Fundamentos Básicos para Redes Neurais

Bruno Fernandes

Conteúdo

- Regressão linear e logística
- *Loss function* x *Cost Function*
- Gradiente descendente
- Grafo computacional
- Vetorização
- Broadcasting
- Implementação da regressão logística

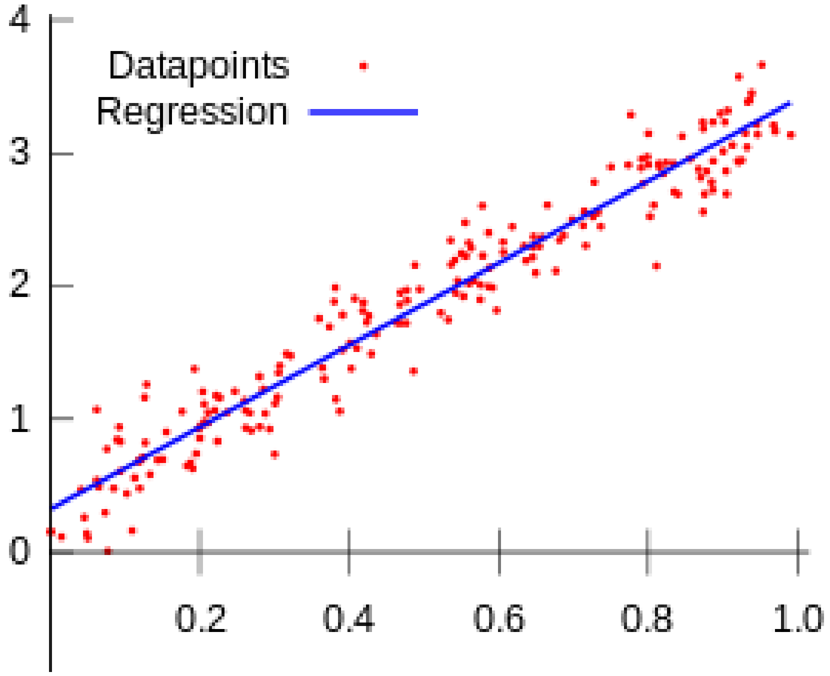
Conteúdo

- **Regressão linear e logística**
- *Loss function* x *Cost Function*
- Gradiente descendente
- Grafo computacional
- Vetorização
- Broadcasting
- Implementação da regressão logística

Regressão Linear

- Em estatística ou econometria, regressão linear é uma equação para se estimar a condicional de uma variável y, dados os valores de algumas outras variáveis x. A regressão, em geral, tem como objetivo tratar de um valor que não se consegue estimar inicialmente.

Wikipédia



Regressão Linear

- $X = \begin{array}{cccc} \vdots & & \vdots & \uparrow \\ X^1 & \ddots & X^m & n_x \\ \vdots & & \vdots & \downarrow \end{array}$ -> tamanho do vetor

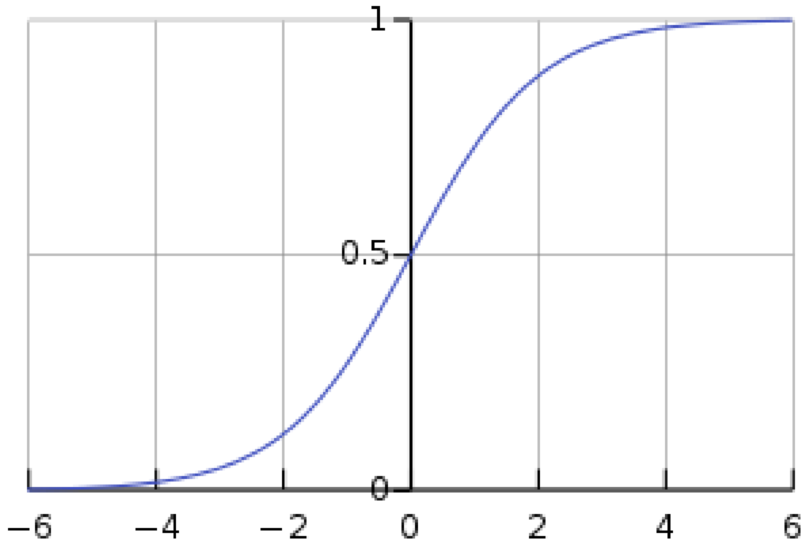
- $X.shape = (n_x, m)$

- $Y.shape = (1, m)$

- $\hat{y} = w^T x + b$ -> não é uma probabilidade

Regressão Logística

- Retorna uma probabilidade
- $\hat{y} = P(y = 1|x)$
- $\hat{y} = \sigma(w^T x + b)$
- $\sigma(z) = \frac{1}{1+e^{-z}}$
  - Se Z é muito grande, σ tende a 1
  - Se Z é um negativo muito grande, σ tende a 0



