# Linear Regression Worked Examples

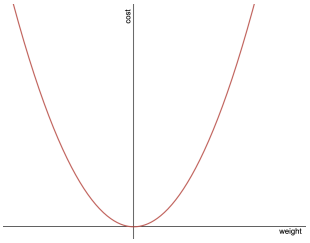
Reading:
Linear Regression Materials
Patrick Loeber video / code

Linear regression models are used to predict a continuous value. They find the best-fitting line (in the 1-feature case), plane (in the 2-feature case), etc. **through the training data**.

As with perceptrons, the features and gold values are numeric values. But now the gold values are continuous (float) values (remember that we're predicting continuous values).

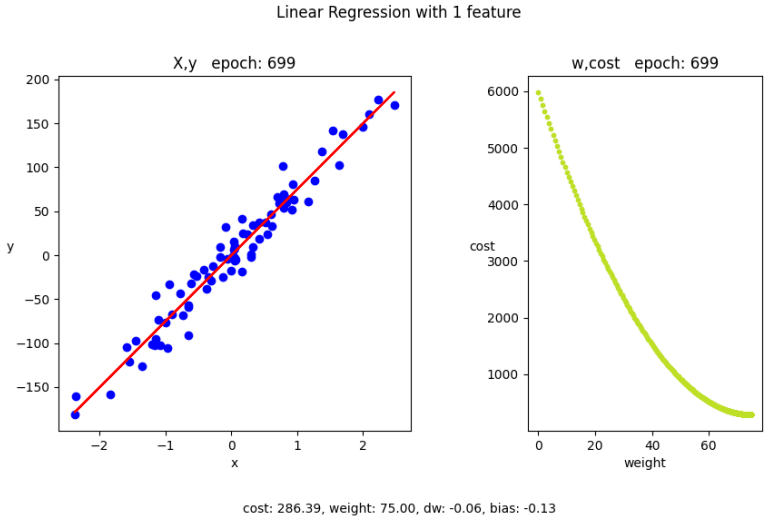
Linear regression differs from perceptrons in that we now have a **cost function**, and we use **gradient descent** to minimize the cost.

We use Mean Squared Error (**MSE**) as the cost function (also called **loss** function). As the name implies, it calculates the average of the squared errors, where the error is the difference between y_pred and the gold value y, averaged over all training samples.




The plot on the right shows the MSE on the y-axis and the weight on the x-axis. We use just one weight to demonstrate, but normally there are many weights (one for each feature in the data). The goal is to find the weight where the cost is near 0. Using the current weights and bias, we calculate the cost (MSE, a scalar value). Find the point on the curve. The **gradient** is the slope of the curve at this point, and it tells us if we need to increase or decrease the weight so that the weight is closer to the lowest point on the curve (where the cost is 0). But how do we calculate the gradient (slope) if we only have one point? That's done with **partial derivatives**.

The **learning rate** controls how much the weights and bias are adjusted at each epoch.




The plot on the left shows the line through a 1-feature training dataset at training epoch 700. The feature values are plotted on the x-axis, and the y (gold) values are on the y-axis. The goal of training is to calculate the line (more precisely, the gradient (slope) of the line, and the bias (the point where the line intersects with the y-axis)). When we use the model for **inference** (to predict y based on x), we use the linear formula $y = xw + b$.

On the right, the weight is plotted against the cost at each epoch. Recall that for this training data, we have only one feature (and therefore one weight). The derivative of the cost function with respect to (wrt) w $\frac{\partial f}{\partial w}$ is the slope of the weight on the cost function curve. Since the slope is negative, the weight is increased at each epoch. Note that $\frac{\partial f}{\partial w}$ at epoch 699 is approaching 0, which means that the gradient descent is nearly finished.

## Shapes - Overview

A training matrix **X** of shape (100,30) has 100 training samples, each with 30 features.

The weights vector represents weights of features in the training data, and has shape (num_features,1)

Bias is a float.

The **y** vector (gold values) has 1 value for each sample. Its shape is (num_samples, 1)

The gradient is a vector of derivatives, one for each feature weight. Its shape is (num_features,1)

# 1 Prediction

Linear regression prediction is just a matter of applying the weights and bias to the sample(s). That is, calculate the value of the linear_output $X \cdot w + b$.

### 1.1 Prediction Exercise 1

A linear regression model was trained on a 3-feature dataset. The final weights and bias after training are:

$$\mathbf{w} = \begin{bmatrix} 4 \\ -2 \\ 3 \end{bmatrix} \qquad bias = -1$$

Calculate the predicted value of the following test data, which contains one sample.

$$\mathbf{X\_test} = \begin{bmatrix} 2 & -1 & -2 \end{bmatrix}$$

**Solution** Apply the predict() function, which simply calculates the linear output:

$$\begin{aligned} linear\_output &= \mathbf{X} \cdot \mathbf{w} + b \\ &= \begin{bmatrix} 2 & -1 & -2 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ -2 \\ 3 \end{bmatrix} + -1 \\ &= (2 * 4) + (-1 * -2) + (-2 * 3) - 1 \\ &= 8 + 2 - 6 - 1 \\ &= 3 \end{aligned}$$

y_pred = 3

### 1.2 Prediction Exercise 2

Using the same weights and bias as in Exercise 1, calculate the predicted value of the following test data, which contains two samples.

