Lisa Yan and Jerry Cain
CS109

Lecture Notes #25
November 9, 2020

Linear Regression and Gradient Ascent

Based on a chapter by Chris Piech and Lisa Yan

Pre-recorded lecture: Sections 1, 2, 3.1, and 3.3.
In-lecture: Rest of Section 3.

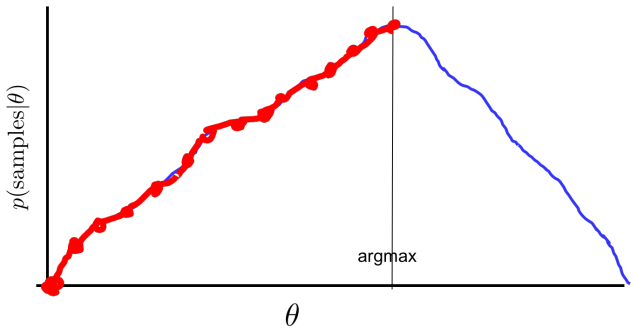
1 Regression

Regression is a second category of Machine Learning prediction algorithms. You have a prediction function $\hat{Y} = g(\mathbf{X})$ as before, but you would like to predict a Y that takes on a **continuous** number.

We won't elaborate on the regression task too much, because classification (with discrete Y) already has a plethora of modern computer science applications—image recognition, sentiment analysis of text, and text authorship, to name a few. However, we will explore *linear regression* (where we model g as a linear function) and learn a truly valuable iterative optimization algorithm (the "butter" to machine learning's "bread," if you will) called **gradient ascent.**

2 Gradient Ascent Optimization

In many cases we can't solve for argmax mathematically. Instead we use a computer. To do so we employ an algorithm called gradient ascent (a classic in optimization theory). The idea behind gradient ascent is that if you continuously take small steps in the direction of your gradient, you will eventually make it to a local maxima.



Start with theta as any initial value (often 0). Then take many small steps towards a local maxima. The new theta after each small step can be calculated as:

$$\theta_j^{\text{new}} = \theta_j^{\text{old}} + \eta \cdot \frac{\partial LL(\theta^{\text{old}})}{\partial \theta_j}$$

Where “eta" (η) is the magnitude of the step size that we take. If you keep updating θ using the equation above you will (often) converge on good values of θ. As a general rule of thumb, use a small value of η to start. If ever you find that the function value (for the function you are trying to argmax) is decreasing, your choice of η was too large. Here is the gradient ascent algorithm in pseudo-code:

```
Initialize: θ_j = 0 for all 0 ≤ j ≤ m

Repeat many times:
    gradient[j] = 0 for all 0 ≤ j ≤ m

    Calculate all gradient[j]'s based on data
    and current setting of theta

    θ_j += η * gradient[j] for all 0 ≤ j ≤ m
```

2.1 Example: Gradient Ascent

Let’s take a function that has an analytical maximum and see how we can use gradient ascent to reach a computational maximum:

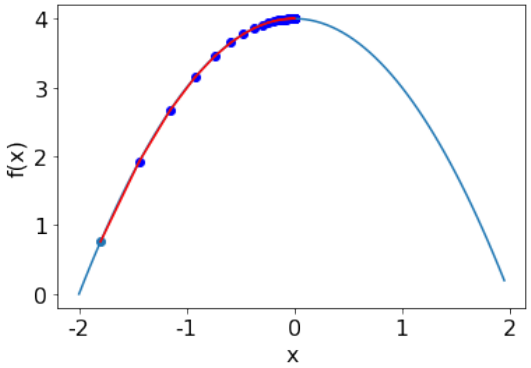
$$f(x) = -x^2 + 4, \text{ where } -1 < x < 2.$$

Analytically, we can set the first derivative to zero and solve. $\frac{df}{dx} = -2x = 0$, and therefore $\arg\max_x f(x) = 0$.

