Neural networks

Prof. Asim Tewari
IIT Bombay

Projection Pursuit Regression

An input vector X with p components, and a target Y . Let ω_m, m = 1, 2, . . . ,M, be unit p-vectors of unknown parameters. The projection pursuit regression (PPR) model has the form

$$f(X) = \sum_{m=1}^{M} g_m(\omega_m^T X)$$

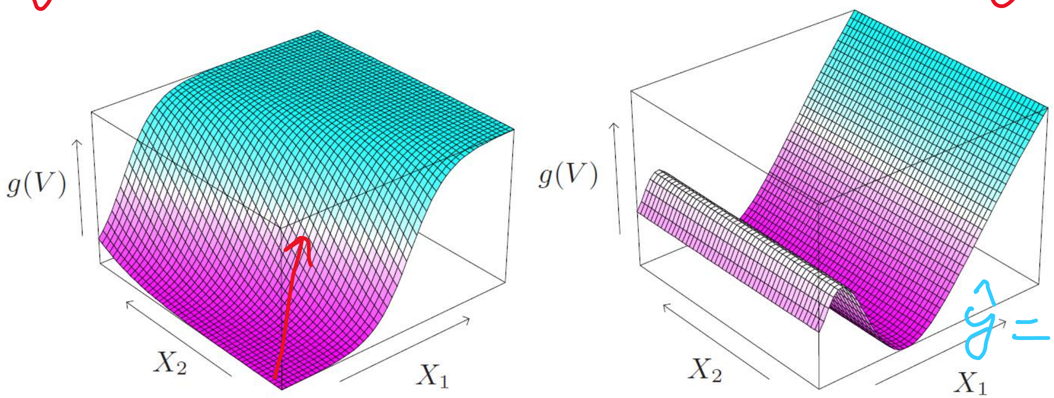
This is an additive model, but in the derived features $V_m = \omega_m^T X$ rather than the inputs themselves. The functions g_m are unspecified and are estimated along with the directions !m using some flexible smoothing method.

We seek the approximate minimizers of the error function

$$\sum_{i=1}^{N} \left[y_i - \sum_{m=1}^{M} g_m(\omega_m^T x_i) \right]^2$$

Projection Pursuit Regression

$\hat{y}_i = g_1(\omega_1 x_i) + g_2(\omega_2 x_i) + \dots + g_m(\omega_m x_i)$



$\hat{y} = g_1(x_1) + g_2(x_2)$

$\omega = \frac{(1,1)}{\sqrt{2}}$

$\omega = (1,0) \rightarrow x_1$

$\omega = (0,1) \rightarrow x_2$

Perspective plots of two ridge functions.

- (Left:) g(V) = 1/[1 + exp(−5(V − 0.5))], where V = (X1 + X2)/p2.
- (Right:) g(V) = (V + 0.1) sin(1/(V/3 + 0.1)), where V = X1.

$\omega_1 = (1,1)/\sqrt{2}$ $\quad \omega_2 = (1,-1)/\sqrt{2}$

$f(x_1 \cdot x_2) \rightarrow x_1 x_2 = \frac{(x_1+x_2)^2 - (x_1-x_2)^2}{4}$

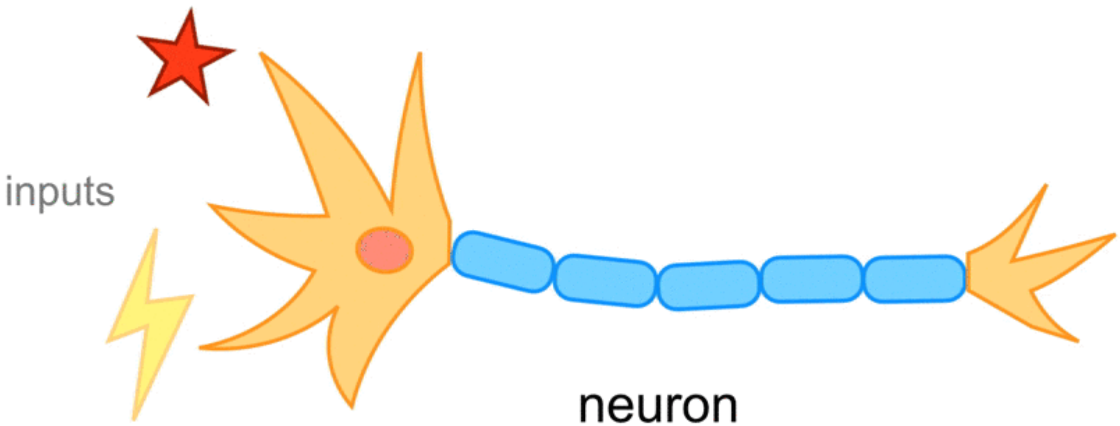
Neural Networks

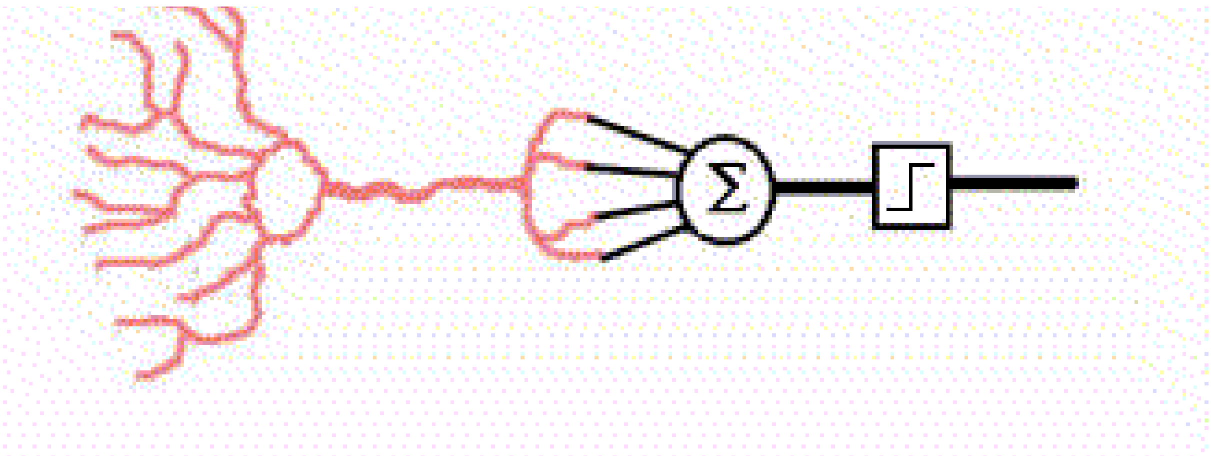
- A popular class of nonlinear regression methods are the so-called universal approximators

