Adapting Auxiliary Losses Using Gradient Similarity

Yunshu Du*, Wojciech M. Czarnecki*, Siddhant M. Jayakumar, Razvan Pascanu, Balaji Lakshminarayanan

DeepMind

yunshu.du@wsu.edu, {lejlot, sidmj, razp, balajiln}@google.com

1. Problem Setup

Given a **main task** of interest and an **auxiliary task** which is not of direct interest, how do we weight the auxiliary loss? The typical multi-task approach uses:

$$\arg\min_{\boldsymbol{\theta},\boldsymbol{\phi}_{main},\boldsymbol{\phi}_{aux}} \mathcal{L}_{main}(\boldsymbol{\theta},\boldsymbol{\phi}_{main}) + \lambda\mathcal{L}_{aux}(\boldsymbol{\theta},\boldsymbol{\phi}_{aux})$$

However, this could be sub-optimal since we care only about performance on the main task. E.g., the auxiliary loss might help initially but hurt later. We want to solve the following problem:

$$\arg\min_{\lambda^{(t)}} \mathcal{L}_{main}\left(\boldsymbol{\theta}^{(t)} - \alpha\nabla_{\boldsymbol{\theta}}(\mathcal{L}_{main} + \lambda^{(t)}\mathcal{L}_{aux}), \boldsymbol{\phi}^{(t)}_{main} - \alpha\nabla_{\boldsymbol{\phi}_{main}}\mathcal{L}_{main}\right)$$

Question: How to automatically adapt the auxiliary loss so that it does not hurt the main loss?

2. Using Gradient Cosine Similarity To Adapt Auxiliary Loss

Motivating example: main function $\mathcal{L}_{main} = (\theta - 10)^2$ auxiliary function $\mathcal{L}_{aux} = \theta^2$

Given two tasks with shared parameters $\boldsymbol{\theta}$ & task-specific parameters $\boldsymbol{\phi}_{main}$ & $\boldsymbol{\phi}_{aux}$), the update:

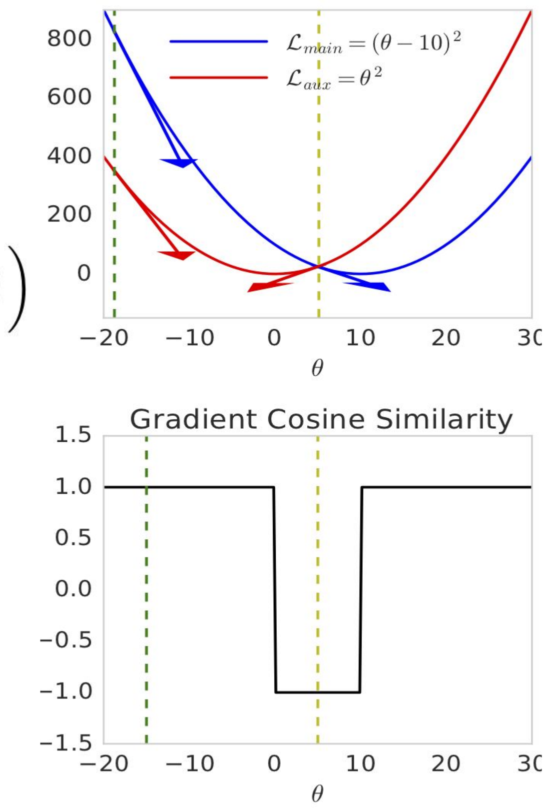
$$\boldsymbol{\theta}^{(t+1)} := \boldsymbol{\theta}^{(t)} - \alpha^{(t)}\left(\nabla_{\boldsymbol{\theta}}\mathcal{L}_{main}(\boldsymbol{\theta}^{(t)}) + \nabla_{\boldsymbol{\theta}}\mathcal{L}_{aux}(\boldsymbol{\theta}^{(t)})\max\left(0, \cos(\nabla_{\boldsymbol{\theta}}\mathcal{L}_{main}(\boldsymbol{\theta}^{(t)}), \nabla_{\boldsymbol{\theta}}\mathcal{L}_{aux}(\boldsymbol{\theta}^{(t)}))\right)\right)$$

$$\boldsymbol{\phi}^{(t+1)}_{main} := \boldsymbol{\phi}^{(t)}_{main} - \alpha^{(t)}\nabla_{\boldsymbol{\phi}_{main}}\mathcal{L}_{main}(\boldsymbol{\Theta}^{(t)}) \quad \text{and} \quad \boldsymbol{\phi}^{(t+1)}_{aux} := \boldsymbol{\phi}^{(t)}_{aux} - \alpha^{(t)}\nabla_{\boldsymbol{\phi}_{aux}}\mathcal{L}_{aux}(\boldsymbol{\Theta}^{(t)})$$

