

Neural Networks

A dark blue diagonal gradient bar that starts from the bottom left and extends towards the top right, covering the lower half of the slide.

Welcome to YCBS 258: Practical Machine Learning

Nicolas Feller

A dark blue diagonal gradient bar that starts from the bottom left corner and extends towards the top right corner, covering the lower half of the slide.

In this Course

Prerequisites

- Python, python, python
- Some statistics
- Some calculus
- Understanding of Classical Machine Learning

What you will learn

- Modern ML approaches
- Keras
- Neural Network Architectures
- Performance Optimization
- Deploy a Neural Network
- Sequence modeling, NLP, Computer vision, Generative Models

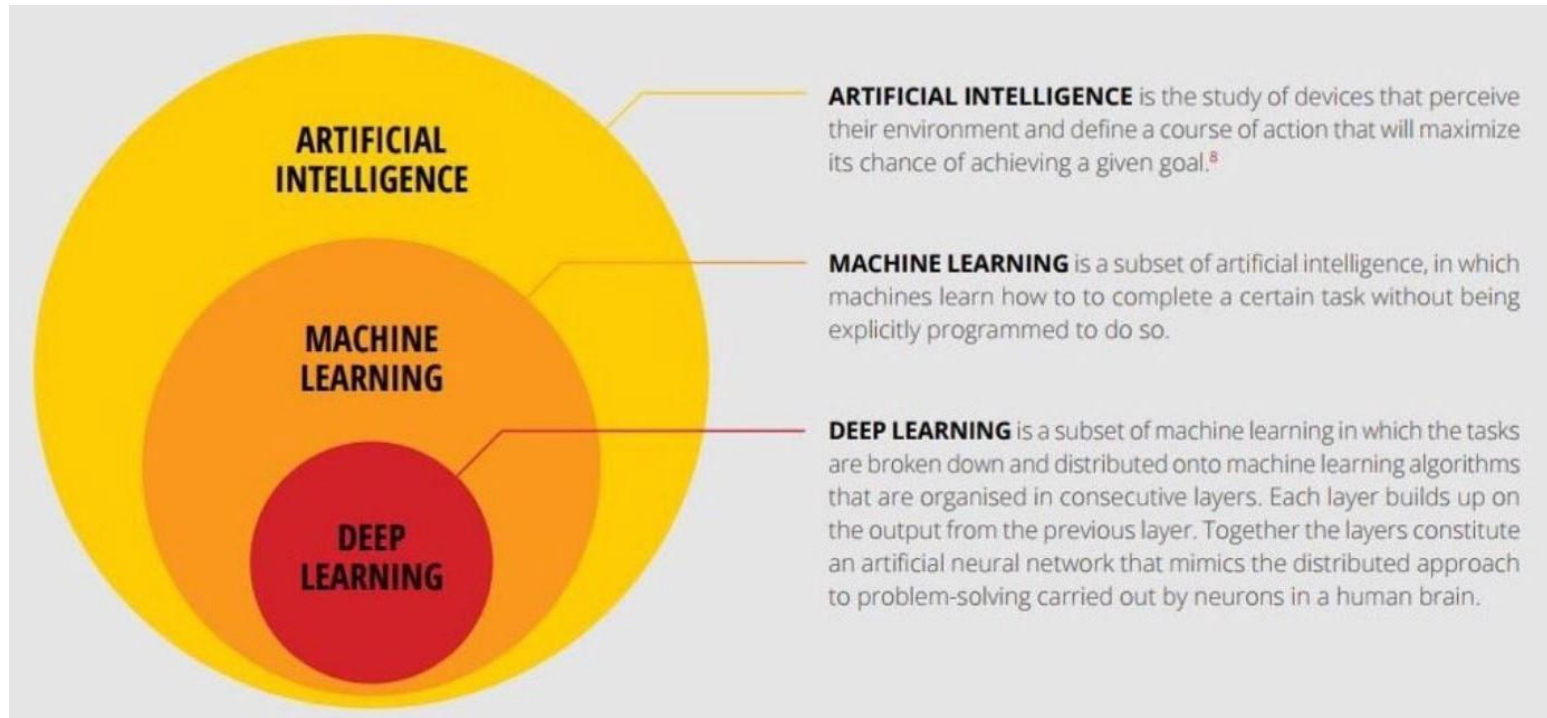
Outline



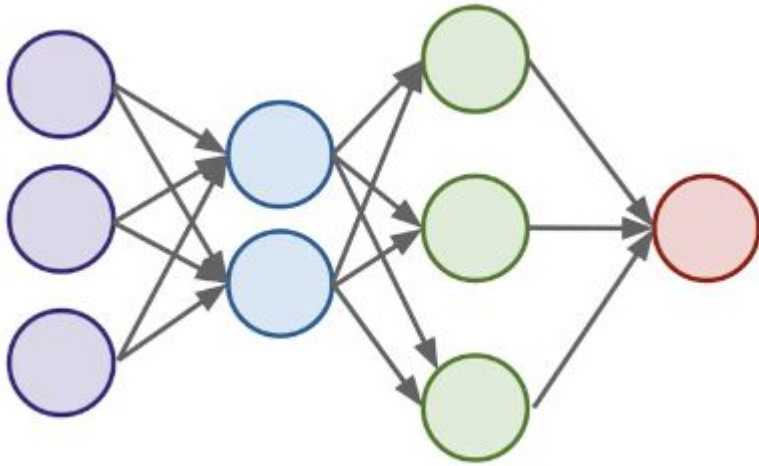
Today

- What is deep learning?
- Why should we care about deep learning?
- The history of deep learning

What is Intelligence?

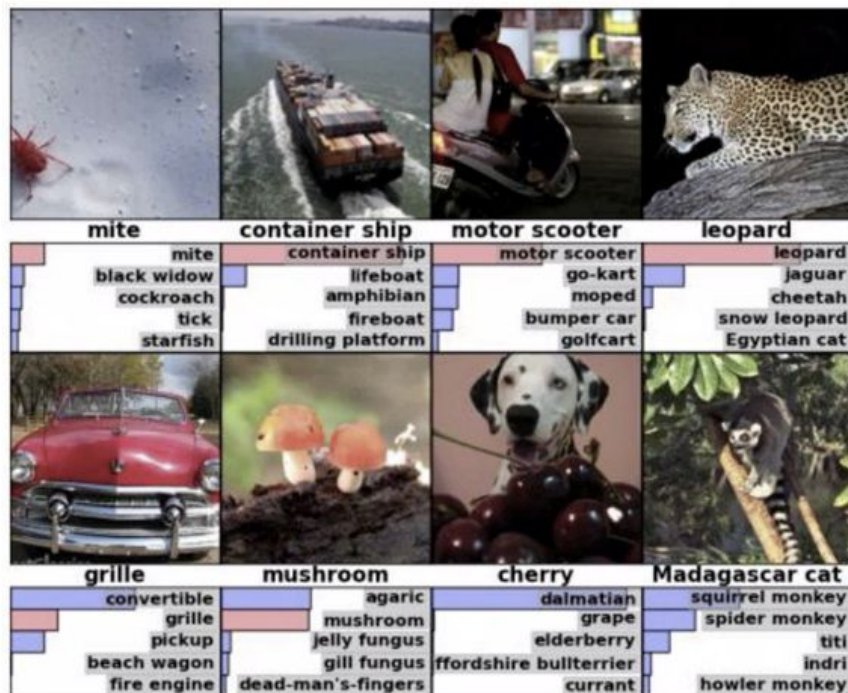


Why Neural Networks?

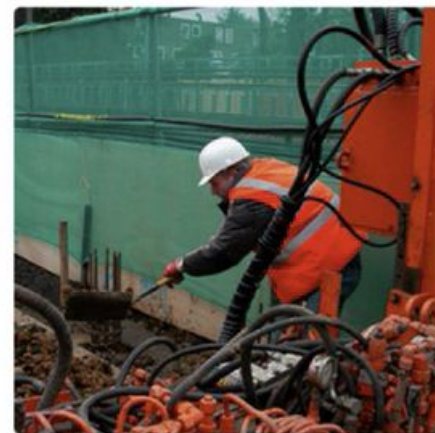


?

Object Recognition and Caption

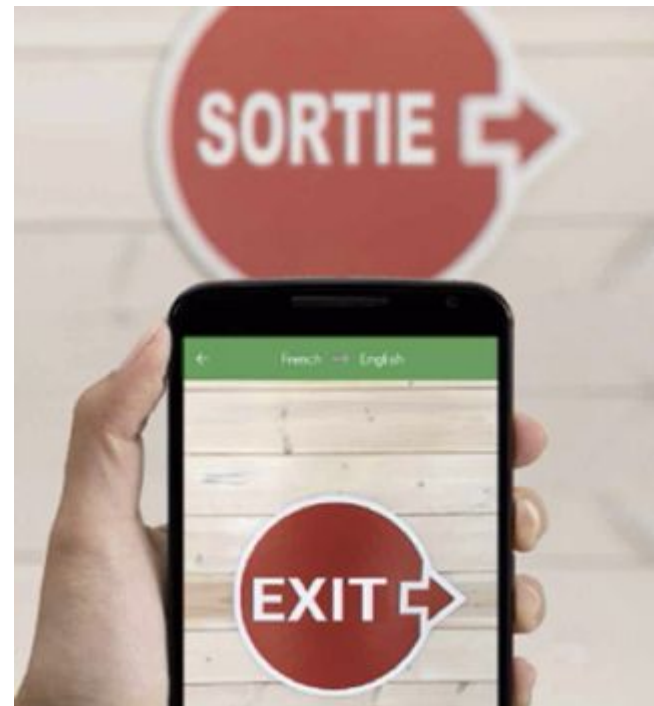
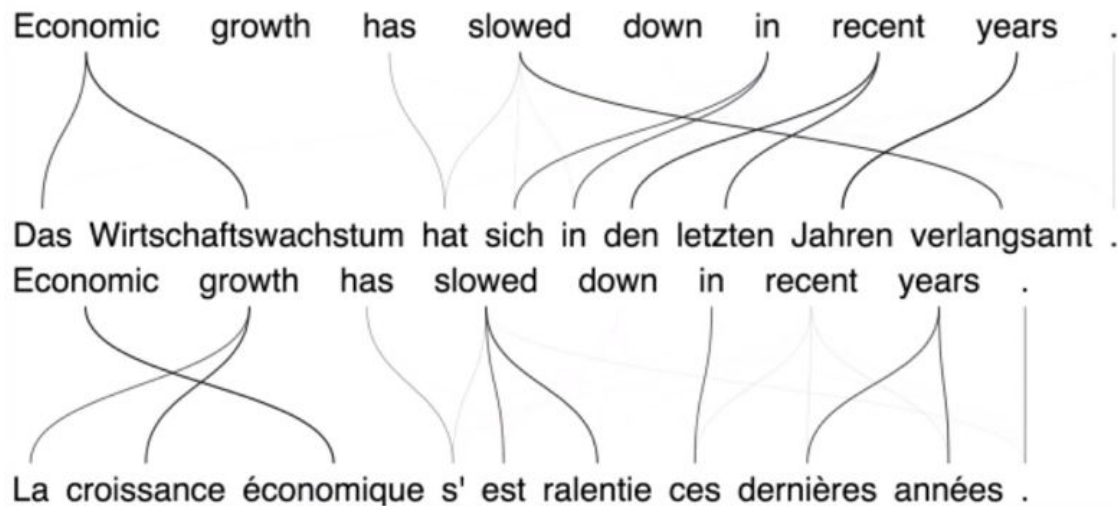


"man in black shirt is playing guitar."

