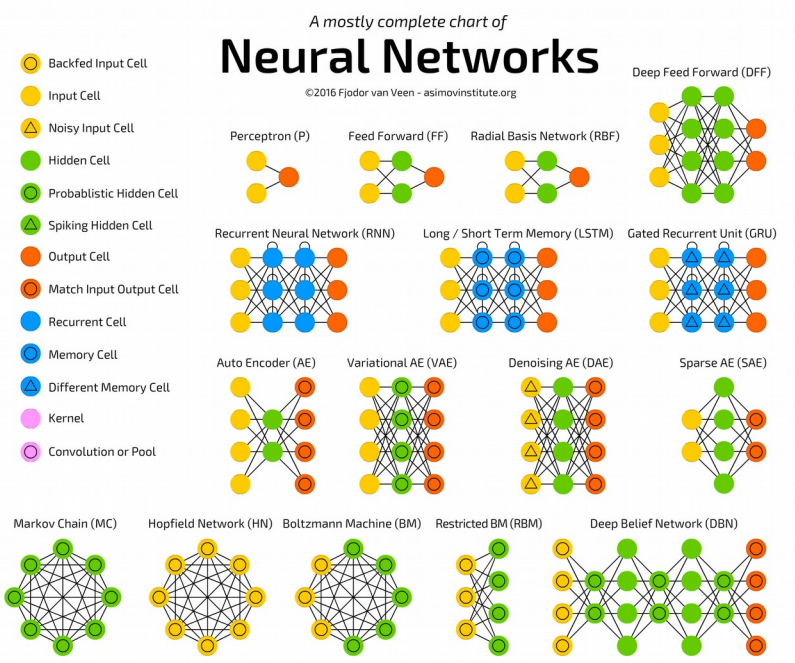
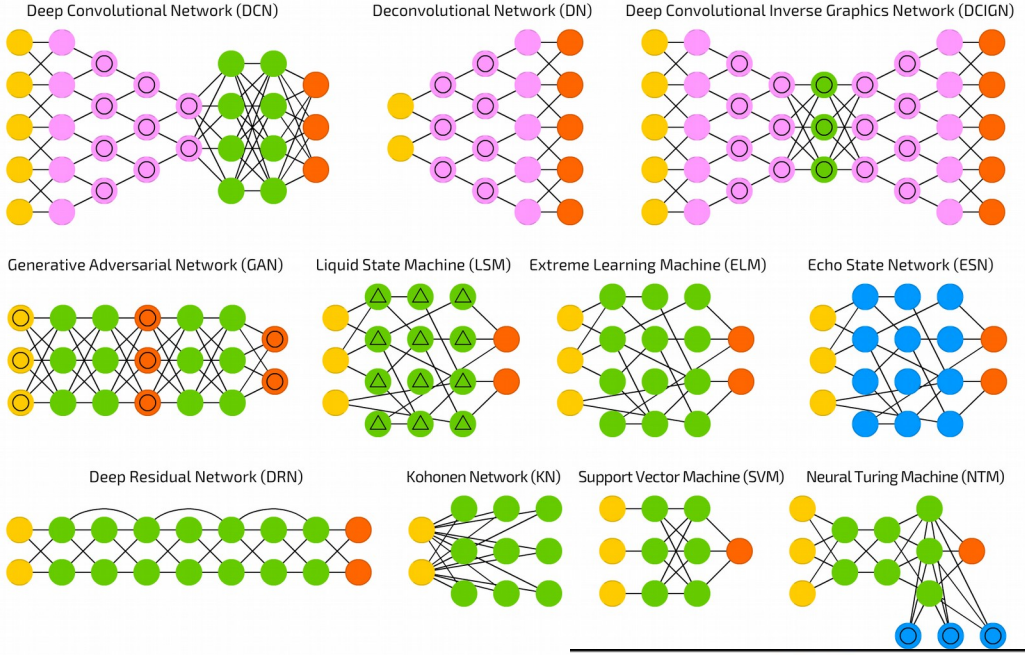
