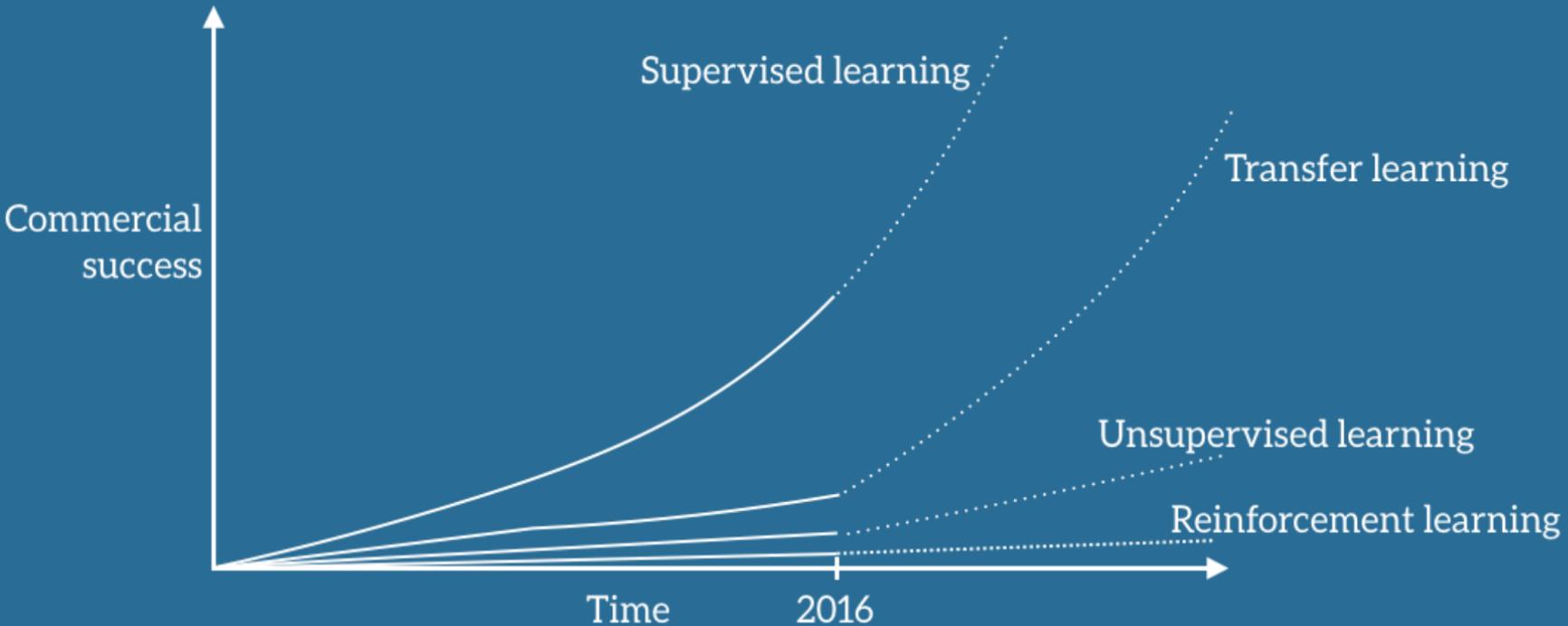


# Transfer- and Meta-Learning

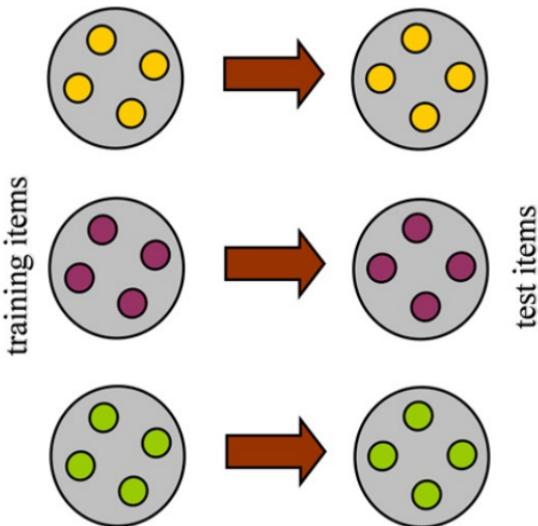
# Drivers of ML success in industry



- Andrew Ng, NIPS 2016 tutorial

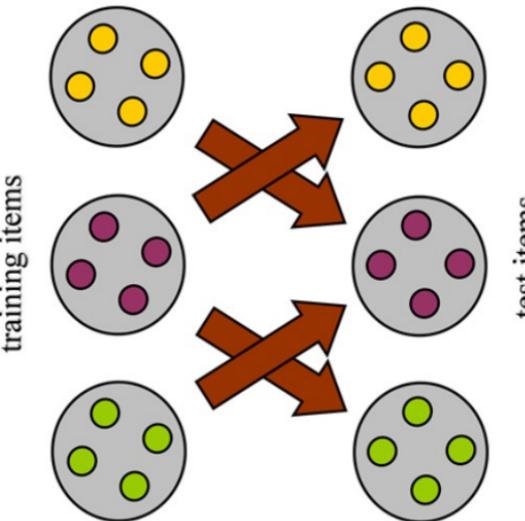
# Transfer Learning

Traditional ML in  
multiple domains



Humans can learn in many domains.

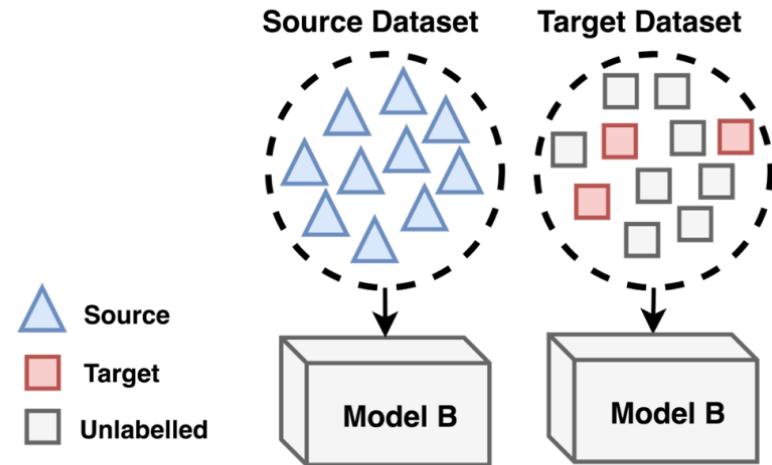
Transfer of learning  
across domains



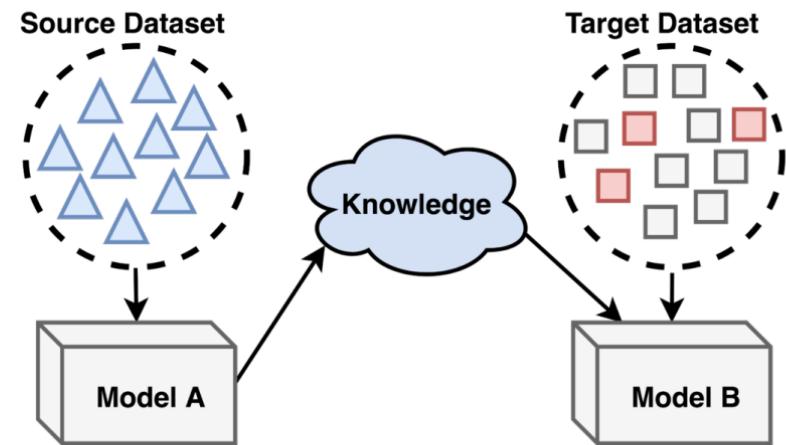
Humans can also transfer from one  
domain to other domains.

