This handout describes simple linear regression as well as the application of gradient descent (and stochastic gradient descent) to fitting a linear regression model.

Setting

We are given data $(x_1, y_1), \ldots, (x_n, y_n)$, where x and y are numerical attributes and we want to explore the relationship between x and y. In particular, we want to model y, the target (aka response) variable, as a function of x, known as a feature (aka predictor variable). We'll write $y \approx h(x)$ to mean y is "approximately modeled as" $h(x)$.

Simple Linear Model

Let's assume h is a simple linear function of x:

$$h_\beta(x) = \beta_0 + \beta_1 x$$

where β_0 is the intercept and β_1 is slope. (Aside: the book uses (α, β) instead of (β_0, β_1)).

The coefficients β_0 and β_1 are *unknown* but we can use the given data to *estimate* them. This is called "fitting" the model.

Fitting the model

Given data, how should we set β_0 and β_1? We want the model to fit the data as closely as possible. There are many different ways to measure *closeness*.

Use a **cost function** to measure how well a particular model fits the data. The "sum of squared errors" cost function is

$$\begin{aligned} J(\beta_0, \beta_1) &= \sum_{i=1}^{n} (y_i - h_\beta(x_i))^2 \\ &= \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_i))^2 \end{aligned}$$

The expression $(y_i - h_\beta(x_i))$ captures the *error* of the model on the i^{th} input. This is also known as a *residual*. The J function computes the sum of *squared* errors. (This is also called residual sum of squares, or RSS.)

▷**Important**: This is a function of the *coefficients* β_0 and β_1. The data points—the x_i's and y_i's—are treated as *fixed*. Each assignment of β_0, β_1 corresponds to a different line through the data. *Goal*: We want to find the β_0, β_1 that *minimizes* the cost.

What does $J(\beta_0, \beta_1)$ look like?

Aside: Why do we want to minimize the sum of squared errors?

- *Maximum Likelihood Estimate* Let's assume $y_i = h_\beta(x_i) + \epsilon_i$ where each ϵ_i is an independent random sample from a Normal distribution with mean of zero. Then y_i is the observed value of a random variable that follows a Normal distribution with a mean of $\beta_0 + \beta_1 x_i$. It turns out that the coefficients that minimize J also maximize the *likelihood* of the data (i.e., the probability of producing the observed data is highest when we set β to minimize J).

- The math works out nicely. (Seriously, "nice" math has practical implications, such as numerical stability under floating point operations.)
- We may not in all cases! Depending on problem setting, a different cost function may be more appropriate.

Gradients

Quick math review: Derivative $\frac{d}{dx}f(x)$ is rate of change of f as a function of x. Gradient is the analogue of a derivative for multi-variable functions. The gradient of function $f(x, y, z)$ is denoted ∇f and is a vector of *partial* derivatives, one for each parameter. Partial derivative $\frac{\partial}{\partial x}f(x, y, z)$ is the rate of change of f as a function of x when you hold y and z fixed.

Gradient of cost function J. The cost is a function of two parameters, β_0 and β_1. Its gradient is a vector of length two: $\nabla J = \left(\frac{\partial}{\partial\beta_0}J, \frac{\partial}{\partial\beta_1}J\right)$

Partial derivative of J with respect to β_1:

$$\begin{aligned}
\frac{\partial}{\partial\beta_1}J(\beta_0,\beta_1) &= \frac{\partial}{\partial\beta_1}\sum_{i=1}^{n}\left(y_i - h_\beta(x_i)\right)^2 \\
&= \sum_{i=1}^{n}\frac{\partial}{\partial\beta_1}\left(y_i - h_\beta(x_i)\right)^2 && \text{(sum rule)} \\
&= \sum_{i=1}^{n}2\left(y_i - h_\beta(x_i)\right)\frac{\partial}{\partial\beta_1}\left(-h_\beta(x_i)\right) && \text{(chain rule)} \\
&= -\sum_{i=1}^{n}2\left(y_i - h_\beta(x_i)\right)\frac{\partial}{\partial\beta_1}(\beta_0 + \beta_1 x_i) && \text{(definition of } h_\beta(x)\text{)} \\
&= -\sum_{i=1}^{n}2\left(y_i - h_\beta(x_i)\right)x_i && \text{(because } \frac{\partial}{\partial\beta_1}(\beta_0+\beta_1 x) = x_i\text{)}
\end{aligned}$$

