联邦学习：从idea构建到代码实战

—— TianLi三篇串讲

2025年6月28日

Overview

抛砖引玉，支持孙导的学术沙龙

内容介绍：本次从TianLi老师的FedProx、q-FFL、Ditto三篇文章入手，详细剖析这三篇文章的idea构建、论文写作手法、以及部分关键代码，以期对新入门的朋友们稍有启发。

个人简介：

- 昵称：丸一口
- 院校：华东师范大学 软件工程学院 硕士
- 研究方向：联邦学习、差分隐私
- 履历：
  - SMU科研助理
  - 运营BIlibili学术账号“丸一口”，目前3000+粉丝
  - 一篇C会、两篇泡池子；若干非一作paper

目录

第1章

FedProx

Li T, Sahu A K, Zaheer M, et al.

Federated optimization in heterogeneous networks[J].

Proceedings of Machine learning and systems, 2020, 2: 429-450.

Federated optimization in **heterogeneous** networks

(1) 系统异构性：指网络中每台设备的系统特性存在显著差异性

(2) 统计异质性：指网络中各节点数据呈非独立同分布状态

[1]

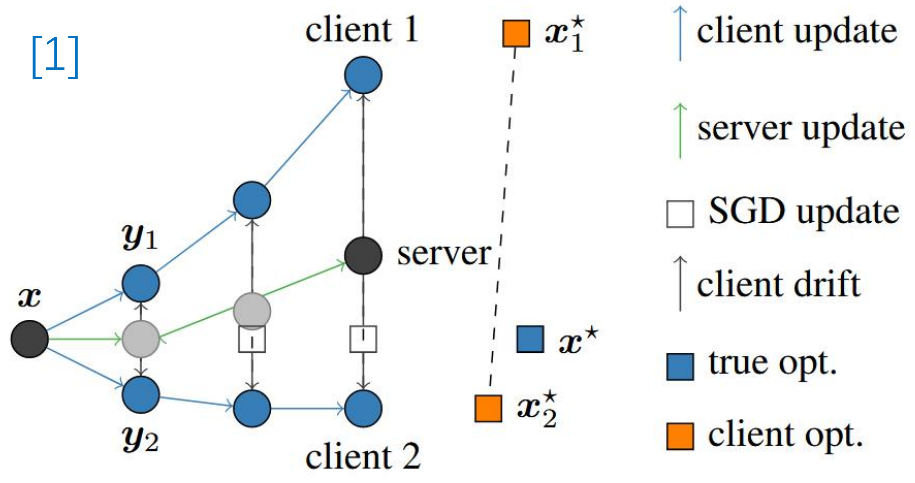


Figure 1. Client-drift in FEDAVG is illustrated for 2 clients with 3 local steps ($N = 2$, $K = 3$). The local updates $\boldsymbol{y}_i$ (in blue) move towards the individual client optima $\boldsymbol{x}_i^\star$ (orange square). The server updates (in red) move towards $\frac{1}{N}\sum_i \boldsymbol{x}_i^\star$ instead of to the true optimum $\boldsymbol{x}^\star$ (black square).

[1] SCAFFOLD: Stochastic Controlled Averaging for Federated Learning

[1]

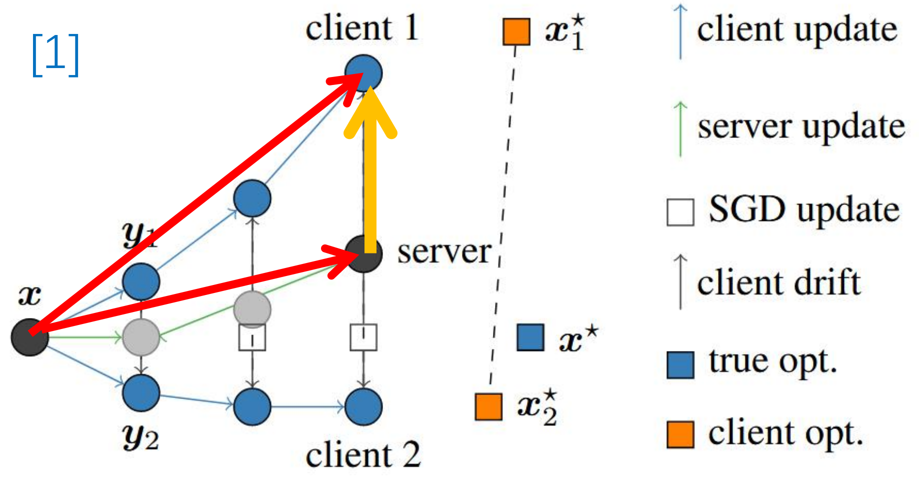
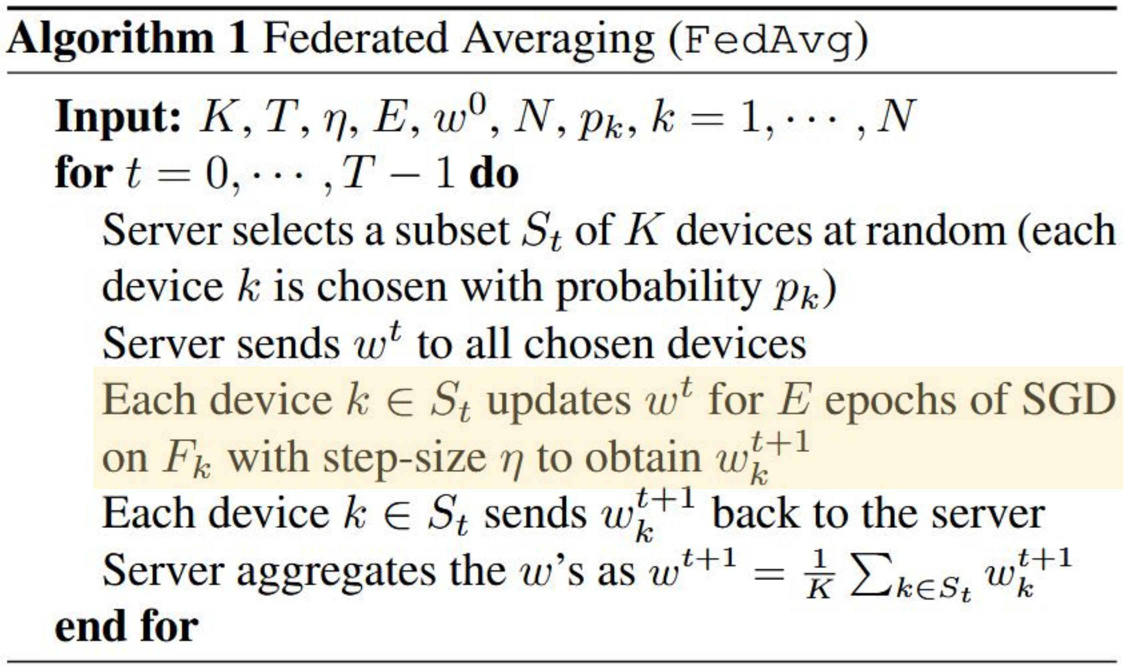


Figure 1. Client-drift in FedAvg is illustrated for 2 clients with 3 local steps ($N = 2$, $K = 3$). The local updates $\boldsymbol{y}_i$ (in blue) move towards the individual client optima $\boldsymbol{x}_i^\star$ (orange square). The server updates (in red) move towards $\frac{1}{N}\sum_i \boldsymbol{x}_i^\star$ instead of to the true optimum $\boldsymbol{x}^\star$ (black square).

