CS7140: Advanced Machine Learning **Spring 2020**

Lecture 10: Exact Inference, Hopfield network

Lecturer: Rose Yu *Scribes: Zlatan Fric, Jinxi Li*

10.1 Recap: Exact Inference

Some refreshers from previous lectures were presented which is useful in understanding Hopfield Network architecture.

10.1.1 Exact Inference Problems

Assume that a function P of a set of N variables $x \equiv \{x_n\}_{n=1}^N$ is defined as a product of M factors as follows:

$$P(\mathbf{X}) = \frac{1}{Z} \prod_{m=1}^{M} f(\mathbf{x}_m) \tag{10.1}$$

where the normalization constant Z is given by:

$$Z = \sum_{x} \prod_{m=1}^{M} f(\mathbf{x}_m) \tag{10.2}$$

The marginalization problem:

$$Z_n(x_n) = \sum_{\{x_n'\}, n' \neq n} P(\mathbf{x}) \tag{10.3}$$

10.1.2 Message Passing

As discussed in previous lectures, one way to solve the exact inference problem is through use of a message passing scheme which can be formulated as follows:

From variable to factor:

$$q_{n \to m(x_n)} = \prod_{m' \in \mathcal{M}(n) m} r_{m' \to n}(x_n) \tag{10.4}$$

From factor to variable:

$$r_{m \to}(x_n) = \sum_{x_m n} (f_m(x_m) \prod_{n' \in \mathcal{N}(m) \setminus n} q_{n' \to m}(x_{n'})) \tag{10.5}$$

Two types of messages passing along the edges:

- messages from $q_{n\to m}$ from variable nodes to factor nodes
- messages $r_{m\to n}$ from factor noces to variable nodes

For more detail please refer to previous lecture on Exact Inference.

10.1.3 Review of CNN/RNN

Please refer to lecture 9 to review weight and feedback architecture of CNN and RNN models.

10.2 Hopfield Network

10.2.1 Feedforward/Feedback Networks

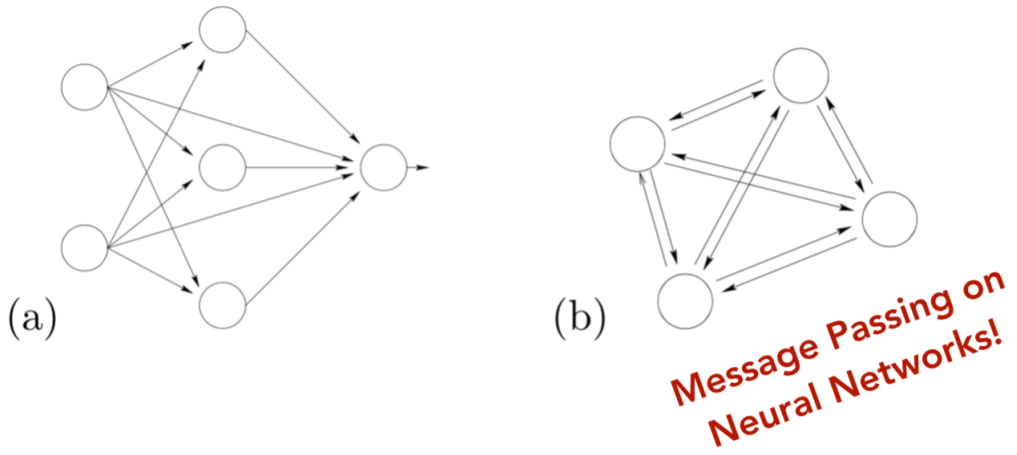

Figure 10.1: a) feed forward neural network b) fully connected feedback network (a case of message passing)

- Feed-forward: The first kind of networks that were developed were simple feed-forward networks. In a feedforward network, all the connections are directed such that the network forms a directed acyclic graph.
- Feedback: Any network that does not fall into the category of a feedforward network will be called a feedback network.
- Applications: Associated Memory, Optimization Problem (for example, using Hopfield network to solve travelling salesman problem)

10.2.2 Hopefield Network

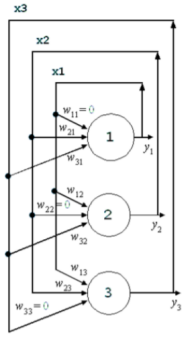

Figure 10.2: Diagram of a Hopfield Network with 3 nodes (1,2,3). Every pair of neurons is connected, as opposed to previous feed forward networks. We also note the feed-back pattern from y back to the hidden nodes.

