Short overview on Multi-Task Learning and Gradient Surgery

Alex Fedorov

April 10, 2020

Informal Problem Definitions

The multi-task learning problem: Learn all of the tasks more quickly or more proficiently than learning them independently.

The meta-learning problem: Given data/experience on previous tasks, learn a new task more quickly and/or more proficiently.

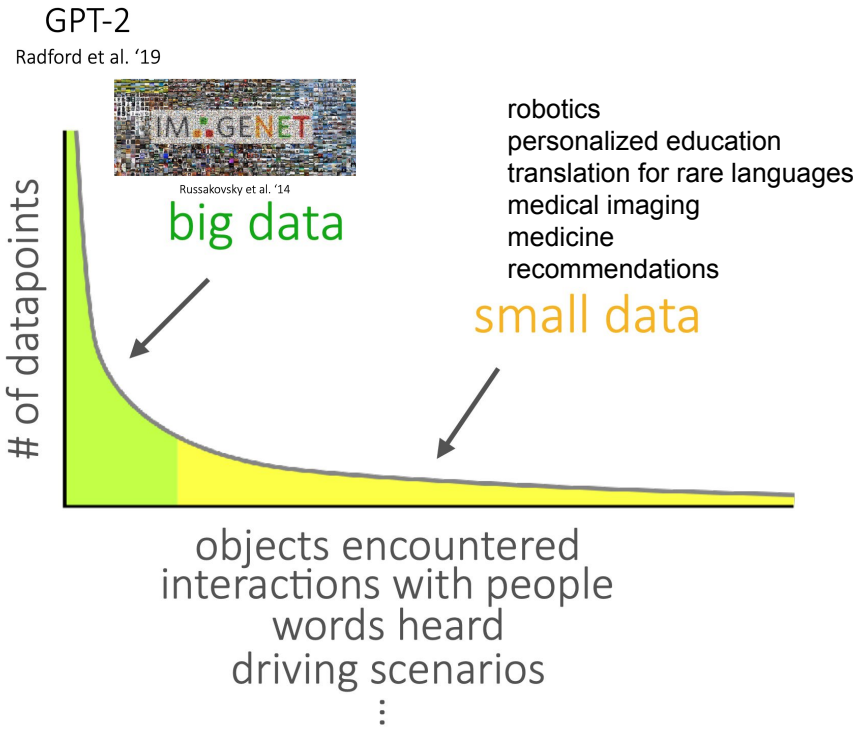
https://cs330.stanford.edu/slides/cs330_lecture1.pdf

Outline

- Part 1 - Multi Task learning
- Part 2 - Multi Task as multi-objective optimization

Part I

Why deep multi-task?



- **Small data**
- **Impractical** to learn **from scratch** for each disease, each robot, each person, each language, each task
- We want to learn **quickly** something new

https://cs330.stanford.edu/slides/cs330_lecture1.pdf

Critical Assumption

The bad news: Different tasks need to share some structure.

If this doesn't hold, you are better off using single-task learning.

The good news: There are many tasks with shared structure!



Even if the tasks are seemingly unrelated:

- The **laws of physics** underly real data.
- People are **all organisms with intentions**.
- The **rules of English** underly English language data.
- Languages all develop for **similar purposes**.

This leads to far greater structure than random tasks.

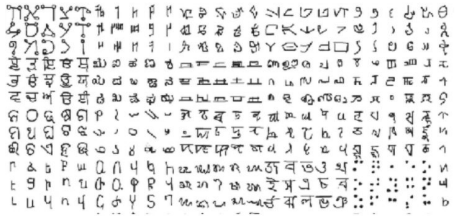
https://cs330.stanford.edu/slides/cs330_lecture1.pdf

Examples

Multi-task classification: $\mathscr{L}_i$ same across all tasks

e.g. per-language handwriting recognition

e.g. personalized spam filter



Multi-label learning: $\mathscr{L}_i, p_i(\mathbf{x})$ same across all tasks

e.g. CelebA attribute recognition

e.g. scene understanding

$$L_{\text{tot}} = w_{\text{depth}} L_{\text{depth}} + w_{\text{kpt}} L_{\text{kpt}} + w_{\text{normals}} L_{\text{normals}}$$

When might $\mathscr{L}_i$ vary across tasks?

- mixed discrete, continuous labels across tasks
- if you care more about one task than another

Age ?
Gender ?
Diagnosis ?
Site ?

https://raminnazer.com/products/rainbow-brain-skull

https://cs330.stanford.edu/slides/cs330_lecture1.pdf

Multitask Learning*

RICH CARUANA

Multitask Learning (MTL) is an inductive transfer mechanism whose principle goal is to improve generalization performance. MTL improves generalization by leveraging the domain-specific information contained in the training signals of *related* tasks. It does this by training tasks in parallel while using a shared representation. In effect, the training signals for the extra tasks serve as an inductive bias. Section 1.2 argues that inductive transfer is important if we wish to scale tabula rasa learning to complex, real-world tasks. Section 1.3 presents the simplest method we know for doing multitask inductive transfer, adding extra tasks (i.e., extra outputs) to a backpropagation net. Because the MTL net uses a shared hidden layer trained in parallel on all the tasks, what is learned for each task can help other tasks be learned better. Section 1.4 argues that it is reasonable to view training signals as an inductive bias when they are used this way.

Caruana, 1997

Is Learning The n-th Thing Any Easier Than Learning The First?

Sebastian Thrun[1]

They are often able to generalize correctly even from a single training example [2, 10]. One of the key aspects of the learning problem faced by humans, which differs from the vast majority of problems studied in the field of neural network learning, is the fact that humans encounter a whole stream of learning problems over their entire lifetime. When faced with a new thing to learn, humans can usually exploit an enormous amount of training data and experiences that stem from other, related learning tasks. For example, when learning to drive a car, years of learning experience with basic motor skills, typical traffic patterns, logical reasoning, language and much more precede and influence this learning task. The transfer of knowledge across learning tasks seems to play an essential role for generalizing accurately, particularly when training data is scarce.

