Xi'an Jiaotong-Liverpool University
西交利物浦大学

Optimization Strategies for Multi-Task Learning (MTL)

- Part 1 – Loss Balancing Methods
- Part 2 – Gradient Balancing Methods
- Part 3 – ~~Find the Pareto Set (Frontier)~~

Yuanyuan Zhang

GitHub: https://github.com/amazing0844 LinkedIn: https://www.linkedin.com/in/yuanyuan-16

Part 1- Loss Balancing Methods

- Part 1 – Uncertainty Weights (UW)
- Part 2 – Dynamic Weight Average (DWA)
- Part 3 – Geometric Loss Strategy (GLS)
- Part 3 – Smooth Tchebycheff Scalarization (STCH)

Uncertainty Weight

2018, CVPR:

作者的数学模型通过贝叶斯模型建立。作者首先提出贝叶斯建模中存在两类不确定性：

认知不确定性（Epistemic uncertainty）：由于缺少训练数据而引起的不确定性
偶然不确定性（Aleatoric uncertainty）：由于训练数据无法解释信息而引起的不确定性

而对于偶然不确定性，又分为如下两个子类：

数据依赖地（Data-dependant）或异方差（Heteroscedastic）不确定性
任务依赖地（Task-dependant）或同方差（Homoscedastic）不确定性

多任务中，任务不确定性捕获任务间相关置信度，反应回归或分类任务的内在不确定性。

- *Epistemic uncertainty* is uncertainty in the model, which captures what our model does not know due to lack of training data. It can be explained away with increased training data.

Aleatoric uncertainty captures our uncertainty with respect to information which our data cannot explain. Aleatoric uncertainty can be explained away with the ability to observe all explanatory variables with increasing precision.

Aleatoric uncertainty can again be divided into two subcategories.

- *Data-dependent* or *Heteroscedastic* uncertainty is aleatoric uncertainty which depends on the input data and is predicted as a model output.
- *Task-dependent* or *Homoscedastic* uncertainty is aleatoric uncertainty which is not dependent on the input data. It is not a model output, rather it is a quantity which stays constant for all input data and varies between different tasks. It can therefore be described as task-dependent uncertainty.

Uncertainty Weight

2018, CVPR:

For **regression** tasks: likelihood as a Gaussian with mean given by the model output:

$$p(\mathbf{y}|\mathbf{f}^{\mathbf{W}}(\mathbf{x})) = \mathcal{N}(\mathbf{f}^{\mathbf{W}}(\mathbf{x}), \sigma^2)$$

Don't ask me why →

$$\log p(\mathbf{y}|\mathbf{f}^{\mathbf{W}}(\mathbf{x})) \propto -\frac{1}{2\sigma^2}||\mathbf{y} - \mathbf{f}^{\mathbf{W}}(\mathbf{x})||^2 - \log \sigma$$

For **classification**: pass through Softmax:

$$p(\mathbf{y}|\mathbf{f}^{\mathbf{W}}(\mathbf{x})) = \text{Softmax}(\mathbf{f}^{\mathbf{W}}(\mathbf{x})).$$

Don't ask me why →

$$\log p(\mathbf{y} = c|\mathbf{f}^{\mathbf{W}}(\mathbf{x}), \sigma) = \frac{1}{\sigma^2} f_c^{\mathbf{W}}(\mathbf{x}) - \log \sum_{c'} \exp\left(\frac{1}{\sigma^2} f_{c'}^{\mathbf{W}}(\mathbf{x})\right)$$

Multi-task likelihood:

$$p(\mathbf{y}_1, ..., \mathbf{y}_K|\mathbf{f}^{\mathbf{W}}(\mathbf{x})) = p(\mathbf{y}_1|\mathbf{f}^{\mathbf{W}}(\mathbf{x}))...p(\mathbf{y}_K|\mathbf{f}^{\mathbf{W}}(\mathbf{x}))$$

σ： noise parameter (learnable) capturing how much noise we have in the outputs

Final Loss

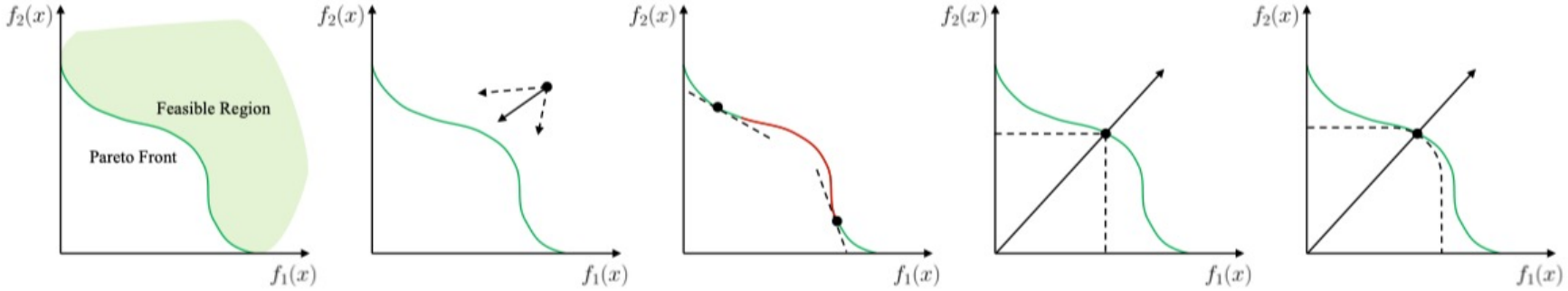
$$\mathcal{L}(\mathbf{W}, \sigma_1, \sigma_2) \approx \frac{1}{2\sigma_1^2}\mathcal{L}_1(\mathbf{W}) + \frac{1}{\sigma_2^2}\mathcal{L}_2(\mathbf{W}) + \log \sigma_1 + \log \sigma_2$$

large uncertainty, small weight

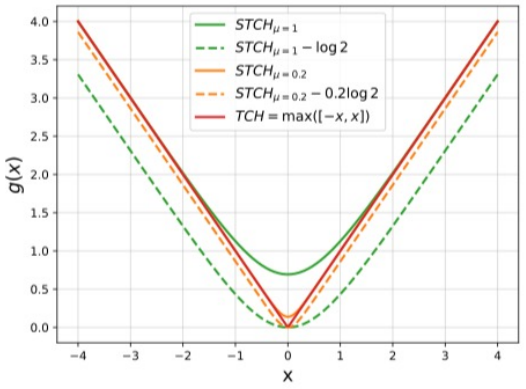
Smooth Tchebycheff Scalarization (STCH)

2024, ICML poster:

1. No access to Grad. info (fast), but could find solutions on non-convex pareto set
2. Rigorous in mathematical analyses
3. Can be used for pareto set search (don't ask me how)



(a) Problem (b) Adaptive Gradient (c) Linear Scalarization (d) TCH Scalarization (e) STCH (This Work)



(a) Smoothing Function

Tchebycheff Scalarization In this work, we focus on the Tchebycheff (TCH) scalarization with promising theoretical properties (Bowman, 1976; Steuer & Choo, 1983):

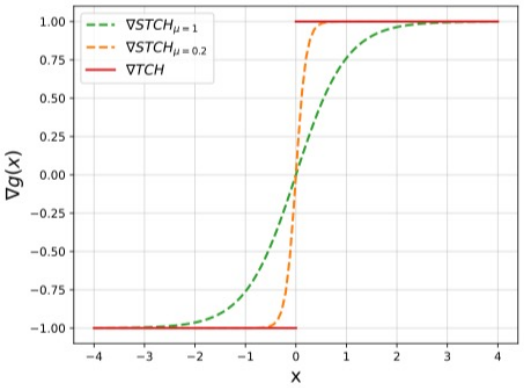
$$\min_{\boldsymbol{x}\in\mathcal{X}} g^{(\mathrm{TCH})}(\boldsymbol{x}|\boldsymbol{\lambda}) = \min_{\boldsymbol{x}\in\mathcal{X}} \max_{1\le i\le m} \{\lambda_i(f_i(\boldsymbol{x}) - z_i^*)\}, \quad (5)$$

where $\boldsymbol{\lambda} \in \boldsymbol{\Delta}^{m-1}$ is the preference vector and $\boldsymbol{z}^* \in \mathbb{R}^m$ is the ideal objective values (e.g., $\boldsymbol{z}_i^* = \min f_i(\boldsymbol{x}) - \epsilon$ with a small $\epsilon > 0$).

lowing smooth Tchebycheff (STCH) scalarization for multi-objective optimization:

$$g_\mu^{(\mathrm{STCH})}(\boldsymbol{x}|\boldsymbol{\lambda}) = \mu \log\left(\sum_{i=1}^{m} e^{\frac{\lambda_i(f_i(\boldsymbol{x}) - z_i^*)}{\mu}}\right), \quad (9)$$

where μ is the smoothing parameter, m is the number of objectives, $\boldsymbol{\lambda} \in \boldsymbol{\Delta}^{m-1}$ and $\boldsymbol{z}^* \in \mathbb{R}^m$ are the preference vector and ideal objective values respectively as in the classical Tchebycheff scalarization (5). Like other scalarization



