Neural Network Lectures

ME 60033 IMS 2014

Reinforcement Learning, Recurrent Networks, Unsupervised Learning

Dr C S Kumar, Robotics and Intelligent Systems Lab

Other Neural Network Architectures

- Supervised Learning Architectures
  - Perceptron, MLP
  - Back Propagation
- Unsupervised Learning Architectures
- Recurrent Networks
  - Hopfield Networks
  - Associative Memory
  - Competitive Learning
  - Self Organizing Maps
- Pseudo recurrent Networks
- Reinforcement learning

Hopfield Networks, Constraint Satisfaction, and Optimization

- ***I.3 Dynamics and Adaptation in Neural Networks (Arbib)***
- ***III. Associative Networks (Anderson)***
- ***III. Energy Functions for Neural Networks (Goles)***

- ***8.2 Connectionist Models of Adaptive Networks***

Hopfield Networks

- A paper by John Hopfield in 1982 was the catalyst in attracting the attention of many physicists to "Neural Networks".

- In a network of McCulloch-Pitts neurons
- whose output is 1 iff $\Sigma w_{ij} s_j \geq \theta_i$ and is otherwise 0,
- neurons are updated synchronously: every neuron processes its inputs at each time step to determine a new output.

Hopfield Networks

- A Hopfield net (Hopfield 1982) is a net of such units subject to the asynchronous rule for updating one neuron at a time:

- "Pick a unit i at random.
- If $\sum w_{ij} s_j \geq \theta_i$, turn it on.
  - Otherwise turn it off."

- Moreover, Hopfield assumes symmetric weights:
- $w_{ij} = w_{ji}$

“Energy” of a Neural Network

- Hopfield defined the “energy”:

$$E = -\tfrac{1}{2} \sum_{ij} s_i s_j w_{ij} + \sum_i s_i \theta_i$$

- If we pick unit i and the firing rule (previous slide) does not change its s_i, it will not change E.

s_i: 0 to 1 transition

- If s_i initially equals 0, and $\Sigma\, w_{ij}s_j \geq \theta_i$

- then s_i goes from 0 to 1 with all other s_j constant,
- and the "energy gap", or change in E, is given by

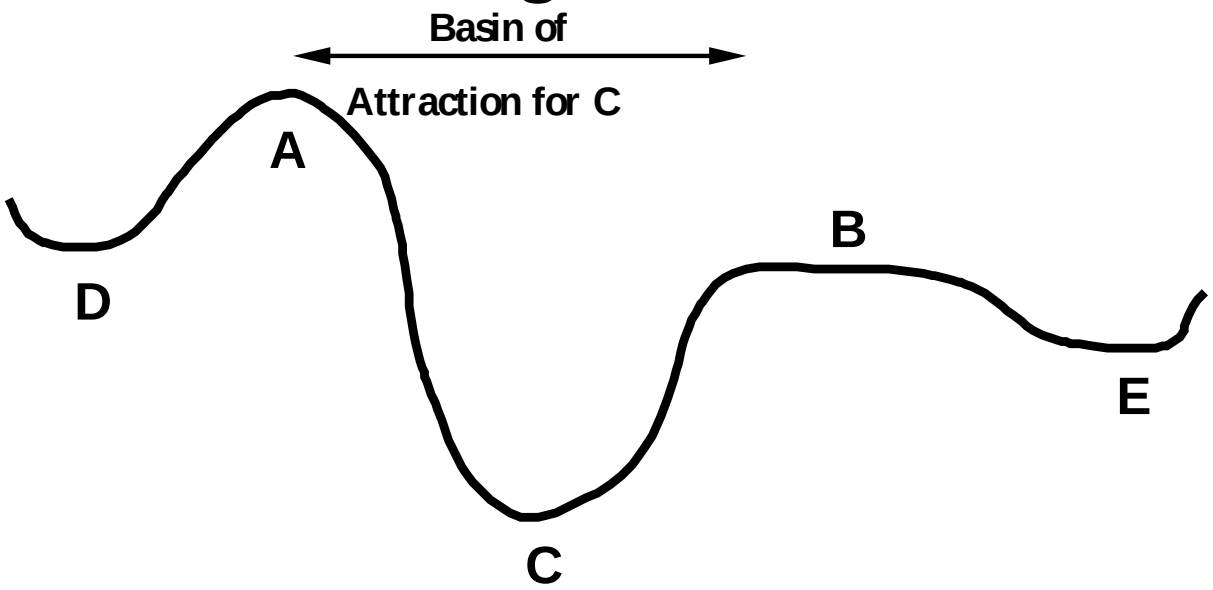
- $\Delta E = -\,½\, \Sigma_j\, (w_{ij}s_j + w_{ji}s_j) + \theta_i$
- $= -\,(\Sigma_{\,j}\, w_{ij}s_j - \theta_i)$ (by symmetry)
- $\leq\ 0.$

