Recitation 2

Gradient Descent and Stochastic Gradient Descent

Vishakh

CDS

February 2, 2022

Announcement

- Updates on Office Hours of the Instructors
- HW 2 is due in two weeks
- Optional Questions in the Homework

Agenda

- Gradient Descent
    - Adaptive Learning Rates
- Stochastic Gradient Descent
- Applications
    - Linear Regression
    - Logistic Regression

Recap

Statistical Learning Theory to Gradient Descent

We are given the data set $(x_1, y_1), \ldots, (x_n, y_n)$ where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$.
We want to fit a **linear function** to this data by performing **empirical risk minimization**.
What is the hypothesis space? And what loss function can we use?

Recap

Statistical Learning Theory to Gradient Descent

We are given the data set $(x_1, y_1), \ldots, (x_n, y_n)$ where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$.
We want to fit a **linear function** to this data by performing **empirical risk minimization**.
We search the hypothesis space $\mathcal{F} = \{h_\theta(x) = \theta^T x \mid \theta \in \mathbb{R}^d\}$ to find the function that minimizes the loss as calculated by, let's say, $\ell(a, y) = (a - y)^2$.

Recap

Statistical Learning Theory to Gradient Descent

- The empirical risk is given by

$$J(\theta) := \hat{R}_n(h_\theta) = \frac{1}{n}\sum_{i=1}^{n}\ell(h_\theta(x_i), y_i) = \frac{1}{n}\sum_{i=1}^{n}(\theta^T x_i - y_i)^2$$

- The hypothesis space is parameterized by θ, so finding a good decision function means finding a good θ.
- To find this function we will minimize the empirical loss on the training data. How?

Recap

- Given a current guess for θ, we will use the gradient of the empirical loss (w.r.t. θ) to get a local linear approximation.
- If the gradient is non-zero, taking a small step in the direction of the negative gradient is guaranteed to decrease the empirical loss.
- This motivates the minimization algorithm called gradient descent.

Gradient Descent

Gradient Descent Algorithm

- Goal: find $\theta^* = \arg\min_\theta J(\theta)$
- θ^0 :=[initial condition]
- $i := 0$
- while not [termination condition]:
  - compute $\nabla J(\theta^i)$
  - ϵ_i :=[choose learning rate at iteration i]
  - $\theta^{i+1} := \theta^i - \epsilon_i \nabla J(\theta^i)$
  - $i := i + 1$
- return θ^i

Gradient Checker

- Recall the mathematical definition of the derivative as:

$$\frac{\partial}{\partial\theta} f(\theta) = \lim_{\epsilon \to 0} \frac{f(\theta+\epsilon) - f(\theta-\epsilon)}{2\epsilon}$$

- Approximate the gradient by setting ϵ to a small constant, say $\epsilon = 10^{-4}$. Compare that this is close to your computed value within some tolerance level.
- Now let's expand this method to deal with vector input $\theta \in \mathbb{R}^d$. Let's say we want to verify out gradient at dimension i $(\nabla f(\theta))_i$. We can make use of one-hot vector e_i in which all dimension except the ith are 0 and the ith dimension has a value of 1: $e_i = [0, 0, ..., 1, ..., 0]^T$
- The gradient at ith dimension can be then approximated as

$$[\nabla f(\theta)]^{(i)} \approx \frac{f(\theta + \epsilon e_i) - f(\theta - \epsilon e_i)}{2\epsilon}$$

Gradient Descent

- How to initialize θ^0?
  - sample from some distribution
  - compose θ^0 using some heuristics
  - Glorot et. al, He at. al, PyTorch initializations
- How to choose termination conditions?
  - run for a fixed number of iteration
  - the value of $J(\theta)$ stabilizes
  - θ^i converges
- What is a good learning rate?

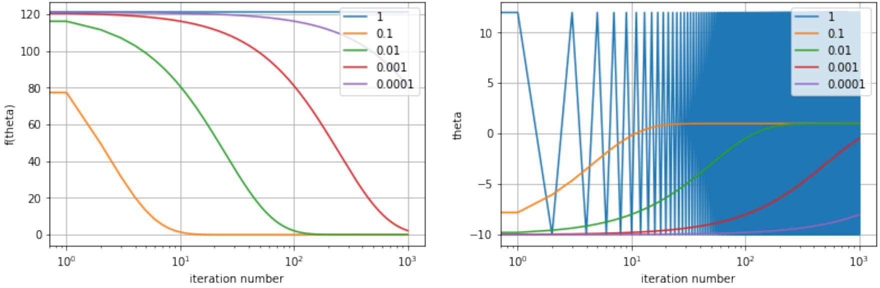
Learning Rate

Application

Suppose we would like to find $\theta^* \in \mathbb{R}$ that minimizes $f(\theta) = \theta^2 - 2\theta + 1$. The gradient (in this case, the derivative) $\nabla f(\theta) = 2\theta - 2$. We can easily see that $\theta^* = \text{argmin}_\theta f(\theta) = 1$.

