Engineering Analytics & Machine Learning (ECSE202)
Seminar 7

Artificial Neural Network

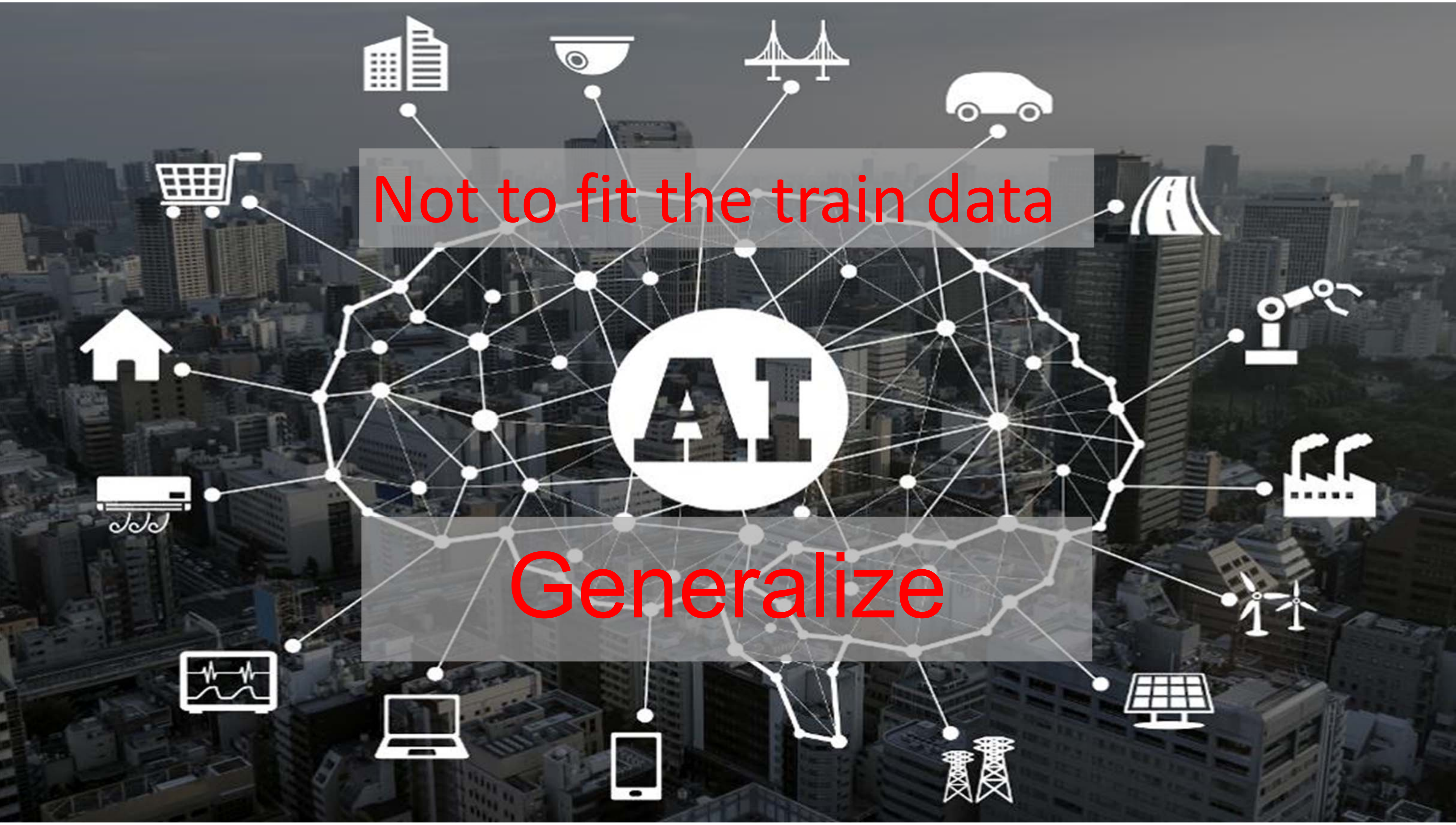
AY2018/2019 OCT SEMESTER

School of Engineering
TEMASEK POLYTECHNIC

Objective of AI

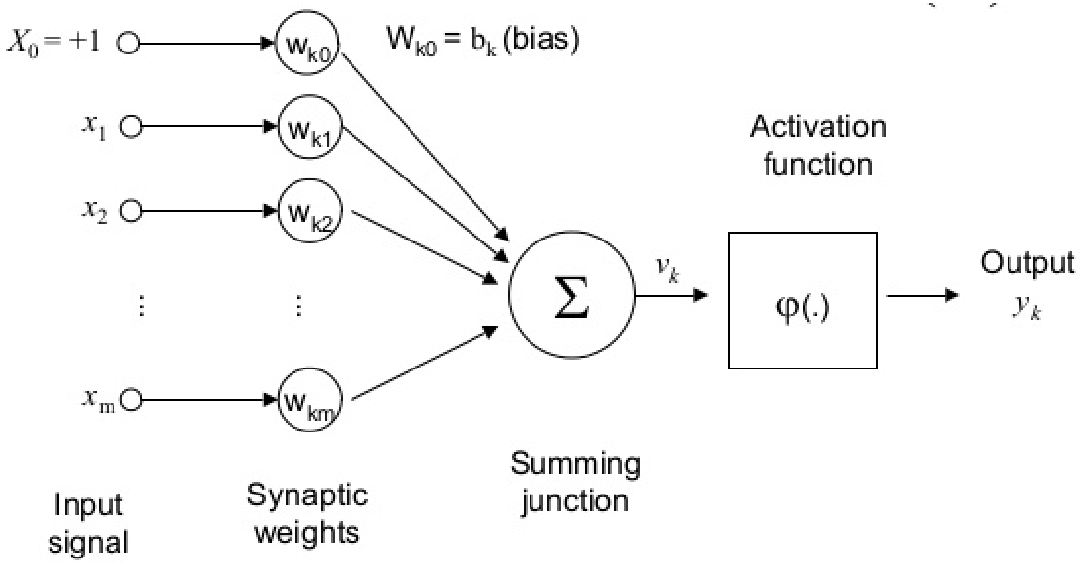
Human Learning

- Show sample of examples for learning
- Not for the kid to be fixed to the material
- Able to apply
- Able to extrapolate
- Able to adapt to similar cases
- Able to innovate


