Lecture3 : Auto-Regressive Models

Ki Pyo	완료
선택	Auto-Regressive Model

1. Distribution Learning With Neural Models

- **Given:** a dataset $\mathcal{D}$ of handwritten digits (binarized MNIST)

- Each image has $n = 28 \times 28 = 784$ pixels. Each pixel can either be black (0) or white (1).
- **Goal:** Learn a probability distribution $p(x) = p(x_1, \cdots, x_{784})$ over $x \in \{0, 1\}^{784}$ such that when $x \sim p(x)$, x looks like a digit

MNIST image에 대한 joint probablity distribution을 학습하고 싶을때,

- 해당 분포를 표현하기 위한, 파라미터로 조절 가능한 분포 모델링 (Modeling) : 데이터가 나올 확률을 수학적으로 계산할 수 있는 함수의 형태(모델 구조)로 먼저 정하기 ⇒ 파라미터화 하기.

$$\{p_\theta(x), \theta \in \Theta\}$$

  가능한 모든 파라미터에 대해 만들어지는 Distribution 집합 ⇒ Model Family

**(그래서 보통 Pdata(모집단)과 우리가 가정하는 Ptheta(Parameterized Distribution)은 상이함.)

- Training Sample을 기반으로 모델의 최적 파라미터 찾기 (Learning)

이 챕터에서는 Autoregressive Model(modeling, parameterize)을 통해 해당 joint distribution을 학습(Learning)하고 싶음. 이번 강의에서는 특히나 Modeling을 다룸.

2. Autoregressive Model for image pixels

이미지 픽셀 간에는 큰 상관관계(및 인과관계)를 보이지 않음

→ 픽셀 확률 변수 간에 ordering에 민감 하지 않을 것임

→ Ordering : 샘플링 할 시에 어떤 순서로 샘플링? (조건부 의존 관계를 표현)
→ Joint probability distribution factorizing할 시에 ordering을 어떤 방향이든 상관없이 하면됨. (여기서는 ordering을 x1(이미지의 왼쪽 위)부터 시작하여 xn(오른쪽 아래)까지 했다고 설정 => raster scan ordering)

Without loss of generality, we can use chain rule for factorization

$$p(x_1, \cdots, x_{784}) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \cdots p(x_n \mid x_1, \cdots, x_{n-1})$$

→ 그럼 joint distribution은 다음과 같이 factorize됨

$$p(x_1, \cdots, x_{784}) = p_{\mathrm{CPT}}(x_1; \alpha^1) p_{\mathrm{logit}}(x_2 \mid x_1; \boldsymbol{\alpha}^2) p_{\mathrm{logit}}(x_3 \mid x_1, x_2; \boldsymbol{\alpha}^3) \cdots p_{\mathrm{logit}}(x_n \mid x_1, \cdots, x_{n-1}; \boldsymbol{\alpha}^n)$$

→ 각각 Factorize된 분포는 이제 Tabular Form으로 나타내기에는 경우의 수가 너무 많아 너무나 많은 값들을 저장해야하기에

→ 각각의 분포(분포에 대한 파라미터 : 위의 예시는 각 픽셀이 베르누이 분포를 따르니... P(Y=1|x1,...xn))를 neural network로 functionalize 하여 출력하도록함

$$p_{\mathrm{CPT}}(X_1 = 1; \alpha^1) = \alpha^1,\ p(X_1 = 0) = 1 - \alpha^1$$
$$p_{\mathrm{logit}}(X_2 = 1 \mid x_1; \boldsymbol{\alpha}^2) = \sigma(\boldsymbol{\alpha}_0^2 + \boldsymbol{\alpha}_1^2 x_1)$$
$$p_{\mathrm{logit}}(X_3 = 1 \mid x_1, x_2; \boldsymbol{\alpha}^3) = \sigma(\boldsymbol{\alpha}_0^3 + \boldsymbol{\alpha}_1^3 x_1 + \boldsymbol{\alpha}_2^3 x_2)$$

→ 이런식으로! 각 픽셀 분포에 대한 파라미터를 neural network로 함수화하여 출력

3. Auto-Regressive Model Summary

- Auto-regressive 모델은 joint probability distribution을 각 확률 변수간의 order를 부여함과 동시에 체인 룰(chain rule)로 factorization하여, 시퀀스나 이미지의 각 요소를 이전에 생성된 요소들에 조건부로 순차적으로 예측하는 모델.
- 예를 들어, 이미지에서의 auto-regressive model은 픽셀 각각의 값에 대한 확률 변수를 나열한 뒤, 첫 번째 픽셀의 분포를 예측하고, 그 값을 조건으로 두 번째 픽셀의 분포를 예측하는 식으로 반복.
- 각 조건부 확률은 경우의 수가 너무 많으므로 테이블 형태 대신 로지스틱 회귀나 신경망처럼 **파라미터화된 함수**로 근사(정확하게는 조건부 확률 분포에 대한 파라미터를 근사)하며, 이를 통해 복잡한 데이터의 전체 분포를 모델링 할 수 있음.

