ESC 2024 Winter Session 6th Week

Probabilistic Neural Network 2

장덕재, 유채원, 이상윤, 황보유민

Introduction

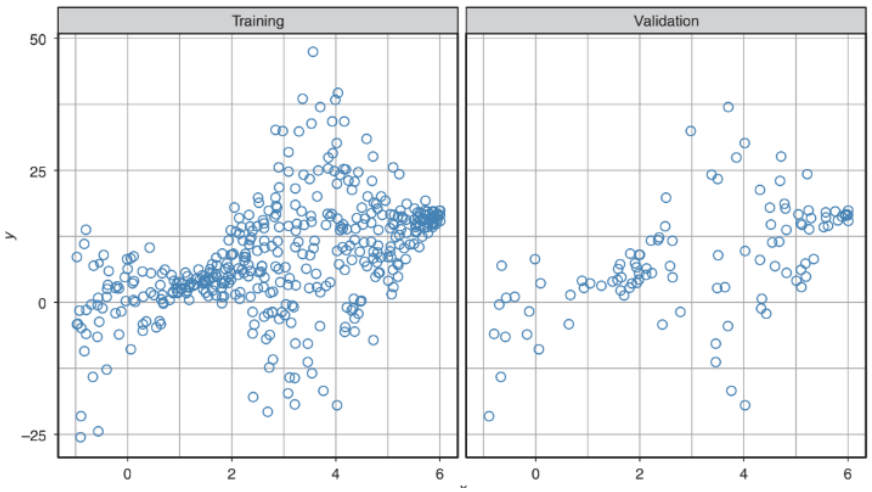
확률적 심층 학습 모델(Probabilistic Deep Learning; PDL)의 구축 원리는 다음과 같다. 첫째, 출력 데이터(Outcome) 혹은 타겟 변수(Target variable) Y에 잘 들어맞는 확률분포를 모델로 선택한다. 둘째, 인공 신경망(Neural net)의 출력 노드(Output node)의 수를 모델이 가지고 있는 모수(Parameter)의 수와 일치시킨다. 셋째, 선택한 분포로부터 음의 로그가능도함수(Negative-log-likelihood; NLL)를 손실함수(Loss function)로 삼아 모델을 훈련시킨다.

1 Evaluating & Comparing Probabilistic Prediction Models

확률적 예측 모델(Probabilistic prediction model)의 목적은 새로운 데이터를 입력받았을 때, 그에 대해 정확한 확률적 예측을 수행하는 데 있다. 그러므로 특정 모델 A가 모델의 학습에 사용되지 않은 새로운 테스트 데이터(Test data)에 대해 모델 B보다 상대적으로 더 낮은 음의 로그가능도함수(Negative-log-likelihood; NLL) 값을 보였다면, A보다 B가 더 성능이 좋다고 말할 수 있다.

2 Fitting & Evaluating a Linear Regression Model

다음과 같은 데이터 세트(Dataset)에 선형회귀분석(Linear regression)를 수행하고자 한다. 왼편은 훈련 데이터 세트(Training dataset)이고, 오른편은 유효성 검사 데이터 세트(Validation data set)이다.

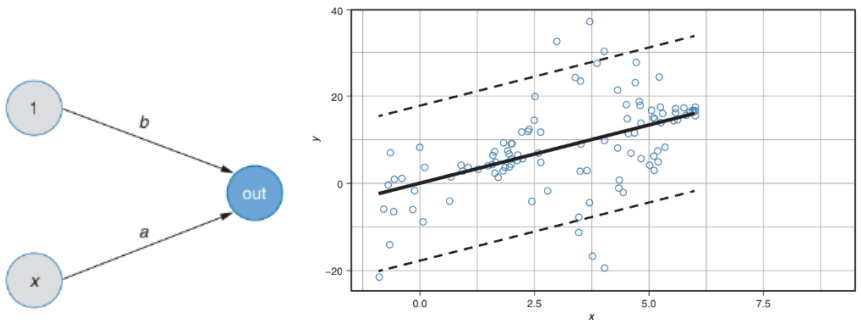


2.1 (Model A) With Given Standard Deviation

여기에서 우리가 활용할 확률 모델은 분산이 고정된 정규분포(Normal distribution with constant variance)로, 수식으로 표현하면 다음과 같다.

$$Y \mid X \sim N(\mu_x = ax + b, \sigma^2 = 1)$$

수행한 선형회귀분석을 단층 퍼셉트론(Single layer perceptron), 혹은 한 개의 뉴런(Neuron)으로 구성된 네트워크(Network)의 형태로 도해하면 왼쪽 아래와 같다. 추정할 확률변수의 모수(Parameter)가 μ_x로 하나이니, 출력 노드(Output node)도 하나인 것을 확인할 수 있다. 오른쪽 아래의 파선은 유효성 검사 데이터 세트(Validation data set)에 대한 95% 예측 구간(Prediction interval)으로, 다시 말해 학습된 정규분포 누적분포함수(Cumulative distribution function ; CDF)의 0.025와 0.975에 해당하는 y의 지점을 모든 x의 범위에 대하여 연속하여 이어놓은 것이다.

