Probabilistic Neural Network 2

ESC 2024 Winter Session 6주차
2조 유채원, 이상윤, 장덕재, 황보유민

Contents

1. Probabilistic DL with Gaussian
2. Probabilistic DL with Poisson
3. Probabilistic DL Models in the Wild
   - Multinomial Distribution
   - Discritized Logistic Mixture
   - Normalizing Flow

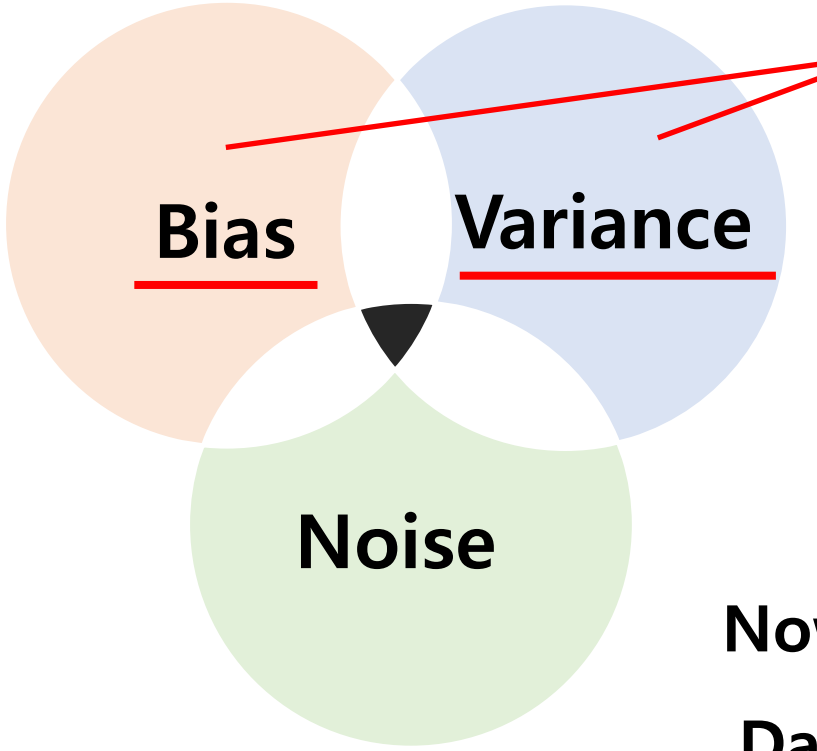
1. Probabilistic DL with Gaussian

Introduction to Probabilistic Deep Learning

1. Three sources of error
2. Noise
3. Example : If you pick Gaussian
4. Find μ, σ as an estimators without DNN
5. Find μ, σ as an estimators with DNN

The 3 Sources of Error

There are 3 sources in element of error



But, We only focused

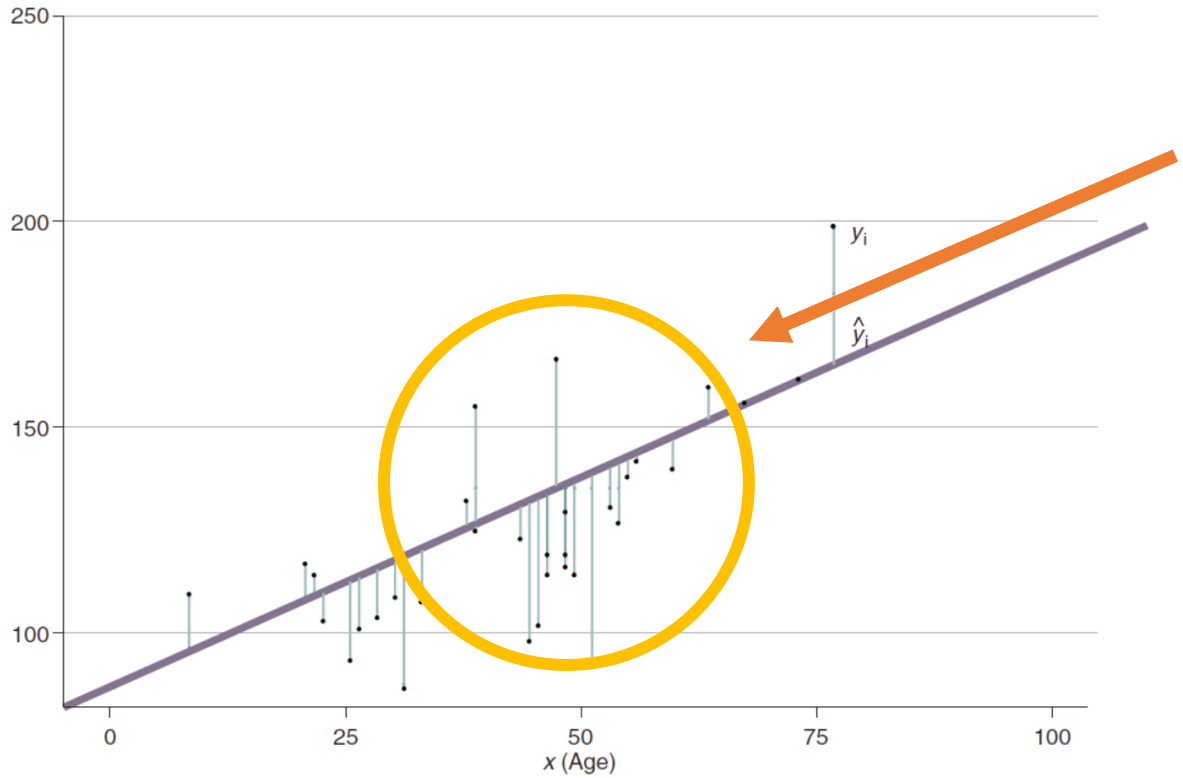
Bias-Variance trade-off relationship!

Now we want to discuss on

Data-inherent Uncertainty :

Noise

Noise



They have different blood pressure

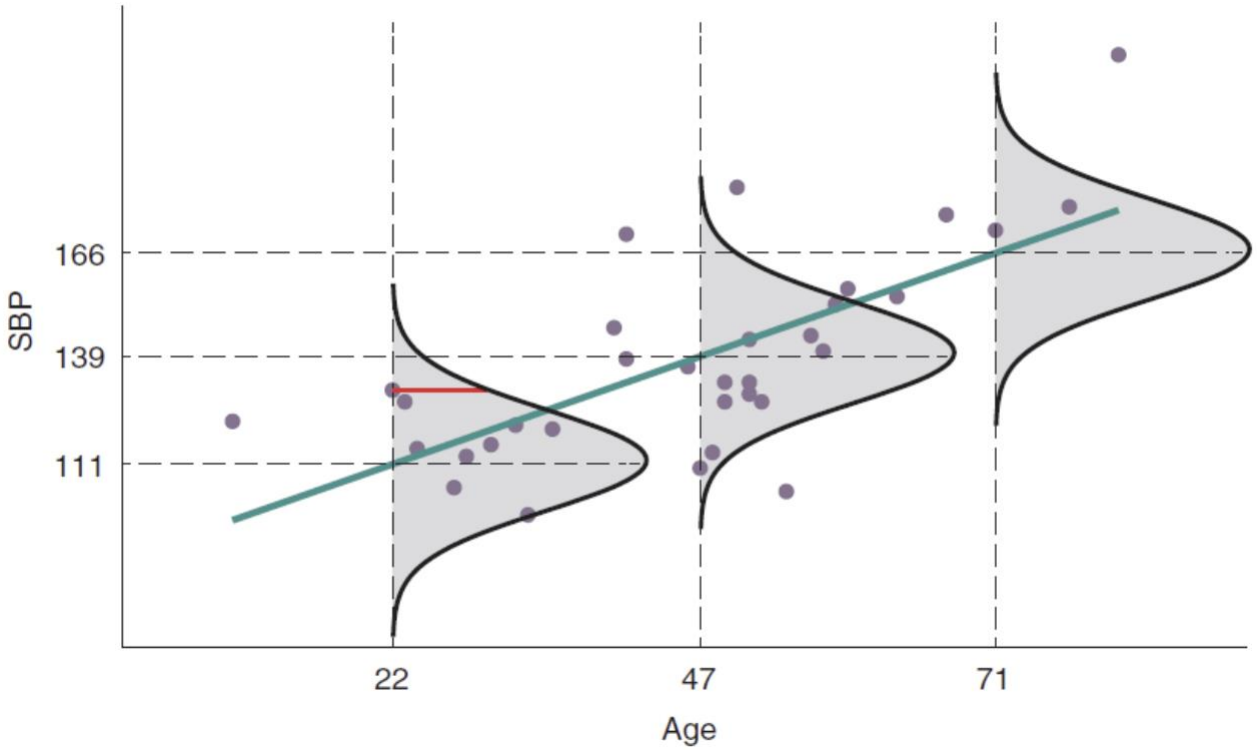
even though they have same age !

Even further, same woman could show

different blood pressure

Measured at two different times

Noise



Thus, we try to assume

CPD(Conditional Probability Distribution)

of the Ground Truth data

It has real practical importance to area needed critical or costly decision

: Medical, Financial

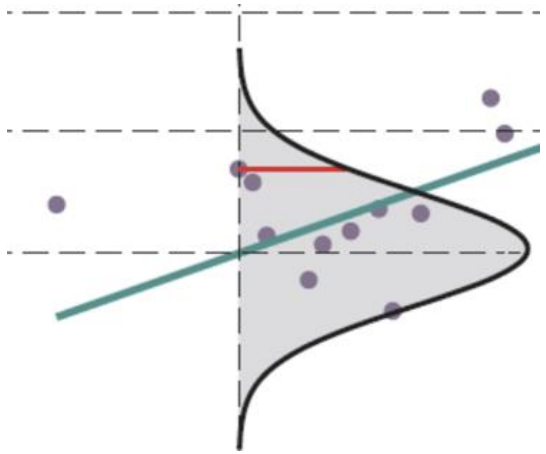
Example : If you pick a Gaussian…

There are only two parameters

of μ and σ!

Then you can define a 95% prediction interval

$$\mu - 1.96\sigma \leq X \leq \mu + 1.96\sigma$$



Now we're going to set

likelihood function and define Energy…

We obtain μ and σ

by optimizing Energy

with DNN !

Example : If you pick a Gaussian…

Assume i.i.d (Identical and Independent Distribution)

Likelihood Distribution $\prod_{\{i=1\}}^{\{n\}} \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{(x-\mu)^2}{2\sigma^2})$

Log-likelihood $\sum_{\{i=1\}}^{\{n\}} \ln\frac{1}{\sigma\sqrt{2\pi}} - \frac{(x-\mu)^2}{2\sigma^2}$

Energy $\frac{1}{n}\sum_{\{i=1\}}^{\{n\}} -\ln\frac{1}{\sigma\sqrt{2\pi}} + \frac{(x-\mu)^2}{2\sigma^2}$

Maximize Log-likelihood is equivalent to Minimize Energy ….

Find μ, σ as an estimators without DNN

Find the point which satisfies derivative function equals zero

- $\frac{\partial E}{\partial \mu} = 0$ ⟶ $\hat{\mu} = \frac{1}{n}\sum_i x_i$ **(Sample mean)**
- $\frac{\partial E}{\partial \sigma} = 0$ ⟶ $\hat{\sigma} = \sqrt{\frac{\sum(x_i - \hat{\mu})^2}{n}}$

Are they biased?

Apply ensemble to estimate True distribution

- $E(\hat{\mu}) = \mu = \frac{1}{n}\sum_i x_i$
- $E(\hat{\sigma}) = \sigma = \sqrt{\frac{\sum_i (x_i - \hat{\mu})^2}{n-1}}$

$\hat{\mu}$ is not biased but

$\hat{\sigma}$ is little bit (under)biased …

Its DOF is n-1 since it is obtained by $\hat{\mu}$

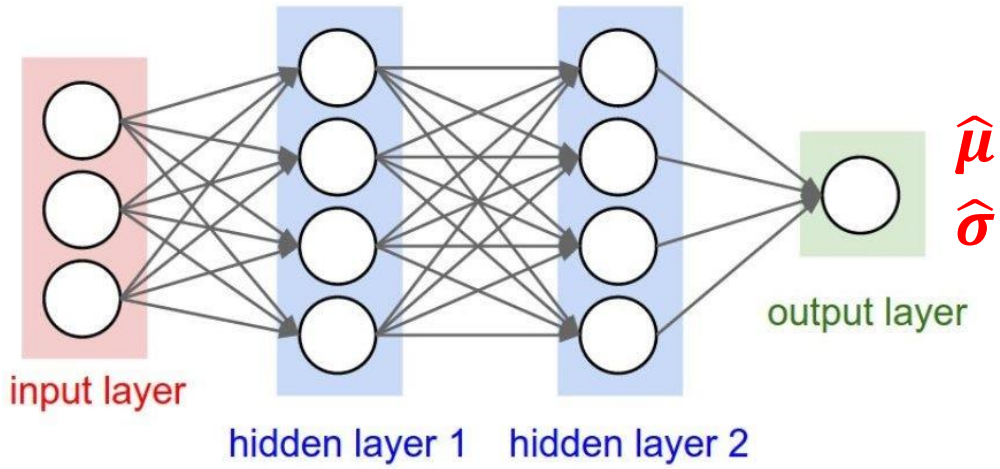
It is not important if n is sufficiently large(Deep Learning)