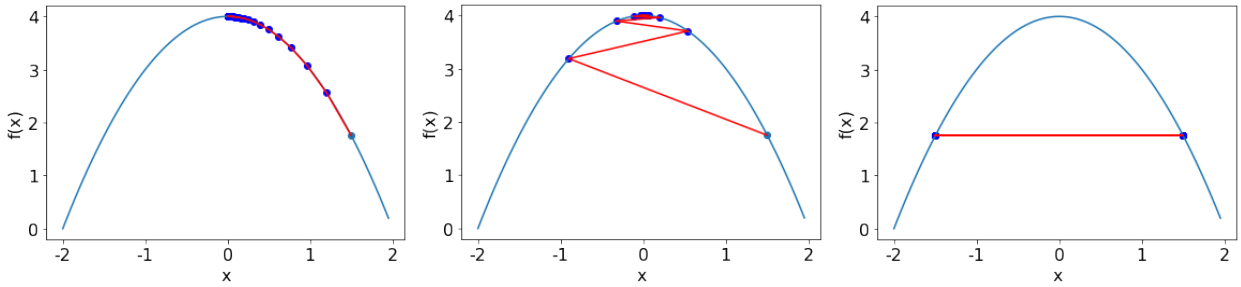
Computationally, we could start at a particular value of x and perform gradient ascent to update x until we reach the maximal value of $f(x)$. The below algorithm starts with an initial guess of $x = -1.8$ and updates with a learning rate $\eta = 0.1$:

```
eta = 0.1       # learning rate
x = -1.8        # initial argmax guess
niters = 100    # number of updates

for i in range(niters):
    x += eta * (-2*x)
```



Depending on the learning rate η that you pick, gradient ascent will take different steps to converge to the maximum; if your learning rate is too large, your algorithm may not converge at all. Supposing we started with an initial guess of $x = 1.5$ and picked different learning rates:



(a) $\eta = 0.1$ (b) $\eta = 0.8$ (c) $\eta = 1.0$

For many of the gradient ascent tasks we study, a reasonable initial guess to start with is $x = 0$. For more advanced tasks, you will want to perturb the initial guess by some random value (e.g., a Gaussian around zero).

3 Linear Regression

Suppose we are working with 1-dimensional observations, i.e., $\mathbf{X} =< X_1 >= X$. Linear Regression assumes the following linear model for prediction, which has two parameters: a and b:

$$\hat{Y} = g(\mathbf{X}) = aX + b$$

Using this model, we would like to determine the optimal parameters according to some optimization objective. We discuss two approaches: an analytical approach that minimizes mean squared error, and a computational approach that maximizes training data likelihood. With one important assumption (which we'll get to later), the two approaches are equivalent.

3.1 Analytical Solution with Mean Squared Error

For regression tasks, we usually decide a prediction $\hat{Y} = g(X)$ that minimizes the *mean squared error* (MSE) "loss" function:

$$\theta_{MSE} = \underset{\theta}{\operatorname{argmin}}\, E[(Y - \hat{Y})^2] = \underset{\theta}{\operatorname{argmin}}\, E[(Y - g(\mathbf{X}))^2] = \underset{\theta}{\operatorname{argmin}}\, E[(Y - aX - b)^2]$$

With our linear prediction model, we determine $\theta_{MSE} = (a_{MSE}, b_{MSE})$ by differentiating the mean squared error with respect to a and b:

$$\frac{\partial}{\partial a} E[(Y - aX - b)^2] = E[-2(Y - aX - b)X] = -2E[XY] + 2aE[X^2] + 2bE[X]$$
$$\frac{\partial}{\partial b} E[(Y - aX - b)^2] = E[-2(Y - aX - b)] = -2E[Y] + 2aE[X] + 2b$$

Setting derivatives to 0 and solving for simultaneous equations:

$$a_{MSE} = \frac{E[XY] - E[X]E[Y]}{E[X^2] - (E[X])^2} = \frac{\text{Cov}(X,Y)}{\text{Var}(X)} = \rho(X,Y)\frac{\sigma_X}{\sigma_Y}$$
$$b_{MSE} = E[Y] - aE[X] = \mu_Y - a\mu_X$$
$$\hat{Y} = \rho(X,Y)\frac{\sigma_Y}{\sigma_X}(X - \mu_X) + \mu_Y$$

