Lecture4 : Learning

Ki Pyo	완료
선택	Auto-Regressive Model MLE

1. Learning a generative model

지금까지는 실제 모집단 분포에 근사시키기 위한 분포를 어떻게 모델링할 것인지에 대해 배움.

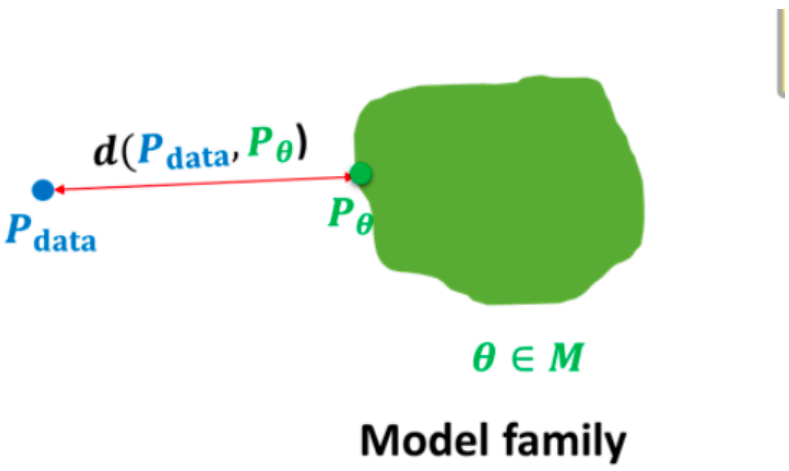
Recap)

- 확률변수 간의 Conditional Independency를 가정한 “Bayesian Network” → 가정으로 인해 Joint Probability를 구성하는 파라미터 수를 줄일 수 있음. Chain Rule로 분해되는 각각의 조건부 분포의 몇몇 조건을 배재 가능
- Chain Rule에 기반한 “Autoregressive Model with Neural Network”(이는 확률 변수간의 가정을 가져가지는 않고 Neural Network가 변수간의 관계를 학습함)→ 신경망의 파라미터로 Joint Probability에서 Factorize된 각각의 분포의 모양이, 정확하게는 파라미터가 결정됨. 확률 변수간의 가정(조건부 독립)은 하지 않으나 확률 변수의 ordering은 부여함(생성 순서)

이렇게 모집단에 근사하도록 학습이 된 Ptheta는 Sampling(Generation), Densitiy Estimation(Anomaly Detection), Unsupervised representation learning에 이용이 될 수 있다.

“이제는 이렇게 파라미터화한 Generative Model을 어떻게 실제 모집단과의 간극 = d(Pdata, Ptheta)을 줄이는 방향으로 파라미터를 학습시킬 것인가?에 대해 배울 것임”

2. Setting



1. Assume that the domain is governed by some underlying distribution Pdata → 어떤 domain이 모집단 Pdata에 의해 지배를 받는다고 가정
2. IID : Pdata로 부터 추출된 Dataset의 각각의 Sample들은 독립이며, 같은 Distribution에서 나왔다는 가정으로 시작 IID
3. 파라미터로 조절가능한 Distribution (BayesNet, FVSBN)으로 우리가 모델링한 후에 해당 Distribution을 실제 Pdata와 가장 가깝도록(근사하도록) 파라미터를 조정. 파라미터의 조정으로 생겨날 수 있는 가능한 모든 distribution의 집합을 M, model family.

3. What is Best?

학습을 목표로 하는 “Best Distribution”은 무엇일까?

