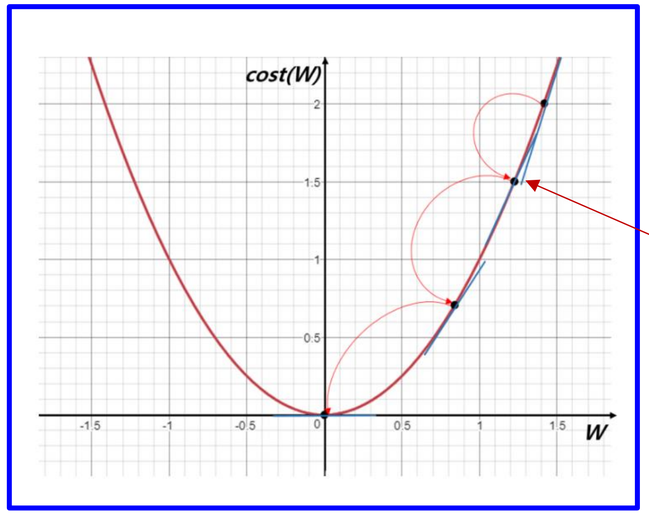
Lecture 07

# Application & Tips: Learning rate, data preprocessing, overfitting

# Gradient descent

*Cost (Loss)*




*STEP (learning rate)*

*Loss (Cost)*

$$L = \frac{1}{N}\sum_{i} D(S(WX_i + b), L_i)$$

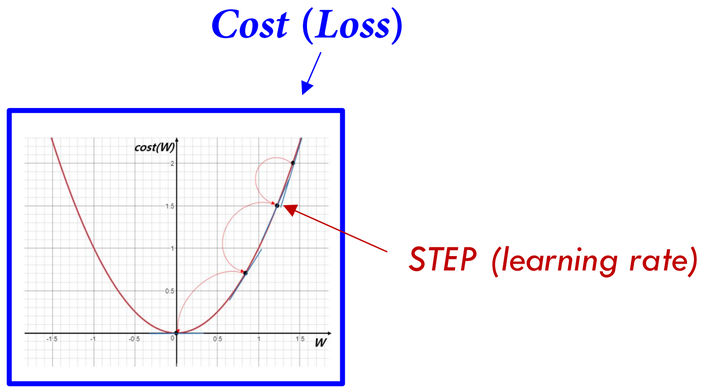
*Training set*

# Gradient descent

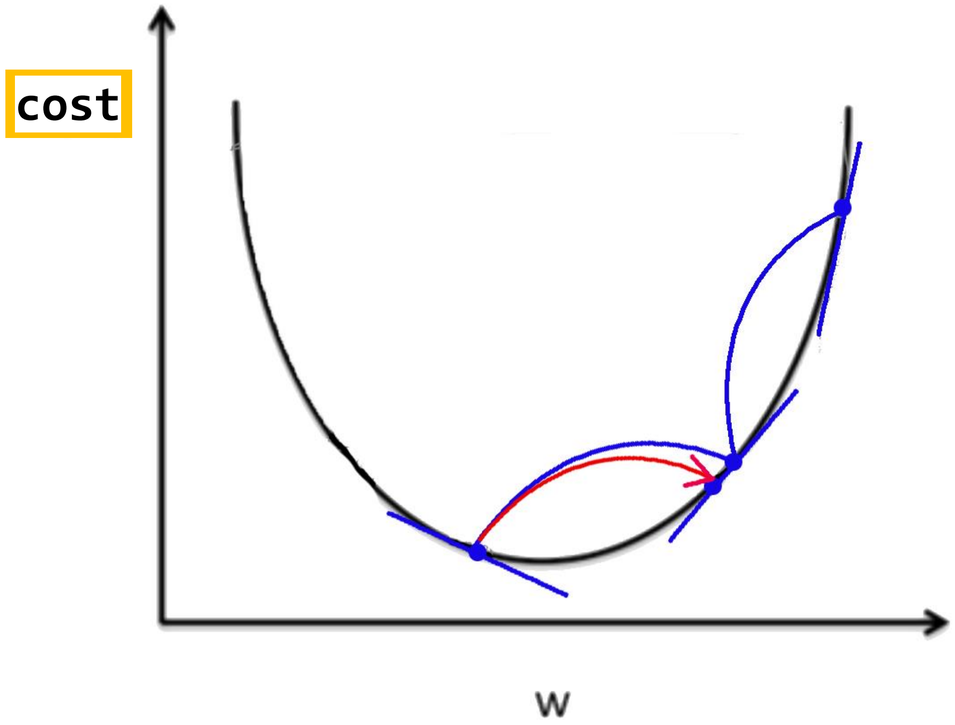
```
# Mininmize error using cross entropy
# reduce_sum( , axis=1 ) axis=1 --> row에서 합계를 계산, axis=0 --> column에서 합계를 계산

Learning_rate = 0.001
