# COMP 3200
# Artificial Intelligence

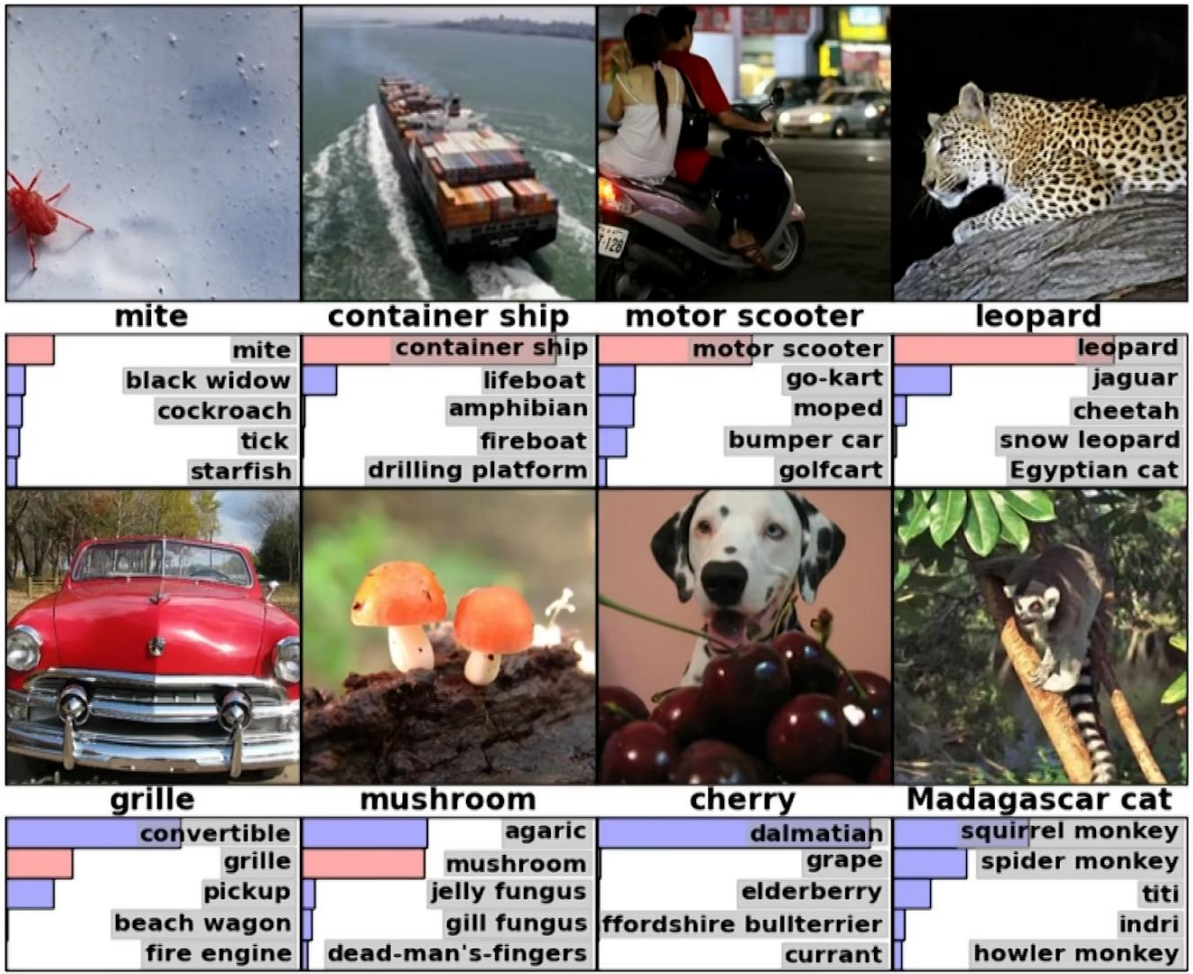

**Lecture 21**
Intro to Neural Networks

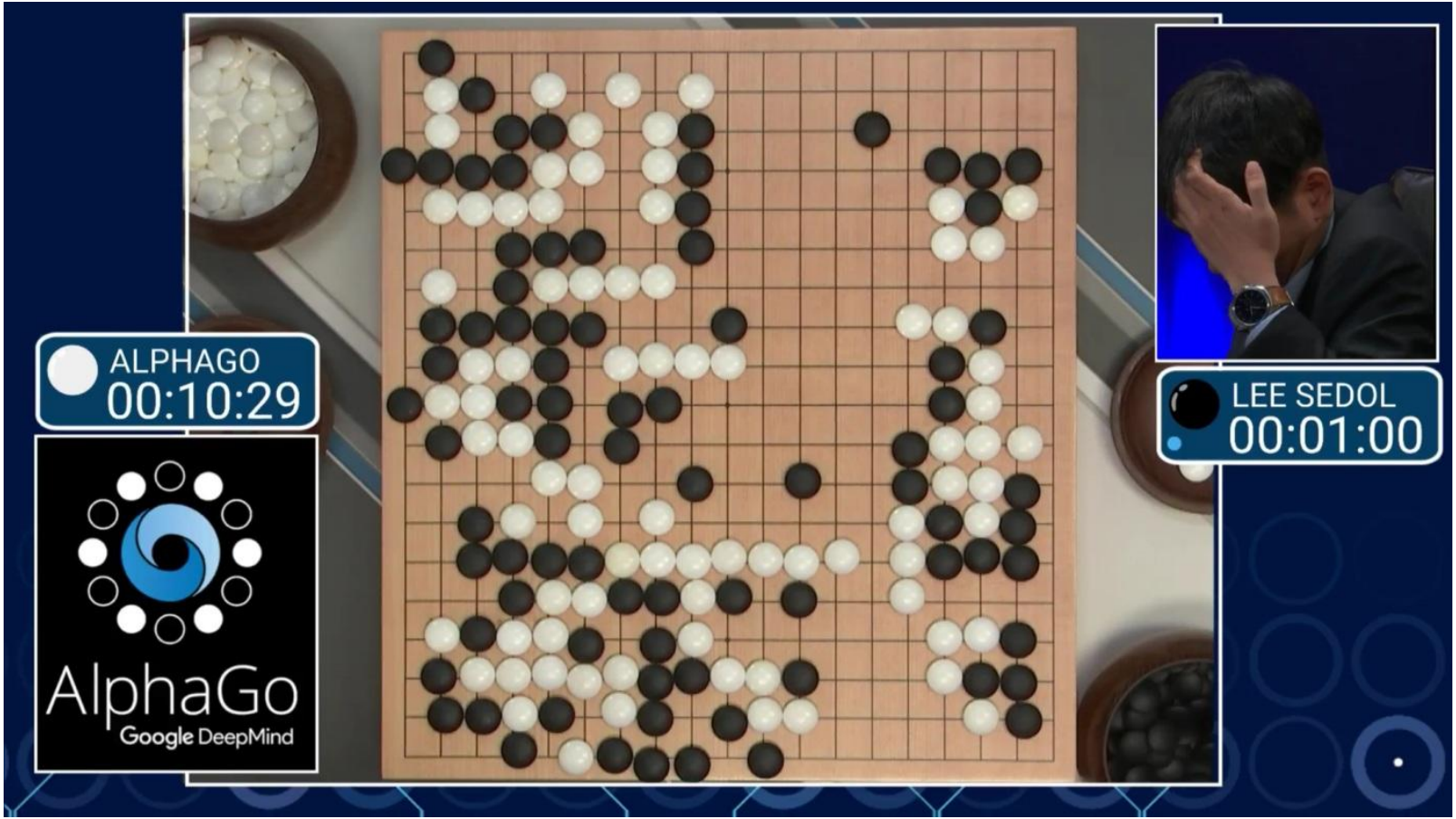
# Neural Networks

- In 2010, MIT was reviewing what topics it should drop from its AI course to make room for ‘more modern’ techniques
- Neural Networks were on the way out
- They decided to leave them in, just to show a possible model of the brain
- In almost 30 years of research, Neural Networks had failed to yield any significant results

# Deep Neural Networks

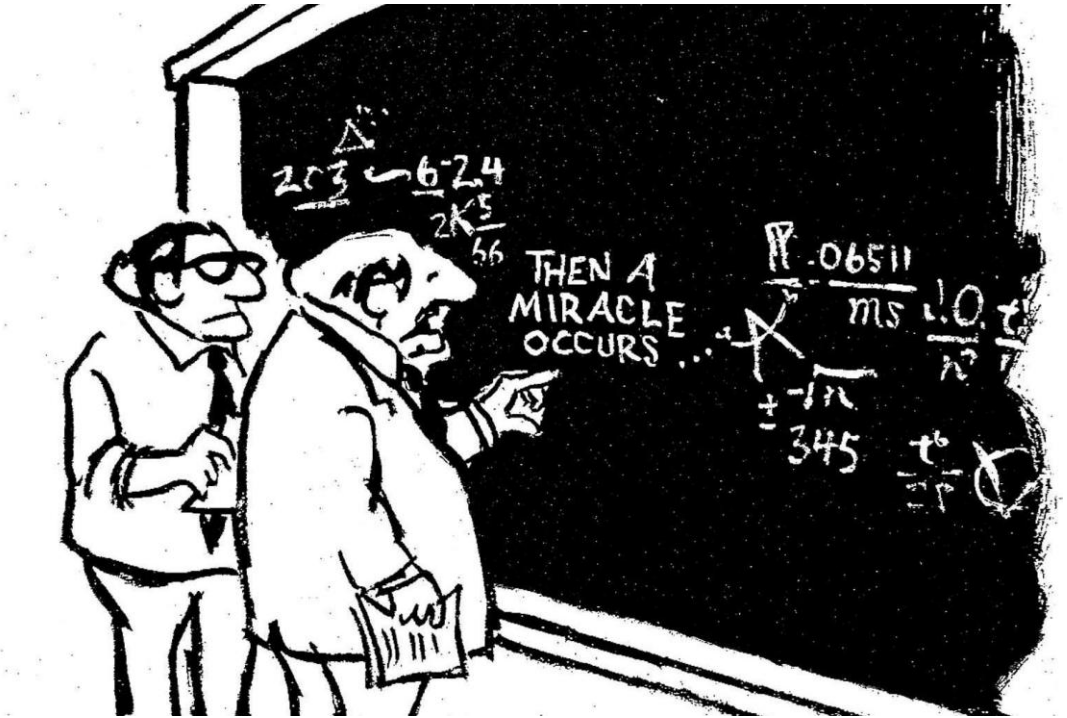
- Geoffrey Hinton, 2012
  - ImageNet Classification with Deep Convolutional Neural Networks
- 60,000,000 parameters in the network
- Purpose: which of 1000 categories best characterized a given picture
- Blew away the competition


mite
mite
black widow
cockroach
tick
starfish
container ship
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
motor scooter
go-kart
moped
bumper car
golfcart
leopard
leopard
jaguar
cheetah
snow leopard
Egyptian cat
grille
convertible
grille
pickup
beach wagon
fire engine
mushroom
agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers
cherry
dalmatian
grape
elderberry
ffordshire bullterrier
currant
Madagascar cat
squirrel monkey
spider monkey
titi
indri
howler monkey

