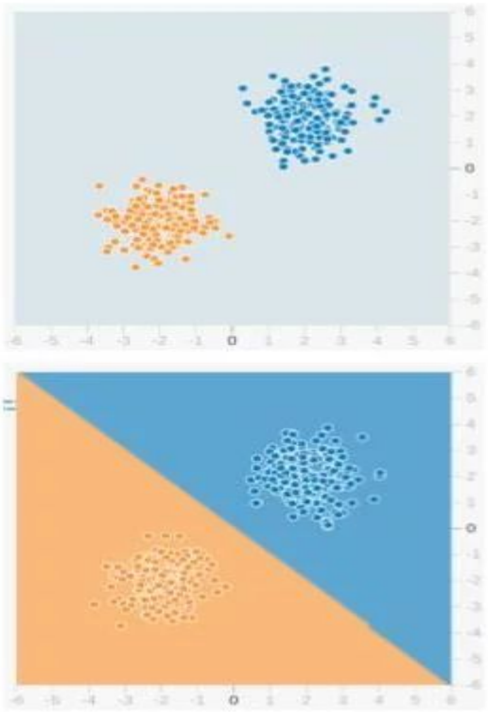
Deep Learning

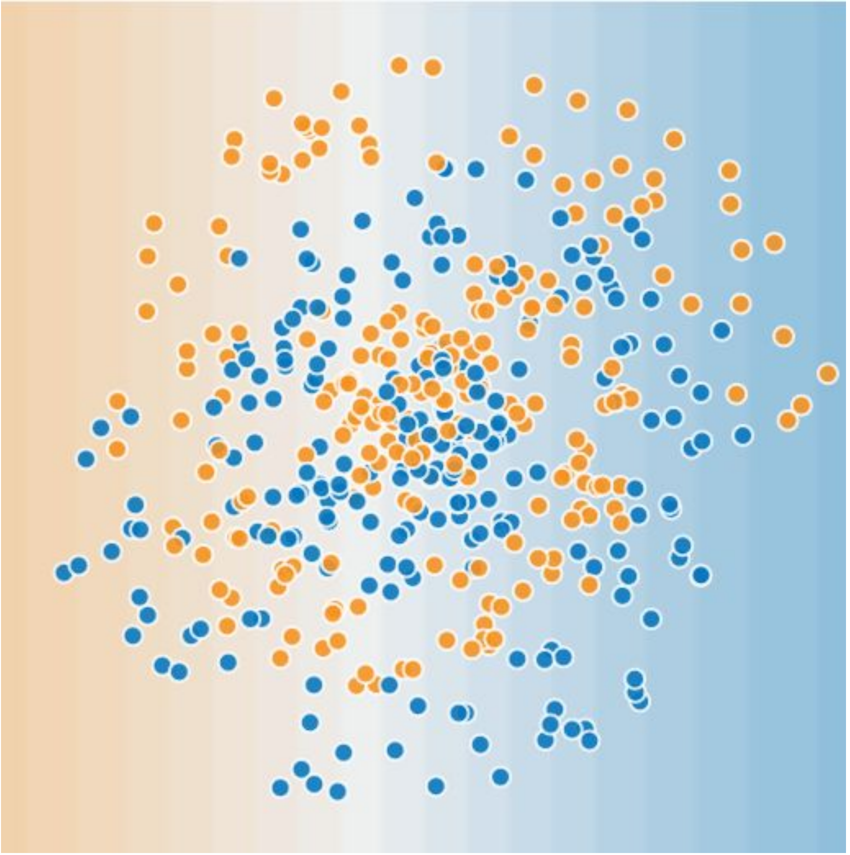
Neural Networks

A “Simple” Classification Problem



How about this classification problem?

Linear model can not solve the problem



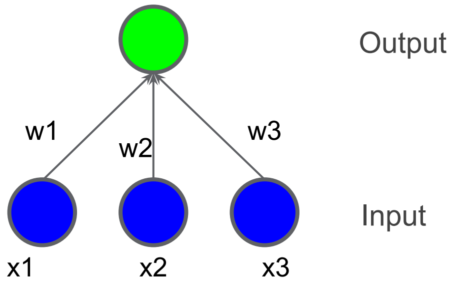
We need **non-linear models**

A Linear Model

- Linear Regression if output is continuous
- Logistic Regression if output is discrete