Not to fit the train data
AI
Generalize

Fundamental

Single Perceptron



$$v_k = \sum_{j=0}^{m} w_{kj} x_j$$

$$y_k = \varphi(v_k)$$

Type of Activation Function

Activation functions are transfer functions which is nonlinear that act as threshold.
Desired characteristic:

- ✓ Mostly smooth, continuous, differentiable
- ✓ Fairly linear

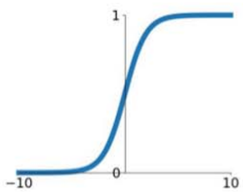
- Activation functions are need as they provide the nonlinearities to handle complex problem.
- Layers of linear function is just a combined linear function
- A combined linear function can only learn linear transformation of the input

Type of Activation Function

Activation Functions

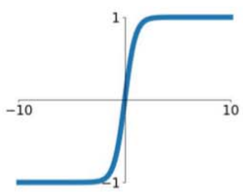
Sigmoid

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



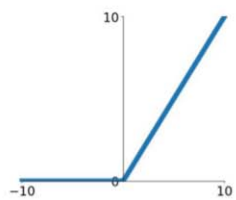
tanh

$\tanh(x)$



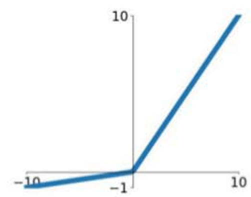
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

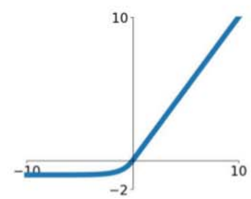


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

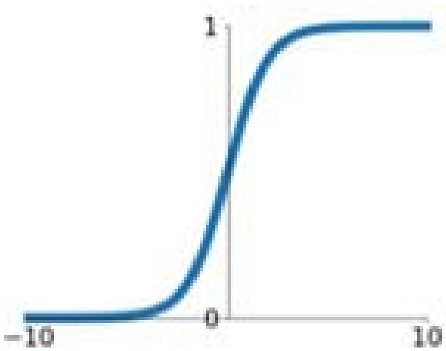
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation Functions

Sigmoid

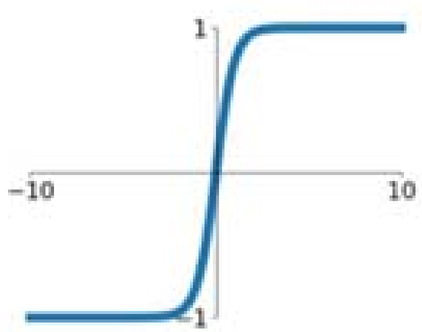


$$\sigma(x) = \frac{1}{1+e^{-x}}$$

- Squashes number to range from 0 to 1
- Historically popular as it could be interpreted as a “firing rate” of a neuron
- Issues:
  - Saturated Neuron “kill” the gradient
  - Output is not zero centered
  - Exponential is computational expensive

Activation Functions

tanh

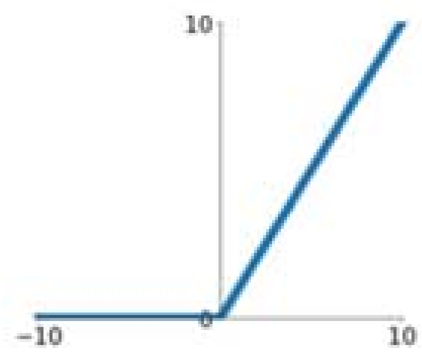


$\tanh(x)$

- Squashes number to range from 0 to 1
- Zero centred
- Issues:
  - Saturated Neuron “kill” the gradient

