Недорогая градиентная оптимизация

Виктор Панкратов

06/12/2021г

Мотивация

Описание задачи

Одной из подзадач при работе с нейронными сетями является выбор или оптимизация гиперпараметров. Существуют техники для этого, когда размерность небольшая (отдельно выделяют < 5 и от 5 до порядка 100),но для больших получить решение сложнее.

Проблема

Веса модели - функции от гиперпараметров. Таким образом оптимизация гиперпараметров сводится к вычислениям якобиана и это слишком сложно для вычислений.

Основные обозначения

$\mathcal{L}_{\mathrm{T}}, \mathcal{L}_{\mathrm{V}}$ – train и validation losses
w – веса сети
λ – гиперпараметры
$\lambda^* = \underset{\lambda}{\arg\min}\ \mathcal{L}_{\mathrm{V}}^*(\lambda)$, где
$\mathcal{L}_{\mathrm{V}}^*(\lambda) = \mathcal{L}_{\mathrm{V}}(\lambda, \mathrm{w}^*(\lambda))$ и $\mathrm{w}^*(\lambda) = \underset{\mathrm{w}}{\arg\min}\ \mathcal{L}_{\mathrm{T}}(\lambda, \mathrm{w})$

Гиперградиент

$$\frac{\partial \mathcal{L}_\mathrm{V}^*(\lambda)}{\partial \lambda} = \left(\frac{\partial \mathcal{L}_\mathrm{V}}{\partial \lambda} + \frac{\partial \mathcal{L}_\mathrm{V}}{\partial \mathrm{w}^*} \frac{\partial \mathrm{w}^*}{\partial \lambda} \right) \Bigg|_{\lambda, \mathrm{w}^*(\lambda)} =$$

$$= \frac{\partial \mathcal{L}_\mathrm{V}(\lambda, \mathrm{w}^*(\lambda))}{\partial \lambda} + \frac{\partial \mathcal{L}_\mathrm{V}(\lambda, \mathrm{w}^*(\lambda))}{\partial \mathrm{w}^*(\lambda)} \times \frac{\partial \mathrm{w}^*(\lambda)}{\partial \lambda}$$

- $\frac{\partial \mathcal{L}_\mathrm{V}(\lambda, \mathrm{w}^*(\lambda))}{\partial \lambda}$ – прямая производная гиперпараметров
- Легко считать, но часто равна 0 - приходится обращаться к другому слагаемому
- $\frac{\partial \mathcal{L}_\mathrm{V}(\lambda, \mathrm{w}^*(\lambda))}{\partial \mathrm{w}^*(\lambda)} \times \frac{\partial \mathrm{w}^*(\lambda)}{\partial \lambda}$ – непрямая производная гиперпараметров
- $\frac{\partial \mathcal{L}_\mathrm{V}(\lambda, \mathrm{w}^*(\lambda))}{\partial \mathrm{w}^*(\lambda)}$ – прямая производная параметров
- $\frac{\partial \mathrm{w}^*(\lambda)}{\partial \lambda}$ – best response Jacobian

IFT

В предположении существования λ', w': $\frac{\partial \mathcal{L}_{\mathrm{T}}}{\partial \mathrm{w}}|_{\lambda', \mathrm{w}'} = 0$ в окрестности λ', w' выполнено

$$\frac{\partial \mathrm{w}^*}{\partial \lambda}\bigg|_{\lambda'} = -\left[\frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathrm{w} \partial \mathrm{w}^{\mathrm{T}}}\right]^{-1} \times \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathrm{w} \partial \lambda}\bigg|_{\lambda', \mathrm{w}^*(\lambda')}$$

Как считать обратный гессиан?

$$\left[\frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathrm{w} \partial \mathrm{w}^{\mathrm{T}}}\right]^{-1} = \lim_{\mathrm{i} \to \infty} \sum_{\mathrm{j}=0}^{\mathrm{i}} \left[\mathrm{I} - \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathrm{w} \partial \mathrm{w}^{\mathrm{T}}}\right]^{\mathrm{j}}$$

Основной алгоритм

Algorithm 1 Gradient-based HO for $\boldsymbol{\lambda}^*, \mathbf{w}^*(\boldsymbol{\lambda}^*)$

```
Initialize hyperparameters λ' and weights w'
while not converged do
    for k = 1 ... N do
        w' −= α · ∂L_T/∂w |_{λ',w'}
    λ' −= hypergradient(L_V, L_T, λ', w')
return λ', w'                          ▷ λ*, w*(λ*) from Eq.1
```

Algorithm 2 hypergradient($\mathcal{L}_V, \mathcal{L}_T, \boldsymbol{\lambda}', \mathbf{w}'$)

```
v_1 = ∂L_V/∂w |_{λ',w'}
v_2 = approxInverseHVP(v_1, ∂L_T/∂w)
v_3 = grad(∂L_T/∂λ, w, grad_outputs = v_2)
return ∂L_V/∂λ |_{λ',w'} − v_3          ▷ Return to Alg. 1
```

Algorithm 3 approxInverseHVP($\mathbf{v}, \mathbf{f}$): Neumann approximation of inverse-Hessian-vector product $\mathbf{v}[\frac{\partial \mathbf{f}}{\partial \mathbf{w}}]^{-1}$

```
Initialize sum p = v
for j = 1 ... i do
    v −= α · grad(f, w, grad_outputs = v)
    p −= v
return p                               ▷ Return to Alg. 2.
```