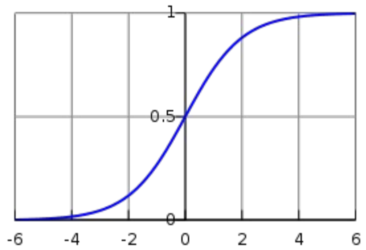
Linear Regression

$$y = \mathbf{wx} + b$$

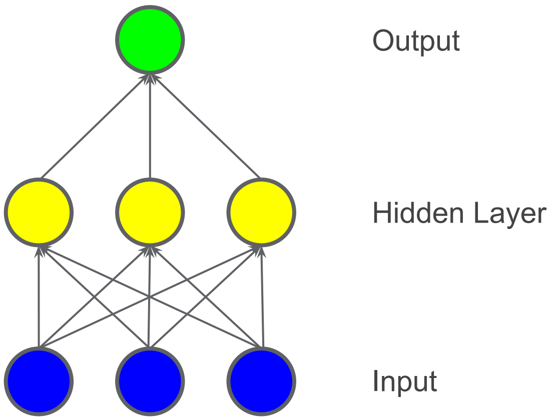


Logistic Regression

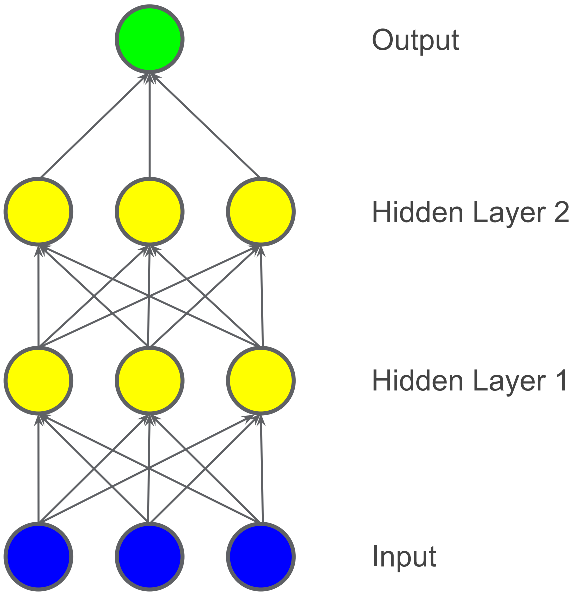
$$y = \sigma(\mathbf{wx} + b)$$



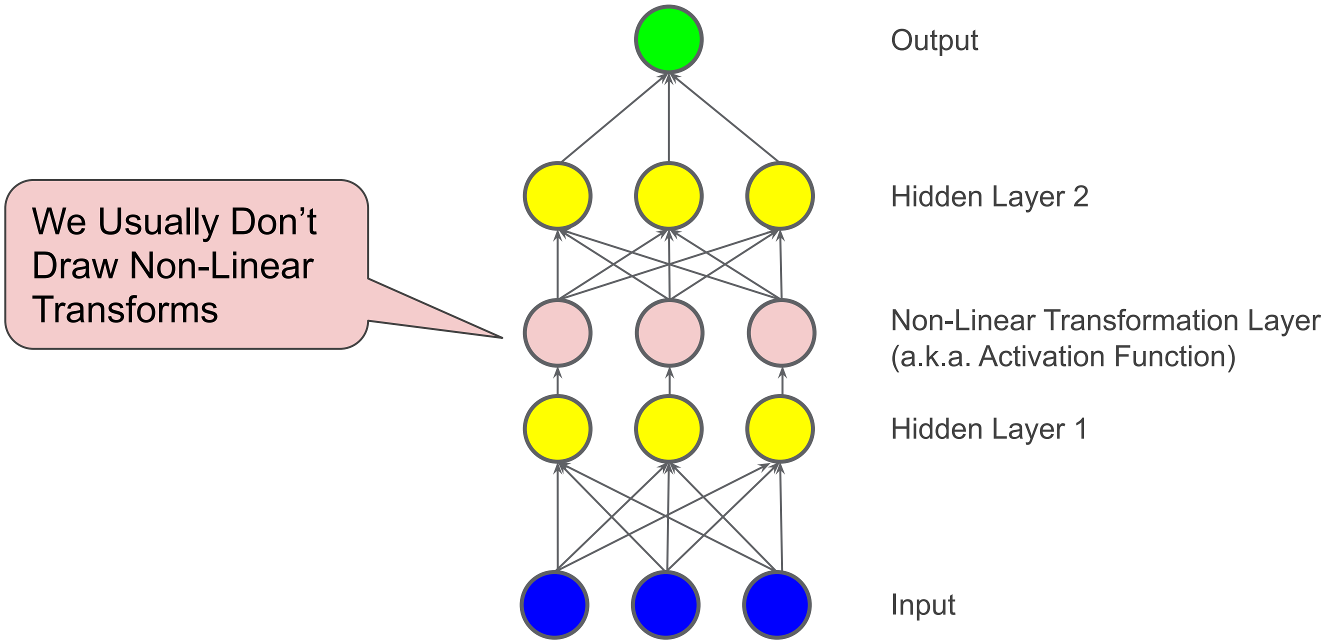
Add Complexity



How about now?



Make it non-linear

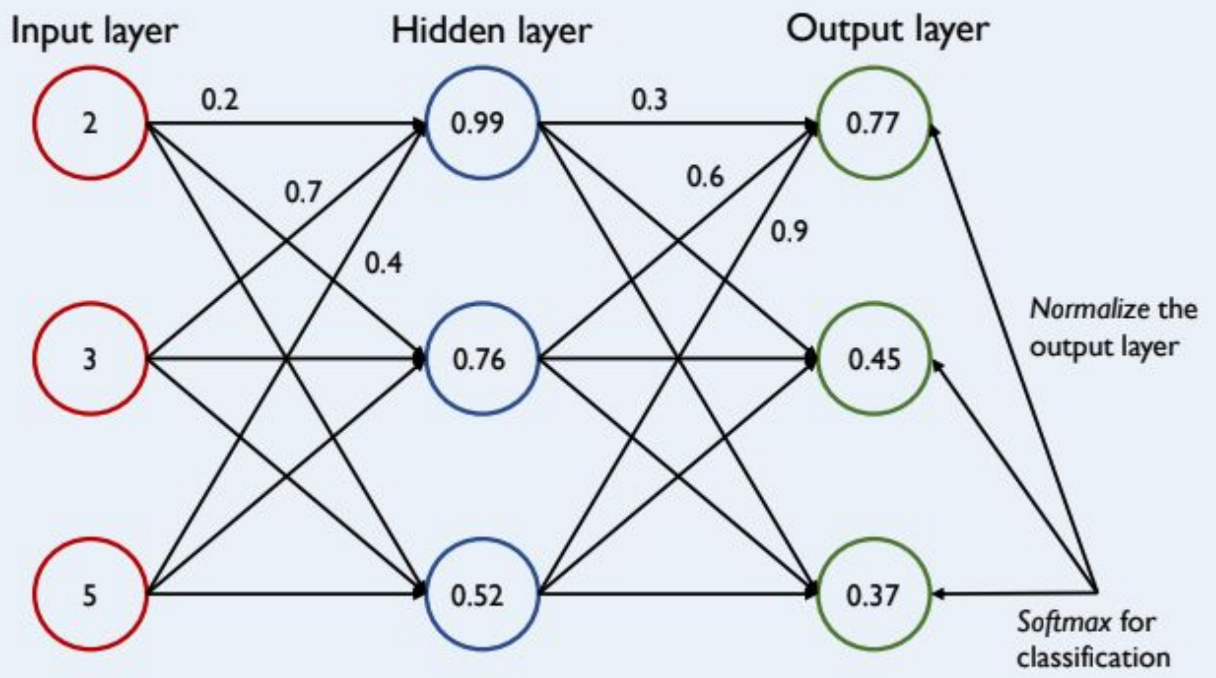


Why Non-linear Activation

- The non-linearities activation function increases the capacity of model
- Without non-linearities, deep neural networks is meaningless: each extra layer is just one linear transform.
- How to select activation functions?

  You can select an activation function which will approximate the distribution faster leading to faster training process.

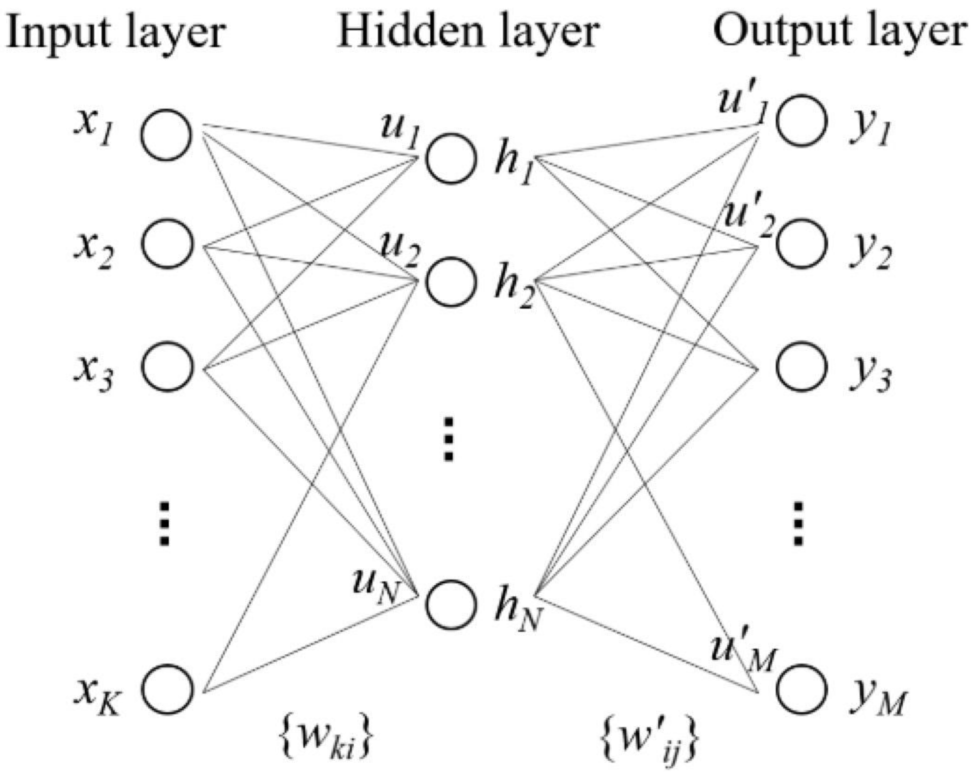
Forward Computation



$2 * 0.2 + 3 * 0.7 + 4 * 0.5 = 4.5$ $\qquad$ $0.98 * 0.3 + 0.76 * 0.6 + 0.52 * 0.9 = 1.218$

$sigmoid(x) = \frac{1}{1+e^{-x}} = 0.99$ $\qquad$ $sigmoid(x) = \frac{1}{1+e^{-x}} = 0.77$