s_i: 1 to 0 transition

- If s_i initially equals 1, and $\Sigma\, w_{ij}s_j < \theta_i$
- then s_i goes from 1 to 0 with all other s_j constant
- The "energy gap," or change in E, is given, for symmetric w_{ij}, by:
- $\Delta E = \Sigma_j\, w_{ij}s_j - \theta_i < 0$
- ***On every updating we have $\Delta E \leq 0$***

Minimizing Energy

- On every updating we have $\Delta E \leq 0$
- Hence the dynamics of the net tends to move E toward a minimum.
- We stress that there may be different such states — they are *local* minima. Global minimization is not guaranteed.

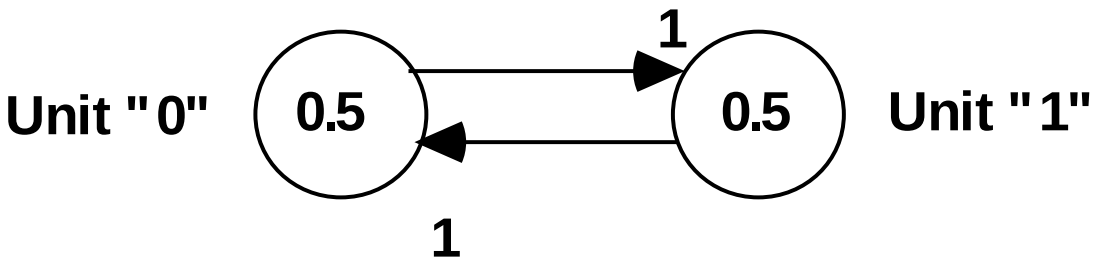


The Symmetry Condition $w_{ij} = w_{ji}$ is crucial for $\Delta E \leq 0$

- Without this condition
- $\frac{1}{2} \Sigma_j (w_{ij} + w_{ji}) s_j - \theta_i$ cannot be reduced to $(\Sigma_j w_{ij} s_j - \theta_i)$,
- so that Hopfield's updating rule cannot be guaranteed to yield a passage to energy minimum.
- It might instead yield a limit cycle -
- which can be useful in modeling control of action.
- In most vision algorithms: constraints can be formulated in terms of symmetric weights, so that $w_{ij} = w_{ji}$ is appropriate.

- In a control problem: a link w_{ij} might express the likelihood that the action represented by i should precede that represented by j, and thus $w_{ij} = w_{ji}$ is normally inappropriate.
-

