Multi-Task Learning Notes (II): Acting on Gradients

Su Jianlin

February 8, 2022

In "Multi-Task Learning Notes (I): In the Name of Loss", we initially explored the problem of multi-task learning from the perspective of loss functions. We eventually found that if we want the results to possess both scaling invariance and translation invariance, using the reciprocal of the gradient norm as the task weight is a relatively simple choice. We further analyzed that this design is equivalent to normalizing the gradient of each task individually before summing them. This means the "battlefield" of multi-task learning has shifted from the loss function to the gradients: it appears we are designing loss functions, but in reality, we are designing better gradients—the so-called "acting on gradients in the name of loss."

So, what are the criteria for a "better" gradient? How can we design better gradients? In this article, we will understand multi-task learning from the perspective of gradients and attempt to construct multi-task learning algorithms directly from the idea of designing gradients.

1 Overall Idea

We know that for single-task learning, the commonly used optimization method is gradient descent. How is it derived? Can the same logic be directly applied to multi-task learning? This is the question this section aims to answer.

1.1 Descent Direction

In fact, we answered the first question in "Optimization Algorithms from a Dynamics Perspective (III): A More Holistic View". Suppose the loss function is $\mathcal{L}$ and the current parameter is $\boldsymbol{\theta}$. We want to design a parameter increment $\Delta\boldsymbol{\theta}$ that makes the loss function smaller, i.e., $\mathcal{L}(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) < \mathcal{L}(\boldsymbol{\theta})$. To this end, we consider the first-order Taylor expansion:

$$\mathcal{L}(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) \approx \mathcal{L}(\boldsymbol{\theta}) + \langle \nabla_{\boldsymbol{\theta}}\mathcal{L}, \Delta\boldsymbol{\theta}\rangle \tag{1}$$

Assuming the accuracy of this approximation is sufficient, $\mathcal{L}(\boldsymbol{\theta}+\Delta\boldsymbol{\theta}) < \mathcal{L}(\boldsymbol{\theta})$ implies $\langle \nabla_{\boldsymbol{\theta}}\mathcal{L}, \Delta\boldsymbol{\theta}\rangle < 0$. This means the angle between the update amount and the gradient must be at least greater than 90 degrees. The most natural choice is:

$$\Delta\boldsymbol{\theta} = -\eta\nabla_{\boldsymbol{\theta}}\mathcal{L} \tag{2}$$

This is gradient descent, where the update is taken in the opposite direction of the gradient, and $\eta > 0$ is the learning rate.

1.2 No Exceptions

Returning to multi-task learning, if we assume each task is equally important, we can interpret this assumption as requiring $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_n$ to all decrease or remain unchanged at each update step. If, after the parameters reach $\boldsymbol{\theta}^*$, any further change leads to an increase in at least one $\mathcal{L}_i$, then $\boldsymbol{\theta}^*$ is said to be a Pareto Optimal solution. Simply put, Pareto optimality means

we cannot improve one task by sacrificing another; it means there is no "internal competition" (involution) between tasks.

Assuming the approximation (1) still holds, seeking Pareto optimality means we need to find $\Delta\boldsymbol{\theta}$ such that:

$$\begin{cases} \langle \nabla_{\boldsymbol{\theta}} \mathcal{L}_1, \Delta\boldsymbol{\theta} \rangle \leq 0 \\ \langle \nabla_{\boldsymbol{\theta}} \mathcal{L}_2, \Delta\boldsymbol{\theta} \rangle \leq 0 \\ \quad \vdots \\ \langle \nabla_{\boldsymbol{\theta}} \mathcal{L}_n, \Delta\boldsymbol{\theta} \rangle \leq 0 \end{cases} \tag{3}$$

Note that a trivial solution $\Delta\boldsymbol{\theta} = \mathbf{0}$ exists, so the feasible region of the above system of inequalities is certainly non-empty. We are primarily concerned with whether non-zero solutions exist in the feasible region. If they do, we find one to use as the update direction; if not, we may have reached Pareto optimality (necessary but not sufficient), and we call this state a Pareto Stationary point.