(b) Smoothing Gradient

Smooth Tchebycheff Scalarization (STCH)

Implementation (essential but only in Appendix)

1. Warmup epochs with equal weights (dft. Epoch = 4)
2. Normalization vector (as the average loss values after warmup)

```
if self.epoch < warmup_epoch:
    loss = torch.mul(torch.log(losses+1e-20), torch.ones_like(losses).to(self.device)).sum()
    loss.backward()
    return  batch_weight
elif self.epoch == warmup_epoch:
    loss = torch.mul(torch.log(losses+1e-20), torch.ones_like(losses).to(self.device)).sum()
    self.average_loss += losses.detach()
    self.average_loss_count += 1

    loss.backward()
    return  batch_weight
else:
    if self.nadir_vector == None:
        self.nadir_vector = self.average_loss / self.average_loss_count
        print(self.nadir_vector)

    losses = torch.log(losses/self.nadir_vector+1e-20)
    max_term = torch.max(losses.data).detach()
    reg_losses = losses - max_term

    loss = mu * torch.log(torch.sum(torch.exp(reg_losses/mu))) * self.task_num
    loss.backward()
```

$$g_\mu^{(\mathrm{STCH})}(\boldsymbol{x}|\boldsymbol{\lambda}) = \mu \log\left(\sum_{i=1}^{m} e^{\frac{y_i}{\mu}}\right)$$

$$\boldsymbol{y}_i = \boldsymbol{\lambda}_i(\boldsymbol{f}_i(\boldsymbol{x}) - z_i^*).$$

$$\hat{g}_\mu^{(\mathrm{STCH})}(\boldsymbol{x}|\boldsymbol{\lambda}) = \mu \log\left(\sum_{i=1}^{m} e^{\hat{y}_i/\mu}\right)$$

$$\tilde{y} = \max_{\forall i} y_i, \qquad \hat{y}_i = y_i - \tilde{y}$$

stabalized version of the STCH scalarization

Lambda = 1/m, z_i* = 0 in this paper;
Both of these param. can be used for task preference.

Others

Geometric Loss Strategy, 2019, CVPR workshop:

$$\mathcal{L}_{Total} = \prod_{i=1}^{n} \sqrt[n]{\mathcal{L}_i}$$

$$\mathcal{L}_{Total} = \sqrt[3]{\mathcal{L}_1 \mathcal{L}_2 \mathcal{L}_3}$$

Dynamic Weight Average, 2019, CVPR:

With DWA, we define the weighting λ_k for task k as:

$$\lambda_k(t) := \frac{K \exp(w_k(t-1)/T)}{\sum_i \exp(w_i(t-1)/T)}, w_k(t-1) = \frac{\mathcal{L}_k(t-1)}{\mathcal{L}_k(t-2)}, \quad (7)$$

w_k : the ratio between the loss from last iteration and the iteration before

After a large number of iteration (large T) → equal weight

Part 2 - Gradient Balancing Methods

- Part 0 – Nash MTL
- Part 1 – Multiple Gradient Descent Algorithm(MGDA)
- Part 2 – Gradient Normalization (GradNorm)
- Part 3 – Projecting Conflicting Gradients (PCGrad)
- Part 4 – Gradient Vaccine (GradVac)
- Part 5 – Conflict-Averse Gradient descent (CAGrad)
- Part 6 – Gradient Homogenization(RotoGrad)
- Part 7 – Gradient sign Dropout (GradDrop)
- Part 8 – Impartial Multi-task Learning (IMTL)
- Part 9 – Independent Component Alignment (Aligned MTL)

Nash MTL

2022, ICML:

Algorithm 1 Nash-MTL

Input: $\theta^{(0)}$ – initial parameter vector, $\{\ell_i\}_{i=1}^K$ – differentiable loss functions, η – learning rate
for $t = 1, ..., T$ **do**
  Compute task gradients $g_i^{(t)} = \nabla_{\theta^{(t-1)}} \ell_i$
  Set $G^{(t)}$ the matrix with columns $g_i^{(t)}$
  Solve for α: $(G^{(t)})^\top G^{(t)} \alpha = 1/\alpha$ to obtain $\alpha^{(t)}$
  Update the parameters $\theta^{(t)} = \theta^{(t)} - \eta G^{(t)} \alpha^{(t)}$
end for
Return: $\theta^{(T)}$

定义

纳什均衡是指在一个博弈中，所有参与者的策略组合使得每个参与者在其他人策略不变的情况下，无法通过单方面改变自己的策略来提高自己的收益[1][2][3][4][5][6][7][8]。

Claim 3.1. *Let G be the $d \times K$ matrix whose columns are the gradients g_i. The solution to* $\arg\max_{\Delta\theta \in B_\epsilon} \sum_i \log(\Delta\theta^\top g_i)$ *is (up to scaling)* $\sum_i \alpha_i g_i$ *where $\alpha \in \mathbb{R}_+^K$ is the solution to $G^\top G\alpha = 1/\alpha$ where $1/\alpha$ is the element-wise reciprocal.*

$\Delta\theta = \sum_i \alpha_i g_i$

$\Delta\theta^\top g_i = \sum_j \alpha_j g_j^\top g_i = \frac{1}{\alpha_i}$.

update vector $\Delta\theta$ in the ball of radius ϵ

$$\arg\max_{\Delta\theta \in B_\epsilon} \sum_i \log(\Delta\theta^\top g_i)$$

Nash bargaining solution

Utility function

G be the d × K matrix whose columns are the gradients

Nash disagreement is set to 0 (no gradient descent)

normalize

We now provide some intuition for this solution. First, if all g_i are orthogonal we get $\alpha_i = 1/||g_i||$ and $\Delta\theta = \sum \frac{g_i}{||g_i||}$ which is the obvious scale invariant solution. When they are not orthogonal, we get

$$\alpha_i ||g_i||^2 + \sum_{j \neq i} \alpha_j g_j^\top g_i = 1/\alpha_i \qquad (2)$$

We can consider $\sum_{j \neq i} \alpha_j g_j^\top g_i = \left(\sum_{j \neq i} \alpha_j g_j\right)^\top g_i$ as the interaction between task i and the other tasks; If it is positive there is a positive interaction and the other gradients aid the i'th task, and if it is negative they hamper it. When there is a negative interaction, the LHS of Eq. 2 decreases and as a result, α_i increases to compensate for it. Conversely, where there is a positive interaction α_i will decrease.

Summary: verify gradient conflict or not, then try to achieve the equal gradient descent for every task

Multiple Gradient Descent Algorithm(MGDA)

2018, NIPS:

$$\min_{\substack{\boldsymbol{\theta}^{sh},\\ \boldsymbol{\theta}^1,\ldots,\boldsymbol{\theta}^T}} \mathbf{L}(\boldsymbol{\theta}^{sh},\boldsymbol{\theta}^1,\ldots,\boldsymbol{\theta}^T) = \min_{\substack{\boldsymbol{\theta}^{sh},\\ \boldsymbol{\theta}^1,\ldots,\boldsymbol{\theta}^T}} (\hat{\mathcal{L}}^1(\boldsymbol{\theta}^{sh},\boldsymbol{\theta}^1),\ldots,\hat{\mathcal{L}}^T(\boldsymbol{\theta}^{sh},\boldsymbol{\theta}^T))^\intercal.$$

Gradient Descent

$$f(x_{k+1}) = f(x_k + \alpha_k d_k) = f(x_k) + \alpha_k \nabla f(x_k)^T d_k + O(\alpha_k), \quad \alpha_k > 0。$$

MGDA

$$\min_{\alpha^1,\ldots,\alpha^T} \left\{ \left\| \sum_{t=1}^{T} \alpha^t \nabla_{\boldsymbol{\theta}^{sh}} \hat{\mathcal{L}}^t(\boldsymbol{\theta}^{sh},\boldsymbol{\theta}^t) \right\|_2^2 \middle| \sum_{t=1}^{T} \alpha^t = 1, \alpha^t \geq 0 \quad \forall t \right\}$$

Such that

$\sum_{t=1}^{T} \alpha^t = 1$ and $\sum_{t=1}^{T} \alpha^t \nabla_{\boldsymbol{\theta}^{sh}} \hat{\mathcal{L}}^t(\boldsymbol{\theta}^{sh},\boldsymbol{\theta}^t) = 0$

$\nabla_{\boldsymbol{\theta}^t} \hat{\mathcal{L}}^t(\boldsymbol{\theta}^{sh},\boldsymbol{\theta}^t) = 0$

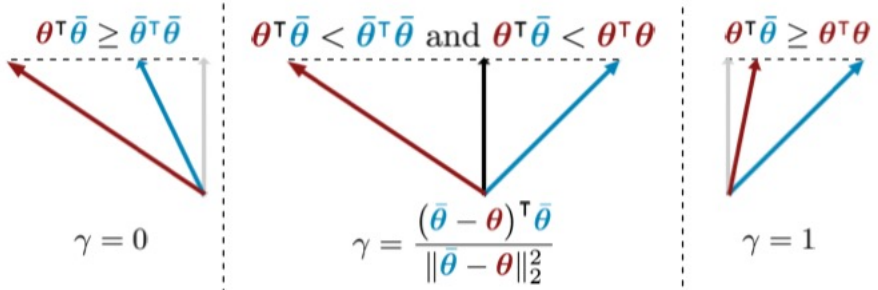


Figure 1: Visualisation of the min-norm point in the convex hull of two points ($\min_{\gamma\in[0,1]} \|\gamma\boldsymbol{\theta} + (1-\gamma)\bar{\boldsymbol{\theta}}\|_2^2$). As the geometry suggests, the solution is either an edge case or a perpendicular vector.

