Computer Vision

CVI620

Session 17
03/2025

What is Left?

11 sessions

1. Optimization and Loss Function
2. Code + Logistic Regression
3. ML and Images
4. Perceptron and Neural Networks
5. Deep Neural Networks
6. Convolution Neural Networks (CNN)
7. Advanced CNNs
8. Project
9. Segmentation
10. Introduction to object detection and image generation methods with AI
11. Project

Agenda

01 Complete GD

02 Example

03 Types of GD

04 Regression with GD

04 Logistic Regression

Review

- Some datasets have millions of features -> impossible to calculate
- They are not always linear
- Some problems do not have closed form formals

Solution -> Optimization

Optimization

Start random in space

Take gradual steps towards your goal

Not the best best answer but a solution close to the best

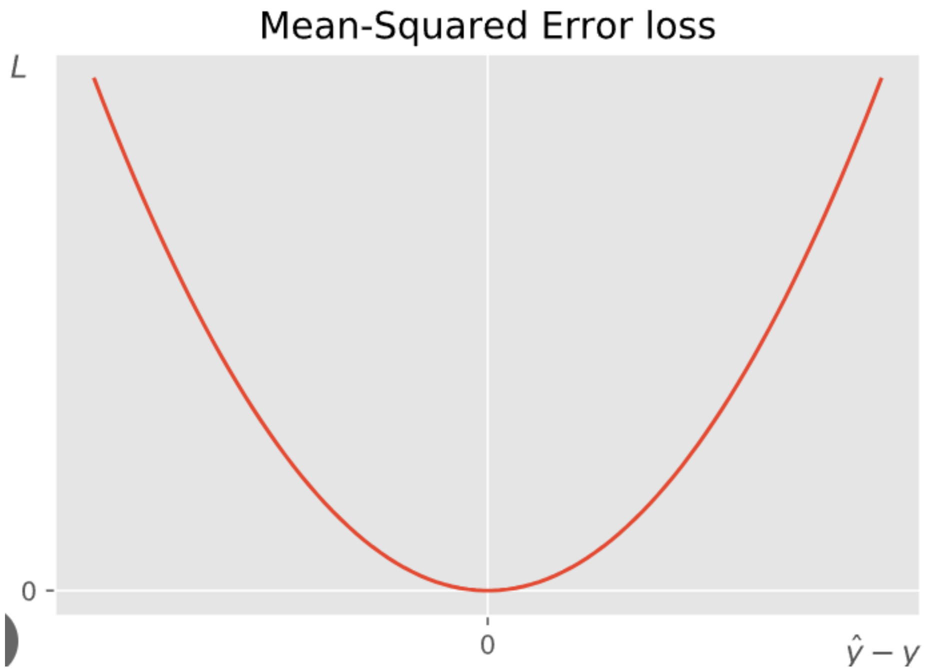
Gradient Descend

- The problem was how we get to update the line in LR
- Gradient descend does this with calculating derivatives again!
- It updates parameters by moving in the opposite direction of their derivatives with respect to loss!

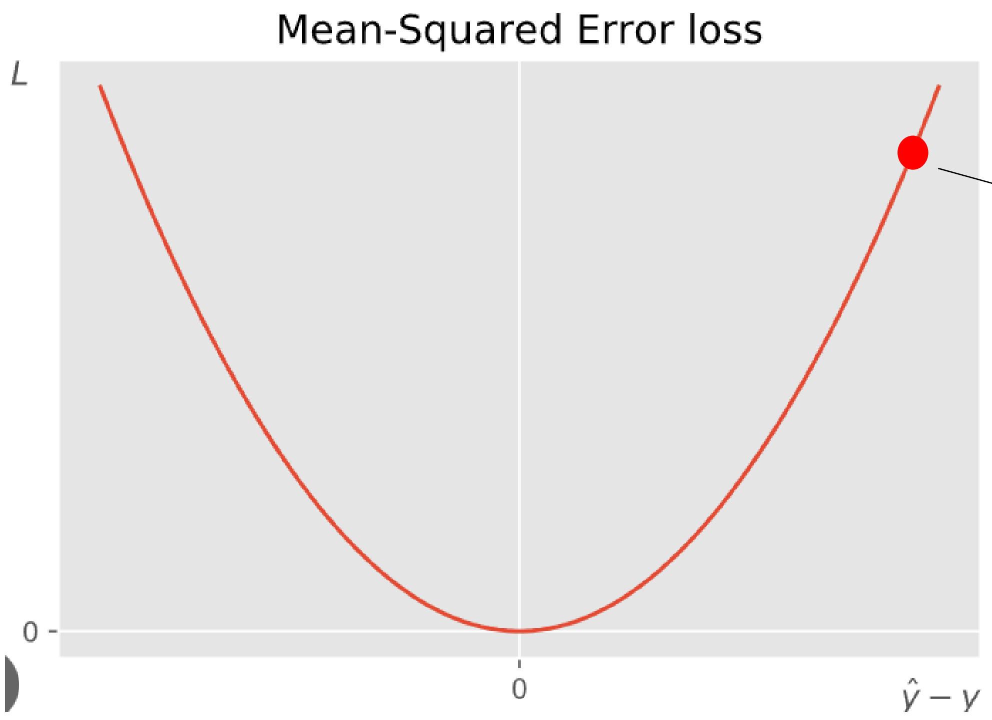
But what do we mean?

