Report

Big Data Physics

경희대학교

Report

Big Data Physics/
Restricted Boltzmann Machine (RBM)

제출일 2022. 12. 21

전공 물리학과

과목 빅데이터 물리학

학번 2017103274

담당교수 고정환 교수님

이름 최정훈

팀원 17 최정훈

19 김은서

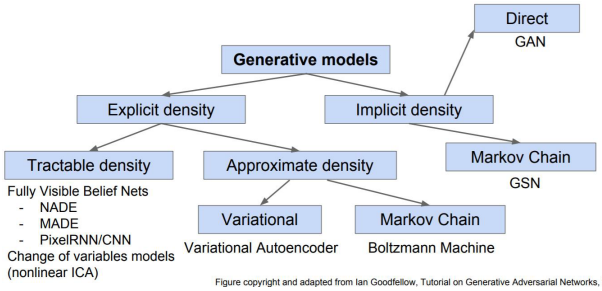
. Introduction 서론

AI 에 대한 연구는 기호 체계를 이용해 다양한 지식들을 논리적 집합으로 표현하는 것(기호 체계 접근법)을 시작으로, 네트워크의 급속한 발전을 통해 저렴한 비용으로 손쉽게 데이터를 사용할 수 있게 되면서, 데이터로부터 정보를 학습하는 지식을 만드는 신호 처리 접근법으로 발전해 나아갔다. 딥러닝 기술은 과거의 전통 통계적 기법들과 비교도 안될 정도로 많은 가중치를 이용해서 보다 복잡한 정보들까지도 학습시킬 수 있는 잠재력을 가진 기술로 정확한 계산을 전제로 하는 물리학과 비교했을 때, 상대적으로 낮은 정확도를 가지지만 훨씬 빠르고 복잡한 문제를 해결할 수 있는 새로운 방법론으로 주목 받고 있다. 딥러닝에 대한 관심이 인류 역사상 가장 높은 현재 TensorFlow 및 PyTorch 등 무료로 제공되는 다양한 라이브러리들을 사용해서 모델을 만드는 것은 더 이상 어려운 과정이 아니게 되었다. 다양한 프로젝트 주제에 대해 고심하던 우리 조는 주제를 선택하는데 있어서 평소에 경험해 보지 못했던 **새로운 모델**을 사용해 보자라는 합의점에 도달 하였으며, 그 기준을 토대로 주제를 선택하기 위한 노력을 하였다. 초기에 주제를 선택하는데 있어서 생성모델과 강화학습 둘 중 하나의 방법을 이용하는 모델을 구현해 보자는 의견이 등장하였고, 우리들은 깊은 고민 끝에 생성모델, 그 중에서 생성모델의 초기모델에 속하는 제한된 볼츠만 머신(Restricted Boltzmann Model; 이하 RBM)에 대해서 공부를 하기로 결정하였고, 이를 토대로 실제로 모델을 생성 및 학습 시켜 본 후, RBM 을 통해 생성한 데이터와 GAN 을 통해 생성한 데이터에 대하여서 비교해 보기로 하였다. 최근에 머신러닝과 딥러닝에 관심을 가지게 된 경우, RBM 에 대해서 많이 들어보지 못했을 것이다. 다른 유명한 모델들에 비교했을 때, 상대적으로 덜 알려진 이름부터 물리학과 친구들의 호기심을 자극하는 제한된 "볼츠만" 머신은 최근 사용되고 있는 다양한 모델들과 상당히 다른 형태로 동작을 하고 있기 때문에 학우들에게 소개(발표)하는 과정에서도 높은 흥미를 끌 수 있을 것이라고 생각했으며, 과거 데이터 전문가(ADP) 과정을 준비하면서 급하게 암기만 하고 넘어갔던 나에게 해당 모델의 작동원리를 알아보며 직접 구현해보는 좋은 경험이 될 것이라고 생각했기 때문에 이를 프로젝트 주제로 선정하게 되었다.