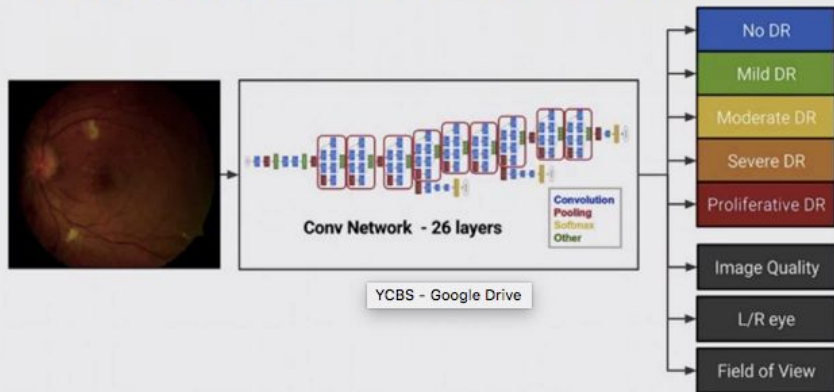


"construction worker in orange safety vest is working on road."

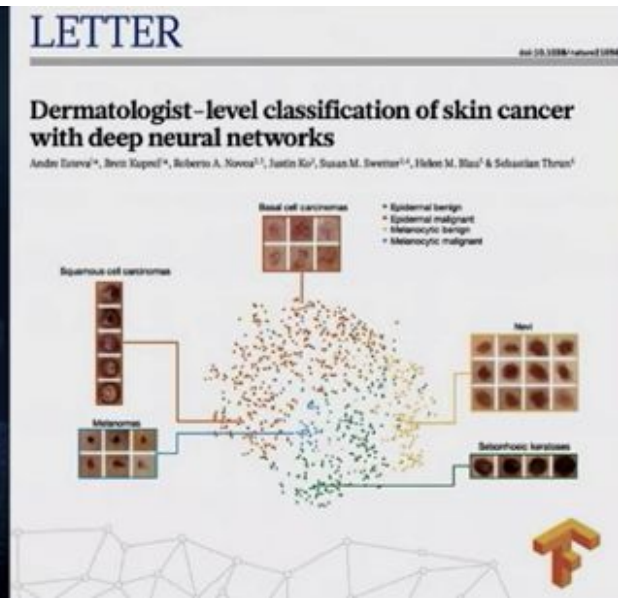
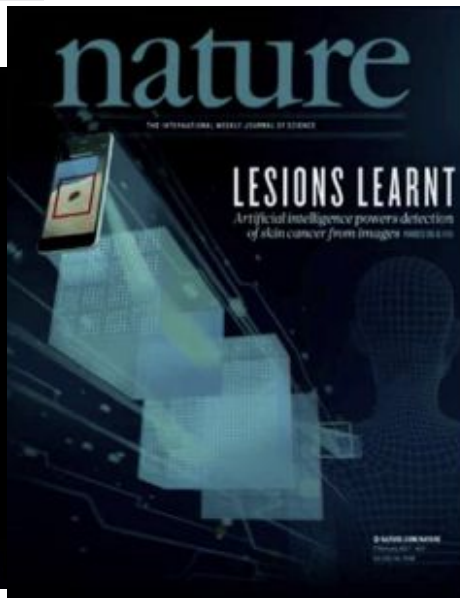
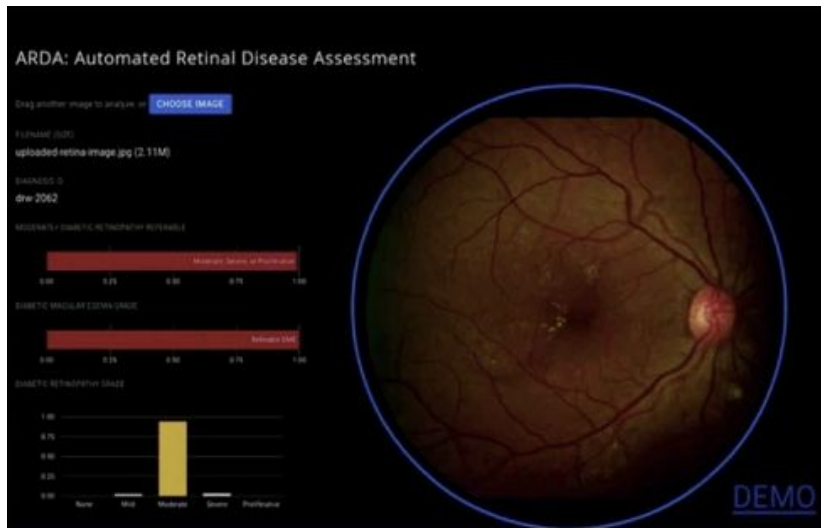
Neural Machine Translation



Adapt deep neural network to read fundus images



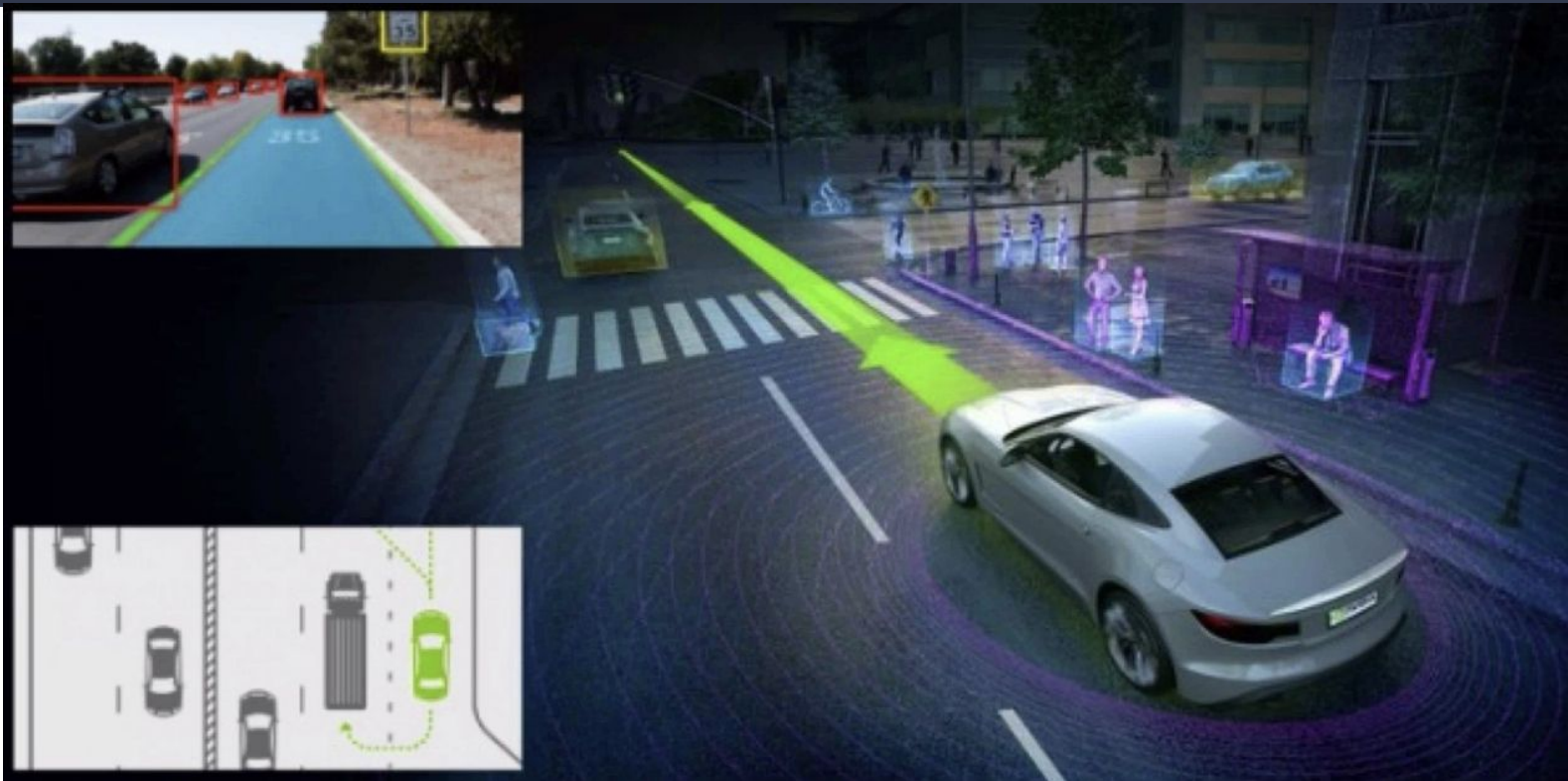
Medical Imaging / Diagnostic



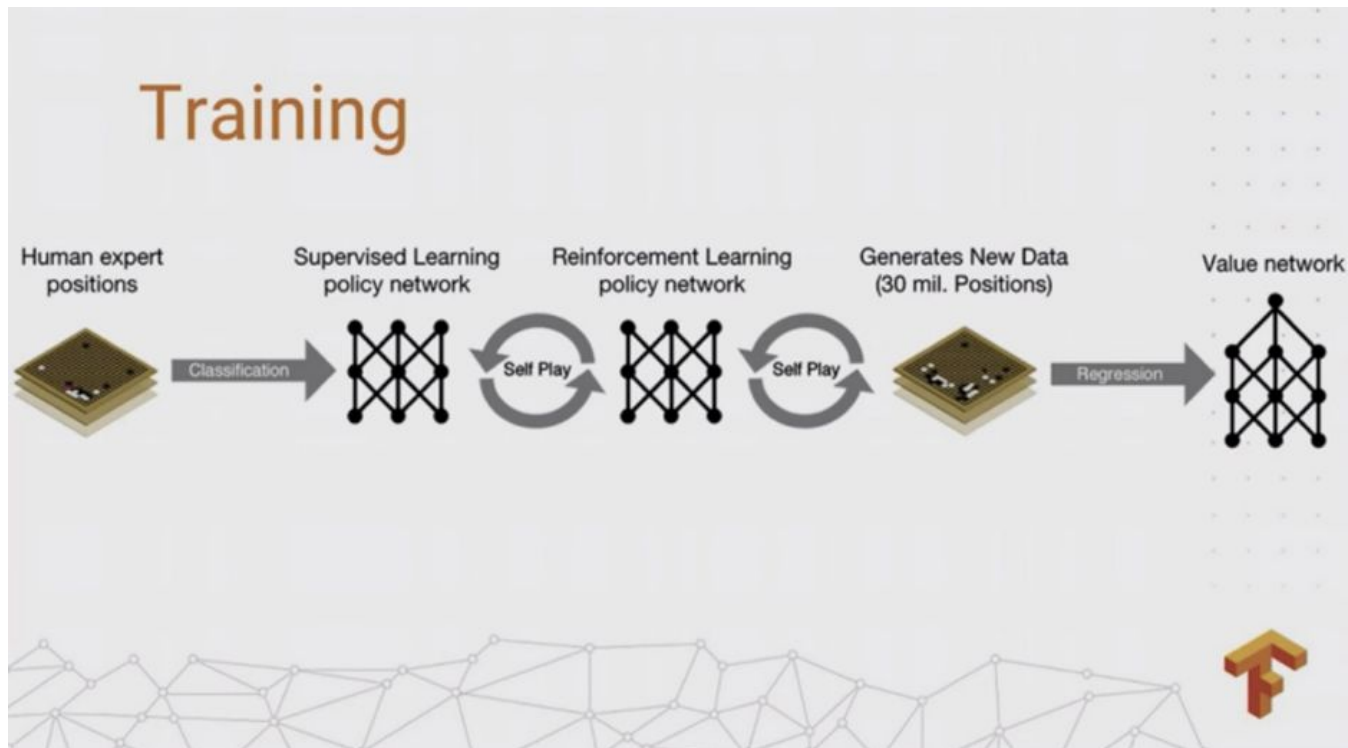
Cooling Data Center Optimizing Agriculture yields