A class F of such functions f is called a *universal approximator* if and only if for any $\epsilon > 0$ there exists a function $f^* \in F$ such that

$$|f(x) - f^*(x)| < \epsilon$$

Neuron is a binary switch





Output of a neuron

- The output of each neuron is a real-valued scalar O_i
- The "effective" input to each neuron is the weighted sum of the inputs plus a bias b_i

$$I_i = \sum_j w_{ji} O_j + b_i$$

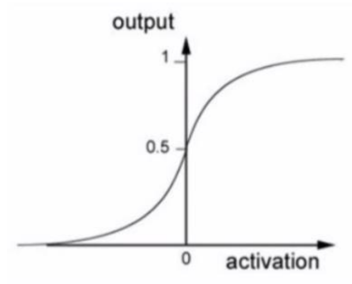
- The output of each neuron is computed from the effective input using a nonlinear *activation function s*

$$O_i = s(I_i)$$

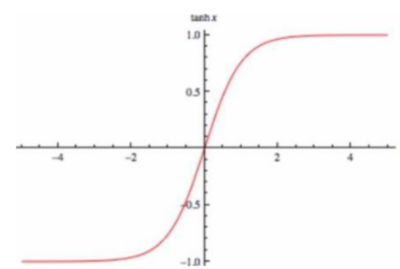
Neuron activation functions

- Frequently used activation functions include
- The *sigmoid* (s-shaped) functions
  - sigmoid(x) = 1/(1 + e^{-x})

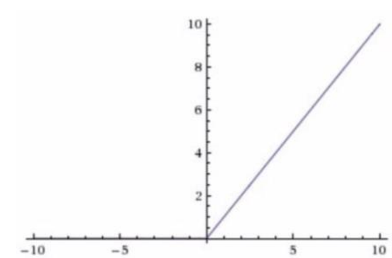
This is same as logistic regression



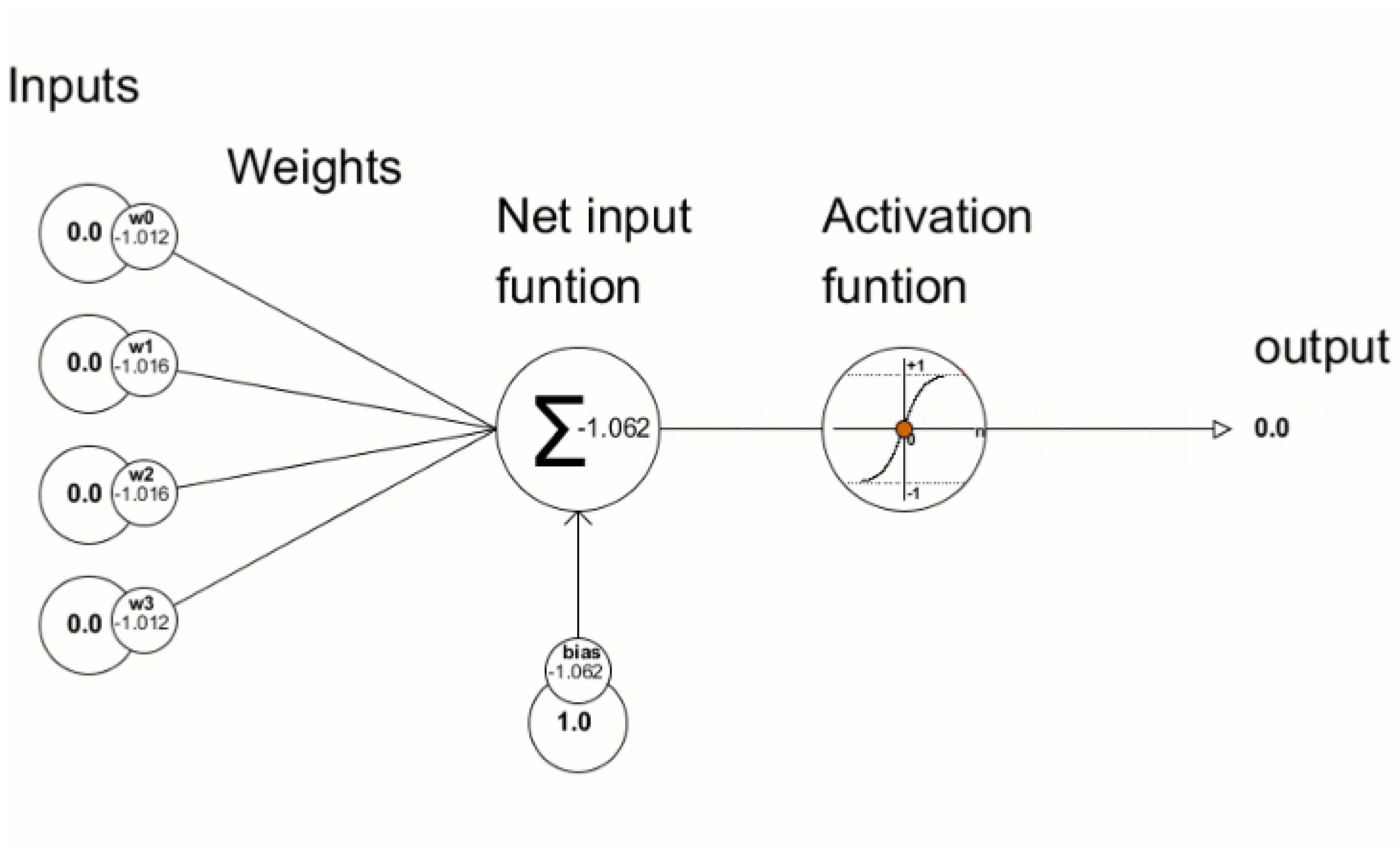
- The hyperbolic tangent function
  - tanh(x) = (e^{x} - e^{-x})/(e^{x} + e^{-x})