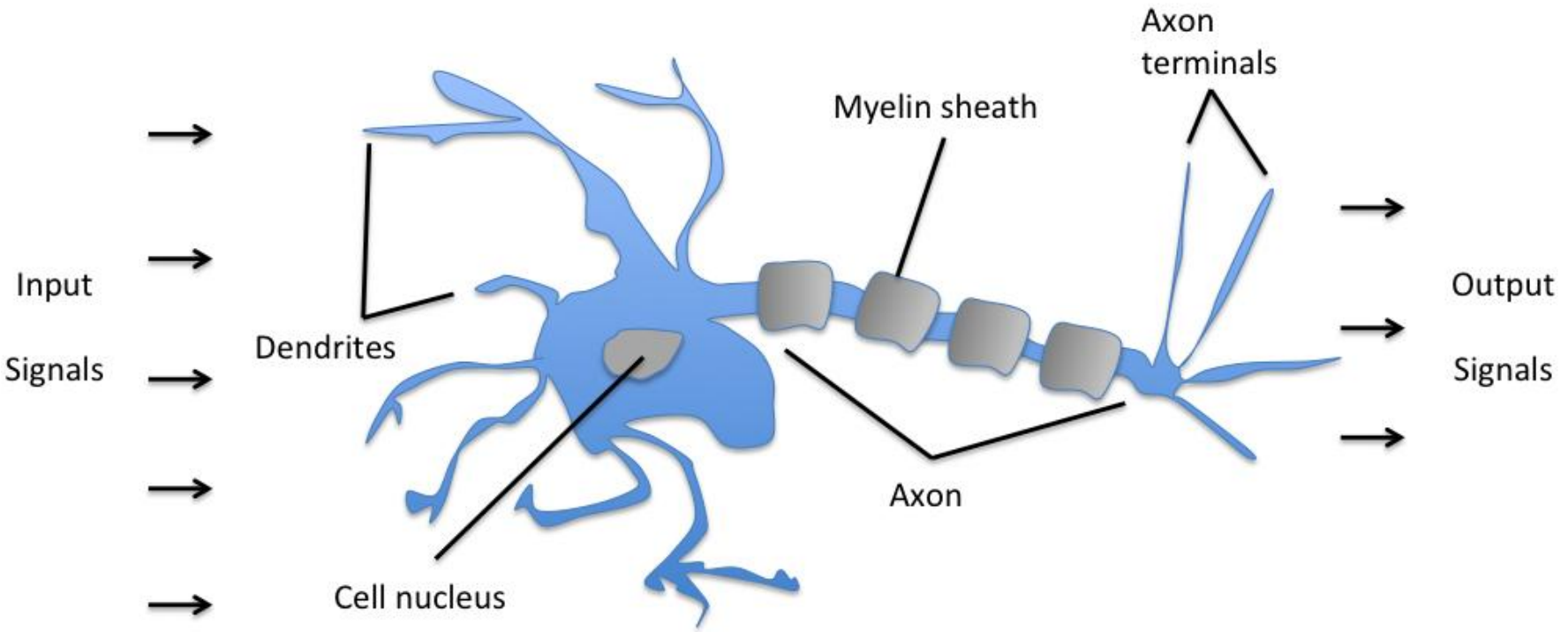

ALPHAGO
00:10:29
AlphaGo
Google DeepMind
LEE SEDOL
00:01:00


The Development of Campus Plans

# Neural Networks



| | Computer | Brain |
|---|---|---|
| **processing units** | 1 CPU, $10^9$ transistors | $10^{11}$ neurons |
| **storage capacity** | $10^9$ B RAM, $10^{12}$ B non-volatile memory | $10^{11}$ neurons, $10^{14}$ synapses |
| **processing speed** | $10^{-8}$ second | $10^{-3}$ second |
| **bandwidth** | $10^9$ bits/sec. | $10^{14}$ bits/sec. |



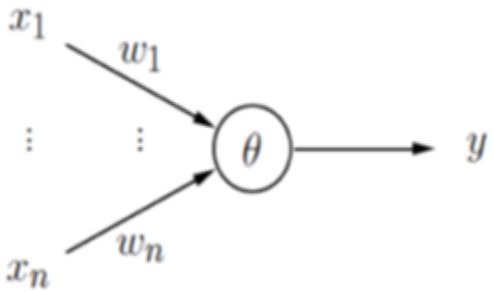

**Schematic of a biological neuron.**

# The Neuron

- Axons from one neuron extend to the dendrites of another neuron
- Stimulation of the dendritic tree of a neuron may cause the axon to ‘spike’, like a transmission line
- After firing, the neuron will go ‘quiet’ for a while (refractory period)
- Firing can affect surrounding / connected neurons, cause a chain reaction

# Threshold Logic Units (TLU)

- Inputs $x_1$, $x_2$, … $x_n$, binary (fire or don't fire)
- Weights $w_1$, $w_2$, …, $w_n$ (real values)
- Inputs multiplied by weights, and summed
- Output spike if result sum meets threshold

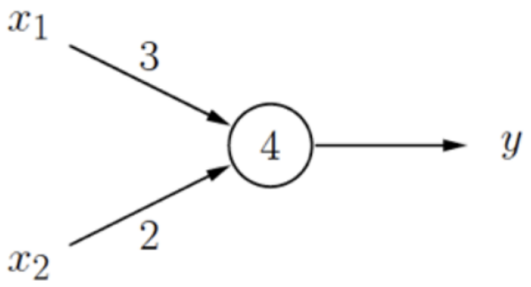



$$y = \begin{cases} 1, & \text{if} \quad \vec{x}\vec{w} = \sum_{i=1}^{n} w_i x_i \geq \theta, \\ 0, & \text{otherwise.} \end{cases}$$

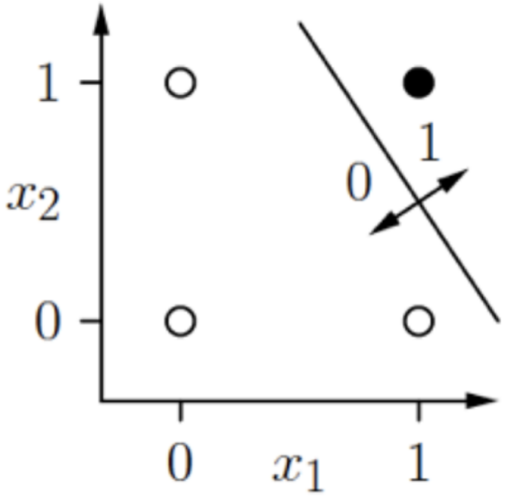
# Threshold Logic Units (TLU)

- By choosing different values for the weight vector, we can have a TLU compute the output of different functions
- Let’s look at some logic operators

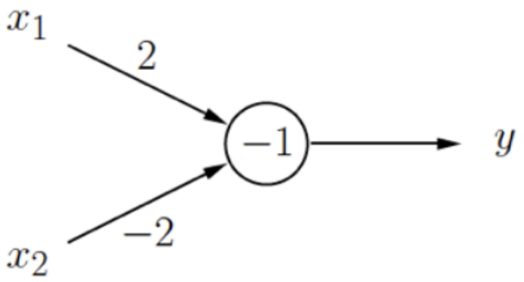
# TLU for the conjunction $x_1 \wedge x_2$



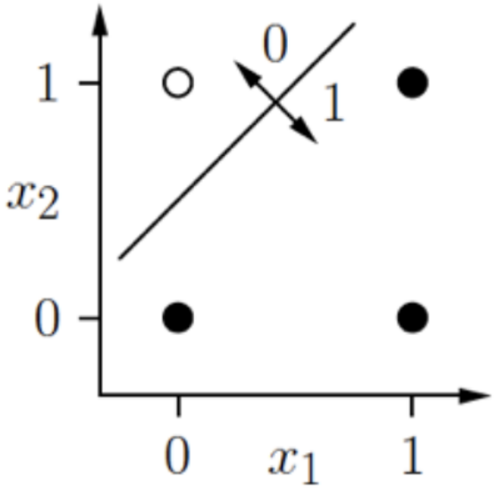

| $x_1$ | $x_2$ | $3x_1 + 2x_2$ | $y$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 3 | 0 |
| 0 | 1 | 2 | 0 |
| 1 | 1 | 5 | 1 |



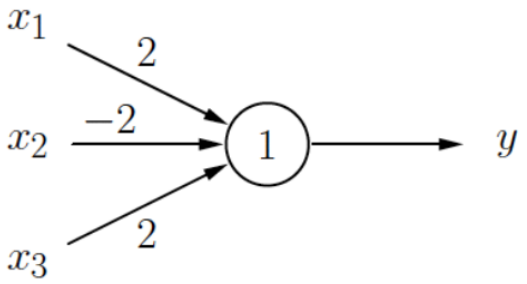
# TLU for the implication $x_2 \rightarrow x_1$



