

Applied Machine Learning for Business Analytics

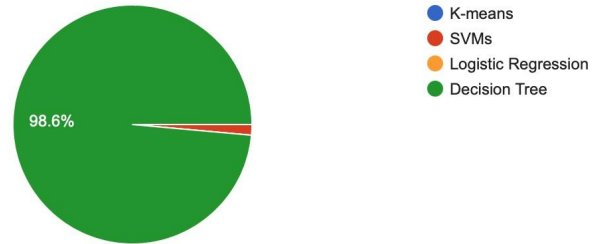
Lecture 3: Neural Networks and Deep Learning

Logistics

- Next Friday, Sanjay Saha will give a tutorial on the deep learning library: Keras after the lecture
- We will finalize the grouping information during this weekend
 - If your group ID is odd number, Sanjay will be your mentor
 - If your group ID is even number, Xiaohui will be your mentor
- The survey for learning mode has been sent out.
- Appreciate if you keeps video on!

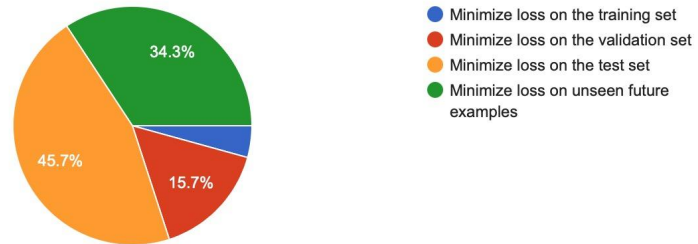
Which of these following machine learning models is not sensitive to scales of input features?

70 responses



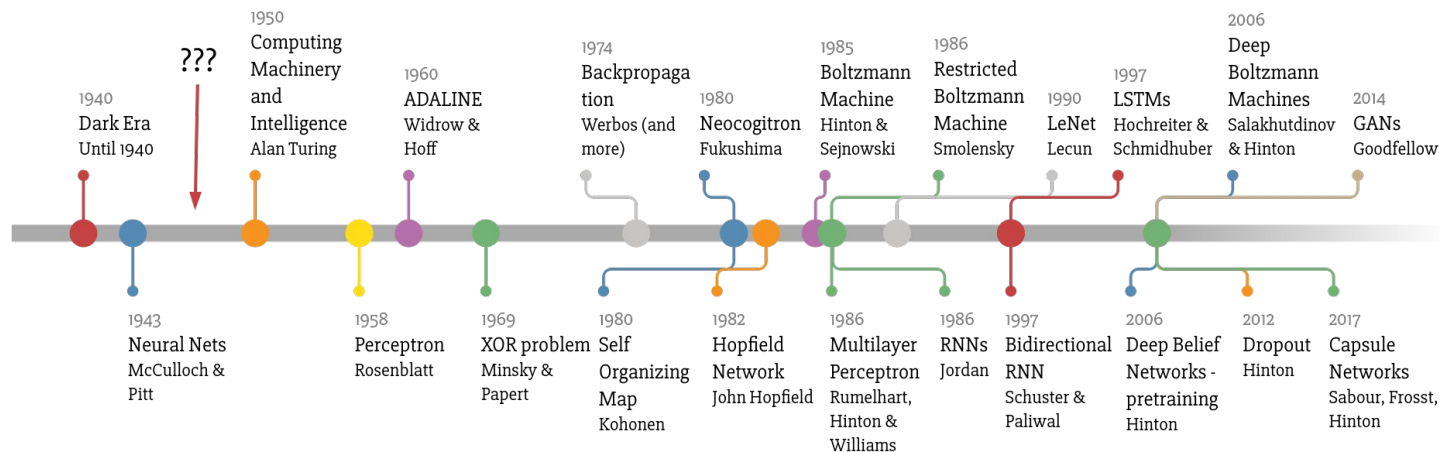
What should be the targets set for machine learning?

70 responses



DL/NN is not New

Deep Learning Timeline



Why DL is 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.

Three Steps in Deep Learning

To approximate the true function, define a function **space**

Neural network structures

Learning
Representation

Need a **measure** to evaluate the quality of each potential function in the previous space

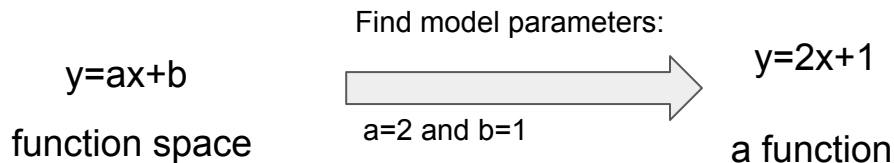
Evaluate the function

Objective Function

Search the function space to find the best function based on the measure.

Pick the best function

Optimization



Agenda

1. Linear Regression
2. Neural Networks
3. Evaluation of Functions
4. Optimization
5. Deep Learning
6. Application of DL

1. Linear Regression

Linear Regression (Single Variable)

- Model architecture: $y=ax+b$
- Objective function: Mean Squared Error Function

$$J(a, b) = \frac{1}{n} \sum_{i=0}^n (y_i - (ax_i + b))^2$$

- Optimization: Gradient Descent Algorithm

Gradient Descent

Gradient for the total loss
function over parameters,

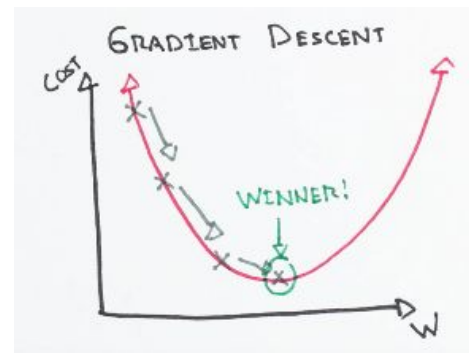
$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t)$$

New Parameters Guess Current Parameters Guess Learning Rate

Blue arrow points to $\nabla J(\theta_t)$



Like hiking down a mountain



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

Simple Math

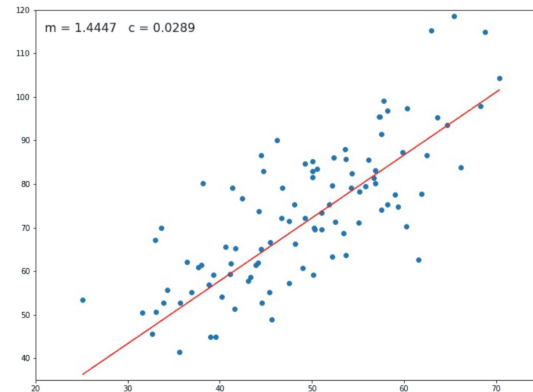
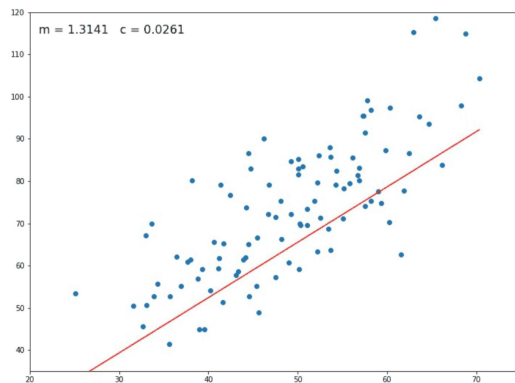
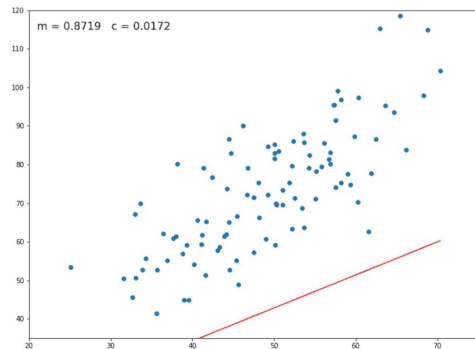
- Gradients for parameters:

$$\frac{\partial}{\partial a} J(a, b) = \frac{1}{n} \sum_{i=0}^n 2(y_i - (ax_i + b))(-x_i)$$

$$\frac{\partial}{\partial a} J(a, b) = \frac{-2}{n} \sum_{i=0}^n (y_i - y'_i)x_i$$

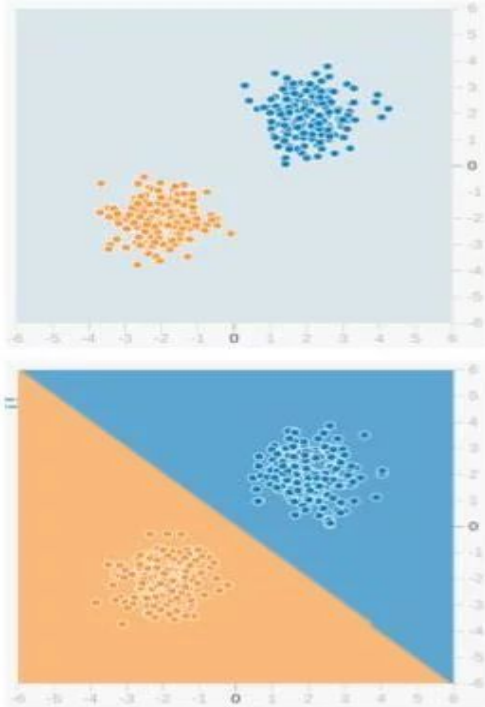
$$\frac{\partial}{\partial b} J(a, b) = \frac{-2}{n} \sum_{i=0}^n (y_i - y'_i)$$

Optimization



2. Neural Networks

A “Simple” Classification Problem



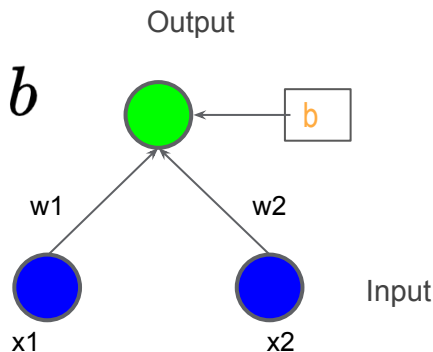
$$y = \mathbf{w}\mathbf{x} + b$$

A Linear Model

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

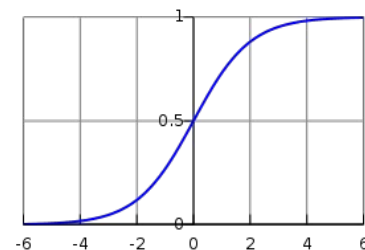
Linear Regression

$$y = w_1 x_1 + w_2 x_2 + b$$



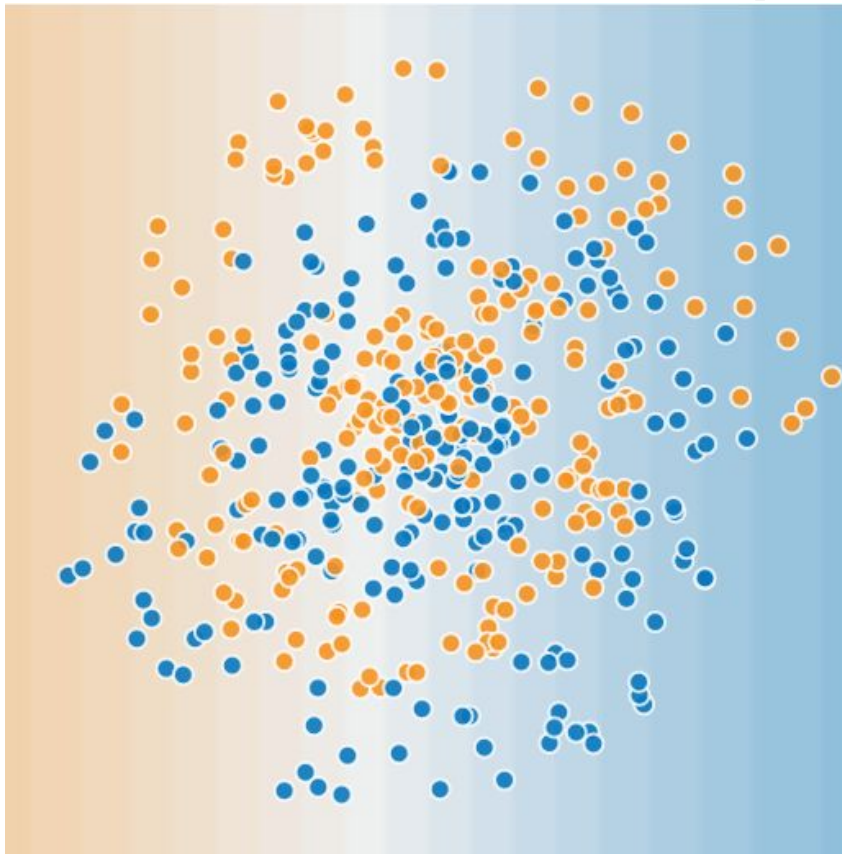
Logistic Regression

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



How about this classification problem?

