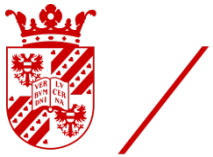
Deep Reinforcement Learning

Lecture 5: Transfer and Meta Reinforcement Learning

Dr. Matthia Sabatelli
m.sabatelli@rug.nl



university of groningen

Today's Agenda

- General Motivation
- Transfer Learning
- Meta-Learning

General Motivation

While successful in many domains it is well known that deep neural networks can be:

- Data Hungry
- Computationally Expensive
- Long Training Times

One might think that these issues are only present when it comes to Supervised Learning, however also in Reinforcement Learnig:

- Agents require millions of trajectories $\langle s_t, a_t, r_t, s_{t+1} \rangle$ that can only be acquired via a very long agent-environment interaction
- Deep Model-Based architectures have at least as many parameters as popular neural architectures that are used in e.g. Computer Vision

General Motivation

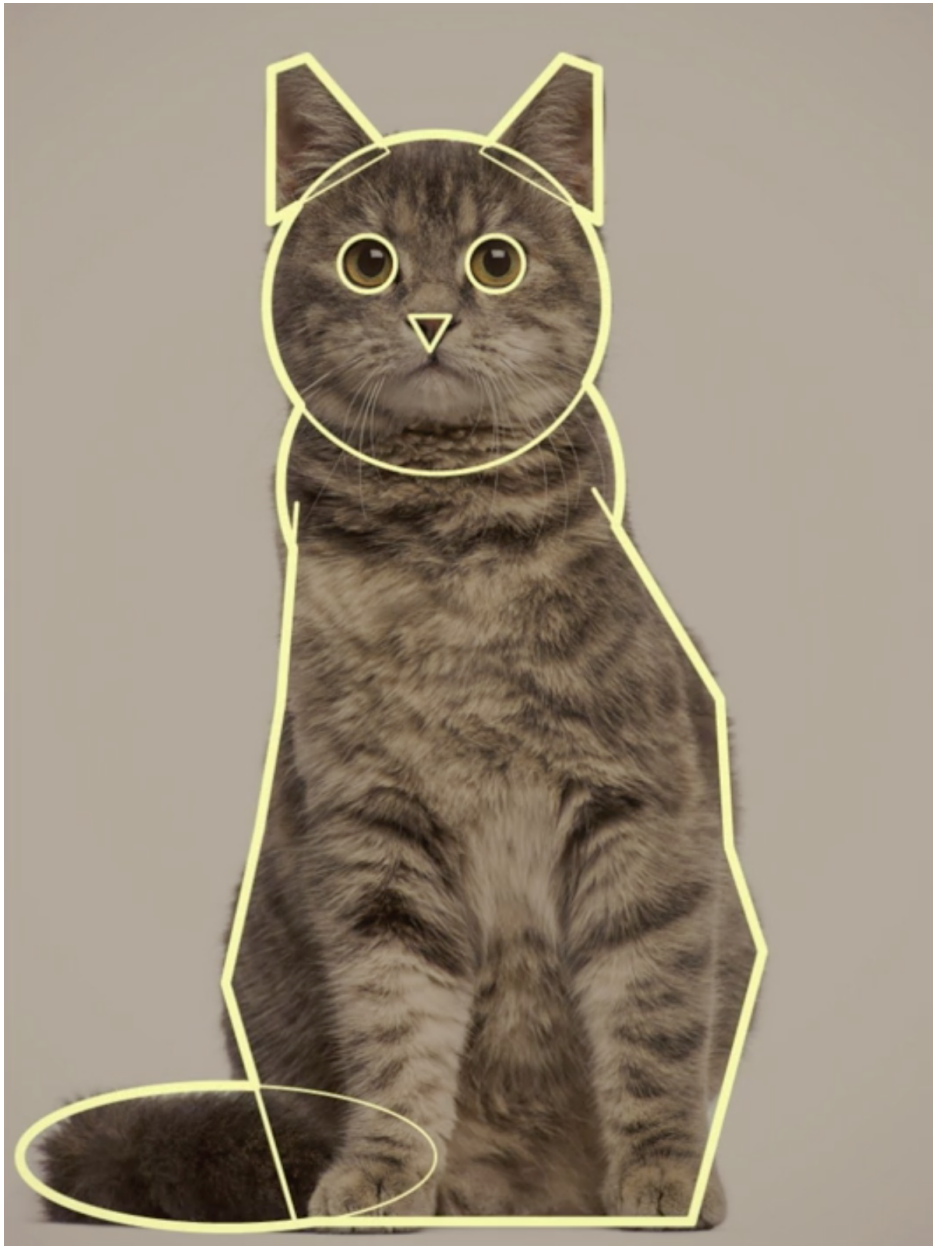
A popular way to deal with data scarcity and long and expensive training runs is to induct some **prior knowledge** within our machine learning system.

- Ideally we don't want to start learning tabula-rasa
- If possible we want to exploit some prior knowledge which we have already learned

Research over the last decade has shown that the fewer human priors get encoded in a machine learning system, the better performing and the more general the model becomes.

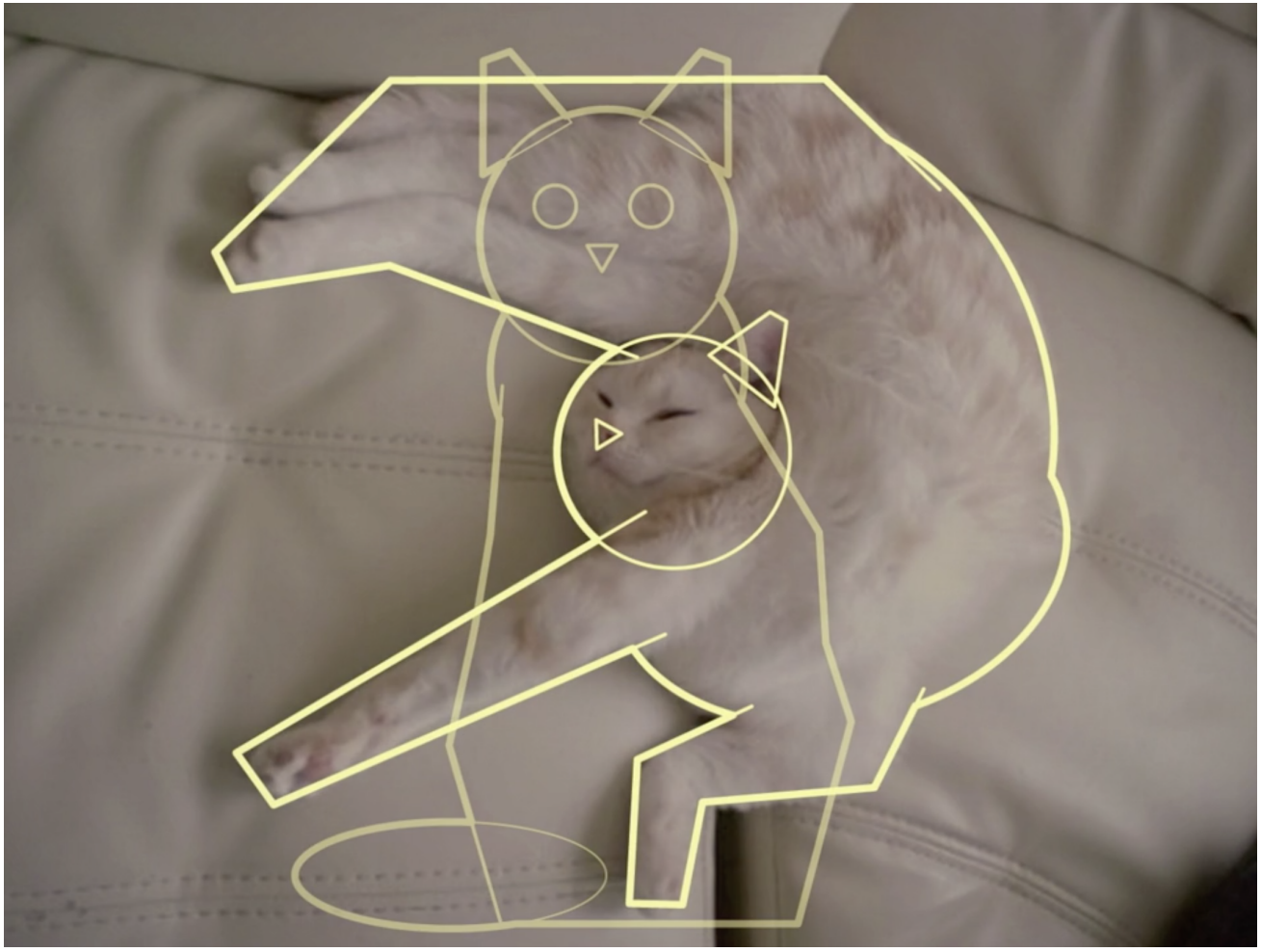
General Motivation

General Motivation



General Motivation

General Motivation



General Motivation

A classic example stemming from Computer Vision is to learn as many possible priors thanks to extremely large datasets, like the popular **ImageNet** dataset.