The condition of asynchronous update is crucial

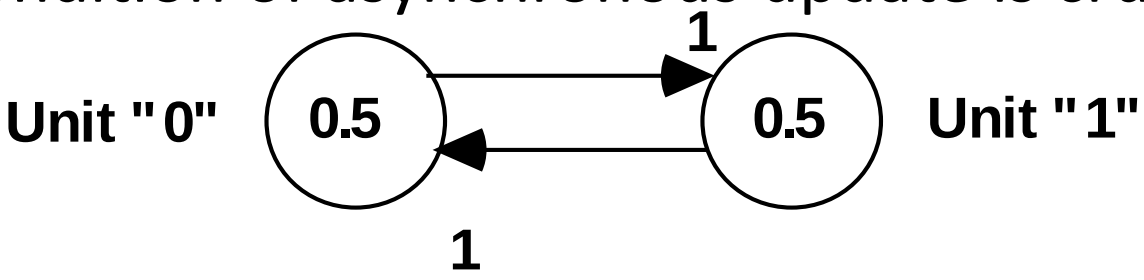


- Consider the above simple "flip-flop" with constant input 1, and with $w_{12} = w_{21} = 1$ and $\theta_1 = \theta_2 = 0.5$

- The McCulloch-Pitts network will oscillate between the states (0,1) and (1,0) or will sit in the states (0,0) or (1,1)

- There is no guarantee that it will converge to an equilibrium.
-

The condition of asynchronous update is crucial



However, with $E = -0.5 \Sigma_{ij} s_i s_j w_{ij} + \Sigma_i s_i \theta_i$

we have $E(0,0) = 0$

$E(0,1) = E(1,0) = 0.5$

$E(1,1) = 0$

and the Hopfield network will converge to the minimum at (0,0) or (1,1).

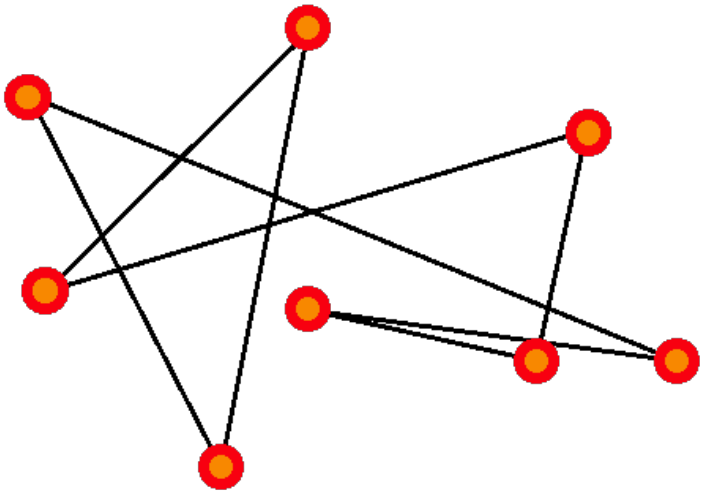
Hopfield Nets and Optimization

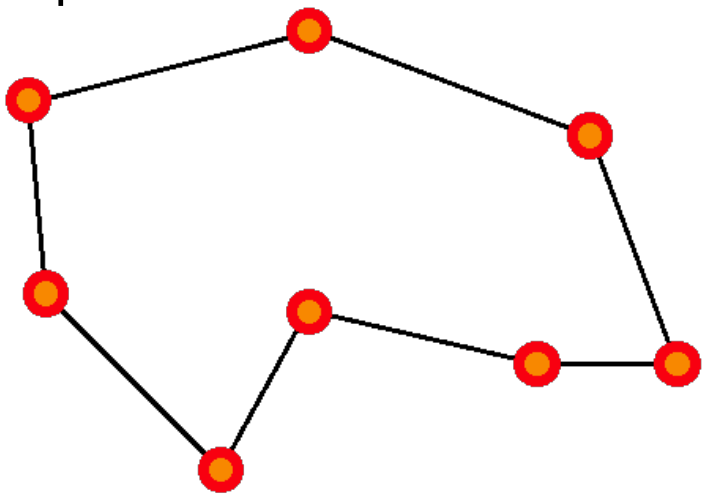
- To design Hopfield nets to solve optimization problems:
- given a problem, choose weights for the network so that E is a measure of the overall constraint violation.
- A famous example is the traveling salesman problem.