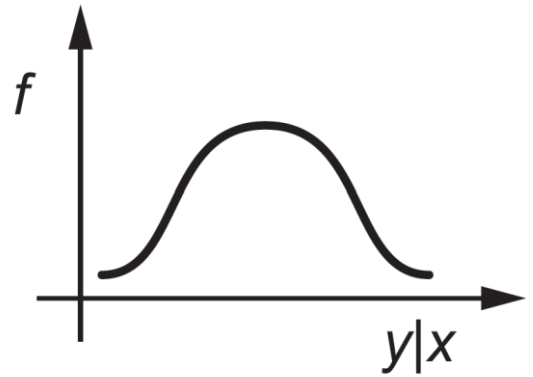
But you are statistician …

Find μ, σ as an estimators with DNN

Probabilistic Deep Learning

Network learns the parameters of the CPD





Predicts CPD for each input

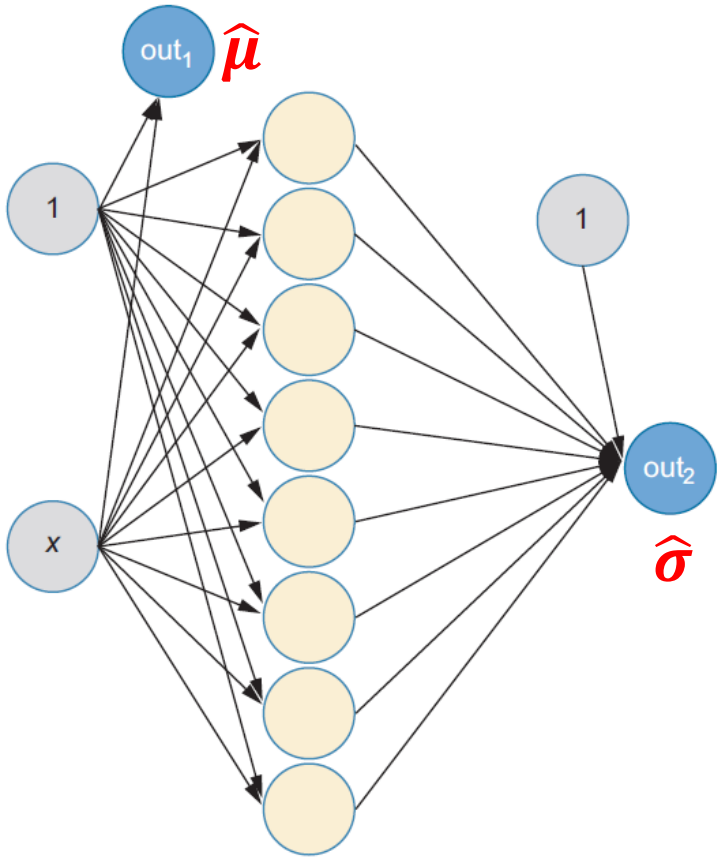
Input Layer size: 입력 변수의 개수

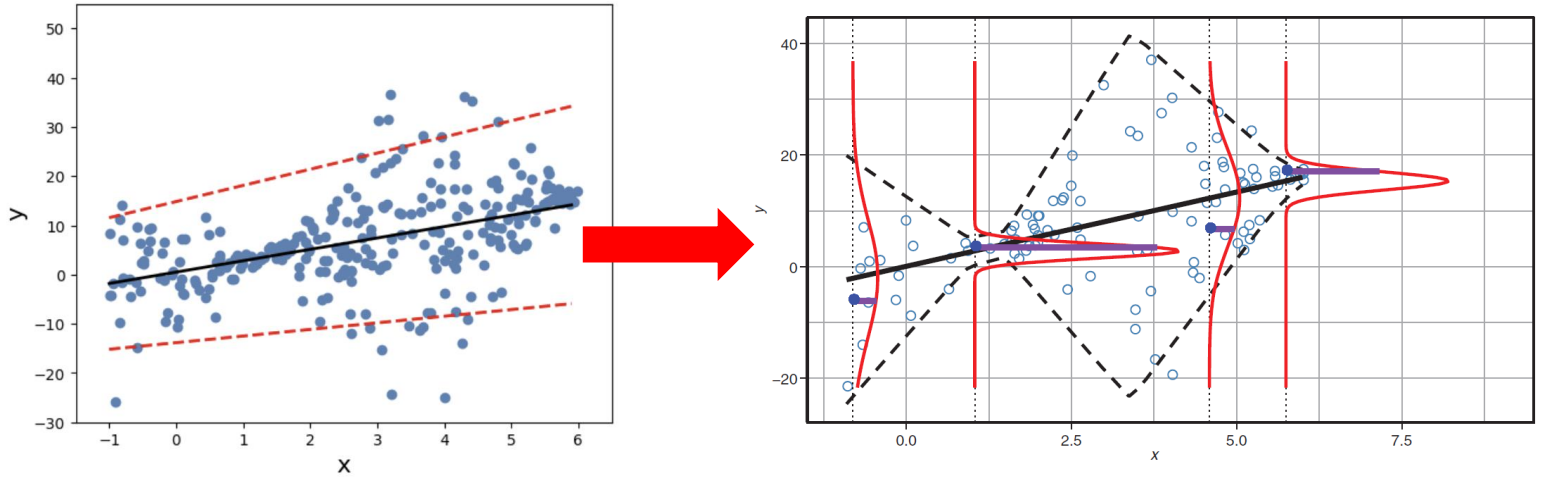
Output Layer size: 필요한 파라미터의 개수

Find μ, σ as an estimators with DNN

Probabilistic Deep Learning

Network learns the parameters of the CPD





Predicts CPD for each input

Hidden layer imports Nonlinearity to $\hat{\sigma}$!

We want to make the SDV flexible over new data groups !

Find μ, σ as an estimators with DNN

We use TensorFlow library

Keras library helps you to construct Neural Network

```
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense, Lambda
from tensorflow.keras.layers import Concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

Find μ, σ as an estimators with DNN

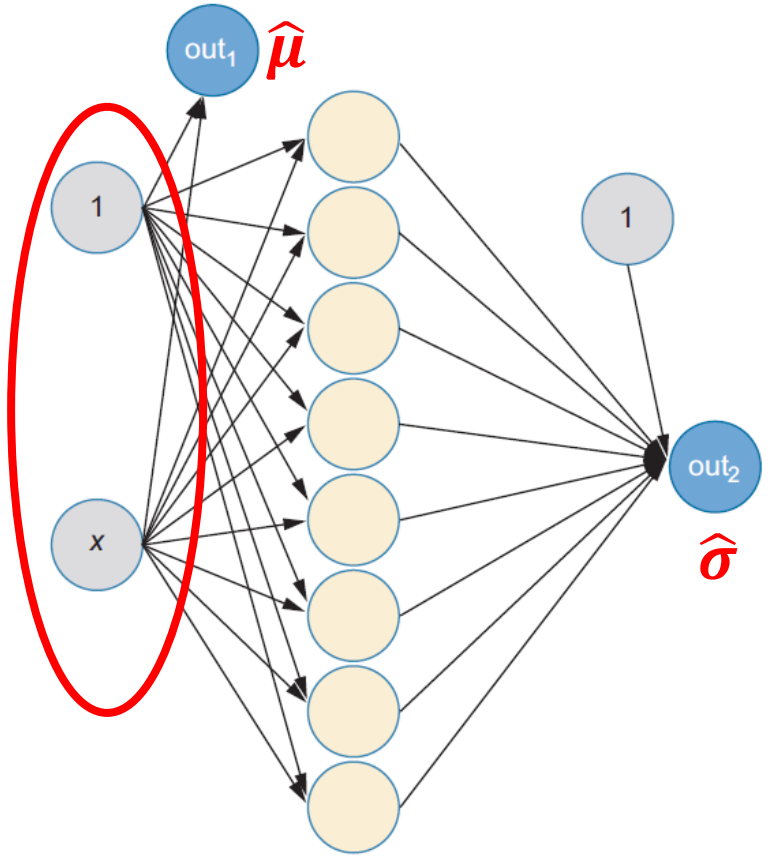
```
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense, Lambda
from tensorflow.keras.layers import Concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

```
inputs = Input(shape=(1,))
```

 # Of input neuron

(except for bias)

Input method helps to construct input layer



Find μ, σ as an estimators with DNN

```
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense, Lambda
from tensorflow.keras.layers import Concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

Dense method helps to "stack" layers

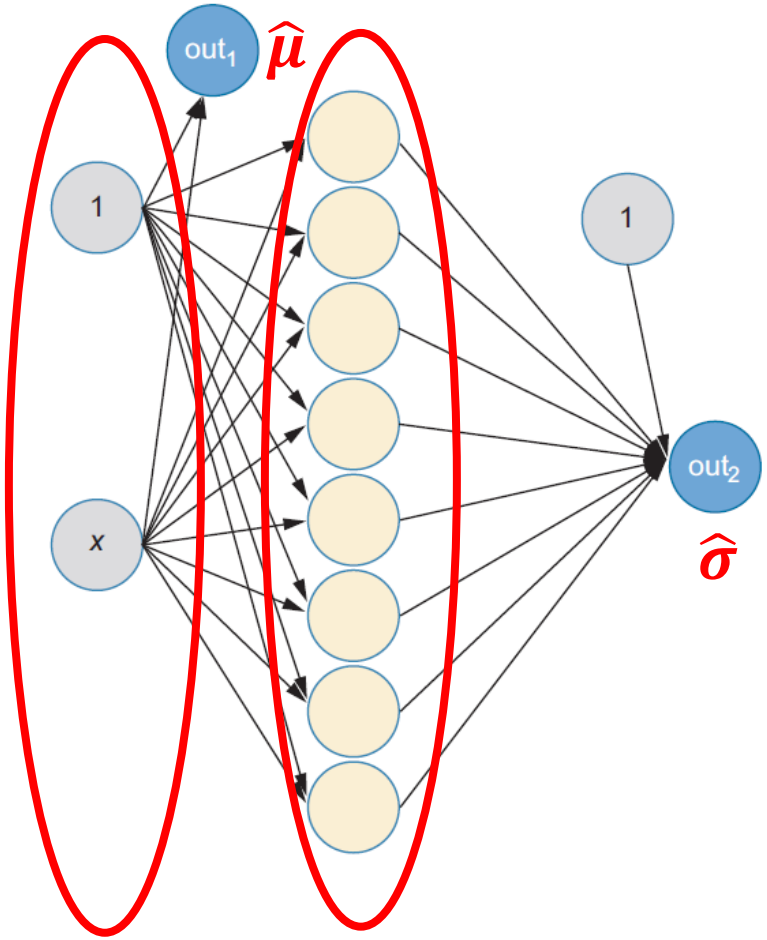
```
hidden1 = Dense(30,activation="relu")(inputs)
hidden1 = Dense(20,activation="relu")(hidden1)
hidden2 = Dense(20,activation="relu")(hidden1)
```

 # Of Neurons on current layer

Activation Function (ReLU is normal)

Previous layer connected to current layer

*** Different layer**



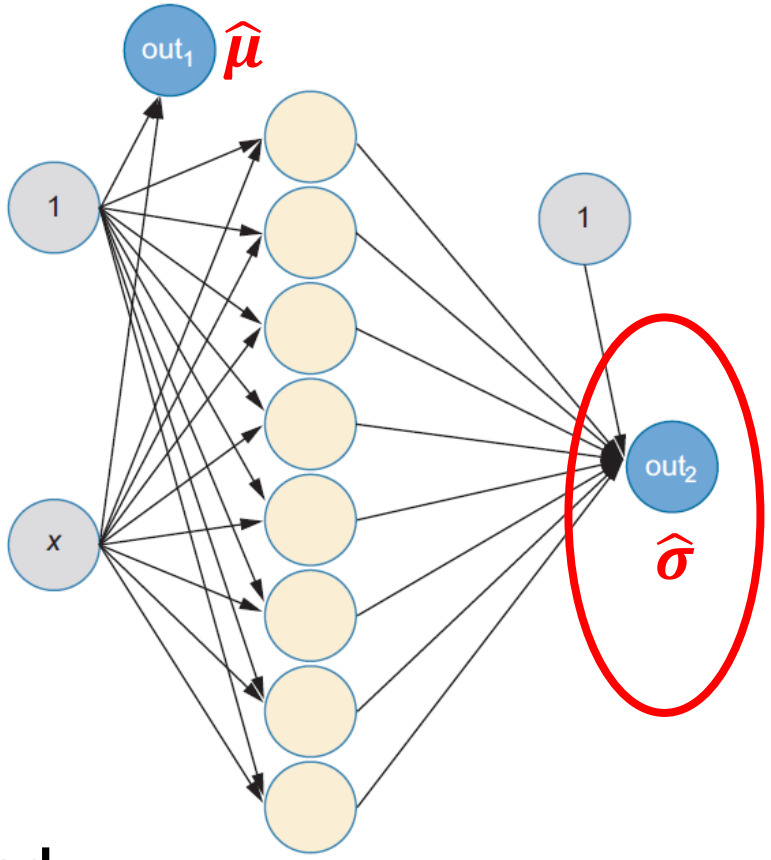
Find μ, σ as an estimators with DNN

```
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense, Lambda
from tensorflow.keras.layers import Concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

```
out1 = Dense(1)(inputs)
out2 = Dense(1)(hidden2)
```

# of output Neurons (except for bias)

Previous Input



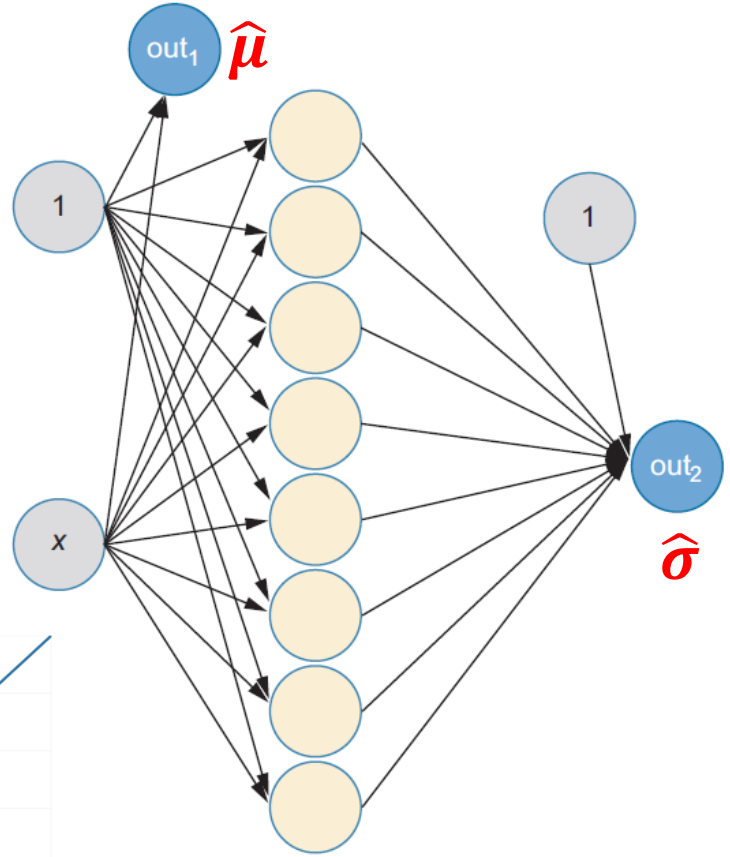
Output layer can be constructed by Dense method

Find μ, σ as an estimators with DNN

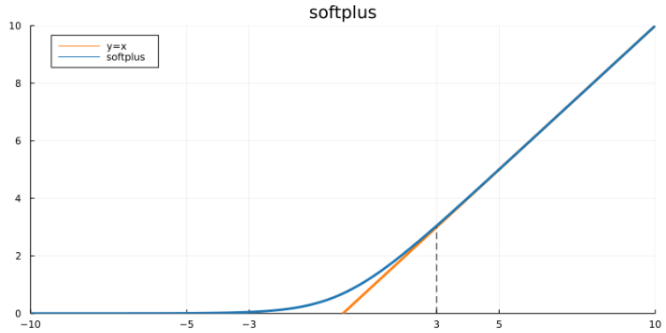
```
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense, Lambda
from tensorflow.keras.layers import Concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

Define a distribution