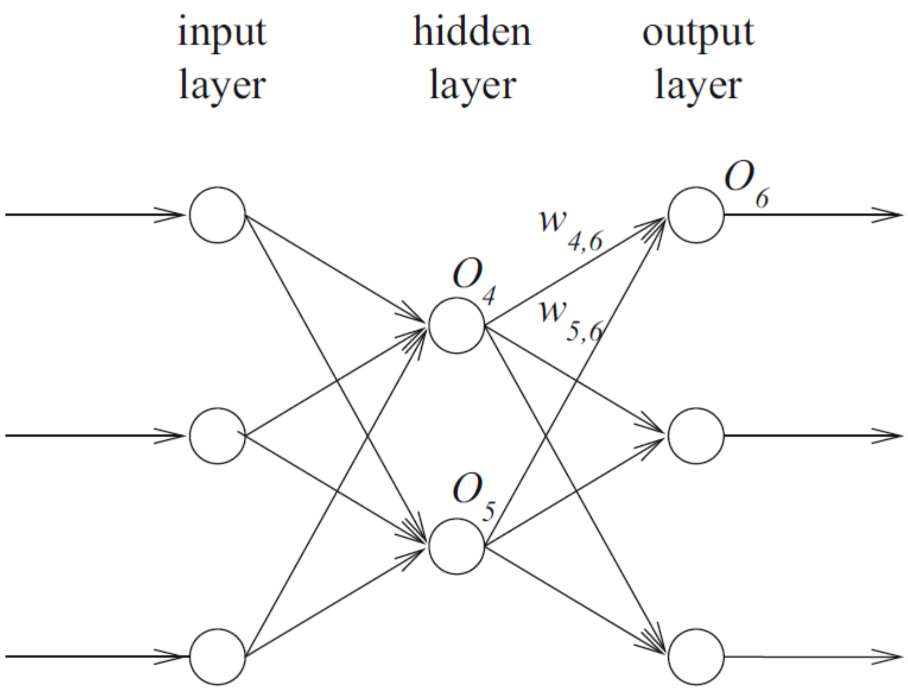
- The rectifier Linear Unit (relu)
  - relu(x) = max(0,x)



