

# Machine Learning

# Contents

Table of contents	1
1 Regression	2
1.1 Linear regression . . . . .	2
1.1.1 Squared error cost function . . . . .	2
1.1.2 Gradient descent . . . . .	2
1.2 Multiple linear regression . . . . .	2
1.3 Logistic regression . . . . .	3
1.4 Softmax regression . . . . .	3
1.5 Feature scaling: z-score normalization . . . . .	3
1.6 Over / underfitting . . . . .	4
1.6.1 Regularization . . . . .	4
2 Neural networks	5
2.1 Choosing an activation function . . . . .	5
2.2 Forward propagation . . . . .	5
2.3 Back propagation . . . . .	5

# 1 Regression

## 1.1 Linear regression

### 1.1.1 Squared error cost function

Measures how well line fits training data

$m$  = num of training examples

$\vec{x}$  holds training example x values (length  $m$ )

$\vec{y}$  holds training example y values (length  $m$ )

$\hat{y}^{(i)} = w\vec{x}^{(i)} + b$

$$J(w, b) = \frac{1}{2m} \sum_{i=1}^m (\hat{y}^{(i)} - \vec{y}_i)^2$$

$\frac{1}{m}$  finds average error for larger data sets,  $\frac{1}{2m}$  makes later calculations neater

### 1.1.2 Gradient descent

Find  $w, b$  for minimum of cost function  $J(w, b)$

1. Start with some  $w, b$  (commonly 0, 0)
2. Look around starting point and find direction that will move the point furthest downwards for a small step size

$\alpha$  = learning rate

Must simultaneously update  $w$  and  $b$

$$\begin{aligned}w_1 &= w_0 - \alpha \frac{\partial}{\partial w} J(w_0, b_0) \\b_1 &= b_0 - \alpha \frac{\partial}{\partial b} J(w_0, b_0) \\ \frac{\partial}{\partial w} J(w, b) &= \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - \vec{y}_i) \vec{x}^{(i)} \\ \frac{\partial}{\partial b} J(w, b) &= \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - \vec{y}_i)\end{aligned}$$

## 1.2 Multiple linear regression

$n_f$  = number of features

$m$  = number of data points

$\vec{w}$  = vector of weights (length  $n_f$ )

$X$  is a list of x vectors which hold  $n_f$  features (size  $m \times n_f$ )

Sum of predictions of all features is the prediction of multiple linear reg

$$f_{\vec{w}, b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$

Gradient descent

$$\begin{aligned}\vec{w}_j &= \vec{w}_j - \alpha \frac{\partial}{\partial \vec{w}_j} J(\vec{w}, b) \\b &= b - \alpha \frac{\partial}{\partial b} J(\vec{w}, b)\end{aligned}$$

Cost function and its partial derivatives

$$\begin{aligned}J(\vec{w}, b) &= \frac{1}{2m} \sum_{i=1}^m (f_{\vec{w}, b}(X^{(i)}) - \vec{y}_i)^2 \\ \frac{\partial}{\partial \vec{w}_j} J(\vec{w}, b) &= \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(X^{(i)}) - \vec{y}_i) X_j^{(i)} \\ \frac{\partial}{\partial b} J(\vec{w}, b) &= \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(X^{(i)}) - \vec{y}_i)\end{aligned}$$

### 1.3 Logistic regression

Sigmoid function

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$0 < g(z) < 1$$

$$z = f_{\vec{w}, b}(\vec{x})$$

The output of  $g$  can be interpreted as the "probability" that class is 1.

ex.  $g(z) = 0.7$  means there is a 70% chance  $y$  is 1

Logistic regression requires a new cost function because  $f_{\vec{w}, b}(\vec{x})$  for logistic regression is non-convex, trapping gradient descend in local minima.