opposite direction of parameter derivatives with respect to loss!

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

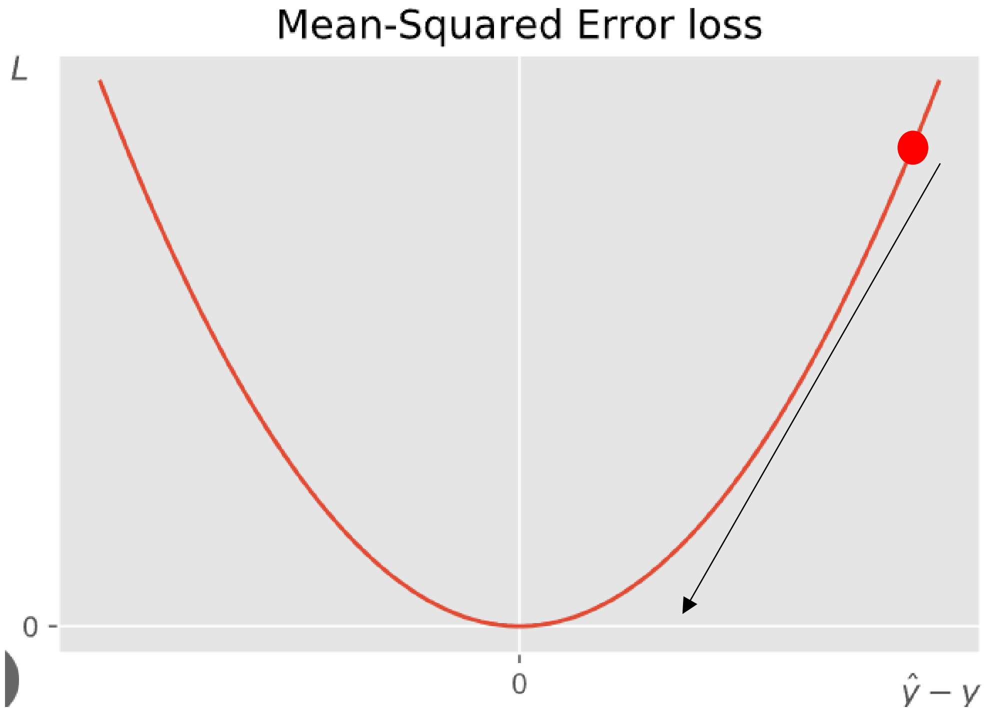


opposite direction of parameter derivatives with respect to loss!