# Transfer Learning

## Traditional Machine Learning



## Transfer Learning



# Why transfer learning?

- Labeled data might be limited, *data-poor*
- Calibration – training – can be very *expensive*
- The learning process can be (too) *long*

## Objectives of transfer learning:

- Extract knowledge learned from related domains to help training in a target domain with a few labeled data
- Extract knowledge learned from related domains to speed up learning in a target domain

**For simplicity, we only consider at most two domains and two tasks.**

**Source domain:**

$$\mathcal{P}(X_S), \text{ where } X_S = \{x_{S_1}, x_{S_2}, \dots, x_{S_{n_S}}\} \in \mathcal{X}_S$$

**Task in the source domain:**

$$\mathcal{P}(Y_S|X_S), \text{ where } Y_S = \{y_{S_1}, y_{S_2}, \dots, y_{S_{n_S}}\} \text{ and } y_{S_i} \in \mathcal{Y}_S$$

**Target domain:**

$$\mathcal{P}(X_T), \text{ where } X_T = \{x_{T_1}, x_{T_2}, \dots, x_{T_{n_T}}\} \in \mathcal{X}_T$$

**Task in the target domain**

$$\mathcal{P}(Y_T|X_T), \text{ where } Y_T = \{y_{T_1}, y_{T_2}, \dots, y_{T_{n_T}}\} \text{ and } y_{T_i} \in \mathcal{Y}_T$$

# Approaches to Transfer Learning

Transfer learning approaches	Description
Instance-transfer	To re-weight some labeled data in a source domain for use in the target domain
Feature-representation-transfer	Find a "good" feature representation that reduces difference between a source and a target domain or minimizes error of models
Model-transfer	Discover shared parameters or priors of models between a source domain and a target domain
Relational-knowledge-transfer	Build mapping of relational knowledge between a source domain and a target domain.

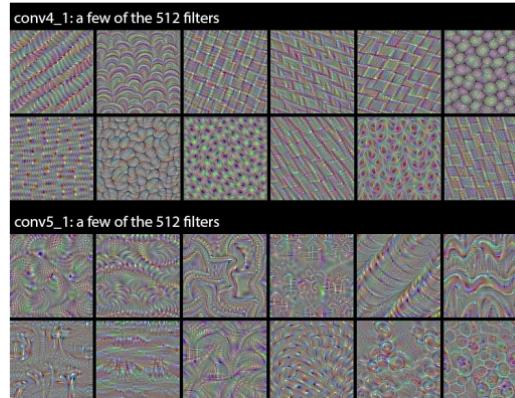
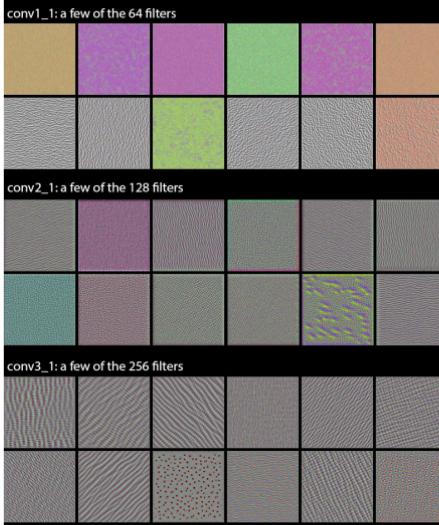
# Transfer learning types (inductive model transfer)

Type	Description	Examples
Inductive	Adapt existing <b>supervised</b> training model on new <b>labeled</b> dataset	Classification, Regression
Transductive	Adapt existing <b>supervised</b> training model on new <b>unlabeled</b> dataset	Classification, Regression
Unsupervised	Adapt existing <b>unsupervised</b> training model on new <b>unlabeled</b> dataset	Clustering, Dimensionality Reduction

# From generic to specific

## Neural Network Layers: General to Specific

- Bottom/first/earlier layers: general learners
  - Low-level notions of edges, visual shapes
- Top/last/later layers: specific learners
  - High-level features such as eyes, nose

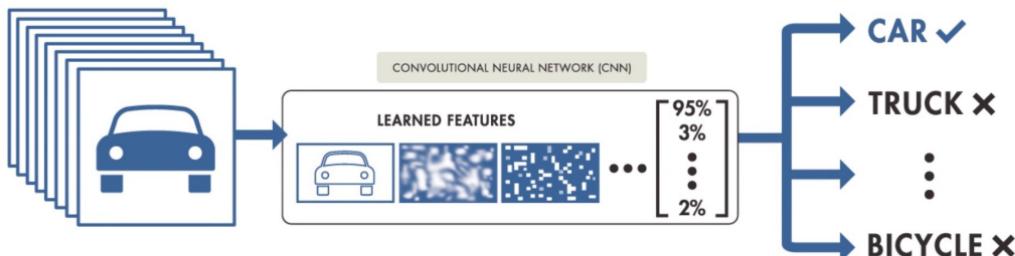


Keep the generic  
adjust the specific

# From generic to specific

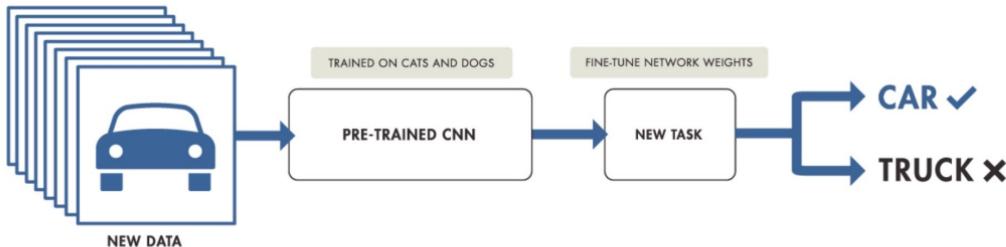
## Neural Network Layers: General to Specific

