* K Means
* Knn
* Conv2d
* ~~Logistic regression~~
* Attention ?
* Implement
  + Batch normalization
  + Gradient descent
  + Layer normalization
  + ~~Softmax~~

<https://www.yuan-meng.com/posts/md_coding/>

import numpy as np

class LogisticRegression:

def \_\_init\_\_(self, lr=0.01, epoch=10) -> None:

self.w = None

self.b = 0

self.lr = lr

self.epoch = epoch

def fit(self, X, y):

# get features and rows

self.n\_obs, self.n\_feat = X.shape

self.w = np.zeros(self.n\_feat)

self.X = X

self.y = y

losses = []

for \_ in range(self.epoch):

y\_pred = self.predict(X)

losses.append(self.compute\_loss(y, y\_pred))

# grad\_w

grad\_w = -np.dot(self.X.T, (self.y - y\_pred)) / self.n\_obs

# grad\_b

grad\_b = -np.sum(self.y - y\_pred) / self.n\_obs

self.w = self.w - self.lr\*grad\_w

self.b = self.b - self.lr\*grad\_b

def predict(self, X):

z = X.dot(self.w) + self.b

# sigmoid

p = 1 / (1 + np.exp(-z))

return np.where(p < 0.5, 0, 1)

def compute\_loss(self, y\_true, y\_pred):

# binary cross entropy

y\_zero\_loss = y\_true \* np.log(y\_pred + 1e-9)

y\_one\_loss = (1-y\_true) \* np.log(1 - y\_pred + 1e-9)

return -np.mean(y\_zero\_loss + y\_one\_loss)

Logistic regression

Steps

1. Fit
2. Compute loss
3. Predict

Class Logistic:

Def \_\_init\_():

Self.w = None

Self.b = 0

Self.lr = 0.01

Self.epoch = 0

Def fit(X):

Num\_obs, num\_feat = X.shape

Self.w = np.zeros(num\_feat )

Def predict(x):

Z = x.dot(self.w) + self.b