Meta-Learning

Designing, Visualizing and Understanding Deep Neural Networks

CS W182/282A

Instructor: Sergey Levine
UC Berkeley

What is meta-learning?

- If you’ve learned 100 tasks already, can you figure out how to *learn* more efficiently?
  - Now having multiple tasks is a huge advantage!
- Meta-learning = *learning to learn*
- In practice, very closely related to multi-task learning
- Many formulations
  - Learning an optimizer
  - Learning an RNN that ingests experience
  - Learning a representation

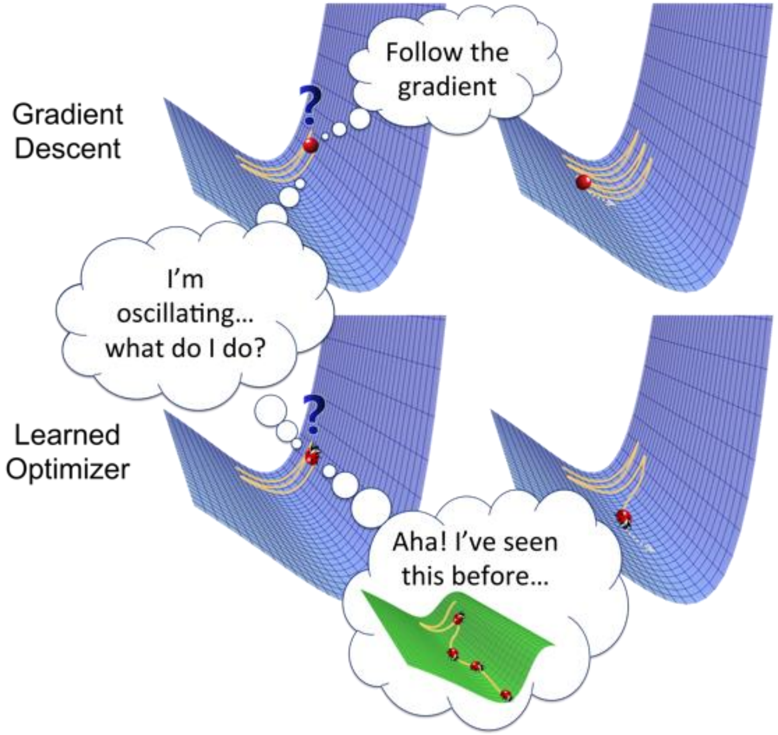


image credit: Ke Li

Why is meta-learning a good idea?

- Deep learning works very well, but requires **large** datasets
- In many cases, we only have a small amount of data available (e.g., some specific computer vision task), but we might have lots of data of a similar type for other tasks (e.g., other object classification tasks)
- How does a *meta-learner* help with this?
  - Use plentiful prior tasks to meta-train a model that can learn a new task quickly with only a few examples
  - Collect a small amount of labeled data for the new task
  - Learn a model on this new dataset that generalizes broadly

Meta-learning with supervised learning

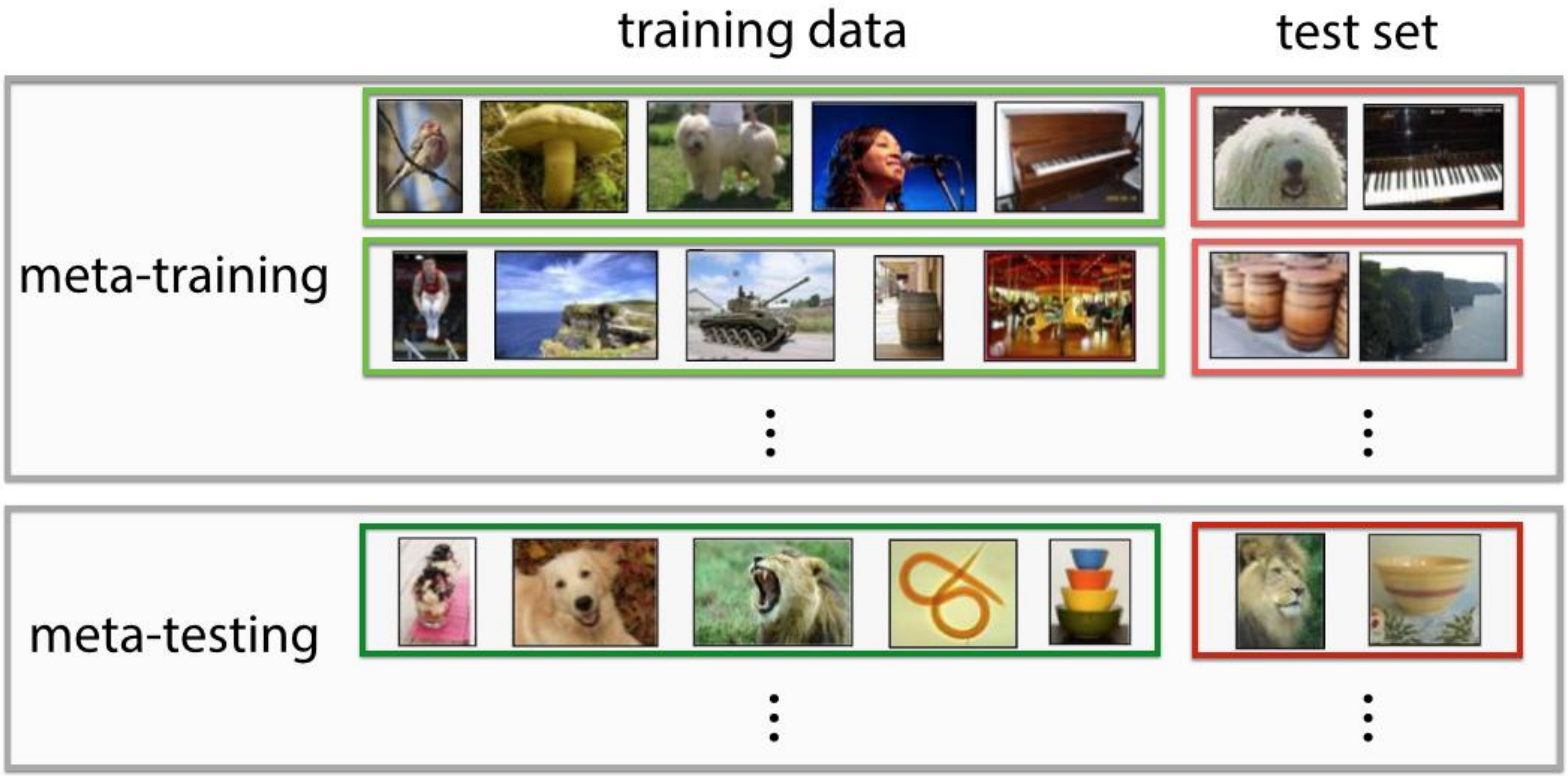
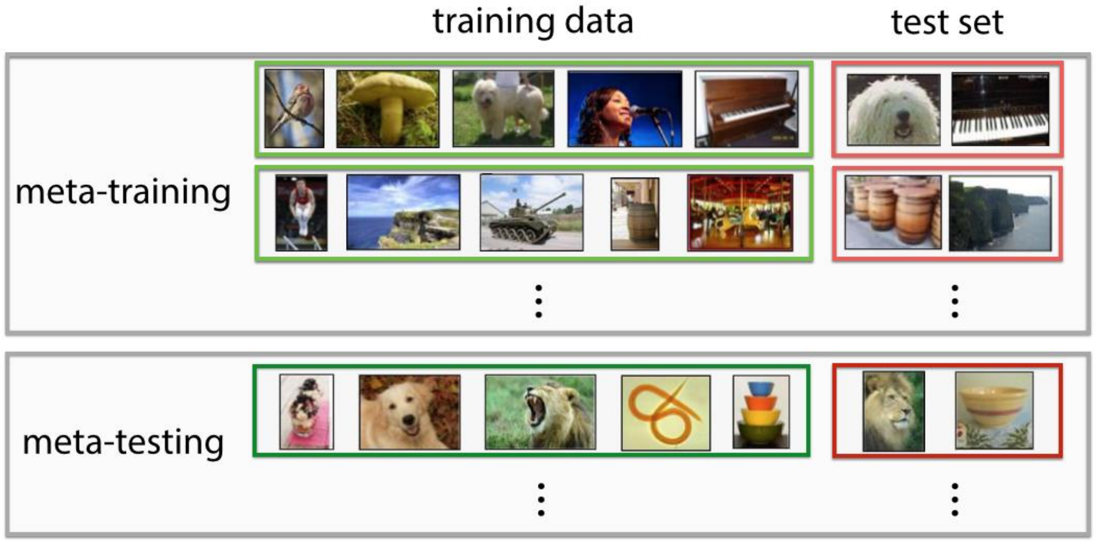