Algorithm 1

$\min_{\gamma\in[0,1]} \|\gamma\boldsymbol{\theta} + (1-\gamma)\bar{\boldsymbol{\theta}}\|_2^2$

if $\boldsymbol{\theta}^\intercal\bar{\boldsymbol{\theta}} \geq \boldsymbol{\theta}^\intercal\boldsymbol{\theta}$ **then**
  $\gamma = 1$
else if $\boldsymbol{\theta}^\intercal\bar{\boldsymbol{\theta}} \geq \bar{\boldsymbol{\theta}}^\intercal\bar{\boldsymbol{\theta}}$ **then**
  $\gamma = 0$
else
  $\gamma = \frac{(\bar{\boldsymbol{\theta}}-\boldsymbol{\theta})^\intercal\bar{\boldsymbol{\theta}}}{\|\bar{\boldsymbol{\theta}}-\boldsymbol{\theta}\|_2^2}$
end if

$\boldsymbol{\theta}^t = \boldsymbol{\theta}^t - \eta\nabla_{\boldsymbol{\theta}^t} \hat{\mathcal{L}}^t(\boldsymbol{\theta}^{sh},\boldsymbol{\theta}^t)$

$\boldsymbol{\theta}^{sh} = \boldsymbol{\theta}^{sh} - \eta \sum_{t=1}^{T} \alpha^t \nabla_{\boldsymbol{\theta}^{sh}} \hat{\mathcal{L}}^t(\boldsymbol{\theta}^{sh},\boldsymbol{\theta}^t)$

Multi-objective gradient with Correction (MoCo)

2023, ICLR:

Stochastic version of MGDA

Stochastic version of any existing opt. methods can be promising in overall performance.

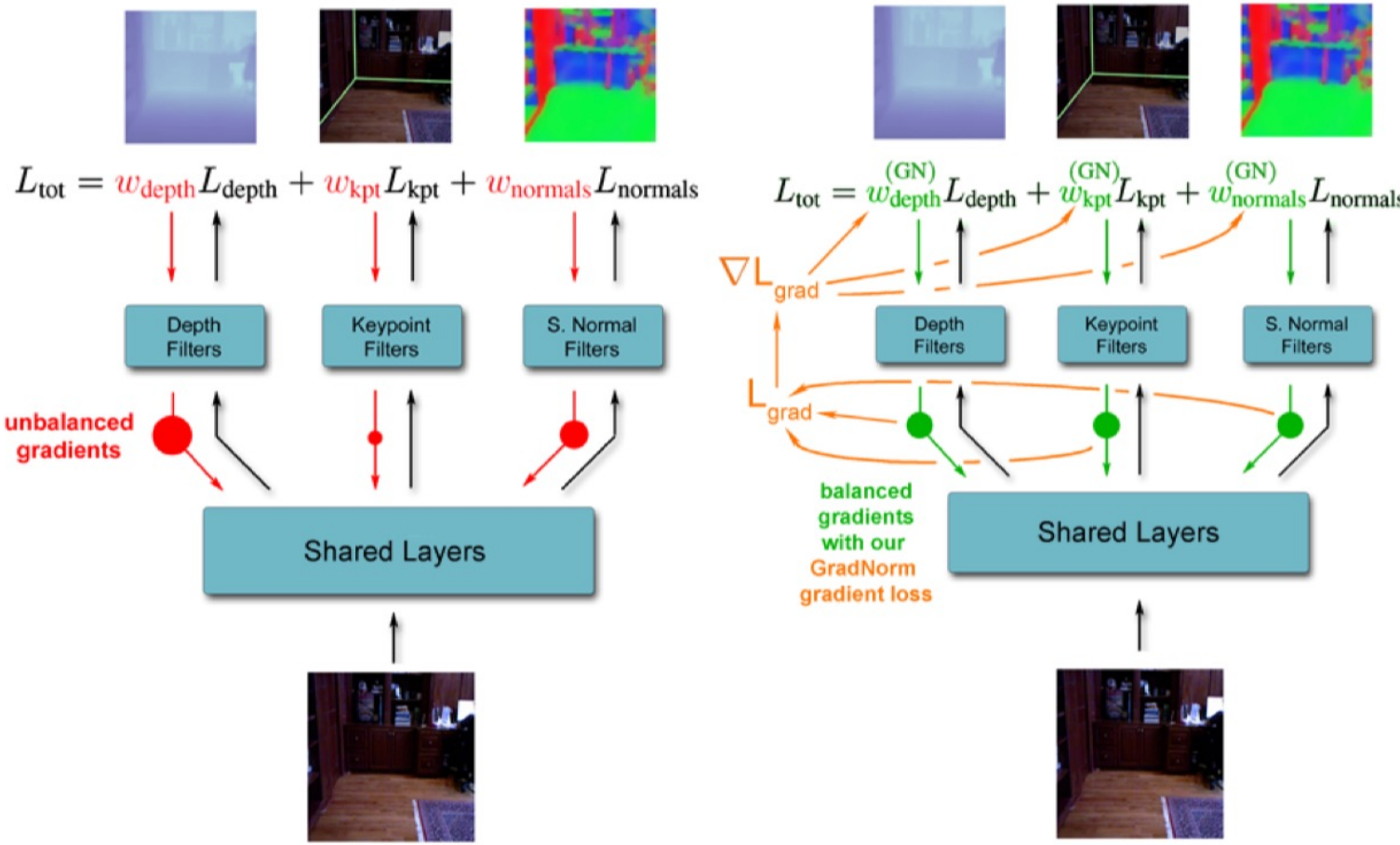
Gradient Normalization (GradNorm)

2018, ICML:

Balance the gradient

Algorithm 1 Training with GradNorm

Initialize $w_i(0) = 1 \; \forall i$
Initialize network weights $\mathcal{W}$
Pick value for $\alpha > 0$ and pick the weights W (usually the final layer of weights which are shared between tasks)
for $t = 0$ **to** max_train_steps **do**
  Input batch x_i to compute $L_i(t) \; \forall i$ and $L(t) = \sum_i w_i(t) L_i(t)$ [standard forward pass]
  Compute $G_W^{(i)}(t)$ and $r_i(t) \; \forall i$
  Compute $\overline{G}_W(t)$ by averaging the $G_W^{(i)}(t)$
  Compute $L_{\text{grad}} = \sum_i |G_W^{(i)}(t) - \overline{G}_W(t) \times [r_i(t)]^\alpha|_1$
  Compute GradNorm gradients $\nabla_{w_i} L_{\text{grad}}$, keeping targets $\overline{G}_W(t) \times [r_i(t)]^\alpha$ constant
  Compute standard gradients $\nabla_{\mathcal{W}} L(t)$
  Update $w_i(t) \mapsto w_i(t+1)$ using $\nabla_{w_i} L_{\text{grad}}$
  Update $\mathcal{W}(t) \mapsto \mathcal{W}(t+1)$ using $\nabla_{\mathcal{W}} L(t)$ [standard backward pass]
  Renormalize $w_i(t+1)$ so that $\sum_i w_i(t+1) = T$
end for



Gradient Normalization (GradNorm)

2018, ICML:

- W: The subset of the full network weights $W \subset \mathcal{W}$ where we actually apply GradNorm. W is generally chosen as the last shared layer of weights to save on compute costs[1].
- $G_W^{(i)}(t) = ||\nabla_W w_i(t) L_i(t)||_2$: the L_2 norm of the gradient of the weighted single-task loss $w_i(t)L_i(t)$ with respect to the chosen weights W.
- $\overline{G}_W(t) = E_{\text{task}}[G_W^{(i)}(t)]$: the average gradient norm across all tasks at training time t.
- $\tilde{L}_i(t) = L_i(t)/L_i(0)$: the loss ratio for task i at time t. $\tilde{L}_i(t)$ is a measure of the *inverse* training rate of task i (i.e. lower values of $\tilde{L}_i(t)$ correspond to a faster training rate for task i)[2].
- $r_i(t) = \tilde{L}_i(t)/E_{\text{task}}[\tilde{L}_i(t)]$: the relative *inverse* training rate of task i.

Concretely, the higher the value of r_i(t), the higher the gradient magnitudes should be for task i in order to encourage the task to train more quickly.

