# APPLICATION DEEP LEARNING GLOBAL FITTING

機械學習 迴歸 分析

Reporter 劉友安

Helper 林謙

Teacher 蔡岳霖

# Machine Learning (Gradient Descent)

- 將變量經過線性組合後，帶入非線性活化函數，活化函數的輸出值作為下一層的輸入變量
- 將最後輸出量和訓練資料(ex:one-hot)，帶入loss function進行比對，算出loss function 對每個權重的微分(梯度)，接著乘上學習率，乘上負號加到原本的權重上，也就是在每一次的梯度計算，都逐步調整權重，使最後輸出的loss function達到最小值。達成目標，便是優化完成。[我們尚未找到證明，可以確保這個步驟收斂，狀況比較接近是" 經驗上會收斂" ]
- 偽代碼(單層示意)：

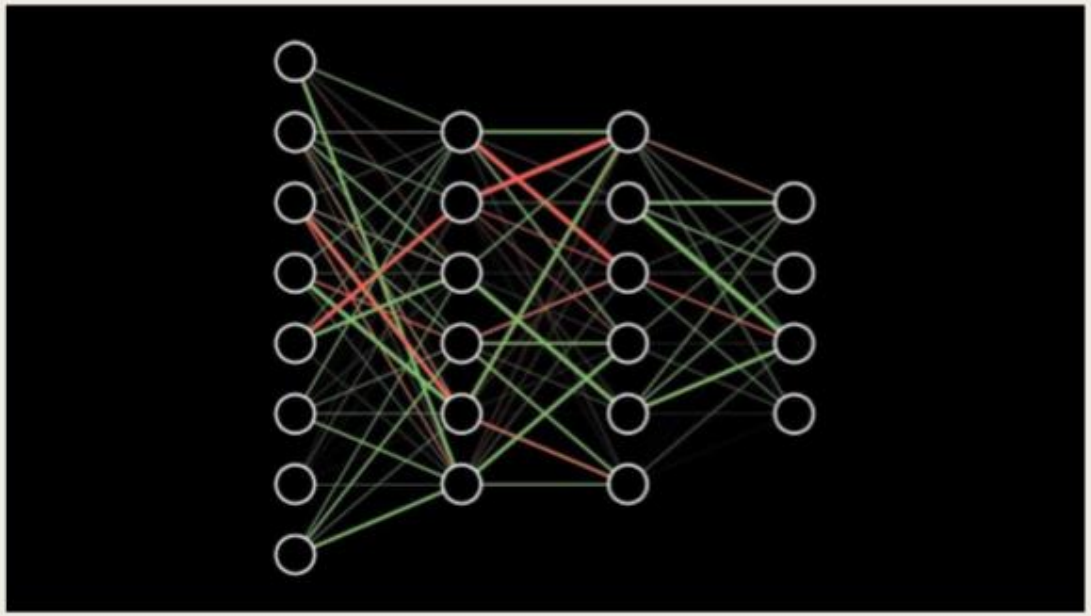
for (int i=0;i<max_epoch;i++){

$$w = w - \frac{\partial(loss\_function(activation\_func(\_w^T x)))}{\partial w} h\ ;$$