Conteúdo

- Regressão linear e logística
- ***Loss function* x *Cost Function***
- Gradiente descendente
- Grafo computacional
- Vetorização
- Broadcasting
- Implementação da regressão logística

Loss function x Cost function

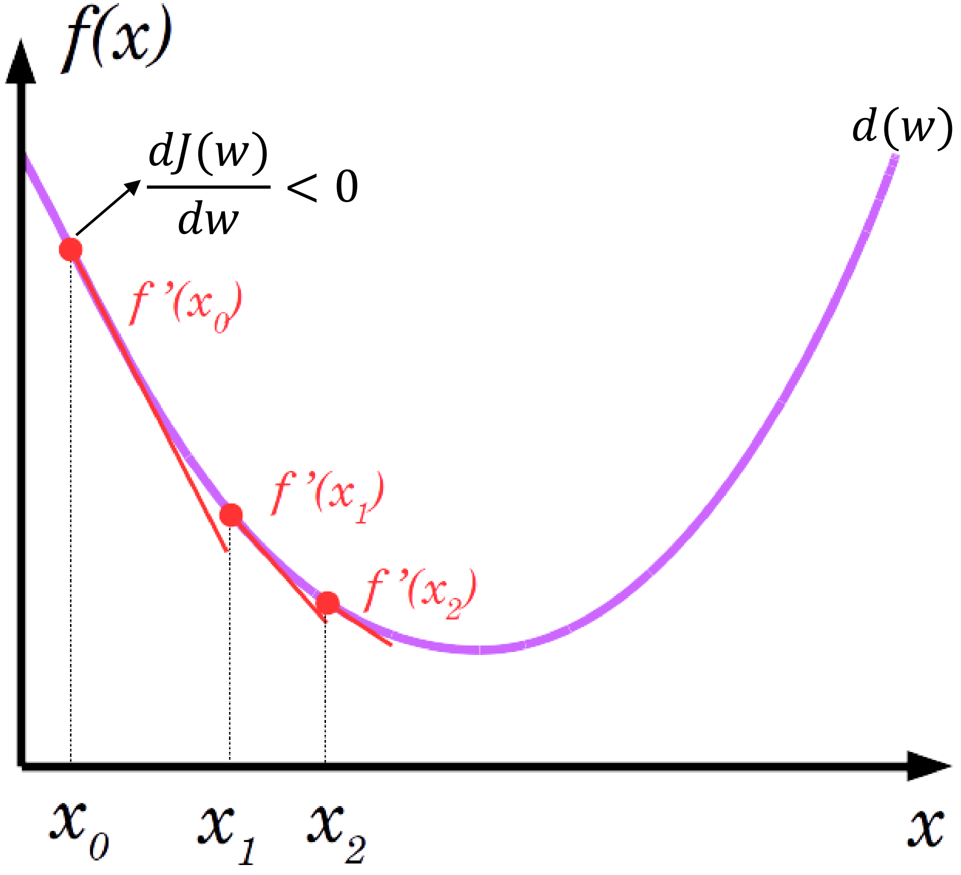
- Loss function
  - Perda associada a um exemplo de treino (utiliza-se uma função convexa para evitar múltiplos mínimos)
  - $L(\hat{y}, y) = -(y \log(\hat{y}) + (1 - y)\log(1 - \hat{y})$
- Cost function
  - Associada aos parâmetros – média de todas loss functions do conjunto
  - $J(w, b) = \frac{1}{m}\sum_{i=1}^{m} L\left(\hat{y}^{(i)}, y^{(i)}\right) = \frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log\left(\hat{y}^{(i)}\right)\left(1 - y^{(i)}\right)\log(1 -\right.$

Conteúdo

- Regressão linear e logística
- *Loss function* x *Cost Function*
- **Gradiente descendente**
- Grafo computacional
- Vetorização
- Broadcasting
- Implementação da regressão logística

Gradiente Descendente

- Método utilizado para minimizar a função de custo



$$w := w - \alpha \frac{dJ(w,b)}{dw}$$

$$b := b - \alpha \frac{dJ(w,b)}{db}$$

Derivadas

- Derivada = inclinação (*slope*)
- A derivada em um ponto de uma função y=f(x) representa a taxa de variação instantânea de y em relação a x neste ponto

f(a)=3a
Para a=2, f(a)=6
Para a=2.001, f(a)=6.003
Inclinação em a=2 é 3
Para a=5, f(a)=15
Para a=5.001, f(a)=15.003
Inclinação em a=5 também é 3

$$\frac{df(a)}{da} = 3$$

$f(a)=a^2$
Para a=2, f(a)=4
Para a=2.001, f(a)=4.004001
$\frac{df(a)}{da} = 4$, quando a=2
Para a=5, f(a)=25
Para a=5.001, f(a)=25.010001
$\frac{df(a)}{da} = 10$, quando a=5

$$\frac{df(a)}{da} = 2a$$

Derivadas

- A definição formal entretanto é a seguinte
- $f'(\theta) = \lim_{\varepsilon \to 0} \frac{f(\theta+\varepsilon)-f(\theta-\varepsilon)}{2\varepsilon}$