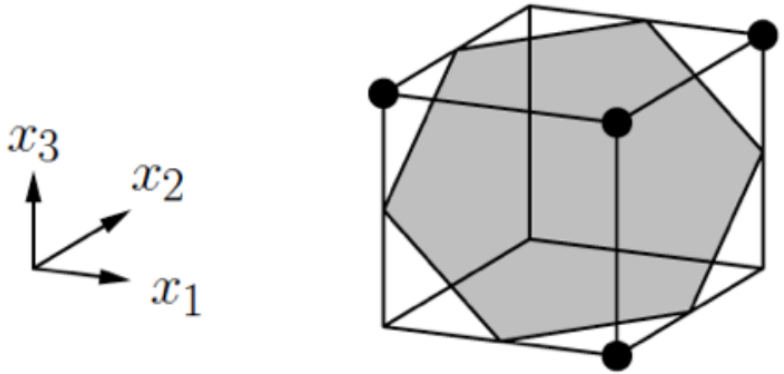

| $x_1$ | $x_2$ | $2x_1 - 2x_2$ | $y$ |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 1 | 0 | 2 | 1 |
| 0 | 1 | $-2$ | 0 |
| 1 | 1 | 0 | 1 |



# TLU for $(x_1 \wedge \overline{x_2}) \vee (x_1 \wedge x_3) \vee (\overline{x_2} \wedge x_3)$




| $x_1$ | $x_2$ | $x_3$ | $\sum_i w_i x_i$ | $y$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 2 | 1 |
| 0 | 1 | 0 | $-2$ | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 2 | 1 |
| 1 | 0 | 1 | 4 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 2 | 1 |



# TLU Linear Separability

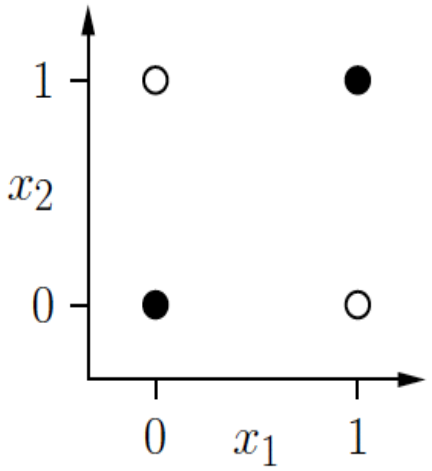
- We call two points in an n-dimensional space linear separable if they can be separated by an (n-1)-dimensional hyperplane. One of these sets may contain points on the hyperplane
- A Boolean function is called linearly separable if the set of points of 0 and the set of points of 1 are linearly separable

# Linear Separability

- Consider the bi-implication problem, in which there is no separation line

$x_1 \leftrightarrow x_2$

| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

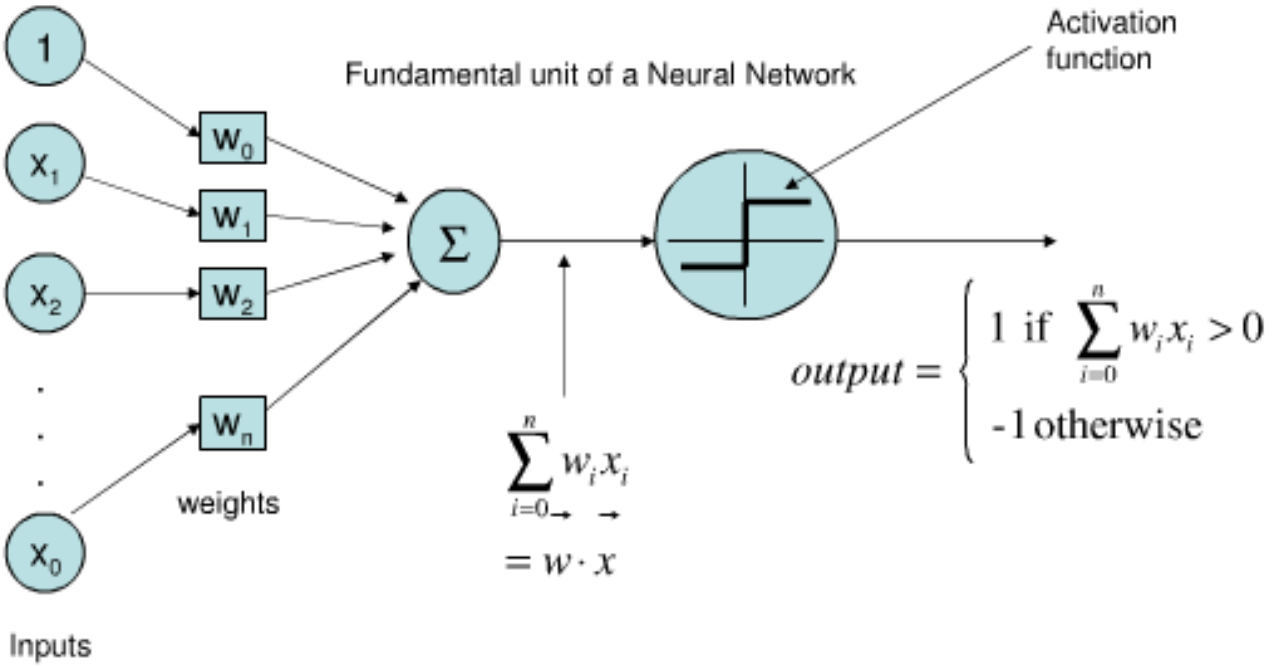


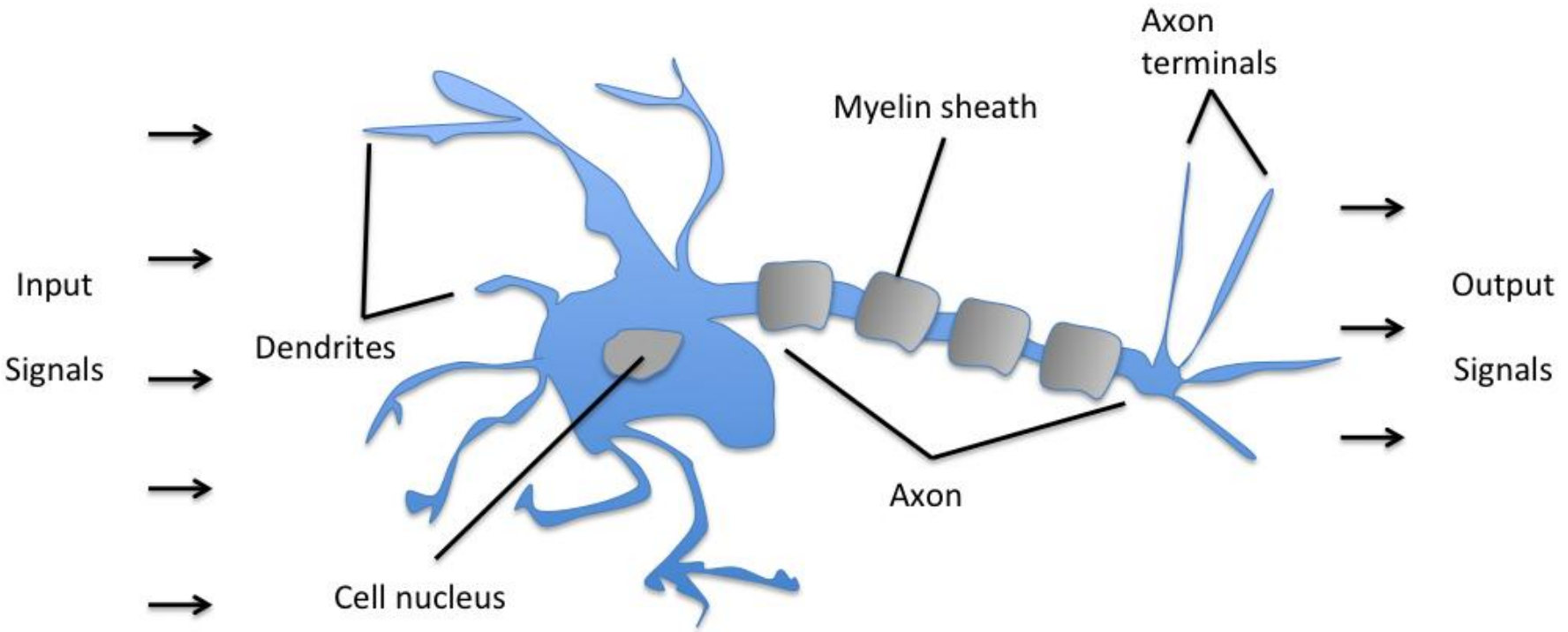
# Linear Separable Boolean Functions

| inputs | Boolean functions | linearly separable functions |
|---|---|---|
| 1 | 4 | 4 |
| 2 | 16 | 14 |
| 3 | 256 | 104 |
| 4 | 65536 | 1774 |
| 5 | $4.3 \cdot 10^9$ | 94572 |
| 6 | $1.8 \cdot 10^{19}$ | $5.0 \cdot 10^6$ |

- For many inputs, TLU can’t compute functions
- Networks of TLUs are needed to overcome this