4. Fully Visible Sigmoid Belief Network (FVSBN)

The conditional variables $X_i \mid X_1, \cdots, X_{i-1}$ are Bernoulli with parameters

$$\hat{x}_i = p(X_i = 1 | x_1, \cdots, x_{i-1}; \boldsymbol{\alpha}^i) = p(X_i = 1 | x_{<i}; \boldsymbol{\alpha}^i) = \sigma(\alpha_0^i + \sum_{j=1}^{i-1} \alpha_j^i x_j)$$

→ Auto-Regressive model이고 각 확률 변수가 베르누이를 따른다고 했을 때, 각각의 조건부 확률 분포에 대한 파라미터는 이전 order의 확률 변수들에 대해 logistic regression을 하여 구한다.

Evaluation of P(x1, x2...xn)?

Sample이 주어졌을 때 해당 sample이 모델링 및 학습한 분포에 대해 얼마나 likely하냐? (있을 법하냐를 evaluate함. Anomaly detection에 이용될 듯)

→ Sigmoid 결과값은 P(y=1|x...)이기에 아래의 식과 같음

How to evaluate $p(x_1, \cdots, x_{784})$? Multiply all the conditionals (factors)

- In the above example:

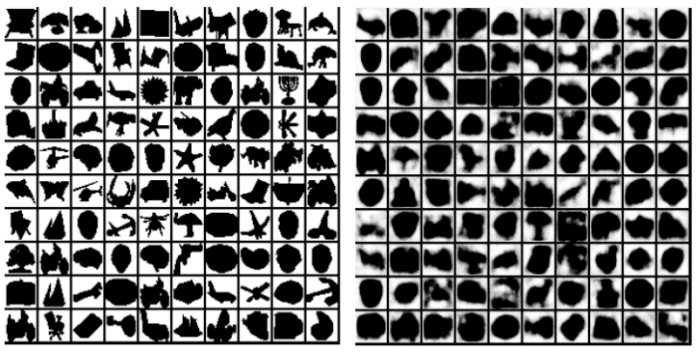
$$p(X_1 = 0, X_2 = 1, X_3 = 1, X_4 = 0) = (1 - \hat{x}_1) \times \hat{x}_2 \times \hat{x}_3 \times (1 - \hat{x}_4)$$
$$= (1 - \hat{x}_1) \times \hat{x}_2(X_1 = 0) \times \hat{x}_3(X_1 = 0, X_2 = 1) \times (1 - \hat{x}_4(X_1 = 0, X_2 = 1, X_3 = 1))$$

Sampling or Generation

- How to sample from $p(x_1, \cdots, x_{784})$?
  1. Sample $\bar{x}_1 \sim p(x_1)$ (`np.random.choice([1,0],p=[`$\hat{x}_1, 1-\hat{x}_1$`])`)
  2. Sample $\bar{x}_2 \sim p(x_2 \mid x_1 = \bar{x}_1)$
  3. Sample $\bar{x}_3 \sim p(x_3 \mid x_1 = \bar{x}_1, x_2 = \bar{x}_2) \cdots$

- 위와 같이 ordering에 기반하여 sampling (generation)

Training + Sampling Results

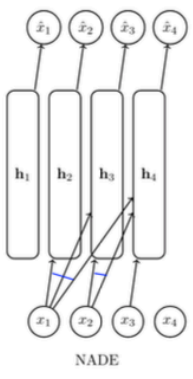


Training Generated Samples

매 픽셀에 대한 분포의 파라미터 예측에 사용되는 logistic regression model이 이전 order로 주어진 픽셀들 간의 복잡한 관계를 표현할 만큼의 capa가 없기 때문에 generation quality가 좋지 않음.

5. NADE: Neural Autoregressive Density Estimation

→ 일반 logistic regression 앞에 linear layer + non-linearity 하나 추가하는 방법



NADE

- To improve model: use one layer neural network instead of logistic regression

$$\mathbf{h}_i = \sigma(A_i \mathbf{x}_{<i} + \mathbf{c}_i)$$

$$\hat{x}_i = p(x_i | x_1, \cdots, x_{i-1}; \underbrace{A_i, \mathbf{c}_i, \alpha_i, b_i}_{\text{parameters}}) = \sigma(\alpha_i \mathbf{h}_i + b_i)$$

- For example $\mathbf{h}_2 = \sigma\left(\underbrace{\begin{pmatrix}\vdots\end{pmatrix}}_{A_2} x_1 + \underbrace{\begin{pmatrix}\vdots\end{pmatrix}}_{c_2}\right)$ $\mathbf{h}_3 = \sigma\left(\underbrace{\begin{pmatrix}\vdots & \vdots\end{pmatrix}}_{A_3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \underbrace{\begin{pmatrix}\vdots\end{pmatrix}}_{c_3}\right)$

issue)

앞의 방법과 같이 auto-regressive (ordering)를 따르는 모델이기에 메 시점(또는 확률 변수를)을 예측할 때마다 model의 structure가 달라지기에 common structure를 공유하는 model을 만들 수가 없음

