Restricted Boltzmann Machines

Theory and Implementation

Vagram Airiian
Software Engineer
LIT JINR
27.11.2020

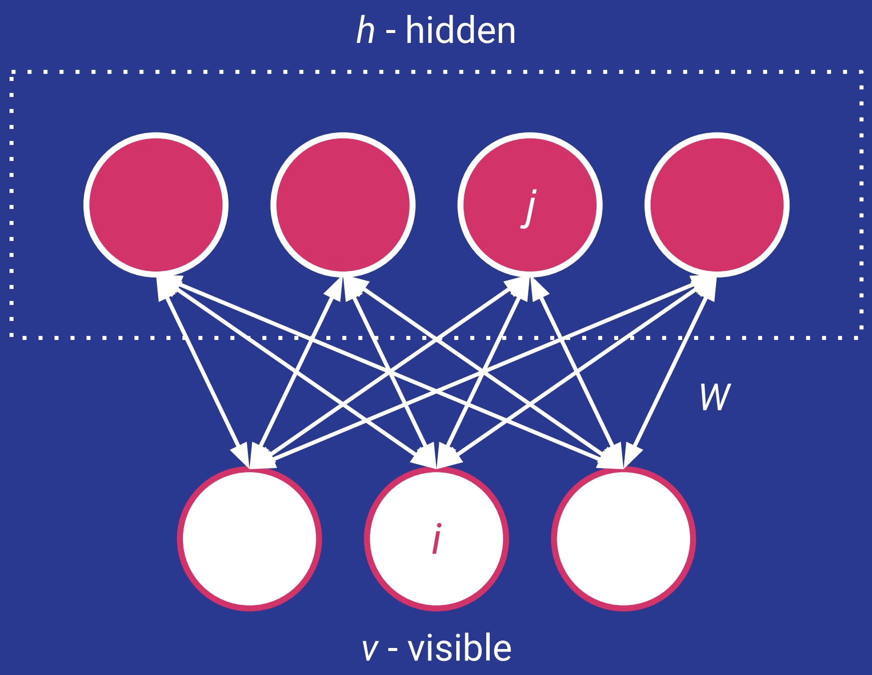
Theory

Restricted Boltzmann Machines

- An RBM defines a probability distribution over binary-valued patterns.
- They're undirected and don't have an output layer.
- All the hidden and visible nodes are all connected with each other.
- Restricted Boltzmann machines can generate data (a generative model).

RBM Model

- Not fully observable.
- One layer of hidden units (more is possible).
- No connections between hidden or visible units.
- A complete bipartite graph (biclique).



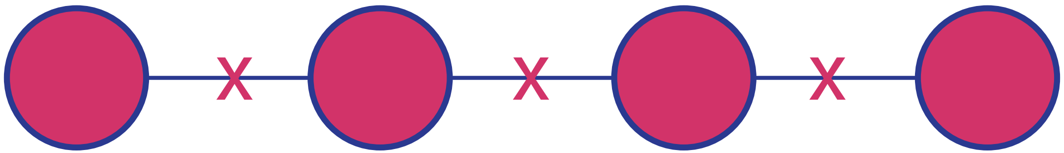
Gibbs Sampling

$$Pr(x_i = 1 \mid x_{-i}) = \sigma\,(\Sigma_{j \neq i}\, w_{ij} x_j + b_i)$$

if t -> ∞, the configurations will be distributed approximately according to the model distribution.

Conditional Independence

- Provides an unbiased sample from the posterior distribution when given a data-vector.
- Allows easy vectorisation of computations.



First Step

$\mathbf{v}^{(0)}$ - initial (input) data

$$h^{(0)} = \mathbb{E}[\mathbf{h} \mid \mathbf{v}] = \sigma\left(\mathbf{W}\mathbf{v}^{(0)} + \mathbf{b}_{\mathbf{h}}\right)$$