Hopefield Nerual Networks were developed by and name after physicist John Hopfield. The first implementation was a binary implementation. The classical Hopfield Network is not used today, but has inspired a lot of feedback architectures used today which rely on the same fundamental principles.

Architecture: A Hopfield network has I number of neurons. All the neurons are fully connected with symmetric, bidirectional connections with weights $w_{ij} = w_{ji}$ Also, there are no self-connections, so $w_{ii}= 0$ for all i. Biases w_{i0} may be included (these may be viewed as weights from a neuron '0' whose activity is permanently $x_0 = 1$). We will denote the activity of neuron i (its output) by x_i

Activity rule single neuron update

$$X(a) = \begin{cases} 1 & a \geq 0 \\ -1 & a \leq 0 \end{cases} \tag{10.6}$$

Updates activation's

$$a_i = \sum_j w_{ij} x_j \tag{10.7}$$

Learning rule make a set of desired memory be stable states $x^{(n)}$

$$w_{ij} = \eta \sum_n x_i^{(n)} x_j^{(n)} \tag{10.8}$$

10.2.3 Hopfield Network Convergence

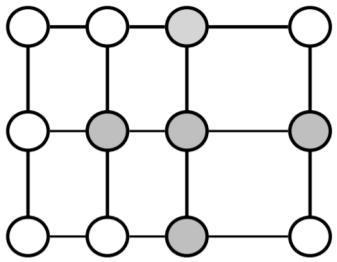

Figure 10.3: We previously encountered the activity rules (1.7,1.8) in the ising model, as the figure shows, with an array of spins (e.g., atoms that can take states ± 1) that are magnetically coupled to each other

Binary probability distribution:

$$P(x|\beta, J) = \frac{1}{Z}\exp[-\beta E(x; J)] \tag{10.9}$$

Spin system energy function:

$$E(x; J) = -\frac{1}{2}\sum_{m,n} J_{mn}x_m x_n - \sum_n h_n x_n \tag{10.10}$$

Approximating distribution (which is the activation in a hopfield network):

$$Q(x; J) = -\frac{1}{Z_Q}\exp(\sum_n a_n x_n) \tag{10.11}$$

Minimize the free energy:

$$\beta F(P^*, Q) = \beta\sum_x Q(x; a)E(x; J) = \sum_x Q(x; a) ln\frac{1}{Q(x; a)} \tag{10.12}$$

Rewrite free energy experssion:

$$F(P^*, Q) = \beta\sum_x Q(x; a)E(x; J) - H_Q \tag{10.13}$$

Variational free energy:

$$F(P^*, Q) = \beta(-\frac{1}{2}\sum_{m,n} J_{mn}\mathbb{E}_Q[x_m]\mathbb{E}_Q[x_n] - \sum_n h_n\mathbb{E}_Q[x_n]) - H_Q \tag{10.14}$$

The key takeaway: If we simply replace J by w, $\mathbb{E}_Q[x]$ by x, and h_n by w_{i0}, we see that the equations of the Hopfield network are identical to a set of mean-field equations that minimize the mean-field equations [MK]

