Today: Outline

- **Neural networks:** artificial neuron, MLP, sigmoid units; neuroscience inspiration, output vs hidden layers; linear vs nonlinear networks;

- **Feed-forward networks**

- Reminders: PS1 grades are due Feb 19 *(today)*
  Pre-lecture Material for Feb 21
  PS2 is due Feb 24

Introduction to Neural Networks

Motivation

Recall: Logistic Regression

$$0 \leq h_\theta(x) \leq 1$$

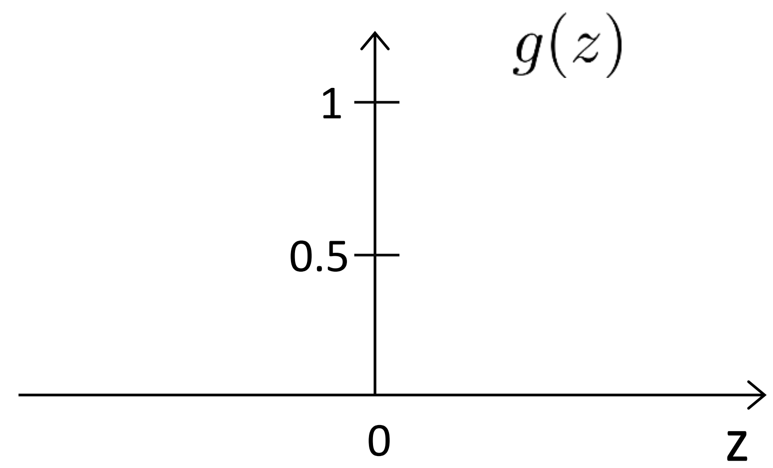
$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

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

Output is probability of label 1 given input

$$p(y = 1|x) = \frac{1}{1 + e^{-\theta^T x}}$$

sigmoid/logistic function



predict "$y = 1$" if $h_\theta(x) \geq 0.5$

predict "$y = 0$" if $h_\theta(x) < 0.5$

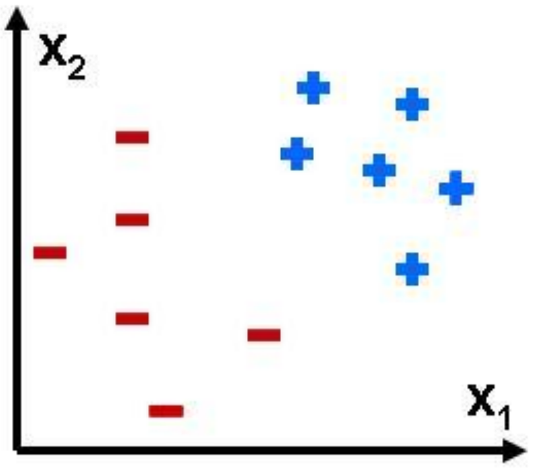
Recall: Logistic Regression Cost

Logistic Regression Hypothesis:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

θ: parameters

$D = \{x^i, y^i\}$: data



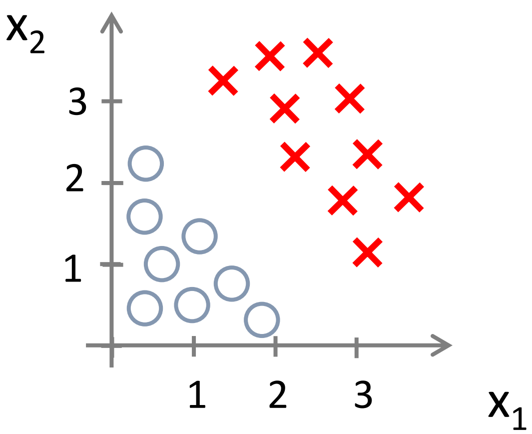
Logistic Regression Cost Function:

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

$$= -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

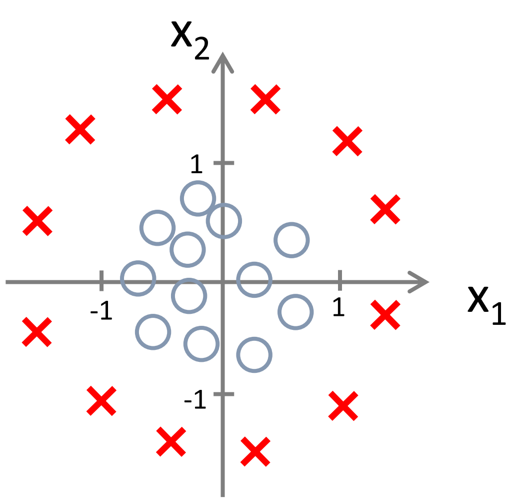
Goal: minimize cost $\min_\theta J(\theta)$

Decision boundary



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

Non-linear decision boundaries



Replace features with non-linear functions
e.g. log, cosine, or polynomial

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

Limitations of linear models

- Logistic regression and other linear models cannot handle nonlinear decision boundaries
  - Must use non-linear feature transformations
  - Up to designer to specify which one

- Can we instead learn the transformation?
  - Yes, this is what neural networks do!

- A Neural network chains together many layers of "neurons" such as logistic units (logistic regression functions)

Neural Networks learn features

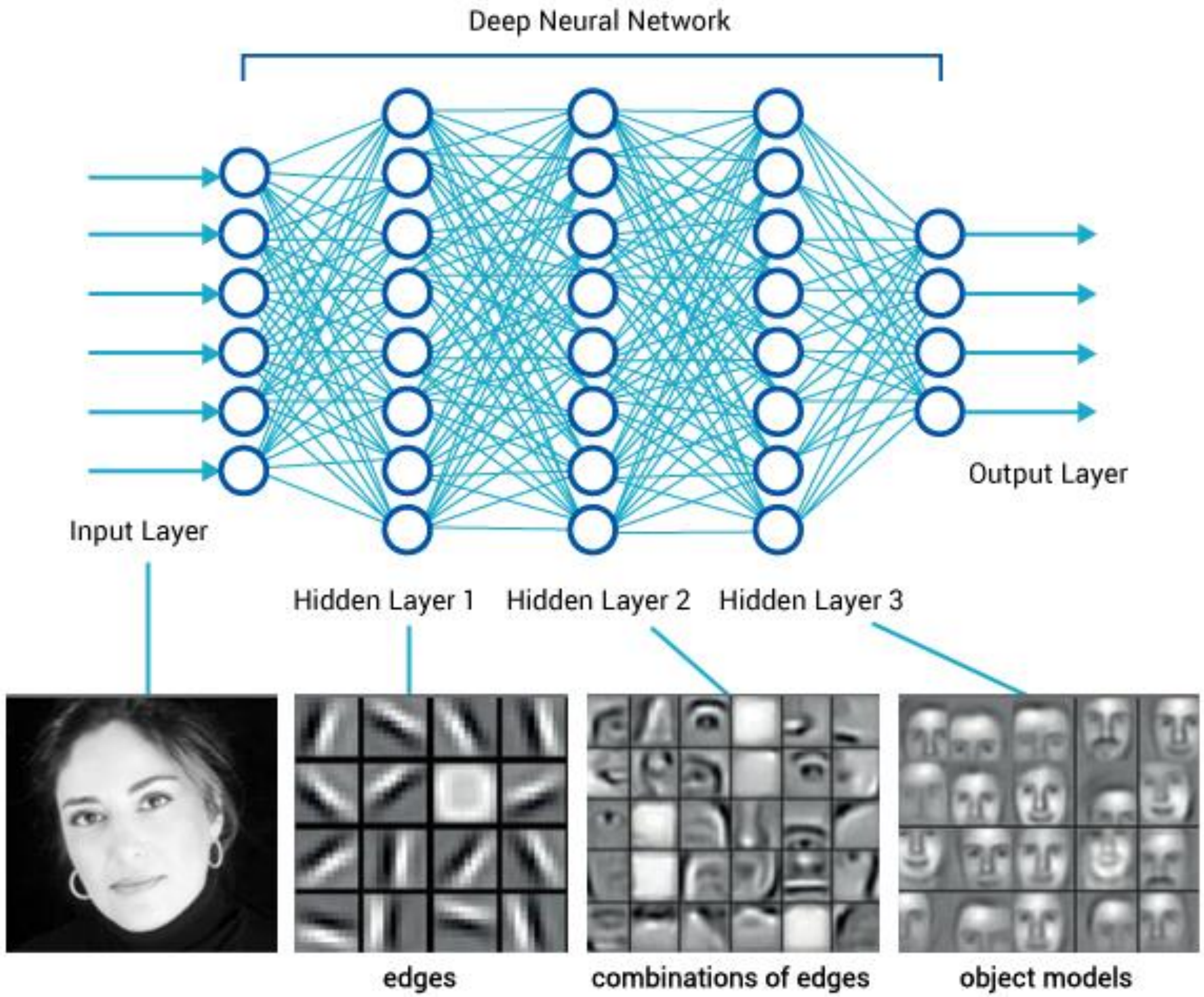
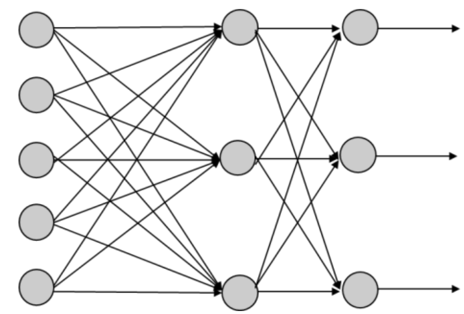


Image: http://www.amax.com/blog/wp-content/uploads/2015/12/blog_deeplearning3.jpg

Neurons in the Brain

Inspired "Artificial Neural Networks"

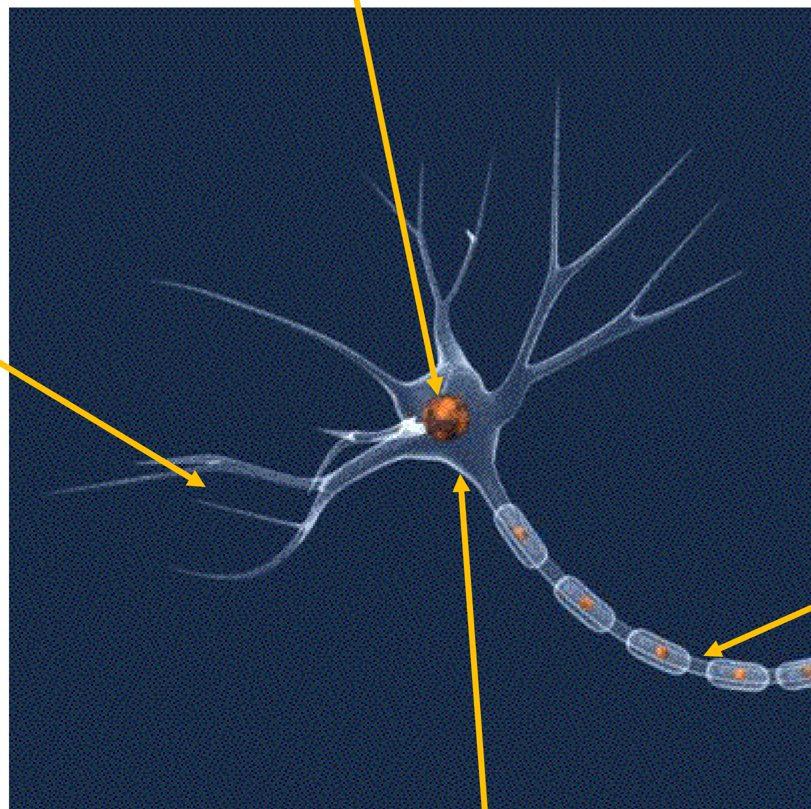


Neurons are cells that process chemical and electrical signals and transmit these signals to neurons and other types of cells