# Perceptron (Simple Neuron)



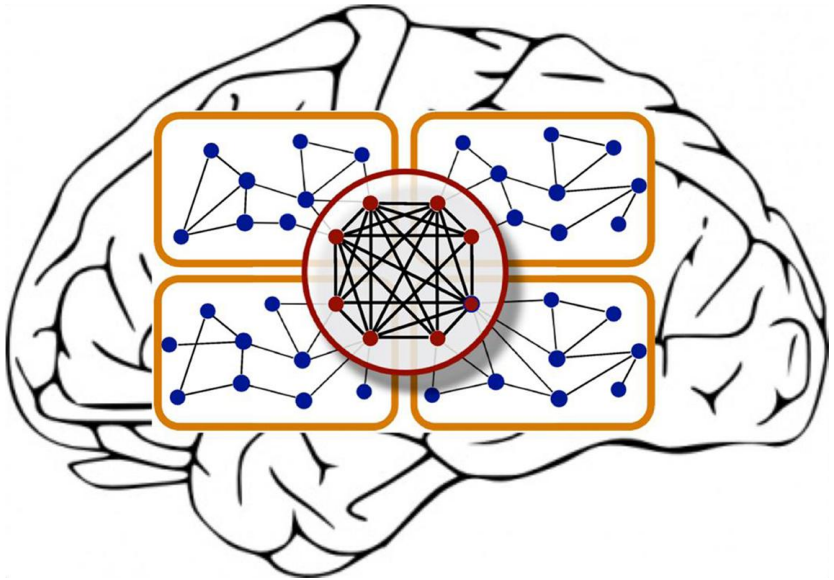



**Schematic of a biological neuron.**

# A Simplified Brain Model

Input

$x_1$
$x_2$
.
.
$x_n$

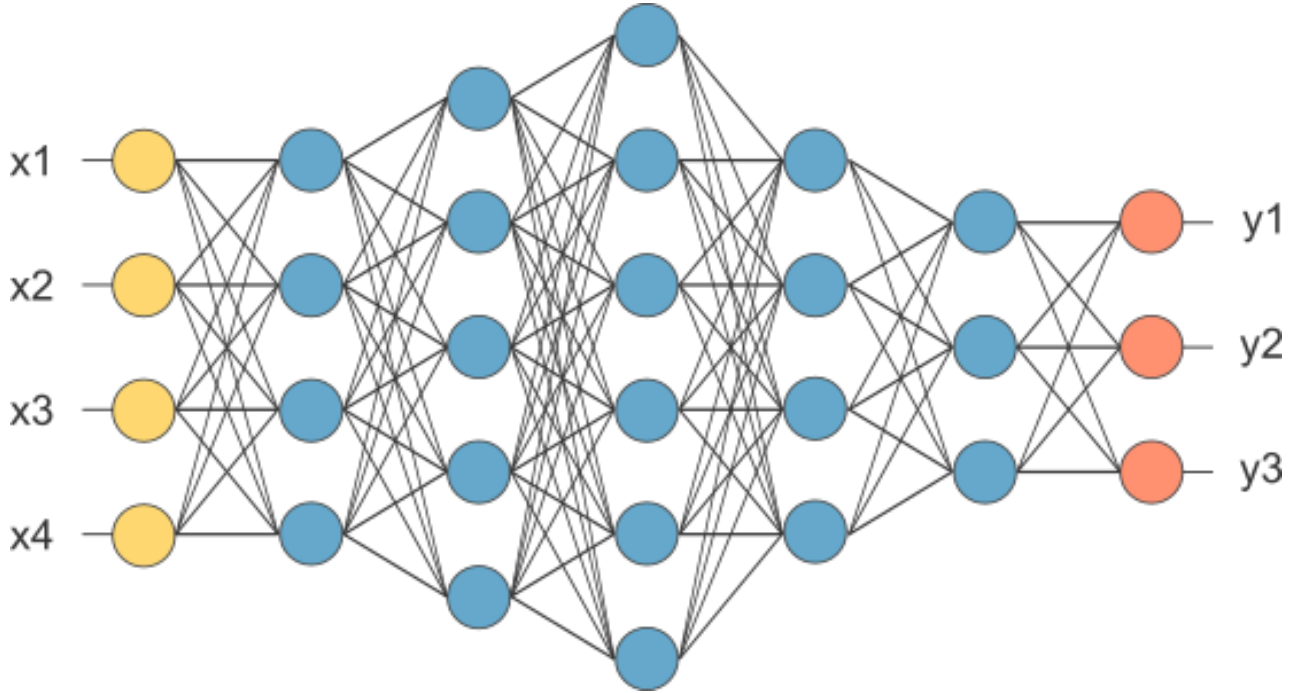


Output

$y_1$
$y_2$
.
.
$y_m$

$$\overline{y} = f(\overline{x}, \overline{w}, \overline{t})$$

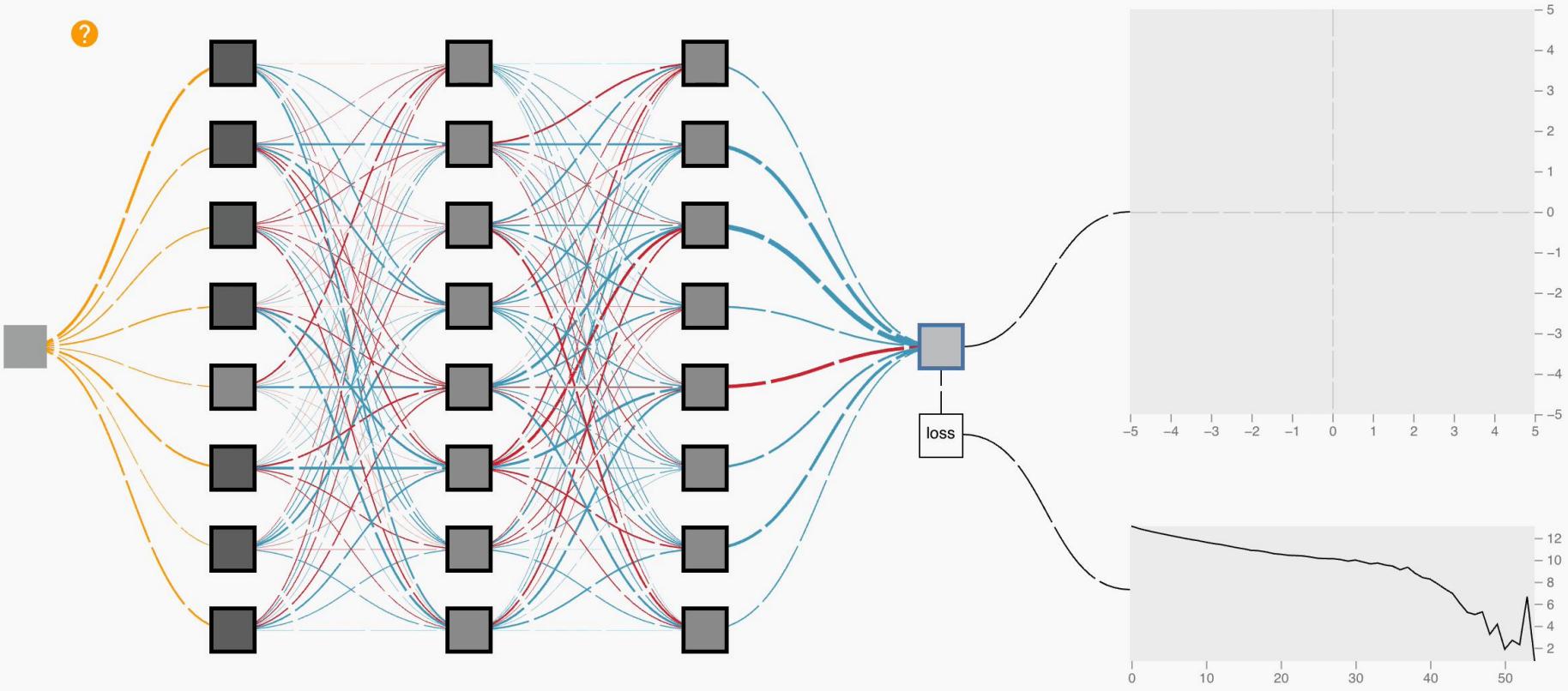
# Neural Network



# ‘Training’ a Neural Network

- Given input data and known outputs
- Running inputs through the network gives us a calculated value $\overline{y} = f(\overline{x}, \overline{w}, \overline{t})$ (feed forward)
- Training a neural network involves tuning the weights/thresholds until the values we get out of the network match our training data
- A neural network is a function approximator

# Back Propagation (Backprop)



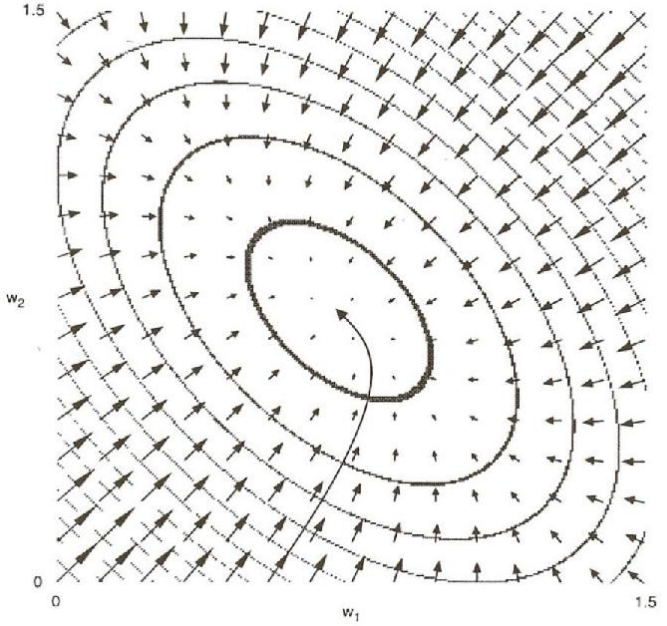

https://xnought.github.io/backprop-explainer/

# Training Progress

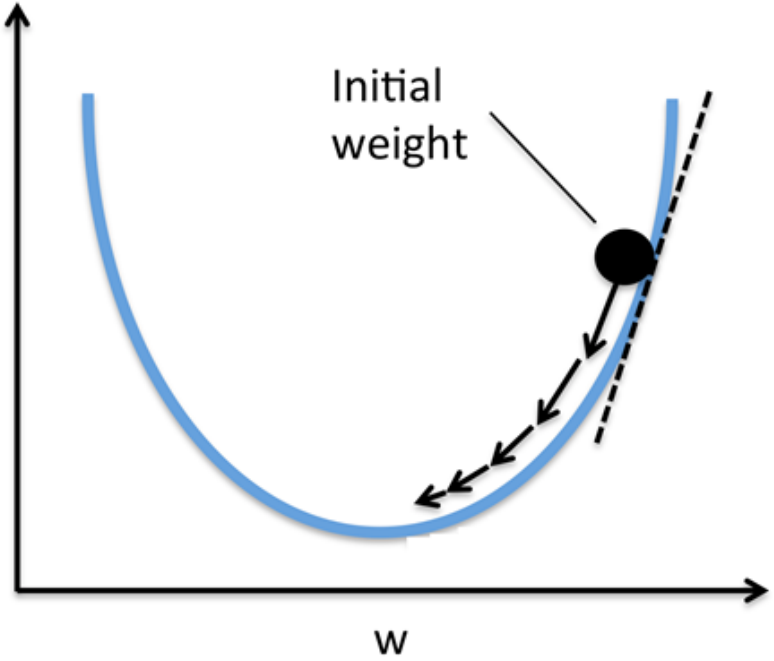
- Desired function: $\overline{d} = g(\overline{x})$
- NN function: $\overline{y} = f(\overline{x}, \overline{w}, \overline{t})$
- Performance should be a function of the desired $\overline{d}$ and the calculated $\overline{y}$
- How about vector distance? $P = \left|\overline{d} - \overline{y}\right|^2$
- Best possible performance would be P = 0
- Attempt to minimize error P