Wait, those are our best parameters? But we don't know the distributions of X and Y, and therefore we don't know true statistics on X and Y. We estimate these statistics based on our observed training data, Our model is therefore as follows (where $\bar{X}$ and $\bar{Y}$ are the sample means computed from the training data:

$$\hat{Y} = g(X = x) = \hat{\rho}(X,Y)\frac{\hat{\sigma_Y}}{\hat{\sigma_X}}(x - \bar{X}) + \bar{Y}$$
$$\hat{a}_{MSE} = \frac{\sum_{i=1}^{n}(x^{(i)} - \bar{X})(y^{(i)} - \bar{Y})}{\sum_{i=1}^{n}(x^{(i)} - \bar{X})^2}$$
$$\hat{b}_{MSE} = \bar{Y} - \hat{a}_{MSE}\bar{X}$$

The estimator that minimizes MSE is useful for regression tasks because Y and $\hat{Y}$ tend to be real numbers, and hence the expected error of $E[(Y - \hat{Y})^2]$ will also be a smooth, differentiable function. On the other hand, for classification tasks, Y and $\hat{Y}$ will be discrete class labels, and therefore the MSE may not be a smooth, differentiable function—so we'll need another objective function to optimize. We'll talk about this more next time.

3.2 Computational Solution with Maximum Likelihood

That seemed somewhat anticlimactic: we had this optimal prediction function, but we had to estimate the parameters of the prediction function by averaging the training data. Let's borrow an idea from our parameter estimation days by maximizing the likelihood of seeing our training data!

Recall that our training data has n datapoints: $((x^{(1)}, y^{(1)}), ((x^{(2)}, y^{(2)}), \ldots, ((x^{(n)}, y^{(n)})$, generated i.i.d. according to the joint distribution of X and Y, $f(X, Y|\theta)$. We can model this joint distribution by incorporating our regression model: $Y = \hat{Y} + Z = aX + b + Z$, where $\hat{Y} = g(X) = aX + b$ is our prediction and Z is our error (i.e., noise) between our prediction $\hat{Y}$ and the actual Y.

We approach the problem of finding a and b that maximize the likelihood of our train data by first finding a distribution involving Y, X, and $\theta = (a, b)$. We then find the value of θ that maximizes the log-likelihood function.

If we assume $Z \sim \mathcal{N}(0, \sigma^2)$ and X follows some unknown distribution, then we can calculate the conditional distribution of Y given X is some number x and we have some parameter values $\theta = (a, b)$ as simply $Y = aX + b + Z$. This is just the sum of a Gaussian and a number, thereby implying that $Y|X, \theta \sim \mathcal{N}(aX + b, \sigma^2)$, which has PDF

$$f(Y = y|X = x, \theta) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y-ax-b)^2}{2\sigma^2}}.$$

3.3 Log conditional likelihood

We'd like to find $\theta_{MLE} = (a_{MLE}, b_{MLE})$ which are parameters that maximize our likelihood function, $L(\theta)$. It turns out that for regression tasks (and classification tasks, which we'll talk about next time), it's often easier to maximize **log conditional likelihood**—since our problem often has a conditional distribution of Y given X and parameter θ.

We can prove that θ_{MLE} maximizes log conditional likelihood, as below.

$$\begin{aligned}
\theta_{MLE} &= \arg\max_{\theta} \prod_{i=1}^{n} f(x^{(i)}, y^{(i)}|\theta) \\
&= \arg\max_{\theta} \sum_{i=1}^{n} \log f(x^{(i)}, y^{(i)}|\theta) && (\theta_{MLE} \text{ also maximizes } LL(\theta)) \\
&= \arg\max_{\theta} \sum_{i=1}^{n} \log \left(f(x^{(i)}) f(y^{(i)}|x^{(i)}, \theta) \right) && \text{(chain rule)} \\
&= \arg\max_{\theta} \sum_{i=1}^{n} \log f(x^{(i)}) + \sum_{i=1}^{n} \log f(y^{(i)}|x^{(i)}, \theta) && \text{(properties of logarithms)} \\
&= \arg\max_{\theta} \sum_{i=1}^{n} \log f(y^{(i)}|x^{(i)}, \theta) && (f(x^{(i)}) \text{ constant w.r.t. } \theta)
\end{aligned}$$