$$F(P^*, Q) = -\beta\frac{1}{2}x^T W x - \sum_i H[(1 + x_n)/2] \tag{10.15}$$

10.2.4 A few items of note

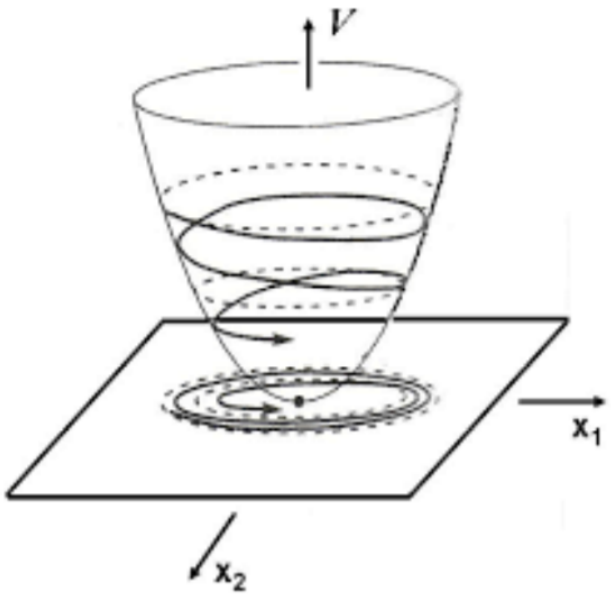

Figure 10.4: Lyapunov functions: if a system has a Lyapunov function then its dynamics are bound to settle down to a fixed point, which is a local minimum of the Lyapunov function, or a limit cycle, along which the Lyapunov function is a constant [MK]

- Stability: Hopfield network's dynamics will always converge to a stable fixed point.
- Generalized energy function for a system characterizes the dynamics of the system.
- A function V is positive definite and for all $\dot{V}(x) \leq 0$ for all x

10.2.5 Continuous Hopfield Network

We assume each neuron's activity x_i is a continuous function of time $x_i(t)$, then the activations are computed in accordance with

$$a_i(t) = \sum_j w_{ij} x_j(t) \tag{10.16}$$

The neuron's response to its activation is assumed to be mediated by the differential equation

$$\frac{d}{dt} x_i(t) = -\frac{1}{\tau}(x_i(t) - f(a_i)) \tag{10.17}$$

where $f(a)$ is the activation function. For a steady activation a_i, the activity $x_i(t)$ relaxes exponentially to $f(a_i)$ with time-constant τ. And as long as the weight matrix is symmetric, this system has the variational free energy as its Lyapunov function.

10.3 Application of Hopfield Network

10.3.1 Associative Memory

This is a brief example for associative memory. Here shows how learning and updates works.
First of all, let's focus on the weight for pixel one and pixel 2, and use the learning rule (10.8):

$$w_{12} = \sum_{n=1}^{4} x_1^{(n)} x_1^{(n)} = 1 + 1 + (-1) + (-1) = 0$$

Then this example shows the updating rule. Focus on case (b), the corresponding weights for the different pixel is highlighted by red line. Then use the activation rule (10.7) and updating rule (10.6):

$$a_{16} = \sum_{j=1, j\neq 16}^{25} w_{16,j} x_j = -28 < 0 \Rightarrow x_{16} = -1$$