$$\min_w h_k(w;\ w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2 . \qquad (2)$$

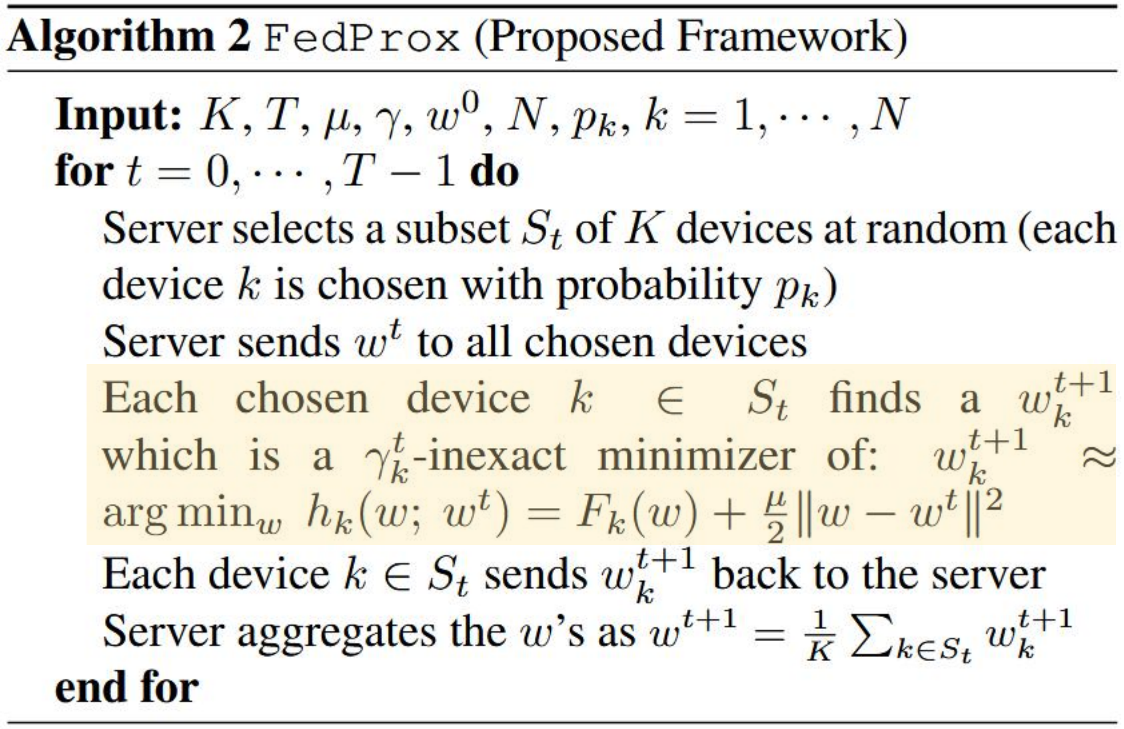
本地模型w - 全局模型wt：被减 指向 减

第二项的目标即：缩小橙色箭头向量的(L2)范数平方



Algorithm 1 Federated Averaging (`FedAvg`)

```
Input: $K, T, \eta, E, w^0, N, p_k, k = 1, \cdots, N$
for $t = 0, \cdots, T - 1$ do
    Server selects a subset $S_t$ of $K$ devices at random (each device $k$ is chosen with probability $p_k$)
    Server sends $w^t$ to all chosen devices
    Each device $k \in S_t$ updates $w^t$ for $E$ epochs of SGD on $F_k$ with step-size $\eta$ to obtain $w_k^{t+1}$
    Each device $k \in S_t$ sends $w_k^{t+1}$ back to the server
    Server aggregates the $w$'s as $w^{t+1} = \frac{1}{K} \sum_{k \in S_t} w_k^{t+1}$
end for
```



Algorithm 2 `FedProx` (Proposed Framework)

```
Input: $K, T, \mu, \gamma, w^0, N, p_k, k = 1, \cdots, N$
for $t = 0, \cdots, T - 1$ do
    Server selects a subset $S_t$ of $K$ devices at random (each device $k$ is chosen with probability $p_k$)
    Server sends $w^t$ to all chosen devices
    Each chosen device $k \in S_t$ finds a $w_k^{t+1}$ which is a $\gamma_k^t$-inexact minimizer of: $w_k^{t+1} \approx \arg\min_w \; h_k(w; \; w^t) = F_k(w) + \frac{\mu}{2} \|w - w^t\|^2$
    Each device $k \in S_t$ sends $w_k^{t+1}$ back to the server
    Server aggregates the $w$'s as $w^{t+1} = \frac{1}{K} \sum_{k \in S_t} w_k^{t+1}$
end for
```

非精确解

Definition 1 (γ-inexact solution). For a function $h(w; w_0) = F(w) + \frac{\mu}{2}\|w - w_0\|^2$, and $\gamma \in [0, 1]$, we say w^* is a γ-inexact solution of $\min_w h(w; w_0)$ if $\|\nabla h(w^*; w_0)\| \leq \gamma \|\nabla h(w_0; w_0)\|$, where $\nabla h(w; w_0) = \nabla F(w) + \mu(w - w_0)$. Note that a smaller γ corresponds to higher accuracy.

[1]