```
def my_dist(params):
  return tfd.Normal(loc=params[:,0:1], scale=1e-3 + tf.math.softplus(0.05 * params[:,1:2]))
```




```
params = Concatenate()([out1,out2])
```



*Params is a list for parameters $[\hat{\mu} \quad \hat{\sigma}]$

*Softplus is used to make SDV obtained by NN to be always positive

Find μ, σ as an estimators with DNN

```
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense, Lambda
from tensorflow.keras.layers import Concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

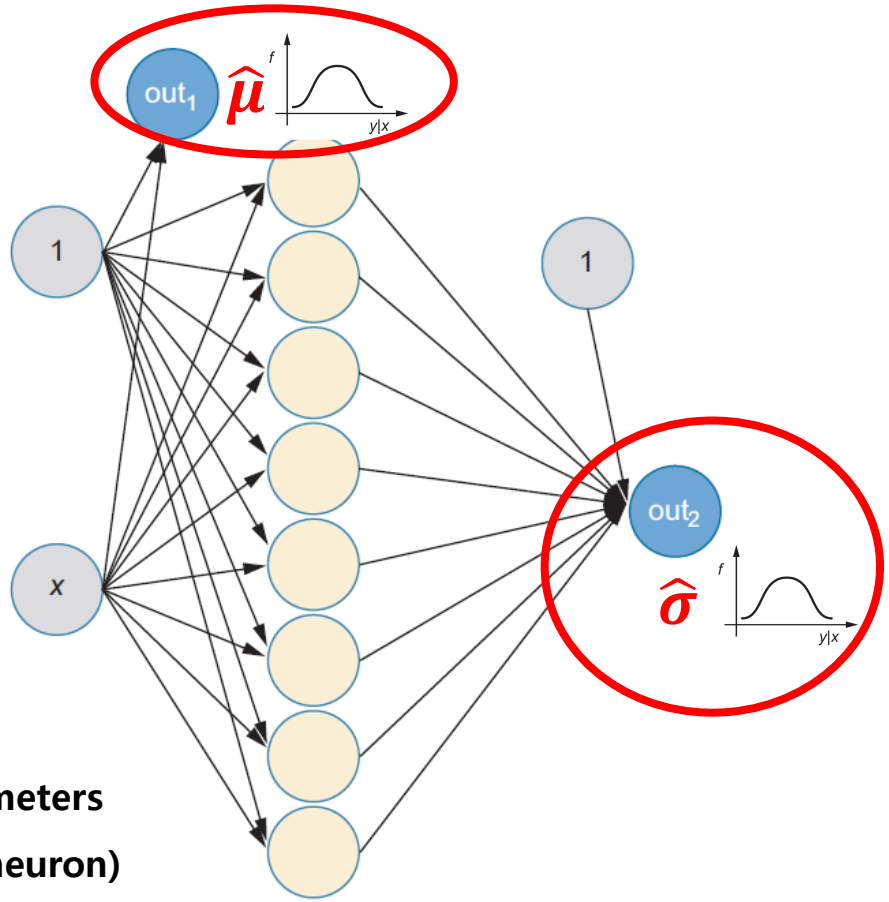
DistributionLambda provides distribution to output neuron

```
dist = tfp.layers.DistributionLambda(my_dist)(params)
```

New Output

User-defined distribution

Its parameters (Output neuron)



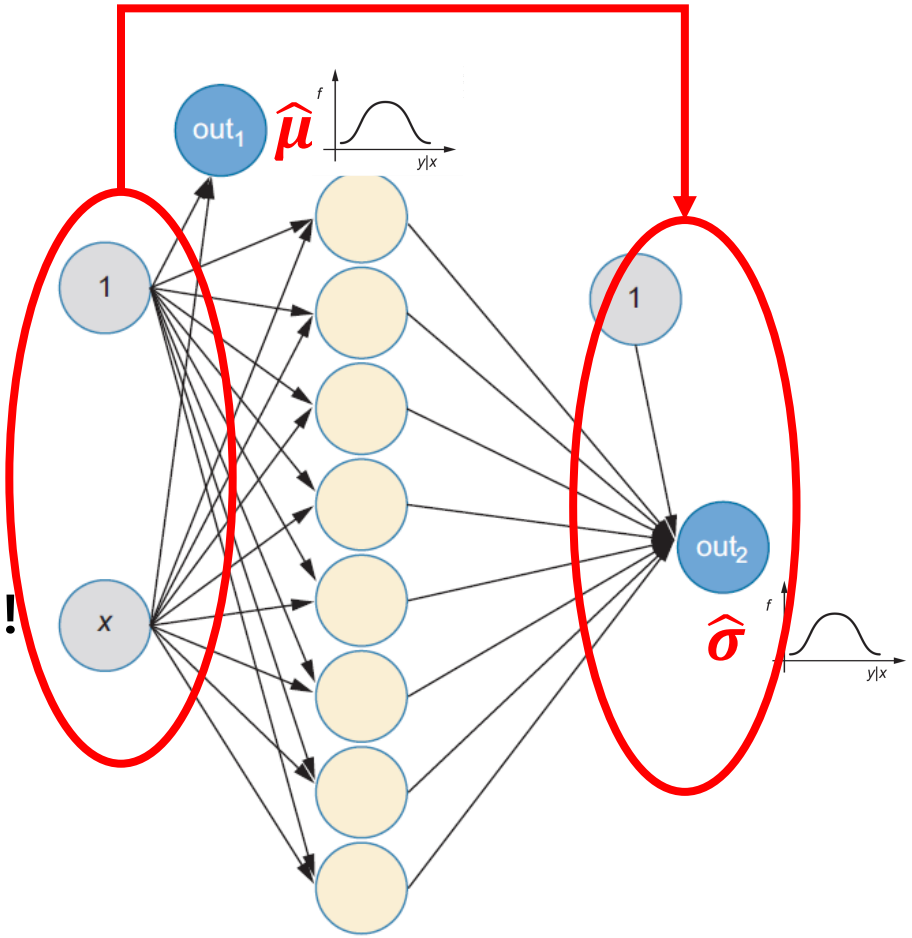
Find μ, σ as an estimators with DNN

```
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense, Lambda
from tensorflow.keras.layers import Concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

Model method connects input and output directly

```
model_flex_sd = Model(inputs=inputs, outputs=dist)
model_flex_sd.compile(Adam(learning_rate=0.01), loss=NLL)
```

Adam is the one of most famous optimization method!



Then, we finished to construct our Neural Network architecture

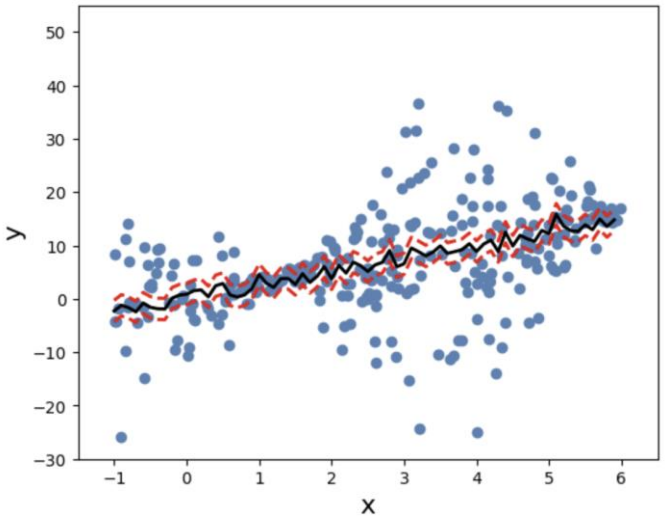
(Just architecture)

Find μ, σ as an estimators with DNN

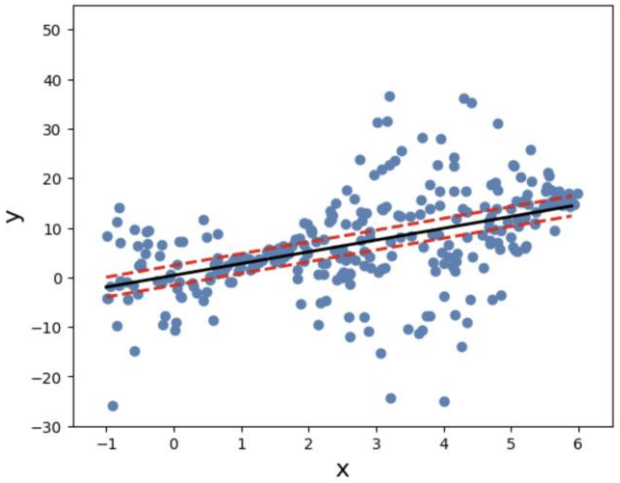
```
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense, Lambda
from tensorflow.keras.layers import Concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

Learning

```
model_flex_sd.fit(x_train, y_train, epochs=2000, verbose=0, validation_data=(x_val,y_val))
```



After fitting



Get mean and SDV

with Lambda

```
flex_sd_mean = Lambda(lambda x: x.mean())(dist)
flex_sd_sd = Lambda(lambda x: x.stddev())(dist)

model_flex_sd_mean = Model(inputs=inputs, outputs=flex_sd_mean)
model_flex_sd_sd = Model(inputs=inputs, outputs=flex_sd_sd)
```

Find μ, σ as an estimators with DNN

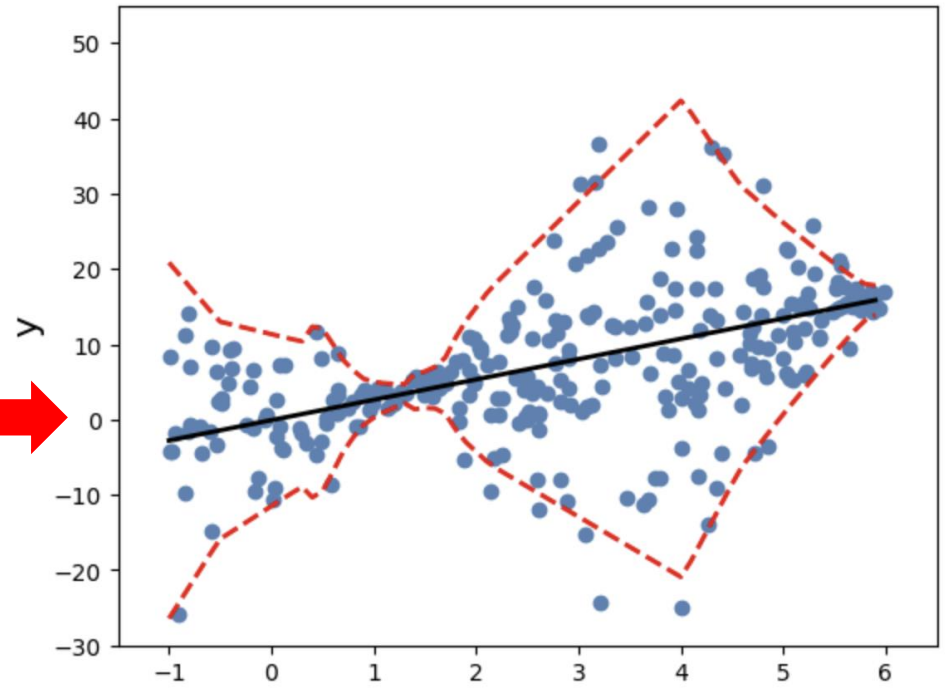
```
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense, Lambda
from tensorflow.keras.layers import Concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

```
plt.figure(figsize=(14,5))

x_pred = np.arange(-1,6,0.1)

