Deep Learning Workshop

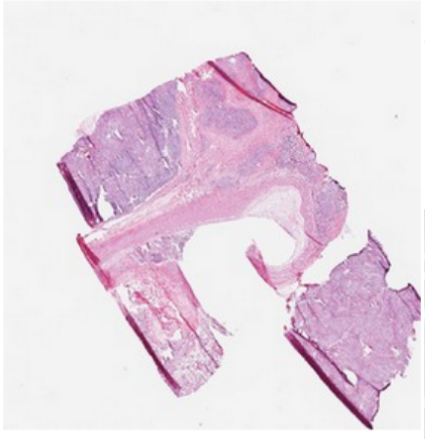
"...what we want is a machine
that can learn from experience."

Alan Turing, 1947

Xenofon Karagiannis

MEGAN MOLTENI SCIENCE 09.17.18 11:00 AM

GOOGLE AI TOOL IDENTIFIES A TUMOR'S MUTATIONS FROM AN IMAGE

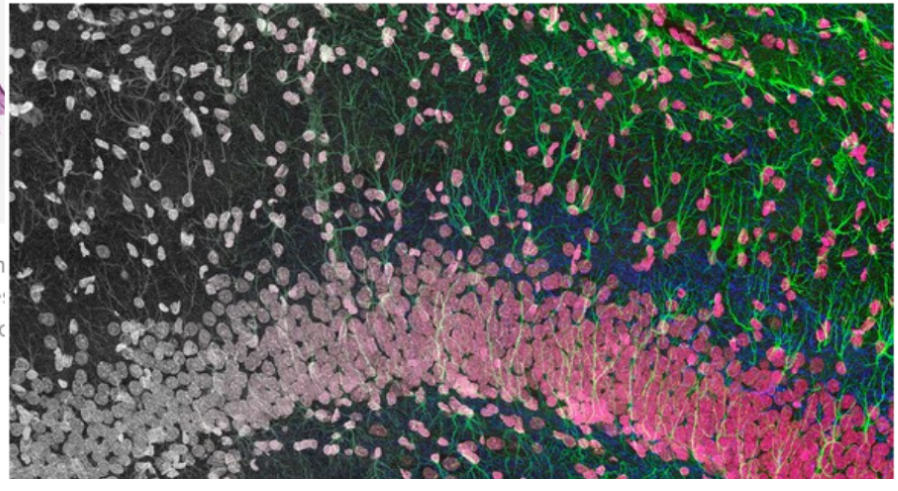


The image shows the process by which an
a map that tells apart two lung cancer type
cell carcinoma in red, lung squamous cell c

NYU SCHOOL OF MEDICINE

ROBBIE GONZALEZ SCIENCE 04.12.18 12:00 PM

AI LEARNS A NEW TRICK: MEASURING BRAIN CELLS



A.I. Shows Promise Assisting Physicians

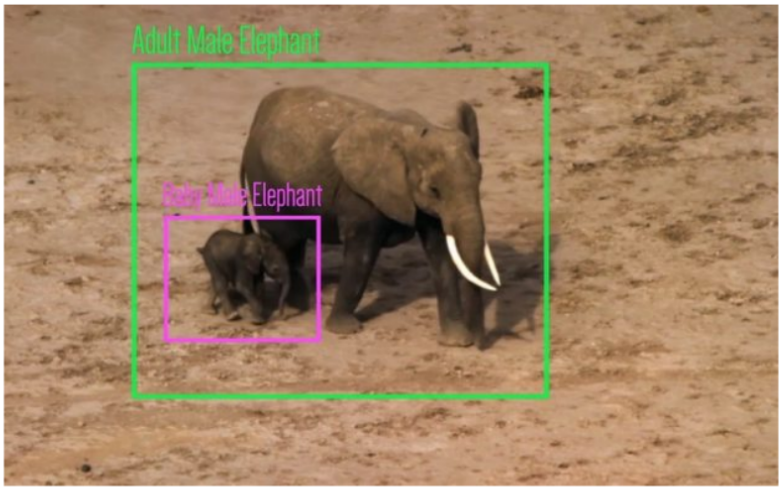
ers to recognize illnesses on magnetic resonance images of a
Beijing last year. The human doctors lost.

A.I. Is Helping Scientists Predict When and Where the Next Big Earthquake Will Be

Paul Allen enlists machine-learning tools for monitoring wildlife and ecosystems

BY ALAN BOYLE on September 26, 2018 at 10:54 am

Post a Comment | Email



Machine-learning technology can contribute to image recognition programs that could identify elephants in aerial imagery on their own. (Vulcan Photo)

The Google Pixel 3 Review: Phone's Smarts Shine Through Its A.I.-Driven Camera

Hardware innovations? Nope. Instead, Google is emphasizi software improvements — particularly for images — with i newest Pixel smartphones.

By Brian X. Chen

Oct. 15, 2018

CADE METZ BUSINESS 05.22.17 03:12 PM

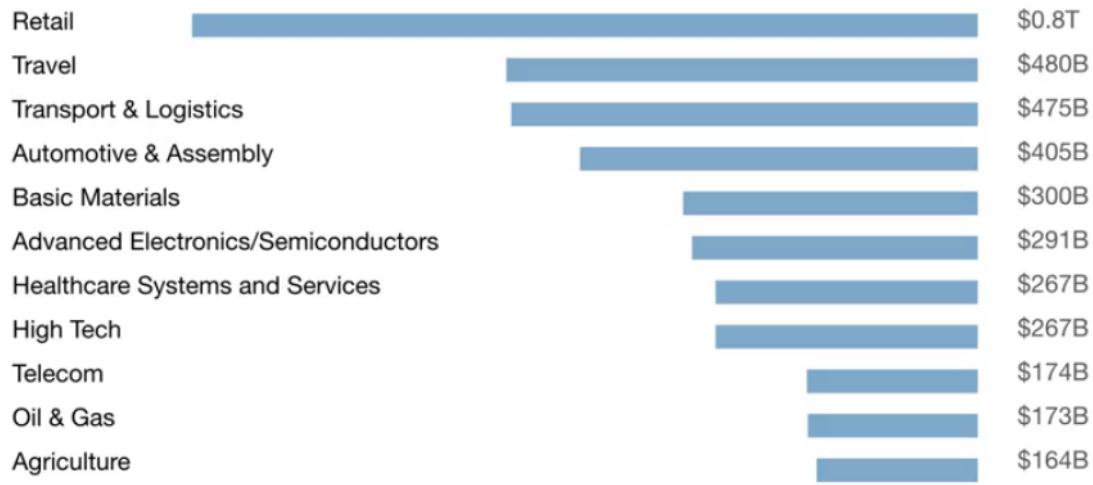
ALPHAGO IS BACK TO BATTLE MERE HUMANS—AND IT'S SMARTER THAN EVER

“AI is the new electricity”

Andrew NG

AI value creation by 2030

$13 trillion



[Source: McKinsey Global Institute.]

Buzzword Bingo!

Data Science	Cloud Computing	Machine Learning
Artificial Intelligence	Big Data	IoT
Reinforcement Learning	Deep Learning	Blockchain

Machine Learning

“Field of study that gives computers the ability to learn without being explicitly programmed.”

-Arthur Samuel (1959)

An ML projects results in a piece of software that runs.

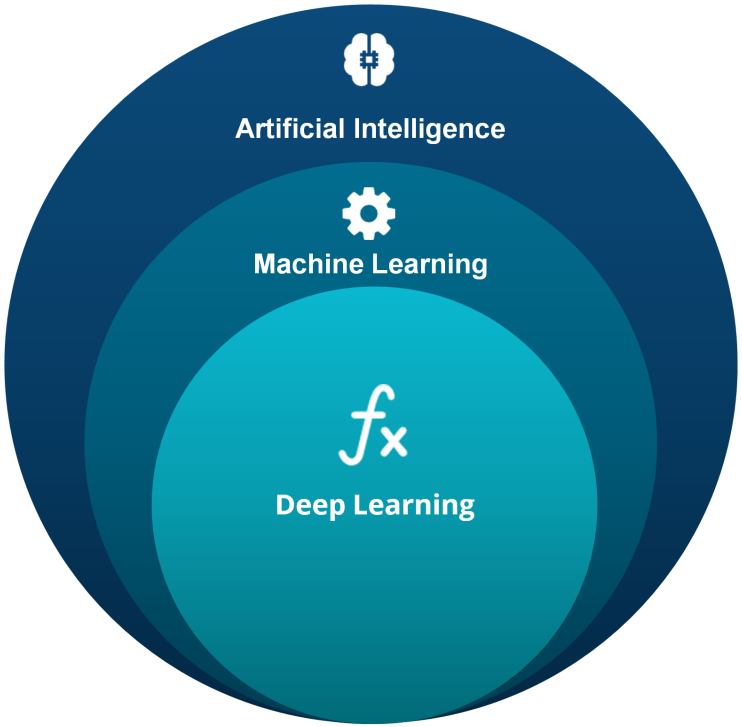
Data Science

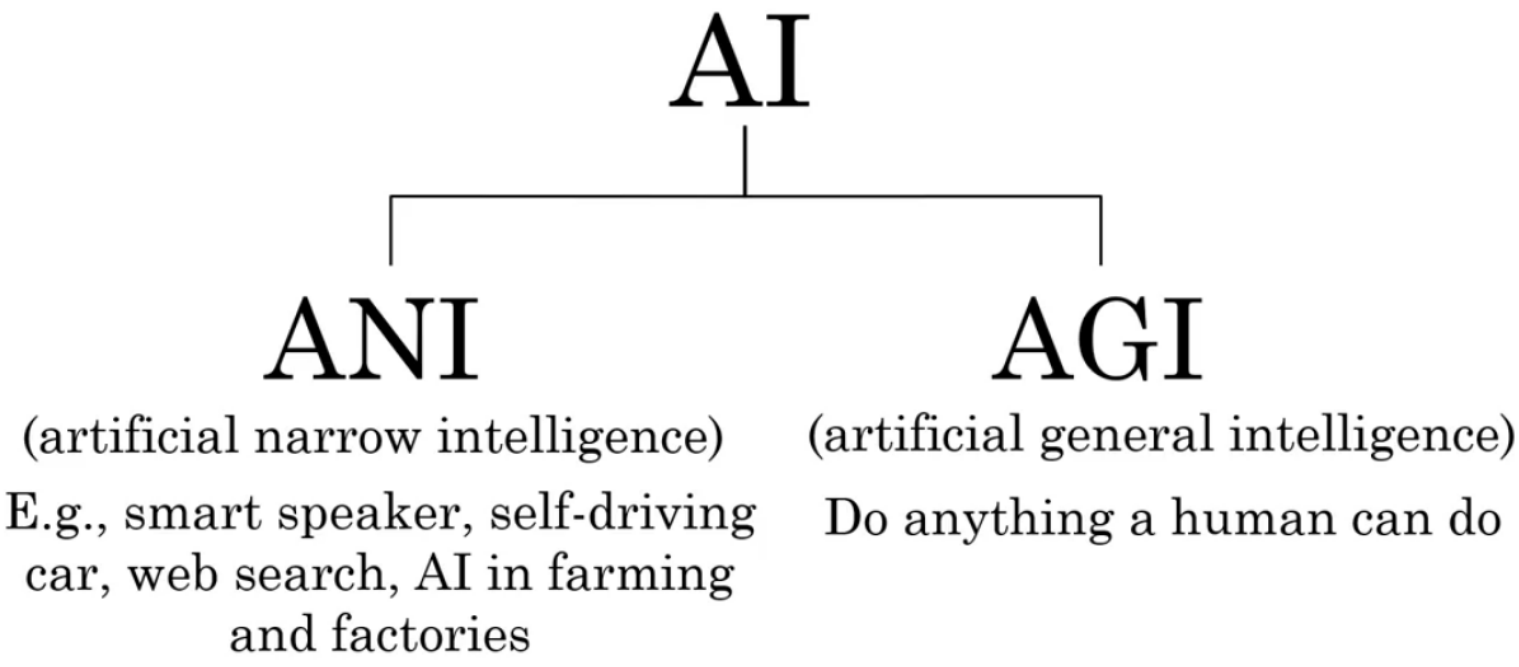
Science of extracting knowledge and insights from data.

A Data Science projects is often a presentation that summarizes conclusions for executives to take business actions or that summarizes conclusions for a product team to decide how to improve a website.

Artificial Intelligence

“the study and design of intelligent agents” where an intelligent agent is a system that perceives its environment and takes actions which maximizes its chances of success.




AI
ANI
(artificial narrow intelligence)
E.g., smart speaker, self-driving car, web search, AI in farming and factories
AGI
(artificial general intelligence)
Do anything a human can do

Color Restoration [link]

Automatic colorization and color restoration in black and white images.

(a) Colorado National Park, 1941 *(b) Textile Mill, June 1937* *(c) Berry Field, June 1909* *(d) Hamilton, 1936*

Figure 1: *Colorization results of historical black and white photographs from the early 20th century. Our model can obtain realistic colorization results whether it is a picture of landscapes (a), man-made environments (b), human portraits (c), or close-ups (d). Photos taken by Ansel Adams (a) and Lewis Hines (b,c,d), and are taken from the US National Archives (Public Domain).*

Speech Reenactment [link]

Synthesizing Obama: Learning Lip Sync from Audio.