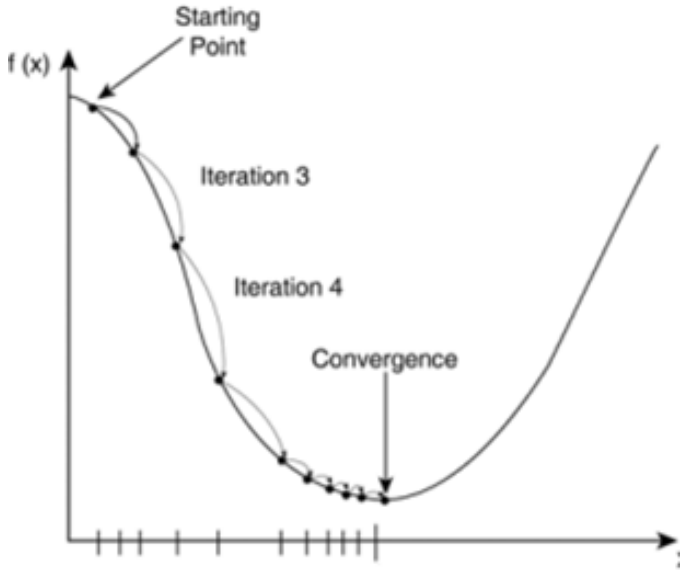
# Adjusting the Weights

- Weights $w_1$, $w_2$
- Performance contour
- At any given time we are at a given ($w_1$, $w_2$)
- Find the action that brings us toward the optimal weights (min err)

