4. Perceptron & ANN with Tensorflow

Table of Contents

Tensorflow

- Tensor
  - N-dimensional array.
    - Like *np.array*
- Tensorflow
  - Library for machine learning.
    - From the definion of flow of tensors (data), tensorflow can calcuate gradients of parameters automatically.
  - There are similar libraries such as Caffe, Torch, …

Tensorflow

- Tensor
  - Variable
    - Mutable tensor.
    - For example, weight.
  - Non-Variable
    - Immutable tensor.
    - For example, constant.
- Dataflow graph
  - Parallelism, Distributed execution, Portability, …

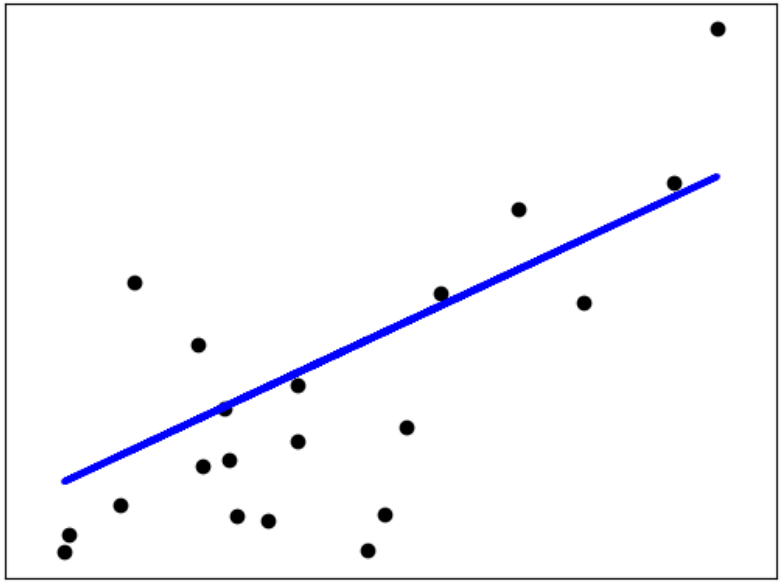
Tensorflow exercise

- Remind ‘Least Squares’.
- Let’s implement least squares with tensorflow.

Least Squares Estimation (Linear Regression)

```
### LR
import numpy as np
from sklearn import datasets, linear_model
import matplotlib.pyplot as plt

# Load the diabetes dataset
diabetes = datasets.load_diabetes()

# diabetes has two attributes: data, target
print(diabetes.data.shape)
print(diabetes.target.shape)

# diabetes consists of 442 samples
#with 10 attributes and 1 real target value.

# Use only one feature
diabetes_X = diabetes.data[:, 2:3]

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)
```

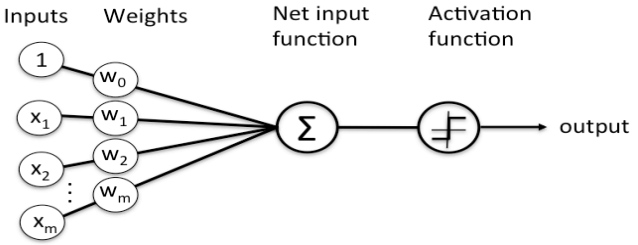
```
# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % np.mean((regr.predict(diabetes_X_test)
          - diabetes_y_test) ** 2))


# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, regr.predict(diabetes_X_test),
         color='blue', linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()
```



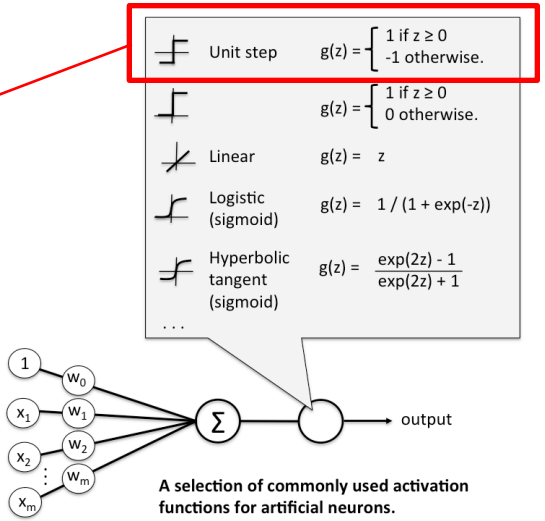
Perceptron(퍼셉트론)