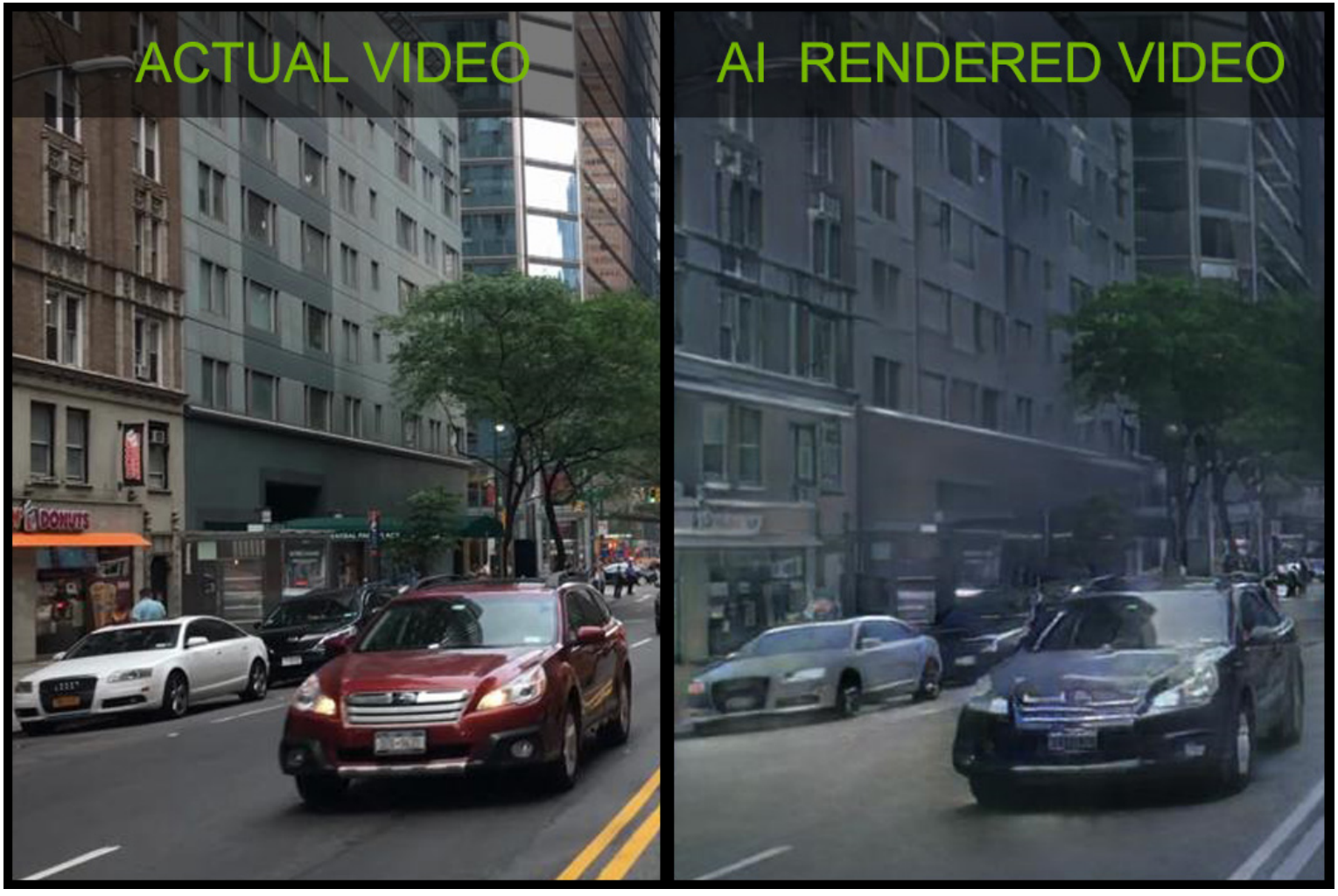
Diagnose crop diseases [link]

Artificial intelligence could help farmers diagnose crop diseases

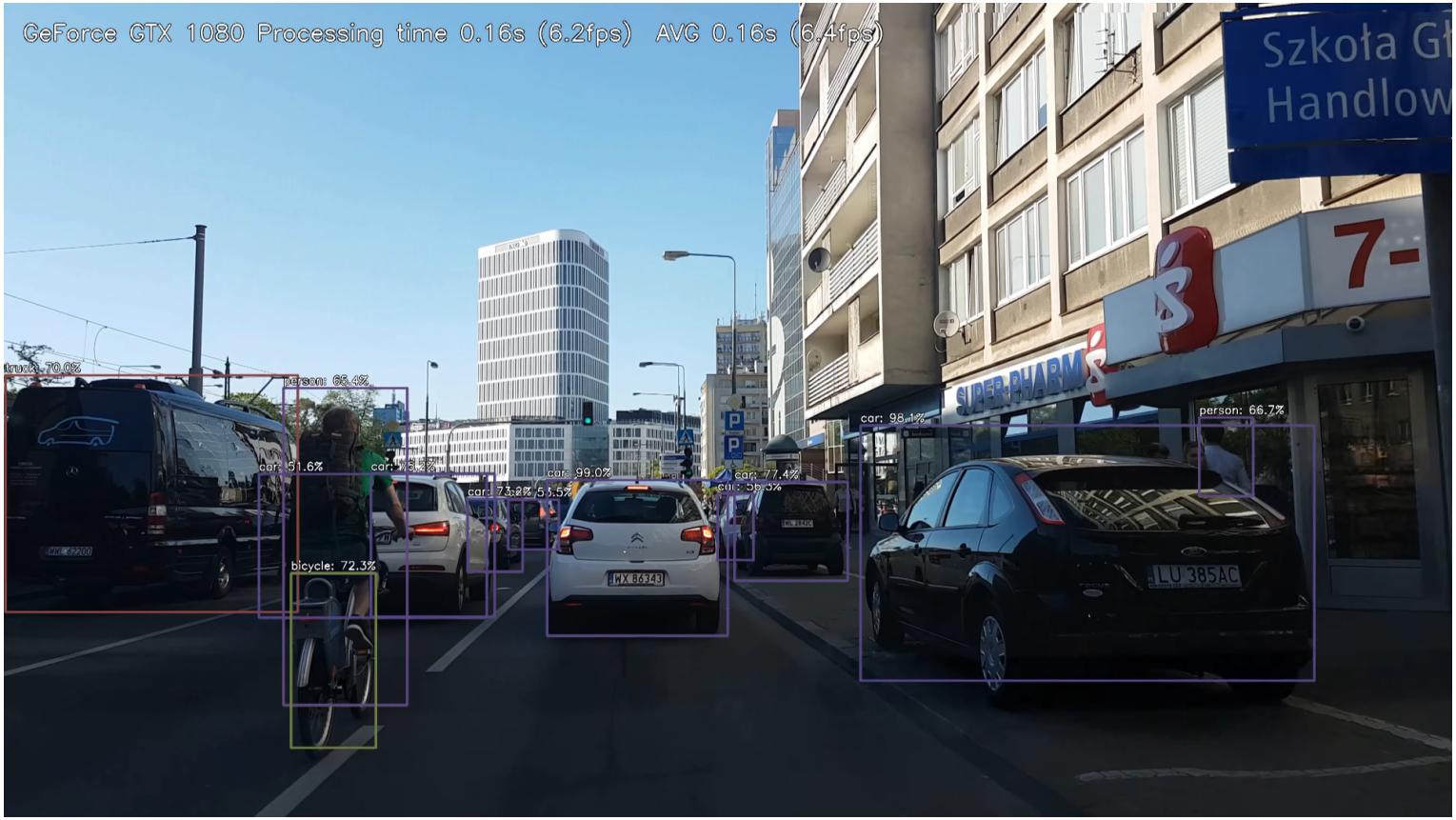
Kelsey Pryze, undergraduate researcher, captures photographs of potato leaves at Penn State's Russell E. Larson Agricultural Research Center at Rock Springs. **IMAGE: PENN STATE / EPFL**

Car Vehicle / Traffic Lights / Pedestrian detection with YOLO [link]

NVIDIA Invents AI Interactive Graphics [link]

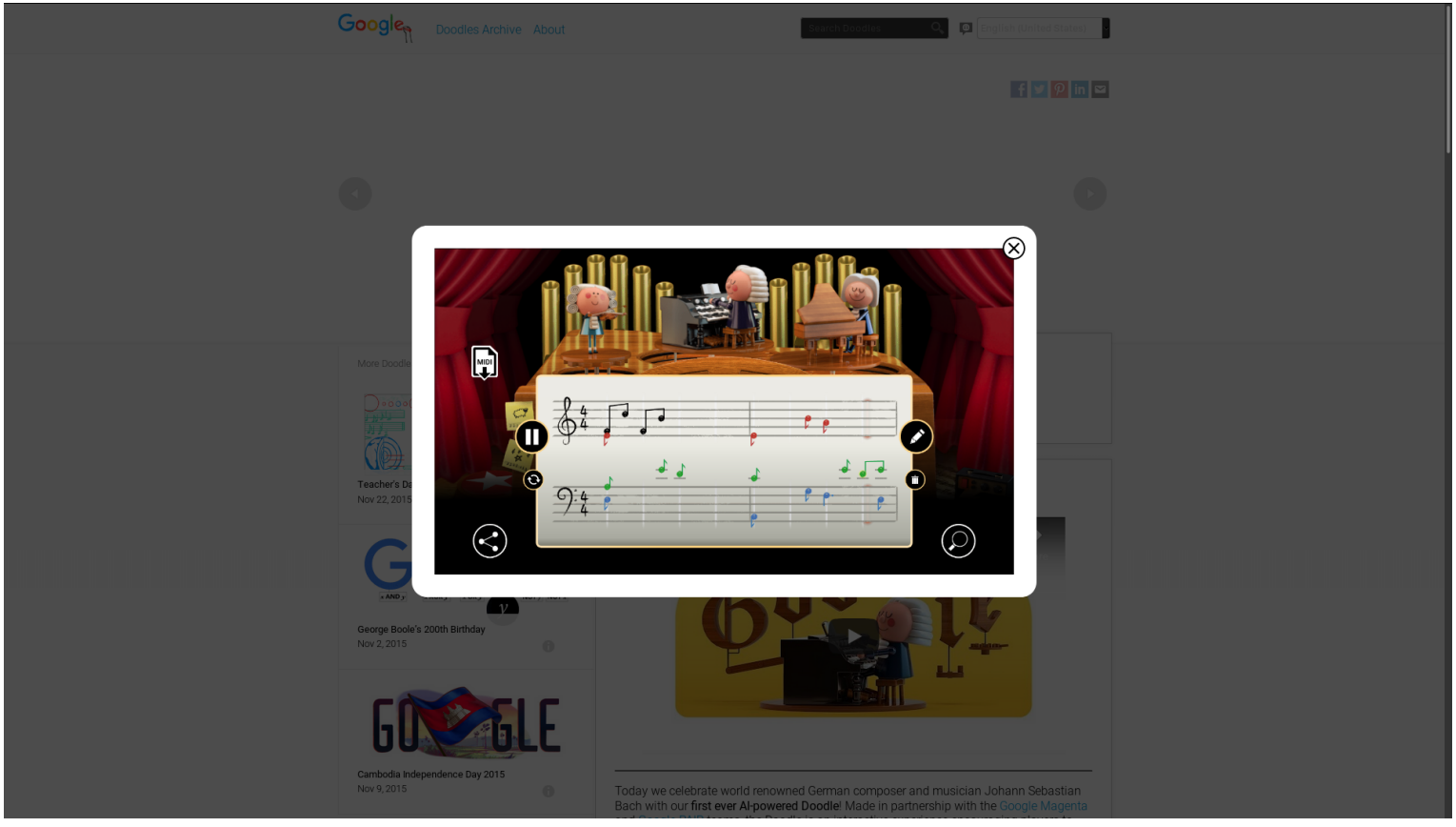


Real Time Object Detection in 4K Video - RetinaNet [link]



This person does not exist - GAN [link]

Google Doodle - Celebrating Johann Sebastian Bach [link]