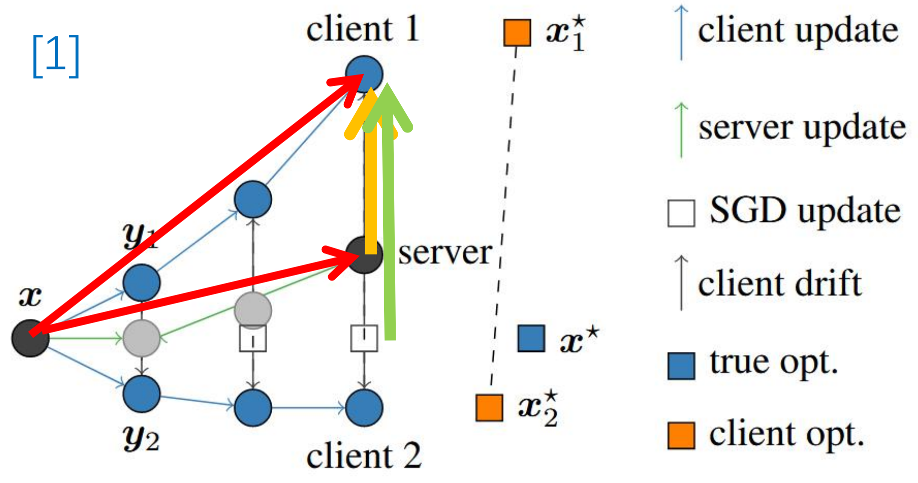
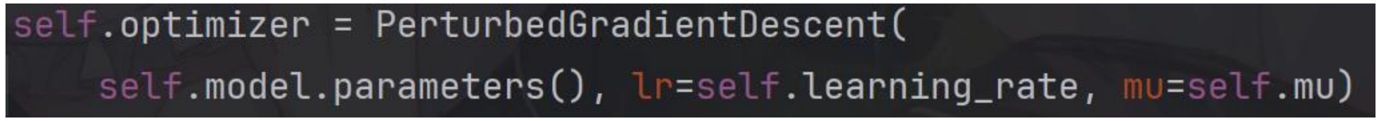
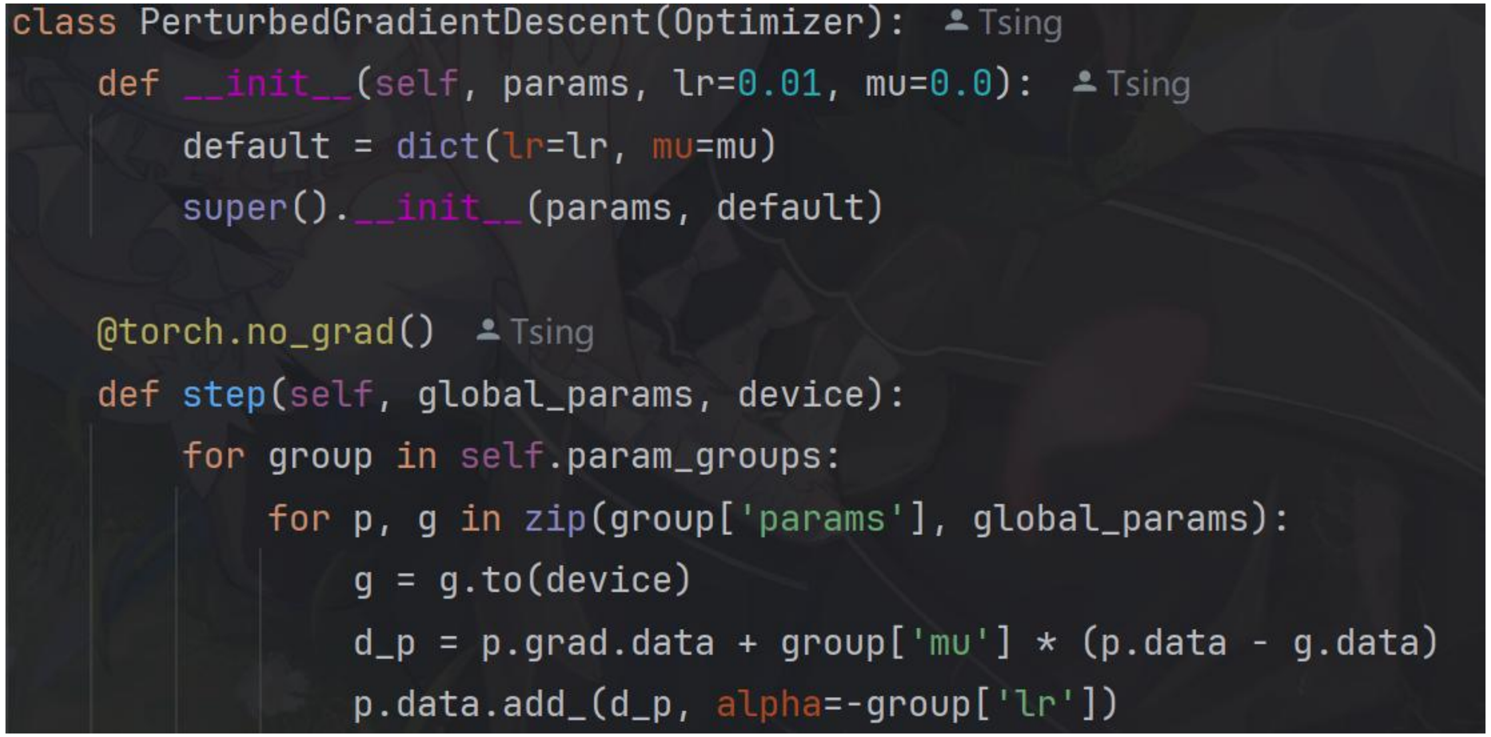


Figure 1. Client-drift in FEDAVG is illustrated for 2 clients with 3 local steps ($N = 2$, $K = 3$). The local updates $\boldsymbol{y}_i$ (in blue) move towards the individual client optima $\boldsymbol{x}_i^\star$ (orange square). The server updates (in red) move towards $\frac{1}{N}\sum_i \boldsymbol{x}_i^\star$ instead of to the true optimum $\boldsymbol{x}^\star$ (black square).

$$\min_{w} h_k(w;\ w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2 . \qquad (2)$$



```
self.optimizer = PerturbedGradientDescent(
    self.model.parameters(), lr=self.learning_rate, mu=self.mu)
```



```
class PerturbedGradientDescent(Optimizer):  Tsing
    def __init__(self, params, lr=0.01, mu=0.0):  Tsing
        default = dict(lr=lr, mu=mu)
        super().__init__(params, default)

    @torch.no_grad()  Tsing
    def step(self, global_params, device):
        for group in self.param_groups:
            for p, g in zip(group['params'], global_params):
                g = g.to(device)
                d_p = p.grad.data + group['mu'] * (p.data - g.data)
                p.data.add_(d_p, alpha=-group['lr'])
```

[2] https://github.com/TsingZ0/PFLlib

第四章收敛性分析和第五章实验设置暂略

第2章

q-FFL

Li T, Sanjabi M, Beirami A, et al.

Fair resource allocation in federated learning[J].

arXiv preprint arXiv:1905.10497, 2019.

[1]

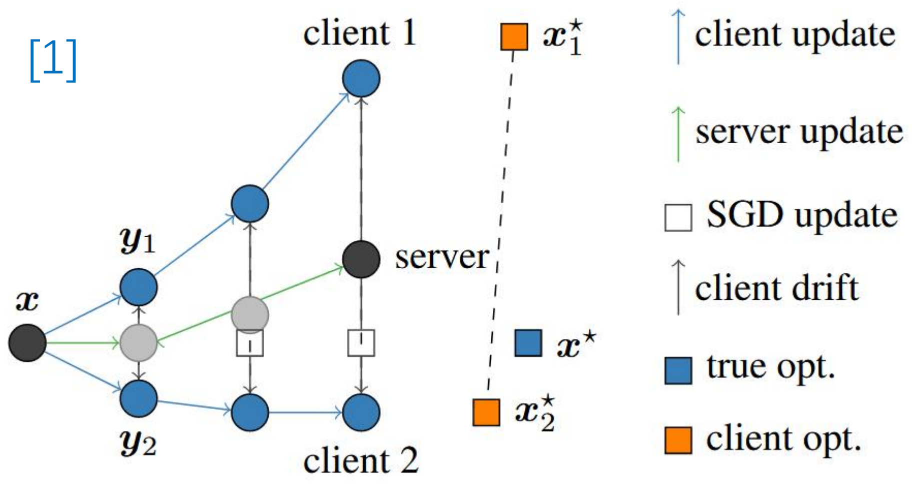


Figure 1. Client-drift in FEDAVG is illustrated for 2 clients with 3 local steps ($N = 2$, $K = 3$). The local updates y_i (in blue) move towards the individual client optima $x_i^\star$ (orange square). The server updates (in red) move towards $\frac{1}{N}\sum_i x_i^\star$ instead of to the true optimum $x^\star$ (black square).