Neuron in the brain

nucleus

dendrites
“Input wire”



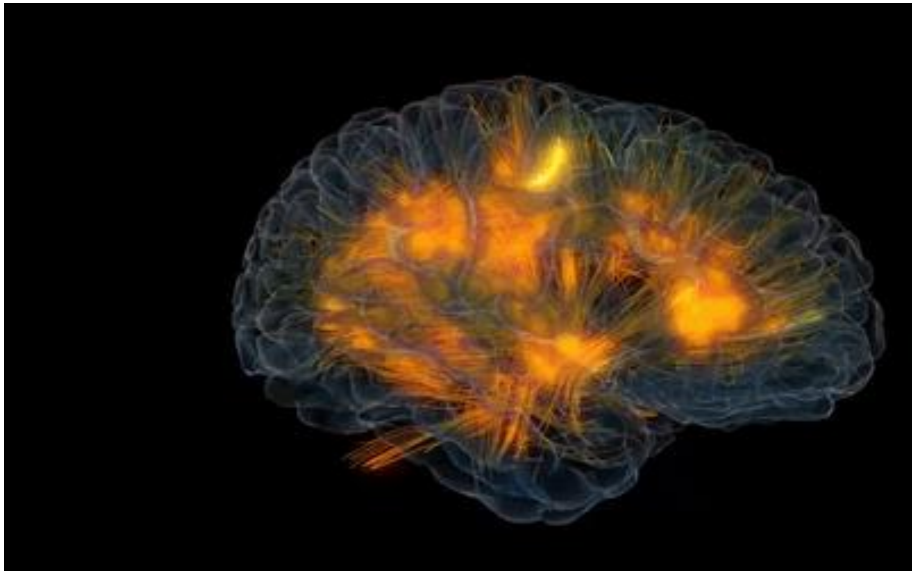
Can measure electrical activity (spike) of a single neuron by placing electrodes

axon
“Output wire”

Cell body

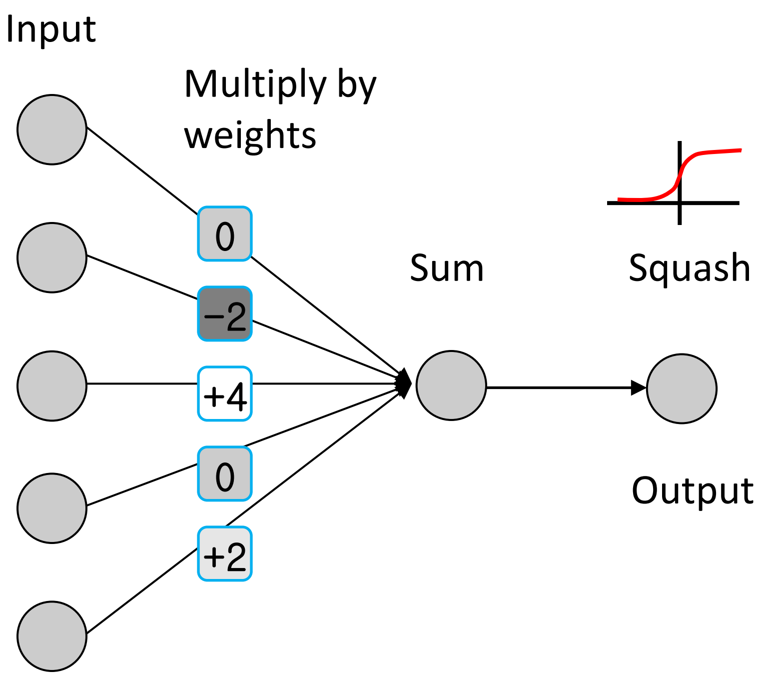
Image: http://webspace.ship.edu/cgboer/neuron.gif

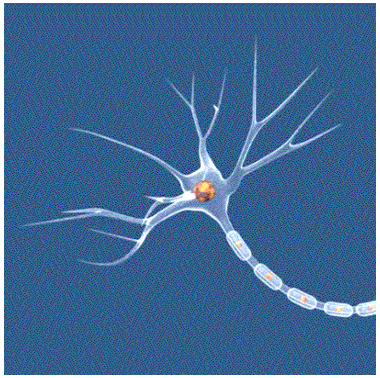
Neural network in the brain



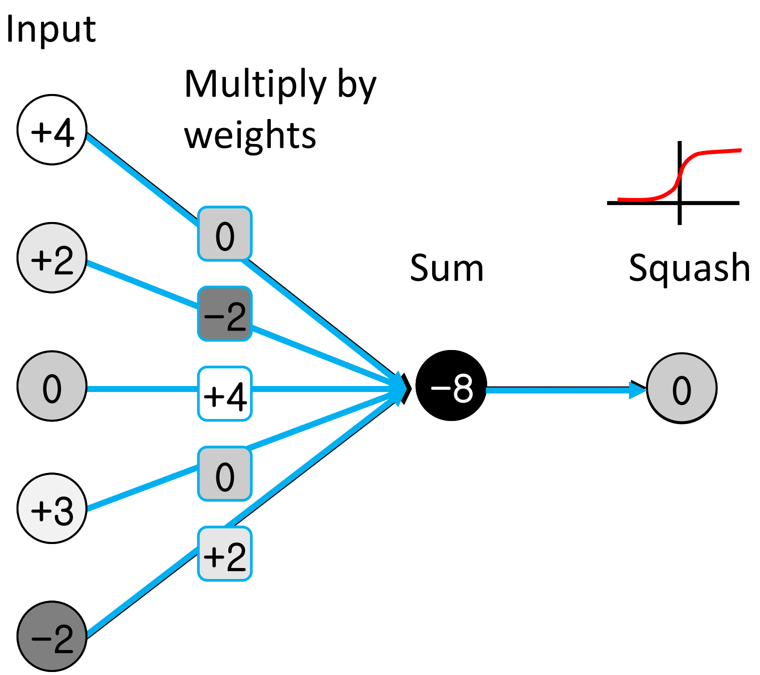
- **Micro networks**: several connected neurons perform sophisticated tasks: mediate reflexes, process sensory information, generate locomotion and mediate learning and memory.
- **Macro networks**: perform higher brain functions such as object recognition and cognition.

