Learning to Reweight and Bilevel Optimization

Sixu Li

1 Preliminaries

Let (x, y) be an input-target pair, and $\{(x_i, y_i), 1 \leq i \leq N\}$ be the training set. We assume that there is a small unbiased and clean validation set $\{(x_i^v, y_i^v), 1 \leq i \leq M\}$, and $M \ll N$. Denote $f(x, \theta)$ as our neural network model and θ be the model parameters. We consider a loss function $\mathcal{L}(f(x, \theta), y)$ to minimize during training.

2 Learning to Reweight

As defined in [4, 6], we can write their objective problem into a bilevel optimization framework,

$$\min_{\omega} \frac{1}{m} \sum_{j=1}^{m} \mathcal{L}(f(x_j^v, \theta(\omega)), y_j^v) \tag{1}$$

$$\text{s.t.} \quad \min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \omega_i \mathcal{L}(f(x_i, \theta), y_i) \tag{2}$$

Denote $F(\omega, \theta(\omega)) \triangleq \frac{1}{m} \sum_{j=1}^{m} \mathcal{L}(f(x_j^v, \theta(\omega)), y_j^v)$. And they do the $t+1$ steps updating as follows,

$$\hat{\theta}_{t+1} = \theta_t - \frac{\alpha}{n} \sum_{i=1}^{n} \omega_{i,t} \nabla_\theta \mathcal{L}(f(x_i, \theta), y_i) \tag{3}$$

$\hat{\theta}_{t+1}$ is regarded as the approximate local optimum of lower-level problem in t steps and then use this $\hat{\theta}_{t+1}$ to update $\omega_{i,t}$.
In [4], they update $\omega_{i,t}$ by,

$$\begin{aligned} \omega_{i,t+1} &= -\eta \frac{\partial F(\omega, \hat{\theta}_{t+1}(\omega))}{\partial \omega_{i,t}} \\ &= -\eta \frac{\partial F(\omega, \hat{\theta}_{t+1}(\omega))}{\partial \theta} \frac{\partial \hat{\theta}_{t+1}(\omega)}{\partial \omega_{i,t}} \\ &\propto \sum_{j=1}^{m} \nabla_\theta \mathcal{L}(f(x_j^v, \theta_t), y_j^v)^T \nabla_\theta \mathcal{L}(f(x_i, \theta_t), y_i) \end{aligned} \tag{4}$$

In [6], they update $\omega_{i,t}$ with,

$$\omega_{i,t+1} = \omega_{i,t} - \eta \frac{\partial F(\omega, \hat{\theta}_{t+1}(\omega))}{\partial \omega_{i,t}} \tag{5}$$

And given $\omega_{i,t+1}$ in $t+1$ step, they updating θ_t by,

$$\theta_{t+1} = \theta_t - \frac{\alpha}{n} \nabla_\theta \big(\sum_{i=1}^{n} \omega_{i,t+1} \mathcal{L}(f(x_i, \theta_t), y_i) \big) \tag{6}$$

3 Challenges of Deep Bilevel Optimization

We just consider the unconstrained bilevel optimization as follows,

$$\min_u f(u, v) \tag{7}$$

$$\text{s.t.} \quad \min_v g(u, v) \tag{8}$$

Assuming we can express the solution to the lower-level problem $v^*(u) = \arg\min_v g(u, v)$ explicitly, we can write the bilevel problem as an equivalent single-level problem as,

$$\min_u f(u, v^*(u)) \tag{9}$$

Then we can use gradient-based approach on this single-level problem and compute the total derivative $\frac{df}{du}(u, v^*(u))$, called the hypergradient and it can be writen as,

$$\frac{df}{du} = \nabla_u f + \frac{dv}{du} \nabla_v f \tag{10}$$

Usually, $\nabla_u f$ and $\nabla_v f$ are easy to calculate. Hence, the key problem now is how to obtain $\frac{dv}{du}$.

4 Methods for Solving Deep Bilevel Optimization

4.1 Approxiamate Hypergradient

4.1.1 Based on Implicit Function Theorem

Since $\nabla_v g = 0$ at $v = v^*(u)$, by implicit function theorem, we have $\frac{dv}{du} = -\nabla^2_{uv} g (\nabla^2_{vv} g)^{-1}$. Thus we can write the hypergradient as,

$$\frac{df}{du} = \nabla_u f - \nabla^2_{uv} g (\nabla^2_{vv} g)^{-1} \nabla_v f \tag{11}$$

However, for recent deep learining models, with millions of parameters, it is nearly impossible to calculate the inverse Hessian to get the exact hypergradient.
Hence, in [3], they propose δ–approximate, i.e., let $\boldsymbol{w}$ be a vector such that

$$\|\boldsymbol{w} - \nabla^2_{uv} g(\nabla^2_{vv} g)^{-1}|_{v=v^*(u)}\| \leq \delta \tag{12}$$

And $\boldsymbol{w}$ can be obtained as an approximate solution to the optimization problem:

$$\min_{w} \quad \frac{1}{2}\boldsymbol{w}^T(\nabla^2_{vv} g)\boldsymbol{w} - \boldsymbol{w}^T\nabla^2_{uv} g \tag{13}$$

The conjugate gradient(CG) method is particularly well suited for this problem.
However, when directly applying this method to lower-level problem (2), it will bring some troubles. Because, our goal is to train a robust network with noisy data, however, if we directly minimize problem (2), especially at begining of the training that the weight parameters ω_i's are not correct, it might lead networks to learn the wrong pattern.