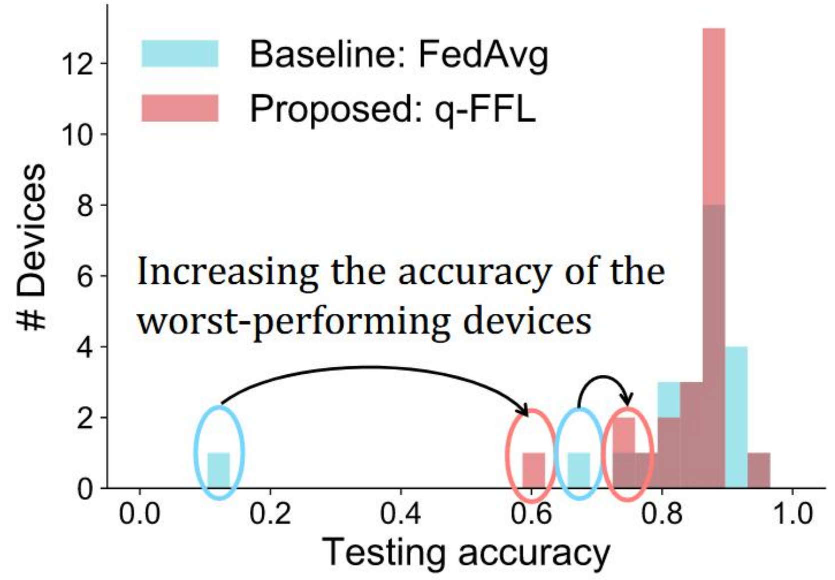
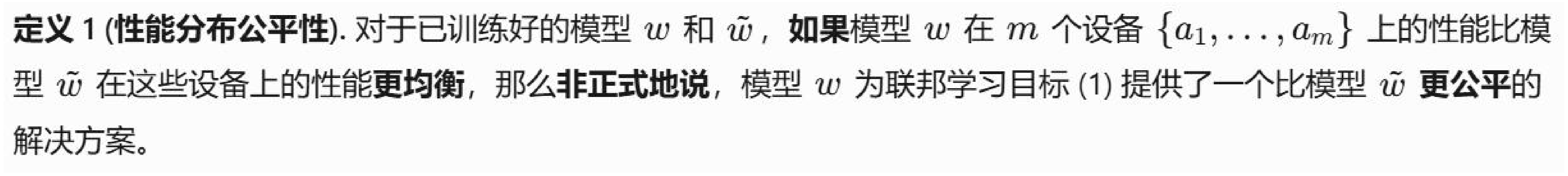


Figure 1: Model performance (e.g., test accuracy) in federated networks can vary widely across devices. Our objective, q-FFL, aims to increase the fairness/uniformity of model performance while maintaining average performance.

Group Fairness [丸式理解]：model在不同的label之间性能差异不大（通常以政治正确来讲故事）


定义 1 (性能分布公平性). 对于已训练好的模型 w 和 $\tilde{w}$，**如果**模型 w 在 m 个设备 $\{a_1, \ldots, a_m\}$ 上的性能比模型 $\tilde{w}$ 在这些设备上的性能**更均衡**，那么**非正式地说**，模型 w 为联邦学习目标 (1) 提供了一个比模型 $\tilde{w}$ **更公平**的解决方案。

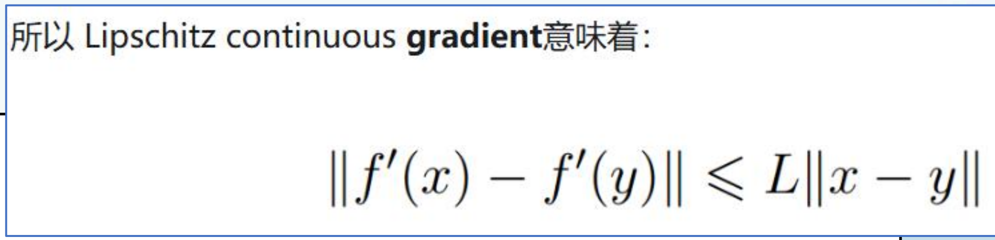
即model在不同的client之间性能差异不大（更符合联邦学习场景）

$$\min_{w} \; f_q(w) = \sum_{k=1}^{m} \frac{p_k}{q+1} F_k^{q+1}(w)\,, \tag{2}$$

q次方的意义：让loss较大的client model获得较大的权重，从而使得全局模型向此模型偏移

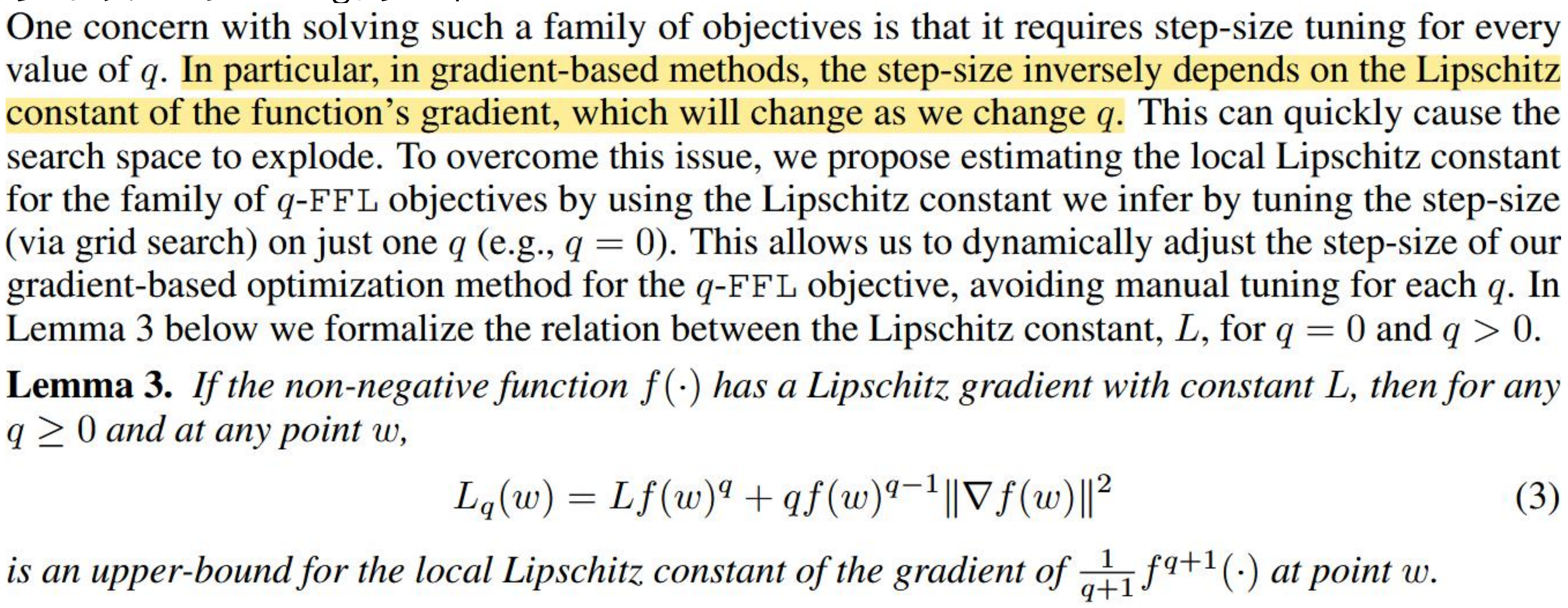
$$\begin{aligned}\min_{w} f_q(w) &= \sum_{k=1}^{m} \frac{p_k}{q+1} F_k^{q+1}(w) \\ &= \sum_{k=1}^{m} \underbrace{\left[\frac{p_k F_k^q(w)}{q+1}\right]}_{\text{视作权重}} F_k(w)\end{aligned}$$