Классический подход

Algorithm 1 Stochastic gradient descent with momentum

input: initial $\mathbf{w}_1$, decays $\boldsymbol{\gamma}$, learning rates $\boldsymbol{\alpha}$, loss function $L(\mathbf{w}, \boldsymbol{\theta}, t)$
initialize $\mathbf{v}_1 = \mathbf{0}$
for $t = 1$ **to** T **do**
    $\mathbf{g}_t = \nabla_{\mathbf{w}} L(\mathbf{w}_t, \boldsymbol{\theta}, t)$ ▷ evaluate gradient
    $\mathbf{v}_{t+1} = \gamma_t \mathbf{v}_t - (1 - \gamma_t)\mathbf{g}_t$ ▷ update velocity
    $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t \mathbf{v}_t$ ▷ update position
end for
output trained parameters $\mathbf{w}_T$

Algorithm 2 Reverse-mode differentiation of SGD

input: $\mathbf{w}_T$, $\mathbf{v}_T$, $\boldsymbol{\gamma}$, $\boldsymbol{\alpha}$, train loss $L(\mathbf{w}, \boldsymbol{\theta}, t)$, loss $f(\mathbf{w})$
initialize $d\mathbf{v} = \mathbf{0}$, $d\boldsymbol{\theta} = \mathbf{0}$, $d\alpha_t = \mathbf{0}$, $d\gamma = \mathbf{0}$
initialize $d\mathbf{w} = \nabla_{\mathbf{w}} f(\mathbf{w}_T)$
for $t = T$ **counting down to** 1 **do**
    $d\alpha_t = d\mathbf{w}^{\mathsf{T}} \mathbf{v}_t$

$$\left.\begin{aligned} &\mathbf{w}_{t-1} = \mathbf{w}_t - \alpha_t \mathbf{v}_t \\ &\mathbf{g}_t = \nabla_{\mathbf{w}} L(\mathbf{w}_t, \boldsymbol{\theta}, t) \\ &\mathbf{v}_{t-1} = [\mathbf{v}_t + (1 - \gamma_t)\mathbf{g}_t]/\gamma_t \end{aligned}\right\} \text{exactly reverse gradient descent operations}$$

    $d\mathbf{v} = d\mathbf{v} + \alpha_t d\mathbf{w}$
    $d\gamma_t = d\mathbf{v}^{\mathsf{T}}(\mathbf{v}_t + \mathbf{g}_t)$
    $d\mathbf{w} = d\mathbf{w} - (1 - \gamma_t) d\mathbf{v} \nabla_{\mathbf{w}} \nabla_{\mathbf{w}} L(\mathbf{w}_t, \boldsymbol{\theta}, t)$
    $d\boldsymbol{\theta} = d\boldsymbol{\theta} - (1 - \gamma_t) d\mathbf{v} \nabla_{\boldsymbol{\theta}} \nabla_{\mathbf{w}} L(\mathbf{w}_t, \boldsymbol{\theta}, t)$
    $d\mathbf{v} = \gamma_t d\mathbf{v}$
end for
output gradient of $f(\mathbf{w}_T)$ w.r.t $\mathbf{w}_1$, $\mathbf{v}_1$, $\boldsymbol{\gamma}$, $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$

$$\mathrm{w}_{\mathrm{i+1}}(\lambda) = \mathrm{w}_{\mathrm{i}}(\lambda) - \alpha \frac{\partial \mathcal{L}_{\mathrm{T}}(\lambda, \mathrm{w}_{\mathrm{i}}(\lambda))}{\partial \mathrm{w}}$$

Lemma. *Given the recurrence from unrolling SGD optimization in Eq. 5, we have:*

$$\frac{\partial \mathbf{w}_{i+1}}{\partial \boldsymbol{\lambda}} = -\sum_{j \le i} \left[\prod_{k<j} I - \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \mathbf{w}^T} \Big|_{\boldsymbol{\lambda}, \mathbf{w}_{i-k}(\boldsymbol{\lambda})} \right] \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T} \Big|_{\boldsymbol{\lambda}, \mathbf{w}_{i-j}(\boldsymbol{\lambda})}$$

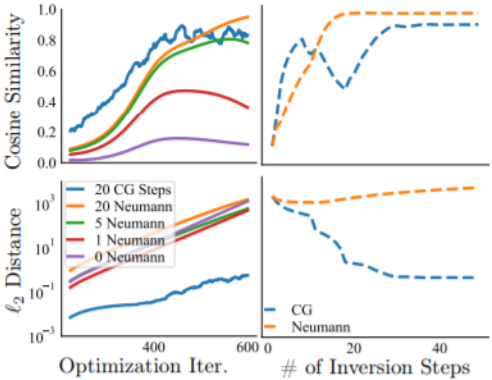

Figure 3: Comparing approximate hypergradients for inverse Hessian approximations to true hypergradients. The Neumann scheme often has greater cosine similarity than CG, but larger ℓ_2 distance for equal steps.

Method	Validation	Test	Time(s)
Grid Search	97.32	94.58	100k
Random Search	84.81	81.46	100k
Bayesian Opt.	72.13	69.29	100k
STN	70.30	67.68	25k
No HO	75.72	71.91	18.5k
Ours	69.22	66.40	18.5k
Ours, Many	**68.18**	**66.14**	18.5k

Table 4: Comparing HO methods for LSTM training on PTB. We tune millions of hyperparameters faster and with comparable memory to competitors tuning a handful. Our method competitively optimizes the same 7 hyperparameters as baselines from [26] (first four rows). We show a performance boost by tuning millions of hyperparameters, introduced with per-unit/weight dropout and DropConnect. "No HO" shows how the hyperparameter initialization affects training.