Energy Based Model: Restricted Boltzmann Machine

Tutorial by Sandeep Kumar 23095447

Theory

We normally use classification or regression in our models. Energy-based models take a different approach, it uses **density estimation.** Consider a dataset with large amount of data-points, the goal would be find the **probability distribution** over the whole database. It means all the data points will be converted to probability and sum of all the datapoints will be 1.

The probability distributions require atleast 2 minimal properties:
1) P(x) needs to assign any possible input value as a non negative value.
2) Probability density must integrate to 1 over all possible inputs.

Energy based models (EBM) can be of use here. The idea is that an EBM can turn any function that predicts value larger than zero, into a probability distribution by dividing by its volume. The volume is all the possible score that a NN or DL generates.

Lets take a simple neural network Eɵ(x), where ɵ is the model parameters and x is the input, the output of the NN will be a simple scalar value between -∞ to ∞. We can us the basic probability theory to normalize the score of all possible inputs.

$$q_ɵ = \frac{\exp(-E_ɵ(x))}{Z_ɵ}$$

Zɵ is the volume, which is
1) $\int exp\,(-E_ɵ(\mathrm{x})\mathrm{dx})$ when x is continous
2) $\sum exp(-E_ɵ(\mathrm{x}))$ when x is discrete

The exponent will ensure that NN output is always greater than 1 no matter the input. The NN function E is negative because in energy based model lower energy means high probability and high energy means low probability.

It is important to understand that energy is like a cost factor, is associated with "likeliness" or "natural" factor. The more likely or natural something is, the low energy it consumes, thus higher probability. Similarly, when an object is not in its likely state, it consumes energy, so high energy means low likeliness, resulting in low probability.

Benefits of EBM: Flexibility

1) Simple model: Could be based on few features of data like linear function or simple quadratic function.

2) Complex model : Energy function can be any neural network or deep network. It can represent sophisticated data and can be highly non linear.

3) It is used in image processing, natural language processing and reinforcement learning.

Constraints

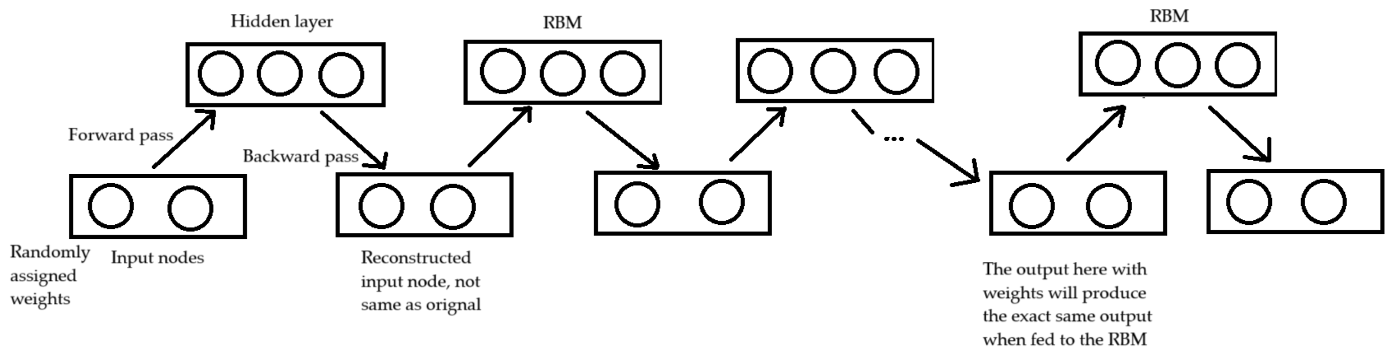
1) Normalize constraint: The value of probability distribution strictly should be between 0 to 1 and its sum as 1. The NN must be normalized properly.
2) Computational constraint: Since we can choose extremely complicated model, the z also gets expensive and difficult, specifically for large database.
3) Differentiablity constraint: Energy function must be differentiable with respect to parameters so optimizer like gradient descent can be used but energy function must be strongly structured.