写作+实验 trick


所以 Lipschitz continuous **gradient**意味着：

$$\|f'(x) - f'(y)\| \leqslant L\|x - y\|$$

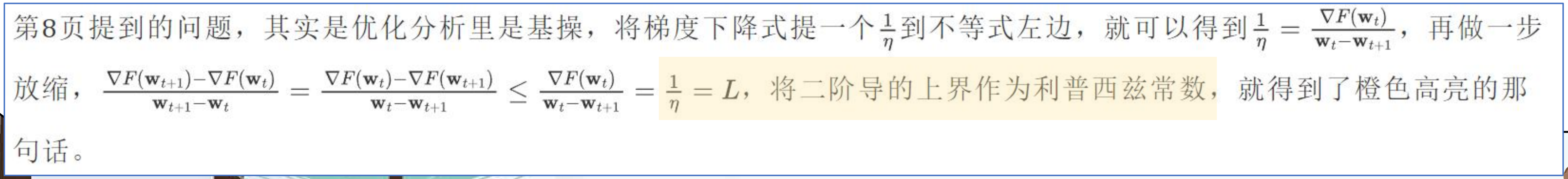
引入了新的超参数q，为超参的网格搜索复杂度增加了一个维度；TianLi在Lemma3里将q和学习率η联系起来，使得调参难度回到FedAvg的水平


One concern with solving such a family of objectives is that it requires step-size tuning for every value of q. In particular, in gradient-based methods, the step-size inversely depends on the Lipschitz constant of the function's gradient, which will change as we change q. This can quickly cause the search space to explode. To overcome this issue, we propose estimating the local Lipschitz constant for the family of q-FFL objectives by using the Lipschitz constant we infer by tuning the step-size (via grid search) on just one q (e.g., $q = 0$). This allows us to dynamically adjust the step-size of our gradient-based optimization method for the q-FFL objective, avoiding manual tuning for each q. In Lemma 3 below we formalize the relation between the Lipschitz constant, L, for $q = 0$ and $q > 0$.

Lemma 3. *If the non-negative function $f(\cdot)$ has a Lipschitz gradient with constant L, then for any $q \geq 0$ and at any point w,*

$$L_q(w) = Lf(w)^q + qf(w)^{q-1}\|\nabla f(w)\|^2 \tag{3}$$

is an upper-bound for the local Lipschitz constant of the gradient of $\frac{1}{q+1}f^{q+1}(\cdot)$ at point w.


第8页提到的问题，其实是优化分析里是基操，将梯度下降式提一个$\frac{1}{\eta}$到不等式左边，就可以得到$\frac{1}{\eta} = \frac{\nabla F(\mathbf{w}_t)}{\mathbf{w}_t - \mathbf{w}_{t+1}}$，再做一步放缩，$\frac{\nabla F(\mathbf{w}_{t+1}) - \nabla F(\mathbf{w}_t)}{\mathbf{w}_{t+1} - \mathbf{w}_t} = \frac{\nabla F(\mathbf{w}_t) - \nabla F(\mathbf{w}_{t+1})}{\mathbf{w}_t - \mathbf{w}_{t+1}} \leq \frac{\nabla F(\mathbf{w}_t)}{\mathbf{w}_t - \mathbf{w}_{t+1}} = \frac{1}{\eta} = L$，将二阶导的上界作为利普西兹常数，就得到了橙色高亮的那句话。

$$\min_{w} \; f_q(w) = \sum_{k=1}^{m} \frac{p_k}{q+1} F_k^{q+1}(w)\,, \tag{2}$$

写作+实验 trick

引入了新的超参数q，为超参的网格搜索复杂度增加了一个维度；TianLi在Lemma3里将q和学习率η联系起来，使得调参难度回到FedAvg的水平

Lemma 3. *If the non-negative function $f(\cdot)$ has a Lipschitz gradient with constant L, then for any $q \geq 0$ and at any point w,*

$$L_q(w) = Lf(w)^q + qf(w)^{q-1}\|\nabla f(w)\|^2 \tag{3}$$

is an upper-bound for the local Lipschitz constant of the gradient of $\frac{1}{q+1}f^{q+1}(\cdot)$ at point w.

对于单个client的目标函数

Proof. At any point w, we can compute the Hessian $\nabla^2\left(\frac{1}{q+1}f^{q+1}(w)\right)$ as:

$$\nabla^2\left(\frac{1}{q+1}f^{q+1}(w)\right) = qf^{q-1}(w)\underbrace{\nabla f(w)\nabla^T f(w)}_{\preceq\|\nabla f(w)\|^2\times I} + f^q(w)\boxed{\underbrace{\nabla^2 f(w)}_{\preceq L\times I}}. \tag{4}$$

As a result, $\|\nabla^2 \frac{1}{q+1}f^{q+1}(w)\|_2 \leq L_q(w) = Lf(w)^q + qf(w)^{q-1}\|\nabla f(w)\|^2$. □

对$\frac{1}{q+1}f^{q+1}(w)$求一阶导：$\nabla\left(\frac{1}{q+1}f^{q+1}(w)\right) = f^q(w) * \nabla f(w)$

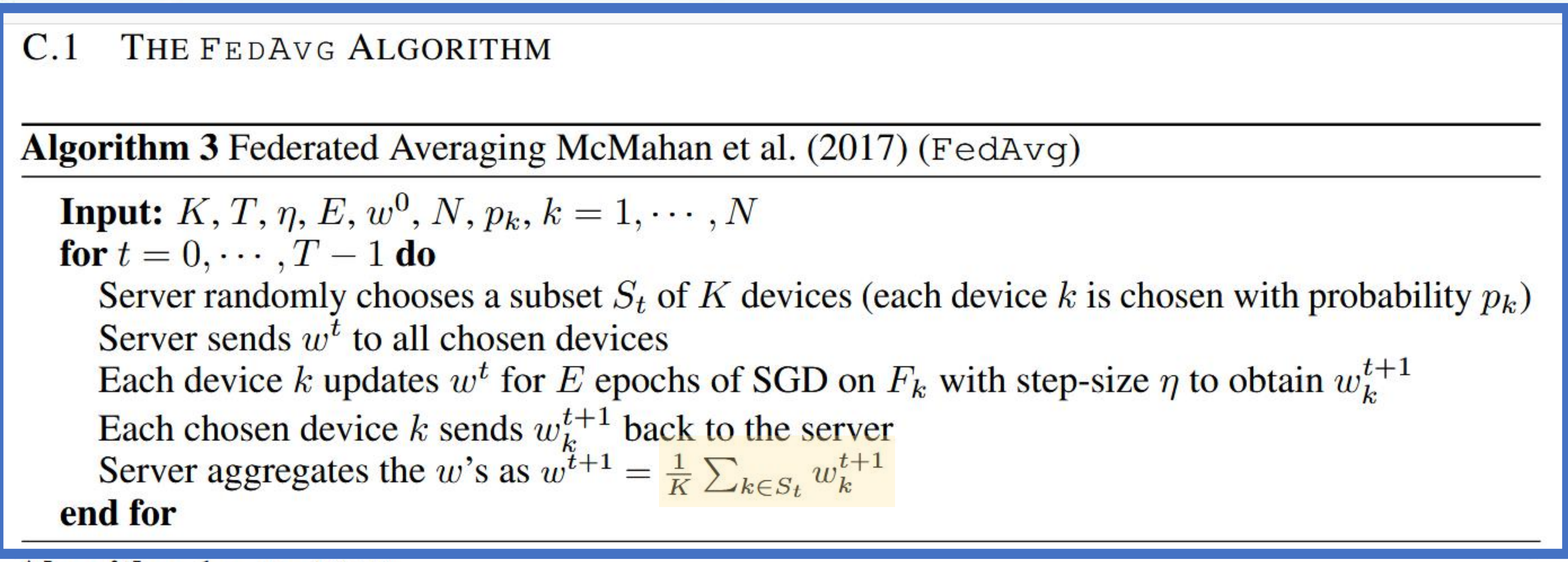
对$\frac{1}{q+1}f^{q+1}(w)$求二阶导：$\nabla^2\left(\frac{1}{q+1}f^{q+1}(w)\right) = q \cdot f^{q-1}(w) \cdot \nabla f(w) * \nabla f(w) + f^q(w) * \nabla^2 f(w)$