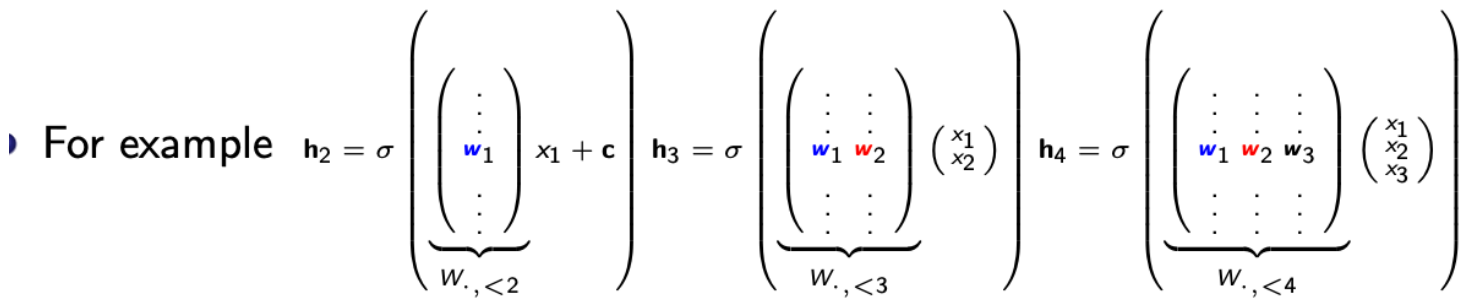
→ 각 시점마다 들어오는 입력에 맞는 예측을 하는 모델을 구축해야함 (파라미터가 많을 것임)
ex) P(x1)은 1일지에 대한 확률 파라미터 하나로 결정되고

x2는 x1만을 입력으로 받는 모델이어야 할 것이고, x3에 대한 prob를 예측하는 구조에서는 x1,x2를 모두 입력으로 받아야할 것임.

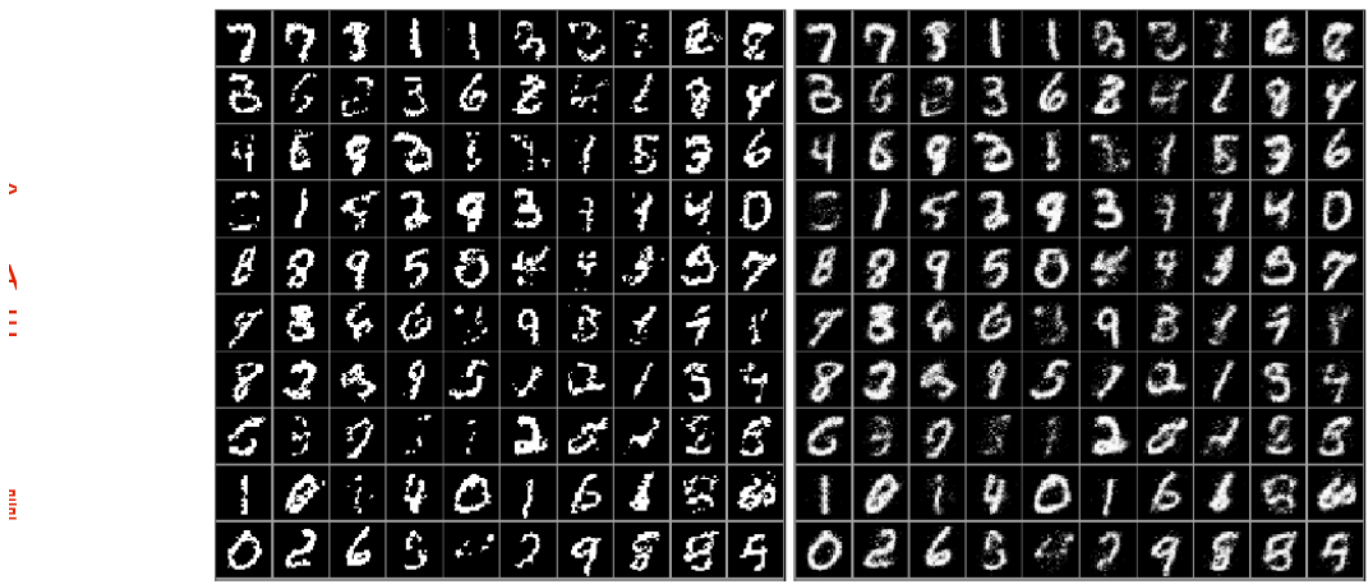
parmeter 수를 줄이는 방법이 없을까?

→ Tie parameters

- 파라미터를 공유하는 방식을 쓴다
- x1-xN을 모두 처리할 수 있는 matrix를 구축하고, 매 시점(또는 order)마다 해당 order 전까지에 대응하는 column들만 추출.이런 식으로 전체 네트워크에 대한 파라미터를 하나의 matrix로 구현 가능
- 가중치 공유는 이루어지나 여전히 n개의 서로 다른 네트워크 구조를 이용해야 함 (하나의 matrix에서 weight column들을 추출하는 방법으로 구성), 뒤의 autoencoder auto-regressive version의 경우 single neutal network로 구현가능함.