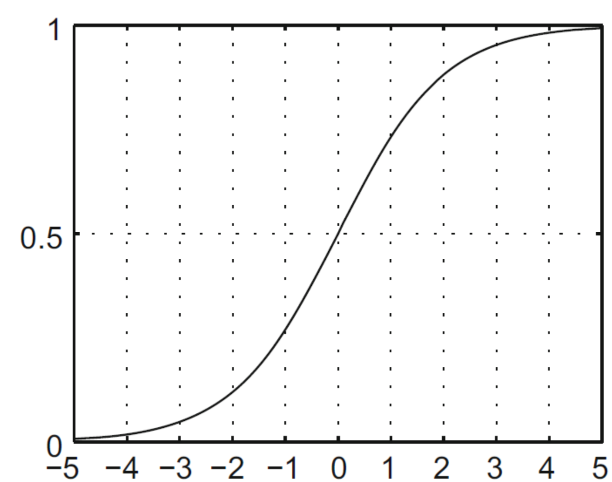
ANN and smooth switch



Multilayer perceptron



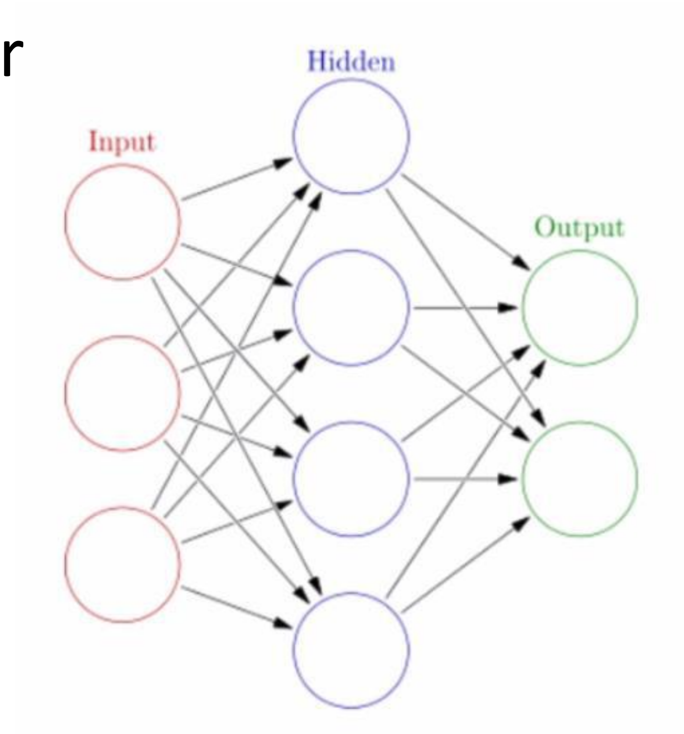
Multilayer perceptron



sigmoidal function

Neural networks

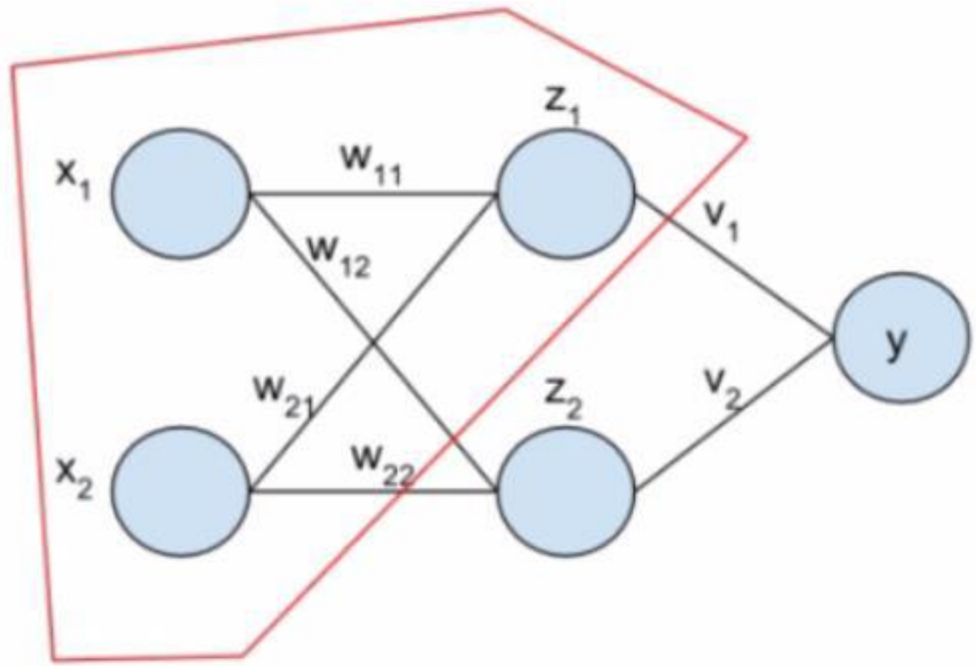
- Hidden layer (one or more)
- From left to right: a node in one layer is connected to every other node in the next layer
- Left-most layer = Input
- Right-most layer = Output



Graphs with nodes and edges

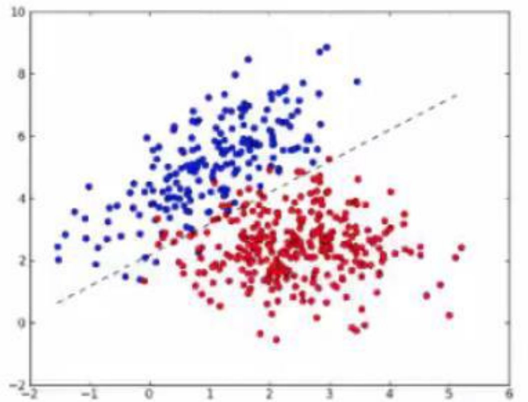
Neural networks

It can be perceived as a multiple layers of logistic regression units



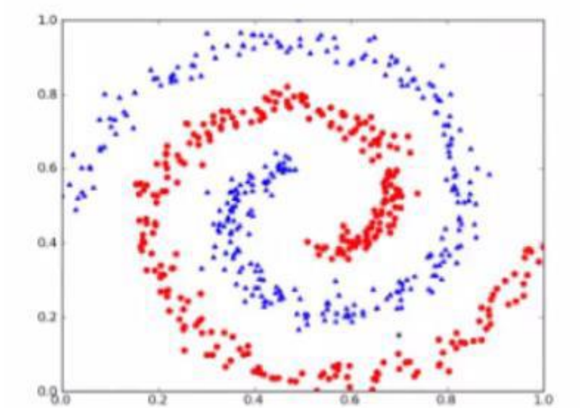
Linear v/s non-linear classifiers

- Logistic regression : linear classifier



Boundary can be expressed as: **WTX**

- Neural networks : non linear classifier



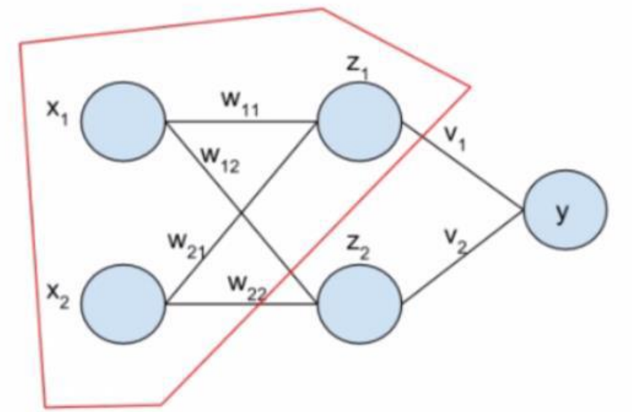
Boundary cannot be expressed as: **WTX**

But can be achieved by a combination of many logistic units, each of which can be expressed as **WTX**

Neural Network Inference

Feed forward calculations

- x1=>0, x2 => 1
- All the ‘w’ weights => 1
- All the ‘v’ weights => 1
- bias: b= 0; c=0



- z(1) = sigmoid(0*1 + 1*1) = 0.731
- z(2) = sigmoid(0*1 + 1*1) = 0.731
- p(y|x) = sigmoid(0.731*1 + 0.731*1) = 0.812

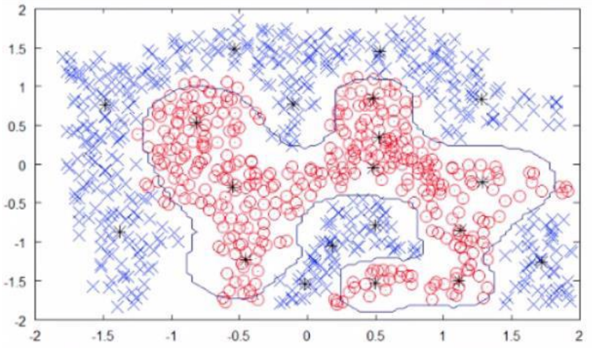
$$z_j = \text{sigmoid}(\textstyle\sum_i (W_{ij} x_i) + b_j)$$

$$p(y|x) = \text{sigmoid}(\textstyle\sum_j (v_j z_j) + c)$$

- Challenge:
  – How to choose the weights and biases so that the neural network predicts accurately