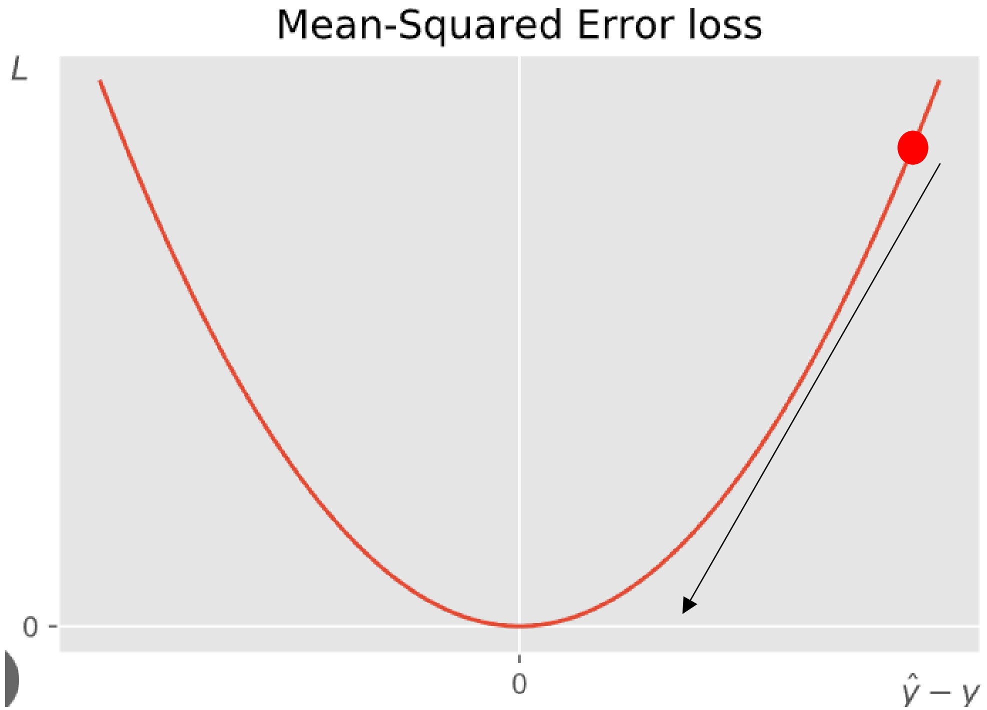
Our first average loss with the random line

opposite direction of parameter derivatives with respect to loss!



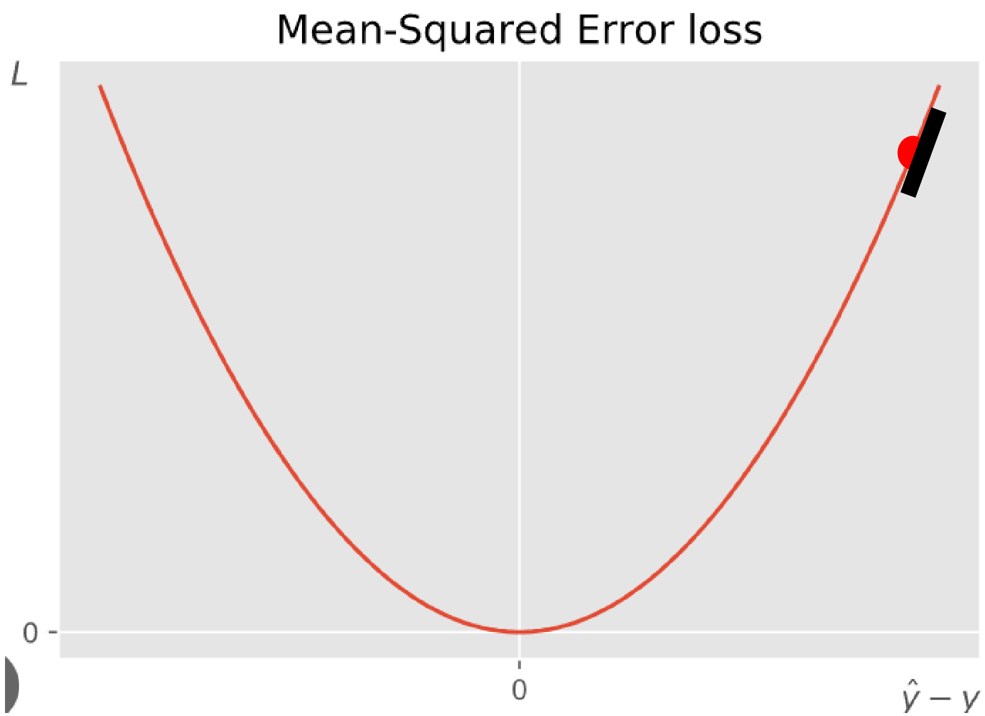
We want this loss to be as close as to 0

opposite direction of parameter derivatives with respect to loss!

