CS 188 Introduction to Artificial Intelligence
Fall 2023 Note 23

Author (all other notes): Nikhil Sharma

Author (Bayes' Nets notes): Josh Hug and Jacky Liang, edited by Regina Wang

Author (Logic notes): Henry Zhu, edited by Peyrin Kao

Credit (Machine Learning and Logic notes): Some sections adapted from the textbook *Artificial Intelligence: A Modern Approach*.

Last updated: August 26, 2023

In this note we will cover how to optimize functions using the gradient descent algorithm. We will also learn about a classification method called logistic regression and how it can be extended to multi-class classification. This will motivate our further discussion on neural networks and backpropagation.

Optimization

We saw in the previous note in the linear regression method that we can derive a closed form solution for the optimal weights by just differentiating the loss function and setting the gradient equal to zero. In general though, a closed form solution may not exist for a given objective function. In cases like that we have to use **gradient-based methods** to find the optimal weights. The idea behind this is that the gradient points towards the direction of steepest increase of the objective. We maximize a function by moving towards the steepest **ascent**, and we minimize a function by moving towards the steepest **descent** direction.

Gradient ascent is used if the objective is a function which we try to maximize.

Randomly initialize $\mathbf{w}$
while $\mathbf{w}$ *not converged* **do**
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha \nabla_{\mathbf{w}} f(\mathbf{w})$
end

Algorithm 1: Gradient ascent

Gradient descent is used if the objective is a loss function that we are trying to minimize. Notice that this only differs from gradient ascent in that we follow the opposite direction of the gradient.

Randomly initialize $\mathbf{w}$
while $\mathbf{w}$ *not converged* **do**
  $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} f(\mathbf{w})$
end

Algorithm 2: Gradient descent

At the beginning, we initialize the weights randomly. We denote the learning rate, which captures the size of the steps we make towards the gradient direction, with α. For most functions in the machine learning world it is hard to come up with an optimal value for the learning rate. In reality, we want a learning rate that is large enough so that we move fast towards the correct direction but at the same time small enough so that the method does not diverge. A typical approach in machine learning literature is to start gradient descent with a relatively large learning rate and reduce the learning rate as the number of iterations increases

(**learning rate decay**).

If our dataset has a large number of n data points then computing the gradient as above in each iteration of the gradient descent algorithm might be too computationally intensive. As such, approaches like stochastic and batch gradient descent have been proposed. In **stochastic gradient descent** at each iteration of the algorithm we use only one data point to compute the gradient. That one data point is each time randomly sampled form the dataset. Given that we only use one data point to estimate the gradient, stochastic gradient descent can lead to noisy gradients and thus make convergence a bit harder. **Mini-batch gradient descent** is a compromise between stochastic and the ordinary gradient descent algorithm as it uses a batch of size m of data points each time to compute the gradients. The batch size m is a user specified parameter.

Let's see an example of gradient descent on a model we've seen before—linear regression. Recall that in linear regression, we defined our loss function as

$$Loss(h_{\mathbf{w}}) = \frac{1}{2} \|\mathbf{y} - \mathrm{X}\mathbf{w}\|_2^2$$

Linear regression has a celebrated closed form solution $\hat{\mathbf{w}} = (\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T\mathbf{y}$, which we saw in the last note. However, we could have also chosen to solve for the optimal weights by running gradient descent. We'd calculate the gradient of our loss function as

$\nabla_{\mathbf{w}} Loss(h_{\mathbf{w}}) = -\mathrm{X}^T\mathbf{y} + \mathrm{X}^T\mathrm{X}\mathbf{w}$

Then, we use this gradient to write out the gradient descent algorithm for linear regression:

Randomly initialize **w**
while **w** *not converged* **do**
  $\mathbf{w} \leftarrow \mathbf{w} - \alpha(-\mathrm{X}^T\mathbf{y} + \mathrm{X}^T\mathrm{X}\mathbf{w})$
end

Algorithm 3: Least squares gradient descent

It is a good exercise to create a linear regression problem and confirm that the closed form solution is the same as the converged solution you obtain from gradient descent.