Linear model can
not solve it



We need **non-linear
functions**

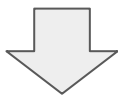
Add Complexity

For Simplicity, the bias term is ignored here.

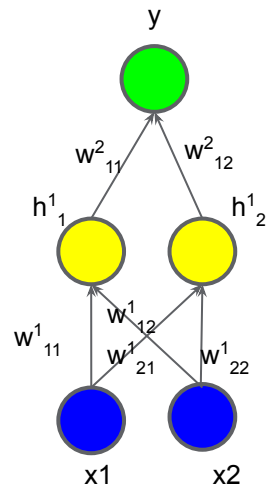
$$h_1^1 = w_{11}^1 x_1 + w_{12}^1 x_2$$

$$h_2^1 = w_{21}^1 x_1 + w_{22}^1 x_2$$

$$y = w_{11}^2 h_1^1 + w_{12}^2 h_2^1 \quad \mathbf{=} \quad y = \mathbf{W} \mathbf{X}$$



$$y = (w_{11}^2 w_{11}^1 + w_{21}^2 w_{12}^1) x_1 + (w_{12}^2 w_{11}^1 + w_{22}^2 w_{12}^1) x_2$$



Output

Hidden Layer

Input

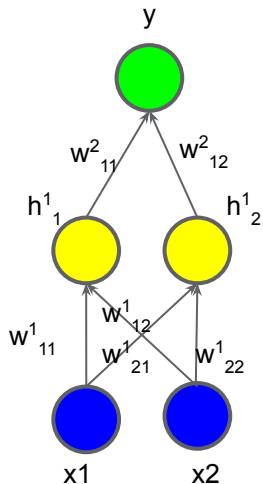
Add Complexity

Associative Law

$$h_1^1 = w_{11}^1 x_1 + w_{12}^1 x_2$$

$$h_2^1 = w_{21}^1 x_1 + w_{22}^1 x_2$$

$$y = w_{11}^2 h_1^1 + w_{12}^2 h_2^1$$



Matrix Format

Output

$$\begin{bmatrix} h_1^1 \\ h_2^1 \end{bmatrix} = \begin{bmatrix} w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Hidden Layer

$$y = \begin{bmatrix} w_{11}^2 & w_{12}^2 \end{bmatrix} \begin{bmatrix} h_1^1 \\ h_2^1 \end{bmatrix}$$

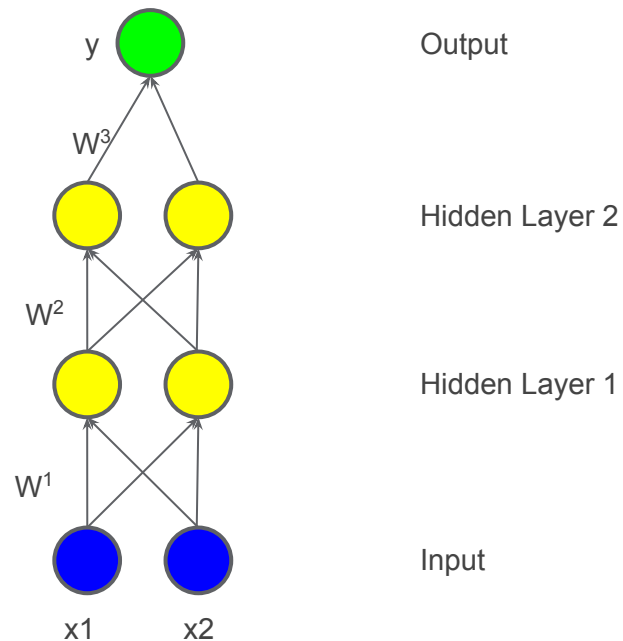
Input



$$y = W^2 W^1 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = (W^2 W^1) \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = W \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Add Complexity

$$y = \mathbf{W}^3 \mathbf{W}^2 \mathbf{W}^1 \begin{bmatrix} x1 \\ x2 \end{bmatrix} = (\mathbf{W}^3 \mathbf{W}^2 \mathbf{W}^1) \begin{bmatrix} x1 \\ x2 \end{bmatrix}$$



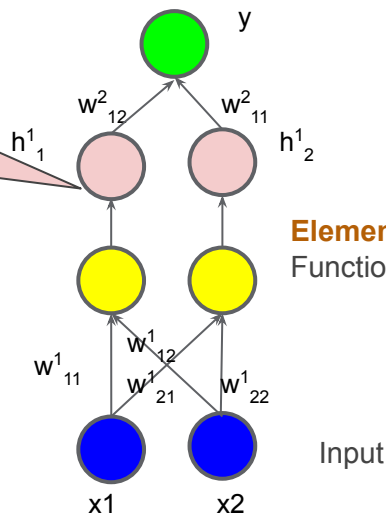
Make it non-linear

$$h_1^1 = \sigma(w_{11}^1 x_1 + w_{12}^1 x_2)$$

$$h_2^1 = \sigma(w_{21}^1 x_1 + w_{22}^1 x_2)$$

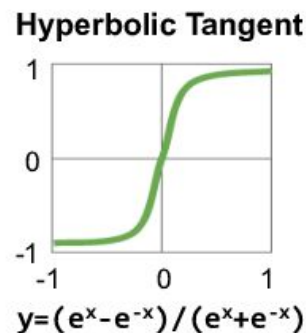
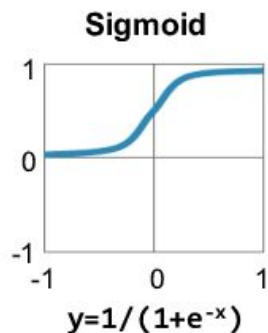
$$y = w_{11}^2 h_1^1 + w_{12}^2 h_2^1 \neq y = \mathbf{w}\mathbf{x}$$

We Usually Don't
Draw Non-Linear
Transforms

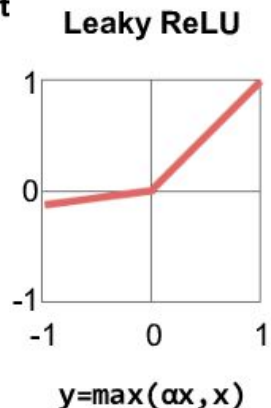
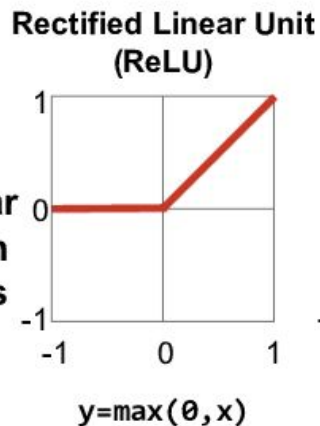


Non-linear Activation Functions

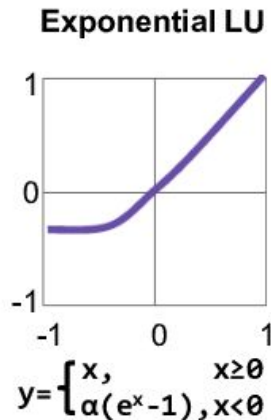
Traditional
Non-Linear
Activation
Functions



Modern
Non-Linear
Activation
Functions

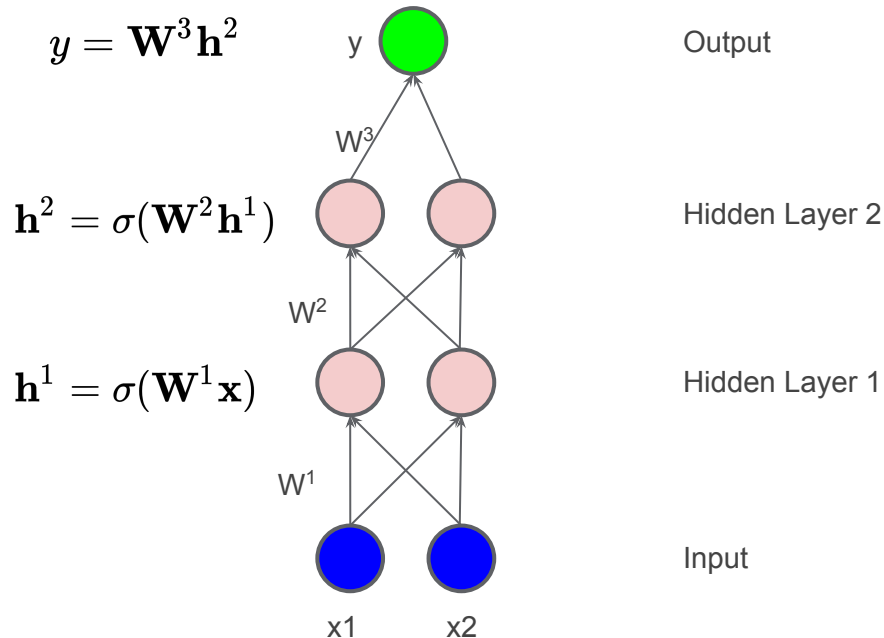


α = small const. (e.g. 0.1)



Add Non-linear Activation Function

$$y = \mathbf{W}^3 \sigma(\mathbf{W}^2 \sigma(\mathbf{W}^1 \mathbf{x}))$$

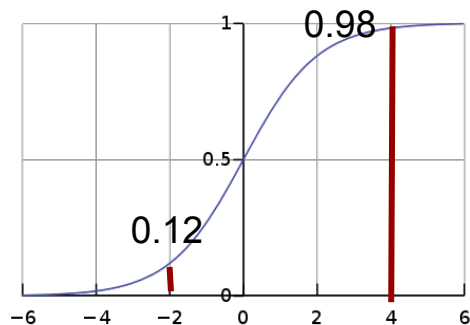


Why Non-linear Activation

- 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.

