GENERALIZATION FOR ROBOTICS

What is Generalization?

→ Can be working with diff. versions of the state space that hasn't seen before.
→ Perform well on $\boxed{\text{similar}}$ problems
→ Out of distribution:

→ s': validation set (concept of : out of - distribution)

state space (model well trained here)

Why is Generalization Good?

1. Do More with Less (information or structure)
   → The agent works even if it has not exp. that state. → strong form of data effiency

2. We generalize well because we make good use of data.

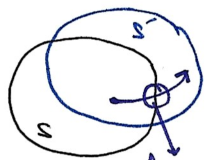
3. Good Agents should generalize to similar situations → discover best ind./causal mechanisms
   → Compression, less deep connections.
   → $\boxed{\text{Occam's Razor}}$

   $y \leftarrow f(x;\theta)$
   x is the cause of y.

4. Avoid Overfitting. ⇒ Data Augmentation + adversial examples as solutions.

   Policy ⇒ max $\mathbb{E}_S(r)$ for state space
   ⇒ However if $\mathbb{E}_{S'}(r)$ low: overfitting in RL

   ⇒ We want the policy to generalize on sequences of states!



from here, once we reach (a_t, s_t) outside of state spaces
⇒ and the rewards ↓
⇒ overfitting.

S: some computation simulation
S': the real world

$$\max \; \mathbb{E}_{\substack{\boxed{s \sim p(s,a)} \\ a \sim \pi(s)}} \left[\sum_{t=0}^{\infty} r(s_t, a_t) \right]$$

$S \leftarrow p_{sim}(s,a)$
$S' \leftarrow p_{real}(s,a)$

Overfitting to the
1. Dynamics (distribution $p(s,a)$) → ∅ use hands after putting gloves
2. Rewards → task missspecifiestion
   ∅ want to be a poker player for example.

5. Avoid Wasting Experience

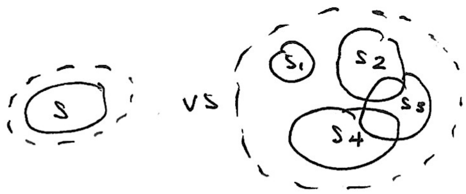
Generalization in Robotics

Overfitting in...
- → To a specific state distribution
- → To the dynamics
- → To Reward functions

erceptual Generalization
- → Learning to overfit appearance + ⇒ with physics changes
- → Method : Data Aug. + Adversial Examples

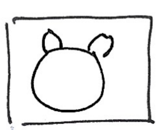
en. in Dynamics
- → $p(s_{t+1} | s_t, a_t)$: forward dynamics can change between tasks
- → Ex: Motor down or breaks.
- → Ex: changes in friction on $\emptyset$ surfaces

ulti-Task Learning
- → We need the agent to train various tasks:



Transfer Learning
- → Distribution $p(y|x)$, $\exists$ lower dim. manifold (z) where $p(y|x) = p(y|z)$
  ↳ lower dimension
- → Want to learn transformation $p(z|x)$

Ex:



x → $\begin{bmatrix} \text{has ears} \\ \text{nose} \\ \vdots \end{bmatrix}$ z

① CNN Model from scratch
↓
Knowledge
↓
② Pretrained CNN

- → If there's an overlap between tasks
  - Training on one improves perf. on others
  - Learning a better $p(z|x)$ for future tasks.

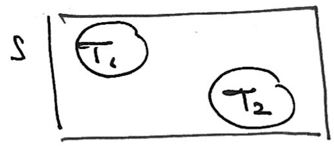
Multi-Task Learning ⇒ learn to be robust to tasks.

init θ to rand. network and $\mathcal{D} \leftarrow \{\}$
while True do
    for $i \in \{0, \dots, n\}$ do
        $T \sim p(T)$ ⇒ Select a task
        for $t \in \{0, \dots, T\}$ do
            Select + Apply Action $a_t = \pi(\cdot | s_t, \theta)$ in $p(s_{t+1} | s_t, a_t, T)$ ⇒ collect data for environment
            Get experience $\underbrace{\{s_t, a_t, r_t, s_{t+1}\}}$ and add to $\mathcal{D}$. ⇒ collect exp. across env.
        end for
    end for
    Update $\boxed{\text{policy } \theta \text{ using } \mathcal{D}}$, then $\mathcal{D} \leftarrow \{\}$
end while

Challenges

- Progress not easy
  - → Goal: increase training speed + gen.
  - → Multi-Task often makes things worse