4.1.2 Based on Reverse-Mode and Forward-Mode

In [1], they treat the iterative optimization algorithm that solves the lower-level problem as a dynamical systerm. Given an initial condition $v_0 = \Phi_{t+1}(u)$ at $t = 0$, the update rule can be written as,

$$v_{t+1} = \Phi_{t+1}(v_t, u) \tag{14}$$

in which Φ_t defines the transition function and T is the number of iterations performed. For example, in gradient descent,

$$\Phi_{t+1}(v_t, u) = v_t - \alpha\nabla_u g(v_t, u) \tag{15}$$

And we denote that $\hat{v}^* = v_T$.
By unrolling the iterative update scheme (14) as a computational graph, we can view $\hat{v}^*$ as a function of u and compute the required derivative $\frac{df}{du}$. Specifically, it can be shown by the chain rule,

$$\frac{df}{du} = \nabla_u f + \sum_{t=0}^{T} B_t A_{t+1} \cdots A_T \nabla_{\hat{v}^*} f \tag{16}$$

where $A_{t+1} = \nabla_{v_t}\Phi_{t+1}(v_t, u)$, $B_{t+1} = \nabla_u \Phi_{t+1}(v_t, u)$ for $t \geq 0$ and $B_0 = d_u\Phi_0(u)$. And this computation can be implemented either in reverse mode or forward mode.
In [5], they perform K–step truncated back–propagation(K–RMD) and use the intermediate variable h_{T-K} to approximate $\frac{df}{du}$:

$$h_{T-K} = \nabla_u f + \sum_{t=T-K+1}^{T} B_t A_{t+1} \cdots A_T \nabla_{\hat{v}^*} f \tag{17}$$

This approach requires storing only the last K iterates v_t, and it also saves computation time.

4.2 Penalty Method

In [2], they propose to use penalty method to solve deep bilevel optimization.
With assumption that function $g(u, v)$ is strongly convex, we can replace the lower-level problem by the first-order necessary condition and resulting in the following problem:

$$\min_{u,v} f(u, v) \quad \text{s.t. } \nabla_v g(u, v) = 0 \tag{18}$$

Then, we can write penalty function as,

$$\widetilde{f}(u, v; \gamma) \triangleq f(u, v) + \frac{\gamma}{2}\|\nabla_v g(u, v)\|^2 \tag{19}$$

Let $(\hat{u}_k, \hat{v}_k)$ be the minimum of the penalty function $\widetilde{f}$ for a given γ_k,

$$(\hat{u}_k, \hat{v}_k) = \arg\min_{u,v} f(u, v) + \frac{\gamma_k}{2}\|\nabla_v g(u, v)\|^2 \tag{20}$$

My Proposition: I think we could use Augmented Lagrangian to replace Penalty Method. For Problem (18), we can write augmented Lagrangian function as,

$$\mathcal{L}_A(u, v; \lambda, \mu) = f(u, v) - \lambda \nabla_v g(u, v) + \frac{\mu}{2}(\nabla_v g(u, v))^2 \tag{21}$$

References

[1] Franceschi, L., Donini, M., Frasconi, P., and Pontil, M. Forward and reverse gradient-based hyperparameter optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (2017), JMLR. org, pp. 1165–1173.

[2] Mehra, A., and Hamm, J. Penalty method for inversion-free deep bilevel optimization. *arXiv preprint arXiv:1911.03432* (2019).

[3] Rajeswaran, A., Finn, C., Kakade, S. M., and Levine, S. Meta-learning with implicit gradients. In *Advances in Neural Information Processing Systems* (2019), pp. 113–124.

[4] Ren, M., Zeng, W., Yang, B., and Urtasun, R. Learning to reweight examples for robust deep learning. *arXiv preprint arXiv:1803.09050* (2018).

[5] Shaban, A., Cheng, C.-A., Hatch, N., and Boots, B. Truncated back-propagation for bilevel optimization. *arXiv preprint arXiv:1810.10667* (2018).

[6] Shu, J., Xie, Q., Yi, L., Zhao, Q., Zhou, S., Xu, Z., and Meng, D. Meta-weight-net: Learning an explicit mapping for sample weighting. In *Advances in Neural Information Processing Systems* (2019), pp. 1917–1928.