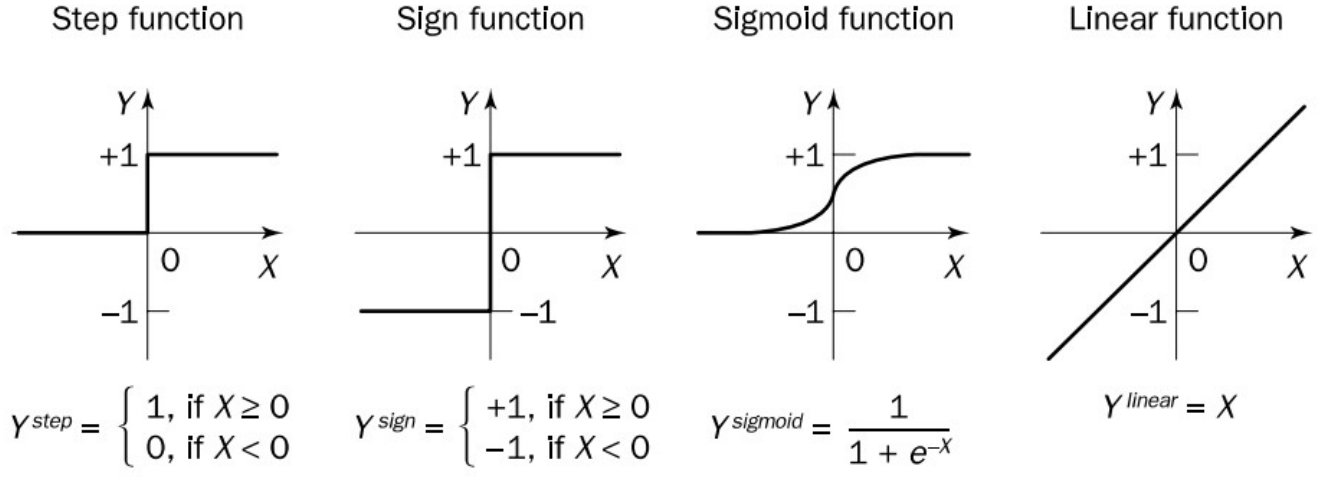
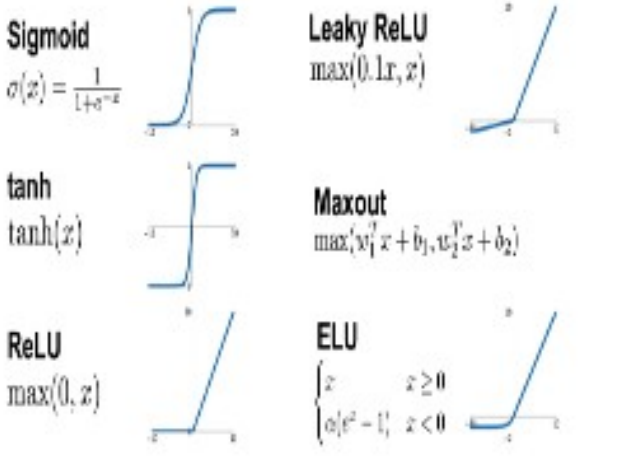