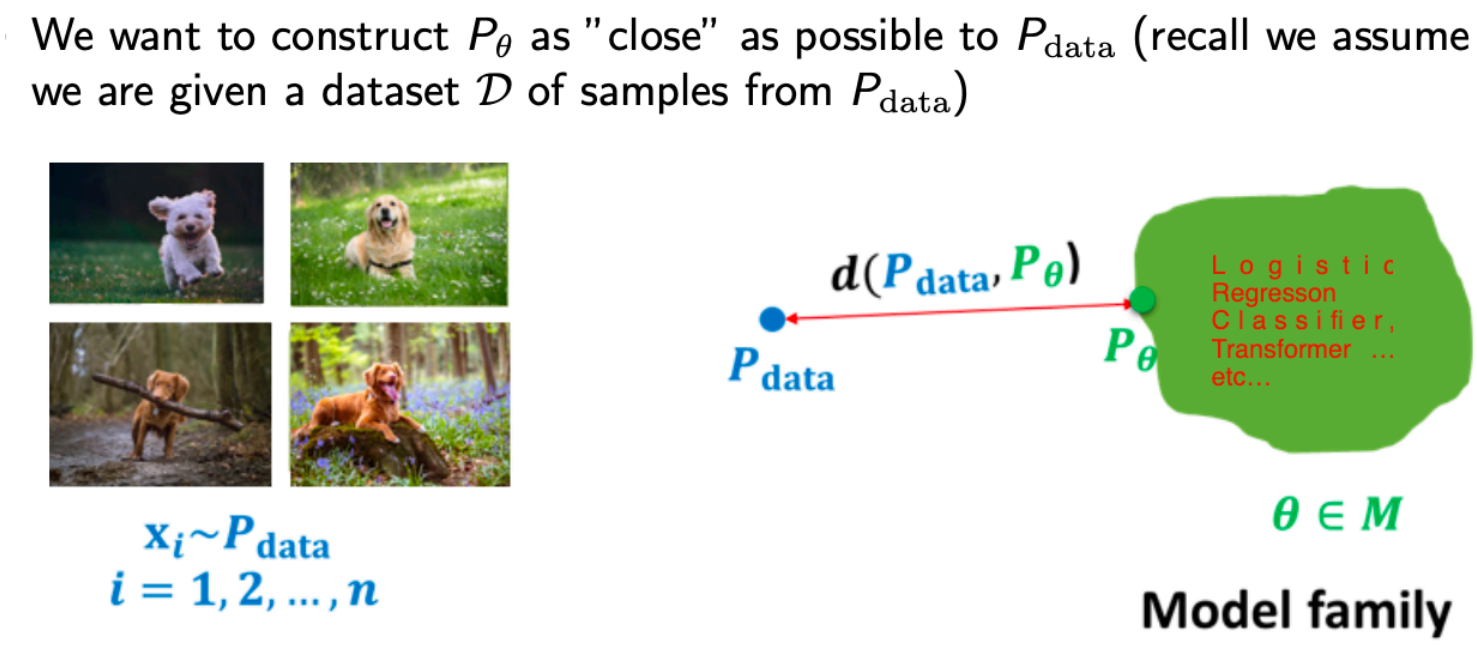
우리가 학습시킨 Generative Model를 통해 목표로 하는 Task가 무엇이냐에 따라 다름

- Density Estimation이라면 full joint probability를 다 구해야함
  => Distribution for sampling

- 이게이게 주어졌을 때 이게 뭐야를 예측하는 Prediction Task인경우 => P(xi | xi-1,...x1)과 같이 일부 조건부 분포만 학습시켜도 ok.
  ⇒ Distribution For Prediction
- Structure or knowledge discovery에 대한 것은 이 강의에서는 다루지 않음

4. Learning Problem as density estimation

- 각각 확률 변수에 대해 assign된 값에 대한 distribution의 density에 대한 regression task와 같은 것 : 얼마나 이 분포에서 나올 법해?를 예측하는 task를 목표로 할 때 !
- 확률변수에 대한 Full Joint Probability Distribution을 학습해야함
- 어떤 종류의 probabilistic inference도 해결할 수 있음 P(X,Y)를 구하면 P(Y|X)를 구할 수 있는 것처럼 (with bayes rule)



- 앞서 말했듯이 우리가 모델링한 파라미터로 조절가능한 분포가 실제 모집단 분포와 최대한 가깝도록 해야한다.
- 그럼 이 가까움 → d(Pdata, Ptheta)를 어떻게 evaluate 할까?
- d(Pdata, Ptheta)를 어떤 것으로 결정하느냐에 따라 우리의 generative model 종류가 달라진다.