Forward Computation



$$u_i = \sum_{k=1}^{K} w_{ki} x_k$$

$$h_i = f(u_i)$$

$$u'_j = \sum_{i=1}^{N} w'_{ij} h_i$$

$$y_j = f(u'_j)$$

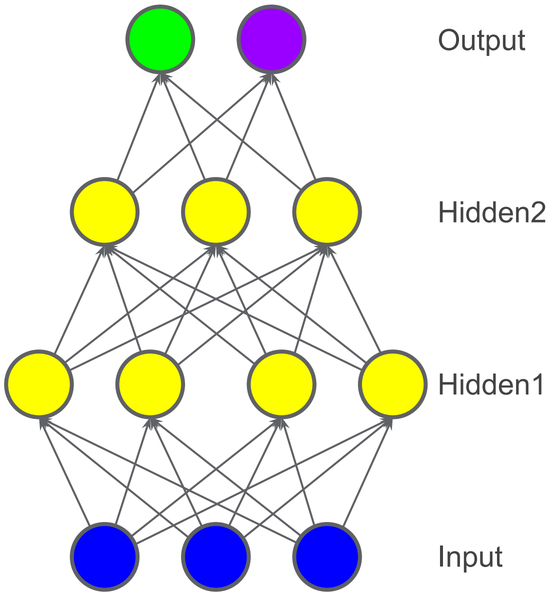
Forward Computation

1. Take f as the non-linear activation
2. Linear Transformation: $$h = W_1 x$$
3. 2-layer Neural Network: $$h = W_2 f(W_1 x)$$
4. 3-layer Neural Network: $$h = W_3 f(W_2 f(W_1 x))$$

- Neural Network is a model that **recursively** applies the matrix multiplication and non-linear activation function.

Backpropagation

Neural networks can be arbitrarily complex



Training done via BackProp algorithm: gradient descent in very non-convex space

$$\underbrace{\min}_{\text{optimizer}} \underbrace{E}_{\text{error function}}(\underbrace{f}_{\text{architecture}}(\underbrace{x), t}_{\text{data}}) + \underbrace{R}_{\text{regularization term}}$$

Gradient Descent

Gradient for loss function f for $\mathbf{x}_n$, which computed by BP algorithm

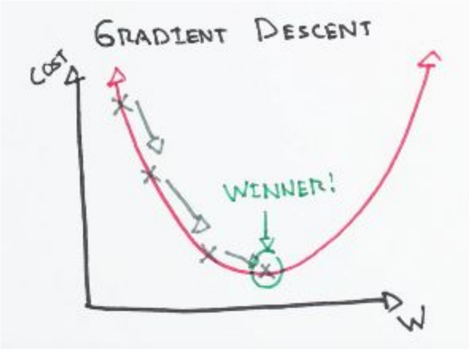
$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha \boxed{\nabla f(\mathbf{x}_n)}$$

New Parameters Guess

Current Parameters Guess

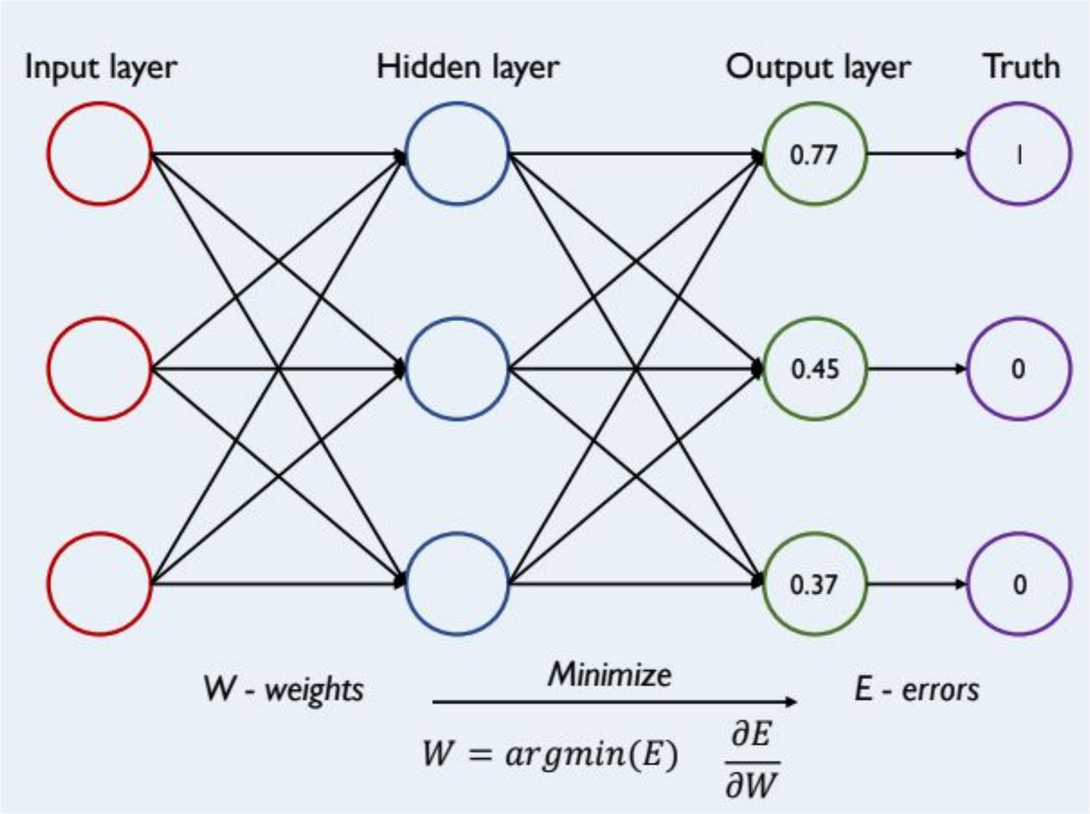
Learning Rate

Like hiking down a mountain



Credit:https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html

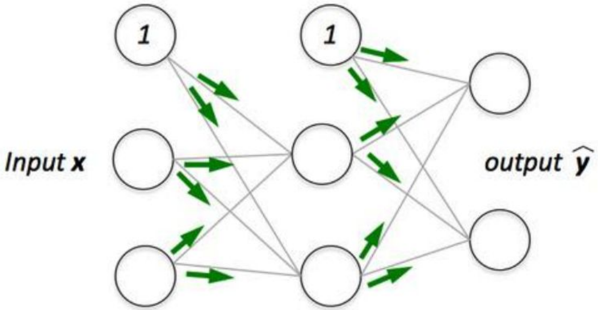
Backpropagation



Backpropagation

Step 1:

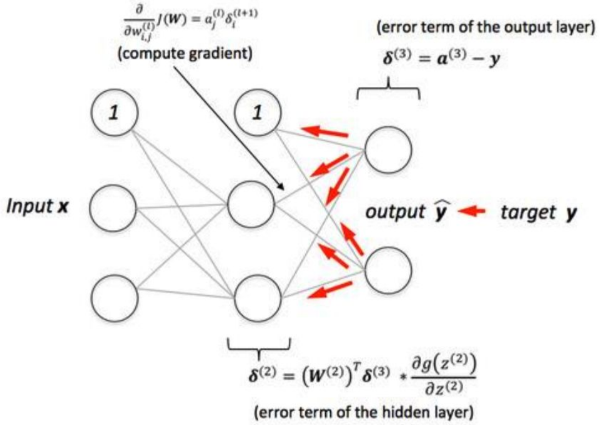
Forward pass to compute the network output and “error”