Activation Functions

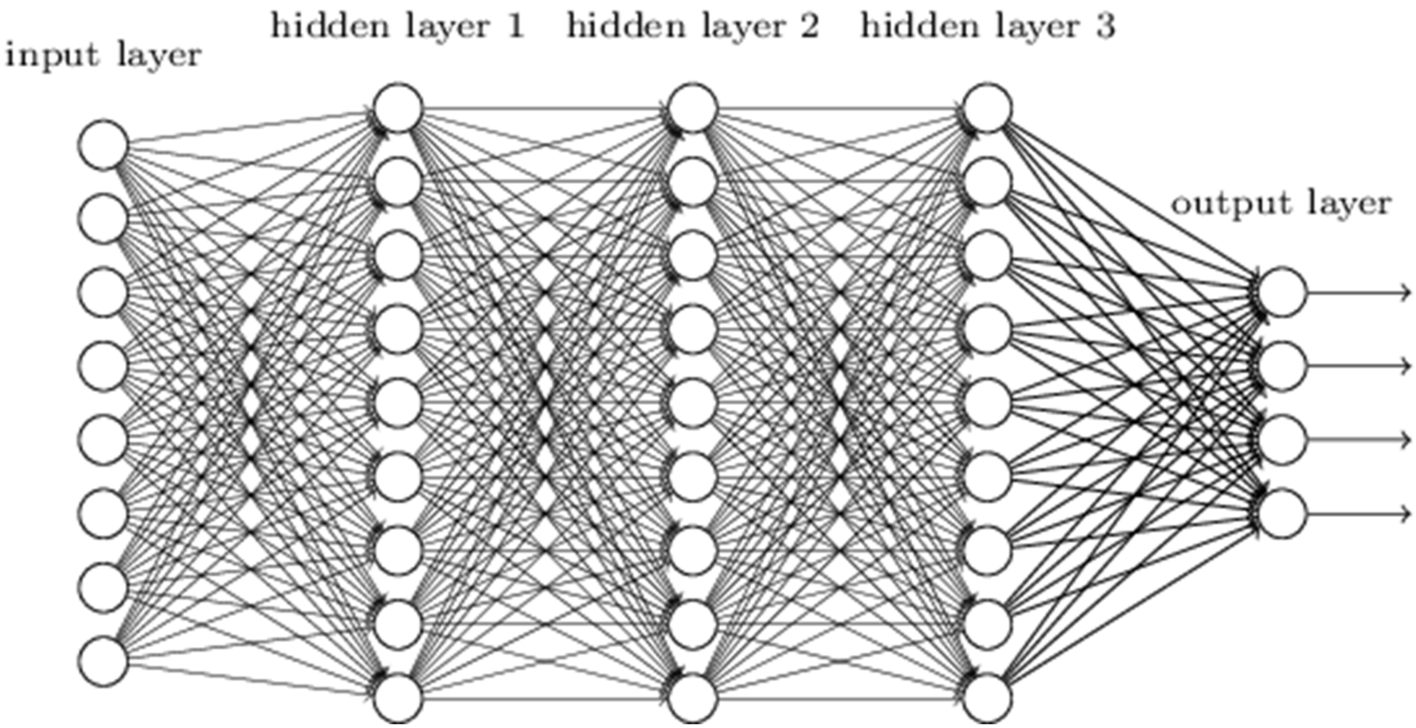
Relu



$$\max(0, x)$$

- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh
- Issues:
    - Not zero centred

Fully Connected Multiple Layer Perceptron Network (MLP)



Source: https://hackernoon.com/training-an-architectural-classifier-iii-84dd5f3cf51c

Multi-Layer Perceptron(MLP)

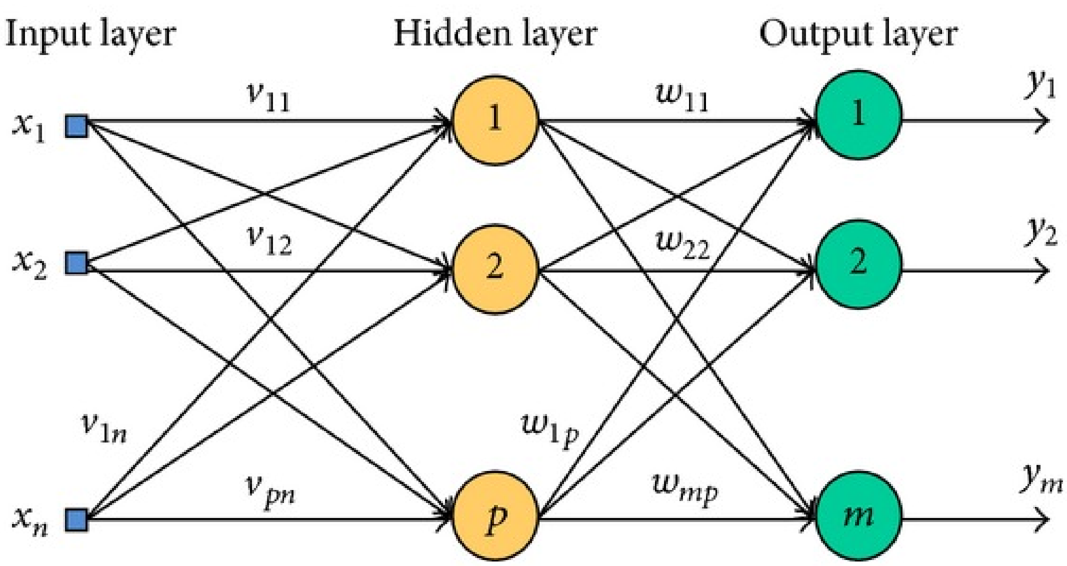


Image Source: https://www.researchgate.net/figure/258524366_fig5_Neural-network-with-one-hidden-layer-of-neurons

Output of hidden layer:

$$h_i^1 = \emptyset\left(\sum_{j=0}^{p} v_{ij} x_j + b_i^1\right)$$

Output of output layer:

$$y_k = \mathrm{f}\left(\sum_{j=0}^{m} w_{kj} y_j + b_k^o\right)$$

Classification Loss functions

Cross entropy loss function with sigmoid as output activation function:

$$\mathrm{E} = -\sum_{i=1}^{N} \left(t_i \log y_i\right) + (1 - t_i) \log(1 - y_i))$$

Where t_i is the target vector where it is either 1 or 0 and yi is the output vector