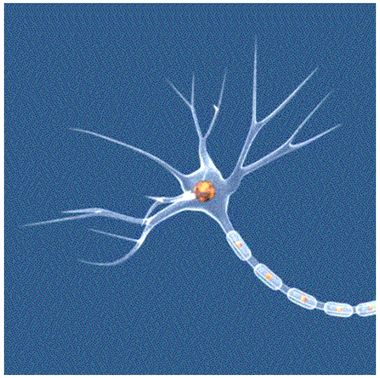
Logistic Unit as Artificial Neuron



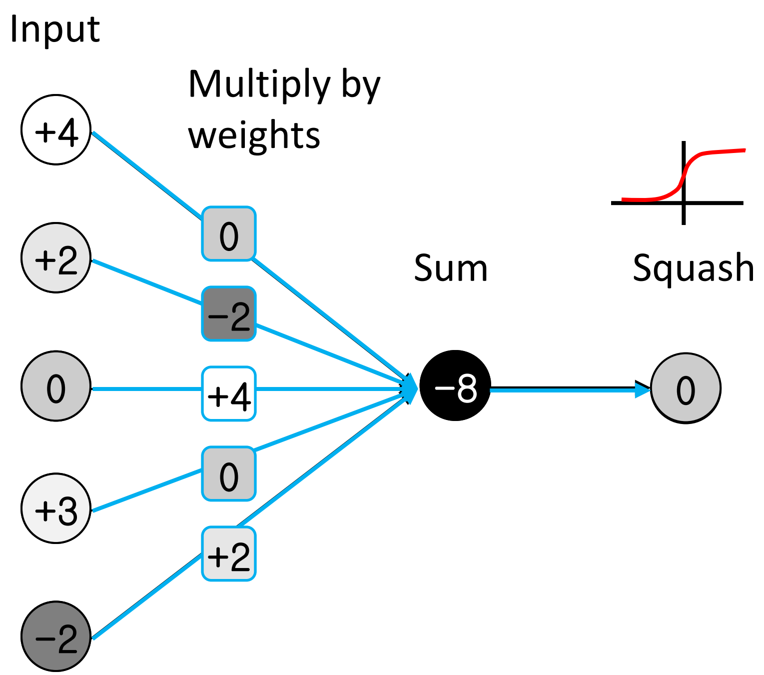


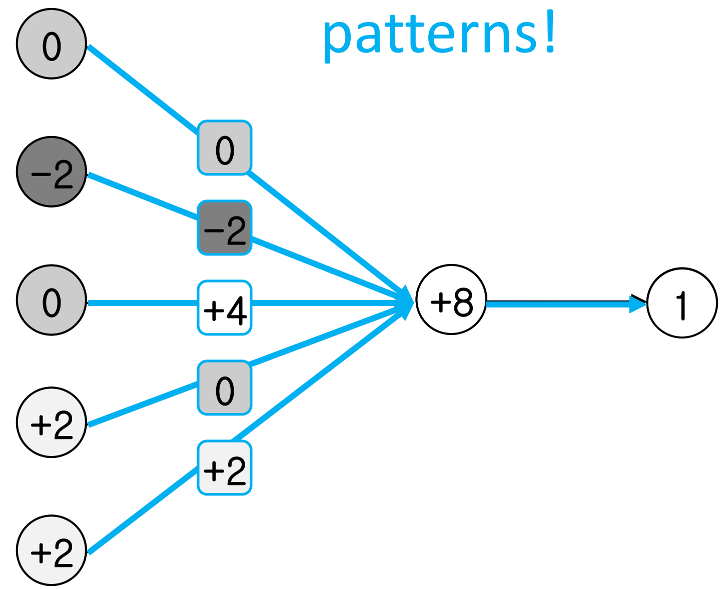
Logistic Unit as Artificial Neuron





Logistic Unit as Artificial Neuron

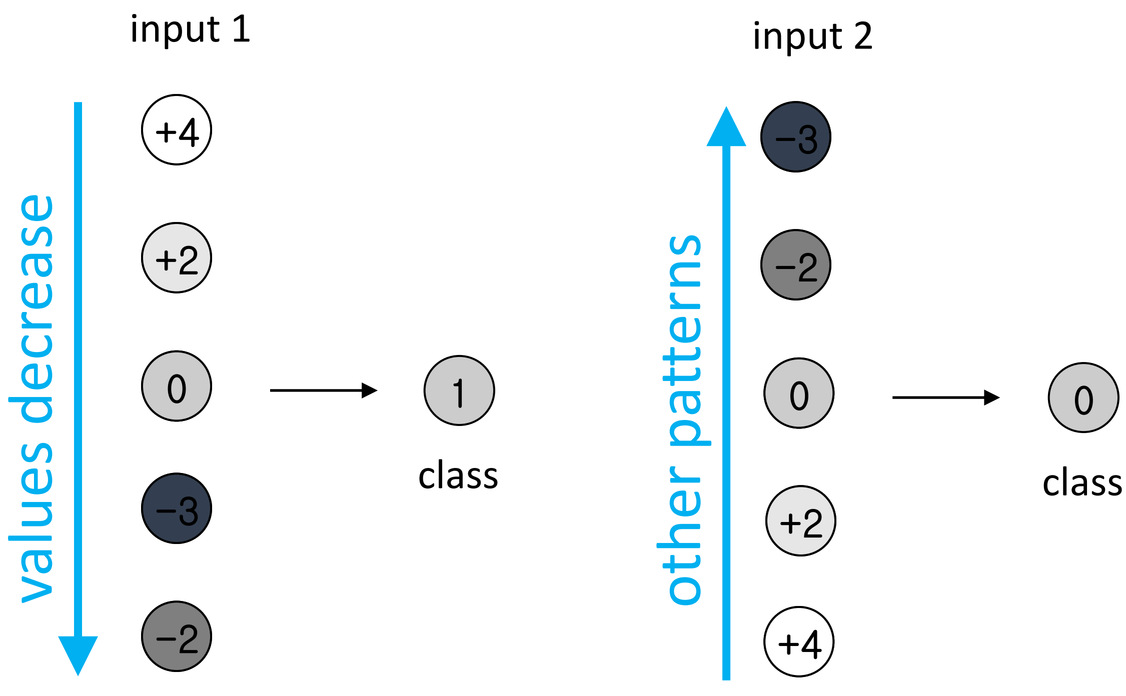




Neurons learn patterns!

Artificial Neuron Learns Patterns

- Classify input into class 0 or 1
- Teach neuron to predict correct class label
- Detect presence of a simple "feature"