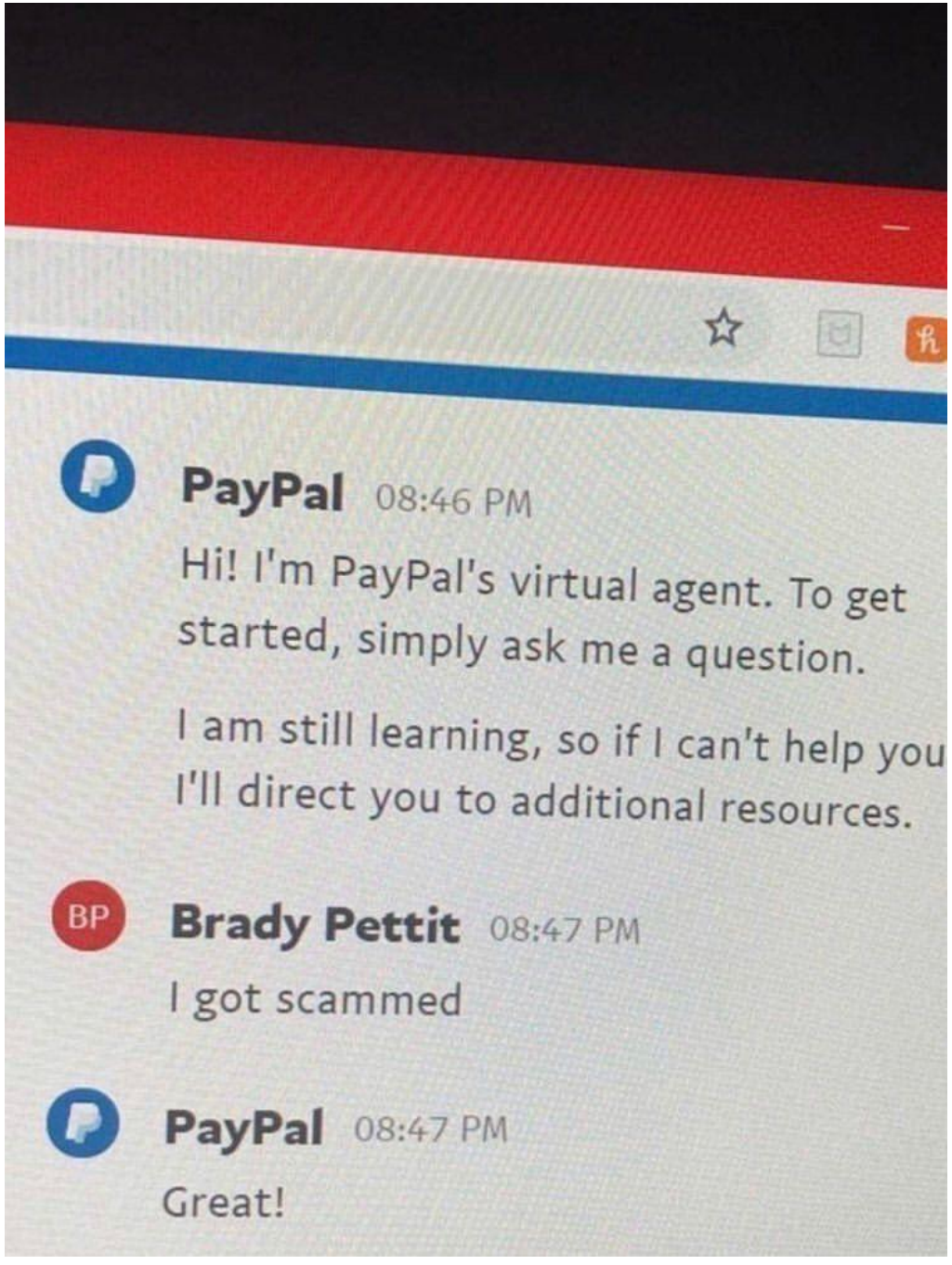


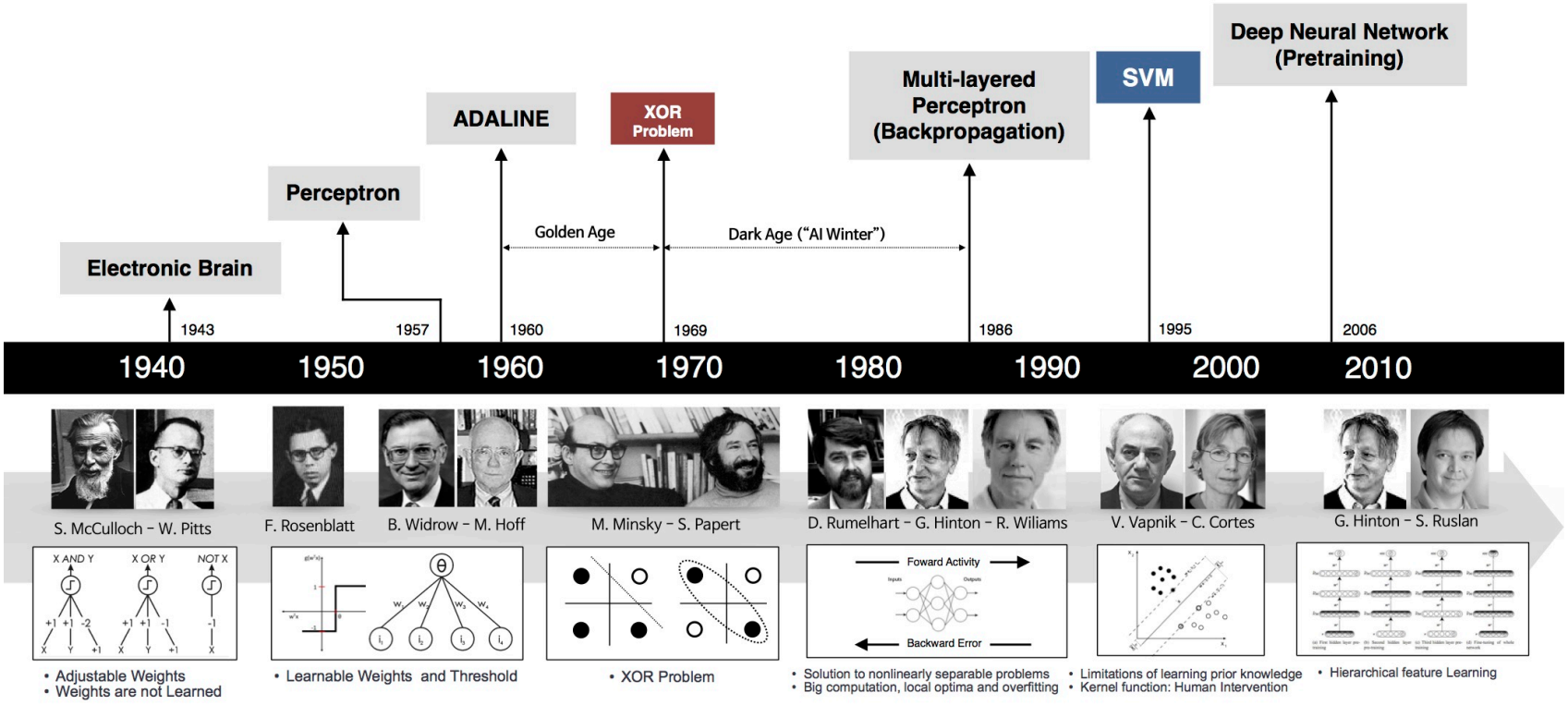
Google DeepMind - AlphaGo [link]

Google DeepMind - AlphaStar Starcraft II [link]

OpenAI Dota2 [link]


Kaenbyou 01/13/2018
60+ hours on 16 GPU nvidia CUDA cluster.
dog


PayPal 08:46 PM
Hi! I'm PayPal's virtual agent. To get started, simply ask me a question.
I am still learning, so if I can't help you I'll direct you to additional resources.
BP
Brady Pettit 08:47 PM
I got scammed
PayPal 08:47 PM
Great!


Deep Neural Network (Pretraining)
SVM
Multi-layered Perceptron (Backpropagation)
XOR Problem
ADALINE
Perceptron
Electronic Brain
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton - R. Wiliams
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

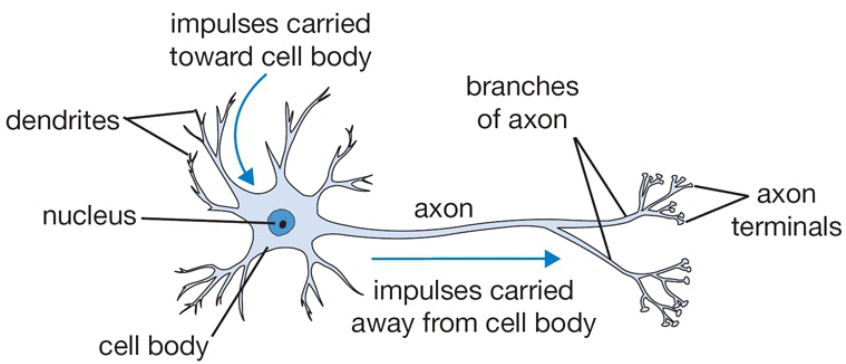
Neuron: The basic computational unit of the brain

Walter Pitts and Warren McCulloch [1943]:

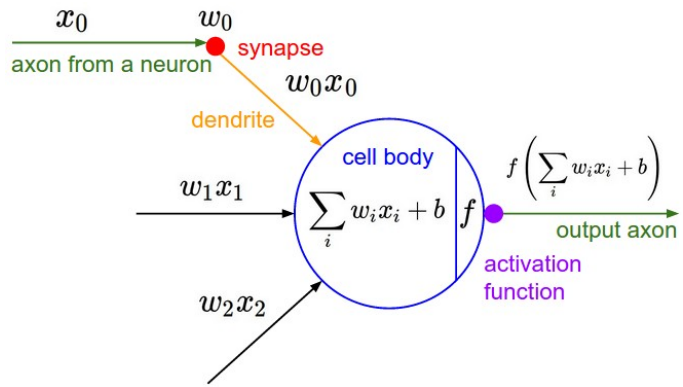
Thresholded logic unit (designed to mimic the way a neuron was thought to work)

adjustable but *not learned* weights

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$



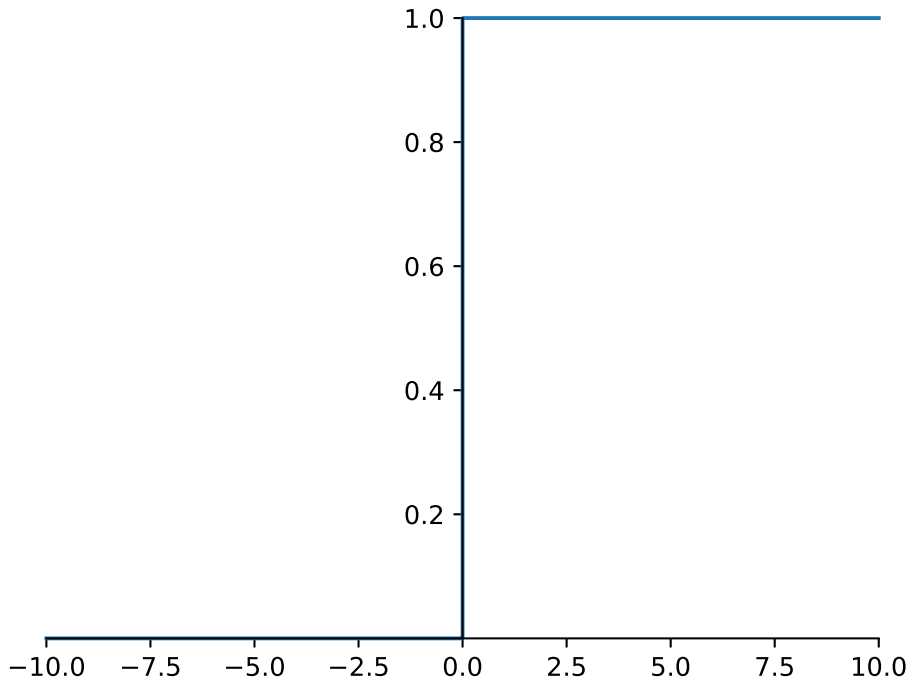
(i) Biological Neuron



(ii) Artificial Neuron

Thresholded logic unit: Maps inputs to 1 or 0

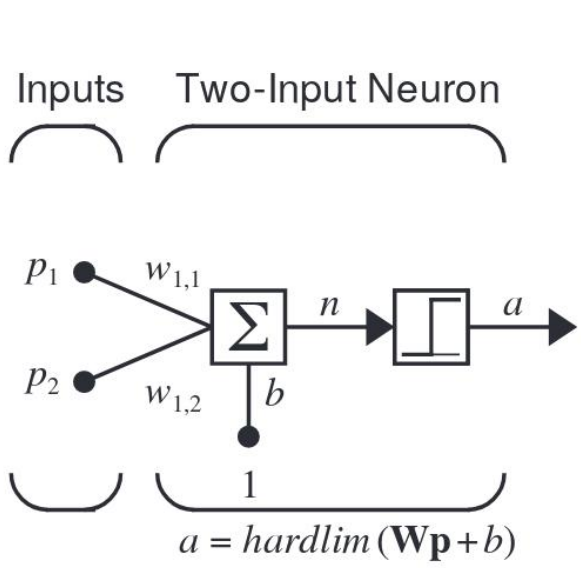
$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$



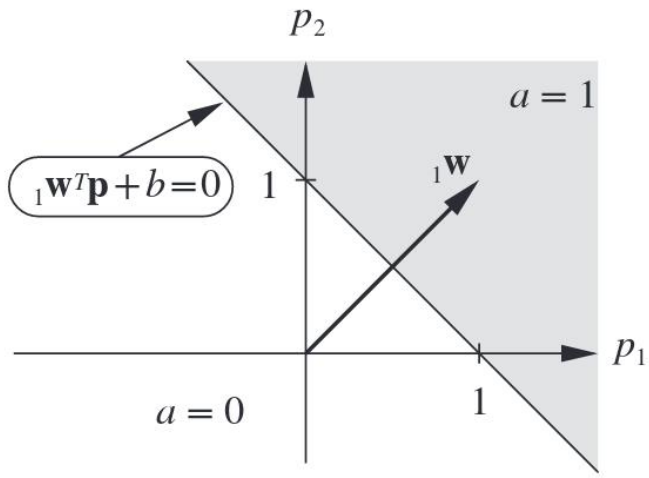
Frank Rosenblatt's "perceptron" [1957]:

First real precursor to modern neural networks

Developed **rule for learning weights**



$$w_{1,1} = 1 \qquad w_{1,2} = 1 \qquad b = -1$$



$$a = hardlim(_1\mathbf{w}^{\mathrm{T}}\mathbf{p} + b) = hardlim(w_{1,1}p_1 + w_{1,2}p_2 + b)$$

Decision Boundary: $w_{1,1}p_1 + w_{1,2}p_2 + b = 0$

Linear equation: $ax + by + c = 0$

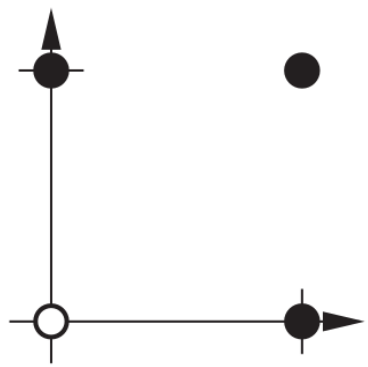
Supervised Learning

Network is provided with a set of examples of proper network behavior (inputs/targets)

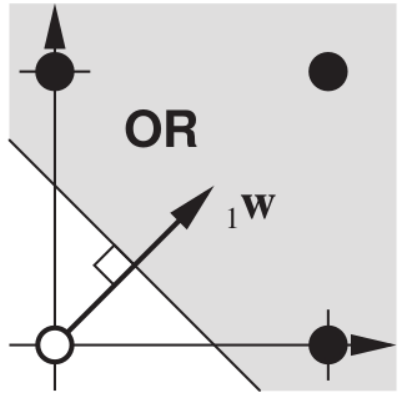
$$\{p_1, t_1\}, \{p_2, t_2\}, \ldots, \{p_Q, t_Q\}$$

Example - OR gate

$$\left\{\mathbf{p}_1 = \begin{bmatrix}0\\0\end{bmatrix}, t_1 = 0\right\} \quad \left\{\mathbf{p}_2 = \begin{bmatrix}0\\1\end{bmatrix}, t_2 = 1\right\} \quad \left\{\mathbf{p}_3 = \begin{bmatrix}1\\0\end{bmatrix}, t_3 = 1\right\} \quad \left\{\mathbf{p}_4 = \begin{bmatrix}1\\1\end{bmatrix}, t_4 = 1\right\}$$



Example - OR gate



Weight vector should be orthogonal to the decision boundary.

$$_1\mathbf{w} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

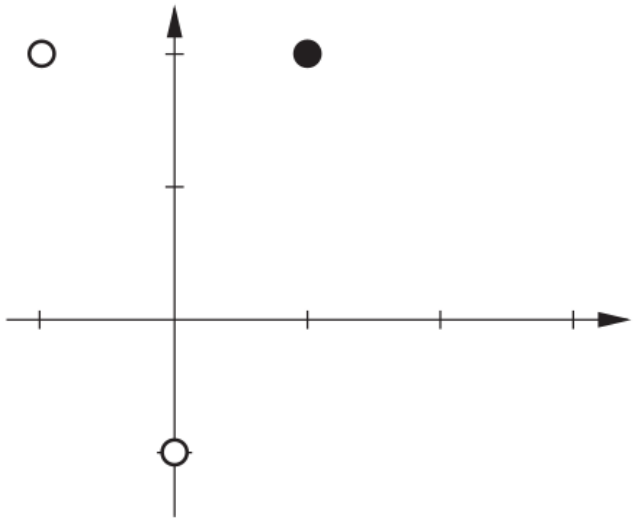
Pick a point on the decision boundary to find the bias.