Thrun, 1998

On the Optimization of a Synaptic Learning Rule

Samy Bengio Yoshua Bengio Jocelyn Cloutier Jan Gecsei

Université de Montréal, Département IRO

This paper presents a new approach to neural modeling based on the idea of using an automated method to optimize the parameters of a synaptic learning rule. The synaptic modification rule is considered as a parametric function. This function has *local* inputs and is the same in many neurons. We can use standard optimization methods to select appropriate parameters for a given type of task. We also present a theoretical analysis permitting to study the *generalization* property of such parametric learning rules. By generalization, we mean the possibility for the learning rule to learn to solve *new* tasks. Experiments were performed on three types of problems: a

Bengio et al. 1992

https://cs330.stanford.edu/slides/cs330_lecture1.pdf

Doesn't multi-task learning reduce to single-task learning?

$$\mathcal{D} = \bigcup \mathcal{D}_i \qquad \mathcal{L} = \sum \mathcal{L}_i$$

https://cs330.stanford.edu/slides/cs330_lecture1.pdf

Doesn't multi-task learning reduce to single-task learning?

$$\mathcal{D} = \bigcup \mathcal{D}_i \qquad \mathcal{L} = \sum \mathcal{L}_i$$

Doesn't multi-task learning reduce to single-task learning?

Yes, it can! Aggregating the data across tasks & learning a single model is one approach to multi-task learning.

But, we can often do better! Exploit the fact that we know that data is coming from different tasks.

https://cs330.stanford.edu/slides/cs330_lecture1.pdf

Auxiliary tasks

- Related task
- Adversarial
- Hints
- Focusing attention
- Quantization smoothing
- Predicting inputs
- Using the future to predict the present
- Representation learning

Machine Learning Multi-Task Learning

- Block-sparse regularization
  - block-sparse and element-wise sparse regularization (Jalali et al., NIPS, 2010)
  - distributed version of group-sparse regularization (Liu et al, arXiv:1612.04022)
- Task relationship
  - Clustering constraints (Evgeniou et al., 2005, JMLR)
    - Extension with variance within and between clusters (Jacob, NIPS 2008)
  - Many more extension in Ruders, 2017

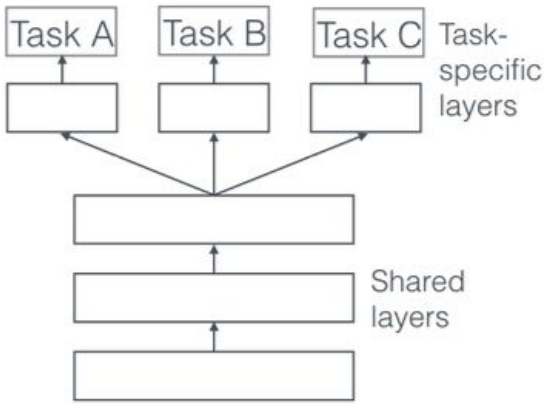
$$\Omega = \|\bar{a}\|^2 + \frac{\lambda}{T}\sum_{t=1}^{T}\|a_{\cdot,t} - \bar{a}\|^2$$

An Overview of Multi-Task Learning in Deep Neural Networks. Ruder, 2017

Neural Architectural solutions

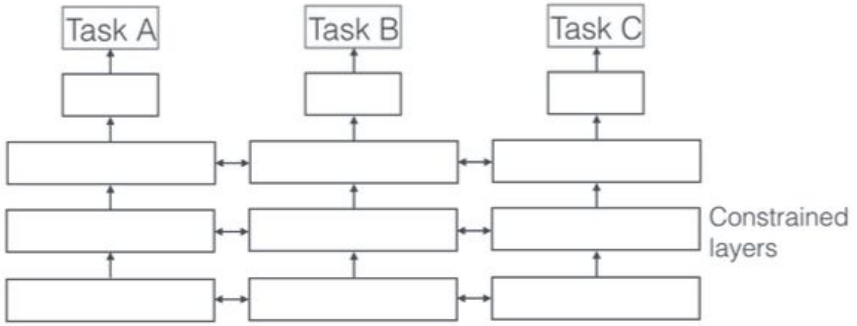
Hard parameter sharing

- Goes back to Caruana, 1993
- Baxter, 1997: More tasks less overfitting
- Breaks down if tasks are not closely related or require reasoning on different levels.



Soft parameter sharing

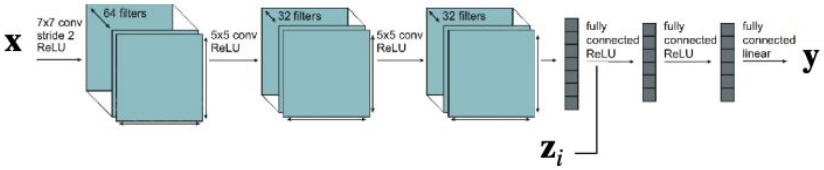
- Duong et al., 2015 (l2 norm)
- Yang and Hospedales, 2017b (trace norm)
- **Learning to share**



$$\min_{\theta^{sh},\theta^1,\ldots,\theta^T} \sum_{i=1}^{T} \mathscr{L}_i(\{\theta^{sh},\theta^i\},\mathscr{D}_i) + \underbrace{\sum_{t'=1}^{T} \|\theta^t - \theta^{t'}\|}_{\text{“soft parameter sharing”}}$$