General Motivation

The same ideas also hold in the field of Natural Language Processing, where nowadays popular **Large Language Models (LLMs)** come as pre-trained on large text corpora and can then easily fine-tuned in a later stage.

General Motivation

Before we start, there are a few questions which I would like to discuss with you:

- "Can you give me three reasons why you think adopting Transfer Learning in a Deep Reinforcement Learning context should be **possible**"

General Motivation

Before we start, there are a few questions which I would like to discuss with you:

- "Can you give me three reasons why you think adopting Transfer Learning in a Deep Reinforcement Learning context should **not be possible**"

The Transfer Learning Problem

Given a source MDP $\mathcal{M}_S$ and a target MDP $\mathcal{M}_T$, transfer learning in reinforcement learning aims to learn an optimal policy π^* for $\mathcal{M}_T$ by exploiting some prior knowledge related to $\mathcal{M}_S$, denoted as $\mathcal{K}_S$, together with the knowledge that underlies $\mathcal{M}_T$, denoted as $\mathcal{K}_T$, such that:

$$\pi^* = \mathbf{argmax}_\pi \mathbb{E}_{s \sim \mu_0^t, a \sim \pi} \left[Q^\pi_{\mathcal{M}}(s, a) \right],$$

where $\pi^* = \zeta(\mathcal{K}_S, \mathcal{K}_T) : \mathcal{S}^t \rightarrow \mathcal{A}^t$ is a function mapping from the states to actions for $\mathcal{M}_T$ learned thanks to both $\mathcal{K}_S$ and $\mathcal{K}_T$.

The Transfer Learning Problem

Particularly important here is the concept of **knowledge** $\mathcal{K}$ which is what we would like to retain when moving from a source MDP $\mathcal{M}_S$ to a target MDP $\mathcal{M}_T$.

- Identifying $\mathcal{K}$ is already a challenging problem given the complex nature of Reinforcement Learning.
- Correctly identifying which kind of knowledge to transfer between $\mathcal{M}_S$ and $\mathcal{M}_T$ is just as important as developing a method that successfully transfers this knowledge in the first place

Typically $\mathcal{K}$ comes in two different forms:

- ?
- ?

The Transfer Learning Problem

Particularly important here is the concept of **knowledge** $\mathcal{K}$ which is what we would like to retain when moving from a source MDP $\mathcal{M}_S$ to a target MDP $\mathcal{M}_T$.

- Identifying $\mathcal{K}$ is already a challenging problem given the complex nature of Reinforcement Learning.
- Correctly identifying which kind of knowledge to transfer between $\mathcal{M}_S$ and $\mathcal{M}_T$ is just as important as developing a method that successfully transfers this knowledge in the first place

Typically $\mathcal{K}$ comes in two different forms:

- Instances
- Parameters

The Transfer Learning Problem

When transferring Instances we transfer RL trajectories coming in the form $\langle s_t, a_t, r_t, s_{t+1} \rangle$ and that have been collected on one, or possibly multiple, source MDPs $\mathcal{M}_S$.

Such trajectories can then be used both in a model-based RL setting, or for speeding up the process of learning a value function.

Ideally, transferring RL trajectories should result in **highly sample efficient** algorithms, although this property, albeit desirable, has one major limitation ...

The Transfer Learning Problem

When we are transferring RL trajectories we have to face the constrain that the source task $\mathcal{M}_S$ and the target task $\mathcal{M}_T$ have:

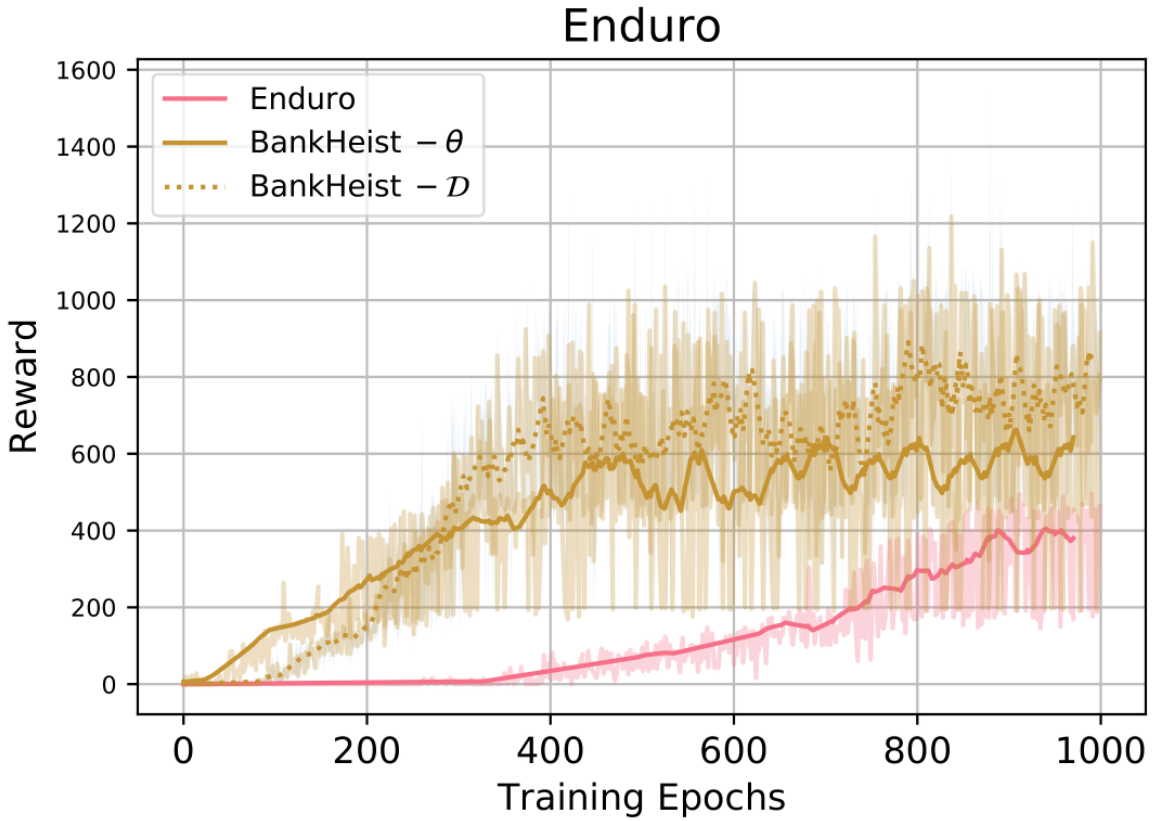
- similar transition models
- the same reward functions.

This instance of TL is usually used within the **batch RL** setup, where gathering experience samples for $\mathcal{M}_T$ can be particularly expensive or time-consuming, which is a constraint that usually does not hold for $\mathcal{M}_S$.

The typical challenge then consists of correctly identifying which of the samples coming from $\mathcal{M}_S$ are the most informative ones for solving $\mathcal{M}_T$.

The Transfer Learning Problem

One way to transfer RL trajectories in a Deep Reinforcement Learning context is by transferring the **Experience Replay Memory Buffers**.



The Transfer Learning Problem

While transferring Experience Replay Memory Buffers is one suitable approach, a way more popular one, inspired by Computer Vision research consists in transferring parameters θ.

As we have seen throughout the course it is desirable to integrate RL algorithms with parametric function approximators to learn an approximation of an optimal value function, policy or model of the Reinforcement Learning environment.