}

## w 權重(透過normal distribution 初始化)

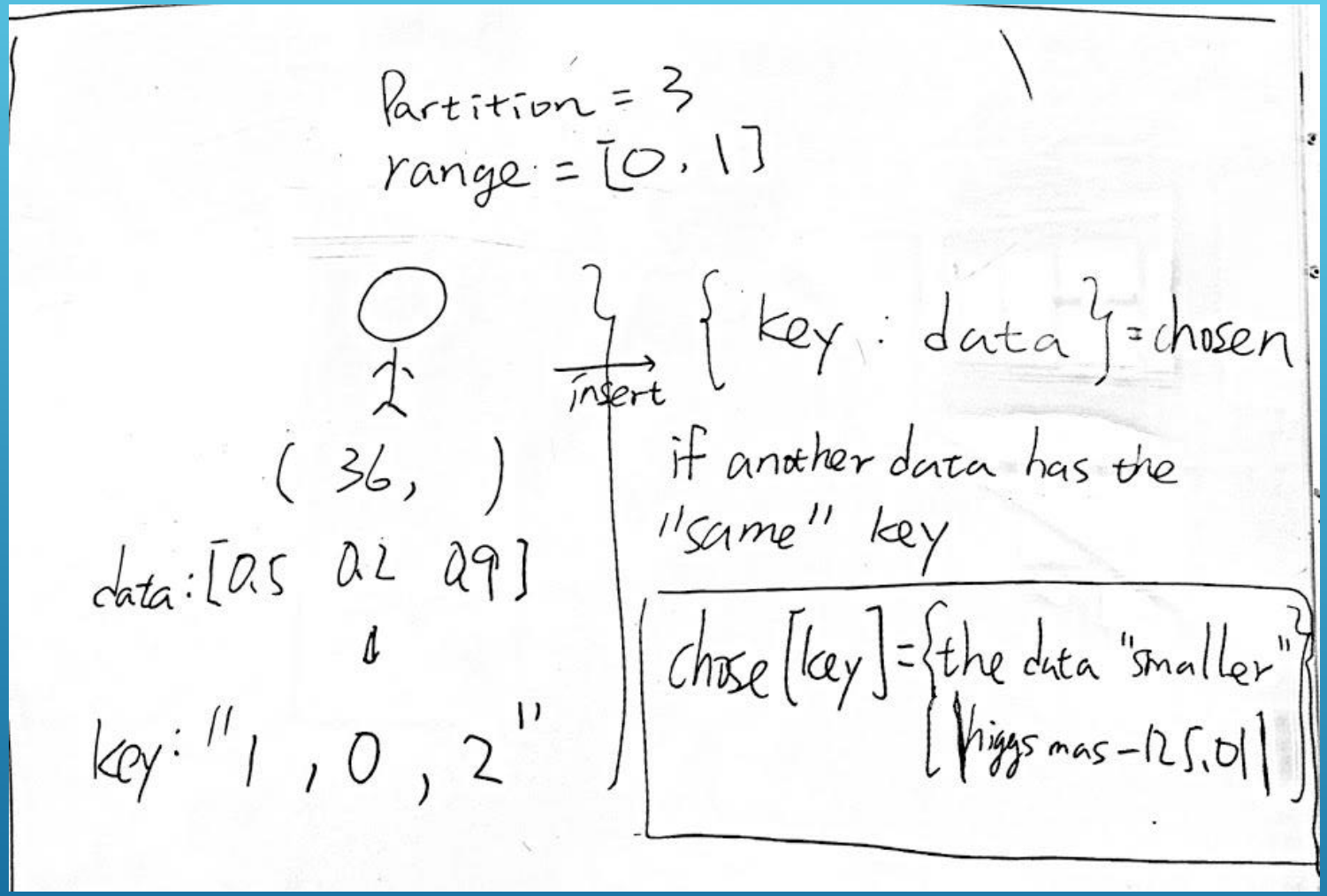
## h 學習率



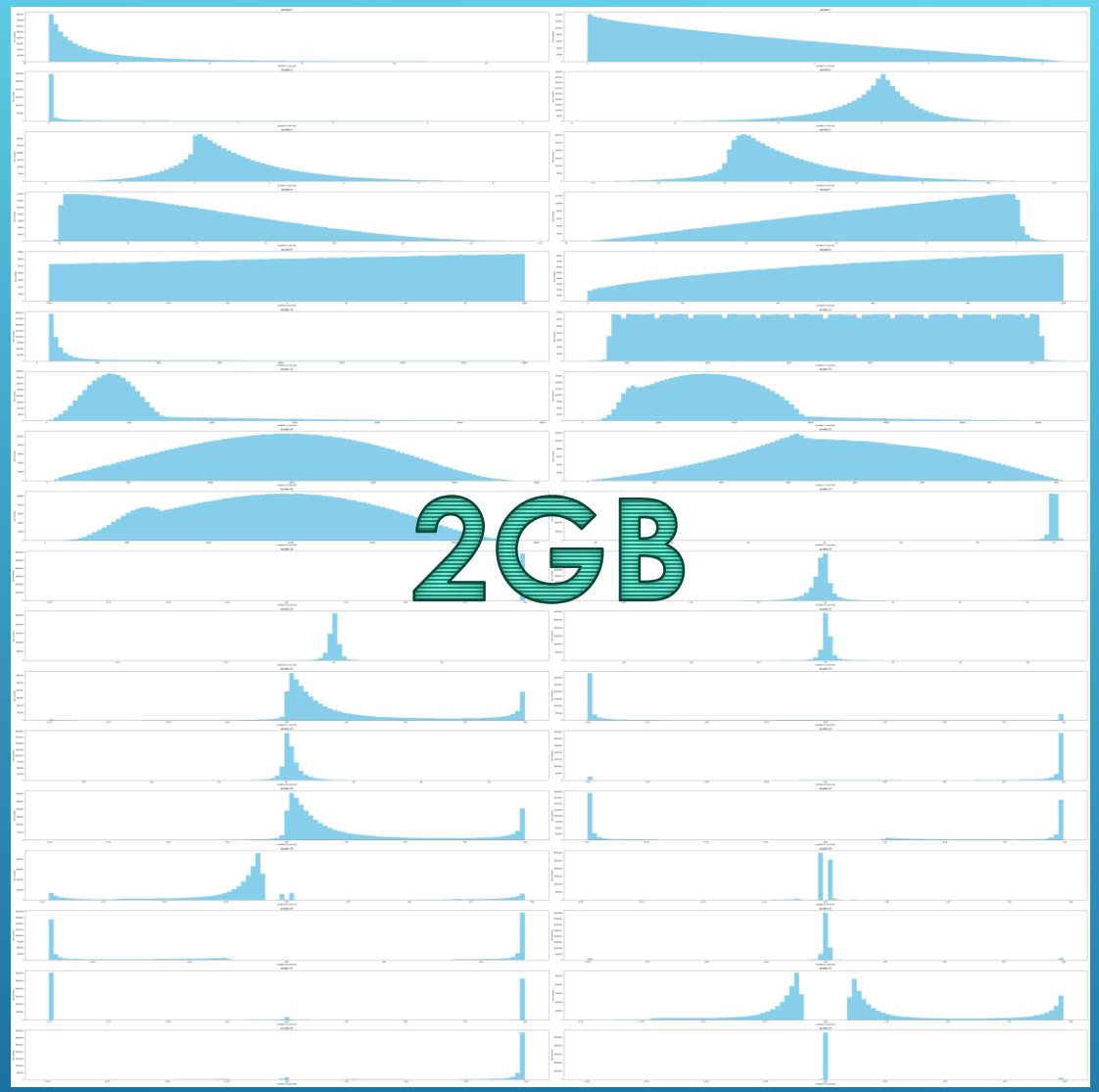
# CONTENT

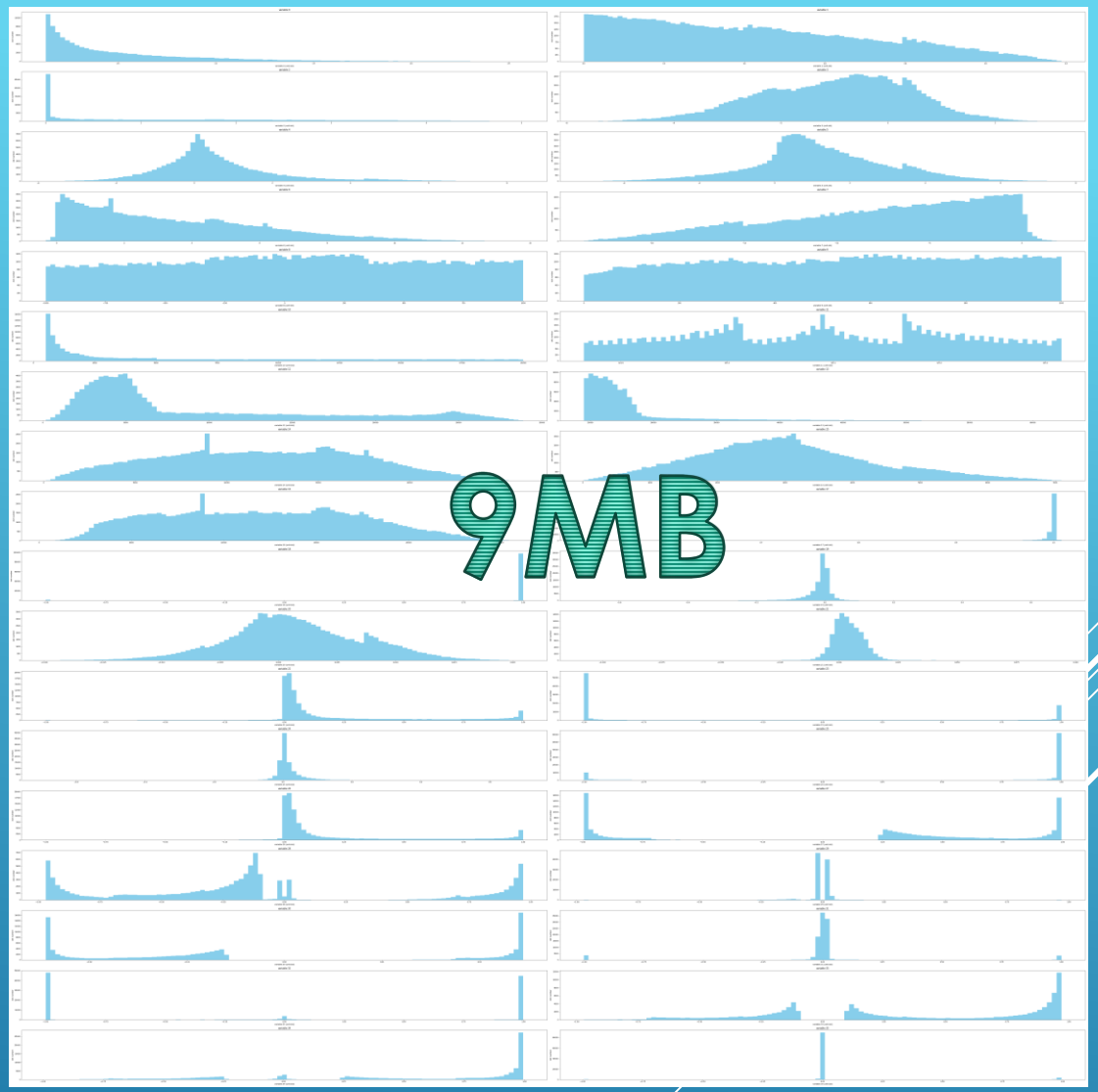
- (1)Data Thinning
- (2)Model Training
- Appendix : SVM--Support Vector Machine

# (1)Data Thinning




→ DATA THINNING BY HASH TABLE




# → RESULT

# (2) MODEL TRAINING

1.Normalise

2.None

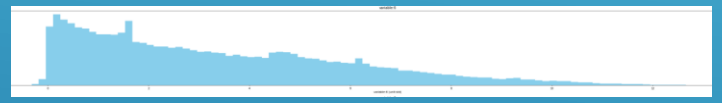
3.Exp

4.Log

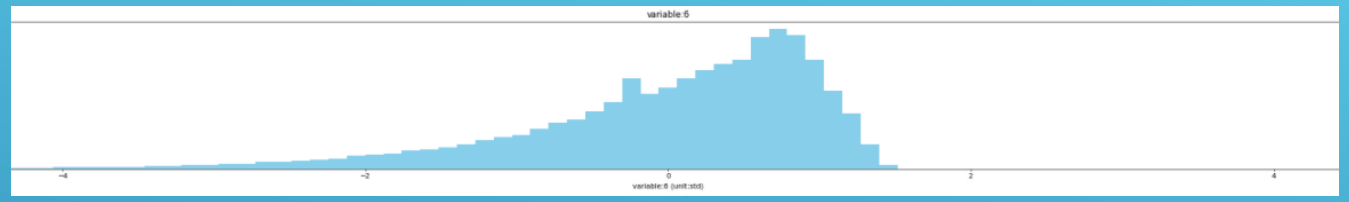
# Preprocessing

# (2) MODEL TRAINING

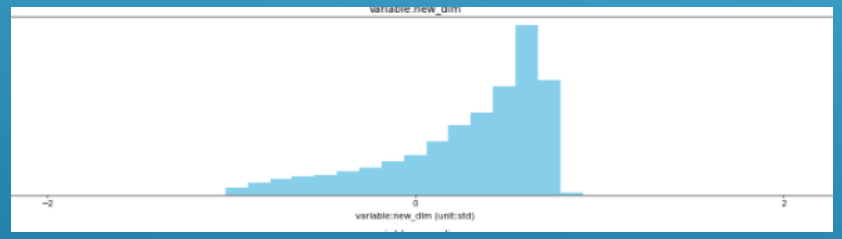
## 5.Negative_2_dim



Log(|data|)




tanh(data)




# Preprocessing

# (2) MODEL TRAINING

```
output_layer = Dense( units = 100 , input_dim = input_array.shape[1]
                    , kernel_initializer = normal, bias_initializer=normal , activation = 'relu')
#BN_1 = BatchNormalization(axis=-1, momentum=0.99, epsilon=0.01, center=True, scale=True, beta_init
dense_3 = Dense(units = 1000 , input_dim = inputLen , kernel_initializer= normal,
                bias_initializer=normal , activation = 'relu')
#dense_2 = Dense(units = 100, kernel_initializer=normal,
                bias_initializer=normal , activation = 'relu')
dense_1 = Dense( units = output_array.shape[1] , kernel_initializer= normal,
                activation = 'linear')
layer_list = [output_layer,dense_3,dense_1 ]
```