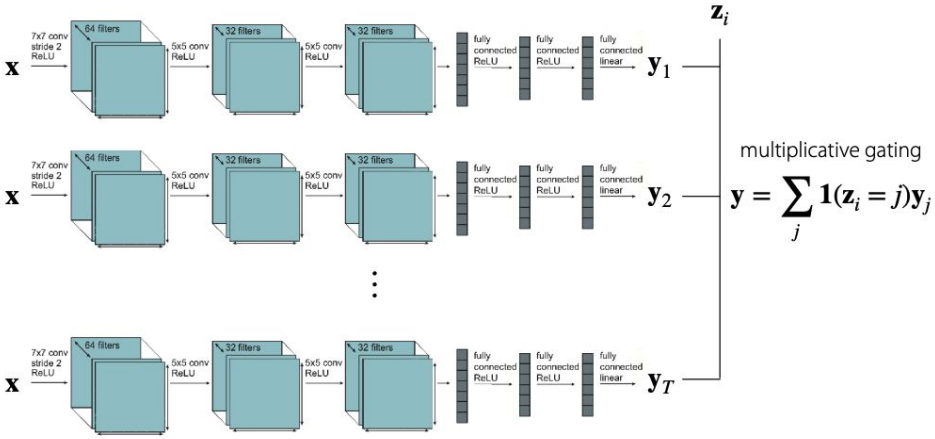
Conditioning on the task

- **Goal:** Find parameters θ of a model f_θ which solve $\min_\theta \mathbb{E}_{\mathcal{T}_i \sim p(\mathcal{T})}[\mathcal{L}_i(\theta)]$.
- Multi-task loss as $\mathcal{L} = \sum_i \mathcal{L}_i(\theta)$
- Gradients $g_i = \nabla \mathcal{L}_i(\theta)$
- Task-conditioned model $f_\theta(y|x, z_i)$



Concatenate $\mathbf{z}_i$ with input and/or activations

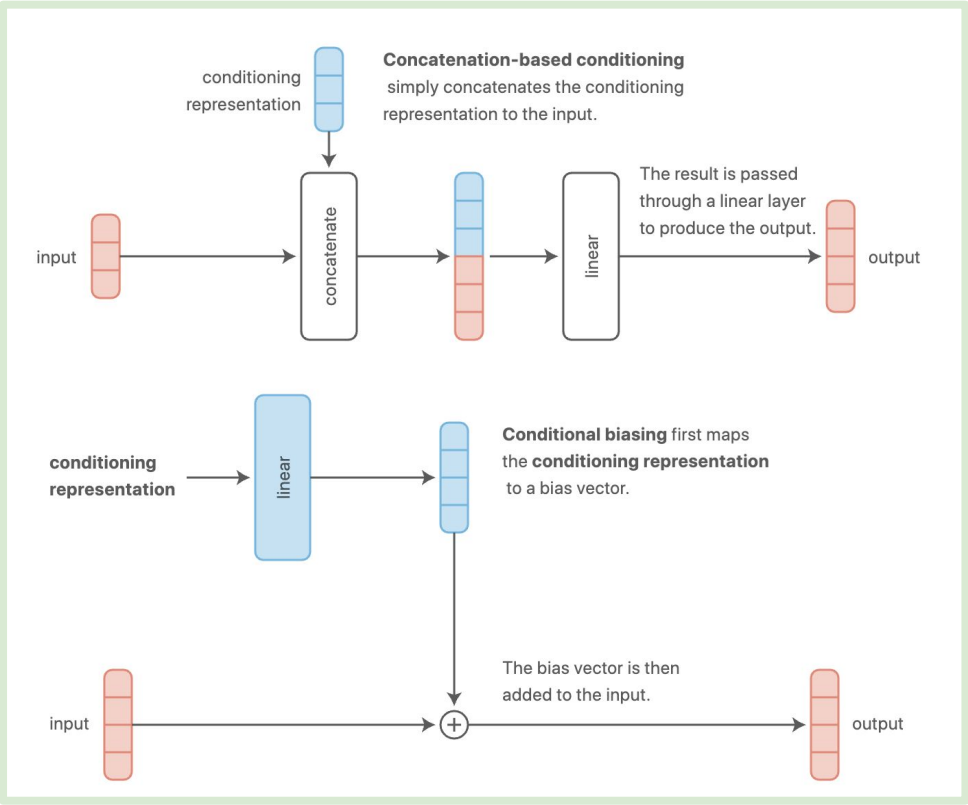
all parameters are shared
except
the parameters directly following $\mathbf{z}_i$

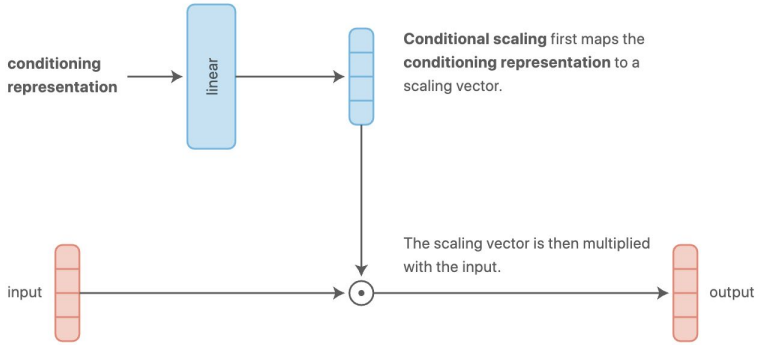


—> independent training within a single network!
with no shared parameters

https://cs330.stanford.edu/slides/cs330_lecture1.pdf

Conditioning types





Unfortunately, these design decisions are like neural network architecture tuning: - problem dependent - largely guided by intuition or knowledge of the problem - currently more of an art than a science

https://distill.pub/2018/feature-wise-transformations/

https://cs330.stanford.edu/slides/cs330_lecture1.pdf

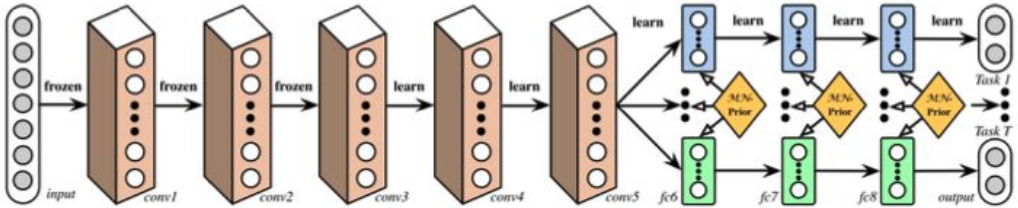


Figure 3: A Deep Relationship Network with shared convolutional and task-specific fully connected layers with matrix priors [Long and Wang, 2015]