. About Restricted Boltzmann Machine 볼츠만 모델에 관하여

- Types and History of Generative Models 생성모델의 종류와 역사

보통 머신 러닝에 대한 책을 처음 읽기 시작하면, 가장 첫 번째 장에서는 앞으로 배운 머신 러닝을 구분하는 방법에 대해서 다루는 내용들이 주로 등장한다. $Deep\ Learning \subset Machine\ Learning \subset AI$ 와 같은 포함 관계를 본적이 있을 것이다. 일반적으로 머신러닝 방법들은 $Machine\ Learning\{\ Supervised\ Learning, Unsupervised\ Learning, Reinforcement\ Learning\ \}$ 와 같이 분류한다. 생성모델은 위의 분류법에 따라 지도적 생성모델과 비지도적 생성모델 모두 존재하며, 대부분은 비지도적 생성모델이기 때문에 Unsupervised Learning 의 범주에 넣기도 한다. 최근에 언론에서 많은 관심을 가지고 있는 가상인간이나 새로운 화풍의 이미지 생성, 새로운 음악 생성, 이미지 색채화 및 해상도 개선과 같이 다양한 산업에서 생성모델이 사용되고 있다. 혹은 생성하는데 비용이 많이 들고 많이 관측될 수 없는 현상에 대한 데이터의 경우, 생성모델을 통해서 새로운 데이터를 대량으로 생산할 수 있으며 이를 다시 학습에 사용하는 방법이 다양한 연구에서 사용되고 있다. 이처럼 재미있고 활용도 높은 생성모델 중 RBM 을 소개하기 앞서서 생성모델의 종류와 그들의 역사에 대해서 간략하게 이야기 해보고자 한다.



Ian Goodfellow, Tutorial on Generative Adversarial networks, 2017

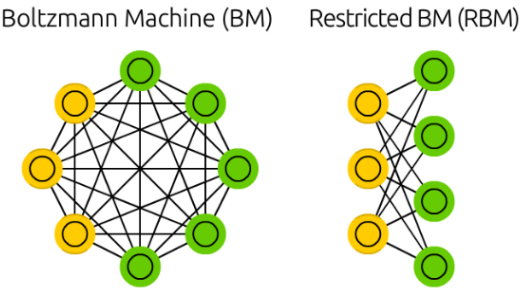
머신러닝 역사를 이야기 할 때, 이론적의 시작은 일반적으로 18 세기 중반 베이즈 정리(Bayes' theorem)의 토대가 등장한 시기를 이야기한다. 이 후 컴퓨터가 등장한 20 세기 중순부터 우리가 들어보았을 법한 많은 분류 및 회귀 모델들이 등장하였고, 생성모델이라는 범주는 1980 년에 처음으로 세상에 등장하게 된다. 생성 모델의 선구자로 불리며, GAN(Generative Adversarial Network)을 2014 년 논문으로 발표한 이안 굿펠로우(Ian Goodfellow)의 생성모델을 분류한 바에 따르면, Generative models 는 크게 Explicit density estimation 와 Implicit density estimation 범주로 나눈다. 생성 모델의 목적이 입력된 $p_{data}(x)$ 분포에 대해서 이와 유사한 $p_{model}(x)$ 분포를 얻는 것이라고 할 수 있다. $p_{model}(x)$ 분포를 얻는데 있어서 Explicit density estimation 는 $p_{data}(x)$ 분포를 직접적으로 구하는 방법으로 Tractable density 와 $p_{data}(x)$를 단순히 추정하는 Approximate density 방법으로 나누어 진다. Tractable density 에는 Fully Visible Belief Nets(PixelCNN, PixelRNN, NADE, MADE)등이 있으며, 학습 데이터의 분포를 학습하였기 때문에 뚜렷한 이미지를 얻을 수 있지만, 높은 연산량으로 속도가 느리다는 문제점이 있다. Approximate density 에는 $p_{data}(x)$ 분포의 잠재 코드 값(확률)을 추출하는 (Variational) Auto-Encoder 가 포함되며[1], 우리 조의 주제인 **RBM** 또한 Markov Chain 의 특성을 이용해 입력 $p_{data}(x)$의 분포를 근사하기 때문에 이의 범주에 포함된다. 위의 Explicit density estimation 를 이용한 방법들이 생성모델에서 가장 먼저 사용된 분야로 RBM 과 Auto-Encoder 는 1980 년대 초창기 생성모델로 등장했다. 현대에도 RBM 의 hidden layer 를 다층으로 구성한 심층 신뢰 신경망(Deep Belief Network)의 형태로 Auto-Encoder 는 Variation Auto-Encoder(2014)와 심층 인공신경망에 포함되어 사용되고 있다.

Implicit density estimation 을 사용하는 모델에서는 $p_{data}(x)$ 분포에 대해서 학습하는 과정이 없다. Implicit density 에는 생성기(Generator Network)와 판별기(Discriminator Network)들의 상반되는 목적을 서로 달성하기 위해 경쟁하는 방식으로 학습을 수행하는 GANs 이 있다. 생성기는 latent space z(랜덤 벡터)로부터 가짜 이미지를 Mapping 하는 과정이기에 $p_{data}(x)$의 분포가 직접적으로 입력되는 경우는 없기 때문에 이는 Implicit density estimation 에 포함되며, 이 후 2016 년 GAN 의 Fully-connected layer 구조를 Convolution Layer 로 치환한 DCGAN(Deep Convolutional GAN) 등장으로 생성 모델의 역사가 새롭게 시작되었다. DCGAN 은 GAN 보다 안정적인 학습이 가능하며, 이미지에서의 산술 연산(vector arithmetic)이 가능하다는 점에서 LSGAN, PGGAN, SRGAN, CycleGAN, StarGAN 등 다양한 GANs 들의 기본 구조로 사용되고 있다.