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

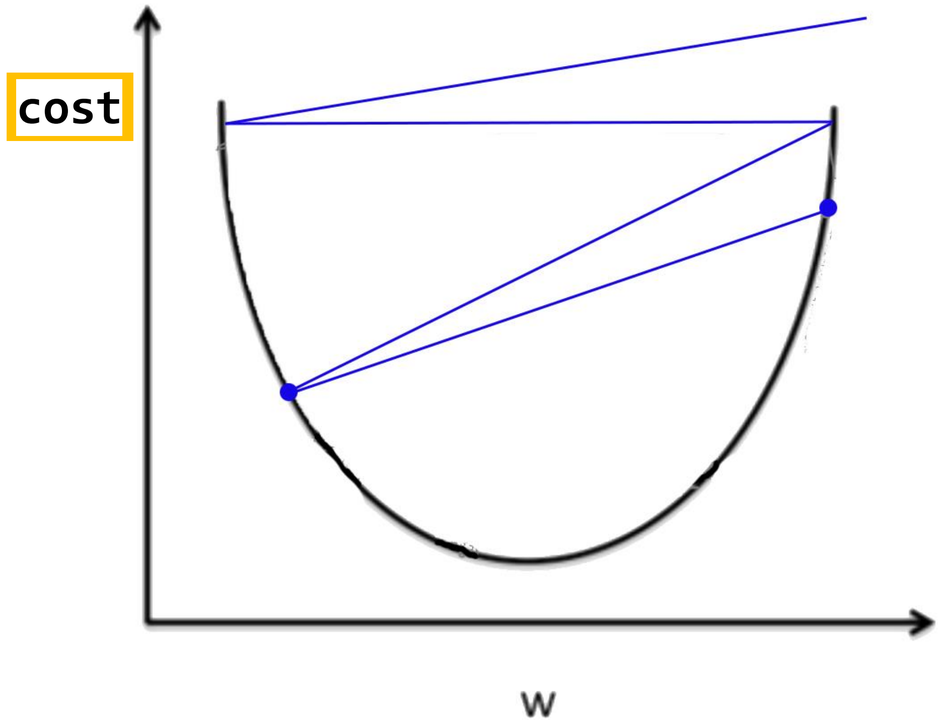
=

```
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001).minimize(cost)
```

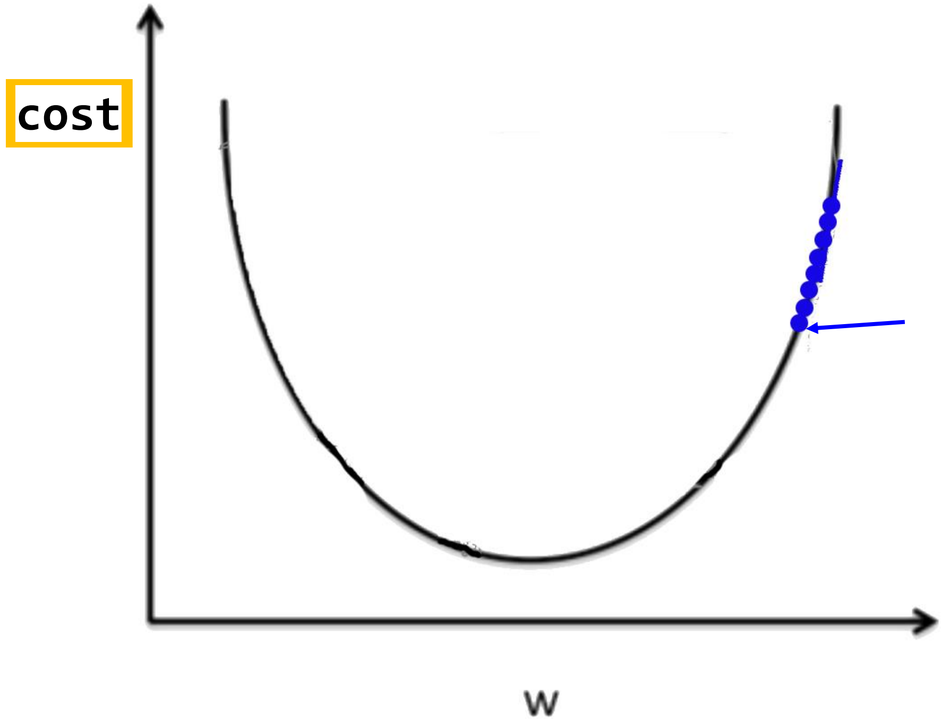


# Large learning rate: overshooting



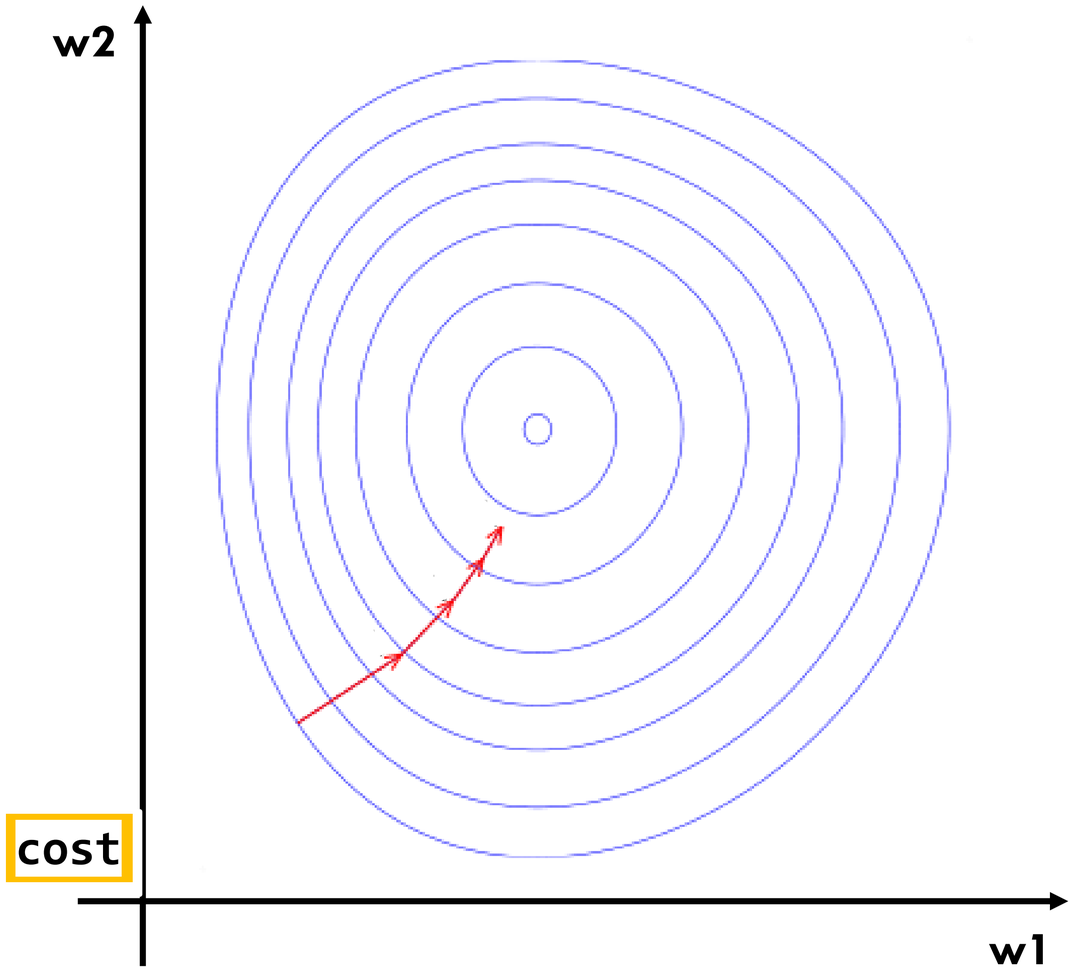
# Small learning rate: takes too long, stops at local minimum

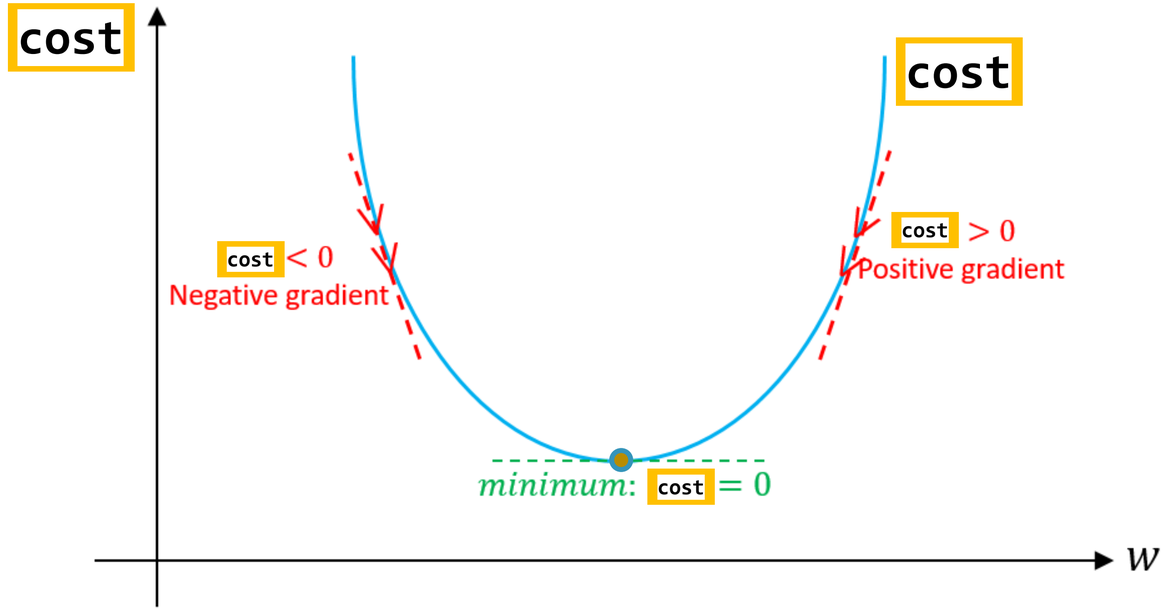


# Try several learning rates

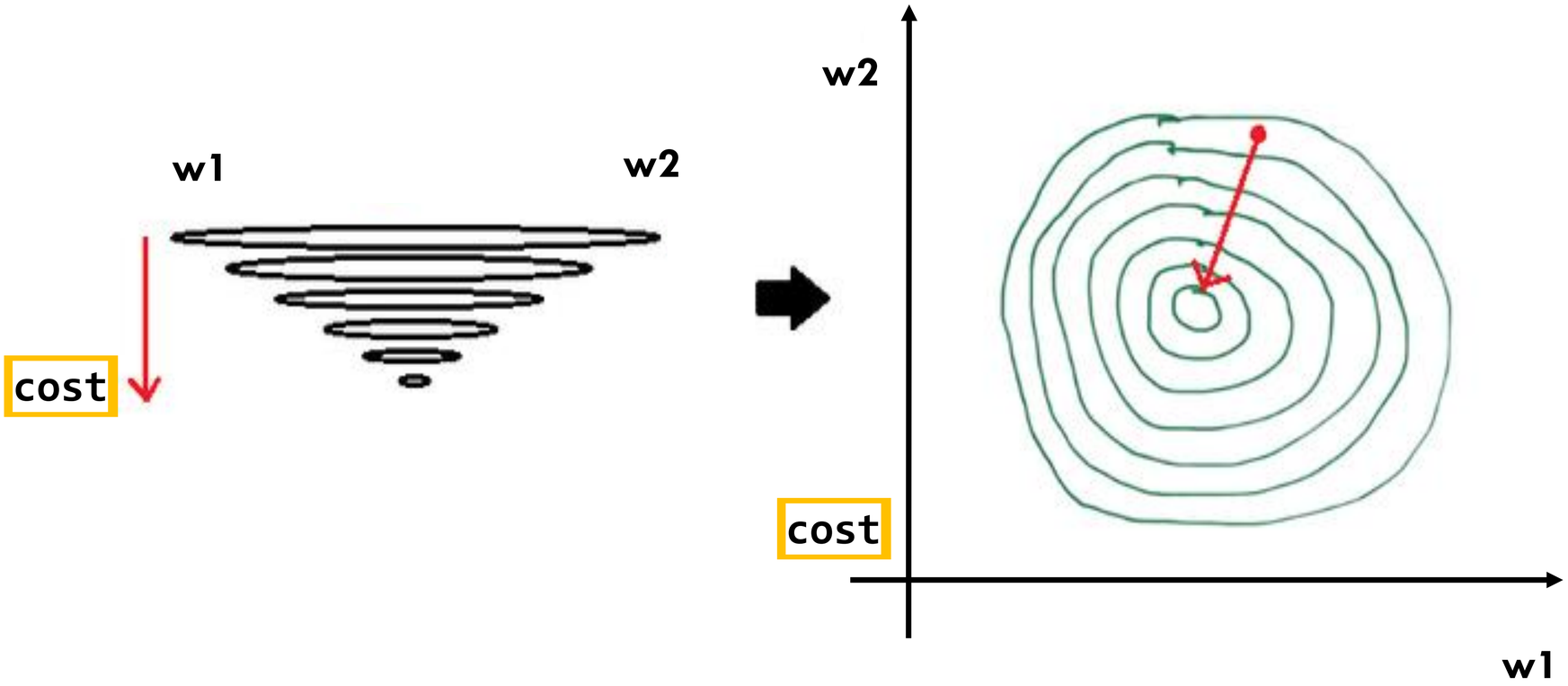