plt.subplot(1,2,1)
plt.scatter(x_train,y_train,color="steelblue") #observerd
preds = model_flex_sd_mean.predict(x_pred)
plt.plot(x_pred,preds,color="black",linewidth=2)
plt.plot(x_pred,preds+2*model_flex_sd_sd.predict(x_pred),color="red",linestyle="--",linewidth=2)
plt.plot(x_pred,preds-2*model_flex_sd_sd.predict(x_pred),color="red",linestyle="--",linewidth=2)
plt.xlabel("x",size=16)
plt.ylabel("y",size=16)
plt.title("train data")
plt.xlim([-1.5,6.5])
plt.ylim([-30,55])
```

Noise in data is expressed!

Black : Fitted model

Red : Fitted model + Noise



2. Probabilistic DL with Poisson

Introduction to Probabilistic Deep Learning

1. Is Gaussian always best?
2. Examples with count datas
3. Modeling Count Data with Poisson Dist.
4. Modeling Count Data with ZIP
5. Dilemma on Probabilistic model

Is Gaussian always the best?

Normal distribution is “Normal”, standard distributions

since they maximizes entropy

But it is not always the best approach!

For instance, you might want to pick Poisson distribution instead

when you model discrete count data with low average counts.

Is Gaussian always the best? : Examples with count datas

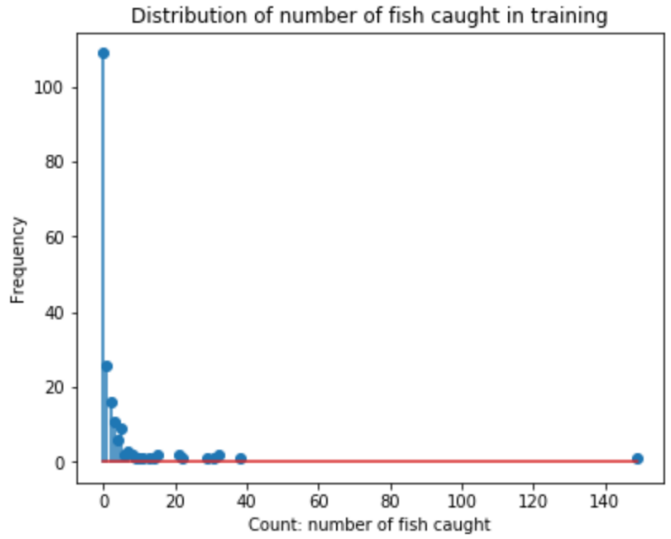
Find the best distribution

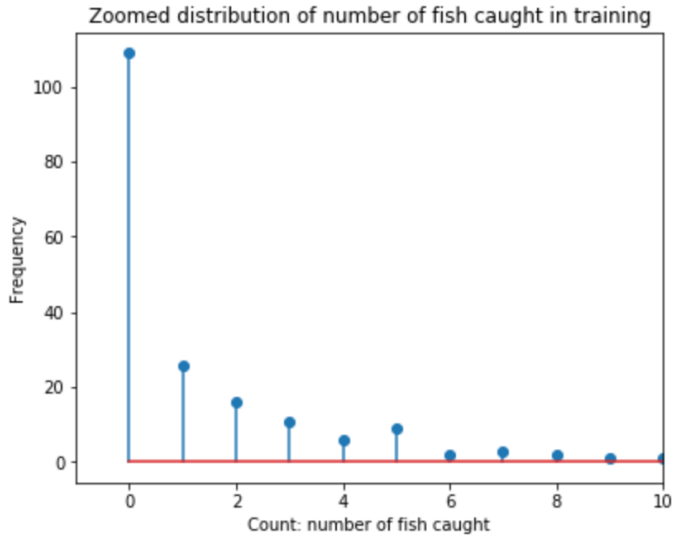
X1: # of People

X2: # of Children

X3: existence camping car

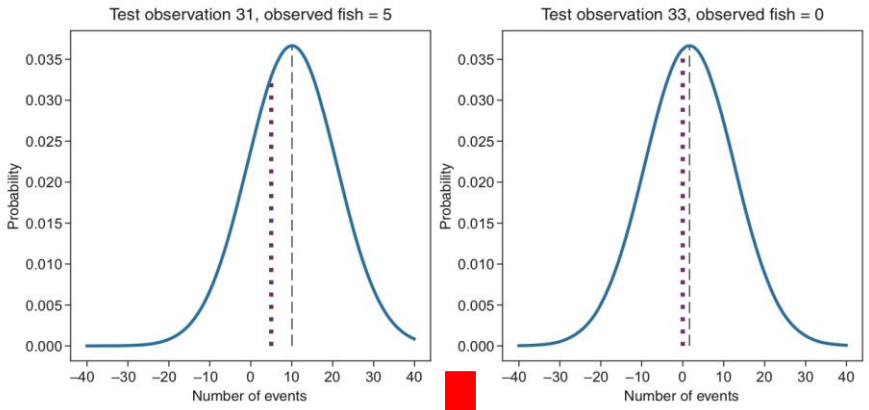
Y: Caught fish w.r.t each group

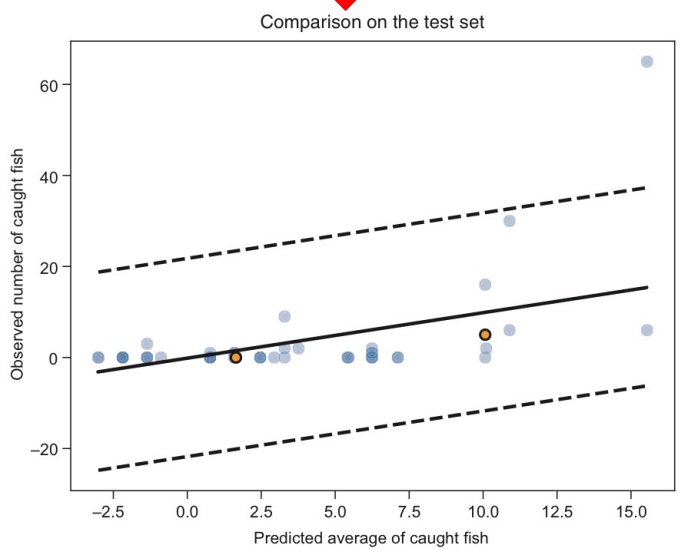




Is Gaussian always the best? : Examples with count datas

Distribution for counting # of caught Fish





If : Linear Regression with constant variance

Then …

Problems!

1. **Negative numbers also have high likelihood**
2. **Predicted Outcome is continuous**

Solutions

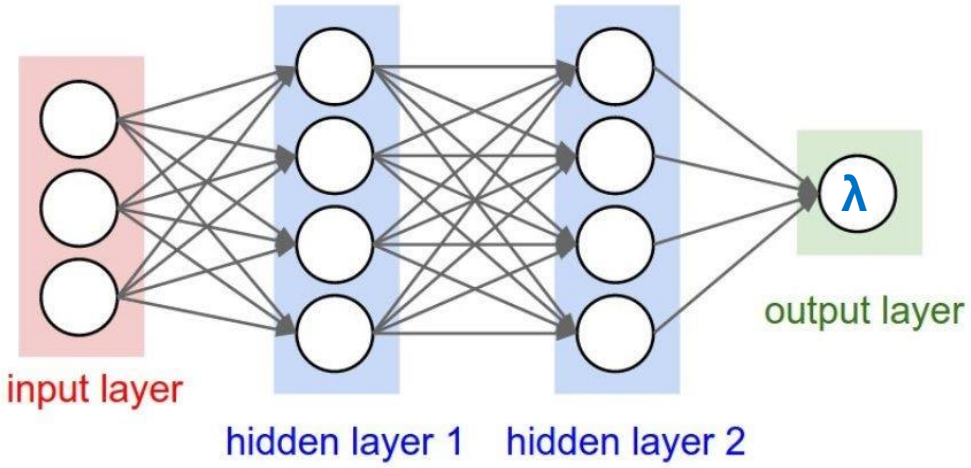
1. **Only positive numbers have likelihood**
2. **Make discrete outcome**

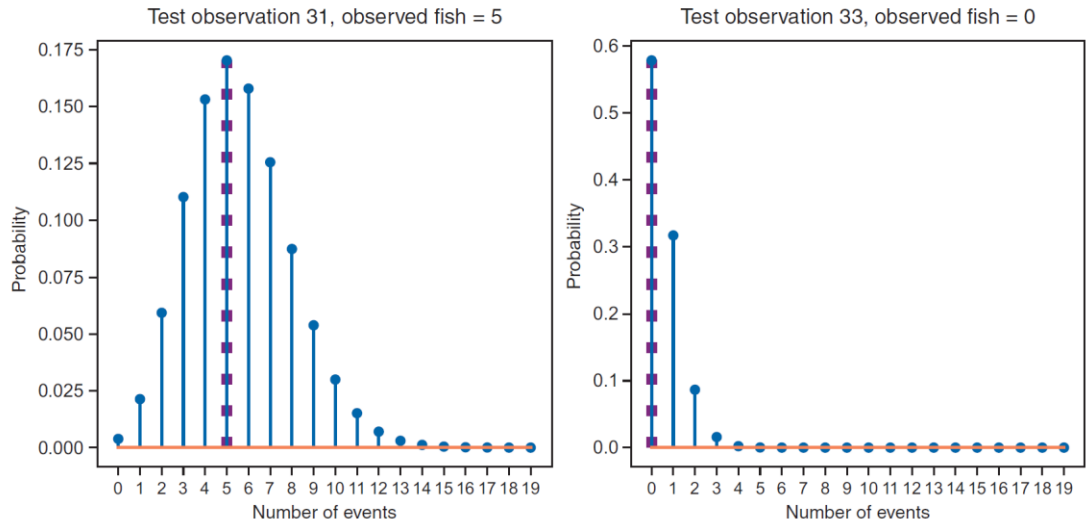
Modeling Count Data with Poisson Dist.

Probabilistic Deep Learning

Network learns the parameters of the CPD

$$P(y = k) = \frac{\lambda^k \cdot e^{-\lambda}}{k!} \qquad \boldsymbol{\lambda \geq 0}$$





Activation function which range ≥ 0

E.g) exponential, softplus

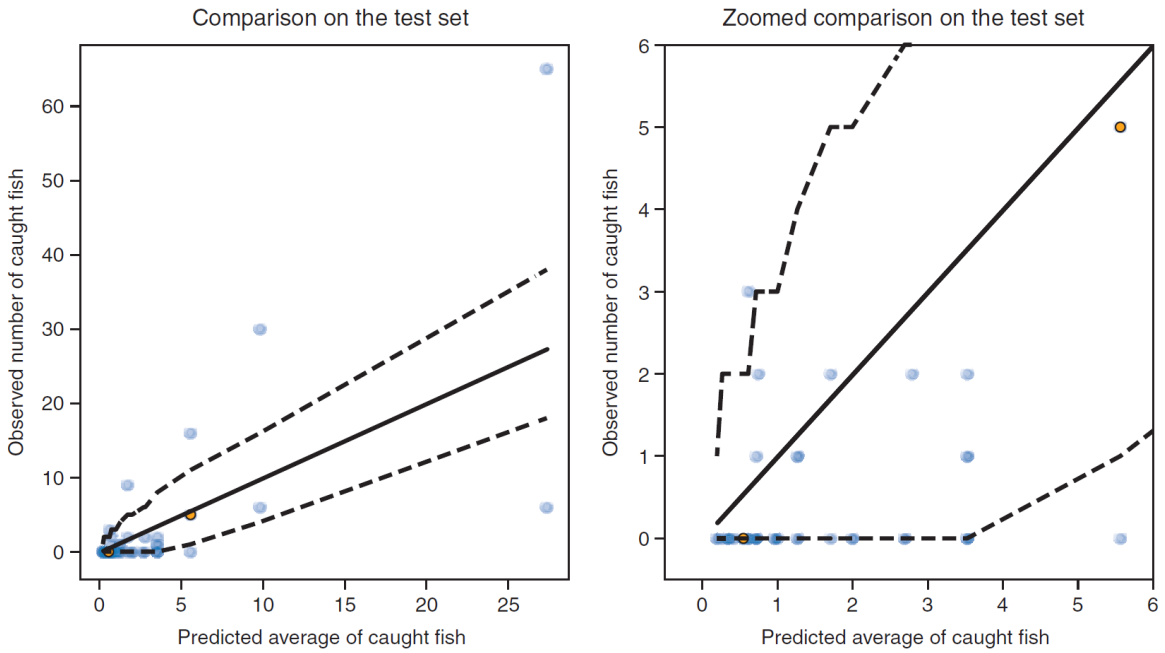
Solutions

1. Only **positive numbers** have likelihood
2. Make **discrete outcome**

Modeling Count Data with Poisson Dist.

Poisson Distribution

Fishing is too difficult !!!

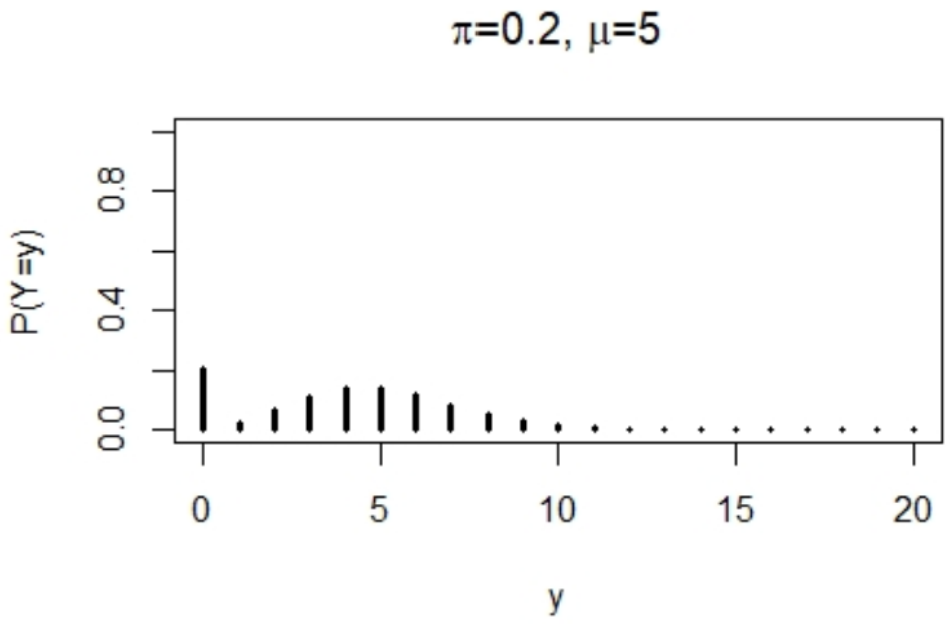


There are too many values near zeros

Modeling Count Data with ZIP

Use Zero-Inflated Poisson(ZIP) Distribution

Instead just Poisson!



There are two parameter ; π and λ

Main $\Pr(Y=0) = \pi + (1-\pi)e^{-\lambda}$

Co-main $\Pr(Y=y_i) = (1-\pi)\dfrac{\lambda^{y_i}e^{-\lambda}}{y_i!}, \qquad y_i = 1,2,3,\ldots$

$\lambda = 0$: Pr(y=0) = 1

$\lambda = \infty$: Pr(y=yi) = π (0< π <1)

π : Probability (Use sigmoid for activation function)

Modeling Count Data with ZIP

TFP doesn't provide ZIP distribution -> set custom function

Listing 5.7 Custom distribution for a ZIP distribution

The first component codes the rate. We used exponential to guarantee values > 0 and the squeeze function to flatten the tensor.

The second component codes zero inflation; using the sigmoid squeezes the value between 0 and 1.