Contrastive Divergence (Training algorithm)

When we train a model in generative modeling it uses a **maximum likelihood estimation**. Maximizing the likelihood of the examples in training set. Exact likelihood cannot be determined due to "unknown normalization count", Zɵ aka volume.

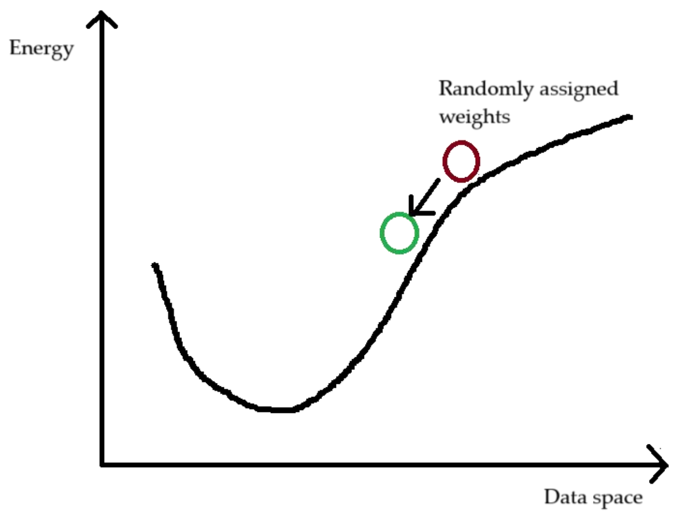
One thing to keep in mind is that we cannot just maximize the energy function as we aim to work on a model that generates, more likely data with low energy and low likely data with high enegry.

Using a Contrastive Divergence ensures that energy function learns meaning-full differences or compare the likelihood of points to create a stable objective.



Hidden layer
RBM
RBM
Forward pass
Backward pass
...
Randomly assigned weights
Input nodes
Reconstructed input node, not same as orignal
The output here with weights will produce the exact same output when fed to the RBM

The hidden layer is defined from the input data, the hidden layer and the input layer are fully connected but nodes are not connected to each other. This gives an output that is capable of generating similar data but not the original. Through large amount of iterations we try minimize the energy state, via wights such that its as close to original as it can be.



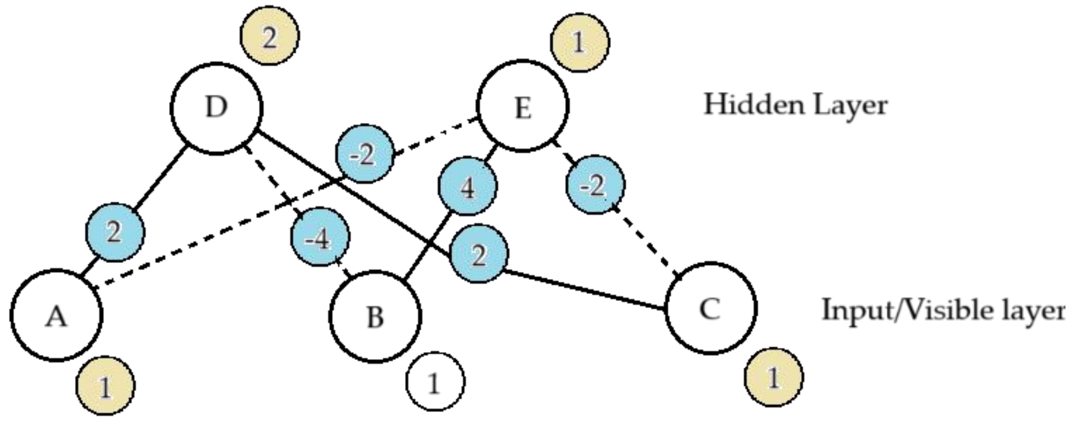
After the first pass the ball moves inside. The ball tries to end up in the lowest energy state possible. Although it is similar to gradient descent where we have a curve and we try to find a global minima, here we can control the curve by controlling the weights and change the shape of the graph as per our need. We can push down the curve when energy is low and probability is high and vise versa when probability is low.