- **Observe the cost function**
- **Check it goes down in a reasonable rate**

# Data (X) preprocessing for gradient descent



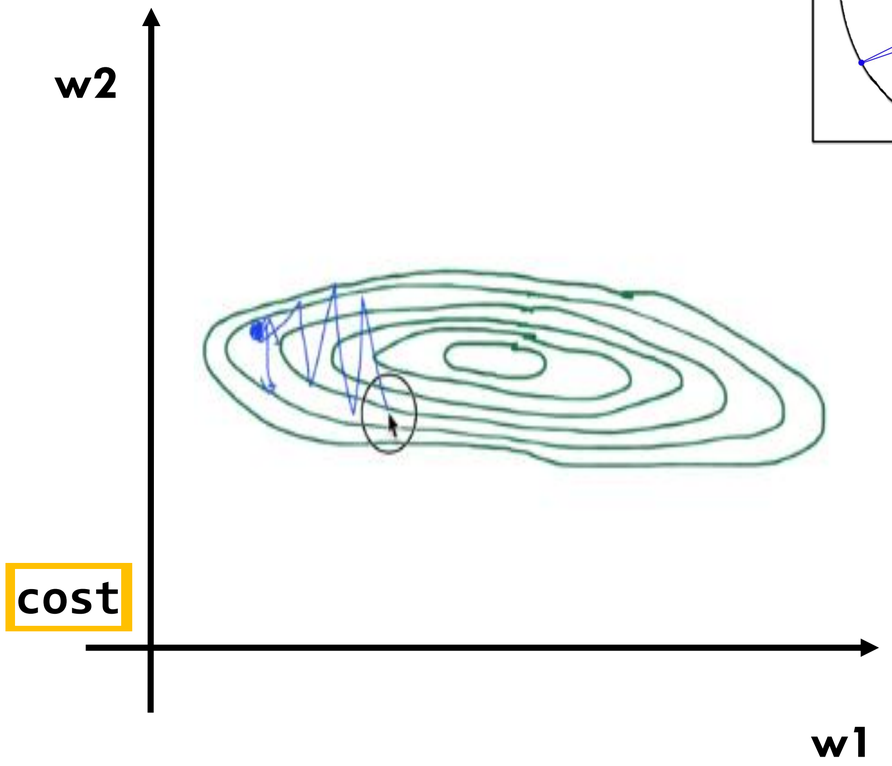



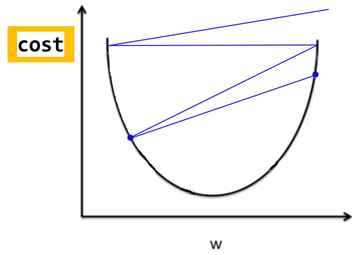
# Data (X) preprocessing for gradient descent

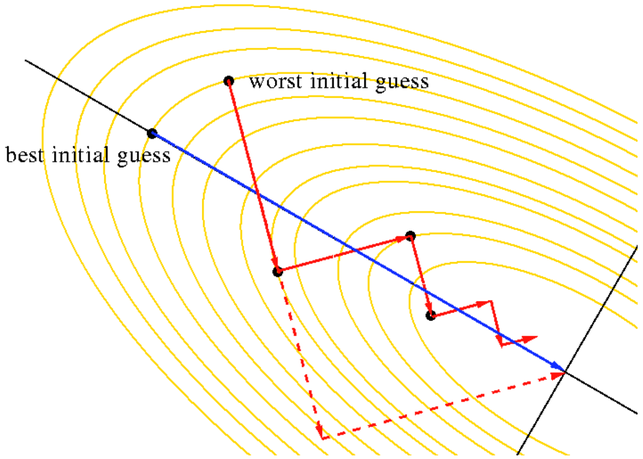


# Data (X) preprocessing for gradient descent

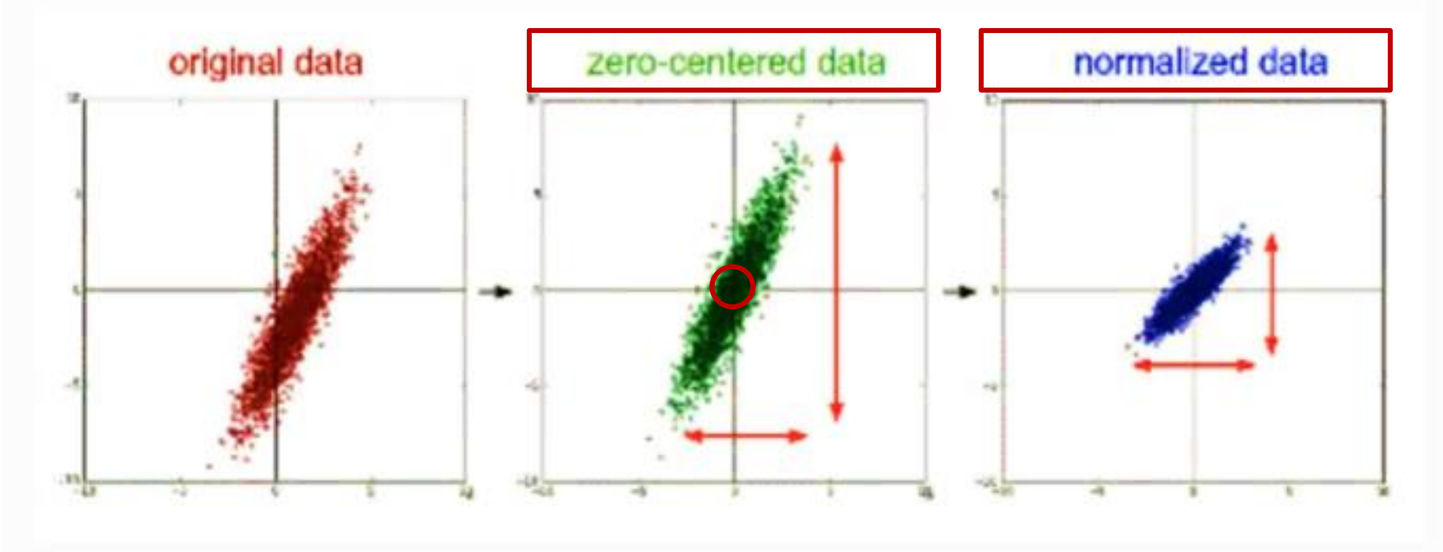
| x1 | x2 | y |
|---|---|---|
| 1 | 9000 | A |
| 2 | -5000 | A |
| 4 | -2000 | B |
| 6 | 8000 | B |