### TRAINING FROM SCRATCH



Keep the generic  
remove the specific

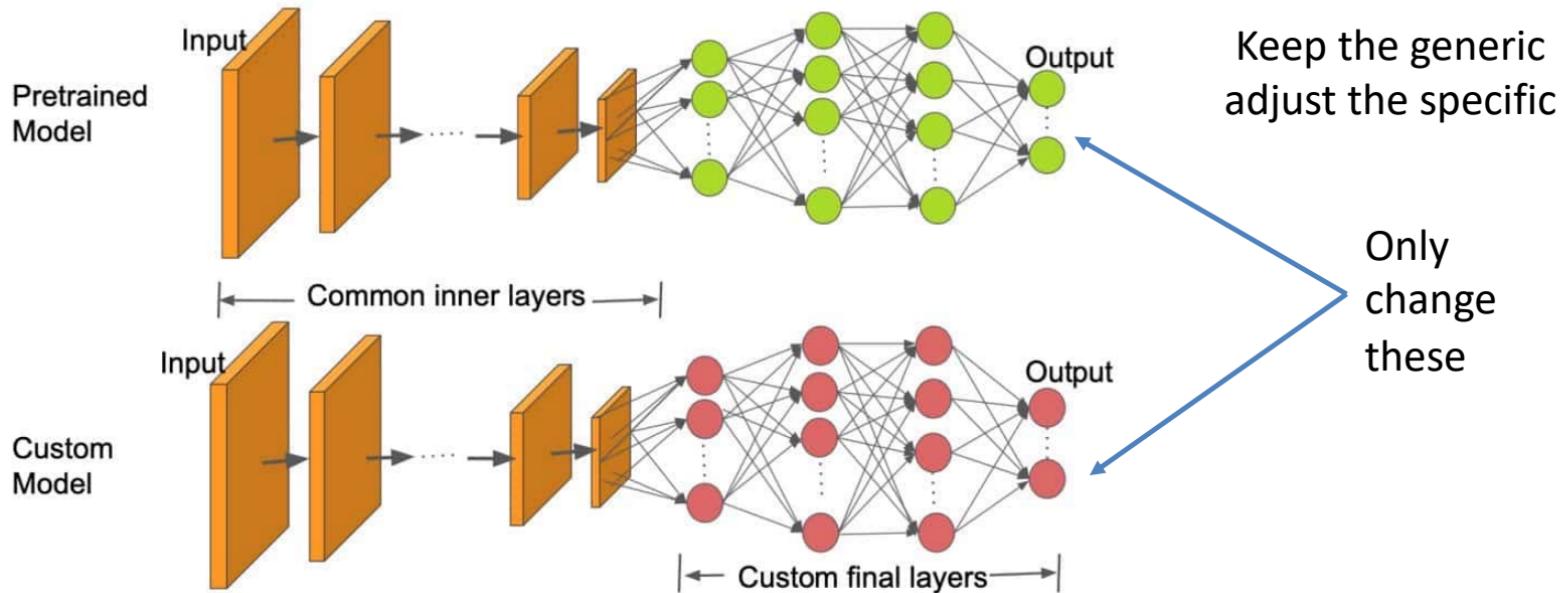
### TRANSFER LEARNING



Only  
change  
these

# From generic to specific

## Neural Network Layers: General to Specific



# Current example

## ChatGPT fine tuning

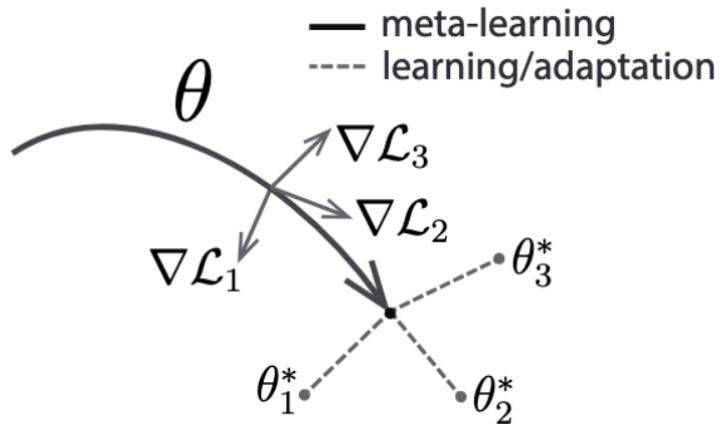
<https://platform.openai.com/docs/guides/fine-tuning/when-to-use-fine-tuning>

Some common use cases where fine-tuning can improve results:

- Setting the style, tone, format, or other qualitative aspects
- Improving reliability at producing a desired output
- Correcting failures to follow complex prompts
- Handling many edge cases in specific ways
- Performing a new skill or task that's hard to articulate in a prompt

# MetaLearning: learning to learn

**Learning across tasks** for rapid adaptation,  
a problem setting that is often formalized as few-shot learning



# MetaLearning: learning to learn

## Model-Agnostic Meta-Learning MAML over different **tasks**

---

**Algorithm 1** Model-Agnostic Meta-Learning

---

**Require:**  $p(\mathcal{T})$ : distribution over tasks

**Require:**  $\alpha, \beta$ : step size hyperparameters

1: randomly initialize  $\theta$

2: **while** not done **do**

3:    Sample batch of tasks  $\mathcal{T}_i \sim p(\mathcal{T})$

4:    **for all**  $\mathcal{T}_i$  **do**

5:     Evaluate  $\nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$  with respect to  $K$  examples

6:     Compute adapted parameters with gradient descent:  $\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$

7:    **end for**

8:    Update  $\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$

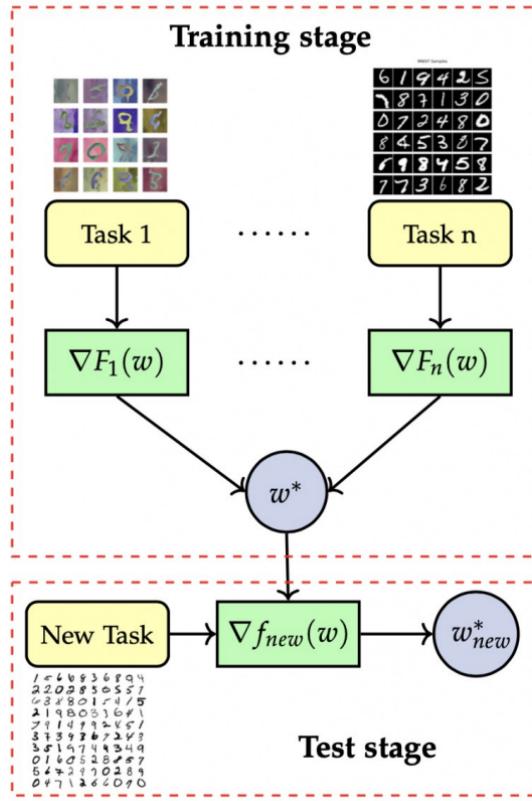
$\beta$  is the meta step size.

9: **end while**

---

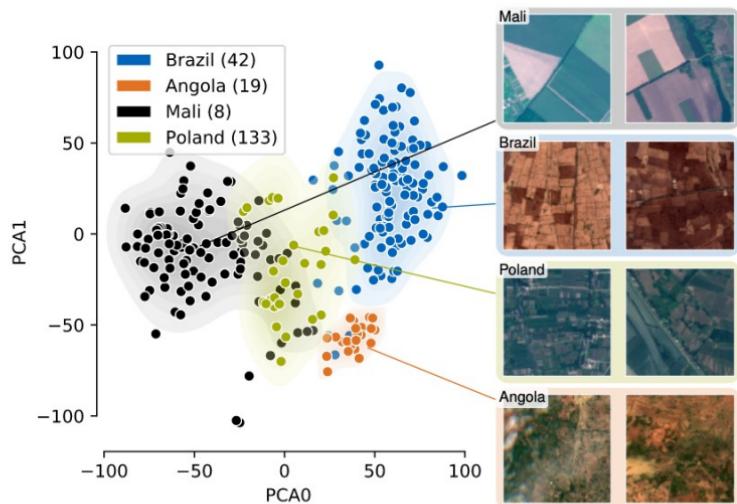
# MetaLearning

## Model-Agnostic Meta-Learning MAML



# MetaLearning

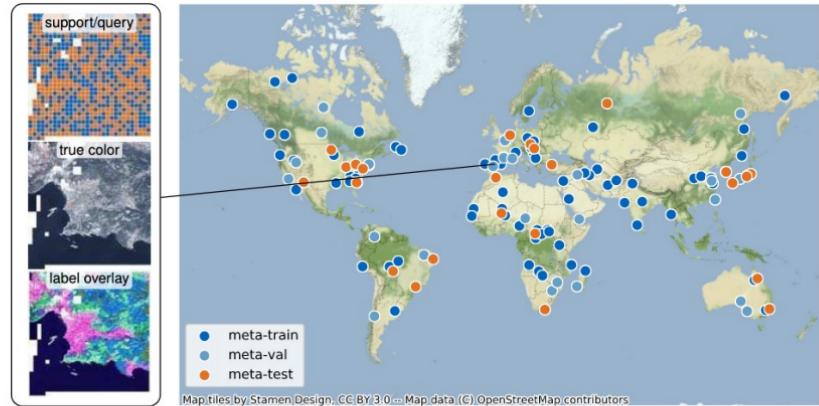
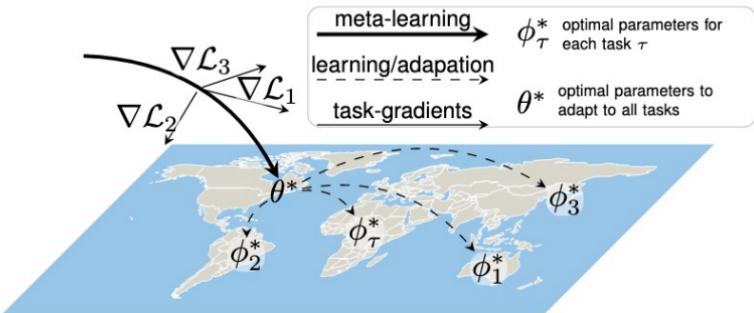
## Example Meta-Learning for Few-Shot Land Cover Classification



Principal component analysis (PCA) on VGG-16 features of cropland images from different countries. Representations of the same class vary geographically; applying models trained on one geography to another would violate the assumption in traditional supervised learning that train and test distributions are the same.