2 Solving Algorithms

For convenience, we denote $\boldsymbol{g}_i = \nabla_{\boldsymbol{\theta}} \mathcal{L}_i$. We seek a vector $\boldsymbol{u}$ such that $\langle \boldsymbol{g}_i, \boldsymbol{u} \rangle \geq 0$ for all i. Then we can take $\Delta\boldsymbol{\theta} = -\eta \boldsymbol{u}$ as the update amount, similar to single-task gradient descent. If there are only two tasks, it can be verified that $\boldsymbol{u} = \boldsymbol{g}_1/\|\boldsymbol{g}_1\| + \boldsymbol{g}_2/\|\boldsymbol{g}_2\|$ automatically satisfies $\langle \boldsymbol{g}_1, \boldsymbol{u} \rangle \geq 0$ and $\langle \boldsymbol{g}_2, \boldsymbol{u} \rangle \geq 0$. That is to say, in dual-task learning, the previously mentioned gradient normalization can reach a Pareto stationary point.

When the number of tasks is greater than 2, the problem becomes more complex. Here we introduce two solving methods. The first is a derivation provided by the author, and the second is the "standard answer" given in "Multi-Task Learning as Multi-Objective Optimization".

2.1 Problem Transformation

First, we further transform the problem. Note that:

$$\forall i, \langle \boldsymbol{g}_i, \boldsymbol{u} \rangle \geq 0 \quad \Leftrightarrow \quad \min_i \langle \boldsymbol{g}_i, \boldsymbol{u} \rangle \geq 0 \tag{4}$$

Therefore, we only need to maximize the minimum $\langle \boldsymbol{g}_i, \boldsymbol{u} \rangle$ as much as possible to find the ideal $\boldsymbol{u}$. The problem becomes:

$$\max_{\boldsymbol{u}} \min_i \langle \boldsymbol{g}_i, \boldsymbol{u} \rangle \tag{5}$$

However, this is somewhat dangerous because if a non-zero $\boldsymbol{u}$ exists such that $\min_i \langle \boldsymbol{g}_i, \boldsymbol{u} \rangle > 0$, then letting the norm of $\boldsymbol{u}$ tend to infinity would cause the maximum value to tend to infinity. For stability, we add a regularization term:

$$\max_{\boldsymbol{u}} \min_i \langle \boldsymbol{g}_i, \boldsymbol{u} \rangle - \frac{1}{2}\|\boldsymbol{u}\|^2 \tag{6}$$

In this way, a $\boldsymbol{u}$ with an infinite norm cannot be the optimal solution. Note that substituting $\boldsymbol{u} = 0$ yields $\min_i \langle \boldsymbol{g}_i, \boldsymbol{u} \rangle - \frac{1}{2}\|\boldsymbol{u}\|^2 = 0$. Thus, if the optimal solution for max is $\boldsymbol{u}^*$, it must satisfy:

$$\min_i \langle \boldsymbol{g}_i, \boldsymbol{u}^* \rangle - \frac{1}{2}\|\boldsymbol{u}^*\|^2 \geq 0 \quad \Leftrightarrow \quad \min_i \langle \boldsymbol{g}_i, \boldsymbol{u}^* \rangle \geq \frac{1}{2}\|\boldsymbol{u}^*\|^2 \geq 0 \tag{7}$$

So the solution to problem (6) must satisfy condition (4), and if it is a non-zero solution, its opposite direction must be a direction that decreases the loss of all tasks.