Self-Driving Cars



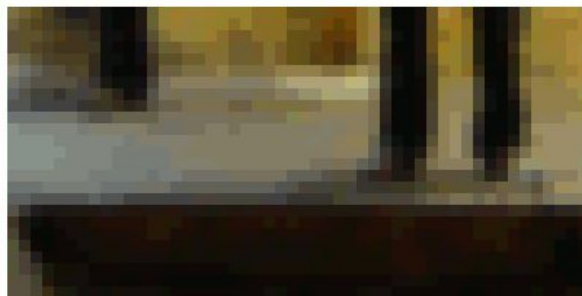
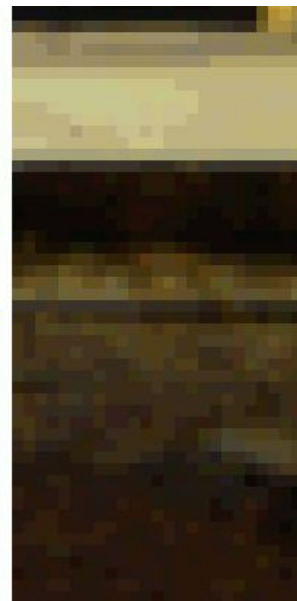
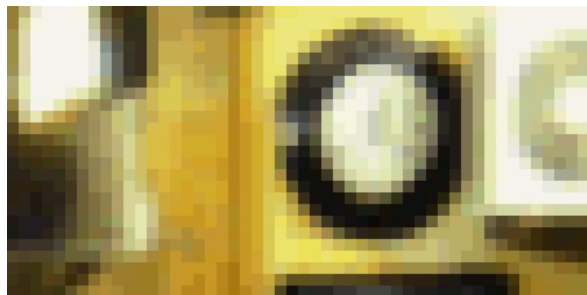
Alpha Go



Speech and Image Synth



Enhance! ([link](#))



Lipreading



Aside from that...

Businesses use DNN's for

Decision making on similar data

“Gloried Curve Fitting”

- Judea Pearl

([link](#))

- Decision Making
 - Diagnostics
 - Identifying anomalies
 - Classification: fraud, CRM, Tagging, Text, Responding to events, Marketing, Recommendations, Spam
 - Regression: Valuation, Timing, Correlations, Requirements
- Forecasting trends/events
- System / Process / Parameter optimization

A Brief History of Deep Learning

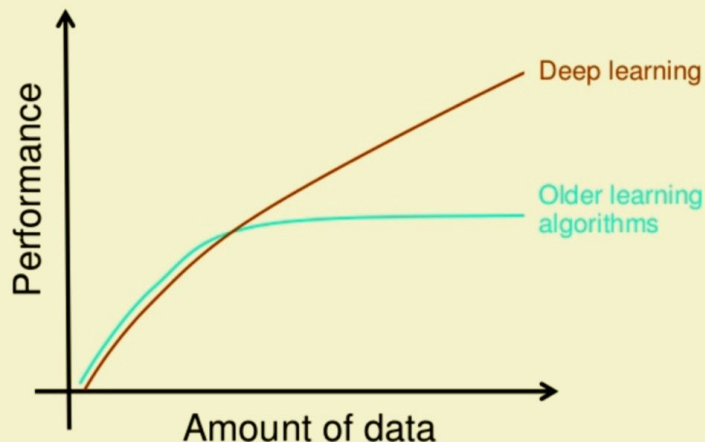
- 1943: Neural networks
- 1957: Perceptron Algorithm
- 1984: Backpropagation, Boltzman Machines, RNN
- 1989: Convolutional Neural Networks
- 1997: Long Short Term Memory Networks
- 2006: Deep Learning, Deep Belief Networks
- 2009: Imagnet
- 2012: Alexnet wins Imagenet, Dropout
- 2014: Generative Adversarial Neural Networks
- 2016: Alpha GO
- 2017: Alpha Zero, Attention Networks
- 2018: BERT

A Brief History of Deep Learning Tools

- 1960: Mark 1 Perceptron
- 2002: Torch
- 2007: CUDA
- 2008: Theano
- 2014: Caffe
- 2015: Tensorflow 0.1
- 2015: Keras
- 2017: Pytorch 0.1
- 2017: Tensorflow 1.0
- 2017: Pytorch 1.0
- 2019: Tensorflow 2.0

But Why? (oversimplification)

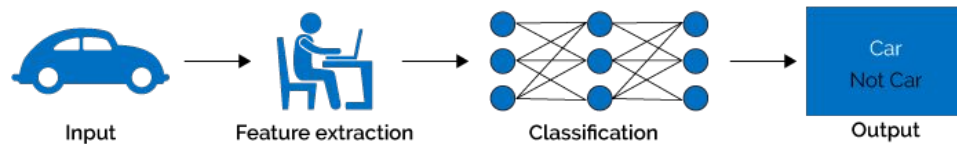
Why deep learning



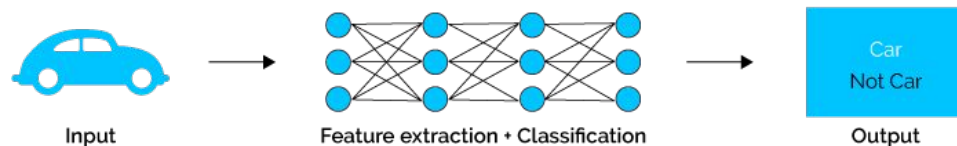
How do data science techniques scale with amount of data?

*think no free lunch

Machine Learning



Deep Learning



Deep Learning	Traditional ML
Input Model Output	Input Feature Extraction Features Model Output

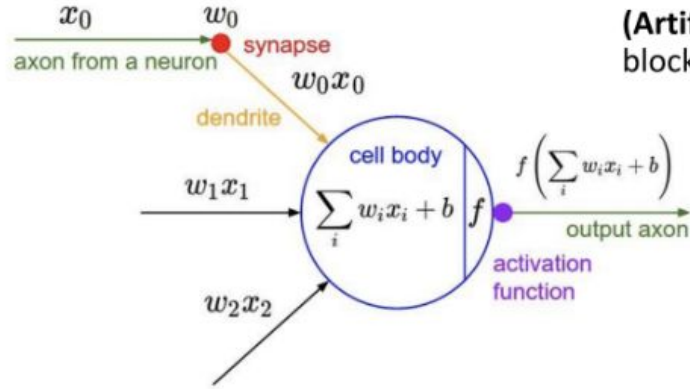
So why DL?

1. It works for complex unstructured problems like images, videos, audio, time series and games
2. Performance with more data
3. Auto feature, no/little domain expertise needed

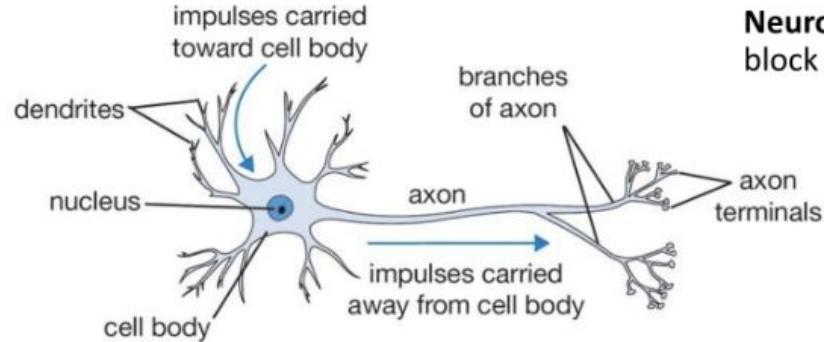


A review of the Perceptron





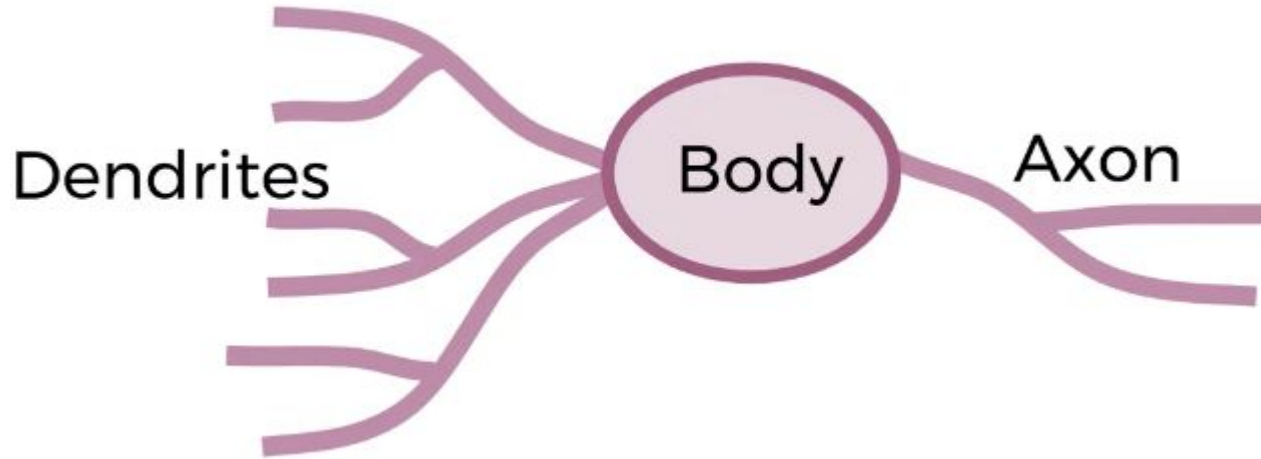
(Artificial) Neuron: computational building block for the “neural network”