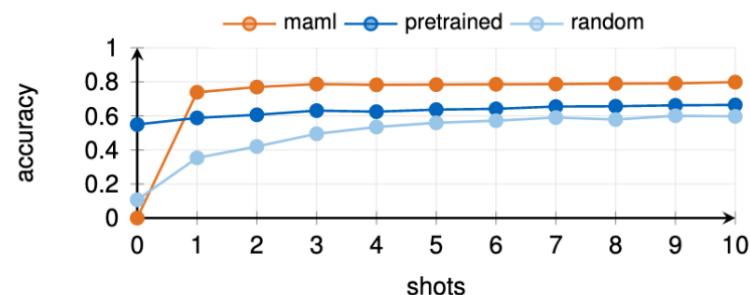
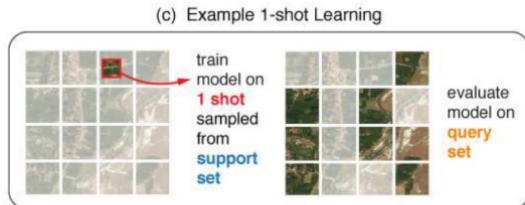
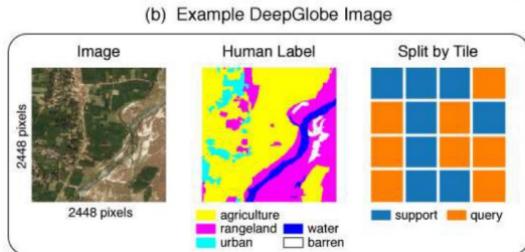
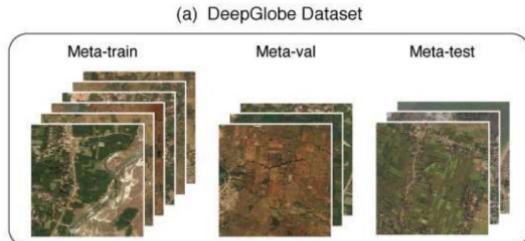
# MetaLearning

## Example Meta-Learning for Few-Shot Land Cover Classification



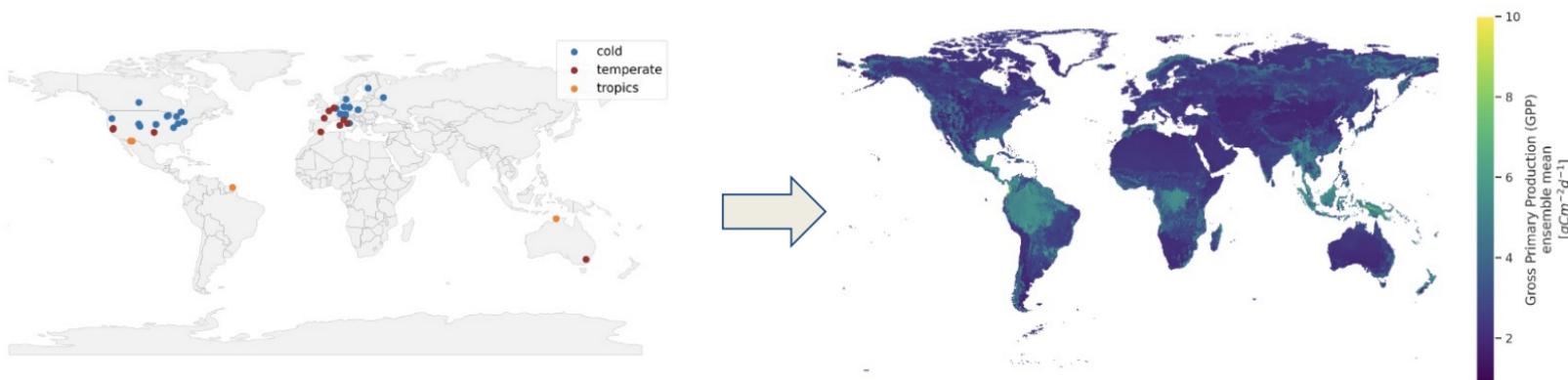
# MetaLearning

## Example Meta-Learning for Few-Shot Land Cover Classification



# MetaLearning

## Example Meta-Learning for Carbon Fluxes



# MetaLearning

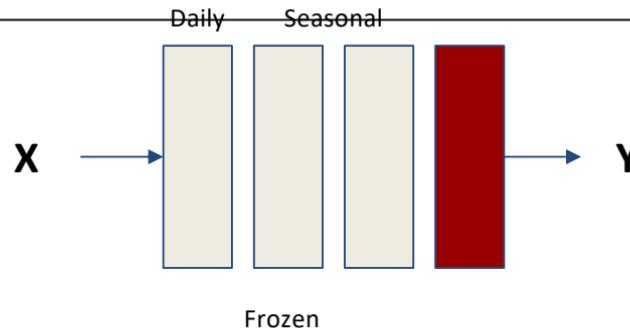
## Example Meta-Learning for Carbon Fluxes

### Data processing

*Balancing the data*

Idea: get the same number of data across classes

Problem: reduce sample size significantly



### Modeling

*Transfer learning*

Idea: train a model on classes with a lot of data, and transfer the learned knowledge to new domains (probably with few data)

Problem(s):

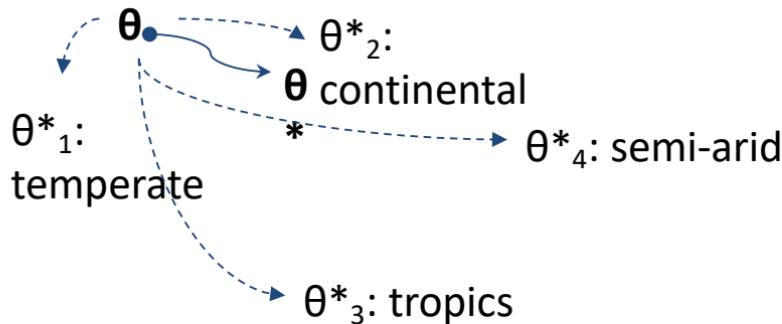
- (i) freeze/unfreeze which part of the model?
- (ii) assumes frozen parts of the model are not changing across tasks → not capturing the right representation