5. KL-divergence

The **Kullback-Leibler divergence** (KL-divergence) between two distributions p and q is defined as

$$D(p\|q) = \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})}.$$

- 어떤 두 확률분포간의 차이를 나타내는 term
- 항상 양수이며 p = q일 때 KL값이 0이됨
- 항상 0 이상의 값을 가짐

$$\mathbf{E}_{\mathbf{x}\sim p}\left[-\log \frac{q(\mathbf{x})}{p(\mathbf{x})}\right] \geq -\log\left(\mathbf{E}_{\mathbf{x}\sim p}\left[\frac{q(\mathbf{x})}{p(\mathbf{x})}\right]\right) = -\log\left(\sum_{\mathbf{x}} p(\mathbf{x})\frac{q(\mathbf{x})}{p(\mathbf{x})}\right) = 0$$

저 뒤에 0 유도하려고 분모 분자 위치 바꾼듯

Jensen's Inequality

- 비대칭성을 가짐

Notice that KL-divergence is **asymmetric**, i.e., $D(p\|q) \neq D(q\|p)$

6. Detour on KL-divergence

→ KL Divergence가 의미하는 바 (정보 압축의 관점에서)

데이터 압축 시에 데이터의 분포에 따라 많이 나타나는 데이터는 적은 비트로 압축을 하고 적게 나타나는 데이터는 비교적 많은 비트로 압축을 하면 모두 동일한 비트로 압축을 하는 것보다 memory efficient함.

→ 모든 데이터에 대해 사용하는 비트 수(메모리 용량)의 기댓값이 줄어들음

ex) morse code

Like Morse code: $E = \bullet$, $A = \bullet-$, $Q = --\bullet-$

많이 쓰이는 E, A는 적은 비트 수로 비교적 적게 쓰이는 Q는 비교적 많은 비트 수로 !

KL-divergence: if your data comes from p, but you use a scheme optimized for q, the divergence $D_{KL}(p||q)$ is the number of *extra* bits you'll need on average

compression scheme

그러나 실제 데이터의 분포와 다르게 분포를 예측하고 이렇게 에측한 분포에 따라 실제 데이터를 압축하면, 실제 분포에 따라 비트 수를 조절하여 압축했을 때보다 메모리 낭비가 커지게 압축을 하게됨

=> 이 메모리 사용의 차이를 KL이 나타낸다고 볼 수 있음.

⇒ 정확하게는 위의 경우 DKL(p||q)이므로 p를 따르는 분포의 데이터를 우리가 예측한 q의 분포에 따라 압축을 했을 때, 실제 p에 따라 압축했을 때보다 얼마나 더 많은 비트수를 평균적으로 쓰게 될지(사용 비트 수의 기댓값 증가값)를 나타냄

7. Closeness as KL Divergence

$$\mathbf{D}(P_{\text{data}}||P_\theta) = \mathbf{E}_{\mathbf{x}\sim P_{\text{data}}}\left[\log\left(\frac{P_{\text{data}}(\mathbf{x})}{P_\theta(\mathbf{x})}\right)\right] = \sum_{\mathbf{x}} P_{\text{data}}(\mathbf{x}) \log \frac{P_{\text{data}}(\mathbf{x})}{P_\theta(\mathbf{x})}$$

→ Density Estimation을 위한 full joint probability를 예측한다고 했을 때, 실제 Pdata와의 closeness를 판단할 수 있는 기준은 KL Divergence가 될 수 있다.

→ 모집단과 우리가 학습한 Ptheta가 아예 똑같다면 KL은 0.