- Hopfield and Tank 1986 have constructed VLSI chips for such networks which do indeed settle incredibly quickly to a local minimum of E.
- Unfortunately, there is no guarantee that this minimum is an optimal solution to the traveling salesman problem. Experience shows it will be "a pretty good approximation," but conventional algorithms exist which yield better performance.

The traveling salesman problem 1

- There are n cities, with a road of length l_{ij} joining
- city i to city j.
- The salesman wishes to find a way to visit the cities that
- is optimal in two ways: each city is visited only once, and
- the total route is as short as possible.





Exponential Complexity

- Why is exponential complexity a problem?

- It means that the number of operations necessary to compute the exact solution of the problem grows exponentially with the size of the problem (here, the number of cities).

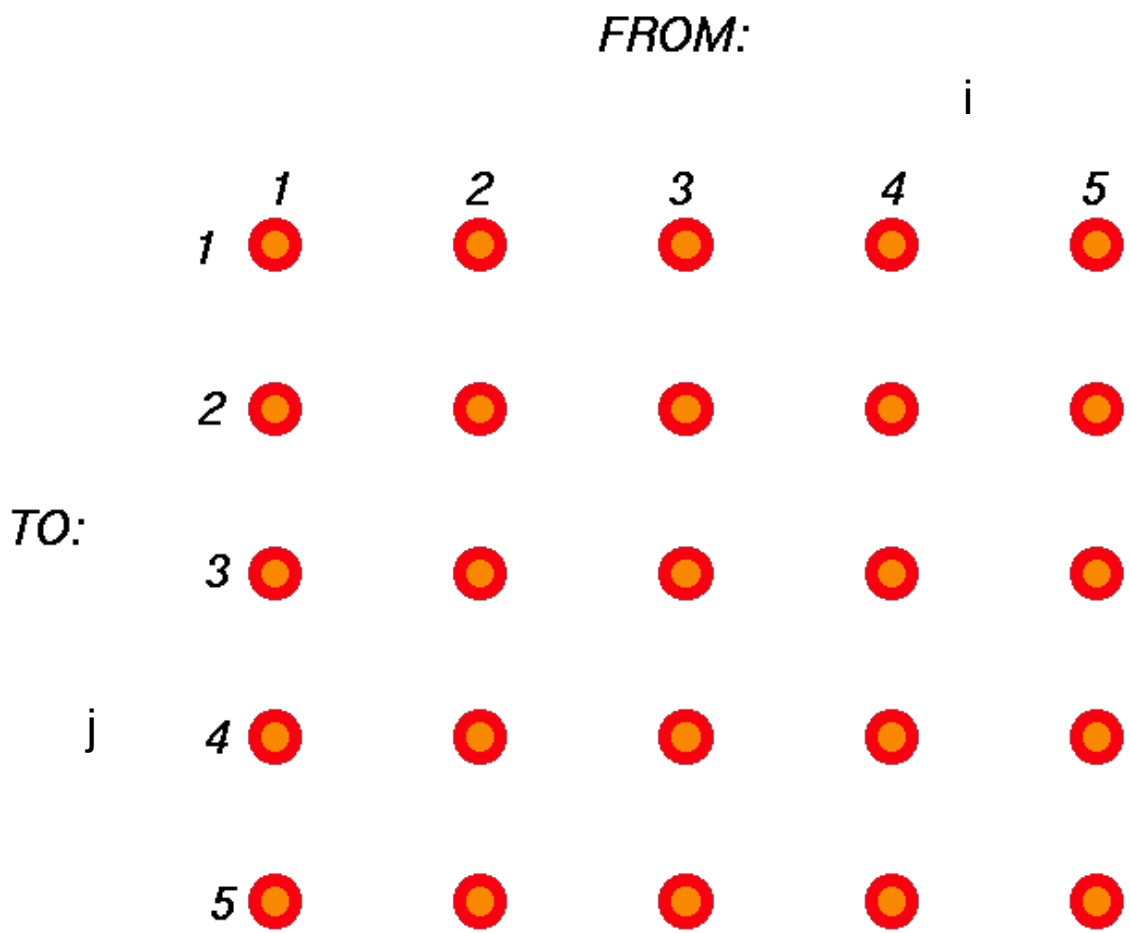
- exp(1) = 2.72
- exp(10) = $2.20\ 10^{4}$
- exp(100) = $2.69\ 10^{43}$
- exp(500) = $1.40\ 10^{217}$
- exp(250,000) = $10^{108,573}$ (Most powerful computer
- = 10^{12} operations/second)

