# Linear Regression Math

Sasmit Datta

## Introduction

Linear regression finds the best fit line for a data set which can be modelled linearly.

$$h_w(x) = b + w_1x_1 + w_2x_2 + \ldots + w_nx_n$$

where, $n$ is the total number of features. Let's assume $b$ as $w_0$.

$$h_w(x) = w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n$$

We can compress this notation.

$$h_w(x) = \sum_{j=0}^{n} w_jx_j$$

where, $x_0 = 1$. Now let's vectorise this notation.

$$h_w(x) = \sum_{i=0}^{n} w_ix_i = \mathbf{w}^\intercal\mathbf{x}$$

where,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \ \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

$\mathbf{w}$ is the vector we want to learn. For that we define a cost function which computes the error of our model ($\mathbf{w}^\intercal\mathbf{x}$) with respect to $y$ value, where $y$ is the ground-truth label. We define the **loss function** as:

$$J(\mathbf{w}) = \frac{1}{2}(h_w(\mathbf{x^{(i)}}) - y^{(i)})^2 = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(\mathbf{w}^\intercal\mathbf{x^{(i)}} - y^{(i)})^2$$

where, $m$ is the total number of training examples.

## Gradient Descent

Our goal is to choose a $\mathbf{w}$ so as to minimise $J(\mathbf{w})$. We can find this by the gradient descent algorithm. It starts with some initial $w_j$ ($j'th$ element of $\mathbf{w}$ vector), and repeatedly performs an update,

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

until $J(\mathbf{w})$ converges.

Here, $\alpha$ is the **learning rate** which tells the algorithm how much big of a step it should take towards the minimum. This is a **hyper-parameter** (a parameter set by the human) - an important one as it pops up almost everywhere, even in deep learning.

Now let's calculate the partial derivative given above.

$$
\begin{aligned}
\frac{\partial}{\partial w_j} J(\mathbf{w}) &= \frac{\partial}{\partial w_j} \frac{1}{2m} \sum_{i=1}^{m} (h_w(x^{(i)}) - y^{(i)})^2 \\
&= \frac{1}{2m} \sum_{i=1}^{m} \frac{\partial}{\partial w_j} (h_w(x^{(i)}) - y^{(i)})^2 \\
&= \frac{1}{2m} \sum_{i=1}^{m} 2 \cdot (h_w(x^{(i)}) - y^{(i)}) \frac{\partial}{\partial w_j} h_w(x^{(i)}) \\
&= \frac{1}{m} \sum_{i=1}^{m} (h_w(x^{(i)}) - y^{(i)}) \frac{\partial}{\partial w_j} (\sum_{k=0}^{n} w_k x_k^{(i)}) \\
&= \frac{1}{m} \sum_{i=1}^{m} (h_w(x^{(i)}) - y^{(i)}) \frac{\partial}{\partial w_j} (w_1 x_1^{(i)} + w_2 x_2^{(i)} \ldots + w_j x_j^{(i)} + \ldots w_n x_n^{(i)}) \\
&= \frac{1}{m} \sum_{i=1}^{m} (h_w(x^{(i)}) - y^{(i)}) x_j^{(i)}
\end{aligned}
$$

Hence, the final update rule will be (for all training examples):

$$
w_j := w_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_w(x^{(i)}) - y^{(i)}) x_j^{(i)}
$$

And this will be done for every feature $j$.

We have an update rule for the weights. We need one for the bias $b$ as well. Earlier we considered $x_0$ to $1$ and $w_0$ to be b. Therefore we can conclude:

$$
b := b - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_w(x^{(i)}) - y^{(i)})
$$

# Matrix Derivatives

Before we jump into the vectorised notation of this equation let's delve a bit into the derivatives of a matrix with respect to a **scaler** real number.