- Statistical Principles of RBM RBM 통계적 원리

위에서 RBM 은 Explicit density estimation 에서 Approximate density 범주에 포함되는 생성 모델임을 기억하면서 RBM 의 통계적 원리에 대해 알아보자. 이를 사람의 얼굴을 학습하는 모델에 대하여 쉽게 생각해 볼 수 있다. $p_{data}(x)$는 입력된 데이터의 분포를 나타내는 것으로 이를 완벽하게 학습하기 위해서는 모집단의 얼굴 부위별 모양에 대해서 완벽하게 확률밀도함수(PDF)를 알아야 한다. 이 후, 학습된 분포를 토대로 확률적으로 얼굴의 부위별 모양을 결정하게 되면, 우리는 새로운 사람의 얼굴 이미지를 생성할 수 있게 된다. 생성 모델에서 학습된 분포 $p_{model}(x)$를 토대로 결과물을 출력하는 과정을 Sampling 이라고 칭한다.

[1] 학습데이터에 대해서 AE의 경우 잠재 코드의 값으로 근사하며, VAE는 잠재 코드의 확률 분포(Gaussian Distribution)를 이용해 근사한다.



The Neural Network Zoo
https://www.asimovinstitute.org/neural-network-zoo/

위의 모델을 간략화 시킨 그림에서 초록색은 Hidden Cell 을 의미하며, 노란색은 Input Cell(RBM 에서는 Visible Cell)이며, 각 Cell 은 별개의 특징에 대한 상태를 담고 있으며 각각 내재적인 특성과 외재적인 특성을 설명하기 위해 구분한다. 좌측의 Boltzmann Machine 과 우측의 Restricted BM 의 차이점은 같은 Layer 에 대한 연결의 유무이다.[2] BM 대신 RBM 을 사용하는 가장 큰 이유는 Hidden Cell 과 Visible Cell 사이의 독립을 가정함으로써 $p(h|v)$ & $p(v|h)$를 계산할 수 있다는 점이며, 이는 이후에 샘플링 과정을 단순화 할 수 있다. 즉, RBM d 개의 Visible Cell 와 n 개의 Hidden Cell 사이의 가중치 $w^{n\times d}$를 가지는 모델로 우리가 흔히 알고 있는 Single Layer Perceptron 과 유사하게 생각할 수도 있지만, RBM 은 데이터를 입력하는 층과 결과를 출력 받는 층이 Visible Cell Layer 로 동일하다는 차이점이 있다. 모델의 형태를 정했기 때문에, 입력되고 출력되는 값들을 비교하기 위해서 정량적(수치적)로 사용할 수 있는 기준이 필요한데 이 때 RBM 에서 사용되는 기준을 **Energy** 라고 부르기 때문에 이와 같은 모델들을 **Energy Based Models(EBMs)**라 부른다.

$$E(v,\ h) = -h^T W\ v\ - b^T v - c^T h \qquad \textit{Energy}$$

정의해주는 Energy 는 위와 같다. 우리는 앞으로 $E(v,h)$의 값을 v,h 상태와 매칭(or Mapping)되는 값이라고 생각할 것이다. 위에서 첫 번째 항은 hidden-visible cells 사이의 interaction 을 두 번째, 세 번째 항은 각각 visible, hidden cell 의 상태 빈도를 학습하기 위한 항이다.) Energy 의 분포를 Boltzmann Distribution[3]로 가정할 경우, 모든 $visible, hidden$ 상태에 대해서 특정 v,h의 확률은 아래와 같다.

$$p_{w,\,b,\,c}(v,\ h) = \frac{\exp\left(-E(v,h)\right)}{Z}, \text{where } Z = \sum_{v,\,h} \exp\left(-E(v,\ h)\right)$$

하지만, RBM 모델을 구성할 때, hidden layer 는 우리가 모르는 상태에 대한 항이기 때문에 hidden Cell 값을 식으로 포함 시킬 경우, 새로운 함수 $F(v) = -log \sum_h \exp\left(-E(v,\ h)\right)$를 정의함으로써 아래와 같이 나타낼 수 있으며, 이 때 새로 정의해준 함수를 Helmholt Free Energy 에서 이름을 따서 Free Energy 라고 부른다.