The **idea** is to simply transfer pre-trained networks across MDPs

$$\mathcal{M}_S \rightarrow \mathcal{M}_T$$

On The Transferability of Deep-Q Networks

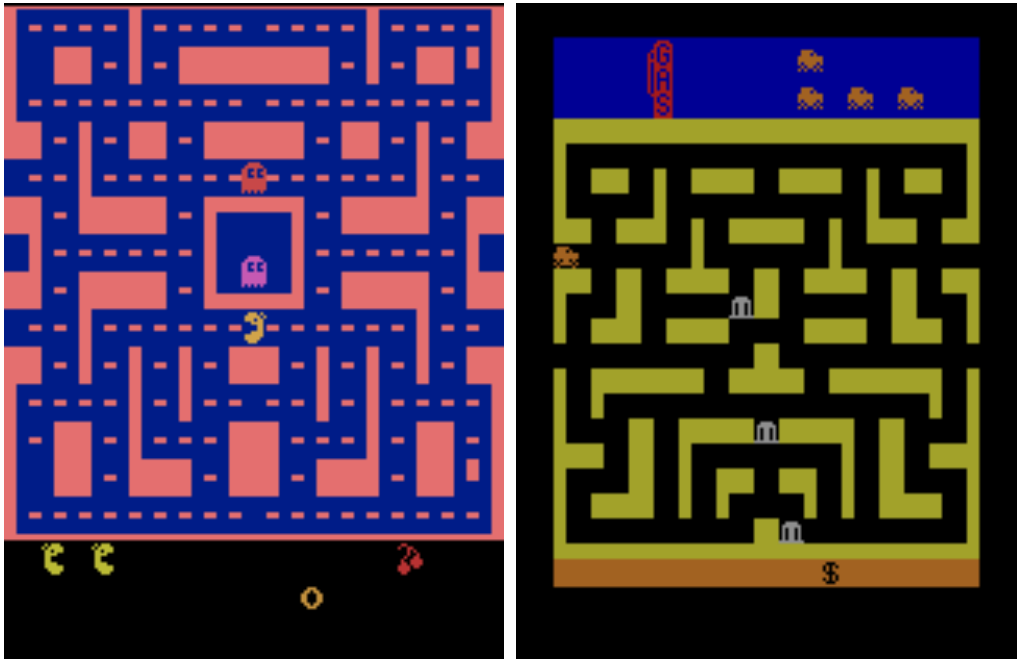
We consider the first case where we have a neural network which has learned to approximate the optimal state-action value function for a certain $\mathcal{M}_S$ and wish to transfer this network to $\mathcal{M}_T$.

The Transfer Learning strategies that one might adopt are the usual learning strategies that characterize transfer learning in Computer Vision:

- Off-the-Shelf Extraction
- Fine-Tuning

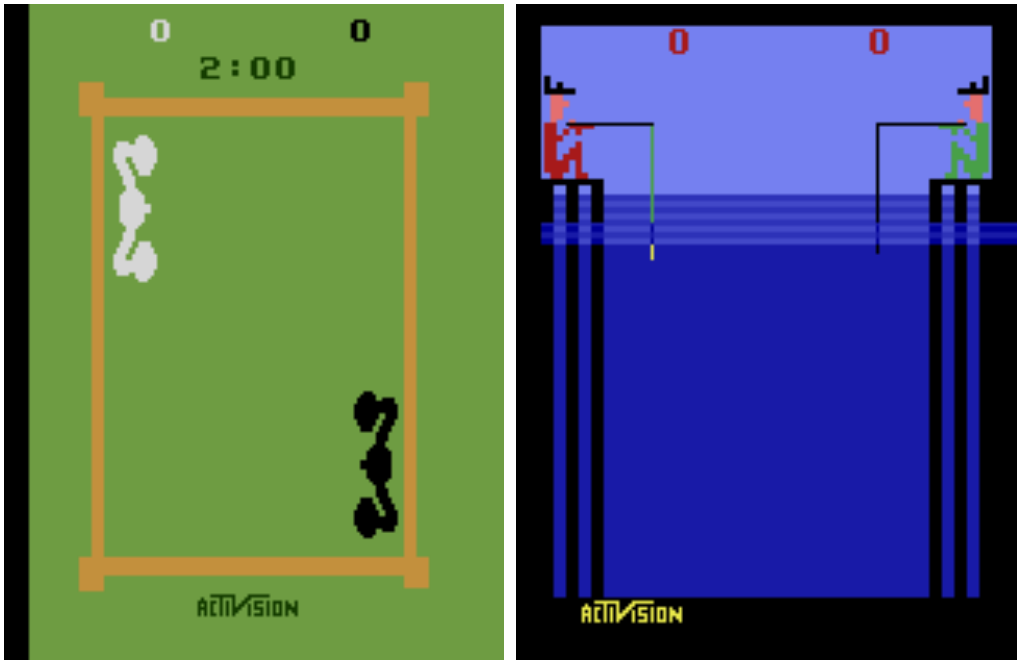
On The Transferability of Deep-Q Networks

- Do you expect positive or negative transfer?



On The Transferability of Deep-Q Networks

- Do you expect positive or negative transfer?



On The Transferability of Deep-Q Networks

DQV	BankHeist	Boxing	CrazyClimber	Enduro	FishingDerby	Frostbite	JamesBond	MsPacman	Pong	Zaxxon
BankHeist	-	-0.019	-1	-0.317	0.5	0.729	0.973	-0.089	-1.238	-0.998
Boxing	-0.494	-	-0.278	-0.852	0.552	-0.01	0.247	-0.184	-0.841	-0.999
CrazyClimber	-0.569	-0.261	-	-0.593	0.19	0.277	0.621	-0.111	-1.206	-0.178
Enduro	-0.571	-0.018	-0.25	-	0.726	-0.017	-0.41	-0.08	-0.466	-0.164
FishingDerby	-1	-0.893	-0.093	-0.45	-	0.068	0.197	-0.136	-3.083	-0.999
Frostbite	-0.933	0.024	-1	-0.348	0.222	-	0.569	0.009	-0.663	-0.076
JamesBond	-0.123	-0.106	-0.131	-0.033	0.519	0.262	-	0.218	-1.329	-1
MsPacman	-0.985	-0.219	-0.012	-0.494	0.6	0.346	0.398	-	-1.646	-0.997
Pong	-1	-0.083	-0.428	-0.476	0.725	-0.024	0.896	0.123	-	-0.729
Zaxxon	-0.76	-0.028	0.037	-0.116	0.385	0.16	-0.253	0.06	-1.602	-

DDQN	BankHeist	Boxing	CrazyClimber	Enduro	FishingDerby	Gopher	IceHockey	Jamesbond	MsPacman	Pong
BankHeist	-	0.121	-0.378	-0.006	-0.107	0.042	-0.006	-0.058	0.001	-3.013
Boxing	-0.316	-	-0.104	-0	0.038	0.06	0.015	-0.225	-0.027	0.936
CrazyClimber	-0.192	-0.487	-	-0.012	-0.084	0.016	0.015	0.016	-0.015	-2.64
Enduro	-0.296	0.193	-0.167	-	0.039	0.03	0.019	-0.235	-0.039	0.248
FishingDerby	-0.212	-0.545	-1	-0.085	-	0.016	0.001	-0.055	-0.026	-0.935
Gopher	-0.466	0.044	-0.108	-0.005	0.007	-	-0.005	-0.094	-0.02	-1.816
IceHockey	-0.046	0.245	-0.067	0.014	-0.178	0.072	-	0.037	-0.015	0.112
Jamesbond	-0.145	0.232	-0.064	0.005	-0.267	0.031	-0.092	-	-0.006	-1.578
MsPacman	-0.173	-1.179	-0.129	-0.06	0.003	-0.019	0.007	0.071	-	-2.774
Pong	-0.127	0.028	-0.12	0.01	0.037	0.042	0.002	-0.174	-0.006	-

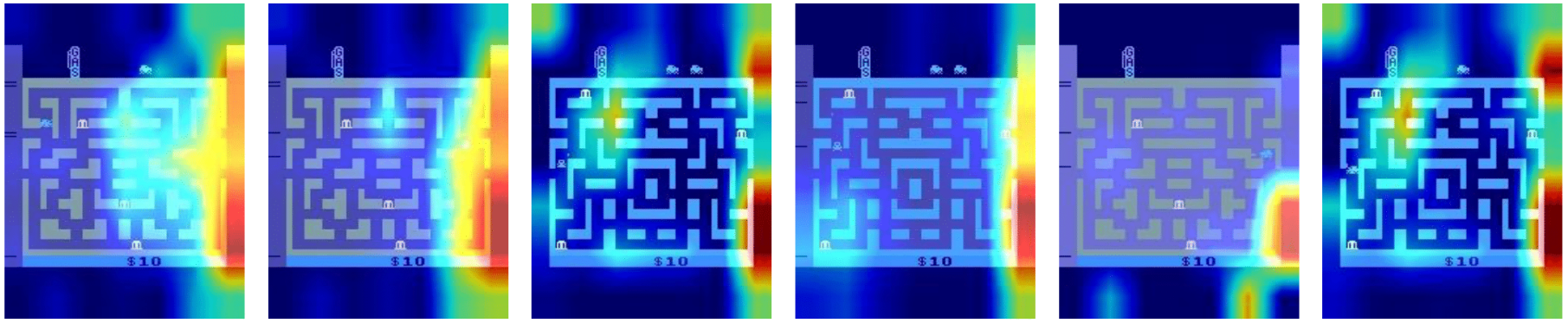
On The Transferability of Deep-Q Networks

When transferring neural networks in a value-based Deep Reinforcement Learning setting:

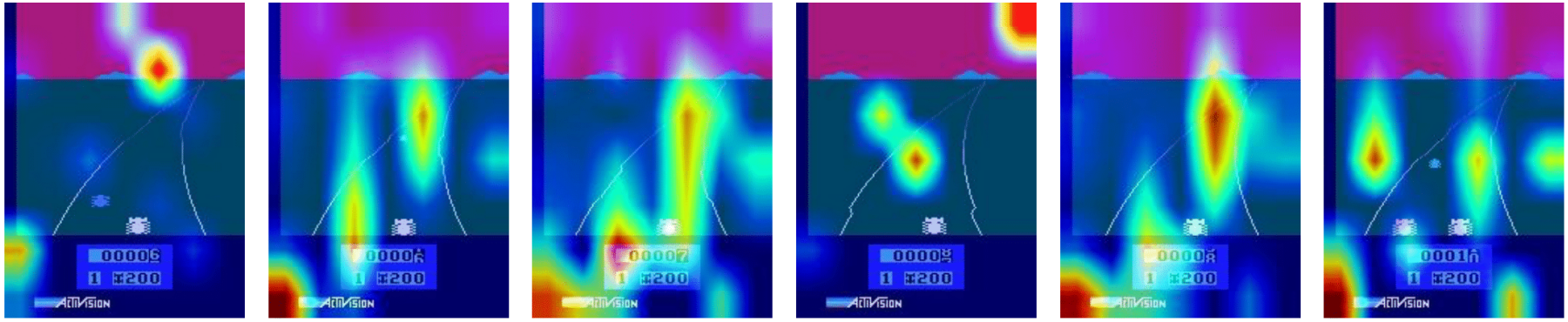
- Positive Transfer happens very rarely, even when the source task $\mathcal{M}_S$ and the target task $\mathcal{M}_T$ share visual features.
- Transfer is not bi-directional: it is possible to obtain positive transfer $\mathcal{M}_S \rightarrow \mathcal{M}_T$ but not $\mathcal{M}_S \leftarrow \mathcal{M}_T$
- Different value-based networks result in different transfer performance, e.g. DQV has better transfer potential than DDQN.

On The Transferability of Deep-Q Networks

What happens in the case of **negative** transfer?



What should happen to achieve **positive** transfer?



On The Transferability of Deep-Q Networks

Unfortunately obtaining the previous results is a very **complicated** task in the value-based Deep Reinforcement Learning.

There are several reasons for this:

- Differently from the supervised learning case Convolutional Neural Networks do not only serve as feature extractors.
- The parameters θ of a Deep-Q Network also encode information about $\approx Q(s, a; \theta)$.
- It is not clear which layers of a Deep-Q Network are responsible for serving as feature extractors and which ones as value function approximators.
- How to tune exploration?
- Training on one source task might not be enough.

On The Transferability of Policy Gradients

This last limitation is addressed by the **Actor-Mimic** algorithm, which exploits a multi-task learning strategy for learning visual representations that can generalize to new tasks.

- Actor-Mimic starts by defining a set of source tasks $\mathcal{M}_{S1}, \mathcal{M}_{S2}, ..., \mathcal{M}_{SN}$
- Each source task is associated to an "expert", E, a DQN agent that is already trained on its respective source task.

It then rescales the Q-values of each expert through the softmax function:

$$\pi_E(a|s) = \frac{e^{\tau Q_E(s,a)}}{\sum_{a' \in \mathcal{A}_E} e^{\tau Q_E(s,a')}}$$

distilling every value function into a policy π.

On The Transferability of Policy Gradients

Given a state s from task $\mathcal{M}_S$ Actor-Mimic's **objective** is defined over the multi-task network as the cross-entropy between the expert's policy and the current multi-task policy:

$$\mathcal{L}^i_{policy}(\theta) = \sum_{a \in \mathcal{A}_E} \pi E_i(a|s) \ \log \ \pi_{\text{AMN}}(a|s;\theta).$$

On The Transferability of Deep Model-Based Algorithms

One learning scenario in which transferring pre-trained models is much more successful is that of Deep Model-Based Reinforcement Learning.

As seen last week, the core idea of model-based Reinforcement Learning is that of learning an approximation of the transition and reward functions:

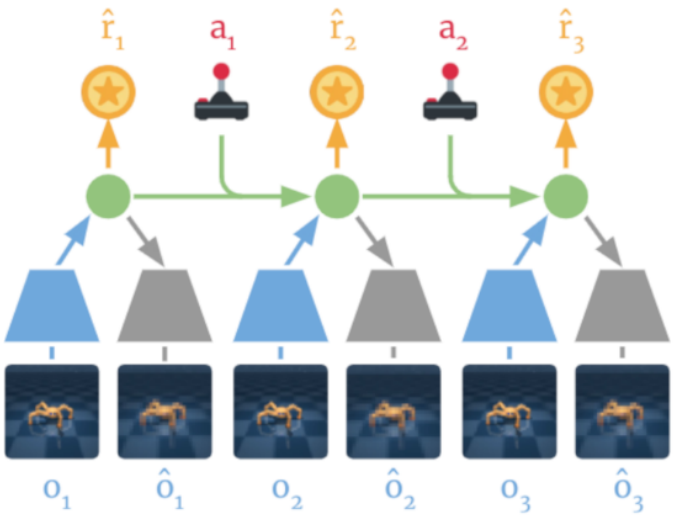
- $\mathcal{P} \approx (\hat{\mathcal{P}}; \theta)$
- $\mathcal{R} \approx (\hat{\mathcal{R}}; \theta)$.

This typically corresponds to solving a **Supervised Learning** problem as given a RL trajectory $\langle s_t, a_t, r_t, s_{t+1} \rangle$ we can train f such that:

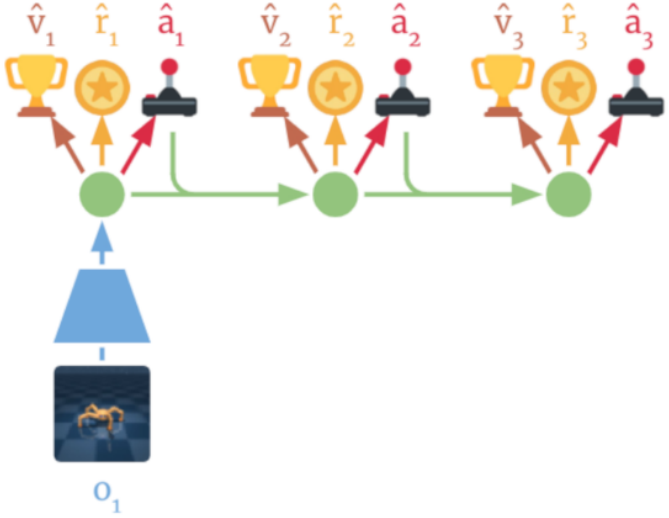
- $f(s_t, a_t; \theta) \rightarrow s_{t+1}$
- $f(s_t, a_t; \theta) \rightarrow r_t$

On The Transferability of Deep Model-Based Algorithms

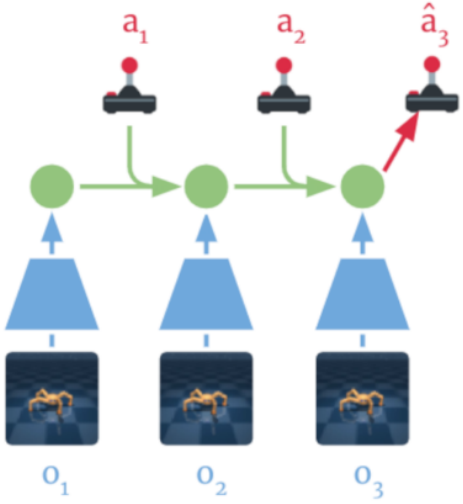
A popular Deep Model-Based Algorithm is **Dreamer**



Learn dynamics from dataset



Learn behavior in imagination



Environment interaction

On The Transferability of Deep Model-Based Algorithms

Dreamer consists of six main components:

- Representation model: $p_{\theta_{\text{REP}}}(s_t|s_{t-1}, a_{t-1}, o_t)$.
- Observation model: $q_{\theta_{\text{OBS}}}(o_t|s_t)$.
- Reward model: $q_{\theta_{\text{REW}}}(r_t|s_t)$.
- Transition model: $q_{\theta_{\text{TRANS}}}(s_t|s_{t-1}, a_{t-1})$.
- Actor: $q_\phi(a_\tau|s_\tau)$.
- Critic: $v_\psi(s_t) \approx \mathbb{E}_{q(\cdot|s_\tau)}(\sum_{\tau=t}^{t+H} \gamma^{\tau-t} r_\tau)$.