```
def zero_inf(out):
    rate = tf.squeeze(tf.math.exp(out[:,0:1]))
    s = tf.math.sigmoid(out[:,1:2])

    probs = tf.concat([1-s, s], axis=1)
    return tfd.Mixture(
        cat=tfd.Categorical(probs=probs),
        components=[
        tfd.Deterministic(loc=tf.zeros_like(rate)),
        tfd.Poisson(rate=rate),
    ])
```

The two probabilities for 0's or Poissonian distribution

tfd.Categorical allows creating a mixture of two components.

Zero as a deterministic value

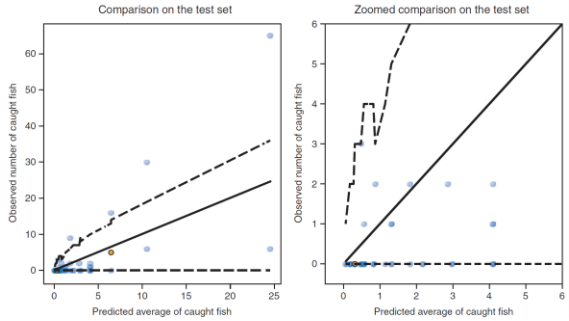
Value drawn from a Poissonian distribution

$$\text{Rate} = \lambda$$

$$S = \pi$$

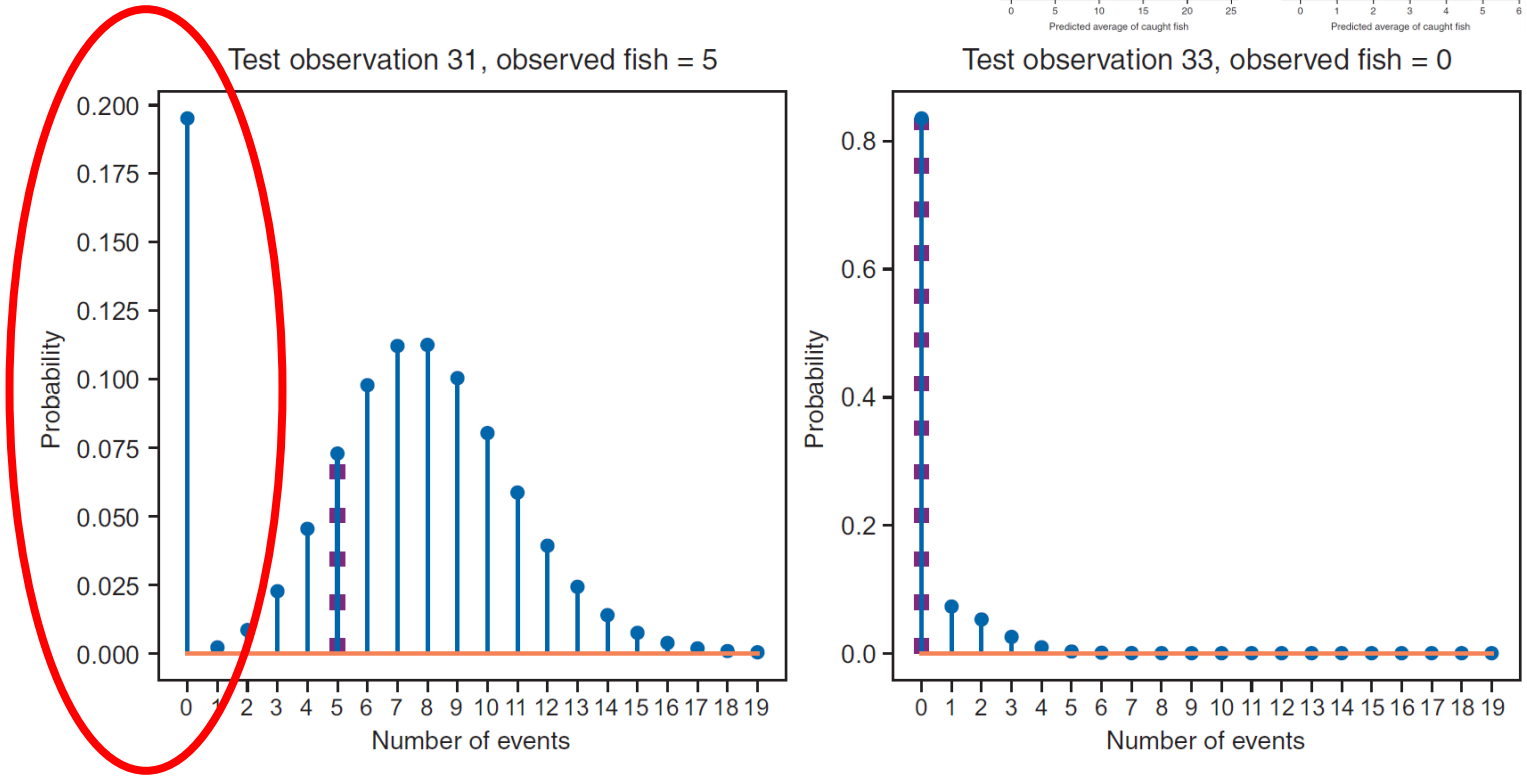
Modeling Count Data with ZIP

Fitted Zero-Inflated Poisson Distribution



2.5% percentile stays at zero over whole range

Fishing is too difficult !!!



High Likelihood on zero !

Modeling Count Data with ZIP

Root Mean Square Error

Mean Absolute Error

	Linear Regression	Poisson	Zero-Inflated
RMSE	8.6	7.2	7.3
MAE	4.7	3.1	3.2
NLL	3.6	2.7	2.2

NLL is our measure

NLL should only be compared in discrete models

참고: count data에서 negative binomial distribution을 이용하기도 함

Is Gaussian always the best?

Our suggestion

If a property of output is

Continuous(Regression) → **Gaussian**

Discrete(Classification) → **Multinomial Dist.**

Count Data → **Poisson**

(# of ~)

Dilemma on Probabilistic model

You use DNN to determine the parameters of the CPD …

1. You pick an appropriate distribution model for the outcome

2. And optimize likelihood or Energy with NN

3. Apply your trained CPD model to test dataset

Hence, There are two ways for optimizing Probabilistic DL Models …

1. Choose appropriate neural network architectures

2. Choose right distributions to the real-world data that makes lowest error

Dilemma on Probabilistic model

You use DNN to determine the parameters of the CPD …

2. Choose right distributions to the real-world data that makes lowest error

Is it easy or always possible? No

Let us consider advantage of the Deep Neural Network…

There are two approach in AI world,

Parametric and Non-parametric approach

Parametric Model

A learning model that summarizes data with a set of parameters of fixed size

(Independent of the number of training examples)

Assume that the data distribution can be defined in terms of a finite set of parameters

Dilemma on Probabilistic model

Parametric Model

Pros and cons

- **Constrained (Not flexible), weak at outliers**
- **Low complexity**

Since we assumed certain distribution!

Examples :

Linear Regression, Logistic Regression, Perceptron, Simple Neural Network

LDA, Support Vector Machine, K-means clustering

Non-parametric model

Assume that the data distribution cannot be defined in terms of a finite set of parameters

Pros and cons

- Flexibility(Data distribution is not assumed)
- Complexity can grow with the number of observations N
- Good when you have a lot of data and no prior knowledge,