The traveling salesman problem 2

- We build a constraint satisfaction network as follows:

- Let neuron N_{ij} express the decision to go straight from city i to city j. The cost of this move is simply l_{ij}.
- We can re-express the "visit a city only once" criterion by saying that, for city j, there is one and only one city i from which j is directly approached. Thus $(\Sigma_i N_{ij}-1)^2$ can be seen as a measure of the extent to which this constraint is violated for paths passing on from city j.
- Thus, the cost of a particular "tour" — which may not actually be a closed path, but just a specification of a set of paths to be taken — is
  - $\Sigma_{ij}\, N_{ij} l_{ij} + \Sigma_{\,j}\, (\Sigma_{\,i} N_{ij}-1)^2$.

Constraint Optimization Network

- 

The traveling salesman problem 3

- Cost to minimize: $\Sigma_{ij}\ N_{ij}l_{ij} + \Sigma_{\ j}\ (\Sigma_{\ i}N_{ij}-1)^2$

- Now $(\Sigma_{\ i}N_{ij}-1)^2 = \Sigma_{\ ik}N_{ij}N_{kj} - 2\ \Sigma_{\ i}N_{ij} + 1$
- and so $\Sigma_{\ j}(\Sigma_{\ i}N_{ij}-1)^2 = \Sigma_{\ ijk}N_{ij}N_{kj} - 2\ \Sigma_{\ ij}N_{ij} + n$
- $= \Sigma_{\ ij,kl}\ N_{ij}N_{kl}v_{ij,kl} - 2\ \Sigma_{\ ij}N_{ij} + n$
- where n is the number of cities
- $v_{ij,kl}$ equals 1 if $j = l$, and 0 otherwise.
- Thus, minimizing $\Sigma_{ij}\ N_{ij}l_{ij} + \Sigma_{\ j}\ (\Sigma_{\ i}N_{ij}-1)^2$ is equiv to minimizing

