Neural network

# Neurones

* Many input (– one output y
* A processing unit
  + transfer function
  + activation
  + if activation is low => y = -1, vice versa
  + simplify threshold b: =>

# The perception

* single neutron
* output is the characteristic function of half space, which is limited by hyperplane
* Training
  + Training set: with
  + Iterative algorithm
    - Initial weight value
    - Each training vector j compute
    - If then
  + Continue until no update
* Theorem: if 2 sets (positive and negative example) are linearly separable then the perception training algorithm will stop after finite loop
  + Proof: . So we need to find . Calculate to show that n cannot be infinite
* **Non-separable case**
  + . Good perception when
* Gradient descent
  + decrease f by moving opposite direction of gradient
  + New update
* Stochastic Gradient
  + Split training set into minibatches of size k. Updating w uses only minibatches. => quick convergence and less computation
* Cannot model XOR

# Multilayer

* Back-propagation algorithm:
  + Consider **one sample** (
  + Expect output and real output (
  + Error
  + Update weight to minimize error . I number of nodes in previous layer and J number of node in current layer
  + Layer 0:
  + Layer 1, 2,3: =
  + If
* Training
  + Converge to local minimum => try several random point
  + Stop iterations: maximum number of iterations, error is small or maximum time is reached
  + Overfitting: training size is insufficient

# Cybenko’s weak theorem

* Let f a continuous function [-1,1]^n -> R
* MLP with 2 hidden layers so that
* Idea: partition into hypercube
* First hidden layer: convert n continuous dimension into sub-intervals
* Second layer: create hypercube from sub-interval
* Output layer: linear transfer function with

# Summary

* Advantages:
  + General, applied into many case
  + Neural interpretation
  + Powerful in theory, efficient in practice
* Disadvantage
  + Slow training
  + Hard to find numbers of layers and neurons
  + Little understanding of real model

Hopfield Networks

# Associative Memory Network

* 1 layer network: Traning pattern
* weighting matrix
* properties: a\_i orthogonal, stable under small noise and low correlation

# Hopfield networks

* auto-associative network
* contain n neurons and activates others except itself
* weight are symmetric
* operation: initialize, fire and evolve, final stable state
* non-deterministic evolution:
  + select a neuron k
  + change state
* fundamental memory
* Energy
* Intuitive idea:
  + Fundamental memory are attractors on the energy surface
  + Fundamental memories are close are confusable
* Hebb weight: how to build a network with **known fundamental memories**?
  + Pattern ( what we want to learn)
  + Hebb weights with
    - contritbutes to w >0
    - contritbutes to w < 0

Self-Organization Map

* Network with 2 layers
* Topology: define neighbour of output layer
* Unsupervised method
* Convert continuous high dimension into discrete 2d-grid

# Competitive learning

### Best matching unit

* Input vector
* Output neuron k: weight vector
* Distance between output k and input x
* Closest :

### Training – Update for their neighbour

* Initialize random weight
* Iterative all training sample
* Compute best neuron k
* Update weight and their neighbour
* For better convergence, \lambda decreases with t

### Comment – Neuronal Interpretation

* Distance with input vectors
* Min distance => max => Winner neuron is also neuron with largest output

Simulated Annealing

# Thermodynamics

* Dynamical system S with states {x}
* Energy function E(x)
* For temperature T>0, system S converge to thermal equilibrium (state)
* At the thermal equilibrium, S will randomly fluctuates around values

# Metropolis algorithm

* Initialize state x\_0
* Select state x neighbour of x\_i
* Compute
  + accept move
  + accept move with
* reduce temperature
  + cooling slowly, global minimum energy
  + cooling quickly, local minimum
  + hard to find optimal one

# Algorithm Parameters

* Parameter:
  + Initial temperature
    - Hot enough to reach to neighbour solution
    - 2 methods: known problem to calculate or decide on proportion of up-hill moves
  + Cooling schedule
    - Empirical and theoretical show combination of cooling rate and number of cycles per iterations is a key to optimal convergence
  + Stopping condition
* Problem:
  + Choice of space of feasible solution
  + Form of the cost function
  + Neighbourhood structure employed
* Initialize Temperature problem
  + is changes in cost of j solution, is average change in cost of cost-increasing solution
  + we have .