RPL CV / DL Reading Group

26 March 2019

Multitask Learning as Multiobjective Optimization

Ozan Sener and Vladlen Koltun

Outline

- Motivation
- Related Work
- M
- Model definition
- Training details
- Results
- MTL challenges analysis
- Conclusions & Future Work

Multi-task learning

Definition: jointly learn T tasks, sharing inductive bias across them designing a parameterized hypothesis that shares some parameters across tasks.

Strategies:

- Soft-sharing: All parameters specific to each task, jointly constrained.
- Hard-sharing: Part of the parameters fully-shared between tasks.

Multi-task learning

Definition: jointly learn T tasks, sharing inductive bias across them designing a parameterized hypothesis that shares some parameters across tasks.

Strategies:

- Soft-sharing: All parameters specific to each task, jointly constrained.
- **Hard-sharing: Part of the parameters fully-shared between tasks.**

Multi-task learning

Definition: jointly learn T tasks, sharing inductive bias across them designing a parameterized hypothesis that shares some parameters across tasks.

Strategies:

- Soft-sharing: All parameters specific to each task, jointly constrained.
- **Hard-sharing: Part of the parameters fully-shared between tasks.**
  - ***Using deep neural networks as model***

Multi-task learning

Usually,

weighted sum of empirical risk for each task

$$\min_{\substack{\boldsymbol{\theta}^{sh}, \\ \boldsymbol{\theta}^1,\ldots,\boldsymbol{\theta}^T}} \sum_{t=1}^{T} c^t \hat{\mathcal{L}}^t(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^t)$$

… cannot handle competing tasks

… ct can be static or dynamic

... uniform weights, found as hyperparameter, heuristics

Multi-task learning

Uncertainty weighting (Kendall et al. 2018)

1. Predict heteroscedastic uncertainty model as mean $\hat{y}_t$ and variance σ_t^2 for each task *t* as new model output
2. Weight loss $\mathcal{L}_t$ with $1 / 2\sigma_t^2$

GradNorm (Chen et al. 2018)

$$G_W^{(i)}(t) = ||\nabla_W w_i(t) L_i(t)||_2$$

1. For each task, compute gradient wrt selected layer and its norm.
2. Compute average gradient norm $\overline{G}_W(t)$
3. Compute rel. training speed as loss / avg. loss. $r_i(t) = \tilde{L}_i(t) / E_{\text{task}}[\tilde{L}_i(t)]$
4. Compute loss to learn loss weights

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

5. Update loss weights, then model parameters

Multi-task learning as multi-objective opt.

Instead,

MTL as multi-objective optimization, optimizing set of possibly contrasting objectives......

$$\min_{\substack{\boldsymbol{\theta}^{sh}, \\ \boldsymbol{\theta}^1,\ldots,\boldsymbol{\theta}^T}} \mathbf{L}(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^T)$$

New goal: *Find Pareto optimal solution, not dominated by any other solution.*

A solution $\boldsymbol{\theta}$ *dominates a solution* $\bar{\boldsymbol{\theta}}$ *if* $\hat{\mathcal{L}}^t(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^t) \leq \hat{\mathcal{L}}^t(\bar{\boldsymbol{\theta}}^{sh}, \bar{\boldsymbol{\theta}}^t)$ *for all tasks* t *and* $\mathbf{L}(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^T) \neq \mathbf{L}(\bar{\boldsymbol{\theta}}^{sh}, \bar{\boldsymbol{\theta}}^1, \ldots, \bar{\boldsymbol{\theta}}^T)$.

Multi-task learning as multi-objective opt.

Focus on gradient-based multi-objective optimization...

MGDA - Multiple Gradient Descent Algorithm

...well-suited for multitask deep networks trained with stochastic gradient descent. But two issues need solving:

1. **Does not scale** to high-dimensional gradients
2. Requires separate computation of gradients for each task, i.e. one **backward pass per task**

MGDA - Multiple Gradient Descent Algorithm

Use *Karush-Kuhn-Tucker (KKT)* conditions to find common descent direction of shared parameters for all objectives, necessary for optimality.

- There exist $\alpha^1, \ldots, \alpha^T \geq 0$ such that $\sum_{t=1}^{T} \alpha^t = 1$ and $\sum_{t=1}^{T} \alpha^t \nabla_{\boldsymbol{\theta}^{sh}} \hat{\mathcal{L}}^t(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^t) = 0$
- For all tasks t, $\nabla_{\boldsymbol{\theta}^t} \hat{\mathcal{L}}^t(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^t) = 0$

Solution satisfying these is Pareto stationary but not necessarily Pareto optimal.

MGDA - Multiple Gradient Descent Algorithm

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

- Solution is zero:
  - Pareto stationary
- Non-zero:
  - Gives a descent direction improving all objectives

Equivalent to finding a minimum-norm solution in the convex hull of the set of solutions.

