Artificial Neural Networks
Lecture Notes

Prof. Dr. Sen Cheng

Winter Semester 2019/20

14 Boltzmann machine

Boltzmann machines are stochastic neural networks that represent statistical distributions. The units in the Boltzmann machine are divided into visible units, v, and hidden units, h (Fig. 1). The visible units receive inputs, i.e. the training set is a set of binary vectors over the set V. The hidden variables represent *latent variables* that influence the activity of the visible variables, but are not observed directly.

Boltzmann machines are generalizations of the Hopfield net and have several advantages over the Hopfield net or ffw networks.

1. They are *generative*, i.e., once they are trained, they can generate different instances of the data according to a certain distribution.
2. They can assign probabilites to states they have never seen before. In many situations, e.g., for a nuclear power plant, we cannot sample all possible states, because some states are dangerous. In these cases, supervised learning of desirable and undesirable states doesn't work.
3. The Hopfield net gets stuck in local minima quite easily. Stochasticity helps to move on to a better local minimum.

Boltzmann machines use the same energy function as the Hopfield net.

$$E(\theta, s) \quad = \quad -\frac{1}{2}\sum_{j,k} w_{jk} s_j s_k - \sum_j b_j s_j \tag{1}$$

where s_i is the state of the i-th unit, w_{ij} the symmetric weight between unit i and j, and b_i the bias.

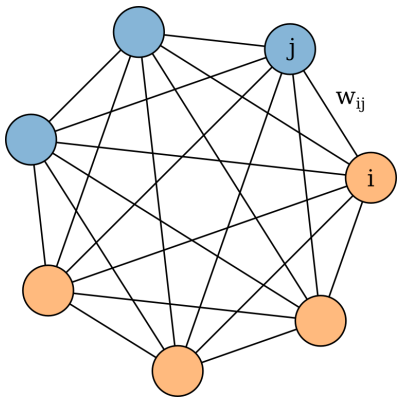

Figure 1: Boltzmann machine. Orange: visible units; blue: hidden units.

Additional material:

- Video of lecture by Geoffrey Hinton.

14.1 Stochastic neural network

Unlike all the networks we have discussed before, the units in a Boltzmann machine are stochastic. Describe the state as the vector s. Then each unit is either active ($s_i = 1$) or inactive ($s_i = 0$). The probability of finding the system in the state s is given by the *Boltzmann distribution*:

$$P(s) = \frac{\exp(-E(s))}{\sum_{s'} \exp(-E(s'))}. \tag{2}$$

In many treatments of the Boltmann machine, the energy is divided by a scalar T, the temperature of the system. The temperature has an important meaning in statistical physics and influences the networks dynamics. However, in practice it turns out that the temperature is not very relevant for Boltzmann machines and we therefore set the temperature to $T = 1$. The Hopfield network could be viewed as a Boltzmann machine at temperature $T = 0$. The denominator in Eq. 2 is known as the *partition function*:

$$Z = \sum_{s} \exp(-E(s)), \tag{3}$$

The summation is over all possible states of the system, of which there is a very large number in any reasonably sized system. Therefore, the partition function is very costly to calculate and the probability of observing a particular state is not practicably computable.

The trick is to find ways to compute meaningful quantities without having to compute the partition function. For instance, the differences in energy ΔE_i that results from a single unit i being off ($s_i = 0$) versus on ($s_i = 1$). Assuming a symmetric matrix of weights:

$$\Delta E_i = E(s_i = 0|s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n) - E(s_i = 1|s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n) \tag{4}$$

To simplify the notation, we will write $i = \text{off}$ for $s_i = 0|s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n$ and $i = \text{on}$ for $s_i = 1|s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n$

$$\Delta E_i = E(i = \text{off}) - E(i = \text{on}) \tag{5}$$

Since the energy is given by Eq. 1, we can compute the energy difference from the network activity (see also the derivation of a similar equation in the Hopfield lecture)

$$\Delta E_i = \sum_{j \neq i} w_{ij} s_j + b_i \tag{6}$$

Next, we would like to also set the network activity based on the energy difference. To this end, we substitute the energy of each state in Eq. 5 with its relative probability according to the Boltzmann factor (the property that the energy of a state is proportional to the negative log probability of that state) gives:

$$\Delta E_i = -\log P(i = \text{off}) + \log P(i = \text{on}) \tag{7}$$

We then rearrange terms and consider that the probabilities of the unit being on and being off must sum to one, and $p_i = P(i = \text{on})$:

$$\Delta E_i = \log p_i - \log(1 - p_i) \tag{8}$$

$$\Delta E_i = \log\left(\frac{p_i}{1 - p_i}\right) \tag{9}$$

$$-\Delta E_i = \log\left(\frac{1 - p_i}{p_i}\right) \tag{10}$$

$$-\Delta E_i = \ln\left(\frac{1}{p_i} - 1\right) \tag{11}$$

$$\exp(-\Delta E_i) = \frac{1}{p_i} - 1 \tag{12}$$

Solving for p_i, the probability that the i-th unit is on gives:

$$p_i = \frac{1}{1+\exp(-\Delta E_i)} = \sigma(\Delta E_i) \tag{13}$$

This relation is the source of the logistic function found in probability expressions in variants of the Boltzmann machine. Plugging in Eq. 6 yields the final equation:

$$p_i = \sigma\left(\sum_{j \neq i} w_{ij} s_j + b_i\right) \tag{14}$$

Eq. 14 defines a neural network, in which each unit calculates a linear weighted sum of its inputs from other units and then applies a stochastic activation function.

14.2 Thermal equilibrium

An important concept in the study of stochastic systems, is the concept of *equilibrium*, or *thermal equilibrium*. It doesn't mean that the network state is constant. This cannot occur since the state of the network is stochastic and therefore constantly changing. However, if the network is left alone and updates its state a sufficient number of times, then it will reach a point in which the states generated by the network will be distributed according to the Boltzmann distribution Eq. 2. If the network is disturbed, e.g. by external inputs, then the previous statement will not be true. The distribution of states will depend on the initial state from which the process was started, and not just on T.

Running the network beginning from a high temperature, its temperature gradually decreases until reaching a thermal equilibrium at a lower temperature. It then may converge to a distribution where the energy level fluctuates around the global minimum. This process is called simulated annealing.

To train the network so that the chance it will converge to a global state is according to an external distribution over these states, the weights must be set so that the global states with the highest probabilities get the lowest energies. This is done by training.

14.3 Learning rule

We define the loss function such that minimizing the the loss function corresponds to maximizing the likelihood of observing the data vectors v, i.e.,

$$L(\theta) = -\log \prod_v p(v) \tag{15}$$

where we use the log-function because it greatly simplifies the math and since the log function is strictly monotonically increasing, it does not change the solution.

$$L(\theta) = -\sum_v \log p(v) \tag{16}$$

We define

$$l(\theta, v) = \log p(v) \tag{17}$$

To find the weights, we need the gradient:

$$\Delta w_{ij} = -\eta \frac{\partial L(\theta)}{\partial w_{ij}} = \eta \sum_v \frac{\partial l(\theta, v)}{\partial w_{ij}} \tag{18}$$

$$l(\theta, v) = \log p(v) = \log \sum_h p(v, h) \tag{19a}$$

At thermal equilibrium, the probability is given by the Boltzman equation Eq. 2

$$= \log \sum_h e^{-E(v,h)} - \log \sum_{v,h} e^{-E(v,h)} \tag{19b}$$

$$\frac{\partial l(\theta,v)}{\partial w_{ij}} = -\frac{1}{\sum_h e^{-E(v,h)}}\sum_h e^{-E(v,h)}\frac{\partial E(v,h)}{\partial w_{ij}} + \frac{1}{\sum_{v,h} e^{-E(v,h)}}\sum_{v,h} e^{-E(v,h)}\frac{\partial E(v,h)}{\partial w_{ij}} \tag{19c}$$

The gradient of the energy function is $\frac{\partial E(v,h)}{\partial w_{ij}} = -\frac{1}{2}s_i s_j$. Using the relationship $e^{-E(v,h)} = Zp(v,h)$ and $Z = \sum_{v,h} e^{-E(v,h)}$

$$= \frac{1}{2}\frac{1}{\sum_h Zp(v,h)}\sum_h Zp(v,h)s_i s_j - \frac{1}{2}\frac{1}{Z}\sum_{v,h} Zp(v,h)s_i s_j \tag{19d}$$

Since Z is a constant

$$= \frac{1}{2}\frac{1}{\sum_h p(v,h)}\sum_h p(v,h)s_i s_j - \frac{1}{2}\sum_{v,h} p(v,h)s_i s_j \tag{19e}$$

Using $\sum_h p(v,h) = p(v)$ and $p(h|v) = p(v,h)/p(v)$

$$= \frac{1}{2}\sum_h \frac{p(v,h)}{p(v)}s_i s_j - \frac{1}{2}\sum_{v,h} p(v,h)s_i s_j \tag{19f}$$

$$= \frac{1}{2}\sum_h p(h|v)s_i s_j - \frac{1}{2}\sum_{v,h} p(v,h)s_i s_j \tag{19g}$$

$$= \frac{1}{2}\mathrm{E}\left[s_i s_j\right]_{p(h|v)} - \frac{1}{2}\mathrm{E}\left[s_i s_j\right]_{p(v,h)} \tag{19h}$$

