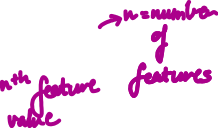
1. **Linear Regression**

A linear model makes a prediction by computing the weighted sum of the input features plus a bias term/intercept term



**Mean Squared Error (MSE)**

Most well suited measure of linear regression model performance is RMSE. Therefore we need to find a theta that minimizes the RMSE. Alternatively, in practice MSE is used.



**The Normal Equation**

There is a closed-form solution (a mathematical way) to compute theta without having to train the model



**Note:** Sklearn’s implementation has a time complexity of O(n^2) where n is the number of features. So, if there are too many features, the time required to compute the parameters increases quadratically.

1. **Gradient Descent**

The general idea is to tweak parameters iteratively to minimize cost function.

* Initialize theta with random numbers (or 0s)
* Change value of theta gradually by taking baby steps at a time, with each step attempting to reduce cost function
* Continue till the algorithm converges to a minimum

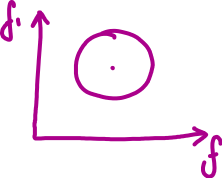
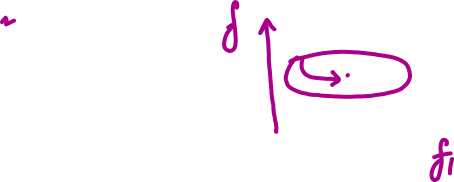
**Learning Rate**

An important parameter is the step size, determined by the learning rate hyperparameter.

Too small a LR means it will take too long to converge. Too large a LR means the algorithm might actually diverge and never find a minimum.

**Problem of differing scales of features**

If feature 1 has values ranging from 0-100 where as feature 2 has values between 0-1, then their scales are different and this may lead to the following problem:



On the left the Gradient Descent algorithm goes straight toward the minimum, thereby reaching it quickly, whereas on the right it first goes in a direction almost orthogonal to the direction of the global minimum, and it ends with a long march down an almost flat valley. It will eventually reach the minimum, but it will take a long time.

We can use Sklearn’s StandardScaler class to do this in code.

**Batch Gradient Descent**