MGDA - Multiple Gradient Descent Algorithm

Case for 2 tasks has analytical solution:

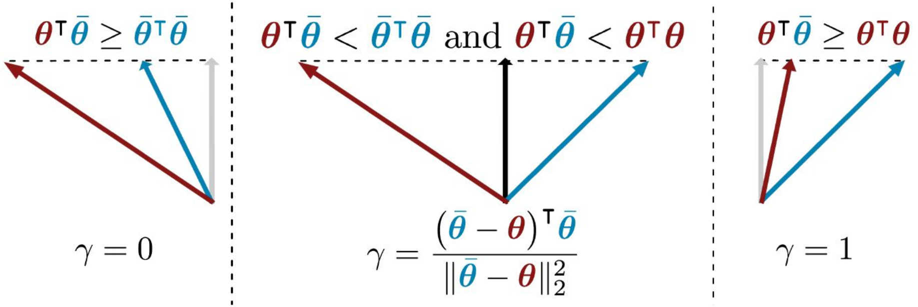


Figure 1: Visualisation of the min-norm point in the convex hull of two points ($\min_{\gamma\in[0,1]} \|\gamma\boldsymbol{\theta} + (1-\gamma)\bar{\boldsymbol{\theta}}\|_2^2$). As the geometry suggests, the solution is either an edge case or a perpendicular vector.

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

1: **if** $\boldsymbol{\theta}^\intercal\bar{\boldsymbol{\theta}} \geq \boldsymbol{\theta}^\intercal\boldsymbol{\theta}$ **then**
2: $\quad\gamma = 1$
3: **else if** $\boldsymbol{\theta}^\intercal\bar{\boldsymbol{\theta}} \geq \bar{\boldsymbol{\theta}}^\intercal\bar{\boldsymbol{\theta}}$ **then**
4: $\quad\gamma = 0$
5: **else**
6: $\quad\gamma = \frac{(\bar{\boldsymbol{\theta}}-\boldsymbol{\theta})^\intercal\bar{\boldsymbol{\theta}}}{\|\boldsymbol{\theta}-\bar{\boldsymbol{\theta}}\|_2^2}$
7: **end if**

MGDA - Multiple Gradient Descent Algorithm

Use 2D case as subroutine for line search in Frank-Wolfe optimizer.

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
$\alpha^1, \dots, \alpha^T = \textsc{FrankWolfeSolver}(\boldsymbol{\theta})$ ▷ 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 $\textsc{FrankWolfeSolver}(\boldsymbol{\theta})$
  Initialize $\boldsymbol{\alpha} = (\alpha^1, \dots, \alpha^T) = (\frac{1}{T}, \dots, \frac{1}{T})$
  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
    $\hat{t} = \arg\max_r \sum_t \alpha^t \mathbf{M}_{rt}$
    $\hat{\gamma} = \arg\min_\gamma \left((1-\gamma)\boldsymbol{\alpha} + \gamma \boldsymbol{e}_{\hat{t}}\right)^\intercal \mathbf{M}\left((1-\gamma)\boldsymbol{\alpha} + \gamma \boldsymbol{e}_{\hat{t}}\right)$ ▷ 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 $\alpha^1, \dots, \alpha^T$
end procedure

MGDA - Multiple Gradient Descent Algorithm

Use 2D case as subroutine for line search in Frank-Wolfe optimizer.

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
$\alpha^1, \dots, \alpha^T$ = 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 $\boldsymbol{\alpha} = (\alpha^1, \dots, \alpha^T) = (\frac{1}{T}, \dots, \frac{1}{T})$
    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
        $\hat{t} = \arg\max_r \sum_t \alpha^t \mathbf{M}_{rt}$
        $\hat{\gamma} = \arg\min_\gamma \left((1-\gamma)\boldsymbol{\alpha} + \gamma \boldsymbol{e}_{\hat{t}}\right)^\intercal \mathbf{M} \left((1-\gamma)\boldsymbol{\alpha} + \gamma \boldsymbol{e}_{\hat{t}}\right)$ ▷ 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 $\alpha^1, \dots, \alpha^T$
end procedure

MGDA - Multiple Gradient Descent Algorithm

Use 2D case as subroutine for line search in Frank-Wolfe optimizer.

