# 1 Introduction

In this course note, we will be discussing advanced optimization techniques for machine learning. We start with a simple review of gradient descent and stochastic gradient descent.

Let $L(w)$ denote the objective function to be minimized during training. For instance, in machine learning, $L(w)$ often includes a loss function summed over the training examples $S$ as well as the regularization penalty:

$$L(w) = \frac{\lambda}{2}\|w\|_2^2 + \sum_{(x_i,y_i)\in S} \ell(x_i, y_i, w),$$

where $\lambda$ denotes the regularization strength of the (in this case) L2 regularization penalty, and $\ell(x, y, w)$ denotes the training error of $w$ on training example $(x, y)$ (e.g., squared error on a linear model: $(y - w^T x)^2$).

The goal then is to solve the following unconstrained optimization problem:

$$\text{argmin}_w \, L(w). \tag{1}$$

We mostly will assume that $L$ is convex.

The simplest way to minimize (1) is via **gradient descent**,

**Algorithm 1** Gradient Descent

1: Initialize $w_0$, typically as 0
2: $t \leftarrow 1$
3: Compute gradient: $g_t \leftarrow \nabla_w L(w = w_{t-1})$
4: Update: $w_t \leftarrow w_{t-1} - \eta_t g_t$
5: $t \leftarrow t + 1$
6: Repeat from Step 3 until some termination condition is met

The main design decision is the choice of step size $\eta_t$. In general, each $\eta_t$ needs to be sufficient small, or else gradient descent will not converge. However, larger choices of $\eta_t$ leads to faster convergence. The simplest approach is to use a fixed step size.

Another way, which is often more efficient in practice, is **stochastic gradient descent**,

**Algorithm 2** Stochastic Gradient Descent

1: Initialize $w_0$, typically as 0
2: $t \leftarrow 1$
3: Compute stochastic gradient: $g_t$ s.t. $E[g_t] = \nabla_w L(w = w_{t-1})$
4: Update: $w_t \leftarrow w_{t-1} - \eta_t g_t$
5: $t \leftarrow t + 1$
6: Repeat from Step 3 until some termination condition is met

The main difference between stochastic gradient and gradient descent is in Step 3, where the gradient used $g_t$ is now a stochastic gradient. In practice, virtually all objective functions can be decomposed additively:

$$L(w) = \sum_{i=1} L_i(w).$$

For instance, the most straightforward decomposition decomposition is:

$$L_i(w) = \frac{\lambda}{2N}\|w\|_2^2 + \ell(x_i, y_i, w).$$

For any given decomposition, stochastic gradient descent chooses an $L_i$ randomly at Step 3. It is easy to verify that

$$E[g_t] = E_i[\nabla_w L_i(w = w_{t-1})] = \frac{1}{N}\nabla_w L(w = w_{t-1}),$$

where $N$ is the number of components in decomposition of $L$. In practice, one often loops through all the $L_i$'s in some order. The key benefit of stochastic gradient descent over gradient descent is that it requires much less computation time (e.g., only processing a few training examples rather than the entire training set). However, the gradient is noisier, and so typically the step size $\eta_t$ is much smaller than that for standard gradient descent. The term **mini-batch SGD** is often used to refer to the setting where each $L_i$ contains a small batch of training data (e.g., 10-500 samples).

## 2 Accelerated Gradient Descent

Nesterov's accelerated gradient descent essentially adds a momentum term to the gradient descent procedure. The general form will look something like this:

1. Compute gradient $g_t$
2. Update momentum term $m_t$
3. Update model $w_t$ using both $g_t$ and $m_t$

We first present the original Nesterov's method, which was designed for gradient descent.