- Simplest form of artificial neuron.
- Linear classification algorithm.
- Formulation.
  - Data: $D = \{(\mathrm{X}, t)^n\}_{n=1}^{N}$
  - Input features: $\mathrm{X} = (x_1, .., x_k)$
  - Output: $t \in \{-1, 1\}$
  - Model: $\hat{t} = g(f(\mathrm{X}))$

$$f(X) = \sum_{i=0}^{k} w_i x_i$$



Schematic of Rosenblatt's perceptron.



A selection of commonly used activation functions for artificial neurons.

Perceptron(퍼셉트론)

- Example) Make ‘AND’ function with perceptron.
  - Data: $D = \{(\mathrm{X}, t)^n\}_{n=1}^{N}$
  - Input features: $\mathrm{X} = (x_1, x_2)$
  - Output: $t \in \{0, 1\}$
  - Model: $\hat{t} = g\big(f(\mathrm{X})\big)$

$$f(X) = \sum_{i=0}^{k} w_i x_i = w_0 + w_1 x_1 + w_2 x_2$$

$$g(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

  - Find (w_0, w_1, w_2) that makes model as logical ‘AND’ function.
- There are multiple answers!
  - One possible answer is $(w_0, w_1, w_2) = (-0.8, 0.5, 0.5)$.

x_1	x_2	x_2 ***AND*** x_2
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Implement 'AND' function

```
################
### Perceptron for AND function
################
import tensorflow as tf
import math


### define graph
x = tf.placeholder(tf.float32, shape=(None, 2), name = 'x-input')
y = tf.placeholder(tf.float32, shape=(None, 1), name = 'y-input')

v_weight = tf.Variable(
  tf.random_uniform(shape=(2, 1), minval=-1, maxval=1),
  dtype=tf.float32,
  name = "W")
v_bias = tf.Variable(
  tf.zeros(shape=(1)),
  dtype=tf.float32,
  name = "w0")

y_h = tf.sigmoid( tf.matmul(x, v_weight) + v_bias )


### define loss function
# # prevent nan loss
# epsilon = 1e-10
# loss = tf.reduce_mean(
#   -1 * y * tf.log(y_h + epsilon) -
#   (1 - y) * tf.log(1 - y_h + epsilon) )

loss = tf.reduce_mean(
  -1 * y * tf.log(y_h) -
  (1 - y) * tf.log(1 - y_h) )

### define optimization function
learning_rate = 0.1
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)
```

Implement 'AND' function

```
### define input data
DATA = {
  'X': [[0,0],[0,1],[1,0],[1,1]],
  'Y': [[0],[1],[1],[1]]
}


### Starting sessions
with tf.Session() as sess:
  ## initialize variables
  init = tf.global_variables_initializer()
  sess.run(init)

  max_iter = 100000

  for i in range(max_iter):
    _, v_w_val, v_b_val, y_h_val, loss_val = sess.run(
      [train_step, v_weight, v_bias, y_h, loss],
      feed_dict={x: DATA['X'], y: DATA['Y']})

    if i % 1000 == 0:
      print('Epoch ', i)
      print('Y_prediction ', y_h_val)
      print('True', DATA['Y'])
      print('Loss', loss_val)
      print('Weight', v_w_val)
      print('Bias', v_b_val)

    if math.isnan(loss_val):
      print('LOSS is NAN!')
      break
```

SNU BIOINFORMATICS INSTITUTE

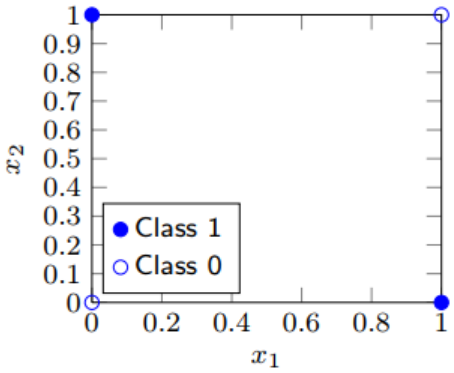
Perceptron Exercise.