and when you don’t want to worry too much about choosing just the right features

Non-parametric model

Assume that the data distribution cannot be defined in terms of a finite set of parameters

Examples :

Decision trees, PCA, Deep neural networks

Dilemma on Probabilistic model

If you have only few datas ...

1. If you still need nonparametric approach

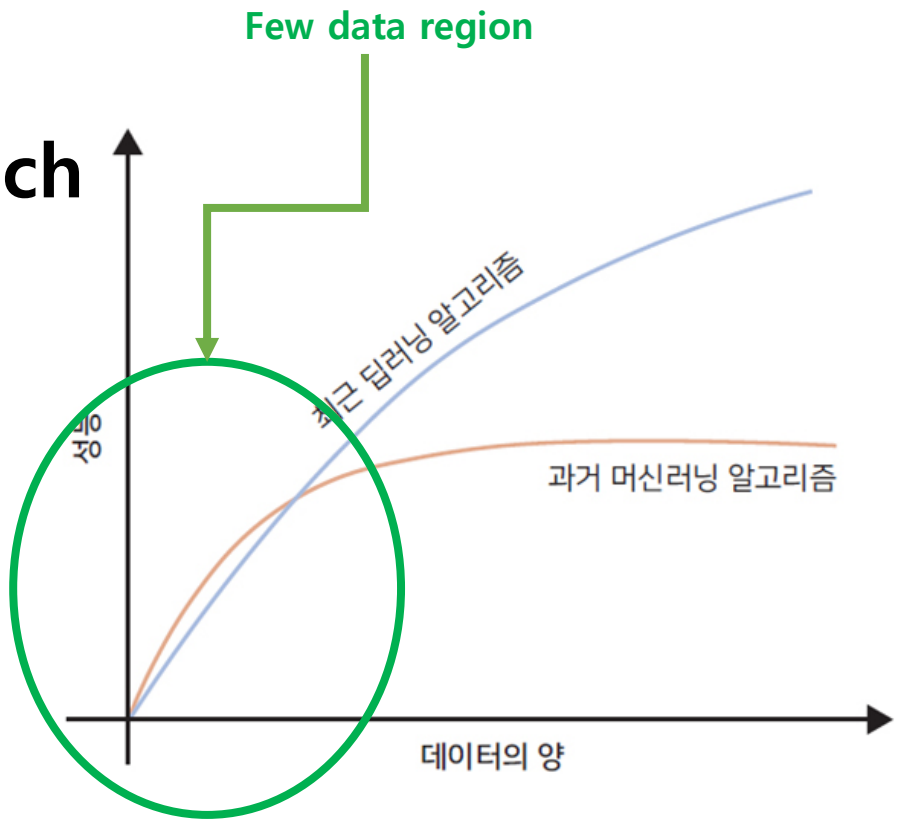
Using xgBoost for fewer datas instead of DNN

can be complementary method (Structured Data)

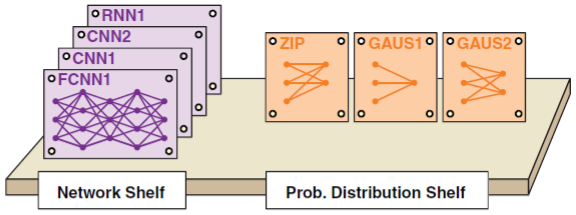
2. if you need parametric approach

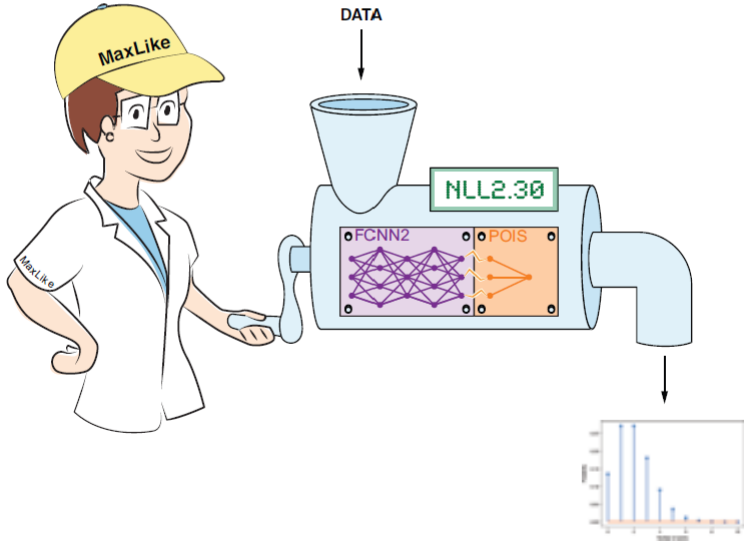
Support Vector Machine(SVM) instead of DNN

can be complementary method (Unstructured Data)



Probabilistic DL Summary





Summary

1. A probabilistic model predicts for each input a CPD
2. NLL measures how well the CPD matches so use it as loss function
3. Proper choice for CPD enhance the performance
4. For continuous, first choice is Normal
5. For count data, common choice is Poisson, Negative Binomial Dist, ZIP

3. Probabilistic DL Models in the Wild

Part 2: Real world distributions

Contents

1
Multinomial Distribution

Very flexible CPD for categorical data

Multinomial Distribution

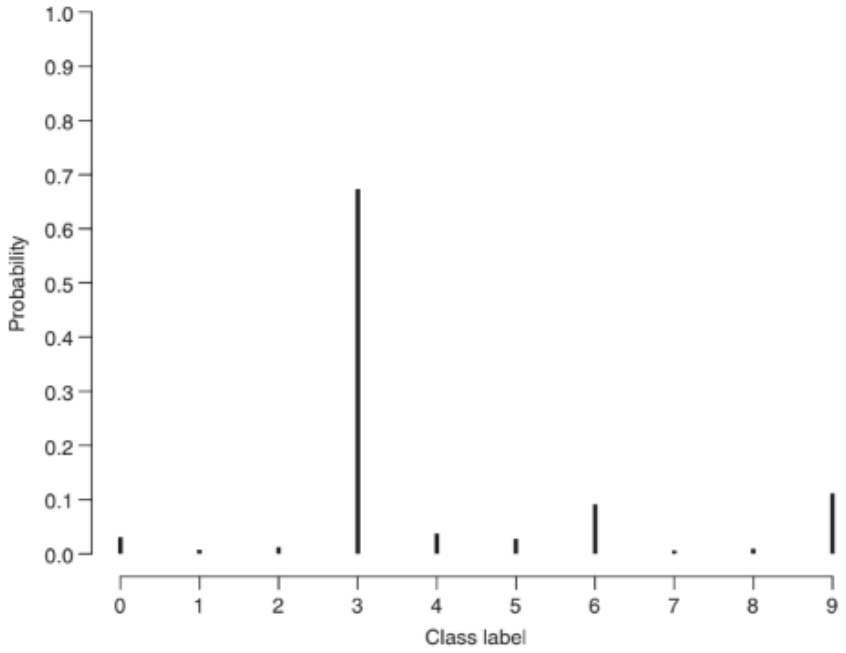


Figure 6.1 Multinomial distribution with ten classes: $MN(p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9)$

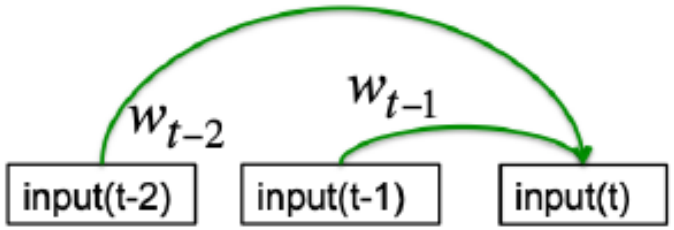
According to this criterion, the Multinomial distribution is especially flexible because it has as many parameters as possible values.

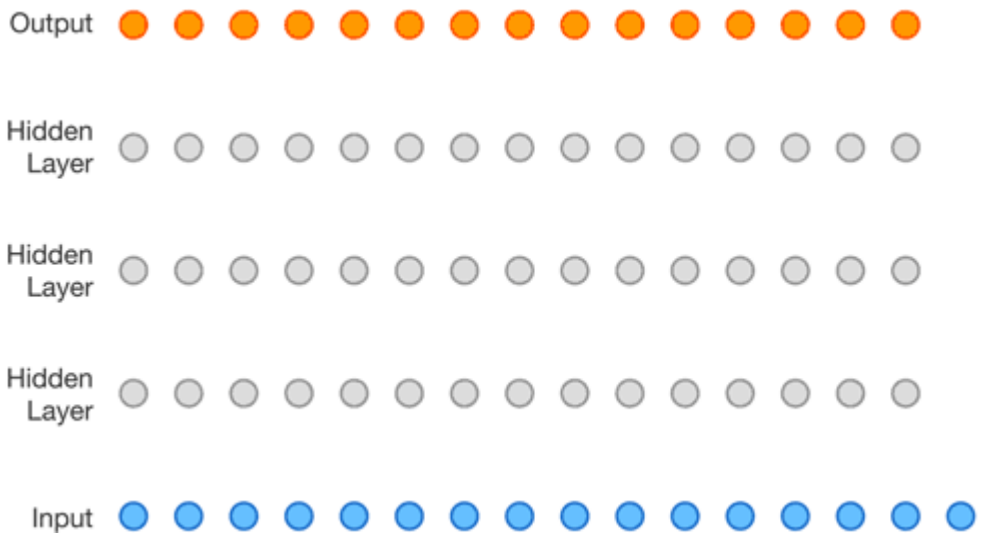
e.g. MNIST example - 9 params.

$$Y|X \sim Multinomial(p_1, p_2, \ldots, p_9)$$

Autoregressive Model – Multinomial as CPD

$$P(x_t) = P(x_t|x_{t-1}, x_{t-2}, \ldots x_0)$$





Autoregressive Models

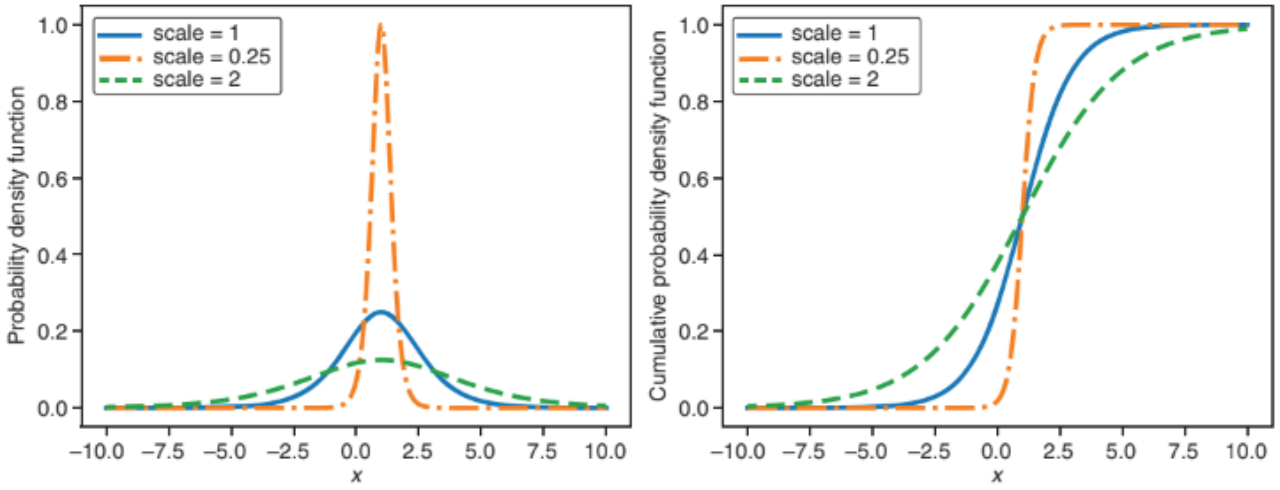
- Predict the next term in a sequence from a fixed number of previous terms using “delay taps”.
- Kind of memoryless models, unlike RNN(Recursive Neural Net).
- Can be generative model also.

2

Discretized Logistic Mixture Distribution

Less parameters than multinomial CPD, but still efficient

Logistic Distribution



$X \sim Logistic(\mu, s)$

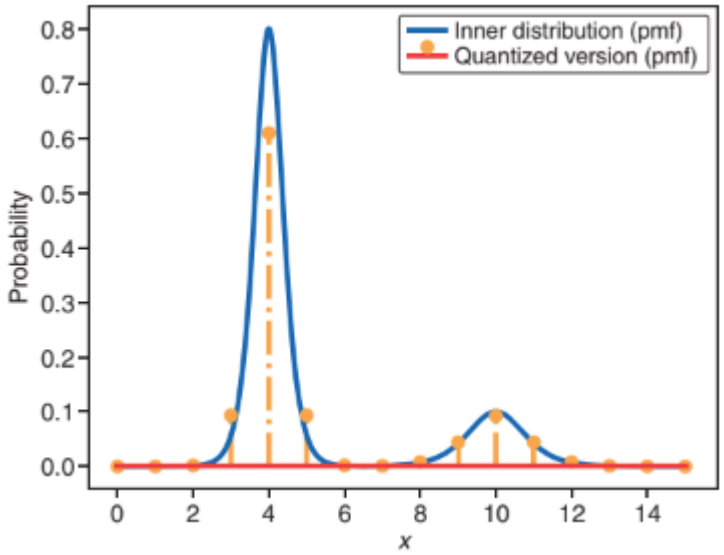
-pdf: $f(x;\mu,s) = \dfrac{e^{-(x-\mu)/s}}{s\left(1+e^{-(x-\mu)/s}\right)^2}$

-cdf: $F(x;\mu,s) = \dfrac{1}{1+e^{-(x-\mu)/s}}$

<Code Implement>

tfd. Logistic(loc= μ, scale = s)

Logistic Mixture Distribution



$X_1 \sim Logistic(\mu_1, s_1)$

$X_2 \sim Logistic(\mu_2, s_2)$

...

$X_n \sim Logistic(\mu_n, s_n)$

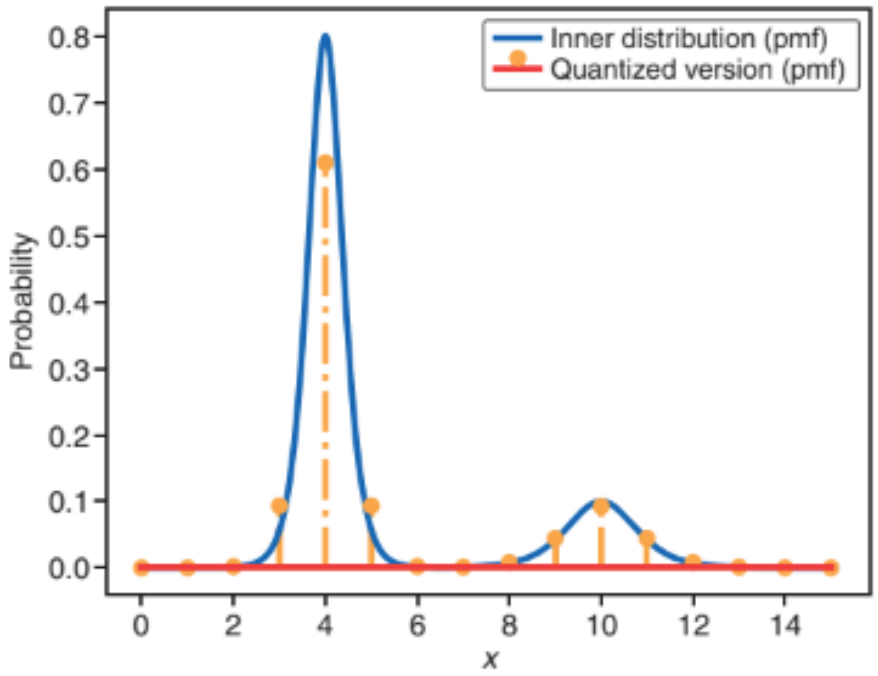
$$X = \sum p_i X_i$$

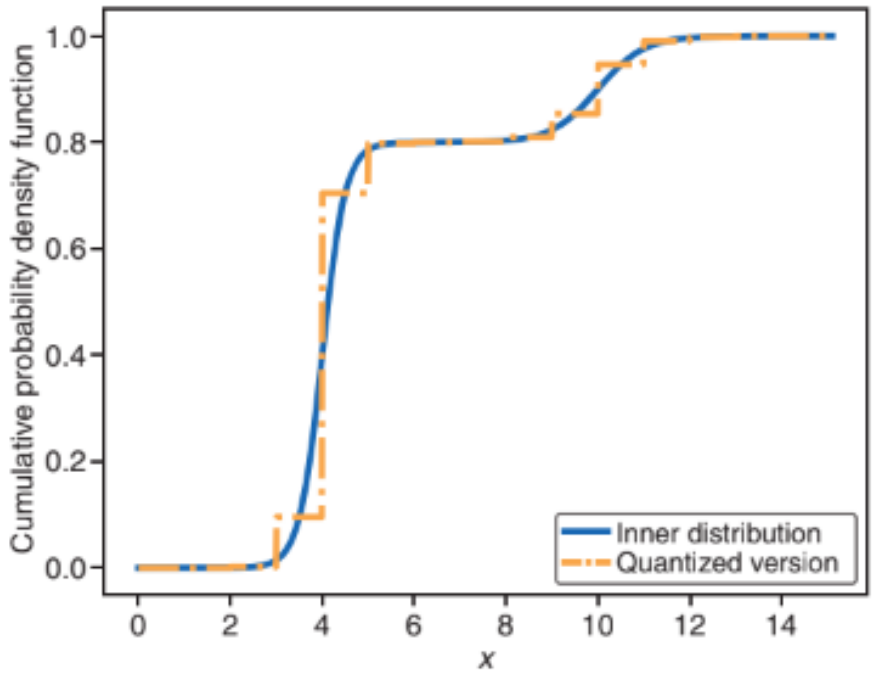
To specify r.v, *X*, *3n* parameters required.

By setting mixture distribution,

we can get simple yet flexible distribution.

Logistic Mixture Distribution



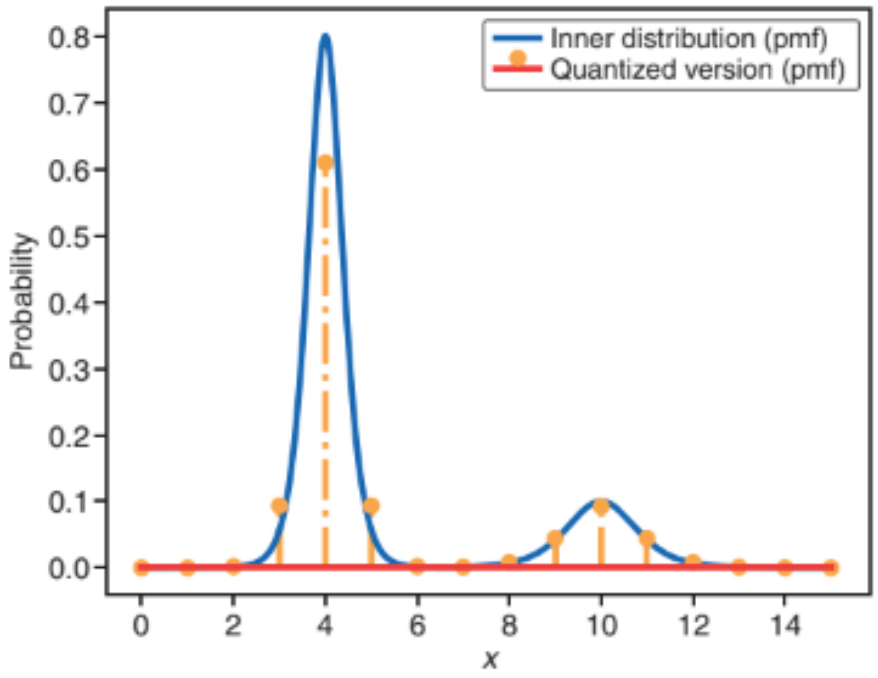


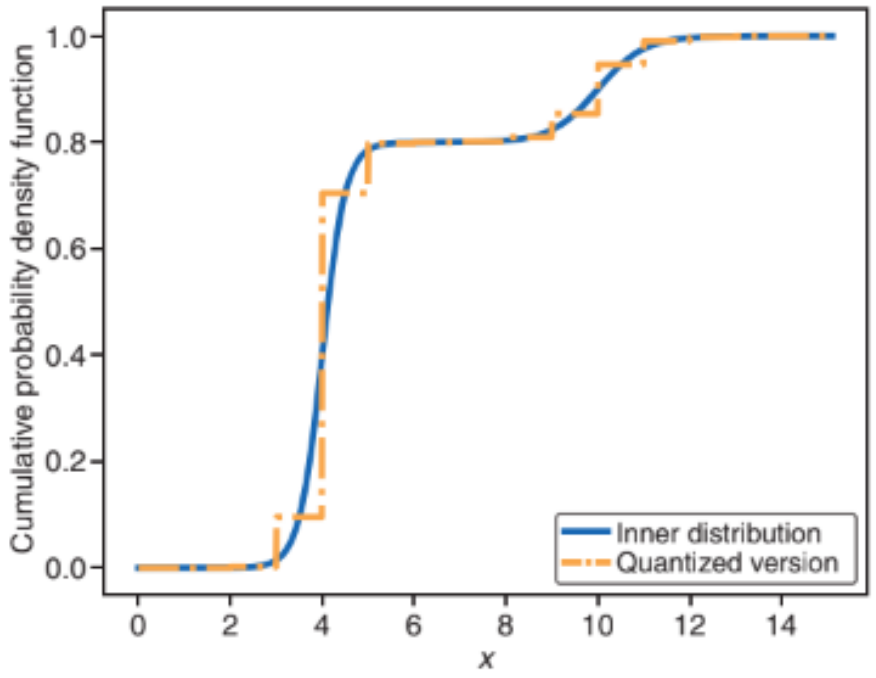