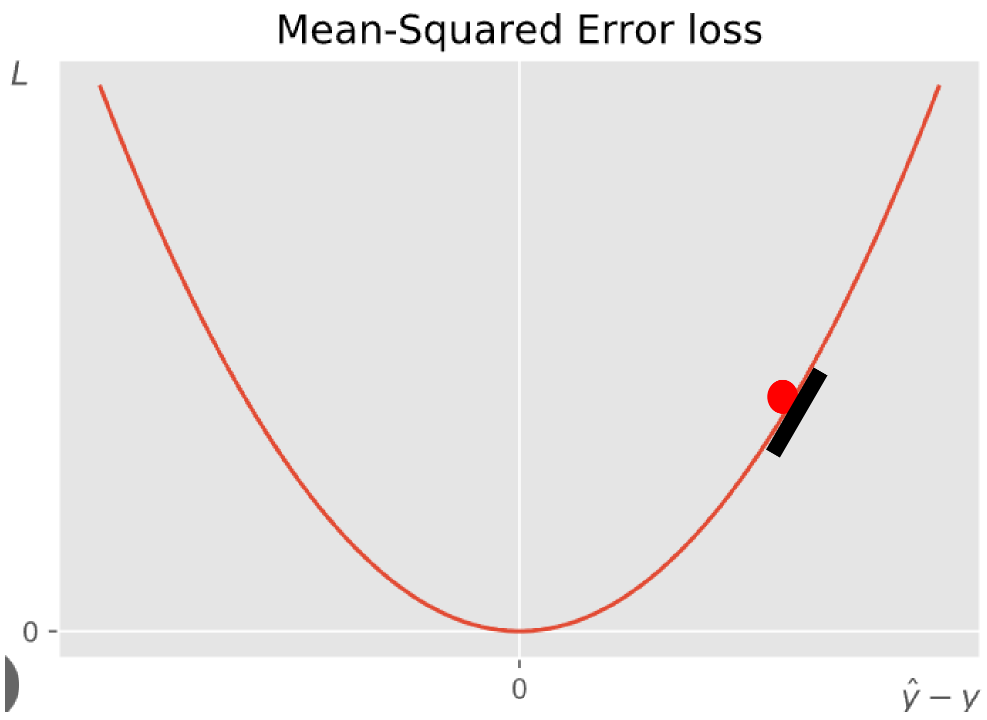


In order to move towards that direction, we have to move in the opposite direction of slope (derivate)

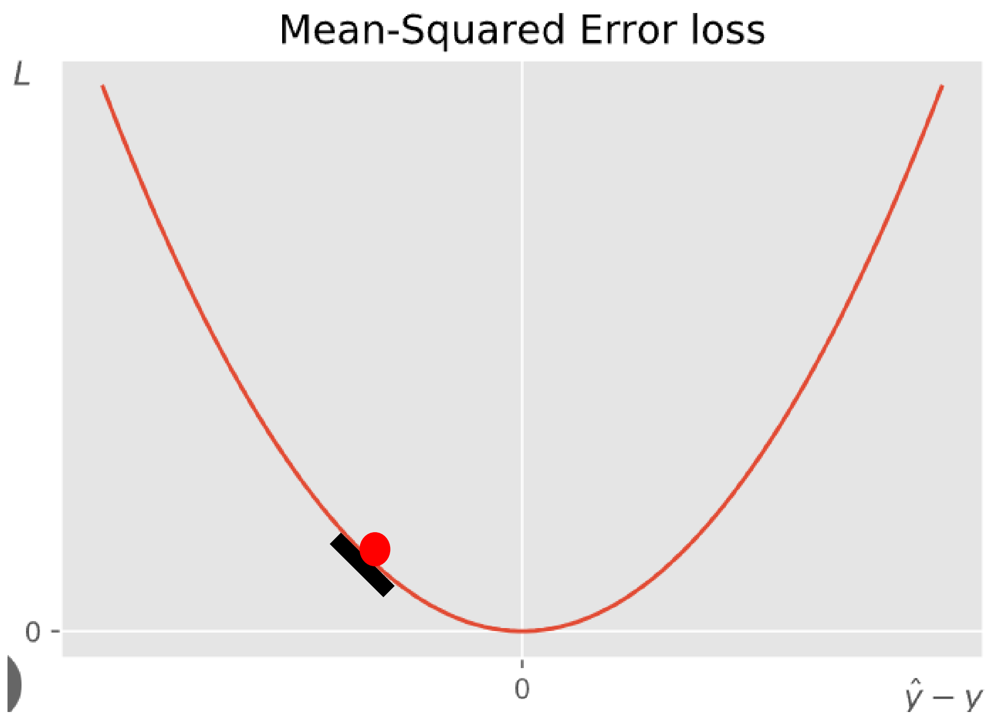
opposite direction of parameter derivatives with respect to loss!