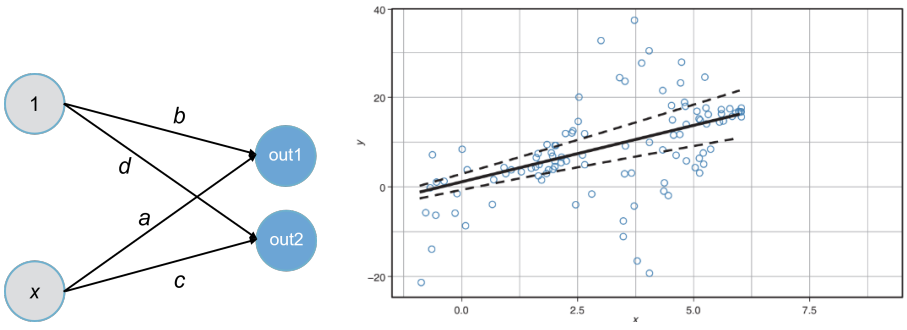


2.2 (Model B) With Learned, Monotonous Standard Deviation

여기에서 우리가 활용할 확률 모델은 분산이 x에 대한 간단한 함수인 정규분포(Normal distribution)로, 수식으로 표현하면 다음과 같다.

$$Y \mid X \sim N(\mu_x = \mathbf{out1} = ax + b, \sigma_x^2), \;\; \sigma_x = \exp(\mathbf{out2}) = \exp(cx + d)$$

수행한 선형회귀분석과 표준편차 분석을 네트워크(Network)의 형태로 도해하면 왼쪽 아래와 같다. 추정할 확률변수의 모수(Parameter)가 μ_x와 σ_x로 두 개니, 출력 노드(Output node)도 두 개인 것을 확인할 수 있다. 여기서 두 번째 노드(**out2**)의 활성화 함수(Activation function)로 지수함수(Exponential function)을 차용한 이유는 표준편차(Standard deviation)를 항상 0보다 크도록 만들어야 하기 때문이다. 오른쪽 아래의 두 파선은 유효성 검사 데이터 세트(Validation data set)에 그린 95% 예측 구간(Prediction interval)이다.

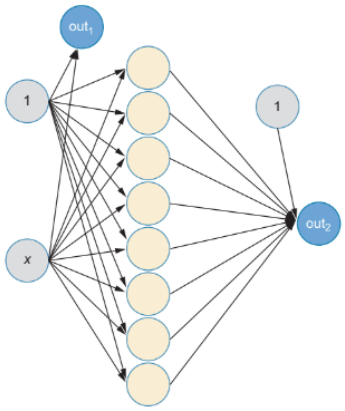


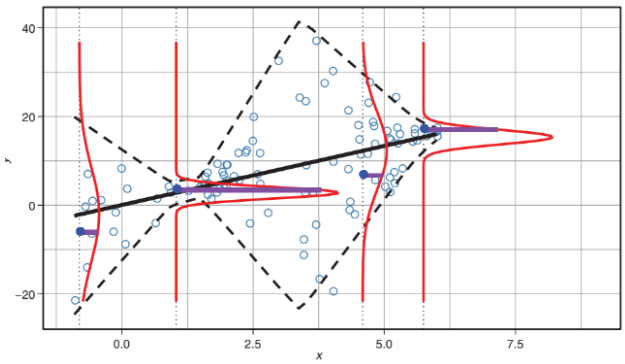
2.3 (Model C) With Learned, Flexible Standard Deviation

여기에서 우리가 활용할 확률 모델은 분산이 x에 대한 복잡한 함수인 정규분포(Normal distribution)로, 수식으로 표현하면 다음과 같다.

$$Y \mid X \sim N(\mu_x = \textbf{out1} = ax + b, \sigma_x^2), \;\; \sigma_x = \exp(\textbf{out2})$$

수행한 선형회귀분석과 표준편차 분석을 심층 신경망(Deep neural network ; DNN)의 형태로 도해하면 왼쪽 아래와 같다. 오른쪽 아래의 두 파선은 유효성 검사 데이터 세트(Validation data set)에 그린 95% 예측 구간(Prediction interval)이다. 분산 함수 σ_x^2 가 Model 2에 비해 용량이 커지고 유연해졌으므로, x 값의 변화에 감응하여 부드럽게 파선이 유동하는 것을 확인할 수 있다.