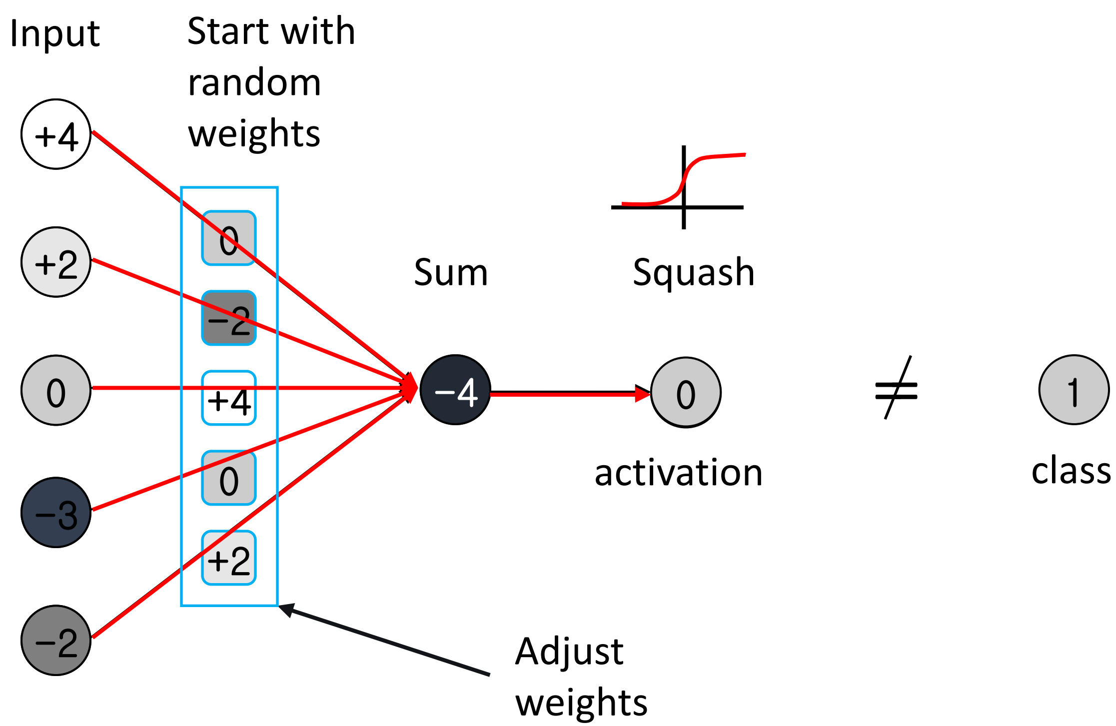


Example

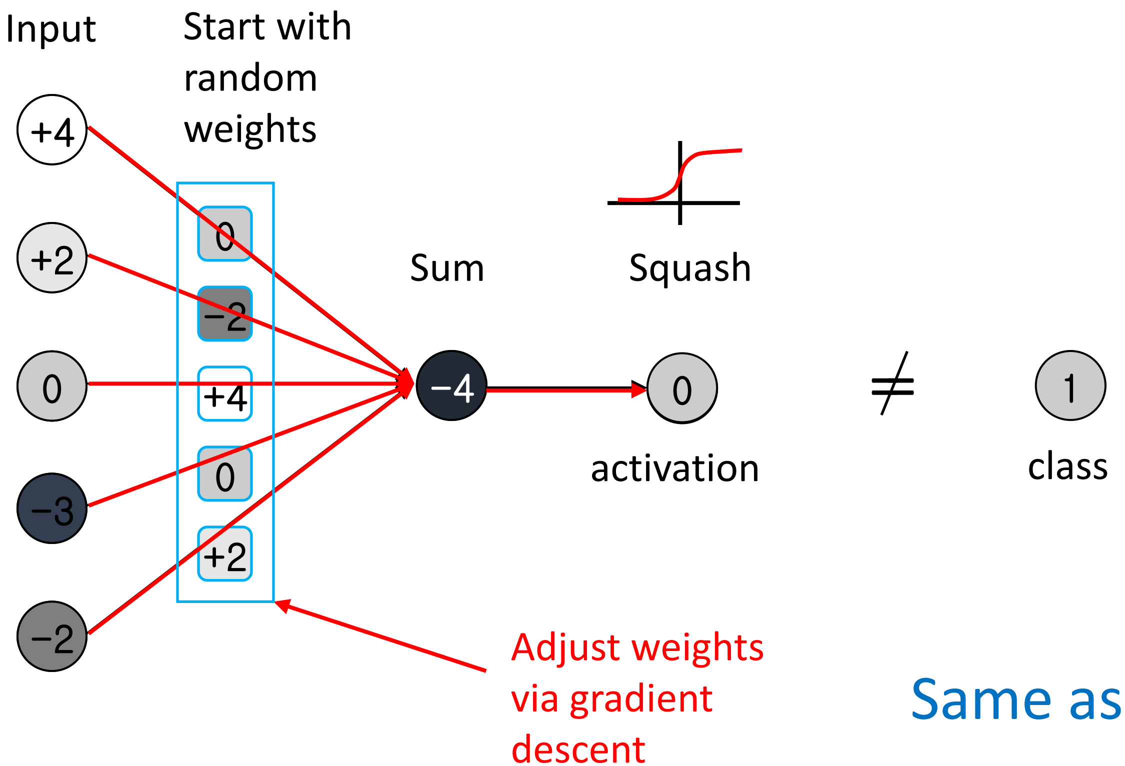
Neural Networks: Learning

Intuition

Artificial Neuron: Learning

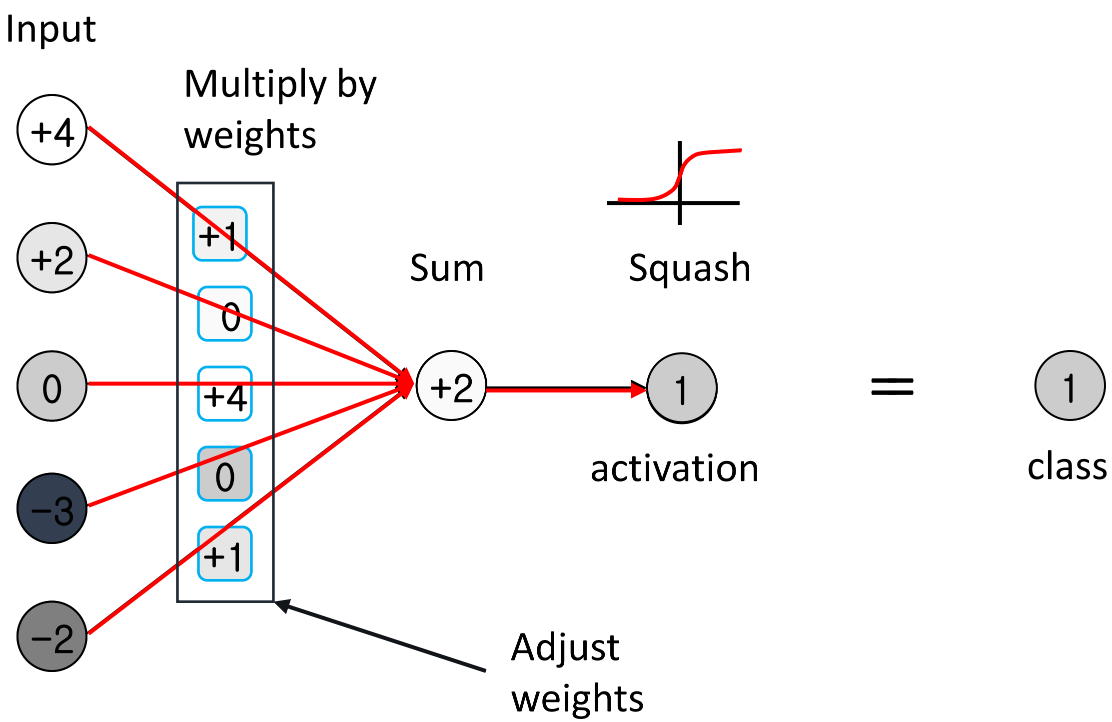


Artificial Neuron: Learning

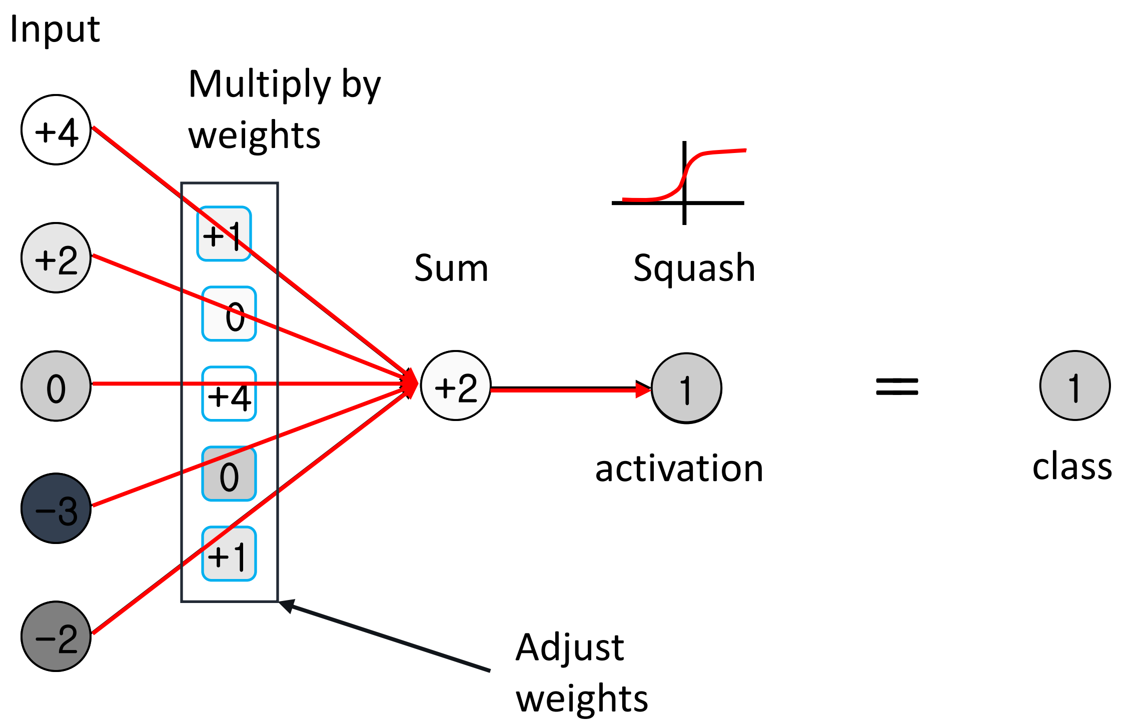


Same as in logistic regression

Artificial Neuron: Learning



Artificial Neuron: Learning

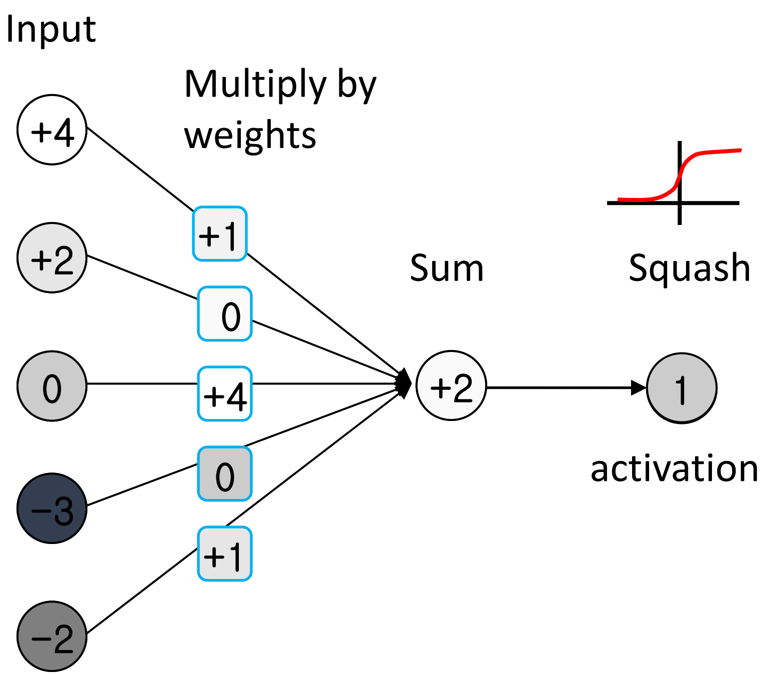


Forward propagation of information through a neuron

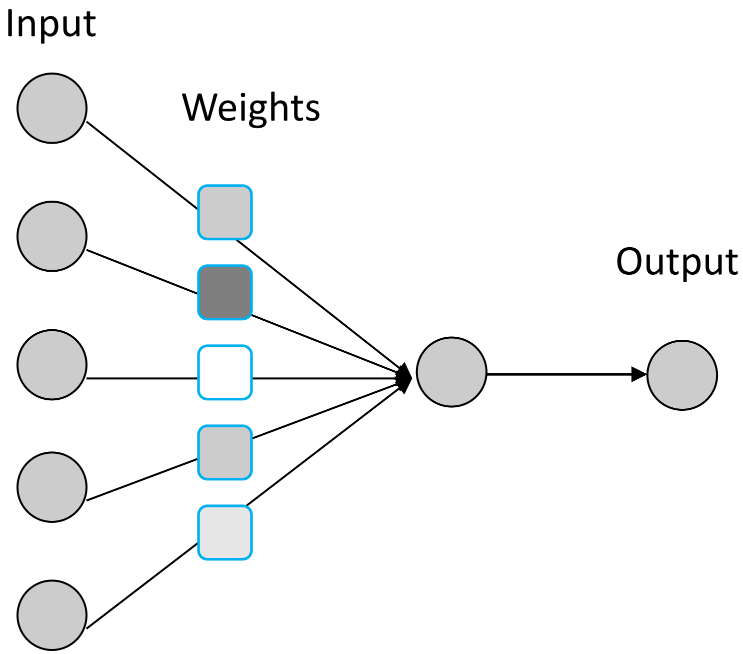
Neural Networks: Learning

Multi-layer network

Artificial Neuron: simplify

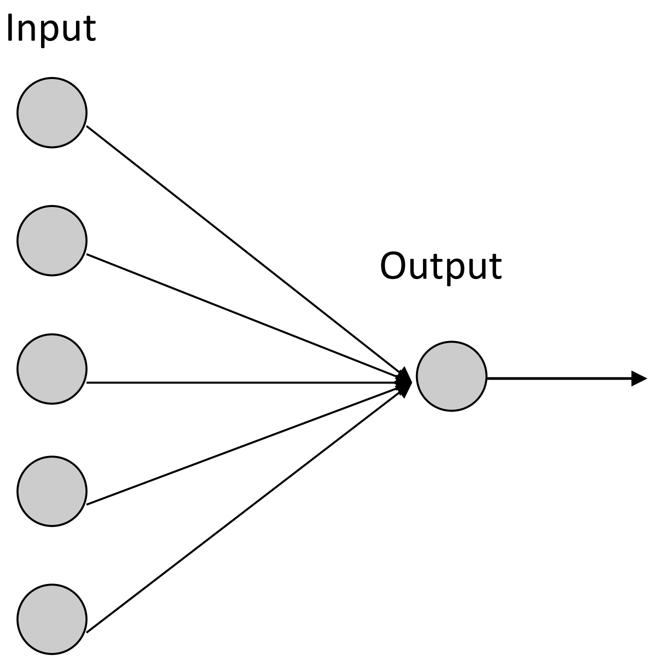


Artificial Neuron: simplify

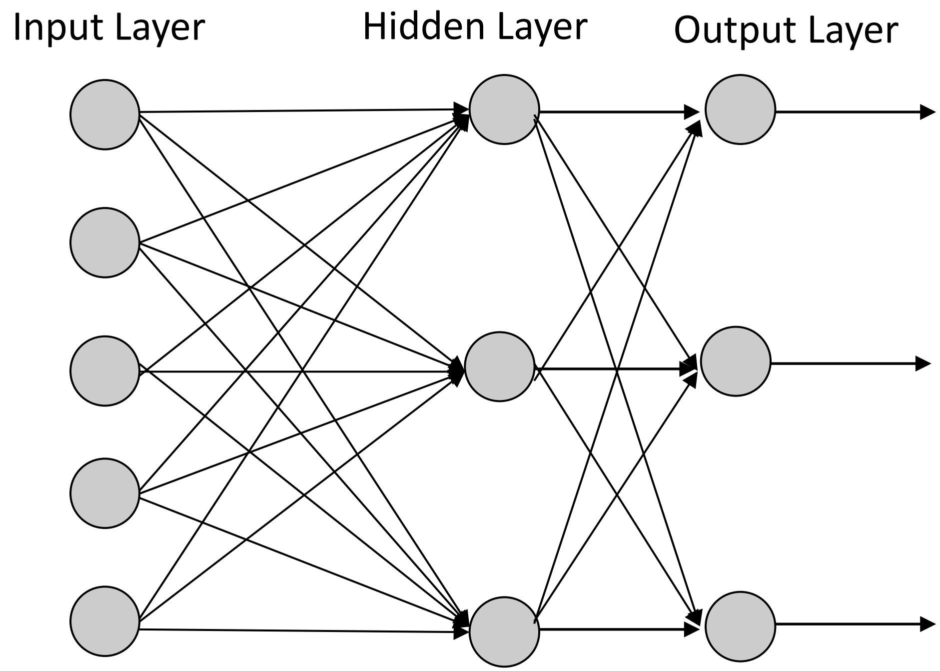


A single neuron is also called a perceptron

Artificial Neural Network



Single Neuron

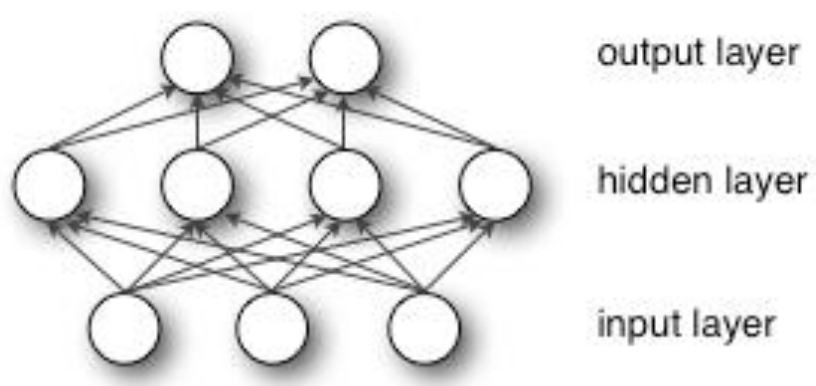


Neural Network (fully connected)

Deep Network: many hidden layers

Multi-layer perceptron (MLP)

- Just another name for a feed-forward neural network
- Logistic regression is a special case of the MLP with no hidden layer and sigmoid output.