More On KL

$$\begin{aligned}\mathbf{D}(P_{\text{data}}||P_\theta) &= \mathbf{E}_{\mathbf{x}\sim P_{\text{data}}}\left[\log\left(\frac{P_{\text{data}}(\mathbf{x})}{P_\theta(\mathbf{x})}\right)\right] \\ &= \mathbf{E}_{\mathbf{x}\sim P_{\text{data}}}[\log P_{\text{data}}(\mathbf{x})] - \mathbf{E}_{\mathbf{x}\sim P_{\text{data}}}[\log P_\theta(\mathbf{x})]\end{aligned}$$

→ 첫번쨰 항은 모델의 파라미터에는 영향을 받지 않음

→ 두 항으로 쪼개면 다음과 같음

Then, *minimizing* KL divergence is equivalent to *maximizing* the **expected log-likelihood**

$$\arg\min_{P_\theta} \mathbf{D}(P_{\text{data}}||P_\theta) = \arg\min_{P_\theta} -\mathbf{E}_{\mathbf{x}\sim P_{\text{data}}}[\log P_\theta(\mathbf{x})] = \arg\max_{P_\theta} \mathbf{E}_{\mathbf{x}\sim P_{\text{data}}}[\log P_\theta(\mathbf{x})]$$

→ KL항을 최소화 하는 것은 뒤의 항을 최대화하는 것과 같음

Issue?

어쨌든 P data는 알지 못하는 상황이기에..

$$\mathbf{E}_{\mathbf{x}\sim P_{\text{data}}}[\log P_{\text{data}}(\mathbf{x})] - \mathbf{E}_{\mathbf{x}\sim P_{\text{data}}}[\log P_\theta(\mathbf{x})]$$

1. Although we can now compare models...
   어떤 파라미터로 정의된 모델이 더 p data에 가까운지는 알수 있겠지만(비교는 가능, 뒤의 argmax항이 어떤 모델이 더 크게 가져가는지 알면 비교가능함), 실제 p data와의 절대적인 거리는 알 수 없음.

Approximate the expected log-likelihood

$$\mathbf{E}_{\mathbf{x}\sim P_{\text{data}}}[\log P_\theta(\mathbf{x})]$$

with the *empirical log-likelihood*:

$$\mathbf{E}_{\mathcal{D}}[\log P_\theta(\mathbf{x})] = \frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}} \log P_\theta(\mathbf{x})$$

2. P data에 대한 정보를 알 수 없다. 그러나 P data에서 IID로 샘플링된 데이터셋은 가지고 있으니 기댓값에 대한 monte carlo approximation을 이용하여 위의 항을 근사시킨다. P data에 대한 sample 몇 개(Training Dataset)를 기반으로 logthetaP(x)에 대한 값을 모두 구해 더한 후, sample의 개수 |D|만큼 나누어 평균내면됨. D의 크기가 커질 수록 실제 이 항과 가까워 질 것임.

8. In Conclusion.. Maximum likelihood learning is then

$$\max_{P_\theta} \frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}} \log P_\theta(\mathbf{x})$$

→ 실제 모집단 분포와의 간극을 나타내는 criterion을 KL divergence로 나타내었을 때, KL항을 줄이는 것은 실제 모집단 분포에서 샘플링된 데이터(training data)들 각각에 대해 모델의 파라미터로 정의한 분포가 최대한의 likelihood를 부여해야하는 것과 같다.

로그 합이므로 로그 안에 곱으로 집어 넣으면..

Equivalently, maximize likelihood of the data
$P_\theta(\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}) = \prod_{\mathbf{x}\in\mathcal{D}} P_\theta(\mathbf{x})$

→ 다음과 같으며, 위의 식 앞에 log하나만 붙인 식을 최대화 시키는 것이 목표가 됨. (그럼 위위에 사진의 식과 같아지는 것임, |D|도 생략)

→ 앞의 상수 |D|는 theta에 대한 최대화 문제이기에 생략

→ 위의 항의 경우 sample 전체에 대한 likelihood(iid 가정으로 인해 전체에 대한 likelihood는 각 샘플에 대한 likelihood의 곱으로 나타낼 수 있음)