Restricted Boltzmann Machine

Restricted Boltzmann machine (RBM) is an energy based model technique which is defined from the Boltzmann Machine, as Boltzmann Machine is computationally expensive and takes a lot of time to process, the restricted version is fast and effective.

A good way to learn RBM is through examples. Lets start with,
1) Weights
Lets take a dataset (sample) were we have 3 columns. We can try to define the relation of these attributes by assuming a hidden layer of 2 nodes as shown in the diagram.

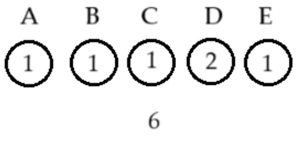
A	B	C
1	0	1
0	1	0
1	0	1
1	0	1
0	1	0
1	0	1

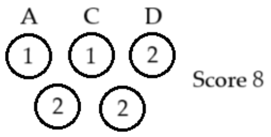


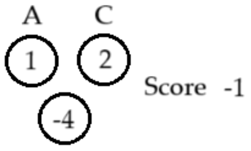
The way the network is defined here is A, B, and C are 3 people and D and F (hidden layer/imaginary) are fruits (say Apple and Mango) that they love. From the input data we can see that when A and C occurs B is absent and vice versa. We assume that there is no correlation between the labels while we assign the weights, A likes Apple so weight of 2 is assigned, B likes Mango very much so a weight of 4 is assigned, C is same as A and hates mango so a negative weight is assigned (Blue).

2) Scores
Every node has its own score (Yellow), D has a score of 2 because 2 people like apple. The total score can be calculated by summing the weights and scores of all the nodes and edges in a scenario. The total number of scenario is 2 ^ Number of nodes, here 32.

After calculating the scores of all scenario we get
Highest score
BE : 7
ACD : 8
Lowest score
BD : -1
ACE : -1

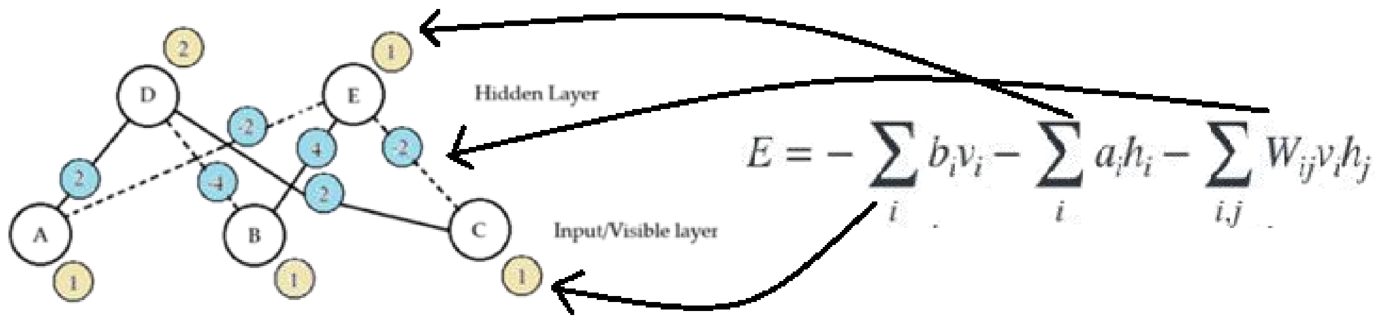






Which make sense according to the “small” dataset sample that we have.

Formula



$$E = -\sum_{i} b_i v_i - \sum_{i} a_i h_i - \sum_{i,j} W_{ij} v_i h_j$$

Energy is the negative of the sum of the scores of weights times the state of the visible layer plus the same sum for the hidden layer plus the sum of the score for visible layer times hidden layer times weights of the edges. Why negative? Energy is -Score (Explained in theory)

3) Probablities
We can now turn these scores into probabilities. Low score for low probabilities and high score for high probabilities.

To understand this lets take random scores (not from data)

score	Probability
3	1/2
2	1/3
1	1/6
6	1

This works, low score for low probability and high score for high probability. But this wont work if the values were 1, 0, -1, we cannot divide by zero which is a problem.
To solve this (mentioned in theory earlier) we can get **exponents** of the scores so the value is always greater than 0.