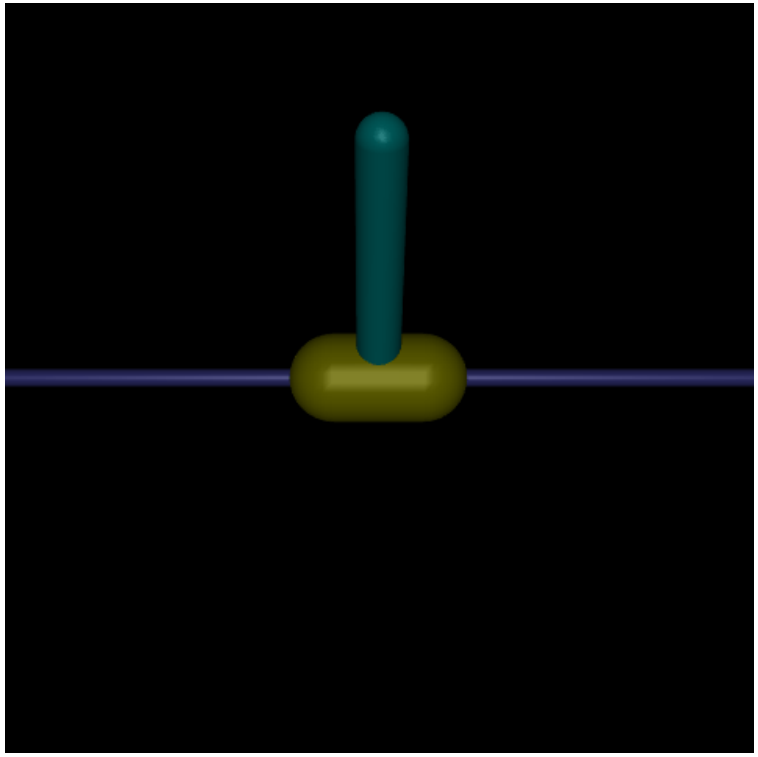
On The Transferability of Deep Model-Based Algorithms

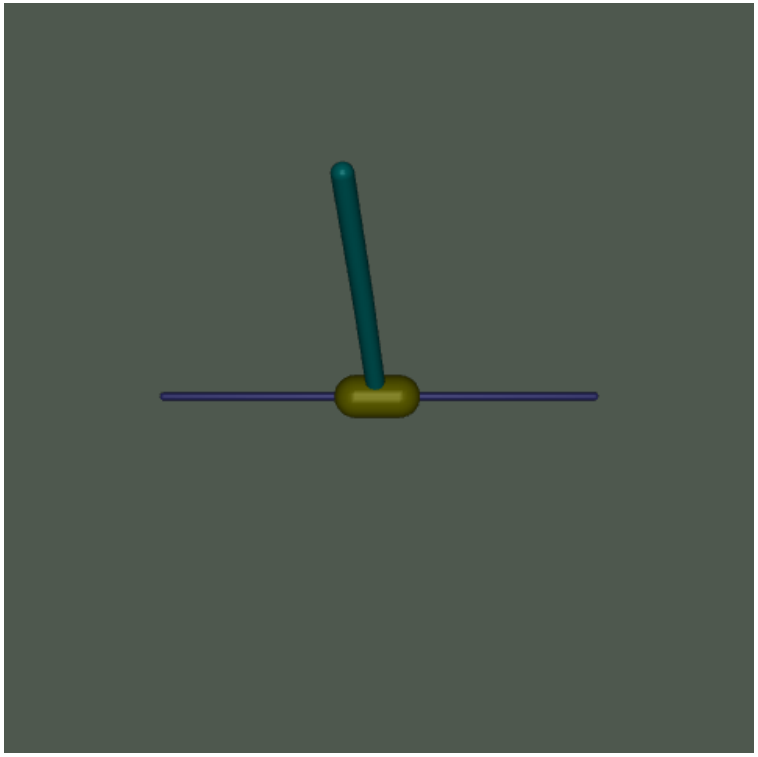
Dreamer's highly **modular** nature does not present the same issues that characterize Deep-Q Networks, meaning that it is very clear which part of the architecture is responsible for learning a certain component of the MDP.

- The parameters of the representation and observation models are the ones that are responsible for extracting features from high dimensional state spaces $\mathcal{S}$.
- The transiton model encodes the physical properties of the environments, and is therefore responsible for learning $\hat{\mathcal{P}}$
- The actor encodes information about the action space $\mathcal{A}$ of the MDP
- The reward model learns to approximate $\hat{\mathcal{R}}$.

On The Transferability of Deep Model-Based Algorithms

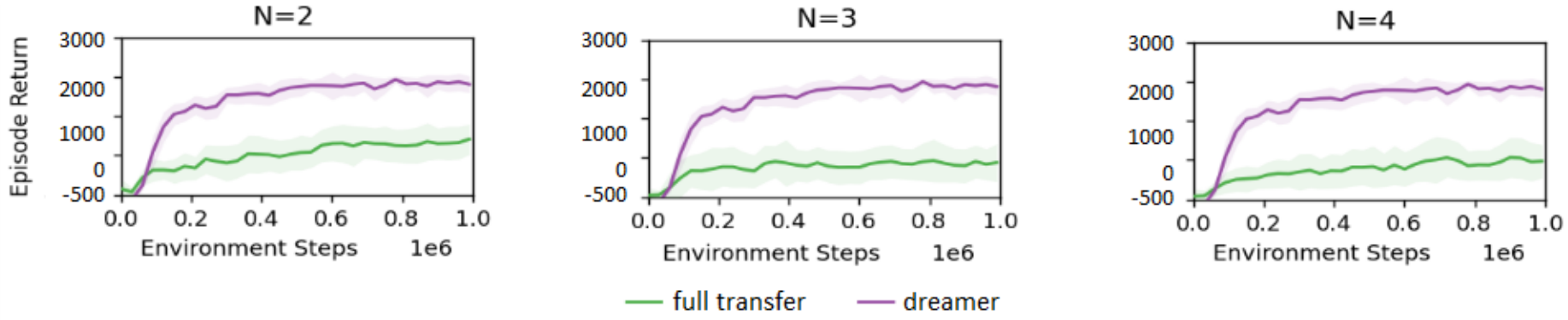
That being said, assuming we have some prior knowledge about the source $\mathcal{M}_S$ and target $\mathcal{M}_T$ MDPs, it is easy to decide which parameters to transfer across tasks.





On The Transferability of Deep Model-Based Algorithms

However, fully transferring pre-trained parameters as is typically done in Computer Vision is **not a good idea** even in the deep model-based context.



- The reason is similar to the one shown earlier in the context of Deep-Q Networks.
- Deep Reinforcement Learning algorithms strongly overfit on the source task $\mathcal{M}_S$ and tend to not forget what they learn on $\mathcal{M}_S$.

On The Transferability of Deep Model-Based Algorithms

One way to avoid this from happening, at least in the deep model-based context, is by adopting **Fractional Transfer Learning** (FTL).

The motivation underlying FTL is very **simple**: it is extremely rare, if not impossible, that two tasks are completely identical to each other.

Therefore fully transferring all parameters across models is not beneficial.