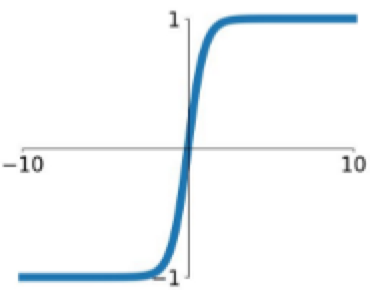


Other Non-linearities

- Also called activation functions

tanh

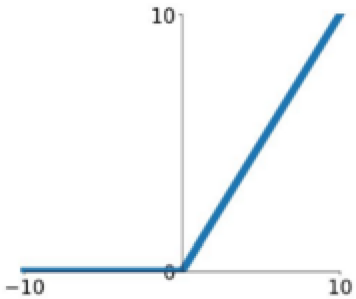
$\tanh(x)$



$$tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

ReLU

$\max(0, x)$

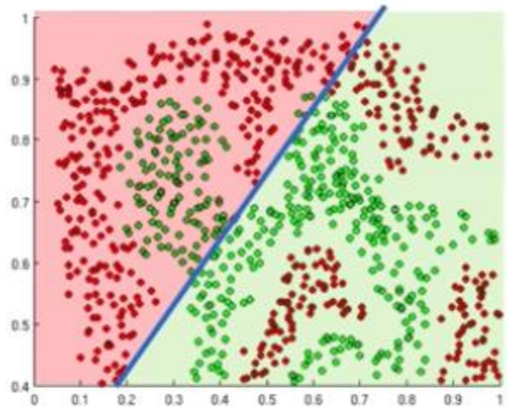


$$RELU(x) = \begin{cases} 0 \text{ if } x < 0 \\ x \text{ if } x >= 0 \end{cases}$$

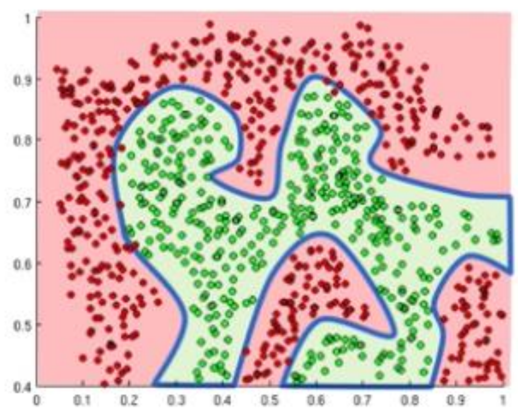
https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044

Importance of Non-linearities

The purpose of activation functions is to ***introduce non-linearities*** *into the network*



Linear activation functions produce linear decisions no matter the network size



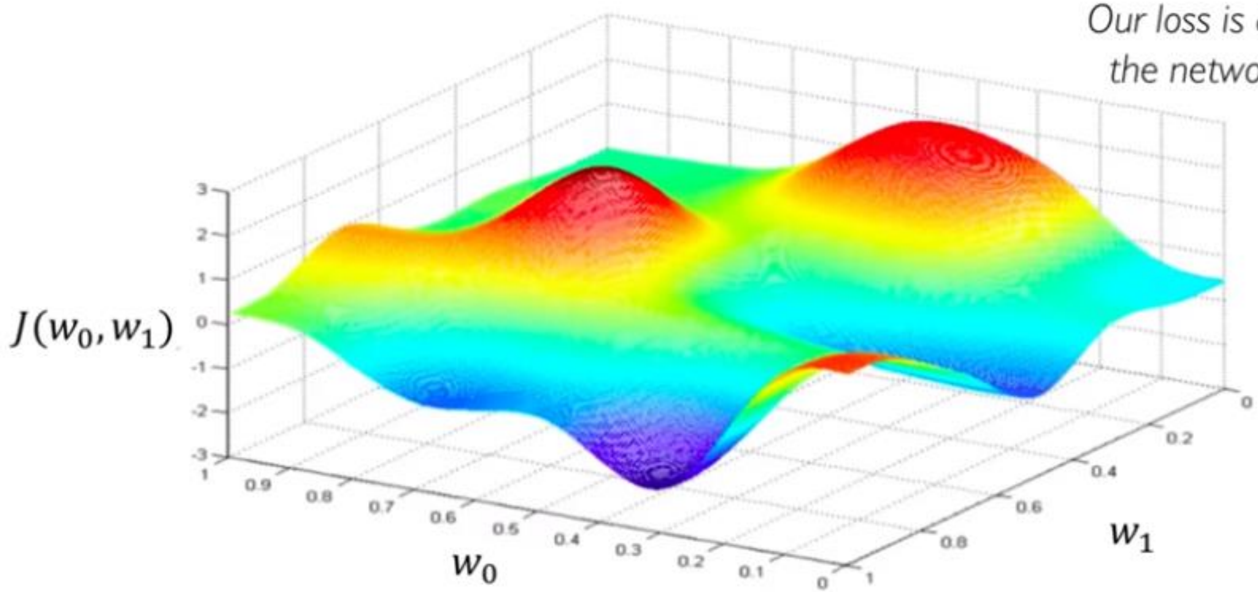
Non-linearities allow us to approximate arbitrarily complex functions

Loss Optimization

- Neural network parameters $\boldsymbol{\theta}$ are often referred to as weights $\boldsymbol{W}$.

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W})$$

Remember:
Our loss is a function of the network weights!



Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\boxed{\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}}$?
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\boxed{\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}}$ ← *Backpropagation*
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\boxed{\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Not feasible to compute over all dataset

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\boxed{\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Compute over a mini-batch

Gradient Descent

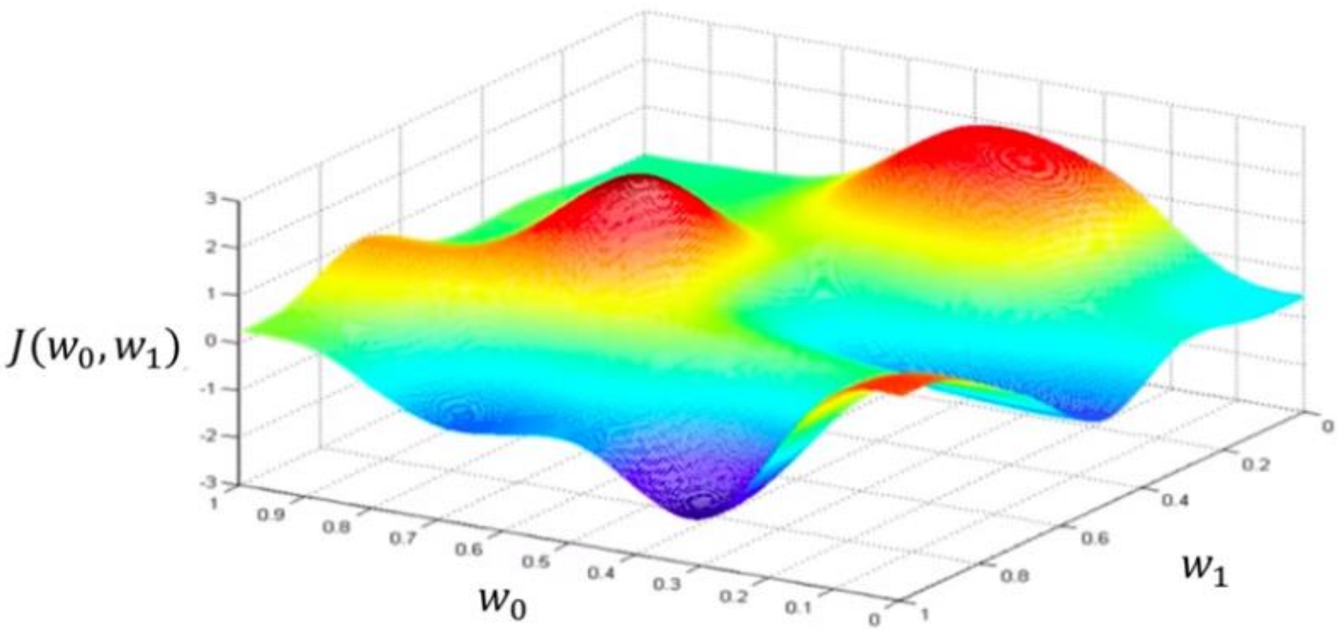
Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\boxed{\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

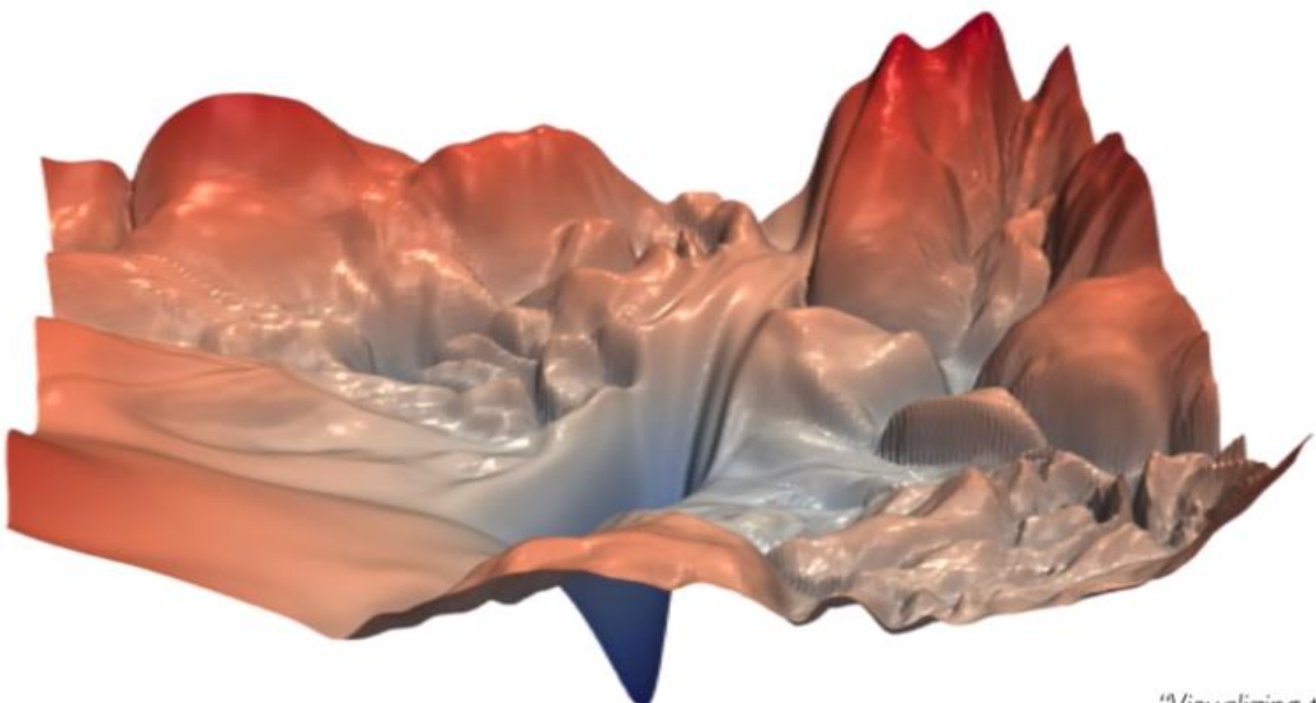
Compute over a mini-batch

Parallelization: Batches can be split onto multiple GPUs

Loss/Cost Function



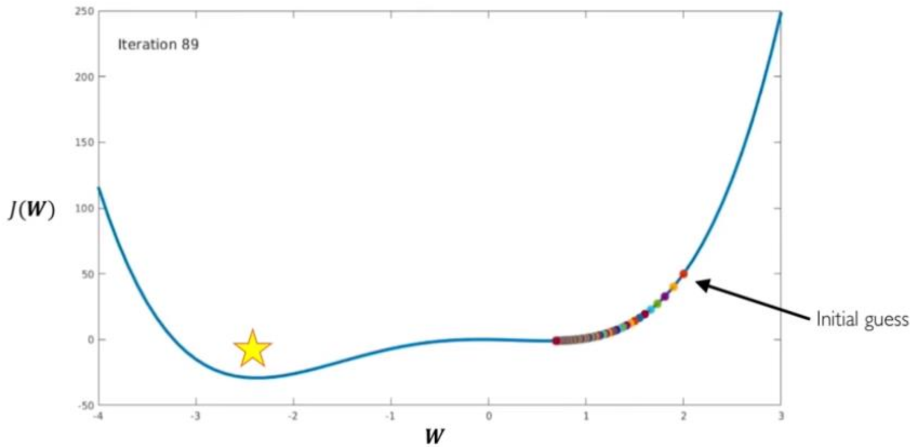
Landscape Visualization



"Visualizing the loss landscape of neural nets". Dec 2017.