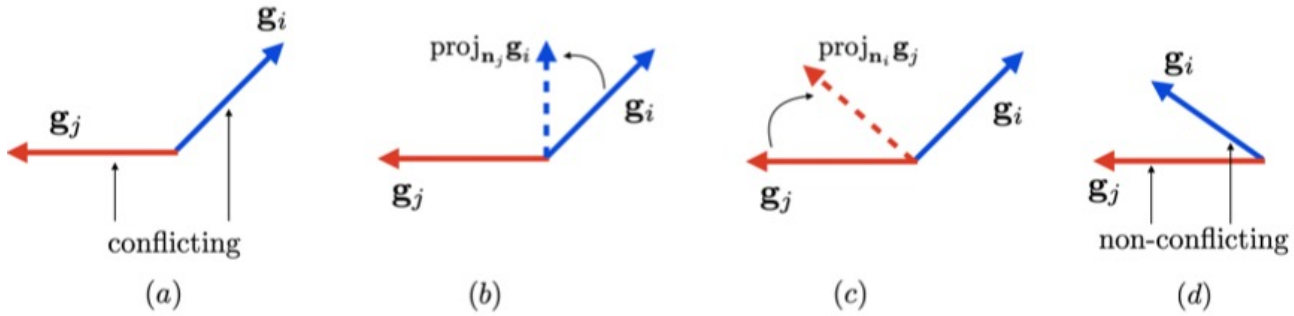
Algorithm 1 Training with GradNorm

Initialize $w_i(0) = 1 \; \forall i$
Initialize network weights $\mathcal{W}$
Pick value for $\alpha > 0$ and pick the weights W (usually the final layer of weights which are shared between tasks)
for $t = 0$ **to** max_train_steps **do**
  Input batch x_i to compute $L_i(t) \; \forall i$ and $L(t) = \sum_i w_i(t) L_i(t)$ [standard forward pass]
  Compute $G_W^{(i)}(t)$ and $r_i(t) \; \forall i$
  Compute $\overline{G}_W(t)$ by averaging the $G_W^{(i)}(t)$
  Compute $L_{\text{grad}} = \sum_i |G_W^{(i)}(t) - \overline{G}_W(t) \times [r_i(t)]^\alpha|_1$
  Compute GradNorm gradients $\nabla_{w_i} L_{\text{grad}}$, keeping targets $\overline{G}_W(t) \times [r_i(t)]^\alpha$ constant
  Compute standard gradients $\nabla_{\mathcal{W}} L(t)$
  Update $w_i(t) \mapsto w_i(t+1)$ using $\nabla_{w_i} L_{\text{grad}}$
  Update $\mathcal{W}(t) \mapsto \mathcal{W}(t+1)$ using $\nabla_{\mathcal{W}} L(t)$ [standard backward pass]
  Renormalize $w_i(t+1)$ so that $\sum_i w_i(t+1) = T$
end for

Projecting Conflicting Gradients (PCGrad)

2020, NIPS:

1. Gradient conflict with each other
2. Magnitude of gradient is imbalanced
3. High positive curvature along the directions of the task gradients



Definition 1. *We define ϕ_{ij} as the angle between two task gradients $\mathbf{g}_i$ and $\mathbf{g}_j$. We define the gradients as* ***conflicting*** *when* $\cos\phi_{ij} < 0$.

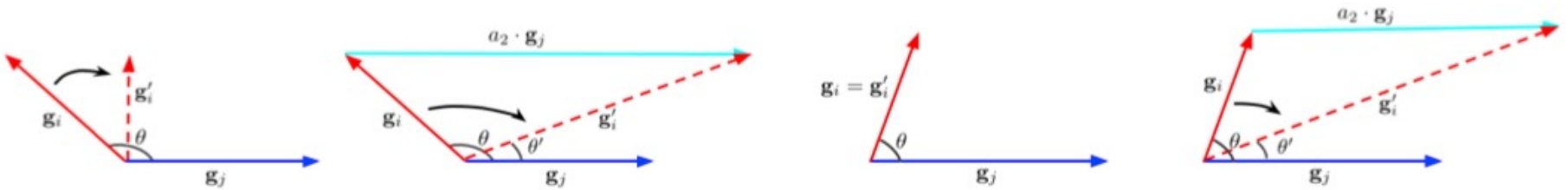
Definition 2. *We define the* ***gradient magnitude similarity*** *between two gradients $\mathbf{g}_i$ and $\mathbf{g}_j$ as* $\Phi(\mathbf{g}_i, \mathbf{g}_j) = \frac{2\|\mathbf{g}_i\|_2\|\mathbf{g}_j\|_2}{\|\mathbf{g}_i\|_2^2+\|\mathbf{g}_j\|_2^2}$.

When the magnitude of two gradients is the same, this value is equal to 1. As the gradient magnitudes become increasingly different, this value goes to zero.

Definition 3. *We define* ***multi-task curvature*** *as* $\mathbf{H}(\mathcal{L};\theta,\theta') = \int_0^1 \nabla\mathcal{L}(\theta)^T\nabla^2\mathcal{L}(\theta + a(\theta' - \theta))\nabla\mathcal{L}(\theta)da$, *which is the averaged curvature of $\mathcal{L}$ between θ and θ' in the direction of the multi-task gradient $\nabla\mathcal{L}(\theta)$.*

Gradient Vaccine (GradVac)

2021, ICLR:



Problem in PCGrad: stall in situation (b), where cosine similarity is positive but low

$$\mathbf{g}_i' = \mathbf{g}_i + \frac{\|\mathbf{g}_i\|(\phi_{ij}^T\sqrt{1-\phi_{ij}^2} - \phi_{ij}\sqrt{1-(\phi_{ij}^T)^2})}{\|\mathbf{g}_j\|\sqrt{1-(\phi_{ij}^T)^2}} \cdot \mathbf{g}_j.$$

arbitrary gradient similarity objective ϕ_{ij}^T

$$\hat{\phi}_{ijk}^{(t)} = (1-\beta)\hat{\phi}_{ijk}^{(t-1)} + \beta\phi_{ijk}^{(t)},$$

exponential moving average

gradient changes across tasks (i,j), layers (k) and training steps (t).

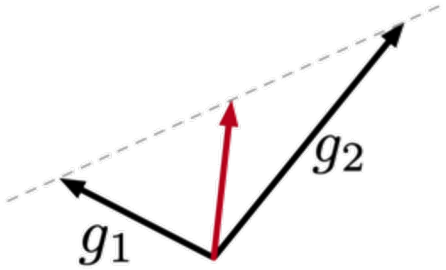
Conflict-Averse Gradient descent (CAGrad)

2021, NIPS:

Gradients from different tasks

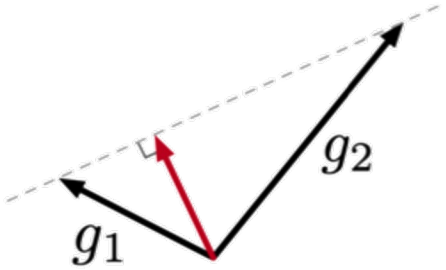
1) may have varying scales with the largest gradient dominating the update
2) may point in different directions so that directly optimizing the average loss can be quite detrimental to a specific task's performance.
3) MGDA can not control which specific point it will converge to

The idea of CAGrad is simple: it looks for an update vector that maximizes the worst local improvement of any objective in a neighborhood of the average gradient.



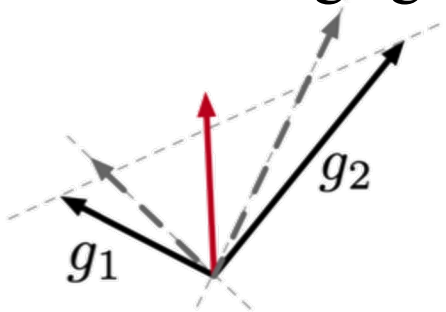
GD

$$d = (g_1 + g_2)/2$$



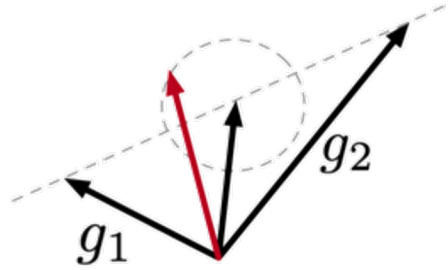
MGDA

$$\max_d \min_i g_i^\top d$$
$$\text{s.t. } \|d\| \leq 1$$



PCGrad

$$d = (g_{1\perp 2} + g_{2\perp 1})/2$$
$$\text{where } g_{i\perp j} = g_i - \frac{g_i^\top g_j}{\|g_j\|} g_j$$



CAGrad (ours)

$$\max_d \min_i g_i^\top d$$
$$\text{s.t. } \|d - g_0\| \leq c\,\|g_0\|$$

Conflict-Averse Gradient descent (CAGrad)

2021, NIPS:

update θ by $\theta' \leftarrow \theta - \alpha d$,

Choose update vector d to decrease not only the average loss L_0, but also every individual loss

1. Calculate minimum decrease rate across the losses (R(-) is negative)

$$R(\theta, d) = \max_{i\in[K]} \left\{ \frac{1}{\alpha} \left(L_i(\theta - \alpha d) - L_i(\theta) \right) \right\} \approx - \min_{i\in[K]} \langle g_i, d \rangle,$$

conflict among objectives (worst one)