Cost function

$$\hat{y}^{(i)} = g(f_{\vec{w}, b}(X^{(i)}))$$

$$J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^m L(g_z, \vec{y}_i)$$

$$L(\hat{y}^{(i)}, \vec{y}_i) = \begin{cases} -\log(\hat{y}^{(i)}) & \text{if } \vec{y}_i = 1 \\ -\log(1 - \hat{y}^{(i)}) & \text{if } \vec{y}_i = 0 \end{cases}$$

Simplified form

$$L(\hat{y}^{(i)}, \vec{y}_i) = -\vec{y}_i \log(\hat{y}^{(i)}) - (1 - \vec{y}_i) \log(1 - \hat{y}^{(i)})$$

The loss function will decrease as  $\hat{y}^{(i)}$  approaches  $\vec{y}_i$  on a graph of  $L$  vs  $f$ .

$\frac{\partial J(\vec{w}, b)}{\partial \vec{w}_j}$  and  $\frac{\partial J(\vec{w}, b)}{\partial b}$  are the same as in linear regression, just the definition of  $f$  has changed.

### 1.4 Softmax regression

Generalization of logistic regression,  $y$  can have more than two possible values.

The most probable value of  $y$  is the value that when given to  $L$  yields the largest loss.

Calculate  $z_i$  with  $\vec{x}$  only consisting of data points that have label  $i$ ?

$n_f$  = num features

$n_y$  = number of possible  $y$  outputs

$W$  is a matrix of dimensions  $n_y \times n_f$ .

$\vec{b}$ ,  $\vec{z}$ ,  $\vec{a}$  are vectors of length  $n_y$ .

$$1 \leq i \leq n_y$$

$$\vec{z}_i = W^{(i)} \cdot \vec{x} + \vec{b}_i$$

$$\vec{a}_i = \frac{e^{\vec{z}_i}}{\sum_{k=1}^{n_y} e^{\vec{z}_k}}$$

$$L(\vec{a}, y) = \begin{cases} -\log \vec{a}_1 & \text{if } y = 1 \\ -\log \vec{a}_2 & \text{if } y = 2 \\ \vdots \\ -\log \vec{a}_n & \text{if } y = n \end{cases} \quad (1)$$

### 1.5 Feature scaling: z-score normalization

After z-score normalization, all features will have a mean of 0 and a standard deviation of 1

$n_f$  = num features

$\vec{\mu}_j$  = mean of all values for feature  $j$  (length  $n_f$ )

$\vec{\sigma}_j$  = standard deviation of feature  $j$  (length  $n_f$ )

$$X_j^{(i)} = \frac{X_j^{(i)} - \vec{\mu}_j}{\vec{\sigma}_j}$$

$$\vec{\mu}_j = \frac{1}{m} \sum_{i=0}^{m-1} X_j^{(i)}$$

$$\vec{\sigma}_j^2 = \frac{1}{m} \sum_{i=0}^{m-1} (X_j^{(i)} - \vec{\mu}_j)^2$$

## 1.6 Over / underfitting

Underfit / high bias: does not fit training set well ( $wx + b$  fit onto data points with  $x + x^2$  shape)

Overfit / high variance: fits training set extremely well but does not generalize well ( $w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$  fit onto training set of shape  $x + x^2$  can have zero cost but predicts values outside the training set inaccurately)

Addressing overfitting

- Collect more data
- Select features ("Feature selection")
- Reduce size of parameters ("Regularization")

### 1.6.1 Regularization

Small values of  $w_1, w_2, \dots, w_n, b$  for simpler model, less likely to overfit

Given  $n_f$  features, there is no way to tell which features are important and which features should be penalized, so all features are penalized.

$$J_r(\vec{w}, b) = J(\vec{w}, b) + \frac{\lambda}{2m} \sum_{j=1}^{n_f} \vec{w}_j^2$$

Can include  $b$  by adding  $\frac{\lambda}{2m}b^2$  to  $J_r$  but typically doesn't make a large difference.

The extra term in  $J_r$  is called the regularization term.

Effectively,  $\lambda \propto \frac{1}{w}$ . When trying to minimize cost, either the error term or the regularization term must decrease. The larger the lambda, the more the regularization term should decrease to minimize cost, decreasing  $w$  parameters.