9. Monte Carlo Estimation

- **Express the quantity of interest as the expected value of a random variable.**

$$E_{x\sim P}[g(x)] = \sum_x g(x)P(x)$$

- Generate T samples $\mathbf{x}^1, \ldots, \mathbf{x}^T$ from the distribution P with respect to which the expectation was taken.
- Estimate the expected value from the samples using:

$$\hat{g}(\mathbf{x}^1, \cdots, \mathbf{x}^T) \triangleq \frac{1}{T}\sum_{t=1}^{T} g(\mathbf{x}^t)$$

where $\mathbf{x}^1, \ldots, \mathbf{x}^T$ are independent samples from P. Note: $\hat{g}$ is a random variable. Why? 샘플링되는 x1. ....xT가 일정 하지 않기 때문 => g_hat값은 분포를 이룰 것임

- Monte Carlo Estimation에 따라 g_hat의 값은 실제 기댓값에 approximation하기 위한 추정값임.
- 샘플링되는 x1. ....xT가 일정 하지 않기 때문 => g_hat값은 분포를 이룰 것임 ⇒ g_hat은 random_variable

10. Monte Carlo 추정값의 성질

- **Unbiased:**

$$E_P[\hat{g}] = E_P[g(x)]$$

- **Convergence:** By law of large numbers

$$\hat{g} = \frac{1}{T}\sum_{t=1}^{T} g(x^t) \to E_P[g(x)] \text{ for } T \to \infty$$

샘플의 크기 T가 커짐에 따라 추정값은 실제 기대값에 수렴함

- **Variance:**

$$V_P[\hat{g}] = V_P\left[\frac{1}{T}\sum_{t=1}^{T} g(x^t)\right] = \frac{V_P[g(x)]}{T}$$

1. Unbiased : Monte Carlo를 통한 추정값은 불편하다. 실제 기댓값에 대해 추정한 값에 대한 평균(샘플링을 여러 번하여 만든 추정값들에 대한 평균)은 실제 값과 같다.

<u>Supplement</u>

1. 큰 수의 법칙에 따라 샘플의 크기가 커질 수록 근사값은 실제 값에 가까워 진다
2. Variance : 추정치에 대한 분산은 샘플 수가 많아질 수록 작아진다(안정된 예측을 하게된다.)

→ Unbiased한 상황에서 T의 값이 클수록 더 실제값에 가까워질 확률이큼

11. Example of MLE Learning

Single variable example: A biased coin

- Two outcomes: *heads* (H) and *tails* (T)
- Data set: Tosses of the biased coin, e.g., $\mathcal{D} = \{H, H, T, H, T\}$
- Assumption: the process is controlled by a probability distribution $P_{\text{data}}(x)$ where $x \in \{H, T\}$
- Class of models $\mathcal{M}$: all probability distributions over $x \in \{H, T\}$.
- Example learning task: How should we choose $P_\theta(x)$ from $\mathcal{M}$ if 3 out of 5 tosses are heads in $\mathcal{D}$?

1. Assume that the domain is governed by some underlying distribution Pdata → 어떤 domain이 모집단 Pdata에 의해 지배를 받는다고 가정 → 이때 x ∈ {H, T}, Pdata가 어떤 분포인지는 아무도 모름(이 가정에서는 알 수도 있겠다는 생각...?)
2. M은 파라미터로 조절 가능한 모든 분포의 집합(이 예시에서는 베르누이 분포를 따른다고 가정) → Ptheta들의 집합

We represent our model: $P_\theta(x = H) = \theta$ and $P_\theta(x = T) = 1 - \theta$

3. D sample이 위와 같은 상황일 때 어떤 M에서 어떤 Ptheta를 선택해야 Pdata와 가장 가깝게 만들 수 있을까? → MLE Learning을 통해 결정

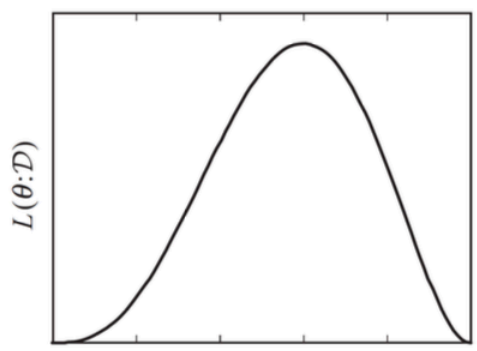
Example data: $\mathcal{D} = \{H, H, T, H, T\}$

