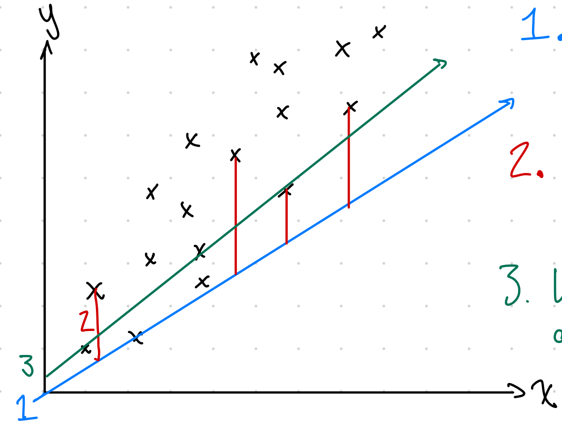
Stochastic Gradient Descent Theory



1. Create some initial guess
   $h_\theta(x) = \theta_1 + \theta_2 x$
   where θ_1, θ_2 are random
2. Measure the error between the prediction and some random batch of data
3. Use the error to update θ_1 & θ_2 to make a better prediction
4. Repeat!

Hypotheses function $h_\theta(x) = \theta_1 + \theta_2 \vec{x}$

Update function:

$$\theta_1 \mathrel{+}= \alpha \cdot (-y_i - h_\theta(x)) / m$$

$$\theta_2 \mathrel{+}= \alpha \cdot -x_i (y_i - h_\theta(x)) / m$$

Where m is the size of the data (# of y points) and α is the learning rate

Learning Rate α: determines the size of the steps taken towards minimizing the loss function during each iteration of training

- The learning rate typically adjusts on a schedule determined in testing

Stochastic Gradient Descent Implementation

```
//learning rate
double learning_rate = 0.000001; α //can be altered
//dacay rates and steps
double decay_rate = 0.9;     //can be altered
int decay_steps = 5;    //can be altered

Random random = new Random();
//initialize slope and intercept as random numbers n, where -0.01 <= n <= 0.01
double slope = random.nextDouble() * 0.02 - 0.01;
double intercept = random.nextDouble() * 0.02 - 0.01;

//define the numver of iterations SGD will run
int num_iterations = 10000000;  //can be altered

//define two variables for gradients
double slope_gradient = 0.0;
double intercept_gradient = 0.0;

for (int i = 0; i < num_iterations; i++) {

    slope_gradient = 0.0;
    intercept_gradient = 0.0;

    int num_data = random.nextInt(data.length);
    for (int j = 0; j < num_data; j++) {
        int random_index = random.nextInt(data.length);
        int x_i = data[random_index][0];
        int y_i = data[random_index][1];

        double predicted_y = slope * x_i + intercept;

        slope_gradient += -x_i * (y_i - predicted_y); -x(y - h_θ(x))
        intercept_gradient += -(y_i - predicted_y);   -(y - h_θ(x))
    }

    //update parameters  - α·((-x(y-h_θ(x)))/ m)
    slope -= learning_rate * (slope_gradient / data.length);
    intercept -= learning_rate * (intercept_gradient / data.length);
                                   - α((-y-h_θ(x))/ m)
    //update the learning rate
    if(i != 0 && decay_steps % i ==0){
        learning_rate *= decay_rate; decay learning rate
    }
}
    System.out.println(slope + "x" + " + " + intercept);
```

More Complicated Stochastic Gradient Descent

The previous formulas assume we only have one single input value x, The implementation takes the square footage of a house, and estimates a price. We know that the price of a house depends on other things, for simplicity, we'll say size, number of bedrooms, and number of bathrooms. We now have x, is a vector! ($\vec{x}$)

$$\vec{x} \begin{bmatrix} \text{Size} \\ \text{\# bedrooms} \\ \text{\# of bathrooms} \end{bmatrix}$$

Notice $\vec{x} \in \mathbb{R}^n$, where n is the number of parameters

We can no longer visually represent our hypothesis $h_\theta(\vec{x})$ as $\vec{x}$ has too many dimensions!

How does changing $\vec{x}$ to a vector affect our algorithm? Lets look at how the variables change

$x \longrightarrow \underset{1 \times n}{\vec{x}} \Longrightarrow \underset{1 \times n}{\vec{x}_i}$ We already knew $\vec{x}$ is a vector! Nice!

$y \longrightarrow y$ No Change!

$\theta_1 \rightarrow \theta_1$ No change!

$h_\theta(x) \rightarrow h_\theta(x)$ No Change!

$\theta_2 \longrightarrow \underset{1 \times n}{\vec{\theta}_2}$ This is the only thing that Changes. Why?

If $\vec{x}$ is $1 \times n$, then, each element in $\vec{x}$ needs some slope in $\vec{\theta}_2$, then, we take $\vec{\theta}_2^T \vec{x}$ to get a constant! Nice!

But how do the formulas change?

More Complicated Stochastic Gradient Descent

Recall original functions:

$$h_\theta(x) = \theta_1 + \theta_2 x$$

$$\theta_1 \mathrel{+}= \alpha \cdot (-y_i - h_\theta(x))/m$$

$$\theta_2 \mathrel{+}= \alpha \cdot -x_i(y_i - h_\theta(x))/m$$

Lets see what this data is now that $\vec{x}$ & $\vec{\theta_2}$ are vectors!

$$\theta_1 \mathrel{+}= \alpha \cdot (-y_i - h_\theta(\vec{x}))/m$$

$$\underset{1\times n}{\vec{\theta_2}} = \alpha \cdot -\underset{1\times n}{\vec{x_i}}(y_i - h_\theta(\vec{x}))/m$$

$$\Rightarrow h_\theta(\vec{x}) = \theta_1 + \underset{n\times 1}{\theta_2^T}\,\underset{1\times n}{\vec{x}}$$

Not too much changes! Nice!