Restricted Boltzmann Machine

Luyang Huang(14307110020)

I. Theory

Restricted Boltzmann Machine is a bipartite graph model in which variable can be divided into two groups, the input layer and the hidden layer. The hidden layer can take Boolean value (0 or 1). In the model, hidden values are independent to each other. To use gradient descent to calculate loss, we need to use Gibbs Sampling Method to sample in each step. The RBM is a kind of generative model and a unsupervised model. In training step, we don't need label to train.

i. Some tricks in implementing RBM

The weights are initialized to small random values chosen from a zero-mean Gaussian with a standard deviation of about 0.01. Initial hidden biases of 0 are usually fine. It is also possible to start the hidden units with quite large negative biases as a crude way of encouraging sparsity.

RBMs typically learn better models if more steps of alternating Gibbs sampling are used. The sampling step includes:

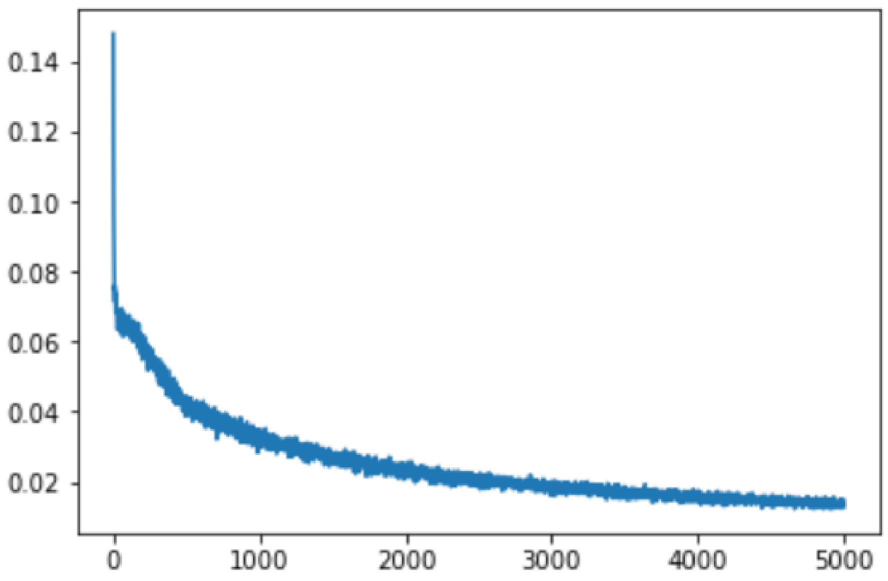
1. set visible state to training sample(x) and compute hidden state(h0) of data then we have binary units of hidden state computed. It is very important to make these hidden states binary, rather than using the probabilities themselves.
2. compute new visible state of reconstruction based on computed hidden state reconstruction. However, it is common to use the probability, instead of sampling a binary value. So this can be binary or probability (so I choose to not use sampled probability)
3. compute new hidden state of reconstruction based on computed visible reconstruction when hidden units are being driven by reconstructions, always use probabilities without sampling.

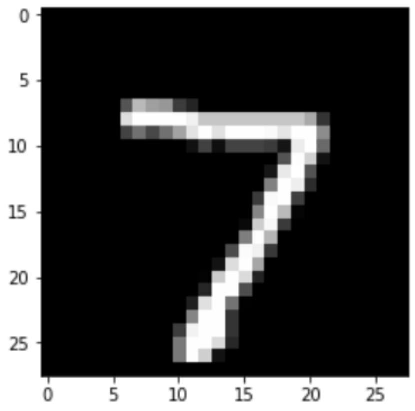
Usually, we can't trust traditional error function because it can't define the difference between input and the reconstructed visible layer very well. Sometimes, the error gets slightly bigger but the result comes better.

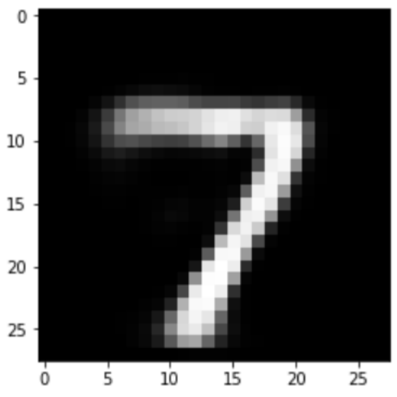
II. Experiment

I set batch size as 100. After about 3000 steps, the errors start to converge. I train for 5000 steps, using only about 1 minute. The difference between input and visible layer output is rather small

The results after decoding is shown below. We can see the results are rather good compared to initial one.


0.14
0.12
0.10
0.08
0.06
0.04
0.02
0
1000
2000
3000
4000
5000


0
5
10
15
20
25


0
5
10
15
20
25