Likelihood of data $= \prod_i P_\theta(x_i) = \theta \cdot \theta \cdot (1 - \theta) \cdot \theta \cdot (1 - \theta)$

위 항을 최대로 하는 theta를 찾아야함

→ 데이터셋 샘플에 대해 가장 큰 likelihood를 부여할 수 있도록

모델의 파라미터에 따라 해당 dataset 샘플에 대한 likelihood값이 달라지고 이를 최대화하는 theta를 찾아야함



지금은 이 theta가 하나의 값이지만 neural network에서는 이 theta가 신경망의 파라미터들이 될 것임

- Optimize for θ which makes $\mathcal{D}$ most likely. What is the solution

→여기서는 closed form으로 theta가 구해졌지만 closed form으로 구해지지 않는 경우도 많음. 다음 auto-regressive model의 예시와 같음

12.MLE Learning in Autoregressive Models

$$P_\theta(\mathbf{x}) = \prod_{i=1}^{n} p_{\text{neural}}(x_i | \mathbf{x}_{<i}; \theta_i)$$

- Auto-Regressive model에서 특정 하나의 데이터 포인트 X에 대한 likelihood는 chain-rule로 구할 수 있음. (n은 factorize된 분포의 개수)

$\theta = (\theta_1, \cdots, \theta_n)$ are the parameters of all the conditionals. Training data $\mathcal{D} = \{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}\}$. Maximum likelihood estimate of the parameters θ?

- Decomposition of Likelihood function

$$L(\theta, \mathcal{D}) = \prod_{j=1}^{m} P_\theta(\mathbf{x}^{(j)}) = \prod_{j=1}^{m} \prod_{i=1}^{n} p_{\text{neural}}(x_i^{(j)} | \mathbf{x}_{<i}^{(j)}; \theta_i)$$

- Goal : maximize $\arg\max_\theta L(\theta, \mathcal{D}) = \arg\max_\theta \log L(\theta, \mathcal{D})$
- We no longer have a closed form solution

- 이 데이터 들의 모임인 D에 대한 likelihood는 각 데이터 포인트들에 대한 likelihood를 곱하면됨.
- theta 1,..n은 모두 factorize된 각각 분포에서의 파라미터 값이다. ⇒ 일반적으로 autoregressive model에서는 weight sharing 됨.
- 이 값을 maximize 시키는 것이 실제 Pdata 분포와의 KL을 minimize하는 것과 같다.
- 신경망 기반의 Auto-Regressive Model의 경우 Likelihood를 maximize시키는 parameter를 closed form으로 구할 수 없음 → Gradient Descent 이용 하기.

"Log Likelihood가 나온 이유는 → 모집단과 Ptheta의 KL항에서 나왔기 때문이라고 생각함"

앞에 log를 붙여서 합으로 바꾸면?

$$L(\theta, \mathcal{D}) = \prod_{j=1}^{m} P_\theta(\mathbf{x}^{(j)}) = \prod_{j=1}^{m} \prod_{i=1}^{n} p_{\text{neural}}(x_i^{(j)} | \mathbf{x}_{<i}^{(j)}; \theta_i)$$

Goal : maximize $\arg\max_\theta L(\theta, \mathcal{D}) = \arg\max_\theta \log L(\theta, \mathcal{D})$

$$\ell(\theta) = \log L(\theta, \mathcal{D}) = \sum_{j=1}^{m} \sum_{i=1}^{n} \log p_{\text{neural}}(x_i^{(j)} | \mathbf{x}_{<i}^{(j)}; \theta_i)$$

across variables

across datapoints

classfier의 output값 => softmax normalized vector

→ gradient-ascent 기법으로 step-by-step으로 파라미터 업데이트 (descent이려면 objective function의 부호가 바뀌어야한다고 생각함)

→ Non-convex optimization problem이지만, practical하게는 잘 작동함

13. MLE Learning with SGD

$$\ell(\theta) = \log L(\theta, \mathcal{D}) = \sum_{j=1}^{m} \sum_{i=1}^{n} \log p_{\text{neural}}(x_i^{(j)} | \mathbf{x}_{<i}^{(j)}; \theta_i)$$