3.4 Gradient of Log Conditional Likelihood

Continuing where we left off, we sub in the conditional distribution of $f(y^{(i)}|x^{(i)}, \theta)$ from our model:

$$\begin{aligned}
\sum_{i=1}^{n} \log f(y^{(i)}|x^{(i)}, \theta) &= \sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi}\sigma} e^{-(y^{(i)}-ax^{(i)}-b)^2/(2\sigma^2)} \\
&= \sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi}\sigma} e^{-(y^{(i)}-ax^{(i)}-b)^2/(2\sigma^2)} && \text{(more logs)} \\
&= -n \log \sqrt{2\pi} - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y^{(i)} - ax^{(i)} - b)^2 && \text{(using natural log)}
\end{aligned}$$

Our goal is to find parameters a, b that maximize likelihood. Remember that argmax is invariant of logarithmic transformations *and* positive scalar constants *and* additive constants? Let's remove positive constant multipliers and terms that don't include θ. We are left with trying to find a value of θ that maximizes:

$$\hat{\theta} = \underset{\theta}{\text{argmax}} \left[-\sum_{i=1}^{n} (y^{(i)} - ax^{(i)} - b)^2 \right]$$

3.5 Use Gradient Ascent

To solve this argmax we are going to use Gradient Ascent. In order to do so we first need to find the derivative of the function we want to argmax with respect to both parameters in θ:

$$\begin{aligned}
\frac{\partial}{\partial a} \left[-\sum_{i=1}^{n} (y^{(i)} - ax^{(i)} - b)^2 \right] &= -\sum_{i=1}^{n} \frac{\partial}{\partial a} (y^{(i)} - ax^{(i)} - b)^2 \\
&= -\sum_{i=1}^{n} 2(y^{(i)} - ax^{(i)} - b)(-x^{(i)}) \\
&= 2\sum_{i=1}^{n} (y^{(i)} - ax^{(i)} - b)(x^{(i)}) \\
\frac{\partial}{\partial b} \left[-\sum_{i=1}^{n} (y^{(i)} - ax^{(i)} - b)^2 \right] &= 2\sum_{i=1}^{n} (y^{(i)} - ax^{(i)} - b)
\end{aligned}$$

This first derivative can be plugged into gradient ascent to give our final algorithm:

If you run gradient ascent for enough training (i.e., update) steps, you will find that for linear regression, the maximum likelihood estimators (assuming zero-mean, normally distributed noise between predicted $\hat{Y}$ and actual Y) is equivalent to the mean squared error estimators. Cool!!

```
a, b = 0, 0                  # initialize θ
repeat many times:
  gradient_a, gradient_b = 0, 0
  for each training example (x, y):
    diff = y − (a ∗ x + b)
    gradient_a += 2 ∗ diff ∗ x
    gradient_b += 2 ∗ diff
  a += η ∗ gradient_a        # θ += η ∗ gradient
  b += η ∗ gradient_b
```

3.6 Interpreting the Gradient Step

This is probably your first time seeing gradient ascent in action on your linear regression model—and it may seem like magic. We're going to take a deep dive and see that the magic is in computational power, but the theory is understandable and intuitive.

For each training example $x^{(i)}$, notice that our linear regression model for some parameters $\theta = (a, b)$ is going to give us an estimated $\hat{y^{(i)}} = ax^{(i)} + b$. We know that it's going to be off by some $y - y^{(i)}$, because that's our error (remember that normal noise Z?). This error term is actually in the gradient ascent step!

```
a, b = 0, 0                  # initialize θ
repeat many times:
  gradient_a, gradient_b = 0, 0
  for each training example (x, y):
    prediction_error = y − (a ∗ x + b)
    gradient_a += 2 ∗ prediction_error ∗ x
    gradient_b += 2 ∗ prediction_error
  a += η ∗ gradient_a        # θ += η ∗ gradient
  b += η ∗ gradient_b
```

Gradient ascent is effectively using this prediction error to update its current parameters a and b! The lower the error, the smaller the update. We love it when iterative computations make sense!