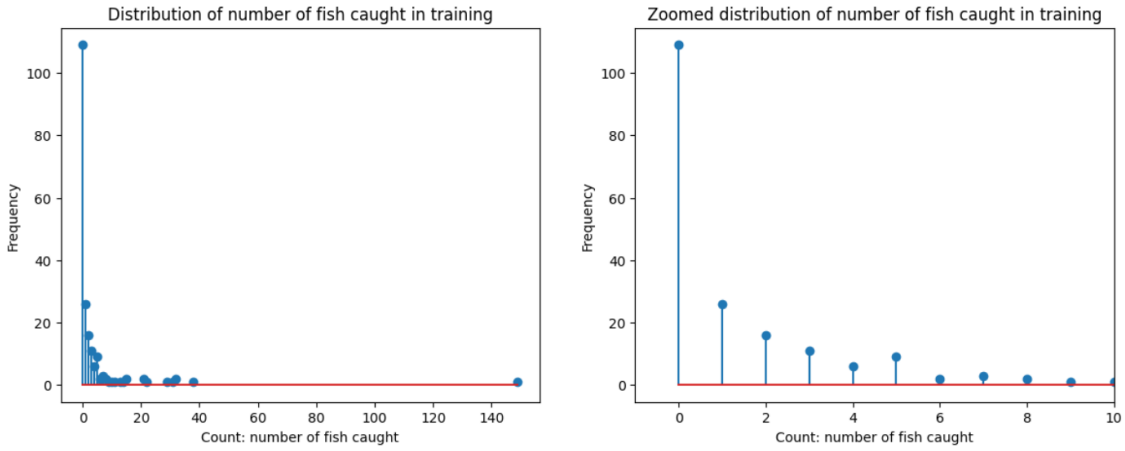
2.4 Comparing Three Models with Negative-Log-Likelihood Criteria

유효성 검사 데이터 세트(Validation data set)를 활용해 세 모델의 성능을 평가하면 다음과 같다. 정량적 관점에서 보았을 때, Model C의 성능이 제일 우수하다는 것을 확인할 수 있다.

$$\begin{aligned} \text{NLL(Model A)} &= 3.53 \\ \text{NLL(Model B)} &= 3.55 \\ \text{NLL(Model C)} &= 3.15 \end{aligned}$$

3 Fitting & Evaluating a Poisson Regression Model

다음과 같은 데이터 세트(Dataset)를 분석하고자 한다. 왼편과 오른편 모두 훈련 데이터 세트(Training dataset) 중 타겟 변수(Target variable) y만 골라내어 막대 그래프로 표현한 것이다. 여기서 y는 가산 자료(Count data)이다.

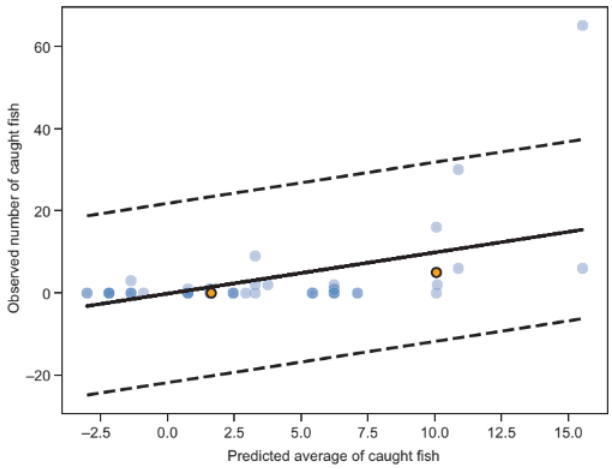


3.1 (Model A) Multiple Linear Regression for Count Data

여기에서 우리가 활용할 확률 모델은 분산이 고정된 정규분포(Normal distribution with constant variance)로, 수식으로 표현하면 다음과 같다.

$$Y \mid \mathbf{X} \sim N(\mu_{\mathbf{x}} = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b, \sigma^2_{\mathbf{x}}),\ \sigma^2_{\mathbf{x}} = \mathrm{MSE}$$

다중선형회귀분석(Multiple linear regression)을 수행한 뒤, 검사 데이터 세트(Test data set) 위에 95% 예측 구간(Prediction interval)을 파선 형태로 그리면 다음과 같다. 예측 구간이 음수 영역에 걸쳐 있는 것으로 보아, 0 이상의 정수(integer)만 가질 수 있는 가산 자료인 타겟 변수 y의 특성을 충분히 고려하지 못한 모델이라는 결론을 내릴 수 있다.

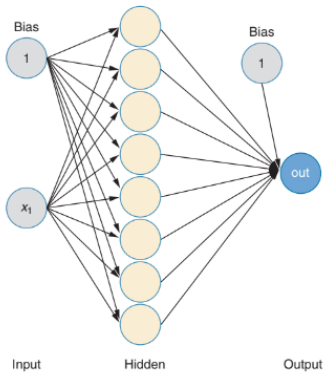


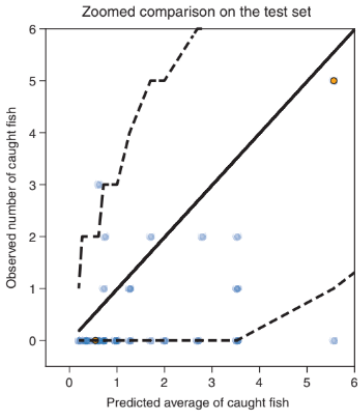
3.2 (Model B) Poisson Regression for Count Data

여기에서 우리가 활용할 확률 모델은 포아송 분포(Poisson distribution)로, 수식으로 표현하면 다음과 같다. 노드(**out**)의 활성화 함수(Activation function)로 지수함수(Exponential function)을 차용한 이유는 $\lambda_{\mathbf{x}}$를 항상 0보다 크도록 만들어야 하기 때문이다.

$$Y \mid \mathbf{X} \sim \text{Poisson}(\lambda_{\mathbf{x}} = \exp(\mathbf{out}))$$

포아송 회귀분석(Poisson regression)을 수행한 뒤, 검사 데이터 세트(Test data set) 위에 95% 예측 구간(Prediction interval)을 파선 형태로 그리면 오른쪽 아래와 같다. 왼쪽 아래는 심층 신경망(Deep Neural Net ; DNN) 모델을 도해한 것이다.