Regression Loss functions

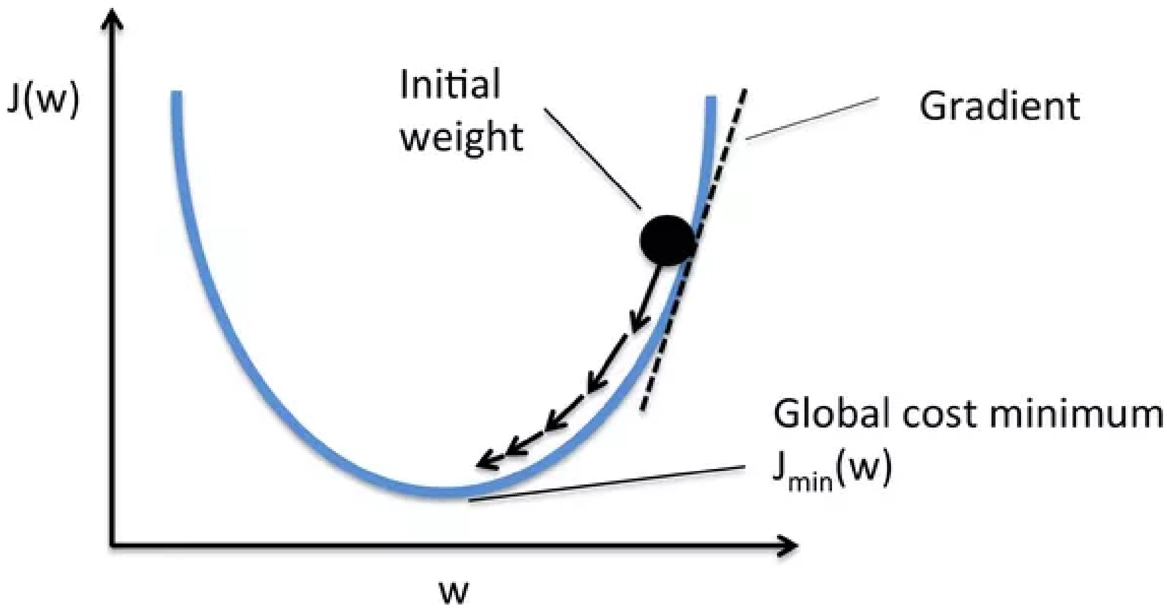
The loss function when the task is to predict real value such as measurement of houses or blood pressure:

E=$\sum \| t_i - y_i \|$ L1 Norm

E=$\sum \| t_i - y_i \|^2$ L2 Norm

Where ti is the target vector and y is the output vector

Gradient Descent (GD)



$$J(w) = \frac{1}{2}\sum_i \left(Target^i - Actual^i\right)^2$$

$$\Delta w_j = -\mu \frac{\partial J}{\partial w_j}$$

$$w_j = w_j + \Delta w_j$$

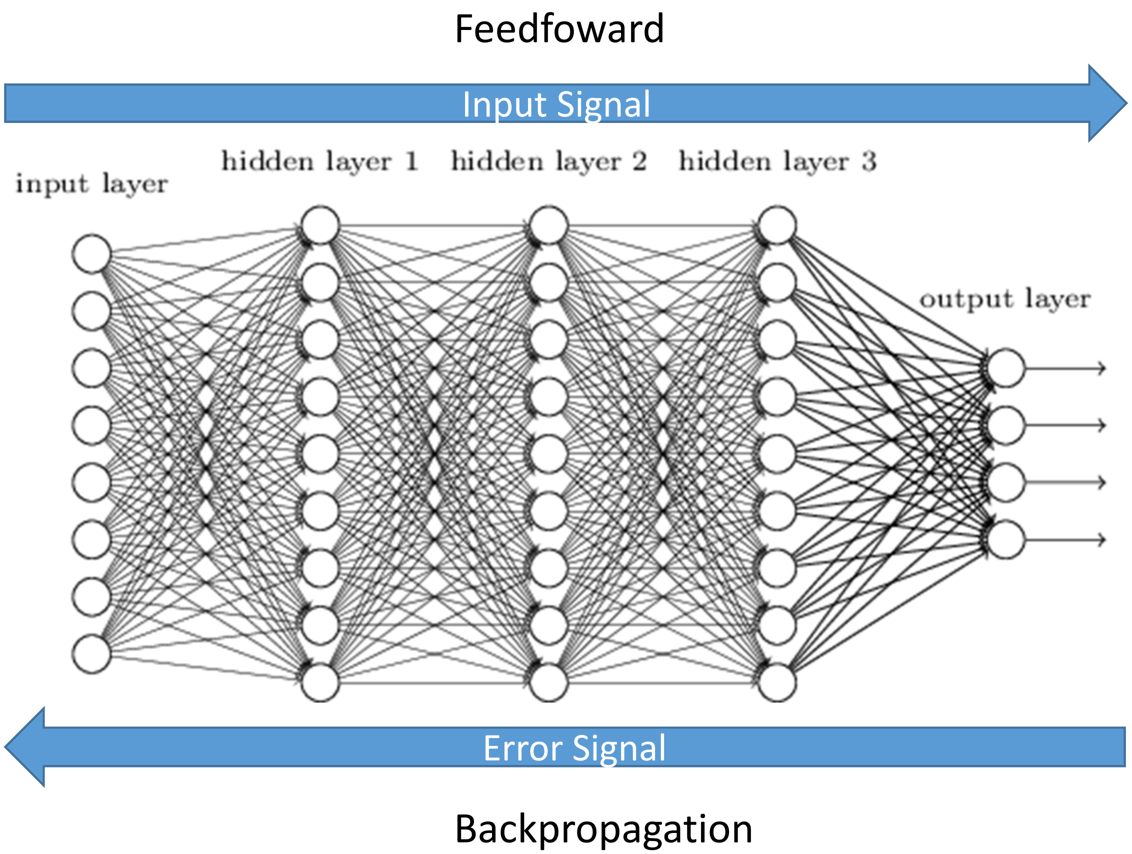
Where μ is the learning rate

Source: https://www.quora.com/Whats-the-difference-between-gradient-descent-and-stochastic-gradient-descent

Stochastic Gradient Descent (SGD)

- GD is too computation expensive for the entire training set
- For SGD instead the entire training set is divide into mini-batch and weights are update in each mini-batch
- The name stochastic as the gradient is estimated by each sample (mini-batch) rather the entire training set
- SGD had shown to almost surely converges to the global cost minimum if the cost function is convex (or pseudo-convex).