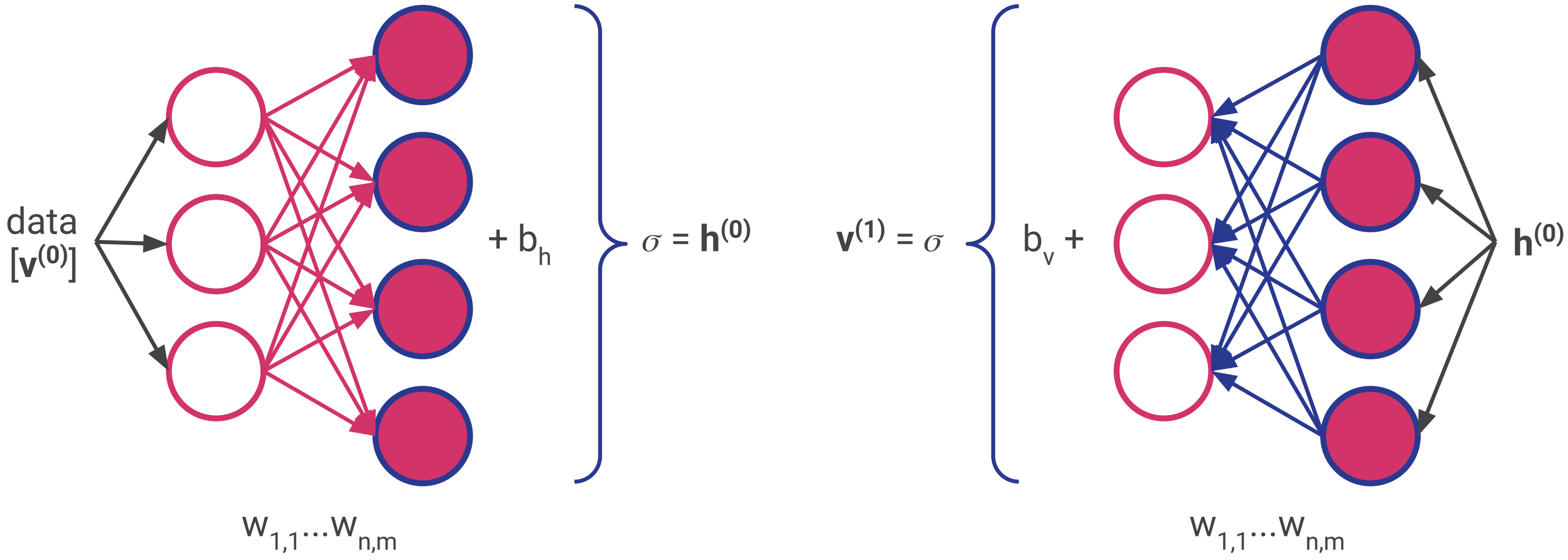
$$v^{(1)} = \mathbb{E}[\mathbf{v} \mid \mathbf{h}] = \sigma\left(\mathbf{W}^{\mathbf{T}}\mathbf{h}^{(0)} + \mathbf{b}_{\mathbf{v}}\right)$$

where

$v^{(1)}$ - reconstructed data

$$\sigma(x) = (1 + e^{-x})^{-1}$$

First Step



Derivation

A joint configuration energy:

$$E(\mathbf{v}, \mathbf{h}) = -\Sigma a_i v_i - \Sigma b_i h_i - \Sigma v_i h_j w_{ij}$$

The network assigns a probability to every pair via:

$p(\mathbf{v}, \mathbf{h}) = Z^{-1} e^{-E(\mathbf{v}, \mathbf{h})}$, $p(\mathbf{v}) = Z^{-1} \Sigma_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}$ - for a visible vector, where $Z = \Sigma_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}$

Then, the derivative of the log probability:

$$\partial \log p(\mathbf{v}) / \partial w_{ij} = \mathbb{E}_{\text{data}}[v_i h_j] - \mathbb{E}_{\text{model}}[v_i h_j]$$

and the learning rule:

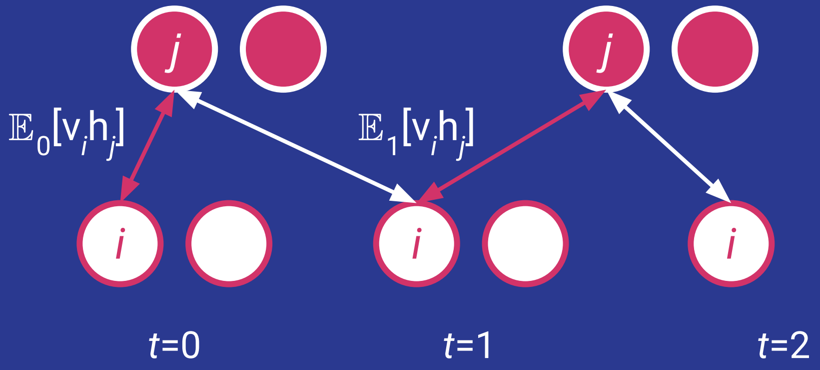
$\Delta w_{ij} = \varepsilon(\mathbb{E}_{\text{data}}[v_i h_j] - \mathbb{E}_{\text{model}}[v_i h_j])$, where ε - learning rate

Contrastive Divergence

- Start with a training vector on the visible units.
- Update all the hidden units in parallel.
- Update the all the visible units in parallel to get a "reconstruction".
- Update the hidden units again.
- Repeat *k* times.

$$\Delta w_{ij} = \varepsilon(\mathbb{E}_{data}[v_i h_j] - \mathbb{E}_{reconstruction}[v_i h_j])$$

(because *t* is finite)

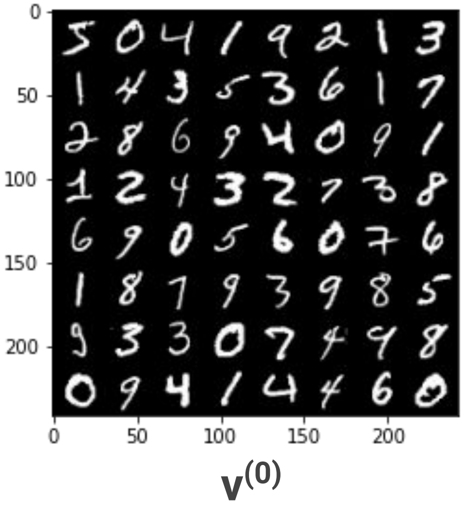


$$\mathbb{E}_{data}[v_i h_j] = \mathbb{E}_0[v_i h_j]$$

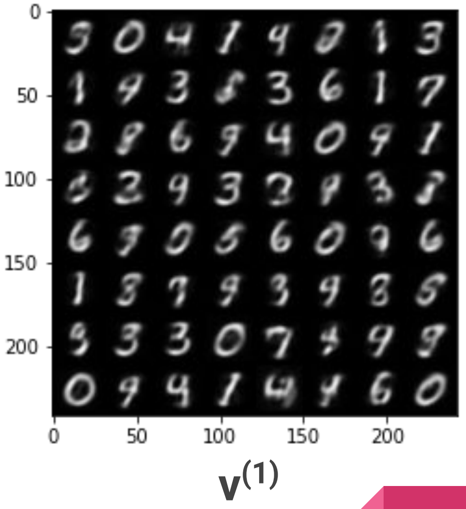
$$\mathbb{E}_{reconstruction}[v_i h_j] = \mathbb{E}_1[v_i h_j]$$

MNIST Example

Training Sample

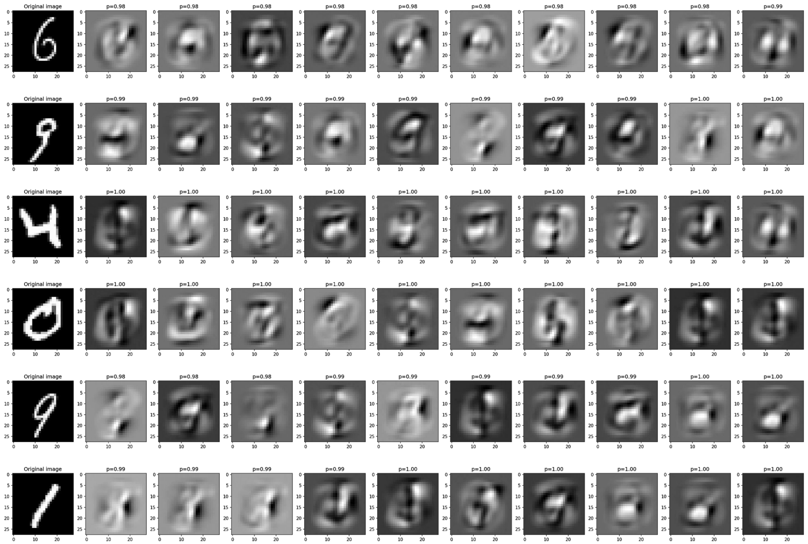


Generated Sample, k=2



Sample Features

Samples of the learned features



Netflix Challenge

You are given most of the ratings that half a million Users gave to 18 000 movies on a scale from 1 to 5 (stars).