- $\Sigma_{\ ij,kl}\ N_{ij}N_{kl}v_{ij,kl} + \Sigma_{\ ij}N_{ij}(l_{ij}-2)$

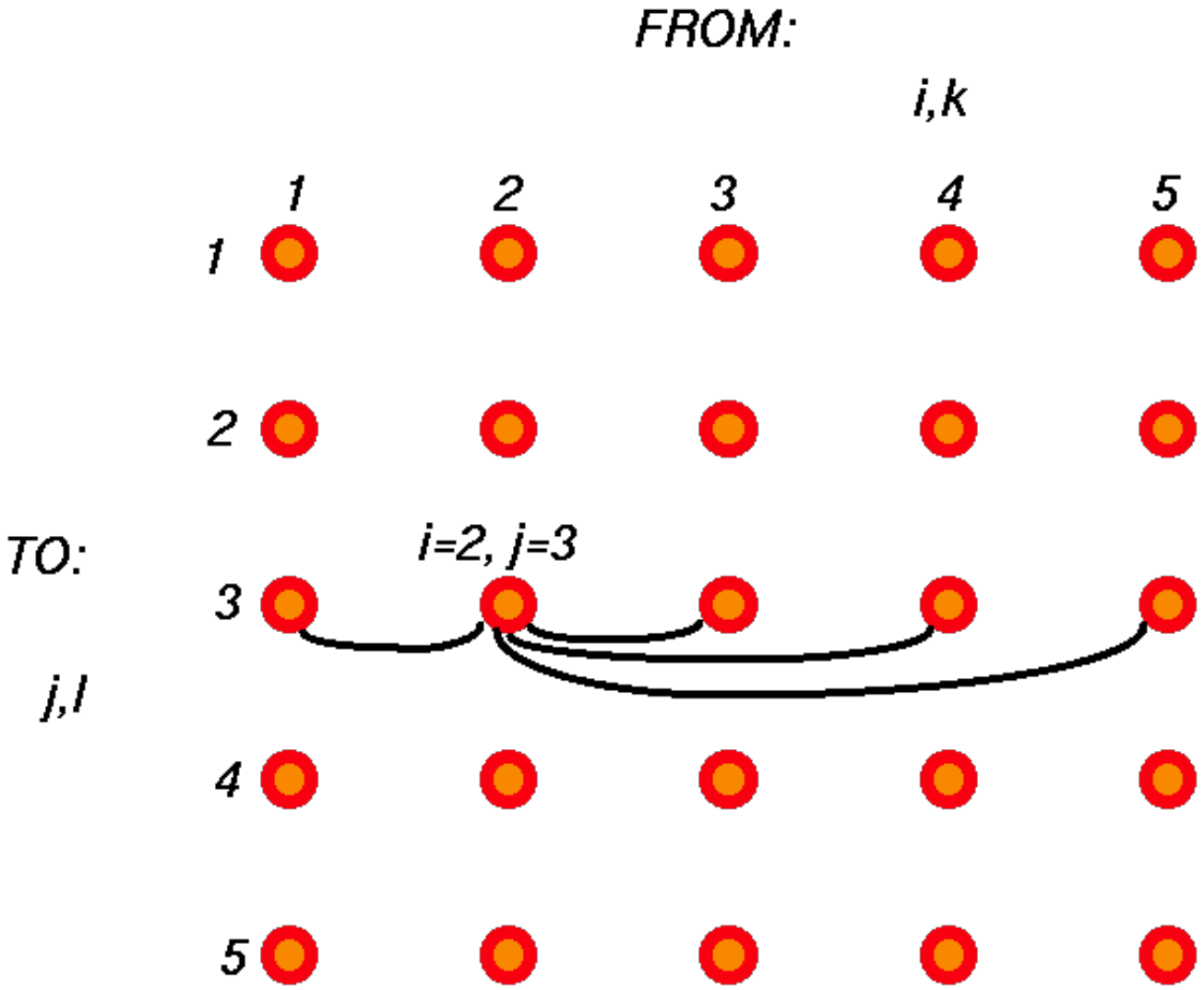
The traveling salesman problem 4

- minimize: $\Sigma_{ij,kl} N_{ij}N_{kl}v_{ij,kl} + \Sigma_{ij} N_{ij}(l_{ij}-2)$

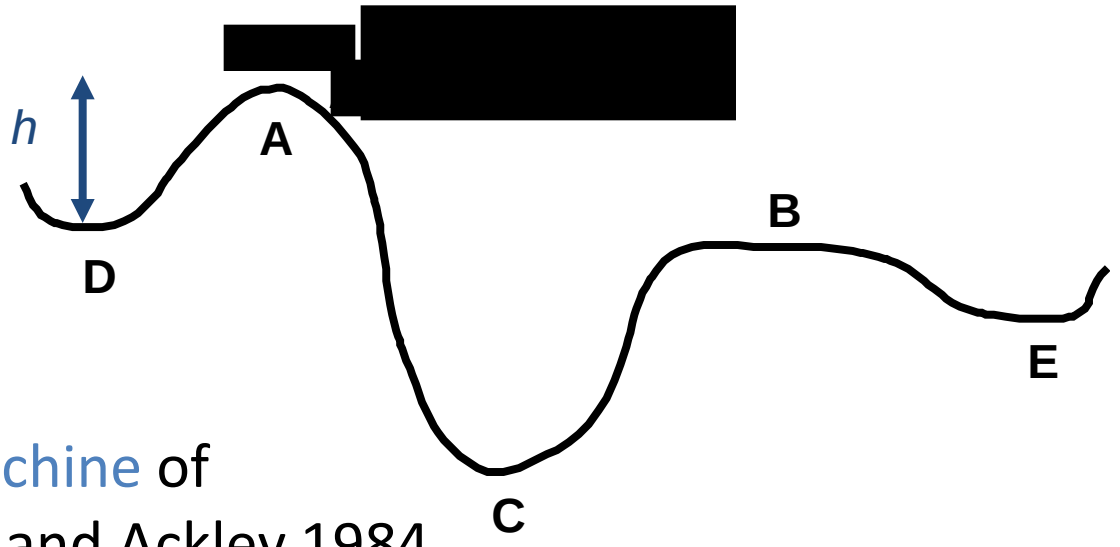
- Compare this to the general energy expression (with s_i now replaced by N_{ij}):
-
- $E = -1/2 \Sigma_{ij,kl} N_{ij}N_{kl}w_{ij,kl} + \Sigma_{ij} N_{ij}\theta_{ij}$.