**Algorithm 3** Nesterov's Accelerated Gradient Descent

1: Initialize $w_0$ and $v_0$, typically as $0$
2: Initialize $\alpha_0 \leftarrow 0$
3: $t \leftarrow 1$
4: Compute gradient: $g_t \leftarrow \nabla_w L(w = w_{t-1})$
5: Update to intermediate value: $v_t \leftarrow w_{t-1} - \eta_t g_t$
6: $\alpha_t \leftarrow \frac{1+\sqrt{1+4\alpha_{t-1}^2}}{2}$
7: $\gamma_t \leftarrow \frac{1-\alpha_{t-1}}{\alpha_t}$
8: Update with momentum: $w_t \leftarrow (1-\gamma_t)v_t + \gamma_t v_{t-1}$
9: $t \leftarrow t+1$
10: Repeat from Step 4 until some termination condition is met

In other words, Nesterov's method keep track of an intermediate solution $v_t$ that is the direct result of gradient descent (see Line 5). However, the model update rule is one that combines both the immediate gradient as well as a momentum term (Line 8). One can rewrite Line 8 as:

$$w_t \leftarrow v_t + \gamma_t(v_{t-1} + v_t) = v_t - \gamma_t(v_t - v_{t-1}), \tag{2}$$

which can be interpreted as moving further along the direction of $v_t - v_{t-1}$ by a magnitude of $-\gamma_t$. Note that $\gamma_1 = 0$, and that $\gamma_t$ very quickly converges to $-1$. So in the limit, (2) is behaving as:

$$w_t \leftarrow v_t + (v_t - v_{t-1}). \tag{3}$$

So to summarize:

- $v_t \leftarrow w_{t-1} - \nabla_w L(w = w_{t-1})$ is computed via the standard gradient descent update rule.

- $w_t \leftarrow v_t - \gamma_t(v_t - v_{t-1})$ is computed via updating $v_t$ with the momentum term $(v_t - v_{t-1})$ scaled by $-\gamma_t$ (where $\gamma_t \leq 0$).

Of course there are many different ways to choose a momentum term, so why this particular choice? It was proven in [1] that

$$L(v_t) - L(w^*) \leq \frac{2\|w_0 - w^*\|^2}{\eta t^2},$$

where $w^*$ is the minimizer of $L$, and $\eta$ lower bounds all $\eta_t$.[1] In other words, for any error tolerance $\epsilon > 0$, it takes $O(1/\sqrt{\epsilon})$ time steps to achieve $L(v_t) - L(w^*) \leq \epsilon$. This can be much faster than the general $O(1/\epsilon)$ convergence rate for ordinary gradient descent for differentiable convex functions.

### 2.1 Accelerated Stochastic Gradient Descent

There is no universally accepted way to incorporate Nesterov's method into stochastic gradient descent. One straightforward approach is to do the momentum update periodically, e.g., once every 10-500 rounds of mini-batch SGD. One version is shown below, with $B$ denoting the number of rounds to iterate per momentum update.

**Algorithm 4** Nesterov's Accelerated Gradient Descent

1: Initialize $w_0$ and $v_0$, typically as 0
2: Initialize $\alpha_0 \leftarrow 0$
3: $t \leftarrow 1$
4: Compute stochastic gradient: $g_t$ s.t. $E[g_t] = \nabla_w L(w = w_{t-1})$
5: **if** $\text{mod}(t, B) = 0$ **then**
6: Update to intermediate value: $v_t \leftarrow w_{t-1} - \eta_t g_t$
7: $\alpha_t \leftarrow \frac{1+\sqrt{1+4\alpha_{t-1}^2}}{2}$
8: $\gamma_t \leftarrow \frac{1-\alpha_{t-1}}{\alpha_t}$
9: Update with momentum: $w_t \leftarrow (1 - \gamma_t)v_t + \gamma_t v_{t-1}$
10: **else**
11: Standard SGD update: $w_t \leftarrow w_{t-1} - \eta_t g_t$
12: $v_t \leftarrow v_{t-1}$
13: **end if**
14: $t \leftarrow t + 1$
15: Repeat from Step 4 until some termination condition is met

If $B = 1$, then the momentum is applied every round. However, this can often lead to unstable estimates since the individual stochastic gradients can be quite noisy. By average over, say, 100 rounds of stochastic gradients, then the momentum estimate is more stable.