| 9 | 9000 | C |



# Data (X) preprocessing for gradient descent



# Standardization

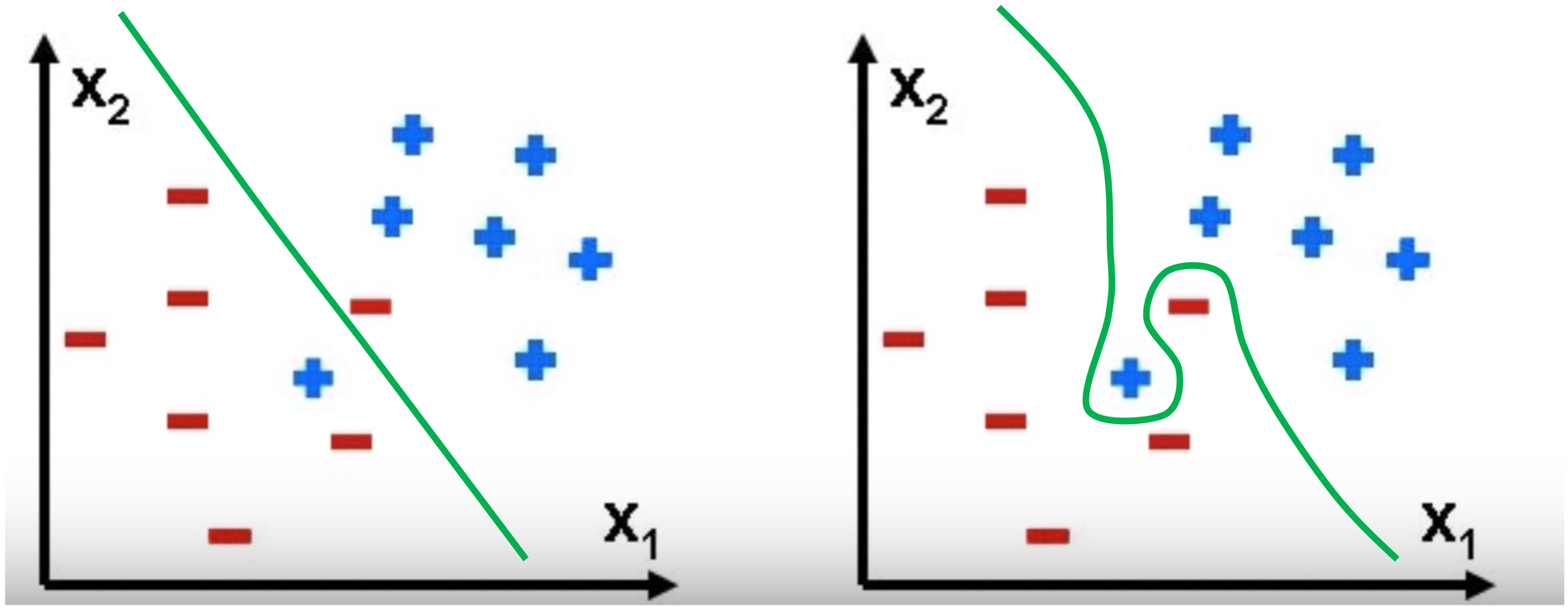
$$x'_j = \frac{x_j - \mu_j}{\sigma_j}$$

```
X_std[:,0] = (X[:,0] - X[:,0].mean()) / X[:,0].std()
```

# Overfitting

- **Our model is very good with training data set (with memorization)**
- **Not good at test dataset or in real use**

# Overfitting

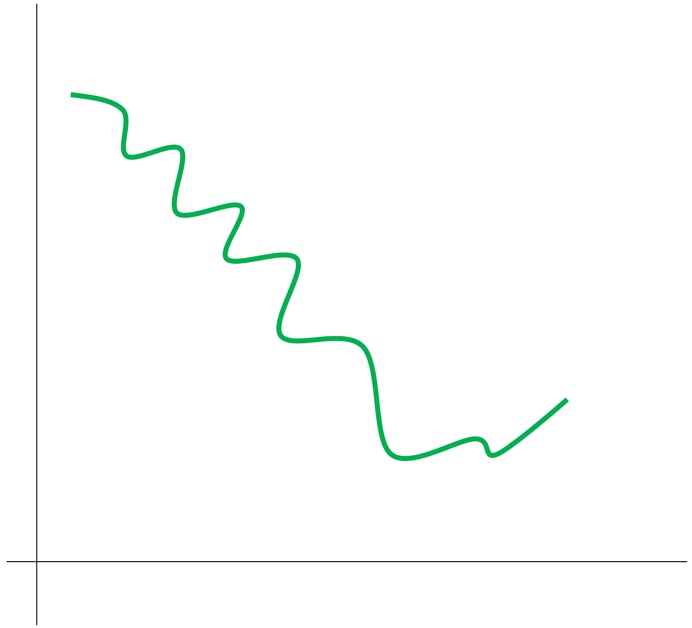


# Solutions for overfitting

- **More training data!**
- **Reduce the number of features**