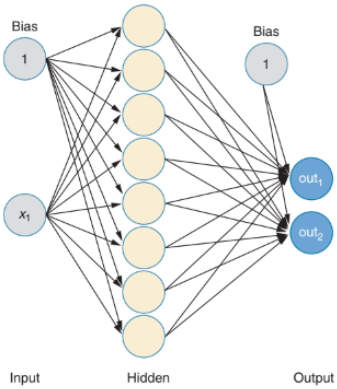


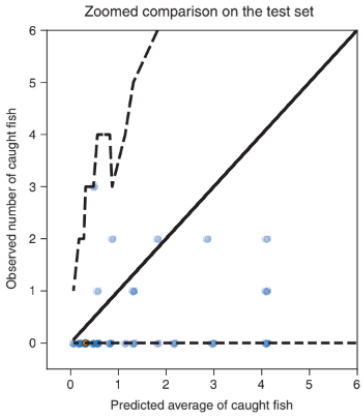
3.3 (Model C) ZIP(Zero-Inflated Poisson) Regression for Count Data

여기에서 우리가 활용할 확률 모델은 영과잉-포아송 분포(Zero-Inflated Poisson distribution)로, 수식으로 표현하면 다음과 같다. 첫 번째 노드(**out1**)의 활성화 함수(Activation function)로 시그모이드 함수(Sigmoid function)을 차용한 이유는 $\pi_{\mathbf{x}}$를 항상 0과 1 사이의 값으로 조정해야 하기 때문이다.

$$Y \mid \mathbf{X} \sim \text{ZIP}(\pi_{\mathbf{x}} = \sigma(\mathbf{out1}),\ \lambda_{\mathbf{x}} = \exp(\mathbf{out2}))$$

영과잉-포아송 회귀분석(Zero-inflated Poisson regression)을 수행한 뒤, 검사 데이터 세트(Test data set) 위에 95% 예측 구간(Prediction interval)을 파선 형태로 그리면 오른쪽 아래와 같다. 왼쪽 아래는 심층 신경망(Deep Neural Net ; DNN) 모델을 도해한 것이다.





3.4 Comparing Three Models with Various Criteria

검사 데이터 세트(Test data set)를 활용해 세 모델의 성능을 평가하면 다음과 같다. 세 평가 기준은 각각 음의 로그가능도(Negative-log-likelihood ; NLL), 평균 제곱근오차(Root Mean Square Error ; RMSE), 그리고 평균 절대오차(Mean Absolute Error ; MAE)이다. 정량적 관점에서 보았을 때, Model B와 C가 A보다 우수하다는 것을 확인할 수 있다.

$$\begin{array}{lll} \text{NLL(Model A)} = 3.6 & \text{RMSE(Model A)} = 8.6 & \text{MAE(Model A)} = 4.7 \\ \text{NLL(Model B)} = 2.7 & \text{RMSE(Model B)} = 7.2 & \text{MAE(Model B)} = 3.1 \\ \text{NLL(Model C)} = 2.2 & \text{RMSE(Model C)} = 7.3 & \text{MAE(Model C)} = 3.2 \end{array}$$

4 Probabilistic DL Models in the Wild

앞서서는 Gaussian, Poisson, ZIP 등의 분포를 Conditional Probability Distribution(CPD)로 사용하는 것을 보았다. 하지만 실세계의 분포는 훨씬 더 복잡한 형태를 지니고 있어서 단순한 분포로는 설명이 힘든 경우가 많다. 따라서 이제부터 복잡한 분포를 설명할 수 있는 flexible probability distribution들을 살펴볼 것이다.

4.1 Multinomial Distribution

일반적으로 분포가 몇 개의 파라미터를 가지고 있냐에 따라서 그 분포의 flexibility를 확인할 수 있다. 당연히 파라미터의 수가 많을수록 분포를 더 flexible할 것이다. 그러한 맥락에서 다항 분포는 매우 유연한 분포라고 할 수 있다. 데이터가 가질 수 있는 클래스의 수만큼 파라미터의 개수를 설정한다면 데이터를 완벽하게 설명할 수 있기 때문이다. 예를 들어서 MNIST 데이터셋에서 이미지 데이터 X가 주어졌을 때, Label Y를 다음과 같이 설정할 수 있다

$$Y|X \ Multinomial(p_0, p_1, ..., p_9)$$

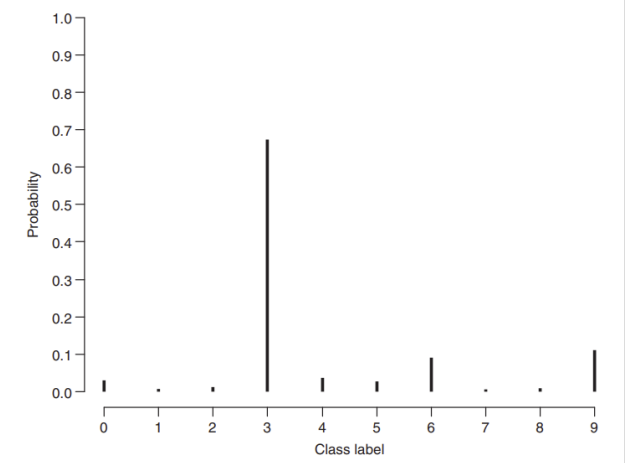


Figure 6.1 Multinomial distribution with ten classes: $MN(p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9)$