2.2 Smooth Approximation

Now we introduce the first solution for problem (6). It assumes the reader, like the author, is unfamiliar with solving min-max problems. We can replace the min in the first step with a smooth approximation (refer to "Seeking a Smooth Maximum Function"):

$$\min_i \langle \boldsymbol{g}_i, \boldsymbol{u}\rangle \approx -\frac{1}{\lambda}\log\sum_i e^{-\lambda\langle \boldsymbol{g}_i, \boldsymbol{u}\rangle} \quad \text{(for sufficiently large } \lambda\text{)} \tag{8}$$

Thus, we can first solve:

$$\max_{\boldsymbol{u}} -\frac{1}{\lambda}\log\sum_i e^{-\lambda\langle \boldsymbol{g}_i, \boldsymbol{u}\rangle} - \frac{1}{2}\|\boldsymbol{u}\|^2 \tag{9}$$

And then let $\lambda \to \infty$. This transforms the problem into an unconstrained maximization of a single function. Taking the gradient and setting it to zero gives:

$$\frac{\sum\limits_i e^{-\lambda\langle \boldsymbol{g}_i, \boldsymbol{u}\rangle}\boldsymbol{g}_i}{\sum\limits_i e^{-\lambda\langle \boldsymbol{g}_i, \boldsymbol{u}\rangle}} = \boldsymbol{u} \tag{10}$$

Assuming the differences between various $\langle \boldsymbol{g}_i, \boldsymbol{u}\rangle$ are larger than the order of $\mathcal{O}(1/\lambda)$, then as $\lambda \to \infty$, the above equation is effectively:

$$\boldsymbol{u} = \boldsymbol{g}_\tau, \quad \tau = \operatorname*{argmin}_i \langle \boldsymbol{g}_i, \boldsymbol{u}\rangle \tag{11}$$

However, if we iterate directly according to $\boldsymbol{u}^{(k+1)} = \boldsymbol{g}_\tau, \tau = \operatorname*{argmin}_i \langle \boldsymbol{g}_i, \boldsymbol{u}^{(k)}\rangle$, it will likely oscillate. This is because it asks us to find the $\boldsymbol{g}_i$ that minimizes $\langle \boldsymbol{g}_i, \boldsymbol{u}^{(k)}\rangle$ and use it as $\boldsymbol{u}^{(k+1)}$. If $\boldsymbol{u}^{(k+1)} = \boldsymbol{g}_{i^*}$, then in the next step, the $\boldsymbol{g}_i$ that minimizes $\langle \boldsymbol{g}_i, \boldsymbol{u}^{(k+1)}\rangle = \langle \boldsymbol{g}_i, \boldsymbol{g}_{i^*}\rangle$ is very likely no longer $\boldsymbol{g}_{i^*}$; in fact, $\boldsymbol{g}_{i^*}$ might be the largest one.

Intuitively, although the algorithm oscillates, it should oscillate around the optimal point $\boldsymbol{u}^*$. Therefore, if we average all results during the oscillation process, we should obtain the optimal point. This means the iteration format converging to the optimal point is:

$$\boldsymbol{u}^{(k+1)} = \frac{k\boldsymbol{u}^{(k)} + \boldsymbol{g}_\tau}{k+1}, \quad \tau = \operatorname*{argmin}_i \langle \boldsymbol{g}_i, \boldsymbol{u}^{(k)}\rangle \tag{12}$$

Note that since each added term is some $\boldsymbol{g}_i$, the final $\boldsymbol{u}^*$ must be a weighted average of the $\boldsymbol{g}_i$'s. That is, there exist $\alpha_1, \alpha_2, \ldots, \alpha_n \geq 0$ and $\alpha_1 + \alpha_2 + \cdots + \alpha_n = 1$ such that:

$$\boldsymbol{u}^* = \sum_i \alpha_i \boldsymbol{g}_i \tag{13}$$

We can also interpret $\alpha_1, \alpha_2, \ldots, \alpha_n$ as the current optimal weight distribution for each $\mathcal{L}_i$.

2.3 Dual Problem

The advantage of the smooth approximation technique is that it is simple and intuitive, requiring little background in optimization algorithms. However, it is ultimately a "non-mainstream" approach with some lack of rigor (though the result is correct). Next, we introduce the "standard answer" based on the idea of duality.

