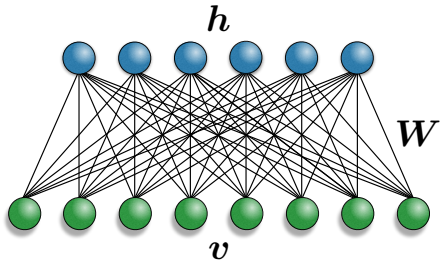
Tutorial 1:
The Restricted Boltzmann Machine

November 13th, 2019



In this tutorial, we will develop the core functionalities of the restricted Boltzmann machine (RBM). As seen in the lectures, the RBM contains a set of visible units $\boldsymbol{v}$ and a set of hidden units $\boldsymbol{h}$, with $v_j, h_i \in \{0, 1\}$. Given the set of parameters $\boldsymbol{\lambda} = \{\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{c}\}$, the RBM defines the following probability distribution over the full graph

$$p_{\boldsymbol{\lambda}}(\boldsymbol{v}, \boldsymbol{h}) = Z_{\boldsymbol{\lambda}}^{-1} e^{-E_{\boldsymbol{\lambda}}(\boldsymbol{v}, \boldsymbol{h})} \,, \tag{1}$$

where the energy function is given by

$$E_{\boldsymbol{\lambda}}(\boldsymbol{v}, \boldsymbol{h}) = -\sum_{j=1}^{N} \sum_{i=1}^{n_h} W_{ij} h_i v_j - \sum_{j=1}^{N} b_j v_j - \sum_{i=1}^{n_h} c_i h_i \,, \tag{2}$$

and $Z_{\boldsymbol{\lambda}} = \sum_{\boldsymbol{v}, \boldsymbol{h}} e^{-E_{\boldsymbol{\lambda}}(\boldsymbol{v}, \boldsymbol{h})}$ is the partition function. The probability distribution over the visible layer is obtained by integrating out the hidden units:

$$p_{\boldsymbol{\lambda}}(\boldsymbol{v}) = \sum_{\boldsymbol{h}} p_{\boldsymbol{\lambda}}(\boldsymbol{v}, \boldsymbol{h}) = Z_{\boldsymbol{\lambda}}^{-1} e^{\,\mathcal{E}_{\boldsymbol{\lambda}}(\boldsymbol{v})} \qquad\qquad \mathcal{E}_{\boldsymbol{\lambda}}(\boldsymbol{v}) = \sum_{j=1}^{N} b_j v_j + \sum_{i=1}^{n_h} \log\left(1 + e^{\,\sum_j W_{ij} v_j + c_i}\right) \,. \tag{3}$$

Given a dataset $\mathcal{D} = \{\boldsymbol{v}_k\}$ of visible configurations, the cost function for unsupervised learning is the average negative log-likelihood

$$\mathcal{C}_{\boldsymbol{\lambda}} = -\sum_{\boldsymbol{v} \in \mathcal{D}} \log p_{\boldsymbol{\lambda}}(\boldsymbol{v}) = \sum_{\boldsymbol{v} \in \mathcal{D}} \mathcal{E}_{\boldsymbol{\lambda}}(\boldsymbol{v}) - \log Z_{\boldsymbol{\lambda}} \,. \tag{4}$$

The gradient with respect to $\boldsymbol{\lambda}$ is given by

$$\begin{aligned} \nabla_{\boldsymbol{\lambda}} \mathcal{C}_{\boldsymbol{\lambda}} &= -\sum_{\boldsymbol{v} \in \mathcal{D}} \nabla_{\boldsymbol{\lambda}} \mathcal{E}_{\boldsymbol{\lambda}}(\boldsymbol{v}) - Z_{\boldsymbol{\lambda}}^{-1} \sum_{\boldsymbol{v}} \nabla_{\boldsymbol{\lambda}} p_{\boldsymbol{\lambda}}(\boldsymbol{v}) \\ &= -\langle \nabla_{\boldsymbol{\lambda}} \mathcal{E}_{\boldsymbol{\lambda}}(\boldsymbol{v}) \rangle_{\mathcal{D}} + \langle \nabla_{\boldsymbol{\lambda}} \mathcal{E}_{\boldsymbol{\lambda}}(\boldsymbol{v}) \rangle_{p_{\boldsymbol{\lambda}}(\boldsymbol{v})} \end{aligned} \tag{5}$$