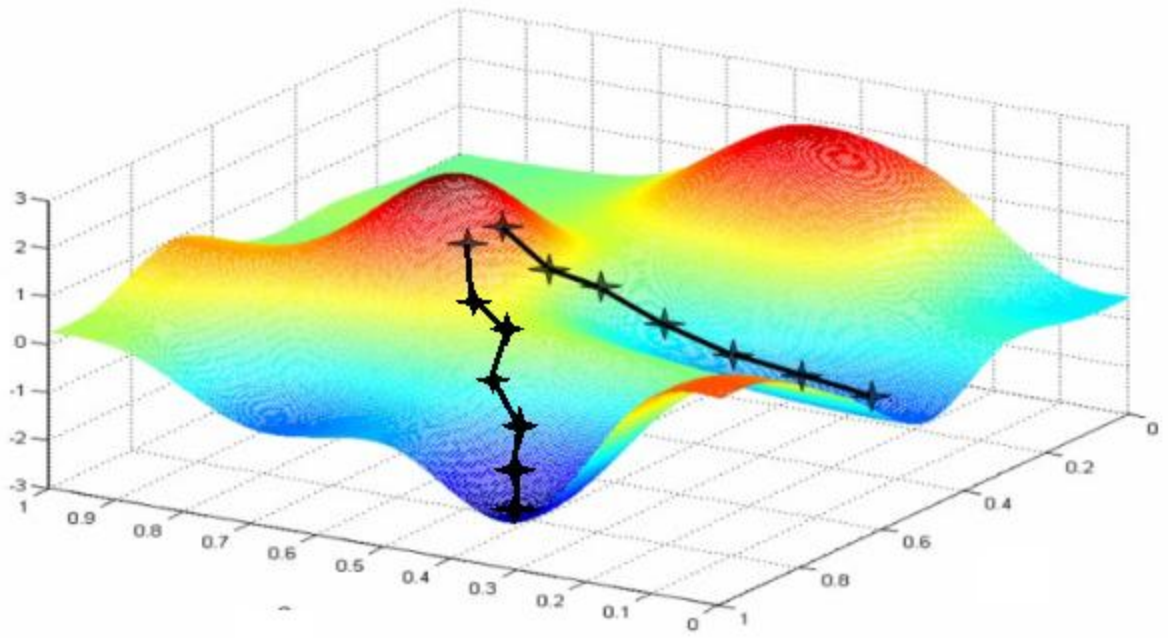


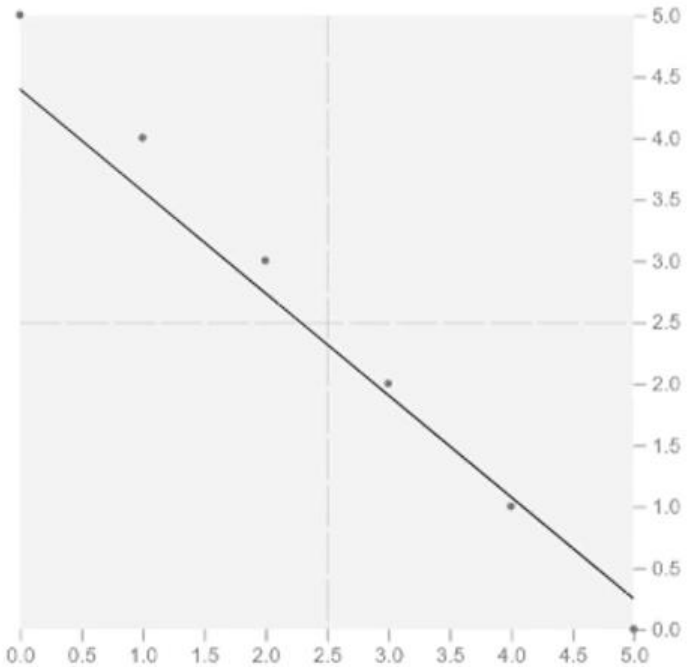
# Gradient Descent

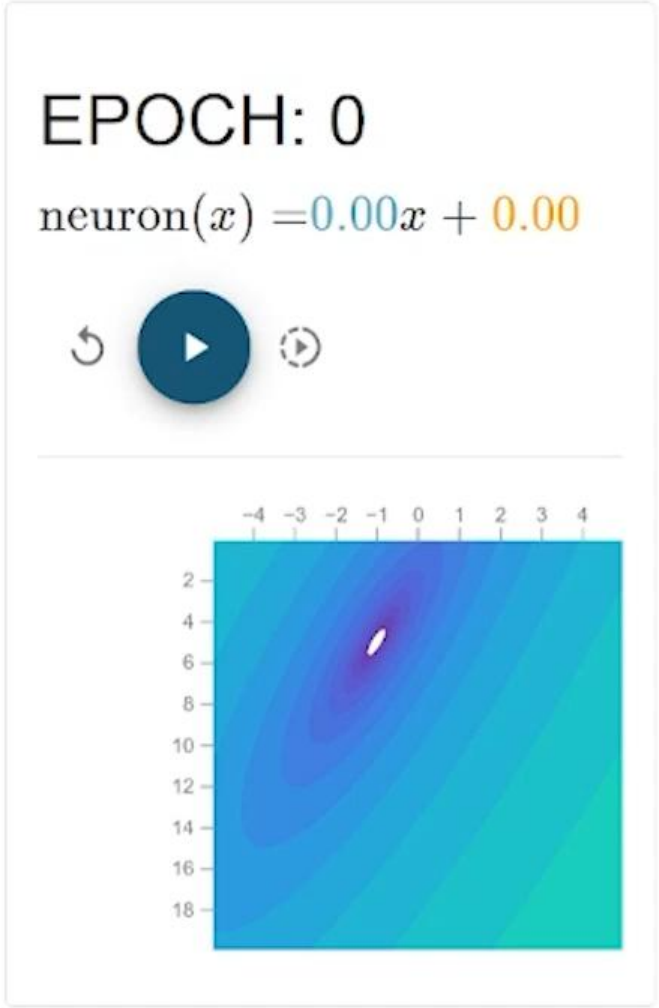


# Gradient Descent

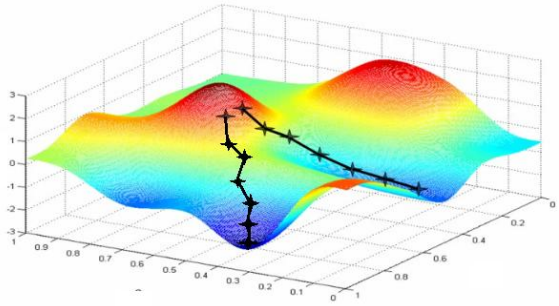




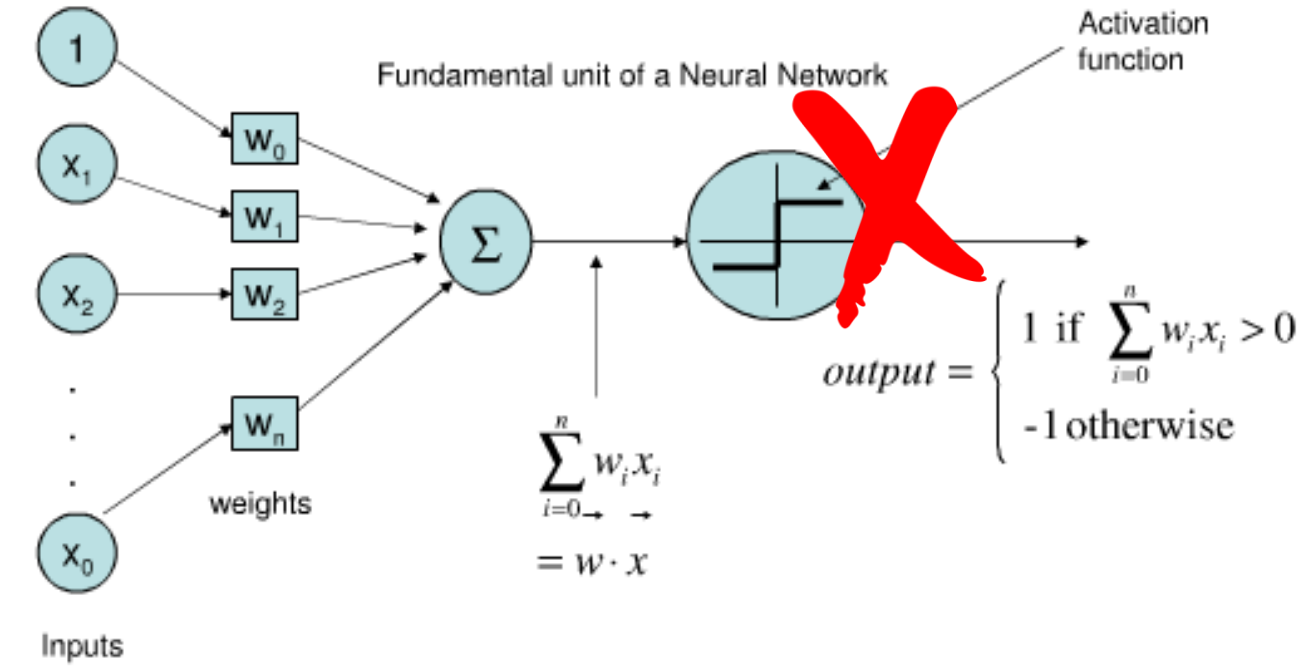

https://xnought.github.io/backprop-explainer/

# Gradient Descent



- Derivative $\frac{\partial P}{\partial w}$
- Update $\Delta w = r\frac{\partial P}{\partial w}$
- Weight(s) $\Delta \overline{w} = r\left(\frac{\partial P}{\partial w_1} i + \frac{\partial P}{\partial w_2} \mathrm{j}\right)$

- Obstacle: When does this work?
  - Only when P is differentiable


Fundamental unit of a Neural Network
Activation function
1
$x_1$
$x_2$
$x_0$
Inputs
$w_0$
$w_1$
$w_2$
$w_n$
weights
$\Sigma$
$\sum_{i=0}^{n} w_i x_i$
$= \vec{w} \cdot \vec{x}$
$output = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

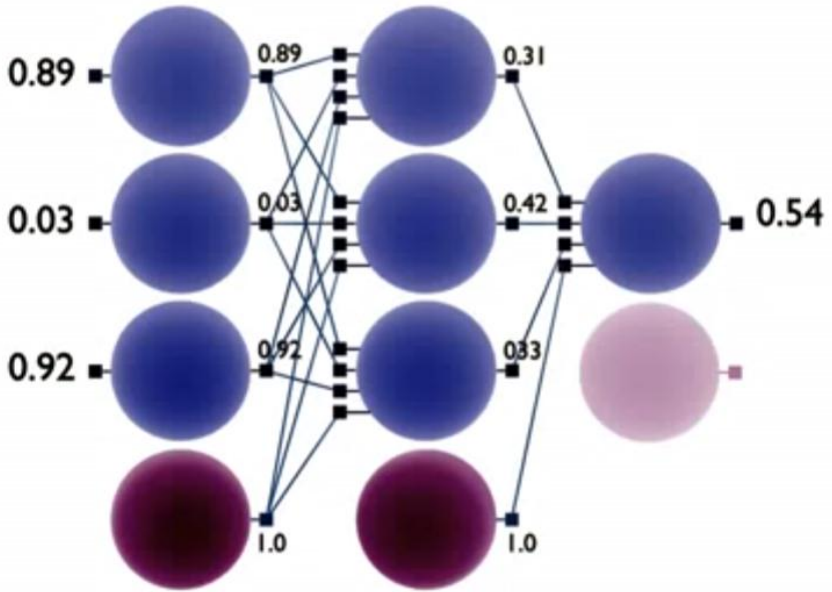
# Step 1: Removing Threshold

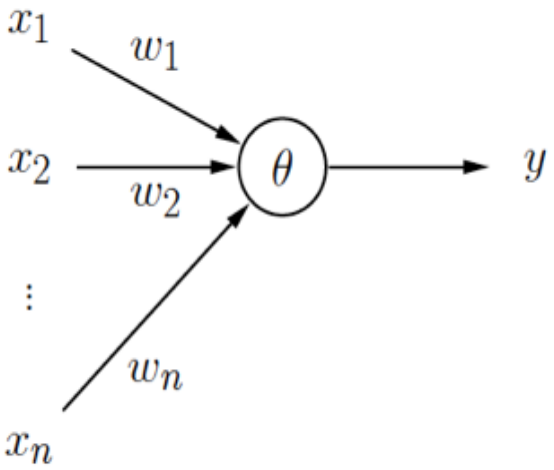
- Thresholding makes our function non-differentiable, also annoying to compute
- Ideally, we want $\overline{y} = f(\overline{x}, \overline{w})$

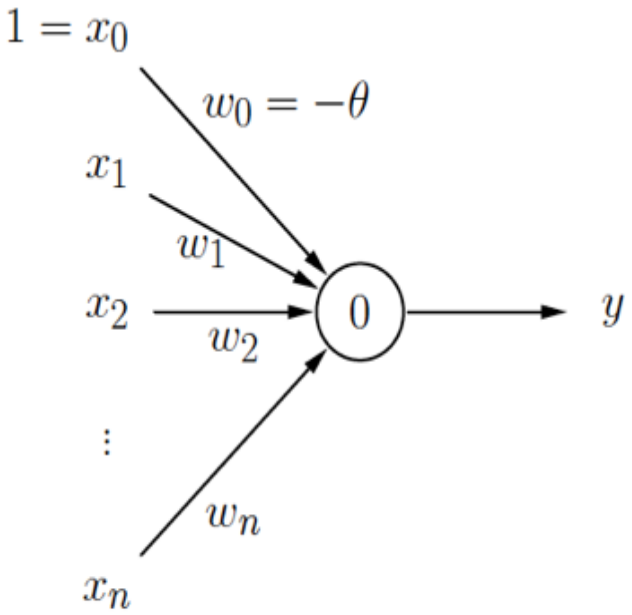
- Enter the Bias Neuron

# Bias Neuron

- An extra neuron added to each layer
- Has fixed output value of 1.0
- Has effect similar to that of thresholding with easier compute