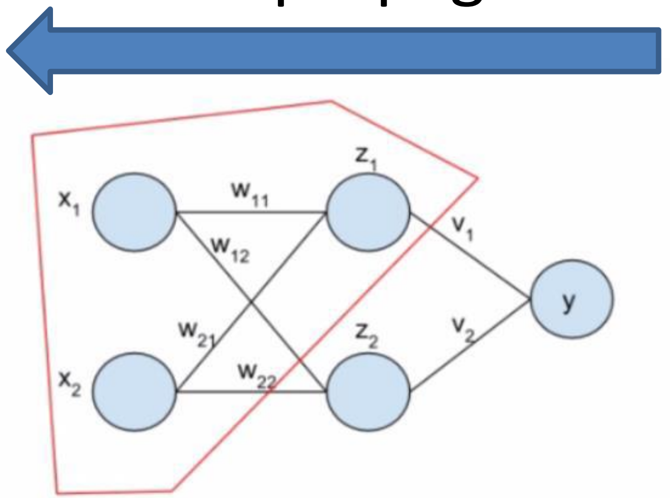
How to interpret the weights and outputs

- The value(s) of Y resulting from these weights is a probability
  - If the probability is greater than 50% => YES
  - If the probability is less than 50% => NO
- In the case of neural networks, beyond the first layer the weights cannot be said to have any meaning!
  - This is what makes neural networks non-linear and powerful



Neural networks Training

- The “training part”
  - Known as “Back propagation”
  - Error gets “propagated” backwards from the right
  - Weights in the intermediate layers get adjusted based on this back-propagated error



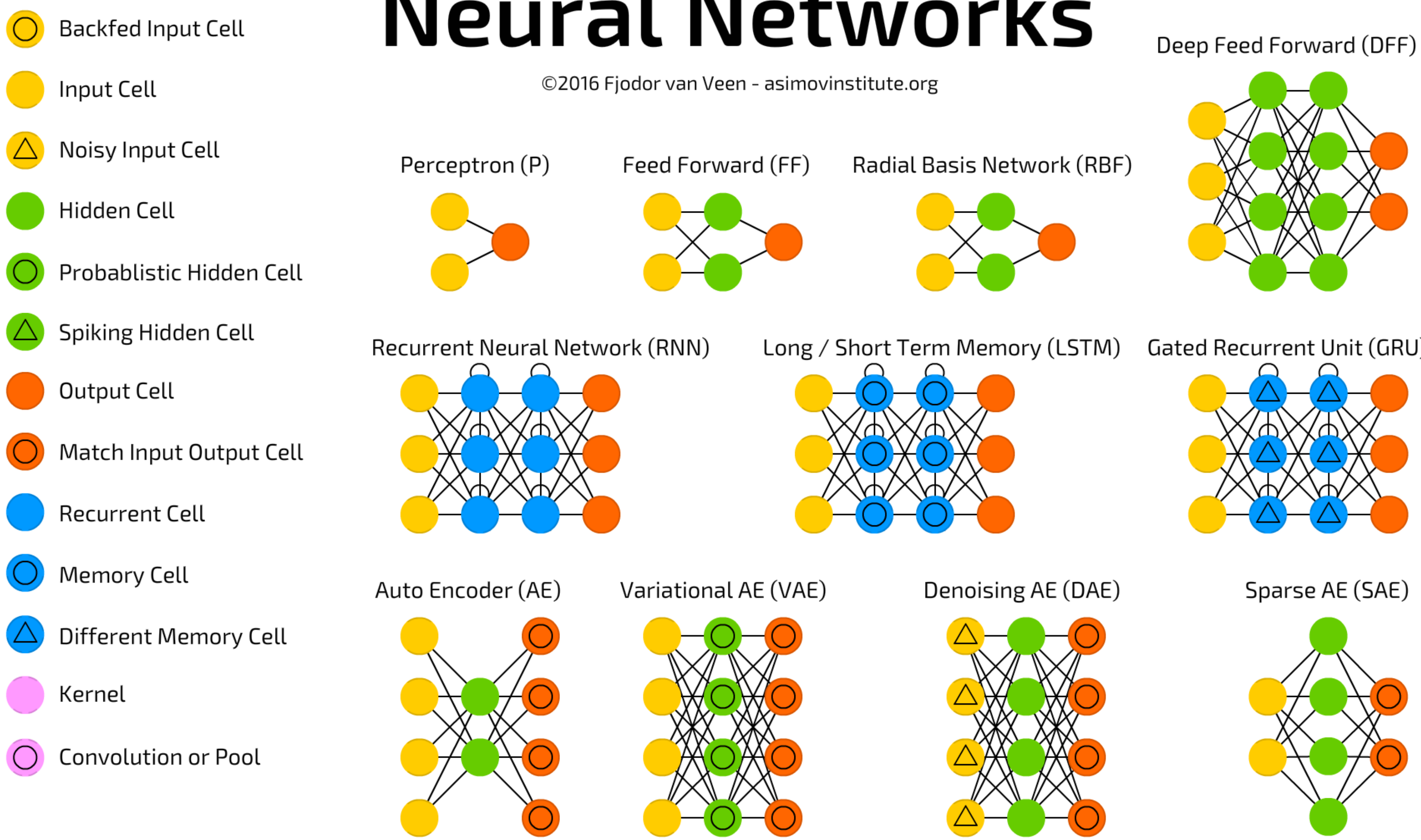
Neural networks Training

- Randomly initialize the weight vectors
- Multiply the input with weight vectors to reach the final output (feed forward)
- Calculate the error : compare result of the forward step to the expected output
- Calculate the gradient of the error and change weights towards the direction of the gradient
- Back-propagate this calculation and change the weights right up to the first hidden layer
- Repeat this process till the stopping criteria is reached