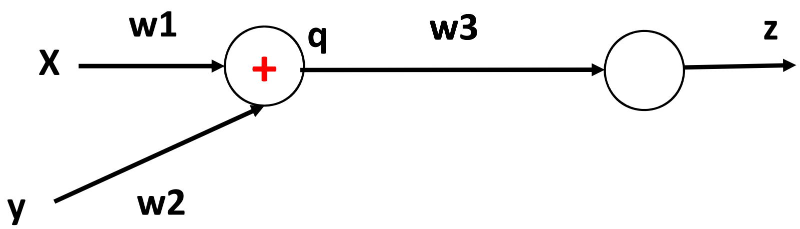
Backpropagation



Backpropagation was invented in the 1970s as a general optimization method for performing automatic differentiation of complex nested functions. However, it wasn't until 1986, with the publishing of a paper by Rumelhart, Hinton, and Williams, titled "Learning Representations by Back-Propagating Errors," that the importance of the algorithm was appreciated by the machine learning community at large.

Source: https://brilliant.org/wiki/backpropagation/

Simple Example of Backpropagation



$$q = w_1 * x + w_{2*}y$$

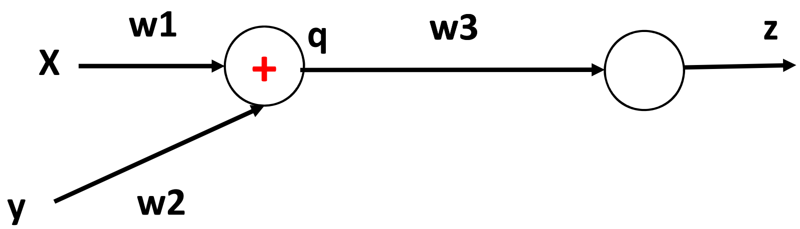
$$z = q * w_3 = (w_1 * x + w_{2*}y) * w_3$$

$$E = \frac{1}{2}(z_t - z)^2$$

$$e = (z_t - z)$$

Where z_t is the desired output

Simple Example of Backpropagation

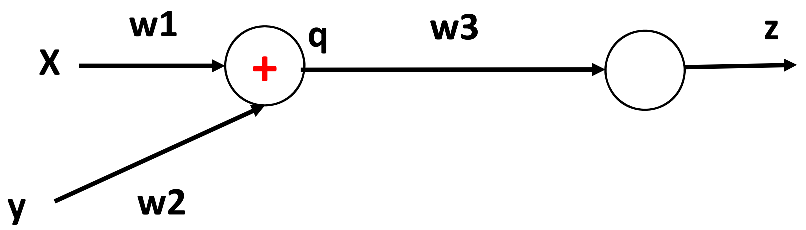


Objective is to find:

$$\Delta w_j = -\mu \frac{\partial E}{\partial w_j}$$

Where μ is the learning rate

Simple Example of Backpropagation



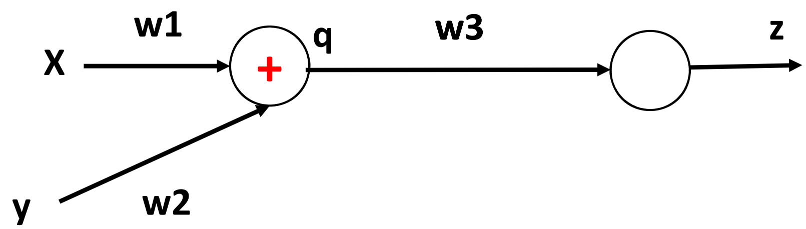
$$\frac{\partial E}{\partial w_3} = \frac{\partial E}{\partial z} * \frac{\partial z}{\partial w_3}$$ chain rule

$$\frac{\partial E}{\partial z} = -(z_t - z) = -e$$

$$\frac{\partial z}{\partial w_3} = (w_1 * x + w_{2*} y) = q$$

$$\frac{\partial E}{\partial w_3} = -q * e$$

Simple Example of Backpropagation



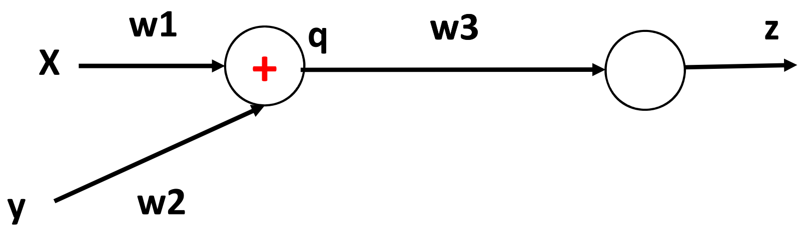
$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial q} * \frac{\partial q}{\partial w_2}$$ chain rule