Learning Rate

- We applied gradient descent for 1000 iterations on $f(\theta) = \theta^2 - 2\theta + 1$ with varying learning rate $\epsilon \in \{1, 0.1, 0.01, 0.001, 0.0001\}$.
- When the learning rate is too large ($\epsilon = 1$), $f(\theta)$ does not decrease through iterations. The value of θ_i at each iteration significantly fluctuates.
- When the learning rate is too small ($\epsilon = 0.0001$), $f(\theta)$ decreases very slowly.



Adaptive Learning Rate

- Instead of using a fixed learning rate through all iterations, we can adjust our learning rate in each iteration using a simple algorithm.
- At each iteration i:
  - $\tilde{\theta} := \theta^{i-1} - \epsilon_{i-1} \nabla f(\theta^{i-1})$
  - $\delta := f(\theta^{i-1}) - f(\tilde{\theta})$
  - if $\delta \geq$ threshold:
    - we achieve a satisfactory reduction on $f(\theta)$
    - $\theta^i = \tilde{\theta}$
    - maybe we can consider increasing the learning rate for next iteration $\epsilon_i := 2\epsilon_{i-1}$
  - else:
    - the reduction is unsatisfactory
    - $\theta^i = \theta^{i-1}$
    - the learning rate is too large, so we reduce the learning rate
    - $\epsilon_i := \frac{1}{2}\epsilon_{i-1}$

Adaptive Learning Rate

How to decide a proper threshold for $f(\theta^{i-1}) - f(\tilde{\theta})$?

Armijo rule

If learning rate ϵ satisfies

$$f(\theta^{i-1}) - f(\tilde{\theta}) \geq \frac{1}{2}\epsilon\|\nabla f(\theta^{i-1})\|^2 \tag{1}$$

then $f(\theta)$ is guaranteed to converge to a (local) maximum under certain technical assumptions.

You can find more details at **this link**.

Stochastic Gradient Descent

Running gradient descent on the entire training dataset can be expensive. Motivates SGD.

SGD Algorithm

- Initialize weights θ^0 :=[initial condition]
- Repeat:
  - Randomly select a data point (x_i, y_i)
  - $\theta^{i+1} = \theta^i - \epsilon \nabla_\theta \ell(f_\theta(x_i), y_i)$
  - Check for stopping critria

Minibatch Gradient Descent

SGD can be noisy. Can we get a better estimate of the gradient?

Minibatch Gradient Descent Algorithm (batch size = n)

- Initialize weights θ^0 :=[initial condition]
- Repeat:
  - Randomly select n data point $(x_i, y_i)_{i=1}^n$
  - $\theta^{i+1} = \theta^i - \epsilon\,[\frac{1}{n}\sum_{i=1}^{n} \nabla_\theta \ell(f_\theta(x_i), y_i)]$
  - Check for stopping critria

- Larger $n \Rightarrow$ Better estimate of gradient but slower
- Smaller $n \Rightarrow$ Worse estimate of gradient but faster

Gradient Descent for Linear Regression

Problem Setup

- Data: $\mathcal{D} = \{x_i, y_i\}_{i=1}^n$ where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$
- Hypothesis Space: $\mathcal{F} = \{f : \mathbb{R}^d \to \mathbb{R} \mid f(x) = \theta^T x,\ \theta \in \mathbb{R}^d\}$
- Action: Prediction on an input x, $\hat{y} = \theta^T x$
- Loss: $l(y, \hat{y}) = (\hat{y} - y)^2$
- Hence, cost function $J(\theta) = \frac{1}{n} \sum_{i=1}^{n} (\theta^T x_i - y_i)^2$
- Goal: Find the optimum theta, $\theta^* = \arg\min_\theta J(\theta)$

Finding θ^*

Approach 1: Closed-Form Solution (HW1)

- $J(\theta) = \frac{1}{n}\sum_{i=1}^{n}(\theta^T x_i - y_i)^2 = \frac{1}{n}||X\theta - b||_2^2$
- To minimize $J(\theta)$, we take the derivative and set it to zero
  - $\nabla J(\theta) = X^T X\theta - X^T y = 0$
  - $\hat{\theta} = (X^T X)^{-1} X^T y$

Pros: One shot algorithm for a direct solution

Cons: Doesn't scale, invertibility constraints

Finding θ^*

Approach 2: Iterative Gradient Descent

- Initialize θ^0
- While not [terminating condition]
  - Compute the gradient $\nabla_\theta J(\theta^{i-1}) = \left[\frac{d}{d\theta_1} J(\theta^{i-1}),\ \frac{d}{d\theta_2} J(\theta^{i-1}) \ldots \frac{d}{d\theta_d} J(\theta^{i-1})\right]^T$
  - $\theta^i = \theta^{i-1} - \epsilon_i \nabla_\theta J(\theta^{i-1})$
- Return θ^i