# MetaLearning

## Example Meta-Learning for Carbon Fluxes

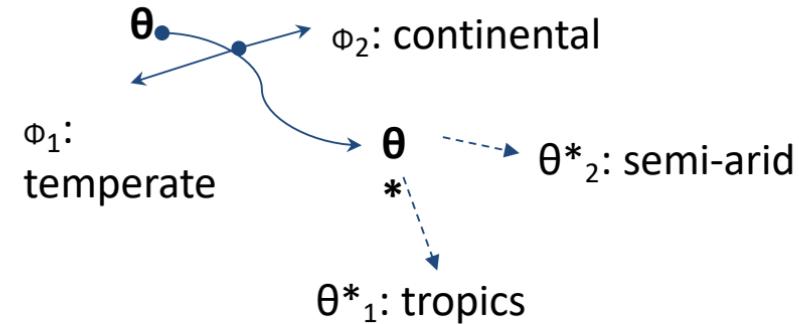
Gradient descent

### Weight Space ( $\theta$ )



Meta-learning

### Weight Space ( $\theta$ )

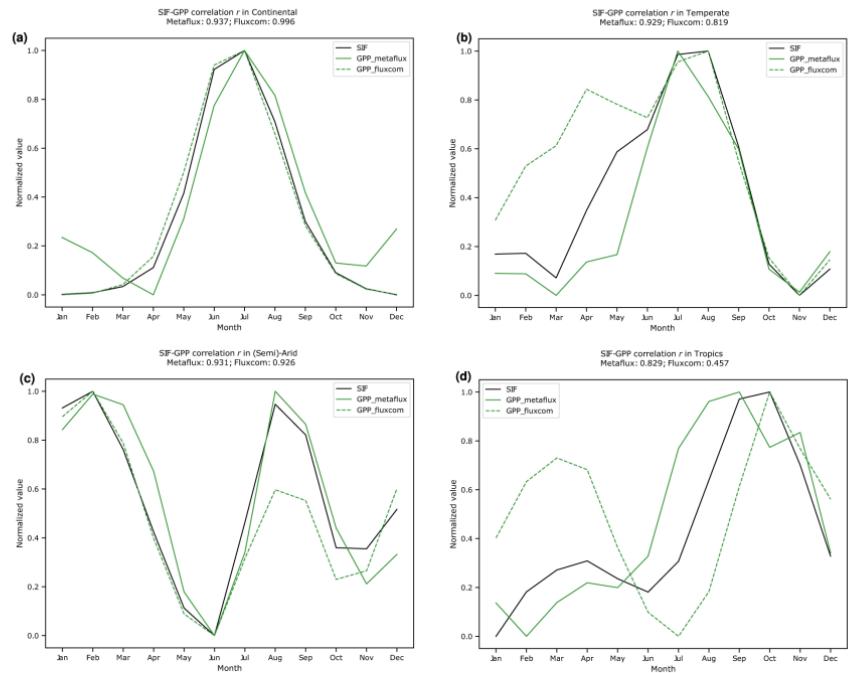


→ meta-learning from data-abundant stations  
→ adaptation from data-sparse stations

# MetaLearning

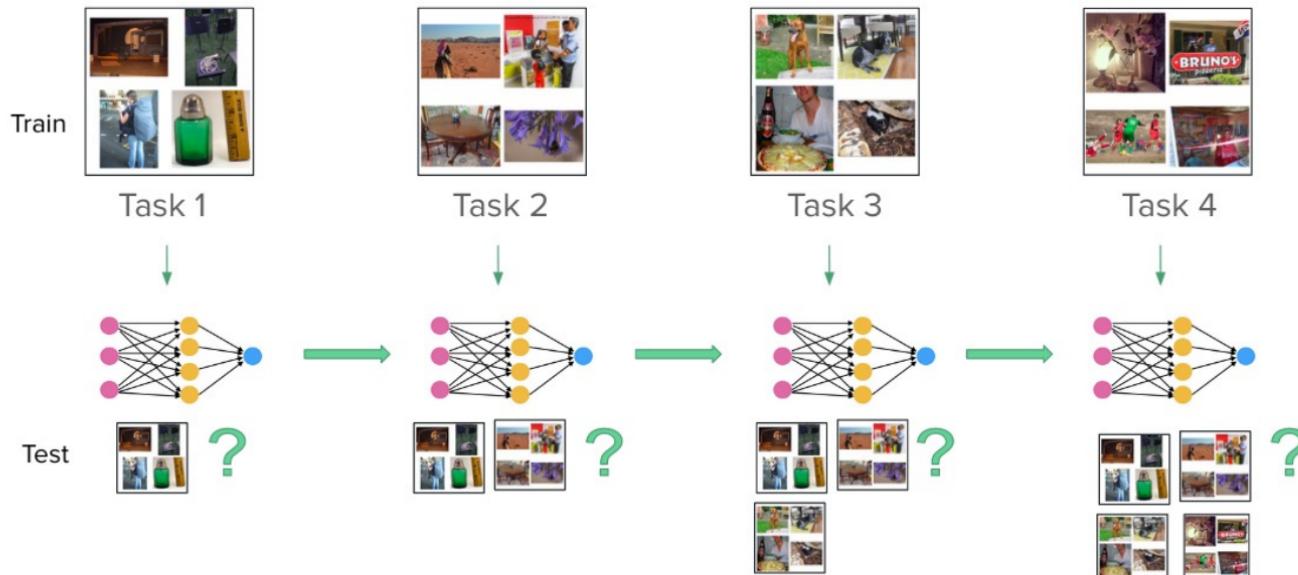
## Example Meta-Learning for Carbon Fluxes

Climate zones	$r(GPP_{fluxcom}, SIF)$	$r(GPP_{metaflux}, SIF)$
Semi-arid	$0.726 \pm 0.165$	<b><math>0.856 \pm 0.083</math></b>
Continental	<b><math>0.965 \pm 0.002</math></b>	$0.937 \pm 0.012$
Temperate	$0.826 \pm 0.021$	<b><math>0.919 \pm 0.002</math></b>
Tropics	$0.343 \pm 0.164$	<b><math>0.546 \pm 0.299</math></b>



# La MAML

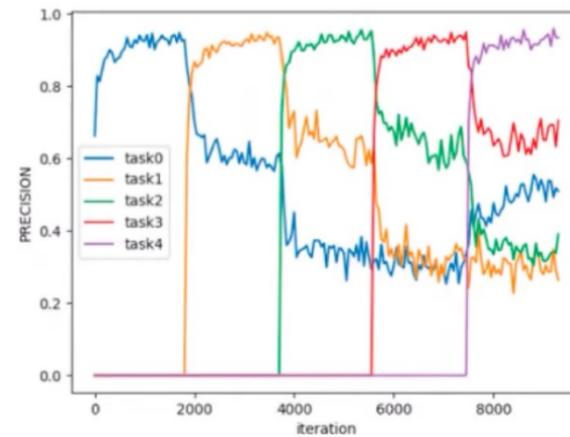
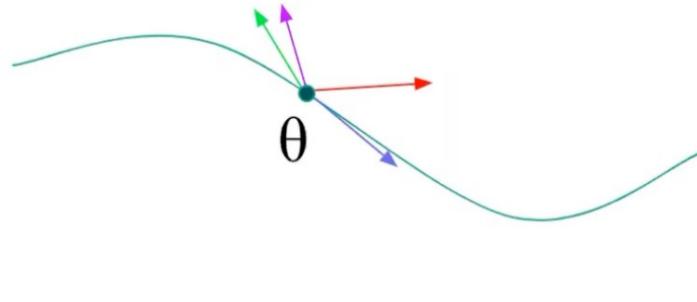
**Continual learning (CL)** problem involves training models with limited capacity to perform well on a set of an unknown number of *sequentially arriving tasks*. Data are not i.i.d when they arrive sequentially.



Potential issue: forget all tasks (interference)

## Catastrophic Forgetting

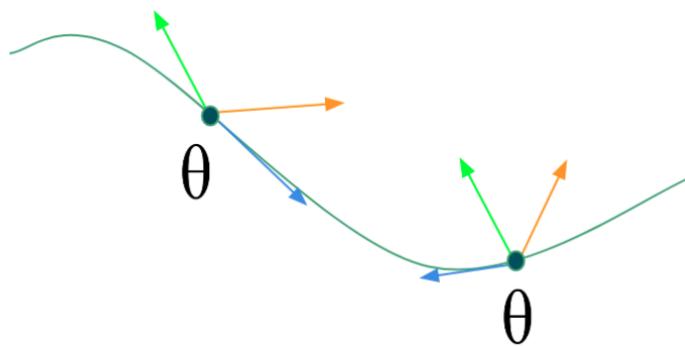
Gradient directions for various tasks at any time  $t$



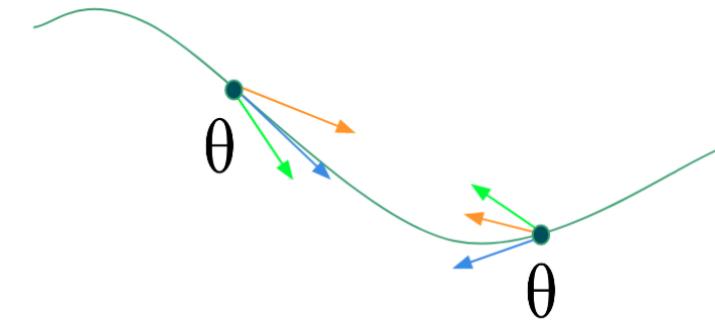
# MetaLearning

Potential issue: forget all tasks (intefreference), gradient-alignment is essential to make shared progress on task-wise objectives under limited availability of training-data.