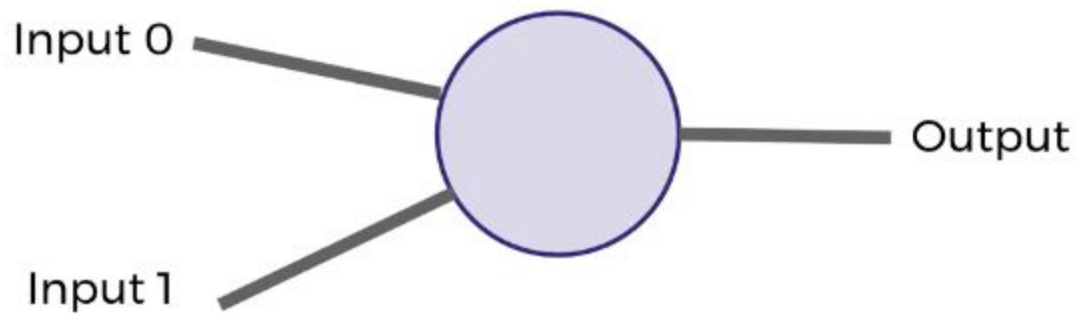
Neuron: computational building block for the brain

The Neuron

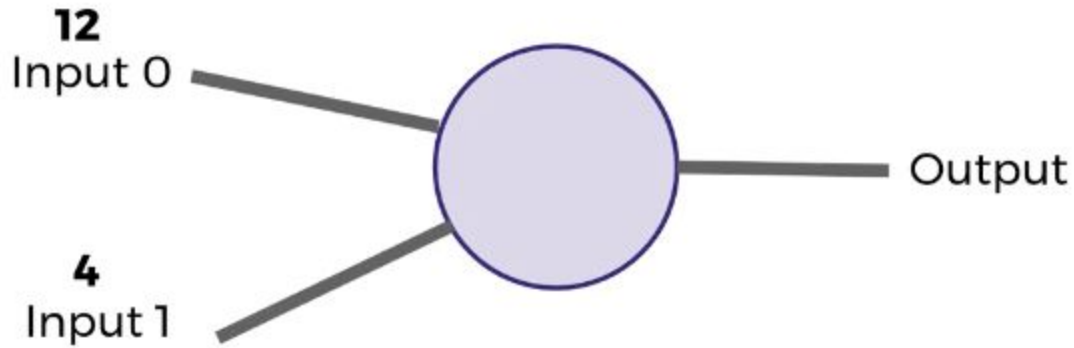
Biological Neuron



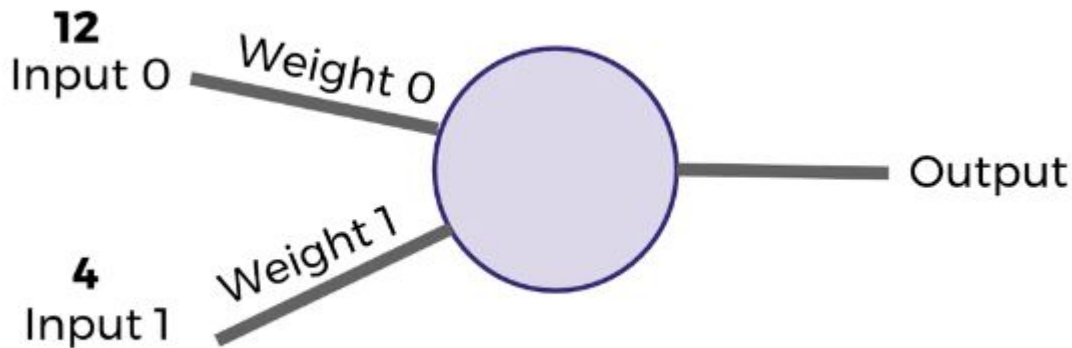
Conceptually



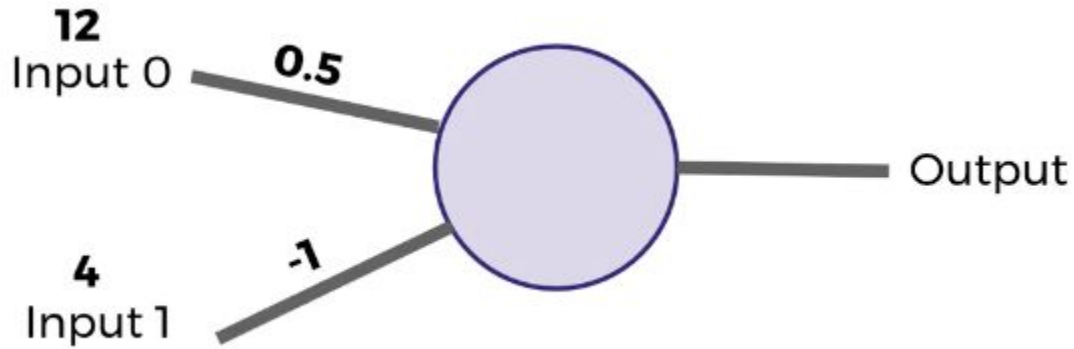
The dendrites receive an input



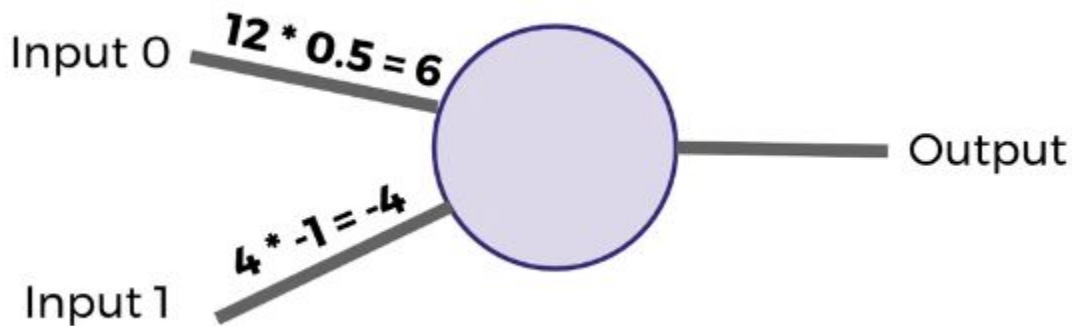
They inputs are subject to a weight



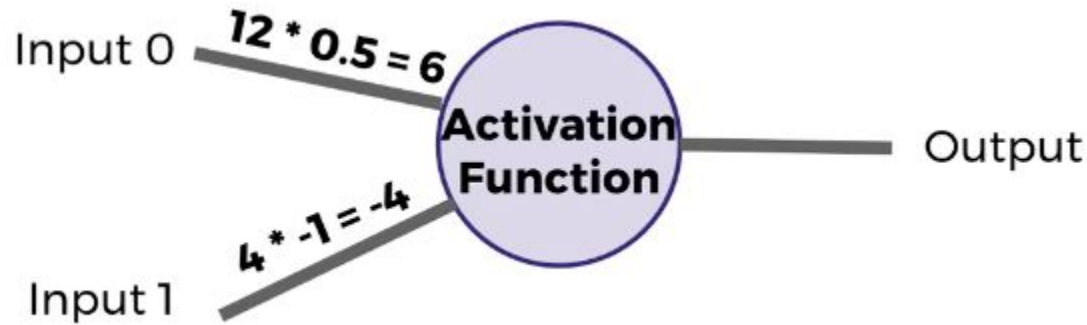
In the weights are initially give some value



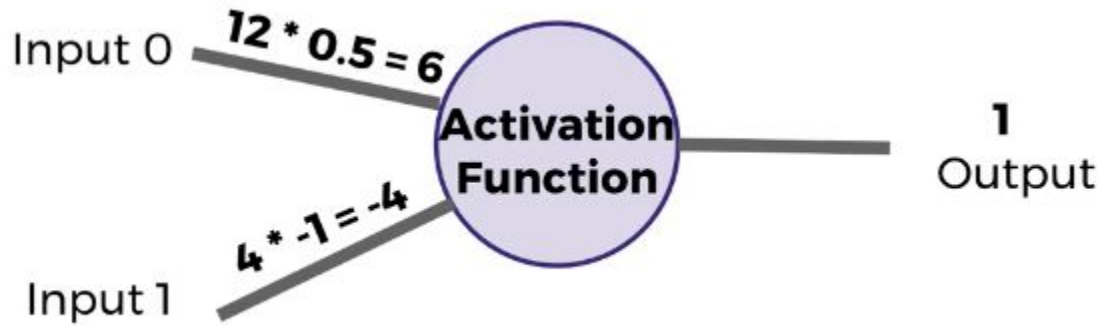
The inputs are multiplied by the weights



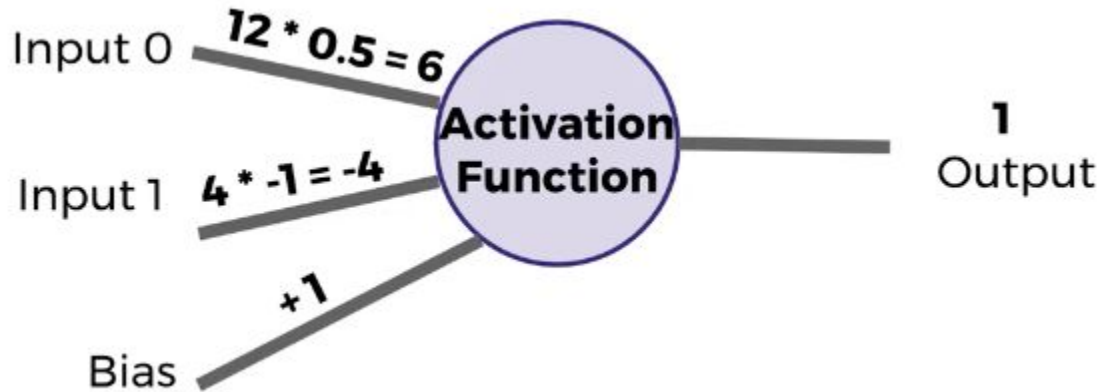
An Activation Function defines how to combine the inputs*weights and what will be passed on



Simple AF: If positive, return 1, otherwise return 0



Add bias, so even if weights are both 0, we still can obtain a non-zero output



Simple Formulas, Sum of products and binary activation

$$z = b + \sum_i x_i w_i$$

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Perceptron Learning Algorithm

1. If the output is correct, leave weights alone
2. If the output unit incorrectly outputs a 0, add the input vector to the weight vector
3. If the output unit incorrectly outputs a 1, subtract the input vector from the weight vector

```
def learn(self, inputs, outputs):  
    if self.forward_pass(inputs) == outputs:  
        pass  
    elif self.forward_pass(inputs) == 0:  
        self.w += inputs  
    elif self.forward_pass(inputs) == 1:  
        self.w -= inputs
```