- 확실히 parameter 수가 줄어드니 overfitting 방지가 될수 있겠다 => 일종의 weight sharing (질문 내용)
- Computation Reusability도 가능할 수 있겠다. → 이전 스텝에서 계산된 column * x값을 저장하는 방식으로



Samples from a model trained on MNIST on the left. Conditional probabilities $\hat{x}_i$ on the right.
Figure from *The Neural Autoregressive Distribution Estimator, 2011*.

→ 왼쪽의 이미지 경우 Sampling 시에 binary하게 (0,1)로 Sampling된 값이고, 오른쪽의 경우 sampling할 때 매 픽셀마다 1이 될 확률을 soft하게 나타낸 것(neural network의 출력값) 이전 스탭의 값들은 당연히 binary한 값을 넣어줌.

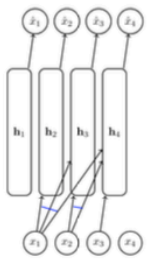
→ 이 case의 경우 P(x1,x2 ... xn, y)를 모델링하는 것이 아닌
P(x1,x2,x3,...,xn)을 모델링하는 것 질의 내용에 따르면 y가 주어졌을 때(일종의 condition?) 더 generation을 잘할 수 있다고 함.

→ 이 NADE 모델은 y를 반영한 분포를 만들어 내는 것이 아니기에 P(x1,x2,..xn) 원하는 label을 뽑아내는 방식이 아니라 아무래도 가장 나올 법한 숫자 이미지를 generate 할 것임(모델 훈련이 잘 되었다는 가정하에)

→ 각각의 확률을 예측하는 모델이 일반 logistic regression 모델보다 더 복잡한 패턴을 capture하도록 만들어져 현 step 기준 이전 픽셀들 사이의 관계를 더 반영한 상태에서 다음 픽셀에 대한 분포 파라미터를 NN이 출력할 수 있게 되겠다

6. General Discrete Distributions

y
n
각
값
각



How to model non-binary discrete random variables $X_i \in \{1, \cdots, K\}$? E.g., pixel intensities varying from 0 to 255
One solution: Let $\hat{\boldsymbol{x}}_i$ parameterize a categorical distribution

$$\mathbf{h}_i = \sigma(W_{\cdot,<i}\mathbf{x}_{<i} + \mathbf{c})$$
$$p(x_i|x_1, \cdots, x_{i-1}) = Cat(p_i^1, \cdots, p_i^K)$$
$$\hat{\boldsymbol{x}}_i = (p_i^1, \cdots, p_i^K) = softmax(A_i\mathbf{h}_i + \mathbf{b}_i)$$

Softmax generalizes the sigmoid/logistic function $\sigma(\cdot)$ and transforms a vector of K numbers into a vector of K *probabilities* (non-negative, sum to 1).

$$softmax(\mathbf{a}) = softmax(a^1, \cdots, a^K) = \left(\frac{\exp(a^1)}{\sum_i \exp(a^i)}, \cdots, \frac{\exp(a^K)}{\sum_i \exp(a^i)}\right)$$

In numpy: `np.exp(a)/np.sum(np.exp(a))`

- Categorical Distribution

⇒ 지금까지는 각각의 확률 변수가 binary했다. 만약 여러 개의 discrete한 value를 가질 수 있다면? neural network는 categorical distribution에 대한 파라미터를 출력해야겠다 : 각 value에 대한 확률 softmax normalization값을 출력하면됨

- Continuous Distribution

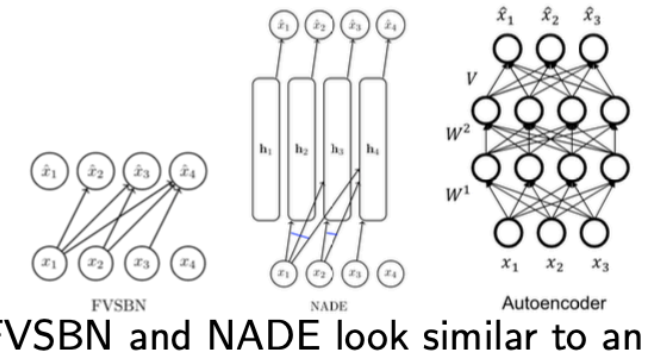
⇒ 역시 똑같이 이에 대한 파라미터를 출력하면됨.