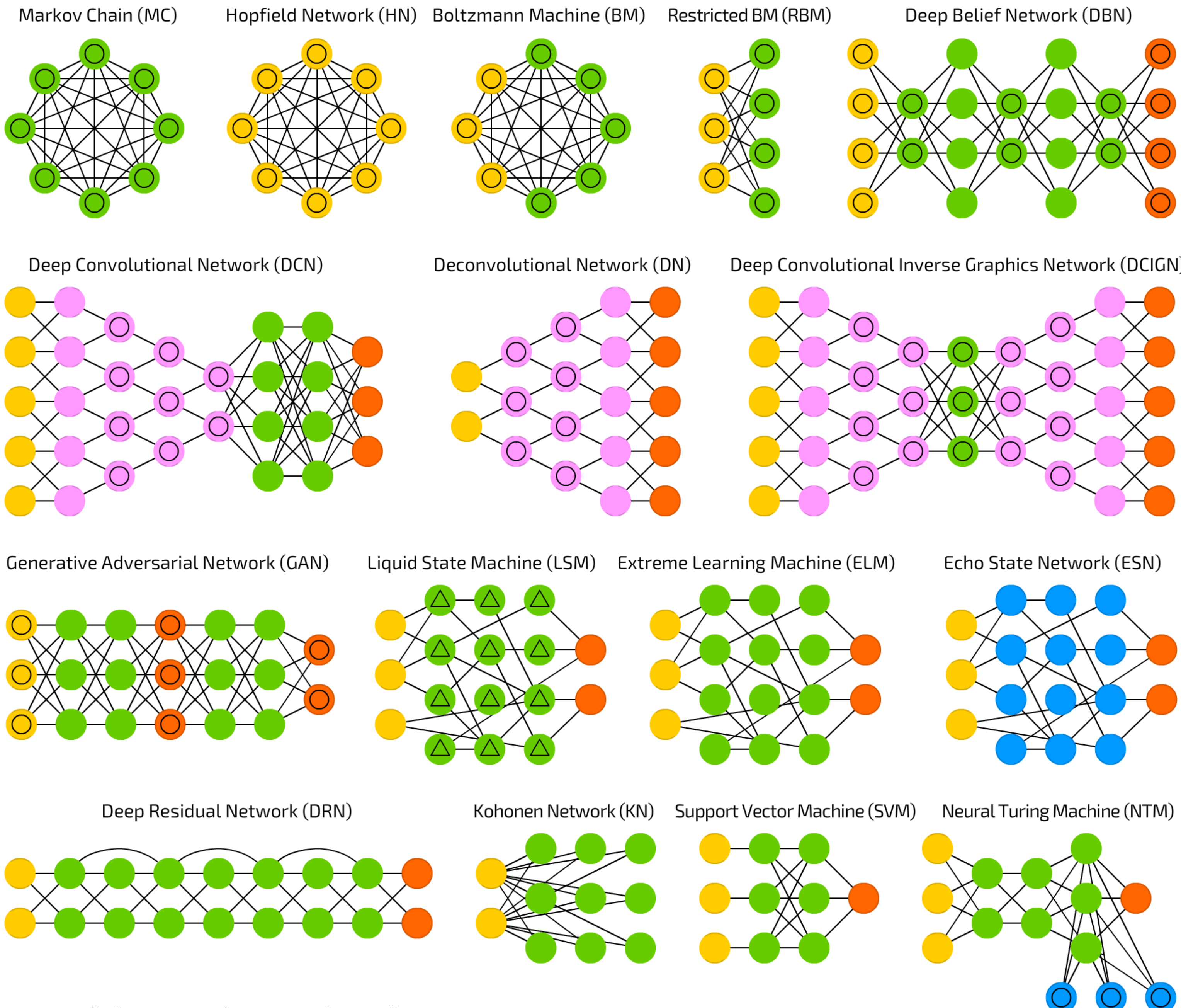
Backpropagation strategies

- Mini batch
  - Uses concepts from *stochastic* gradient descent
  - Perform backpropagation with a sample of the dataset
  - This is done to avoid local minima
- Momentum training
  - Adds a fraction of the previous weight update to the present one
  - This is done to make use of the local trends. Keep following the trend
- Nesterov momentum
  - In addition to looking backwards – using past weights – this technique also looks ahead
- Adaptive gradient (ADAGRAD)
  - Done to decrease the risk of overshooting the global minimum
  - Divide each term by the sum of squares of its previous gradient
  - Used when datasets are relatively smaller (< 10K)
- RMS backpropagation
  - Avoids shrinkage of the learning rate over time
  - Used when datasets are large (> 10K)