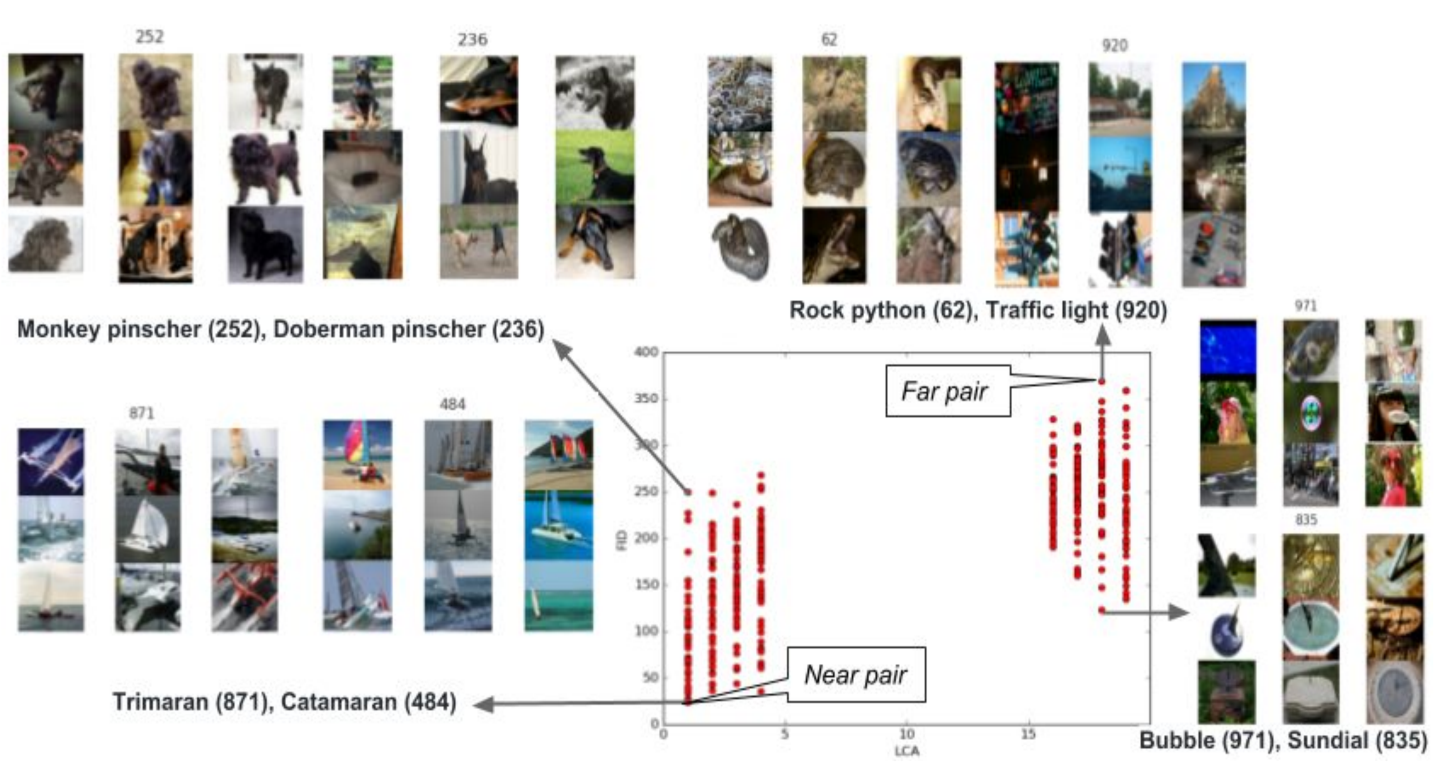
leads to convergence to local minimum of $\mathcal{L}_{main}$ w.r.t. $(\boldsymbol{\theta}, \boldsymbol{\phi}_{main})$ given small enough $\alpha^{(t)}$.

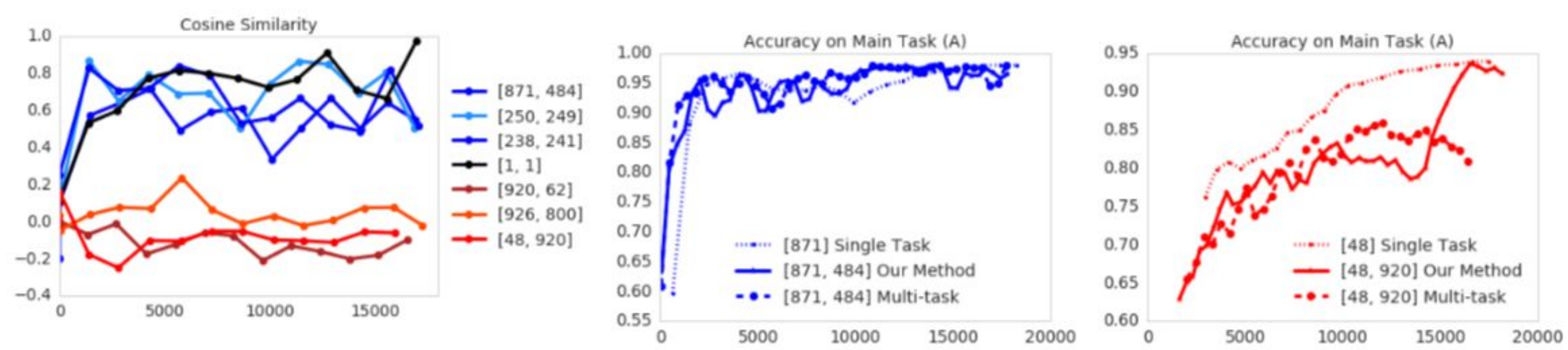
Weighted version: Weight the auxiliary loss by cosine similarity (as above).