## 3 Relationship Between Gradient Descent and Proximal Updates

Let us consider an alternative form of iterative optimization:

$$w_t = \text{argmin}_{w'} \, L(w') + \frac{1}{2\eta_t}\|w' - w_{t-1}\|_2^2. \tag{4}$$

[1] In the case where each $\eta_t$ is constant, then $\eta = \eta_t$.

Here, the idea is that we want $w_t$ to be the best solution to $L$ that is not to far from the previous $w_{t-1}$. So long as $1/\eta_t$ is sufficiently large, then the so-called "proximal" will dominate whenever $w'$ is far from $w_{t-1}$. This form of the optimization problem might seem a bit circular, since if we can solve (4) then we might be able to solve the original problem. However, we shall see later that this form has some very nice properties.

We can show that a variant of (4) is equivalent to gradient descent. By properties of convex functions, we can lower bound (4) as:

$$L(w') + \frac{1}{2\eta_t}\|w' - w_{t-1}\|_2^2 \geq L(w_{t-1}) + \nabla_w L(w = w_{t-1})^T(w' - w_{t-1}) + \frac{1}{2\eta_t}\|w' - w_{t-1}\|_2^2. \tag{5}$$

The closer $w'$ is to $w_t$, the closer the gap, and at $w' = w_t$, (5) becomes an equality.

One can thus consider a variant of (4), which can be thought of as the linear approximation to (4) at $w_{t-1}$:

$$\text{argmin}_{w'} \nabla_w L(w = w_{t-1})^T(w' - w_{t-1}) + \frac{1}{2\eta_t}\|w' - w_{t-1}\|_2^2. \tag{6}$$

Taking the derivative of (6) and setting it to 0 yields:

$$0 = \nabla_w L(w = w_{t-1}) + \frac{1}{\eta_t}(w' - w_{t-1}) \Rightarrow w_t = w_{t-1} - \eta_t \nabla_w L(w = w_t),$$

which is exactly the gradient descent update rule. In other words, the gradient descent update rule is the closed form solution to (6), which is in turn the linear approximation to to (4) at $w_{t-1}$. So one can think of (4) and (6) as a generalization of gradient descent. Specifically, gradient descent is the closed-form solution to (6) whenever $L$ is differentiable.

The more general setting can be thought of as:

**Algorithm 5** Proximal Updates

1: Initialize $w_0$, typically as 0
2: $t \leftarrow 1$
3: Update: $w_t \leftarrow \text{argmin}_{w'}$ (4) or $\text{argmin}_{w'}$ (6)
4: $t \leftarrow t + 1$
5: Repeat from Step 3 until some termination condition is met

# 4 Iterative Soft-Thresholding for Non-Differentiable Functions

We now consider the case where we can decompose $L$ into differentiable and non-differentiable components:

$$L(w) = G(w) + H(w),$$

where $G$ is differentiable and $H$ is not differentiable. For example, $G$ can be a differentiable loss function summed over all the training examples:

$$G(w) = \sum_{(x_i, y_i) \in S} \ell(x, y, w),$$

and $H$ can be the L1 penalty:

$$H(w) = \lambda \|w\|_1.$$

We will show how to solve such an $L$ using the Iterative Soft-Thresholding Algorithm (ISTA). We do so by alternating between solving $G$ and $H$. For the differentiable part $G$, we solve for the next step using (6), which yields a closed form solution of:

$$v_t \leftarrow \text{argmin}_{w'} \nabla_w G(w = w_{t-1})^T (w' - w_{t-1}) + \frac{1}{2\eta_t} \|w' - w_{t-1}\|_2^2 \equiv w_{t-1} - \eta_t \nabla_w G(w = w_{t-1}). \tag{7}$$

Afterwards, for the non-differentiable part $H$, we instead solve using (4), which yields:

$$w_t \leftarrow \text{argmin}_{w'} H(w') + \frac{1}{2\eta_t} \|w' - v_t\|_2^2, \tag{8}$$

where $v_t$ was the intermediate solution to the gradient update of $G$ (7).

