<u>MAKING MODEL FOR BINARY CLASSIFICATION:</u>

<u>Classification- Breast Cancer or Not</u>

INTRODUCTION:

1) ANN: Artificial neural network consists of
   - Input layer
   - Hidden layer
   - Output layer
2) SLP: Single layer perception
   - *If ANN model has no hidden layer its called SLP*
3) Basic equations:

   Output = Weight * Input + Bias

   We already have output and input layers. We don't have weight (a value that can give different weights depending on features and output => [len(features), len(output)])

   And bias (a value that can give different weights depending on features => [len(output)])

IMPORTING LIBRARIES:

```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, Normalizer
from sklearn.decomposition import PCA as sklearnPCA

# Supress unnecessary warnings so that presentation looks clean
import warnings
warnings.filterwarnings("ignore")
```

EXPLORING DATASET:

1) Import data set:

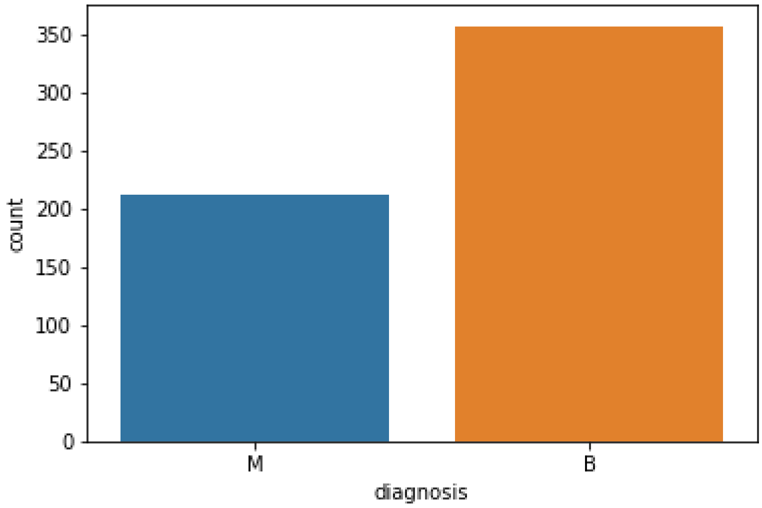
```
wbcd = pd.read_csv("../input/data.csv")
wbcd.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_r
0	842302	M	17.99	10.38	122.80	1001.0	0.11840
1	842517	M	20.57	17.77	132.90	1326.0	0.08474
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960
3	84348301	M	11.42	20.38	77.58	386.1	0.14250
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030

2) Summary the diagnosis:

INP: `sns.countplot(wbcd['diagnosis'],label="Count")`

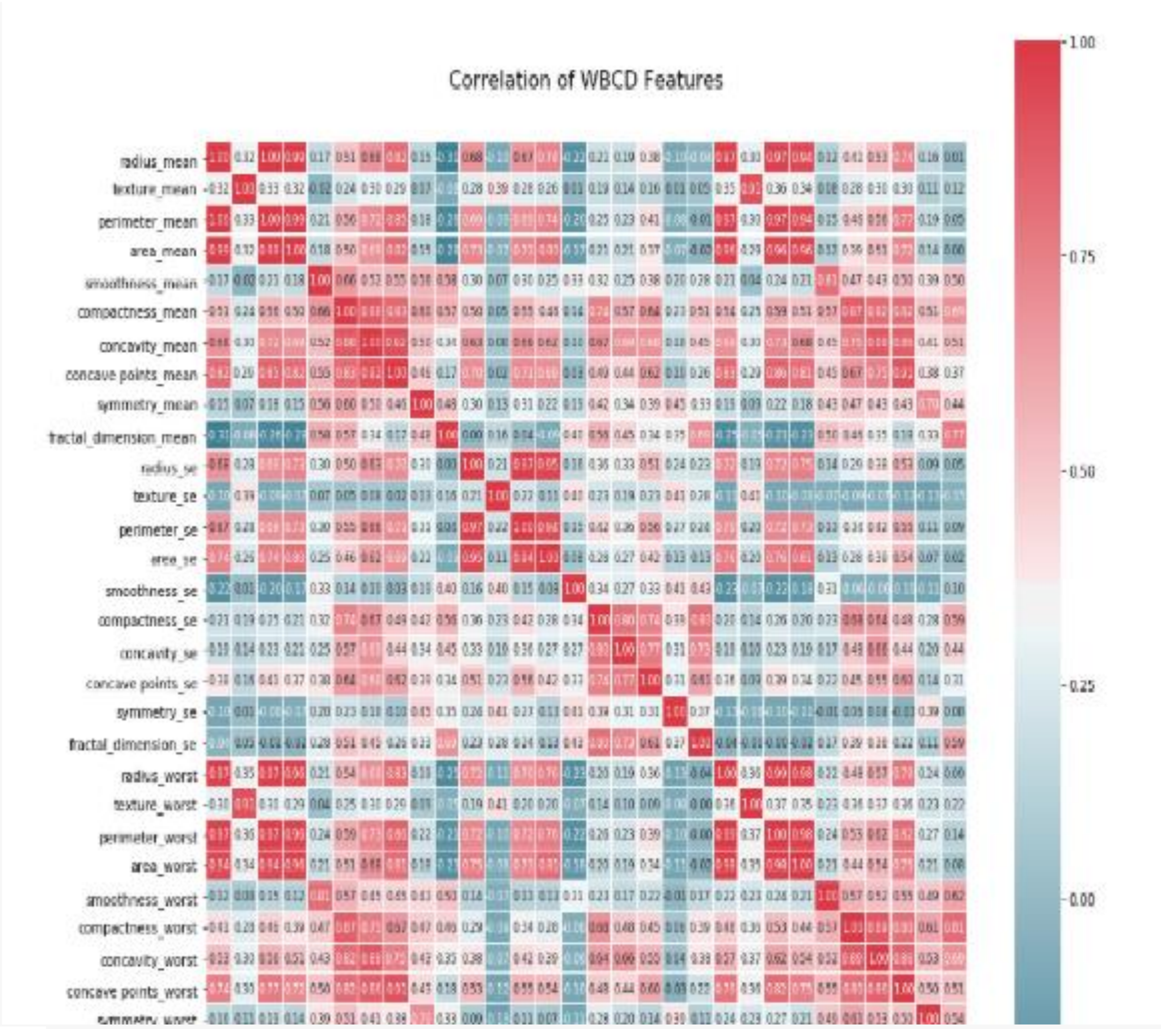
Output: `<matplotlib.axes._subplots.AxesSubplot at 0x7fece72a3470>`



3) Correlation Plot of 30 features

INPUT:

```
corr = wbcd.iloc[:,2:].corr()
colormap = sns.diverging_palette(220, 10, as_cmap = True)
plt.figure(figsize=(14,14))
sns.heatmap(corr, cbar = True,  square = True, annot=True, fmt= '.2f',annot_kws={'size': 8},
         cmap = colormap, linewidths=0.1, linecolor='white')
plt.title('Correlation of WBCD Features', y=1.05, size=15)
```



4) Preparing the data for machine learning:

i) Divide "WBCD data" into Train(70%) / Test data(30%)

```
- train,test = train_test_split(wbcd, test_size=0.3, random_state=42)
- print("Training Data :",train.shape)
- print("Testing Data :",test.shape)
```

II) Drop ID column:

```
-train_id = train['id']
test_id = test['id']

train_data = train.iloc[:,1:]
test_data = test.iloc[:,1:]

print("Training Data :",train_data.shape)
print("Testing Data :",test_data.shape)
```

III) Seperate x:Feature data(30) / y:Result data(1)

-Normalize x_data values for better prediction

```
# Training Data
train_x = train_data.iloc[:,1:]
train_x = MinMaxScaler().fit_transform(train_x)
print("Training Data :", train_x.shape)

# Testing Data
test_x = test_data.iloc[:,1:]
test_x = MinMaxScaler().fit_transform(test_x)
print("Testing Data :", test_x.shape)
```

Change Results(diagnosis) format : String -> Numeric

In [10]:

```
# Training Data
train_y = train_data.iloc[:,:1]
train_y[train_y=='M'] = 0
train_y[train_y=='B'] = 1
print("Training Data :", train_y.shape)