image credit: Ravi & Larochelle '17

Meta-learning with supervised learning

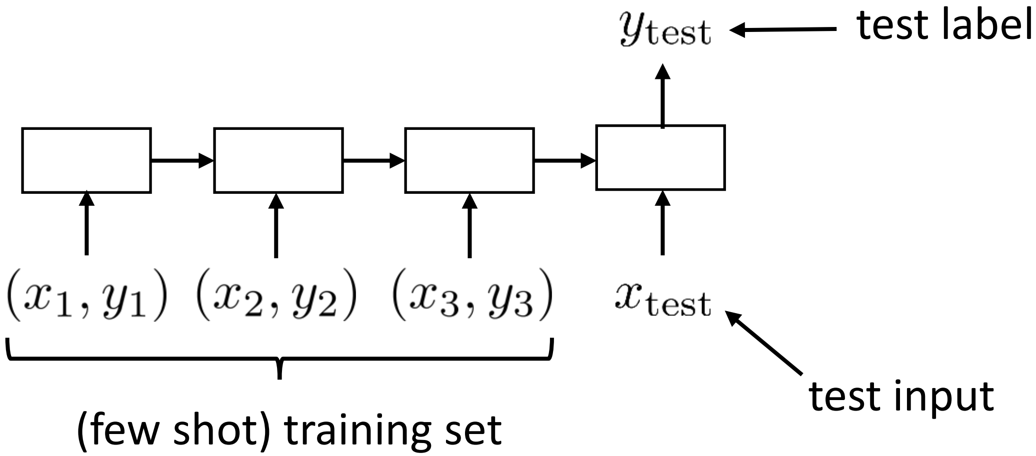


supervised learning: $f(x) \rightarrow y$

input (e.g., image) output (e.g., label)

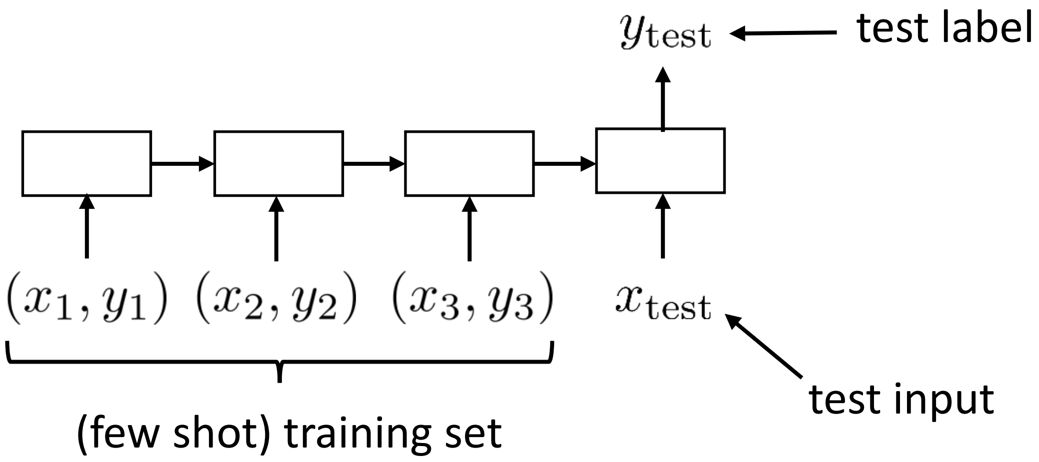
supervised meta-learning: $f(\mathcal{D}^{\text{tr}}, x) \rightarrow y$

training set



- How to read in training set?
  - Many options, RNNs can work
  - More on this later

What is being "learned"?



supervised meta-learning: $f(\mathcal{D}^{\text{tr}}, x) \rightarrow y$

"Generic" learning:

$$\theta^\star = \arg\min_\theta \mathcal{L}(\theta, \mathcal{D}^{\text{tr}})$$

$$= f_{\text{learn}}(\mathcal{D}^{\text{tr}})$$

"Generic" meta-learning:

$$\theta^\star = \arg\min_\theta \sum_{i=1}^{n} \mathcal{L}(\phi_i, \mathcal{D}_i^{\text{ts}})$$

$$\text{where } \phi_i = f_\theta(\mathcal{D}_i^{\text{tr}})$$

What is being “learned”?

“Generic” learning:

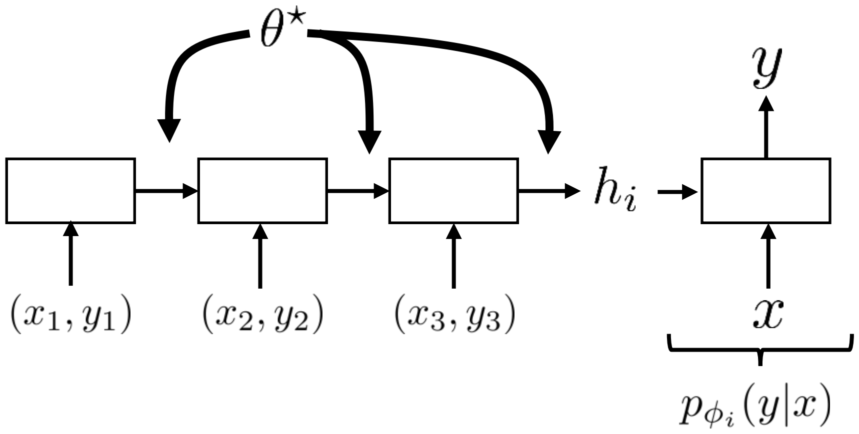
$$\theta^\star = \arg\min_\theta \mathcal{L}(\theta, \mathcal{D}^{\text{tr}})$$

$$= f_{\text{learn}}(\mathcal{D}^{\text{tr}})$$

“Generic” meta-learning:

$$\theta^\star = \arg\min_\theta \sum_{i=1}^{n} \mathcal{L}(\phi_i, \mathcal{D}_i^{\text{ts}})$$

$$\text{where } \phi_i = f_\theta(\mathcal{D}_i^{\text{tr}})$$



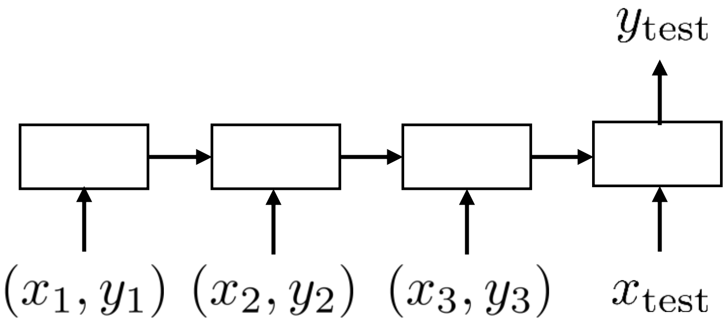
RNN hidden state

meta-learned weights

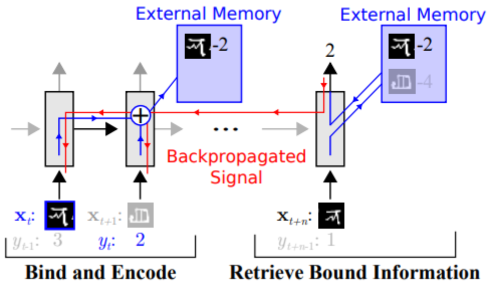
$$\phi_i = [h_i, \theta_p]$$

Meta-learning methods

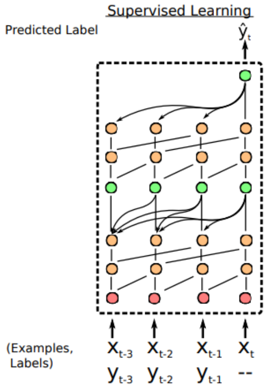
black-box meta-learning



some kind of network that can read in an entire (few-shot) training set

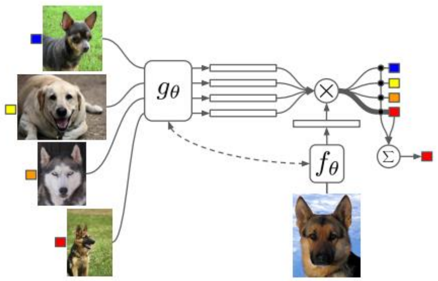


Santoro et al. **Meta-Learning with Memory-Augmented Neural Networks**. 2016.