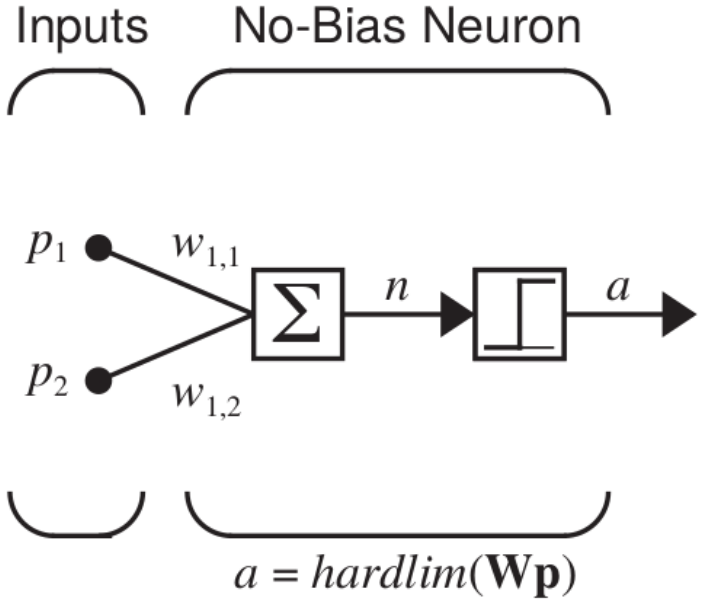
$$_1\mathbf{w}^T\mathbf{p} + b = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} + b = 0.25 + \mathrm{b} = 0 \quad \Rightarrow \quad b = -0.25$$

Learning Rule Test Problem

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

$$\left\{\mathbf{p}_1 = \begin{bmatrix}1\\2\end{bmatrix}, t_1 = 1\right\} \quad \left\{\mathbf{p}_2 = \begin{bmatrix}-1\\2\end{bmatrix}, t_2 = 0\right\} \quad \left\{\mathbf{p}_3 = \begin{bmatrix}0\\-1\end{bmatrix}, t_3 = 0\right\}$$

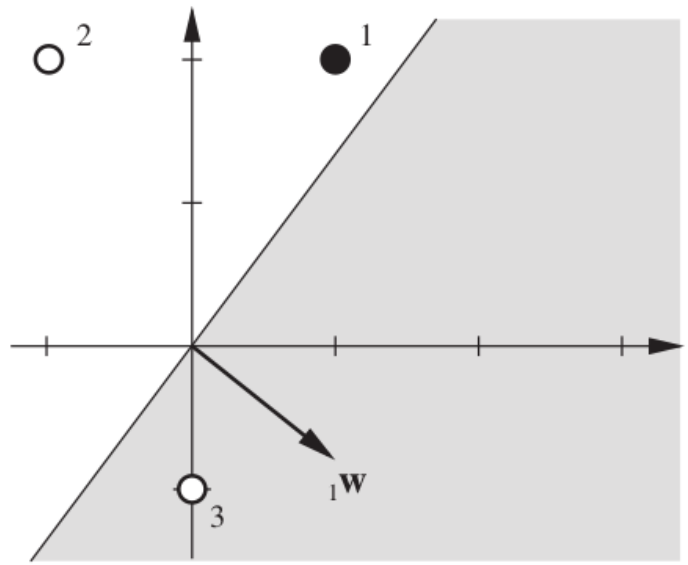




Starting Point

Random initial weight:

$$ {}_1\mathbf{w} = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix} $$



Present $\mathbf{p}_1$ to the network:

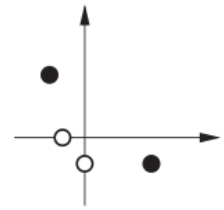
$$ a = hardlim({}_1\mathbf{w}^T \mathbf{p}_1) = hardlim\left(\begin{bmatrix} 1.0 & -0.8 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right) $$

$$ a = hardlim(-0.6) = 0 $$

Incorrect Classification.

Tentative Learning Rule

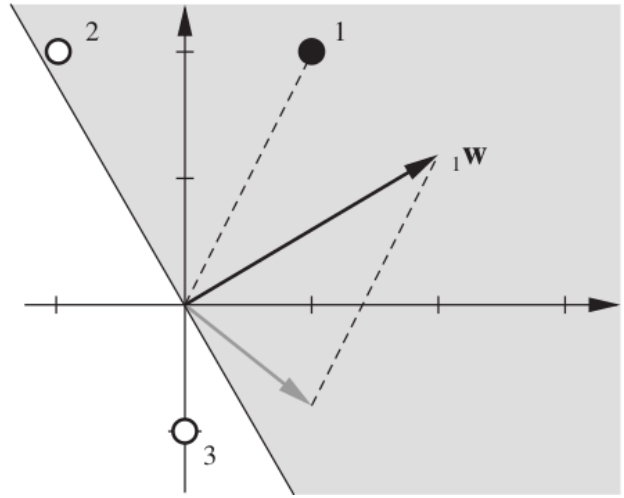
- Set ${}_1\mathbf{w}$ to $\mathbf{p}_1$
  - Not stable ×



- Add $\mathbf{p}_1$ to ${}_1\mathbf{w}$ ✓

Tentative Rule: If $t = 1$ and $a = 0$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}_1 = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2.0 \\ 1.2 \end{bmatrix}$$



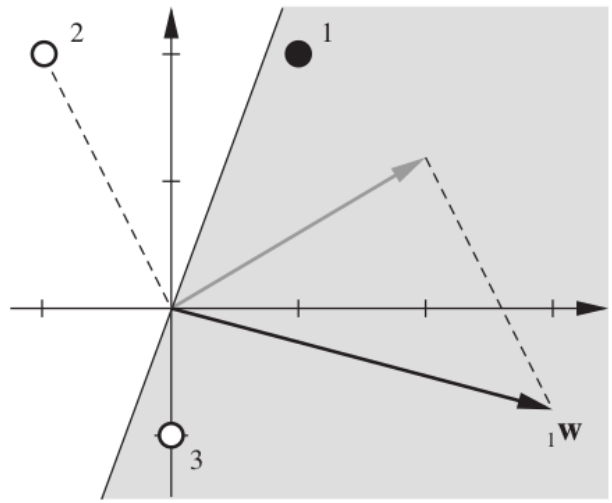
Second Input Vector

$$a = hardlim({}_1\mathbf{w}^T\mathbf{p}_2) = hardlim\left(\begin{bmatrix}2.0 & 1.2\end{bmatrix}\begin{bmatrix}-1\\2\end{bmatrix}\right)$$

$$a = hardlim(0.4) = 1 \quad \text{(Incorrect Classification)}$$

Modification to Rule: If $t = 0$ and $a = 1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}_2 = \begin{bmatrix}2.0\\1.2\end{bmatrix} - \begin{bmatrix}-1\\2\end{bmatrix} = \begin{bmatrix}3.0\\-0.8\end{bmatrix}$$

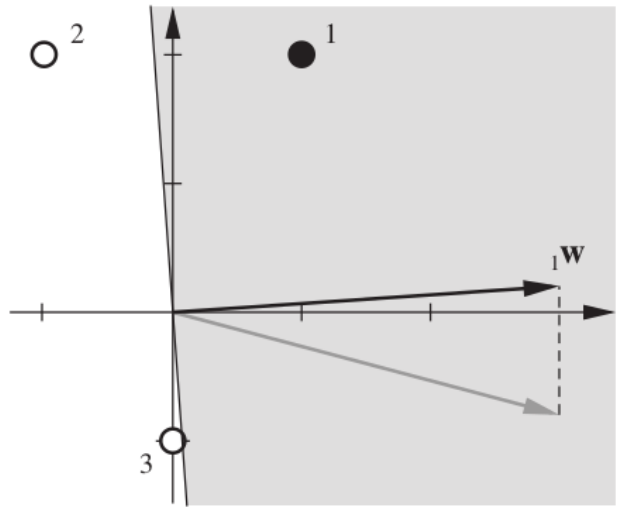


Third Input Vector

$$a = hardlim({}_1\mathbf{w}^T \mathbf{p}_3) = hardlim\left(\begin{bmatrix}3.0 & -0.8\end{bmatrix}\begin{bmatrix}0\\-1\end{bmatrix}\right)$$

$$a = \text{hardlim}(0.8) = 1 \qquad \text{(Incorrect Classification)}$$

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}_3 = \begin{bmatrix}3.0\\-0.8\end{bmatrix} - \begin{bmatrix}0\\-1\end{bmatrix} = \begin{bmatrix}3.0\\0.2\end{bmatrix}$$



Patterns are now correctly classified.

If $t = a$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$.

Unified Learning Rule

If $t = 1$ and $a = 0$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$

If $t = 0$ and $a = 1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$

If $t = a$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$

$$e = t - a$$

If $e = 1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$

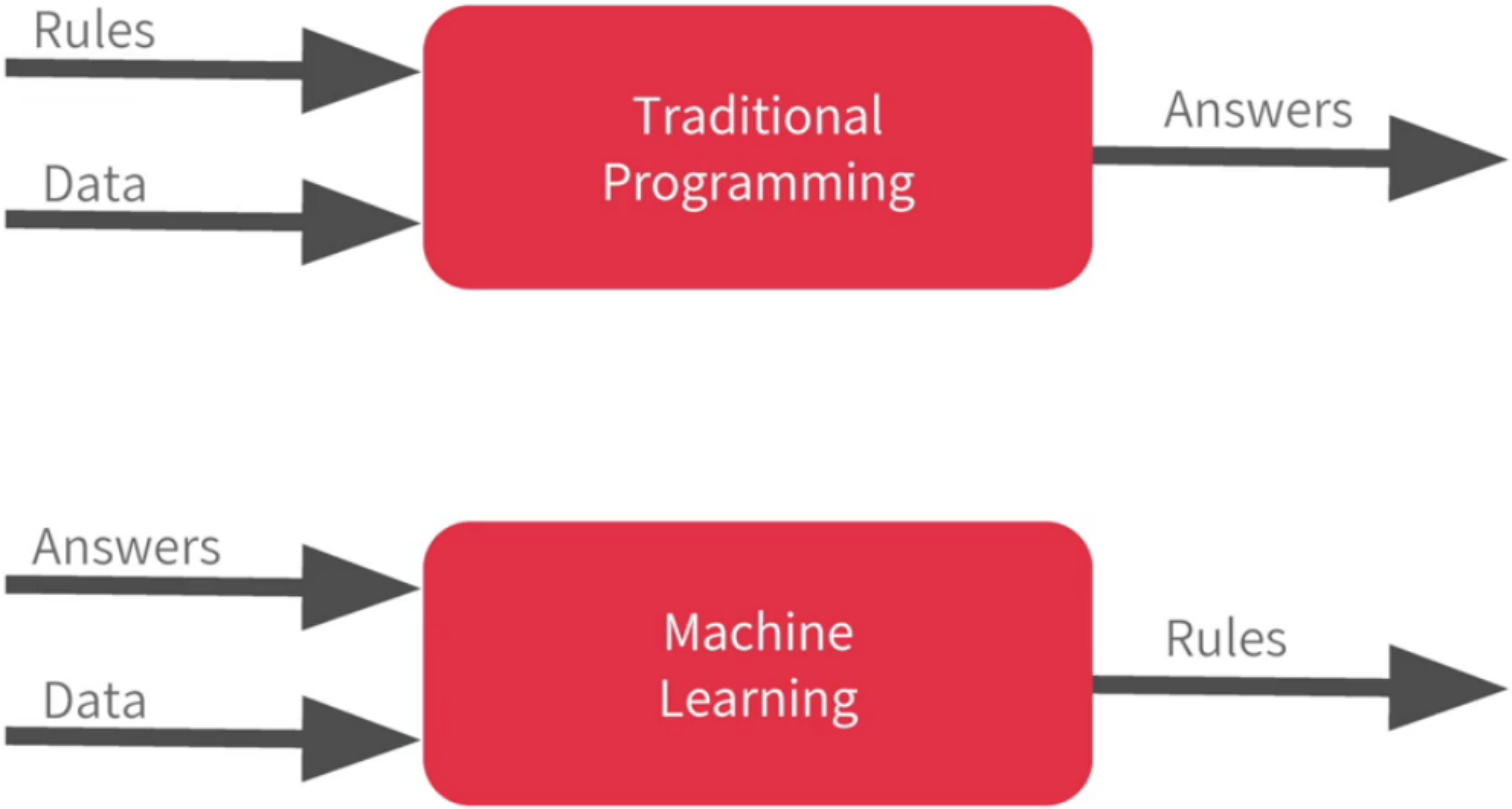
If $e = -1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$

If $e = 0$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$

$$ {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + e\mathbf{p} = {}_1\mathbf{w}^{old} + (t-a)\mathbf{p} $$

$$ b^{new} = b^{old} + e $$

A bias is a weight with an input of 1.


Rules
Data
Traditional Programming
Answers
Answers
Data
Machine Learning
Rules

Perceptron convergence theorem

The perceptron learning rule will converge to a weight vector (not necessarily unique) that gives the correct response for all training patterns, and it will do so in a finite number of steps.