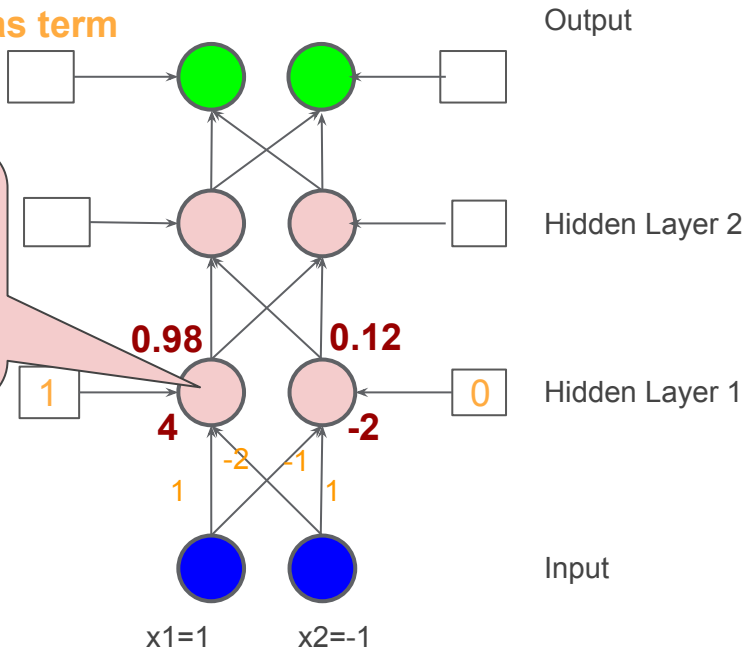
Forward Computation



Sigmoid function:

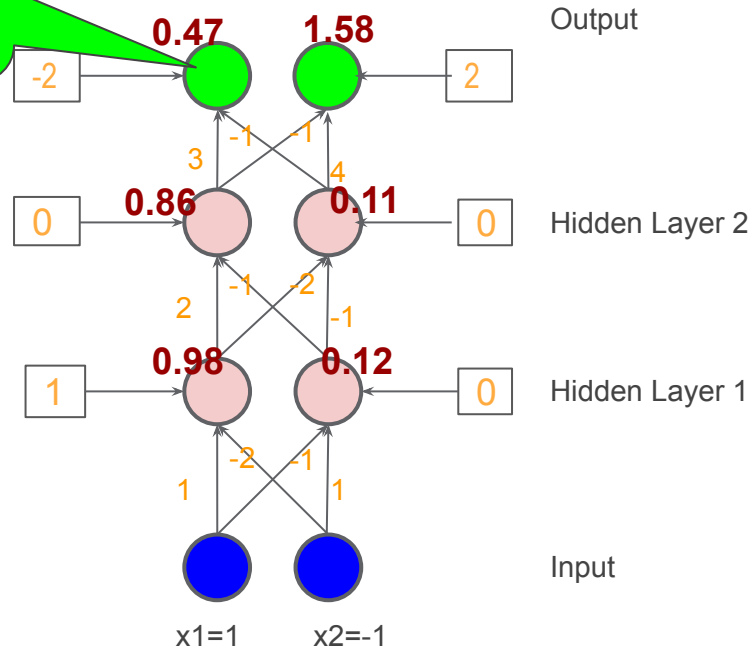
$$h = \sigma(z) = \frac{1}{1+e^{-z}}$$

bias term



Forward Computation

Identity Function. It can be non-linear functions specified by applications.



Forward Computation

1. Neural Network acts as a function that transforms the input vector into the output vector (target)

$$\begin{bmatrix} 0.47 \\ 1.58 \end{bmatrix} = f_{\theta} \left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} \right)$$

$$\begin{bmatrix} 0.04 \\ 1.7 \end{bmatrix} = f_{\theta} \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} \right)$$

$$\mathbf{W}^1 = \begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix} \quad \mathbf{b}^1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\mathbf{W}^2 = \begin{bmatrix} 2 & -1 \\ -2 & -1 \end{bmatrix} \quad \mathbf{b}^2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

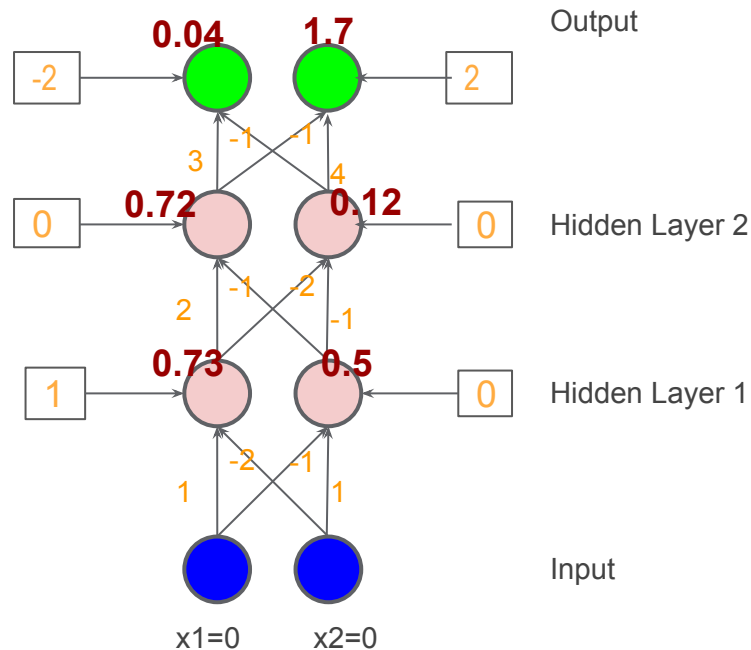
$$\mathbf{W}^3 = \begin{bmatrix} 3 & -1 \\ -1 & 4 \end{bmatrix} \quad \mathbf{b}^3 = \begin{bmatrix} -2 \\ 2 \end{bmatrix}$$

one param. set

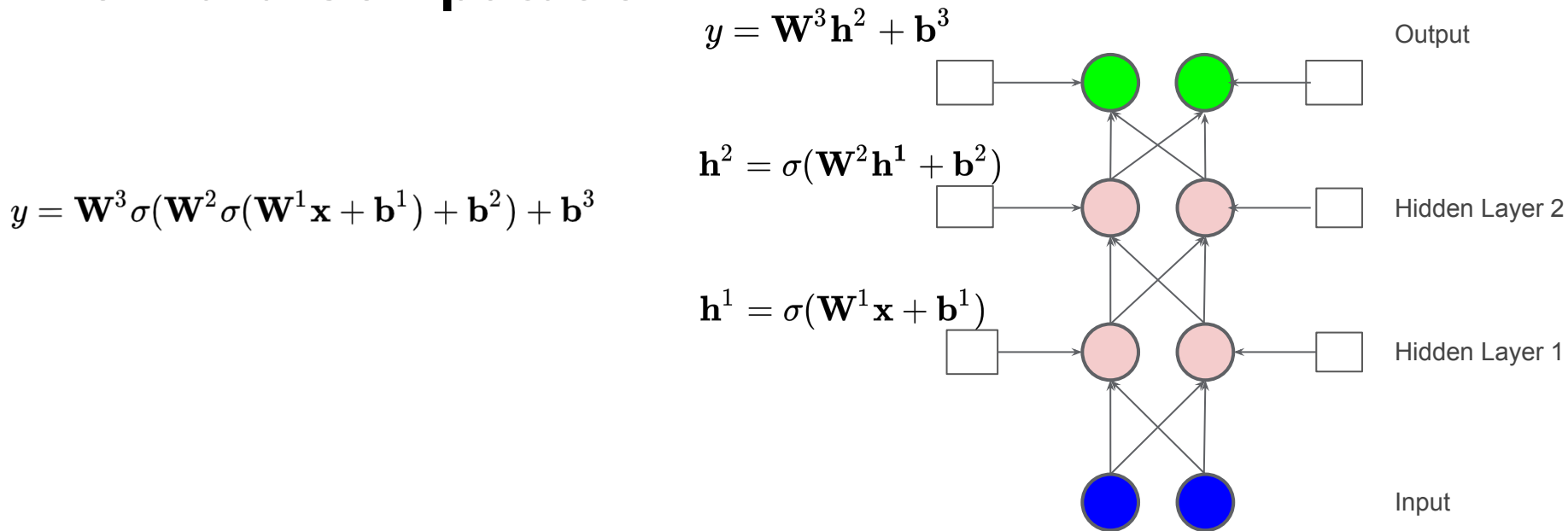
2. It is actually a function space parameterized by weights matrices and bias vectors.

$$\theta = \{\mathbf{W}^1, \mathbf{b}^1, \mathbf{W}^2, \mathbf{b}^2, \mathbf{W}^3, \mathbf{b}^3\}$$

Parameter space

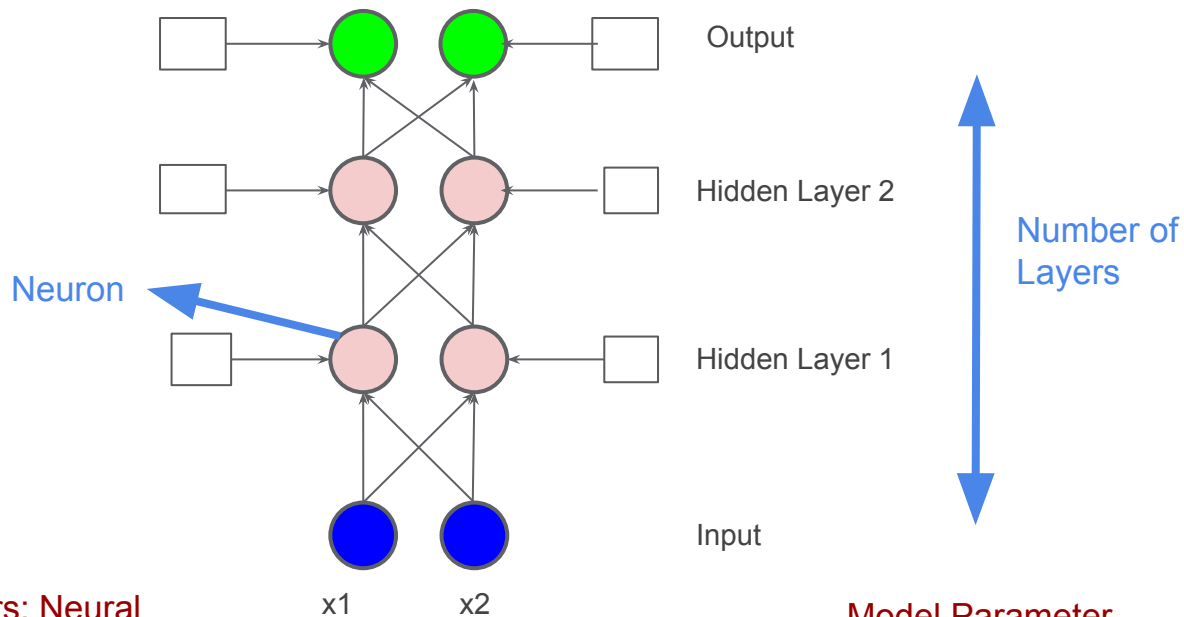


Forward Computation



1. Neural Network is a model that **recursively** applies the matrix multiplication and non-linear activation function.
2. Parallel computing techniques can be used to speed up matrix operation.