1. Initialize θ^0 at random
2. Compute $\nabla_\theta \ell(\theta)$ (by back propagation)
3. $\theta^{t+1} = \theta^t + \alpha_t \nabla_\theta \ell(\theta)$

MLE 기반 Auto-Regressive Model Training 방법

1. 파라미터 초기화
2. Dataset Likelihood에 대한 파라미터 gradient 계산

3. Gradient Ascent ⇒ Likelihood를 최대화 해야하기 때문 (Descent인 경우 Objectie function 부호 바뀌기 => loss function)

What is the gradient with respect to θ_i?

$$\nabla_{\theta_i}\ell(\theta) = \sum_{j=1}^{m} \nabla_{\theta_i} \sum_{i=1}^{n} \log p_{\text{neural}}(x_i^{(j)}|\mathbf{x}_{<i}^{(j)};\theta_i) = \sum_{j=1}^{m} \nabla_{\theta_i} \log p_{\text{neural}}(x_i^{(j)}|\mathbf{x}_{<i}^{(j)};\theta_i)$$

Each conditional $p_{\text{neural}}(x_i|\mathbf{x}_{<i};\theta_i)$ can be optimized separately if there is no parameter sharing. In practice, parameters θ_i are shared (e.g., NADE, PixelRNN, PixelCNN, etc.)

→ i 번째 factorization 분포의 likelihood에 대한 i번째 파라미터 gradient

Stocastic(Mini-Batch Gradient Descent) Gradient Update 기반의 update

$$\ell(\theta) = \log L(\theta, \mathcal{D}) = \sum_{j=1}^{m} \sum_{i=1}^{n} \log p_{\text{neural}}(x_i^{(j)}|\mathbf{x}_{<i}^{(j)};\theta_i)$$

1. Initialize θ^0 at random
2. Compute $\nabla_\theta \ell(\theta)$ (by back propagation)
3. $\theta^{t+1} = \theta^t + \alpha_t \nabla_\theta \ell(\theta)$

$$\nabla_\theta \ell(\theta) = \sum_{j=1}^{m} \sum_{i=1}^{n} \nabla_\theta \log p_{\text{neural}}(x_i^{(j)}|\mathbf{x}_{<i}^{(j)};\theta_i)$$

What if $m = |\mathcal{D}|$ is huge?

$$\begin{aligned} \nabla_\theta \ell(\theta) &= m \sum_{j=1}^{m} \frac{1}{m} \sum_{i=1}^{n} \nabla_\theta \log p_{\text{neural}}(x_i^{(j)}|\mathbf{x}_{<i}^{(j)};\theta_i) \\ &= m E_{x^{(j)} \sim \mathcal{D}} \left[\sum_{i=1}^{n} \nabla_\theta \log p_{\text{neural}}(x_i^{(j)}|\mathbf{x}_{<i}^{(j)};\theta_i) \right] \end{aligned}$$

Monte Carlo: Sample $x^{(j)} \sim \mathcal{D}; \nabla_\theta \ell(\theta) \approx m \sum_{i=1}^{n} \nabla_\theta \log p_{\text{neural}}(x_i^{(j)}|\mathbf{x}_{<i}^{(j)};\theta_i)$

→ 한번에 모든 데이터셋에 대해 gradient를 계산하고 업데이트를 진행하는 것은 데이터셋이 클 경우에 무리

메번 파라미터 업데이트를 진행할 때마다 데이터셋의 subset만을 이용하여서 gradient를 계산하고 업데이트 한다.