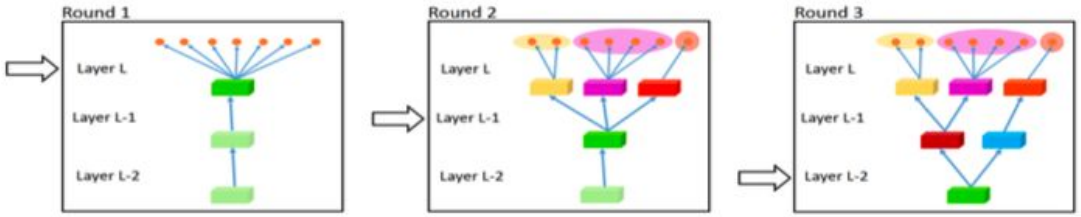


Figure 4: The widening procedure for fully-adaptive feature sharing [Lu et al., 2016]

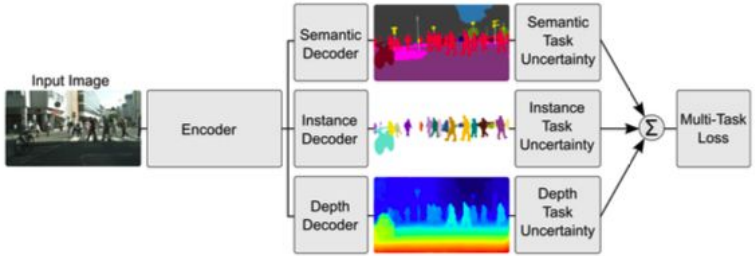


Figure 7: Uncertainty-based loss function weighting for multi-task learning [Kendall et al., 2017]

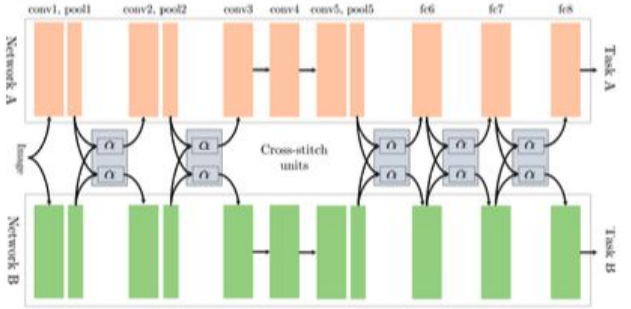


Figure 5: Cross-stitch networks for two tasks [Misra et al., 2016]

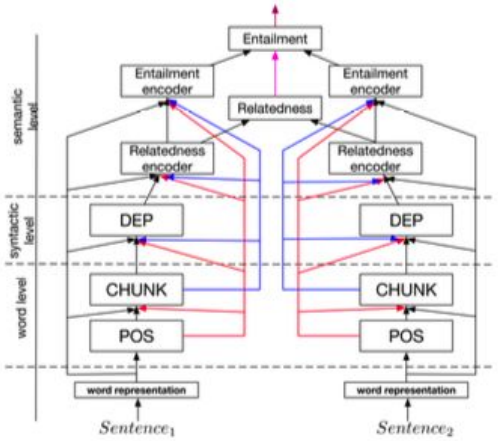


Figure 6: A Joint Many-Task Model [Hashimoto et al., 2016]

Why does it work?

- Implicit data augmentation
  - Learning one task may only distinguish only one type of noise
  - Learning two task simultaneously is able to learn a more general representation
- Attention focusing
  - Tasks can help the model to focus on relevant features
- Eavesdropping
  - Some features are easy to learn for one task, but hard for other task
  - To read *hints (Learning from hints in neural networks, Abu-Mostafa, 1990)*
- Representation bias
  - Some feature are useful for multiple task as long they share something
- Regularization
  - Reduces risk to fit random noise

An Overview of Multi-Task Learning in Deep Neural Networks. Ruder, 2017

Challenges

1) Negative transfer
   a) optimization challenges
      i) caused by cross-task interference
      ii) tasks may learn at different rates
   b) limited representational capacity
      i) multi-task networks often need to be much larger than their single-task counterparts

If you have negative transfer, **share less** across tasks.

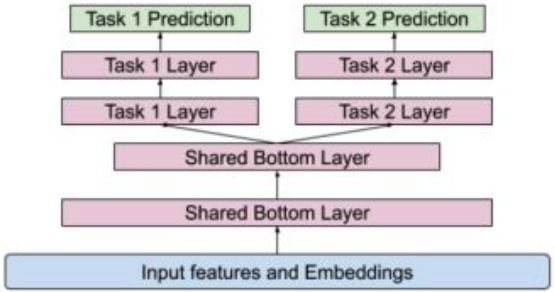
2) Overfitting
   a) Multi-task learning <-> a form of regularization

Solution: Share more.

https://cs330.stanford.edu/slides/cs330_lecture1.pdf

Recommending What Video to Watch Next: A Multitask Ranking System
Z. Zhao et al., 2019

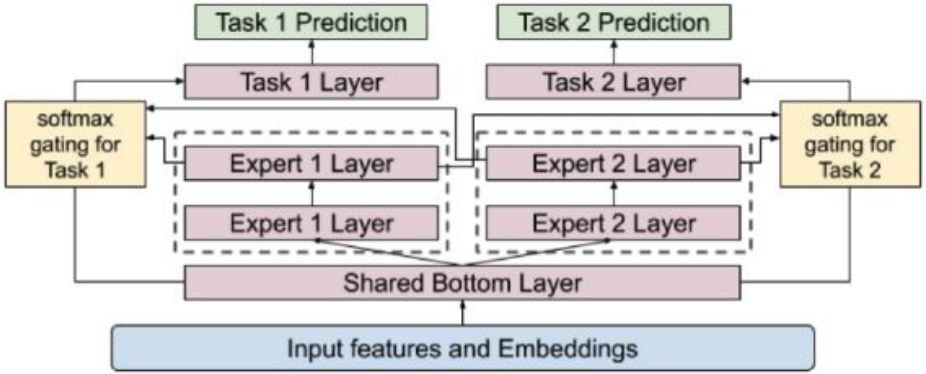
Basic option: "Shared-Bottom Model"
(i.e. multi-head architecture)



(a) Shared-Bottom Model with shared bottom hidden layers and separate towers for two tasks.

