Linear Regression, Training and Loss

1 Linear regression

Definition: Linear regression

A method for finding the straight line or hyperplane that best fits a set of points

$$y = b + w_1 x_1$$

y - the predicted label
b - the bias, sometimes referred to as w_0
w_1 - the weight of feature 1
x_1 - a feature

2 Training and loss

Definition: Training a model

Learning good values for all weights and the bias from labelled examples

Definition: Loss

The penalty for a bad prediction

Definition: Empirical Risk Minimisation

The process of examining many examples and attempting to find a model that minimises loss

2.1 Squared loss

The square of the difference between the label and the prediction

$$(\text{observation} - \text{prediction}(x))^2$$

$$(y - \hat{y})^2$$

2.2 Mean square error

$$MSE = \frac{1}{N} \sum_{(x,y)\in D} (y - \text{predicition}(x))^2$$

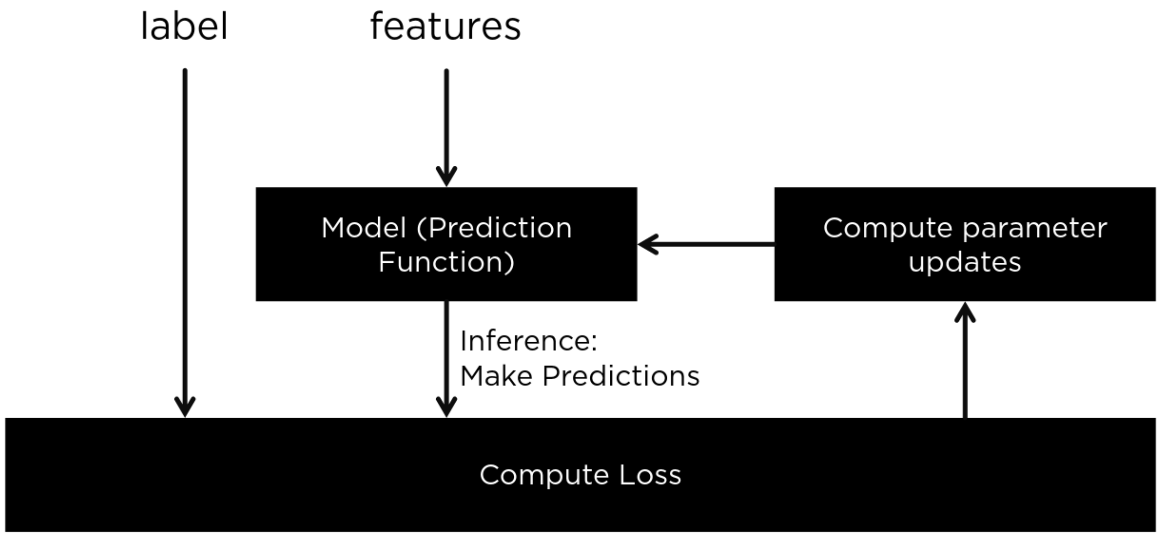
(x,y) is an example where

- x is the set of features used by the model to make predictions
- y is the example's label

prediction(x) is a function of the weights and bias in combination with the set of features x
D is the dataset containing many labelled examples
N is the number of examples in D

3 Reducing loss

- Hyperparameters are the configuration settings used to tune how the model is trained
- Derivative of loss with respect to weights and biases tells us how loss changes for a given example
- So we repeatedly take small steps in the direction that minimises loss, we call these **Gradient steps**



3.1 Weight initialisation

For convex problem, weights can start anywhere forming a graph that looks like x^2

Foreshadowing: not true for neural networks

- More than one minimum
- Strong dependency on initial values

3.2 Efficiency of reducing loss

- Could compute gradient over entire dataset on each step, but this turns out to be unnecessary
- Computing gradient on small data examples works well
- **Stochastic Gradient Descent** - one example at a time
- **Mini-batch Gradient Descent** - batches of 10-1000

3.3 Learning rate

The ideal learning rate in one-dimension is

$$\frac{1}{f(x)''}$$

The ideal learning rate for 2 or more dimension is the inverse of the Hessian (matrix of second partial derivatives)