Neural Network: Function Set



Hyperparameters: Neural
Network Structure

Model Parameter

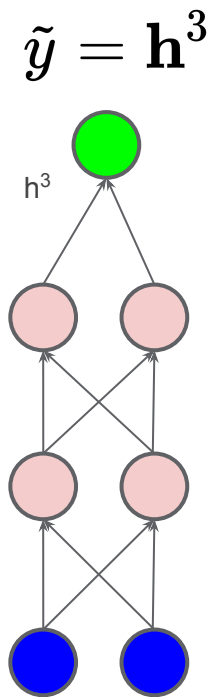
1. Number of Layers
2. Number of neurons in each layer
3. Non-linear Activation function in each layer

**A function space
containing various
functions**

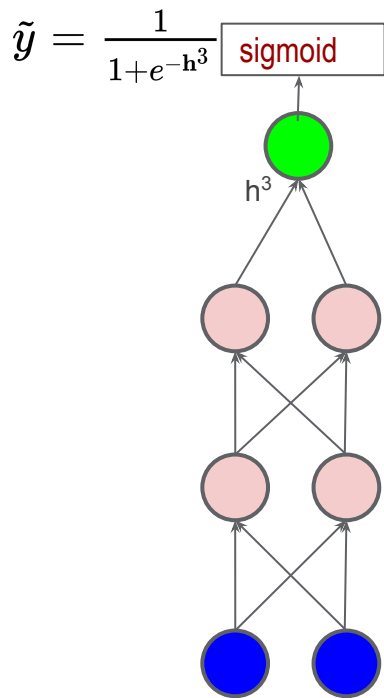
**A specific function
mapping from input
data to targets.**

Output Layer

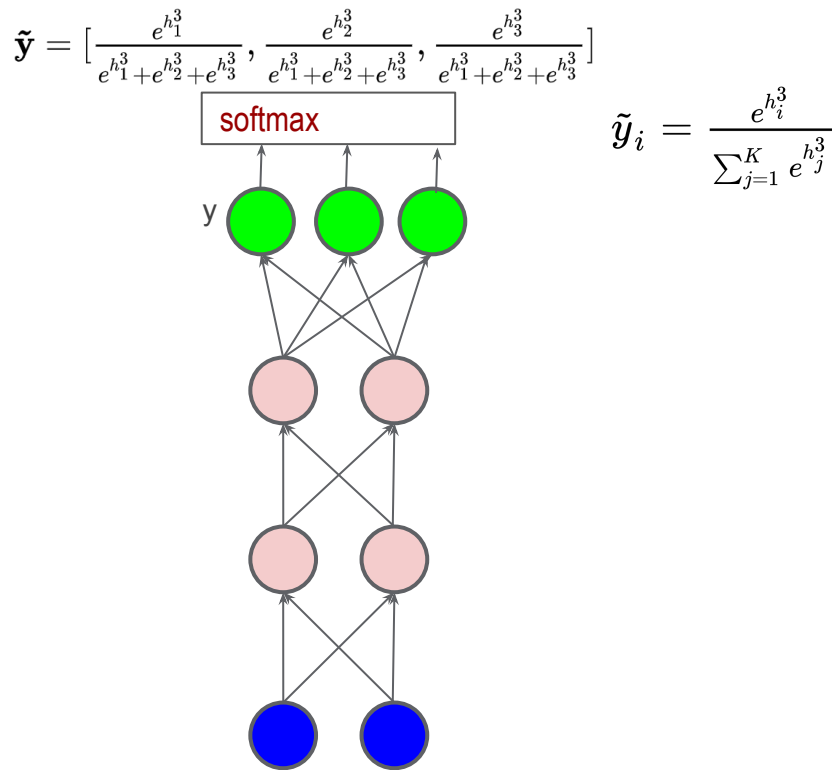
Regression



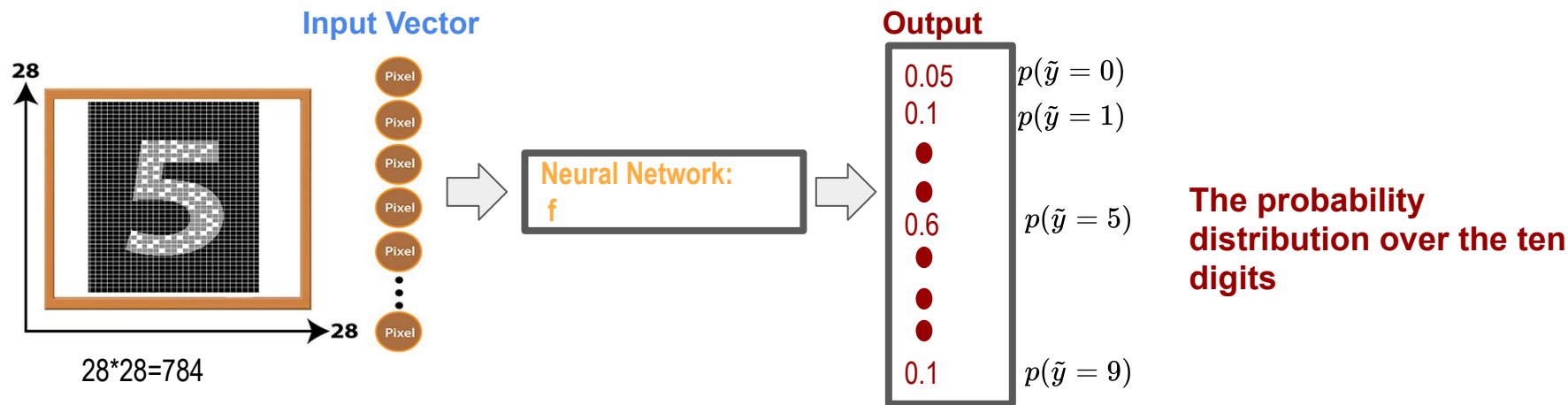
Binary Classification



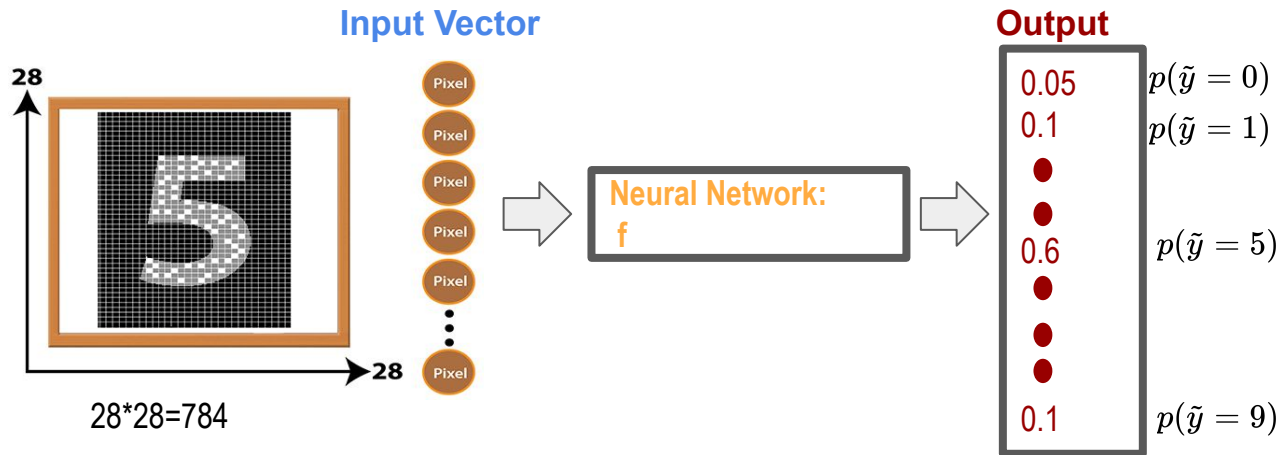
Multi-label Classification



Example: MNIST Dataset



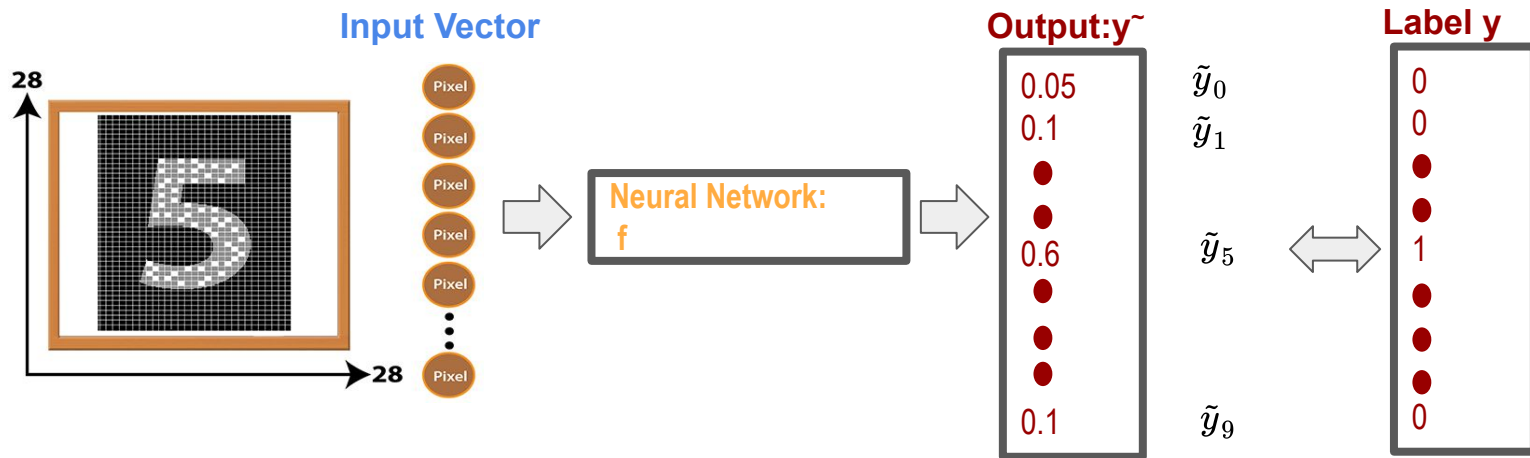
Example: MNIST Dataset



1. In this task, the neural network is a function mapping from the input 784-dim vector to the output 10-dim vector.
2. The neural network structure should be decided to make sure the best function exists in the function set.