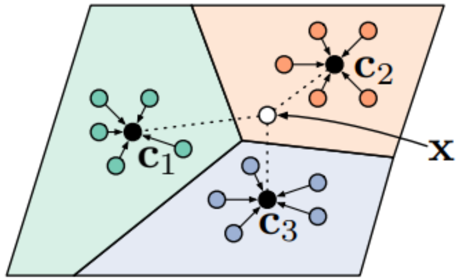


Mishra et al. **A Simple Neural Attentive Meta-Learner**. 2018.

non-parametric meta-learning

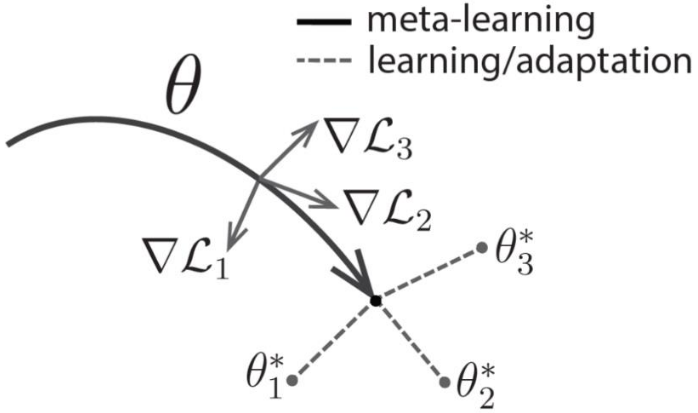


Vinyals et al. **Matching Networks for One Shot Learning**. 2017.



Snell et al. **Prototypical Networks for Few-shot Learning**. 2018.

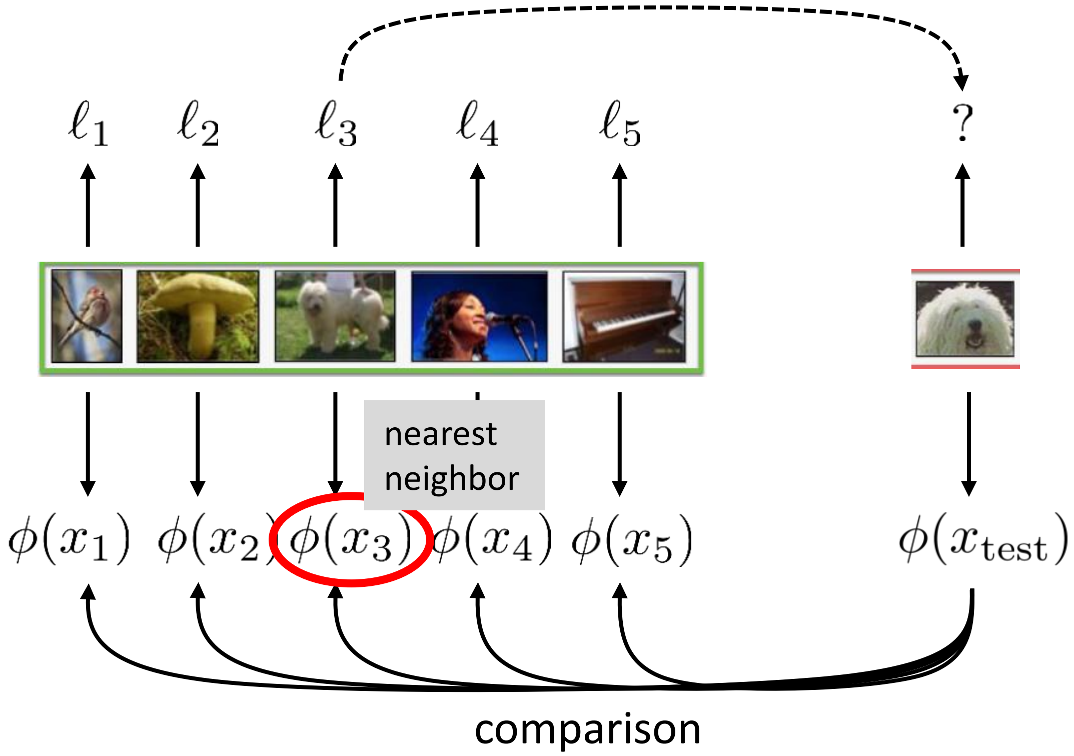
gradient-based meta-learning



Finn et al. **Model-Agnostic Meta-Learning**. 2018.

Non-Parametric & Gradient-Based Meta-Learning

Basic idea



why does this work?

that is, why does the nearest neighbor have the right class?

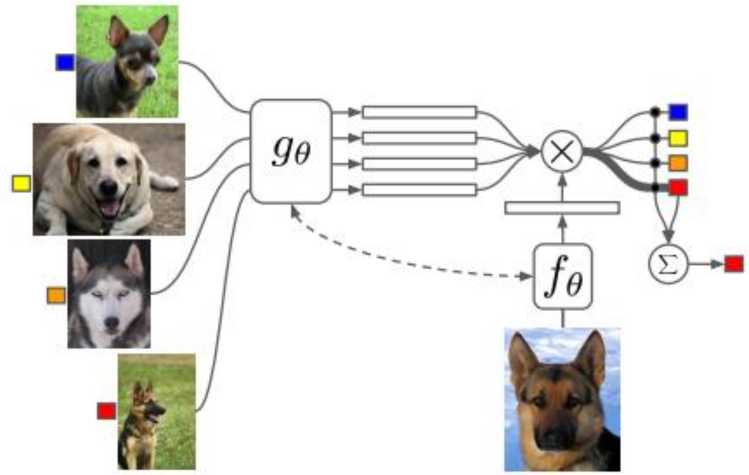
because we **meta-train** the features so that this produces the right answer!

$$p_{\text{nearest}}(x_k^{\text{tr}} | x_j^{\text{ts}}) \propto \exp(\phi(x_k^{\text{tr}})^T \phi(x_j^{\text{ts}}))$$

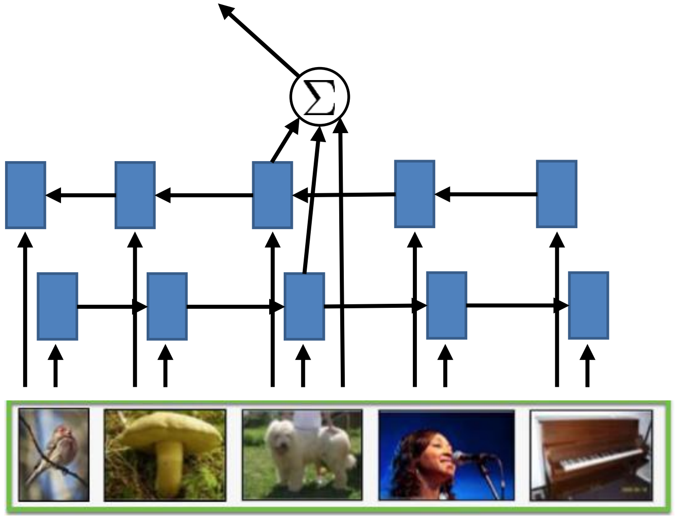
$$p_\theta(y_j^{\text{ts}} | x_j^{\text{ts}}, \mathcal{D}_i^{\text{tr}}) = \underbrace{\sum_{k: y_k^{\text{tr}} = y_j^{\text{ts}}} p_{\text{nearest}}(x_k^{\text{tr}} | x_j^{\text{ts}})}_{\text{all training points that have this label}}$$

$$\theta^\star = \arg\min_\theta \sum_{i=1}^n \mathcal{L}(\underbrace{f_\theta(\mathcal{D}_i^{\text{tr}})}_{\text{learned (soft) nearest neighbor classifier}}, \mathcal{D}_i^{\text{ts}}) = -\sum_{i=1}^n \sum_{j=1}^m \log p_\theta(y_j^{\text{ts}} | x_j^{\text{ts}}, \mathcal{D}_i^{\text{tr}})$$

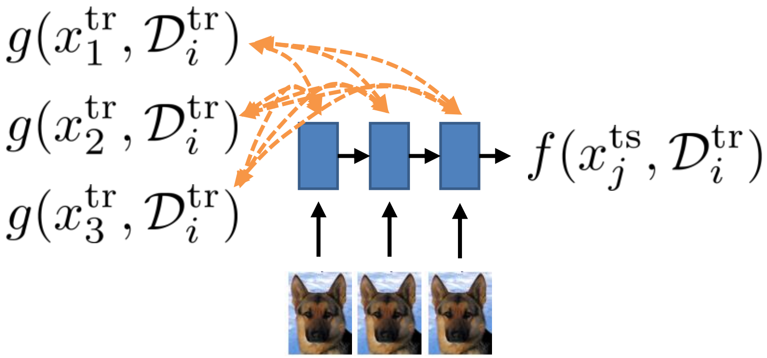
Matching networks



$g(x_k^{\text{tr}}, \mathcal{D}_i^{\text{tr}})$ bidirectional LSTM embedding



$f(x_j^{\text{ts}}, \mathcal{D}_i^{\text{tr}})$ attentional LSTM embedding



$$p_\theta(y_j^{\text{ts}} | x_j^{\text{ts}}, \mathcal{D}_i^{\text{tr}}) = \sum_{k: y_k^{\text{tr}} = y_j^{\text{ts}}} p_{\text{nearest}}(x_k^{\text{tr}} | x_j^{\text{ts}})$$