E.g., In a mixture of K Gaussians,

$$p(x_i|x_1, \cdots, x_{i-1}) = \sum_{j=1}^{K} \frac{1}{K}\mathcal{N}(x_i; \mu_i^j, \sigma_i^j)$$
$$\mathbf{h}_i = \sigma(W_{\cdot,<i}\mathbf{x}_{<i} + \mathbf{c})$$
$$\hat{\boldsymbol{x}}_i = (\mu_i^1, \cdots, \mu_i^K, \sigma_i^1, \cdots, \sigma_i^K) = f(\mathbf{h}_i)$$

→ 각 k개의 가우시안에 대한 mean과 std를 출력하도록 하면됨

7. Autoregressive models & Autoencoders



- On the surface, FVSBN and NADE look similar to an **autoencoder**:
- an **encoder** $e(\cdot)$. E.g., $e(x) = \sigma(W^2(W^1 x + b^1) + b^2)$
- a **decoder** such that $d(e(x)) \approx x$. E.g., $d(h) = \sigma(Vh + c)$.
- Loss function for dataset $\mathcal{D}$ reconstruction

$$\text{Binary r.v.:} \quad \min_{W^1, W^2, b^1, b^2, V, c} \sum_{x \in \mathcal{D}} \sum_i -x_i \log \hat{x}_i - (1 - x_i) \log(1 - \hat{x}_i)$$

$$\text{Continuous r.v.:} \quad \min_{W^1, W^2, b^1, b^2, V, c} \sum_{x \in \mathcal{D}} \sum_i (x_i - \hat{x}_i)^2$$

- e and d are constrained so that we don't learn identity mappings. Hope that $e(x)$ is a meaningful, compressed representation of x (feature learning)
- A vanilla autoencoder is *not* a generative model: it does not define a distribution over x we can sample from to generate new data points.

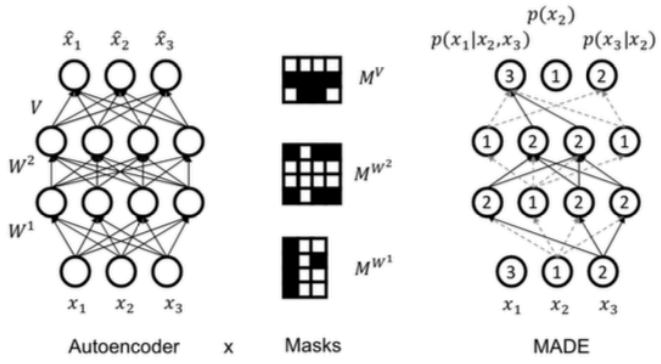
- Encoder/Decoder 구조의 AE :good compressed representation을 얻는 것이 목표이기 때문에, Encoder와 Decoder가 Identity Mapping을 하지 않도록 constraint를 주어야함.
- Difference between Autroregressive Models and Autoencoder: AE는 X에 대한 분포를 define하고 있지 않기에 Auto-regree model과 달리 sampling이 가능하지 않다.
- 반면 auto-regressive model은 확률 변수간의 ordering을 기반으로 joint probability distribution factorization을 통한 모델링 및 학습하기에 sampling(generation)이 가능.
- Auto-Regressive model은 각각 확률 변수에 대한 확률 분포를 뽑아내는 구조이지만, AE는 확률 분포를 뽑아내는 구조가 아님.

8. Autoregressive Autoencoders?

→ 그럼 이 Autoencoder를 autoregressive로 만들어(각 확률변수에 대한 확률분포를 출력하도록 만들어) generative model로 만들 방법이 있을까?

- training 및 generation 과정에서 변수 간의 ordering을 부여하도록 강제하면됨
- 모든 입력이 한번에 들어가는 AE 구조에서 Ordering을 부여한다면 현재 입력을 기준으로 다음을 예측할 때는 다음 입력에 대한 weight는 masking해야함
- Masking 방식을 이용하면 NADE와 달리 single neural network로 "training" 하는 것이 가능해짐(sequential하게 하지 않아도 모든 order에 대해 이전 order에 대한 확률변수들을 기반으로 distribution parameter를 한번에 출력할 수 있음) → Sequential training이 일어나지 않게함
- 여전히 inference(sampling)할 시에는 sequential하게 해야함. 이러한 sequential sampling 방식은 모든 auto-regressive model에 적용됨.

Weight Masking Algorithm



1. **Challenge**: An autoencoder that is autoregressive (DAG structure)
2. **Solution**: use masks to disallow certain paths (Germain et al., 2015). Suppose ordering is x_2, x_3, x_1, so $p(x_1, x_2, x_3) = p(x_2)p(x_3 \mid x_2)p(x_1 \mid x_2, x_3)$.
   1. The unit producing the parameters for $\hat{x}_2 = p(x_2)$ is not allowed to depend on any input. Unit for $p(x_3|x_2)$ only on x_2. And so on...
   2. For each unit in a hidden layer, pick a random integer i in $[1, n-1]$. That unit is allowed to depend only on the first i inputs (according to the chosen ordering). 왜 smaller or equal? : hidden layer에서는 2의 경우 1,2를 다 참고할 수 있지만 output layer의 2는 1까지만 참고해야 하지 않을까?
   3. Add mask to preserve this invariant: connect to all units in previous layer with smaller or equal assigned number (strictly $<$ in final layer)

- n은 총 order number개수

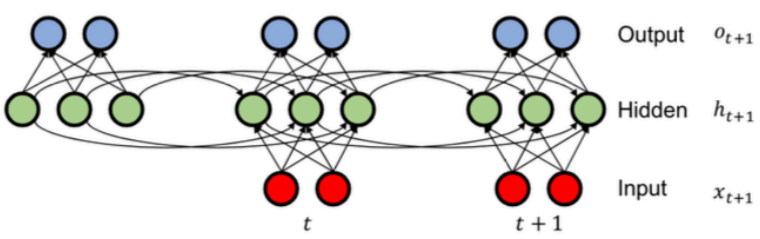
1st order 변수의 출력에는 아무런 입력도 개입 x

2nd order 변수의 출력에는 1st order만 개입하도록...

hidden의 경우 숫자 이하까지 개입가능!

