ANN

- More general than

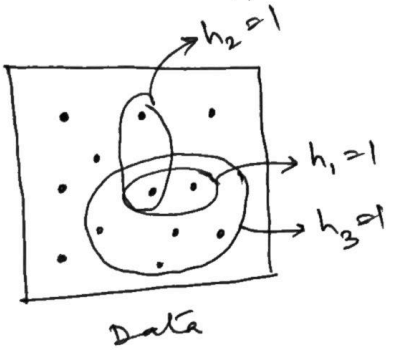
Let $h_j(x)$ & $h_k(x)$ be boolean-valued functions over X.

Then h_j is more-general-than-or-equal-to h_k ($h_j \geq_g h_k$)

iff $(\forall x \in X)\left[h_k(x) = 1 \Rightarrow h_j(x) = 1\right]$

if h_j is more-general-than h_k

then h_k is more-specific-than h_j

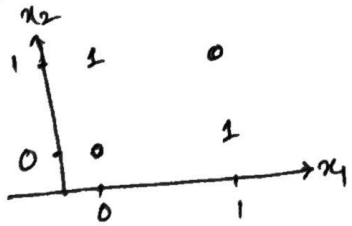


So $h_3 \geq_g h_1$

Perceptron = 1 neuron (node of hidden layer) in ANN. or output

Regularisation for ANN — L1, L2, Dropout.

XOR



SGD is not guaranteed to converge.

can use adaptive learning rate [not necessarily better]

Learning XOR

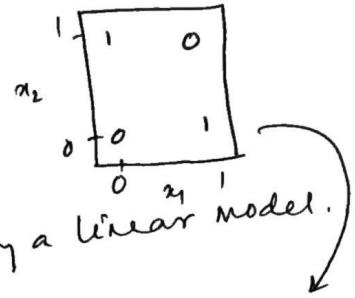
$X = \{[0,0], [0,1], [1,0], [1,1]\}$

MSE loss function.

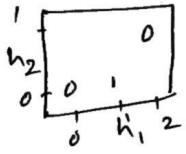
$$J(\theta) = \frac{1}{4} \sum_{x \in X} \left(f^*(x) - f(x;\theta) \right)^2$$

$f(x; w, b) = x^T w + b$

Solving gives $w = 0$ & $b = 1/2$



=> XOR can't be implemented by a linear model.



ML non-linear function
≡ Affine Transformation
+ Activation function
(Rectified Linear Unit, ReLU)

$\hookrightarrow g(z) = \max\{0, z\}$

$h_i = g(x^T W_{:,i} + c_i)$

$f(x; W, c, w, b)$

$= w^T \max\{0, W^T x + c\} + b$

$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ $c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$ $w = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$ $b = 0$.

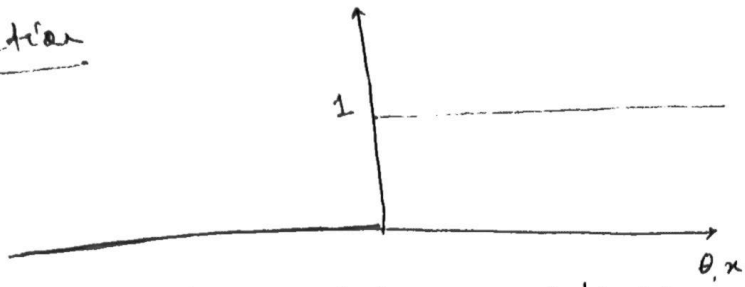
Universal Approximation Theorem
- May not be able to find parameter values corresponding to the desired function
- Overfitting: May converge to the wrong function

No Free Lunch Theorem
- There is no universally superior ML algo.

A feedforward network with single hidden layer is sufficient to represent any arbitrary function.

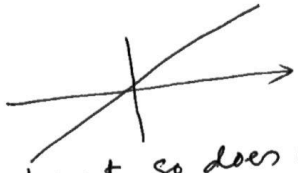
ANN ACTIVATION FUNCTIONS

Step function



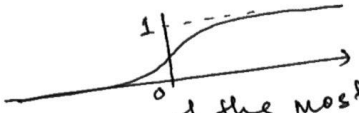
- Can be used in output layer
- derivative is always zero, so cant help in gradient descent & so not for hidden layers.