- **Regularization**

# Regularization

- **Let's not have too big numbers in the weight**



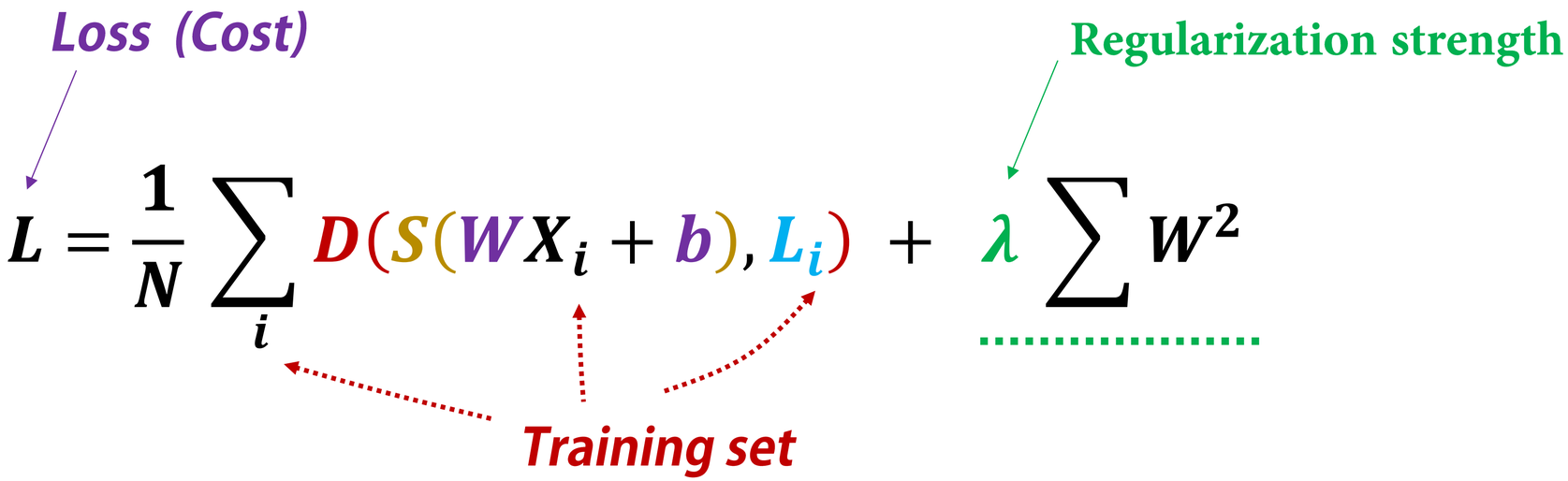
*Loss (Cost)*

$$L = \frac{1}{N}\sum_{i} D(S(WX_i + b), L_i)$$

*Training set*

# Regularization

- **Let's not have too big numbers in the weight**



# Regularization

- **Let's not have too big numbers in the weight**

```
l2reg = 0.001 * tf.reduce_sum(tf.square(W))
```

*Loss (Cost)*

Regularization strength

$$L = \frac{1}{N}\sum_{i} D(S(WX_i + b), L_i) + \lambda \sum W^2$$