UNIVERSIDADE DE PERNAMBUCO

Conteúdo

- Regressão linear e logística
- *Loss function* x *Cost Function*
- Gradiente descendente
- **Grafo computacional**
- Vetorização
- Broadcasting
- Implementação da regressão logística

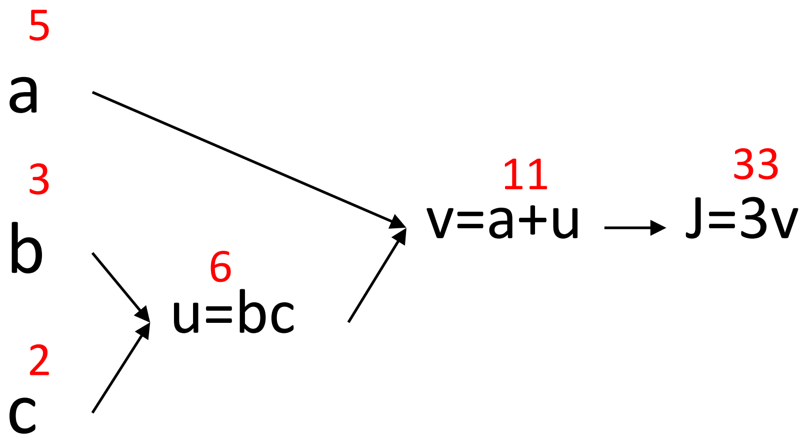
Grafo Computacional

J(ab,c)=3(a+bc)

u=bc

v=a+u

J=3v



A derivada faz o caminho inverso do grafo na propagação backward

$$\frac{dJ}{dv} = 3 \qquad \frac{dJ}{da} = \frac{dJ}{dv}\frac{dJ}{da} = 3 \qquad \frac{dJ}{du} = \frac{dJ}{dv}\frac{dJ}{du} = 3 \qquad \frac{dJ}{db} = \frac{dJ}{du}\frac{dJ}{db} = 6 \qquad \frac{dJ}{dc} = \frac{dJ}{du}\frac{dJ}{dc} = 9$$

$$\frac{dJ}{da} = 1$$

Conteúdo

- Regressão linear e logística
- *Loss function* x *Cost Function*
- Gradiente descendente
- Grafo computacional
- **Vetorização**
- Broadcasting
- Implementação da regressão logística

Vetorização

- Eliminação de for-loops

- Single instruction, multiple data (SIMD)

```
import numpy as np
import time

a = np.random.rand(100,100)
b = np.random.rand(100,100)
inicio = time.time()
z=np.dot(a,b)
print(z.shape)
fim = time.time()
print(fim - inicio)

z2 = np.zeros((100,100))
inicio = time.time()
for i in range(100):
    for j in range(100):
        for k in range(100):
            z2[i][j] += a[i][k]*b[k][j]
print(z2.shape)
fim = time.time()
print(fim - inicio)

>> (100, 100)
>> 0.1940925121307373
>> (100, 100)
>> 1.034226417541504
```

Conteúdo

- Regressão linear e logística
- *Loss function* x *Cost Function*
- Gradiente descendente
- Grafo computacional
- Vetorização
- **Broadcasting**
- Implementação da regressão logística

Broadcasting

- Tratamento de arrays de diferentes tamanhos
  - Sujeito a algumas restrições, o array menor é propagado ao longo do array maior de forma a garantir tamanhos compatíveis

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + 100 = \begin{bmatrix} 101 \\ 102 \\ 103 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 200 & 300 \end{bmatrix} = \begin{bmatrix} 101 & 202 & 303 \\ 104 & 205 & 306 \end{bmatrix}$$

Conteúdo

- Regressão linear e logística
- *Loss function* x *Cost Function*
- Gradiente descendente
- Grafo computacional
- Vetorização
- Broadcasting
- **Implementação da regressão logística**

Implementação da Regressão Logística

- Função sigmoide

```python
def sigmoid(z):
    s = 1 / (1 + np.exp(-z))
    return s
```

Implementação da Regressão Logística

- Fase forward

```python
def forward(w, b, X, Y):
    m = X.shape[1]
    # FORWARD PROPAGATION (FROM X TO COST)
    # compute activation
    A = sigmoid(np.dot(w.T,X) + b)
    # compute cost
    cost = (-1/m) * np.sum(  (Y *np.log(A)) + ((1-Y) * np.log(1-A)) )
    return A, cost
```