First, define $\mathbb{P}^n$ as the set of all n-dimensional discrete distributions:

$$\mathbb{P}^n = \left\{(\alpha_1, \alpha_2, \ldots, \alpha_n)\,\middle|\, \alpha_1, \alpha_2, \ldots, \alpha_n \geq 0, \sum_i \alpha_i = 1\right\} \tag{14}$$

Then it is easy to verify:

$$\min_i \langle \boldsymbol{g}_i, \boldsymbol{u} \rangle = \min_{\alpha \in \mathbb{P}^n} \langle \tilde{\boldsymbol{g}}(\alpha), \boldsymbol{u} \rangle, \quad \tilde{\boldsymbol{g}}(\alpha) = \sum_i \alpha_i \boldsymbol{g}_i \tag{15}$$

Thus, problem (6) is equivalent to:

$$\max_{\boldsymbol{u}} \min_{\alpha \in \mathbb{P}^n} \langle \tilde{\boldsymbol{g}}(\alpha), \boldsymbol{u} \rangle - \frac{1}{2}\|\boldsymbol{u}\|^2 \tag{16}$$

The above function is concave with respect to $\boldsymbol{u}$ and convex with respect to α, and the feasible regions for $\boldsymbol{u}$ and α are both convex sets. Therefore, according to von Neumann's Minimax Theorem, the min and max in (16) can be swapped:

$$\min_{\alpha \in \mathbb{P}^n} \max_{\boldsymbol{u}} \langle \tilde{\boldsymbol{g}}(\alpha), \boldsymbol{u} \rangle - \frac{1}{2}\|\boldsymbol{u}\|^2 = \min_{\alpha \in \mathbb{P}^n} \frac{1}{2}\|\tilde{\boldsymbol{g}}(\alpha)\|^2 \tag{17}$$

The right side of the equation follows because the max part is just an unconstrained maximum problem for a quadratic function, which yields $\boldsymbol{u}^* = \tilde{\boldsymbol{g}}(\alpha)$. Finally, only the min remains, and the problem becomes finding a weighted average of $\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_n$ that minimizes its norm.

When $n = 2$, the solution is relatively simple, equivalent to constructing the altitude of a triangle, as shown below:

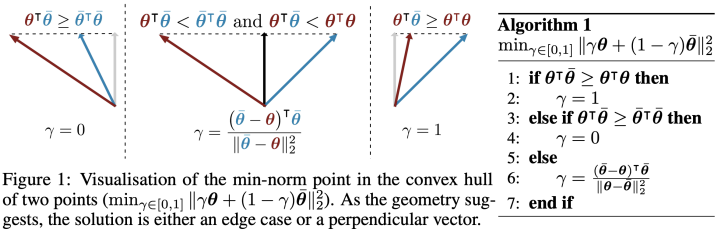

Figure 1: Visualisation of the min-norm point in the convex hull of two points ($\min_{\gamma\in[0,1]} \|\gamma\boldsymbol{\theta} + (1-\gamma)\bar{\boldsymbol{\theta}}\|_2^2$). As the geometry suggests, the solution is either an edge case or a perpendicular vector.

Figure 1: Solving algorithm and geometric meaning when $n = 2$

When $n > 2$, we can use the Frank-Wolfe algorithm to transform it into multiple $n = 2$ cases for iteration. The Frank-Wolfe algorithm can be understood as a constrained gradient descent algorithm suitable for cases where the parameter's feasible region is a convex set. Explaining it in detail would take too much space, so I won't elaborate here; readers are encouraged to find materials to study it. Briefly, the Frank-Wolfe algorithm first linearizes the objective to find the next update direction e_τ, where $\tau = \operatorname{argmin}_i \langle \boldsymbol{g}_i, \tilde{\boldsymbol{g}}(\alpha) \rangle$ and e_τ is a one-hot vector with 1 at position τ. Then it performs an interpolation search between α and e_τ to find the optimal result. The iteration process is:

$$\begin{cases} \tau = \operatorname{argmin}_i \langle \boldsymbol{g}_i, \tilde{\boldsymbol{g}}(\alpha^{(k)}) \rangle \\ \gamma = \operatorname{argmin}_\gamma \|\tilde{\boldsymbol{g}}((1-\gamma)\alpha^{(k)} + \gamma e_\tau)\|^2 = \operatorname{argmin}_\gamma \|(1-\gamma)\tilde{\boldsymbol{g}}(\alpha^{(k)}) + \gamma \boldsymbol{g}_\tau\|^2 \\ \alpha^{(k+1)} = (1-\gamma)\alpha^{(k)} + \gamma e_\tau \end{cases} \tag{18}$$