opposite direction of parameter derivatives with respect to loss!

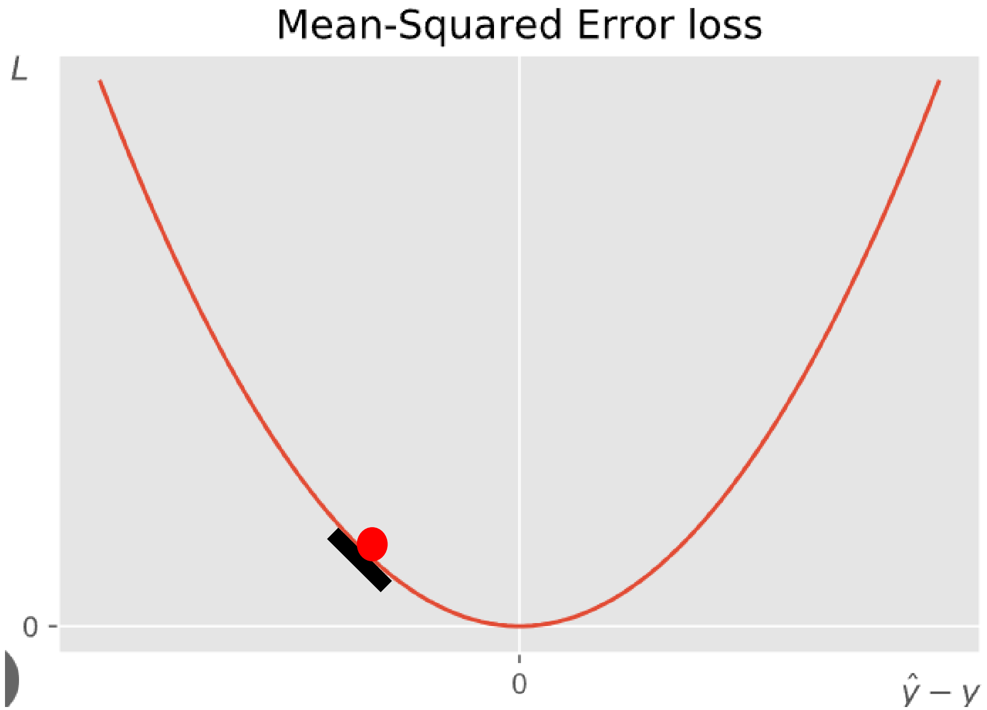


opposite direction of parameter derivatives with respect to loss!



$$w^{+} = w^{-} - \frac{\partial L}{\partial w}$$

opposite direction of parameter derivatives with respect to loss!



$$w^{+} = w^{-} - \alpha \frac{\partial L}{\partial w}$$

Now let's see mathematically!

- opposite direction of **parameter derivatives** with respect to loss!
- Loss is calculated:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Now let's see mathematically!

- opposite direction of **parameter derivatives** with respect to loss!
- Loss is calculated:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Given in data

2x+4

Now let's see mathematically!