3. Evaluation of Functions

Cross-Entropy Loss



Given a set of parameters and one training sample,

$$loss(\tilde{y}, y) = - \sum_{i=0}^9 y_i \ln(\tilde{y}_i)$$

Total Loss

- Training dataset contains N training samples
- The total loss is: $J = \sum_{n=1}^N \text{loss}(\tilde{\mathbf{y}}_n, \mathbf{y}_n)$
- Find a function in the function set that minimizes the total loss J
- Find the network parameters θ that minimizes the total loss J . Modern

For loss function, training data are fixed and model parameters are unknown.

$$\operatorname{argmin}_{\theta} J$$

4. Optimization

Gradient Descent

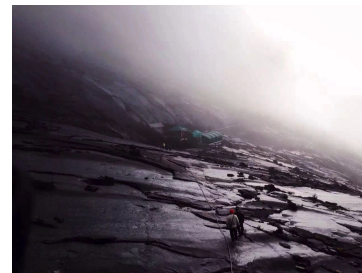
Gradient for the total loss
function over parameters,
which computed by
Backpropagation algorithm

$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t)$$

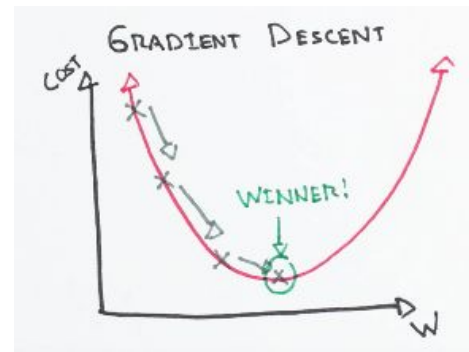
New Parameters Guess

Current Parameters Guess

Learning Rate

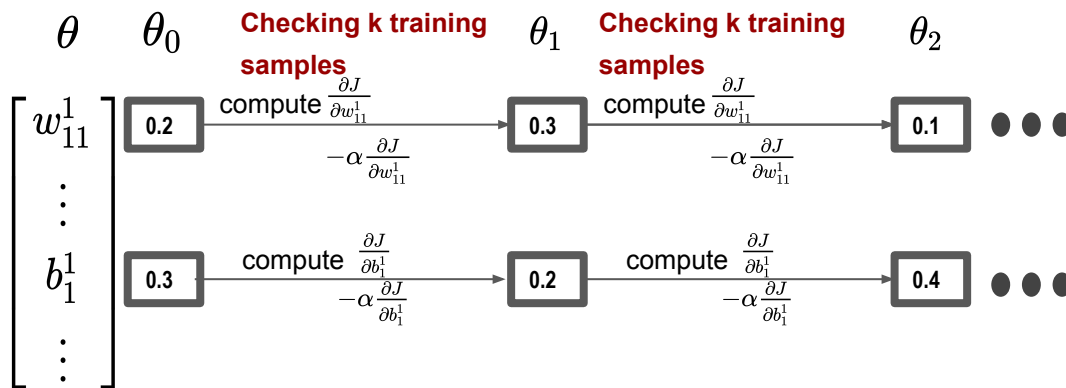


Like hiking down a mountain



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

Gradient Descent



Backpropagation is used to compute gradients in an efficient way. $\frac{\partial J}{\partial \theta}$

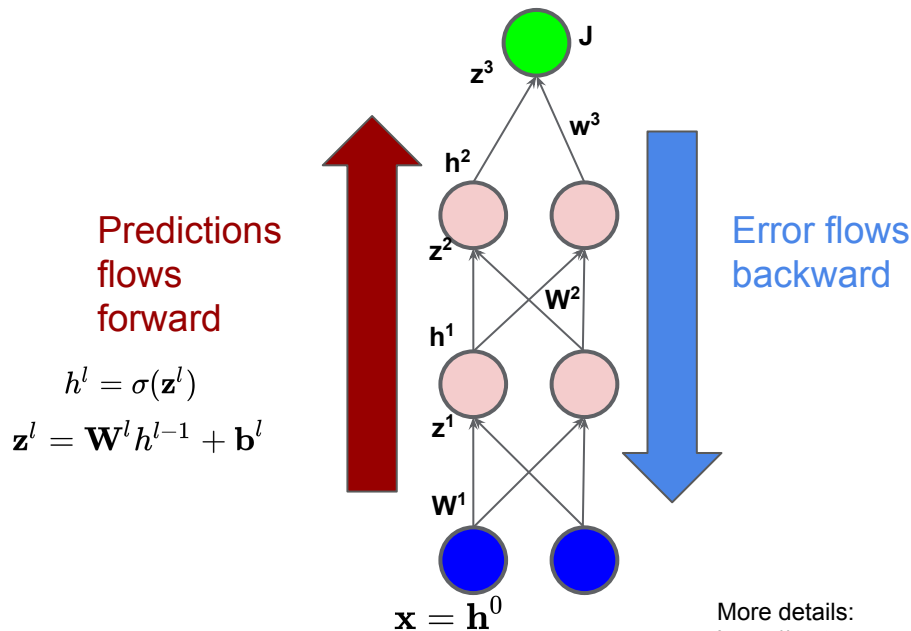
Batch size: **k**

A dataset is [1,2,3,4,5,6] and the batch size is 2, one batch shuffle could be:
batch0=[2,1], batch1=[3,6], batch2=[4,5]

Backpropagation

Definition (from wiki):

By computing the gradient of the loss function with respect to each weight by the **chain rule**, computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule



$$\frac{\partial J}{\partial \mathbf{w}^3} = \frac{\partial J}{\partial \mathbf{z}^3} \frac{\partial \mathbf{z}^3}{\partial \mathbf{w}^3}$$

$$\frac{\partial J}{\partial \mathbf{W}^2} = \frac{\partial J}{\partial \mathbf{z}^3} \frac{\partial \mathbf{z}^3}{\partial \mathbf{h}^2} \frac{\partial \mathbf{h}^2}{\partial \mathbf{z}^2} \frac{\partial \mathbf{z}^2}{\partial \mathbf{W}^2}$$

$$\frac{\partial J}{\partial \mathbf{W}^1} = \frac{\partial J}{\partial \mathbf{z}^3} \frac{\partial \mathbf{z}^3}{\partial \mathbf{h}^2} \frac{\partial \mathbf{h}^2}{\partial \mathbf{z}^2} \frac{\partial \mathbf{z}^2}{\partial \mathbf{h}^1} \frac{\partial \mathbf{h}^1}{\partial \mathbf{z}^1} \frac{\partial \mathbf{z}^1}{\partial \mathbf{W}^1}$$

From \mathbf{w}^3

From \mathbf{w}^3

From \mathbf{W}^2

More details:

<https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>

Batch Size

Three approaches to select batch sizes:

1. Batch Gradient Descent

batch size = Number of training data

2. Mini-batch Gradient Descent

$1 < \mathbf{batch\ size} < \text{number of training data}$

3. Stochastic Gradient Descent

batch size = 1

Training Process

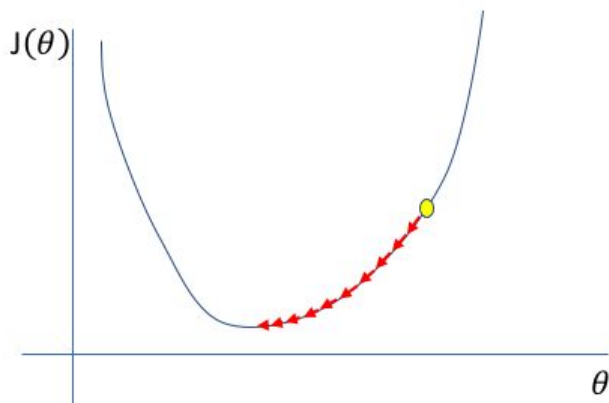
- Initialize neural network randomly
- For _ in range(number of epoch):
 - Shuffle all the training dataset into a list of batches
 - For _ in range(number of batches)
 - Get output with the input data in the batch
 - Compare outputs with ground truth in training data
 - Compute loss function with the batch data
 - Update weights with backpropagation and gradient descent algorithm

**Iteratively
perform**



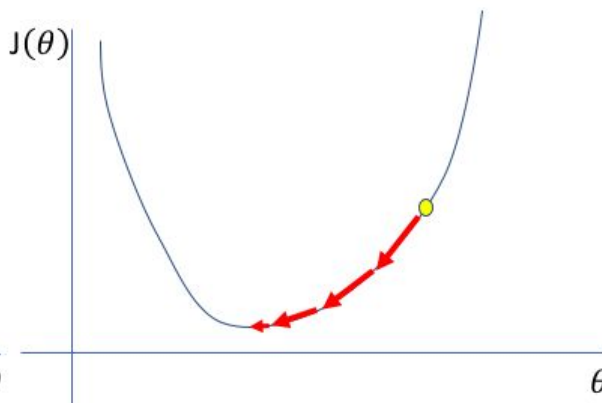
Learning Rate

Too low



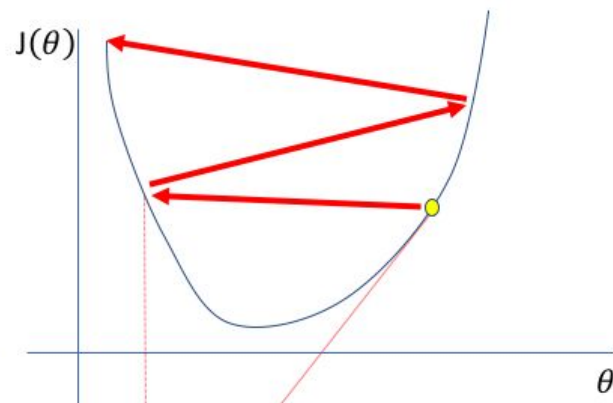
A small learning rate requires many updates before reaching the minimum point

Just right



The optimal learning rate swiftly reaches the minimum point

Too high



Too large of a learning rate causes drastic updates which lead to divergent behaviors