The solution for γ is exactly the $n = 2$ special case, which can be solved using the algorithm in the screenshot above. If γ is not obtained through search but fixed at $1/(k+1)$, the result is equivalent to (12), which is a simplified version of the Frank-Wolfe algorithm. In other words, the result obtained through smooth approximation is equivalent to the result of the simplified Frank-Wolfe algorithm.

2.4 De-constraint

In fact, for the solution of problem (17), we could theoretically solve it directly via gradient descent by removing constraints. For example, we can set parameters $\beta_1, \beta_2, \ldots, \beta_n \in \mathbb{R}$ and:

$$\alpha_i = \frac{e^{\beta_i}}{Z}, \quad Z = \sum_i e^{\beta_i} \tag{19}$$

Then it can be transformed into:

$$\min_{\beta} \frac{1}{2Z^2} \left\| \sum_i e^{\beta_i} \boldsymbol{g}_i \right\|^2 \tag{20}$$

This is an unconstrained optimization problem that can be solved with conventional gradient descent. However, for some reason, the author has not seen this approach used (perhaps to avoid tuning the learning rate?).

3 Some Techniques

In the previous section, we provided two schemes for finding the update direction of a Pareto stationary point. Both require us to perform additional multiple iterations to determine the weight of each task at every step of training before updating the model parameters. As one can imagine, the computational cost is quite high in practice, so we need some techniques to reduce it.

3.1 Gradient Inner Product

As seen, in both schemes, the key step is $\operatorname{argmin}_i \langle \boldsymbol{g}_i, \tilde{\boldsymbol{g}}(\alpha) \rangle$, which means we must traverse gradients to calculate inner products. However, in deep learning scenarios, model parameters are often very large, so the gradient is a very high-dimensional vector. Calculating an inner product at every iteration is computationally expensive. We can utilize the expansion:

$$\langle \boldsymbol{g}_i, \tilde{\boldsymbol{g}}(\alpha) \rangle = \left\langle \boldsymbol{g}_i, \sum_j \alpha_j \boldsymbol{g}_j \right\rangle = \sum_j \alpha_j \langle \boldsymbol{g}_i, \boldsymbol{g}_j \rangle \tag{21}$$

In each iteration, only α changes. Therefore, $\langle \boldsymbol{g}_i, \boldsymbol{g}_j \rangle$ only needs to be calculated once and stored for each training step, avoiding repeated high-dimensional vector inner product calculations.

3.2 Shared Encoding

However, when the model is large enough, it is difficult to calculate the gradient for each task separately and then perform iterative calculations. If we assume that the various tasks share the same encoder, we can further simplify the algorithm with an approximation.

Specifically, assume the batch size is b, and the encoder output for the j-th sample is $\boldsymbol{h}_j$. From the chain rule, we know:

$$\boldsymbol{g}_i = \nabla_{\boldsymbol{\theta}} \mathcal{L}_i = \sum_j (\nabla_{\boldsymbol{h}_j} \mathcal{L}_i)(\nabla_{\boldsymbol{\theta}} \boldsymbol{h}_j) = \underbrace{(\nabla_{\boldsymbol{h}_1} \mathcal{L}_i, \ldots, \nabla_{\boldsymbol{h}_b} \mathcal{L}_i)}_{\nabla_{\boldsymbol{H}} \mathcal{L}_i} \underbrace{\begin{pmatrix} \nabla_{\boldsymbol{\theta}} \boldsymbol{h}_1 \\ \vdots \\ \nabla_{\boldsymbol{\theta}} \boldsymbol{h}_b \end{pmatrix}}_{\nabla_{\boldsymbol{\theta}} \boldsymbol{H}} \tag{22}$$