A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)


Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)


Step function
Sign function
Sigmoid function
Linear function
Y
X
0
+1
−1
$Y^{step} = \begin{cases} 1, \text{ if } X \geq 0 \\ 0, \text{ if } X < 0 \end{cases}$
$Y^{sign} = \begin{cases} +1, \text{ if } X \geq 0 \\ -1, \text{ if } X < 0 \end{cases}$
$Y^{sigmoid} = \frac{1}{1 + e^{-X}}$
$Y^{linear} = X$


Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$
Leaky ReLU
$\max(0.1x, x)$
tanh
$\tanh(x)$
Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$
ReLU
$\max(0, x)$
ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$

Map (SOM) tutorial

## Example

For vector 1100 *(We are using the Euclidean distance squared for convenience)*

$D(1) = (1-0.2)^2+(1-0.6)^2+(0-0.5)^2+(0-0.9)^2 = 1.86$

$D(1) = 1.86$, $D(2) = 0.98$

Hence J = 2. Note that $R = 0$, so we need not update the weights of any neighboring neurons.

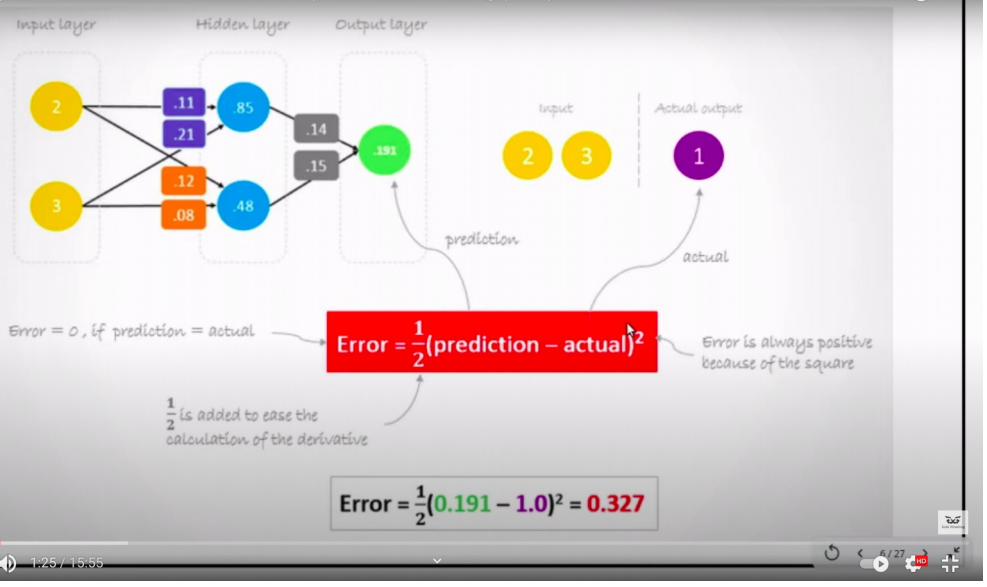
Using $w_{ij}(new) = w_{ij}(old) + \alpha(x_i - w_{ij}(old))$, the new weight matrix is

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = \begin{bmatrix} 0.2 & 0.92 \\ 0.6 & 0.76 \\ 0.5 & 0.28 \\ 0.9 & 0.12 \end{bmatrix}$$

$$w_{12}(new) = w_{12}(old) + \alpha(x_1 + w_{12}(old)) = 0.8 + 0.6(1 - 0.8) = 0.92$$

1:00 / 1:45

Back Propagation in Neural Network with an Example | Machine Learning (2019)




1:25 / 15:55

# Kohonen learning algorithm

- Step 1: Initialization
  - Initial synaptic weights are set to small random values $\in [0,1]$ and the learning rate $\alpha$ is a small positive value.
- Step 2: Activation and similarity matching
  - Apply the input vector $\mathbf{X}$ and find the winner-takes-all neuron $j_{\mathbf{X}}$ at iteration $\boldsymbol{p}$, using the minimum-distance Euclidean criterion

$$j_{\mathbf{X}}(p) = \min_{j} \left\| \mathbf{X} - \mathbf{W}_j(p) \right\| = \left\{ \sum_{i=1}^{n} \left[ x_i - w_{ij} \right]^2 \right\}^{1/2}, \qquad j = 1, 2, \ldots, m$$

where $\boldsymbol{n}$ is the number of neurons in the input layer, and $\boldsymbol{m}$ is the number of neurons in the output or Kohonen layer.

- Step 3: Learning
  - Update the synaptic weights: $w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$

    where $\boldsymbol{\Delta w_{ij}(p)}$ is the weight correction at iteration $\boldsymbol{p}$
  - The **competitive learning rule** (Haykin, 1999) for weight correction

$$\Delta w_{ij}(p) = \begin{cases} \alpha \left[ x_i(p) - w_{ij}(p) \right] & j \in \Lambda_j(p) \\ 0 & j \notin \Lambda_j(p) \end{cases}$$

    where $\boldsymbol{\alpha}$ is the learning rate $(0 < \alpha < 1)$, and $\boldsymbol{\Lambda_j(p)}$ is the neighborhood function centered around the winner-takes-all neuron $\boldsymbol{j_{\mathbf{X}}}$ at iteration $\boldsymbol{p}$
- Step 4: Iteration
  - Increase iteration $\boldsymbol{p}$ by one, go back to Step 2 and continue until the stopping criterion is satisfied.