We take all 32 scenario of out data and get exponent for each score, then get the sum of all score. Divide them by summed value to get the probability.

Scenario	Score	Exp(score)	Probability
BD	-2	0.14	0 (way to low but not 0)
BE	7	1096.63	0.17
ACD	8	29808.96	0.45
ACE	-1	0.37	0 (way to low but not 0)

We got low score for low probability and high score for high probability.

Formula

$$p(v, h) = \frac{e^{-E(v, h)}}{Z} \qquad Z = \sum_{v,h} e^{-E(v,h)}$$

4) Training

We need to understand what we want the RBM to do. There are 32 scenarios and we want all the known one to be high in probability. The known ones are B, AC, BD, BE, ACD, ACE, BDE, ACDE.

None	| | | | |	ABC	| | | | |
A	| | | | |	ABD	| | | | |
B	| | | | |	ABE	| | | | |
C	| | | | |	**ACD**	| | | | | | |
D	| | | | |	**ACE**	| | | | | | |
E	| | | | |	ADE	| | | | |
AB	| | | | |	BCD	| | | | |
AC	| | | | | | |	BCE	| | | | |
AD	| | | | |	BDE	| | | | |
AE	| | | | |	CDE	| | | | |
BC	| | | | |	ABCD	| | | | |
BD	| | | | |	ABCE	| | | | |
BE	| | | | |	ABDE	| | | | |
CD	| | | | |	**ACDE**	| | | | | | |
CE	| | | | |	BCDE	| | | | |
DE	| | | | |	ABCDE	| | | | |

This can be done using the principle of contrastive divergence. We take the first event in our sample dataset (right) where A and C show up but not B, we choose AC, ACD, ACE, ACDE and increase their probability a little bit (left), next we decrease the probability of all the other scenarios.

A	B	C
1	**0**	**1**
0	1	0
1	**0**	**1**
1	**0**	**1**
0	1	0
1	**0**	**1**

In the second row of the data, the B showed up so we increase the probability of B, BD, BE, BDE, and then again decrease the probability of all other scenarios.

A	B	C
1	0	1
0	**1**	**0**
1	0	1
1	0	1
0	**1**	**0**
1	0	1

None	| | | |	ABC	| | | |
A	| | | |	ABD	| | | |
B	| | | | | |	ABE	| | | |
C	| | | |	**ACD**	| | | | | | |
D	| | | |	**ACE**	| | | | | | |
E	| | | |	ADE	| | | |
AB	| | | |	BCD	| | | |
AC	| | | | | | |	BCE	| | | |
AD	| | | |	**BDE**	| | | | | |
AE	| | | |	CDE	| | | |
BC	| | | |	ABCD	| | | |
BD	| | | | | |	ABCE	| | | |
BE	| | | | | |	ABDE	| | | |
CD	| | | |	**ACDE**	| | | | | | |
CE	| | | |	BCDE	| | | |
DE	| | | |	ABCDE	| | | |

None	|	ABC	|
A	|	ABD	|
B	| | | | | |	ABE	|
C	|	**ACD**	| | | | | | |
D	|	**ACE**	| | | | | | |
E	|	ADE	|
AB	|	BCD	|
AC	| | | | | | |	BCE	|
AD	|	**BDE**	| | | | | |
AE	|	CDE	|
BC	|	ABCD	|
BD	| | | | | |	ABCE	|
BE	| | | | | |	ABDE	|
CD	|	**ACDE**	| | | | | | |
CE	|	BCDE	|
DE	|	ABCDE	|

Here we face a little issue, we have nodes in total, so 32 scenarios but, real life data is different, we will have around 100 hidden nodes and 200 input nodes, computation of 2^{300} is virtually impossible. To fix this we use sampling.

5) Gibbs Sampling

To understand Gibbs sampling lets consider a space with positive and negative values, the objective is to fill up the space with positive values but we cant look inside. The approach here would be to take a random value out and put in a positive value in. In case we got unlucky and pick a positive energy, its still in a no loss type of situation. This process when repeated for a long time will eventually remove 99% of the negative values.