对$\frac{1}{q+1}f^{q+1}(w)$求一阶导: $\nabla\left(\frac{1}{q+1}f^{q+1}(w)\right)=f^q(w)*\nabla f(w)$

对$\frac{1}{q+1}f^{q+1}(w)$求二阶导: $\nabla^2\left(\frac{1}{q+1}f^{q+1}(w)\right)=q\cdot f^{q-1}(w)\cdot\nabla f(w)*\nabla f(w)+f^q(w)*\nabla^2 f(w)$

Algorithm 1 q-FedSGD

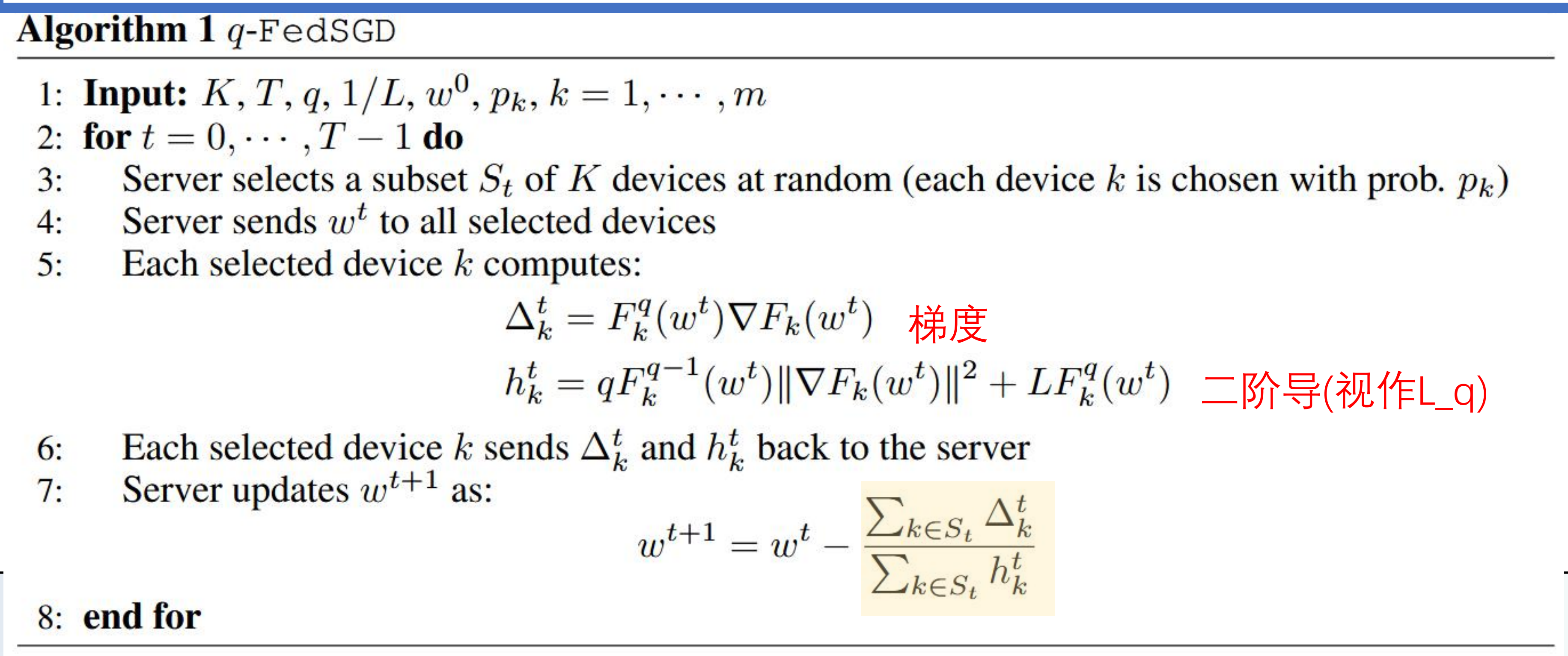
```
1: Input: $K, T, q, 1/L, w^0, p_k, k=1,\cdots,m$
2: for $t=0,\cdots,T-1$ do
3:     Server selects a subset $S_t$ of $K$ devices at random (each device $k$ is chosen with prob. $p_k$)
4:     Server sends $w^t$ to all selected devices
5:     Each selected device $k$ computes:
           $\Delta_k^t = F_k^q(w^t)\nabla F_k(w^t)$   梯度
           $h_k^t = qF_k^{q-1}(w^t)\|\nabla F_k(w^t)\|^2 + LF_k^q(w^t)$   二阶导(视作L_q)
6:     Each selected device $k$ sends $\Delta_k^t$ and $h_k^t$ back to the server
7:     Server updates $w^{t+1}$ as:
           $w^{t+1} = w^t - \frac{\sum_{k\in S_t}\Delta_k^t}{\sum_{k\in S_t}h_k^t}$
8: end for
```



C.1 THE FEDAVG ALGORITHM

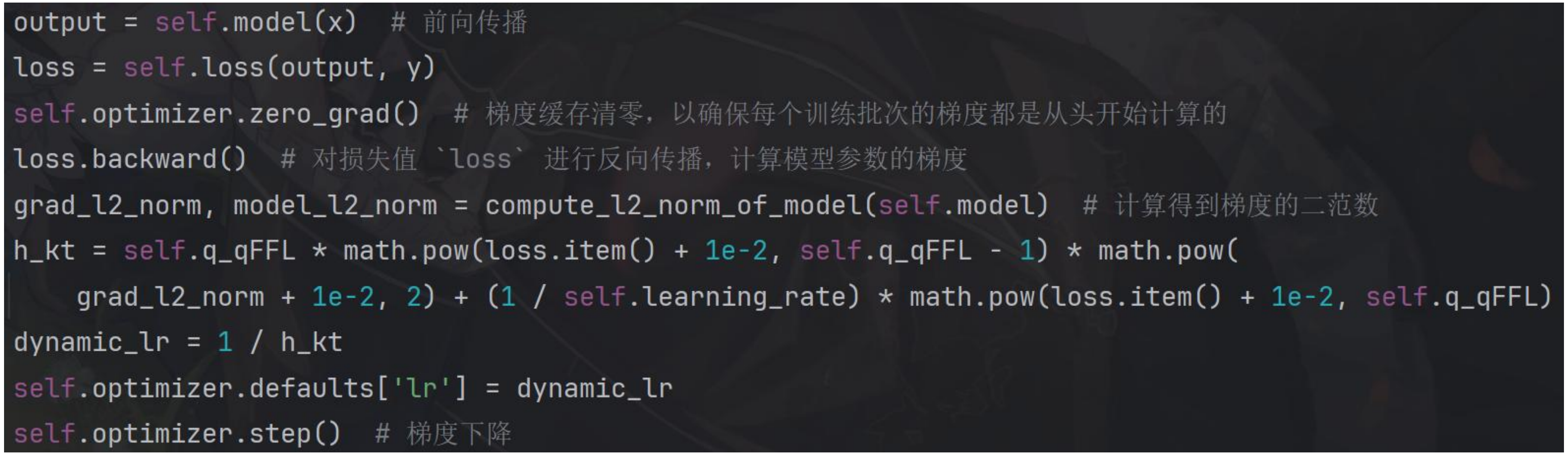
Algorithm 3 Federated Averaging McMahan et al. (2017) (FedAvg)

```
Input: $K, T, \eta, E, w^0, N, p_k, k = 1, \cdots, N$
for $t = 0, \cdots, T-1$ do
    Server randomly chooses a subset $S_t$ of $K$ devices (each device $k$ is chosen with probability $p_k$)
    Server sends $w^t$ to all chosen devices
    Each device $k$ updates $w^t$ for $E$ epochs of SGD on $F_k$ with step-size $\eta$ to obtain $w_k^{t+1}$
    Each chosen device $k$ sends $w_k^{t+1}$ back to the server
    Server aggregates the $w$'s as $w^{t+1} = \frac{1}{K}\sum_{k\in S_t} w_k^{t+1}$
end for
```