- Each user only rates a small fraction of the movies.
- You have to predict the ratings users gave to the U4 held out movies.

Users	Movies (18 000)					
	M1	M2	M3	M4	M5	M6
U1				3		
U2	5		1			
U3		3	5			
U4	4		?			5
U5			4			
U6				2		

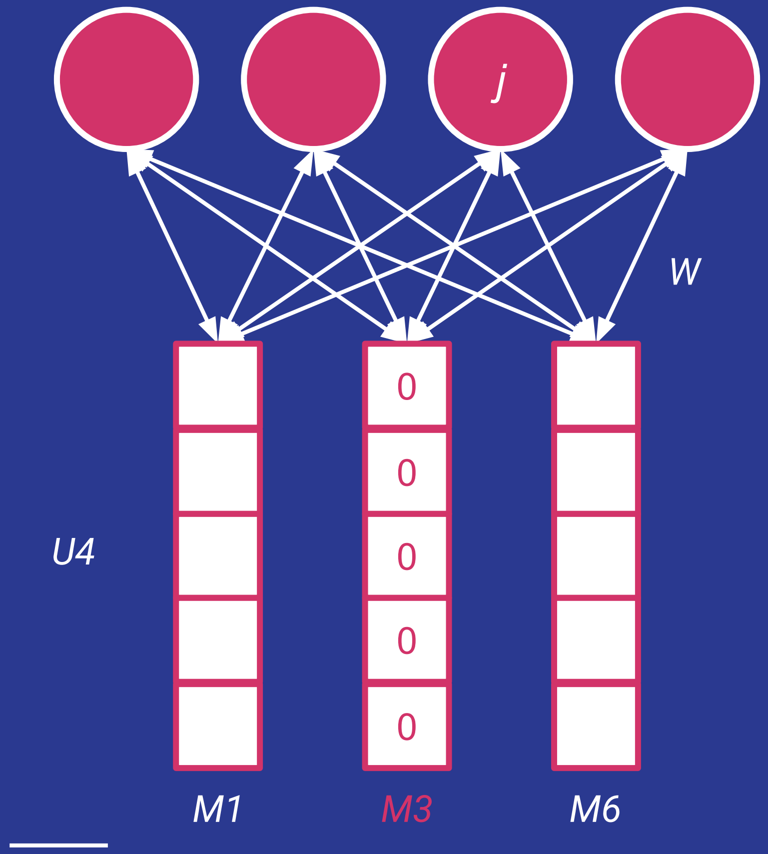
RBM Solution

Treat each user as a training case:

- A user is a vector of movie ratings.
- There is one visible unit per movie and its a 5-way softmax.
- The contrastive divergence learning rule for a softmax is the same as for a binary unit.
- There are ~100 hidden units.

One of the visible values is unknown:

- It needs to be filled in by the model.
- Softmax will yield the most probable rating (rating = argmax(**M3**))



Training

- For each user, use an RBM that only has visible units for the rated movies (skip unrated - only for training!).
- Instead of one RBM for all users, we have a different RBM for every user.
  - All RBMs use the same hidden units.
  - The weights from each hidden unit to each movie are shared by all the users who rated that movie.
- Each user-specific RBM only gets one training case!
  - It works due to the weight-sharing.