초기의 WaveNet이나 PixelCNN과 같은 모델들은 다항 분포를 CPD로 사용했다. WaveNet은 구글에서 만든 생성 모델로, 텍스트가 주어졌을 때 음성을 생성한다. PixelCNN은 OpenAI에서 만든 모델로,

이전 펙셀들을 바탕으로 다음 픽셀을 예측한다. 참고로 WaveNet과 PixelCNN 모두 앞서 보았던 Y|X와 같은 CPD를 사용하는 것이 아니라 자기회귀적인(Autoregressive) CPD를 사용한다.

$$P(x_t|x_{t-1}, ..., x_0)\ Multinomial$$

사실 PixelCNN을 예시로 들자면, 하나의 픽셀이 가질 수 있는 값은 매우 많다(8 bit 이미지의 경우 0 255). 사진의 해상도가 높을수록 픽셀이 가질 수 있는 값은 더 많아질 것이다. 8 bit일 경우, 다항 분포를 쓰게 된다면 총 255개의 파라미터가 필요하게 된다(256개가 아닌 이유는 다항 분포의 파라미터의 합이 1이 되어야 하기 때문). 그런데 파라미터의 수가 많을수록 학습 비용이 더 커지기 때문에 비효율적이다. 그래서 다항 분포의 대안으로 나온 것이 Discretized Logistic Mixture Distribution이다. 물론 이 분포는 다항 분포에 비해 미세하게 flexibility는 떨어질 수 있겠지만 파라미터의 수는 대폭 줄일 수 있다.

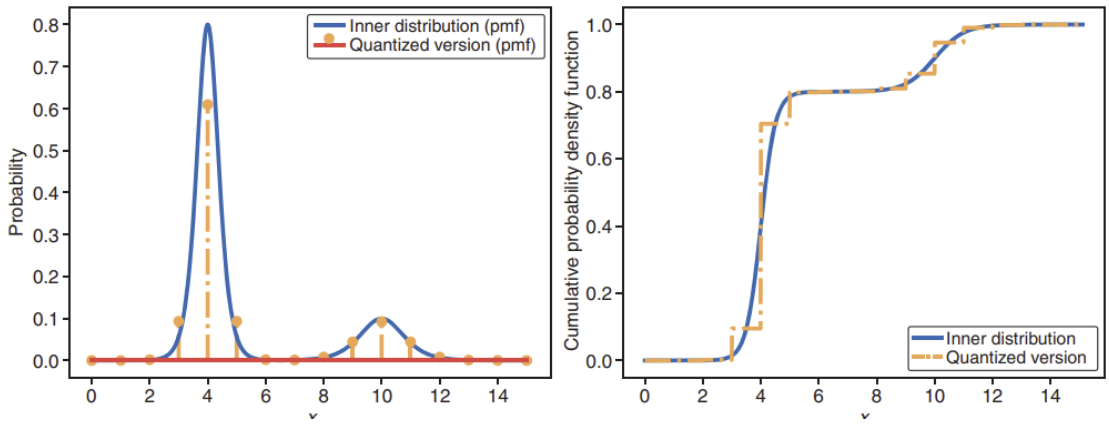
4.2 Discretized Logistic Mixture

일반적으로 복잡한 분포를 나타내기 위해서 사용하는 것은 Gaussian Mixture이다. Logistic 분포 또한 Gaussian 분포와 유사하게 종 모양의 형태를 띄고 있어서 Gaussian의 대안으로 사용되기도 한다. 이때 굳이 Gaussian을 쓰지 않고 Logistic을 쓰는 이유는 여기서 다루는 것이 그냥 mixture distribution이 아니라 'discretized' mixture distribution이기 때문이다. 연속 분포를 이산 분포로 만드는 과정에서 분포의 CDF를 계산하는 것이 필요한데, Gaussian의 경우 CDF를 구하는 것이 어려운 반면, Logistic의 경우 CDF를 쉽게 구할 수 있다.

$$Logistic\,PDF : f(x) = \frac{e^{-\frac{x-\mu}{s}}}{s\left(1 + e^{-\frac{x-\mu}{s}}\right)^2}$$

$$Logistic\,CDF : F(x) = \frac{1}{1 + e^{-\frac{x-\mu}{s}}}$$

Logistic 분포를 Gaussian과 같이 location, scale 파라미터를 지닌다. 그리고 여러 분포를 섞기 위해선 각 분포마다 어느 정도의 가중치를 줄 지도 결정해야 한다. 즉, weight 파라미터가 추가로 필요하게 되는 것이다. 따라서 만약 2개의 Logistic 분포를 섞는다면 2*3=6개의 파라미터를 설정할 필요가 있다. 아래의 그림은 location = 4, scale = 0.25, weight = 0.8인 logistic 분포와 location = 10, scale = 0.5, weight = 0.2인 logistic 분포를 섞은 후 이산화를 한 것이다. 왼쪽이 PDF, 오른쪽이 CDF이다.