These summations are pratically impossible to calculate because the number of states is immense. Therefore, we can only sample these terms as an approximation of the true gradient.

$$\frac{\partial l(\theta,v)}{\partial w_{ij}} \approx \frac{1}{2}\left\langle s_i s_j\right\rangle_{p(h|v)} - \frac{1}{2}\left\langle s_i s_j\right\rangle_{p(v,h)} \tag{20a}$$

So, the weight updates are

$$\Delta w_{ij} = \eta\sum_v \left[\left\langle s_i s_j\right\rangle_{p(h|v)} - \left\langle s_i s_j\right\rangle_{p(v,h)}\right] \tag{20b}$$

$$= \eta\sum_v \left[\left\langle s_i s_j\right\rangle_{data} - \left\langle s_i s_j\right\rangle_{model}\right] \tag{20c}$$

Two phases are required to sample $\langle s_i s_j\rangle_{data}$ and $\langle s_i s_j\rangle_{model}$. The positive phase, where the visible units are clamped to the data, and a negative phase, where the visible units are free to change. The algorithm is summarized in Alg. 14.1. The first term stems from the numerator of the Boltzmann distribution Eq. 2. It increases the weight between two units that are strongly correlated in the data-driven distribution. That is how the network learns correlations between units. By contrast, the second term stems from the partition function, i.e., the denominator in Eq. 2, and reduces weights between units that are strongly correlated in the model distribution.

The update rule for the bias terms are equivalent:

$$\Delta b_i = \frac{\eta}{T}\sum_v \left[\left\langle s_i\right\rangle_{data} - \left\langle s_i\right\rangle_{model}\right] \tag{20d}$$

The learning rule for the Boltzmann machine is far more biologically plausible than backpropagation because the only information needed to change the weights is provided by "local" information. That is, the connection does not need information about anything other than the two units it connects. Nevertheless, the Boltzmann machine learning rule is impractical because the sampling takes a very long time due to the large number of states, and because of the time required to reach thermal equilibrium.

Algorithm 14.1 Training a Boltzmann Machine

repeat
  Permute the order of data points
  for all mini-batches (size around $10-100$) **do**
    {Sample from data distribution (positive phase)}
    repeat
      Randomly select a data point i
      Initialize visible units to x^i (and do not update visible units)
      Initialize hidden units to random values $\sim B(0.5)$
      repeat {burn-in phase}
        Update probabilities of hidden units according to Eq. 14 and sample activities
      until thermal equilibrium is reached
      repeat {sampling phase}
        Update probabilities of hidden units according to Eq. 14 and sample activities {visible units are clamped to input}
        Save state as s_{data}
      until enough samples have been collected
    until sufficient data points have been sampled
    {Sample from model distribution (negative phase)}
    Initialize visible and hidden units to random values $\sim B(0.5)$
    repeat {burn-in phase}
      Update probabilities of visible and hidden units according to Eq. 14 and sample activities
    until thermal equilibrium is reached
    repeat {sampling phase}
      Update probabilities of visible and hidden units according to Eq. 14 and sample activities
      Save state as s_{model}
    until enough samples have been collected
    Calculate $\langle s_i \rangle_{data}$ and $\langle s_i s_j \rangle_{data}$ for the mini-batch
    Calculate $\langle s_i \rangle_{model}$ and $\langle s_i s_j \rangle_{model}$ for the mini-batch
    Update weights and biases
  end for
until predetermined number of epochs is reached

14.4 Restricted Boltzmann machine

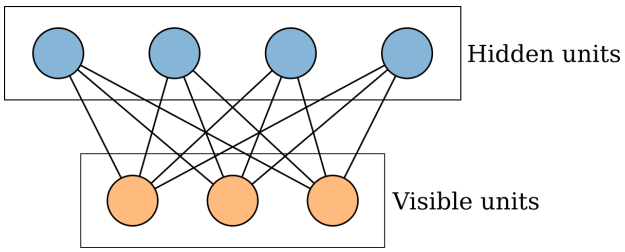

Figure 2: Restricted Boltzmann Machine (RBM).

Although learning is impractical in general Boltzmann machines, it can be made quite efficient in an architecture called the Restricted Boltzmann Machine or RBM, which does not allow intralayer connections.