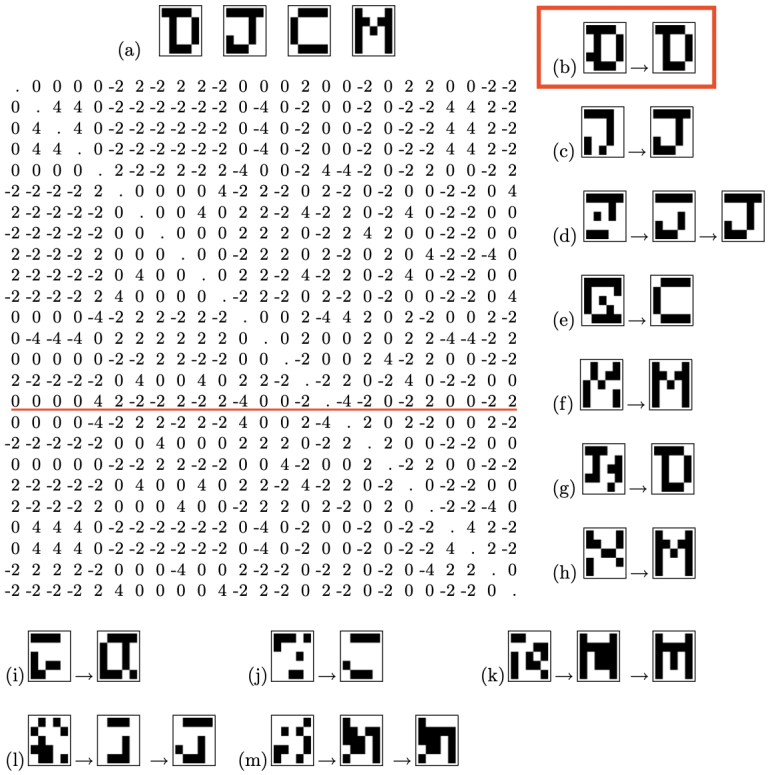

Figure 10.5: Binary Hopfield network storing four memories. a) The four memories and the weight matrix. b-h) Initial states differ by one to five bits from desired memory are restored to that memory. i-m) Some initial conditions that are far from the memories lead to stable states other than the four memories.

We can squash more memories into the network and maintain a stable memory (see figure 10.6), even when

some weights are randomly deleted. This shows the robustness of hopfield associated memory. However, storing too many patterns will cause catastrophical failure (see figure 10.7).

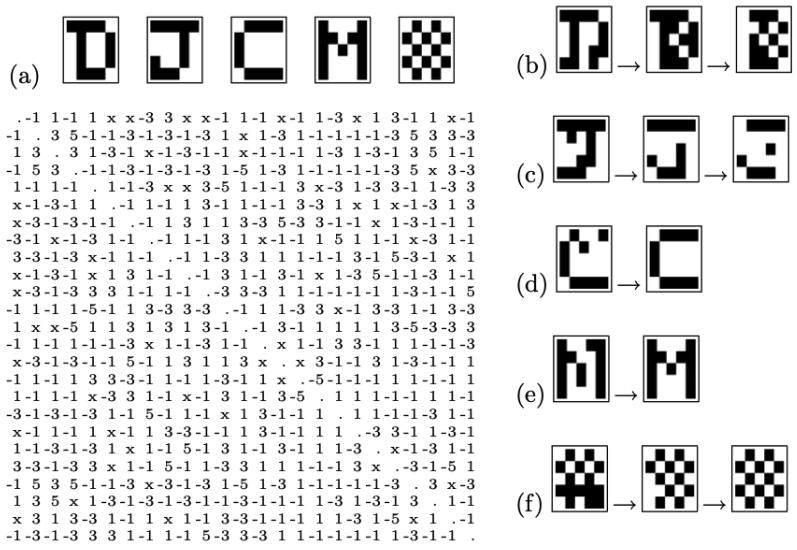

Figure 10.6: Hopfield network storing five memories and suffering deletion of 26 of the network. Initial states that differ by three random bits from a memory: some are restored, but some converge to other states.

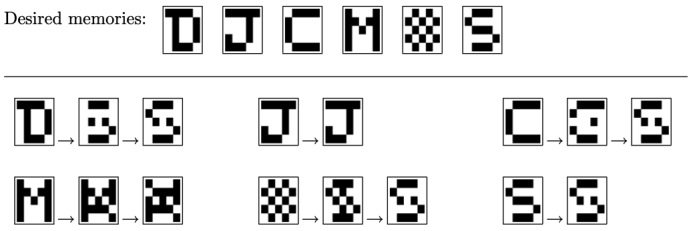

Figure 10.7: An overloaded Hopfield network trained on six memories,, most of which are not stable.

10.3.2 Solving Optimization Problem

Since the Hopfield network's dynamic minimize an energy function, we can map interesting optimization problems onto Hopfield network. The key idea here is constrained satisfaction.

One typical optimization problem is traveling salesman problem. For a set of K cities, and a matrix of K(K-2)/2 distances between cities, the problem is to find a closed tour of the cities, visiting each city once, that has the smallest total distance.