9. RNN

Challenge: model $p(x_t|x_{1:t-1}; \alpha^t)$. "History" $x_{1:t-1}$ keeps getting longer.
Idea: keep a summary and recursively update it



$$\text{Summary update rule: } h_{t+1} = tanh(W_{hh}h_t + W_{xh}x_{t+1})$$
$$\text{Prediction: } o_{t+1} = W_{hy}h_{t+1}$$
$$\text{Summary initalization: } h_0 = \boldsymbol{b}_0$$

1. Hidden layer h_t is a summary of the inputs seen till time t
2. Output layer o_{t-1} specifies parameters for conditional $p(x_t \mid x_{1:t-1})$
3. Parameterized by $\boldsymbol{b}_0$ (initialization), and matrices W_{hh}, W_{xh}, W_{hy}. Constant number of parameters w.r.t n!

RNN 구조도 joint probability를 variable ordering(조건부 의존관계 명시)을 통해 chain rule에 따라 factorization하여 모델링하고 학습할 수 있어 order sampling을 통한 generation에 이용될 수 있음 (확률
변수 간의 관계에 대한 가정(조건부 독립)은 하지 않음)

이도 역시 각 order에 대한 distribution이 이전까지의 order에 대한 conditional distribution이기에 Auto-Regressive model임.

RNN은 입력되는 조건(이전 시점의 정보들)이 후반부로 갈수록 많아지는 문제를 해결하기 위해 이전 시점(ordering)까지의 조건들에 대한 summary(이전 시점까지의 정보)를 hidden state로 요약하는 방법을 씀. 이 hidden state는 recursive하게 시점이 지나면서 update됨.

Density(Likelihood) Estimation in RNN

$$\textbf{Autoregressive: } p(x = hello) = p(x_1 = h)p(x_2 = e|x_1 = h)p(x_3 = l|x_1 = h, x_2 = e) \cdots p(x_5 = o|x_1 = h, x_2 = e, x_3 = l, x_4 = l)$$

For example, ht를 ot로 linear transformation 해당 시점에서의 나올 단어들에 대한 distribution을 뽑아내는 것이 목표이니까 softmax normalization

$$p(x_2 = e|x_1 = h) = softmax(o_1) = \frac{\exp(2.2)}{\exp(1.0) + \cdots + \exp(4.1)}$$

$$o_1 = W_{hy}h_1$$

$$h_1 = tanh(W_{hh}h_0 + W_{xh}x_1)$$

- Density Estimation인 경우: 훈련 다 한후 parameter fixed된 상태에서의 분포에서 해당 값을 가질 확률을 출력
- Likelihood Evaluation인 경우: 훈련 과정에서 변화하는 파라미터에 대한 분포에서 해당 값이 나올 정도를 나타냄

Density & Likelihood

10. Pros & Cons of RNN Structure

Pros

- Can be applied to sequences of arbitrary length

Cons

- Sequential Both in Training & Inferencing (SLOW)
- Training : Sequential Likelihood Evaluation
- Generation : Sequential Sampling in Inference

Generation in RNN

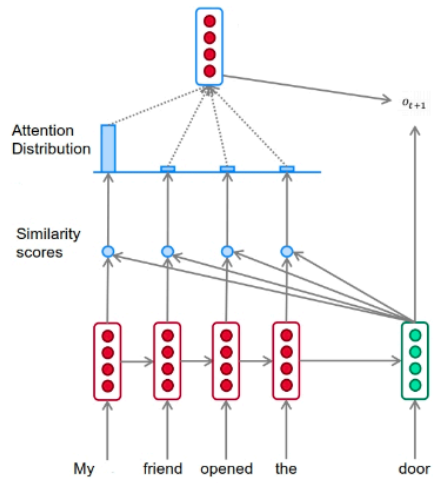
→ RNN은 변수간의 ordering을 가정하기에 joint probability를 chain rule을 통해 Factorization 가능하고 각각 factorization된 distribution에 대한 parameter(예를 들어 vocal에 대한 각 단어에 대한 확률이면 각 단어들이 나올 확률들을 표현하는 softmax normalized vector)는 neural network를 통해 구할 수 있음.
이 모델을 잘 훈련시킨다면 학습때 배운 텍스트들을 기반으로 order기반 sequential sampling을 통해 학습데이터와 비슷한 그럴듯한 샘플을 만들어낼 수 있음

11. Issues of RNN

- Sequential한 likelihood evaluation과 sampling
- Bottleneck : 이전 스텝의 조건들을 어떻게 하나의 hidden vector로 요약해야하지..