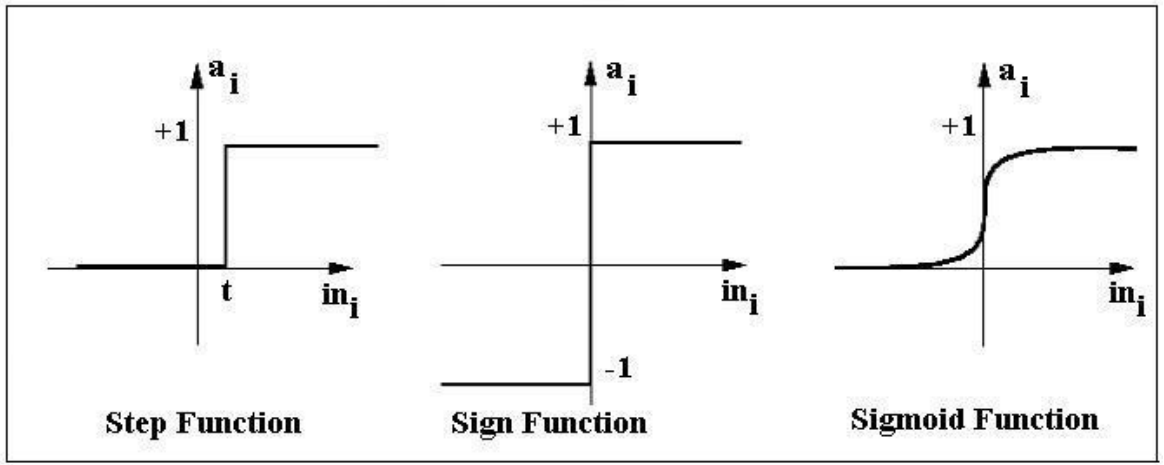




$$\sum_{i=1}^{n} w_i x_i \geq \theta$$

$$\sum_{i=1}^{n} w_i x_i - \theta \geq 0$$

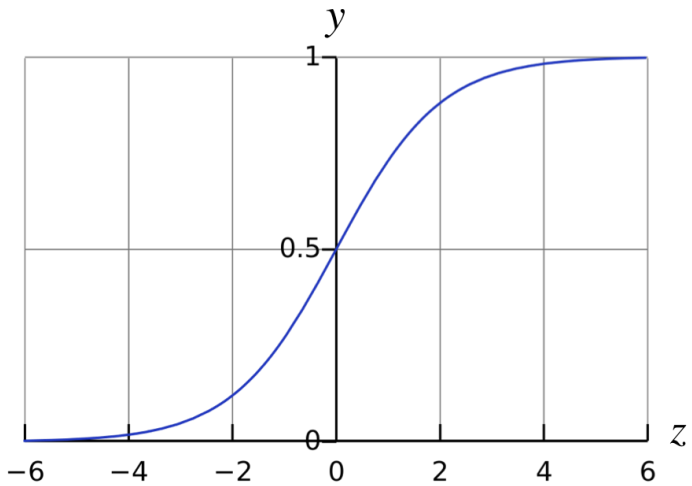
# Step 2: Smoothing the Activation

- Step function non-differentiable
- Apply a sigmoid function to smooth it out



# Step 2: Smoothing the Activation

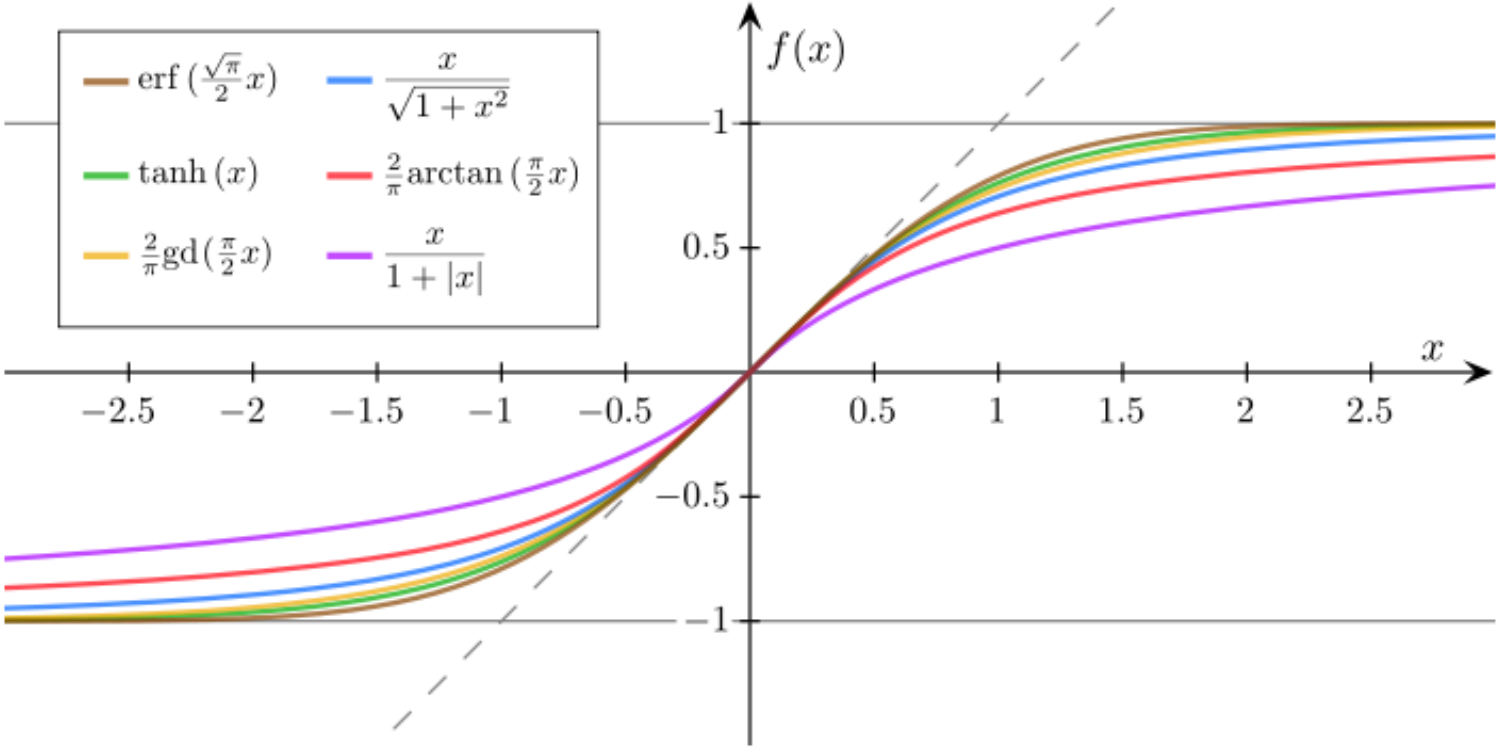
- Step function non-differentiable
- Apply a sigmoid function to smooth it out