In code ([link](#))

```
import numpy as np

class Perceptron(object):

    def __init__(self, input_size):
        self.w = np.random.rand(input_size)

    def forward_pass(self, inputs):
        return self.activation_function(np.multiply(inputs, self.w))

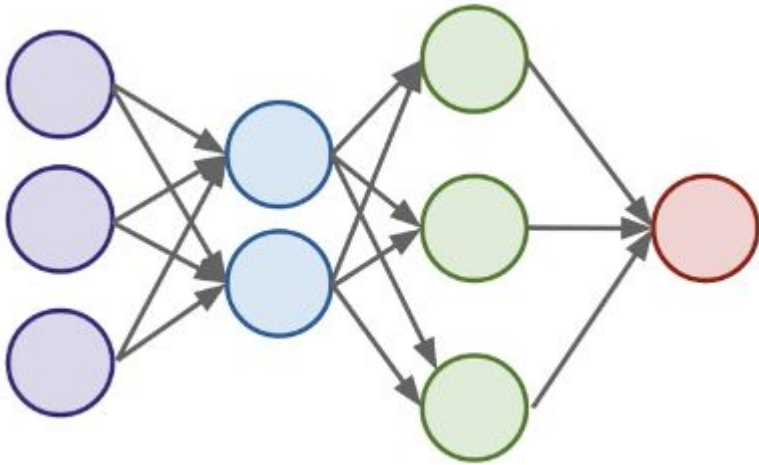
    def activation_function(self, inp):
        return 1 if np.sum(inp) > 0 else 0

    def learn(self, inputs, outputs):
        if self.forward_pass(inputs) == outputs:
            pass
        elif self.forward_pass(inputs) == 0:
            self.w += inputs
        elif self.forward_pass(inputs) == 1:
            self.w -= inputs
```

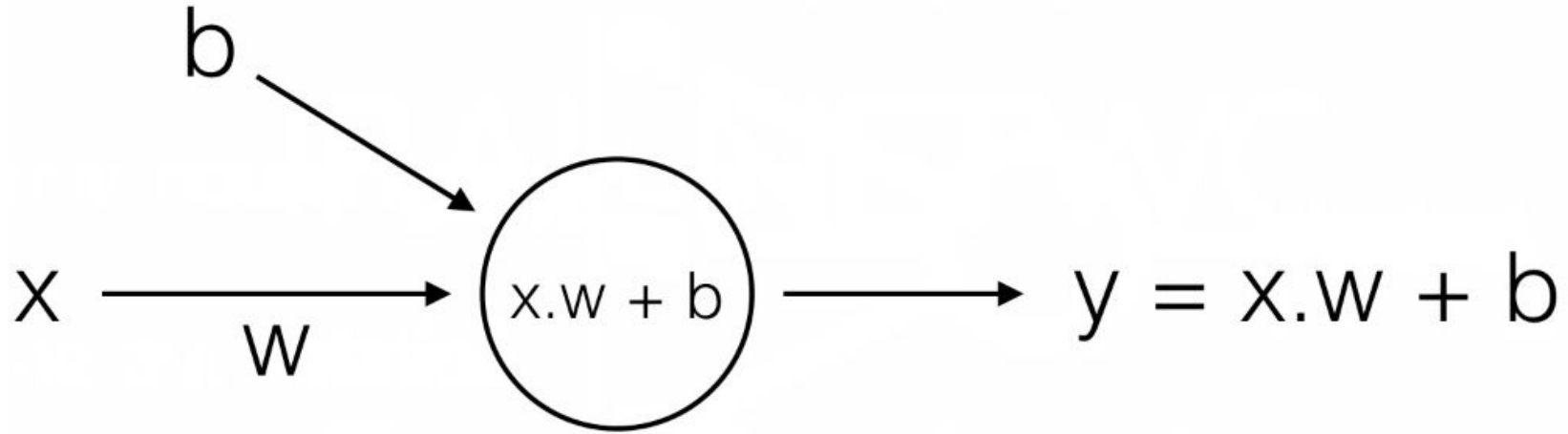
Neural Networks

- Topology
- Activation Functions
- Loss
- Optimizer
- Back-propagation

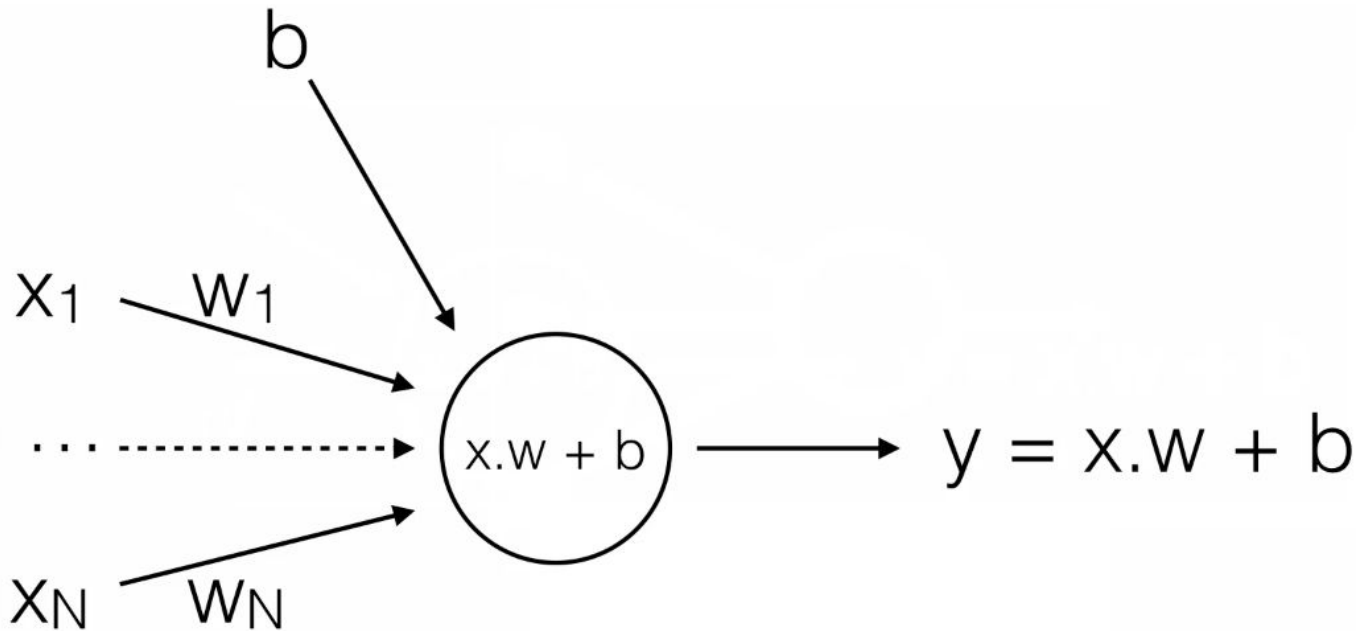
Input – Hidden Layers – Output



Neural Network with one layer and one unit (AKA linear regression)

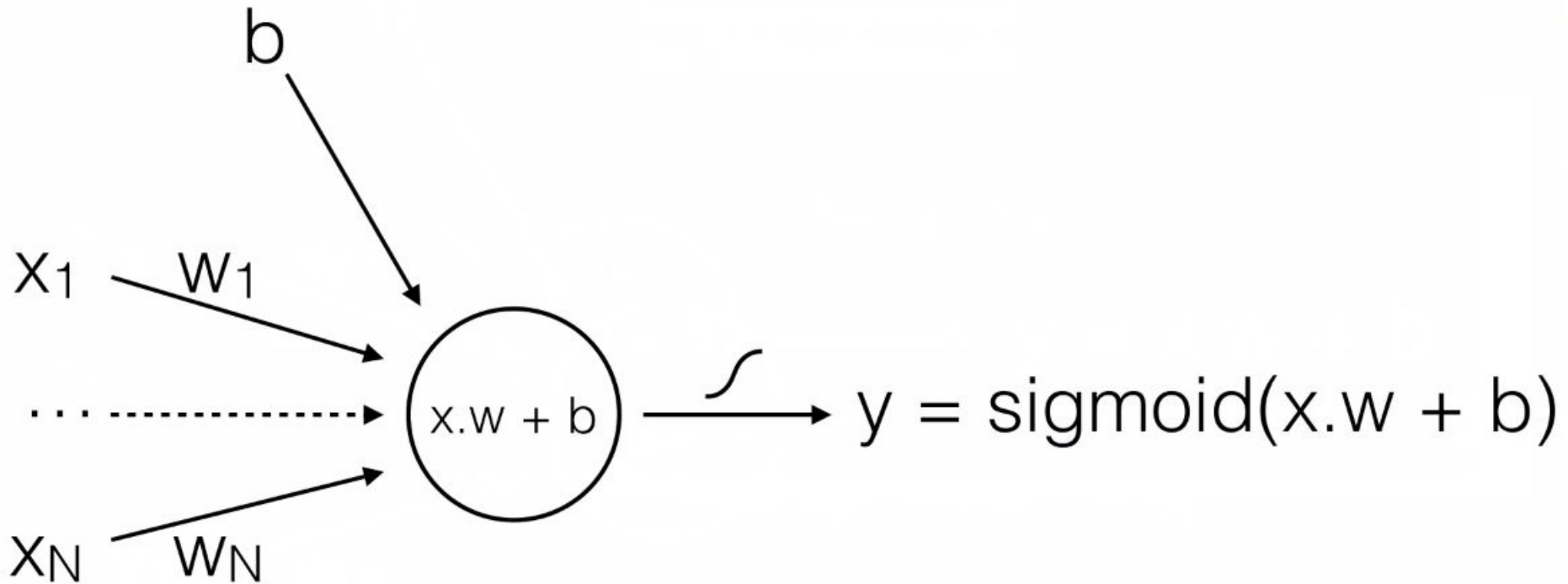


Multivariate Linear Regression (Nothing New Here?)

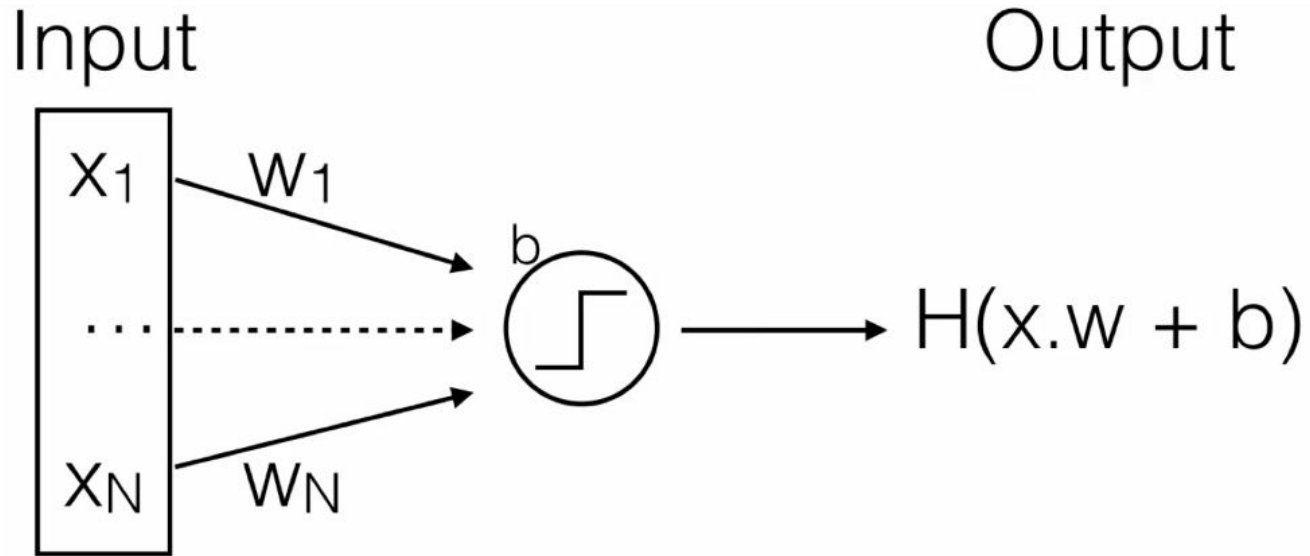


What About Logistic Regression?