$$p_{w,\,b,\,c}(v) = \sum_h \frac{1}{Z} e^{-E(v,\,h)} = \frac{1}{Z'} e^{-F(v)}, \text{where } Z' = \sum_v \exp\left(-F(v)\right)$$

Visible Unit 과 Hidden Unit 의 상태가 {0, 1}인 경우만 생각하는 경우, 이를 **Bernoulli RBM** 이라고 칭한다. Bernoulli RBM 에서 Free Energy 의 h 항은 0, 1 로 아래와 같이 변환 가능하다.[4]

$$F(v) = -b^T v\ -\ \sum_{i=1}^{n} log(1 + e^{c_i + W_i v}) \qquad \textit{Bernoulli Helmholt Free Energy}$$

우리가 사용하는 모델이 Bernoulli RBM 임을 이용하면, 조건부 확률 $p(h|v)$ 와 $p(v|h)$를 아래와 같이 정리할 수 있다.[5]
(∗ 동일한 layer 사이의 독립을 가정하는 RBM 이기 때문에 조건분 확률은 $p(h|v) = \prod_j p(hj|v)$ 를 만족한다.)

---

[2] visible layer와 hidden layer 사이의 interaction을 무시하는 것은 입력 데이터 간의 독립성을 가정한다.
이는 시계열 데이터 분석이나, 연관성 분석처럼 하나의 데이터 사이의 독립성을 가정하지 못하는 데이터를 제외하면 필요 없는 항임

[3] $p_i \propto e^{-\epsilon_i/kT}$

[4] Bernoulli RBM은 입력에 대해서 {0, 1}로 변환하는 인코딩 과정을 거치기 때문에 높은 해상도(resolution)을 얻기는 힘들다.

[5] 이진 분류이기 때문에 $p(h_i = 1 \mid v) = 1 - p(h_i = 0 \mid v)$ 와 같다.

$$p(h_i = 1 \mid v) = \sigma(c_i + W_i v)$$
$$p\left(v_j = 1 \mid h\right) = \sigma(b_j + W_j^T v)$$

정량적으로 조건부 확률과 비용(Energy)에 대한 정의를 끝냈다. 그렇다면, RBM 에서 샘플링(Sampling) 과정에 대해서 자세히 들여다 보고, 모델의 학습 과정에 대해서 알아보자. RBM 은 입력된 데이터 $\vec{v}^{(l)}$ 을 이용해서 hidden layer 의 $\vec{h}^{(l)}$를 얻을 경우 이는 입력 데이터의 분포를 일정 학습한 분포이며, 동일한 방법 $\vec{h}^{(l)}$을 이용해서 $\vec{v}^{(l+1)}$를 얻고 이는 $p_{data}(x)$의 분포를 $p_{model}(x)$이 학습하고 있는 과정이라고 할 수 있다.

$$\vec{h}^{(l)} \sim p\left(\vec{h} \mid \vec{v}^{(l)}\right)$$
$$\vec{v}^{(l+1)} \sim p(\vec{v} \mid \vec{h}^{(l)}) \qquad \textit{sampling process}$$

만일 $p_{data}(x)$ 분포를 직접적으로 구하는 Tractable density 추정을 위해서는 $F(\vec{v}^{(l+1)}) =: F(\vec{v}^{(l)})$ 로 수렴할 때까지 위의 과정을 반복해야 한다. 이와 같은 추정은 높은 시간과 연산을 요구하기 때문에 Approximate density 에 포함된 RBM 모델은 분포에 근사를 이용한다.

***Check √; Sampling Process* 가 *Loss* 가 작아지도록 진행하는 학습 과정이라고 할 수 있는가? {**

가능도의 곱이 최대가 되는 Parameters(W, b, c)를 모수의 추정치로 하는 방법을 최대 우도 추정 방법(Method of Maximum Likelihood)라 부른다. Sampling Process 를 통해 얻은 Parameters 가 학습이 잘 되었다면, 이 때의 가능도의 곱이 최대일 것이다. 학습 과정 Gradient Ascent 를 통해 최대값으로 수렴시킬 수 있다.

$$\theta^{l+1} := \theta^l + \alpha \frac{\partial}{\partial \theta} ln\, p(v)$$

즉, 학습이 될수록 $\frac{\partial}{\partial \theta} log\, p(v)$ 의 값의 크기가 작아질 것이다.