For function $f : \mathbb{R}^{n \times d} \to \mathbb{R}$, mapping from a $n$-by-$d$ matrices to a real number, we define the derivative of $f$ with respect to matrix $\mathbf{A}$ as:

$$
\nabla_{\mathbf{A}} f = \begin{bmatrix}
\frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1d}} \\
\vdots & \ddots & \vdots \\
\frac{\partial f}{\partial A_{n1}} & \cdots & \frac{\partial f}{\partial A_{nd}}
\end{bmatrix}
$$

Thus the gradient, $\nabla_{\mathbf{A}} f$ is in itself a $n$-by-$d$ matrix where,

$$
(\nabla_{\mathbf{A}} f)_{ij} = \frac{\partial f}{\partial A_{ij}}
$$

### Example:

Let's suppose there is a matrix $\mathbf{A}$:

$$
\mathbf{A} = \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
$$

We define a function $f : \mathbb{R}^{2 \times 2} \to \mathbb{R}$,

$$
\begin{aligned}
f(\mathbf{A}) &= \frac{1}{m} sum(\mathbf{A}) \\
&= \frac{1}{m}(A_{11} + A_{12} + A_{21} + A_{22})
\end{aligned}
$$

∴ It's derivative:

$$
\nabla_{\mathbf{A}} f = \begin{bmatrix} \frac{1}{m} & \frac{1}{m} \\ \frac{1}{m} & \frac{1}{m} \end{bmatrix}
$$

# Vectorisation

Before we delve any deeper, the list of formulas derived in this section is more inclined towards the code. For example we are taking each example in $\mathbf{X}$ as a row vector and not a column since that's how we usually represent our data in a numpy array.

Given a training set, we can define a matrix $\mathbf{X}$ with shape: $m \times n$ where, $m$ is the number of training examples and $n$ is the number of features:

$$
\mathbf{X} = \begin{bmatrix} - & - & \mathbf{x}^{(1)\intercal} & - & - \\ - & - & \mathbf{x}^{(2)\intercal} & - & - \\ & & \vdots & & \\ - & - & \mathbf{x}^{(m)\intercal} & - & - \end{bmatrix}
$$

We have a weight vector with shape: $n \times 1$.

$$
\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}
$$

Also a $\mathbf{y}$ vector with the ground truth labels with shape: $m \times 1$.

$$
\mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}
$$

Finally we have a bias term:

$$
b
$$

Hence we can write the predicted $\mathbf{y}$-values as:

$$
\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b
$$

$\hat{\mathbf{y}}$ has a shape: $m \times 1$.

Therefore our final cost function will be:

$$
\frac{1}{2m} sum((\mathbf{X}\mathbf{w} + b - \mathbf{y})^2)
$$

Let's break this equation down into smaller pieces so we have an easier time computing the derivative.

$$
\mathbf{z} = \mathbf{X}\mathbf{w} + b - \mathbf{y}
$$

$$
\mathbf{s} = \mathbf{z}^{\mathbf{2}}
$$

$$\mathbf{L} = \frac{1}{2m} sum(s)$$

The final derivative we want to calculate is:

$$\nabla_{\mathbf{w}} \mathbf{L} = \frac{\partial \mathbf{L}}{\partial \mathbf{w}} \text{ and } \nabla_{b} \mathbf{L} = \frac{\partial \mathbf{L}}{\partial b}$$

But first let's take first calculate:

$$\nabla_{\mathbf{z}} \mathbf{L} = \frac{\partial \mathbf{L}}{\partial \mathbf{z}}$$

Therefore:

$$\frac{\partial \mathbf{L}}{\partial \mathbf{z}} = \frac{\partial \mathbf{L}}{\partial \mathbf{s}} \frac{\partial \mathbf{s}}{\partial \mathbf{z}} = \frac{1}{2m} \cdot 2\mathbf{z} = \frac{\mathbf{z}}{m}$$

$\partial \mathbf{L} / \partial \mathbf{z}$ is easy to calculate since its just a series of simple element-wise operations. We have to work our way through $\mathbf{X}\mathbf{w}$ which is a matrix multiplication to get the final product: $\partial \mathbf{L} / \partial \mathbf{w}$.