We can apply this theory to the dataset. When we loop around the dataset we increase the probability of the BE, that counts as adding a positive value to the bag. Then lower the probability of a completely random scenario (very low chance that it will a positive value), moving to next data point ACD, same process is applied.

Doing this for a long time, we end up with something really good. We get ACD and BE with the highest probability and low probabilities on all the other scenarios including the hidden scenarios because from the start the the hidden layer is just an imaginary layer used to define the relation. The result from this, the RBM generate data that mimics the original dataset pretty well.

None	| | | | |	ABC	| | | | |
A	| | | | |	ABD	| | | | |
B	| | | | |	ABE	| | | | |
C	| | | | |	ACD	| | | | | | |
D	| | | | |	ACE	| | | | |
E	| | | | |	ADE	| | | | |
AB	| | | | |	BCD	| | | | |
AC	| | | | |	BCE	| | | | |
AD	| | | | |	BDE	| | | | |
AE	| | | | |	CDE	| | | | |
BC	| | | | |	ABCD	| | | | |
BD	| | | | |	ABCE	| | | | |
BE	| | | | | | |	ABDE	| | | | |
CD	| | | | |	ACDE	| | | | |
CE	| | | | |	BCDE	| | | | |
DE	| | | | |	ABCDE	| | | | |

After a long repeated process $\longrightarrow$

None	|	ABC	|
A	|	ABD	|
B	|	ABE	|
C	|	ACD	| | | | | | |
D	|	ACE	|
E	|	ADE	|
AB	|	BCD	|
AC	|	BCE	|
AD	|	BDE	|
AE	|	CDE	|
BC	|	ABCD	|
BD	|	ABCE	|
BE	| | | | | | |	ABDE	|
CD	|	ACDE	|
CE	|	BCDE	|
DE	|	ABCDE	|

Lastly when we mess with the weights of the model we pick a dataset scenario and check the relation of the hidden and input layer and assign weights to the nodes and edges, the weights that we start with is 0.1 which is a learning rate, a hyper-parameter. Gibbs sampling repeates the process by decreasing the weights until we find the model which mimics the dataset.

The practical example (although not perfect) is implemented below.

Get Libraries

```
import numpy as np
import torch
import torchvision
import torchvision.transforms as transforms
```

Loading the MINST dataset that contains the handwritten number 0 to 9.

```
batch_size = 64
transform = transforms.Compose([transforms.ToTensor()])

train_loader = torch.utils.data.DataLoader(
    torchvision.datasets.MNIST(root="./data", train=True, transform=transform, download=True),
    batch_size=batch_size, shuffle=True
)
```

Initialize the RBM Restricted Boltzmann parameter

```
# Initialize RBM parameters
visible_units = 784  # 28x28 pixels
hidden_units = 256
learning_rate = 0.005
k = 10  # Increased CD-k steps for better learning
```

Initialize the Weights and Bias

```
# Xavier (Glorot) Initialization
W = torch.randn(visible_units, hidden_units) * np.sqrt(2 / (visible_units + hidden_units))
b_v = torch.zeros(visible_units)
b_h = torch.zeros(hidden_units)
```

Make a loop that loops through 100 epochs converting 28 X 28 images to 784 dimensional vector

```
# Training loop
epochs = 100 # (update: 5)
for epoch in range(epochs):
    epoch_loss = 0
    for batch, (data, _) in enumerate(train_loader):
        data = data.to(device)  # Move data to the correct device
        v_0 = data.view(-1, visible_units)  # Flatten the image
        batch_size = v_0.shape[0]  # Get batch size
```

The forward pass

```
# Positive phase
h_prob_0 = sigmoid(torch.matmul(v_0, W) + b_h)   # P(h|v)
h_0 = (h_prob_0 > torch.rand_like(h_prob_0)).float()  # Sample h
```