It turns out that for many commonly used non-differentiable functions, $H$, (8) has a closed-form solution. Differentiating (8) and setting it to 0 yields:

$$0 = \nabla_w H(w = w') + \frac{1}{\eta_t}(w' - v_t) \Rightarrow \nabla_w \left[\eta_t H(w = w')\right] + w' = v_t. \tag{9}$$

Consider L1 regularization $H(w) = \lambda \|w\|_1$. We can write the sub-differential of $\alpha H$ (for any positive constant $\alpha$) component-wise as:

$$\nabla_w \alpha\lambda \|w\|_1 = \begin{cases} -\alpha\lambda & \text{if} \quad w \leq -\alpha\lambda \\ \left[-\alpha\lambda, \alpha\lambda\right] & \text{if} \quad -\alpha\lambda < w < \alpha\lambda \\ \alpha\lambda & \text{if} \quad w \geq \alpha\lambda \end{cases}$$

In other words, when $w$ is close to 0, there is a continuous range of differentials, and any of them that satisfies (9) is an optimal solution. We can thus write the closed-form solution to (9) component-wise as:

$$w_t = \begin{cases} v_t + \eta_t\lambda & \text{if} \quad v_t \leq -\eta_t\lambda \\ 0 & \text{if} \quad -\eta_t\lambda < v_t < \eta_t\lambda \\ v_t - \eta_t\lambda & \text{if} \quad v_t \geq -\eta_t\lambda \end{cases} \tag{10}$$

The above update rule is known as "soft-thresholding". The entire algorithm is thus:

**Algorithm 6** Iterative Soft-Thresholding Algorithm (ISTA) for optimizing L1 regularized training objectives

1: Initialize $w_0$, typically as 0
2: Decompose $L = G + H$, where $H = \lambda|w|_1$
3: $t \leftarrow 1$
4: Update: $v_t \leftarrow \text{argmin}_{w'}$ (4), the solution is given in (7)
5: Update: $w_t \leftarrow \text{argmin}_{w'}$ (6), the solution is given in (10)
6: $t \leftarrow t + 1$
7: Repeat from Step 4 until some termination condition is met

Of course, one can also use standard sub-gradient descent for optimizing L1 regularized training objectives. However, the goal of using L1 regularization is so that the solution $w$ is sparse (few non-zeros). However, standard gradient descent will not actually produce a sparse solution (you can try it yourself). ISTA is guaranteed to produce a sparse solution because it zeros out small components of $w$.

The above algorithm is gradient descent, and requires differentiating the entire $G$ each iteration. In practice, one often solves for $G$ using stochastic gradient descent. In which case, one straightforward approach is to do some number of rounds of mini-batch SGD on $G$, followed by a soft-thresholding step. Just keep in mind that the step sizes $\eta_t$ need to be adjusted. If you do $B$ rounds of mini-batch SGD on $G$ for each soft-thresholding step, then you need to set the soft-thresholding step-size to $B\eta_t$.

### 4.1 Fast Iterative Soft-Thresholding Algorithm

The new approach presented in [1] is actually the Fast Iterative Soft-Thresholding Algorithm (FISTA), which combines ISTA with Nesterov's accelerated gradient descent.

**Algorithm 7** Fast Iterative Soft-Thresholding Algorithm (FISTA) for optimizing L1 regularized objectives

1: Initialize $w_0$ and $u_0$, typically as $0$
2: Decompose $L = G + H$, where $H = \lambda|w|_1$
3: Initialize $\alpha_0 \leftarrow 0$
4: $t \leftarrow 1$
5: Update: $v_t \leftarrow \text{argmin}_{w'}$ (4), the solution is given in (7)
6: Update: $u_t \leftarrow \text{argmin}_{w'}$ (6), the solution is given in (10)
7: $\alpha_t \leftarrow \frac{1+\sqrt{1+4\alpha_{t-1}^2}}{2}$
8: $\gamma_t \leftarrow \frac{1-\alpha_{t-1}}{\alpha_t}$
9: Update: $w_t \leftarrow (1-\gamma_t)u_t + \gamma_t u_t$
10: $t \leftarrow t+1$
11: Repeat from Step 5 until some termination condition is met

## References

[1] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.