$$\frac{\partial}{\partial \theta} \ln p(v) = \frac{\partial}{\partial \theta} \ln \frac{\exp(-F(v))}{Z'} = \frac{\partial}{\partial \theta}(-F(v) - \ln(Z'))$$

$$= -\frac{\partial}{\partial \theta} F(v) - \frac{\partial}{\partial \theta} \ln\left(\sum_{\tilde{v}} \exp\left(-F(\tilde{v})\right)\right.$$

$$= -[\frac{\partial}{\partial \theta} F(v) - \sum_{\tilde{v}} \frac{\exp(-F(\tilde{v}))}{Z'} \frac{\partial}{\partial \theta} F(\tilde{v})\,]$$

$$= -[\frac{\partial}{\partial \theta} F(v) - \sum_{\tilde{v}} p(\tilde{v}) \frac{\partial}{\partial \theta} F(\tilde{v})\,]$$

Expected Value $E(x) = \sum x P(x)$ 임을 이용하면,

$$\frac{\partial}{\partial \theta} ln\, p(v) = -\,[\,\frac{\partial}{\partial \theta} F(v) - E_p\{\frac{\partial F(\tilde{v})}{\partial \theta}\}\,]$$

와 같이 나타낼 수 있다. 이는 위의 Sampling 의 목적이었던, $F(\vec{v}^{(l+1)}) =: F(\vec{v}^{(l)})$ 와 동일한 형태 임을 알 수 있다. 따라서, Sampling 과정을 통해서 학습하는 과정은 MLE 추정 방법을 사용하는 것과 동일하다고 할 수 있다.

};

Sampling Process 를 이용해서 분포의 approximate density 를 학습한다는 것은 $F(\vec{v}^{(l+1)}) =: F(\vec{v}^{(l)})$ 를 만족할 때까지 반복하는 것이 아닌, 일정 횟수만 Sampling Process 를 반복한 후 이를 이용하는 것을 의미한다. 이와 같은 학습 방법을 Contrastive Divergence(CD)라 한다.[6] Geoffrey Hinton 의 연구에 따르면, 단 한번의 샘플링을 한다고 하여도 Local Minimum 로 수렴하는데 큰 어려움이 없다는 사실을 실험적으로 보였으며, 추후에 이론적 증명이 되었다고 한다.

- RBM Implemented code RBM 구현 코드

Bernoulli RBM 은 사용하는데 한계가 존재한다. 현재의 DCGAN 모델들의 성능을 생각하면서, 데이터를 선택하면 학습이 되었는지 전혀 되지 않았는지 구분하기 어려울 수도 있기 때문에 비교적 단순한 데이터를 사용할 수 밖에 없었다. 최근 픽셀 컬러당 10bit 의 정보를 담고 있는 이미지도 등장하였지만, 우리가 사용하는 Bernoulli RBM 은 0 과 1 만을 입력으로 받을 수 있으며 동일하게 0 과 1 만 출력한다. 픽셀 당 1bit 만으로도 명확하게 구분할 수 있는 데이터를 생각했을 때, 손 글씨 이미지(MNIST)를 학습하는 것이 가장 적절할 것이라고 생각했기에 이를 사용하게 되었다.

MNIST database (Modified National Institute of Standards and Technology Database)는 이미지 학습에 가장 많이 사용되는 데이터베이스 중 하나로 일반적으로 사용되는 데이터는 28x28 개의 픽셀로 구성되어 있으며 하나의 픽셀 당 1byte 의 정보를 가진 흑백 1 채널 데이터이다.

코드는 2 개의 코드로 구성되어 있다. (mnist_load.py, rbm_fit.py) mnist_load.py 는 torchvision.datasets 에서 단순히 데이터를 로드 하는 과정이기에 설명을 생략한다.

rbm_fit.py 는 크게 3 개의 함수 및 클래스를 정의한다.
- func: save_img(file_name, img)
- class: RBM(n_vis, n_hid, k)
- func: main

save_img:
모델을 통해서 얻은 결과값은 이미지를 나타내는 값이다. 따라서 이를 시각화 하지 않을 경우, 학습의 정도를 파악하는 것이 어렵기 때문에 체크 포인트마다 얻은 결과값을 이미지로 디스크에 저장하기 위한 함수이다.

RBM:
rbm_fit.py 에서 사용하는 RBM 모델을 정의하기 위한 클래스이다. 클래스는 아래와 같이 구성되어 있다.

Instance variables {
- w
  visible cell - hidden cell 사이의 가중치 (*hidden_size* x *vissible_size*)
- *v_bias*
  visible cell bias
- *h_bias*
  hidden cell bias
- *k*
  Contrastive Divergence; CD 횟수 지정(위에서 설명한 바와 같이 *k=1* 사용)

}

[6] The learning rule is much more closely approximating the gradient of another objective function called the Contrastive Divergence (Hinton, 2002)

methods {

- init

클래스 생성자로 instance variables 생성 및 초기화를 수행한다.

visible cell 의 개수는 우리가 알고 있는 특징들을 의미하기에 입력 데이터 개수(28x28=784)와 같다.