# 25 to 11 Output to Input

# (2) MODEL TRAINING

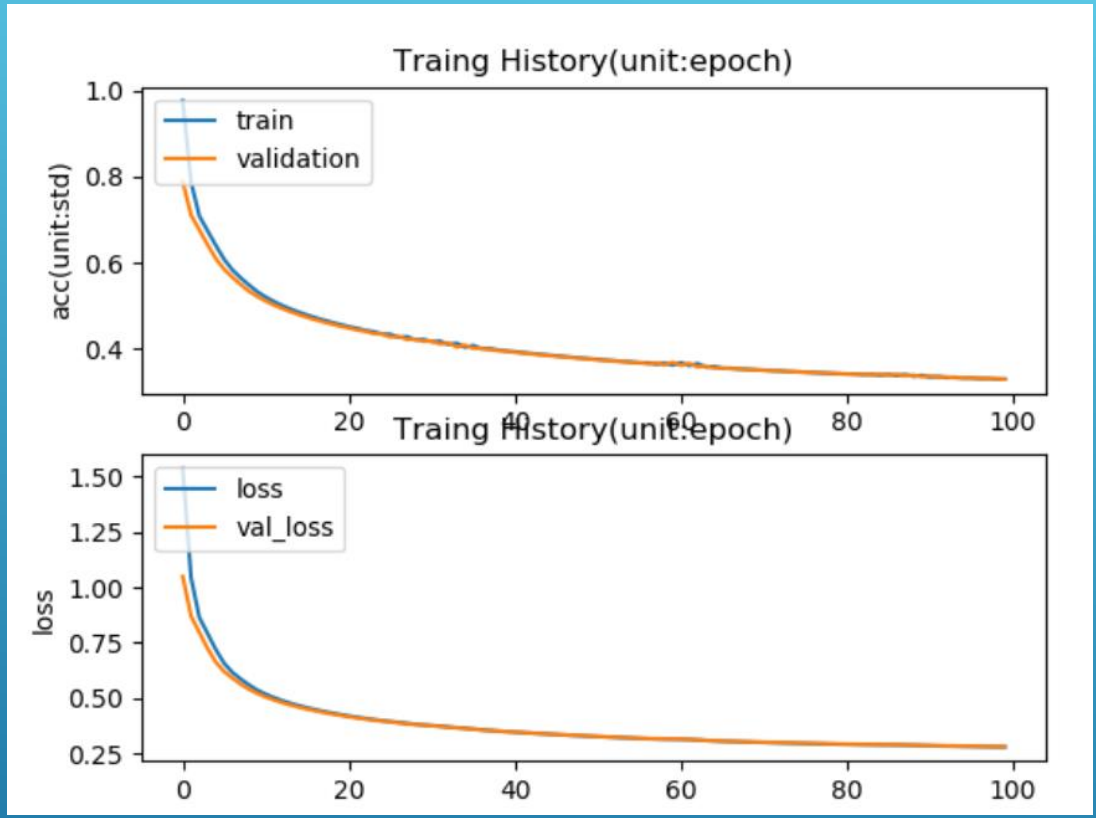
```
normal = initializers.RandomNormal(mean=0.0, stddev=0.1, seed=10)
uniform = initializers.RandomUniform(minval=-0.3, maxval=0.3, seed=10)

output_layer = Dense( units = 100 , input_dim = input_array.shape[1]
                    , kernel_initializer = normal, bias_initializer=normal , activation = 'relu')
#BN_1 = BatchNormalization(axis=-1, momentum=0.99, epsilon=0.01, center=True, scale=True, beta_init
dense_3 = Dense(units = 1000 , input_dim = inputLen , kernel_initializer= normal,
                bias_initializer=normal , activation = 'relu')
#dense_2 = Dense(units = 100, kernel_initializer=normal,
#                bias_initializer=normal , activation = 'relu')
dense_1 = Dense( units = output_array.shape[1] , kernel_initializer= normal,
                activation = 'linear')
layer_list = [output_layer,dense_3,dense_1 ]
```

```
adam = optimizers.Adam(lr=0.001, beta_1=0.5, beta_2=0.999, epsilon=None, decay=0.02, amsgrad=False)
model.compile(optimizer=adam,
              loss= 'mse',
              metrics=['mae'])
```