Pros: Conceptually simple, good chance of convergence

Cons: Slow

Finding θ^*

Approach 3: Stochastic Gradient Descent

- Initialize θ_0
- While not [terminating condition]
  - Select a random training point $\{x_j, y_j\}$
  - $\theta^i = \theta^{i-1} - \epsilon_i \nabla_\theta J^{(j)}(\theta^{i-1})$ where $J^{(j)}(\theta) = (\theta^T x_j - y_j)^2$
- Return θ_i

Pros: Fast and memory efficient

Cons: Practical challenges, noisy gradient steps

Gradient Descent for Logistic Regression

Problem Setup

- Data: $\mathcal{D} = \{x_i, y_i\}_{i=1}^n$ where $x_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$
- Hypothesis Space: (Slightly convoluted)
  $\mathcal{H} = \{h : \mathbb{R}^d \to (0,1) \mid h_\theta(x) = g(\theta^T x),\ \theta \in \mathbb{R}^d, g(z) = \frac{1}{1+e^{-z}}\}$
- Action: Prediction on an input x:
  - $\hat{y} = h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$
  - Interpret this as a probability: $p(y = 1|x;\theta) = h_\theta(x)$ and $p(y = 0|x;\theta) = 1 - h_\theta(x)$
- More succinctly, $p(y|x,\theta) = (h_\theta(x))^y(1 - h_\theta(x))^{1-y}$

Gradient Descent for Logistic Regression

- Goal: Minimize logistic loss over all examples:
  $\ell(h_\theta(x), y) = -y \log h_\theta(x) - (1-y)(1 - \log h_\theta(x))$
- Assuming your data is generated iid.
- Then likelihood, $L(\theta) = P(y_1 \dots y_n | x_1 \dots x_n; \theta) = \prod_{i=1}^{n} p(y_i | x_i; \theta)$
- Maximizing log likelihood,
  $\ell(\theta) = \log L(\theta) = \sum_{i=1}^{n} y_i \log h_\theta(x_i) + (1-y_i)(1 - \log h_\theta(x_i))$
- Equivalent to our goal of loss minimization

Gradient Descent for Logistic Regression

- No analytical one-shot solution
- Find optimum using gradient based approaches
    - The gradient works out to be quite tractable

Gradient Descent for Logistic Regression

- For a single example (x, y):

$$\ell(\theta) = y \log h_\theta(x) + (1 - y)(1 - \log h_\theta(x))$$

$$\frac{\partial}{\partial \theta_j} \ell(\theta) = \frac{\partial \ell(\theta)}{\partial g(\theta^T x)} \frac{\partial g(\theta^T x)}{\partial \theta_j}$$

Gradient Descent for Logistic Regression

- For a single example (x, y):

$$\ell(\theta) = y \log h_\theta(x) + (1 - y)(1 - \log h_\theta(x))$$

$$\begin{aligned} \frac{\partial}{\partial \theta_j}\ell(\theta) &= \frac{\partial \ell(\theta)}{\partial g(\theta^T x)} \frac{\partial g(\theta^T x)}{\partial \theta_j} \\ &= \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial g(\theta^T x)}{\partial \theta_j} \end{aligned}$$

Gradient Descent for Logistic Regression

- For a single example (x, y):

$$\ell(\theta) = y \log h_\theta(x) + (1-y)(1 - \log h_\theta(x))$$

$$\begin{aligned}
\frac{\partial}{\partial \theta_j}\ell(\theta) &= \frac{\partial \ell(\theta)}{\partial g(\theta^T x)} \frac{\partial g(\theta^T x)}{\partial \theta_j} \\
&= \left(y \frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)} \right) \frac{\partial g(\theta^T x)}{\partial \theta_j} \\
&= \left(y \frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)} \right) \left(g(\theta^T x)(1-g(\theta^T x)) \right) \frac{\partial \theta^T x}{\partial \theta_j}
\end{aligned}$$

Gradient Descent for Logistic Regression

- For a single example (x, y):

$$\ell(\theta) = y \log h_\theta(x) + (1-y)(1 - \log h_\theta(x))$$

$$\begin{aligned}
\frac{\partial}{\partial \theta_j}\ell(\theta) &= \frac{\partial \ell(\theta)}{\partial g(\theta^T x)} \frac{\partial g(\theta^T x)}{\partial \theta_j} \\
&= \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial g(\theta^T x)}{\partial \theta_j} \\
&= \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1 - g(\theta^T x)} \right) \left(g(\theta^T x)(1 - g(\theta^T x)) \right) \frac{\partial \theta^T x}{\partial \theta_j} \\
&= \left(y(1 - g(\theta^T x)) - (1-y)(g(\theta^T x) \right) x_j \\
&= (y - h_\theta(x)) x_j
\end{aligned}$$