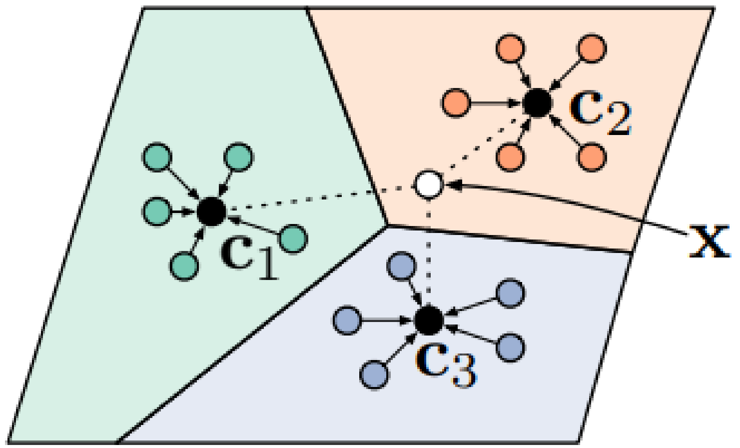
$$p_{\text{nearest}}(x_k^{\text{tr}} | x_j^{\text{ts}}) \propto \exp(g(x_k^{\text{tr}}, \mathcal{D}_i^{\text{tr}})^T f(x_j^{\text{ts}}, \mathcal{D}_i^{\text{tr}}))$$

different nets to embed x^{tr} and x^{ts}

both f and g conditioned on entire set $\mathcal{D}_i^{\text{tr}}$

Vinyals et al. **Matching networks for few-shot learning**. 2016.

Prototypical networks



Two simple ideas compared to matching networks:

1. Instead of "soft nearest neighbor," construct prototype for each class

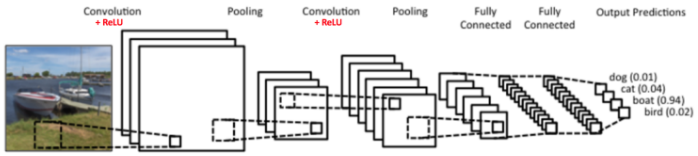
$$p_\theta(y|x_j^{\text{ts}}, \mathcal{D}_i^{\text{tr}}) \propto \exp(c_y^T f(x_j^{\text{ts}})) \quad c_y = \frac{1}{N_y} \sum_{k: y_k^{\text{tr}} = y} g(x_k^{\text{tr}})$$

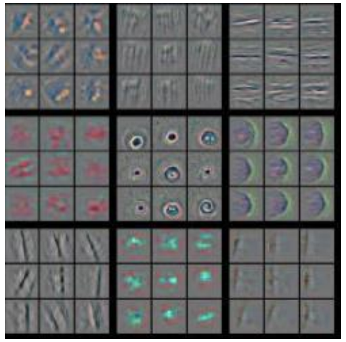
2. Get rid of all the complex junk ~~bidirectional LSTM embedding~~

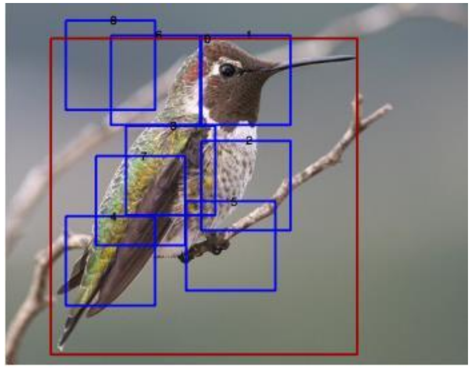
~~attentional LSTM embedding~~

Snell et al. **Prototypical networks for few-shot learning**. 2017.

Back to representations…







is pretraining a *type* of meta-learning?

better features = faster learning of new task!

Meta-learning as an optimization problem

$$\theta^\star = \arg\min_\theta \sum_{i=1}^{n} \mathcal{L}(\phi_i, \mathcal{D}_i^{\mathrm{ts}})$$

$$\text{where } \phi_i = f_\theta(\mathcal{D}_i^{\mathrm{tr}})$$

what if $f_\theta(\mathcal{D}_i^{\mathrm{tr}})$ is just a *finetuning* algorithm?

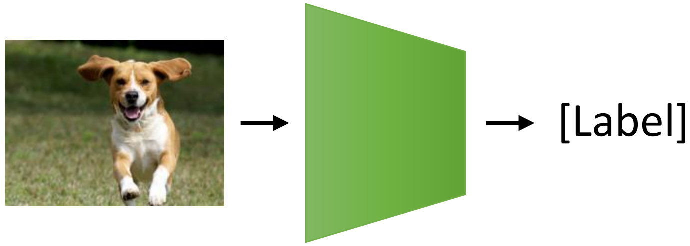
$$f_\theta(\mathcal{D}_i^{\mathrm{tr}}) = \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_i^{\mathrm{tr}})$$

(could take a few gradient steps in general)

This can be trained the same way as any other neural network, by implementing gradient descent as a computation graph and then running backpropagation *through* gradient descent!

Finn, Abbeel, Levine. **Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks.**

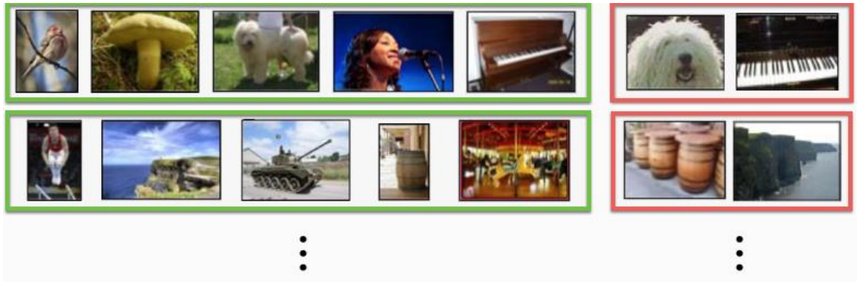
MAML in pictures

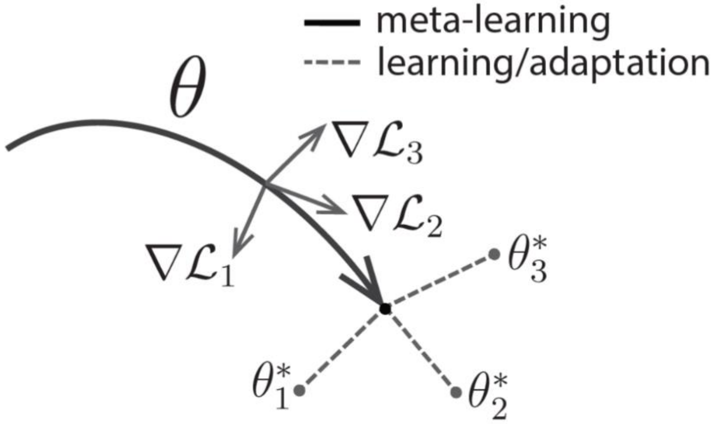


θ

$$\cancel{\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{\text{tr}})}$$

$$\theta \leftarrow \theta - \beta \sum_i \nabla_\theta \mathcal{L}(\theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_i^{\text{tr}}), \mathcal{D}_i^{\text{ts}})$$





θ'

What did we just do??

supervised learning: $f(x) \rightarrow y$

supervised meta-learning: $f(\mathcal{D}^{\mathrm{tr}}, x) \rightarrow y$

model-agnostic meta-learning: $f_{\mathrm{MAML}}(\mathcal{D}^{\mathrm{tr}}, x) \rightarrow y$

$$f_{\mathrm{MAML}}(\mathcal{D}^{\mathrm{tr}}, x) = f_{\theta'}(x)$$

$$\theta' = \theta - \alpha \sum_{(x,y) \in \mathcal{D}^{\mathrm{tr}}} \nabla_\theta \mathcal{L}(f_\theta(x), y)$$