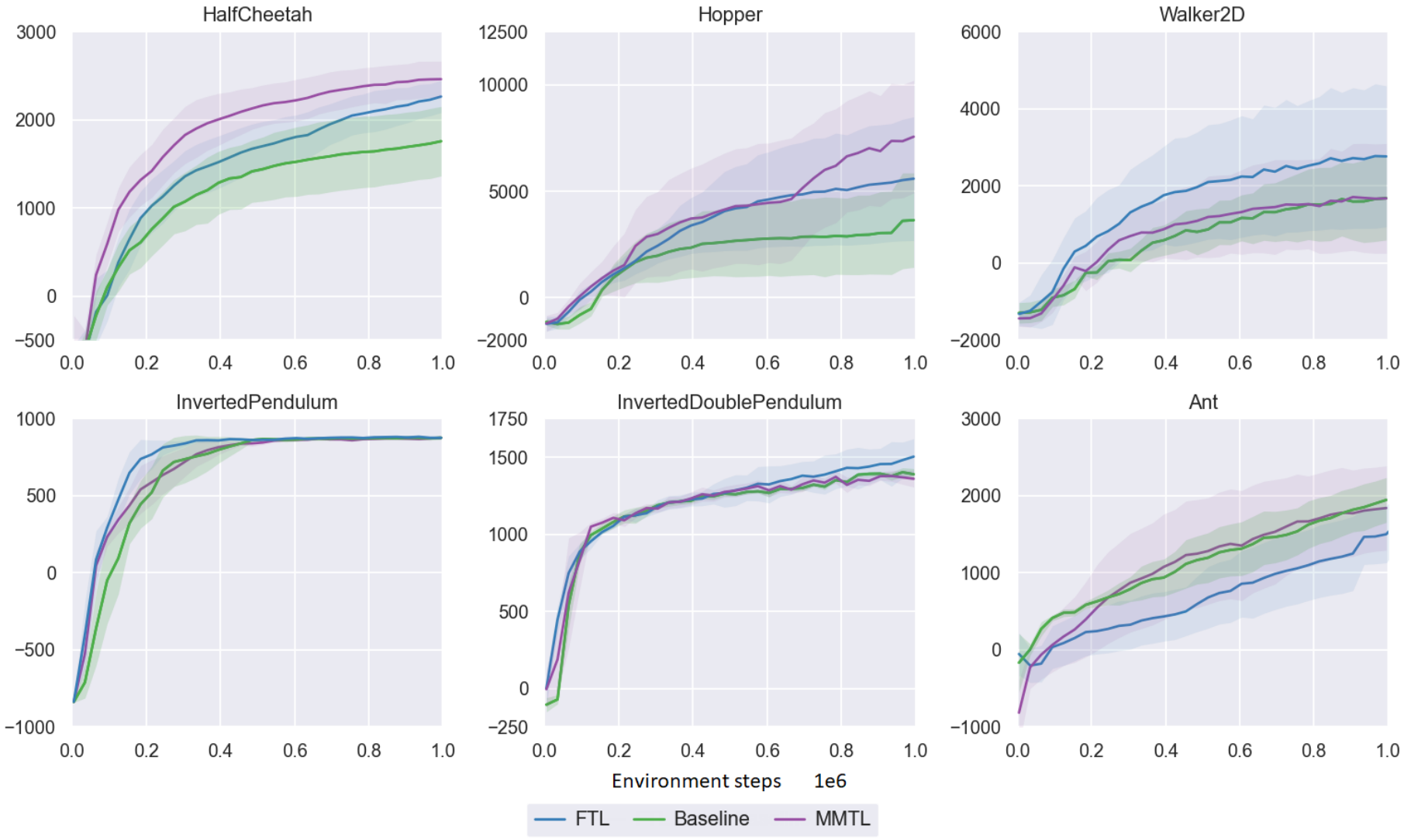
On The Transferability of Deep Model-Based Algorithms

FTL only transfers a **fraction of the pre-trained parameters** such that the network can both benefit from the transferred knowledge and also continue to learn and adapt to the new task.

Let θ_T denote target parameters, θ_ϵ randomly initialized weights, λ the fraction parameter, and θ_S source parameters, FTL is defined as:

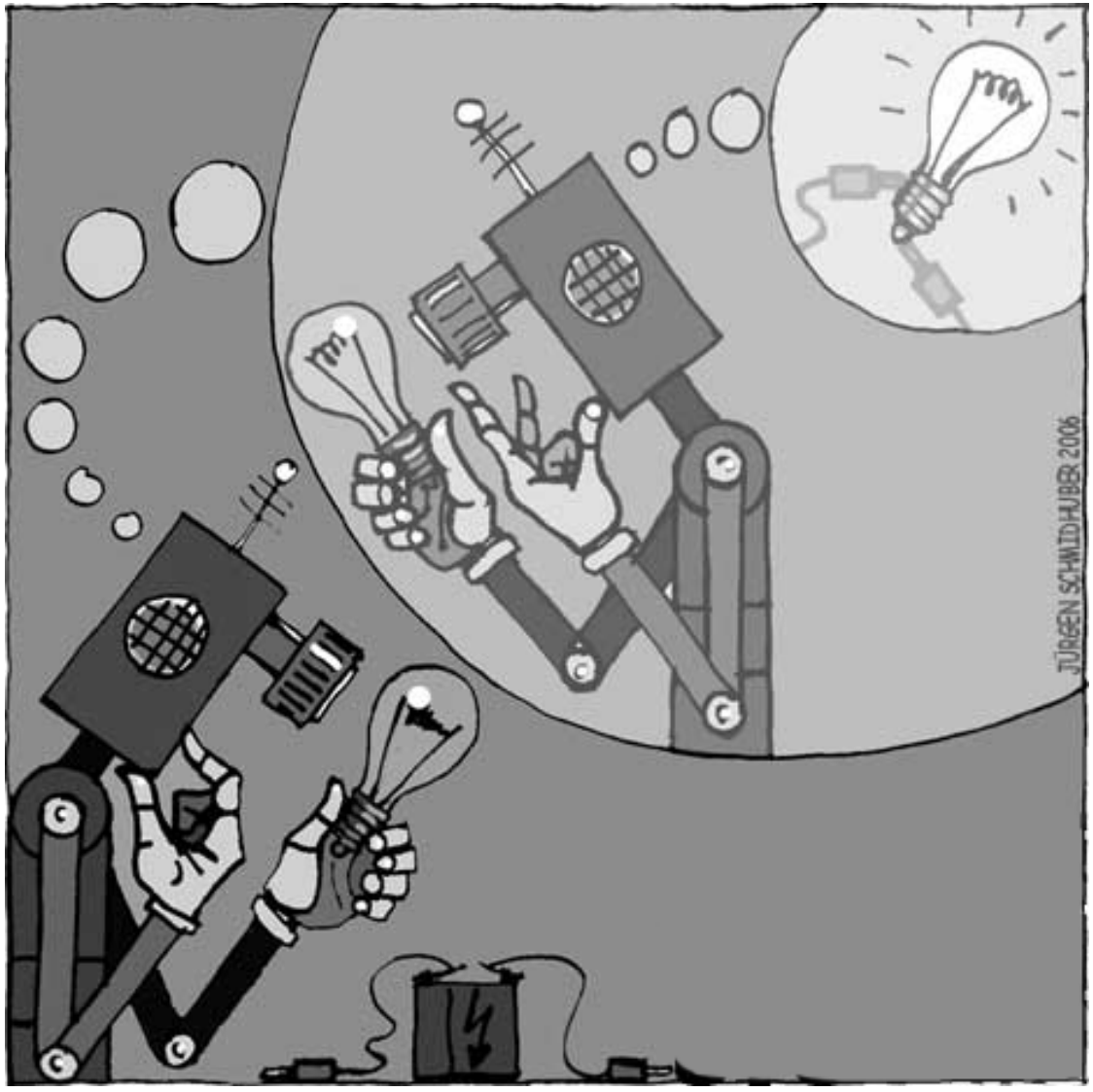
$$\theta_T \leftarrow \theta_\epsilon + \lambda\theta_S.$$

On The Transferability of Deep Model-Based Algorithms



Break

The Meta-Learning Problem



The Meta-Learning Problem

The underlying **idea** of Meta-Learning is similar to that of Transfer Learning, as also in this case we want to leverage previous experience and be able to learn from small amount of data.

In the Reinforcement Learning context we want to learn a value function $Q(s, a)$, policy π or model of the environment $\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathfrak{R}$ with as few trajectories τ as possible.

Differently from the Transfer-Learning scenario, where we explicitly decided which type of Knowledge $\mathcal{K}$ to transfer, we now want to **learn** what the most effective transferable priors are.

The Meta-Learning Problem

We take a step back and again consider the **Supervised Learning** setting.

Our goal is that of learning

$$y = f(\mathbf{x}; \theta)$$

from input-output pairs coming in the following form:

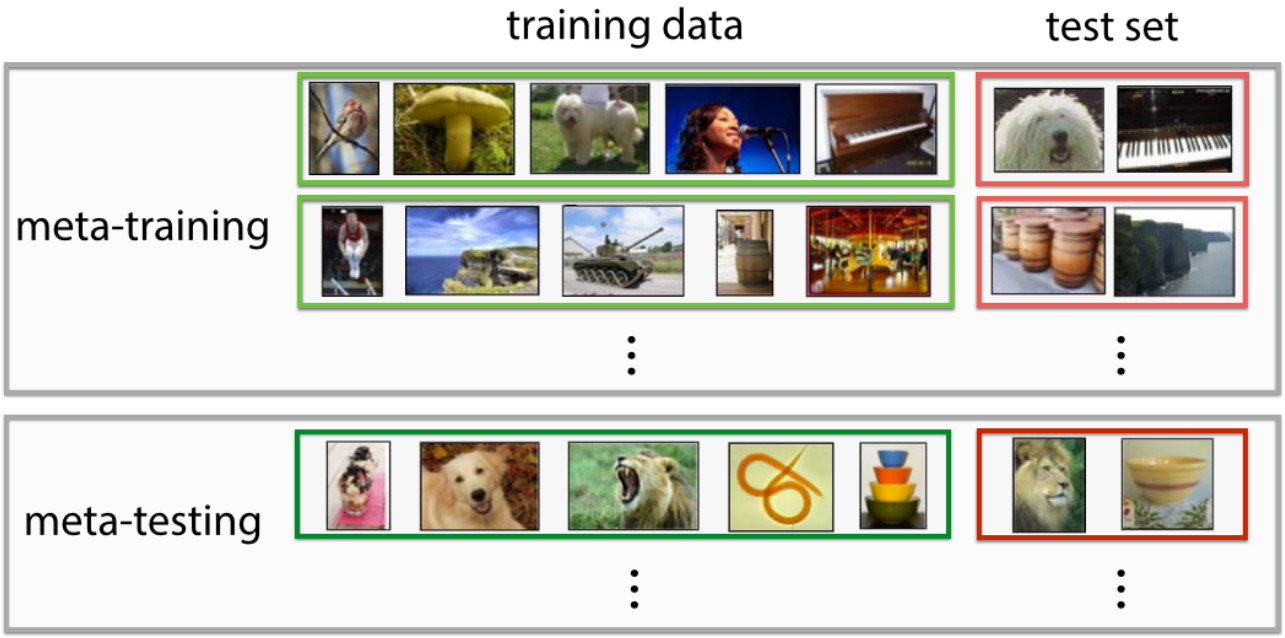
$\mathcal{D}_{\text{train}} = (\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$ where $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$.