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

Except SGD

SGD

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

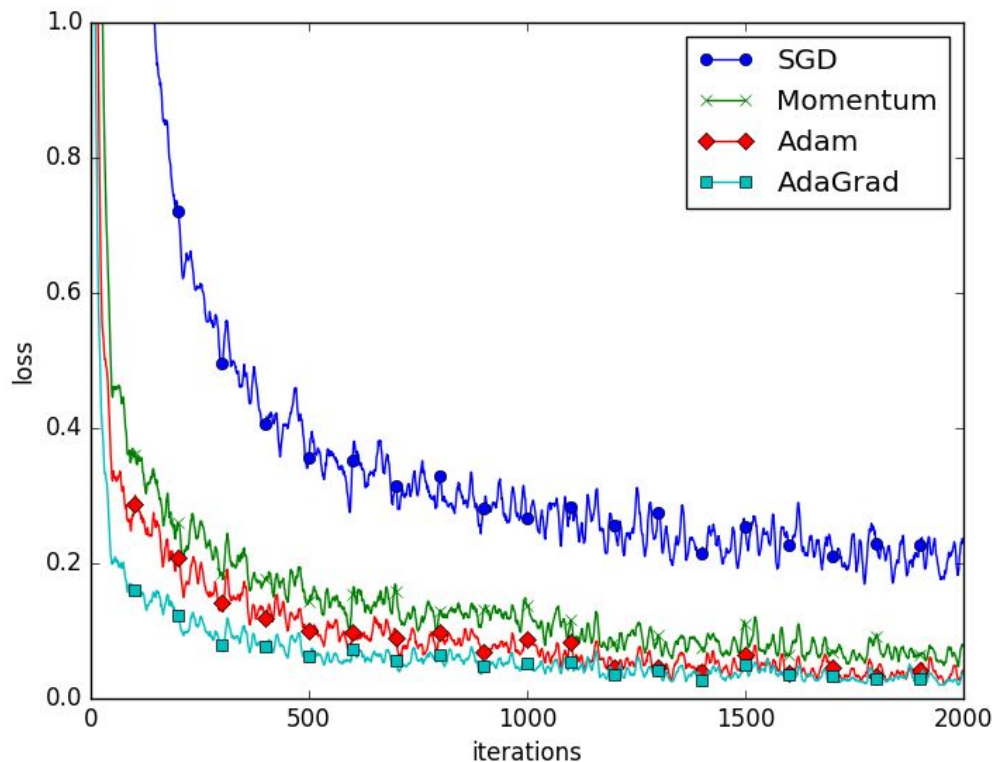


Different Variants

Momentum, Adam, AdaGrad, RMSProp



Auto-tune learning rates



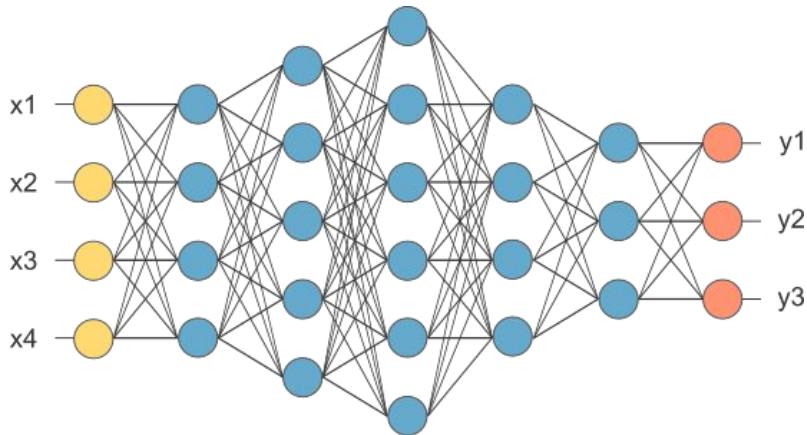
Neural Network Visualization

Playground

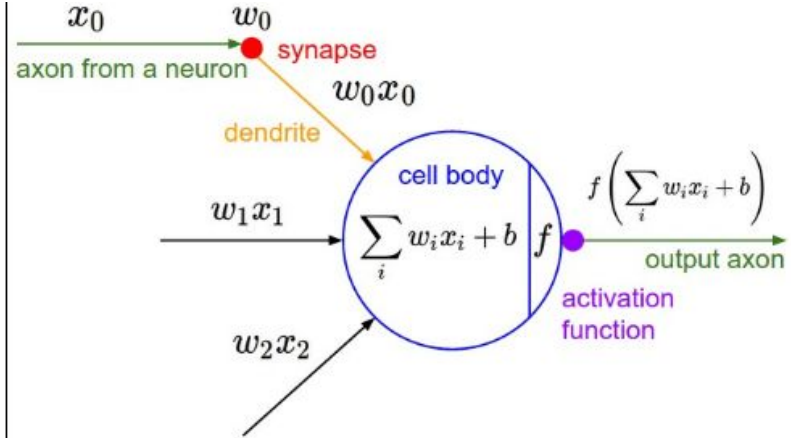
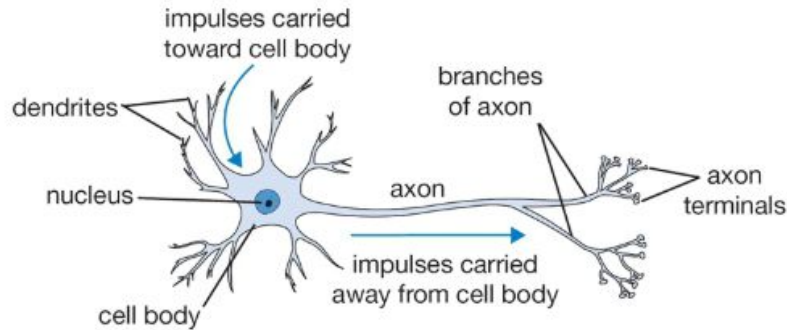
5. Deep Learning

Neural Network

- 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.
- From another way:
 - NN is running several 'logistic regression' at the same time (expanding at width and depth dimensions).



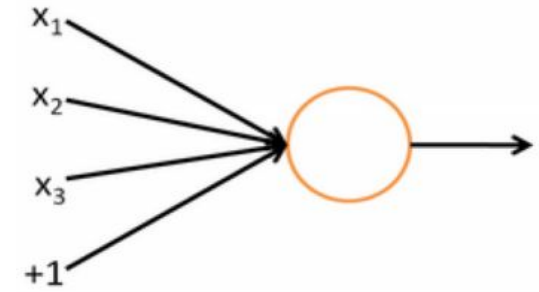
Neural Computation



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

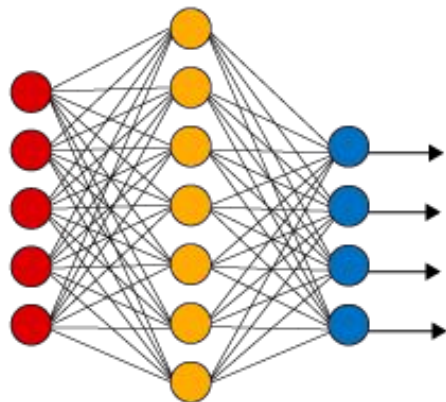
The fact that a neuron is essentially a logistic regression unit:

- 1 performs a dot product with the input and its weights
- 2 adds the bias and apply the non-linearity