Implementation

Initial Values

```
self.weights = torch.randn(num_v, num_h) *0.01
self.v_bias = torch.ones(num_v) * 0.5
self.h_bias = torch.zeros(num_h)
self.momentum_coefficient =0.5
self.k = 2
self.weight_decay = 1e-4
self.learning_rate = 1e-3
```

$\mathbf{W} \sim N(0, 0.01)$

$\mathbf{b_v} = 0.5 \text{ or } \log[p_i/(1-p_i)]$

$\mathbf{b_h} = 0$

$\alpha = 0.5$

$k \in [1, \infty]$

$wd = 0.0001$

$\varepsilon = 0.001$

Hidden Vector

```
def sample_hidden(self, visible):
    activations = torch.matmul(visible,
                               self.weights) + self.bias_h
    p = torch.sigmoid(activations)
    return p
```

$$h^{(0)} = \mathbb{E}[\mathbf{h} \mid \mathbf{v}] = \sigma\left(\mathbf{W}\mathbf{v}^{(0)} + \mathbf{b}_h\right)$$

Visible Vector

```
def sample_visible(self, hidden):
    activations = torch.matmul(hidden,
                               self.weights.t()) + self.bias_v
    p = torch.sigmoid(activations)
    return p
```

$$\mathrm{v}^{(1)} = \mathbb{E}[\mathbf{v} \mid \mathbf{h}] = \sigma\left(\mathbf{W}^{\mathsf{T}}\mathbf{h}^{(0)} + \mathbf{b}_{\mathrm{v}}\right)$$

Binarization

```
def binarize(self, samples):
    p = torch.rand(self.num_h).to(self.device)
    activated_units = (samples >= p).float()
    return activated_units
```

$p_i \sim U(0, 1)$

unit = | 1, if s >= p
| 0, else

RBM Pass

```
def forward(self, input_data, train=True):
    # Positive phase
    pos_hidden_probabilities = self.sample_hidden(input_data)
    hidden_activations = self.binarize(pos_hidden_probabilities)
    positive_associations = torch.matmul(input_data.t(), hidden_activations)

    # Negative phase
    for step in range(self.k):
        # Gibbs sampling
        neg_visible_probabilities = self.sample_visible(hidden_activations)
        neg_hidden_probabilities = self.sample_hidden(neg_visible_probabilities)
        hidden_activations = self.binarize(neg_hidden_probabilities)
    negative_associations = torch.matmul(neg_visible_probabilities.t(), hidden_activations)
```

Parameters Update

```
if train:
    # Update momentum
    self.weights_momentum *= self.momentum_coefficient
    self.weights_momentum += (positive_associations - negative_associations) * self.lr

    self.v_bias_momentum *= self.momentum_coefficient
    self.v_bias_momentum += torch.sum(input_data - neg_visible_probabilities, dim=0) * self.lr

    self.h_bias_momentum *= self.momentum_coefficient
    self.h_bias_momentum += torch.sum(pos_hidden_probabilities - neg_hidden_probabilities, dim=0) * self.lr

    # Update weights and biases
    batch_size = input_data.size(0)
    self.weights += self.weights_momentum / batch_size
    self.v_bias += self.v_bias_momentum / batch_size
    self.h_bias += self.h_bias_momentum / batch_size

    # L2 weight decay
    self.weights -= self.weights * self.weight_decay
```

Contrastive Divergence

```
def contrastive_divergence(self, input_data):
    # Do k sampling steps and updates
    input_data, negative_visible_probabilities =self.forward(input_data)

    # Compute a reconstruction error
    error = F.mse_loss(input_data, neg_visible_probabilities)

    return error
```

Training

```
def train(model, train_loader, input_size, n_epochs=n_epochs):
    for epoch in range(n_epochs):
        epoch_error = 0.0
        for batch, _ in train_loader:          # batch size = 64
            batch = batch.view(-1, input_size)  # flatten input data
            batch = batch.to(device)
            batch_error = model.contrastive_divergence(batch)
            epoch_error += batch_error

        print(f'Epoch: {epoch + 1} | Error: {epoch_error:.4f}')
    return model
```

Inference

```
images = next(iter(train_loader))[0]
v, v_rec = model.forward(images.to(device).view(1, 784), train=False) # Flatten input data
```

Done!

Extra Reading

- Luis Serrano. Restricted Boltzmann Machines - A friendly introduction. https://www.youtube.com/watch?v=Fkw0_aAtwIw
- Geoffrey Hinton. A Practical Guide to Training Restricted Boltzmann Machines. https://www.cs.toronto.edu/~hinton/absps/guideTR.pdf
- Asja Fischer, Christian Igel. An Introduction to Restricted Boltzmann Machines. https://link.springer.com/chapter/10.1007/978-3-642-33275-3_2

Thank you for attention!