Letting $\boldsymbol{H} = (\boldsymbol{h}_1, \ldots, \boldsymbol{h}_b)$, we get $\boldsymbol{g}_i = (\nabla_{\boldsymbol{H}} \mathcal{L}_i)(\nabla_{\boldsymbol{\theta}} \boldsymbol{H})$. Using the matrix norm inequality:

$$\left\| \sum_i \alpha_i \boldsymbol{g}_i \right\|^2 = \left\| \sum_i \alpha_i (\nabla_{\boldsymbol{H}} \mathcal{L}_i)(\nabla_{\boldsymbol{\theta}} \boldsymbol{H}) \right\|^2 \leq \left\| \sum_i \alpha_i \nabla_{\boldsymbol{H}} \mathcal{L}_i \right\|^2 \|\nabla_{\boldsymbol{\theta}} \boldsymbol{H}\|^2 \tag{23}$$

Naturally, if we minimize $\left\|\sum_i \alpha_i \nabla_{\boldsymbol{H}} \mathcal{L}_i\right\|^2$, the computational cost will be significantly reduced because this only requires the gradient of the final output encoding vector, not the gradients of all parameters. The above formula tells us that minimizing $\left\|\sum_i \alpha_i \nabla_{\boldsymbol{H}} \mathcal{L}_i\right\|^2$ is actually minimizing the upper bound of equation (17). Like many problems that are difficult to optimize directly, we hope that minimizing the upper bound will yield similar results.

However, while this upper bound is more efficient, it has limitations. It generally only applies to multi-task learning where each sample has multiple types of labels. it is not suitable for scenarios where training data for different tasks have no intersection (i.e., each task is labeled on different samples, and a single sample has only one type of label). In the latter case, the various $\nabla_{\boldsymbol{H}} \mathcal{L}_i$ are mutually orthogonal, meaning there is no interaction between tasks, and the upper bound becomes too loose to be meaningful.

3.3 Error in Proof

The "standard answer" mentioned earlier and the optimization of the upper bound for shared encoders both come from the paper "Multi-Task Learning as Multi-Objective Optimization". The original paper then attempts to prove that when $\nabla_{\boldsymbol{\theta}} \boldsymbol{H}$ is full rank, optimizing the upper bound can also find a Pareto stationary point. Unfortunately, the proof in the original paper is incorrect.

The proof is located in Appendix A of the original paper, where it uses an incorrect conclusion:

> If $\boldsymbol{M}$ is a symmetric positive definite matrix, then $\boldsymbol{x}^\top \boldsymbol{y} \geq 0$ if and only if $\boldsymbol{x}^\top \boldsymbol{M} \boldsymbol{y} \geq 0$.

It is easy to provide a counterexample to prove this conclusion is wrong. For instance, let $\boldsymbol{x} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}, \boldsymbol{y} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \boldsymbol{M} = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}$. Here $\boldsymbol{x}^\top \boldsymbol{y} < 0$ but $\boldsymbol{x}^\top \boldsymbol{M} \boldsymbol{y} > 0$.

After reflection, the author believes the proof in the original paper is difficult to fix, meaning the original paper's conjecture does not hold. In other words, even if $\nabla_{\boldsymbol{\theta}} \boldsymbol{H}$ is full rank, the update direction derived from the upper bound may not necessarily be a direction that prevents all task losses from rising, and thus may not find a Pareto stationary point. As for why the experimental results of the upper bound optimization in the original paper are good, it can only be said that the parameter space of deep learning models is so large that there is plenty of "room to maneuver," allowing the upper bound approximation to achieve decent results.

4 Summary

In this article, we understood multi-task learning from the perspective of gradients. From this viewpoint, the main task of multi-task learning is to find an update direction that is as opposite as possible to the gradient of each task, so that the loss of each task can decrease as much as possible, without sacrificing one task for the sake of another. This is the ideal state of no "internal competition" between tasks.

Original Address: *https://kexue.fm/archives/8896*