# 25 to 11 Output to Input

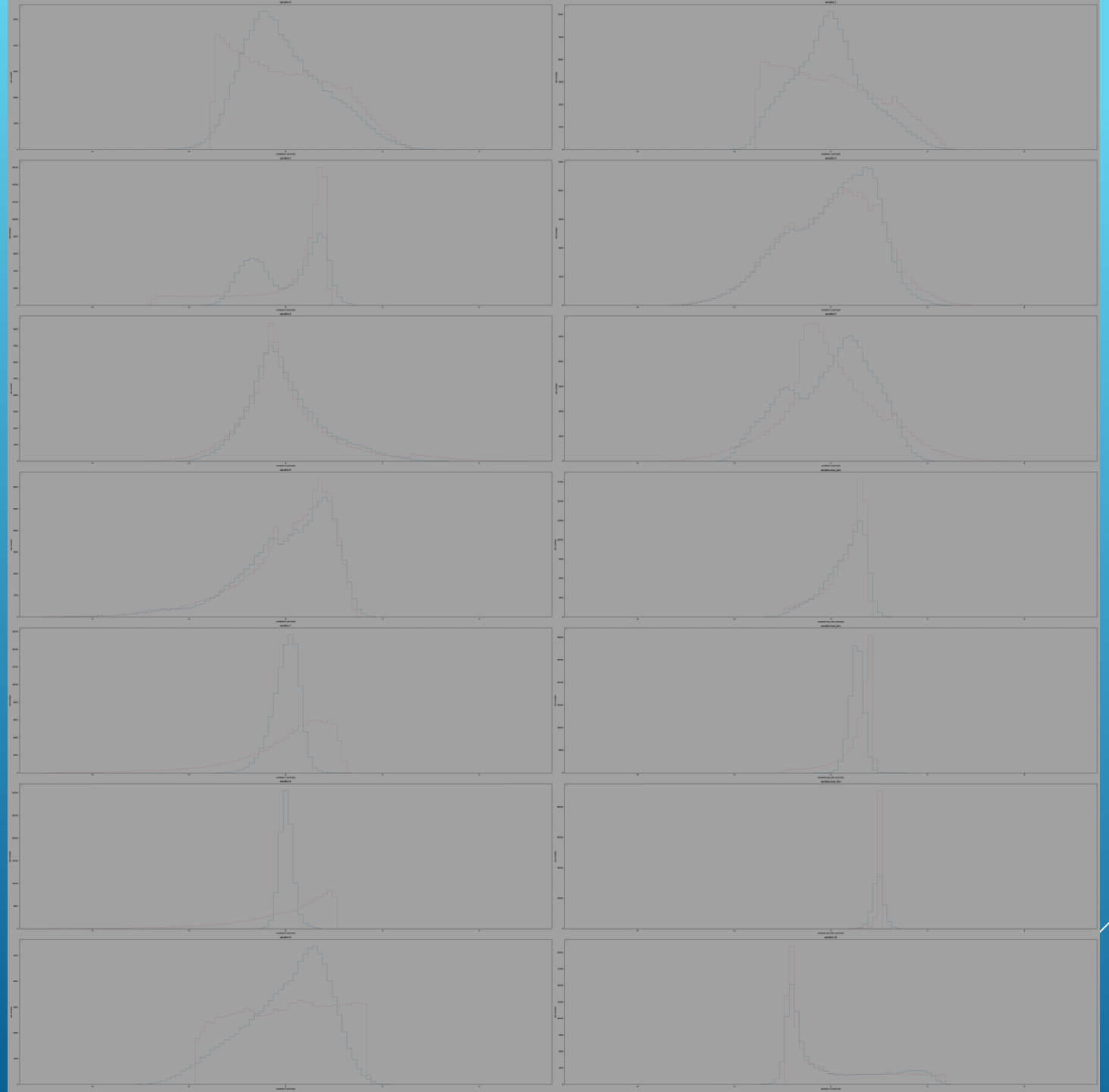
# (2) MODEL TRAINING



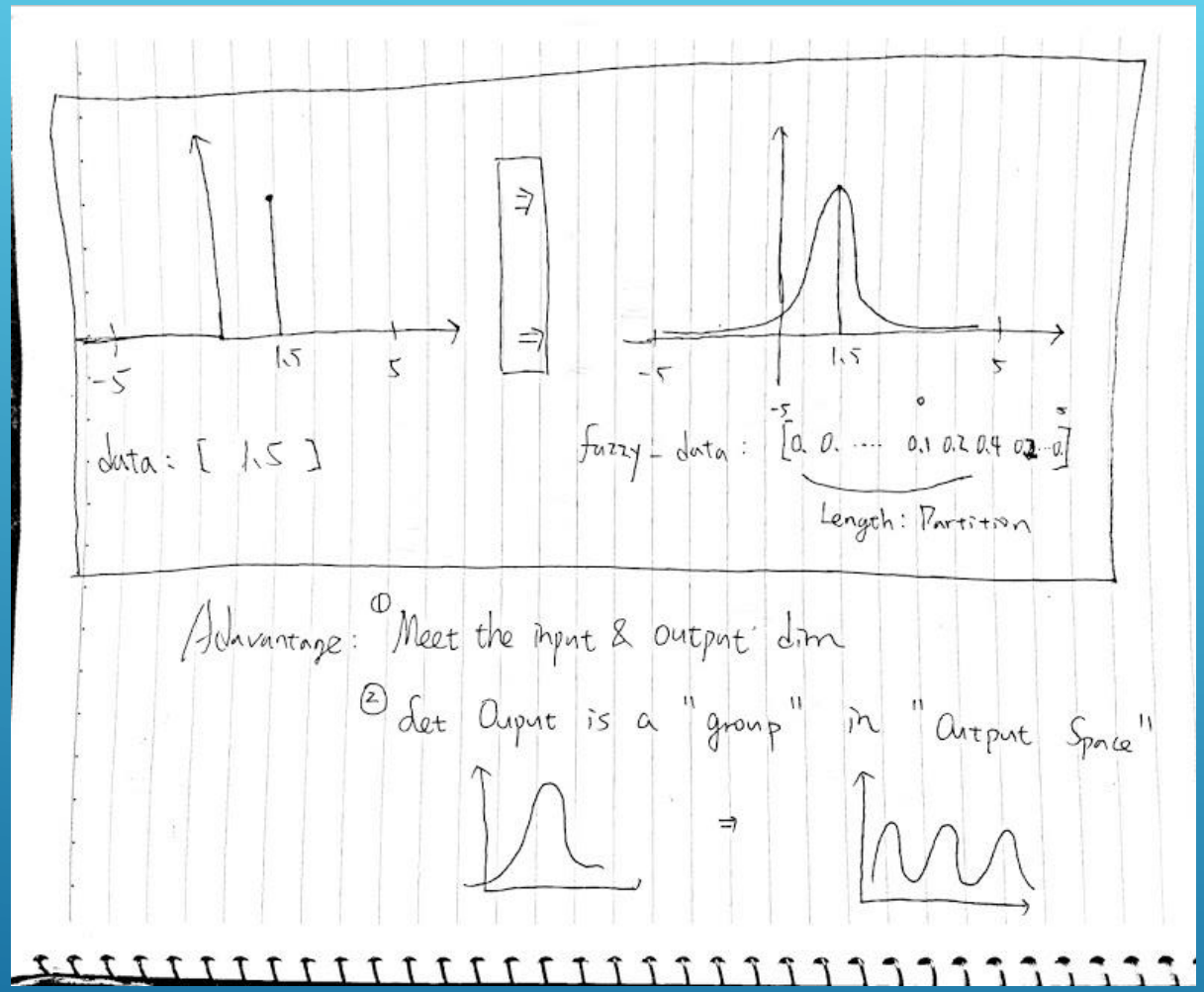

# 25 to 11 Output to Input

# (2) MODEL TRAINING

25 to 11 Output to Input



# (2) MODEL TRAINING



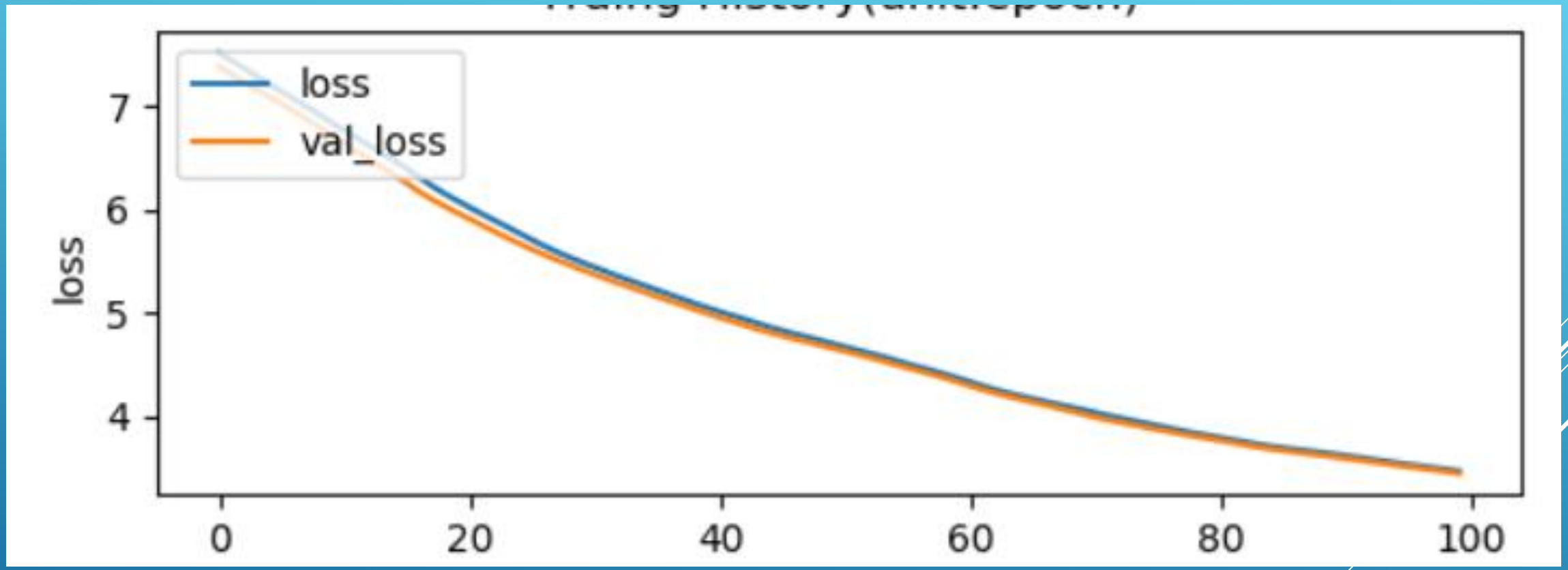

# 2 to 11 Fuzzy Training

# (2) MODEL TRAINING

```
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 120)               0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               12100
_________________________________________________________________
dense_2 (Dense)              (None, 100)               10100
_________________________________________________________________
dense_3 (Dense)              (None, 140)               14140
_________________________________________________________________
reshape_1 (Reshape)          (None, 14, 10)            0
=================================================================
Total params: 36,340
Trainable params: 36,340
Non-trainable params: 0
```