-> harm learning when correlation between tasks is low

Instead: use a form of soft-parameter sharing
"Multi-gate Mixture-of-Experts (MMoE)"



(b) Multi-gate Mixture-of-Expert Model with one shared bottom layer and separate hidden layers for two tasks.

https://cs330.stanford.edu/slides/cs330_lecture1.pdf

Part 2

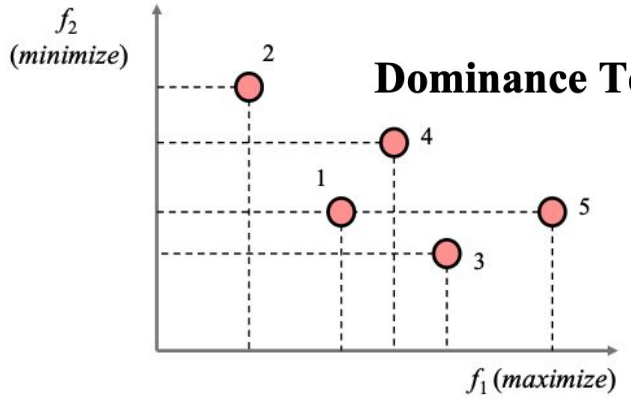
Multi-Objective Optimization

- Involve more than one objective function that are to be minimized or maximized
- Answer is set of solutions that define the best tradeoff between competing objectives

- Mathematically

$$\begin{aligned} &\min/\max \ f_m(\boldsymbol{x}), && m = 1, 2, \cdots, M \\ &\text{subject to } g_j(\boldsymbol{x}) \geq 0, && j = 1, 2, \cdots, J \\ &h_k(\boldsymbol{x}) = 0, && k = 1, 2, \cdots, K \\ &x_i^{(L)} \leq x_i \leq x_i^{(U)}, && i = 1, 2, \cdots, n \end{aligned}$$

($x_i^{(L)}$: lower bound, $x_i^{(U)}$: upper bound)

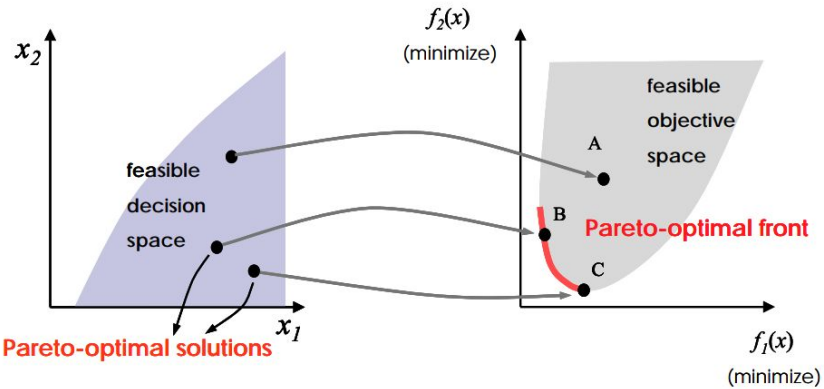


- 1 Vs 2: 1 dominates 2
- 1 Vs 5: 5 dominates 1
- 1 Vs 4: Neither solution dominates

https://engineering.purdue.edu/~sudhoff/ee630/Lecture09.pdf

Pareto Optimal

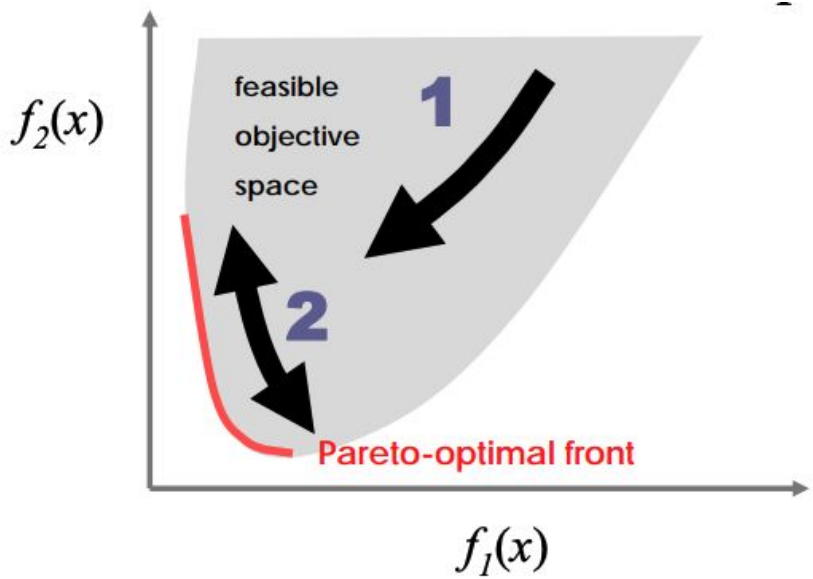
- **Non-dominated solution set**
  - Given a set of solutions, the non-dominated solution set is a set of all the solutions that are not dominated by any member of the solution set
- The non-dominated set of the entire feasible decision space is called the **Pareto-optimal set**
- The boundary defined by the set of all point mapped from the Pareto optimal set is called the **Pareto optimal front**



https://engineering.purdue.edu/~sudhoff/ee630/Lecture09.pdf

Goals in MOO

- Find set of solutions as close as possible to **Pareto optimal front**
- To find a set of solutions as diverse as possible



https://engineering.purdue.edu/~sudhoff/ee630/Lecture09.pdf

Classic methods

- Weighted Sum Method
- ε-Constraint Method
- Weighted Metric Method
- Many Genetic Algorithms
  - SPEA

https://engineering.purdue.edu/~sudhoff/ee630/Lecture09.pdf

Weighted Sum Method

- Scalarize a set of objectives into a single objective by adding each objective pre-multiplied by a user-supplied weight
- Weight of an objective is chosen in proportion to the relative importance of the objective

$$\begin{aligned} \textbf{minimize} \quad & F(\boldsymbol{x}) = \sum_{m=1}^{M} w_m f_m(\boldsymbol{x}), \\ \textbf{subject to} \quad & g_j(\boldsymbol{x}) \geq 0, \qquad j = 1, 2, \cdots, J \\ & h_k(\boldsymbol{x}) = 0, \qquad k = 1, 2, \cdots, K \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)}, \qquad i = 1, 2, \cdots, n \end{aligned}$$