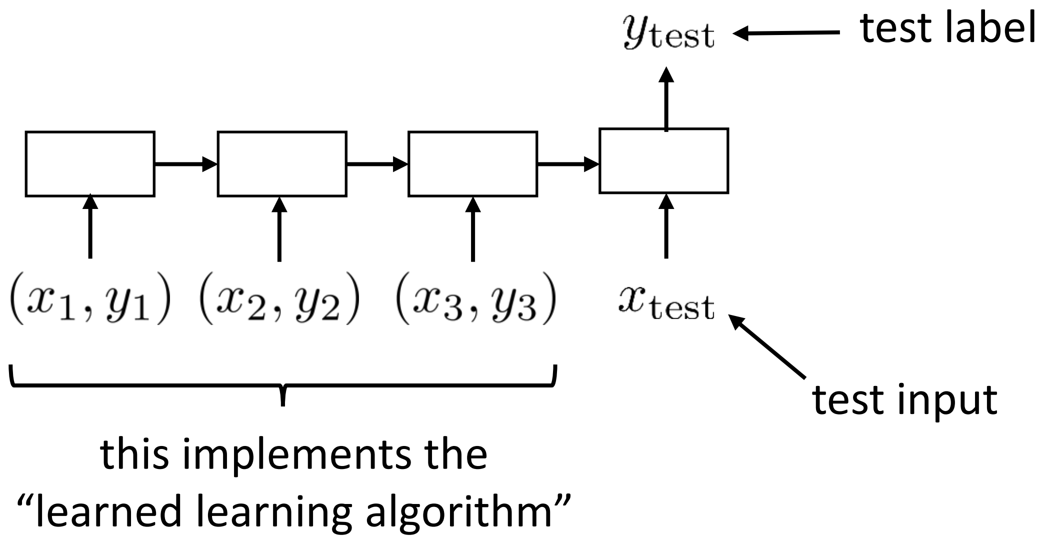
Just another computation graph…

Can implement with any autodiff package (e.g., TensorFlow)

But has favorable inductive bias…

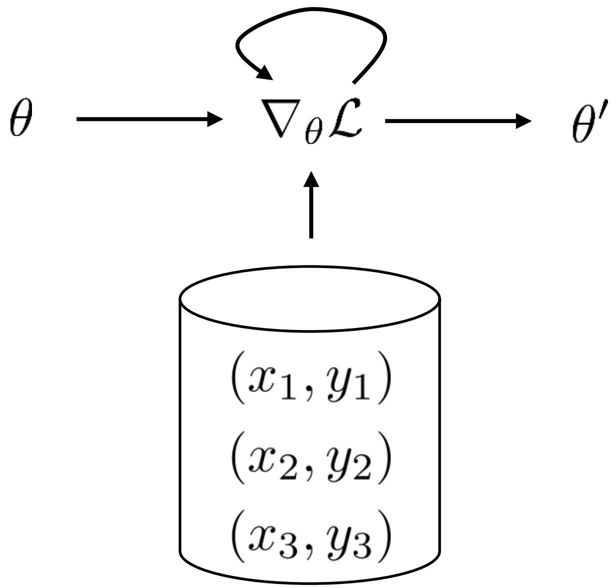
Why does it work?

black-based meta-learning



- Does it converge?
  - Kind of?
- What does it converge to?
  - Who knows…
- What to do if it's not good enough?
  - Nothing…

MAML



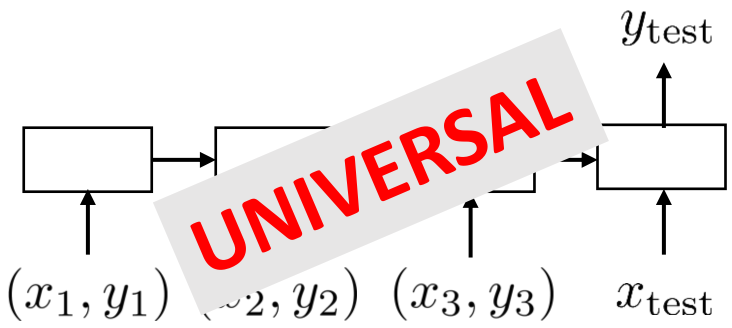
- Does it converge?
  - Yes (it's gradient descent…)
- What does it converge to?
  - A local optimum (it's gradient descent…)
- What to do if it's not good enough?
  - Keep taking gradient steps (it's gradient descent…)

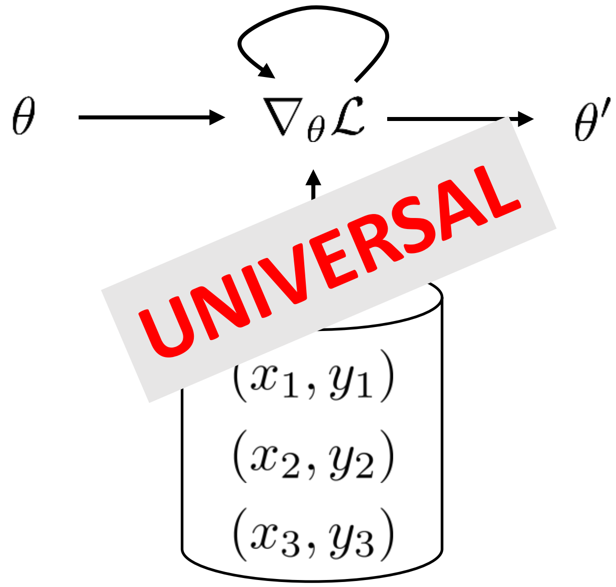
Universality

Did we lose anything?

Universality: meta-learning can learn any "algorithm"

more precisely, can represent any function $f(\mathcal{D}_{\text{train}}, x)$

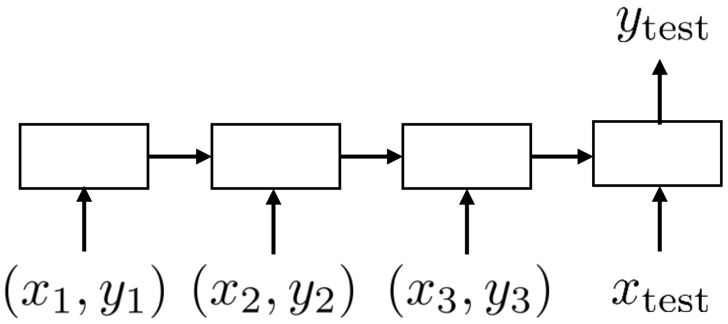




Finn & Levine. "**Meta-Learning and Universality**"

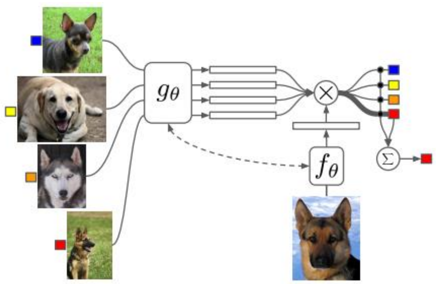
Summary

black-box meta-learning



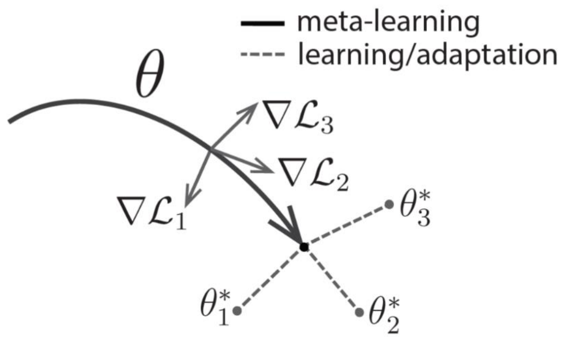
some kind of network that can read in an entire (few-shot) training set

non-parametric meta-learning



Vinyals et al. **Matching Networks for One Shot Learning**. 2017.

gradient-based meta-learning



Finn et al. **Model-Agnostic Meta-Learning**. 2018.

+ conceptually very simple

+ benefits from advances in sequence models (e.g., transformers)

- minimal inductive bias (i.e., *everything* has to be meta-learned)

- hard to scale to “medium” shot (we get long “sequences)

+ can work very well by combining some inductive bias with easy end-to-end optimization

- restricted to classification, hard to extend to other settings like regression or reinforcement learning

- somewhat specialized architectures

+ easy to apply to any architecture or loss function (inc. RL, regression)

+ good generalization to out-of-domain tasks

- meta-training optimization problem is harder, requires more tuning

- requires second derivatives

Meta-Reinforcement Learning

The meta reinforcement learning problem

"Generic" learning:

$$\theta^\star = \arg\min_\theta \mathcal{L}(\theta, \mathcal{D}^{\text{tr}})$$

$$= f_{\text{learn}}(\mathcal{D}^{\text{tr}})$$

"Generic" meta-learning:

$$\theta^\star = \arg\min_\theta \sum_{i=1}^{n} \mathcal{L}(\phi_i, \mathcal{D}_i^{\text{ts}})$$

$$\text{where } \phi_i = f_\theta(\mathcal{D}_i^{\text{tr}})$$

Reinforcement learning:

$$\theta^\star = \arg\max_\theta E_{\pi_\theta(\tau)}[R(\tau)]$$

$$= f_{\text{RL}}(\mathcal{M}) \qquad \mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, r\}$$

MDP

Meta-reinforcement learning:

$$\theta^\star = \arg\max_\theta \sum_{i=1}^{n} E_{\pi_{\phi_i}(\tau)}[R(\tau)]$$

$$\text{where } \phi_i = f_\theta(\mathcal{M}_i)$$

MDP for task i

The meta reinforcement learning problem

$$\theta^\star = \arg\max_\theta \sum_{i=1}^{n} E_{\pi_{\phi_i}(\tau)}[R(\tau)]$$

where $\phi_i = f_\theta(\mathcal{M}_i)$

assumption: $\mathcal{M}_i \sim p(\mathcal{M})$

meta test-time:

sample $\mathcal{M}_{\text{test}} \sim p(\mathcal{M})$, get $\phi_i = f_\theta(\mathcal{M}_{\text{test}})$