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
$\alpha^1, \dots, \alpha^T = \text{FRANKWOLFESOLVER}(\boldsymbol{\theta})$ ▷ Solve (3) to find a common descent direction
$\boldsymbol{\theta}^{sh} = \boldsymbol{\theta}^{sh} - \eta \sum_{t=1}^{T} \alpha^t \nabla_{\boldsymbol{\theta}^{sh}} \hat{\mathcal{L}}^t(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^t)$ $\boxed{\text{▷ Gradient descent on shared parameters}}$

procedure FRANKWOLFESOLVER($\boldsymbol{\theta}$)
  Initialize $\boldsymbol{\alpha} = (\alpha^1, \dots, \alpha^T) = (\frac{1}{T}, \dots, \frac{1}{T})$
  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
    $\hat{t} = \arg\max_r \sum_t \alpha^t \mathbf{M}_{rt}$
    $\hat{\gamma} = \arg\min_\gamma \left((1-\gamma)\boldsymbol{\alpha} + \gamma \boldsymbol{e}_{\hat{t}}\right)^\intercal \mathbf{M}\left((1-\gamma)\boldsymbol{\alpha} + \gamma \boldsymbol{e}_{\hat{t}}\right)$ ▷ 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 $\alpha^1, \dots, \alpha^T$
end procedure

MGDA - Multiple Gradient Descent Algorithm

Frank-Wolfe solver typically converges within a few iterations, negligible addition to training time.

But we still need T backward passes...

MGDA - Upper Bound

For the encoder-decoder(s) case where…

$$f^t(\mathbf{x}; \boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^t) = (f^t(\cdot; \boldsymbol{\theta}^t) \circ g(\cdot; \boldsymbol{\theta}^{sh}))(\mathbf{x}) = f^t(g(\mathbf{x}; \boldsymbol{\theta}^{sh}); \boldsymbol{\theta}^t)$$

Encoder

Decoder
for task t

The shared representation can be expressed as

$$\mathbf{z}_i = g(\mathbf{x}_i; \boldsymbol{\theta}^{sh})$$

MGDA - Upper Bound

Upper bound of objective of min-norm point problem…

$$\left\| \sum_{t=1}^{T} \alpha^t \nabla_{\boldsymbol{\theta}^{sh}} \hat{\mathcal{L}}^t(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^t) \right\|_2^2 \leq \left\| \frac{\partial \mathbf{Z}}{\partial \boldsymbol{\theta}^{sh}} \right\|_2^2 \left\| \sum_{t=1}^{T} \alpha^t \nabla_{\mathbf{Z}} \hat{\mathcal{L}}^t(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^t) \right\|_2^2$$

Does not depend on the alphas

Can be computed with a single backward pass

If $\frac{\partial \mathbf{Z}}{\partial \boldsymbol{\theta}^{sh}}$ *is full-rank (tasks not linearly related),*

optimizing UB is equivalent.

MGDA - Upper Bound

The rest of the algorithm is exactly the same

(:

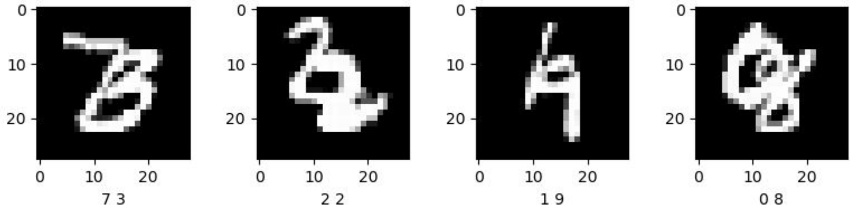
Experiments

Baselines for all experiments

1. Single-task
2. Uniform weights
3. Weights found through grid-search
4. Uncertainty weighting
5. GradNorm

Experiments

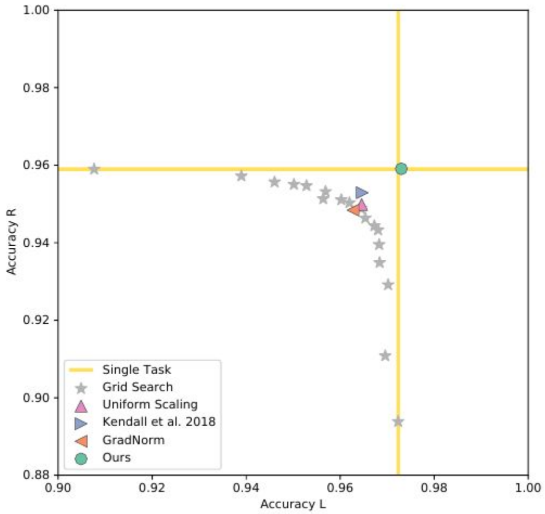
MultiMNIST



2 tasks:

- Classify top-left digit “L”
- Classify bottom-right digit “R”

LeNet-based multi-task network



Experiments

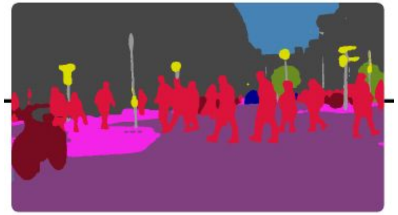
CelebA

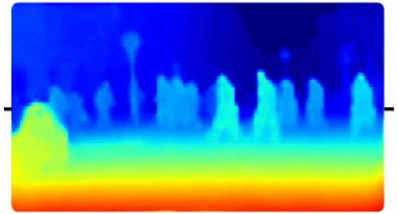
Multi-label classification,
each label is a binary class. task
40 tasks

ResNet-18 encoder, linear decoders.