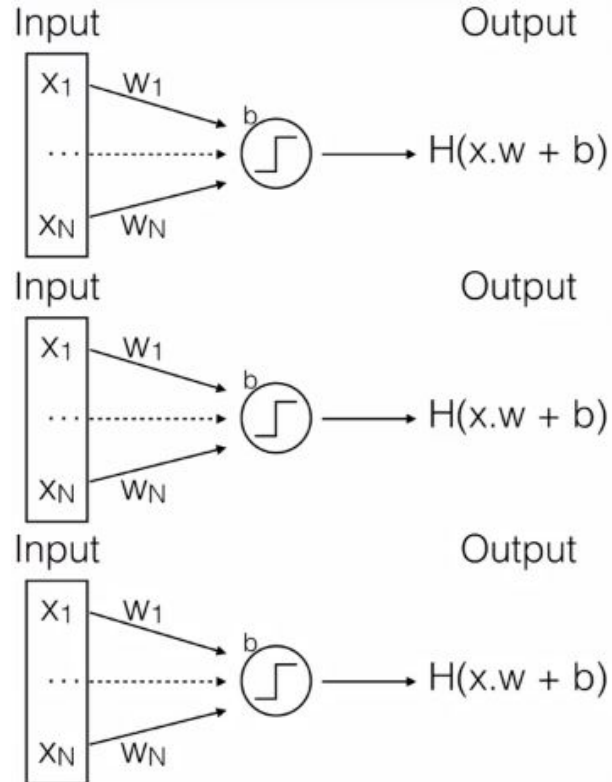
Just add a sigmoid and now we have Logistic Regression (binary output)



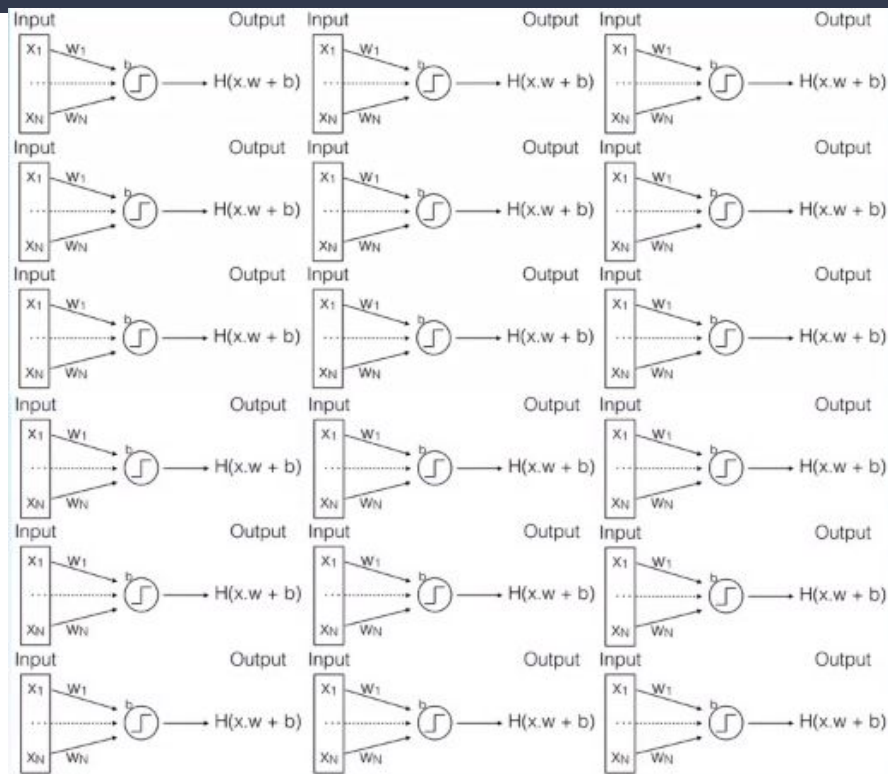
One Unit



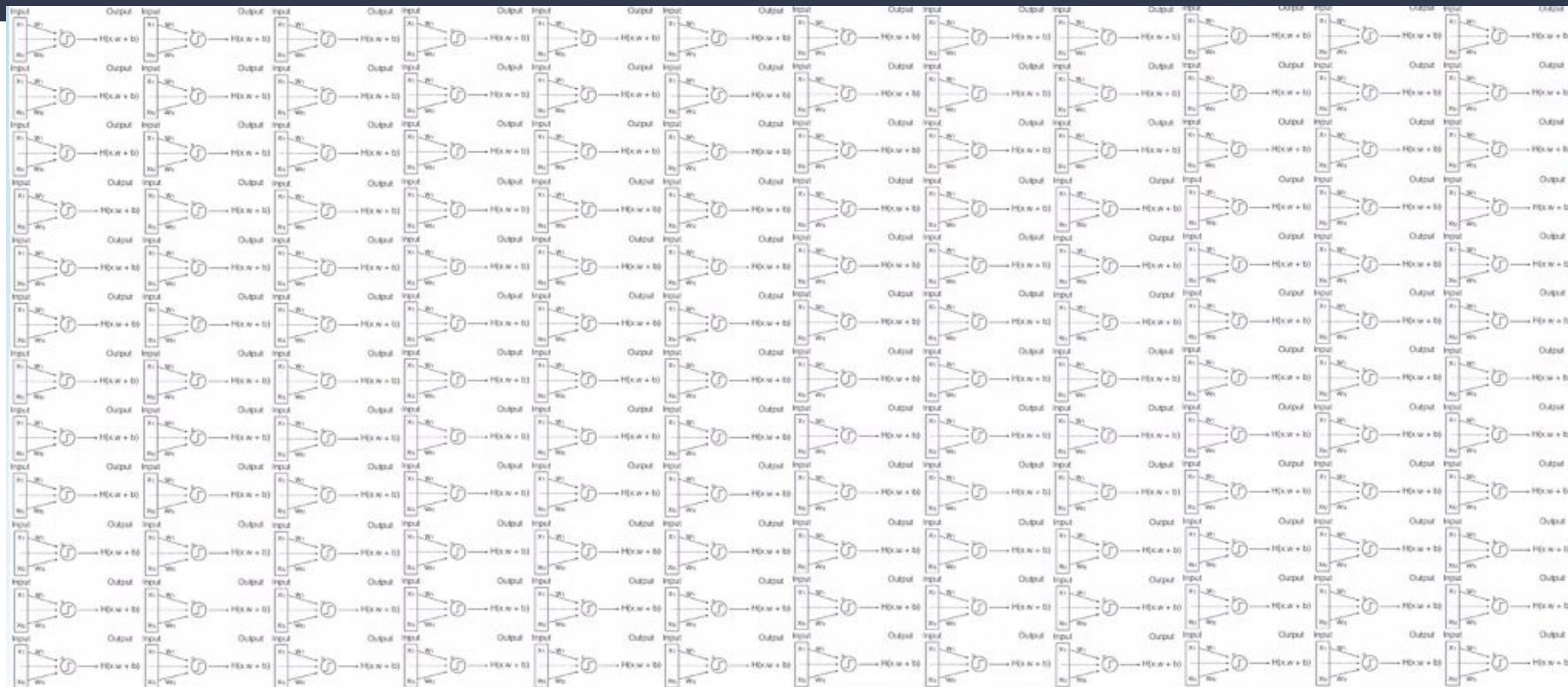
One Layer



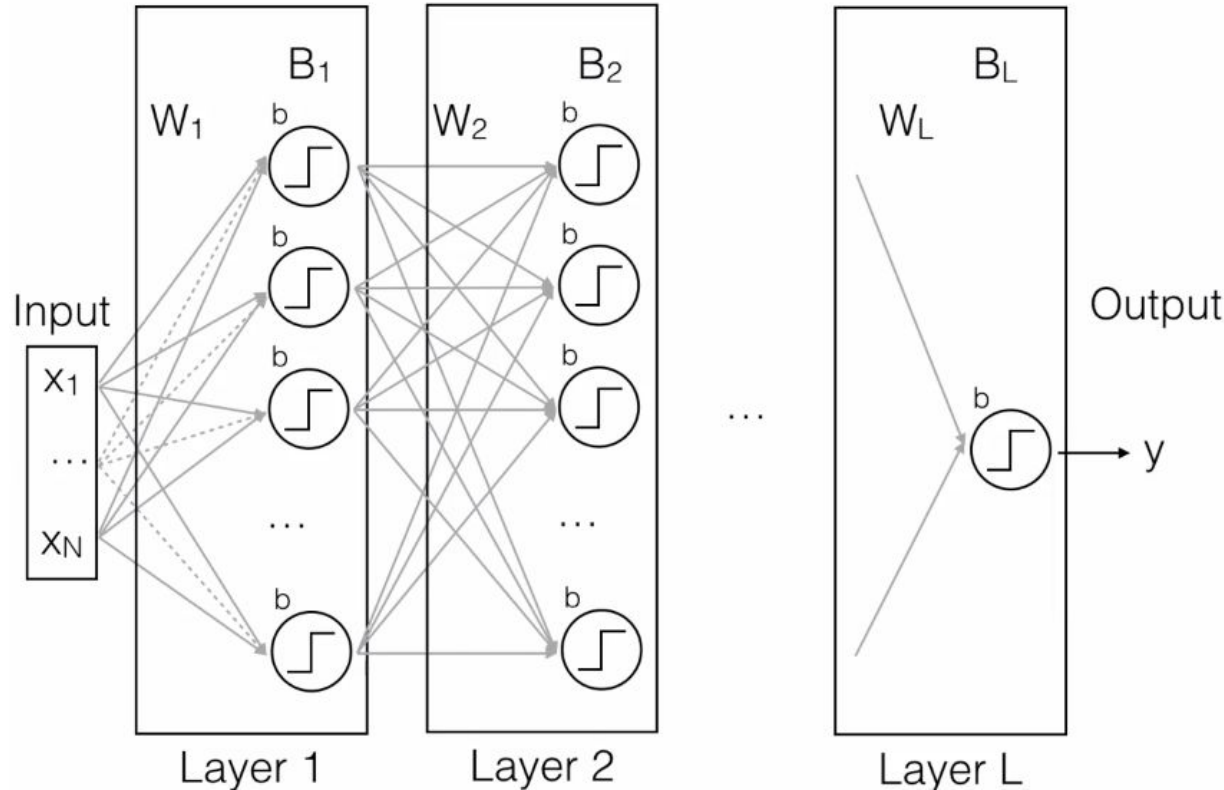
Multiple Layer



And so on...