Gradient directions for various tasks at any time t



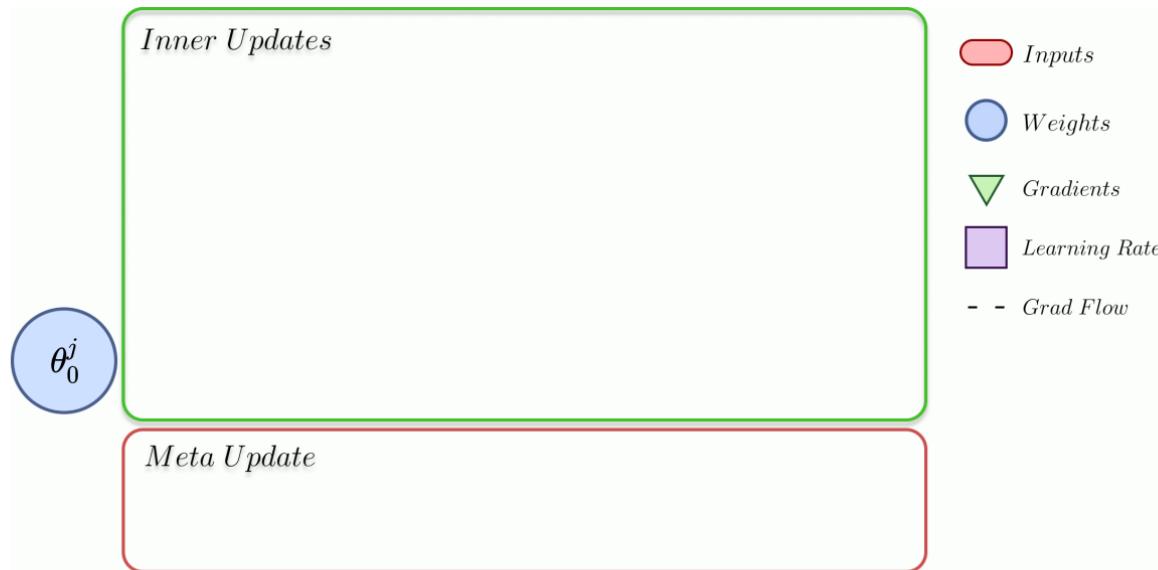
Interference



Alignment

# La MAML

**La-MAML** algorithm: For every batch of data, the initial weights undergo a series of  $k$  fast updates to obtain  $\theta_{jk}$  (here  $j = 0$ ), which is evaluated against a meta-loss to backpropagate gradients with respect to the weights  $\theta_{j0}$  and learning rates  $\alpha_0$ . First  $\alpha_0$  is updated to  $\alpha_1$  which is then used to update  $\theta_{j0}$  to  $\theta_{j1}$ . The blue boxes indicate fast weights while the green boxes indicate gradients for the slow updates. LRs and weights are updated in an asynchronous manner.



# La MAML

Gradient of Learning rates (alpha) :

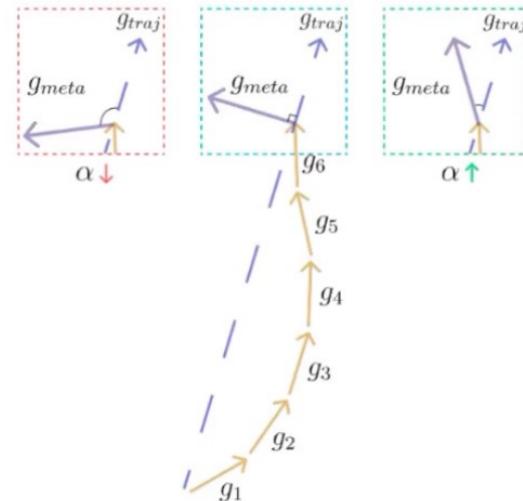
$$g_{\text{MAML}}(\alpha) = \frac{\partial}{\partial \alpha} L_m(\theta_k)$$
$$= \left( \frac{\partial}{\partial \theta_k} L_m(\theta_k) \right) \left( - \sum_{i=0}^{k-1} \frac{\partial}{\partial \theta_i} L_i(\theta_i) \right)$$

Outer Gradient

Old + New Tasks

Inner Gradient

New Task



# La MAML

Updating the network weights and LRs asynchronously in the meta-update

Two losses to simulate continual learning in the inner loop and test forgetting in the outer loop  
is referred to as the Online-Aware Meta Learning objective.

---

**Algorithm 1** La-MAML : Look-ahead MAML

---

**Input:** Network weights  $\theta$ , LRs  $\alpha$ , inner objective  $\ell$ , meta objective  $L$ , learning rate for  $\alpha$  :  $\eta$

$j \leftarrow 0, R \leftarrow \{\}$  ▷ Initialise Replay Buffer

**for**  $t := 1$  **to**  $T$  **do**

**for**  $ep := 1$  **to**  $num_{epochs}$  **do**

**for** batch  $b$  **in**  $(X^t, Y^t) \sim D_t$  **do**

$k \leftarrow sizeof(b)$

$b_m \leftarrow Sample(R) \cup b$

**for**  $n = 0$  **to**  $k - 1$  **do**

                Push  $b[k']$  to R with reservoir sampling

$\theta_{k'+1}^j \leftarrow \theta_{k'}^j - \alpha^j \cdot \nabla_{\theta_{k'}^j}$

**end for**

$\alpha^{j+1} \leftarrow \alpha^j - \eta \nabla_{\alpha^j} L_t(\theta_k^j, b_m)$

$\theta_0^{j+1} \leftarrow \theta_0^j - max(0, \alpha^{j+1}) \cdot \nabla_{\theta_0^j} L_t(\theta_k^j, b_m)$