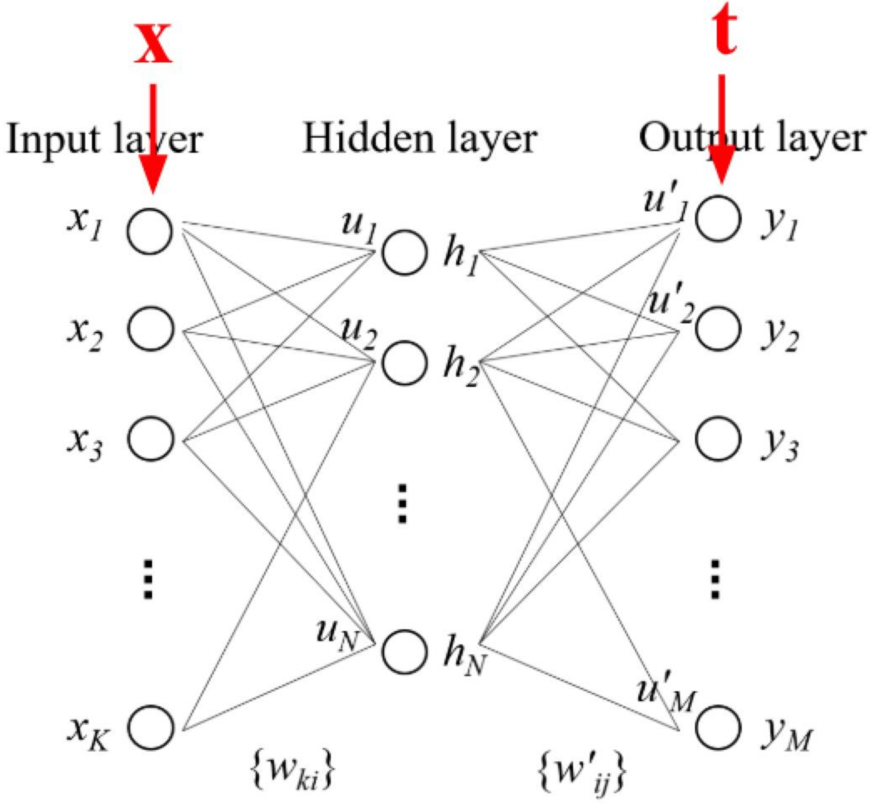
Step 2:

Backward pass to compute gradients

And update the model weights based on gradients.



Backpropagation



$$E = \frac{1}{2}\sum_{j=1}^{M}(y_j - t_j)^2$$

$$\frac{\partial E}{\partial y_j} = y_j - t_j$$

$$\frac{\partial E}{\partial u'_j} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial u'_j}$$

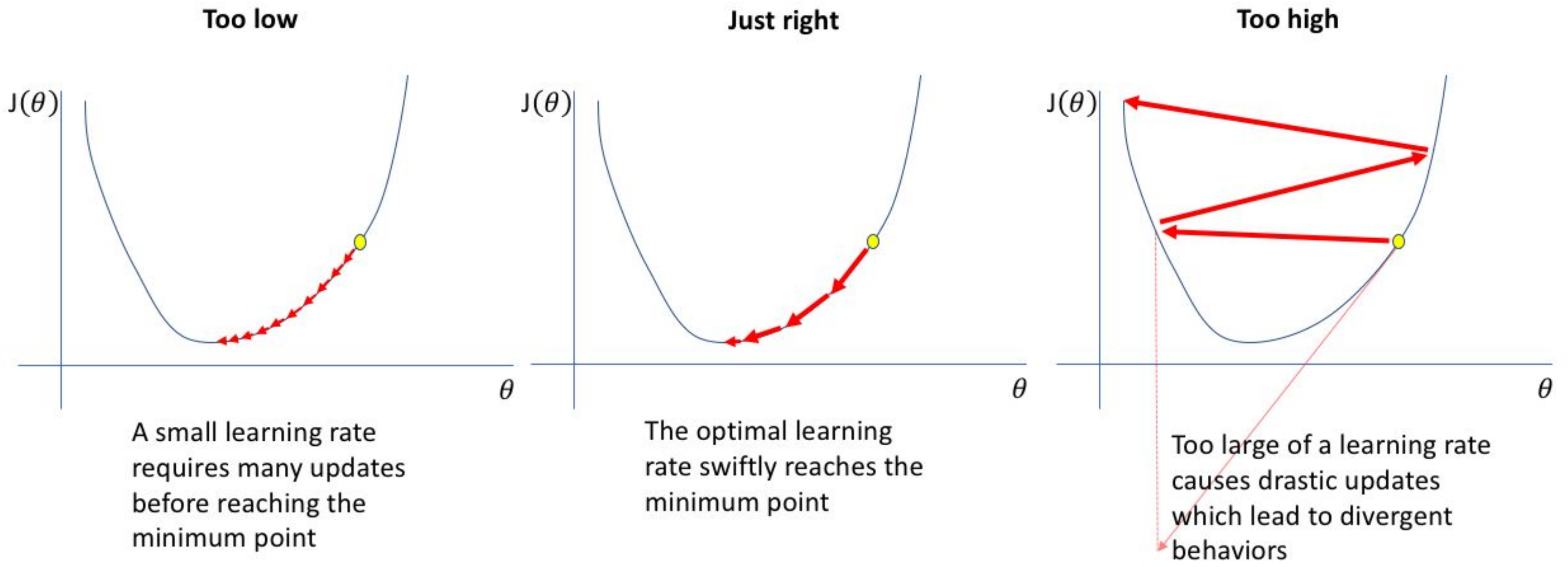
$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u'_j} \cdot \frac{\partial u'_j}{\partial w'_{ij}}$$

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{M} \frac{\partial E}{\partial u'_j} \frac{\partial u'_j}{\partial h_i}$$

$$\frac{\partial E}{\partial u_i} = \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial u_i}$$

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial u_i} \cdot \frac{\partial u_i}{\partial w_{ki}}$$

How to find learning rate?



https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

A Joke

One variant of Gradient Descent
Algorithm

Training Process

1. Initialize neural network randomly
2. Get output with input data
3. Compare outputs with ground truth in training data
4. Get loss function
5. Update weights with backpropagation and gradient descent algorithm

Iteratively perform

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha \bigtriangledown f(\mathbf{x}_n)$$

- Stochastic gradient descent (SGD)
  - Randomly shuffle the data
  - Batch size k: the number of data used for steps 2-5
  - One epoch: the full scan of all the training data. How many times will the weights be updated in one epoch?
  - Number of Epoch T: the number of iterations to stop training

Types of Gradient Descent Algorithms

1. Batch Gradient Descent — *batch size = Number of data*
2. Mini-batch Gradient Descent — 1<*batch size*< *number of data*
3. Stochastic Gradient Descent — *batch size = 1*

Batch SGD

Batch SGD: batch size is the number of training data

1 only update model parameters after all training data have been evaluated.

2 stable error gradient

3 need a large memory

4 may lead to a less optimal solution

Mini-Batch SGD

Mini-batch SGD: split the dataset into small batches and take the average of the gradient over the batch and update the weights

1 more efficient than SGD

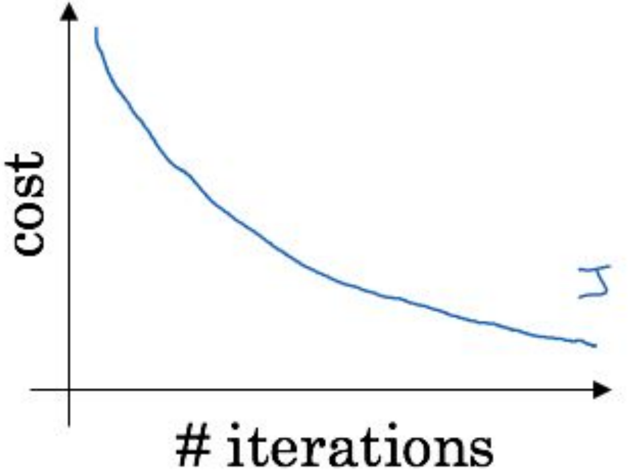
2 requires additional hyperparameter i.e. mini-batch size