Rosenblatt was so confident that the perceptron would lead to true AI, that in 1959 he remarked:

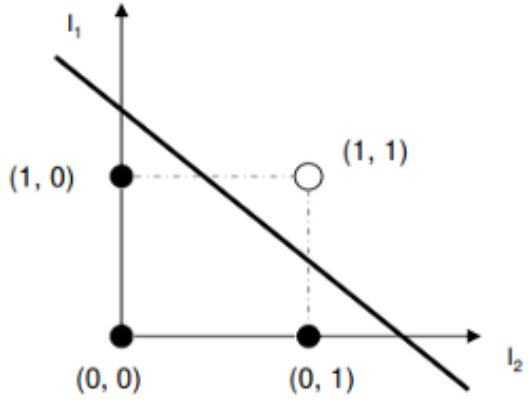
[The perceptron is] the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

Marvin Minsky and Seymor Papert - XOR problem [1969]

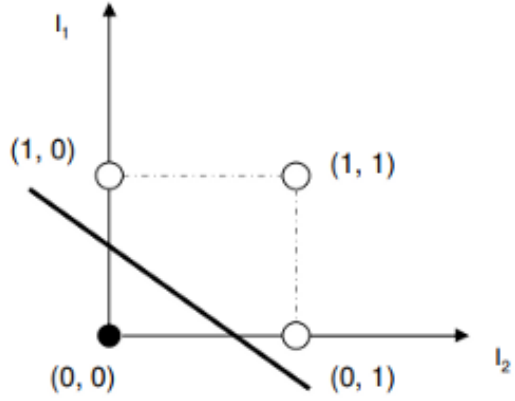
They showed that the perceptron was incapable of learning the simple exclusive-or (XOR) function.
They proved that it was theoretically impossible for it to learn such a function, no matter how long you let it train.
(this isn't surprising to us, as the model implied by the perceptron is a linear one and the XOR function is nonlinear)

At the time this was enough to kill all research on neural nets

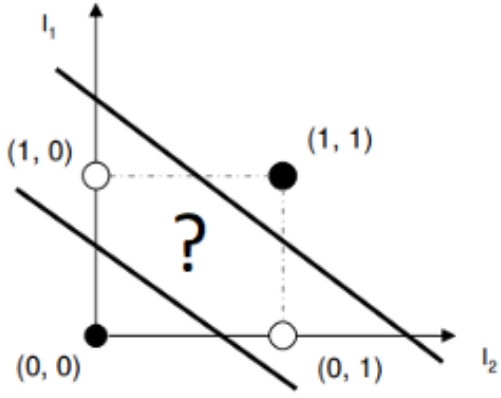
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0



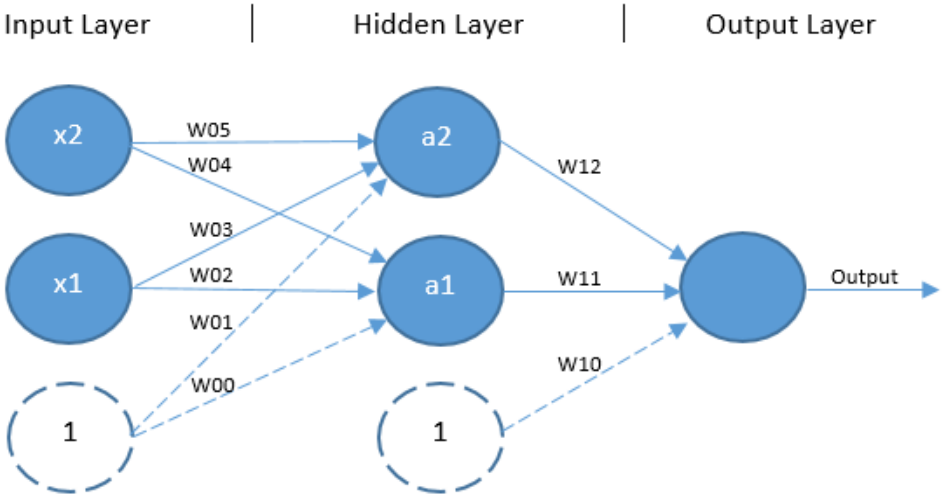

BRACE YOURSELVES
AI WINTER IS COMING
memegenerator.net

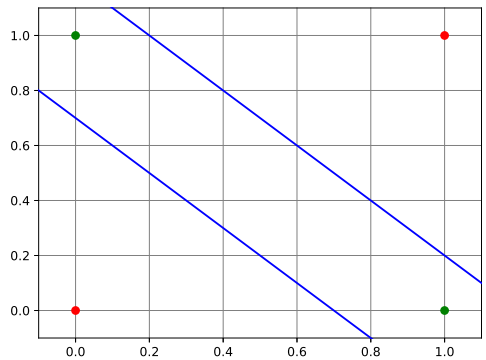
XOR Solution???

One single perceptron neuron: One decision boundary (hyperplane)

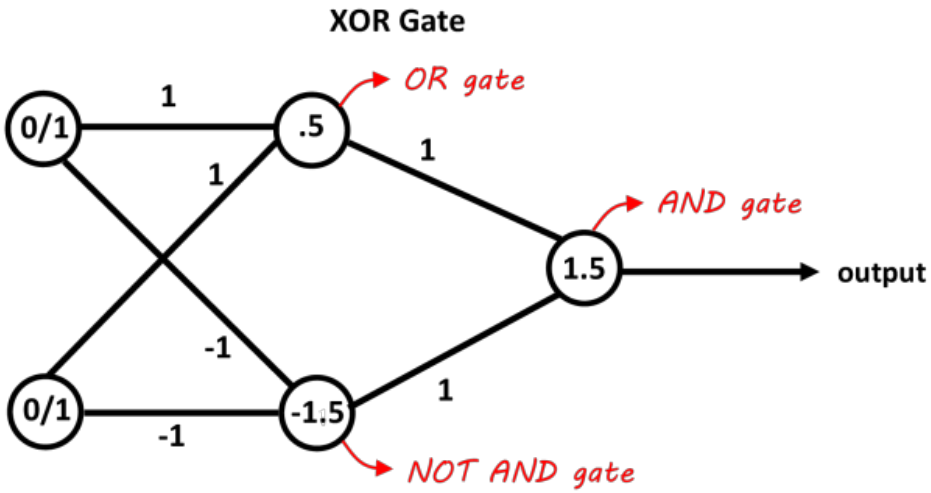
Two perceptron neurons: Two decision boundaries

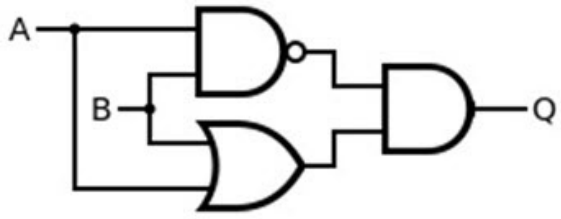
Multi Layer Perceptron (MLP)





As a logic gates





A network of perceptrons can be used to simulate a circuit containing many NAND gates. And because NAND gates are universal for computation, it follows that perceptrons are also universal for computation.

In the mathematical theory of artificial neural networks, the universal approximation theorem states that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of R^n, under mild assumptions on the activation function. The theorem thus states that simple neural networks can represent a wide variety of interesting functions when given appropriate parameters

The problem: There was no equally powerfull rule (to the Perceptron's) for learning in networks with hidden units.

David. Rumelhard, Geoffrey. Hinton and Ronald. Williams - Learning Representations by back-propagating errors [1986]
(aka "Backpropagation")

Perceptron's delta rule: $\Delta_p w_{ji} = \eta(t_{pj} - o_{pj}) i_{pi} = \eta \delta_{pj} i_{pi}$

How was this rule derived?
For linear units, this rule minimizes the squares of the differences between the actual and the desired output values summed over the output units and all pairs of input/output vectors. $E = \sum E_p$

$$E_p = \frac{1}{2} \sum_j (t_{pj} - o_{pj})^2$$

We have: input and target data, network architecture
We want: an algorithm which lets us find weights and biases so that the output from the network approximates y(x) (target *t*) for all training inputs x.
To quantify how well we're achieving this goal we define a cost function.

$$C(w,b) = \frac{1}{2n}\sum_{x}(t-\alpha)^2$$

w the collection of all weights in the network, *b* all the biases, *n* the total number of training inputs, α is the vector of outputs from the network when *x* is input, and the sum is over all training inputs, *x*.

The output α depends on *x*, *w* and *b*.
We'll call *C* the quadratic cost function (or mean squared error - MSE).

$C(w,b)$ is non-negative.
$C(w,b)$ becomes small i.e $C(w,b) \approx 0$ when $t \approx \alpha$.

$$C(w,b) = \frac{1}{2n}\sum_x (t-\alpha)^2$$

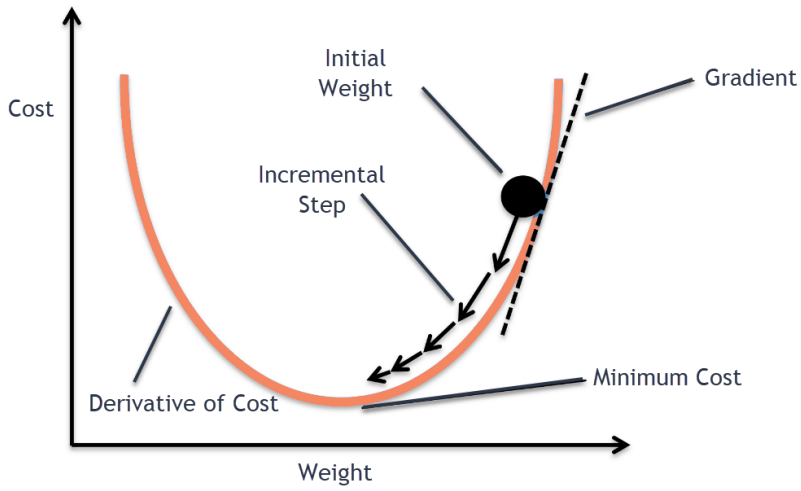
$y = 2x + 4$
$x = [1, 2, 3, 4, 5, 6, 7, 8]$
$t = [6, 8, 10, 12, 14, 16, 18, 20]$
For $w = 3, b = 2$.
$\alpha = [5, 8, 11, 14, 17, 20, 23, 26]$

$C(w,b) = \frac{(6-5)^2+(8-8)^2+(10-11)^2+\ldots+(20-26)^2}{16} = 5.75$

The aim of our training algorithm will be to minimize the cost C(w,b) as a function of the weights and biases. In other words, we want to find a set of weights and biases which make the cost as small as possible. We'll do that using an algorithm known as **gradient descent**.

Minimize some function

The derivative of the function shows us the way the function changes. From Calculus is known that a function has a minimum (or maximum) value when its derivative at that point is zero.

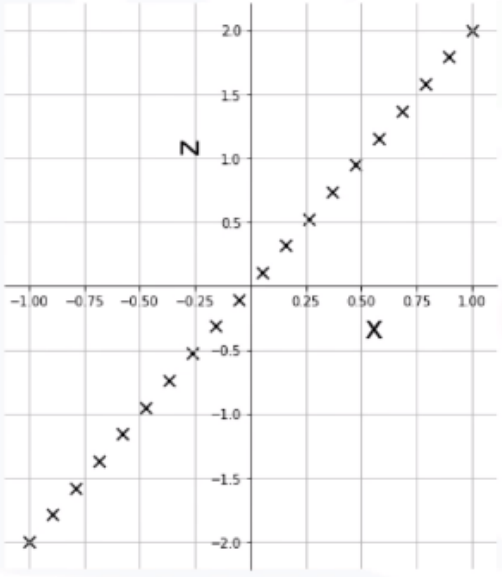


$$f(x+\epsilon) \approx f(x) + \epsilon f'(x)$$

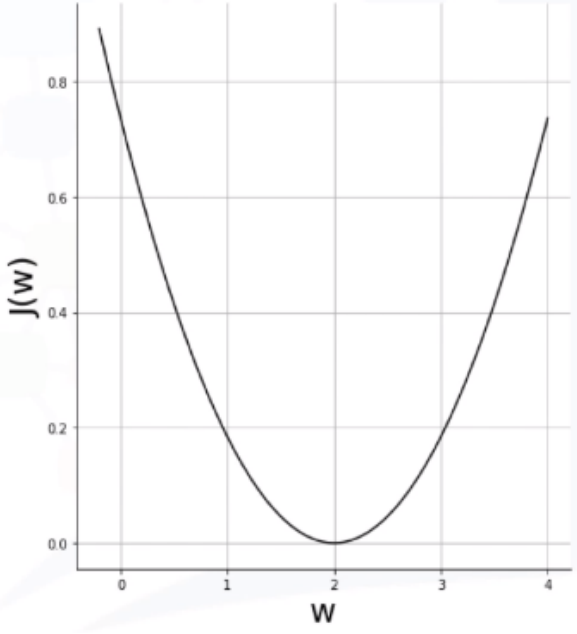
It is useful for minimizing the function because it tells us how to change x in order to make a small improvement in y