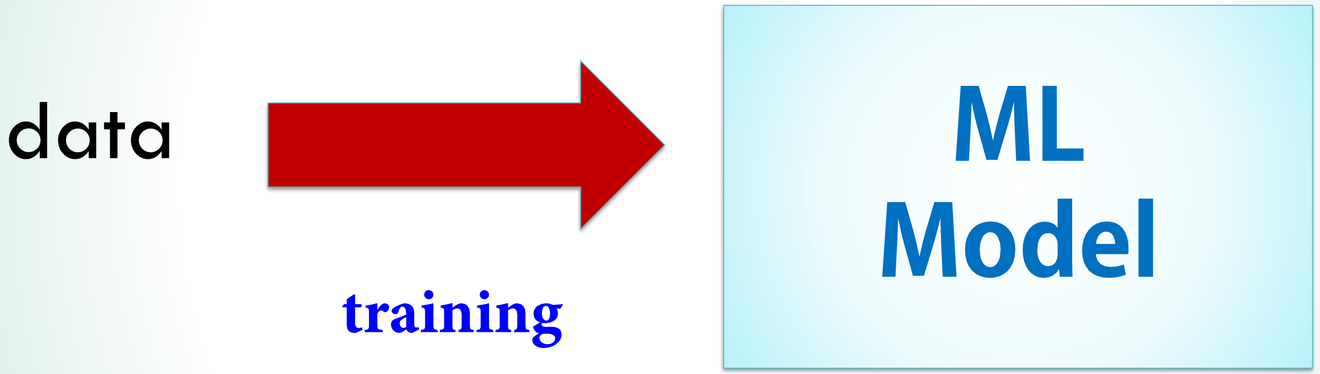
*Training set*

# Summary

- **Learning rate**
- **Data preprocessing**
- **Overfitting**
  - ▷ **More training data**
  - ▷ **Regularization**

# Application & Tips:
# Learning and test data sets

# Performance evaluation: is this good?



# Evaluation using training set?

| Size | Price |
|---|---|
| 2104 | 400 |
| 1600 | 330 |
| 2400 | 369 |
| 1416 | 232 |
| 3000 | 540 |
| 1985 | 300 |
| 1534 | 315 |
| 1427 | 199 |