$$\frac{\partial E}{\partial q} = (z_t - q * w_3) * w_3 = e * w_3$$

$$\frac{\partial q}{\partial w_2} = y$$

$$\frac{\partial E}{\partial w_2} = e * w_3 * y$$

Simple Example of Backpropagation

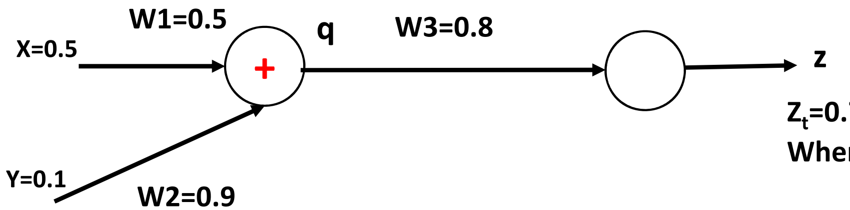


$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial q} * \frac{\partial q}{\partial w_1}$$ chain rule

$$\frac{\partial q}{\partial w_1} = x$$

$$\frac{\partial E}{\partial w_1} = e * w_3 * x$$

Simple Example of Backpropagation



Z_t=0.7
Where Z_t is the target value of z

$$q = w_1 * x + w_{2*}y = 0.5 * 0.5 + 0.1 * 0.9 = 0.34$$

$$z = q * w_3 = 0.272$$

$$e = (z_t - z) = 0.428$$

$$\frac{\partial E}{\partial w_3} = -q * e = -0.14552$$

$$\frac{\partial E}{\partial w_2} = e * w_3 * y = 0.03424$$

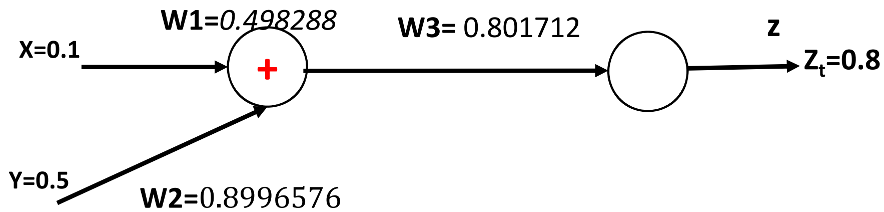
$$\frac{\partial E}{\partial w_1} = e * w_3 * x = 0.1712$$

$$w_3 = w_3 - \mu \frac{\partial E}{\partial w_3} = 0.8 - 0.01 * (0.1712) = 0.801712$$

$$w_2 = w_2 - \mu \frac{\partial E}{\partial w_2} = 0.8996576$$

$$w_1 = w_1 - \mu \frac{\partial E}{\partial w_1} = 0.498288$$

Simple Example of Backpropagation



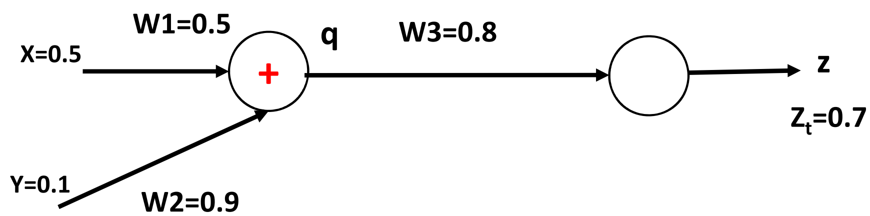
$$q = w_1 * x + w_{2*} y = 0.4996576$$

$$z = q * w_3 = 0.4005814938112$$

$$e = (z_t - z) = 0.3994185061888$$

- Initially the e=0.428
- With the updated weights, e=0.399 which show an improvement of 0.029 (6.78%)