- simple
- It is difficult to set the weight vectors to obtain a **Pareto-optimal solution** in a desired region in the objective space
- It can not find certain **Pareto-optimal solutions** in the case of a *nonconvex objective space*

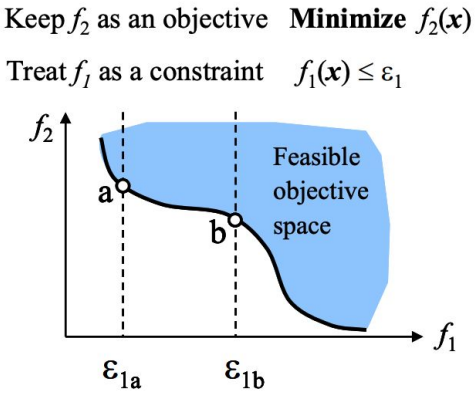
https://engineering.purdue.edu/~sudhoff/ee630/Lecture09.pdf

ε-Constraint Method

- Haimes et. al. 1971
- Keep just one of the objective and restricting the rest of the objectives within user-specific values

- Applicable to either convex or non-convex problems
- The ε vector has to be chosen carefully so that it is within the minimum or maximum values of the individual objective function

$$
\begin{aligned}
&\textbf{minimize} && f_\mu(\boldsymbol{x}), \\
&\textbf{subject to} && f_m(\boldsymbol{x}) \le \varepsilon_m, && m = 1, 2, \cdots, M \text{ and } m \ne \mu \\
& && g_j(\boldsymbol{x}) \ge 0, && j = 1, 2, \cdots, J \\
& && h_k(\boldsymbol{x}) = 0, && k = 1, 2, \cdots, K \\
& && x_i^{(L)} \le x_i \le x_i^{(U)}, && i = 1, 2, \cdots, n
\end{aligned}
$$



https://engineering.purdue.edu/~sudhoff/ee630/Lecture09.pdf

Weighted Metric Method

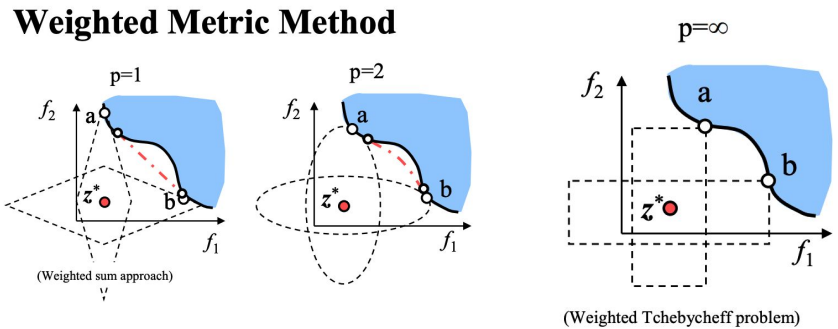
- Combine multiple objectives using the weighted distance metric of any solution from the ideal solution z*

$$\textbf{minimize} \quad l_{\mathbf{p}}(\boldsymbol{x}) = \left(\sum_{m=1}^{M} w_m \left| f_m(\boldsymbol{x}) - z_m^* \right|^p \right)^{1/p},$$

$$\textbf{subject to} \quad g_j(\boldsymbol{x}) \geq 0, \qquad j = 1, 2, \cdots, J$$

$$h_k(\boldsymbol{x}) = 0, \qquad k = 1, 2, \cdots, K$$

$$x_i^{(L)} \leq x_i \leq x_i^{(U)}, \qquad i = 1, 2, \cdots, n$$

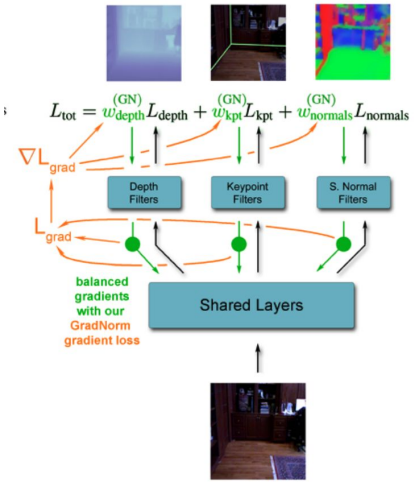


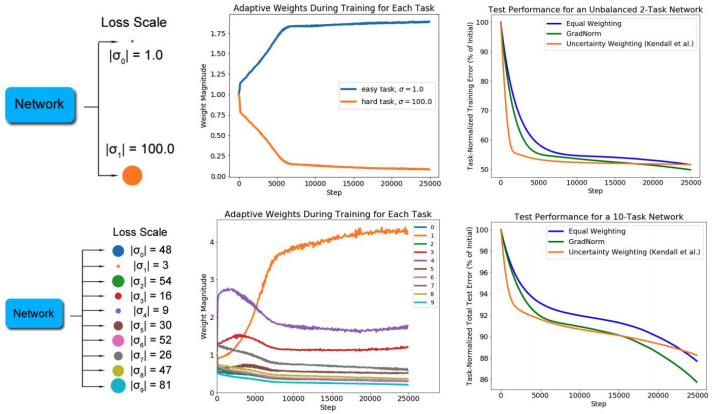
- Advantage
  - Weighted Tchebycheff metric guarantees finding all Pareto-optimal solution with ideal solution z* □
- Disadvantage □
  - Requires knowledge of minimum and maximum objective values
  - Requires z* which can be found by independently optimizing each objective functions
  - For small p, not all Pareto-optimal solutions are obtained
  - As p increases, the problem becomes non-differentiable

https://engineering.purdue.edu/~sudhoff/ee630/Lecture09.pdf

GradNorm Chen et al. 2017

- places gradient norms for different tasks on a common scale
- dynamically adjusts gradient norms so different tasks train at similar rates





$$L_{\text{grad}}(t; w_i(t)) = \sum_i \left| G_W^{(i)}(t) - \overline{G}_W(t) \times [r_i(t)]^\alpha \right|_1$$

- 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.

