Tufts Class #03: Gradient Methods

Machine Learning (COMP 135)

1

Review: Minimizing Squared Error

- For a chosen set of weights, $\mathbf{w}$, used in linear regression, we define error as the (squared) difference between hypothesis function and correct output, summed over all test-cases:

$$Loss(\mathbf{w}) = \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(\mathbf{x}_j))^2$$

- Learning is then the process of finding a weight-sequence that ***minimizes*** this loss:

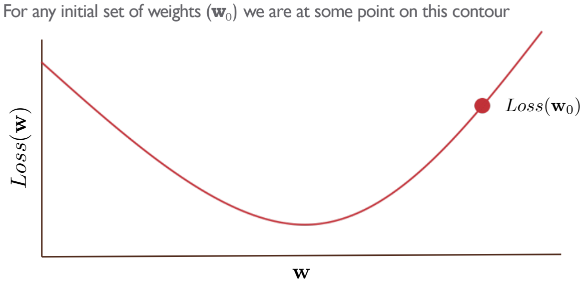
$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} Loss(\mathbf{w})$$

2

Gradient Descent

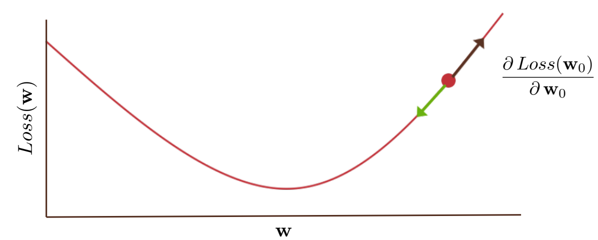
$$Loss(\mathbf{w}) = \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(\mathbf{x}_j))^2$$

- The loss function forms a ***contour*** (here shown for one-dimensional data)
  - For any initial set of weights ($\mathbf{w}_0$) we are at some point on this contour



3

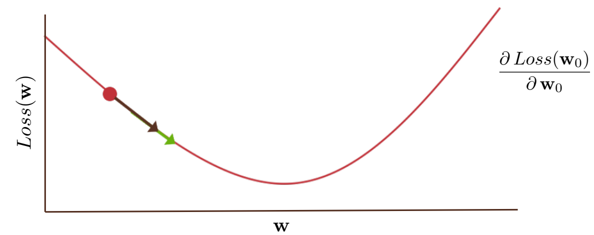
Gradient Descent



- At this point, the derivate of the loss function points "uphill"
  - The gradient descent update moves along the function in the ***opposite*** direction, to decrease loss most significantly

4

Gradient Descent



- At ***other*** points, the derivate points "downhill" ***already***
  - Gradient descent thus moves along the function in the ***same*** direction to decrease loss

5

The Loss Gradient

$$\mathcal{L} = \sum_{j=1}^{n} (y_j - h_{\mathbf{w}}(\mathbf{x}_j))^2 = \sum_{j=1}^{n} (y_j - \mathbf{w} \cdot \mathbf{x}_i)^2$$

- For this loss function, the gradient with respect to any single weight is the first derivative of the loss applied to that weight:

$$\nabla \mathcal{L} = \frac{\partial \mathcal{L}}{\partial w_i} = -2 \sum_{j=1}^{n} x_{j,i} (y_j - \mathbf{w} \cdot \mathbf{x}_j)$$

- We can then modify that weight by ***subtracting*** the gradient:

1. If the gradient is ***positive*** along the weight-axis, we are ***decreasing*** the weight to move in the ***opposite*** direction
2. If the gradient is ***negative***, we are ***increasing*** the weight to move in the ***same*** direction

6

Modifying the Weight Updates

- In theory, we could modify a weight by applying the gradient ***directly***:

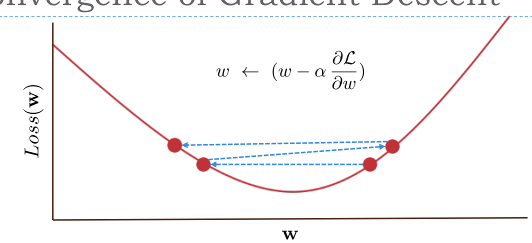
$$w_i \leftarrow (w_i - \frac{\partial \mathcal{L}}{\partial w_i}) = (w_i + 2 \sum_{j=1}^{n} x_{j,i} (y_j - \mathbf{w} \cdot \mathbf{x}_j))$$

- In practice, however, this does not work well
  - Changing weights too much can "over-shoot" minimal points in the loss gradient
- Instead, we apply a step-size parameter to the weights
  - A multiplier, α, most often < 1, to decrease the magnitude of update

$$w_i \leftarrow (w_i - \alpha \frac{\partial \mathcal{L}}{\partial w_i}) = (w_i + 2\alpha \sum_{j=1}^{n} x_{j,i} (y_j - \mathbf{w} \cdot \mathbf{x}_j))$$

7

Convergence of Gradient Descent



- In the presence of large changes to the weights, the result can "ping pong" around the loss space in a way that ***never*** settles near a minimum
- Also known as the learning rate, α provides a control parameter for this process

1. This can be ***fixed*** to some small constant: $\alpha = 0.001$
2. Or, we may ***decay*** the parameter, making it smaller over time, decreasing it as a function of t, the number of iterations of the process: $\alpha_0 = C \qquad \alpha_t = \frac{C}{t} \quad (t \geq 1)$