3 hints on batch size:

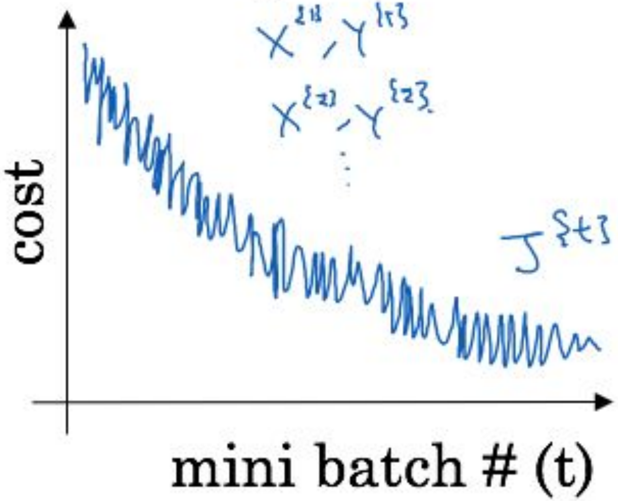
* a power of two that fits the memory requirements of GPU or CPU.
* small -> a learning process that converges quickly at the cost of noise in the training
* large -> a learning process that converges slowly with accurate estimate of the error gradient

Mini-Batch vs Batch

Batch gradient descent



Mini-batch gradient descent



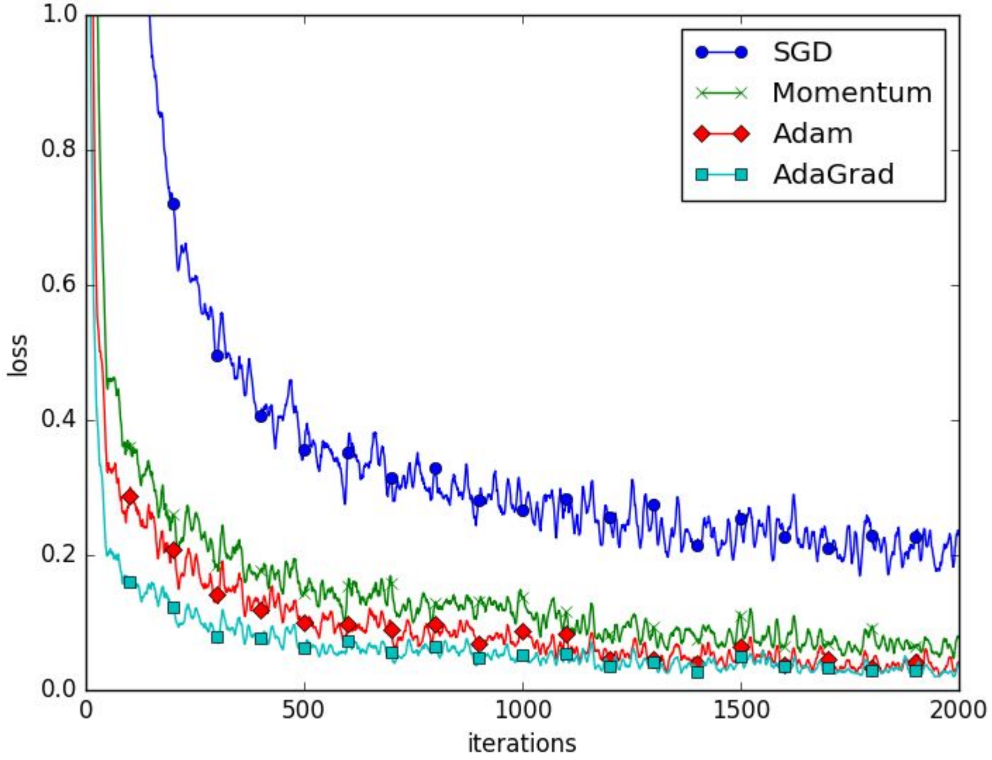
Except SGD

SGD

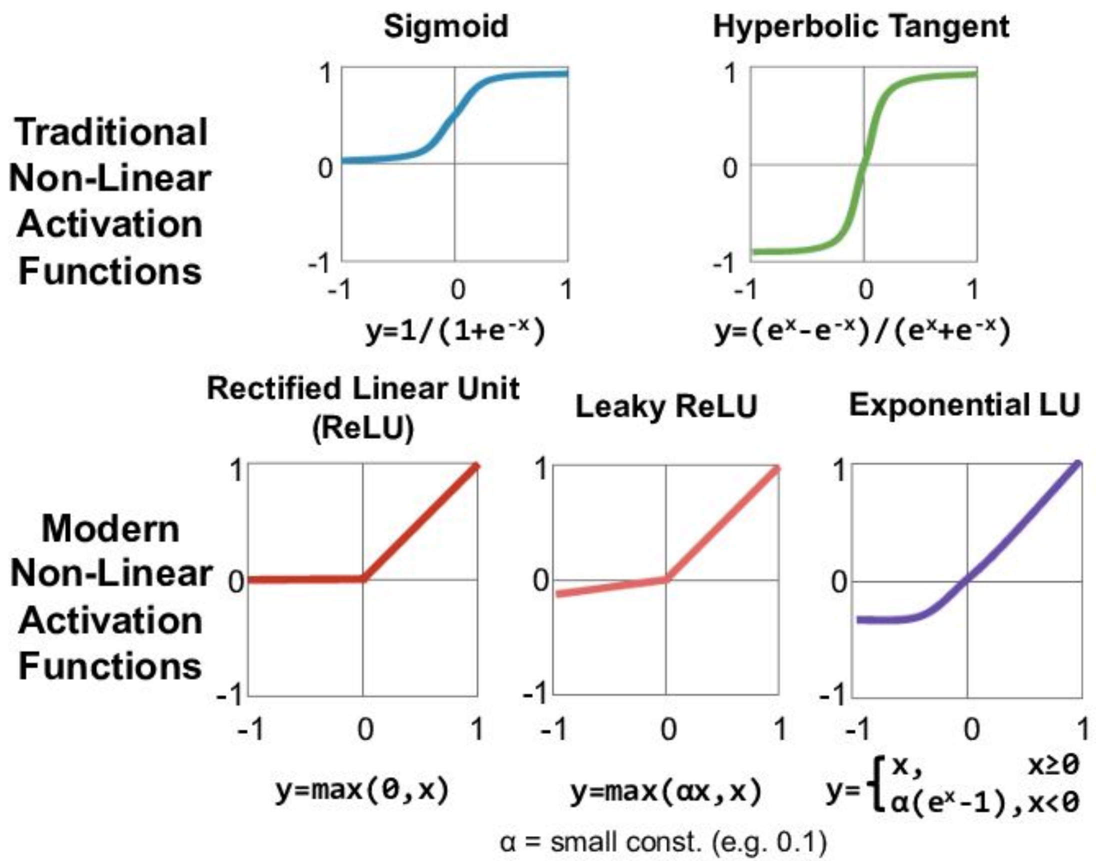
$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha \bigtriangledown f(\mathbf{x}_n)$$