Neural Networks

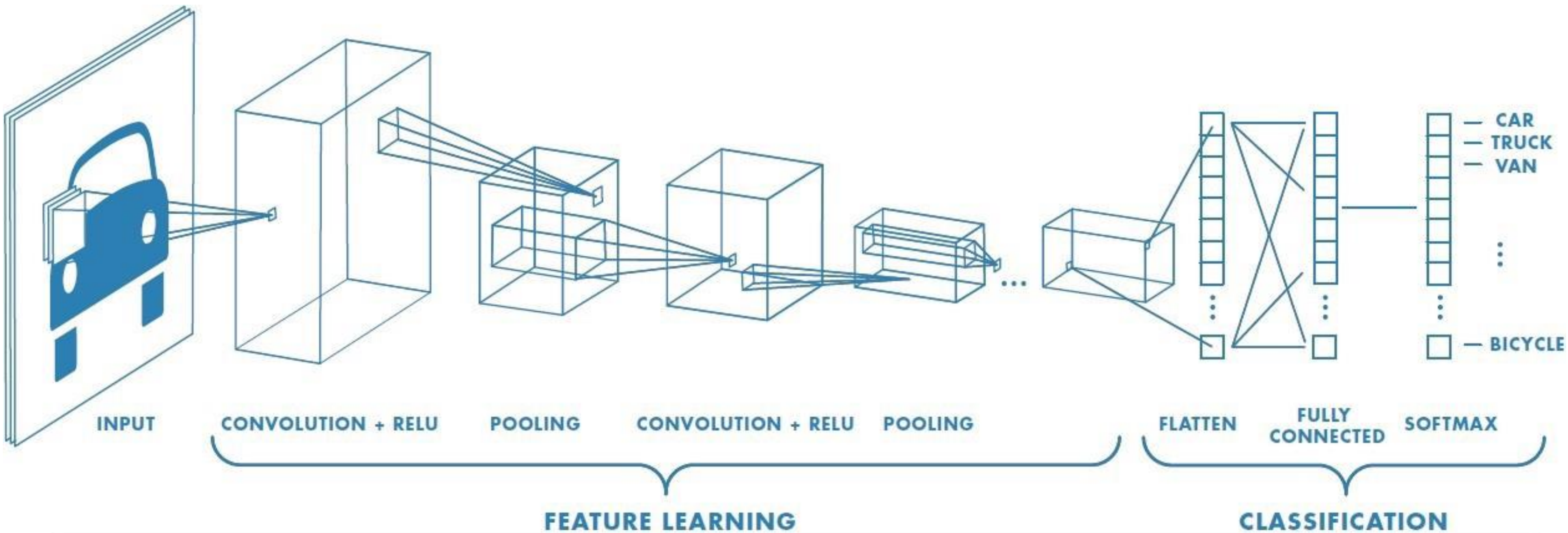


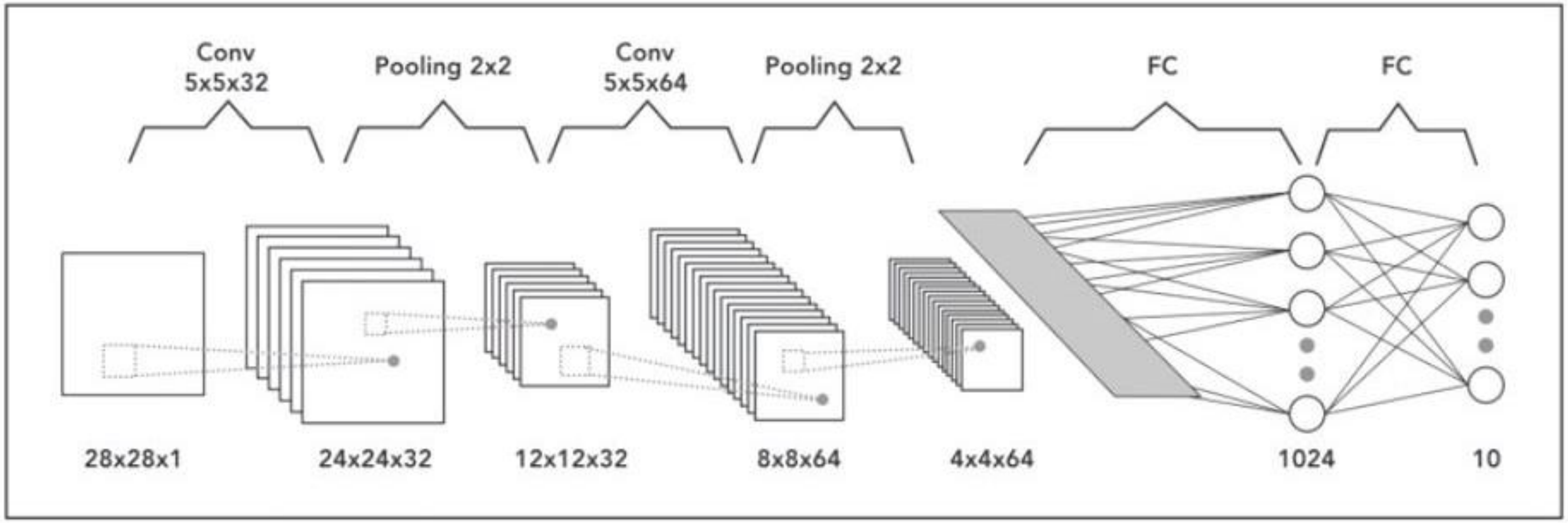
Ref: F. Van Veen "The Neural Network Zoo", 2016