- sampling(p)

입력 받은 확률에 대한 분포를 생성해서 리턴

- v_to_h(v)

$\vec{h}^{(l)} \sim p(\vec{h} \mid \vec{v}^{(l)})$ 과정을 의미

$p(h_i = 1 \mid v) = \sigma(c_i + W_i v)$ 의 값을 *sampling* 함수를 이용해서 $\vec{h}^{(l)}$ 생성

$p(h_i = 1 \mid v)$, $\vec{h}^{(l)}$ 리턴

- h_to_v(h)

$\vec{v}^{(l+1)} \sim p(\vec{v} \mid \vec{h}^{(l)})$ 과정을 의미

$p(v_j = 1 \mid h) = \sigma(b_j + W_j^T v)$ 의 값을 *sampling* 함수를 이용해서 $\vec{v}^{(l+1)}$ 생성

$p(v_j = 1 \mid h)$, $\vec{v}^{(l+1)}$ 리턴

- forward(v)

training data $\vec{v}^{(0)}$ 받아서 설정된 CD 횟수 k 번 만큼 sampling 수행

$\vec{v}^{(0)}$, $\vec{v}^{(k)}$ 리턴

- freeEnergy(v)

Loss 로 사용되는 $F(v) = -b^T v - \sum_{i=1}^{n} log(1 + e^{c_i + W_i v})$ 값을 리턴

}

main:

rbm_fit.py 의 main 함수로 RBM 인스턴스 생성 및 학습 수행. 학습 이 후, 테스트 데이터를 이용한 평가 수행.

(데이터를 학습하기 전 data 의 bernoulli() 메소드를 통해 값을 부호화 처리해 주어야 함 ∵ 우리가 사용하는 모델은 Bernoulli RBM 임)

이하 전체 코드는 아래와 같다.

rbm_fit.py

```
#!/opt/homebrew/Caskroom/miniforge/base/envs/tf_38/bin/python
import numpy as np
import torch
import torchvision
import matplotlib.pyplot as plt

def save_img(file_name, img):
  f="./pic/%s.png"%file_name
  img=np.transpose(img.numpy(), (1, 2, 0))
  plt.imsave(f, img)

# RBM model with torch
class RBM(torch.nn.Module):
  def __init__(self, n_vis=784, n_hid=500, k=5):
    super(RBM, self).__init__()
    # weigths rand initialize
    self.W=torch.nn.Parameter(torch.randn(n_hid, n_vis)*1e-2)
    # bias 0 initialize
    self.v_bias=torch.nn.Parameter(torch.zeros(n_vis))
    self.h_bias=torch.nn.Parameter(torch.zeros(n_hid))
```

```
    # Number of Divergence
    self.k=k

  def sampling(self, p):
    # from Gaussian distribution: achieve probability
    _p=p-torch.autograd.Variable(torch.rand(p.size()))
    # if _p>p return 1 else return -1;
    p_sign=torch.sign(_p)
    # with Relu-> if _p>p return 1 else return 0;
    return torch.nn.functional.relu(p_sign)

  def v_to_h(self, v):
    # P(h_i=1|v)=sigmoid(c_i+W_i*v)
    p_h=torch.sigmoid(torch.nn.functional.linear(v, self.W, self.h_bias))
    sample_h=self.sampling(p_h)
    return p_h, sample_h

  def h_to_v(self, h):
    # P(v_j=1|h)=sigmoid(b_j+W'_j*h)
    p_v=torch.sigmoid(torch.nn.functional.linear(h, self.W.t(), self.v_bias))
    sample_v=self.sampling(p_v)
    return p_v, sample_v

  def forward(self, v):
    p_h0, h0=self.v_to_h(v)
    _h=h0
    for _ in range(self.k): # Divergence count
      _p_v, _v=self.h_to_v(_h)
      _p_h, _h=self.v_to_h(_v)

    return v, _v    # return initial v and after k times divergenced _v

  def freeEnergy(self, v):
    # F(v)=-b'v-SUM_i[ log(1+exp(c_i+W_iv)) ]
    vbias=v.mv(self.v_bias) # b'v
    wx_b=torch.nn.functional.linear(v, self.W, self.h_bias) # c+Wv
    sum_i=torch.log(1+torch.exp(wx_b))
    ssum=torch.sum(sum_i, dim=1)
    # F(v)^(1)
    return (-ssum-vbias).mean()

if __name__=="__main__":
  # k=1 conversion 1 times
  rbm=RBM(k=1)
  # optimizer Adam
  optimizer=torch.optim.Adam(rbm.parameters(), 1e-3)

  batch_size=64
  train_loader = torch.utils.data.DataLoader(
    torchvision.datasets.MNIST('./', train=True, download=True,
transform=torchvision.transforms.Compose([torchvision.transforms.ToTensor()])), batch_size=batch_size)

  test=torchvision.datasets.MNIST('./', train=False,
transform=torchvision.transforms.Compose([torchvision.transforms.ToTensor()]))
  # test with 32 data
  sample_data=test.data[:32, :].view(-1, 784)
  sample_data=sample_data.type(torch.FloatTensor)/255.
```

```
v, v1=rbm(sample_data)
save_img("label", torchvision.utils.make_grid(v.view(32, 1, 28, 28).data))
save_img("before_train", torchvision.utils.make_grid(v1.view(32, 1, 28, 28).data))

for epoch in range(100):
  loss=[]
  for _, (data, label) in enumerate(train_loader):
    data=torch.autograd.Variable(data.view(-1, 784))
    # we consider bernoulli h
    _sample_data=data.bernoulli()
    # forward
    v, v1=rbm(_sample_data)
    _loss=rbm.freeEnergy(v)-rbm.freeEnergy(v1)
    loss.append(_loss.data.item())
    optimizer.zero_grad()
    _loss.backward()
    optimizer.step()
  print("EPOCH: %d\r" %(epoch+1), flush=True, end="")
  save_img("epoch%d" %(epoch+1), torchvision.utils.make_grid(v1.view(32, 1, 28, 28).data))

v, v1=rbm(sample_data)
save_img("after_train", torchvision.utils.make_grid(v1.view(32, 1, 28, 28).data))
plt.plot(loss, "co")
plt.show()
```