Different Variants

Momentum, Adam, AdaGrad, RMSProp



Non-linear Activation Functions



When Gradient is zero

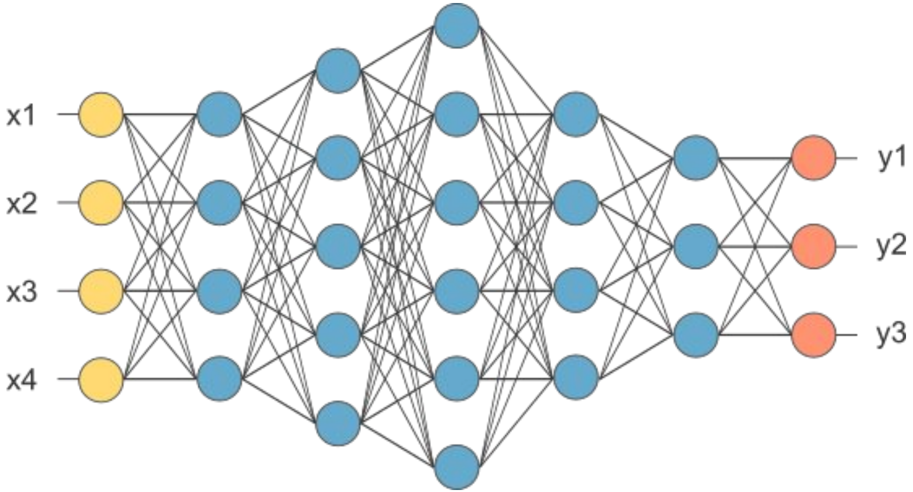
Neural Network

1. From Wiki:

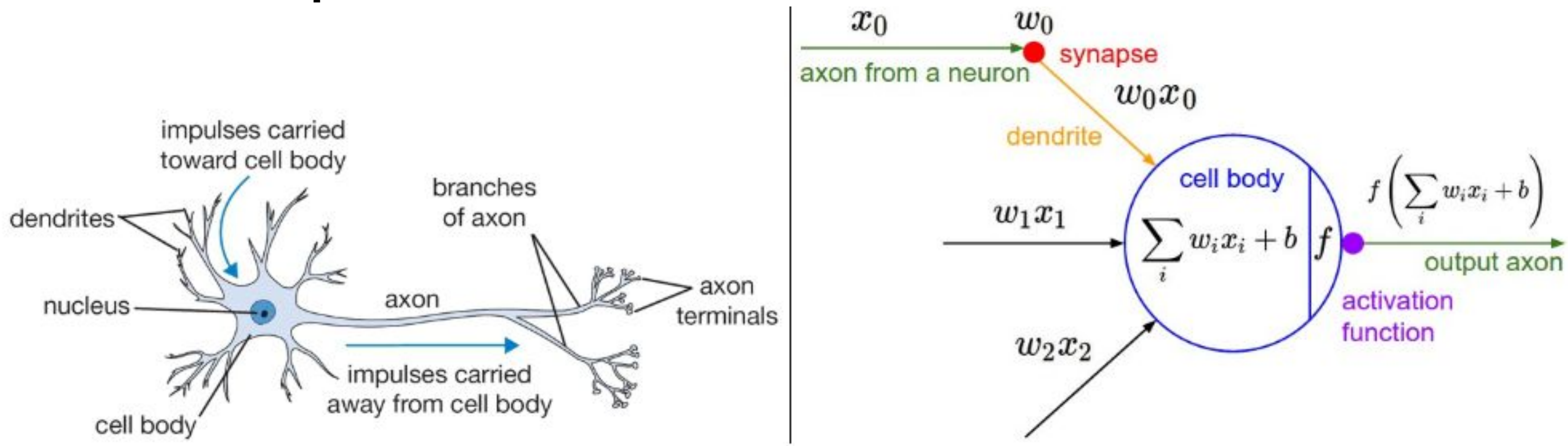
- NN is based on a collection of connected units of nodes called artificial **neurons** which loosely model the neurons in a biological brain.

2. From another way:

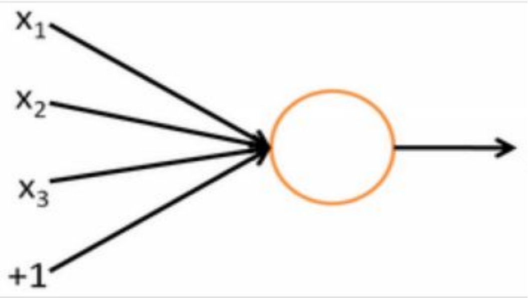
- NN is running several ‘logistic regression’ at the same time (expanding at width and depth dimensions).



Neural Computation



A cartoon drawing of a biological neuron (left) and its mathematical model (right).



The fact that a neuron is essentially a logistic regression unit:

1 performs a dot product with the input and its weights

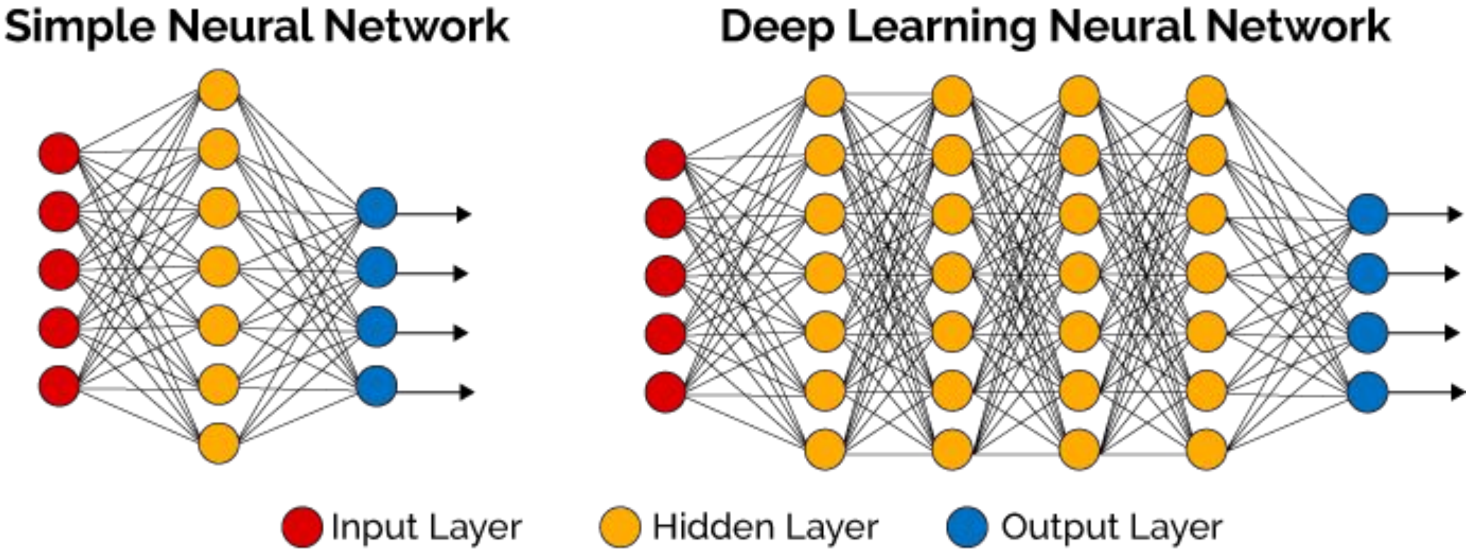
2 adds the bias and apply the non-linearity