Each node in each layer is fully connected












Just one big $f(x)=y$
Takes inputs from a
feature space and maps
onto a target space.
Same as all ML models

1. Multiply previous layer output by weights
2. Deform them with a nonlinearity
3. Pass transformed values along to next layer
4. And so on

Activation Functions

Defines the output of the node or neuron. This output is used as input for the next node

It maps the resulting values into the desired range using a simple function

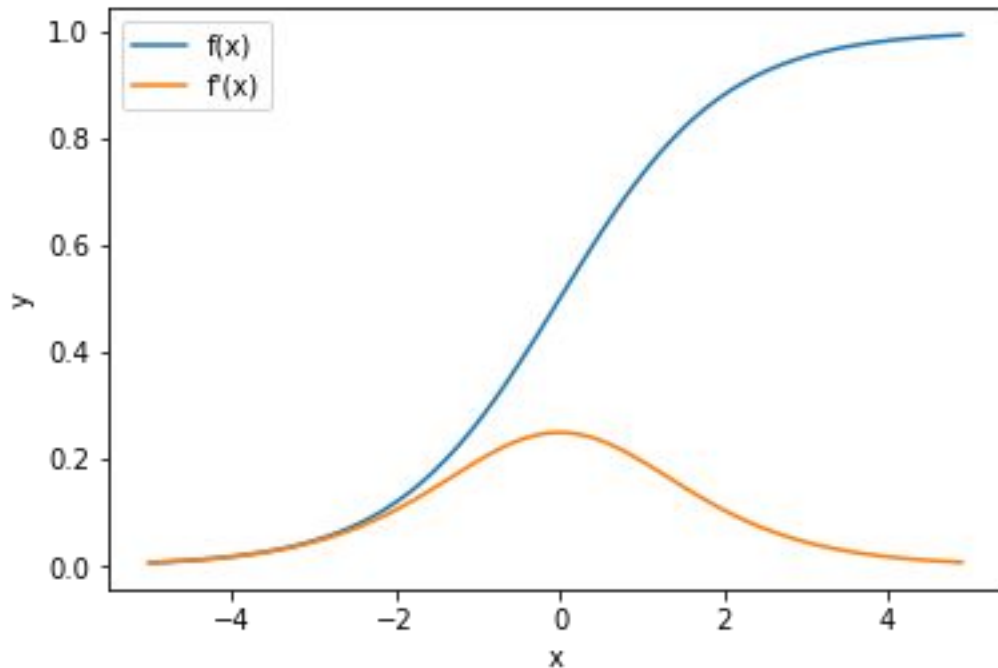
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) ^[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) ^[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Which one should we choose?

Most popular

1. Sigmoid or logistic
2. Tanh - hyperbolic tangent
3. ReLu - Rectified Linear Unit

Sigmoid

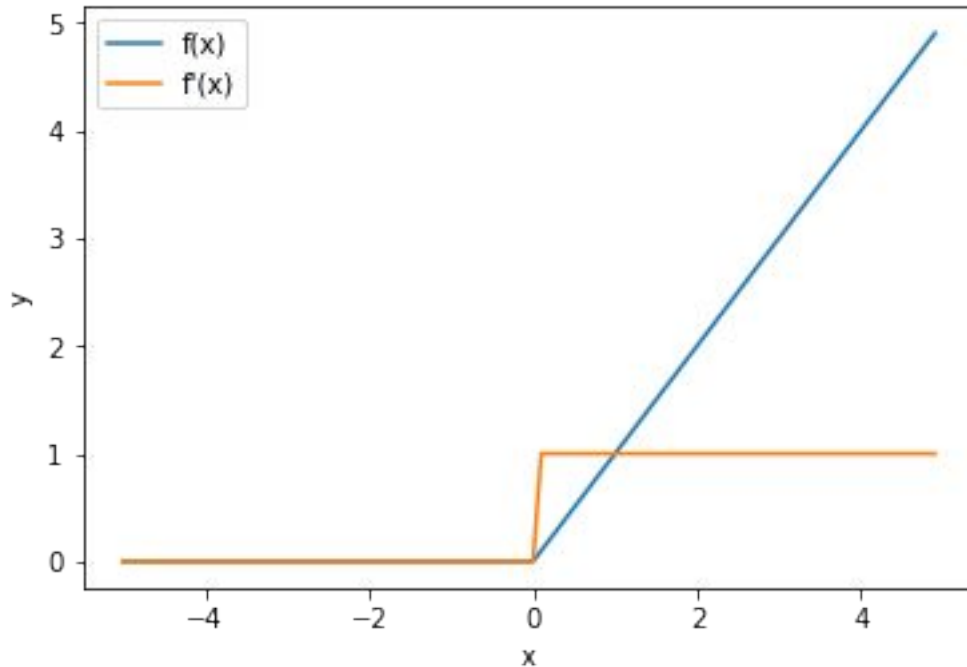


It looks like the step function, but allows us to take into account small changes.

The original NN cost function

1. Vanishing Gradient, (degrading derivative)
derivatives are too small
2. Requires initialization
3. Doesn't work in large networks
4. Saturates and kills gradients
5. Slow convergence

ReLu



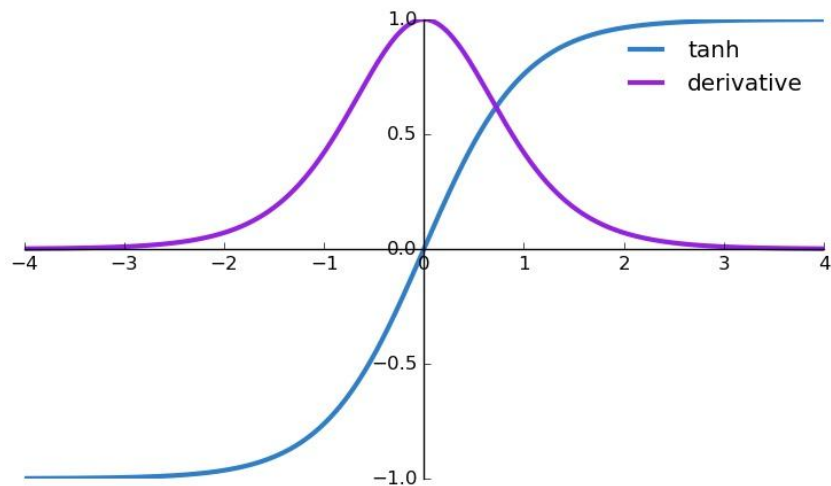
Rectified Linear Unit

$\text{Max}(0, x)$

Most popular AF

1. Very simple and fast to calculate while staying non-linear
2. Derivative is fast to calculate and doesn't degrade
3. Can sometimes "kill" neurons when always 0 derivatives

Tanh



$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

- Similar in most ways to the sigmoid
- But is zero-centered, so easier to optimize
- But still suffers from ...

Loss/Cost

- How far off we are from the expected value, How wrong is my neural network (or my model in general)?
- What are some examples of loss functions?

Types of Loss

Mean Square Error - Regression problems:

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$$

Categorical Cross Entropy - Classification problems:

$$\mathcal{L} = -\frac{1}{n} \sum_{i=1}^n [y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$



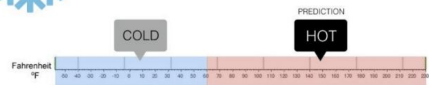
Regression

What is the temperature going to be tomorrow?



Classification

Will it be Cold or Hot tomorrow?



- Loss function quantifies gap between prediction and ground truth
- **For regression:**
 - Mean Squared Error (MSE)
- **For classification:**
 - Cross Entropy Loss

Mean Squared Error

$$MSE = \frac{1}{N} \sum (t_i - s_i)^2$$

Prediction s_i

Ground Truth t_i

Cross Entropy Loss

$$CE = - \sum_i^C t_i \log(s_i)$$

Classes C

Prediction s_i

Ground Truth $\{0,1\}$ t_i

Loss Functions

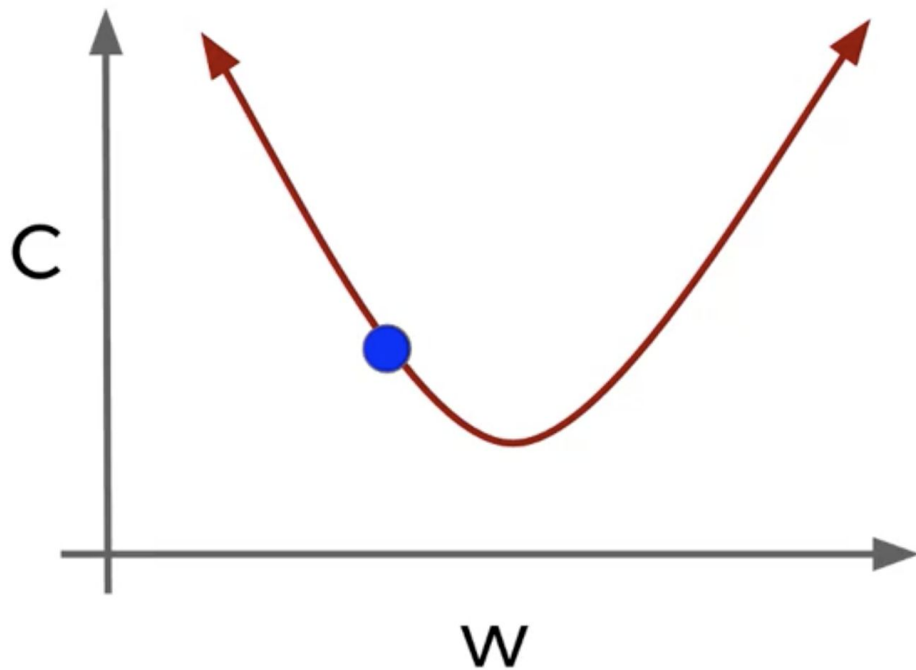
Optimizer – Backpropagation



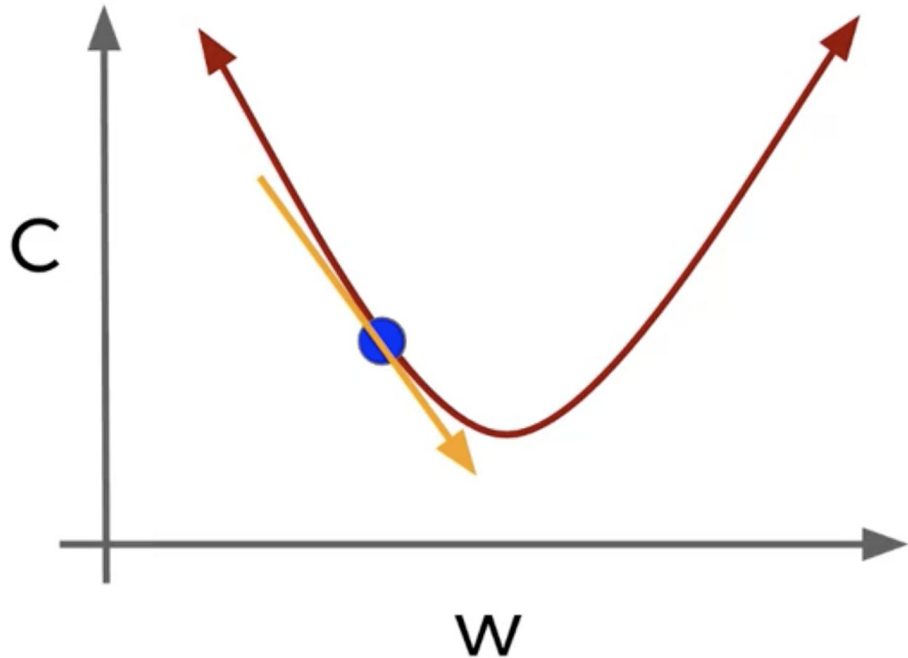
Stochastic Gradient Descent

Stochastic gradient descent (often shortened to **SGD**), also known as **incremental** gradient descent, is an iterative method for optimizing a differentiable objective function, a stochastic approximation of gradient descent optimization