2. Minmise the worst one

averaged gradient g_0

$$\min_{d\in\mathbb{R}^m} \max_{i\in[K]} \left\{ \frac{1}{\alpha} (\hat{L}_i(\theta - \alpha d) - \hat{L}_i) \right\} \quad \text{by} \quad \max_{d\in\mathbb{R}^m} \min_{i\in[K]} \langle g_i, d \rangle \quad \text{s.t.} \quad \|d - g_0\| \leq c \|g_0\|,$$

c: control convergence rate

Within constrained radius $\hat{g}_0 = 1/K \sum_i^K \hat{g}_i, \; c \in [0, 1)$

$$\boldsymbol{g}_i = \nabla_{\boldsymbol{\theta}} \boldsymbol{\mathcal{L}}_i$$

Read paper for ultimate solution

Gradient Homogenization (RotoGrad)

2022, ICLR spotlight:

C can be viewed as step length

1. homogenize the magnitude of the gradients across tasks

$$||\omega_k \boldsymbol{G}_k|| = ||\omega_i \boldsymbol{G}_i|| \quad \forall i \iff \omega_k \boldsymbol{G}_k = \frac{C}{||\boldsymbol{G}_k||}\boldsymbol{G}_k = C\boldsymbol{U}_k \quad \forall k,$$

$$C := \sum_k \alpha_k ||\boldsymbol{G}_k|| \qquad \alpha_k = \frac{||\boldsymbol{G}_k||/||\boldsymbol{G}_k^0||}{\sum_i ||\boldsymbol{G}_i||/||\boldsymbol{G}_i^0||},$$

2. homogenizes task-gradient directions

$$\boldsymbol{x} \xrightarrow{f_{\boldsymbol{\theta}}} \boldsymbol{z} \begin{cases} \xrightarrow{\boldsymbol{R}_1} \boldsymbol{r}_1 \xrightarrow{h_{\boldsymbol{\phi}_1}} L_1(h_1(\boldsymbol{r}_1), \boldsymbol{y}_1) \\ \xrightarrow{\boldsymbol{R}_2} \boldsymbol{r}_2 \xrightarrow{h_{\boldsymbol{\phi}_2}} L_2(h_2(\boldsymbol{r}_2), \boldsymbol{y}_2) \\ \cdots \qquad\qquad \cdots \\ \xrightarrow{\boldsymbol{R}_K} \boldsymbol{r}_K \xrightarrow{h_{\boldsymbol{\phi}_K}} L_K(h_K(\boldsymbol{r}_K), \boldsymbol{y}_K) \end{cases}$$

Backbone with shared params θ: $\boldsymbol{z} = f(\boldsymbol{x}; \boldsymbol{\theta}) \in \mathbb{R}^d$

Head network with params $\boldsymbol{\phi}_k$ $h_k(\boldsymbol{x}) = h_k(\boldsymbol{z}; \boldsymbol{\phi}_k)$

Task-specific rotation matrices $\boldsymbol{R}_k$, $\boldsymbol{r}_k := \boldsymbol{R}_k \boldsymbol{z}$.

normalized batch gradient

$$\boldsymbol{v}_n := \frac{1}{K}\sum_k \boldsymbol{u}_{n,k}$$

New loss for **Rotation:** $\mathcal{L}_{\text{rot}}^k := -\sum \langle \boldsymbol{R}_k^\top \tilde{\boldsymbol{g}}_{n,k}, \boldsymbol{v}_n \rangle,$

New training target: maximize the batch-wise cosine similarity

1. v_n is the target vector that we want all task gradients pointing towards
2. Still have separate optimal direction ,but not convergent to the v_n

Gradient Homogenization (RotoGrad)

2022, ICLR:

Algorithm 1 Training step with RotoGrad.

Input input samples $\boldsymbol{X}$, task labels $\{\boldsymbol{Y}_k\}$, network's (RotoGrad's) learning rate η (η_{roto})
Output backbone (heads) parameters $\boldsymbol{\theta}$ ($\{\boldsymbol{\phi}_k\}$), RotoGrad's parameters $\{\boldsymbol{R}_k\}$

compute shared feature $\boldsymbol{Z} = f(\boldsymbol{X}; \boldsymbol{\theta})$
for $k = 1, 2, \ldots, K$ **do**
  compute task-specific loss $L_k = \sum_n L_k(h_k(\boldsymbol{R}_k \boldsymbol{z}_n; \boldsymbol{\phi}_k), \boldsymbol{y}_{n,k})$
  compute gradient of shared feature $\boldsymbol{G}_k = \nabla_{\boldsymbol{z}} L_k$
  compute gradient of task-specific feature $\widetilde{\boldsymbol{G}}_k = \boldsymbol{R}_k \boldsymbol{G}_k$ ▷ Treated as constant w.r.t. $\boldsymbol{R}_k$.
  compute unitary gradients $\boldsymbol{U}_k = \boldsymbol{G}_k / ||\boldsymbol{G}_k||$
  compute relative task convergence $\alpha_k = ||\boldsymbol{G}_k|| / ||\boldsymbol{G}_k^0||$
end for
make $\{\alpha_k\}$ sum up to one $[\alpha_1, \alpha_2, \ldots, \alpha_K] = [\alpha_1, \alpha_2, \ldots, \alpha_K] / \sum_k \alpha_k$
compute shared magnitude $C = \sum_k \alpha_k ||\boldsymbol{G}_k||$
update backbone parameters $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \boldsymbol{z} \cdot C \sum_k \boldsymbol{U}_k$
compute target vector $\boldsymbol{V} = \frac{1}{K} \sum_k \boldsymbol{U}_k$
for $k = 1, 2, \ldots, K$ **do**
  compute RotoGrad's loss $L_k^{\text{roto}} = -\sum_n \langle \boldsymbol{R}_k^\top \widetilde{\boldsymbol{g}}_{n,k}, \boldsymbol{v}_n \rangle$
  update RotoGrad's parameters $\boldsymbol{R}_k = \boldsymbol{R}_k - \eta_{\text{roto}} \nabla_{\boldsymbol{R}_k} L_k^{\text{roto}}$
  update head's parameters $\boldsymbol{\phi}_k = \boldsymbol{\phi}_k - \eta \nabla_{\boldsymbol{\phi}_k} L_k$
end for

Stackelberg game (主从博弈):

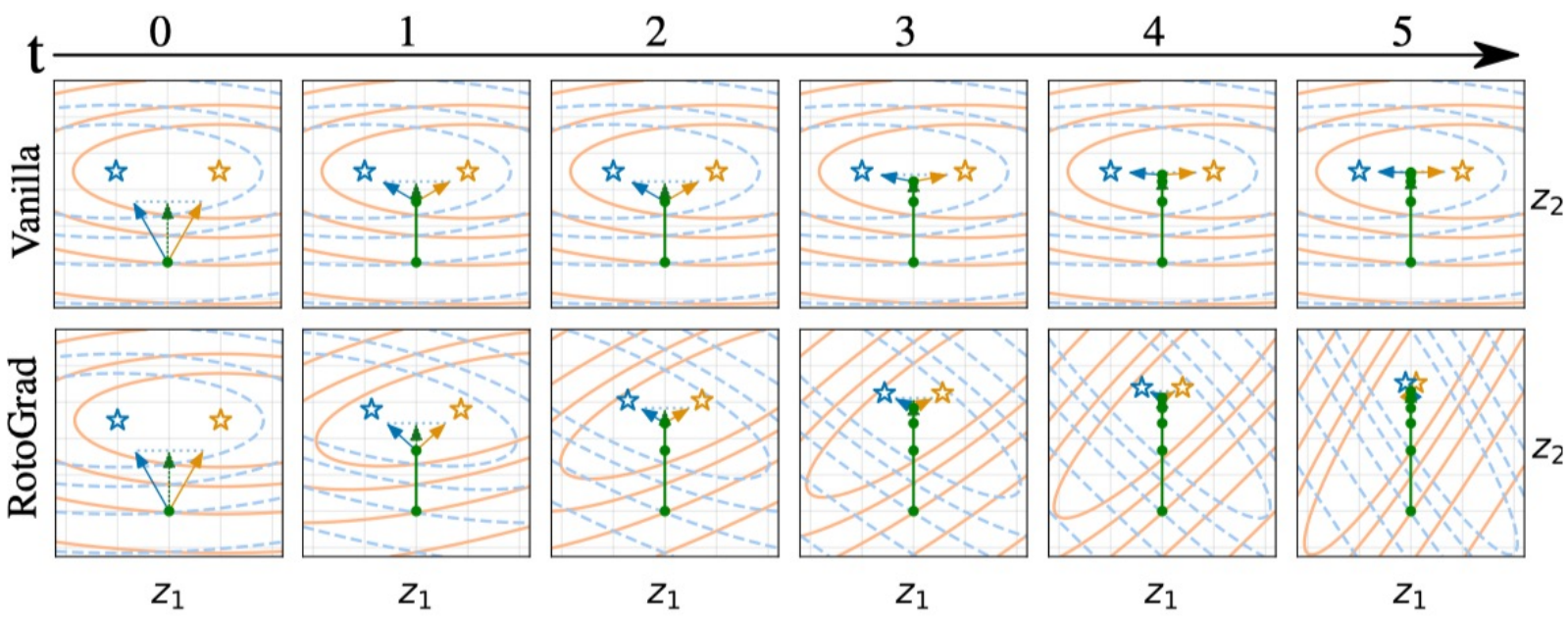
$$\mathcal{N}\text{etwork: } \underset{\boldsymbol{\theta}, \{\boldsymbol{\phi}\}_k}{\text{minimize}} \sum_k \omega_k L_k.,$$

$$\mathcal{R}\text{otation: } \underset{\{\boldsymbol{R}_k\}_k}{\text{minimize}} \sum_k \mathcal{L}_{\text{rot}}^k$$