Using a similar derivation, you can show $\frac{\partial}{\partial\beta_0}J(\beta_0,\beta_1) = -\sum_{i=1}^{n}2\left(y_i - h_\beta(x_i)\right)$.

Intuition The magnitude of the update depends on the error terms $(y_i - h_\beta(x_i))$. Points where the current line is a bad fit have a bigger effect on the gradient.

Gradient points in direction of *increasing* J. To minimize J we want to head in the *opposite* direction.

Gradient Descent

Gradient descent is Algorithm 1. Differences from book: here, the step size is fixed; the book tries several step sizes and uses whichever one lowers cost the most.

Stochastic gradient descent

Stochastic gradient descent is Algorithm 2.

Algorithm 1 Gradient descent for simple linear regression

procedure GRADIENTDESCENT(cost function J, step size η, tolerance t)
  Initialize β_0, β_1 arbitrarily.
  cost $\leftarrow J(\beta_0, \beta_1)$
  repeat
    ▷ Use gradient to update coefficients:
    $\beta_0 \leftarrow \beta_0 - \eta \cdot \frac{\partial}{\partial \beta_0} J(\beta_0, \beta_1)$ ▷ Take step in opposite direction of gradient
    $\beta_1 \leftarrow \beta_1 - \eta \cdot \frac{\partial}{\partial \beta_1} J(\beta_0, \beta_1)$
    oldCost $\leftarrow$ cost
    cost $\leftarrow J(\beta_0, \beta_1)$
  until $|\text{cost} - \text{oldCost}| < t$ ▷ Or some other "convergence" criterion
  Return β_0, β_1.
end procedure

GD vs. SGD: Gradient descent uses the *entire* dataset to compute the gradient, takes one step, and repeats. In contrast, stochastic gradient descent uses a *single* randomly chosen data point to compute the gradient, takes one step, and repeats.

Intuition If you look at the updates of β_0 and β_1, they are somewhat intuitive. Suppose for some $x_i > 0$, the model $h_\beta(x_i)$ *overestimates* y_i. This means $(y_i - h_\beta(x_i))$ will be negative. What does the update do? It lowers the intercept and the slope, bringing the line towards the point.

SGD is noisy: since it uses the gradient of squared error on a single point, that gradient may point in a different direction than the gradient of $J(\beta_0, \beta_1)$! However, it will be in the right direction "on average" and the benefit is that it can converge much faster because every it takes a step with every data point.

Differences from book:

- Here a data point is randomly chosen (aka random sampling with replacement); in book, the data is shuffled (equivalent to sampling without replacement).
- The book uses some tricks to help with convergence. If a pass through the data yields no improvement in cost, the step size is lowered. After 100 passes through the data without an improvement, it stops.

Algorithm 2 Stochastic gradient descent for simple linear regression

```
procedure StochasticGradientDescent(dataset, step size η, tolerance t)
    Initialize β_0, β_1 arbitrarily.
    cost ← J(β_0, β_1)                                    ▷ This is computed using the dataset.
    repeat
        for i = 1 ... n do
            Choose a random data point (x_i, y_i) from dataset.    ▷ The "stochastic" part.
            ▷ Use gradient of squared error on point (x_i, y_i) to update coefficients:
            β_0 ← β_0 + η · 2 (y_i − h_β(x_i))
            β_1 ← β_1 + η · 2 (y_i − h_β(x_i)) x_i
        end for
        oldCost ← cost
        cost ← J(β_0, β_1)
    until |cost − oldCost| < t
    Return β_0, β_1.
end procedure
```