Pieter Abbeel

Neural Network Obrok Logistic Regression

Lojistik Regresyon

İkili sınıflandırma için bir algoritmadır. Bir resimdeki cismin kedi olup olmadığının tahmini ikili sınıflandırmaya örnektir.

Notation

$(x, y) \quad x \in \mathbb{R}^{n_x} \quad , \quad y \in \{0, 1\}$

m => training example: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$

$m = m_{train}$ $\quad$ m_{test} = test examples

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad n_x \qquad Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix}$$

m

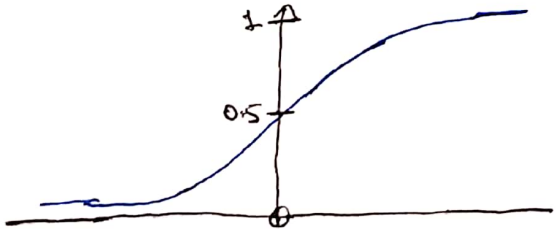
x veriliyor, $\hat{y}$ isteniyor. $\hat{y} = P(y=1 \mid x)$ $\quad 0 \le \hat{y} \le 1$

$x \in \mathbb{R}^{n_x}$

Parameters: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

Output: $\hat{y} = \text{sigmoid}(\underbrace{w^T x + b}_{z})$

z değeri çok büyük veya çok küçük çıkacağı için sigmoid fonksiyonu ile 0 ve 1 arasına düşürülür.



$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Eğer z çok büyükse $\sigma(z) \cong \frac{1}{1+0} = 1$

Eğer z negatif sayı ise $\cong \frac{1}{1 + \text{Bignum}} \cong 0$

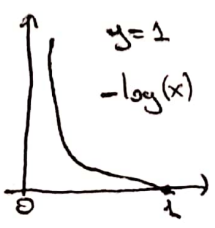
$y = b + w_1 x_1 + w_2 x_2 + \dots + w_m x_m$

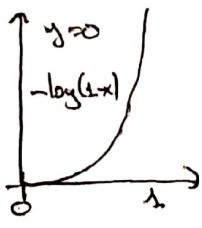
Not: Lojistik regresyonda bulmaya çalıştığımız b ve w ağırlıklarıdır. Bu şekilde tahminimizi yaparız.

Lojistik Regresyon Cost Function

loss $\quad \mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log(1-\hat{y}))$ Tek örnek için kaçar

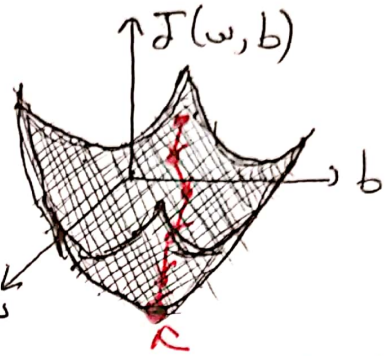
cost $\quad J(w, b) = \frac{-1}{m} \sum_{i=1}^{m} + (y \log \hat{y} + (1-y) \log(1-\hat{y}))$



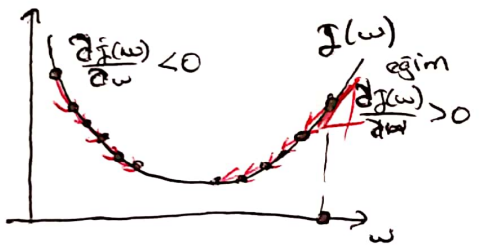


Gradient Descent

$J(w,b)$'yi minimize etmek için en uygun w ve b parametrelerini arıyoruz.