- Thus if we set up a network with connections
- $w_{ij,kl} = -2\, v_{ij,kl}$ ($= -2$ if $j=l$, 0 otherwise) and
- $\theta_{ij} = l_{ij} - 2$,
- it will settle to a local minimum of E.

TSP Network Connections

- 

Boltzmann Machines



- The Boltzmann Machine of
- Hinton, Sejnowski, and Ackley 1984
- uses simulated annealing to escape local minima.
- To motivate their solution, consider how one might get a ball-bearing traveling along the curve to "probably end up" in the deepest minimum. The idea is to shake the box "about h hard" — then the ball is more likely to go from D to C than from C to D. So, on average, the ball should end up in C's valley.

Boltzmann's statistical theory of gases

- In the statistical theory of gases, the gas is described not by a deterministic dynamics, but rather by the probability that it will be in different states.

- The 19th century physicist Ludwig Boltzmann developed a theory that included a probability distribution of temperature (i.e., every small region of the gas had the same kinetic energy).

Boltzmann Distribution

- At thermal equilibrium at temperature T, the
- Boltzmann distribution gives the relative
- probability that the system will occupy state A vs.
- state B as:

$$\frac{P(A)}{P(B)} = \exp\left(-\frac{E(A)-E(B)}{T}\right) = \frac{\exp(E(B)/T)}{\exp(E(A)/T)}$$

- where E(A) and E(B) are the energies associated with states A and B.

Simulated Annealing

- Kirkpatrick et al. 1983:
- ***Simulated annealing*** is a general method for making likely the escape from local minima by allowing jumps to higher energy states.
- The analogy here is with the process of annealing used by a craftsman in forging a sword from an alloy.
- He heats the metal, then slowly cools it as he hammers the blade into shape.
  - If he cools the blade too quickly the metal will form patches of different composition;
  - If the metal is cooled slowly while it is shaped, the constituent metals will form a uniform alloy.

Simulated Annealing in Hopfield Nets

- Pick a unit i at random
- Compute $\Delta E = \sum_j w_{ij} s_j - \theta_i$ that would result from flipping s_i
- Accept to flip s_i with probability $1/[1+\exp(\Delta E/T)]$

NOTE: this rule converges to the deterministic rule in the previous slides when $T \rightarrow 0$

Optimization with simulated annealing:

- set T
- optimize for given T
- lower T (see Geman & Geman, 1984)
- repeat

Statistical Mechanics of Neural Networks

- A good textbook which includes research by physicists studying neural networks:

  – Hertz, J., Krogh. A., and Palmer, R.G., 1991, *Introduction to the Theory of Neural Computation*, Santa Fe Institute Studies in the Sciences of Complexity, Addison-Wesley.