| 1380 | 212 |
| 1494 | 243 |

training set

ML Model

- 100% correct (accuracy)
- Can memorize

# Evaluation using training set?

| Size | Price |
|---|---|
| 2104 | 400 |
| 1600 | 330 |
| 2400 | 369 |
| 1416 | 232 |
| 3000 | 540 |
| 1985 | 300 |
| 1534 | 315 |

training set

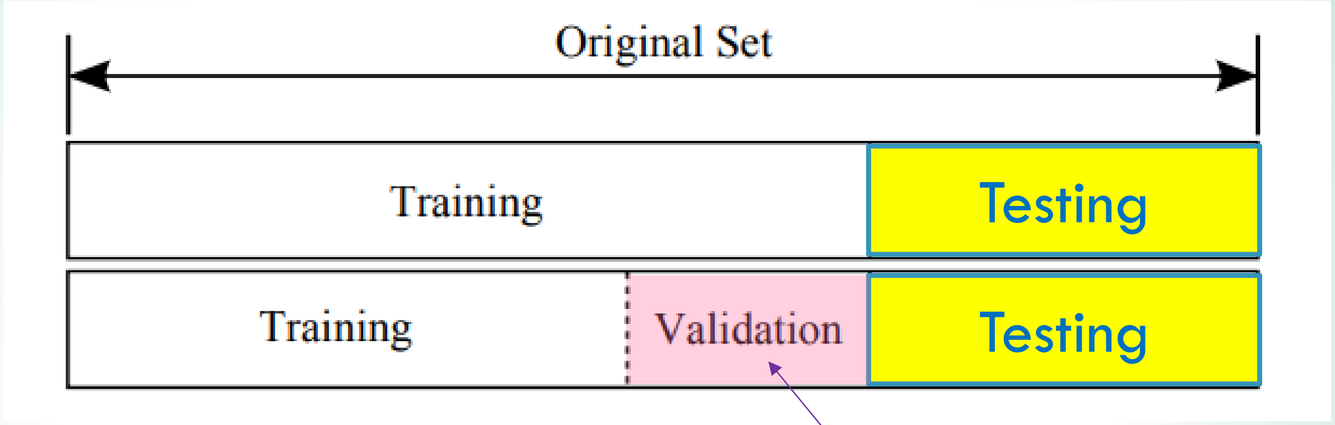
test set

ML Model

**Compare**

$Y$ , $\hat{Y}$

# Training, validation and test sets



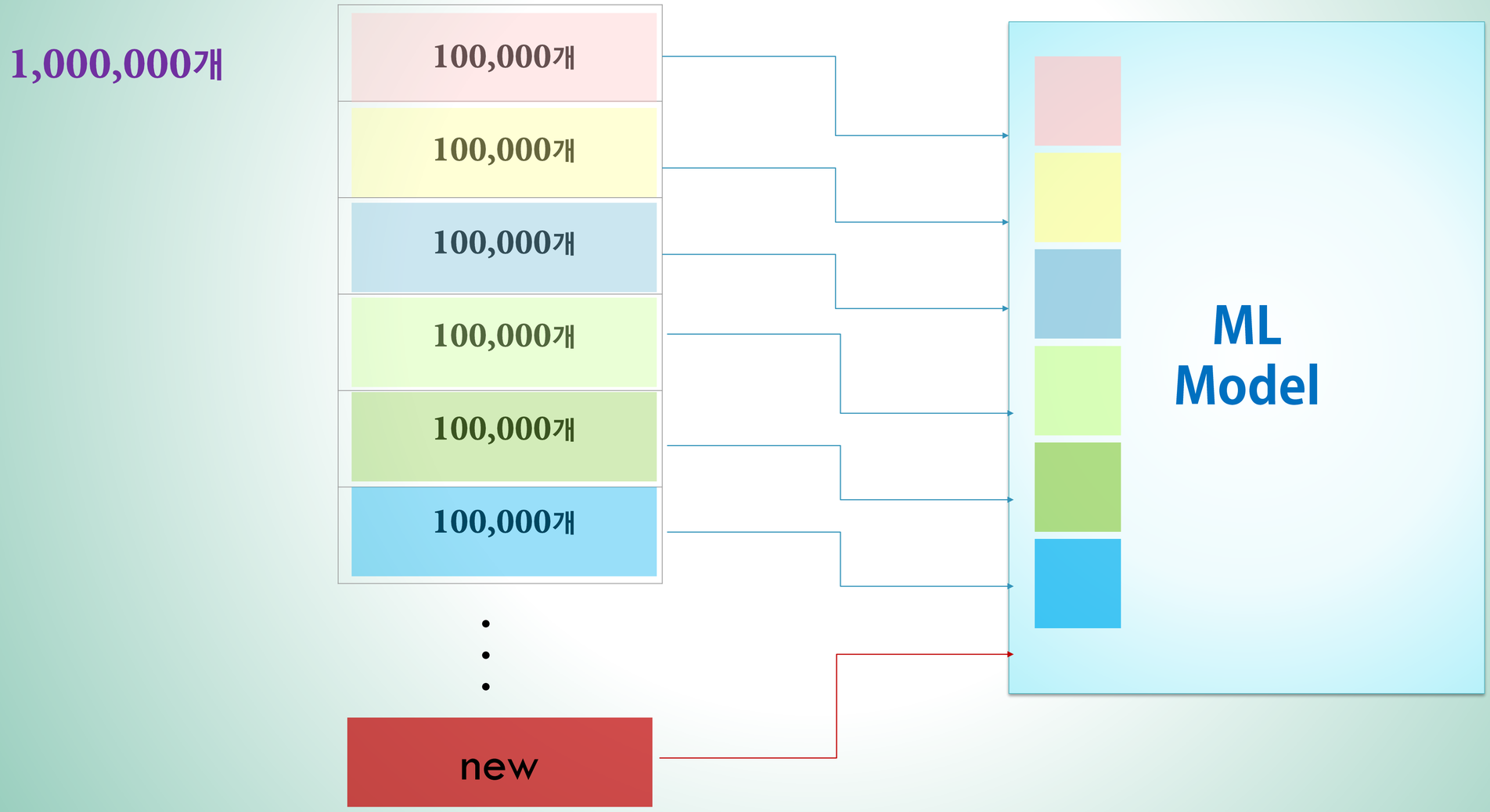

*Cost (Loss)*