Consider K equals 4 as an example (see figure 10.8). States have exactly one '1' in every row/column, which requires putting large negative weights between any pair of neurons in the same row/column. And the weights must encoe the objective function that we want to minimize - the total distance, which requires putting negative weights proportional to the distance between the nodes.

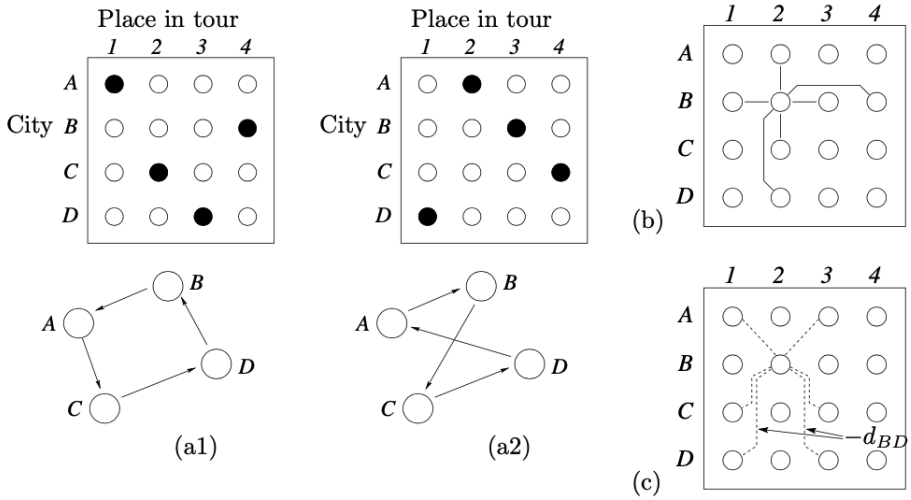

Figure 10.8: Hopfield network for solving a travelling salesman problem with K = 4 cities. a1), a2) Two solution states. b) The negative weights between B2 and other nodes. c) The negative weights that embody the distance objective function.

A solution to the TSP on Hopfield network is shown in figure 9

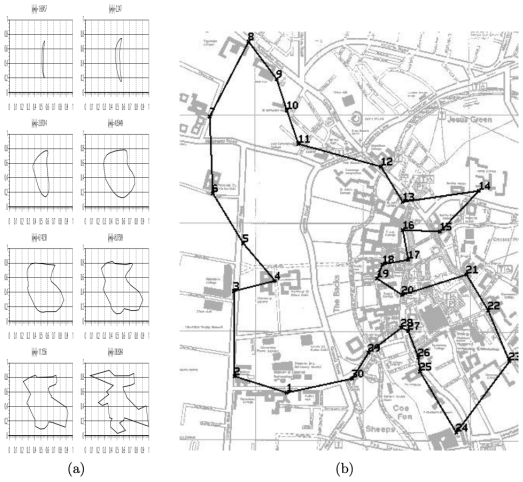

Figure 10.9: A solution to the 'travelling scholar problem' found be Aiyer using a continuous Hopfield network.

10.3.3 Graph Neural Networks

A graph includes nodes and edges. In a graph neural network, the state of a node depends on its neighbors, so we can use a generalized massage passing for graph. As shown in figure 10.10, the state of node x_1 depends on its neighbors and the edges between them, that is

$$x_i = \sum_{j \in \mathcal{N}(i)} f(l_i, l_{i,j}, x_j, l_j) \tag{10.18}$$

For GNN, the model is optimized by minimizing the energy, which is similar to Hopfield network.