Algorithm 1 q-FedSGD

```
1: Input: $K, T, q, 1/L, w^0, p_k, k = 1, \cdots, m$
2: for $t = 0, \cdots, T-1$ do
3:     Server selects a subset $S_t$ of $K$ devices at random (each device $k$ is chosen with prob. $p_k$)
4:     Server sends $w^t$ to all selected devices
5:     Each selected device $k$ computes:
           $\Delta_k^t = F_k^q(w^t)\nabla F_k(w^t)$    梯度
           $h_k^t = qF_k^{q-1}(w^t)\|\nabla F_k(w^t)\|^2 + LF_k^q(w^t)$    二阶导(视作L_q)
6:     Each selected device $k$ sends $\Delta_k^t$ and $h_k^t$ back to the server
7:     Server updates $w^{t+1}$ as:
           $w^{t+1} = w^t - \frac{\sum_{k\in S_t}\Delta_k^t}{\sum_{k\in S_t} h_k^t}$
8: end for
```



```python
output = self.model(x)  # 前向传播
loss = self.loss(output, y)
self.optimizer.zero_grad()  # 梯度缓存清零，以确保每个训练批次的梯度都是从头开始计算的
loss.backward()  # 对损失值 `loss` 进行反向传播，计算模型参数的梯度
grad_l2_norm, model_l2_norm = compute_l2_norm_of_model(self.model)  # 计算得到梯度的二范数
h_kt = self.q_qFFL * math.pow(loss.item() + 1e-2, self.q_qFFL - 1) * math.pow(
    grad_l2_norm + 1e-2, 2) + (1 / self.learning_rate) * math.pow(loss.item() + 1e-2, self.q_qFFL)
dynamic_lr = 1 / h_kt
self.optimizer.defaults['lr'] = dynamic_lr
self.optimizer.step()  # 梯度下降
```

第3章

Ditto

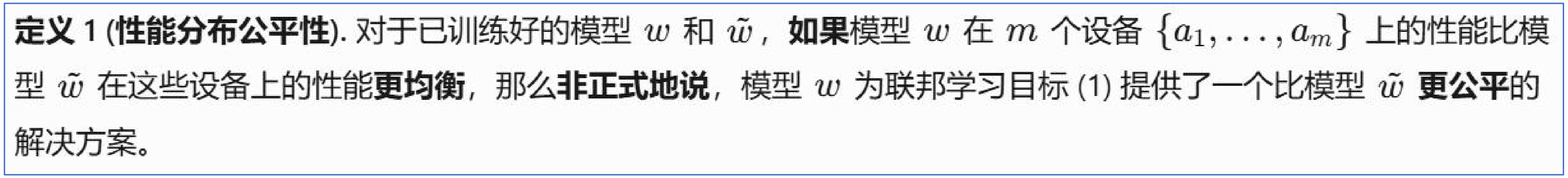
Li T, Hu S, Beirami A, et al.

Ditto: **Fair** and **robust** federated learning through personalization[C]

//International conference on machine learning. PMLR, 2021: 6357-6368.

有点摁加的感觉，为了扩展工作量

鲁棒性相关的内容，我个人觉得做得比较薄弱，跟公平性比起来，跟提出的算法耦合性不强


定义 1 (性能分布公平性). 对于已训练好的模型 w 和 $\tilde{w}$，**如果**模型 w 在 m 个设备 $\{a_1, \ldots, a_m\}$ 上的性能比模型 $\tilde{w}$ 在这些设备上的性能**更均衡**，那么**非正式地说**，模型 w 为联邦学习目标 (1) 提供了一个比模型 $\tilde{w}$ **更公平**的解决方案。

$$\min_{w} h_k(w;\ w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2 .$$

（FedProx） 找一个全局模型w，使得**所有client**的hk(加权)最小

$$\begin{aligned} \min_{v_k} \quad & h_k(v_k; w^*) := F_k(v_k) + \frac{\lambda}{2}\|v_k - w^*\|^2 \\ \text{s.t.} \quad & w^* \in \arg\min_{w} G(F_1(w), \ldots F_K(w)) . \end{aligned}$$

(Ditto)

找一个本地模型vk，使得对于**client k**来说，hk最小

受限于一个w*，这个w*是使得G最小的w

Algorithm 1: `Ditto` for Personalized FL

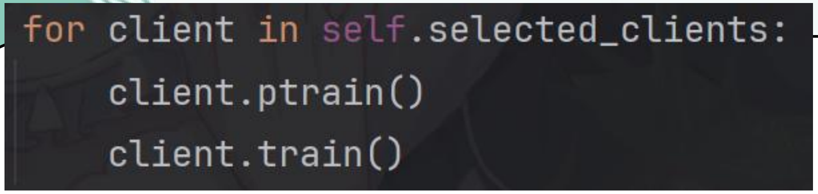
```
1 Input: $K, T, s, \lambda, \eta, w^0, \{v_k^0\}_{k\in[K]}$
2 for $t = 0, \cdots, T-1$ do
3     Server randomly selects a subset of devices $S_t$,
        and sends $w^t$ to them
4     for device $k \in S_t$ in parallel do
5         Solve the local sub-problem of $G(\cdot)$
            inexactly starting from $w^t$ to obtain $w_k^t$:
            $w_k^t \leftarrow \text{UPDATE\_GLOBAL}(w^t, \nabla F_k(w^t))$
          /* Solve $h_k(v_k; w^t)$ */
6         Update $v_k$ for $s$ local iterations:
            $v_k = v_k - \eta(\nabla F_k(v_k) + \lambda(v_k - w^t))$
          Send $\Delta_k^t := w_k^t - w^t$ back
7     Server aggregates $\{\Delta_k^t\}$:
        $w^{t+1} \leftarrow \text{AGGREGATE}\left(w^t, \{\Delta_k^t\}_{k\in\{S_t\}}\right)$
8 return $\{v_k\}_{k\in[K]}$ (personalized), $w^T$ (global)
```

$$\min_{v_k} \quad h_k(v_k; w^*) := F_k(v_k) + \frac{\lambda}{2}\|v_k - w^*\|^2 \qquad \texttt{(Ditto)}$$
$$\text{s.t.} \quad w^* \in \arg\min_w G(F_1(w), \ldots F_K(w))\,.$$

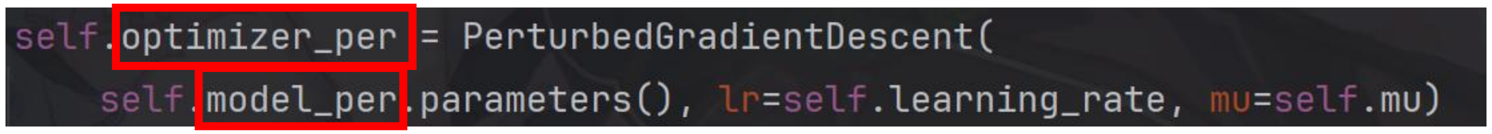
$$\min_w h_k(w;\ w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2\,. \qquad \text{(FedProx)}$$