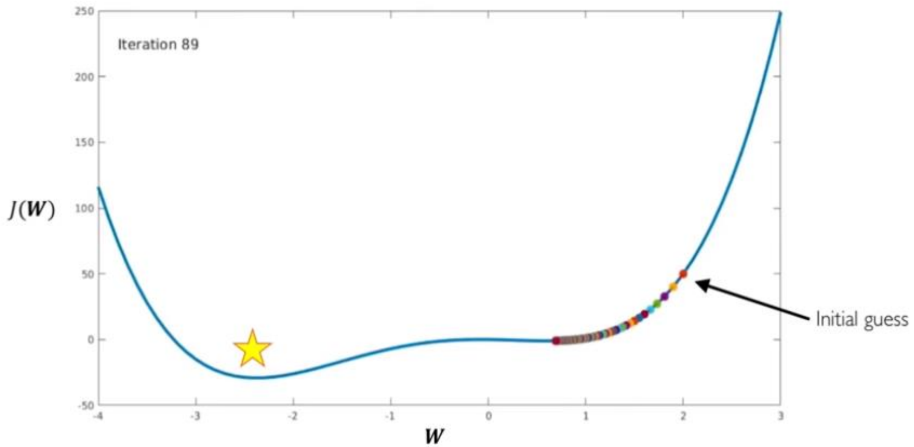
Setting the Learning Rate

Small learning rate *converges slowly and gets stuck in false local minima*

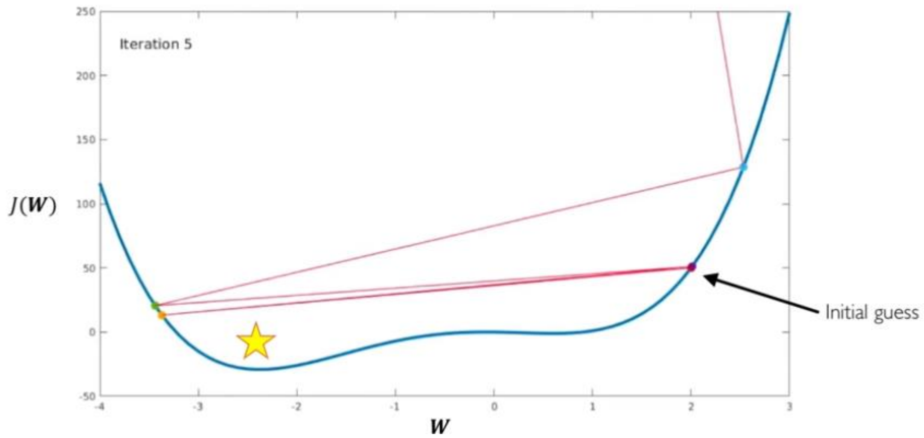


Setting the Learning Rate

***Small learning rate** converges slowly and gets stuck in false local minima*



***Large learning rates** overshoot, become unstable and diverge*



Setting the Learning Rate

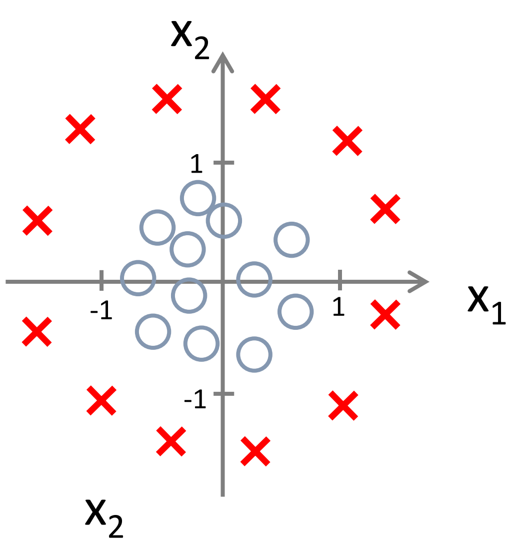
- How to select the learning Rate?
  - Try several, and see which works best
  - Start with a learning rate, and change it ***adaptively*** as the model trains
  - Many are implemented in Neural Network Tools

Neural Networks Learn Features

logistic regression unit == artificial neuron

chain several units together == neural network

“earlier” units learn non-linear feature transformation

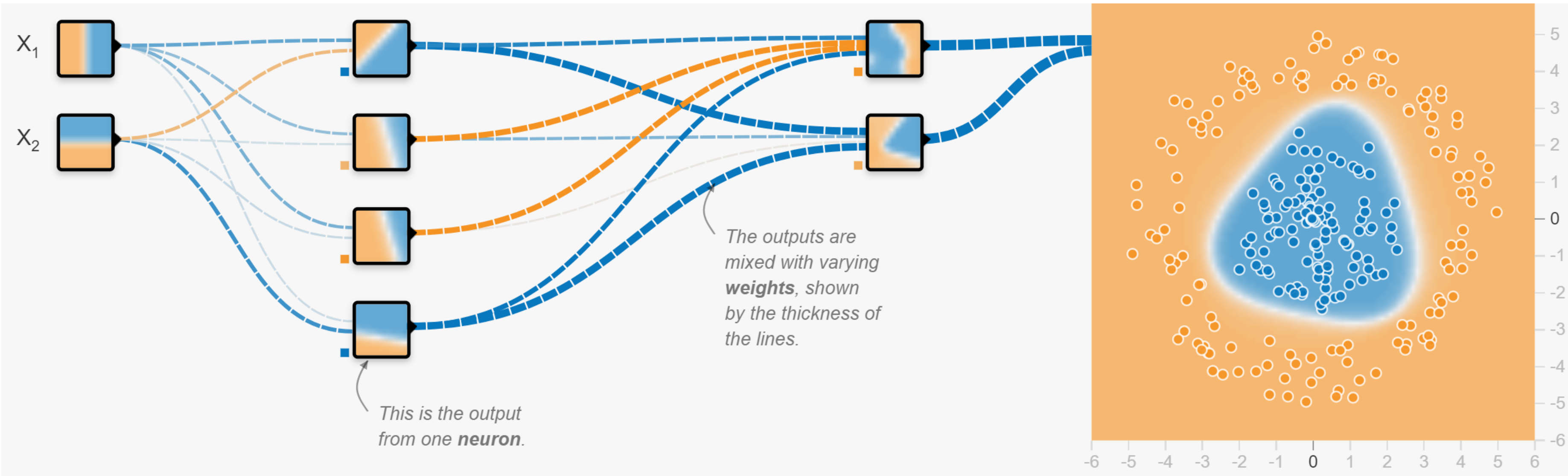


$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

simple neural network

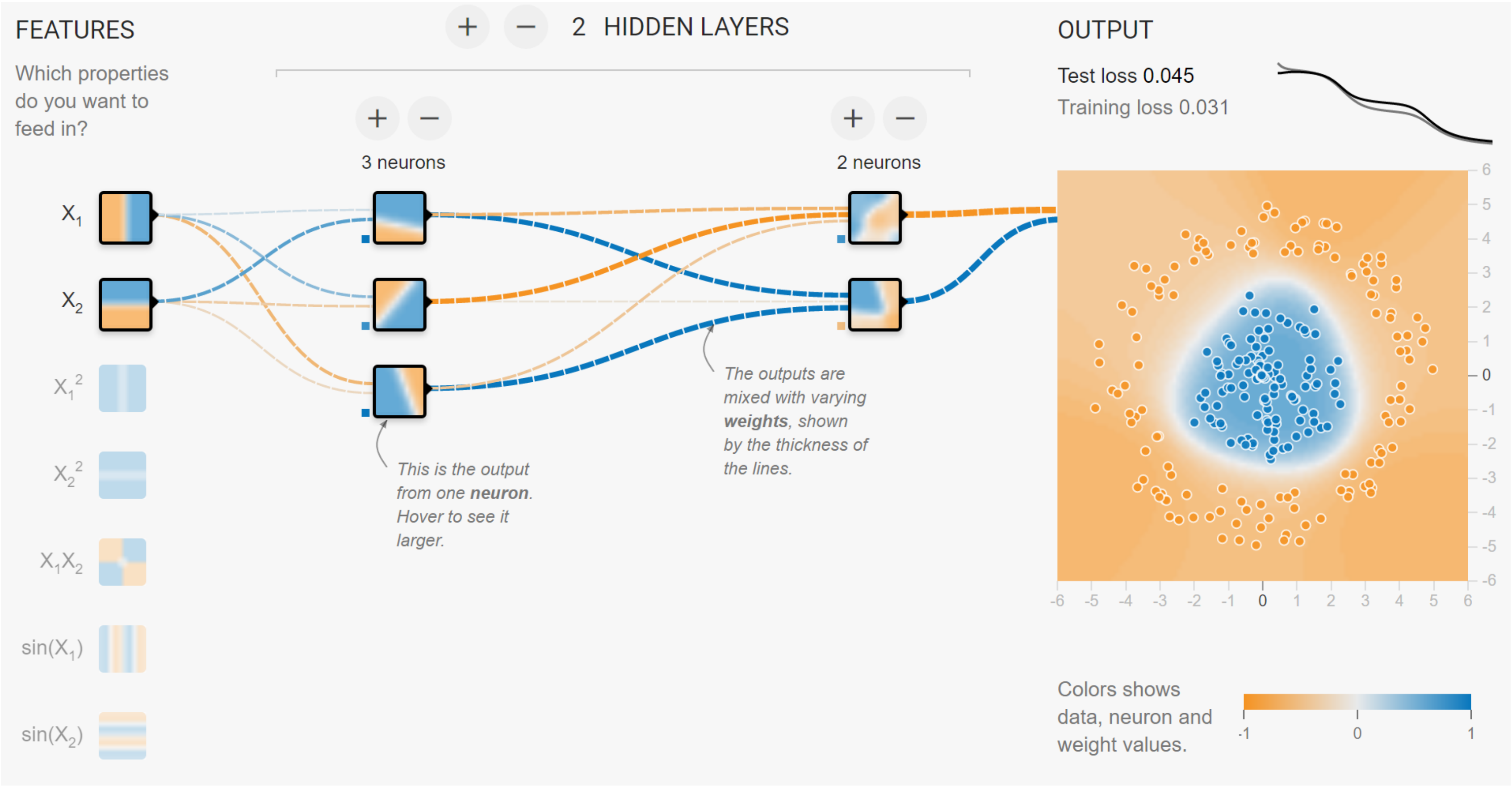
$$h(x) = g(\theta + \theta_1 h^{(1)}(x) + \theta_2 h^{(2)}(x) + \theta_3 h^{(3)}(x))$$