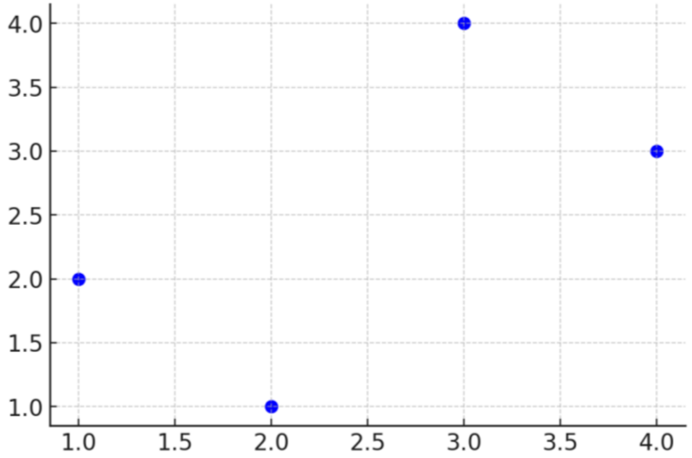
$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

2x+4

Given in data

Now let's see mathematically!

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

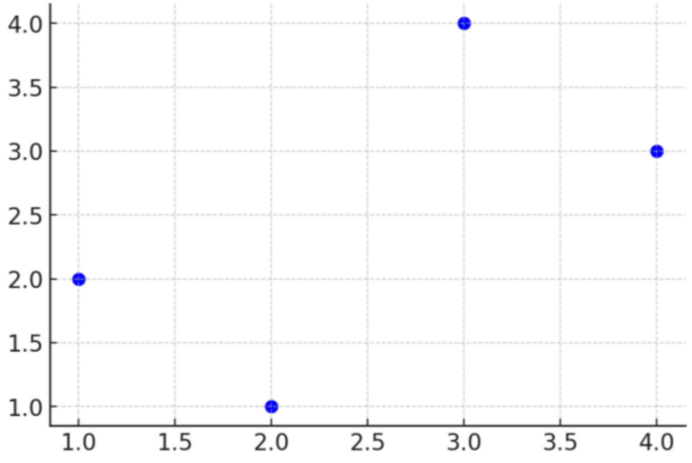


Data = (1,2), (2,1), (3,4), (4,3)
y′ = 2x + 4

Loss = [$(2-6)^2$ + $(1-8)^2$ + $(4-10)^2$ + $(3-12)^2$] / 4 = [16 + 49 + 36 + 81] / 4 = 45.5

Now let's see mathematically!

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$



Data = (1,2), (2,1), (3,4), (4,3)

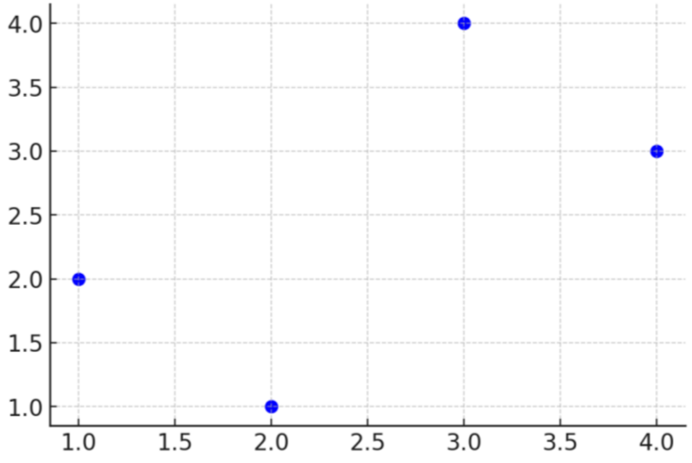
y' = 2x + 4

Loss = [$(2\text{-}6)^2$+ $(1\text{-}8)^2$+ $(4\text{-}10)^2$+ $(3\text{-}12)^2$] / 4 = [16 + 49 + 36 + 81] / 4 = 45.5

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y - (ax + b))^2$$

Now let's see mathematically!

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$



Data = (1,2), (2,1), (3,4), (4,3)
y' = 2x + 4

Loss = [$(2-6)^2 + (1-8)^2 + (4-10)^2 + (3-12)^2$] / 4 = [16 + 49 + 36 + 81] / 4 = 45.5