In Meta Supervised Learning we start by having

- $\mathcal{D}_{\text{train}} = (\mathbf{x}, \mathbf{y})_{\mathbf{1:K}}$ with respective $\mathbf{x}_{\text{test}}$ and $\mathbf{y}_{test}$.
- But we also want to incorporate additional data.

The Meta-Learning Problem

Let's see how this problem looks like in practice:



The **key idea** is to construct a set of tasks $\mathcal{T}$ that resemble as much as possible the task we would like to solve.

Each task comes with its own dataset therefore resulting in a dataset of datasets $\mathcal{D}_{\text{meta-train}} = \mathcal{D}_1, ..., \mathcal{D}_n$.

The Meta-Learning Problem

While for regular supervised learning our objective is simply

$$\mathbf{argmax}_{\phi} \log p(\phi|\mathcal{D}_{\text{train}})$$

,

Now our goal becomes to learn

$$\mathbf{argmax}_{\phi} \log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}})$$

In practice this means learning how to solve the tasks contained in $\mathcal{D}_{\text{meta-train}}$ first (meta-training), and once we have done so provide our learning algorithm with $\mathcal{D}_{\text{train}}$.

The Meta-Learning Problem

Formally we want to learn meta-parameters

$$\theta^\star = \mathbf{argmax}_\theta \log p(\theta|\mathcal{D}_{\text{meta-train}}),$$

which, intuitively, means finding a set of parameters $\theta^\star$ that contain everything we need to know to effectively solve new tasks.

This can be written as follows:

$$\begin{aligned}\log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}}) &= \log \int_\Theta p(\phi|\mathcal{D}_{\text{train}}, \theta) p(\theta|\mathcal{D}_{\text{meta-train}}) d\theta \\ &\approx \log p(\phi|\mathcal{D}_{\text{train}}; \theta^\star) + \log p(\theta^\star|\mathcal{D}_{\text{meta-train}}).\end{aligned}$$

The Meta-Learning Problem

Once we have $\theta^\star$, we still need to find ϕ, which is the set of parameters that we need for solving the problem modeled by $\mathcal{D}_{\text{train}}$.

This can be written as:

$$\text{argmax}_\phi \log p(\phi|\mathcal{D}, \mathcal{D}_{meta-train}) \approx \text{argmax}_\phi \log p(\phi|\mathcal{D}, \theta^\star)$$

Combining everything together we can see the meta-learning optimization process as a two-stage process:

- meta-learning: $\theta^\star = \text{argmax}_\theta \log p(\theta|\mathcal{D}_{\text{meta-train}})$
- adaptation: $\phi^\star = \text{argmax}_\phi \log p(\phi|\mathcal{D}_{\text{train}}, \theta^\star)$

When it comes to the Reinforcement Learning setting we translate our problem formulation and replace:

- Input Data $\mathbf{x} \Rightarrow$ states $\boldsymbol{s}_t$
- Output Data $\mathbf{y} \Rightarrow$ acions $\boldsymbol{a}_t$
- Learning Function $\mathbf{y} = f(\mathbf{x}; \theta) \Rightarrow \boldsymbol{a}_t = \pi(\boldsymbol{s}_t; \theta)$
- Training Data $(\mathbf{x}, \mathbf{y})_i \Rightarrow$ trajectories $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{r}_t, \boldsymbol{s}_{t+1})$

Therefore our objective becomes designing and optimizing f such that

$$a_t = f(\mathcal{D}_{\text{train}}, \boldsymbol{s}_t; \theta),$$

where $\mathcal{D}_{\text{train}}$ is composed of k rollouts from a certain policy π.

Following the notation that we have used until today throughout the course, we can write the meta-learning step as:

$$\theta^\star = \mathbf{argmax}_\theta \sum_{i=1}^{n} \mathbb{E}_{\pi\phi_i(\tau)}[R(\tau)]$$

where $\phi_i = f(\mathcal{M}_i; \theta)$.

The next natural design choice one needs to face is how to design f and how to train this system such that it allows us to find $\phi^\star$ from $\theta^\star$.

Optimization-Based Meta-Learning

These techniques build on top of the idea of fine-tuning, which, as seen earlier, means updating a set of pre-trained parameters θ via some training data that is representative of a new task:

$$\phi_j \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}^j_{\text{train}}(\theta)$$

While this approach works very well when it comes to Computer Vision and Natural Language Processing tasks, in the Reinforcement Learning context things become **harder**:

- There's no large dataset like ImageNet that one can use for finding θ
- Neural Networks serve both as feature extractors as well as value function approximators (see earlier limitations mentioned with DQNs)
- Remember that we are dealing with a not-stationary problem

Optimization-Based Meta-Learning

The **idea** of optimization-based meta-learning is that of finding a set of parameters θ that guarantees successful fine-tuning.

Formally this means finding

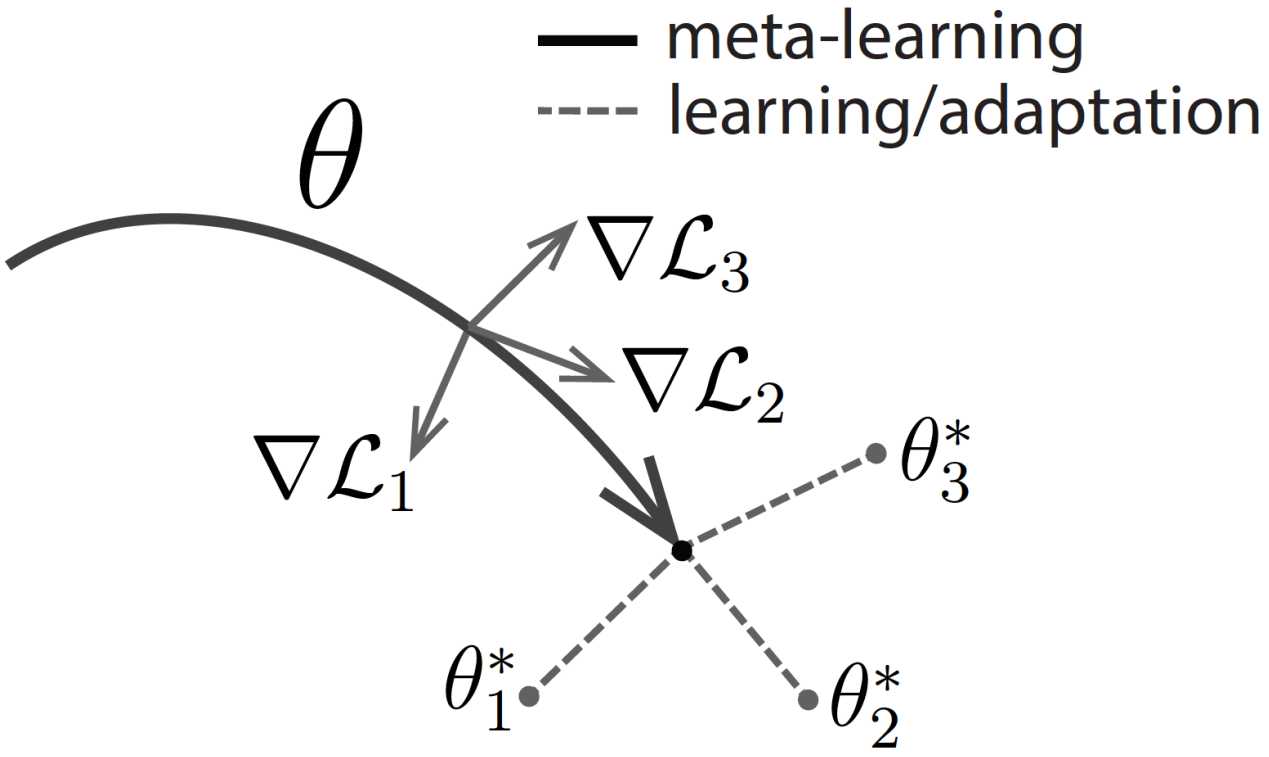
$$\min_{\theta} \sum_{\text{task } i} \mathcal{L}^{i}_{\text{test}}(\theta - \alpha \nabla_{\theta} \mathcal{L}^{i}_{\text{meta-train}}(\theta)).$$