Implementação da Regressão Logística

- Fase backward

```python
def backward(A, X, Y):
    m = X.shape[1]
    db = (1/m) * (np.sum(A-Y))
    dw = (1/m)*(np.dot(X,np.subtract(A,Y).T))
    grads = {"dw": dw,
             "db": db}
    return grads
```

Implementação da Regressão Logística

- Otimização

```
def optimize(w, b, X, Y, num_iterations, learning_rate):
    costs = []
    for i in range(num_iterations):
        A, cost = forward(w, b, X, Y)
        grads = backward(A, X, Y)
        dw = grads["dw"]
        db = grads["db"]
        w = w - learning_rate * dw
        b = b - learning_rate * db
        if i % 100 == 0:
            costs.append(cost)
    params = {"w": w, "b": b}
    grads = {"dw": dw, "db": db}
    return params, grads, costs
```

Implementação da Regressão Logística

- Predição

```
def predict(w, b, X):
    m = X.shape[1]
    Y_prediction = np.zeros((1,m))
    w = w.reshape(X.shape[0], 1)
    A = sigmoid(np.dot(w.T,X) + b)
    for i in range(A.shape[1]):
        if (A[0,i] <=0.5):
            Y_prediction[0,i] = 0
        elif (A[0,i] > 0.5):
            Y_prediction[0,i] = 1
        pass
    return Y_prediction
```

Implementação da Regressão Logística

- Modelo

```
def model(X_train, Y_train, X_test, Y_test, num_iterations = 2000, learning_rate = 0.5):
    w = np.zeros((X_train.shape[0], 1))
    b = 0
    parameters, grads, costs = optimize(w, b, X_train, Y_train, num_iterations, learning_rate)
    w = parameters["w"]
    b = parameters["b"]
    Y_prediction_test = predict(w, b, X_test)
    Y_prediction_train = predict(w, b, X_train)
    print("train accuracy: {} %".format(100 - np.mean(np.abs(Y_prediction_train - Y_train)) * 100))
    print("test accuracy: {} %".format(100 - np.mean(np.abs(Y_prediction_test - Y_test)) * 100))
```

Implementação da Regressão Logística

- Imprimindo a função de custo

```
costs = np.squeeze(costs)
plt.plot(costs)
plt.ylabel('cost')
plt.xlabel('iterations (per hundreds)')
plt.title("Learning rate =" + str(learning_rate))
plt.show()
```

Carregando Conjunto de Dados

- Arquivos: train_catvnoncat.h5 e test_catvnoncat.h5

```python
import numpy as np
import h5py
def load_dataset():
    train_dataset = h5py.File('datasets/train_catvnoncat.h5', "r")
    train_set_x_orig = np.array(train_dataset["train_set_x"][:]) # your train set features
    train_set_y_orig = np.array(train_dataset["train_set_y"][:]) # your train set labels

    test_dataset = h5py.File('datasets/test_catvnoncat.h5', "r")
    test_set_x_orig = np.array(test_dataset["test_set_x"][:]) # your test set features
    test_set_y_orig = np.array(test_dataset["test_set_y"][:]) # your test set labels

    classes = np.array(test_dataset["list_classes"][:]) # the list of classes

    train_set_y_orig = train_set_y_orig.reshape((1, train_set_y_orig.shape[0]))
    test_set_y_orig = test_set_y_orig.reshape((1, test_set_y_orig.shape[0]))

    return train_set_x_orig, train_set_y_orig, test_set_x_orig, test_set_y_orig, classes
```

Ajustando os dados

```
def load_processed_cat_dataset(train_set_x_orig, test_set_x_orig):
    train_set_x_flatten = train_set_x_orig.reshape(train_set_x_orig.shape[0], -1).T
    test_set_x_flatten = test_set_x_orig.reshape(test_set_x_orig.shape[0], -1).T
    train_set_x = train_set_x_flatten/255.
    test_set_x = test_set_x_flatten/255.
    return train_set_x, test_set_x
```

UNIVERSIDADE DE PERNAMBUCO

Executando

```
train_set_x_orig, train_set_y, test_set_x_orig, test_set_y, classes = load_dataset()
train_set_x, test_set_x = load_processed_cat_dataset(train_set_x_orig, test_set_x_orig)
model(train_set_x, train_set_y, test_set_x, test_set_y, num_iterations = 2000, learning_rate = 0.005)
```

Exercício

- Tente melhorar a acurácia do teste alterando o número de épocas ou a taxa de aprendizagem

Fundamentos Básicos para Redes Neurais

Bruno Fernandes

UNIVERSIDADE DE PERNAMBUCO