As output need to be discrete, we can discretize it!

(i.e. converting pdf to pmf)

Use NN to find parameters(MaxLike approach)

Why Not Gaussian Mixture?





Need CDF to discretize pdf

: Logistic is easier than Gaussian to use CDF

3
Normalizing Flow

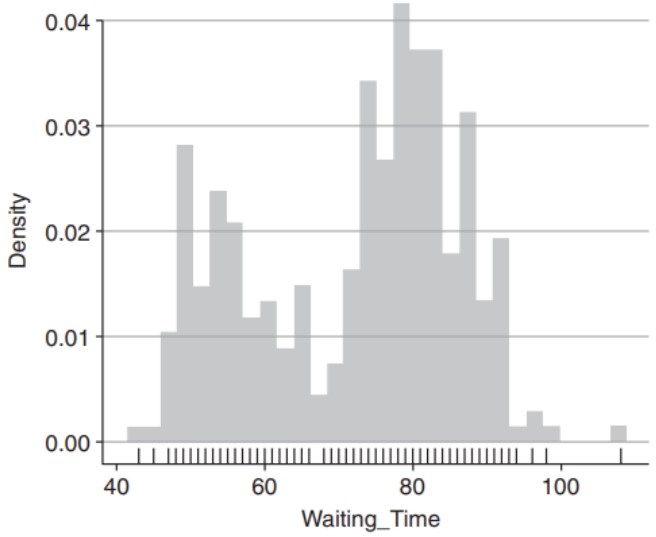
Modeling High Dimensional Distribution

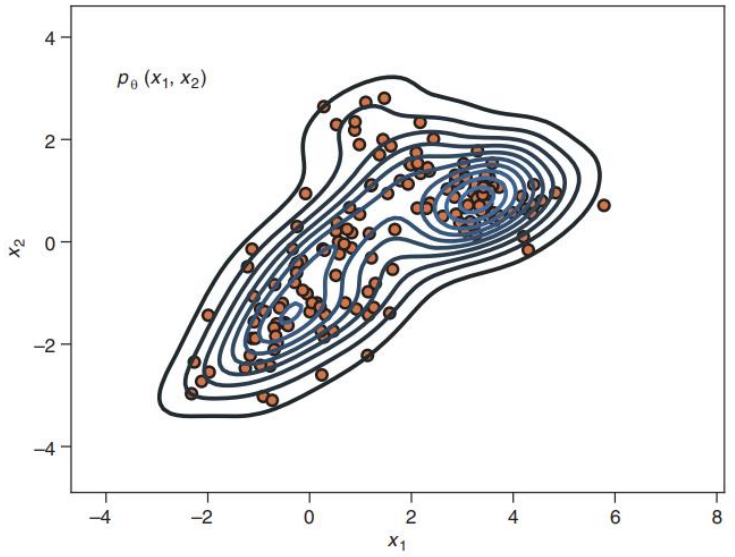
Normalizing Flow

NF의 목적은 복잡한 데이터의 분포를 간단한 분포로부터 변수 변환을 통해 얻어내는 것

1차원의 복잡한 분포는 Logistic Mixture 모델을 통해 구할 수 있었다.

하지만 Logistic Mixture 모델로는 고차원의 복잡한 분포를 설명하기 힘들다 → 따라서 나온 것이 NF!





Normalizing Flow – Main Topic

Variable Transformation : Indirect computation of likelihood & generating samples

Find Bijective Function : Based on MaxLike approach

Chain Flow : Stacking more bijective functions

Variable Transformation

Theorem 1.7.1. *Let X be a continuous random variable with pdf $f_X(x)$ and support $\mathcal{S}_X$. Let $Y = g(X)$, where $g(x)$ is a one-to-one differentiable function, on the support of X, $\mathcal{S}_X$. Denote the inverse of g by $x = g^{-1}(y)$ and let $dx/dy = d[g^{-1}(y)]/dy$. Then the pdf of Y is given by*

$$f_Y(y) = f_X(g^{-1}(y)) \left| \frac{dx}{dy} \right|, \quad \text{for } y \in \mathcal{S}_Y, \tag{1.7.11}$$

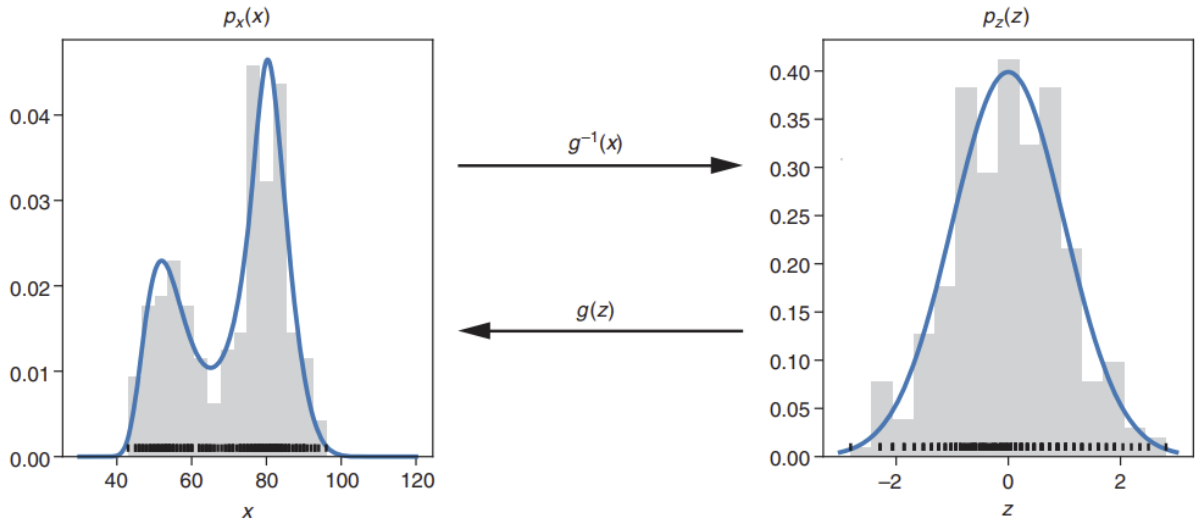
where the support of Y is the set $\mathcal{S}_Y = \{y = g(x) : x \in \mathcal{S}_X\}$.

Variable Transformation을 할 때, Jacobian이 추가로 곱해진다. 이 부분 덕분에 변환된 확률변수의 전체 넓이가 1이 될 수 있고, 이러한 특징 덕분에 Normalizing Flow라고 부른다.

One Dimensional NF

x : 주어진 데이터, **z** : 표준정규분포를 따르는 확률변수

x = g(z), z = g^{-1}(x)

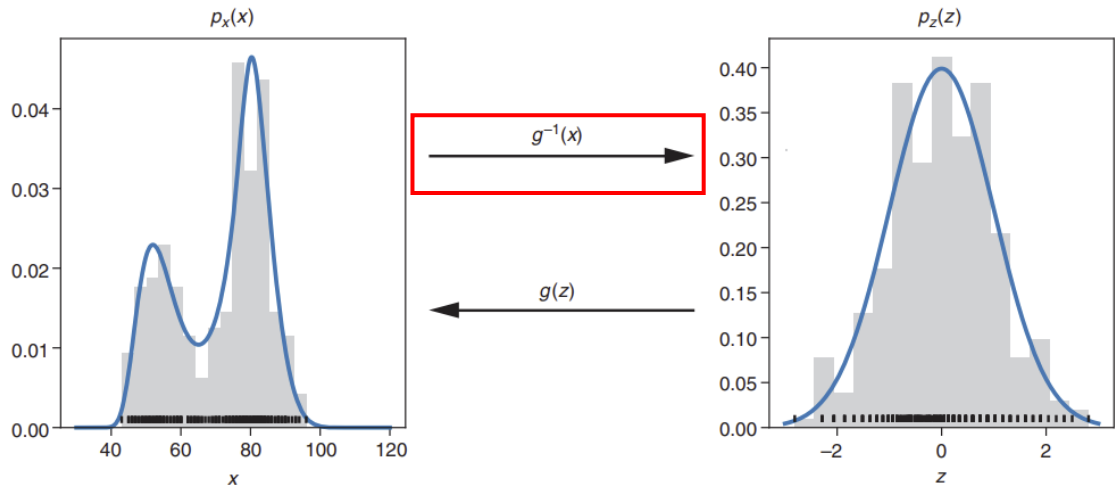


만약 x와 z의 관계를 잘 설명하는 bijective function g를 찾았다고 가정했을 때:

- 새로운 데이터 x_0의 likelihood를 알 수 있다.
- 실제론 존재하지 않는 데이터를 생성할 수 있다.

Why find g?

1. Likelihood 계산 가능

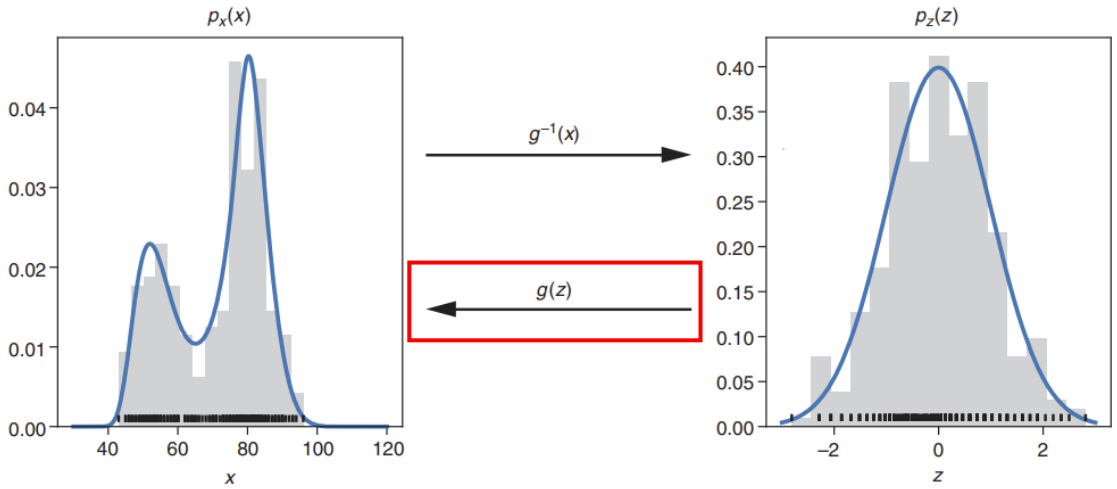


Given $\boldsymbol{x_0}$, we can't compute $\boldsymbol{p_x(x_0)}$ directly

$\boldsymbol{g^{-1}(x_0) = z_0 \rightarrow p_z(z_0)|J| = p_x(x_0)}$

Why find g?

2. Sampling 가능 (Data Generation)



You can't sample from $\boldsymbol{p_x(x)}$ directly

Sample from $\boldsymbol{p_z(z)}$ and than transform!

Bijective Function 찾기 – Data Fitting

Variable Transformation 기법을 쓰기 위해선 일단 Bijective function(일대일함수)이라는 조건이 전제

만약 하나의 함수만을 사용한다면, g를 어떠한 파라미터 $\boldsymbol{\theta}$에 의해 매우 flexible하게 변하는 함수로 설정을
하고, MaxLike approach를 통해 θ를 결정지을 수 있다.

만약 함수 g가 z와 x간의 관계를 잘 설명한다면, 주어진 데이터 $(\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_N})'$를 $(\boldsymbol{z_1}, \boldsymbol{z_2}, \ldots, \boldsymbol{z_N})'$로 변환한 후에 계산한 joint probability도 높은 값을 가질 것이다. 따라서 우리는 MaxLike approach를 통해 g를 구할 수 있게 된다.

→ $\boldsymbol{argmax_\theta \prod p_z(g^{-1}(x_i))}$

물론 하나의 함수만을 사용한다면, 아무리 그 함수가 flexible하다고 한들 한계가 있을 것이다.

→ Chaining Flows 도입

Bijective Function 찾기 – Data Fitting

<Code Implementation>

What if we want to find g which z ~ N(0, 1) & x ~ N(5, 0.2)

Listing 6.5 A simple example in TFP