⇒ Attention Mechanism로 이를 해결하자

12. Attention



Attention mechanism to compare a *query* vector to a set of *key* vectors

1. Compare current hidden state (query) to all past hidden states (keys), e.g., by taking a dot product
2. Construct attention distribution to figure out what parts of the history are relevant, e.g., via a softmax
3. Construct a summary of the history, e.g., by weighted sum
4. Use summary and current hidden state to predict next token/word

13. Generative Transformers

⇒ Masked + Self-Attention

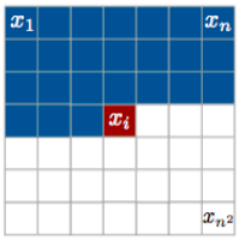
- Self-Attention을 통한 병렬화의 가능으로 Likelihood Evaluation이 빠르다 → 빠르게 Training 시킬 수 있다
- Masked ? : 그러나 다음 변수에 대한 distribution은 현재 + 이전 order까지의 변수들에 대해서만 condition으로 주어야 하기에 병렬화 시에 다음 그리고 그 이후의 변수가 attention 연산에 개입되지 않도록 masking이 필요함

  → Training시에 Masking + Self-Attention으로 인해 병렬적으로 모든 확률 변수에 대해(이전 스텝까지의 변수를 기반으로) likelihood estimation이 가능해짐

  → 그러나 Sampling(Generation)시에는 여전히 Sequential하게 Sampling 해야함.

Summary Till Now

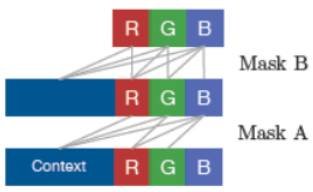
14. Pixel RNN



1. Model images pixel by pixel using raster scan order
2. Each pixel conditional $p(x_t \mid x_{1:t-1})$ needs to specify 3 colors

$$p(x_t \mid x_{1:t-1}) = p(x_t^{red} \mid x_{1:t-1})p(x_t^{green} \mid x_{1:t-1}, x_t^{red})p(x_t^{blue} \mid x_{1:t-1}, x_t^{red}, x_t^{green})$$

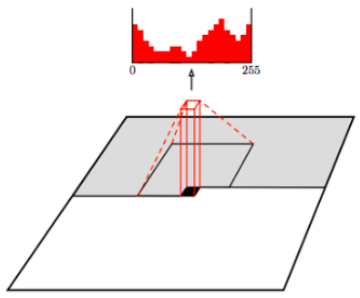
   and each conditional is a categorical random variable with 256 possible values
3. Conditionals modeled using RNN variants. LSTMs + masking (like MADE)



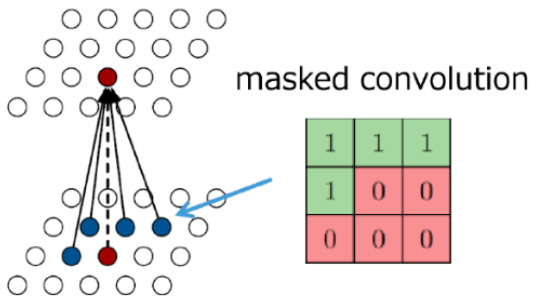
Results on downsampled ImageNet. Very slow: sequential likelihood evaluation.

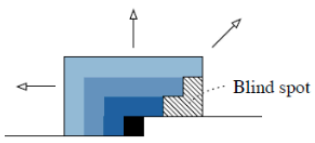
Pixel RNN에서는 P(x1,...., xn^2)을 RNN의 Variant인 LSTM을 통해 autoregressive 모델링(raster scan order). 각 확률 변수의 값은 0-255의 값이므로 Factorize된 각각의 분포는 모두 discrete probability distribution임(0,...,255)
→ 아 그럼 모델은 매 pixel distribution을 정의하기 위해 256 dim의 softmax normalized vector를 뽑아냄을 알 수 있음.
→ 그림과 같이 training시에(likelihood evaluation시에) 이전 order까지의 픽셀까지만을 기반으로 auto regressive하게 다음 픽셀값에 대한 likelihood를 출력할 수 있게함.
→ Pixel RNN의 변수 ordering은 raster scan order를 따른다고 함 (위에서 아래, 왼쪽에서 오른쪽)
→ 하나의 픽셀 안에도 3-channel이기 때문에 R → G → B의 ordering을 가정하여 위와와 같은 식이 나옴
→ Pixel RNN의 경우 RNN의 구조를 따르기에 훈련 과정이 Sequential하게 일어남.

15. PixelCNN



Idea: Use convolutional architecture to predict next pixel given context (a neighborhood of pixels).
Challenge: Has to be autoregressive. Masked convolutions preserve raster scan order. Additional masking for colors order.