**Regularized linear regression**

$$J_r(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^m [(f_{\vec{w},b}(X^{(i)}) - \vec{y}_i)^2] + \frac{\lambda}{2m} \sum_{j=1}^{n_f} \vec{w}_j^2$$

For gradient descent, only  $\frac{\partial J_r}{\partial \vec{w}_j}$  changes ( $b$  is not regularized):

$$\frac{\partial J_r}{\partial \vec{w}_j} = \frac{1}{m} \sum_{i=1}^m [(f_{\vec{w},b}(X^{(i)}) - \vec{y}_i) X_j^{(i)}] + \frac{\lambda}{m} \vec{w}_j$$

**Regularized logistic regression**

$$J_r(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^m L(f_{\vec{w},b}(X^{(i)}), \vec{y}_i) + \frac{\lambda}{2m} \sum_{j=1}^{n_f} \vec{w}_j^2$$

For gradient descent, only  $\frac{\partial J_r}{\partial \vec{w}_j}$  changes ( $b$  is not regularized):

$$\frac{\partial J_r}{\partial \vec{w}_j} = \frac{1}{m} \sum_{i=1}^m [(f_{\vec{w},b}(X^{(i)}) - \vec{y}_i) X_j^{(i)}] + \frac{\lambda}{m} \vec{w}_j$$

## 2 Neural networks

$n_\ell$  = num layers excluding input

$n_n^{[\ell]}$  = n neurons in layer  $\ell$

$n_f$  = num features

$\vec{W}$  is a vector (length  $n_\ell$ ) of matrices of size  $n_n^{[\ell]} \times n_f$

$\vec{x}$  is a vector of outputs from each neuron in previous layer

$\vec{b}$  is a vector (length  $n_\ell$ ), holds a bias value for each layer

$Z$  and  $A$ : vector (length  $n_\ell$ ) of vectors (length  $n_n^{[\ell]}$ )

$g$ : activation function

$1 \leq i \leq n_\ell$

$$Z_i^{[\ell]} = \vec{W}_i^{[\ell]} \vec{x} + \vec{b}^{[\ell]}$$

$$A^{[\ell]} = g(Z^{[\ell]})$$

$a$  (activation) = scalar output of a single neuron

Superscript  $[i]$  is used to notate information relating to the  $i$ th layer in a neural network.

### 2.1 Choosing an activation function

sigmoid:  $g(z) = \frac{1}{1+e^{-z}}$

tanh:  $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

linear:  $g(z) = z$

ReLU:  $g(z) = \max(0, z)$

Leaky ReLU:  $g(z) = \max(\epsilon z, z)$  where  $\epsilon$  is a small nonzero positive value  $< 1$

#### For output layer

Binary classification,  $y = 0$  or  $1$ : use sigmoid

Regression,  $-\infty \leq y \leq \infty$ : use linear activation function

Regression,  $y \geq 0$ : use ReLU

#### For hidden layer

ReLU is most common

### 2.2 Forward propagation

Input  $A^{[\ell-1]}$ , output  $A^{[\ell]}$ , cache  $Z^{[\ell]}$ ,  $W^{[\ell]}$ ,  $\vec{b}^{[\ell]}$

$$Z^{[\ell]} = W^{[\ell]} A^{[\ell-1]} + \vec{b}^{[\ell]}$$

$$A^{[\ell]} = g^{[\ell]}(Z^{[\ell]})$$

Up to  $A^{[n_\ell]}$ , in which case  $\hat{y} = A_0^{[n_\ell]}$  assuming output layer has one unit

### 2.3 Back propagation

Input  $da^{[\ell]}$ , output  $da^{[\ell-1]}$ ,  $dW^{[\ell-1]}$ ,  $d\vec{b}^{[\ell-1]}$

$$dZ^{[\ell]} = dA^{[\ell]} \cdot \frac{dg^{[\ell]}(Z^{[\ell]})}{dZ^{[\ell]}}$$

$$dW^{[\ell]} = \frac{1}{m} dZ^{[\ell]} \cdot A^{[\ell-1]T}$$

$$d\vec{b}^{[\ell]} = \frac{1}{m} \sum_i dZ_i^{[\ell]}$$

$$dA^{[\ell-1]} = W^{[\ell]T} \cdot dZ^{[\ell]}$$