## Derivative of Matrix Multiplication

Let's consider a matrix multiplication:

$$\mathbf{C} = \mathbf{A} \times \mathbf{B}$$

Which we can expand to:

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

If we work out the equations for each value in $\mathbf{C}$ we get:

$$\begin{aligned} c_{11} &= a_{11}b_{11} + a_{12}b_{21} \\ c_{12} &= a_{11}b_{12} + a_{12}b_{22} \\ c_{21} &= a_{21}b_{11} + a_{22}b_{21} \\ c_{22} &= a_{21}b_{12} + a_{22}b_{22} \end{aligned}$$

Now let's assume $\mathbf{C} \in \mathbb{R}^{2\times2}$ converges into a loss $\mathbf{L} \in \mathbb{R}$. Therefore there exists a $\partial \mathbf{L} / \partial \mathbf{C} \in \mathbb{R}^{2\times2}$.

We have to find a relationship between $\partial \mathbf{L} / \partial \mathbf{C}, \partial \mathbf{L} / \partial \mathbf{A}$ and $\mathbf{B}$

$$\begin{aligned} \frac{\partial \mathbf{L}}{\partial a_{11}} &= \frac{\partial \mathbf{L}}{\partial c_{11}} b_{11} + \frac{\partial \mathbf{L}}{\partial c_{12}} b_{12} \\ \frac{\partial \mathbf{L}}{\partial a_{12}} &= \frac{\partial \mathbf{L}}{\partial c_{11}} b_{21} + \frac{\partial \mathbf{L}}{\partial c_{12}} b_{22} \\ \frac{\partial \mathbf{L}}{\partial a_{21}} &= \frac{\partial \mathbf{L}}{\partial c_{21}} b_{11} + \frac{\partial \mathbf{L}}{\partial c_{22}} b_{12} \\ \frac{\partial \mathbf{L}}{\partial a_{22}} &= \frac{\partial \mathbf{L}}{\partial c_{21}} b_{21} + \frac{\partial \mathbf{L}}{\partial c_{22}} b_{22} \end{aligned}$$

We can rearrange this equation as:

$$\begin{bmatrix} \frac{\partial \mathbf{L}}{\partial a_{11}} & \frac{\partial \mathbf{L}}{\partial a_{12}} \\ \frac{\partial \mathbf{L}}{\partial a_{21}} & \frac{\partial \mathbf{L}}{\partial a_{22}} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathbf{L}}{\partial c_{11}} & \frac{\partial \mathbf{L}}{\partial c_{12}} \\ \frac{\partial \mathbf{L}}{\partial c_{21}} & \frac{\partial \mathbf{L}}{\partial c_{22}} \end{bmatrix} \begin{bmatrix} b_{11} & b_{21} \\ b_{12} & b_{22} \end{bmatrix}$$

$$\frac{\partial \mathbf{L}}{\partial \mathbf{A}} = \frac{\partial \mathbf{L}}{\partial \mathbf{C}} \mathbf{B}^{\intercal}$$

Similarly we can find $\partial \mathbf{L} / \partial \mathbf{B}$ by the same method which turns out to be:

$$\frac{\partial \mathbf{L}}{\partial \mathbf{B}} = \mathbf{A}^\intercal \frac{\partial \mathbf{L}}{\partial \mathbf{C}}$$

## Finding $\partial \mathbf{L}/\partial \mathbf{w}$

From above, we have:

$$\frac{\partial \mathbf{L}}{\partial \mathbf{z}} = \frac{1}{m} \cdot \mathbf{z}$$

$$\mathbf{z} = \mathbf{X}\mathbf{w} + b - \mathbf{y}$$

There we finally have $\partial \mathbf{L}/\partial \mathbf{w}$:

$$\frac{\partial \mathbf{L}}{\partial \mathbf{w}} = \mathbf{X}^\intercal \frac{\partial \mathbf{L}}{\partial \mathbf{z}}$$

$$\frac{\partial \mathbf{L}}{\partial \mathbf{w}} = \frac{1}{m} \mathbf{X}^\intercal \mathbf{z}$$

## Finding $\partial \mathbf{L}/\partial b$

$\partial \mathbf{L}/\partial b$ is relatively easy to find since its just a scaler ($1 \times 1$ matrix) which is broadcasted to perform calculations.