Neural Network Visualization

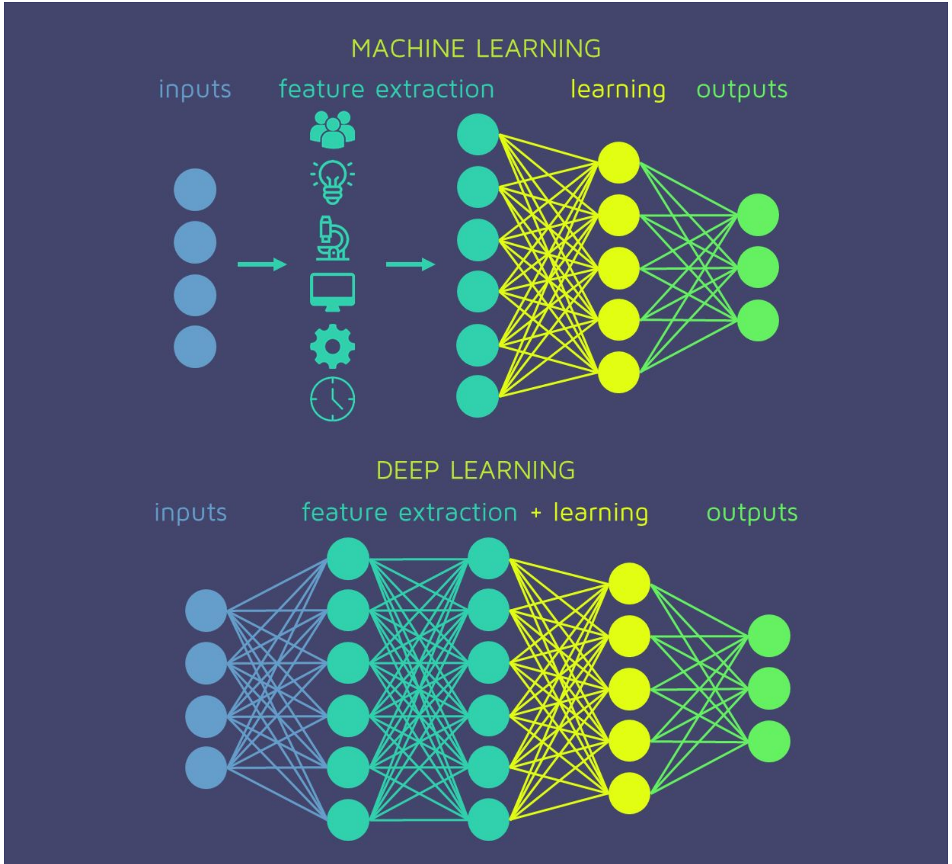
[Playground]

Deep Learning/Deep Neural Networks

Shallow vs Deep

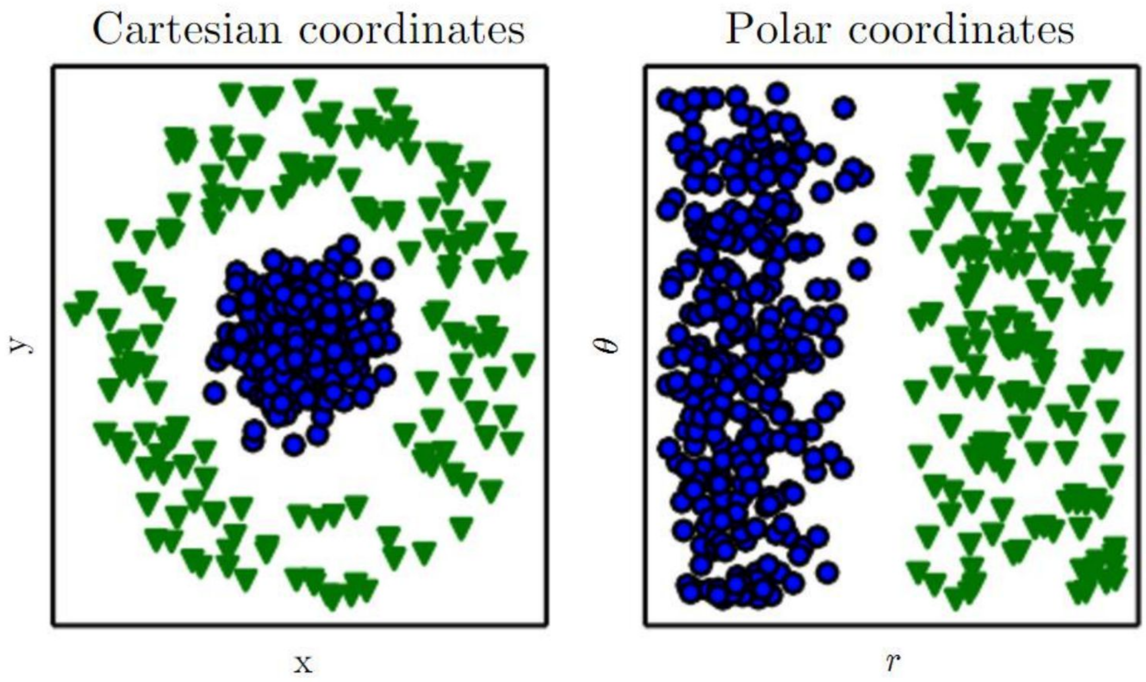


End-to-End Learning

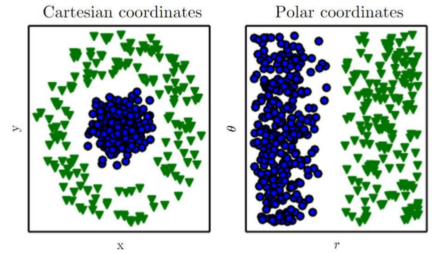


From Aporras

Representation Matters

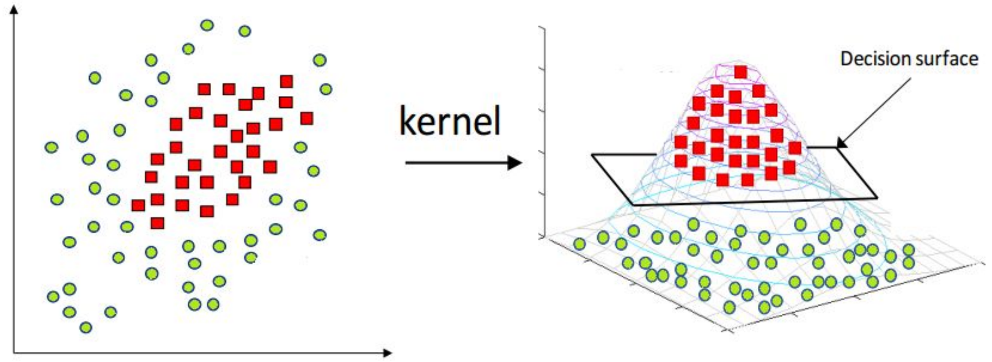


Task: Draw a line to separate the **green triangles** and **blue circles**.



We want to project the data into the new feature/vector space that data is linearly separated