```
a = tf.Variable(1.0)
b = tf.Variable(0.0)
bijector = tfb.AffineScalar(shift=a, scale=b)
dist = tfd.TransformedDistribution(distribution=
        tfd.Normal(loc=0,scale=1),bijector=bijector)

optimizer = tf.keras.optimizers.Adam(learning_rate=0.1)

for i in range(1000):
    with tf.GradientTape() as tape:
        loss = -tf.reduce_mean(dist.log_prob(X))
        gradients = tape.gradient(loss,
                    dist.trainable_variables)
    optimizer.apply_gradients(
        zip(gradients, dist.trainable_variables))
```

Defines the variables

Sets up the flow using an affine transformation defined by two variables

The NLL of the data

Calculates the gradients for the trainable variables

Applies the gradients to update the variables

Chaining Flows – One Dimensional

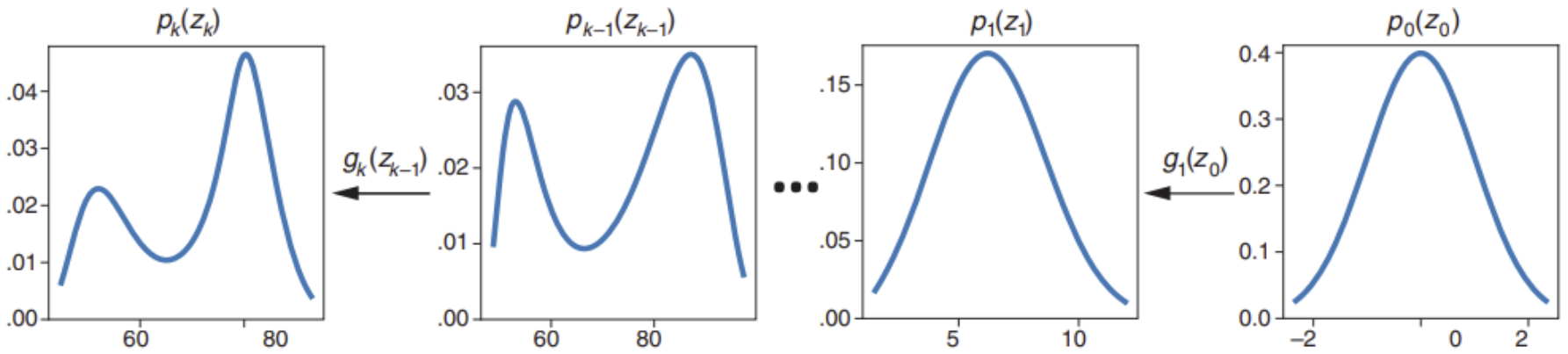


Figure 6.12 A chain of simple transformations makes it possible to create complex transformations needed to model complex distributions. From right to left, starting from a standard Gaussian distribution $z_0 \sim N(0,1)$ changes via successive transformations to a complex distribution with a bimodal shape (on the left).

변환을 여러 번 거치는 것을 Chaining flow라고 한다.

: 일반적으로 Linear Transformation 사이에 Non-linear Transformation을 끼워 넣는 식으로 Chain을 구축

Chaining Flows – One Dimensional

Linear Transformation: g(z) = az + b

- 분포의 모양을 바꾸진 않음.

(즉, z가 가우시안이면, g(z)도 여전히 가우시안을 따름)

따라서 우리는 복잡한 분포를 나타내기 위해

non-linear bijector function인 **SinhArcsinh** function을 이용

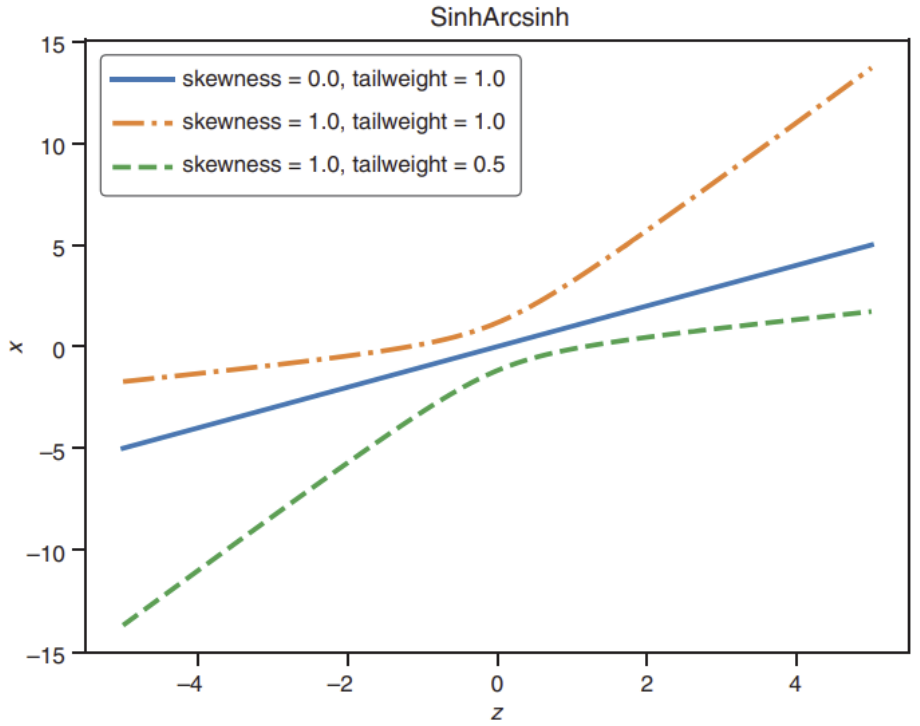


Figure 6.13 The bijector `SinhArcsinh` for different parameter values

Transformation Between High Dimensional Distributions

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{z_1} \\ \boldsymbol{z_2} \\ \dots \\ \boldsymbol{z_D} \end{bmatrix} \sim \boldsymbol{N_D(0, I_D)} \xrightarrow{\quad g \quad} \mathbf{x} = \begin{bmatrix} \boldsymbol{x_1} \\ \boldsymbol{x_2} \\ \dots \\ \boldsymbol{x_D} \end{bmatrix} = \boldsymbol{g(z)} = \begin{bmatrix} \boldsymbol{g_1(z_1, z_2, \dots, z_D)} \\ \boldsymbol{g_2(z_1, z_2, \dots, z_D)} \\ \dots \\ \boldsymbol{g_D(z_1, z_2, \dots, z_D)} \end{bmatrix}$$

Transformation $$p_x(x) = p_z(z) \cdot \left| \det \left(\frac{\partial g(z)}{\partial z} \right) \right|^{-1}$$

Jacobian Matrix

How to find function g in high dimensions

1. Fully Triangular Flow

2. Real Non-Volume Preserving Flow (Real NVP)

Fully Triangular Flow

차원이 높아질수록 Jacobi Matrix의 determinant 계산이 복잡해진다.

→ 따라서 Jacobi Matrix가 삼각행렬이 될 수 있도록 함수 g를 설정해줄 것이다.

$$\frac{\partial g(z)}{\partial z} = \begin{pmatrix} \frac{\partial g_1(z_1,z_2,z_3)}{\partial z_1} & \frac{\partial g_1(z_1,z_2,z_3)}{\partial z_2} & \frac{\partial g_1(z_1,z_2,z_3)}{\partial z_3} \\ \frac{\partial g_2(z_1,z_2,z_3)}{\partial z_1} & \frac{\partial g_2(z_1,z_2,z_3)}{\partial z_2} & \frac{\partial g_2(z_1,z_2,z_3)}{\partial z_3} \\ \frac{\partial g_3(z_1,z_2,z_3)}{\partial z_1} & \frac{\partial g_3(z_1,z_2,z_3)}{\partial z_2} & \frac{\partial g_3(z_1,z_2,z_3)}{\partial z_3} \end{pmatrix} \Rightarrow \frac{\partial g(z)}{\partial z} = \begin{pmatrix} \frac{\partial g_1(z)}{\partial z_1} & 0 & 0 \\ \frac{\partial g_2(z)}{\partial z_1} & \frac{\partial g_2(z)}{\partial z_2} & 0 \\ \frac{\partial g_3(z)}{\partial z_1} & \frac{\partial g_3(z)}{\partial z_2} & \frac{\partial g_3(z)}{\partial z_3} \end{pmatrix}$$

삼각행렬을 만들기 위해서는 $\boldsymbol{\frac{\partial g_1(z_1,z_2,z_3)}{\partial z_2} = 0, \frac{\partial g_1(z_1,z_2,z_3)}{\partial z_3} = 0, \frac{\partial g_2(z_1,z_2,z_3)}{\partial z_2} = 0}$

이는 i번째 함수 $\boldsymbol{g_i}$는 $\boldsymbol{z_j}$ $(\boldsymbol{j > i})$를 포함하지 않아야 한다는 것을 의미한다.

Fully Triangular Flow

$$\frac{\partial g(z)}{\partial z} = \begin{pmatrix} \frac{\partial g_1(z)}{\partial z_1} & 0 & 0 \\ \frac{\partial g_2(z)}{\partial z_1} & \frac{\partial g_2(z)}{\partial z_2} & 0 \\ \frac{\partial g_3(z)}{\partial z_1} & \frac{\partial g_3(z)}{\partial z_2} & \frac{\partial g_3(z)}{\partial z_3} \end{pmatrix}$$

삼각행렬의 Determinant : 대각 원소들의 곱

→ g를 앞서 보았던 기준에 따라 설정한다면, determinant 계산을 위해 3번만 편미분을 진행하면 된다(D차원 분포라면 D번 편미분)

→ 조금 더 자세히 보면, i번째 함수 g_i가 z_i에 의해 편미분되는 것만 고려하면 된다

이때, g_i가 z_i에 Linear한 함수라면(i.e., $g_i(z) = az_i + b$), 편미분이 간편해지기 때문에 이 점을 반영하여 함수를 설정할 것이다.

따라서 만약 D차원 분포를 다루는 상황이라면, g_i를 다음과 같이 나타낼 수 있다

→ $g_i(z_1, z_2, \dots, z_D) = g_i(z_1, z_2, \dots, z_i) = b_i(z_1, z_2, .., z_{i-1}) + exp\big(\alpha_i(z_1, z_2, .., z_{i-1})\big) * z_i$

이렇게 설정하면, 비록 z_i에 대해서는 linear transformation일지 몰라도, slope와 bias를 non-linear한 방식으로 계산할 수 있기에 매우 flexible하게 함수를 fitting할 수 있다.

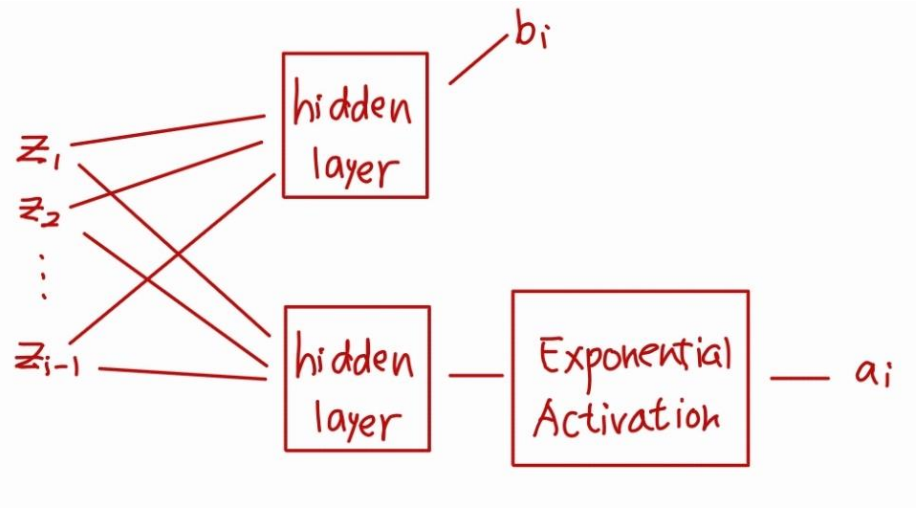
Fully Triangular Flow

$$\boldsymbol{g_i(z_1, z_2, \dots, z_D) = g_i(z_1, z_2, \dots, z_i) = b_i(z_1, z_2, .., z_{i-1}) + exp\big(\alpha_i(z_1, z_2, .., z_{i-1})\big) * z_i}$$

i 번째 slope: $exp\big(\alpha_i(z_1, z_2, .., z_{i-1})\big)$

i 번째 bias: $b_i(z_1, z_2, .., z_{i-1})$

Use Neural Network to get slope and bias!



Why Exponential Activation?

- If slope = 0, then determinant is zero

Fully Triangular Flow - Data Fitting

$$\boldsymbol{p_x(x) = p_z\left(g^{-1}(x)\right)\left|det\left(\frac{\partial g(z)}{\partial z}\right)\right|^{-1}}$$

1차원 분포에서와 마찬가지로 MaxLike 방법을 사용해서 g의 파라미터들을 찾을 것이다.

위의 식의 우변 부분을 Maximize하기 위해서는 $z = g^{-1}(x)$를 알아야 한다.

z를 한번에 다 계산할 수는 없으므로 아래와 같이 sequential한 방법으로 계산해야 한다.

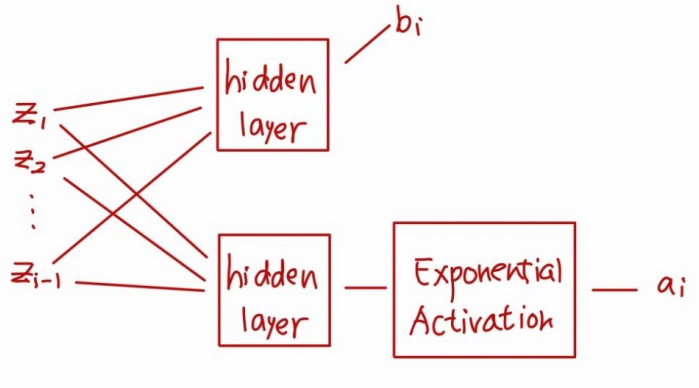
$$z_1 = x_1$$
$$z_2 = \frac{x_2 - b_2(z_1)}{\exp(\alpha_2(z_1))}$$
$$z_3 = \frac{x_3 - b_3(z_1, z_2)}{\exp(\alpha_3(z_1, z_2))}$$
$$z_4 = \frac{x_4 - b_4(z_1, z_2, z_3)}{\exp(\alpha_4(z_1, z_2, z_3))}$$

Real Non-volume preserving flow (Real NVP)

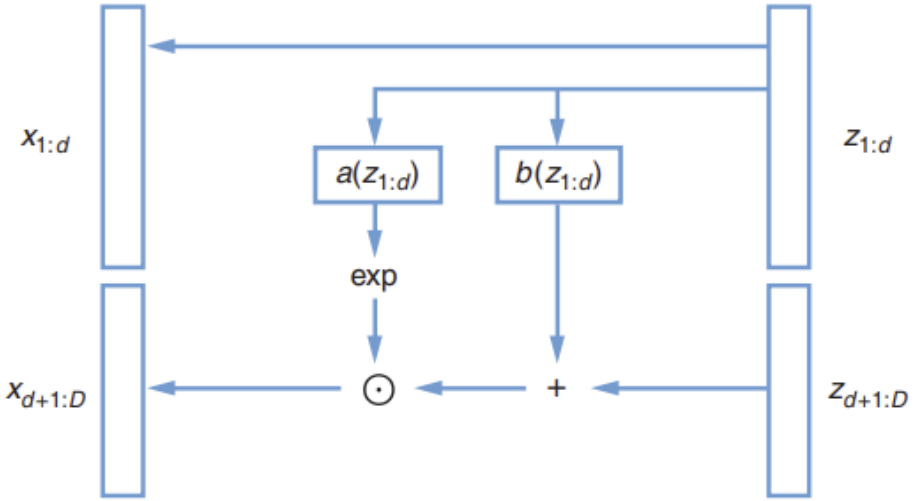
오른쪽 NN은 하나의 함수 g_i를 구하기 위한 NN이다.

따라서 전체 함수들을 다 구해주기 위해선 D개의 NN이 필요

➔ 모두 fitting시키는 건 많은 시간이 소요된다.



대신에 조금 덜 flexible하지만 빠른 계산이 가능한 **Real NVP**라는 것을 사용하기도 한다.



$$
\begin{aligned}
x_1 &= g_1(z_1) = z_1 \\
x_2 &= g_2(z_2) = z_2 \\
x_3 &= g_3(z_1, z_2, z_3) = b_3(z_1, z_2) + \exp(\alpha_3(z_1, z_2)) \cdot z_3 \\
x_4 &= g_4(z_1, z_2, z_4) = b_4(z_1, z_2) + \exp(\alpha_4(z_1, z_2)) \cdot z_4 \\
x_5 &= g_5(z_1, z_2, z_5) = b_5(z_1, z_2) + \exp(\alpha_5(z_1, z_2)) \cdot z_5
\end{aligned}
$$

$$
\begin{aligned}
z_1 &= x_1 \\
z_2 &= x_2 \\
z_3 &= \frac{x_3 - \mu_1(z_1, z_2)}{\exp(\alpha_3(z1, z2))} \\
z_4 &= \frac{x_4 - \mu_2(z1, z2)}{\exp(\alpha_4(z1, z2))} \\
z_5 &= \frac{x_5 - \mu_3(z1, z2)}{\exp(\alpha_5(z1, z2))}
\end{aligned}
$$

$$
\frac{\partial g}{\partial z} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
\frac{\partial g_3}{\partial z_1} & \frac{\partial g_3}{\partial z_2} & e^{\alpha_3} & 0 & 0 \\
\frac{\partial g_4}{\partial z_1} & \frac{\partial g_4}{\partial z_2} & 0 & e^{\alpha_4} & 0 \\
\frac{\partial g_5}{\partial z_1} & \frac{\partial g_5}{\partial z_2} & 0 & 0 & e^{\alpha_5}
\end{pmatrix}
$$

Stack More Layers

1차원 분포에서 봤듯이 여러 번 변환을 거칠 수도 있다!

Listing 6.7 The simple example of a Real NVP TFP

```
bijectors=[]
num_blocks = 5
h = 32
for i in range(num_blocks):
    net = tfb.real_nvp_default_template(
                [h, h])
    bijectors.append(
        tfb.RealNVP(shift_and_log_scale_fn=net,
                    num_masked=num_masked))
    bijectors.append(tfb.Permute([1,0]))
    self.nets.append(net)
bijector = tfb.Chain(list(reversed(bijectors[:-1])))

self.flow = tfd.TransformedDistribution(
    distribution=tfd.MultivariateNormalDiag(loc=[0., 0.]),
    bijector=bijector)
```

Adds num_blocks of coupling permutations to the list of bijectors

Number of hidden layers in the NF model

Size of the hidden layers

Defines the network

A shift and flow with parameters from the network

Permutation of coordinates

Distribution of z with two independent Gaussians

감사합니다