Unweighted version: Use aux loss when cos > threshold and ignore otherwise.



3. Supervised Learning Using Pairs Of Imagenet Classes





(a) Cosine similarities on near pairs (blue) and far pairs (red).

(b) Near pair: 871 vs. 484

(c) Far pair: 48 vs. 920

Ground truth of task similarity: use Least Common Ancestor (LCA) and Frechet Inception Distance (FID) between ImageNet classes.

Near pair: the most similar, such as *Trimaran* and *Catamaran*

Far pair: the least similar, such as *Rock python* and *Traffic light*

Given a pair of classes (A, B), we define the main task as (A vs. rest) and the auxiliary task as (B vs. rest).

Figure (a): we validate that **near pairs have high cosine similarity** and **far pairs have low cosine similarity**.

Figure (b): in a **near pair**, our method uses auxiliary to learn faster and recovers the performance of multi-task

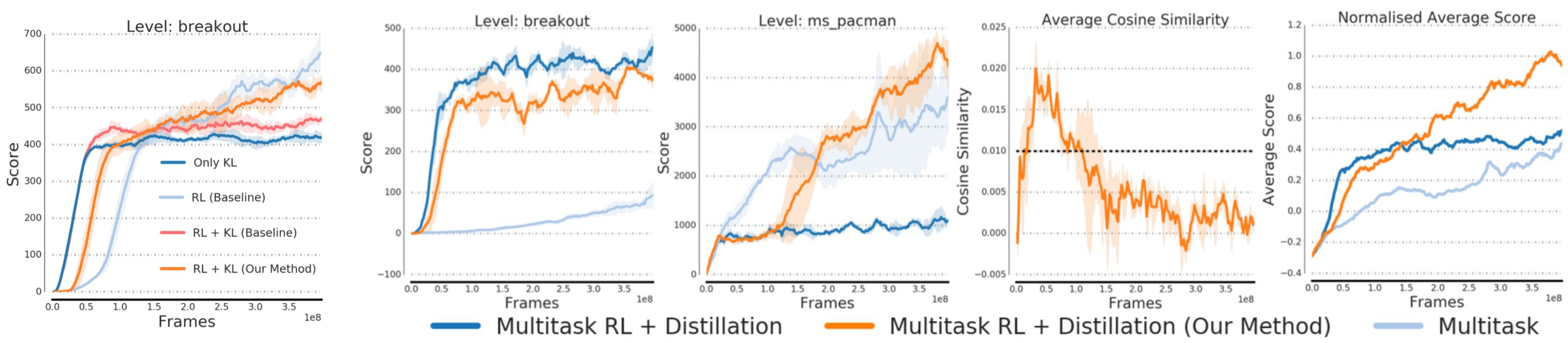
Figure (c): in a **far pair**, our method successfully ignores auxiliary and recovers the performance of single task

Our method automatically uses (ignores) auxiliary when it helps (hurts), achieving the best of both worlds.

4. Reinforcement Learning On Imperfect-Teacher Distillation

Single task on Breakout

Multi-task on Breakout and Ms. PacMan



Single task on Breakout: the main task is Breakout, the auxiliary task is a sub-optimal pre-trained Breakout teacher

Only KL: solely following the teacher leads to sub-optimal solutions

RL (Baseline): single task learning without the teacher

RL + KL (Baseline): the teacher only helps initially

Our Method: **uses the teacher's knowledge when it helps initially and ignores when it hurts later on**

Multi-task on Breakout and Ms. PacMan: the main task is multi-task Breakout + Ms. PacMan, the auxiliary task is a sub-optimal pre-trained Breakout teacher

Multi-task: learns Ms. PacMan at the expense of Breakout

Multi-task RL + Distillation: the teacher helps Breakout but hurts Ms. PacMan

Our Method: **Ms. PacMan ignores the teacher when it hurts; both Breakout and Ms. Pacman learn well**

5. Summary

- Proposed gradient cosine similarity as a simple yet effective way to automatically adapt the auxiliary task to help (& not hurt) the main task.
- Experiments on ImageNet and Atari show empirical success; paper contains additional experiments on cross-domain distillation tasks.
- Paper shows theoretical guarantees on the convergence to local optimum of the main task.

Potential issues and Future directions

- Guarantees convergence to local optimum of the main task but not faster convergence.
- Extend theory to optimizers that rely on statistics of the gradients or second order information (e.g., Adam or RMSprop).
- Apply our method to settings where the auxiliary task hurts initially but helps later.