- Cause: $\boxed{\text{Interference}}$

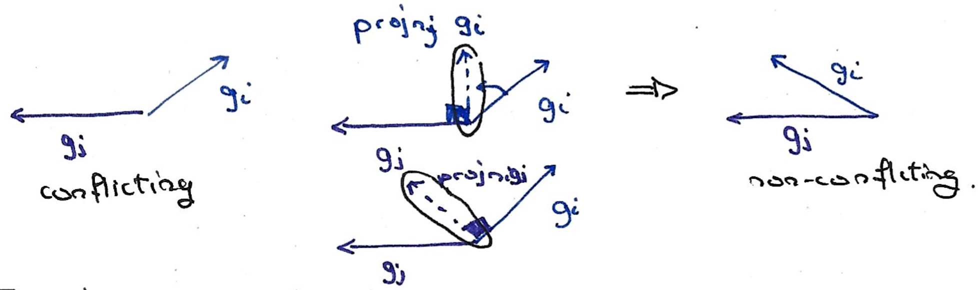


- The agent will focus on easiest task (starts to overfit on some specific tasks)
- Optimization difficulties
  - → Asking policy to learn more
  - → No free lunch theorem

$p(z|x) \approx H(s)$

INTERFERENCE

- Gradients will contradict each other.
- Solution: PCGrad (Projection)

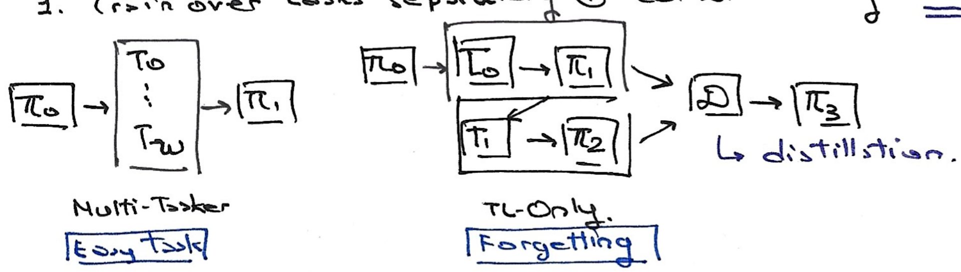


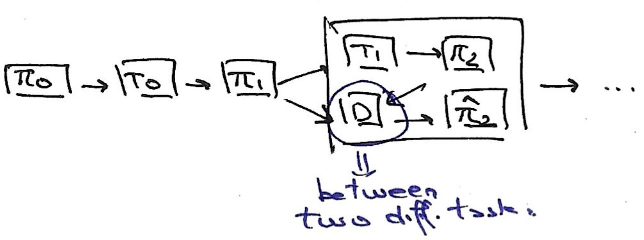
Focusing on Easiest Task

- Common issue in many methods (Options, HRL, Multi-Tasking)
  - → Always check the video results.
- Methods:
  1. Train over tasks separately ⊕ combine using behaviour cloning (Dagger)

Combining new skills together.

↳ Distillation.
doesn't rely on reward functions.





PLAiD (better version and transferability than TL-Only)
→ Avoid forgetting.

⇒ We must «forgetting» and «quality»
- ↳ TL: difficult to predict affectiveness + transfer value function + overfitting specific tasks (↑ forgetting)
- ↳ Plaid: ↓ quality, but ↓ forgetting
- TL: ↑ quality, but ↑ forgetting

Meta Reinforcement Learning.

But before that: Discussion about a paper

PROGRESSIVE RL WITH DISTILLATION FOR MULTI-SKILLED MOTION CONTROL

Combination of Distillation, transfer Learning and MAML.

└ behavior cloning. (Dagger)

Transfer Learning

→ Expert on source T_i ⇒ post-training π_i

→ Student on target task T_{i+1} ⇒ π_{i+1}, trying to match the expert's expectation.

Ex: 2D Biped Controller ⇒ Train on different types of terrains as $\boxed{\text{tasks}}$ ↴

incline, steps, slopes, gaps, etc.

Sequence Learning Schedules.

Issue with the 3 following methods:

ⓐ $\pi_0 \rightarrow \begin{bmatrix} T_0 \\ T_1 \\ \vdots \\ T_{\omega-1} \\ T_\omega \end{bmatrix} \rightarrow \pi_1$

collects easy tasks

MULTI-TASKER

ⓑ $\pi_0 \rightarrow [T_0 \rightarrow \pi_1],\ [T_1 \rightarrow \pi_2],\ \dots,\ [T_{\omega-1} \rightarrow \pi_\omega],\ [T_\omega \rightarrow \pi_{\omega+1}] \rightarrow \mathcal{D} \rightarrow \pi_{\omega+2}$

No experience to remember (⊥ tasks)

PARALLEL

ⓒ $\pi_0 \rightarrow [T_0 \rightarrow \pi_1] \rightarrow [T_1 \rightarrow \pi_2] \rightarrow \dots \rightarrow [T_{\omega-1} \rightarrow \pi_\omega] \rightarrow [T_\omega \rightarrow \pi_{\omega+1}] \rightarrow \mathcal{D} \rightarrow \pi_{\omega+2}$

Forgetting

TL-ONLY

★: Concept of « forgetting »

→ Once we get to Task($\omega-1$) for example, the T_0 will be totally forgotten

→ Hence, if there was overlap between them, no more advantage on them anymore.