Example



X_1
X_2
The outputs are mixed with varying **weights**, shown by the thickness of the lines.
This is the output from one **neuron**.
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
0
1
2
3
4
5
6

Training a neural net: Demo



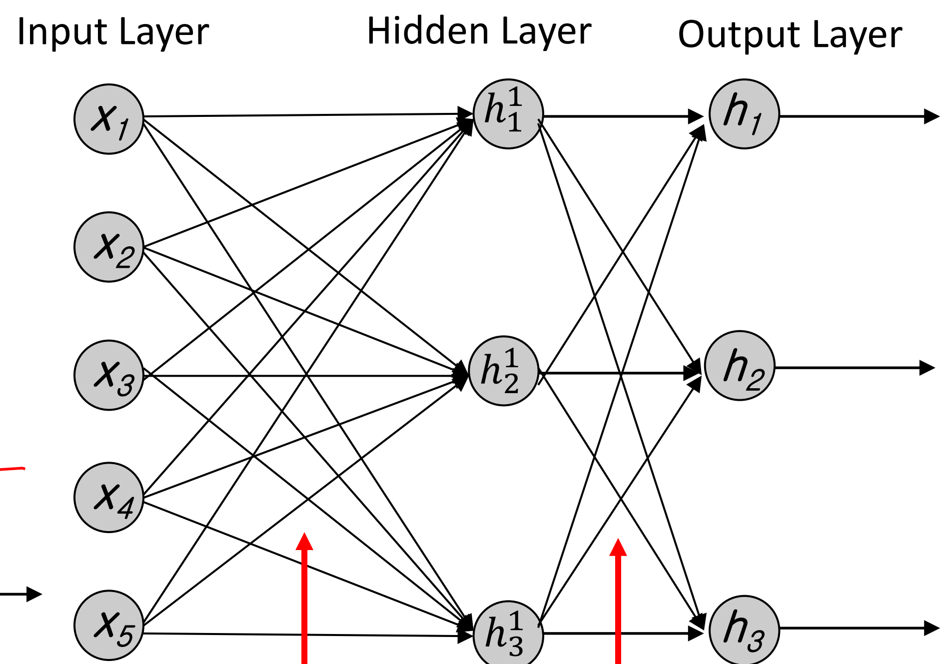
[Tensorflow playground]

Artificial Neural Network:

general notation

input

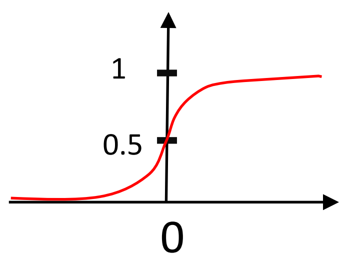
$$x = \begin{bmatrix} x_1 \\ \cdots \\ x_5 \end{bmatrix}$$



hidden layer activations

$$h^i = g(\Theta^{(i)} x)$$

$$g(z) = \frac{1}{1 + \exp(-z)}$$



output

$$h_\Theta(\mathrm{x}) = g(\Theta^{(2)} a)$$

weights

$$\Theta^{(1)} = \begin{pmatrix} \theta_{11} & \cdots & \theta_{15} \\ \vdots & \ddots & \vdots \\ \theta_{31} & \cdots & \theta_{35} \end{pmatrix} \qquad \Theta^{(2)} = \begin{pmatrix} \theta_{11} & \cdots & \theta_{13} \\ \vdots & \ddots & \vdots \\ \theta_{31} & \cdots & \theta_{33} \end{pmatrix}$$

Cost function

Neural network: $h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$

training error

$$J(\Theta) = \boxed{-\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1-(h_\Theta(x^{(i)}))_k)\right]}$$
$$\boxed{+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2}$$

regularization

Gradient computation

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log h_\theta(x^{(i)})_k + (1 - y_k^{(i)}) \log(1 - h_\theta(x^{(i)})_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_j^{(l)})^2$$

$$\min_{\Theta} J(\Theta)$$

Need code to compute:

- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

Gradient computation

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log h_\theta(x^{(i)})_k + (1-y_k^{(i)})\log(1-h_\theta(x^{(i)})_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_j^{(l)})^2$$

$$\min_{\Theta} J(\Theta)$$

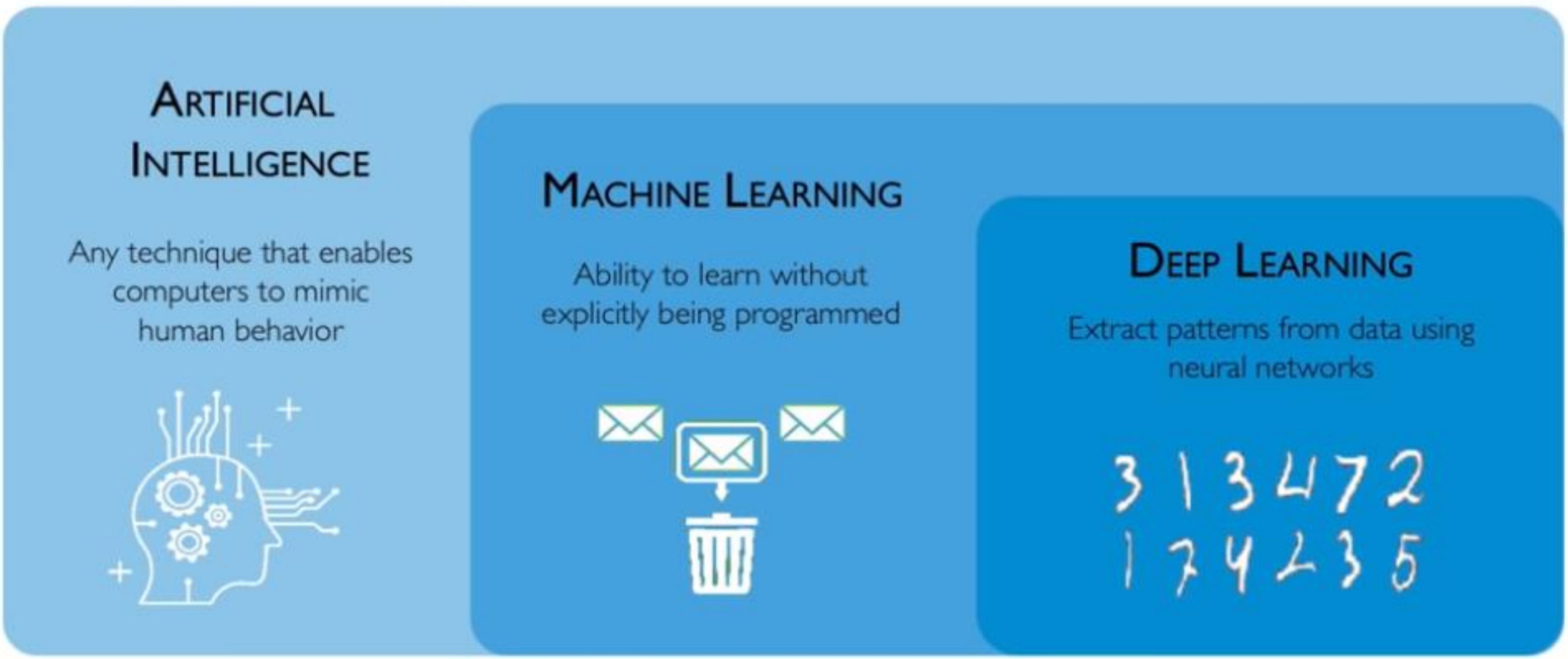
Need code to compute:

- $J(\Theta)$
- $\boxed{\frac{\partial}{\partial\Theta_{ij}^{(l)}} J(\Theta)}$ *Backpropagation*

Deep Learning

Architectures

What is Deep Learning?

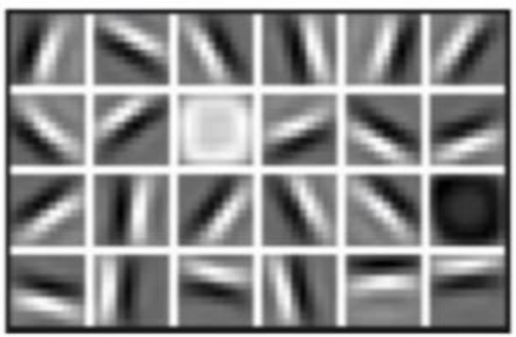


Why Deep Learning?

Hand engineered features are time consuming, brittle, and not scalable in practice

Can we learn the **underlying features** directly from data?

Low Level Features



Lines & Edges

Mid Level Features

Eyes & Nose & Ears

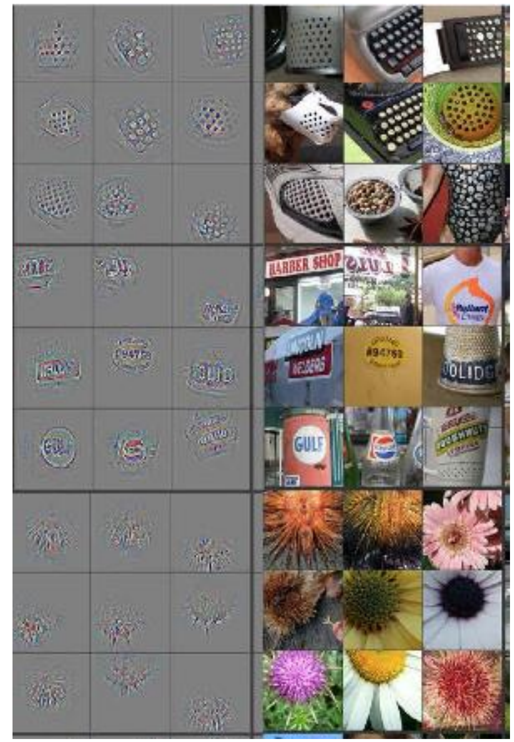
High Level Features

Facial Structure

Why Deep Learning?