Logistic mixture 분포의 파라미터들을 찾기 위해서 Neural Network를 사용한다. 데이터에 기반하여 NLL을 최소화시키는 파라미터를 찾는다.

4.3 Normalizing Flow(NF)

앞서 살펴본 Discretized Logistic Mixture 여러 분포들을 '병렬적으로' 나열해서 복잡한 분포를 설명하고자 한 접근이었다. 이와 달리 Normalizing flow에서는 하나 이상의 변수 변환을 통해 복잡한 분포를 설명하고자 한다. 즉 sequential한 방법인 것이다. Logistic Mixture로는 고차원에서의 복잡한 분포를 설명하기 어렵다는 한계가 있는데, Normalizing flow를 사용한다면 고차원의 분포 또한 설명할 수 있게 된다는 장점이 있다.

4.3.1 1차원 Normalizing Flow

먼저 1차원에서 NF가 진행되는 과정은 다음과 같다. X를 주어진 데이터의 분포라고 하고, Z_0를 표준정규분포로 설정하자. 그리고 다음과 같은 변수 변환을 설정한다 :

$$Z_0 \overset{g_1}{\rightarrow} Z_1 \overset{g_2}{\rightarrow} Z_2 \overset{g_3}{\rightarrow} \dots \overset{g_k}{\rightarrow} X$$

이때 함수 g_i들의 형태는 사용자가 설정하는 것이고, 함수의 파라미터들은 데이터에 의해 결정된다. 파라미터들은 NLL을 최소화시키는 것을 기준으로 Neural Network를 통해 찾게 된다. 참고로 $Z \overset{g}{\rightarrow} X$의 1차원 변수 변환 공식은 다음과 같다 :

$$f_X(x) = f_Z(g^{-1}(x))|\frac{dz}{dx}|$$

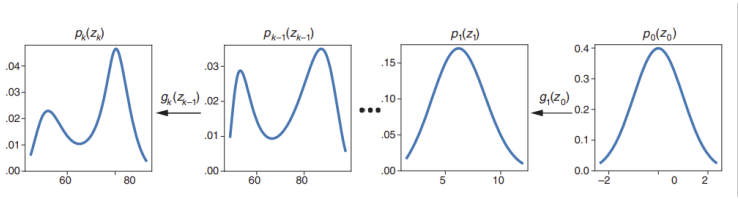
함수를 여러 개 쌓는다면 다음과 같이 나타낼 수 있다 :

$$f_X(x) = f_{Z_0}(z_0)|\frac{dz_0}{dz_1}||\frac{dz_1}{dz_2}|...|\frac{dz_{k-1}}{dx}|$$

하지만 실제 데이터에서는 $f_X(x) \neq f_Z(g^{-1}(x))|\frac{dz}{dx}|$일 수 있다. 예를 들어서 X가 Bimodal distribution을 따를 때, $g(z) = az + b$로 설정한다면, a와 b가 어떤 값을 가지더라도 $g(Z)$와 X가 동일한 분포를 가질 순 없다(Z가 표준정규분포를 따르기 때문). 하지만 $g(Z)$를 X에 가장 근접하게 만들어 주는 a와 b를 찾을 수는 있다. 왜 가능한지 직관적으로 설명을 하자면, 일단 X의 분포는 알 수는 없지만 분명히 존재한다. 그리고 우리에게 주어진 데이터는 X의 분포를 따른다. 그렇기에 X의 분포를 가정하고 계산한 데이터들의 joint probability가 X가 아닌 다른 분포를 가정하고 계산한 joint probability보다 더 높을 것이다. 따라서 데이터 $x_1, x_2, ..., x_N$이 주어졌을 때, $x_1 \overset{g^{-1}}{\rightarrow} z_1, x_2 \overset{g^{-1}}{\rightarrow} z_2, \dots, x_N \overset{g^{-1}}{\rightarrow} z_N$으로 z_i들을 계산하고, 그것들의 joint probability를 계산했을 때 가장 높은 값을 가지게 하는 a와 b가 $g(Z)$를 X에 가장 근접하게 만들어 주는 함수 g의 파라미터들인 것이다. 이를 수식적으로는 다음과 같이 나타낼 수 있다 :

$$\underset{g}{argmax} \prod_{i=1}^{N} f_Z(z_i)$$

위의 예시처럼 단 하나만의 linear transformation $g(z) = az+b$만을 사용한다면 Z로부터 X를 얻어내는 것에 한계가 있을 것이다. Linear transformation의 경우 분포의 location과 scale은 바꿀 수 있지만 형태 자체를 바꾸지는 못하기 때문이다. 따라서 non-linear transformation을 사용한다. 일반적으로는 linear transformation들 사이에 non-linear transformation을 끼워서 사용을 하고, 이렇게 하면 Z로부터 X를 효과적으로 유도할 수 있다.