$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - (mx_i + c))^2$$

$$E = \frac{1}{n} \sum_{i=0}^{n} (y_i - (mx_i + c))^2$$

$$D_m = \frac{1}{n} \sum_{i=0}^{n} 2(y_i - (mx_i + c))(-x_i)$$

$$D_m = \frac{-2}{n} \sum_{i=0}^{n} x_i(y_i - \bar{y}_i)$$

$$m = m - \alpha \times D_m$$

$$D_c = \frac{-2}{n} \sum_{i=0}^{n} (y_i - \bar{y}_i)$$

$$c = c - \alpha \times D_c$$

$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - (mx_i + c))^2$$

$$D_m = \frac{1}{n}\sum_{i=0}^{n} 2(y_i - (mx_i + c))(-x_i)$$

$$D_m = \frac{-2}{n}\sum_{i=0}^{n} x_i(y_i - \bar{y}_i)$$

$$m = m - \alpha \times D_m$$

$$\boxed{\text{new m = 2 - 0.01 * 2 = 0.02}}$$

$$D_c = \frac{-2}{n}\sum_{i=0}^{n}(y_i - \bar{y}_i)$$

$$c = c - \alpha \times D_c$$

$$\boxed{\text{new c = 4 - 0.01 * 2 = 2.02}}$$

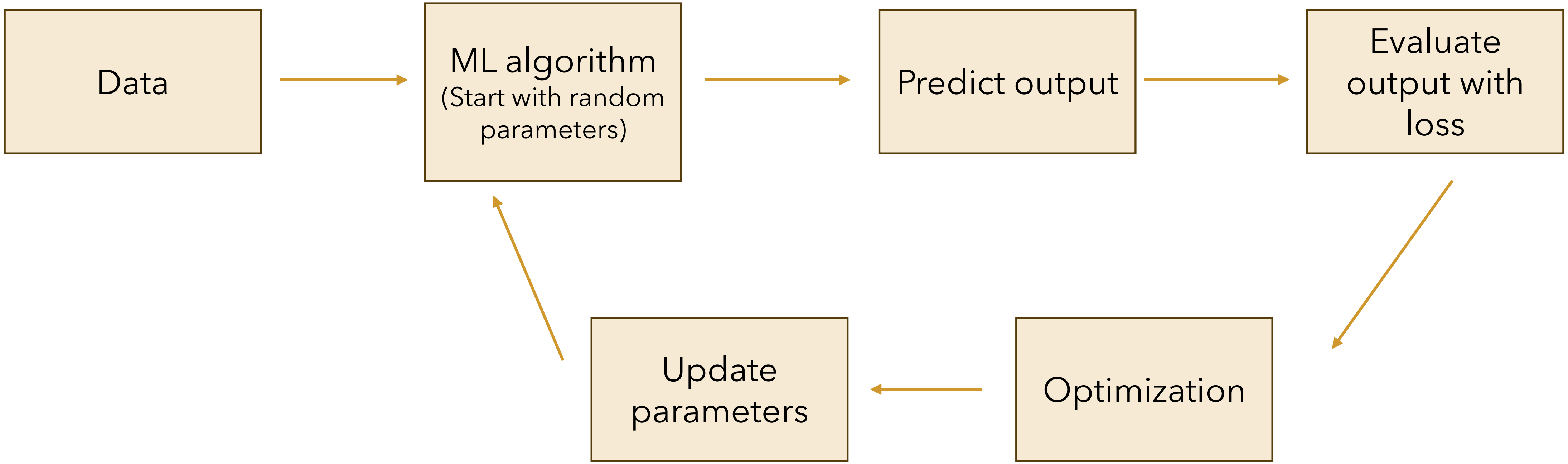
y = 0.02*x + 2.02

New line: $y = 0.02*x + 2.02$

Previous line: $y = 2x+4$

Best fitted line: $y = x + 0.6$

Framework


Data
ML algorithm
(Start with random parameters)
Predict output
Evaluate output with loss
Optimization
Update parameters