$$\mathbf{X\_test} = \begin{bmatrix} 2 & -1 & -3 \\ -1 & 2 & 1 \end{bmatrix}$$

**Solution** Apply the predict() function, which simply calculates the linear output:

$$
\begin{aligned}
linear\_output &= \mathbf{X} \cdot \mathbf{w} + b \\
&= \begin{bmatrix} 2 & -1 & -3 \\ -1 & 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ -2 \\ 3 \end{bmatrix} + -1 \\
&= \begin{bmatrix} (2*4) + (-1*-2) + (-3*3) \\ (-1*4) + (2*-2) + (1*3) \end{bmatrix} - 1 \\
&= \begin{bmatrix} 8+2-9 \\ -4-4+3 \end{bmatrix} - 1 \\
&= \begin{bmatrix} 0 \\ -6 \end{bmatrix}
\end{aligned}
$$

predicted value for sample1 $= 0$, sample2 $= -6$

# 2 Training

During training, feature weights and bias are updated to "fit" the training data. The optimal weights and bias usually can not be reached by processing the training data only once. The training data is processed over many iterations (called epochs). Unlike the perceptron model, which iterates over each training sample individually, linear regression models process the entire training data at once using dot products.

## 2.1 Training Exercise

Suppose you are training a linear regression model on the following 2-feature training data, using a learning rate of .1:

$$\mathbf{X\_train} = \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 2 & 1 \end{bmatrix} \qquad \mathbf{y\_train} = \begin{bmatrix} -1 \\ 1 \\ 2 \end{bmatrix}$$

At epoch x, the weights and bias are:

$$\mathbf{w} = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \qquad bias = -3$$

Calculate the weights and bias at epoch x+1 (i.e. after updating the weights and bias once).

**Solution** Apply the fit() method for 1 iteration.

Make the linear prediction based on the current weights and bias:

$$
\begin{aligned}
\boldsymbol{y\_pred} &= \boldsymbol{X} \cdot \boldsymbol{w} + b \\
&= \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 3 \end{bmatrix} + -3 = \begin{bmatrix} (1*2) + (2*3) \\ (2*2) + (3*3) \\ (2*2) + (1*3) \end{bmatrix} - 3 = \begin{bmatrix} 8 \\ 13 \\ 7 \end{bmatrix} - 3 = \begin{bmatrix} 5 \\ 10 \\ 4 \end{bmatrix}
\end{aligned}
$$

Find the delta between the linear predictions and the gold values.

**The Patrick Loeber code uses (y_pred - y) instead of (y - y_pred) to avoid multiplication by -1 when calculating the gradients in the next step**.

$$\boldsymbol{y\_pred} - \boldsymbol{y} = \begin{bmatrix} 5 \\ 10 \\ 4 \end{bmatrix} - \begin{bmatrix} -1 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 6 \\ 9 \\ 2 \end{bmatrix}$$

The gradient vector is a vector of averaged partial derivatives $\frac{\partial f}{\partial w}$ (averaged over all samples in the training data). Each element in the gradient vector describes the slope of the cost function for one of the feature weights. We use the mean squared error (MSE) as the cost function. The partial derivative with respect to $w$ of the cost function $\frac{\partial f}{\partial w} = -x_i(y_i - y_{ipred})$. We remove the multiplication by -1 and use the delta y calculated in the previous step, resulting in the following formula for the partial derivative of the cost function wrt w:

$\frac{\partial f}{\partial w} = x_i(y_{ipred} - y_i)$ This is the derivative of 1 element in our training matrix.

We want only 1 derivative for each feature weight, so we average the derivatives for each feature:

$\frac{\partial f}{\partial w_i} = \frac{1}{N}\sum_{i=0}^{n} x_i(y_{ipred} - y_i)$

The sum in this formula is a dot product, but we need to transpose the training data so that the calculations are made along the feature axis:

$$\begin{aligned}\frac{\partial f}{\partial w} &= \frac{1}{n\_samples} * \boldsymbol{X}^T \cdot (y\_pred - y) \\ &= (1/3) * \begin{bmatrix}1 & 2 & 2\\ 2 & 3 & 1\end{bmatrix} \cdot \begin{bmatrix}6\\9\\2\end{bmatrix} \\ &= (1/3) * \begin{bmatrix}28\\41\end{bmatrix} \\ &\approx \begin{bmatrix}9.33\\13.67\end{bmatrix}\end{aligned}$$

Now calculate the gradient with respect to $b$. The partial derivative wrt $b$ of the cost function is $-(y_i - y_{ipred}) = (y_{ipred} - y_i)$, so for the bias we just need the average of the y deltas.

$$\begin{aligned}\frac{\partial f}{\partial b} &= \frac{1}{n\_samples} * \Sigma(y\_pred - y) \\ &= (1/3) * (6 + 9 + 2) \\ &\approx 5.67\end{aligned}$$

Finally, update the weights and bias:

$$\begin{aligned}\boldsymbol{w} &= \boldsymbol{w} - lr * \frac{\partial f}{\partial w} \\ &= \begin{bmatrix}2\\3\end{bmatrix} - .1 * \begin{bmatrix}9.33\\13.67\end{bmatrix} \\ &= \begin{bmatrix}2\\3\end{bmatrix} - \begin{bmatrix}.933\\1.367\end{bmatrix} \\ &\approx \begin{bmatrix}1.067\\1.633\end{bmatrix}\end{aligned}$$

$$\begin{aligned}bias &= bias - lr * \frac{\partial f}{\partial b} \\ &= -3 - .1 * 5.67 \\ &\approx -3.567\end{aligned}$$