a) The gradient of the function $\mathcal{E}_{\boldsymbol{\lambda}}(\boldsymbol{v})$ with respect to the different parameters are:

$$\frac{\partial \mathcal{E}_{\boldsymbol{\lambda}}(\boldsymbol{v})}{\partial W_{ij}} = \frac{v_j}{1 + e^{-\sum_j W_{ij} v_j + c_i}} \tag{6}$$

$$\frac{\partial \mathcal{E}_{\boldsymbol{\lambda}}(\boldsymbol{v})}{\partial b_j} = v_j \tag{7}$$

$$\frac{\partial \mathcal{E}_{\boldsymbol{\lambda}}(\boldsymbol{v})}{\partial c_i} = \frac{1}{1 + e^{-\sum_j W_{ij} v_j - c_i}} \tag{8}$$

Complete the function `DerLog(visible)` to compute these derivatives over a batch of data (`visible`). The output of the function should be a list of three arrays corresponding to gradients with respect to the weights and biases. For example, `der_W` (see code) should be computed as

$$\texttt{derW}_{ij} = \frac{1}{M} \sum_{k=1}^{M} \frac{\partial \mathcal{E}_{\boldsymbol{\lambda}}(\boldsymbol{v}_k)}{\partial W_{ij}} \tag{9}$$

where $M =$ `visible.shape[0]` is the number of visible configurations passed as input.

After you implemented the function, you can test that it operates correctly by running the following command: `python test.py`. This scripts runs a numerical test, comparing the results of your function with derivatives computed using finite differences.

b) Implement the function `GradientUpdate` to estimate the gradients in Eq. (5). The first term (called positive phase)

$$\langle \nabla_{\boldsymbol{\lambda}} \mathcal{E}_{\boldsymbol{\lambda}}(\boldsymbol{v}) \rangle_{\mathcal{D}} = \frac{1}{M} \sum_{k=1}^{M} \nabla_{\boldsymbol{\lambda}} \mathcal{E}_{\boldsymbol{\lambda}}(\boldsymbol{v}^{(k)}) \tag{10}$$

computes the average of the gradients $\nabla_{\boldsymbol{\lambda}} \mathcal{E}_{\boldsymbol{\lambda}}(\boldsymbol{v})$ over a batch of M training data points. Within the code, this data is passed as input to the function (i.e. `batch`). The second term (called negative phase)

$$\langle \nabla_{\boldsymbol{\lambda}} \mathcal{E}_{\boldsymbol{\lambda}}(\boldsymbol{v}) \rangle_{p_{\boldsymbol{\lambda}}} = \frac{1}{n_{mc}} \sum_{k=1}^{n_{mc}} \nabla_{\boldsymbol{\lambda}} \mathcal{E}_{\boldsymbol{\lambda}}(\boldsymbol{v}^{(k)}) \tag{11}$$

is evaluated on samples $\{\boldsymbol{v}^{(k)}\}$ drawn from the RBM distribution $p_{\boldsymbol{\lambda}}(\boldsymbol{v})$ using Monte Carlo sampling. The Monte Carlo step has already been implemented (`BlockGibbsSampling`), and returns a collection of n_{mc} visible configurations (`samples`). Once you compute the gradients on this data, the total gradients $\nabla_{\boldsymbol{\lambda}} \mathcal{C}_{\boldsymbol{\lambda}}$ are simply obtained using Eq. (5). Finally, the parameters $\boldsymbol{\lambda}$ need to be updated accordingly. Use vanilla gradient descent:

$$\boldsymbol{\lambda} \to \boldsymbol{\lambda} - \eta \, \nabla_{\boldsymbol{\lambda}} \mathcal{C}_{\boldsymbol{\lambda}} \tag{12}$$

where $\eta =$ `learning_rate` is the step-size of the update.

c) Once the gradients updates are implemented, the RBM can be used to run unsupervised learning on some simple reference data. Try running the command: `python tutorial1.py`. At each training iteration, the cost function is computed and printed on screen. Try adding the following command line arguments and see the effect on the cost:

- `nh`: number of hidden units
- `lr`: learning rate

You can add arguments simply as: `python tutorial1.py -ARG1 value -ARG2 value`. For example:

```
python tutorial1.py -nh 4 -lr 0.05
```