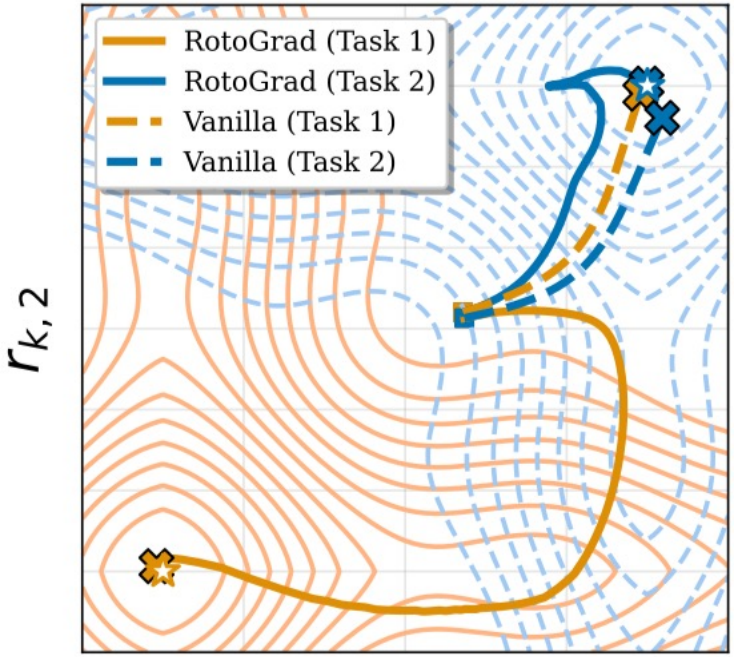
rotations' optimizer (leader)

Gradient Homogenization (RotoGrad)

2022. ICLR:



(a) Convex avocado-shaped experiment.

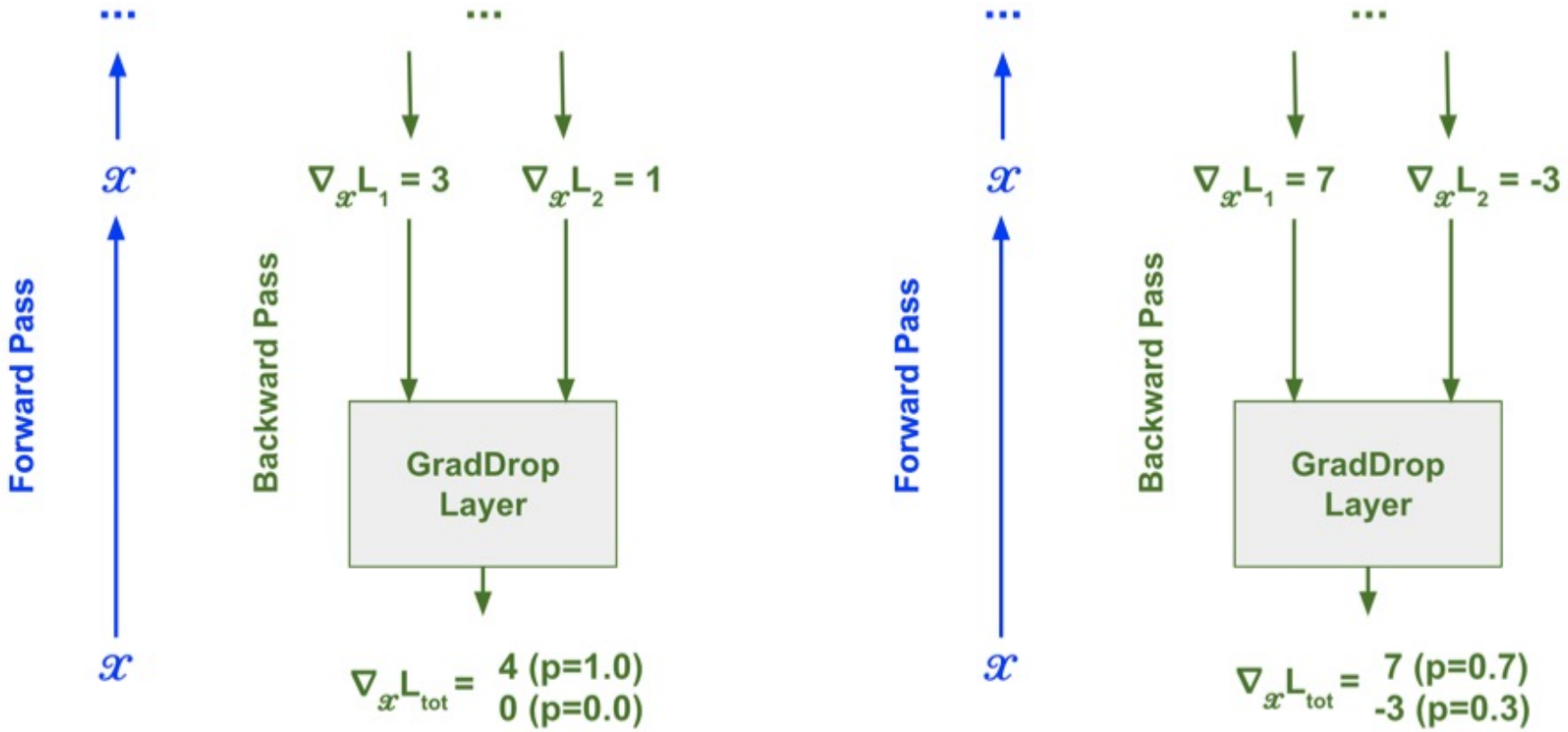


(b) Non-convex experiment.

Figure 1: Level plots showing the evolution of two regression MTL problems with/without RotoGrad, see §4. RotoGrad is able to reach the optimum (☆) for both tasks. *(a)* In the space of z, RotoGrad rotates the function-spaces to align task gradients (blue/orange arrows), finding shared features z (green arrow) closer to the (matched) optima. *(b)* In the space of r_k, RotoGrad rotates the shared feature z, providing per-task features r_k that better fit each task.

Gradient sign Dropout (GradDrop)

2020. NIPS: 0.5 * (1 + (3 + 1)/(| 3 | + | 1 |)) = 1.0 0.5 * (1 + (7 + 3)/(| 7 | + | - 3 |)) = 0.7,



$$\mathcal{P} = \frac{1}{2}\left(1 + \frac{\sum_i \nabla L_i}{\sum_i |\nabla L_i|}\right)$$

Gradient Positive Sign Purity (quality of the current minma)

(P is a measure of how many positive gradients are present at any given value)

Impartial Multi-task Learning (IMTL)

2021. ICLR:

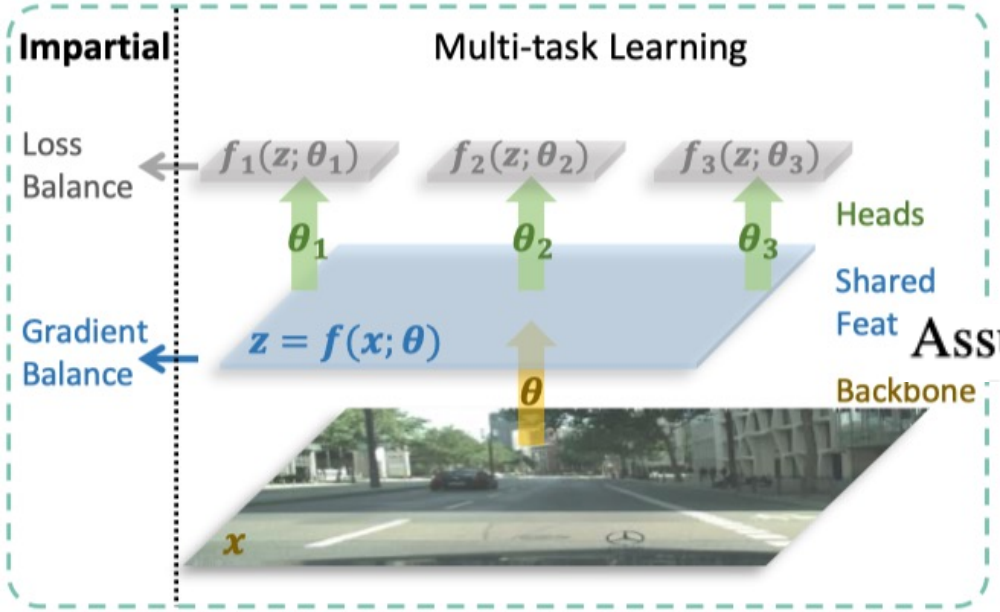


Figure 2: Overview of IMTL.