Gibbs sampling

```
# Gibbs sampling (CD-k)
v_k = v_0.clone()
for _ in range(k):
    h_prob = sigmoid(torch.matmul(v_k, W) + b_h)
    h_k = (h_prob > torch.rand_like(h_prob)).float()
    v_prob = sigmoid(torch.matmul(h_k, W.T) + b_v)
    v_k = v_prob + 0.1 * torch.randn_like(v_prob) # Add nosie to smooth reconstruct
```

Backward pass

```
# Negative phase
h_prob_k = sigmoid(torch.matmul(v_k, W) + b_h)
```

Updating weights and biases using contrastive divergence

```
# Compute gradients
pos_grad = torch.matmul(v_0.T, h_prob_0)
neg_grad = torch.matmul(v_k.T, h_prob_k)

# Update weights and biases
W += learning_rate * (pos_grad - neg_grad) / batch_size
b_v += learning_rate * torch.mean(v_0 - v_k, dim = 0)
b_h += learning_rate * torch.mean(h_prob_0 - h_prob_k, dim = 0)
```

Compute loss and print the output

```
        # Compute loss (reconstruction error) (code for later use)
        #batch_loss = torch.mean((v_0 - v_k) ** 2)
        #epoch_loss += batch_loss.item()

        # Compute loss using Mean Absolute Error (L1 Loss)
        batch_loss = torch.nn.functional.l1_loss(v_k, v_0)
        epoch_loss += batch_loss.item()

    print(f"Epoch {epoch+1}/{epochs}, Loss: {epoch_loss / len(train_loader):.4f}")
```

Select 10 random images from the dataset

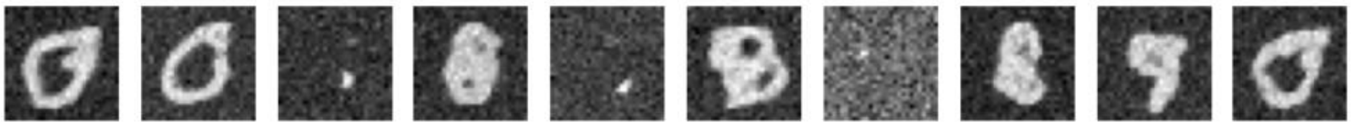
```
import matplotlib.pyplot as plt

# Number of samples & Gibbs steps
num_samples = 10
num_gibbs_steps = 500

# Start from real MNIST data instead of pure randomness
sample_data, _ = next(iter(train_loader))
v = sample_data[:num_samples].view(num_samples, visible_units).to(device)
```

Gibbs sampling for the output

```
# Perform Gibbs sampling
for _ in range(num_gibbs_steps):
    h_prob = sigmoid(torch.matmul(v, W) + b_h)
    h_sample = (h_prob > torch.rand_like(h_prob)).float()
    v_prob = sigmoid(torch.matmul(h_sample, W.T) + b_v)
    v = v_prob + 0.1 * torch.randn_like(v_prob)
```

And finally plot and compare generated image with the original image

```
# Plot generated images
fig, axes = plt.subplots(1, num_samples, figsize=(10, 2.5))
for i in range(num_samples):
    axes[i].imshow(v[i].view(28, 28).numpy(), cmap="gray")
    axes[i].axis("off")
plt.show()
```



```
import matplotlib.pyplot as plt

# Get a batch of training data
sample_data, _ = next(iter(train_loader))
sample_data = sample_data[:10]  # Select 10 images

# Plot the images
fig, axes = plt.subplots(1, 10, figsize=(10, 2))
for i in range(10):
    axes[i].imshow(sample_data[i].view(28, 28).numpy(), cmap="gray")
    axes[i].axis("off")
plt.show()
```

REFERENCES

1) EBM tutorial
Tutorial 8: Deep Energy-Based Generative Models — UvA DL Notebooks v1.2 documentation

2) Implicit generation and generalization of EBM
Implicit generation and generalization methods for energy-based models | OpenAI
[1903.08689] Implicit Generation and Generalization in Energy-Based Models

3) RBM tutorial
Generative model that won the 2024 Physics Nobel Prize - Restricted Boltzmann Machines (RBM)

END
THANKS FOR READING