First let's see what $\partial \mathbf{z}/\partial b$ is:

$$\frac{\partial \mathbf{z}}{\partial b} = 1$$

Now $\mathbf{z}$ is a $m \times 1$ matrix. So is $\partial \mathbf{L}/\partial \mathbf{z}$. So we have to use the chain rule (like before) to get a value of $\partial \mathbf{z}/\partial b$ which should have a shape of $1 \times 1$.

Since through broadcasting, we can say that $b$ was repeated $m$ times to get added to $\mathbf{X}\mathbf{w}$ to get $\mathbf{z}$. So by simple calculus, we can conclude that:

$$\frac{\partial \mathbf{L}}{\partial b} = \frac{\partial \mathbf{z}}{\partial b} \cdot sum\left(\frac{\partial \mathbf{L}}{\partial \mathbf{z}}\right)$$

$$\frac{\partial \mathbf{L}}{\partial b} = \frac{1}{m} sum(\mathbf{z})$$

## Update

Therefore the updates we perform to the parameters $(\mathbf{w}, b)$ will be:

$$\mathbf{w} := \mathbf{w} - \alpha \frac{\partial \mathbf{L}}{\partial \mathbf{w}}$$

$$b := b - \alpha \frac{\partial \mathbf{L}}{\partial b}$$

## Data Normalisation

For training, we have to normalise our train data for stable training. For all the features in $\mathbf{X}$, we find the their mean and standard deviations, and get a mean vector, $\mathbf{x_{mean}}$, and a stand deviation vector, $\mathbf{x_{std}}$. Therefore, $\mathbf{X_{norm}}$ is:

$$\mathbf{X_{norm}} = \frac{\mathbf{X} - \mathbf{x_{mean}}}{\mathbf{x_{std}}}$$

Similarly, we can get the normalised $\mathbf{y}$ as:

$$\mathbf{y_{norm}} = \frac{\mathbf{y} - y_{mean}}{y_{std}}$$

where, $y_{mean}$ and $y_{std}$ are the mean and standard deviations of $\mathbf{y}$ respectively.

# Testing Our Model

We can't use training dataset to test our model as it is something the model has learned on. We'll have to use a separate dataset which the model hasn't trained on to get the actual loss. Let's define our test set:

$$\mathbf{X_{test}} = \begin{bmatrix} - - \mathbf{x_{test}}^{(1)\intercal} - - \\ - - \mathbf{x_{test}}^{(2)\intercal} - - \\ \vdots \\ - - \mathbf{x_{test}}^{(m)\intercal} - - \end{bmatrix}, \mathbf{y_{test}} = \begin{bmatrix} y_{test}^{(1)} \\ y_{test}^{(2)} \\ \vdots \\ y_{test}^{(m)} \end{bmatrix}$$

Now we'll have to normalise our dataset. We'll normalise $\mathbf{X_{test}}$ using $\mathbf{x_{mean}}$ and $\mathbf{x_{std}}$ from our train set but for $\mathbf{y_{test}}$, we'll normalise it using the mean and stand deviations of $\mathbf{y_{test}}$ itself. This is a better measure for how robust the model is for the real world.

Similarly, like before we can calculate the loss for our test set:

$$\frac{1}{2m} sum((\mathbf{X_{test,norm}}\mathbf{w} + b - \mathbf{y_{test,norm}})^2)$$

# Inference

Our model can only input and output normalised data. So when we use our model for inferencing, we have to un-normalise the output we get from the model and we'll do that using $y_{mean}$ and $y_{std}$ from the train set.

So first we will normalise the input data with $\mathbf{x_{mean}}$ and $\mathbf{x_{std}}$ from the training set and get $\mathbf{x_{inference}}$. We will now calculate $y_{norm}$.

$$y_{norm} = \mathbf{x_{inference}}\mathbf{w} + b$$

Un-normalisation:

$$y = y_{norm}y_{std} + y_{mean}$$

$$y = y_{std}(\mathbf{x_{inference}}\mathbf{w} + b) + y_{mean}$$