ⓓ PLAID : adding a new task each time + distillation, solves « forgetting » (a little bit), better forward transfer abilities.

$\pi_0 \rightarrow T_0 \rightarrow \pi_1 \rightarrow \begin{bmatrix} T_1 \rightarrow \pi_2 \\ \mathcal{D} \leftarrow \hat{\pi}_2 \end{bmatrix} \dots \begin{bmatrix} T_\omega \rightarrow \pi_{\omega+1} \\ \mathcal{D} \leftarrow \hat{\pi}_{\omega+1} \end{bmatrix}$

flat surface... incline... steps...

⇒ Generalization: incorporating new tasks while remembering the previous ones.

→ Mesure effectiveness with forgetting and quality.

→ TL: hard to predict effectiveness + overfitting specific tasks + forgetting

→ Distillation needs some work for PLAID.

MetaRL: learn to learn

- So far, discussed how to generalize to unseen tasks
- What if we can collect a little data from the target task

$p(z|X)$ vs. $p(z|x, \mathcal{D})$ ↳ for z.

RL ↑ Meta-RL ↑

- Collect a little bit more data on the target task and make a big jump in the reward function overall for that task.

[Ex:] Using exp from other Tasks.

- Board games (a new one) ⇒ you have played many before
  ⇒ similar dynamics ⊕ figure out how to maximize pts.
- Meta-Learning: operationalize this process.

FRAMEWORK

- Given a function $f(x;\theta)$ with parameters θ and a loss function ℓ.
  → to predict if we can win the board game.
- Samples $\mathcal{D}_i = (x_i, y_i)$
  → predict win/lose based on the action/config. x_i
  ↳ images/configurations of the board

→ Task-Loss: $\mathcal{L}(\mathcal{D}_i, \theta_i) = \mathbb{E}_{(x_i, y_i) \in \mathcal{D}_i}[\ell(f(x_i;\theta_i), y_i)]$

- In supervised setting: use paired data of input x_i and output y_i
- Goal: Enable fast $\boxed{\text{adaptation}}$ on a diff. set of meta-test tasks not seen during training.
  ↳ How to win frequently in this new board game with a few data from this board?

⇒ Solution: MAML Algorithm

MAML: Model-Agnostic-Meta-Learning. ⇒ LIBRARY: $\boxed{\text{HIGHER}}$

- Fixed distribution of tasks $p(T)$ ⇒ all the board games in the shelf.
  → M tasks drawn from $\{T_i\}_{i=0}^{M}$ distribution
  → Tasks $T_j \sim p(T)$ that provided data $\mathcal{D}_j = \{x_j, y_j\}$ ⇒ $\mathcal{D}_j$: play games for board game "j" (assumes with same ppl)
  → Train θ for a model on $\{T_i\}_{i=0}^{M}$
  → Objective $f(x;\theta)$ is low after few gradient updates on data $\mathcal{D}_j$

$$\hookrightarrow \min_{\theta} \sum_{T_i \sim p(T)} \mathcal{L}((\theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_i^{train}), \mathcal{D}_i^{test})$$

$$= \min_{\theta} \sum_{T_i \sim p(T)} \mathcal{L}(\phi, \mathcal{D}_{T_j}^{val})$$

OTHER METHODS.

Train RNN to injest description and predict a MDP

Train RNN to injest description and predict a policy that will max. rewards

→ Sequence of data that helps you to predict a policy that will max. rewards inside this env.

→ Overfitting

Train optimizer or update function MAML → $\boxed{p(z \mid x, D, \theta)}$

→ Good extrapolation + consistent

→ Hard to determine # samples or gradient steps

Train a representation $p(z|x)$ PEARL

→ Inference problem

→ Overfitting (dosen't generalize well)

Transformers: works well for diff. policy changes (humanoids for example)

GENERALIZATION WITHOUT TASKS.

1. Learn to infer the boundaries.
2. Learn without explicit task boundaries (detect local state distributions)

LIMITATIONS.

- Meta-Learning can memorize.
  - → Pickup on obvious indications of what the task is.
  - → ∅ for new data
  - → Need to be a good Bayesian + avoid assuming. model is good.

Needs good average policy

- Training process: very data-sufficient. (lots of data)