4.3.2 고차원 Normalizing Flow

이제 $\mathbf{X} = (X_1, X_2, ..., X_D)'$의 분포를 찾기 위해 $\mathbf{Z} = (Z_1, Z_2, ..., Z_D)'$ $N_D(0, I_D)$를 활용할 것이다 (이때, $\mathbf{X}$와 $\mathbf{Z}$는 같은 차원이다). 그리고 $\mathbf{X}$에 최대한 가까운 $g(\mathbf{Z})$를 찾을 것이다. 이때, $g(\mathbf{Z})$는 다음과 같이 정의된다 :

$$g(\mathbf{Z}) = \begin{bmatrix} g_1(z_1, z_2, \ldots, z_D) \\ g_2(z_1, z_2, \ldots, z_D) \\ \cdots \\ g_D(z_1, z_2, \ldots, z_D) \end{bmatrix}$$

고차원에서의 변수 변환 공식은 다음과 같다 :

$$f_{\mathbf{X}}(x) = f_{\mathbf{Z}}(z)|det(\frac{\partial g(z)}{\partial z})|^{-1}$$

$$\frac{\partial g(z)}{\partial z} = \begin{bmatrix} \frac{\partial g_1}{\partial z_1} & \frac{\partial g_1}{\partial z_2} & \cdots & \frac{\partial g_1}{\partial z_D} \\ \frac{\partial g_2}{\partial z_1} & \frac{\partial g_2}{\partial z_2} & \cdots & \frac{\partial g_2}{\partial z_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g_D}{\partial z_1} & \frac{\partial g_D}{\partial z_2} & \cdots & \frac{\partial g_D}{\partial z_D} \end{bmatrix}$$

$\frac{\partial g(z)}{\partial z}$를 Jacobian Matrix라고 부르기도 한다 함수 g를 설정하는 대표적인 유형으로는 Fully Triangular Flow와 Real NVP가 있다

Fully Triangular Flow Fully Triangular Flow는 Jacobian의 determinant와 편미분 계산의 단순화에 초점을 맞춘 방법이다. 일반적으로 행렬의 determinant는 차원이 높아질수록 계산 비용이 매우 커진다. 하지만 삼각 행렬의 determinant는 대각 원소들의 곱이므로 계산이 매우 간편해진다. 따라서 Fully Triangular Flow에서는 Jacobian이 삼각행렬이 될 수 있도록 g를 설정한다.

$$\frac{\partial g_1}{\partial z_2} = 0, \frac{\partial g_1}{\partial z_3} = 0, ..., \frac{\partial g_1}{\partial z_D} = 0, \frac{\partial g_2}{\partial z_3} = 0, ...$$

이를 일반화하면 다음과 같다 :

$$\frac{\partial g_i}{\partial z_j} = 0 \quad if \quad j > i$$

이는 i번째 함수 g_i는 z_j (j>i)를 포함하지 않아야 한다는 것을 의미한다. 그래서 g를 다음과 같이 단순회시켜 나타낼 수 있게 된다.

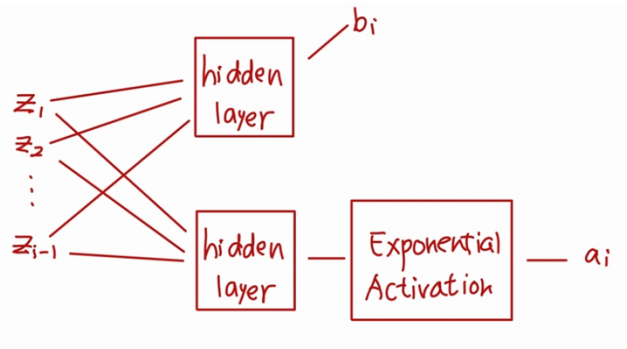
$$g(\mathbf{Z}) = \begin{bmatrix} g_1(z_1, z_2, \ldots, z_D) \\ g_2(z_1, z_2, \ldots, z_D) \\ \cdots \\ g_D(z_1, z_2, \ldots, z_D) \end{bmatrix} = \begin{bmatrix} g_1(z_1) \\ g_2(z_1, z_2) \\ \cdots \\ g_D(z_1, z_2, \ldots, z_D) \end{bmatrix}$$

$$\frac{\partial g(z)}{\partial z} = \begin{bmatrix} \frac{\partial g_1}{\partial z_1} & 0 & \cdots & 0 \\ \frac{\partial g_2}{\partial z_1} & \frac{\partial g_2}{\partial z_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g_D}{\partial z_1} & \frac{\partial g_D}{\partial z_2} & \cdots & \frac{\partial g_D}{\partial z_D} \end{bmatrix}$$