Gradient Descent Algorithm

$$Z = wX$$

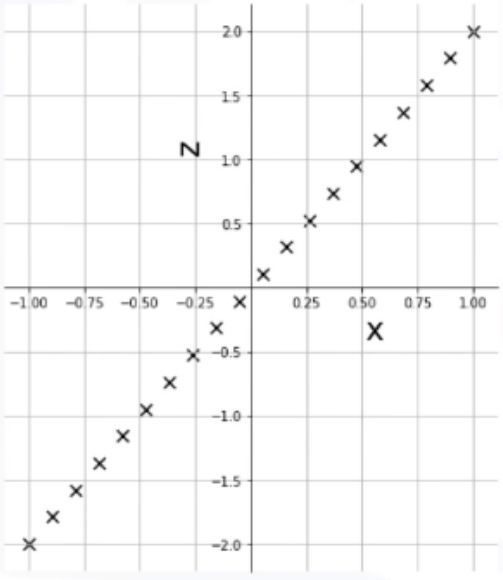


$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(z_i - wx_i)^2$$

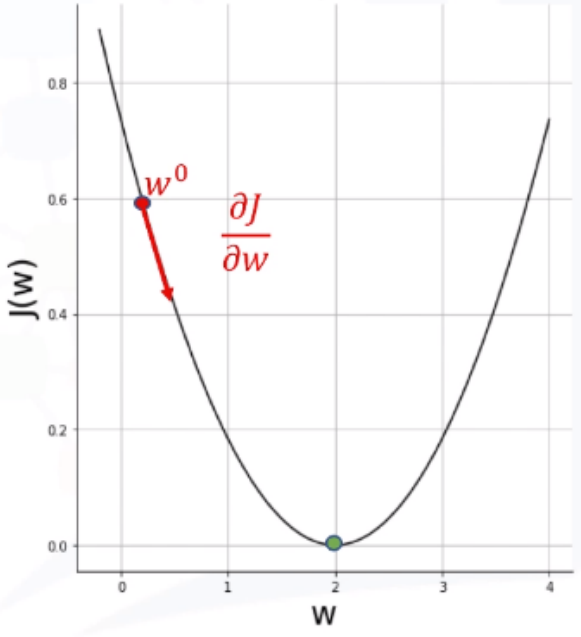


Gradient Descent Algorithm

$$Z = wX$$

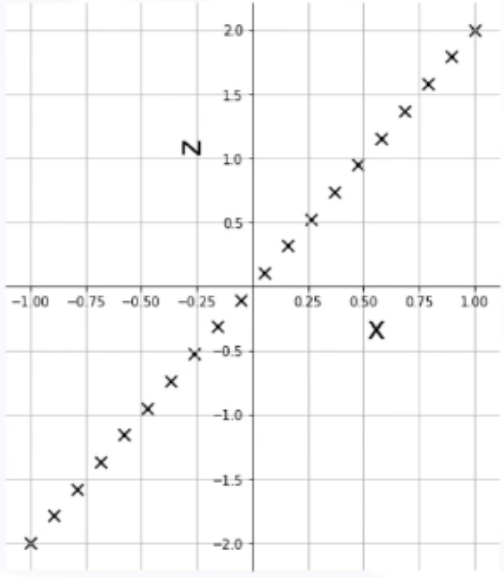


$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(z_i - wx_i)^2$$

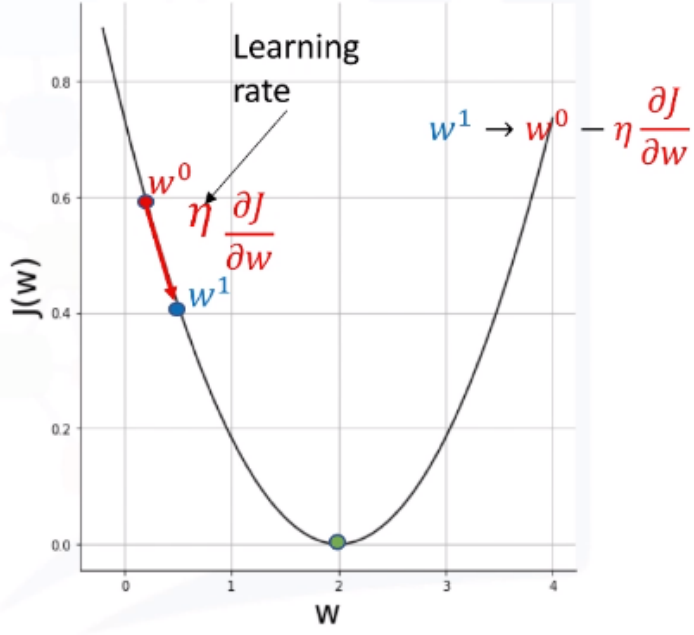


Gradient Descent Algorithm

$$Z = wX$$

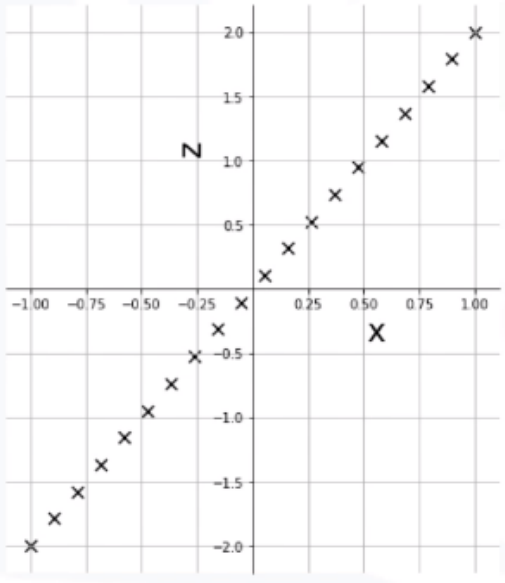


$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(z_i - wx_i)^2$$

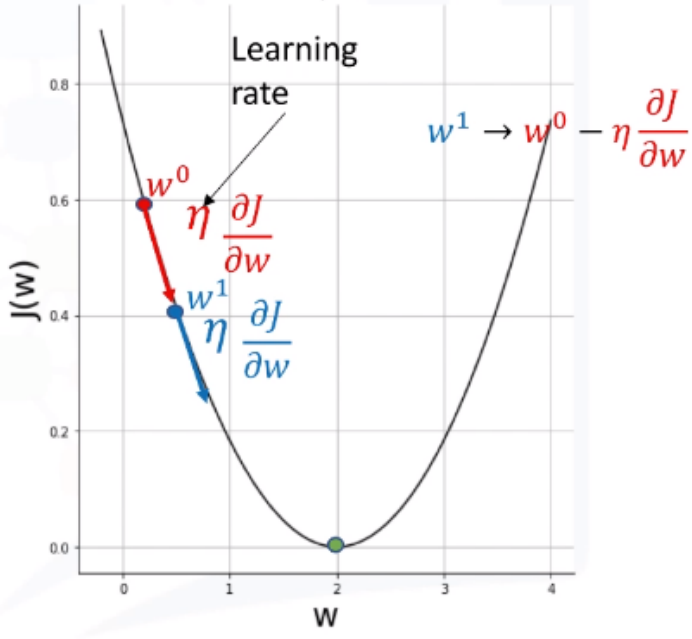


Gradient Descent Algorithm

$$Z = wX$$



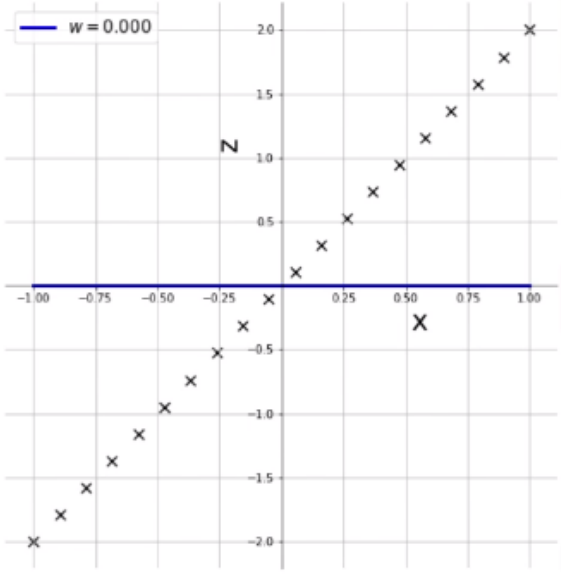
$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(z_i - wx_i)^2$$

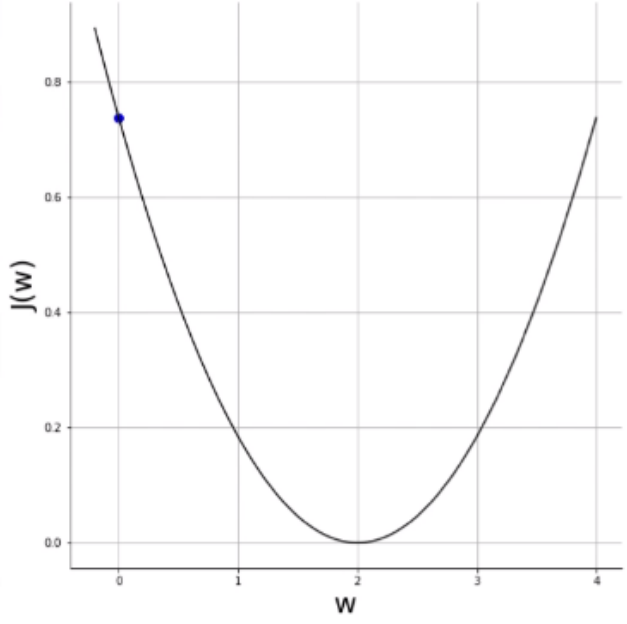


Gradient Descent – Initialization

$$Z = wX$$

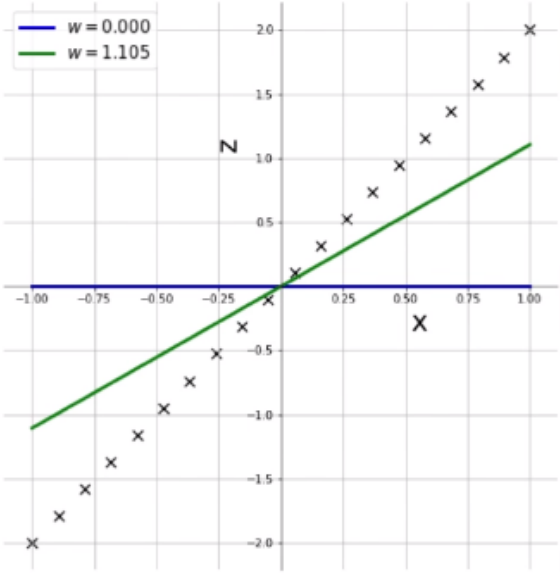
$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(z_i - wx_i)^2$$



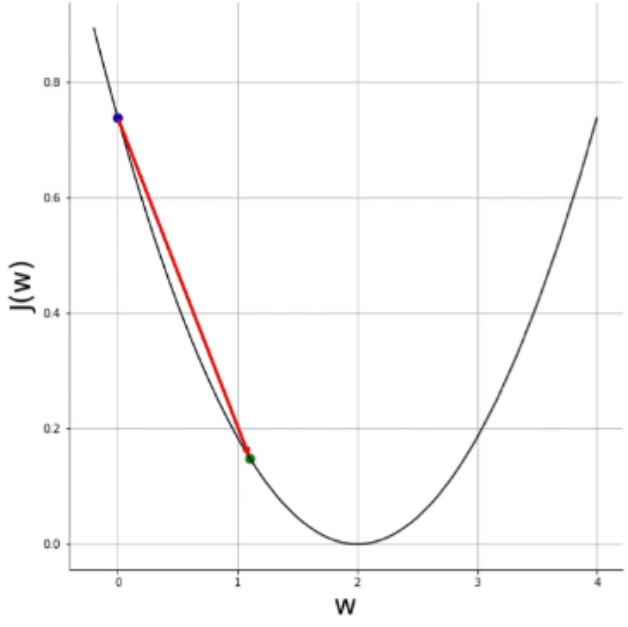


Gradient Descent – 1st Iteration

$$Z = wX$$



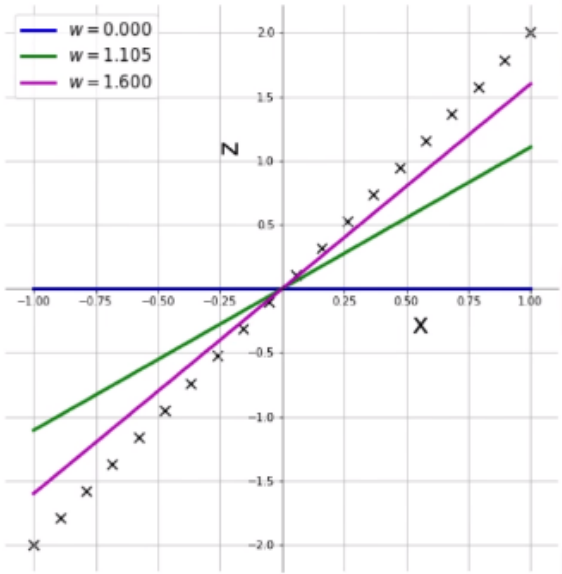
$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(z_i - wx_i)^2$$