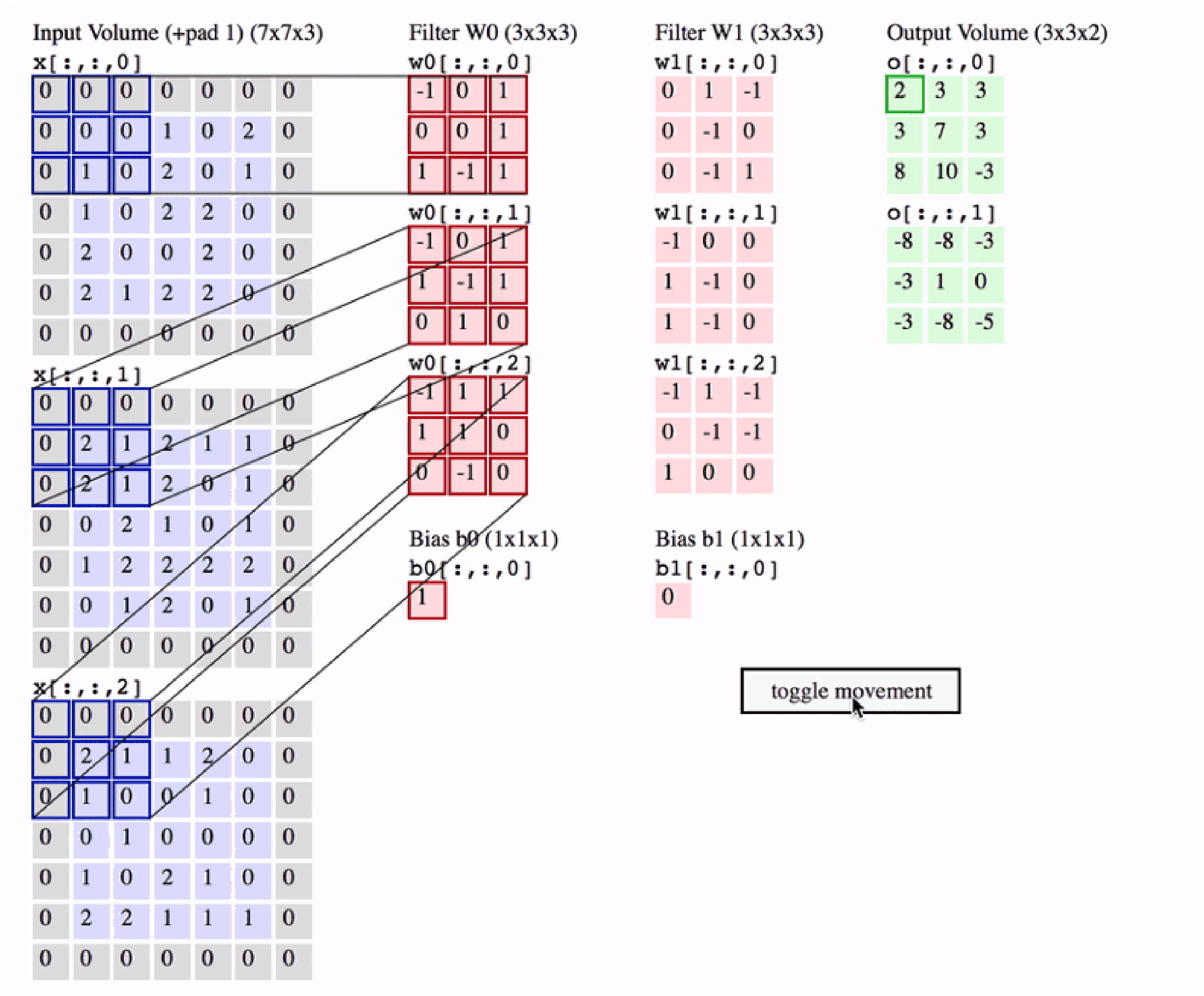
Ref: F. Van Veen "The Neural Network Zoo", 2016

Convolutional Neural Network

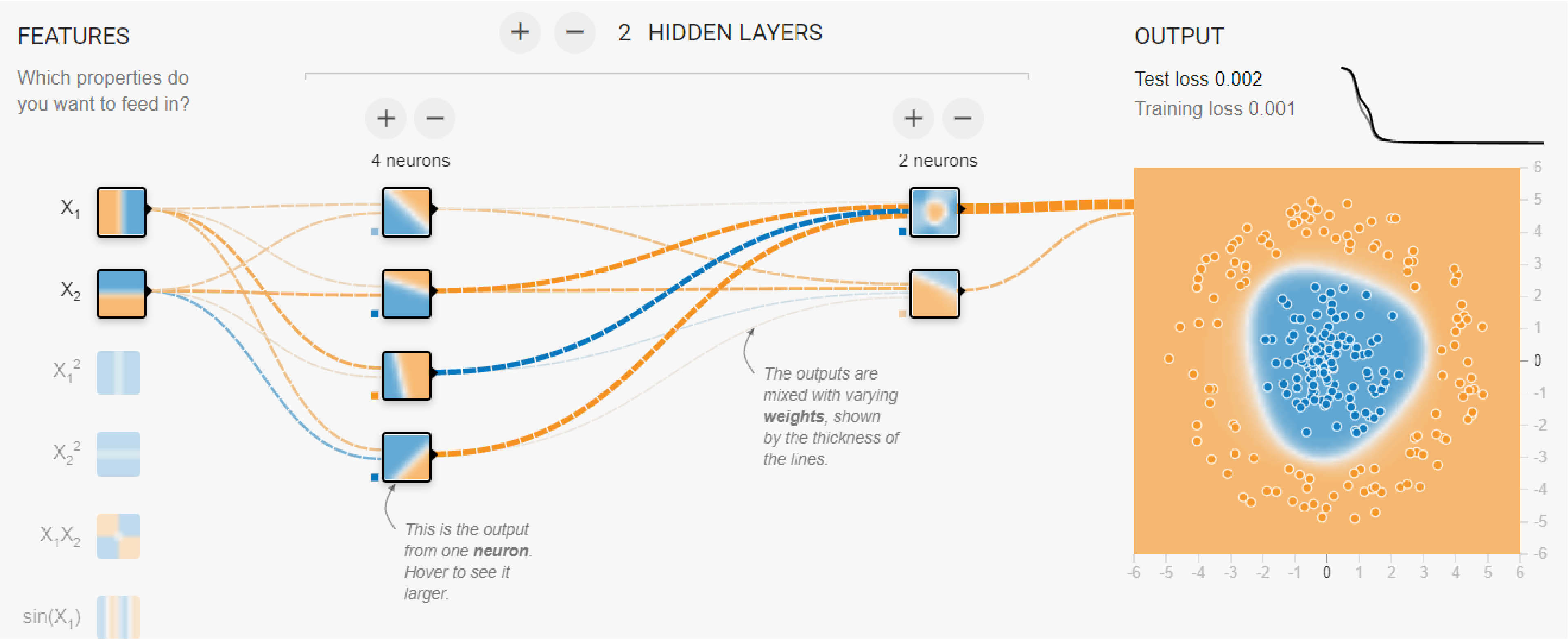




Convolutional Neural Network



Tinker With a Neural Network



https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.75977&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false