Linear function



- Gradient is a constant, so does not help in gradient descent
- Leads to overall linear classifier.

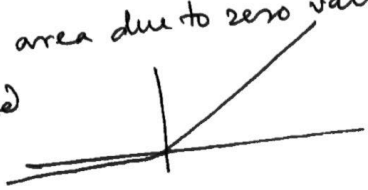
Sigmoid function



- good classifier, one of the most widely used.
- gives probability estimates
- has changing gradient
- but gradient saturates quickly leading to slow learning & can get stuck in local minima Vanishing gradient problem
- can use hyperbolic tangent fn. instead.

ReLU [Rectified Linear Unit] function

- Faster than sigmoid/tanh due to linearity & zero nature.
- But no learning in neg. area due to zero value.
- used Leaky ReLU instead



Softmax function

- Similar to sigmoid but helps in multiclass classification.
- usually used in output layer.

Usually, use ReLU in hidden layer & sigmoid/softmax in output layers.

~~K - nearest neighbour~~

ANN Optimization Methods

BATCH GRADIENT DESCENT

- usual gradient descent
- uses whole dataset for each update
- very slow & resource consuming
- can't use new data on the fly (no online update)
- usually converges to local minima for non-convex loss functions
- done over many epochs

STOCHASTIC GRADIENT DESCENT

- uses one data point at a time
- frequent updates with high variance
- random jumps allow convergence to ~~global minima~~ or potentially better minima. may (not ~~always~~ be global since SGD can overshoot)
- faster convergence
- shuffle data for each epoch

MINI-BATCH GRADIENT

- SGD using small batch of dataset
- faster convergence than vanilla & less fluctuations than SGD with one data point
- usually called SGD.
- usually gets stuck at saddle points.

SGD + Momentum

- Add update vector of past time step
- helps avoiding being caught in ravines.
- like pushing a ball downhill.
- gradient terms add up if they point in same direction

NESTEROV SGD [NAG - Nesterov Accelerated Gradient]

- on SGD + momentum, need to slow down before gradient changes sign again.
- So estimate future parameter value using momentum term & then use gradient at that parameter value

$$v_t = \underbrace{\gamma v_{t-1}}_{\text{usually } 0.9} + \eta \nabla_\theta C(\theta - \gamma v_{t-1})$$

$$\theta = \theta - v_t$$

ADAGRAD

ADAM - faster
SGD - better generalised

- Perform larger updates for frequent parameters
- Helps in dealing with sparsity.
- Basically use different learning rate (η) for each parameter.
- η is modified for each parameter based on past gradients.
- But η can keep decreasing to zero in this method. Resolved by AdaDelta.

ADAM [Adaptive Moment Estimation]

Another method to compute adaptive learning rate.

Solving ODEs & PDEs using ANN

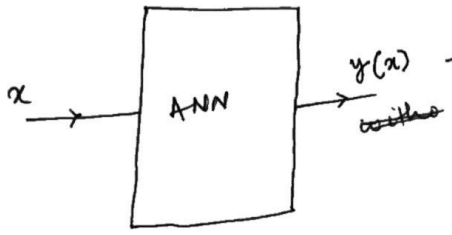
$$\frac{dy}{dx} = -2xy$$

$$y(0) = 1$$

$$\Rightarrow y(x) = e^{-x^2}$$

Analytical solution
Not always possible
Numerical Simulation
- Cannot give output beyond the range on which simulation performed
- Requires human analysis of results.

ANN
- Possibility of learning the actual function [Universal Approx. Thm.]



Train the ANN without having any knowledge of the analytic or numerical solution.

Need a loss (cost) function to be minimised

$$C = \left[\frac{dy}{dx} + 2xy\right]^2 + [y(0) - 1]^2$$

↳ Automatic Differentiation [AutoDiff] in Tensorflow

Different from both symbolic diff & numerical differentiation.

Symbolic Diff (eg. Mathematica) - Inefficient code
Numerical Diff - round-off errors
] unsuitable for higher & partial derivatives with many inputs needed in gradient descent.

AutoDiff is NOT numerical differences
It is both efficient & numerically stable.
(partly symbolic & partly numerical)