1. Change previous code to learn ’OR’ function.
2. Change previous code to learn ’XOR’ function?

Perceptron(퍼셉트론)

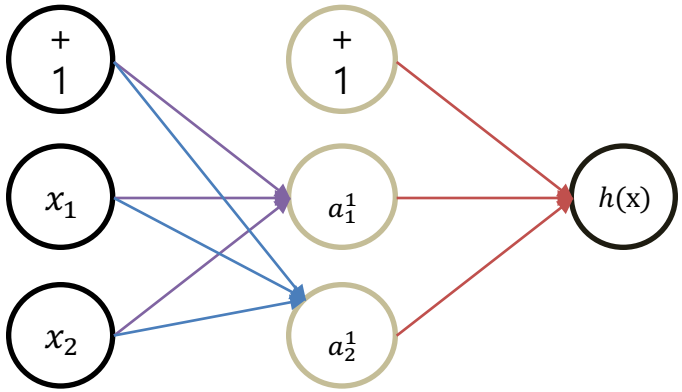
- Then, can perceptron model ‘XOR’ function?
  - No, it cannot.
  - To do so, we can separate two classes by drawing a single line on the plot. However, it is not possible.
- Perceptron cannot model non linearly separable data!
- Multilayer perceptron can represent more complex models.

x_1	x_2	x_2 **XOR** x_2
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**



Multilayer perceptron(MLP)

- Modeling ‘XOR’ function with MLP?
  - $XOR = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$
    - $\vee$: ‘AND’, $\wedge$: ‘OR’, $\neg$: ‘NOT’
  - With two layers

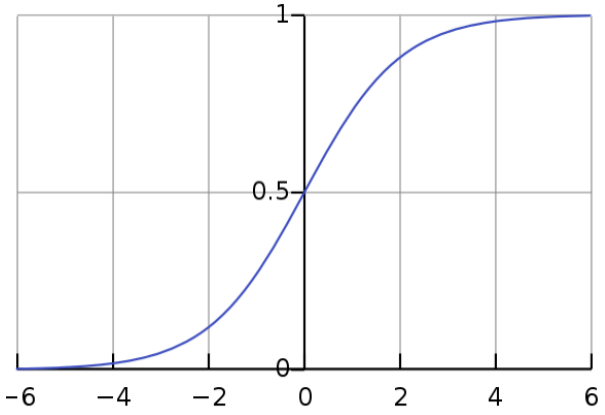


x_1	x_2	x_2 **XOR** x_2
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

Perceptron Exercise.

1. Change previous code to learn ’XOR’ function.
    1. By adding additional layer.

Multiclass-classification



- Logistic loss -> softmax loss
  - Sigmoid (logistic): $\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$
  - Softmax: $\sigma(X)_j = \frac{e^{X_j}}{\sum_i e^{X_i}}$
- Cross entropy

$$H(p,q) = -\sum p_i \log q_i = -y \log \hat{y} - (1-y)\log(1-\hat{y})$$

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log(p_{i,j})$$

Neural network for real data

- Iris data from sklearn.dataset
  - Multiclass-classification problem.
  - 3 classes.
  - 4 features.
  - 150 samples.

Neural network Exercise.

1. Build new model.
    1. Consists of four hidden layers.
    2. Each layer consists of (10, 20, 10, 20) units with biases.
2. Does accuracy increase or decrease?
    1. Explain why.