	Average error
Single task	8.77
Uniform scaling	9.62
Kendall et al. 2018	9.53
GradNorm	8.44
Ours	**8.25**

Experiments



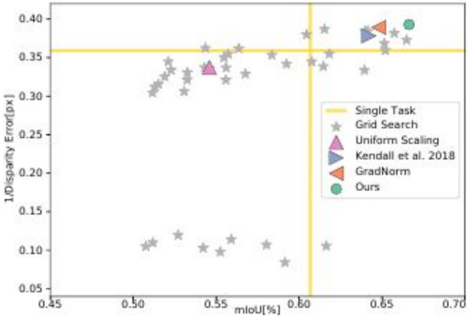


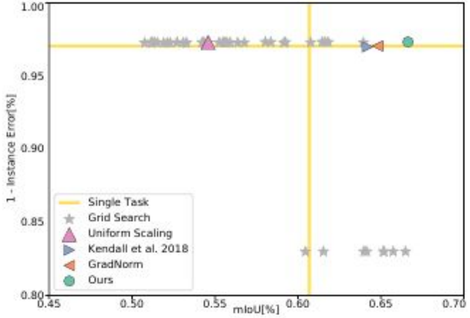
Cityscapes

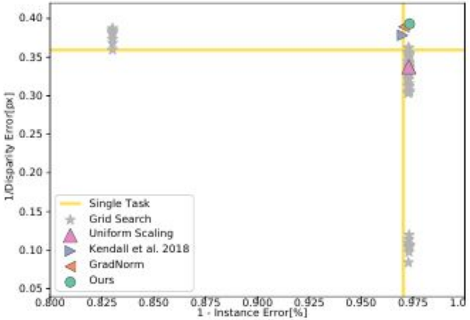
3 scene-understanding tasks:

- Semantic segmentation
- Instance segmentation
- Depth estimation

ResNet-50 encoder,
pyramid pooling decoders







Experiments

Effect of upper bound approximation

Table 2: Effect of the MGDA-UB approximation. We report the final accuracies as well as training times for our method with and without the approximation.

	Scene understanding (3 tasks)				Multi-label (40 tasks)	
	Training time	Segmentation mIoU [%]	Instance error [px]	Disparity error [px]	Training time (hour)	Average error
Ours (w/o approx.)	38.6	66.13	10.28	2.59	429.9	8.33
Ours	**23.3**	**66.63**	**10.25**	**2.54**	**16.1**	**8.25**

60 %

3.7%

Experiments

Effect of upper bound approximation

And *surprisingly* also better accuracies…

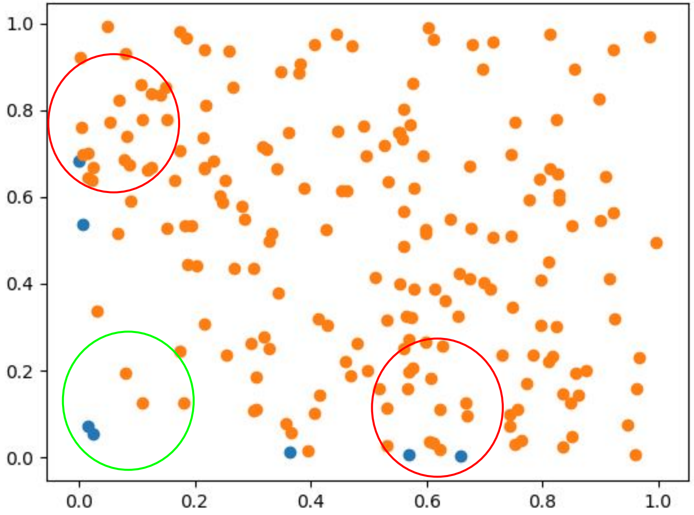
...possibly due to solving problem (min norm point)
in lower-dimensional space
(shared representation instead of parameters)

Conclusions

Applying multi-objective optimization to multi-task learning achieves better results than traditional approaches based on weighted sum of losses.

A method and an approximation with negligible computational overhead are proposed and evaluated on 3 different multi-task problems, showing it is effective on a wide range of scenarios.

Extra!



- MGDA tends to give a shortest path to the Pareto front
  - Not necessarily a balanced Pareto optimal solution

In practice, for cases in which gradient magnitudes differ a lot between tasks, this is important. Need to scale gradients:

- **By the loss**
- By the L2 norm
- Other...