Demo: Backpropagation



One-Hot Encoding

Not onehot encoding case

explanation variable

Fruits	numeric(x)
apple	0
meat	1
grape	2

Simple linear classification

$y = wx$

possible value

$y = 0$
$y = w$
$y = 2w$

classification threshold

$y > constant$

$y <= constant$

possible classification result

Class1	Class2
apple meat	grape
apple meat grape	

Class1	Class2
apple	meat grape
	apple meat grape

never happen classification case

Class1	Class2
apple grape	meat

onehot encoding case

explanation variable

Fruits	numeric(x)
apple	001
meat	010
grape	100

Simple linear classification

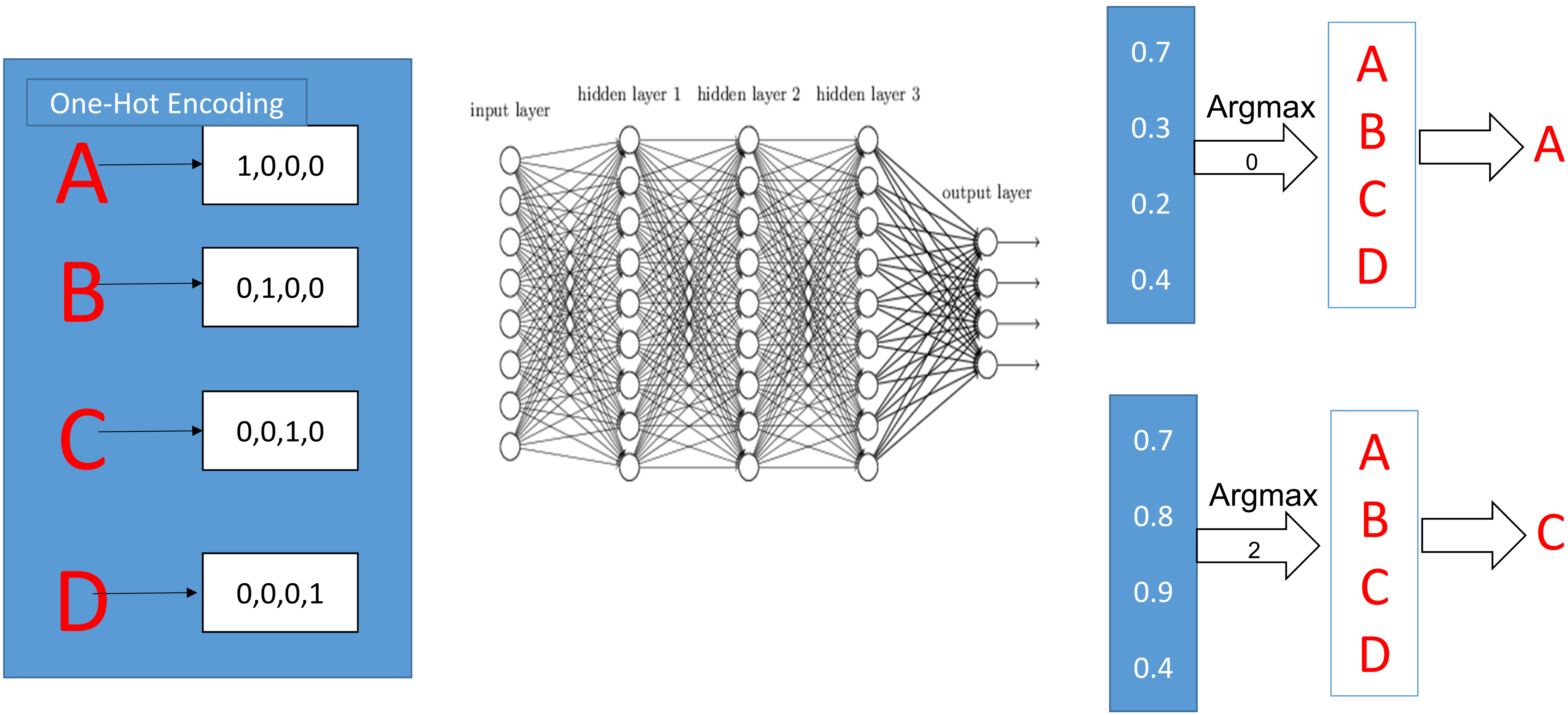
$$y = w_{apple}x_{apple} + w_{meat}x_{meat} + w_{grape}x_{grape}$$