**Regularization strength**

$$L = \frac{1}{N} \sum_i D\left(S\left(WX_i + b\right), L_i\right) + \lambda \sum W^2$$

*Training set*

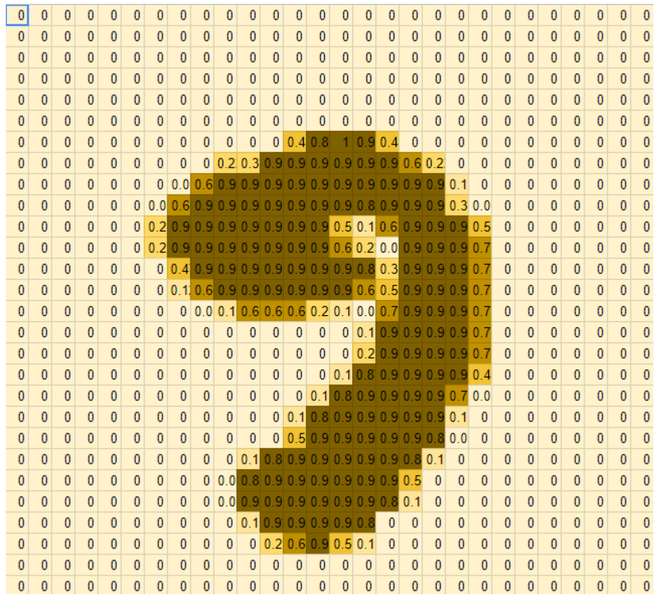
# Online learning



# MNIST Dataset

train-images-idx3-ubyte.gz: training set images (9912422 bytes)
train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)

# MNIST Dataset



https://docs.google.com/spreadsheets/d/1Phwp2oCrHz08939_T0LqMyX-kbxun96x8yq3g0Rgitl/edit#gid=0

# Accuracy

- **How many of your predictions are correct?**
- **95% ~ 99%?**