prompt.
- Anaconda installation on Windows recommends not adding Anaconda to the PATH, so we have a “procedure” to get to a proper prompt. Do the following:
- Search for applications with names starting with Anaconda. One of them is `Anaconda Prompt (Anaconda3)`.
- Run that application as an administrator. That is important since the administrator enables you to install software packages. Also, on that command prompt you have access to all Anaconda packages, including the latest Python.
- In that command prompt you can do all you want to accomplish with virtual environments.

# Prompt on MacOS

- On Mac things are even simpler , no need for admin etc..
- Install Anaconda using instructions below ,

<https://docs.anaconda.com/free/anaconda/install/mac-os.html>

- These steps add records to you `~/.bash_profile` so every time a terminal window is initialized, proper Anaconda environmental variables are set.

- To test, go to the terminal window and run commands

```
$ conda --version
```

```
conda 22.9.0
```

```
(chatdocs) rjoglekar74 at MacBook-Pro-4 in ~/AppDevelopment
```

```
$ python3 --version
```

```
Python 3.11.5
```

- Using Conda and Python you can create virtual environments, run Jupyter, and do other things.

# Manipulation of Virtual Environments

## Creating virtual environments

- Type:

```
conda create --name myenv
```

- Replace `myenv` with the environment name.
- When conda asks you to proceed, type y:

```
proceed ([y]/n)?
```

- This creates the `myenv` environment in the directory `/envs/`. No packages are installed in this environment, yet.
- To create an environment with a specific version of Python, type:

```
conda create -n myenv python=3.11.4
```

- To create an environment without specifying package version, type:

```
conda create -n myenv scipy
```

- or:

```
conda create -n myenv python
```

```
conda install -n myenv scipy
```

# Virtual Environments, continued

- To create an environment with a specific version of a package, type:

```
conda create -n myenv scipy=1.10.3
```

- or:

```
conda create -n myenv python
```

```
conda install -n myenv scipy=1.10.3
```

- To create an environment with a specific version of Python and multiple packages:

```
conda create -n myenv python=3.9 scipy=1.10.3 astroid babel
```

- To activate an environment:

```
conda activate myenv
```

- To remove an environment, type:

```
conda remove -name myenv
```

- To install TensorFlow, activate an environment with a desired version of Python, e.g. 3.11.4, and type:

```
pip install tensorflow
```

- That is all.

- In that same environment, you might want to install Jupyter (notebook, server)

```
pip install jupyter
```

- As you are running TensorFlow or Jupyter, errors might appear suggesting that some Python packages are missing. Read the errors and install missing packages.

# Installing TensorFlow

- To install TensorFlow, activate an environment with a desired version of Python, e.g., 3.11.4, activate that environment and then type:

```
pip install tensorflow
```

- That is all.
- In that same environment, you might want to install Jupyter (notebook, server)

```
pip install jupyter
```

```
pip install matplotlib
```

- and other packages. As you are running TensorFlow or Jupyter, errors might appear suggesting that some Python packages are still missing. Read the errors and install missing packages.