More on Gradient Descent

Dimitris Papadimitriou

INFO 251: Applied Machine Learning

March 11, 2020

Contents

- Some python technicalities
- Why standardization helps
- GD with OLS in matrix form
- Iteration termination
- Mini-Batch GD
- AdaGrad

Python Vectors and Arrays

- Be consistent on vector representation
- v1.shape = (n,1) or v2.shape = (n,)
- but not both!

GD and Standardization

- Loss function becomes more “spherical"
- Gradient will always point to the minimum
- Avoids “zig-zagging"
- Notebook

OLS in Matrix Form

- What if we have more the 2 or 3 regressors?
- Reminder: $||z||_2^2 = \sum_i^n z_i^2$
- Estimate $\alpha, \beta_1 \cdots, \beta_m$ by
  $\min_{\alpha,\beta_i} \frac{1}{2N} \sum_{i=1}^{N} (y_i - \alpha - \beta_1 x_i^{(1)} - \cdots - \beta_m x_i^{(m)})^2 =$
  $\frac{1}{2N} ||Y - \boldsymbol{\alpha} - X\boldsymbol{\beta}||_2^2$

where $\boldsymbol{\alpha} = \alpha \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$, $X = \begin{bmatrix} x_1^{(1)} & \cdots & x_1^{(m)} \\ x_2^{(1)} & \cdots & x_2^{(m)} \\ \vdots & \vdots & \vdots \\ x_N^{(1)} & \cdots & x_N^{(m)} \end{bmatrix}$ and $Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$

OLS in Matrix Form

- Overview of derivation
- Note the following rules from vector calculus:
- $\frac{\partial AX}{\partial X} = A^T$, $\frac{\partial X^T A}{\partial X} = A$, $\frac{\partial X^T AX}{\partial X} = 2AX$ (the last when A is symmetric)
- Write

$$J(\alpha, \boldsymbol{\beta}) =$$
$$\frac{1}{2N}||Y - \boldsymbol{\alpha} - X\boldsymbol{\beta}||_2^2 = \frac{1}{2N}(Y - \boldsymbol{\alpha} - X\boldsymbol{\beta})^T(Y - \boldsymbol{\alpha} - X\boldsymbol{\beta}) =$$
$$\frac{1}{2N}(Y^T Y - Y^T\boldsymbol{\alpha} - Y^T X\boldsymbol{\beta} - \boldsymbol{\alpha}^T Y + \boldsymbol{\alpha}^T\boldsymbol{\alpha} + \boldsymbol{\alpha}^T X\boldsymbol{\beta}$$
$$- \boldsymbol{\beta}^T X^T Y + \boldsymbol{\beta}^T X^T\boldsymbol{\alpha} + \boldsymbol{\beta}^T X^T X\boldsymbol{\beta})$$

OLS in Matrix Form

- $\frac{1}{2N}||Y - \boldsymbol{\alpha} - X\boldsymbol{\beta}||_2^2$
- We must compute the gradient w.r.t. to the intercept and the slope of
- $J(\alpha, \boldsymbol{\beta}) = \frac{1}{2N}||Y - \boldsymbol{\alpha} - X\boldsymbol{\beta}||_2^2$

$$\frac{J(\alpha, \boldsymbol{\beta})}{\partial \alpha} = \frac{1}{N}\mathbf{1}^T(Y - \boldsymbol{\alpha} - X\boldsymbol{\beta}) = \frac{1}{N}(Y - \boldsymbol{\alpha} - X\boldsymbol{\beta}).sum()$$

(Python notation)

$$\frac{J(\alpha, \boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \frac{1}{N}(X^T Y - X^T\boldsymbol{\alpha} - X^T X\boldsymbol{\beta}) = \frac{1}{N}X^T(Y - \boldsymbol{\alpha} - X\boldsymbol{\beta})$$

Batch Gradient Descent Algorithm (with matrix version of OLS)

Algorithm 1 Batch GD

1: Initialize $\alpha, \beta_1, \ldots, \beta_m$, select R
2: **for** $i = 1, 2, \ldots$ **do**
3:

$$\alpha \leftarrow \alpha - \frac{R}{N}\mathbf{1}^T(Y - \alpha - X\boldsymbol{\beta})$$

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \frac{R}{N}X^T(Y - \alpha - X\boldsymbol{\beta})$$

4: If convergence criterion achieved **break**
5: **end for**

Batch Gradient Descent Algorithm (with matrix version of OLS)

Algorithm 2 Batch GD

1: Initialize $\alpha, \beta_1, \ldots, \beta_m$, select R
2: **for** $i = 1, 2, \ldots$ **do**
3:

$$\alpha \leftarrow \alpha - \frac{R}{N}\mathbf{1}^T(Y - \alpha - X\beta)$$

$$\beta \leftarrow \beta - \frac{R}{N}X^T(Y - \alpha - X\beta)$$

4: If $max(|\beta^{new} - \beta^{old}|) < 1e^{-6}$ and $|\alpha^{new} - \alpha^{old}| < 1e^{-6}$ **break**
5: **end for**