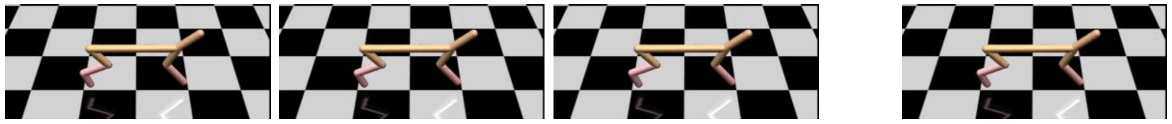
$\{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$

meta-training MDPs

Some examples:

$\mathcal{M}_1$ $\mathcal{M}_2$ $\mathcal{M}_3$ $\mathcal{M}_{\text{test}}$

$\mathcal{M}_1$ $\mathcal{M}_2$ $\mathcal{M}_3$ $\mathcal{M}_{\text{test}}$

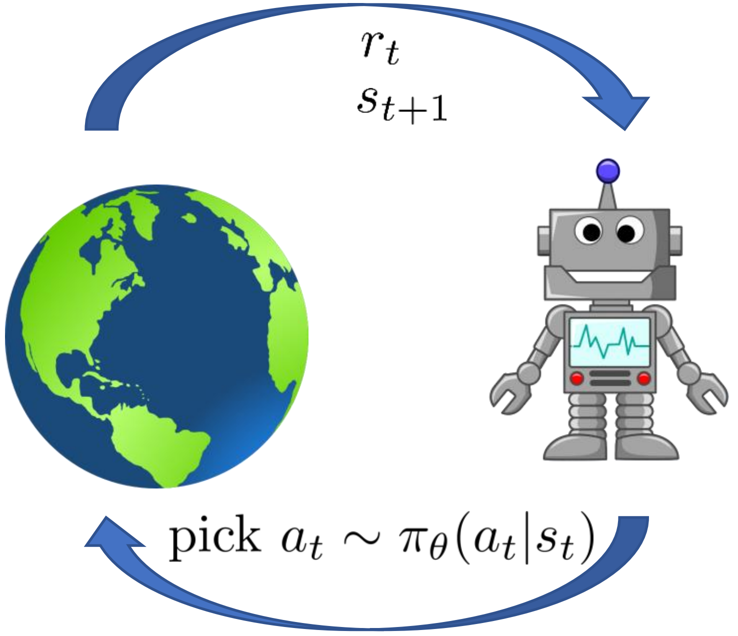


0.5 m/s 0.7 m/s -0.2 m/s -0.7 m/s

Meta-RL with recurrent policies

$$\theta^\star = \arg\max_\theta \sum_{i=1}^{n} E_{\pi_{\phi_i}(\tau)}[R(\tau)]$$

where $\phi_i = f_\theta(\mathcal{M}_i)$



use (s_t, a_t, s_{t+1}, r_t) to improve π_θ

main question: how to implement $f_\theta(\mathcal{M}_i)$?

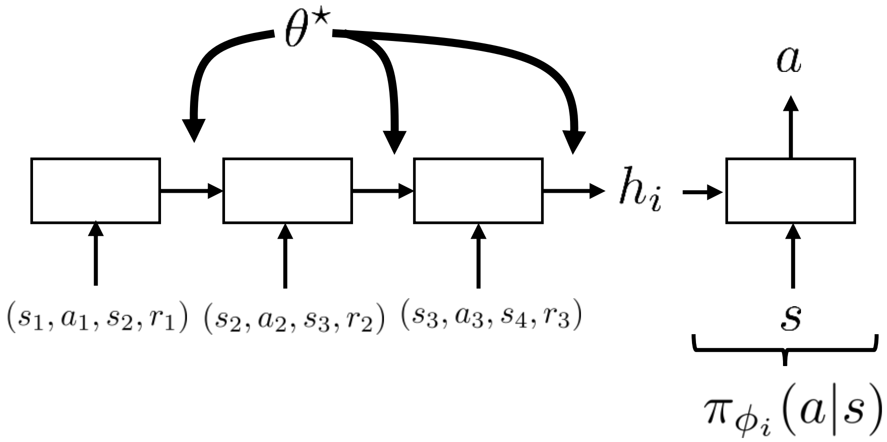
what should $f_\theta(\mathcal{M}_i)$ *do*?

1. improve policy with experience from $\mathcal{M}_i$

   $\{(s_1, a_1, s_2, r_1), \ldots, (s_T, a_T, s_{T+1}, r_T)\}$

2. (new in RL): choose how to interact, i.e. choose a_t

   meta-RL must also *choose* how to *explore*!



RNN hidden state

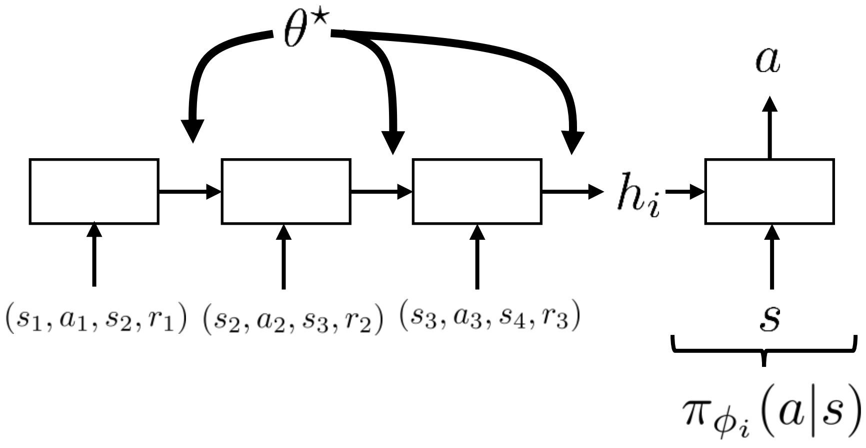
meta-learned weights

as before, $\phi_i = [h_i, \theta_\pi]$

Meta-RL with recurrent policies

$$\theta^\star = \arg\max_\theta \sum_{i=1}^{n} E_{\pi_{\phi_i}(\tau)}[R(\tau)]$$

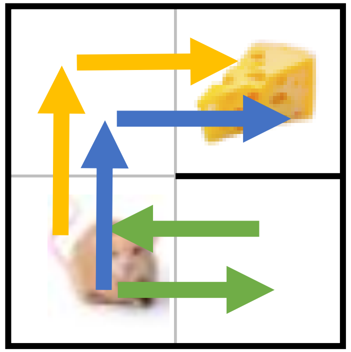
$$\text{where } \phi_i = f_\theta(\mathcal{M}_i)$$

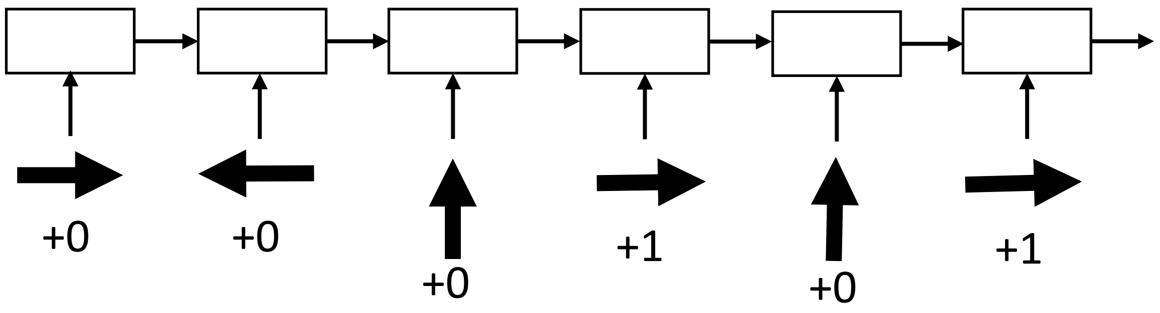


so... we just train an RNN policy?