The Unreasonable Effectiveness of Deep Features

Maximal activations of pool_5 units [R-CNN]



conv_5 DeConv visualization
[Zeiler-Fergus]

Rich visual structure of features deep in hierarchy.

Why Now?

1952 Stochastic Gradient Descent

1958 Perceptron
- Learnable Weights

⋮

1986 Backpropagation
- Multi-Layer Perceptron

1995 Deep Convolutional NN
- Digit Recognition

⋮

Neural Networks date back decades, so why the resurgence?

1. Big Data
- Larger Datasets
- Easier Collection & Storage

IMAGENET

WIKIPEDIA
The Free Encyclopedia

2. Hardware
- Graphics Processing Units (GPUs)
- Massively Parallelizable

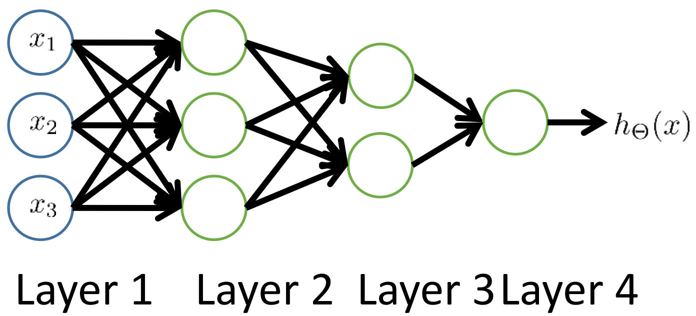
3. Software
- Improved Techniques
- New Models
- Toolboxes

TensorFlow

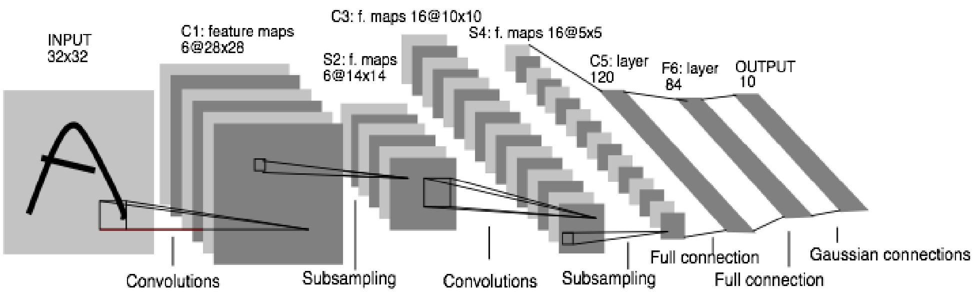
Network architectures

Feed-forward

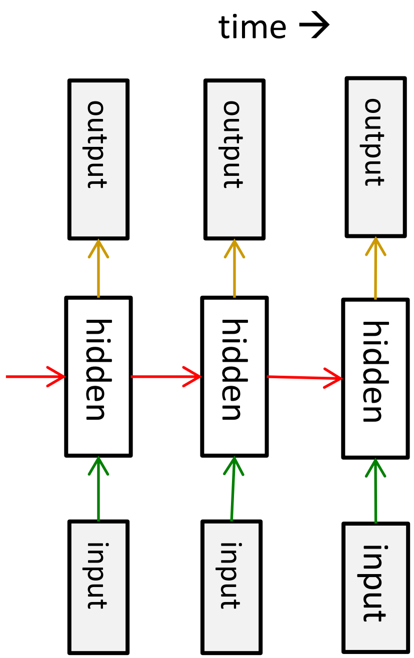
Fully connected



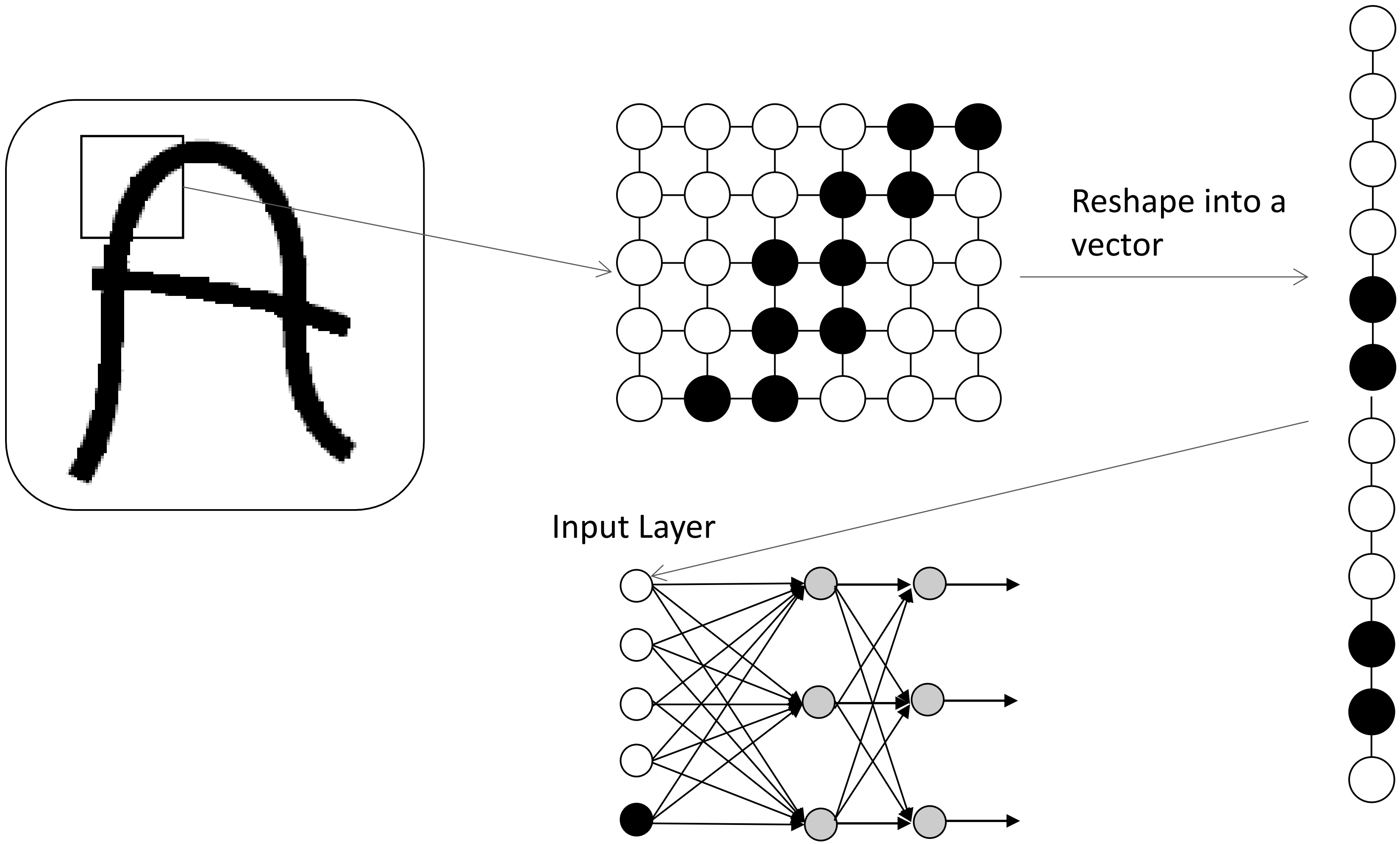
Convolutional



Recurrent



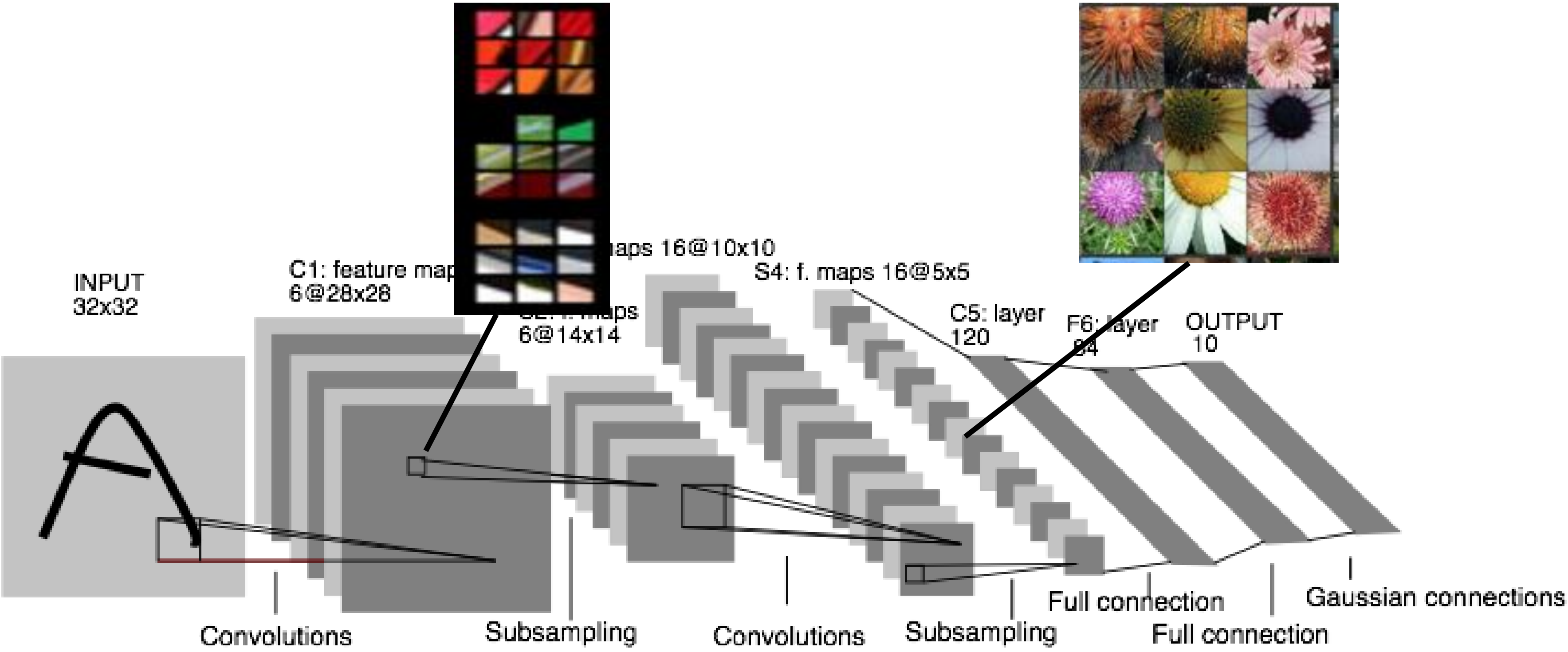
Fully Connected



Not ideal for representing images

Convolutional Neural Network

A better architecture for 2d signals



LeNet

Summary so far

- Neural network chains together many layers of “neurons” such as logistic units
- Hidden neurons learn more and more abstract non-linear features