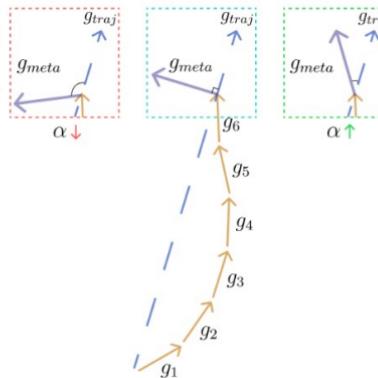
$j \leftarrow j + 1$

**end for**

**end for**

**end for**

---

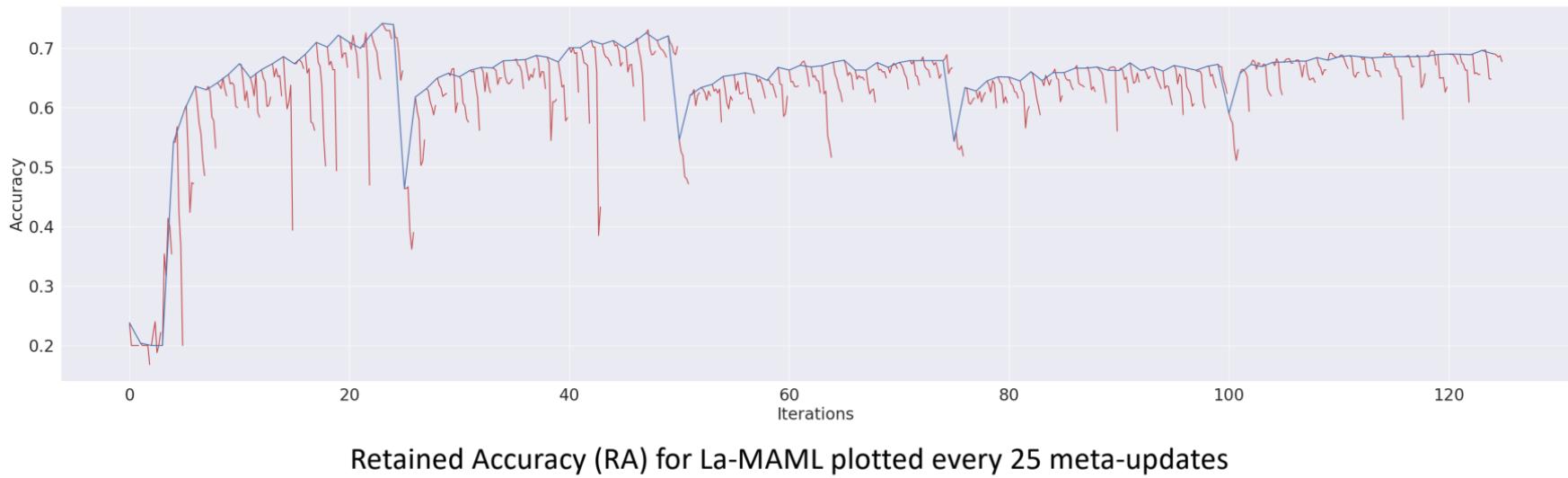


**(a) Figure 2:** Different scenarios for the alignment of  $g_{traj}$  (blue dashed line) and  $g_{meta}$ , going from interference (left) to alignment (right). Yellow arrows denote the *inner updates*. The LR  $\alpha$  increases (decreases) when gradients align (interfere).

# La MAML

Updating the network weights and LRs asynchronously in the meta-update

Two losses to simulate continual learning in the inner loop and test forgetting in the outer loop  
is referred to as the Online-Aware Meta Learning objective.



# Reinforcement-Learning

# Reinforcement learning

Example for video games



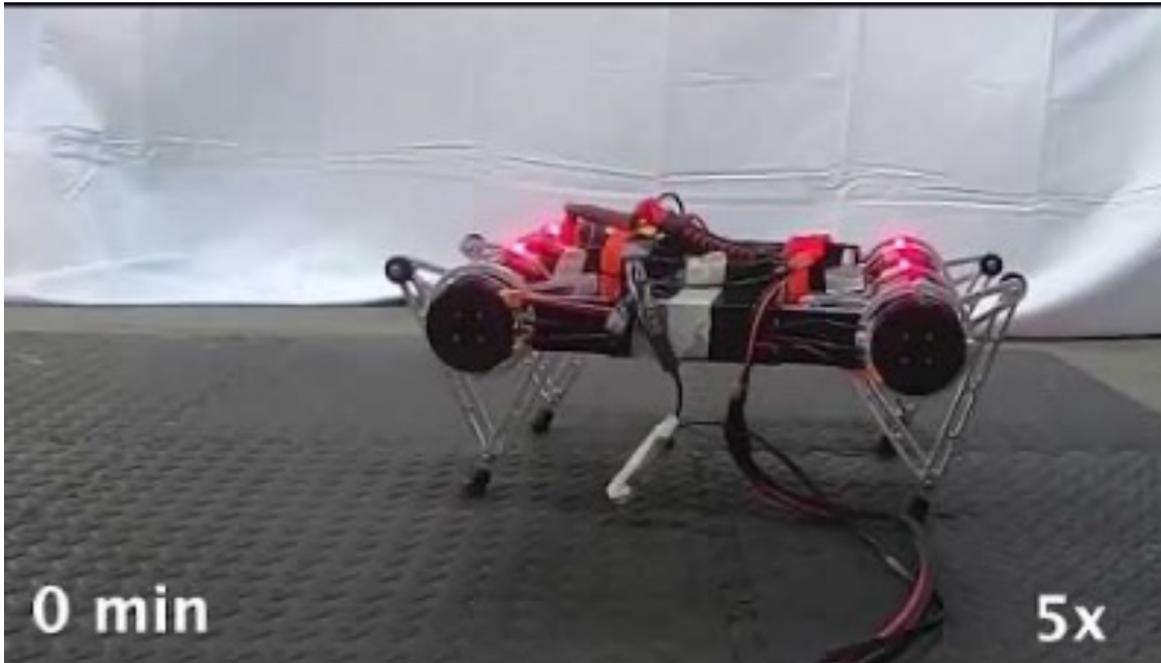
# Reinforcement learning

Example for walking robots



# Reinforcement learning

Example for walking robots

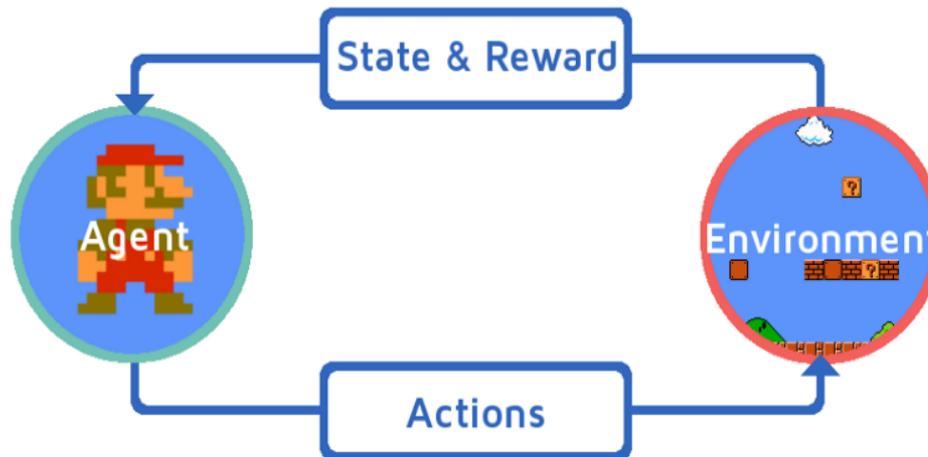


# Reinforcement learning: learning from rewards

- The Reward  $R_t$  defines the goal of an RL problem.
  - Gives the agent a sense of what is good and bad.
  - A reward of a higher magnitude is better.
  - Usually a function of environment state (situation).
- 
- Explore and exploit:
    - Reinforcement Learning is like trial and error
  - The agent should explore the environment to search for better policies.
  - After selection of optimal policy, the agent maximises the rewards.
    - Exploration finds more information about the environment.
    - Exploitation uses known information to maximise reward.