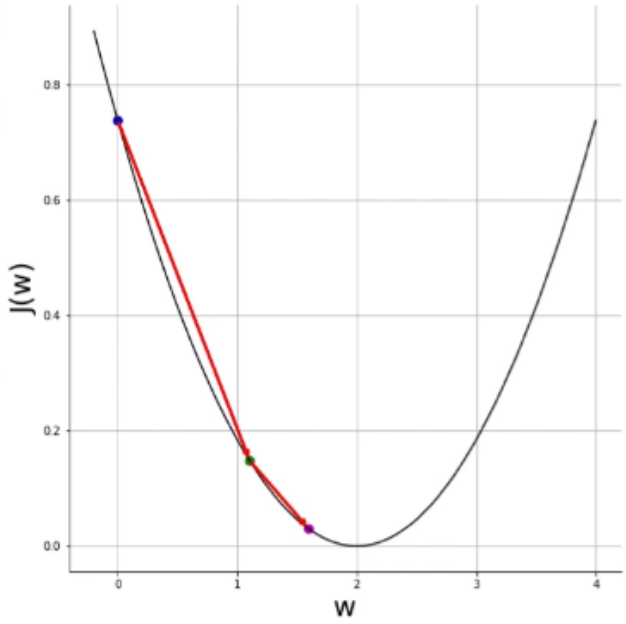


Gradient Descent – 2nd Iteration

$$Z = wX$$

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(z_i - wx_i)^2$$

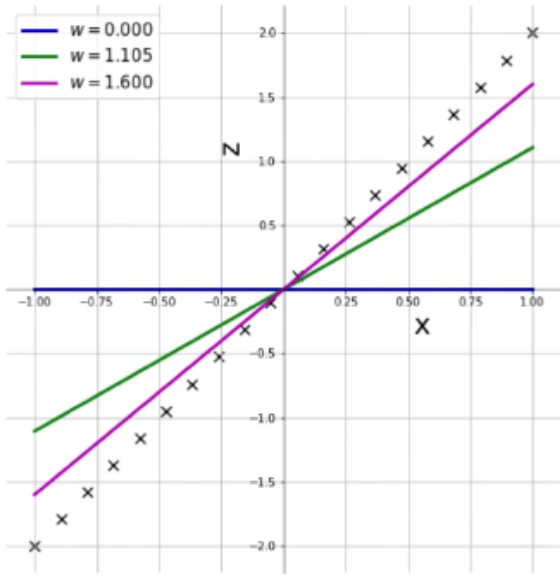


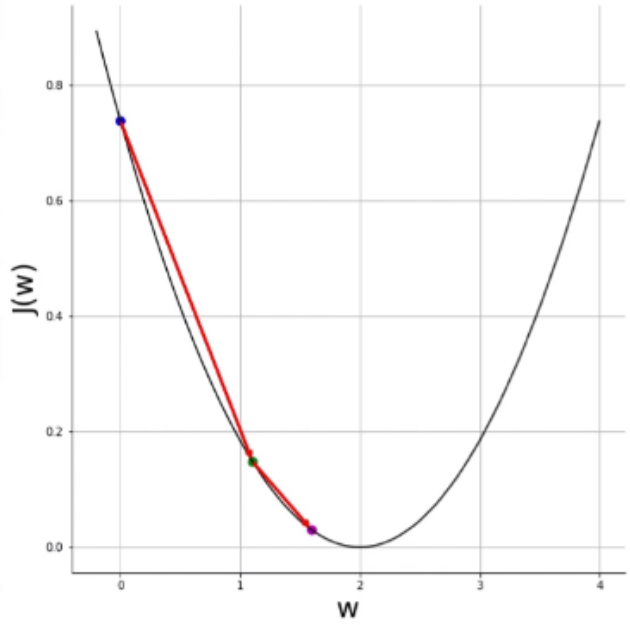


Gradient Descent – 2nd Iteration

$$Z = wX$$

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(z_i - wx_i)^2$$

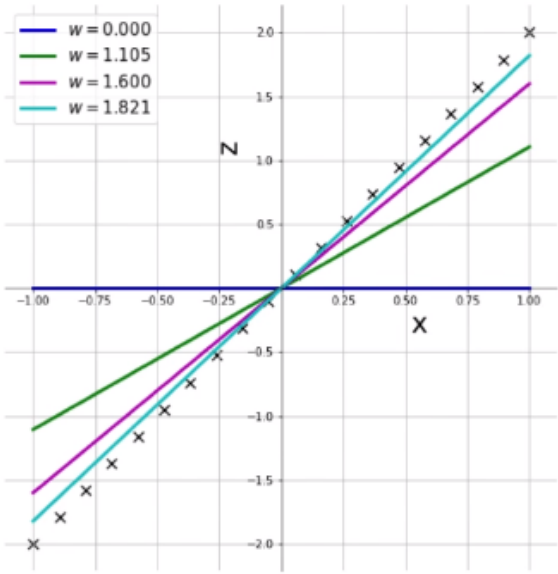


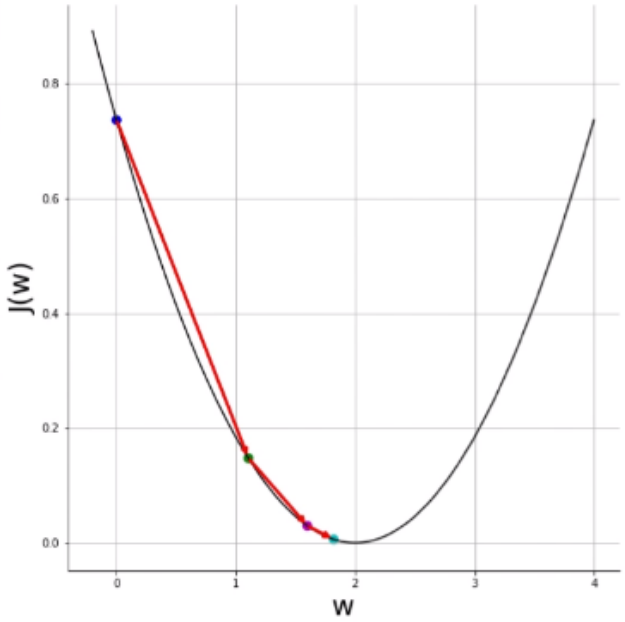


Gradient Descent – 3rd Iteration

$$Z = wX$$

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(z_i - wx_i)^2$$

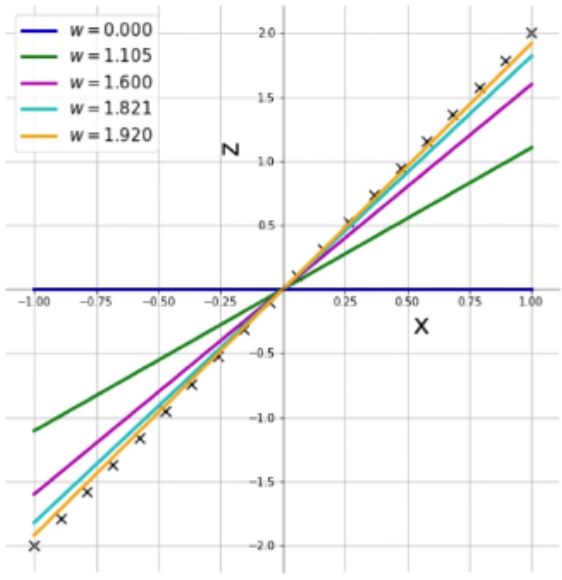


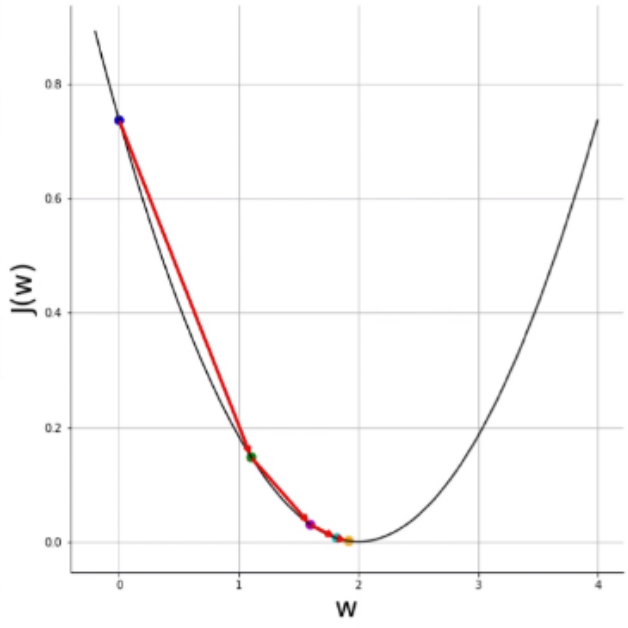


Gradient Descent – 4th Iteration

$$Z = wX$$

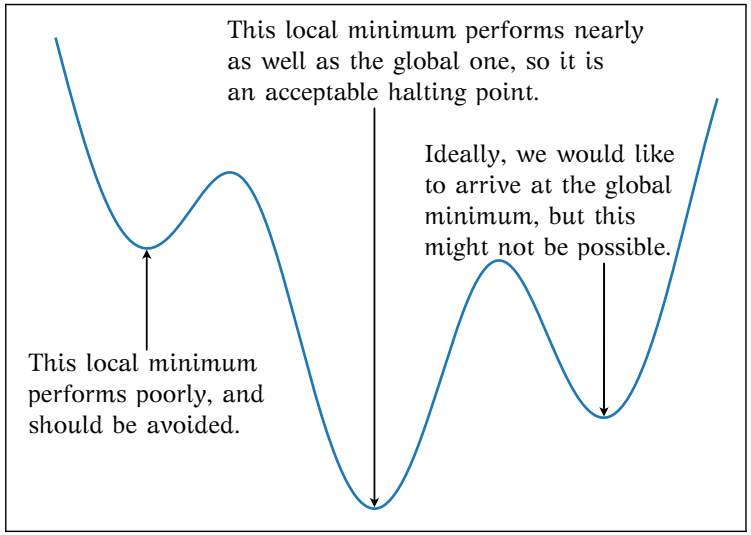
$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(z_i - wx_i)^2$$



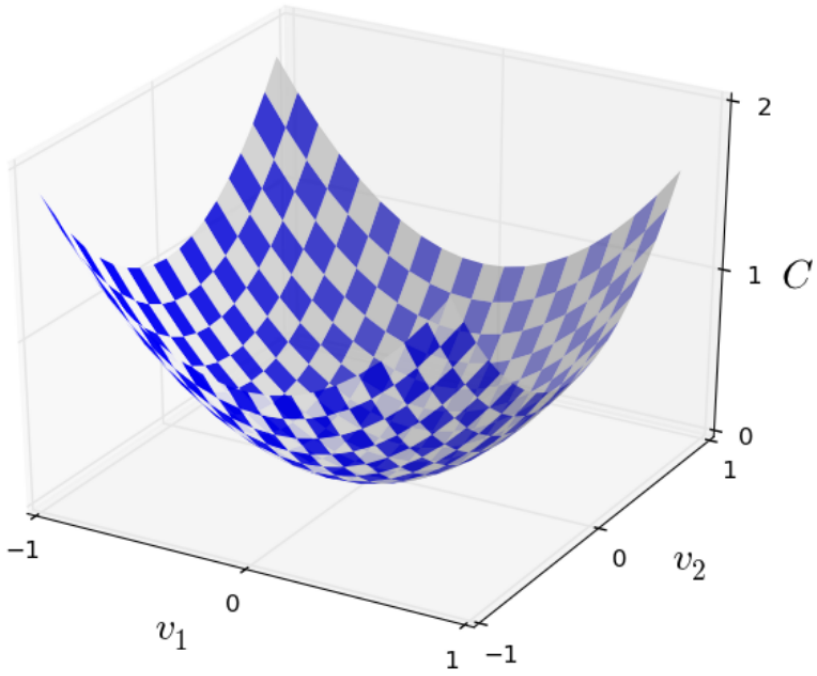


Critical points:

The cost functions may have many local minima that are not optimal.
This makes the optimization difficult, especially when the input is multidimensional.
Usually we settle for finding a value of f which is low, but not necessarily minimal.



Multidimensional inputs:



Usually we have multiple inputs and for that we use partial derivatives.
There is a partial derivative for each input variable.
The gradient of f is the vector containing all the partial derivatives $\nabla_x f(x)$. Element i of the gradient is the partial derivative of f with respect to x_i.
In multiple dimesions, critical points are these where every element of the gradient is equal to zero.

Gradient descent:

To minimize f we need to find the direction in which f decreases the fastest.

We can find the direction using the directional derivative.

Moving in the direction of the negative gradient we can decrease f. This method is known as gradient descent.

$$x' = x - \epsilon \nabla_x f(x)$$

ϵ is the learning rate, a positive scalar determing the size of the step.

There are many ways to set the value of this parameter

Stochastic Gradient Descent:

Deep Learning needs large datasets which increase the computational cost of the gradient descent method.

For the cost function:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} L(x^{(i)}, y^{(i)}, \theta)$$

the gradient descent method needs to calculate the:

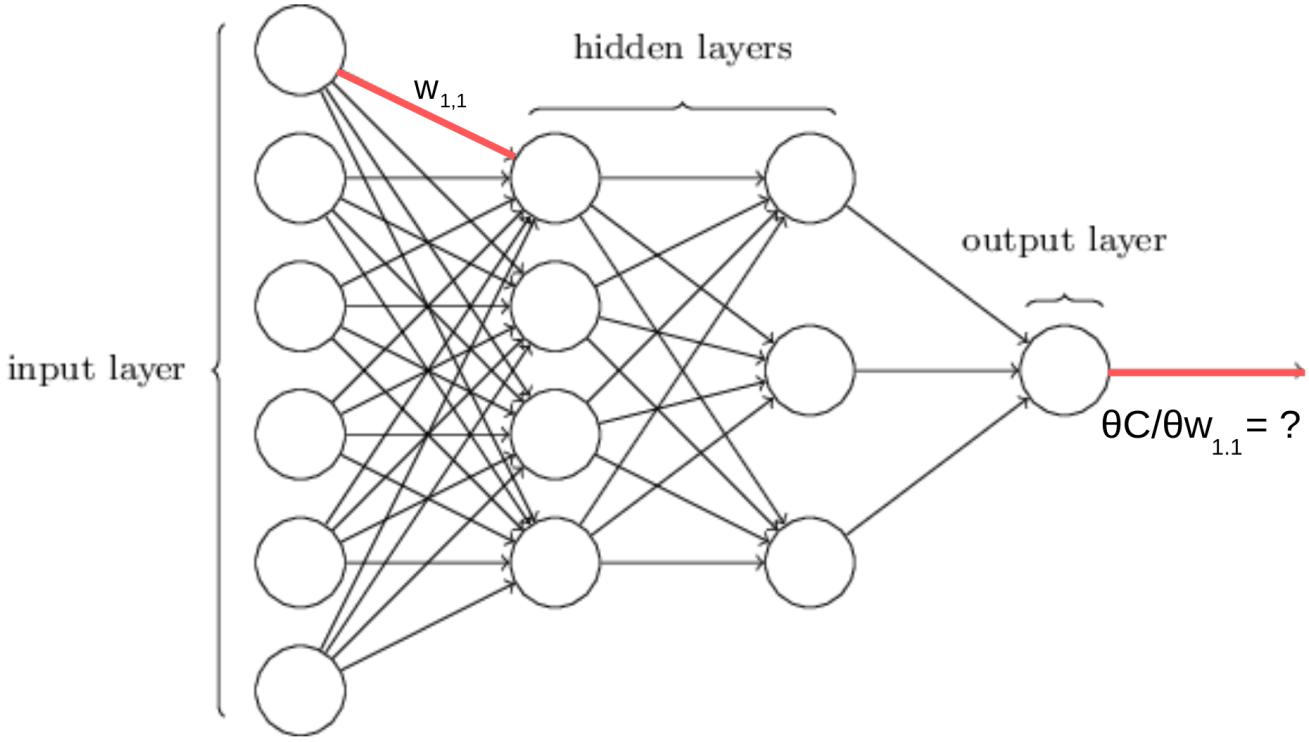
$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta L(x^{(i)}, y^{(i)}, \theta)$$

which is $O(m)$ However, the gradient is a prediction and because of that it can be an approximation using a small subset of the training dataset.

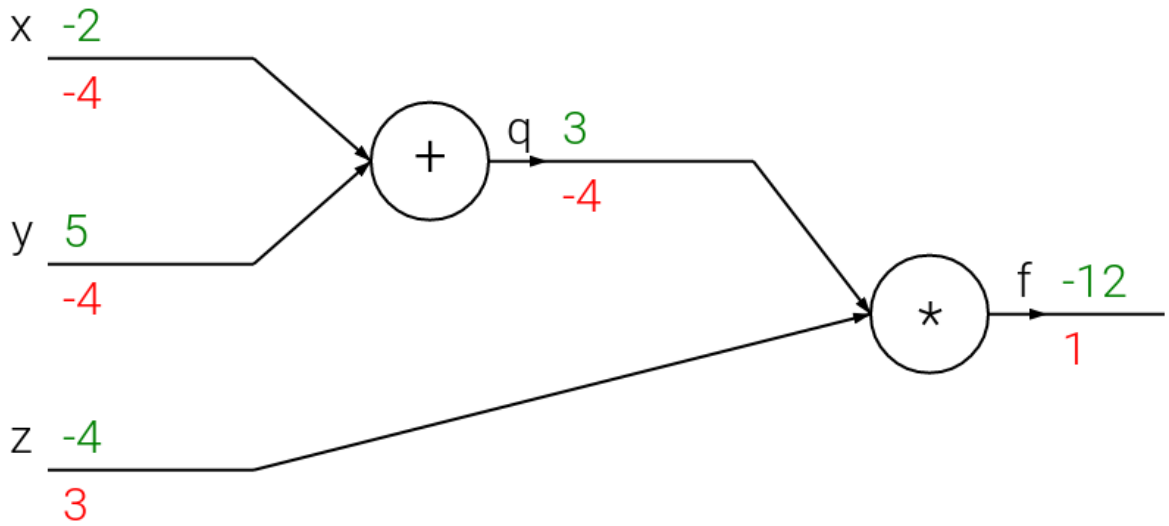
This is the Stochastic Gradient Descent methods which is an extension of the gradient descent.