Therefore, when the visible units are known, the activity of the hidden units can be updated in one step, and no Gibbs sampling is required. This greatly speeds up the learning process.

$$P(h_j = 1|v) = \sigma\left(\sum_i w_{ij} v_i + b_j^h\right) \tag{21}$$

$$P(v_i = 1|h) = \sigma\left(\sum_j w_{ij} h_j + b_i^v\right) \tag{22}$$

Algorithm 14.2 Training an RBM

repeat
  Permute the order of data points
  for all mini-batches **do**
    {Sample from data distribution (positive phase)}
    for all data points i in mini-batch **do** {data distribution}
      Initialize visible units to x^i (and do not update visible units)
      Update probabilities of hidden units according to Eq. 21 and sample activities
      Save state as s_{data}
    end for
    {Sample from model distribution (negative phase)}
    Initialize visible and hidden units to random values $\sim B(0.5)$
    repeat {burn-in phase}
      Update probabilities of hidden units according to Eq. 21 and sample activities
      Update probabilities of visible units according to Eq. 22 and sample activities
    until thermal equilibrium is reached
    repeat {sampling phase}
      Update probabilities of hidden units according to Eq. 21 and sample activities
      Update probabilities of visible units according to Eq. 22 and sample activities
      Save state as s_{model}
    until enough samples have been collected
    Calculate $\langle s_i \rangle_{data}$ and $\langle s_i s_j \rangle_{data}$ for the mini-batch
    Calculate $\langle s_i \rangle_{model}$ and $\langle s_i s_j \rangle_{model}$ for the mini-batch
    Update weights and biases
  end for{mini-batches}
until predetermined number of epochs is reached

14.4.1 Contrastive divergence

The difference between the positive and negative phases, is simply the number of iterations through the network. With every iteration $\langle s_i s_j \rangle_{model}$ becomes more and more different from $\langle s_i s_j \rangle_{data}$. This is called *contrastive divergence*. We could speed up Alg. 14.2 further by giving up on thermal equilibrium before we sample from the model distribution. So, in the contrastive divergence algorithm (Alg. 14.3), a fixed number of iterations k are used in the negative phase. Particularly in the beginning of training, the energy function is not correct and the gradient is very inexact. So, little

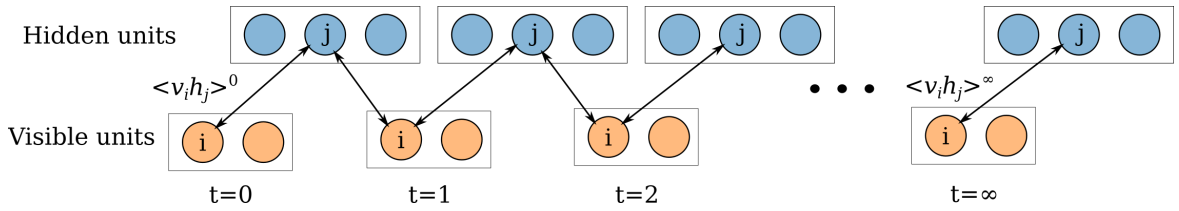

Figure 3: Contrastive divergence.

is gained by spending a lot of time to reach thermal equilibrium, which is needed to calculate the gradient precisely. Therefore, in the beginning of training, only one step of contrastive divergence, i.e., $k = 1$, is good enough to compute a gradient that moves the weights towards a better solution. As learning progresses and gradient descent approaches a good solution, more steps of contrastive divergence are needed to compute the gradient more exactly. There are several different schedules for increasing k with the number of gradient descent steps.

Algorithm 14.3 Constrastive divergence CD_k for RBM

repeat
  Permute the order of data points
  for all mini-batches **do**
    for all data points i in mini-batch **do**
      {Sample from data distribution (positive phase)}
      Initialize visible units to x^i (and do not update visible units)
      Update probabilities of hidden units according to Eq. 21 and sample activities
      Save state as s_{data}
      for k times **do**
        Update probabilities of visible units according to Eq. 22 and sample activities
        Update probabilities of hidden units according to Eq. 21 and sample activities
      end for
      Save state as s_{model}
    end for
    Calculate $\langle s_i \rangle_{data}$ and $\langle s_i s_j \rangle_{data}$ for the mini-batch
    Calculate $\langle s_i \rangle_{model}$ and $\langle s_i s_j \rangle_{model}$ for the mini-batch
    Update weights and biases
  end for{mini-batches}
until predetermined number of epochs is reached

Applications

- Pre-training a ffw network (unsupervised feature learning) Hinton (2006).

Figure 4: Reconstructing images of hand-written digits. 1. row: original image. 2. row: reconstruction by RBM. 3. row: reconstruction by logistic PCA. 4. row: reconstruction by regular PCA. Source: Hinton (2006), Fig. 2B.

- Unsupervised learning of feature representations when little labelled data is available.

Additional material:

- "A Practical Guide to Training Restricted Boltzmann Machines" by Geoffrey Hinton

References

Hinton, G. E. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, *313*(5786), 504–507.