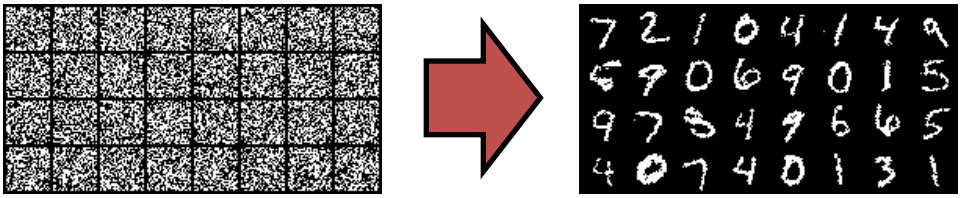
. Conclusion and Impression 결론 및 느낀 점

처음 머신러닝과 딥러닝에 관심이 생기고, 다양한 형태의 딥러닝을 공부해 보고 싶었기에 많은 모델들을 구현해보고 이해하기 위해서 노력해 본적이 있다. 그러던 중 만나게 된 Restricted Boltzmann Machine 은 이전의 모델들과는 많이 다른 모습을 가지고 있었다. 다른 모델들에 비해 압도적으로 많은 수학 수식들로 인해 다양한 커뮤니티에서 악명이 높았던 RBM 및 Energy Based Models 에 나 또한 겁이 났었고, GANs 라는 더 높은 성능을 발휘하는 모델에 묻혀 현대에는 많이 사용되지 않는 "역사속으로 사라진 모델" 이라고 스스로 위로하며 애써 못 본 척 넘어 갔던 기억이 난다. 교수님께서 이번 프로젝트에서 평소라면 귀찮아서 하지 않았을, 하지만 이렇게 시간이 주어진다면 하고 싶은 주제에 대해서 프로젝트를 진행해 보라는 말씀이 듣고, RBM 이 스치듯이 떠올랐고 팀원들과 함께 공부해보고 한번 구현해 보기로 마음을 먹게 되었다.

RBM 은 학습시키기 어렵다고 악명이 높은 GAN 에 비교했을 때, 굉장히 학습 시키는 것이 쉬웠다. 학습을 시킬 때 이전에 GAN 로 이미지를 생성시켰을 때, 100 번 정도의 Epoch 를 수행시켰을 때, 그럴 듯한 MNIST 이미지가 생성되었던 것이 생각이 나 동일하게 100 Epoch 로 설정하고 수행하였으나, 대략 10 Epoch 에서부터는 더 이상 성능이 향상되지 않는다는 것을 느꼈다. 이를 토대로 봤을 때, RBM 은 굉장히 학습 시키기가 쉬운 모델이라는 생각이 들었다.