IMTL-G (for shared params) $\{\boldsymbol{u}_t = \boldsymbol{g}_t / \|\boldsymbol{g}_t\|\}$

$$\boldsymbol{g}\boldsymbol{u}_1^\top = \boldsymbol{g}\boldsymbol{u}_t^\top \Leftrightarrow \boldsymbol{g}(\boldsymbol{u}_1 - \boldsymbol{u}_t)^\top = 0,\ \forall\, 2 \leqslant t \leqslant T.$$

$$\boldsymbol{\alpha} = \boldsymbol{g}_1 \boldsymbol{U}^\top \left(\boldsymbol{D}\boldsymbol{U}^\top\right)^{-1}. \qquad \text{(IMTL-G)} \qquad \sum_t \alpha_t = 1.$$

Assume $\boldsymbol{\alpha} = [\alpha_2, \cdots, \alpha_T]$, $\boldsymbol{U}^\top = \left[\boldsymbol{u}_1^\top - \boldsymbol{u}_2^\top, \cdots, \boldsymbol{u}_1^\top - \boldsymbol{u}_T^\top\right]$,

$$\boldsymbol{D}^\top = \left[\boldsymbol{g}_1^\top - \boldsymbol{g}_2^\top, \cdots, \boldsymbol{g}_1^\top - \boldsymbol{g}_T^\top\right]$$

IMTL-L (for task params.)

Scale loss $\{\alpha_t L_t\}$ to constant

learnable scale parameters $\{s_t\}$ $\{h(s_t) > 0\}$

encouraging the scaled losses $h(s)L(\boldsymbol{\theta})$ to be 1 for all tasks,

$$g(s) = e^s L(\boldsymbol{\theta}) - s, \qquad \text{(IMTL-L).}$$

Similar with UW by log(-)

Impartial Multi-task Learning (IMTL)

tasks by encouraging the scaled losses $h(s)L(\boldsymbol{\theta})$ to be 1 for all tasks, so the optimality $s^\star$ of s is achieved when $h(s)L(\boldsymbol{\theta}) = 1$, or equivalently:

$$f(s) \equiv h(s)L(\boldsymbol{\theta}) - 1 = 0, \text{ if } s = s^\star. \tag{3}$$

Cannot be figured out directly

One may expect to minimize $|f(s)| = |h(s)L(\boldsymbol{\theta}) - 1|$ to find $s^\star$, however when $h(s)L(\boldsymbol{\theta}) < 1$, the gradient with respect to $\boldsymbol{\theta}$, $\nabla_{\boldsymbol{\theta}}|f(s)| = -h(s)\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta})$, is in the opposite direction. On the other hand, assume our scaled loss $g(s)$ is a differentiable convex function with respect to s, then its minimum is achieved if and only if $s = s^\star$, where the derivative of $g(s)$ is zero:

$$g'(s) = 0, \text{ if } s = s^\star. \tag{4}$$

From Eq. (3) and (4) we find that the values of $f(s)$ and $g'(s)$ are both 0 when $s = s^\star$, we can then regard $f(s)$ as the derivative of $g(s)$, which is our target scaled loss and used to optimize both the network parameters $\boldsymbol{\theta}$ and loss scale parameter s, then we have:

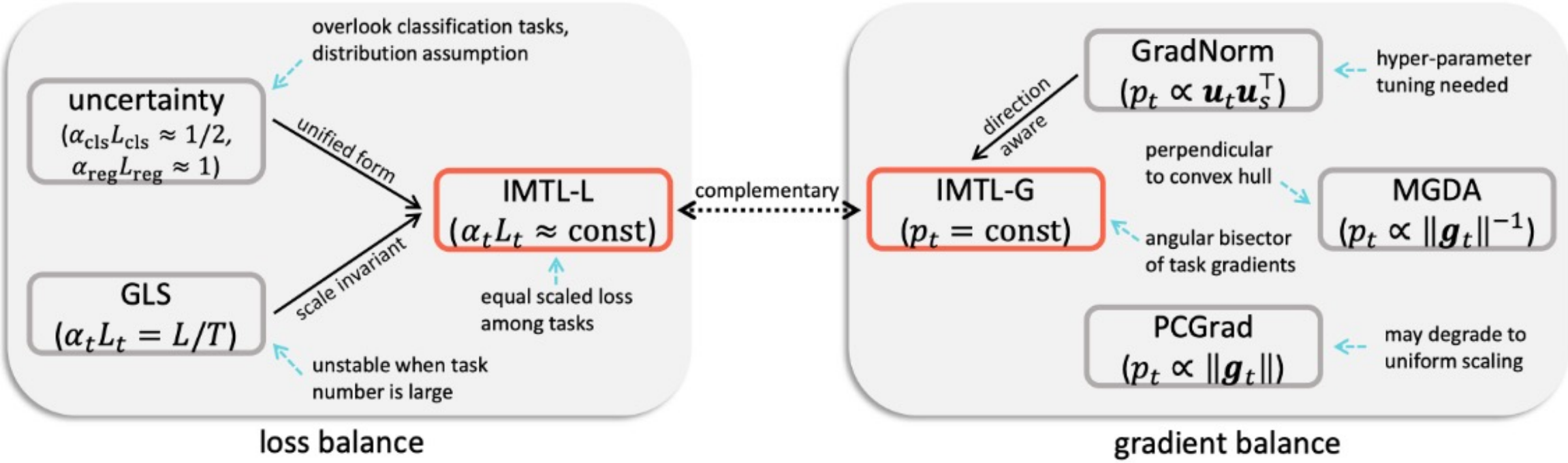
$$g'(s) = f(s) \Leftrightarrow g(s) = \int f(s)\,\mathrm{d}s = L(\boldsymbol{\theta})\int h(s)\,\mathrm{d}s - s. \tag{5}$$

From Eq. (3) and (5), we notice that both $h(s)$ and $\int h(s)\,\mathrm{d}s$ denote loss scales, so we have $\int h(s)\,\mathrm{d}s = Ch(s)$, where $C > 0$ is a constant. According to ordinary differential equation, $\int h(s)\,\mathrm{d}s$ must be the exponential function: $\int h(s)\,\mathrm{d}s = ba^s$ with $a > 1, b > 0$ (see Appendix B.2). We then have $g''(s) = ka^s$, $k > 0$, which is always positive and verifies our assumption about the convexity of $g(s)$. Also note that the gradient of $g(s)$ with respect to $\boldsymbol{\theta}$, $\nabla_{\boldsymbol{\theta}}g(s) = \int h(s)\,\mathrm{d}s\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}) = ba^s\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta})$, is in the appropriate direction since $ba^s > 0$. As an instantiation, we set $\int h(s)\,\mathrm{d}s = e^s$ $(a = e, b = 1)$, then

$$g(s) = e^s L(\boldsymbol{\theta}) - s, \qquad \text{(IMTL-L)}. \tag{6}$$

Impartial Multi-task Learning (IMTL)

2021, ICLR:



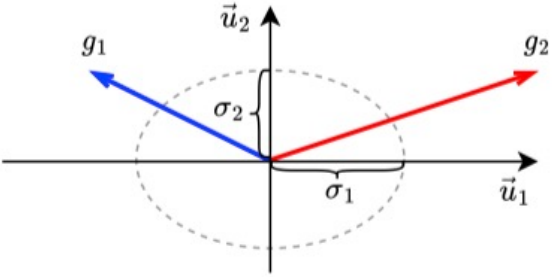
Basically the same

MGDA： fixed on convex hull

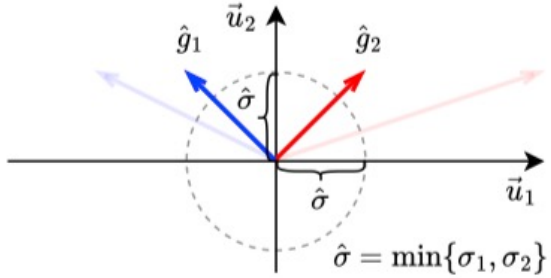
PCGrad: not 雨露均沾

Independent Component Alignment (Aligned MTL)

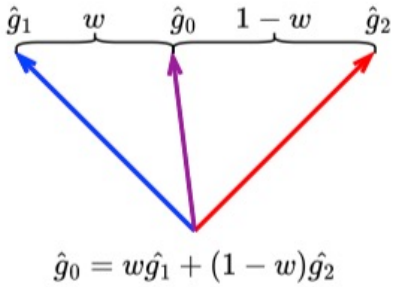
2023, CVPR:



(a) Initial gradients



(b) Gradients aligned via Aligned-MTL



(c) Accumulated MTL gradient

Find: $\hat{\boldsymbol{G}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, where $\boldsymbol{\Sigma} = \sigma \boldsymbol{I}$ $\kappa(\hat{\boldsymbol{G}}) = 1$

must be orthogonal with equal singular values:

Minimizing the condition number of the linear system of gradients leads to mitigating dominance and conflicts within this system.