Algorithm 2 `FedProx` (Proposed Framework)

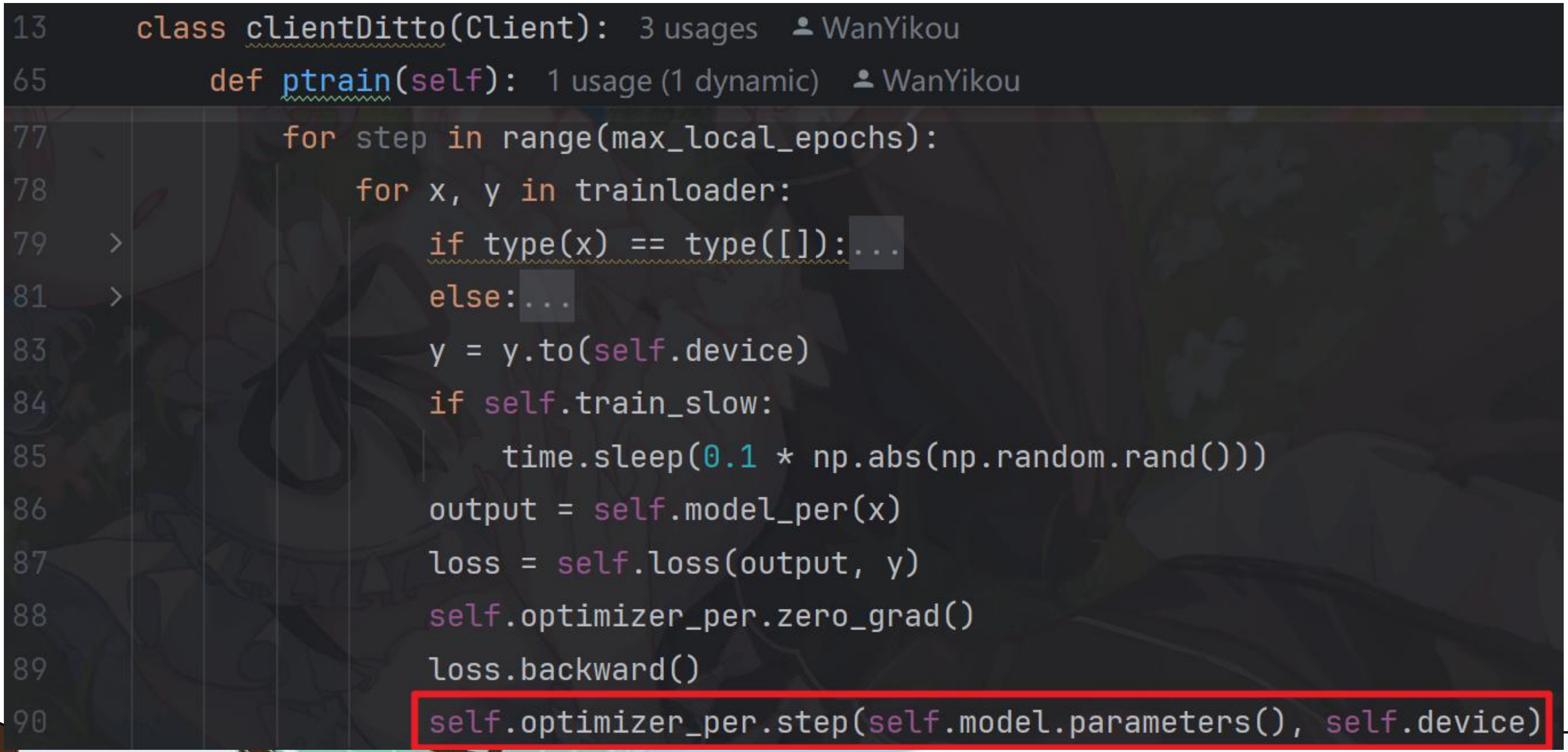
```
Input: $K, T, \mu, \gamma, w^0, N, p_k, k = 1, \cdots, N$
for $t = 0, \cdots, T-1$ do
    Server selects a subset $S_t$ of $K$ devices at random (each device $k$ is chosen with probability $p_k$)
    Server sends $w^t$ to all chosen devices
    Each chosen device $k \in S_t$ finds a $w_k^{t+1}$ which is a $\gamma_k^t$-inexact minimizer of: $w_k^{t+1} \approx \arg\min_w h_k(w;\ w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2$
    Each device $k \in S_t$ sends $w_k^{t+1}$ back to the server
    Server aggregates the $w$'s as $w^{t+1} = \frac{1}{K}\sum_{k\in S_t} w_k^{t+1}$
end for
```



```python
for client in self.selected_clients:
    client.ptrain()
    client.train()
```



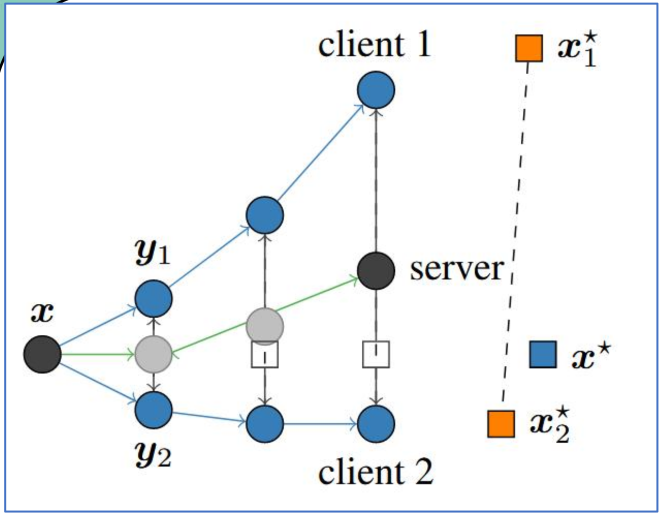
```python
self.optimizer_per = PerturbedGradientDescent(
    self.model_per.parameters(), lr=self.learning_rate, mu=self.mu)
```



```python
class clientDitto(Client):  3 usages  WanYikou
    def ptrain(self):  1 usage (1 dynamic)  WanYikou
        for step in range(max_local_epochs):
            for x, y in trainloader:
                if type(x) == type([]):...
                else:...
                y = y.to(self.device)
                if self.train_slow:
                    time.sleep(0.1 * np.abs(np.random.rand()))
                output = self.model_per(x)
                loss = self.loss(output, y)
                self.optimizer_per.zero_grad()
                loss.backward()
                self.optimizer_per.step(self.model.parameters(), self.device)
```

第4章

总结



异质性 ⟷ 公平性

$$\min_w h_k(w;\ w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2 . \qquad (2)$$

FedProx 2020

目标函数形式接近

q-FFL2019 ⟷ 公平性 ⟷ Ditto2021

$$\min_w \ f_q(w) = \sum_{k=1}^{m} \frac{p_k}{q+1} F_k^{q+1}(w)\,,$$

$$\begin{aligned} \min_{v_k} \quad & h_k(v_k; w^*) := F_k(v_k) + \frac{\lambda}{2}\left\|v_k - w^*\right\|^2 \\ \text{s.t.} \quad & w^* \in \arg\min_w G(F_1(w), \dots F_K(w))\,. \end{aligned} \qquad \text{(Ditto)}$$

感谢各位聆听~