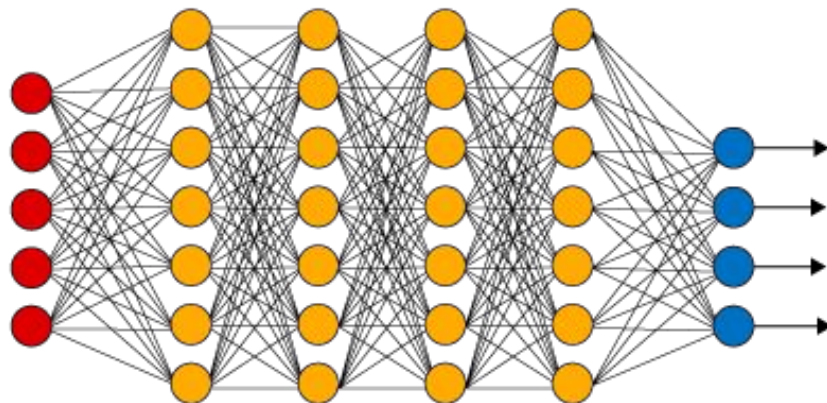
Shallow vs Deep

Simple Neural Network



● Input Layer

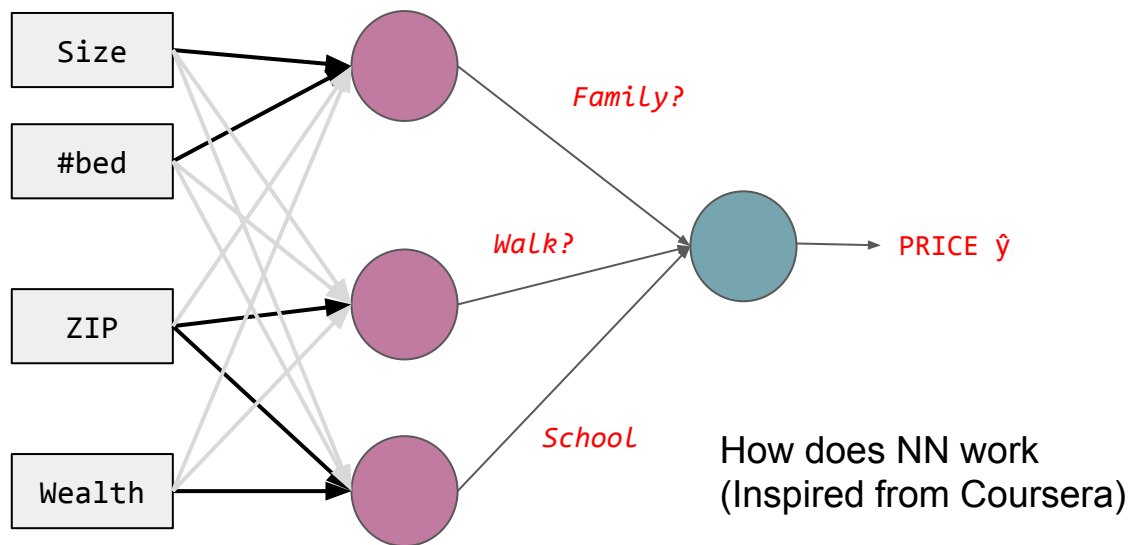
Deep Learning Neural Network



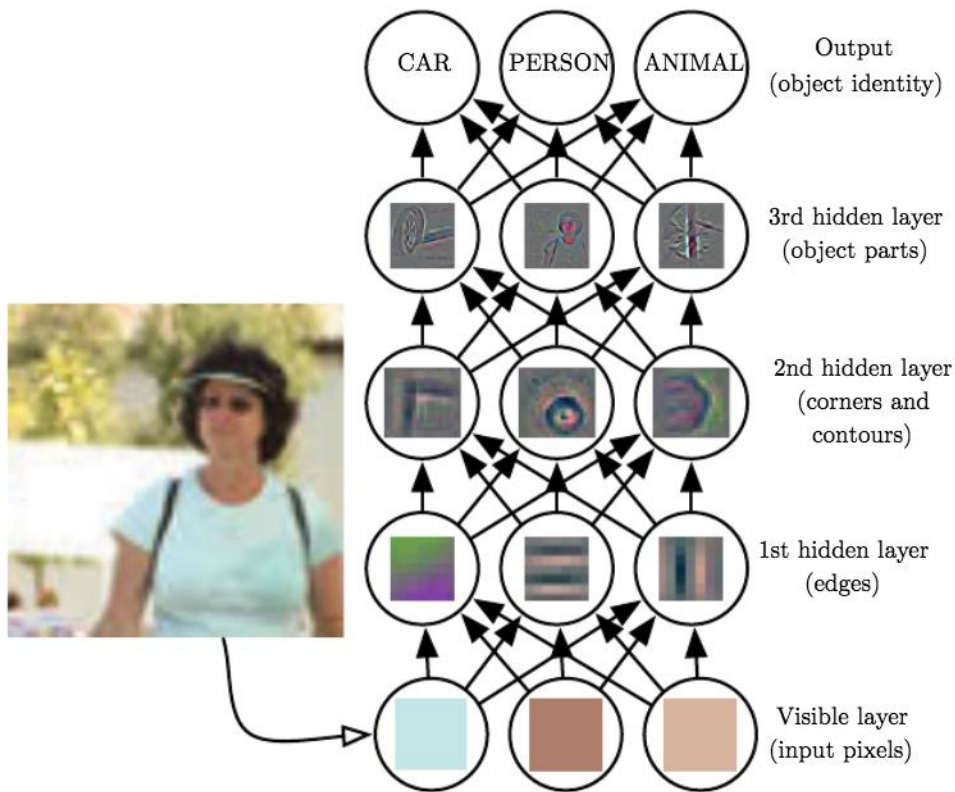
● Hidden Layer

● Output Layer

Representation Learning in DL

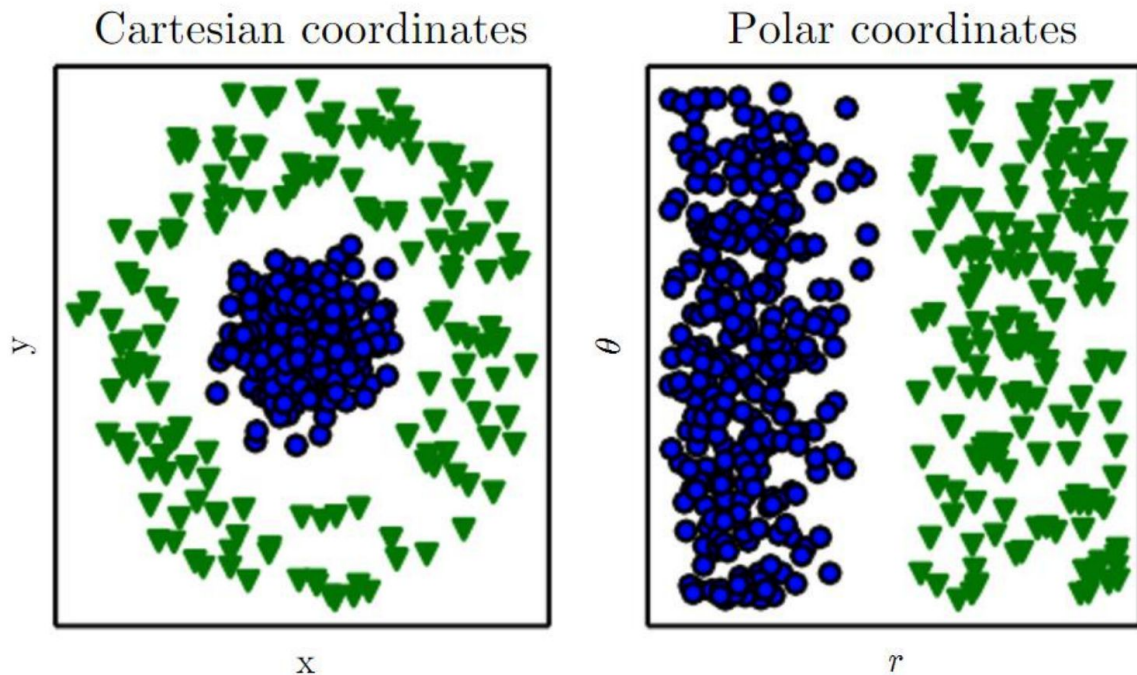


Representation Learning in DL

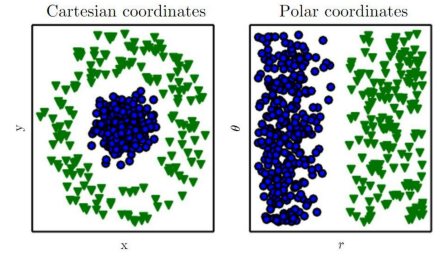


From Deep Learning (Goodfellow)

Representation Matters

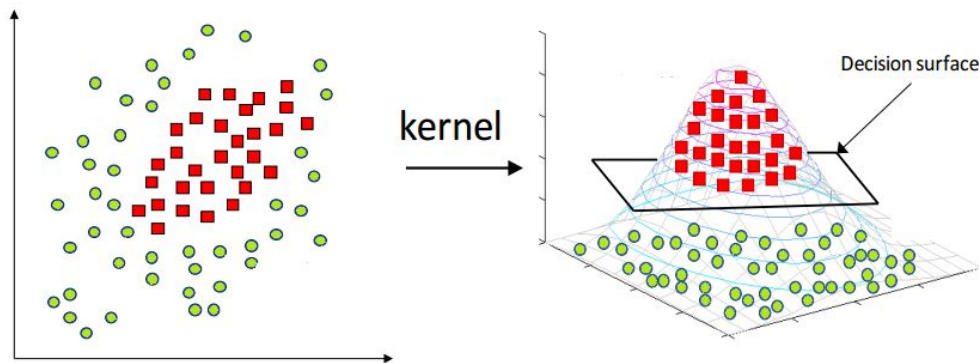


Task: Draw a line to separate the **green triangles** and **blue circles**.



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

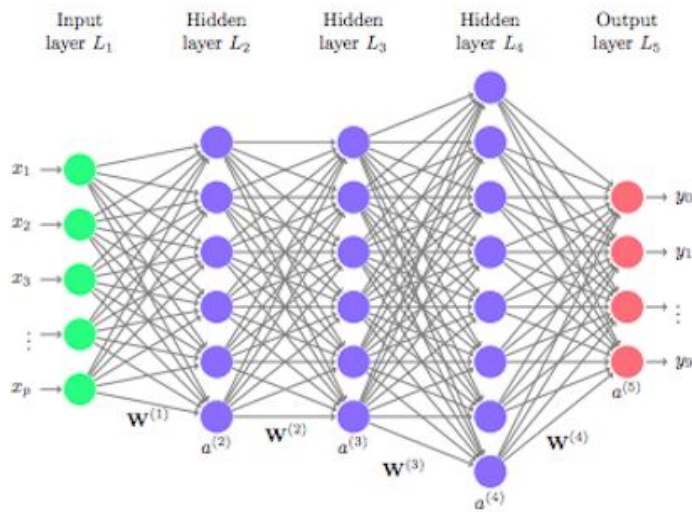
Kernel Tricks in SVM



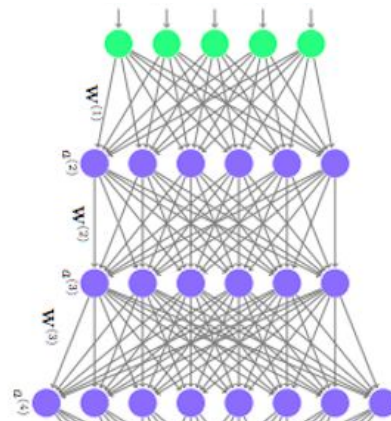
Low-dim, Original Space

High-dim, **Linearly Separated** Space

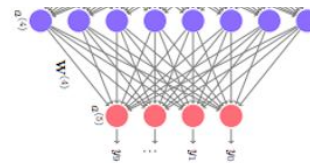
Hidden Representation in Deep Learning



Low-dim, Original Space

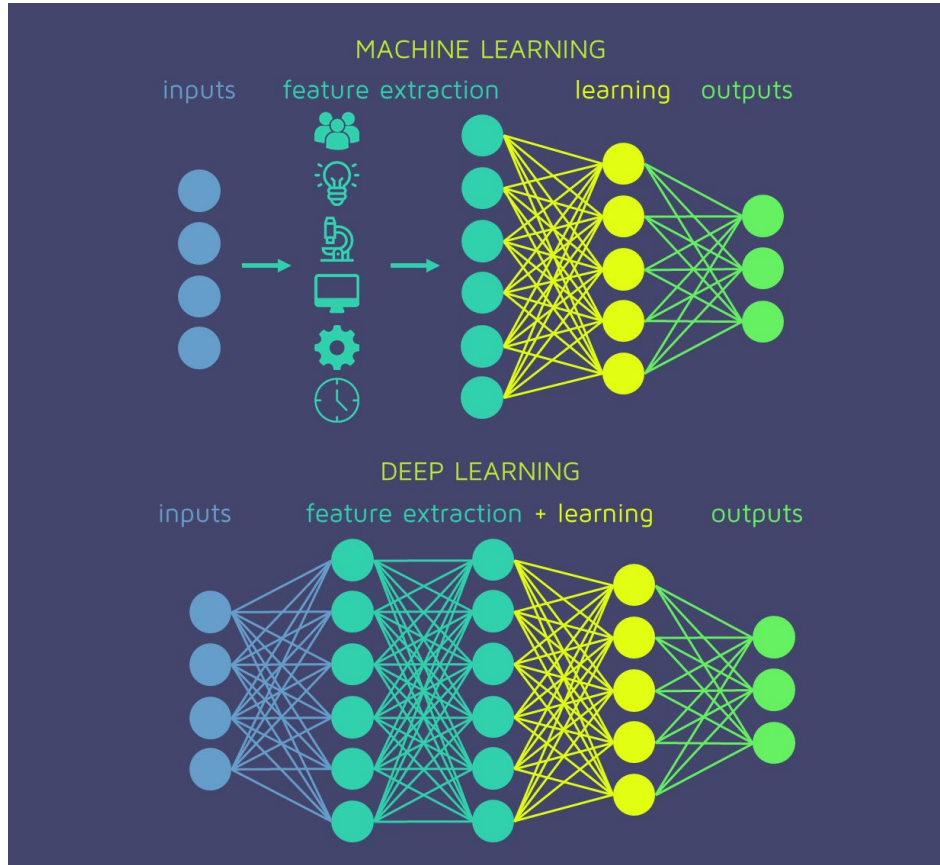


High-dim, **Linearly Separated** Space



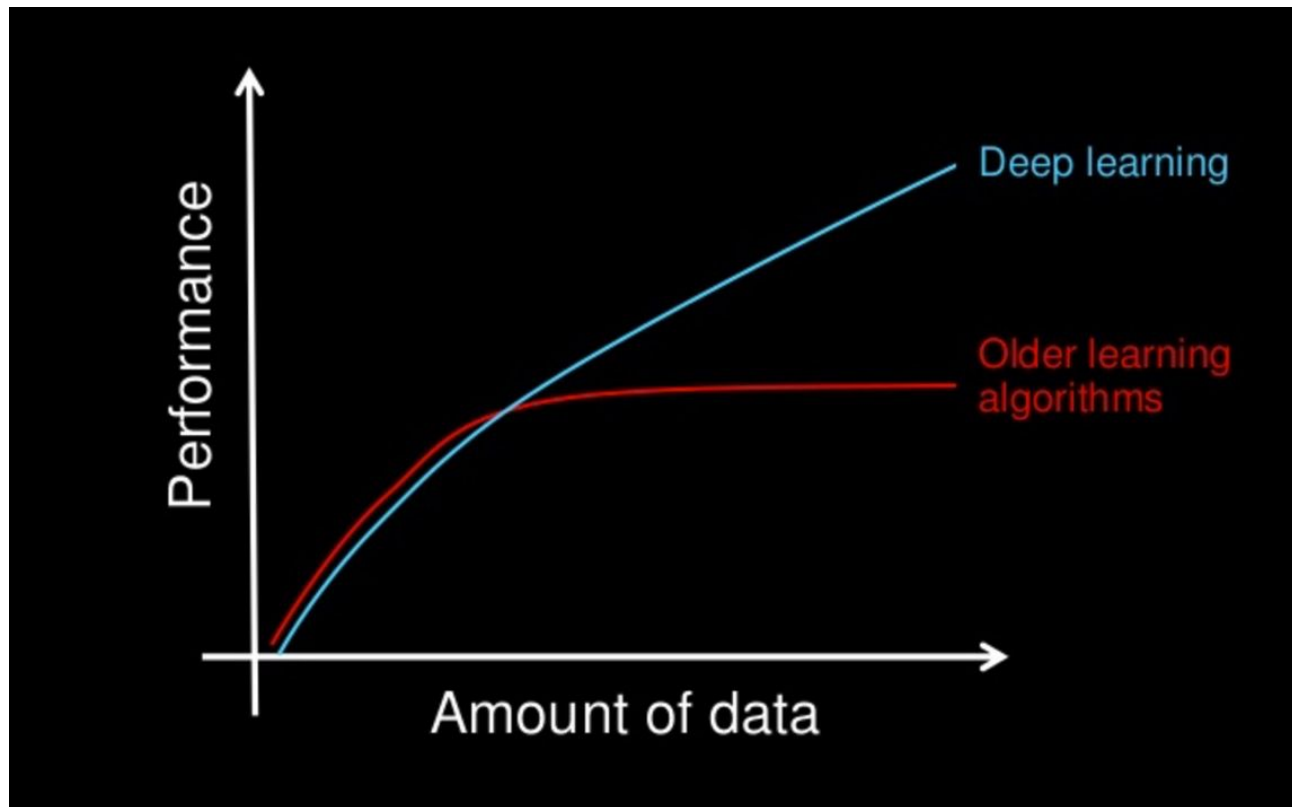
Softmax Classifier
(Linear Model)