→ PixelCNN은 각 픽셀에 대한 distribution값을 CNN 연산을 통해 한번에 뽑아낼 수 있게함 (Training Time Likelihood Estimation Parallazation이 가능해짐)

→ 물론 일반적인 convolution이 아닌 auto-regressive 성질을 유지하기 위해 masked convolution을 이용함 (pixel, pixel 안에서도 RGB ordering을 고려한 masked convolution)

→ raster scan order는 optimal한 variable ordering이 아닐 수도 있음

→ Pixel RNN보다 빠르면서도 비슷한 성능을 보임

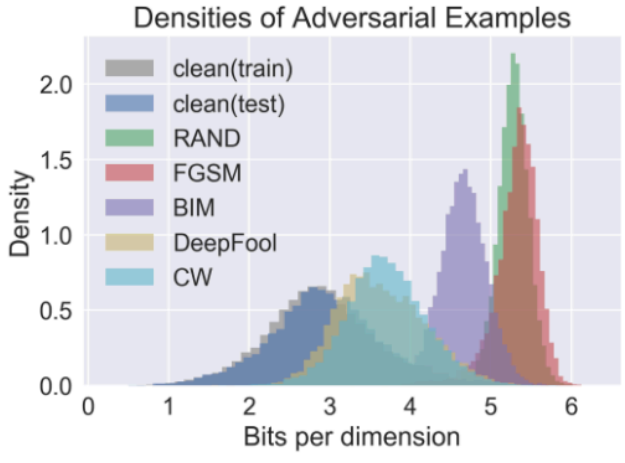
16. Generative Models for anomaly detection : An Adversarial Sample

잘 훈련된 Generative Model이라면 Adversarial Attack을 잘 찾아낼 수 있음(Anomaly Detection). 잘 훈련된 분포로부터 낮은 likelihood를 가졌을 때, 이를 anomaly라고 판단하여 우리 눈으로는 식별하지 못하는 adversarial attack을 감지할 수 있음.

Adversarial Sample : 모델을 속이기 위해 입력에 아주 작은 변경을 가한 샘플 → 원본 이미지에 아주 작은 노이즈를 추가해서 모델을 헷갈리게 만들기 위함

PixelCNN과 같은 방식으로 joint distribution을 모델링한다면 단순히 sampling을 넘어 + 해당 분포에서 추출된 만한 샘플인지 아닌지를 구별하는 task(anomaly detection)도 가능(density estimation)

이 모든 과정(sampling , anomaly detection 등등)을 잘하기 위해서는 이미지의 semantic한 정보를 알아내는 것에 더해 실제 조건 간의 관계(픽셀간의 관계)도 잘 학습해야함 !
(P(xi | xi-1,xi-2...x1)도 결국에 neural network가 functional하게 표현하는 것이니 이 neural network가 어떻게 디자인 및 학습되냐에 따라 조건 간의 관계를 얼마나 잘 반영하는지가 달라질 수 있음.



- Train a generative model $p(x)$ on clean inputs (PixelCNN)
- Given a new input $\overline{x}$, evaluate $p(\overline{x})$
- Adversarial examples are significantly less likely under $p(x)$

위 사진과 같이 일반적인 사진에 대해 training된 분포에 비해
FGSM, CW와 같은 adversarial attack이 적용된
사진들은 더 scale된 범위에서 분포를 이루기에 Adversarial Attack Sample들은 일반 training 분포에서는 낮은 likelihood값을 가질 것임.

17. Summary of Autoregressive Models

Easy to sample

훈련된 모델에 대해 P(x0)에서 하나 추출 : x0_bar

P(x1|x0=x0_bar)에서 하나 추출 : x1_bar

P(x2|x0=x0_bar, x1=x1_bar)에서 하나 추출 : x2_bar

이런식으로 sequential하게 추출!

Easy for Density Estimation And Likelihood Evaluation

Training의 likelihood evaluation뿐만 아니라 Anomaly Detection에서의 density estimation도 역시 변수 ordering(데이터 생성 순서라고 생각하자)대로 모델에 넣고 현재 sample의 값에 대응하는 factorize된 density들을 곱하면 수행할 수 있음

예를들어 density estimation 을 하고 싶은 샘플이 (x1=1, x2=0, x3=1 ...)일 때 auto-regressive model이 모델링한 P(x1,x2,x3...)에서의 density를 구하는 방법은
모델이 학습과정에서 P(x1), P(x2|x1) ...P(xn | xn-1, xn-2 ....)을 학습했으니 이제 P(x1 = 1), P(x2=0|x1 = 1), P(x3=1|x2=0, x1=1)을 모두 구해 곱하면됨. P(x2=0|x1 = 1)의 경우 모델의 입력으로 x1=1을 넣었을 때 뽑아낸 softmax output에서 P(x2=0)의 값이 되겠다!

Auto-regressive modeling의 단점

- feature extraction이나 clustering과 같은 task에는 크게 유용하게 쓰이지 않음. 모델이 학습하는 방식이 순차적인 예측(각 변수에 대한 확률 분포 예측)에 초점을 맞추고 있기 때문에.