Find new orthogonal matrix

$$\hat{\boldsymbol{G}} \quad \min_{\hat{\boldsymbol{G}}} \|\boldsymbol{G} - \hat{\boldsymbol{G}}\|_F^2 \quad \text{s.t.} \quad \hat{\boldsymbol{G}}^\top \hat{\boldsymbol{G}} = \boldsymbol{I}$$

Algorithm 1 Gradient matrix alignment

Require: $G \in \mathbb{R}^{|\theta| \times T}$ – gradient matrix,
$w \in \mathbb{R}^T$ – task importance

/* Compute task space Gram matrix */
$\boldsymbol{M} \leftarrow \boldsymbol{G}^\top \boldsymbol{G}$
/* Compute eigenvalues and eigenvectors of M */
$(\lambda, \boldsymbol{V}) \leftarrow eigh(\boldsymbol{M})$
$\boldsymbol{\Sigma}^{-1} \leftarrow diag\left(\sqrt{\frac{1}{\lambda_1}}, \cdots \sqrt{\frac{1}{\lambda_R}}\right)$
/* Compute balance transformation */
$\boldsymbol{B} \leftarrow \sqrt{\lambda_R}\boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{V}^\top$
$\boldsymbol{\alpha} \leftarrow \boldsymbol{B}\boldsymbol{w}$
return $\boldsymbol{G}\boldsymbol{\alpha}$

Independent Component Alignment (Aligned MTL)

2023, CVPR:

Measurement of the system stability: Condition number $\kappa(\boldsymbol{G}) = \frac{\sigma_{max}}{\sigma_{min}}$.

max/min singular values
=1 is the best

linear combination of task gradients $\boldsymbol{g} = \boldsymbol{G}\boldsymbol{w}$

Optimal direction

$$\boldsymbol{g}_i = \nabla \mathcal{L}_i(\boldsymbol{\theta}) \quad \boldsymbol{G} = \{\boldsymbol{g}_1, \cdots, \boldsymbol{g}_T\}$$

$$\hat{\boldsymbol{G}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$$

$\boldsymbol{G}^\top\boldsymbol{G}$ and $\boldsymbol{G}\boldsymbol{G}^\top$

$$\hat{\boldsymbol{G}} = \sigma\boldsymbol{U}\boldsymbol{V}^\top = \sigma \underbrace{\boldsymbol{U}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^\top}_{\text{Parameter space}}\boldsymbol{G} = \sigma\boldsymbol{G}\underbrace{\boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{V}^\top}_{\text{Task space}}$$

Use min sigma

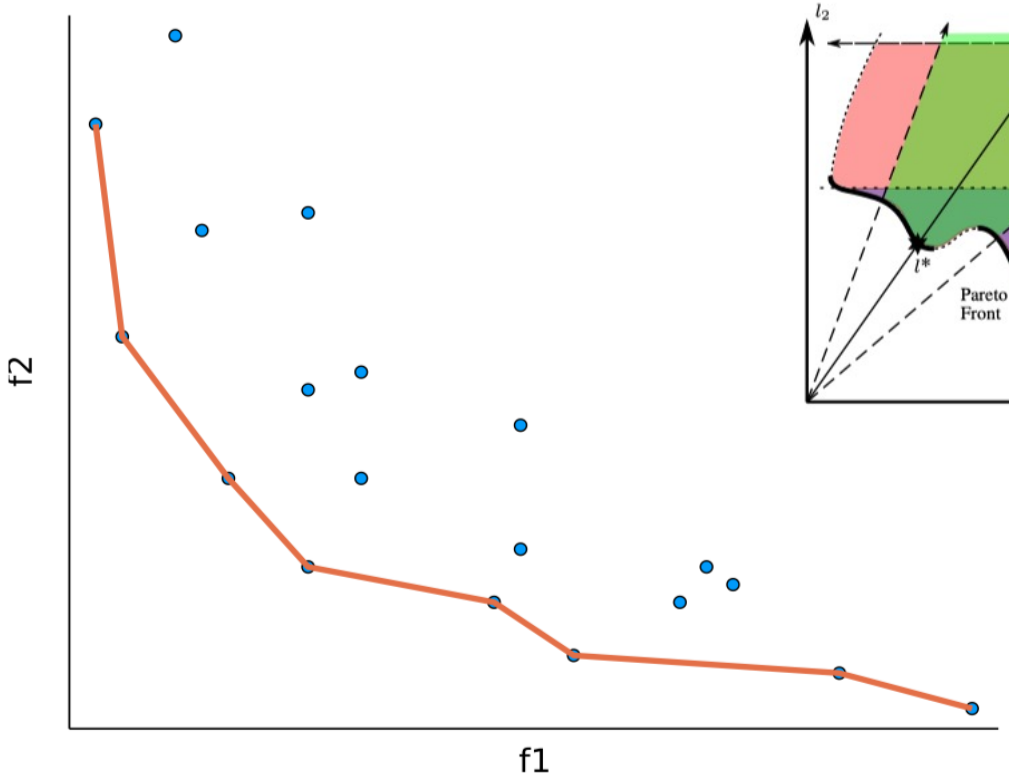
1. minimizing the **condition number** of the gradients system leads to mitigating dominance and conflicts within this system.
2. The initial matrix G (k(G)>1) has the conflict grad. w.r.t tasks, we use SVD to balance them (use singular values to achieve condition number = 1)
3. Force the same descent on the orthogonal basis, (not on the orginal grad. direction)

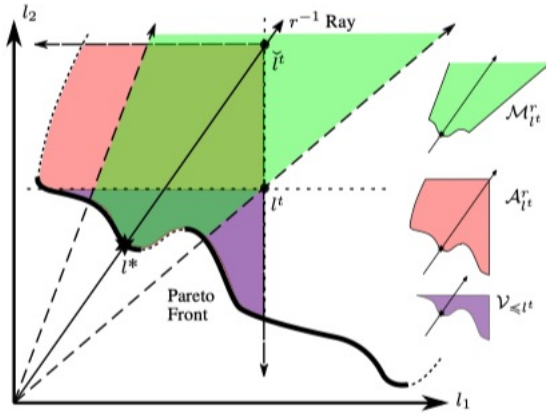
Part 3 - Find the Pareto Set (Frontier)

Typical Dilemma:

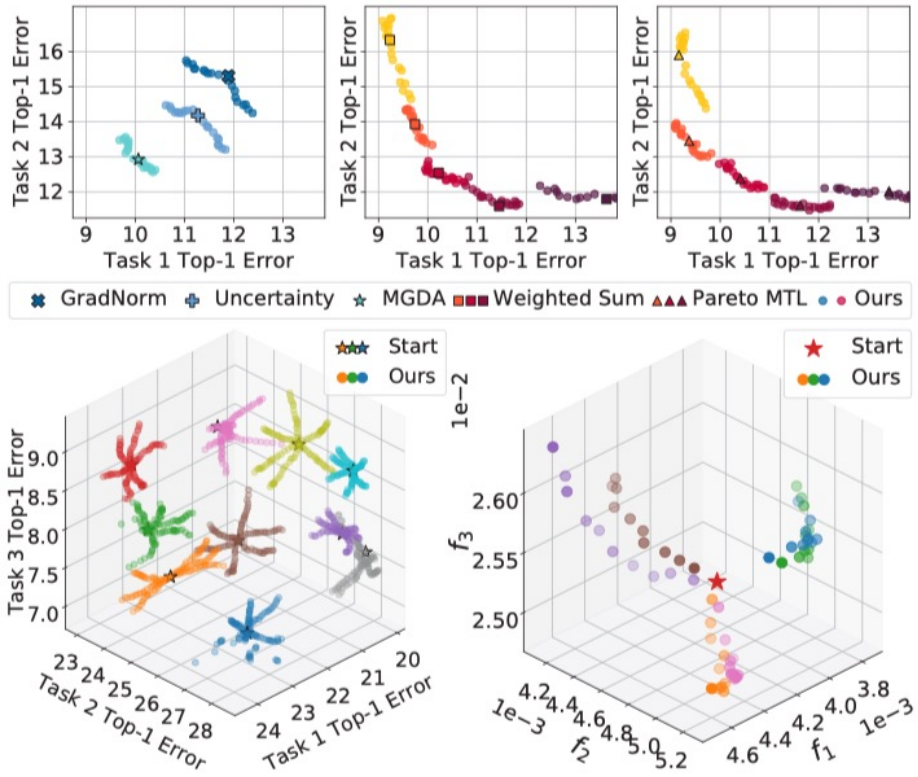
1. Most real-life problems involve decisions that depend on multiple and conflicting criteria.
2. There is usually no single solution that simultaneously minimizes all the objective functions

e.g., 1. Time and Risk; 2. Settling Time and Robustness





Pareto Set and Pareto Front



Q&A

Thanks for your time!