GD in 1 dimension



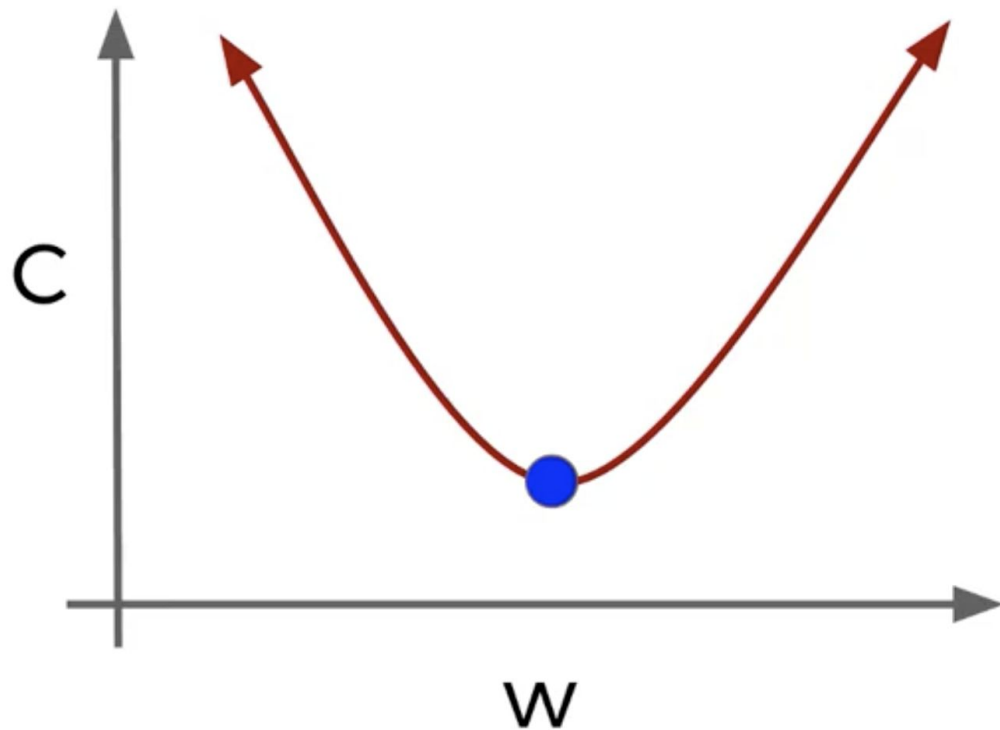
Slope of Tangent line and 'descend' along that line



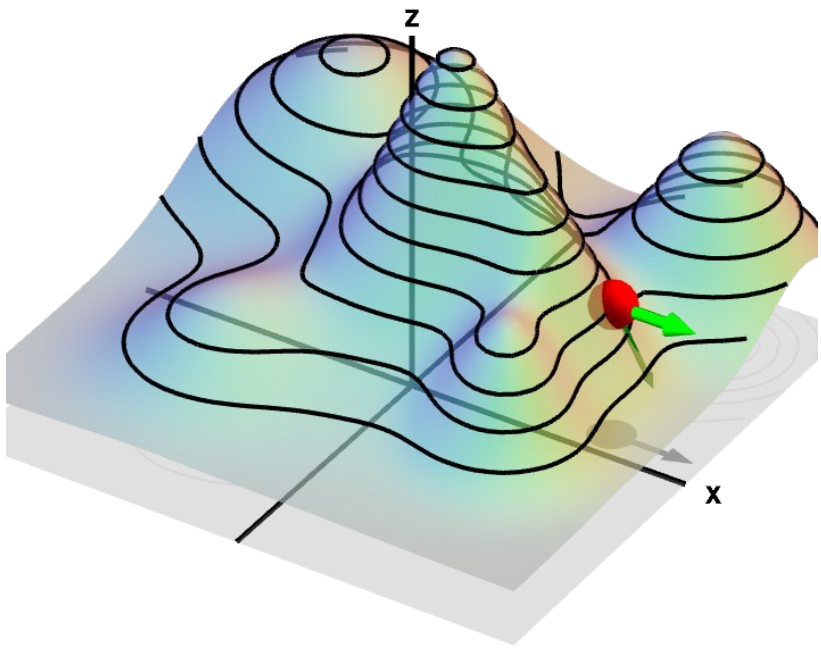
New weight is equal to the old weight subtracted by the slope of the tangent line

$$w = w - \delta f / \delta w$$

Until we can descent no longer



Partial Derivatives



*assume z is the error dimension

The shortest path down is not necessarily along the x or y dimensions, but rather a combination of both - partial derivative

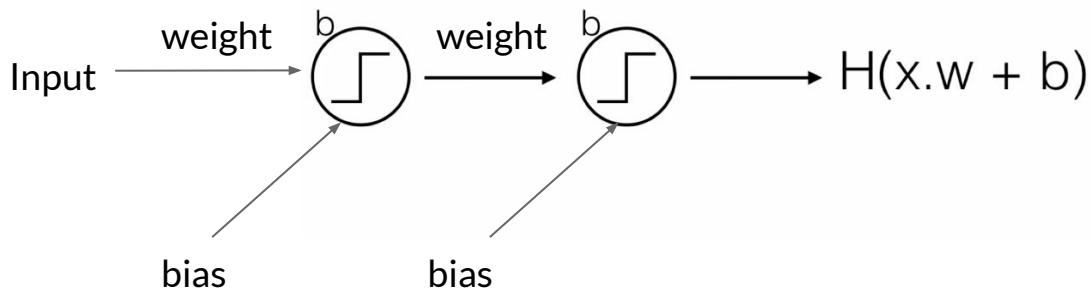
Chain Rule

The derivative of the entire neural network with respect to the input

Is equal to..

The derivative of the entire neural net wrt the previous node and the derivative of the previous node wrt the input

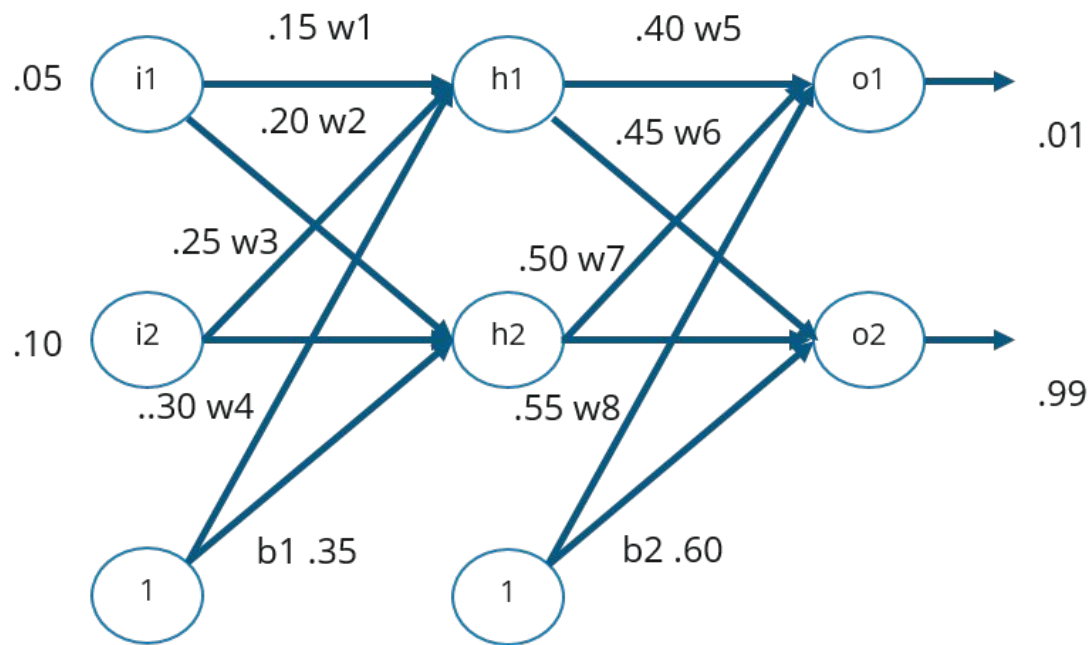
Output



$$f = f(g) ; g = g(x)$$

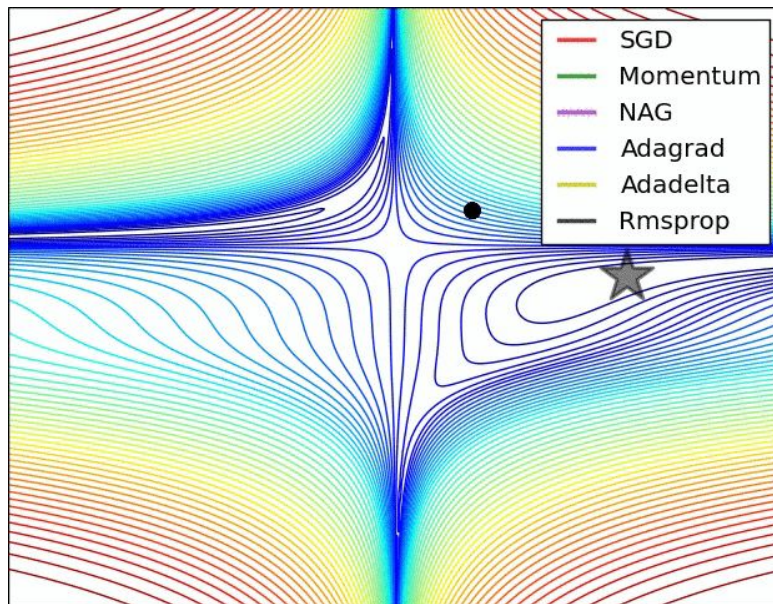
$$\frac{df}{dx} = \frac{df}{dg} \frac{dg}{dx}$$

How it works



- Forward pass gives you a prediction - and an error
- Backward pass gives you a gradient

Different Flavors of SGD ([link](#))



Tensorflow Playground

<https://playground.tensorflow.org>

Simple Example of NN

<http://bit.ly/2TAybmD>

More Advanced Example <http://bit.ly/2XWKic1>

Datacamp

- [Intro to python](#)
- [Intermediate python for data science](#)
- [Object Oriented Programming](#)
- [Deep Learning in Python \(keras\)](#)
- [Software Engineering for Data Scientists](#)