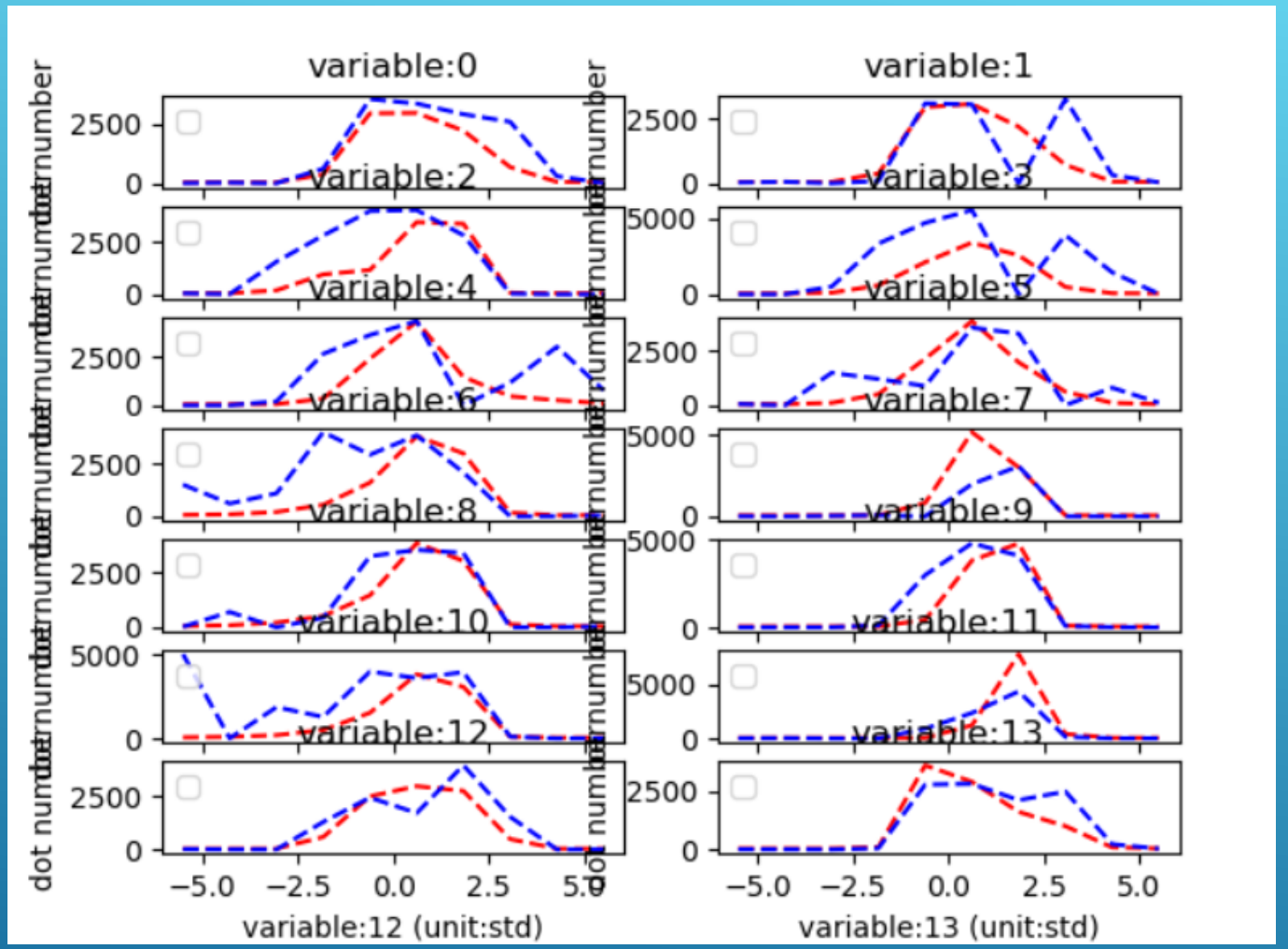
# 2 to 11 Fuzzy Training

# (2) MODEL TRAINING



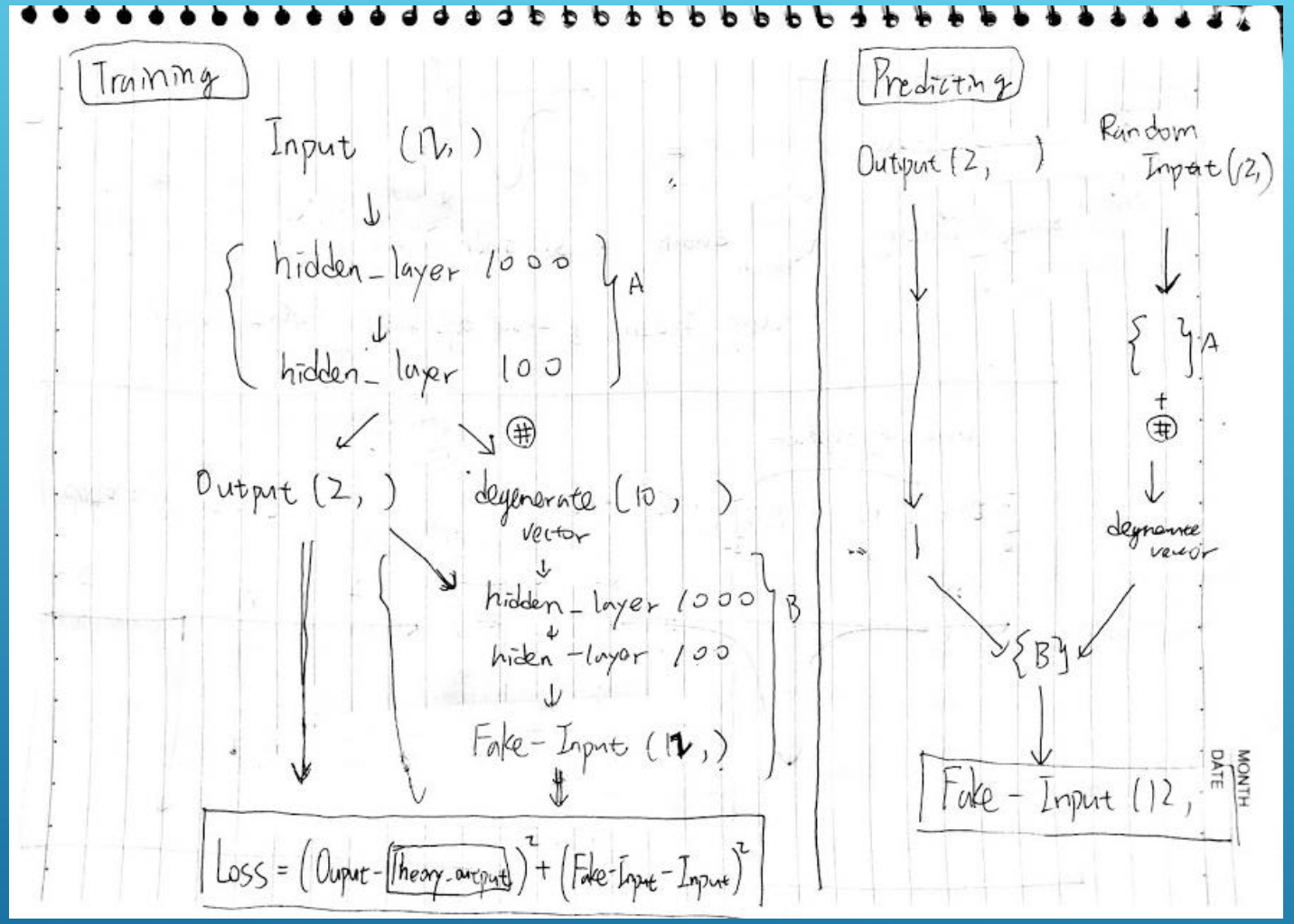

# 2 to 11 Fuzzy Training :Result

# (2) MODEL TRAINING




# 2 to 11 Fuzzy Training :Result

# (2) MODEL TRAINING



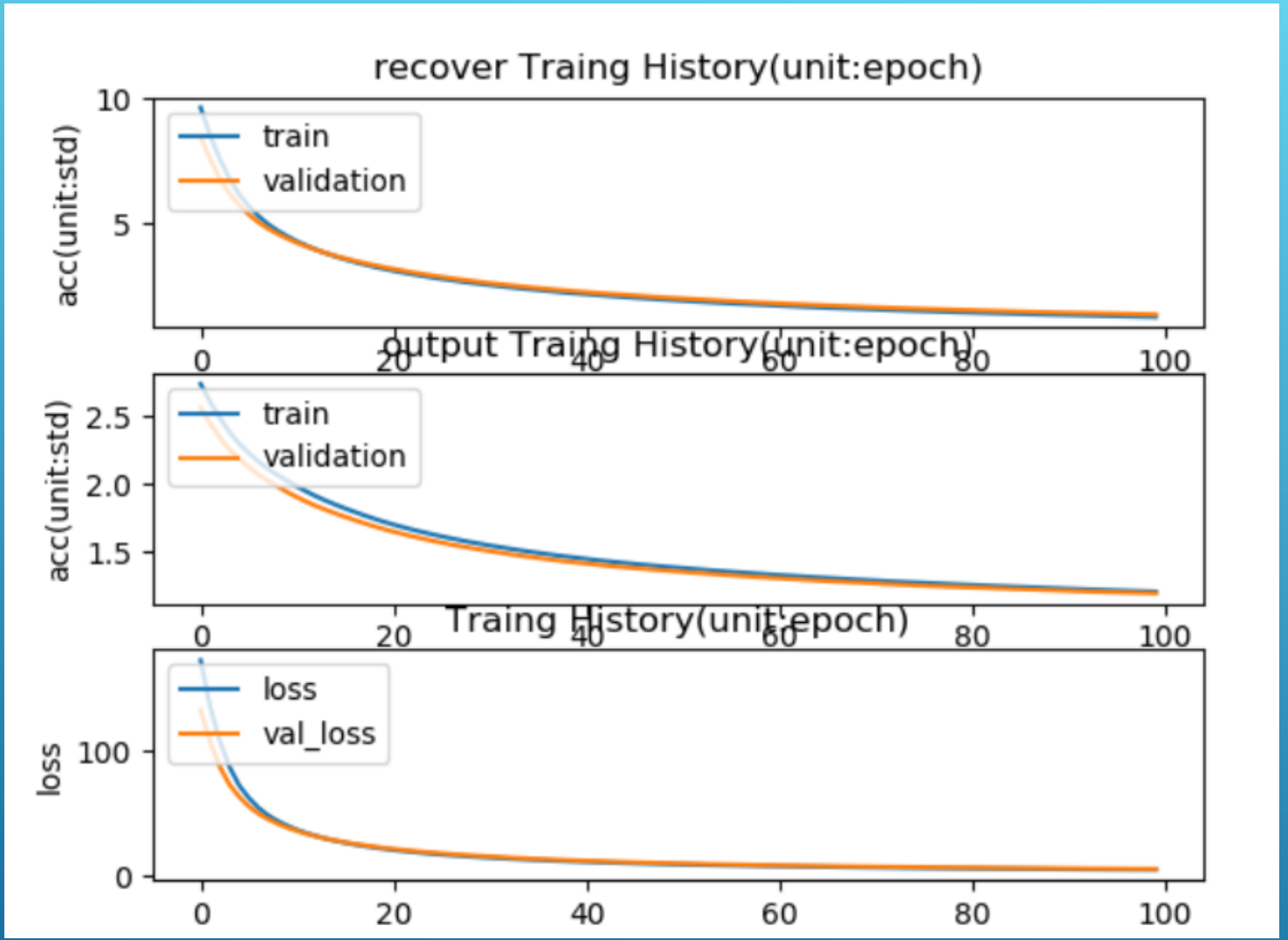

# 2 to12 Degenerate Vector Training

# (2) MODEL TRAINING

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 12)                0
_________________________________________________________________
forward_1 (Dense)            (None, 1000)              13000
_________________________________________________________________
forward_2 (Dense)            (None, 100)               100100