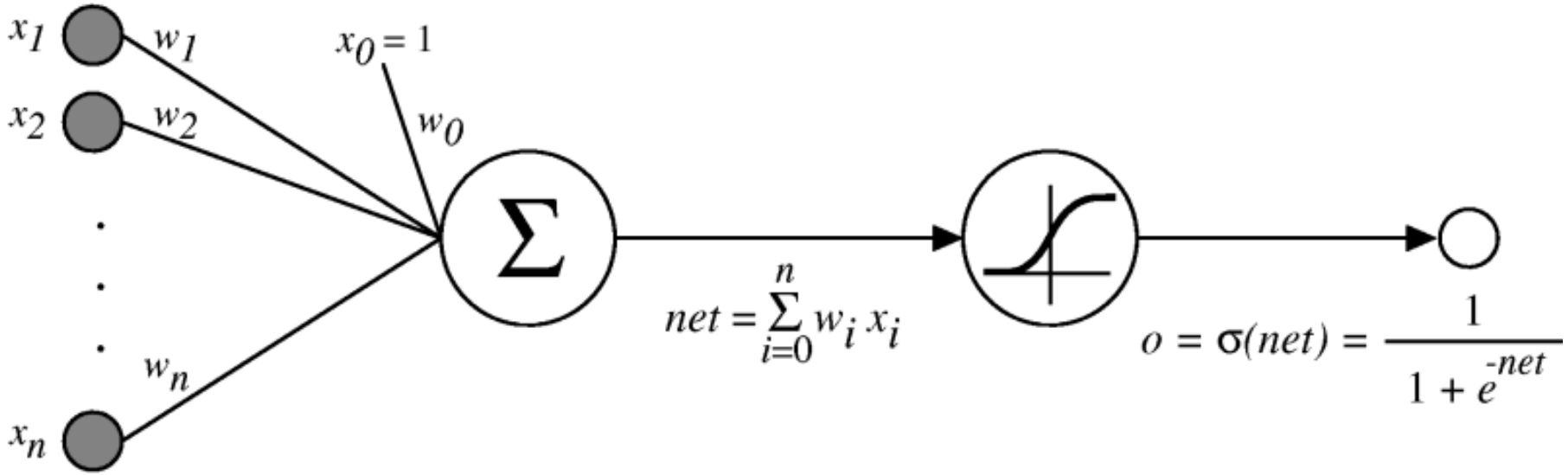

$$z = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

$$y = \frac{1}{1 + e^{-z}}$$

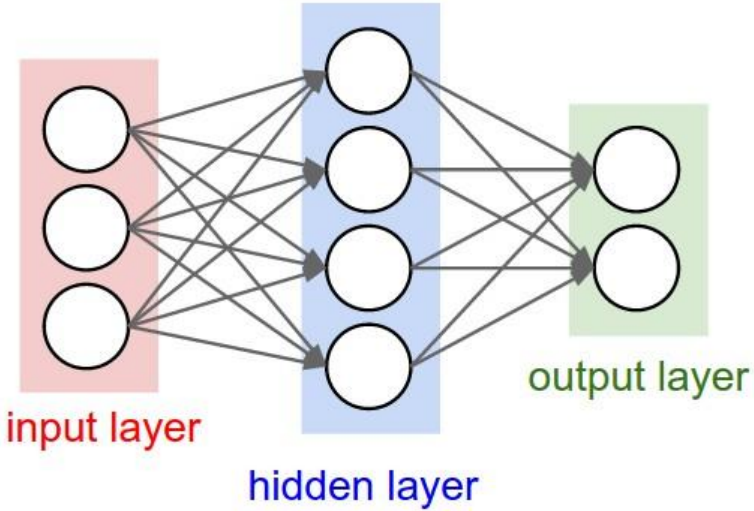
# Sigmoid Functions

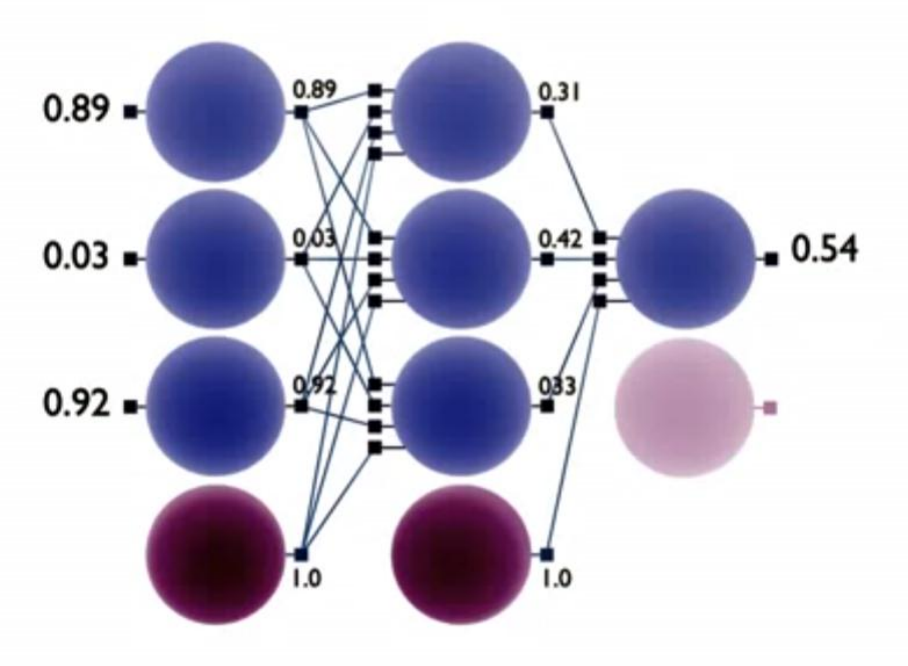


# The Improved Neuron



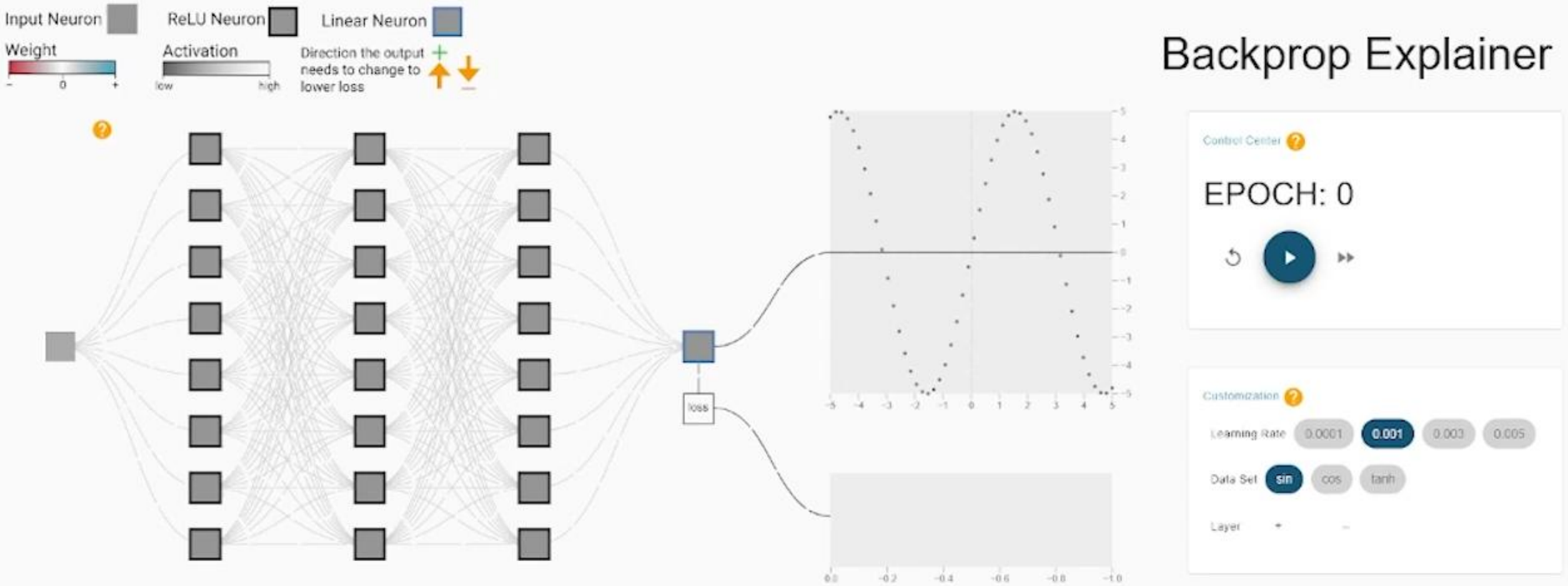
# Neural Net Structure



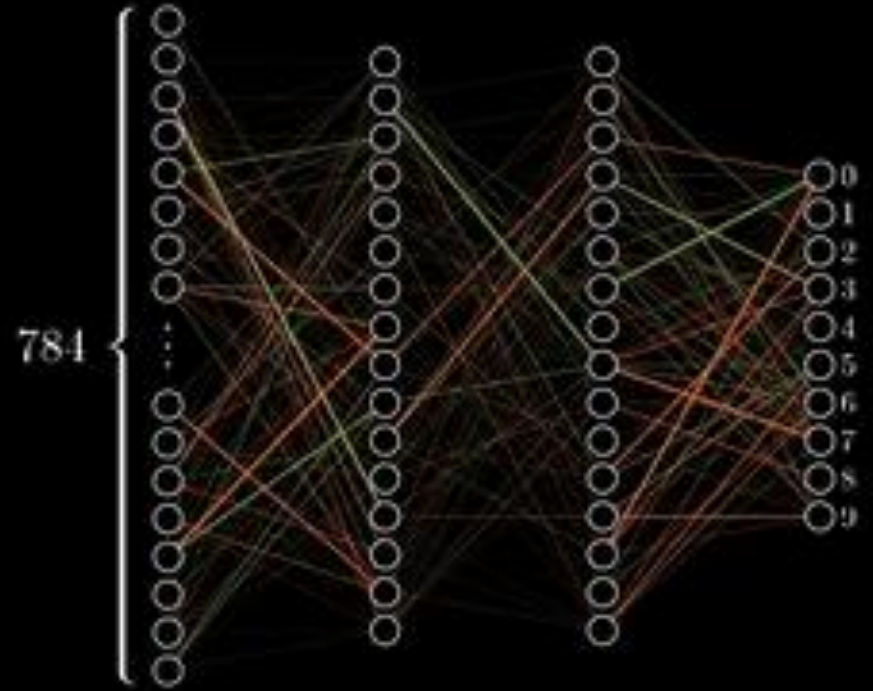
# Neural Net Structure

- Common Structure: Fully Connected
- If we start with a fully connected network, then unimportant connections will eventually train to a low weight
- Network can have an arbitrary structure
- Some research has learned net structure via genetic algorithms




https://xnought.github.io/backprop-explainer/

# Training in progress...



# Neural Network Implementation in C++

https://youtu.be/sK9AbJ4P8ao

# Neural Net Demo

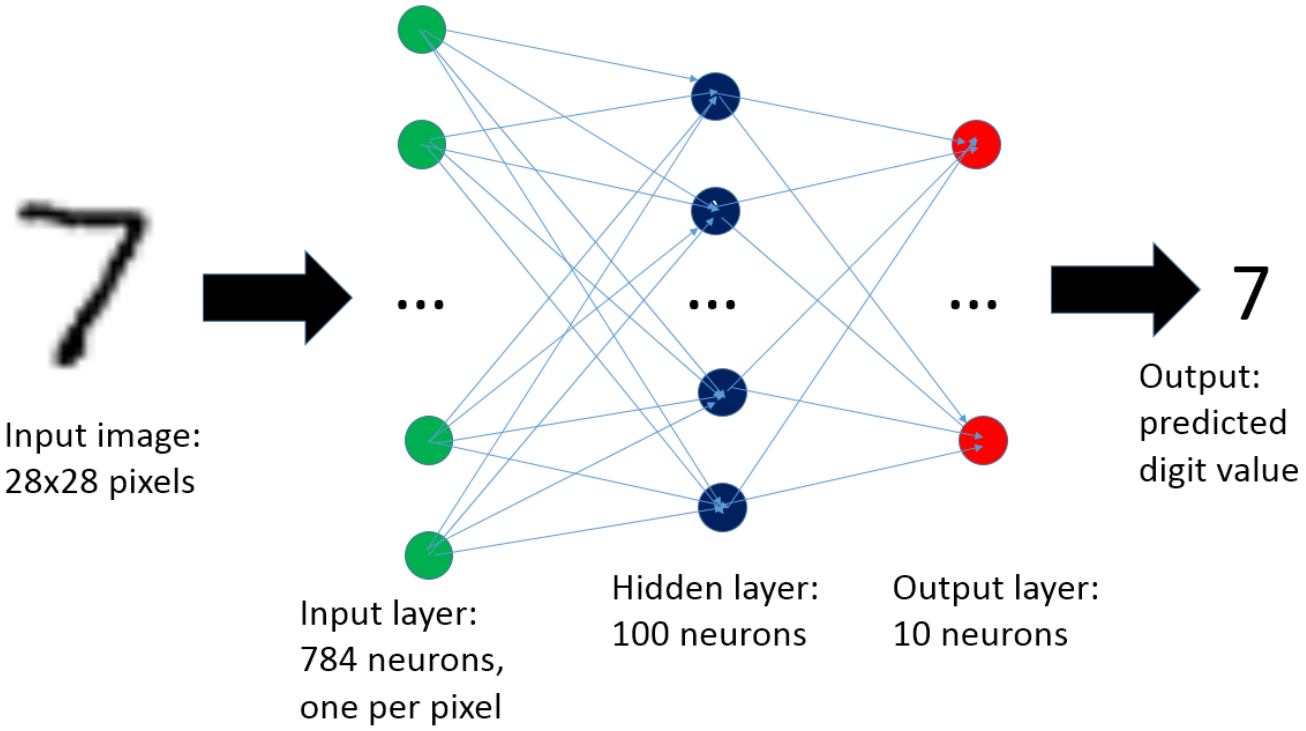
http://playground.tensorflow.org/

# Neural Network Applications

- Neural Networks are a method for performing function approximation

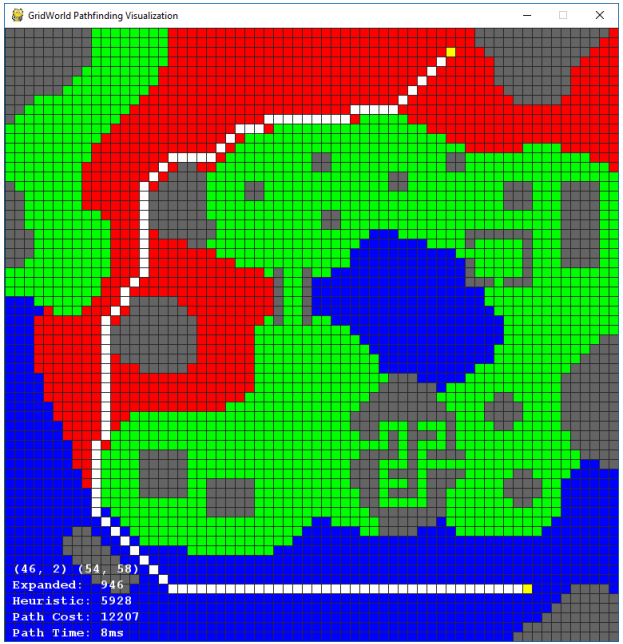
- Given a number of inputs, and desired outputs (training data), try and learn the function that computes correct values

# Image Classification



# Learning Heuristics

- Compute all pairs shortest path distances on map
- Neural Network
  - Inputs = (x, y) location
  - Target = Path Distance
- Train neural network, then use it as the heuristic function for future calls for A*



# AlphaGo

- Combined Deep Learning with Heuristic Search (MCTS)

- Two Networks were trained:
  - Value Network (how good is state S)
  - Policy Network (what move to do at state S)

# Learning from Replays

- DeepMind had thousands of professional human games to learn from
- Policy Network
  - Input = State
  - Target = Move the expert human performed
- Value Network
  - Input = State
  - Target = Who won the game from that state

# Learning from Self Play

- Once it had learned all it could from humans, it played against itself
- Continued to learn, and get stronger
- Now it learns from a stronger player each time it plays a new game
- Also key: throw in some random moves to learn about states you may not visit

# Heuristic Search

- Go has a large branching factor (300)
- Humans have good abstractions of what moves to perform at a state
- AlphaGo learned a policy network
- Now it can narrow the heuristic search to only consider moves in the policy network
- Search also needs a heuristic function to determine how good a state is (value network)

# Exam Questions

- Explain and compute the output for a TLU
- Explain linear separability (LS) and compute whether a simple function is LS
- Describe the structure of a Neuron / NN
- Implementation of NN / AlphaGo Videos
- Will not ask any gradient descent