So, how do we train a Multi Layer Perceptron?

$$w_{1.1}^{new} = w_{1.1}^{old} - \eta \frac{\partial C}{\partial w_{1.1}}$$



Back Propagation: PROPAGATE the errors BACK to the input weights!



Chain rule to the rescue!

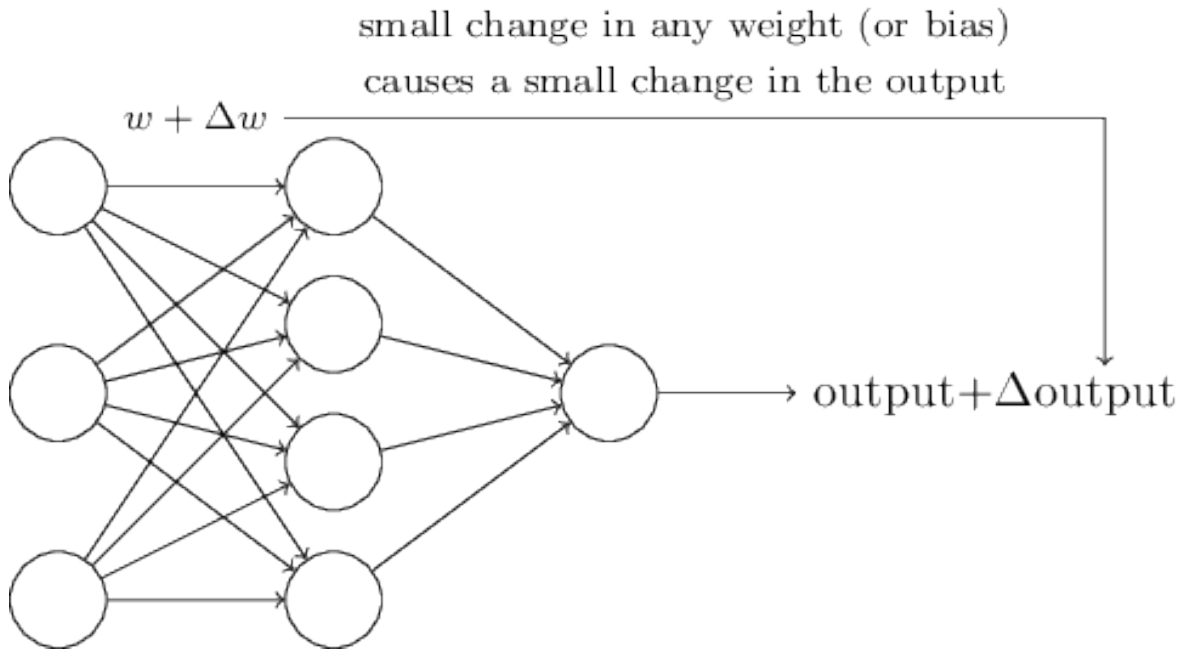
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = \frac{\partial (q \cdot z)}{\partial q}\frac{\partial (x + y)}{\partial x} = z \cdot 1 = z = -4$$

$$\frac{\partial f}{\partial z} = \frac{\partial (q \cdot z)}{\partial z} = q = 3$$

Sigmoid neurons - Sigmoid activation function

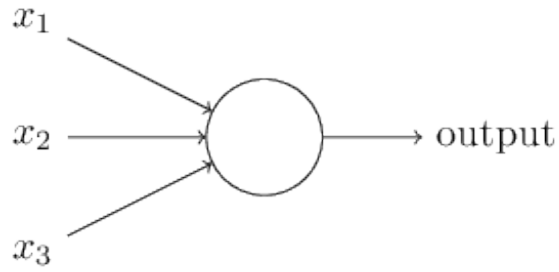
We want a small change in weight to cause only a small corresponding change in the output from the network.

Not possible with perceptrons! - a small change in the weights or bias of any single perceptron in the network can sometimes cause the output of that perceptron to completely flip, say from 0 to 1

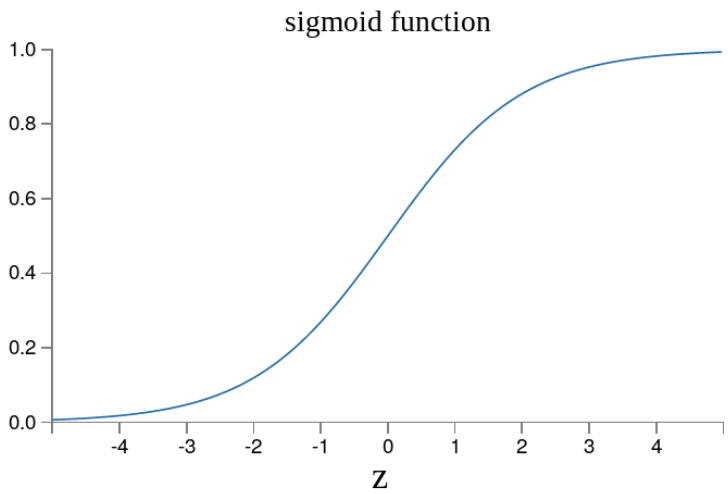


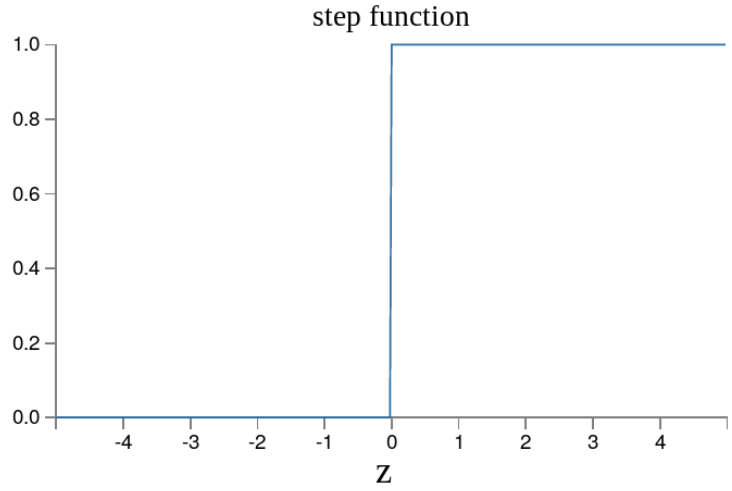
The ouputs of the sigmoid neuron is NOT 0 or 1.

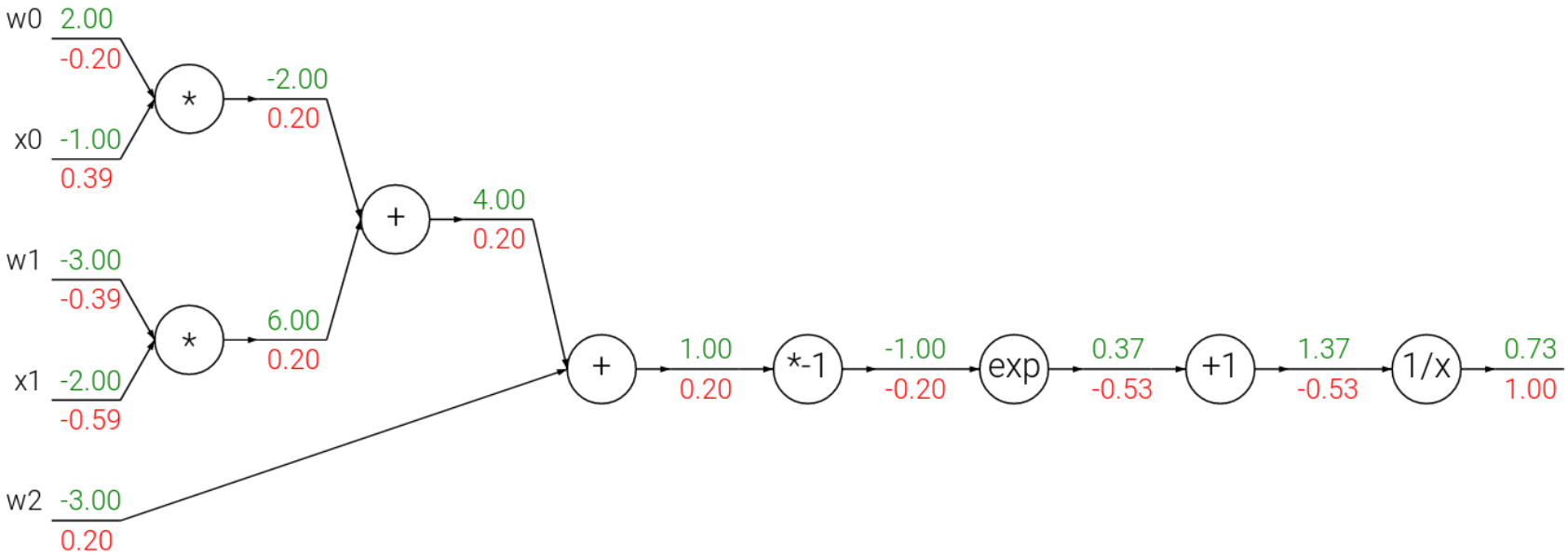
$output = \sigma(w \cdot x + b)$, where σ is the sigmoid function.



$$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+exp(-\sum_j w_j x_j - b)}$$



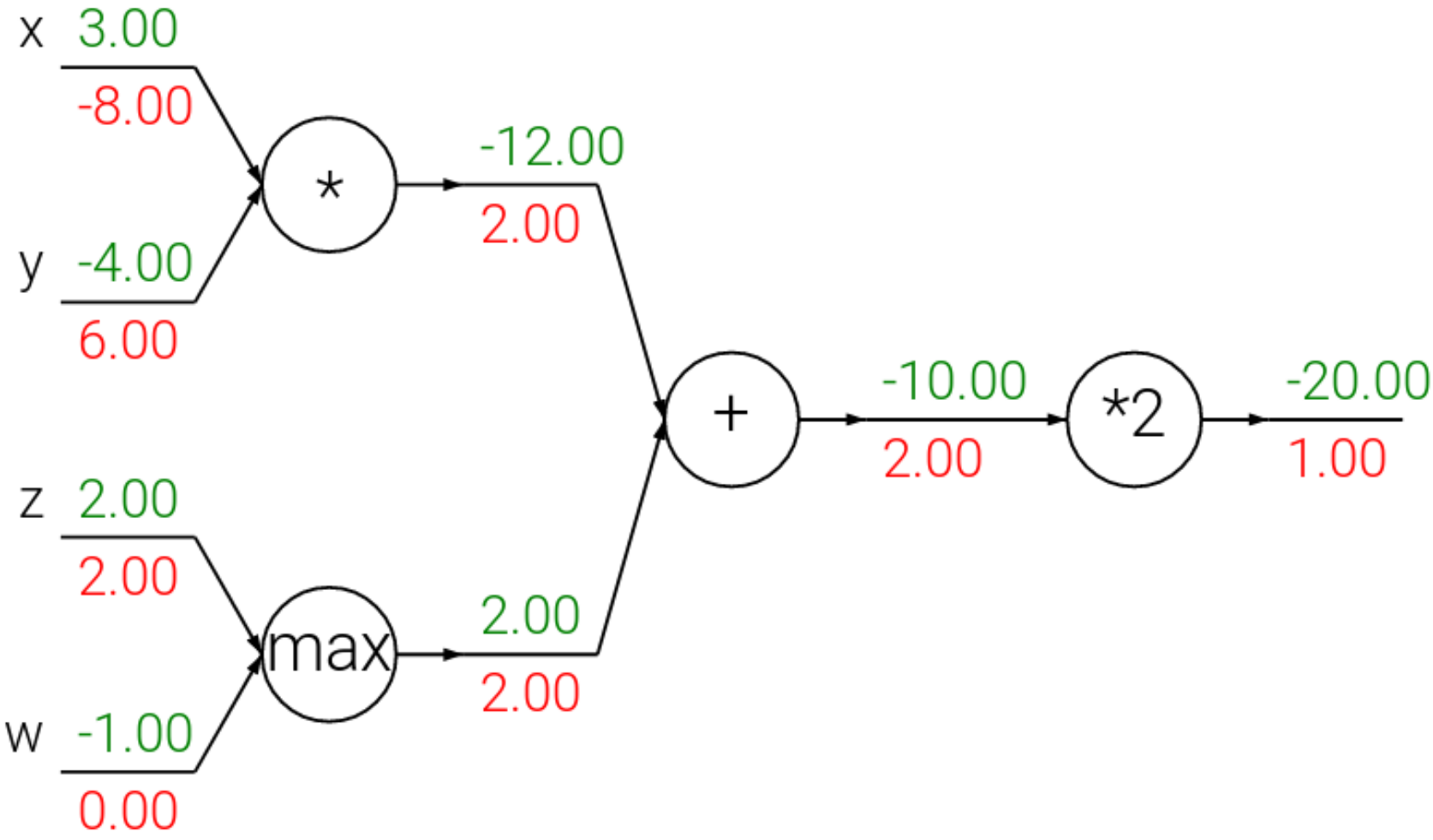




$q = w_0 \cdot x_0, \quad p = w_1 \cdot x_1, \quad s = q + p, \quad t = w_2 + s, \quad r = -t, \quad u = e^r,$
$v = u + 1$ and $z = \frac{1}{v}$

$$\frac{\partial z}{\partial w_0} = \frac{\partial z}{\partial v}\frac{\partial v}{\partial u}\frac{\partial u}{\partial r}\frac{\partial r}{\partial t}\frac{\partial t}{\partial s}\frac{\partial s}{\partial q}\frac{\partial q}{\partial w_0} =$$

$$-0.53 \cdot 1 \cdot 0.37 \cdot (-1) \cdot 1 \cdot 1 \cdot x_0 = -0.1961 \approx -0.20$$


x
3.00
-8.00
y
-4.00
6.00
*
-12.00
2.00
z
2.00
2.00
w
-1.00
0.00
max
2.00
2.00
+
-10.00
2.00
*2
-20.00
1.00

softmax,
NN as matrices,
MNIST example keras (dense)
overfitting / underfitting

CNN

https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html
python demo - XOR problem (boundaries cant converge)
Linear model!!! meaning a straight line or a linear hyperplane

Artificial intelligence could help farmers diagnose crop diseases
TensorFlow: an ML platform for solving impactful and challenging problems
TensorFlow: an ML platform for solving impactful and challenging problems [link]
Let there be Color! [link]
Let there be Color! [github]

refs
AI - Def.