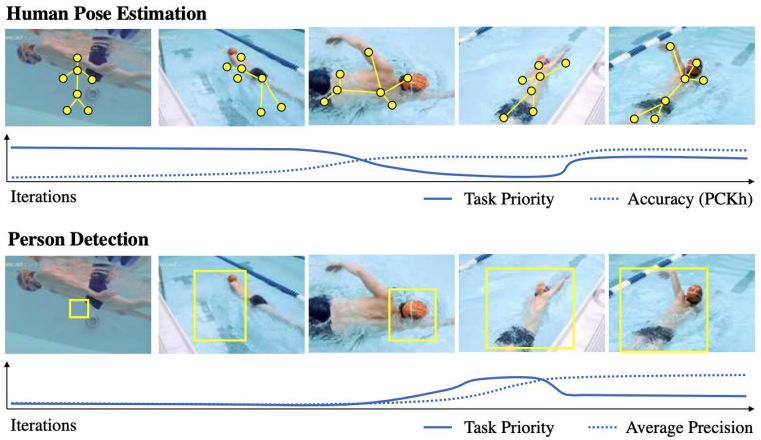
https://arxiv.org/pdf/1711.02257.pdf

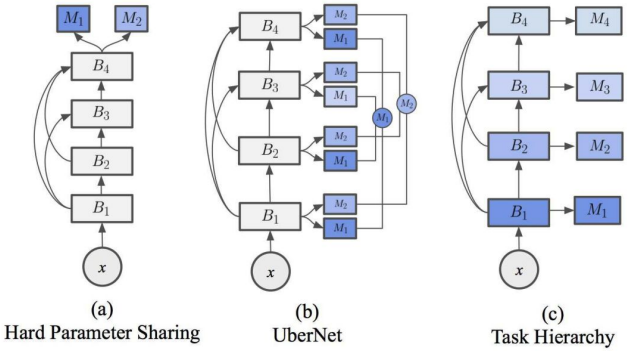
Dynamic Task Prioritization for Multitask Learning
Guo et al. ECCV 2018

$$\mathrm{FL}(p_c;\gamma_0) = -(1-p_c)^{\gamma_0}\log(p_c)$$

$$\bar{\kappa}_t^{(\tau)} = \alpha\kappa_t^{(\tau)} + (1-\alpha)\bar{\kappa}_t^{(\tau-1)}$$

$$\mathcal{L}_{\mathrm{DTP}}(\cdot) = \mathcal{L}^*_{\mathrm{Total}}(\cdot) = \sum_{t=1}^{|T|}\mathrm{FL}(\bar{\kappa}_t;\gamma_t)\mathcal{L}^*_t(\cdot)$$





http://openaccess.thecvf.com/content_ECCV_2018/papers/Michelle_Guo_Focus_on_the_ECCV_2018_paper.pdf

Multi-Task Learning as Multi-Objective Optimization
Sener & Koltun, NIPS 2018

- Multi-task learning as multi-objective optimization with the overall objective of finding a Pareto optimal solution
- Multiple Gradient Descent Algorithm
- Frank-Wolfe solver
- The optimization problem defined is equivalent to finding a minimum-norm point in the convex hull of the set of input points.

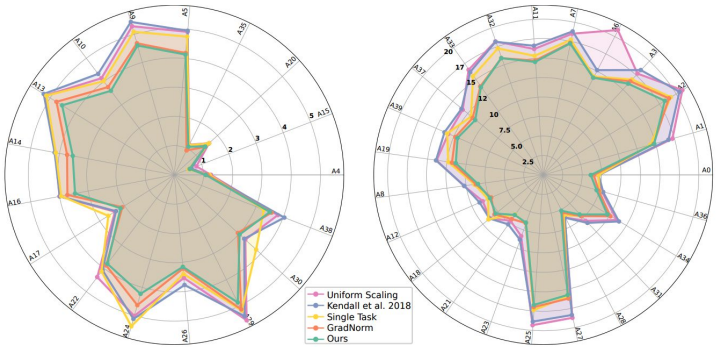


Figure 2: Radar charts of percentage error per attribute on CelebA (Liu et al., 2015b). Lower is better. We divide attributes into two sets for legibility: easy on the left, hard on the right. Zoom in for details.

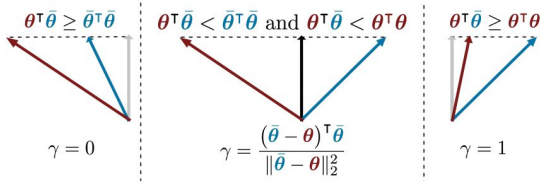


Figure 1: Visualisation of the min-norm point in the convex hull of two points ($\min_{\gamma\in[0,1]} \|\gamma\boldsymbol{\theta} + (1-\gamma)\bar{\boldsymbol{\theta}}\|_2^2$). As the geometry suggests, the solution is either an edge case or a perpendicular vector.

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

Algorithm 2 Update Equations for MTL

for $t = 1$ **to** T **do**
  $\boldsymbol{\theta}^t = \boldsymbol{\theta}^t - \eta\nabla_{\boldsymbol{\theta}^t}\hat{\mathcal{L}}^t(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^t)$ ▷ Gradient descent on task-specific parameters
end for
$\alpha^1, \ldots, \alpha^T$ = FrankWolfeSolver($\boldsymbol{\theta}$) ▷ Solve (3) to find a common descent direction
$\boldsymbol{\theta}^{sh} = \boldsymbol{\theta}^{sh} - \eta\sum_{t=1}^T \alpha^t\nabla_{\boldsymbol{\theta}^{sh}}\hat{\mathcal{L}}^t(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^t)$ ▷ Gradient descent on shared parameters