This corresponds to:

- Starting from a set of parameters θ
- Run a few steps of gradient descent on your new task $\mathcal{L}^{i}_{\text{meta-train}}$
- Optimize the performance of these fine-tuned parameters on hold-out data $\mathcal{L}^{i}_{\text{test}}$

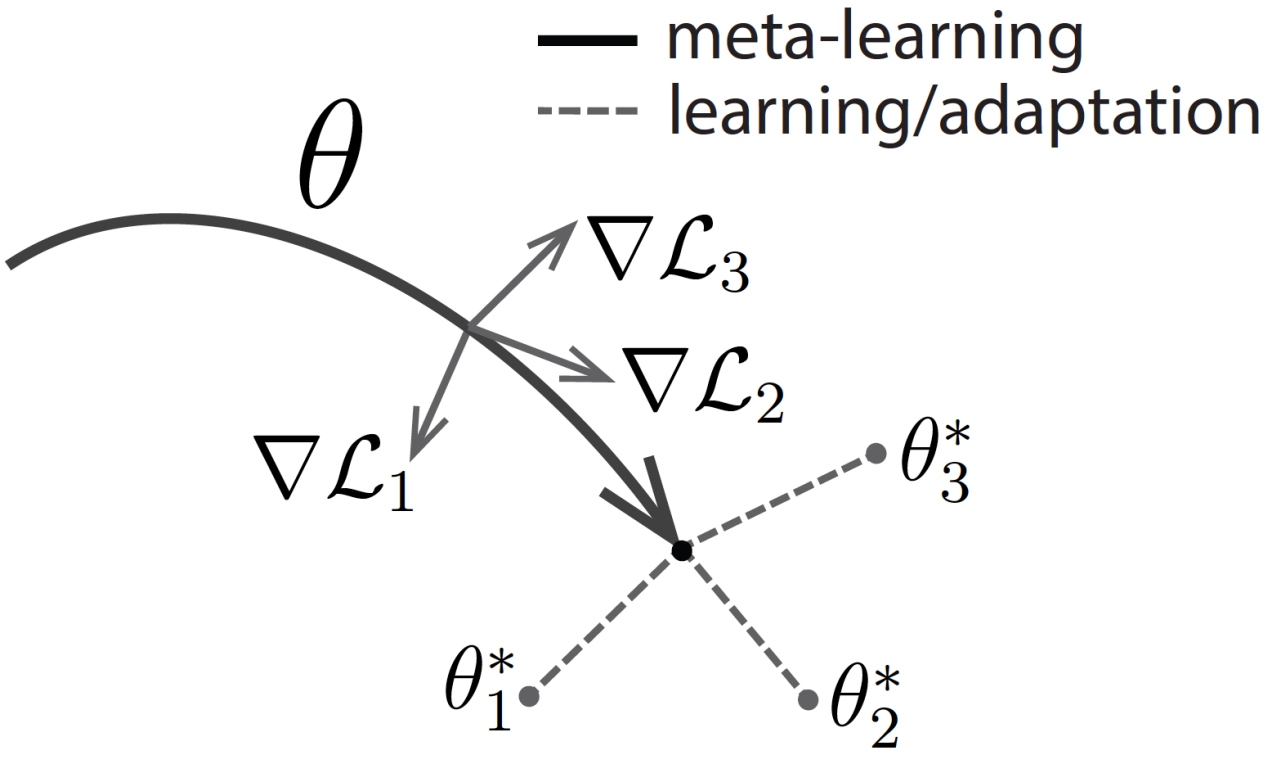
Optimization-Based Meta-Learning

Conceptually we can visualize this process as follows, where ϕ_i is replaced with θ_i^*:



Optimization-Based Meta-Learning

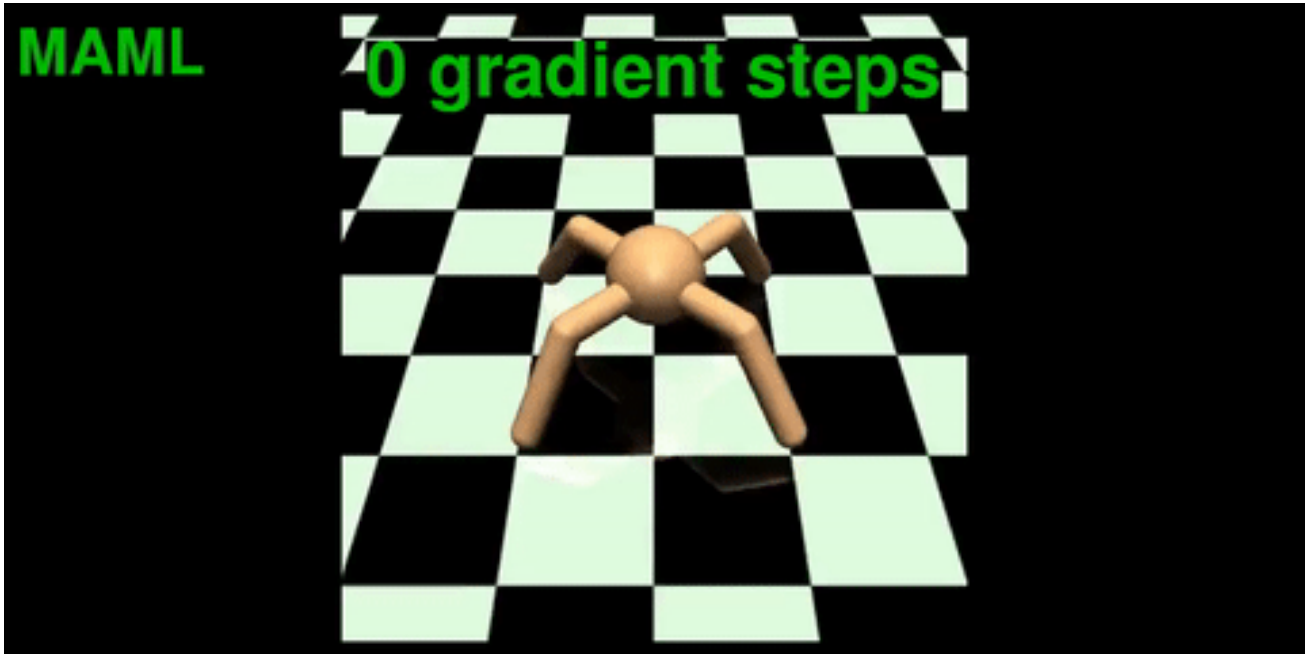
Conceptually we can visualize this process as follows, where ϕ_i is replaced with θ_i^*:



This algorithm comes with the name **M**odel **A**gnostic **M**eta **L**earning (MAML)

Optimization-Based Meta-Learning

How does this algorithm look like in practice?



Optimization-Based Meta-Learning

Optimization Based Meta-Learning methods:

- Incorporate the inductive bias of Stochastic Gradient Descent
- Are model-agnostic, as they are independent from the neural architecture that one wants to train
- Depend on the gradients that are obtained during meta-training, which might be noisy or not very informative (see Lecture 3)

References

- Sabatelli, Matthia. "Contributions to Deep Transfer Learning: from Supervised to Reinforcement Learning." (2022).
- Tirinzoni, Andrea. "Exploiting structure for transfer in reinforcement learning." (2021).
- Sabatelli, Matthia, and Pierre Geurts. "On The Transferability of Deep-Q Networks." Deep Reinforcement Learning Workshop of the 35th Conference on Neural Information Processing Systems. 2021.
- Parisotto, Emilio, Jimmy Lei Ba, and Ruslan Salakhutdinov. "Actor-mimic: Deep multitask and transfer reinforcement learning." arXiv preprint arXiv:1511.06342 (2015).

References

- Sasso, Remo, Matthia Sabatelli, and Marco A. Wiering. "Multi-Source Transfer Learning for Deep Model-Based Reinforcement Learning." Transactions on Machine Learning Research (TMLR).
- Finn, Chelsea, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks." International conference on machine learning. PMLR, 2017.

See you next week