End-to-End Learning



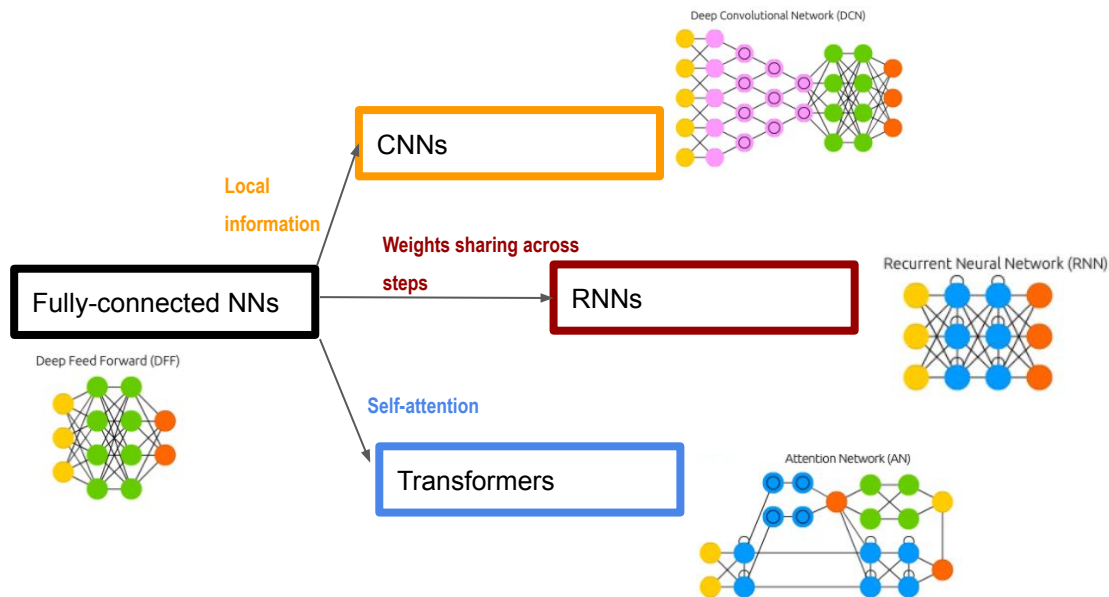
From Aporras

Why Deep Learning



From Andrew Ng

Deep Learning Structures



<https://www.asimovinstitute.org/author/fjodorvanveen/>

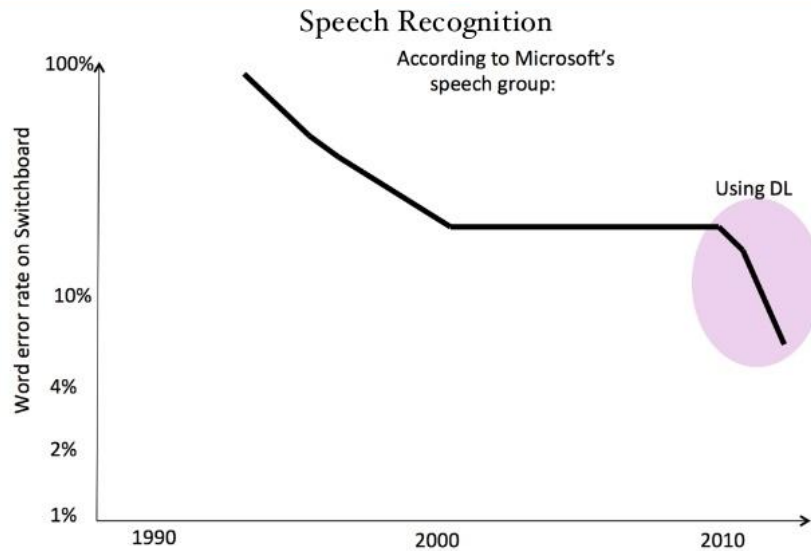
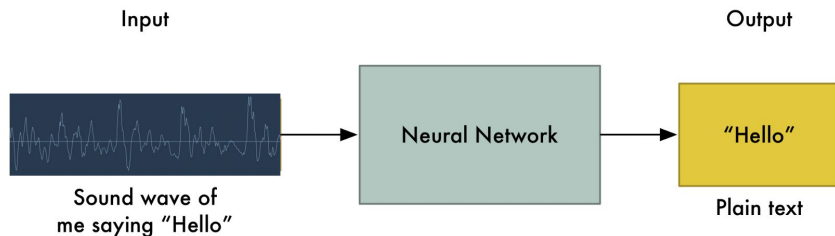
Deep Learning

- 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.

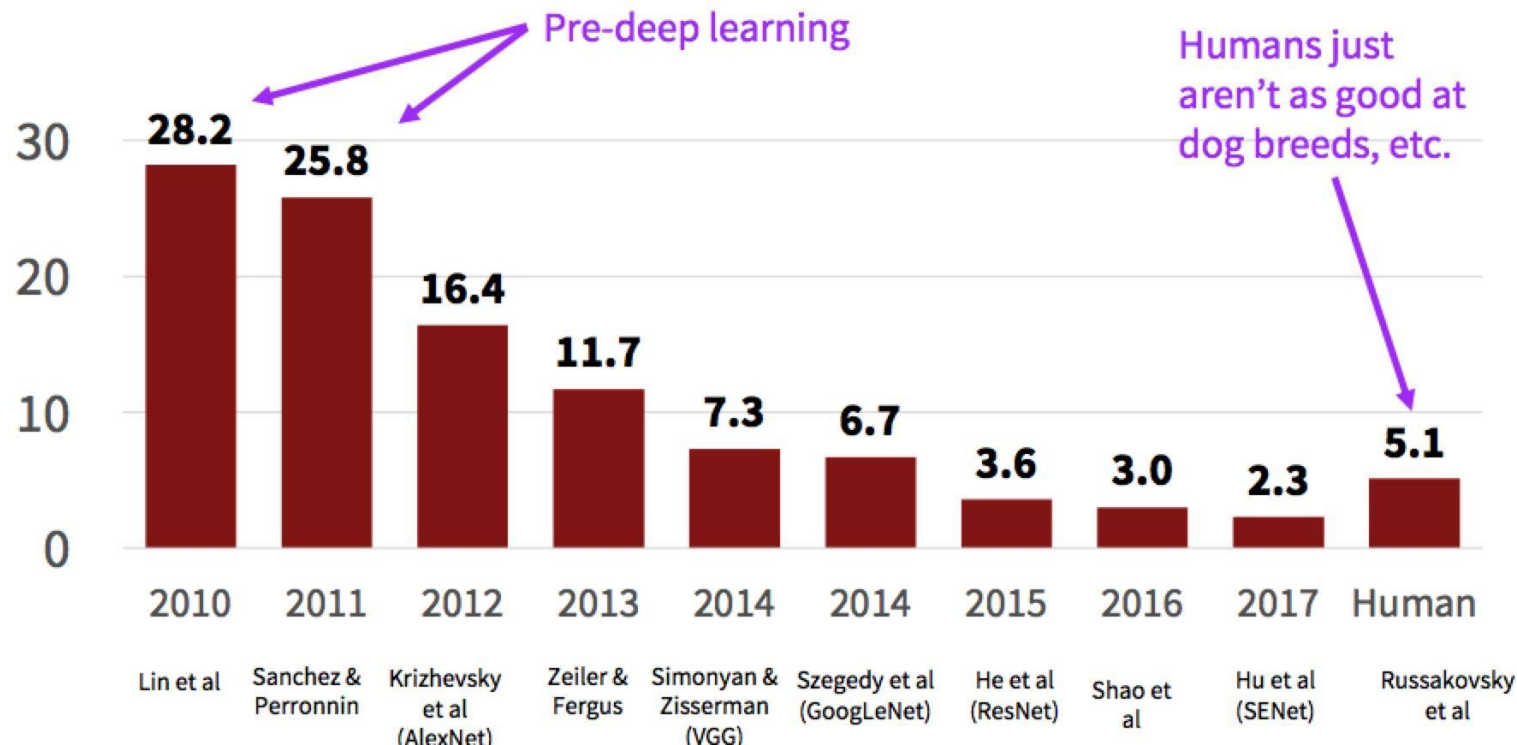
6. Applications of DL

Deep Learning for Speech

The first real-world tasks addressed by deep learning is speech recognition



Deep Learning for Computer Vision



Deep Learning for Arts



Original photo

Reference photo

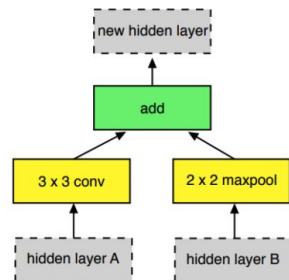
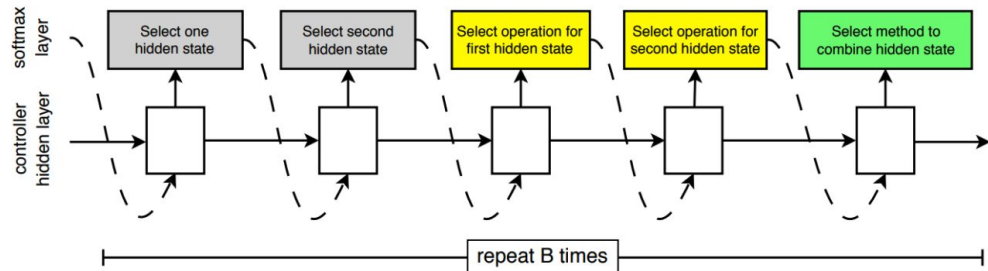
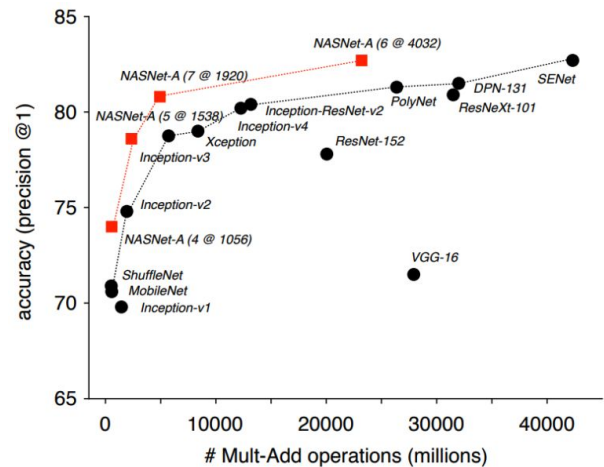
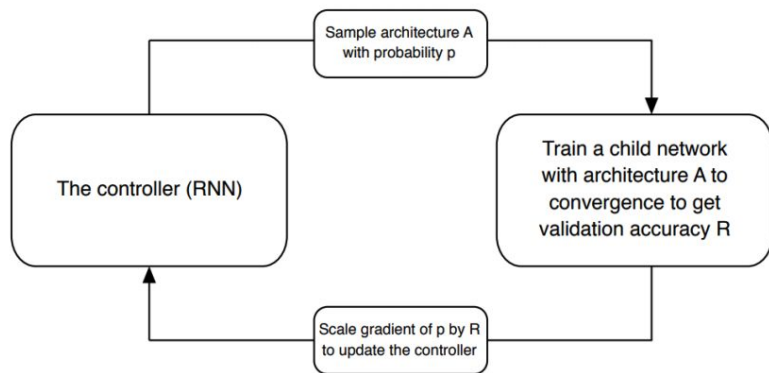
Result

Deep Learning for Data Generations

- Given training data, generate new samples from same distribution



AutoML and Neural Architecture Search



Source: Lex Fridman

Happy Lunar New Year