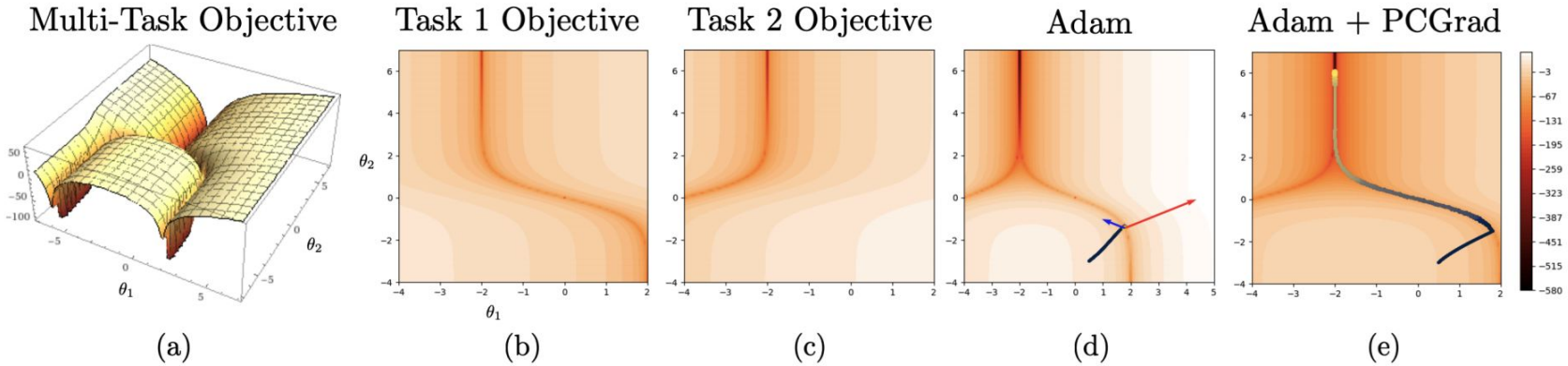
procedure FrankWolfeSolver($\boldsymbol{\theta}$)
  Initialize $\boldsymbol{\alpha} = (\alpha^1, \ldots, \alpha^T) = (\frac{1}{T}, \ldots, \frac{1}{T})$
  Precompute $\mathbf{M}$ st. $\mathbf{M}_{i,j} = \left(\nabla_{\boldsymbol{\theta}^{sh}}\hat{\mathcal{L}}^i(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^i)\right)^\intercal\left(\nabla_{\boldsymbol{\theta}^{sh}}\hat{\mathcal{L}}^j(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j)\right)$
  repeat
    $\hat{t} = \arg\min_r \sum_t \alpha^t\mathbf{M}_{rt}$
    $\hat{\gamma} = \arg\min_\gamma \left((1-\gamma)\boldsymbol{\alpha} + \gamma\boldsymbol{e}_{\hat{t}}\right)^\intercal\mathbf{M}\left((1-\gamma)\boldsymbol{\alpha} + \gamma\boldsymbol{e}_{\hat{t}}\right)$ ▷ Using Algorithm 1
    $\boldsymbol{\alpha} = (1-\hat{\gamma})\boldsymbol{\alpha} + \hat{\gamma}\boldsymbol{e}_{\hat{t}}$
  until $\hat{\gamma} \sim 0$ **or** Number of Iterations Limit
  return $\alpha^1, \ldots, \alpha^T$
end procedure

https://arxiv.org/pdf/1810.04650.pdf

Tragic Triad

Hypothesis: Conflict is detrimental when conflicting gradients coincide with high positive curvature and a large difference in gradient magnitudes.

- Conflicting Gradients
- Dominating Gradients
- High Curvature

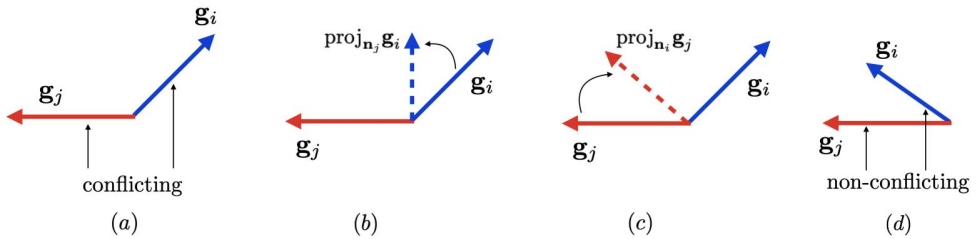


Conflicting Gradients

- Detrimental to the learning
- Simple solution is too average gradients
  - Still worsen performance
  - Might not solve different scales
- To deal with scales is trade-off hyperparameter

If two gradients are conflicting, we alter the gradients by projecting each onto the normal plane of the other, preventing the interfering components of the gradient from being applied to the network.

$$\Phi(\mathbf{g}_i, \mathbf{g}_j) = \frac{2\|\mathbf{g}_i\|_2\|\mathbf{g}_j\|_2}{\|\mathbf{g}_i\|_2^2 + \|\mathbf{g}_j\|_2^2}$$



PCGrad

Algorithm 1 PCGrad Update Rule

Require: Model parameters θ, task minibatch $\mathcal{B} = \{\mathcal{T}_k\}$

$\mathbf{g}_k \leftarrow \nabla_\theta \mathcal{L}_k(\theta) \;\; \forall k$

for $\mathcal{T}_i \in \mathcal{B}$ **do**

  for $\mathcal{T}_j \overset{\text{uniformly}}{\sim} \mathcal{B} \setminus \mathcal{T}_i$ in random order **do**

    if $\mathbf{g}_i^T \mathbf{g}_j < 0$ **then**

      // Subtract the projection of $\mathbf{g}_i$ *onto* $\mathbf{g}_j$

      Set $\mathbf{g}_i = \mathbf{g}_i - \frac{\mathbf{g}_i \cdot \mathbf{g}_j}{\|\mathbf{g}_j\|^2} \mathbf{g}_j$

    end if

  end for

  Store $\mathbf{g}_i^{\text{PC}} = \mathbf{g}_i$

end for

return update $\Delta\theta = \mathbf{g}^{\text{PC}} = \sum_i \mathbf{g}_i^{\text{PC}}$

PCGrad

- Pros
  - Agnostic to the model architecture
  - Shown that improves performance on multiple experiments
- Cons
  - What about Pareto?
  - Comparison is limited to the multi-task architectural solutions and method is not compared with GradNorm, convex hull + FrankWolfe

Thank you