Cost function dışbükey bir fonksiyonu sebebiyet verdiği için gradient descent'i kullanarak tek bir optimum değere ulaşabiliriz. Dışbükey olmayan fonksiyonlarda birden çok yerel optimum noktası olduğundan net bir değer bulamayız. (mean square cost function'da bu yüzden kullanmadık)



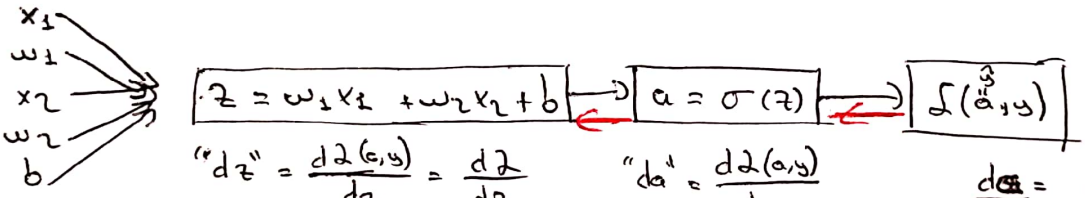
repeat {

$$w := w - \alpha \frac{\partial J(w)}{\partial w}$$

(learning rate)

}

Logistic Regression Derivaties



"dz" $= \frac{dL(a,y)}{dz} = \frac{dL}{dz}$

$= \frac{dL}{da} \cdot \frac{da}{dz}$ $a.(1-a)$

$= \left(\frac{-y}{a} + \frac{1-y}{1-a}\right) . a.(1-a)$

$= a - y$

"da" $= \frac{dL(a,y)}{da}$

$= \frac{-y}{a} + \frac{1-y}{1-a}$

$\frac{da}{dz} = \frac{1}{1+e^{-z}}$

$=$

$\frac{dL}{dw_1} =$ "dw_1" $= x_1 . dz$ "dw_2" $= x_2 dz$ $db = dz$

$w_1 := w_1 - \alpha\, dw_1$

$w_2 := w_2 - \alpha\, dw_2$

$b := b - \alpha\, db$

$J = 0$; $dw_1 = 0$; $dw_2 = 0$; $db = 0$;

For $i = 1$ to m

$z^{(i)} = w^T x^{(i)} + b$

$a^{(i)} = \sigma(z^{(i)})$

$J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$

$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$

$db \mathrel{+}= dz^{(i)}$

$J /= m$; $dw_1 /= m$; $dw_2 /= m$; $db /= m$;

$w_1 := w_1 - \alpha\, dw_1$

$w_2 := w_2 - \alpha\, dw_2$

$b := b - \alpha\, db$

$\min_\theta J(\theta)$:

repeat {

$$w_j := w_j - \frac{\alpha}{m} \sum_{i=1}^{m} \left(a^{(i)} - y^{(i)}\right) x_j^{(i)}$$

}

What is vectorization?

$$z = w^T x + b$$

Non-Vectorizel

```
z=0
for i in range(n-x):
    z+ = w[i] * x[i]
z+=b
```

Vectorizel

```
z = np.dot(w,x) + b
```

GPU, CPU } SIMD - single instrution multiple data

SIMD ile paralel veri işleme ile çok kısa sürede bir sürü veri işlenebiliyor. Numpy gibi kütüphanelerde bunu kullanıyor. GPU bu işte çok iyi ancak CPU ile de yapılabiliyor.

Not Derin öğrenmedeki en önemli nokta döngülerden kaçınmaktır.

repeat {

$$z = w^T x + b$$

$$= np.dot(w.T, x) + b$$

$$A = \sigma(z)$$

$$dz = A - Y$$

$$dw = \frac{1}{m} . X . dz$$

$$db = \frac{1}{m} . np.sum(dz)$$

}

$$w = w - \alpha dw$$

$$b = b - \alpha db$$

$$\theta = w \qquad x w^T + b$$

$$\theta := \theta - \frac{\alpha}{m} X^T (\sigma(X\theta) - y)$$

Not Logistic regresyonun vectorize edilmiş genel cost functionlı aşağıdadır.

$$J(w) = \frac{1}{m} . (-y^T \log(h) - (1-y)^T \log(1-h))$$

$$h = \sigma(w^T X + b)$$

Not Bir nöron lineer fonksiyon ($z = wx + b$) hesaplar ardından bir aktivasyon fonksiyonu hesaplar.

Not Numpy'de broadcasting sayesinde kopyalamalar yapılarak +, -, /, * işlemleri yapılır.

$a(m,n)$

$b(m,x)$

or

$b(x,n)$ olması gerekir.