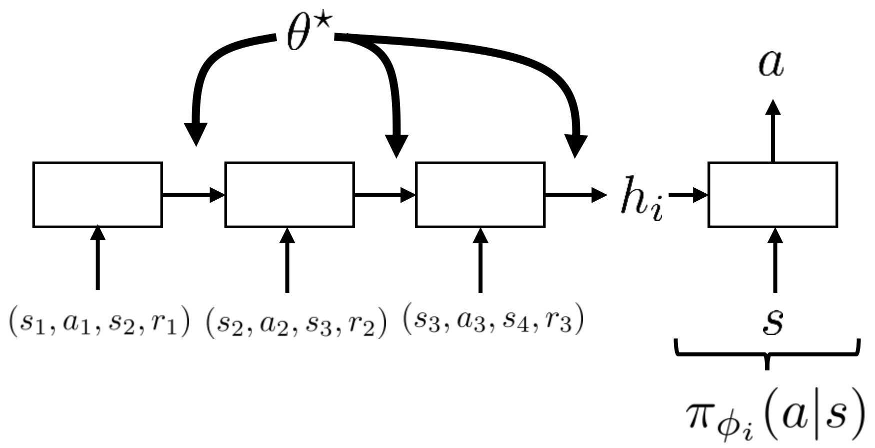
yes!

crucially, RNN hidden state is **not** reset between episodes!





Why recurrent policies *learn to explore*



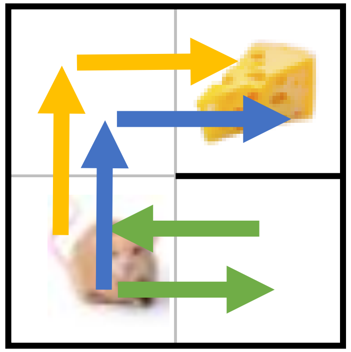
1. improve policy with experience from $\mathcal{M}_i$

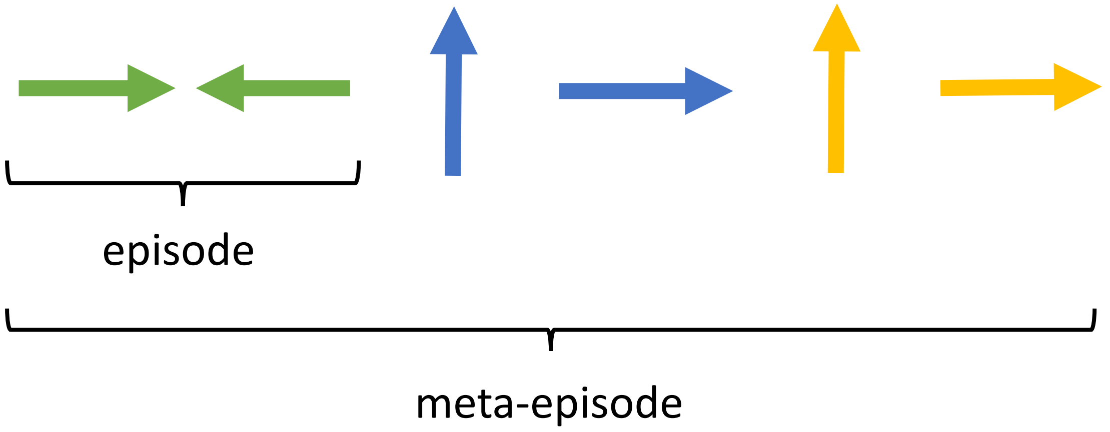
   $$\{(s_1, a_1, s_2, r_1), \ldots, (s_T, a_T, s_{T+1}, r_T)\}$$

2. (new in RL): choose how to interact, i.e. choose a_t

   meta-RL must also *choose* how to *explore*!

$$\theta^\star = \arg\max_\theta E_{\pi_\theta}\left[\sum_{t=0}^{T} r(s_t, a_t)\right]$$



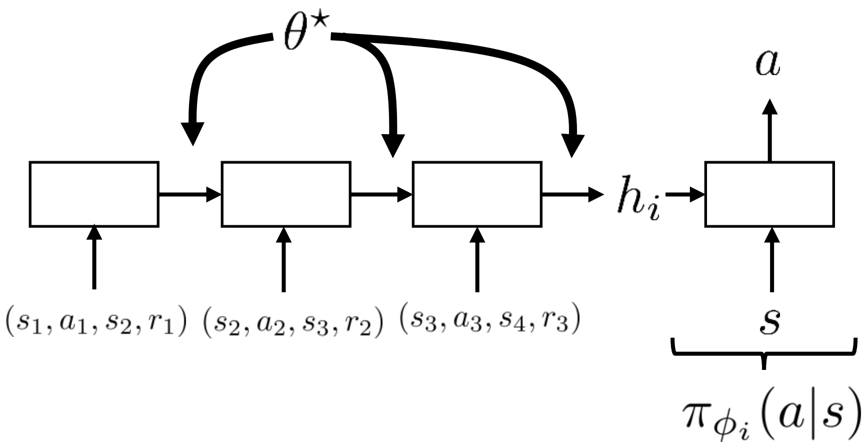


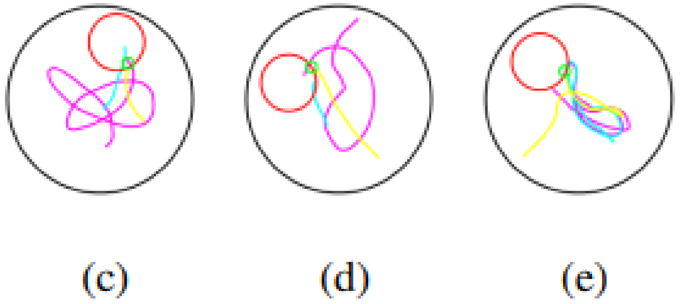
optimizing total reward over the entire **meta-**episode with RNN policy **automatically** learns to explore!

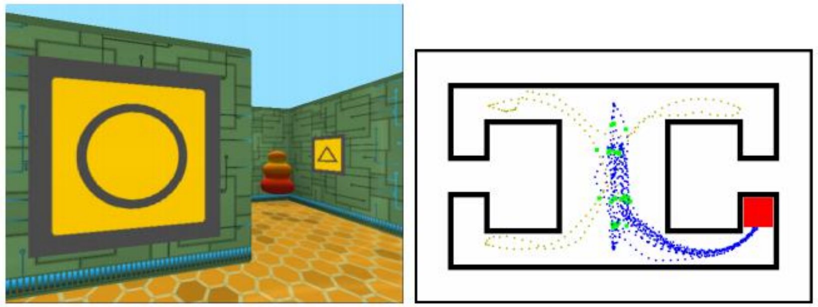
Meta-RL with recurrent policies

$$\theta^\star = \arg\max_\theta \sum_{i=1}^{n} E_{\pi_{\phi_i}(\tau)}[R(\tau)]$$

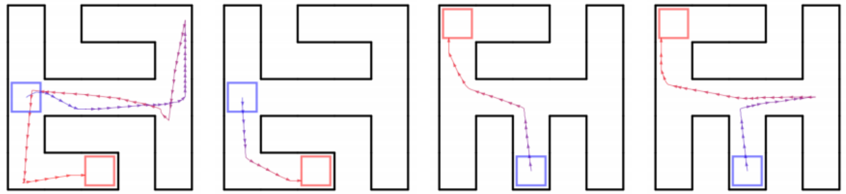
$$\text{where } \phi_i = f_\theta(\mathcal{M}_i)$$







(a) Labryinth I-maze (b) Illustrative Episode



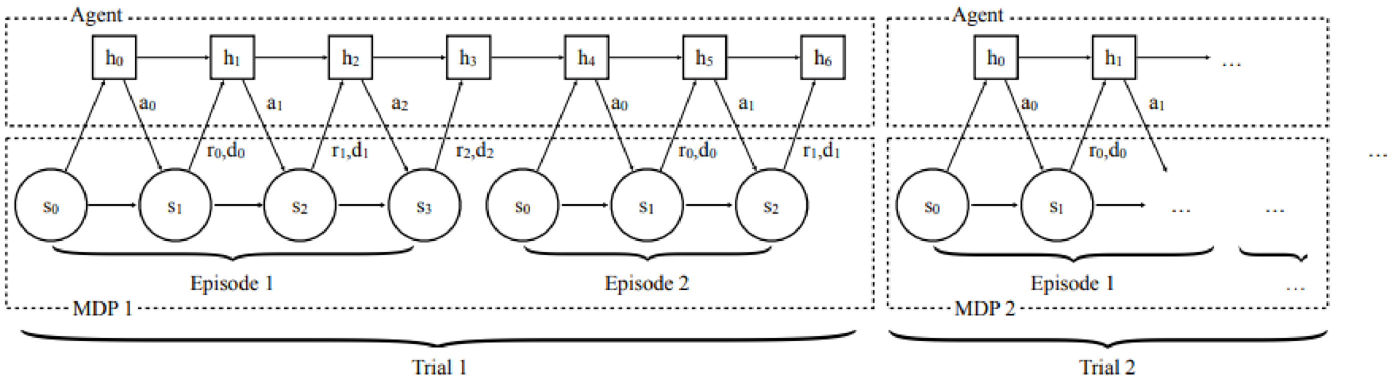
(a) Good behavior, 1st episode (b) Good behavior, 2nd episode (c) Bad behavior, 1st episode (d) Bad behavior, 2nd episode

Heess, Hunt, Lillicrap, Silver. **Memory-based control with recurrent neural networks.** 2015.