Batch Gradient Descent Algorithm (with matrix version of OLS)

Algorithm 3 Batch GD

1: Initialize $\alpha, \beta_1, \ldots, \beta_m$, select R
2: **for** $i = 1, 2, \ldots$ **do**
3:

$$\alpha \leftarrow \alpha - \frac{R}{N}\mathbf{1}^T(Y - \alpha - X\beta)$$

$$\beta \leftarrow \beta - \frac{R}{N}X^T(Y - \alpha - X\beta)$$

4: If $|Cost^{new} - Cost^{old}| < 1e^{-6}$ equivalent **break**
5: **end for**

SGD and Mini-Batch GD Algorithms

Algorithm 4 Mini-Batch GD

1: Initialize α, β, select R, batch-size K
2: **for** $i = 1, 2, \ldots$ **do**
3: Randomly shuffle data
4: **for** $j = 1, 2, \ldots, \lfloor N/K \rfloor$ **do**
5: Choose jth batch (batch(j)) from data
6:

$$\alpha \leftarrow \alpha - R\frac{\partial J^j(\alpha, \beta)}{\partial \alpha}$$
$$\beta \leftarrow \beta - R\frac{\partial J^j(\alpha, \beta)}{\partial \beta}$$

7: **end for**
8: If convergence criterion achieved **break**
9: **end for**

where for our bivariate regression problem

$$\frac{\partial J^j(\alpha, \beta)}{\partial \alpha} = \frac{1}{K} \sum_{k \in batch(j)} y_k - \alpha - \beta x_k, \quad \frac{\partial J^j(\alpha, \beta)}{\partial \beta} = \frac{1}{K} \sum_{k \in batch(j)} (y_k - \alpha - \beta x_k) x_k$$

SGD and Mini-Batch GD Algorithms

- The SGD and Mini-Batch implementation above is called cyclic
- In practice instead of shuffling and going over the data set in batches just sample randomly without replacement a subset os K data points in each iteration.
- See notebook for implementation

Logistic Regression

- Classification method
- Given features x_i predict label Y_i.
- $P(Y_i = 1) = \hat{Y}_i = \frac{1}{1+e^{-\alpha-\beta x_i}}$
- Cost function; $\sum_{i=1}^{N} -Y_i log(\hat{Y}_i) - (1 - Y_i) log(1 - \hat{Y}_i)$
- Where does this come from $\rightarrow$ Maximum Likelihood
- Max $J(\alpha, \beta) = \prod_{i=1}^{N} (\frac{1}{1+e^{-\alpha-\beta x_i}})^{Y_i} (1 - \frac{1}{1+e^{-\alpha-\beta x_i}})^{1-Y_i}$
- Key in gradient calculations: $h(x) = \frac{1}{1+e^{-x}}$ then $\frac{\partial h(x)}{\partial x} = h(x)(1 - h(x))$

Logistic Regression

- Update rules:

$$\frac{\partial J(\alpha,\beta)}{\partial \alpha} = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \frac{1}{1+e^{-\alpha-\beta x_i}})$$

$$\frac{\partial J(\alpha,\beta)}{\partial \beta} = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \frac{1}{1+e^{-\alpha-\beta x_i}})x_i$$

- Prediction: for new data point x_i we predict:
- after estimating $\hat{\alpha}, \hat{\beta}$ given new x_i
- label 1 if $\frac{1}{1+e^{-\hat{\alpha}-\hat{\beta}x_i}} > 0.5$ and zero otherwise
- Report accuracy N_{test} data points: $\sum_{i=1}^{N_{test}} 1\{\hat{Y} == Y_i\}/N_{test}$

Ridge Regression

- $J(\alpha, \boldsymbol{\beta}) = \frac{1}{2N}||Y - \boldsymbol{\alpha} - X\boldsymbol{\beta}||_2^2 + \lambda||\beta||_2^2$
- What do we penalize?
- Need to normalize?
- Only difference from not penalized OLS is the extra $\lambda\beta$ term in the gradients

AdaGrad

- Intuition: Reduce the learning rates of parameters slower when corresponding data is sparse
- Update rule:

$$\beta_j = \beta_j - \frac{R}{\sqrt{G_j^{(k)}}} \frac{\partial J(\alpha, \beta_1, \ldots)}{\partial \beta_j}$$

where $G_j^{(k)} = \sum_{i=1}^{k} (\frac{\partial J^{(i)}(\alpha, \beta_1, \ldots)}{\partial \beta_j})^2$ and R is your learning rate.