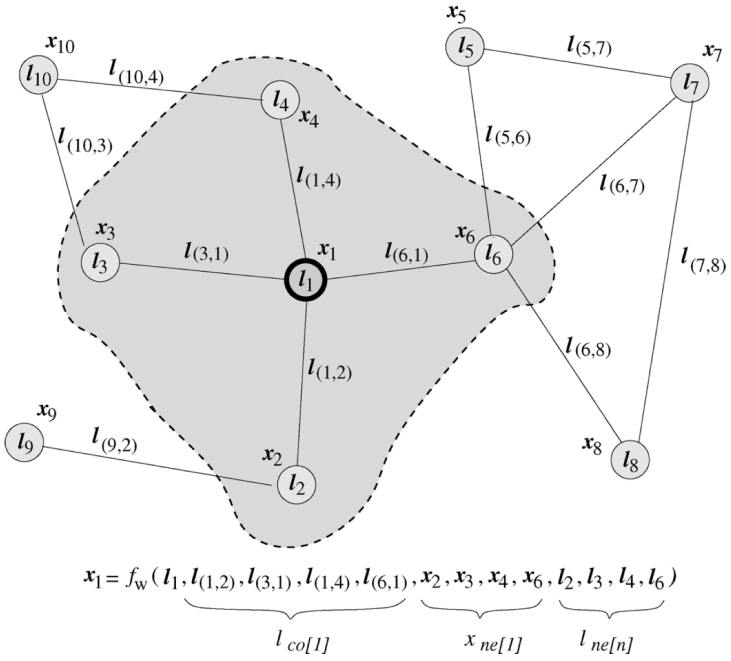

Figure 10.10: Graph and the neighborhood of a node. The state of the node x_i depends on the information contained in its neighborhood. [GNN]

10.3.4 Neural Message Passing

We can generalize message passing for graph, i.e. neural message passing. Suppose a graph G with node features x_v and edge features e_{vw}, we can apply message passing. For each step t, we first have message process. This process uses message function to sum over all the message from the neighborhood of feature x_v. The message function is

$$m_v^{t+1} = \sum_{w \in \mathcal{N}(v)} M_t(h_v^t, h_w^t, e_{vw}) \tag{10.19}$$

Update process updates the hidden state for feature v with the formal hidden state and the new message. The update function is thus given by

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}) \tag{10.20}$$

The readout process takes the hidden states for all nodes as input to make prediction. The readout function is defined as

$$\hat{y} = R(\{h_v\}) \tag{10.21}$$

A typical application of neural message passing can be shown from figure 10.11.

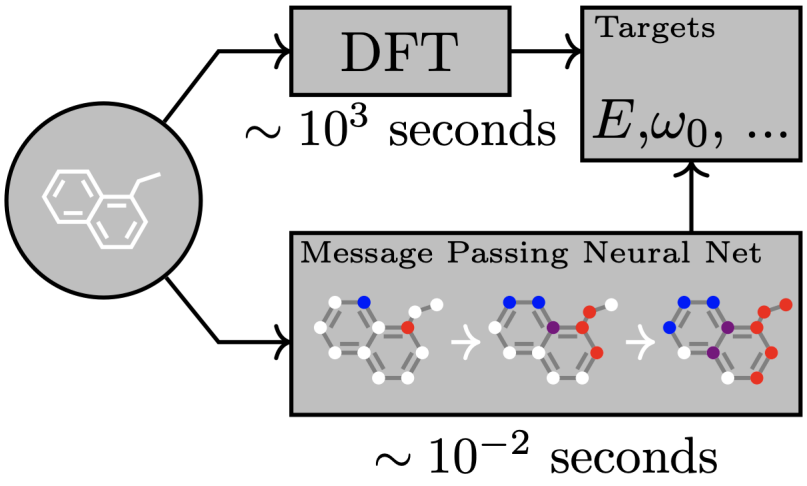

Figure 10.11: A Message Passing Neural Network predicts quantum properties of an organic molecule by modeling a computationally expensive DFT calculation. [NMP]

References

[MK] MacKay, David JC, "Information theory, inference and learning algorithms", *Cambridge university press*, 2013

[GNN] Scarselli, Franco, et al., "The graph neural network model", *IEEE Transactions on Neural Networks 20.1*, 2008, pp. 61-80

[NMP] Gilmer, Justin, et al., "Neural message passing for quantum chemistry", *Proceedings of the 34th International Conference on Machine Learning-volume 70. JMLR. org*, 2017