_________________________________________________________________
degenerate (Dense)           (None, 10)                1010
=================================================================
Total params: 114,110
Trainable params: 114,110
Non-trainable params: 0
_________________________________________________________________
```

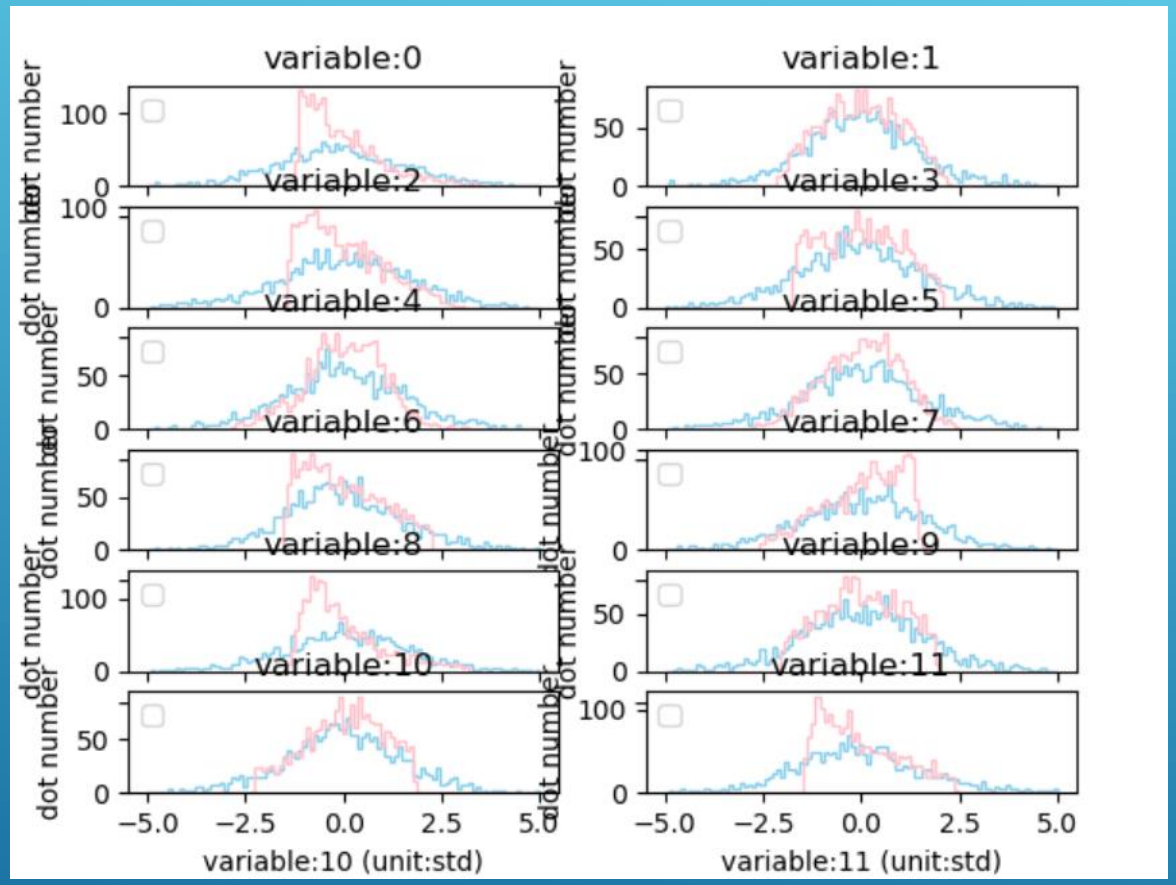
# 2 to12 Degenerate Vector Training

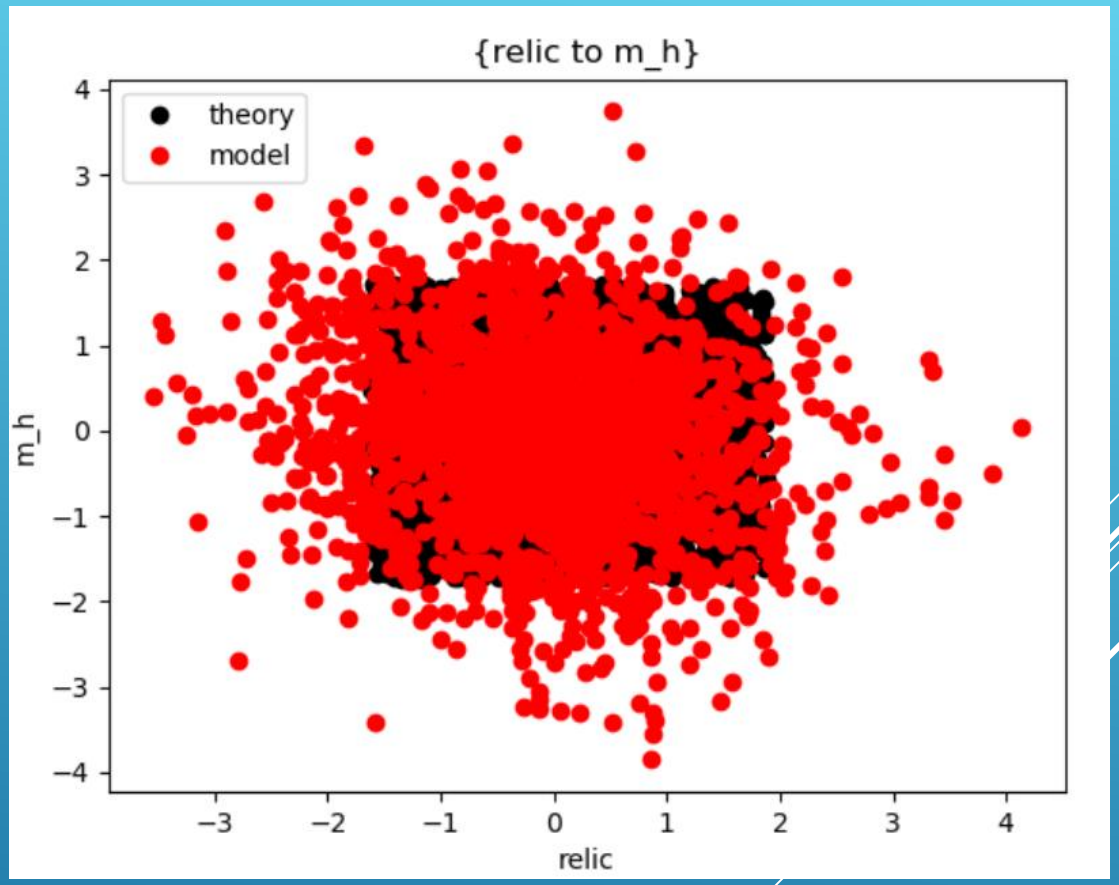
# (2) MODEL TRAINING




# 2 to12 Degenerate Vector Training

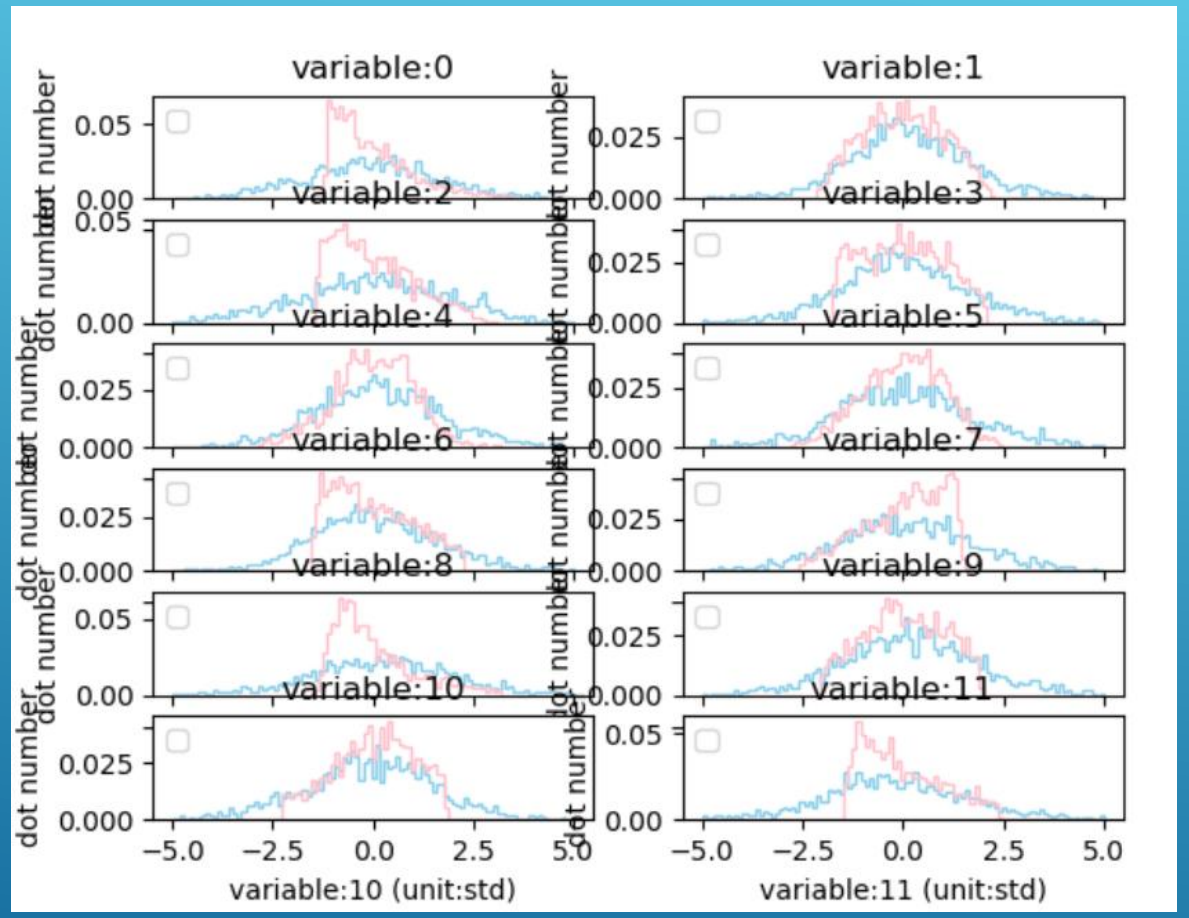
# (2) MODEL TRAINING

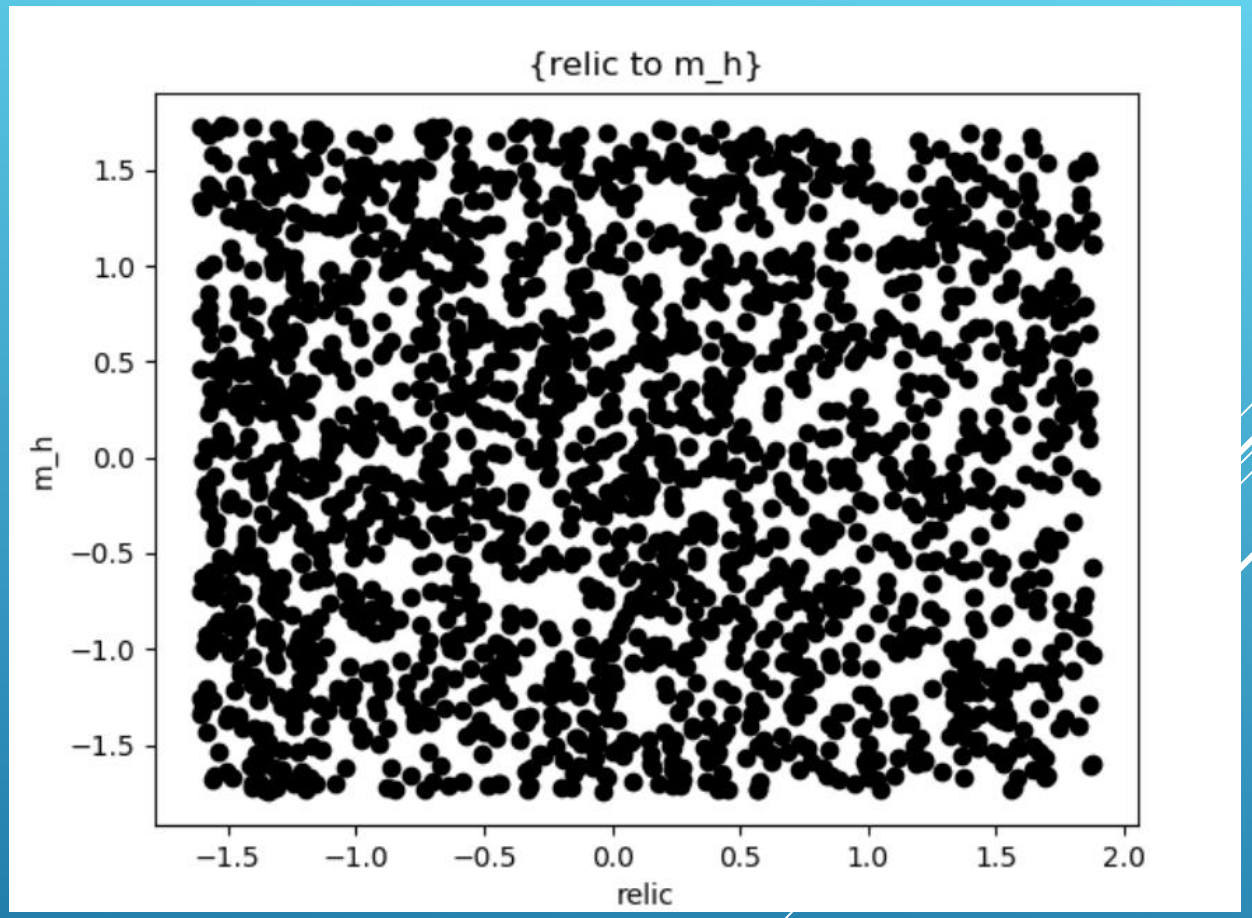