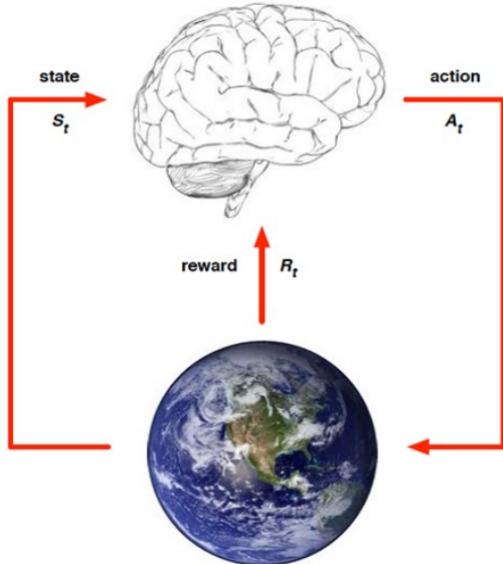
# Reinforcement learning: learning from rewards

- The RL Agent receives **state  $S^0$**  from the **environment** i.e. Mario
- Based on that **state  $S^0$** , the RL agent takes an **action  $A^0$** , say — our RL agent moves right. Initially, this is random.
- Now, the environment is in a new state  **$S^1$**  (new frame from Mario or the game engine)
- Environment gives some **reward  $R^1$**  to the RL agent. It probably gives a  $+1$  because the agent is not dead yet.
- This RL loop continues until we are dead or we reach our destination, and it continuously outputs a sequence of **state, action and reward**.
- Aim of our RL agent is to maximize the reward

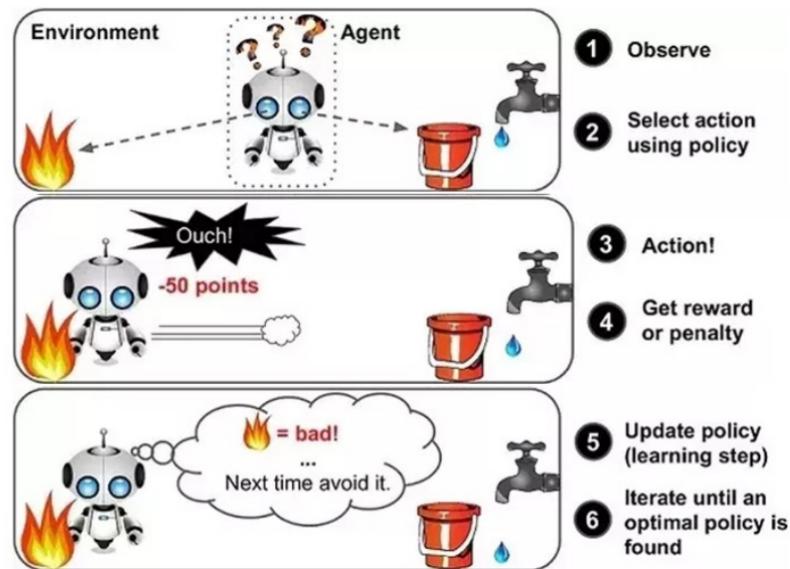


# Reinforcement learning: learning from rewards

## Example



An example



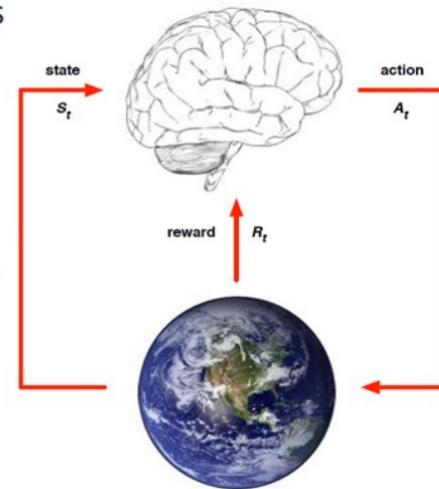
## RL Agent Basics

- An RL agent may include one or more of these components
  - Policy: agent's behaviour function
  - Value function: how good is each state and/or action
  - Model: agent's representation of the environment

### Characteristics

What makes reinforcement learning different from other machine learning paradigms?

- There is no supervisor, only a *reward* signal
- Feedback is delayed, not instantaneous
- Time really matters (sequential, non i.i.d data)
- Agent's actions affect the subsequent data it receives



# Reward maximization

- The cumulative rewards at each time step with the respective action is written as:

$$G_t = \sum_{k=0}^T R_{t+k+1}$$

- But discounting might be important: reward “closer” might be discounted (like yield/rate)

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \text{ where } \gamma \in [0, 1]$$

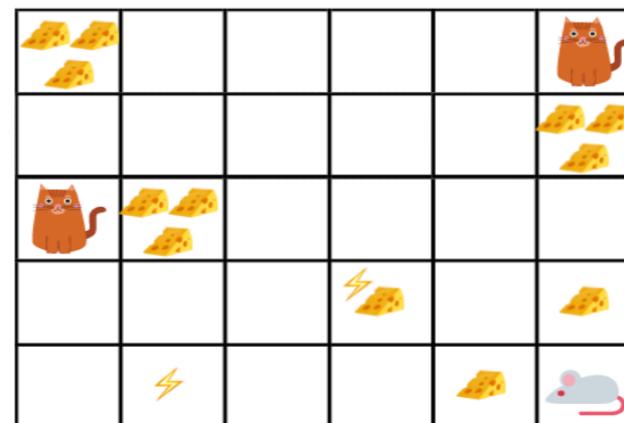
$$R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots$$

Sigma (Sum up)

Discount rate

Rewards received at each state

Expanded form of the Equation



# Reinforcement learning approaches

## 1. Policy-based approach

Mapping each state to the best action (deterministic or stochastic- gives a distribution)

$$a = \pi(s)$$

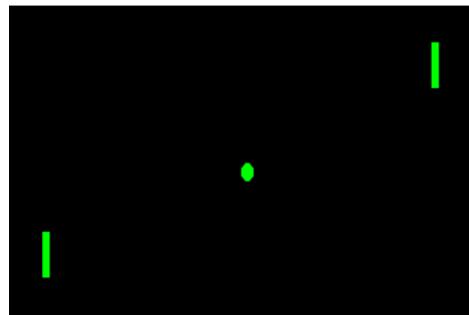
a: actions. S: state,  $\pi$  : Policy function

## 2. Value-based

optimize the value function: maximum expected future reward

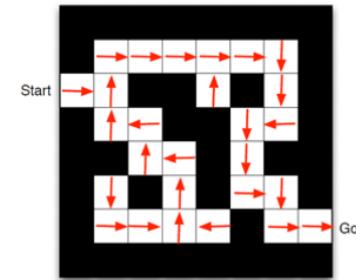
$$v_{\pi}(s) = \mathbb{E}_{\pi} [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s]$$

Expected	Reward discounted	Given that state
----------	-------------------	------------------

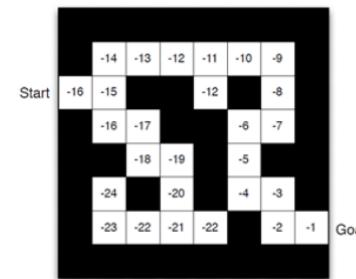


## Policy, Value Function & Model

- A **policy** is the agent's behaviour
- It is a map from state to action, e.g.
- Deterministic policy:  $a = \pi(s)$
- Stochastic policy:  $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$



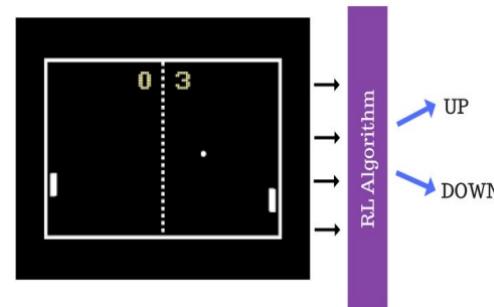
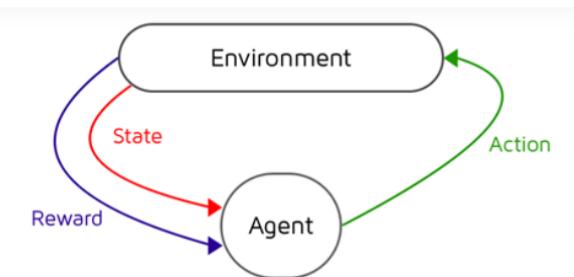
- Arrows represent policy  $\pi(s)$  for each state  $s$



- Numbers represent value  $v_\pi(s)$  of each state  $s$

# Reinforcement learning approaches

Not all actions were bad, maybe only last moves



Do not discard entire sequence by only weight last ones negatively (due to sparsity in reward setting).