$$w_{apple} = 0.5, w_{meat} = 0, w_{grape} = 0.5$$

Class1	Class2
apple grape	meat

onehot encoding is needed for categorical data in neural network classification

One-Hot Encoding example



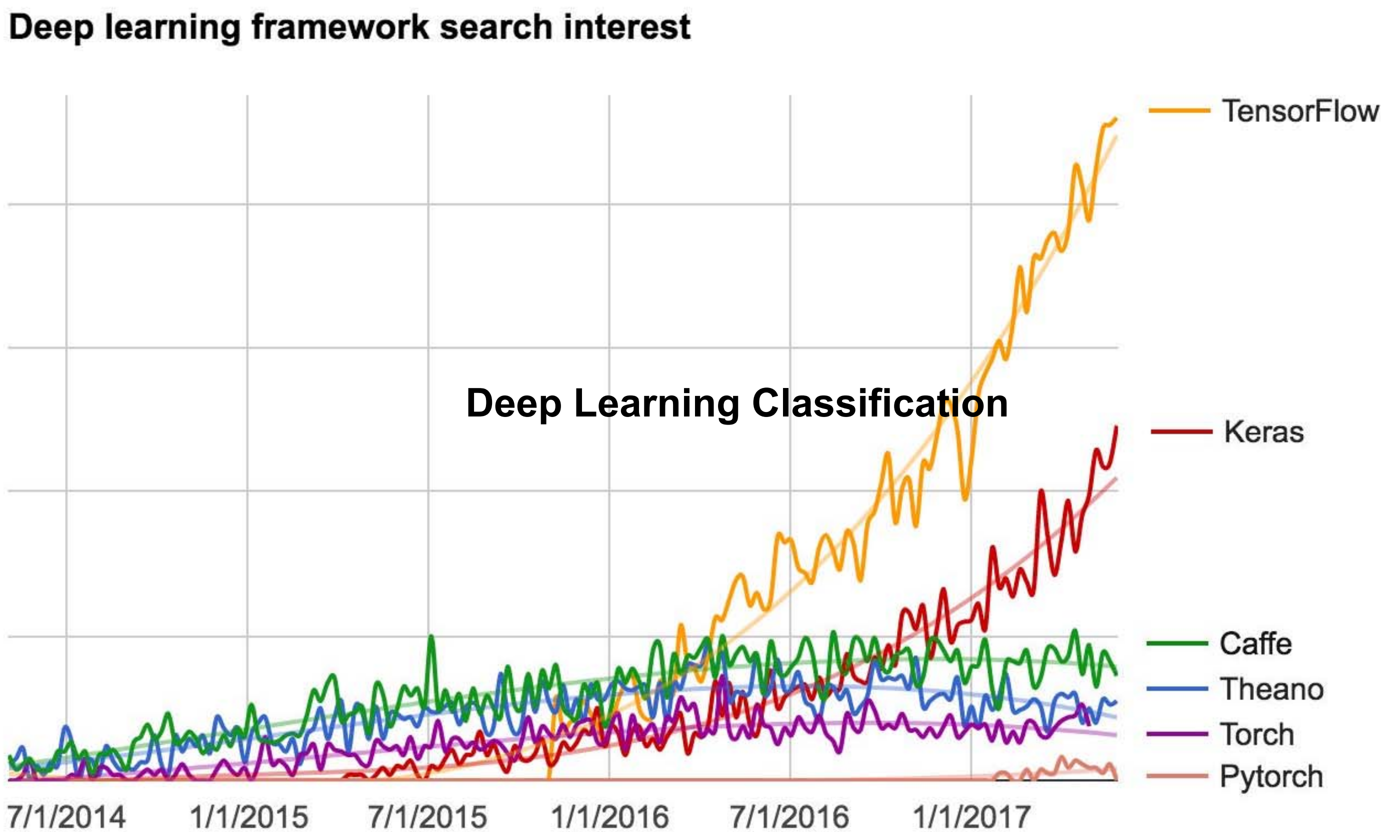
AI Frameworks

AI Framework

	Languages	Tutorials and training materials	CNN modeling capability	RNN modeling capability	Architecture: easy-to-use and modular front end	Speed	Multiple GPU support	Keras compatible
Theano	Python, C++	++	++	++	+	++	+	+
Tensor-Flow	Python	+++	+++	++	+++	++	++	+
Torch	Lua, Python (new)	+	+++	++	++	+++	++	
Caffe	C++	+	++		+	+	+	
MXNet	R, Python, Julia, Scala	++	++	+	++	++	+++	
Neon	Python	+	++	+	+	++	+	
CNTK	C++	+	+	+++	+	++	+	

Source: https://www.kdnuggets.com/2017/03/getting-started-deep-learning.html

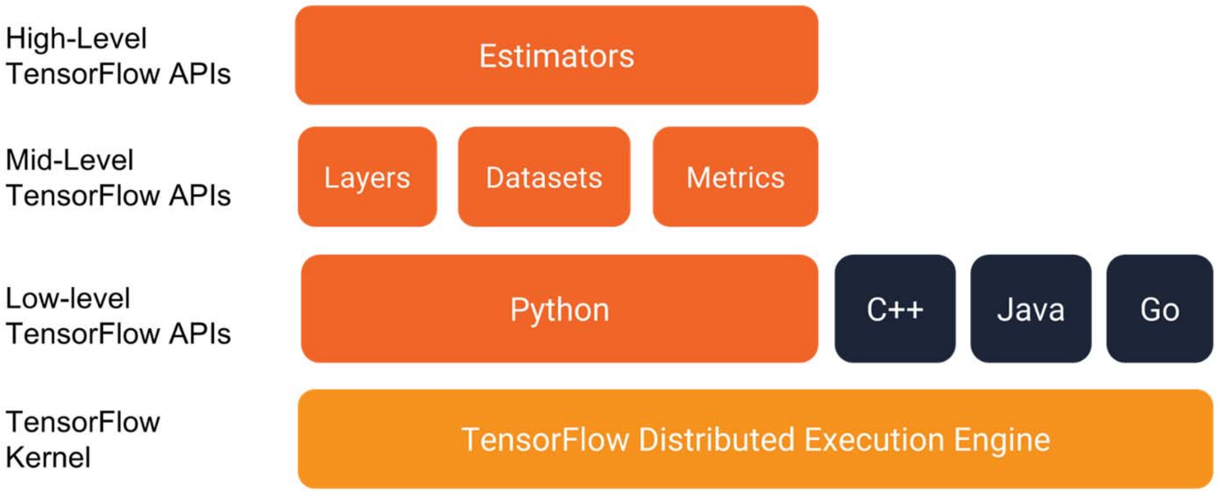
AI Framework



Source: https://twitter.com/fchollet/status/871089784898310144

Tensorflow

Tensorflow Layers



- TensorFlow™ is an open source software library for high performance numerical computation.
- Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.
- Originally developed by researchers and engineers from the Google Brain team within Google’s AI organization.
- it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains.

You can access link below the access the official site:
https://www.tensorflow.org/

Resources

CS231n: Convolutional Neural Networks for Visual Recognition
(http://cs231n.stanford.edu/)

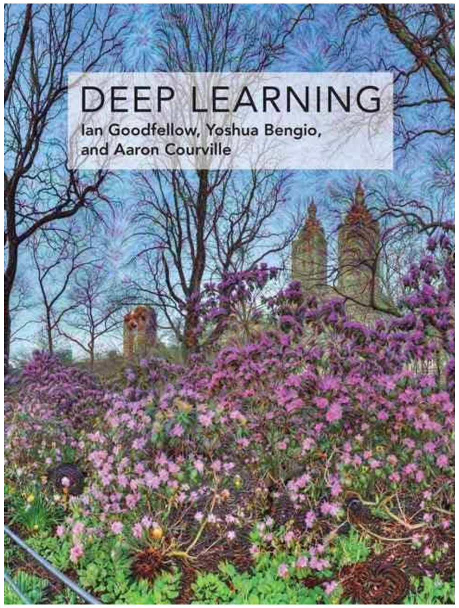
NUS SoC CS6101 Deep Learning vis Fast.AI
(http://www.comp.nus.edu.sg/~kanmy/courses/6101_2017_2/)

Coursera Deep Learning Specialization
https://www.coursera.org/specializations/deep-learning

Fast.ai
http://www.fast.ai/

Awesome AI Papers (Deep Learning, Computer Vision, Robotics, NLP etc.)
https://www.facebook.com/groups/awesomeaipapers/

My archive of papers to be read
https://github.com/coolingozone/readingdeeplearning



http://www.deeplearningbook.org/

TP Library Open Shelf, Level 7 Q325.5 Goo