일차적으로 Jacobian을 삼각행렬로 단순하게 만들었지만, 우리에게 필요한 것은 전체 Jacobian이 아니라 그것의 determinant이므로 대각 원소 부분만 계산하면 된다. 즉, 우리는 i번째 함수를 i번째 변수로 편미분하기만 하면 된다는 것이다. 이때 편미분 계산을 간편화하기 위해서 i번째 함수를 i번째 변수에 대한 linear function으로 설정한다. 그리고 나머지 변수들은 slope와 bias를 결정짓도록 설정한다.

$$g_i(z_1, z_2, ..., z_i) = exp(\alpha_i(z_1, z_2, ..., z_{i-1})) \times z_i + b_i(z_1, z_2, ..., z_{i-1})$$

이렇게 하면 Jacobian의 determinant는 $\prod_{i=1}^{D} exp(\alpha_i(z_1, z_2, ..., z_{i-1}))$가 된다. 이때 exponential이 곱해지는 이유는 기울기가 0이 되는 것을 방지하기 위함이다(만약 기울기가 0이 된다면 Jacobian의 determinant는 0이 되게 된다). 각 파라미터들은 Neural Network를 활용해 다음과 같이 학습이 된다 : 이때 추가로 주목해야 할 점은 주어진 데이터는 $x_1, .., x_N$인 반면, NN에 들어가는 데이터는 $z_1, .., z_N$



라는 점이다. 따라서 g^{-1}를 이용해서 z_i들을 구할 필요가 있다. 그런데 1차원에서와 달리 모든 z_i들을 한번에 다 계산할 수는 없고, 다음과 같은 sequential한 방법을 이용하여 계산해야 한다

$$z_1 = x_1$$

$$z_2 = \frac{x_2 - b_2(z_1)}{exp(\alpha_2(z_1))}$$

$$z_3 = \frac{x_3 - b_3(z_1, z_2)}{exp(\alpha_3(z_1, z_2))}$$

$$z_4 = \frac{x_4 - b_4(z_1, z_2, z_3)}{exp(\alpha_4(z_1, z_2, z_3))}$$

$$\cdots$$

Real non-volume preserving flow(Real NVP) Fully Triangle Flow는 determinant 계산의 단순화를 위해 제약 조건을 걸었음에도 불구하고 파라미터의 수가 굉장히 많은 방법이다. 각 함수 g_i마다 NN을 사용하여 파라미터를 구하는데, 이러한 함수가 총 D개 있으니(차원의 크기=D) D개의 NN의 모든 파라미터를 fitting을 시켜야 하기 때문이다. 그래서 Fully Triangle Flow의 기본적인 특징들은 유지한 채 조금 더 계산을 간편화시킨 것이 Real NVP이다. Real NVP는 $g_1, g_2, .., g_d$까지를 identity function으로 설정한다(이때, d는 사용자 지정이다). 그리고 i>d에 대해선 g_i를 Fully Triangle Flow에서와 비슷하게 z_i에 대해 linear하게 설정을 한다. 그런데 Fully Triangle Flow에서는 j<i인 모든 z_j를 사용하여 slope와 bias를 계산했었는데, Real NVP에서는 $z_1, z_2, ..., z_d$만을 사용하여 slope와 bias를 구한다. 5차원 분포(D=5)에서 d=2로 설정하면 g_i들을 다음과 같이 나타낼 수 있다 :

$$x_1 = g_1(z_1) = z_1$$

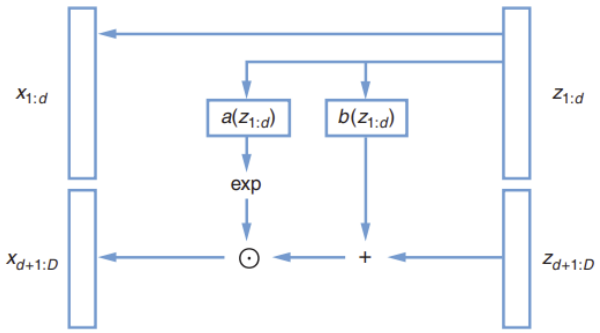
$$x_2 = g_2(z_2) = z_2$$

$$x_3 = g_3(z_1, z_2, z_3) = exp(\alpha_3(z_1, z_2)) \times z_3 + b_3(z_1, z_2)$$

$$x_4 = g_4(z_1, z_2, z_4) = exp(\alpha_4(z_1, z_2)) \times z_4 + b_4(z_1, z_2)$$

$$x_5 = g_5(z_1, z_2, z_4) = exp(\alpha_5(z_1, z_2)) \times z_5 + b_5(z_1, z_2)$$

Real NVP의 구조를 그림으로 나타내면 다음과 같다 : Real NVP에서는 d차원에 대해서 아무런 변화를



주고 있지 않다. 그런데 이러한 방법으로 과연 복잡한 고차원 분포를 온전하게 표현할 수 있는지 의문이 들 수 있다. 주어진 데이터의 분포는 모든 차원에서 복잡한 양상을 띌 수 있으니 말이다. 하지만 어차피 고차원에서도 여러개의 함수를 사용해 chain을 구축할 것이고, 중간 중간에 permutation function을 섞어 준다면 앞서 변화가 없었던 부분들도 충분히 변화를 줄 수 있다.