Types of GD

Mini Batch GD

1< Batch size < n

Stochastic GD

Batch size = 1

Batch GD

Batch size = n

Code

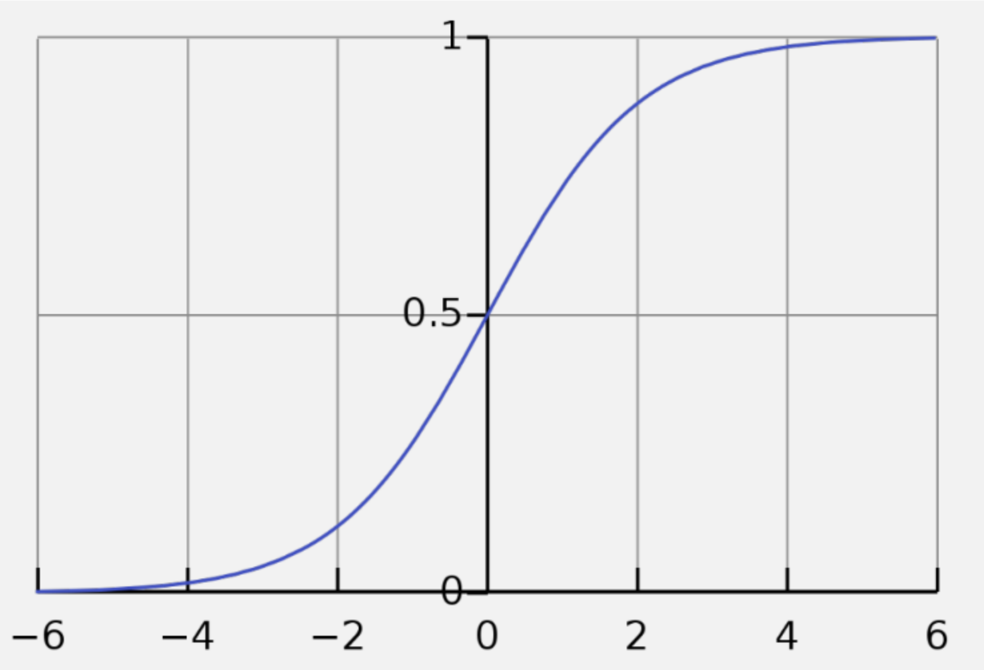
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.preprocessing import StandardScaler

# DATA
dataset = pd.read_csv('S16/petrol_consumption.csv')
X = dataset[['Petrol_tax', 'Average_income', 'Paved_Highways', 'Population_Driver_licence(%)']]
y = dataset['Petrol_Consumption']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# MODEL
regressor = LinearRegression()
regressor.fit(X_train, y_train)
coeff_df = pd.DataFrame(regressor.coef_, X.columns, columns=['Coefficient'])
print(coeff_df)
print(regressor.intercept_)

sgd_regressor = SGDRegressor(max_iter=1000)
sgd_regressor.fit(X_train, y_train)
sgdcoeff_df = pd.DataFrame(sgd_regressor.coef_, X.columns, columns=['Coefficient'])
print(sgdcoeff_df)
print(sgd_regressor.intercept_)

# RESULTS AND EVALUATION
y_pred = regressor.predict(X_test)
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print('Mean Absolute Error:', mean_absolute_error(y_test, y_pred))

y_pred_gd = sgd_regressor.predict(X_test)
df_gd = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_gd})
print('Mean Absolute Error:', mean_absolute_error(y_test, y_pred))
```

Terms overview

- Train-Test sets
- Normalization
- KNN for classification
- Linear regression for prediction
- Gradient Descent for prediction
- Epoch
- Batch size
- Accuracy
- Loss function

Logistic Regression

Logistic Function



$$y = \frac{1}{1 + e^{-x}}$$

$$e \approx 2.71828$$

Where do you think it is useful?

Classification

$$out = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_{n-1} x_{n-1}$$

$$\longrightarrow$$

$$y = \frac{1}{1 + e^{-out}}$$

Logistic Regression Loss Function

If y=1 and y′=0 or the opposite -> loss should be high

Else -> loss low

$$Loss = -ylog(y') - (1-y)\log(1-y')$$

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

df = pd.read_csv('S17/diabetes.csv')

zero_not_accepted = ['Glucose', 'BloodPressure',
                     'SkinThickness', 'Insulin', 'BMI']

for columns in zero_not_accepted:
    df[columns] = df[columns].replace(0, np.nan)
    mean = int(df[columns].mean(skipna=True))
    df[columns] = df[columns].replace(np.nan, mean)


X = df.drop(columns=['Outcome'])
y = df['Outcome']

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True)

sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)

model = LogisticRegression()
model.fit(x_train, y_train)

preds = model.predict(x_test)
print(accuracy_score(y_test, preds))
```