8

A Mathematical Convenience

- We can use a trick to make gradient computation more efficient, multiplying our loss function by a constant, c:

$$\mathcal{L}' = c \times \mathcal{L} = c \sum_{j=1}^{n} (y_j - \mathbf{w} \cdot \mathbf{x}_j)^2$$

- For a positive constant, this ***does not*** affect the target, minimizing weights:

$$\arg\min_{\mathbf{w}} \mathcal{L}' \;=\; \arg\min_{\mathbf{w}} c\,\mathcal{L} \;=\; \arg\min_{\mathbf{w}} \mathcal{L}$$

- Furthermore, the gradient for a single weight is also simple:

$$\nabla \mathcal{L}' = \frac{\partial \mathcal{L}'}{\partial w_i} = -2c \sum_{j=1}^{n} x_{j,i} (y_j - \mathbf{w} \cdot \mathbf{x}_j)$$

9

A Mathematical Convenience

$$\nabla \mathcal{L}' = \frac{\partial \mathcal{L}'}{\partial w_i} = -2c \sum_{j=1}^{n} x_{j,i} (y_j - \mathbf{w} \cdot \mathbf{x}_j)$$

- Given this form of the gradient, we can simplify things by setting our multiplier $c = 1/2$, which means that the derivative calculation reduces somewhat:

$$\mathcal{L}' = \frac{1}{2} \sum_{j=1}^{n} (y_j - \mathbf{w} \cdot \mathbf{x}_j)^2$$

$$\nabla \mathcal{L}' = \frac{\partial \mathcal{L}'}{\partial w_i} = - \sum_{j=1}^{n} x_{j,i} (y_j - \mathbf{w} \cdot \mathbf{x}_j)$$

10

A Mathematical Convenience

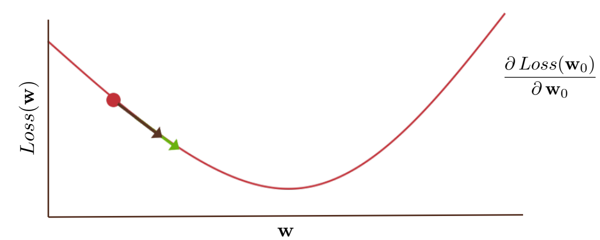
$$\nabla \mathcal{L}' = \frac{\partial \mathcal{L}'}{\partial w_i} = - \sum_{j=1}^{n} x_{j,i} (y_j - \mathbf{w} \cdot \mathbf{x}_j)$$

- This final form of the gradient gives us the weight update rule for linear regression
  - Descending the gradient by ***subtracting*** it out updates each
  - We use the learning rate to control the ***speed*** of this descent (i.e., the exact amount we are subtracting)

$$w_i \;\leftarrow\; w_i + \alpha \sum_{j} x_{j,i} \, (y_j - h_{\mathbf{w}}(\mathbf{x}_j))$$

11

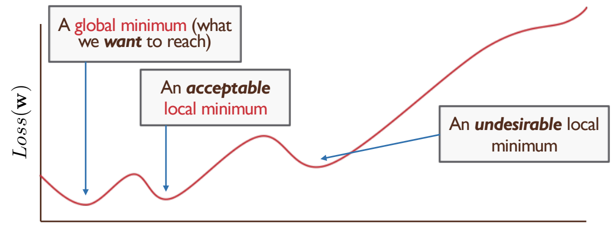
Potential Issues in Gradient Descent



- The squared-error loss for linear regression has a convex functional form
  - This means that, handled properly, gradient descent will converge upon a solution that is (very close to) optimal

12

Potential Issues in Gradient Descent



- When loss functions are complex and non-convex, descending the gradient ***may not*** guarantee optimality
  - Local minima in the loss function are possible
  - Can be dealt with by a variety of techniques, e.g. randomly repeating starts
  - Can often be tolerated, so long as a ***reasonable*** minimum is found
  - We will see such non-convex scenarios later in the course

Variants of Gradiant Descent: Batch

- The gradient descent technique we described updates weights across ***all*** data in the training set:

1. Loop over all weights w_i, updating them:

$$w_i \leftarrow w_i + \alpha \sum_j x_{j,i} \left(y_j - h_{\mathbf{w}}(\mathbf{x}_j)\right)$$

2. Stop on convergence

- This is also known as batch gradient descent
  - A very stable algorithm: as long as learning rate α is not too large, will converge well to optimal or near-optimal solutions
  - Can be quite slow when data-set is large

Variants of Gradiant Descent: Stochastic

- Another version is stochastic gradient descent, where we use only ***one*** data-point at a time:

1. Pick some data-point $\mathbf{x}_j$
2. Loop over all weights w_i, updating them:

$$w_i \leftarrow w_i + \alpha(x_{j,i}(y_j - h_{\mathbf{w}}(\mathbf{x}_j)))$$

3. Stop on convergence

- A faster technique, but less stable
  - We must be careful to reduce α slowly in order to converge to a good solution

Variants of Gradiant Descent: Mini-Batch

- An "in-between" version is mini-batch gradient descent
- Like the batch version, we sum the overall weighted error, but only do so over some ***fixed-size proper subset*** of the data at any point
  - Can be more efficient than batch when data-set is very large
  - More stable than stochastic method, but care still needed to ensure that the algorithm will converge properly to minima
- Each of these approaches is used with many ML models
  - All that effectively changes is the loss function used, its gradient, and the resulting derivative calculation