# Testing Data
test_y = test_data.iloc[:,:1]
test_y[test_y=='M'] = 0
test_y[test_y=='B'] = 1
print("Testing Data :", test_y.shape)
```

5) Making ANN-SLP Model

1) Make "Placeholder" for dynamic variable allocation

Placeholder is one of the function in tensorflow. It is a space to put and change values while the program is running.

- for X, a place must have 30 columns, since wbcd data has 30 features.
- for Y, a place must have 1 columns, since the results has 1 outcome.

- If you see the row "None", it means it has no size limits. (You can write -1 instead of "None")

In [11]:

```
X = tf.placeholder(tf.float32, [None,30])
Y = tf.placeholder(tf.float32, [None, 1])
```

6-2) Make Weight, Bias value with randomly

- W(weight) : why **[30,1]**? 16 for 16 features, 1 for 1 Outcome(results).
- P(weight): why **[10,1]**? 10 for 10 PCA features, 1 for 1 Outcome(results).
- b(bias) : why **[1]**? outcome has 1 layers.

In [12]:

```
# weight
W = tf.Variable(tf.random_normal([30,1], seed=0), name='weight')

# bias
b = tf.Variable(tf.random_normal([1], seed=0), name='bias')
```

6-3) Make Output Results

- **Output = Weight * Input + Bias**
- tf.matmul() : for array multiply

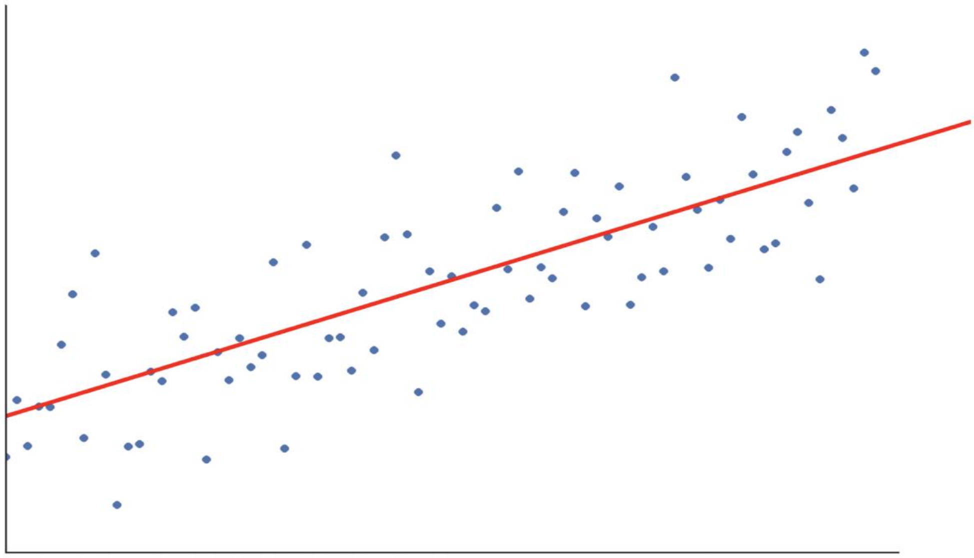
In [13]:

```
logits = tf.matmul(X,W) + b
```

6-4) Cross Entropy

Before this, you have to know **How Linear Regression Works**

- Linear Regression: Draw a random line to find the **mean square root error** and find the slope and intercept to minimize this value (reduce the error to the minimum)
- Since Logits is also linear equation, you have to find minimum cost!



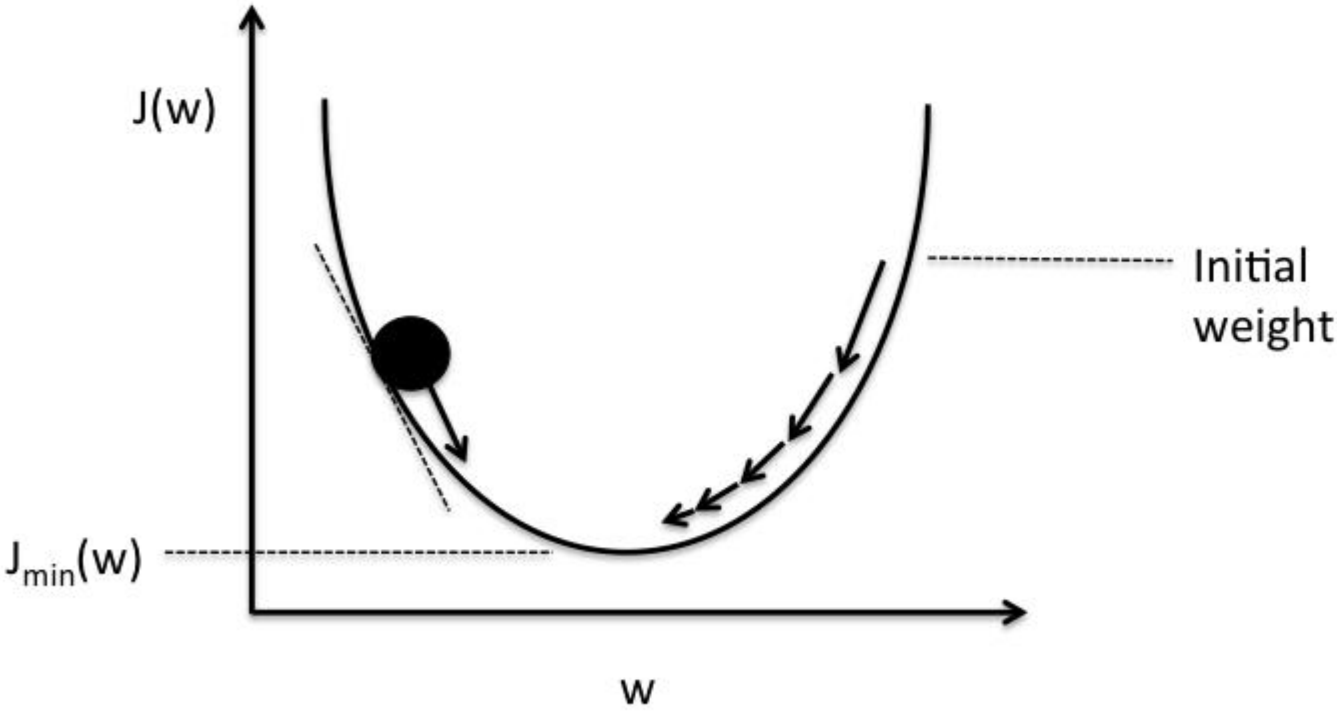
For example, logits(we get above) is **red line**, and the real dataset is **blue dot**.

1. For finding cost, you have to substract all blue dot value with red line.
2. Next, You add all distance you find and get average.
3. For good prediction, this average distance of red line & blue dot must be minimum value.
4. tf.nn.sigmoid_cross_entropy_with_logits(): for gradient_descent with sig results(hypothesis).

In [14]:

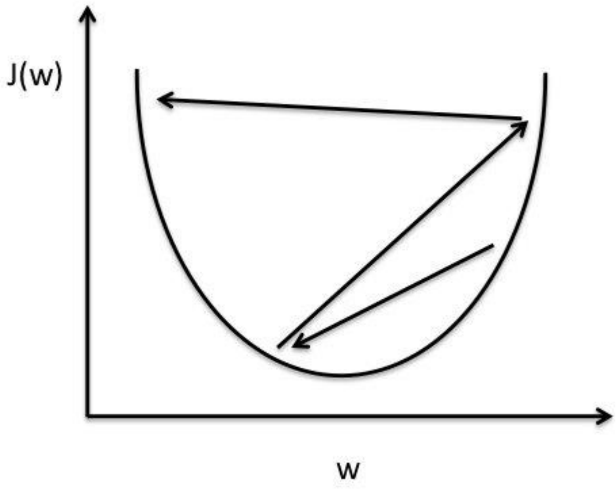
```
hypothesis = tf.nn.sigmoid(logits)