RBM 20 epoch 이 후, 학습데이터에 대한 생성한 이미지

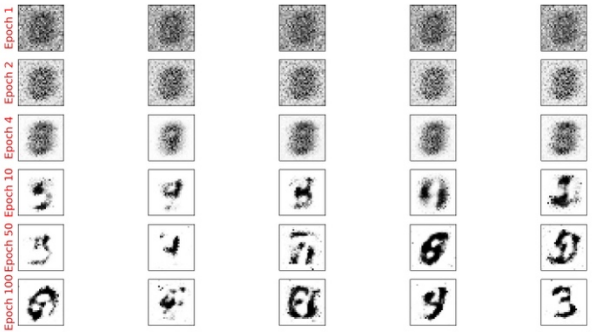
이미지를 살펴보면, 손 글씨가 인간이 판단할 수 있는 수준 이상으로 생성해 낸 것을 확인할 수 있다. 이미지가 기대보다 흐릿하다고 생각할 수 있을 것이다. 하지만, 해당 모델이 0 과 1 만 출력하는 Bernoulli RBM 임을 생각하면, 상당히 잘 학습하였다는 생각이 들었다.



학습이 되지 않았을 때, 랜덤(임의의) 분포에 대한 결과물을 리턴 하였지만, 학습에 사용된 이미지의 분포를 학습한 이후, 우리가 원하는 수준의 손 글씨를 생성했다고 결론을 내릴 수 있었다.

입력된 데이터보다 화질이 떨어지거나 거의 유사한 데이터를 왜 생성하는 것일까? 이에 대해서는 생성 모델의 사용 범주에 대해서는 위에서 설명한 바[7]가 있지만, 우리의 프로젝트 목표와 같기 때문에 다시 한번 언급하겠다. 생성 모델의 가장 큰 목적은 입력되는 데이터의 분포 즉, 양상을 학습하는데 있다. 분포를 학습하게 되면, 이를 토대로 생성모델은 무수히 많은 데이터를 생성해 낼 수 있는 능력을 갖게 된다. 이를 이용해서 학습 할 때, 부족한 데이터이거나 관측이 어려운 데이터의 패턴을 학습해서 쉽게 생성해 내거나, 기존에 존재하지 않는 것을 창조해 낼 수 있는 능력이 있기 때문에 생성 모델이 중요하다고 생각한다.

마지막으로, 우리가 공부하였던 Energy Based Models 와 최근 생성 모델 연구에서 주목받고 있는 Explicit Sampling 모델들에 대한 간단한 비교하면서 보고서를 마무리 해보고자 한다.



GAN (not DCGAN) MNIST Validation

위의 사진은 대략 1 년 전 GAN 에 대해 공부해 보면서 MNIST 데이터를 단순한 모델을 만들어서 학습시켰던 데이터가 남아 있어 비교를 위해 가져왔다. 해당 GAN 모델은 Tensorflow 로 구현하였고, 이번 프로젝트에서 생성한 RBM 모델과 거의 유사한 가중치 개수를 가지는 모델이었다. 비교를 해보면 GAN 이 학습하는 과정이 훨씬 느리다는 느낌을 받을 수 있을 것이다. 최근에 보다 다양하고 강력한 GANs 들은 보다 높은 정확도와 학습 안정성을 지니고 있지만, 나는 RBM 의 직접적인 통계 샘플링

[7] 생성 모델의 종류와 역사 (해당 보고서 pg. 3)

과정에서 더 높은 안정성을 가질 수 있다고 생각한다. 그와 같은 조사를 하던 중, 위의 내용에 관한 논문이 있어서 이의 내용을 일부 발췌하면, Implicit Models 는 아래와 같은 장점을 가지고 있다고 한다.[8]

- Simple and Stability	*단순성과 안정성*
- Sharing of Statistical Strength	*통계적 힘의 공유*
- Adaptive Computation Time	*소요시간 대비 높은 성능*
- Flexibility Of Generation	*유연하게 데이터를 생성(구조에 대한 구속이 적음)*
- Compositionality	*학습의 난이도가 낮음*

따라서 Implicit Models 의 기본적인 원리를 이해하고, Explicit Models 를 보완하고자 할 때, 이와 같은 지식들은 분명히 큰 힘으로 작용할 것이라고 생각한다.

. My Contribution 본인의 프로젝트 기여 내용

나는 위 RBM 모델 생성 프로젝트에서 **주제선정**, **ppt 제작**과 **코드 구현**을 맡았다. 평소 다양한 머신러닝 모델들에 대한 관심이 많았기 때문에 RBM 이라는 모델을 사전에 알고 있었고, 이를 조원들에게 소개하고 프로젝트를 수행할 수 있었다. 그리고 평소에 공부해둔 코드들을 정리하는 것을 좋아하고 프로젝트를 수행하기 위해 필요한 컴퓨팅 환경을 갖추고 있었기 때문에 직접 코드 구현을 수행하고, 결과를 얻는데 기여할 수 있었다.

[8] Implicit Generation and Modeling with Energy Based Models (2014)