# 2 to12 Degenerate Vector Training

# (2) MODEL TRAINING

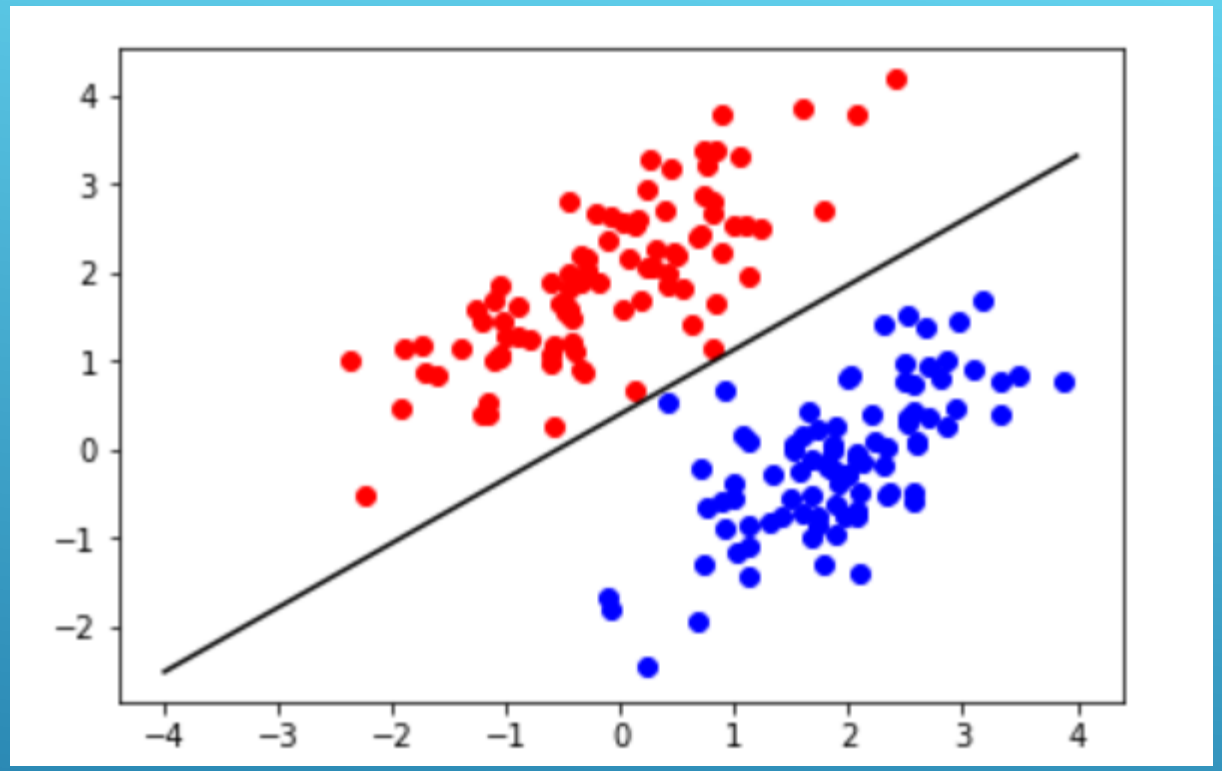


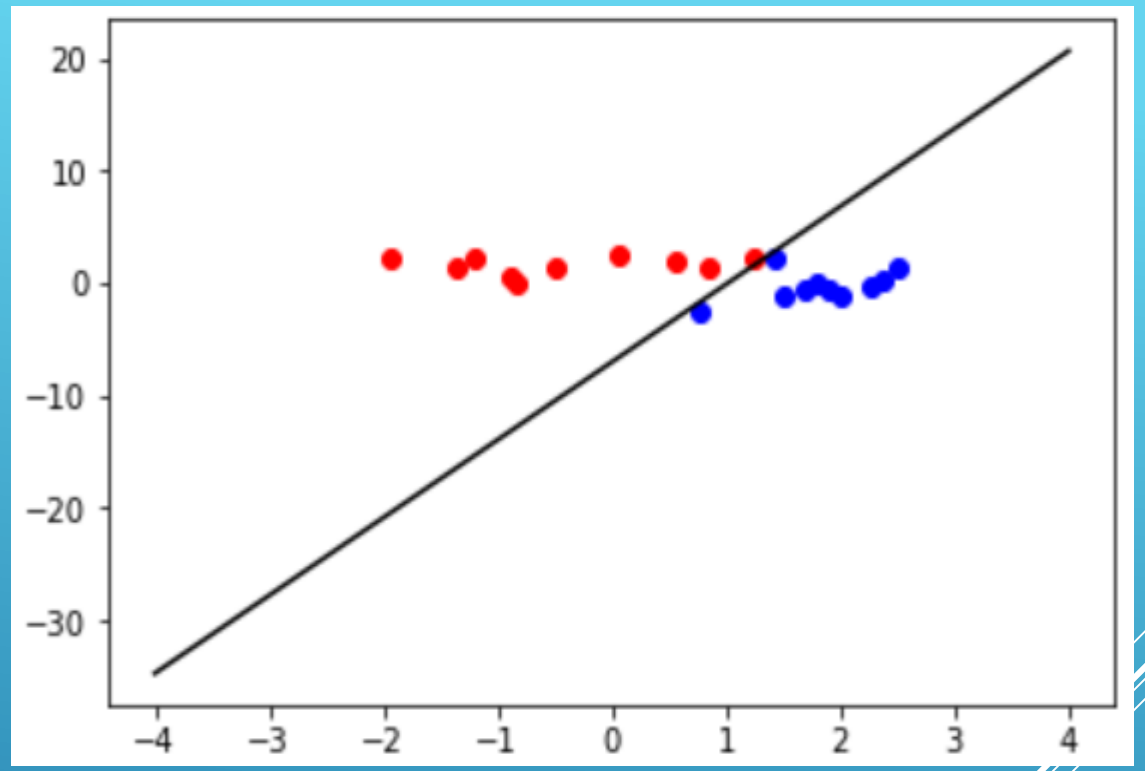




# 2 to12 Degenerate Vector Training

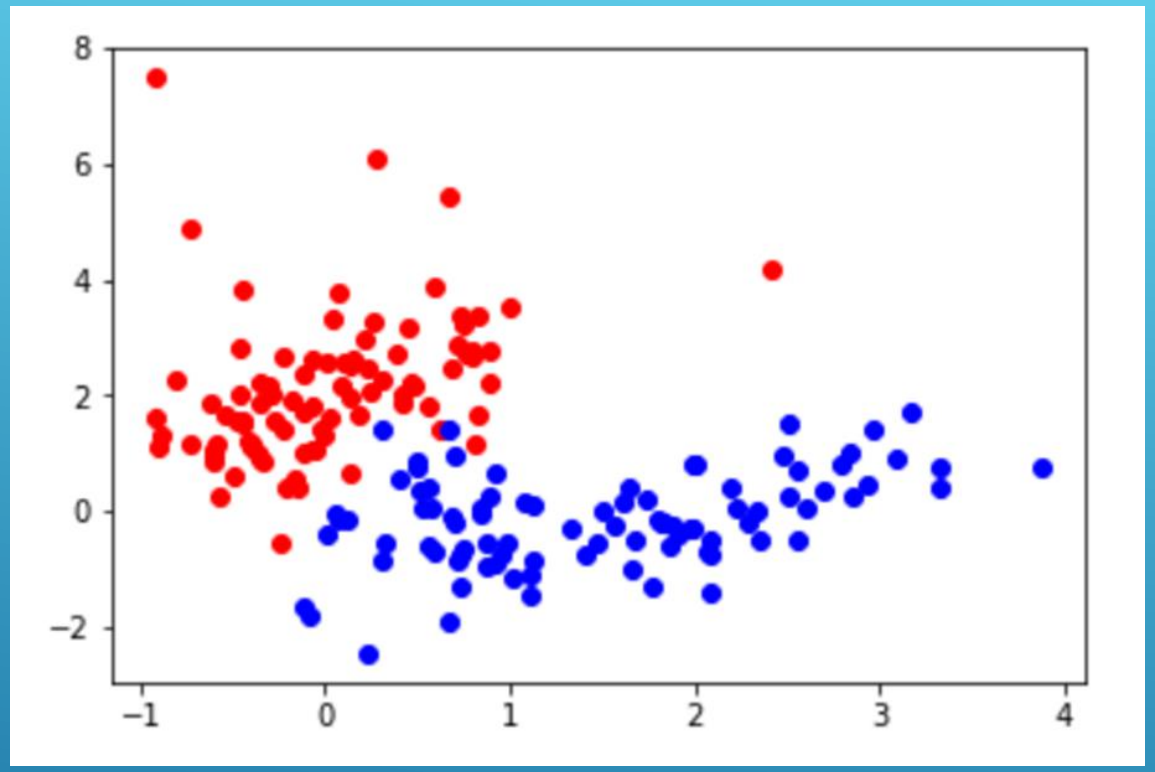
# APPENDIX : SUPPORT VECTOR MACHINE

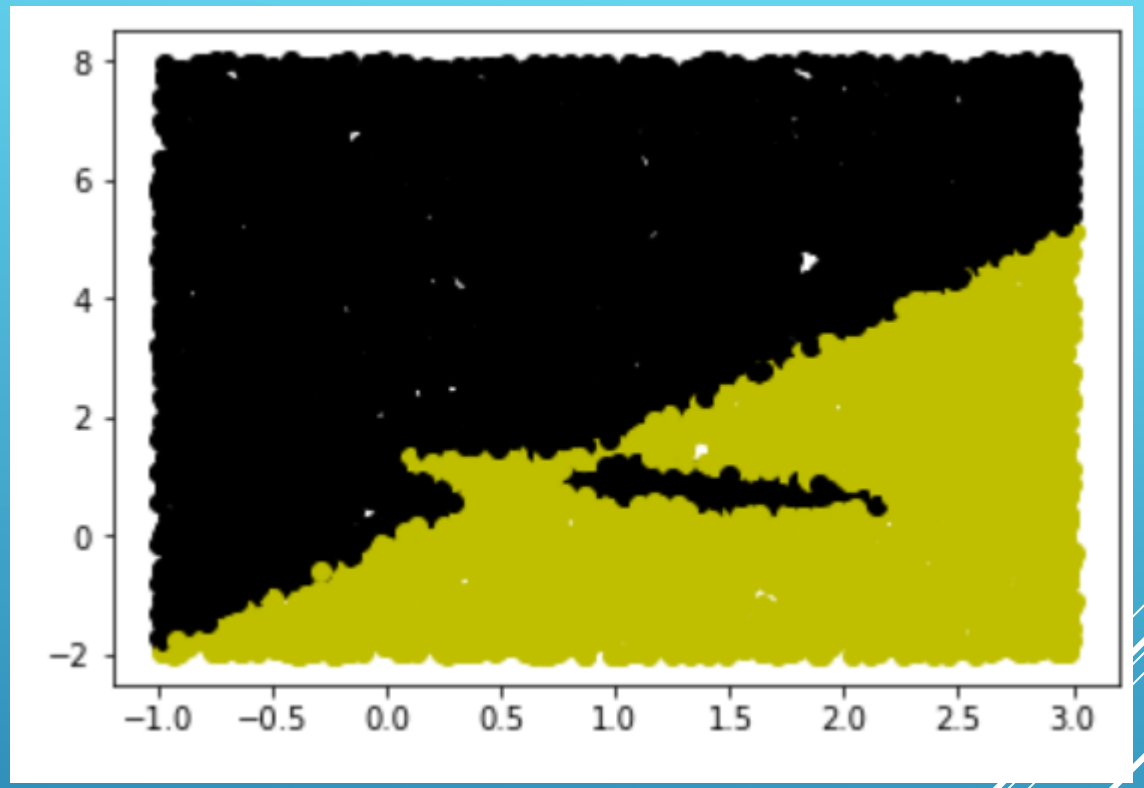
小小的工具研究




# RESULT BY FUKUSVM



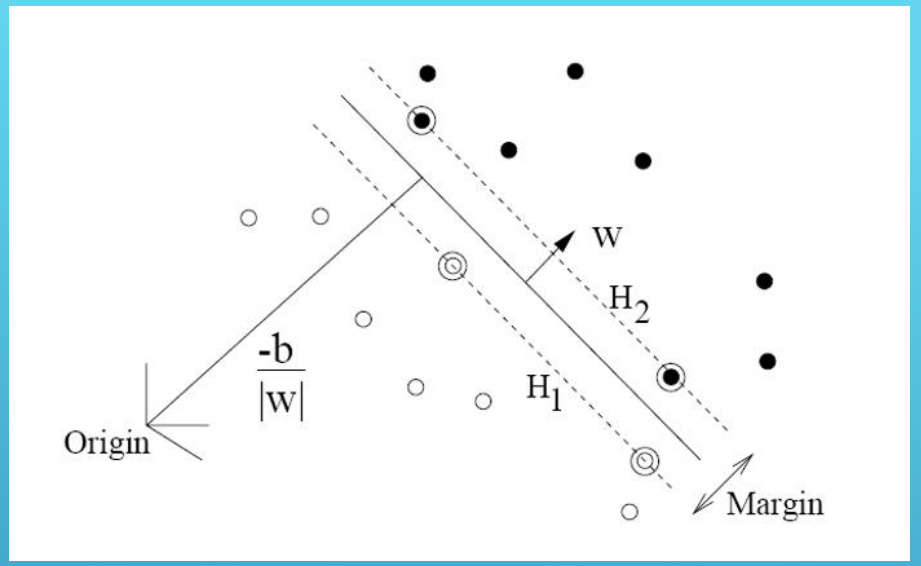




# RESULT BY FUKUSVM- POLYMONIAL KERNEL

$$\min_{b,\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

$$\text{s.t.} \quad y_n(\mathbf{w}^T\mathbf{z}_n + b) \geq 1,$$

$$\text{for } n = 1, 2, \ldots, N$$




$$\text{minimize} \quad \frac{1}{2}\|w\|^2 + C\sum_i \xi_i$$

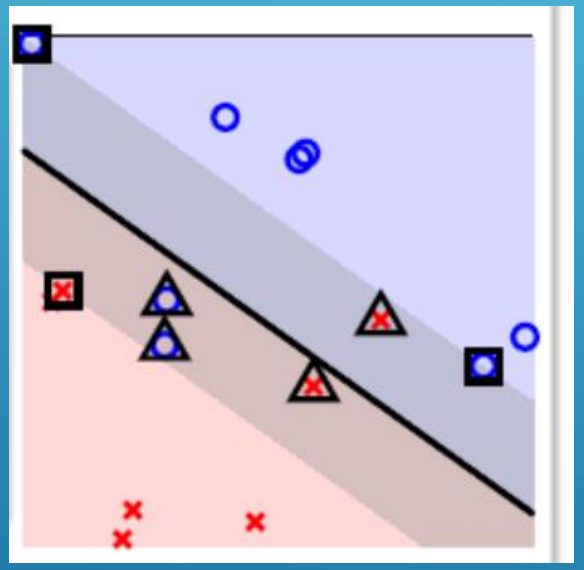
$$w^T x_i - b \leq -1 + \xi_i \quad \forall y_i = -1$$

$$w^T x_i - b \geq +1 - \xi_i \quad \forall y_i = +1$$

$$\xi_i \geq 0 \quad \forall i$$

$$\text{subject to} \quad y_i\left(w^T x_i - b\right) - 1 + \xi_i \geq 0 \quad \forall i$$

$$\xi_i \geq 0 \quad \forall i$$



# SUPPORT VECTOR MACHINE：理論

$$\text{maximize } f(c_1 \dots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i (\varphi(\vec{x}_i) \cdot \varphi(\vec{x}_j)) y_j c_j$$

$$= \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i k(\vec{x}_i, \vec{x}_j) y_j c_j$$

$$\text{subject to } \sum_{i=1}^{n} c_i y_i = 0, \text{ and } 0 \leq c_i \leq \frac{1}{2n\lambda} \text{ for all } i.$$

# SVM: KERNEL FUNCTION