cost_i = tf.nn.sigmoid_cross_entropy_with_logits(logits=logits,labels=Y)
cost = tf.reduce_mean(cost_i)
# cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
```

6-5) Gradient Descent Optimizer

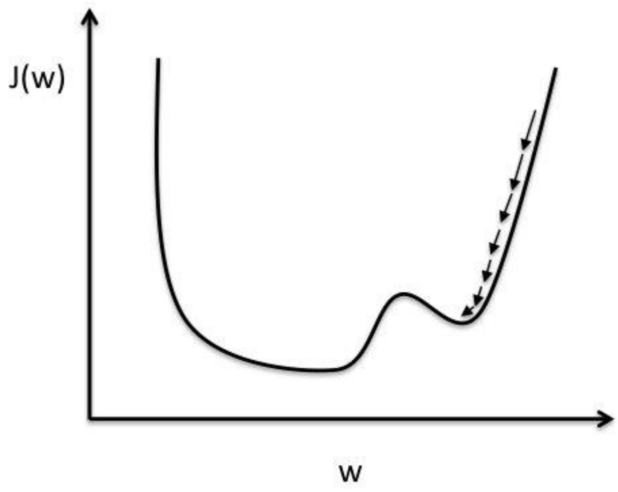


Schematic of gradient descent.

- GradientDescentOptimizer: It makes the best result with the least error
- There are lots of optimizer methods provided in tensorflow. (GradientDescent, Adam, RMSProp, etc.)
- learning rate : It indicates the degree of descending size.



Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.

In [15]:

```
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

6-6) Compare : original vs. prediction

In [16]:

```
prediction = tf.cast(hypothesis > 0.5, dtype=tf.float32)
correct_prediction = tf.equal(prediction, Y)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, dtype=tf.float32))
```

6-7) Activate Model

In [17]:

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for step in range(10001):
        sess.run(train, feed_dict={X: train_x, Y: train_y})
        if step % 1000 == 0:
            loss, acc = sess.run([cost, accuracy], feed_dict={X: train_x, Y: train_y})
            print("Step: {:5}\tLoss: {:.3f}\tAcc: {:.2%}".format(step, loss, acc))

    train_acc = sess.run(accuracy, feed_dict={X: train_x, Y: train_y})
    test_acc,test_predict,test_correct = sess.run([accuracy,prediction,correct_pr
ediction], feed_dict={X: test_x, Y: test_y})
    print("Model Prediction =", train_acc)
    print("Test Prediction =", test_acc)
```

5) ANN-SLP MODEL:

- train_x, test_x : normalization data
- 30 features
- train_y, test_y

In [18]:

```
def ann_slp():
    print("==========Data Summary==========")
    print("Training Data :", train_x.shape)
    print("Testing Data :", test_x.shape)

    X = tf.placeholder(tf.float32, [None,30])
    Y = tf.placeholder(tf.float32, [None, 1])

    W = tf.Variable(tf.random_normal([30,1], seed=0), name='weight')
    b = tf.Variable(tf.random_normal([1], seed=0), name='bias')

    logits = tf.matmul(X,W) + b
    hypothesis = tf.nn.sigmoid(logits)

    cost_i = tf.nn.sigmoid_cross_entropy_with_logits(logits=logits,labels=Y)
    cost = tf.reduce_mean(cost_i)

    train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

    prediction = tf.cast(hypothesis > 0.5, dtype=tf.float32)
    correct_prediction = tf.equal(prediction, Y)
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, dtype=tf.float32))

    print("\n===========Processing===========")
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        for step in range(10001):
            sess.run(train, feed_dict={X: train_x, Y: train_y})
            if step % 1000 == 0:
                loss, acc = sess.run([cost, accuracy], feed_dict={X: train_x, Y: train_y})
                print("Step: {:5}\tLoss: {:.3f}\tAcc: {:.2%}".format(step, loss, acc))

        train_acc = sess.run(accuracy, feed_dict={X: train_x, Y: train_y})
```

```
    test_acc,test_predict,test_correct = sess.run([accuracy,prediction,correct_
prediction], feed_dict={X: test_x, Y: test_y})

    print("\n============Results============")
    print("Model Prediction =", train_acc)
    print("Test Prediction =", test_acc)

    return train_acc,test_acc

ann_slp_train_acc, ann_slp_test_acc = ann_slp()
```