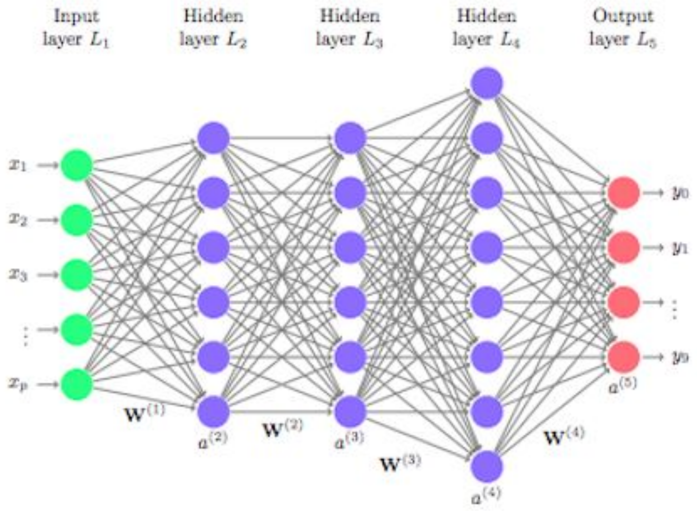
Kernel Tricks in SVM



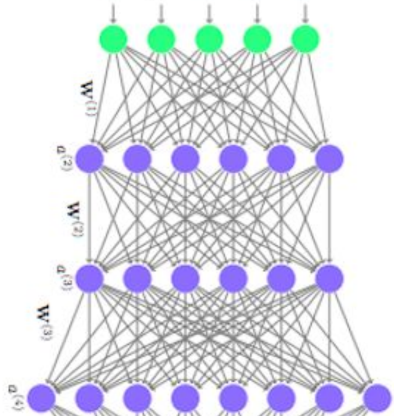
Low-dim, Original Space

High-dim, **Linearly Separated** Space

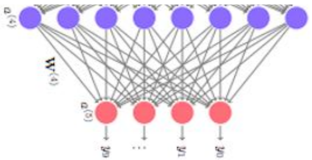
“Trick” in Deep Learning



Low-dim, Original Space

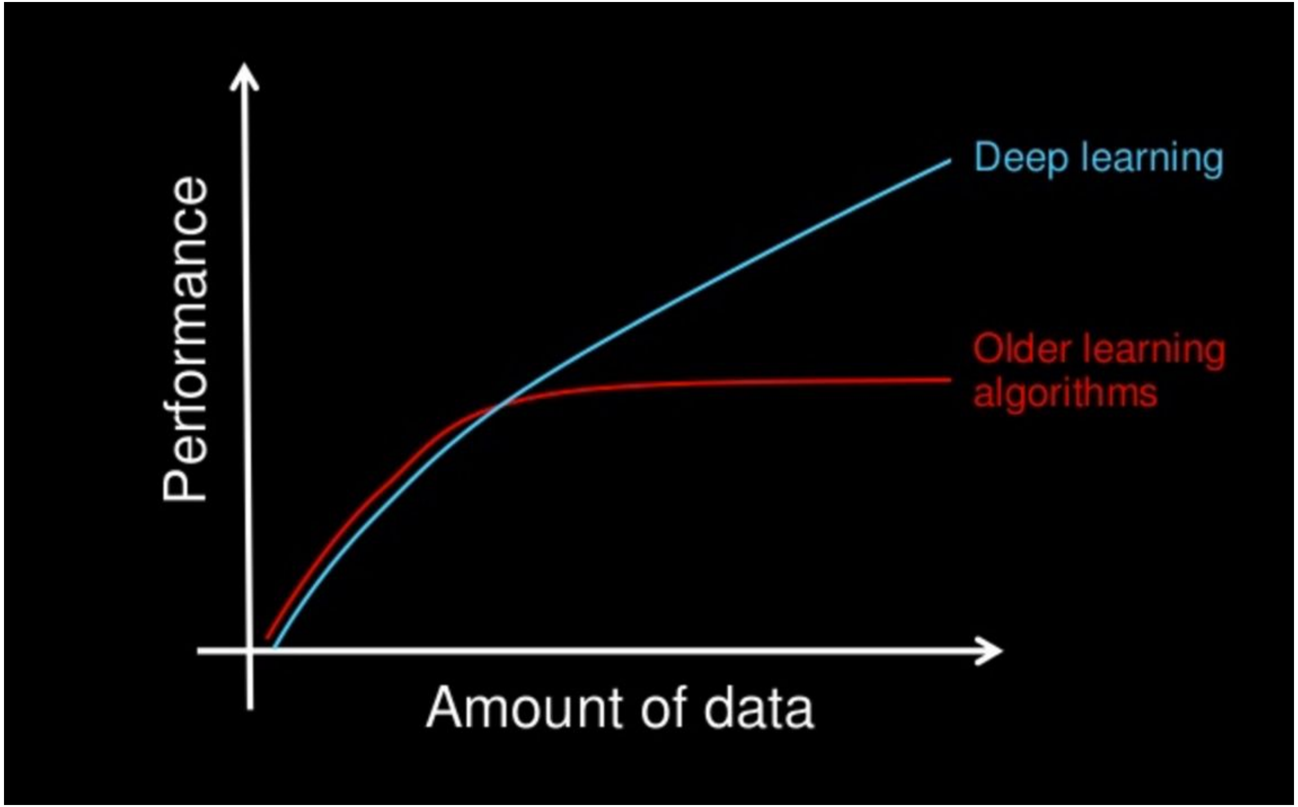


High-dim, **Linearly Separated** Space



Softmax Classifier
(Linear Model)

Why Deep Learning

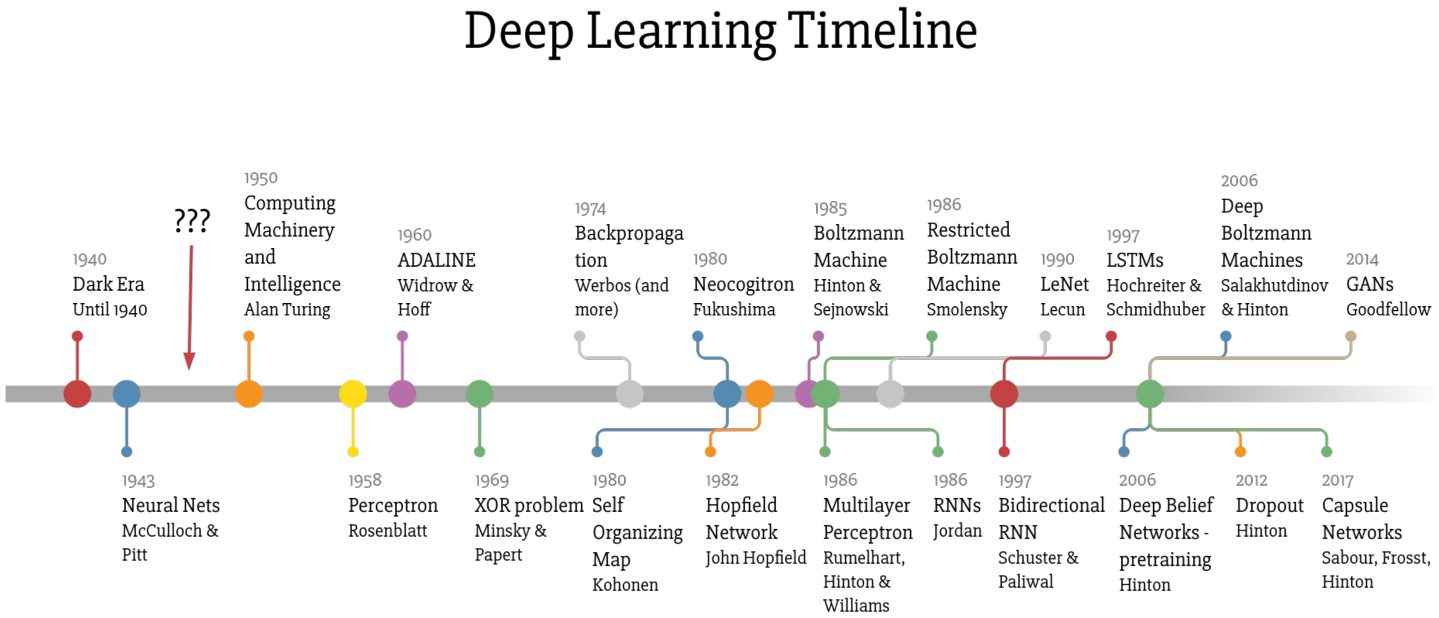


From Andrew Ng

Deep Learning

- Deep learning is a subfield of machine learning
- Most machine learning methods work well because of high-quality feature engineering/representation learning.
- Deep learning is an **end-to-end** structure, which supports automatic representation learning
- Different network structures: CNN, RNN, LSTM, GRU, Attention model, etc

DL/NN is not New



Made by Favio Vázquez

Why is Deep Learning Powerful Now?

- Feature engineering require high-level expert knowledge, which are easily over-specified and incomplete.
- Large amounts of training data
- Modern multi-core CPUs/GPUs/TPUs
- Better deep learning ‘tricks’ such as regularization, optimization, transfer learning etc.