Machine Learning

Contents

1 Regression

1.1 Linear regression

1.1.1 Squared error cost function

Measures how well line fits training data

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

m = num of training examples
$y^{(i)}$ = training example
$\hat{y}^{(i)} = wx^{(i)} + b$
$\frac{1}{m}$ 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

α = learning rate
Must simultaneously update w and b

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

1.2 Multiple linear regression

x is a list of lists in multiple linear regression. Notation for accessing by row and column is $x_{col}^{(row)}$
n = number of features
Sum of predictions of all features is the prediction of multiple linear reg

$$\begin{aligned}
\vec{w} &= [w_1, w_2, w_3, \cdots, w_n] \\
\vec{x} &= [x_1, x_2, x_3, \cdots, x_n] \\
f_{\vec{w},b}(\vec{x}) &= \vec{w}\cdot\vec{x} + b
\end{aligned}$$

Gradient descent

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

Cost function and its partial derivatives

$$\begin{aligned}
J(\vec{w}, b) &= \frac{1}{2m}\sum_{i=0}^{m-1}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2 \\
\frac{\partial}{\partial w_j}J(\vec{w}, b) &= \frac{1}{m}\sum_{i=0}^{m-1}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})x_j^{(i)} \\
\frac{\partial}{\partial b}J(\vec{w}, b) &= \frac{1}{m}\sum_{i=0}^{m-1}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})
\end{aligned}$$

1.3 Logistic regression

Sigmoid function

$$g(z) = \frac{1}{1+e^{-z}}$$
$$0 < g(z) < 1$$

From sigmoid function to logistic regression formula

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

The output of f can be interpreted as the "probability" that class is 1.
ex. $f_{\vec{w},b}(\vec{x}) = 0.7$ means there is a 70% chance y is true
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

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m} L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$$
$$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w},b}(\vec{x}^{(i)}) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$

Simplified form

$$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = -y^{(i)} \log(f_{\vec{w},b}(\vec{x}^{(i)})) - (1 - y^{(i)}) \log(1 - f_{\vec{w},b}(\vec{x}^{(i)}))$$

The loss function will decrease as f approaches $y^{(i)}$ on a graph of L vs f.
$\frac{\partial J(\vec{w},b)}{\partial w_j}$ 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 now a matrix of dimensions $n \times n_f$.
$\vec{b}$ is now a vector of length n_y.

$$1 \leq i \leq n_y$$
$$z_i = \vec{W}_i \cdot \vec{x} + b_i$$
$$a_i = \frac{e^{z_i}}{\sum_{k=1}^{n_y} e^{z_k}}$$

$$L(\vec{a}, y) = \begin{cases} -\log a_1 \text{ if } y = 1 \\ -\log a_2 \text{ if } y = 2 \\ \vdots \\ -\log a_n \text{ if } y = n \end{cases} \tag{1}$$

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
μ_j = mean of all values for feature j
σ_j = standard deviation of feature j

$$x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j}$$
$$\mu_j = \frac{1}{m}\sum_{i=0}^{m-1} x_j^{(i)}$$
$$\sigma_j^2 = \frac{1}{m}\sum_{i=0}^{m-1}(x_j^{(i)} - \mu_j)^2$$

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 ($w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$ 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 $w_1, w_2, \cdots, w_n, b$ for simpler model, less likely to overfit

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

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

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

$$J_r(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^{m} \left[(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2 \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2$$

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

$$\frac{\partial J_r}{\partial w_j} = \frac{1}{m} \sum_{i=1}^{m} \left[(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} \right] + \frac{\lambda}{m} w_j$$

Regularized logistic regression

$$J_r(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^{m} L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) + \frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2$$

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

$$\frac{\partial J_r}{\partial w_j} = \frac{1}{m} \sum_{i=1}^{m} \left[(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} \right] + \frac{\lambda}{m} w_j$$

2 Neural networks

a (activation) = scalar output of a single neuron
Superscript $[i]$ is used to notate information relating to the ith layer in a neural network.
Activation functions are functions a neuron uses to output a value, more in section 2.2.
ReLU activation function: $g(z) = \max(0, z)$

2.1 Choosing an activation function

For output layer

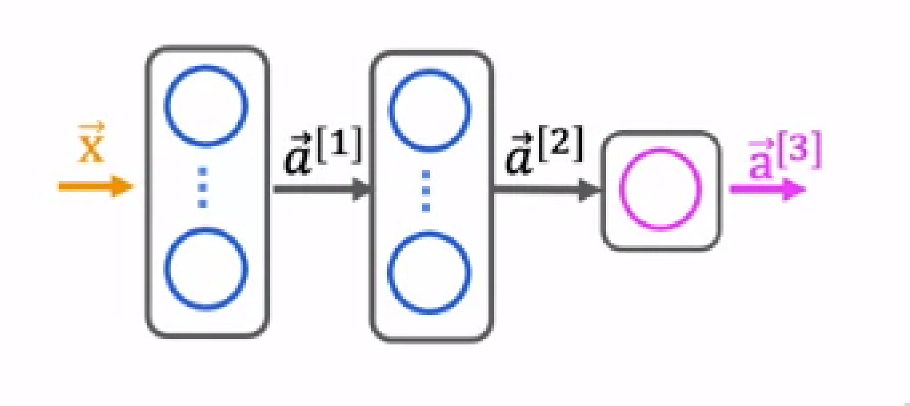
Binary classification, y = 0/1: use sigmoid
Regression, y = +/-: use linear activation function
Regression, y = 0/+: use ReLU

For hidden layer

ReLU is most common

2.2 Layer types

Dense layer



Activation value of unit (neuron) j in layer ℓ

$$a_j^{[\ell]} = g(\vec{w}_j^{[\ell]} \cdot \vec{a}^{[\ell-1]} + b_j^{[\ell]})$$

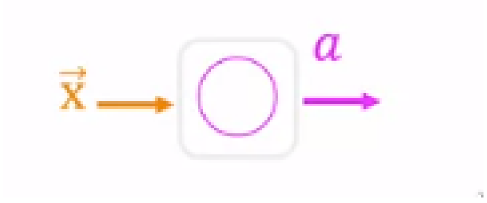
Input layer is $\ell = 0$.

Convolutional layer

Each neuron only looks at a part of the previous layer's output.
May have faster computation, and needs less training data (less prone to overfitting)

2.3 Computation graph

Computes derivatives of neural networks automatically
Assume neural network shown below:

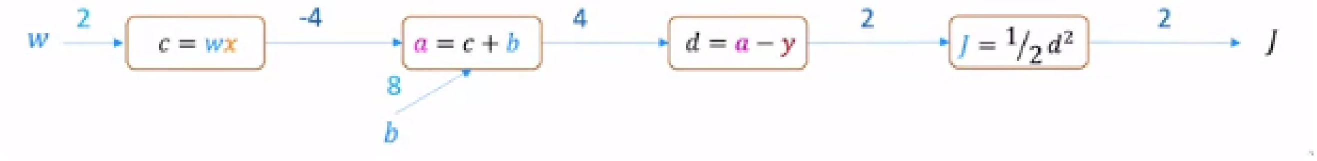


where:

$$a = g(z) = z$$
$$z = wx + b$$
$$J(w, b) = \frac{1}{2}(a - y)^2$$

Variables: $w = 2$, $b = 8$, $x = -2$, $y = 2$

Use forward propagation to calculate cost function



Computing $\frac{\partial J}{\partial w}$ and $\frac{\partial J}{\partial b}$ will require back propagation.
Start from last node in the graph, and determine how J changes if d changes by some small value ϵ.
If $\epsilon = 0.001$, $\delta J =$