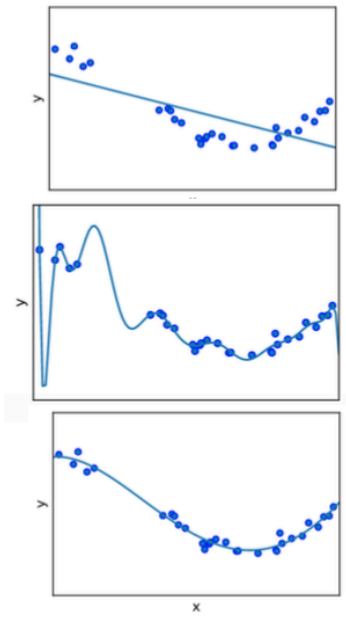
→ 이는 uniform distribution 기반의 monte carlo estimation으로부터 출발한것인데(양쪽에 m과 1/m을 곱하기) 전체 데이터셋에 대한 gradient를 |D| * Expectation of Uniform Distribution으로 표현하고 이때 Expectation of Uniform Distribution을 monte carlo를 통해 근사하려함. monte carlo 시에 샘플이 n개 라면 n-mini batch stochastic update가 되겠다.

→ 전체 데이터셋에 대한 gradient를 monte carlo approximation을 통해 일부 미니 배치에 대한 gradient로 근사 시키려함. (Mini Batch SGD)

→ 위 그림의 마지막 식의 경우 하나의 샘플에 대해(one-minibatch) 각각 factorize된 분포의 likelihood의 parameter gradient를 모두 더하고 m을 곱함. (원래는 전체 데이터 샘플 들에 대해 각각 factorize된 분포의 likelihood의 parameter gradient를 모두 더하고 (sigma n) 이를 모두 더해야(sigma m) D에 대한 log likelihood gradient가 나오는 것임_

14. Overfitting?

- There is an inherent **bias-variance trade off** when selecting the hypothesis class. Error in learning due to both things: bias and variance.
- Hypothesis space: linear relationship
  - Does it fit well? Underfits
- Hypothesis space: high degree polynomial
  - Overfits
- Hypothesis space: low degree polynomial
  - Right tradeoff



Unseen data에 대해 얼마나 좋은 성능을 내는지에 대한 generalization 성능이 좋아야함
=> 모델에 대한 적절한 규제 필요

Bais-Variance Trade-Off

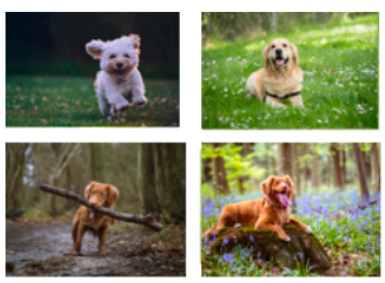
1. make model capacity limited ⇒ underfitting 초래함, bias 증가
2. make model capacity too flexible? ⇒ overfitting, variance 증가

→ find the sweet spot!

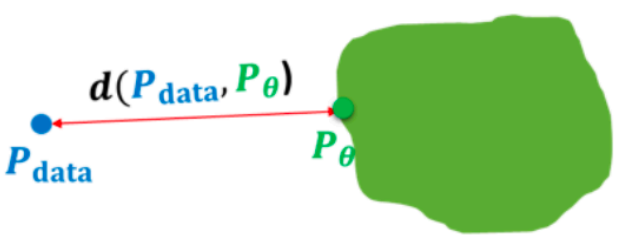
15. How to avoid overfitting?

Hard constraints, e.g. by selecting a less expressive model family:

- Smaller neural networks with less parameters
- Weight sharing



$\mathbf{x}_i \sim P_{\text{data}}$
$i = 1, 2, \ldots, n$



$\theta \in M$

Model family

Soft preference for "simpler" models: **Occam Razor**.

Augment the objective function with **regularization**:

$$objective(\mathbf{x}, \mathcal{M}) = loss(\mathbf{x}, \mathcal{M}) + R(\mathcal{M})$$

Evaluate generalization performance on a held-out validation set

1. Parameter의 수 줄이기 => Weight Sharing을 통해
   ex) Auto-Regressive model에서 factorize된 각각의 확률 분포에 대한 parameter출력하는 데에 이용되는 신경망의 파라미터를 sharing하도록하자.
2. Objective Function이 Overfitting 방지를 위한 Regularization Term을 포함하도록 한다
3. Training 과정 중 Vaildation Set을 도입하여 Generalization 성능을 monitoring한다.