Introduction to Radial Basis Functions

The idea of ***Radial Basis Function (RBF) Networks*** derives from the theory of function approximation. We have already seen how Multi-Layer Perceptron (MLP) networks with a hidden layer of sigmoidal units can learn to approximate functions. RBF Networks take a slightly different approach. Their main features are:

1. They are two-layer feed-forward networks.
2. The hidden nodes implement a set of radial basis functions (e.g. Gaussian functions).
3. The output nodes implement linear summation functions as in an MLP.
4. The network training is divided into two stages: first the weights from the input to hidden layer are determined, and then the weights from the hidden to output layer.
5. The training/learning is very fast.
6. The networks are very good at interpolation.

Exact Interpolation

The ***exact interpolation*** of a set of N data points in a multi-dimensional space requires every one of the D dimensional input vectors $\mathbf{x}^p = \{x_i^p : i = 1,...,D\}$ to be mapped onto the corresponding target output t^p. The goal is to find a function $f(\mathbf{x})$ such that

$$f(\mathbf{x}^p) = t^p \qquad \forall\, p = 1,...,N$$

The radial basis function approach introduces a set of N ***basis functions***, one for each data point, which take the form $\phi\left(\left\|\mathbf{x} - \mathbf{x}^p\right\|\right)$ where $\phi(\cdot)$ is some non-linear function whose form will be discussed shortly. Thus the pth such function depends on the distance $\left\|\mathbf{x} - \mathbf{x}^p\right\|$, usually taken to be Euclidean, between $\mathbf{x}$ and $\mathbf{x}^p$. The output of the mapping is then taken to be a linear combination of the basis functions, i.e.

$$f(\mathbf{x}) = \sum_{p=1}^{N} w_p \phi\left(\left\|\mathbf{x} - \mathbf{x}^p\right\|\right)$$

The idea is to find the "weights" w_p such that the function goes through the data points.

Determining the Weights

It is easy to determine equations for the weights by combining the above equations:

$$f(\mathbf{x}^q) = \sum_{p=1}^{N} w_p \phi\left(\left\|\mathbf{x}^q - \mathbf{x}^p\right\|\right) = t^q$$

We can write this in matrix form by defining the vectors $\mathbf{t} = \{t^p\}$ and $\mathbf{w} = \{w_p\}$, and the matrix $\mathbf{\Phi} = \left\{\Phi_{pq} = \phi\left(\left\|\mathbf{x}^q - \mathbf{x}^p\right\|\right)\right\}$. This simplifies the equation to $\mathbf{\Phi}\,\mathbf{w} = \mathbf{t}$. Then, provided the inverse of $\mathbf{\Phi}$ exists, we can use any standard matrix inversion techniques to give

$$\mathbf{w} = \mathbf{\Phi}^{-1}\,\mathbf{t}$$

It can be shown that, for a large class of basis functions $\phi(\cdot)$, the matrix $\mathbf{\Phi}$ is indeed non-singular (and hence invertable) providing the data points are distinct.

Once we have the weights, we have a function $f(\mathbf{x})$ that represents a continuous differentiable surface that passes exactly through each data point.

Linear models

- It is mathematically easy to fit linear models to data.
  - We can learn a lot about model-fitting in this relatively simple case.
- There are many ways to make linear models more powerful while retaining their nice mathematical properties:
  - By using non-linear, non-adaptive basis functions, we can get generalised linear models that learn non-linear mappings from input to output but are linear in their parameters – only the linear part of the model learns.
  - By using kernel methods we can handle expansions of the raw data that use a huge number of non-linear, non-adaptive basis functions.
  - By using large margin kernel methods we can avoid overfitting even when we use huge numbers of basis functions.
- But linear methods will not solve most AI problems.
  - They have fundamental limitations.

References

1. Simon Haykins – Neural Networks and Learning Machines
2. Christopher Bishop – Neural Networks in Pattern Recognition
3. Kevin Gurney – Introduction to Neural Networks
4. Hertz, Krogh, Palmer – Introduction to theory of Neural computation (Santa Fe Institute Series)
5. Handbook of Brain Sciences & Neural Networks