Wang, Kurth-Nelson, Tirumala, Soyer, Leibo, Munos, Blundell, Kumaran, Botvinick. **Learning to Reinforcement Learning.** 2016.

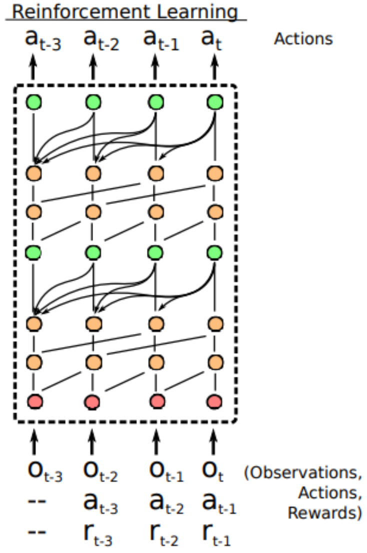
Duan, Schulman, Chen, Bartlett, Sutskever, Abbeel. **RL2: Fast Reinforcement Learning via Slow Reinforcement Learning.** 2016.

Architectures for meta-RL



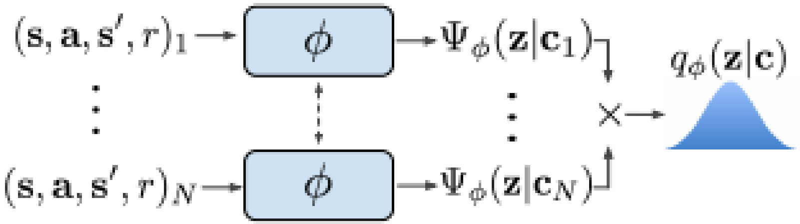
standard RNN (LSTM) architecture

Duan, Schulman, Chen, Bartlett, Sutskever, Abbeel. **RL2: Fast Reinforcement Learning via Slow Reinforcement Learning.** 2016.



attention + temporal convolution

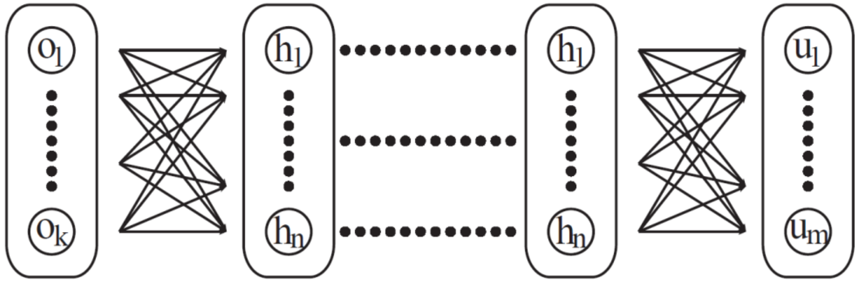
Mishra, Rohaninejad, Chen, Abbeel. **A Simple Neural Attentive Meta-Learner.**



parallel permutation-invariant context encoder

Rakelly*, Zhou*, Quillen, Finn, Levine. **Efficient Off-Policy Meta-Reinforcement learning via Probabilistic Context Variables.**

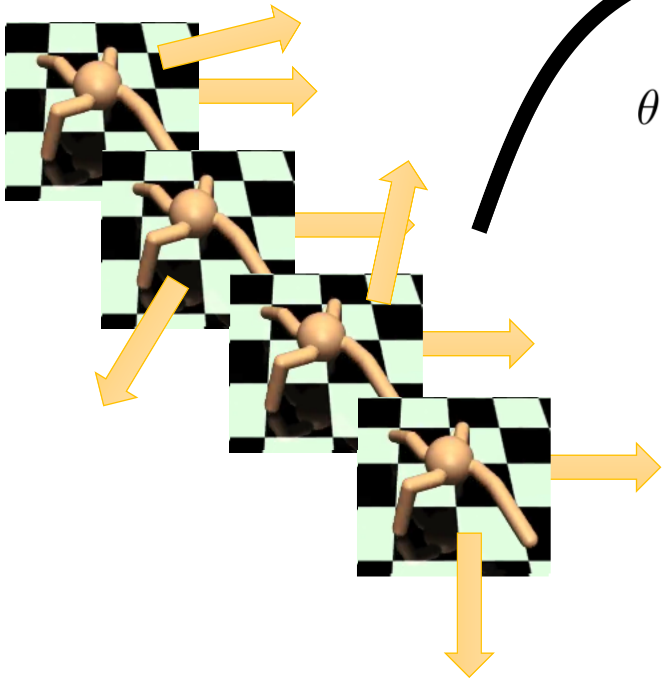
MAML for RL

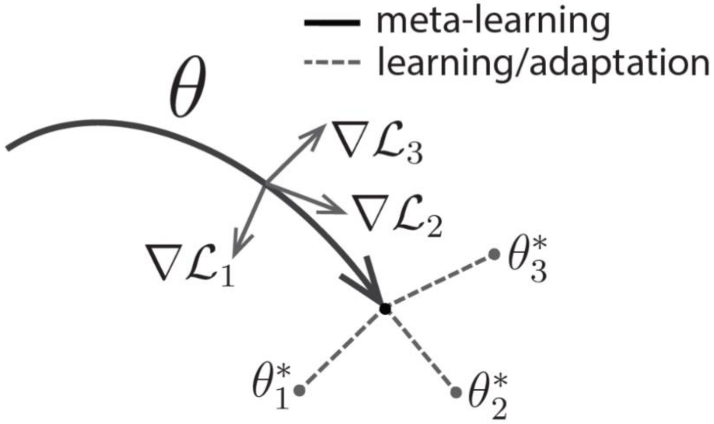


$$\theta$$

$$\cancel{\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)}$$

$$\theta \leftarrow \theta + \beta \sum_i \nabla_\theta J_i[\theta + \alpha \nabla_\theta J_i(\theta)]$$

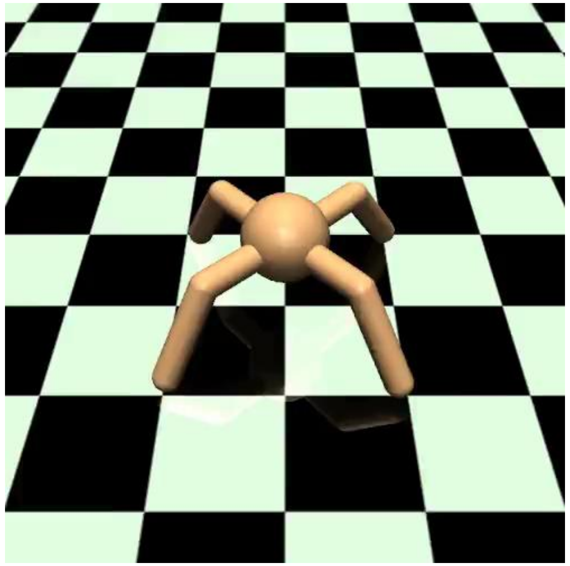


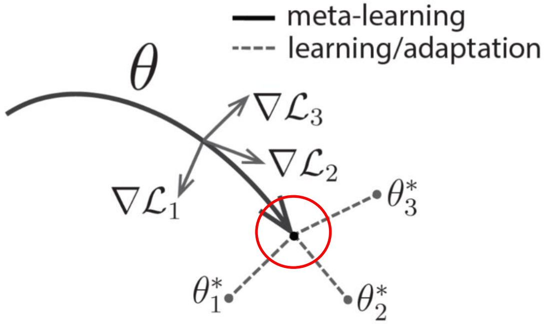


$$\theta'$$

MAML for RL videos

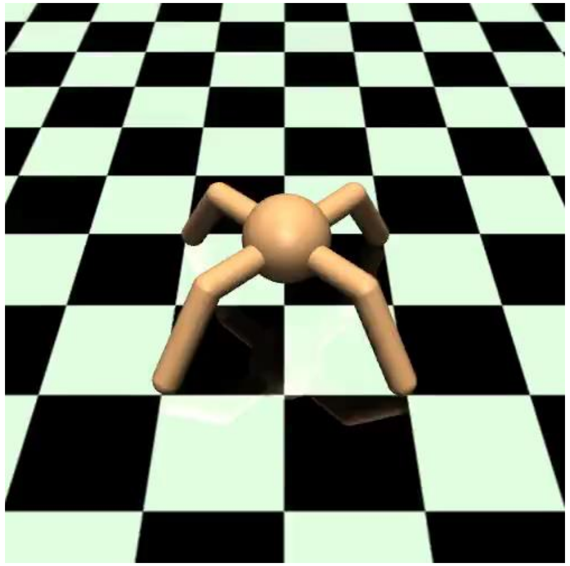
after MAML training

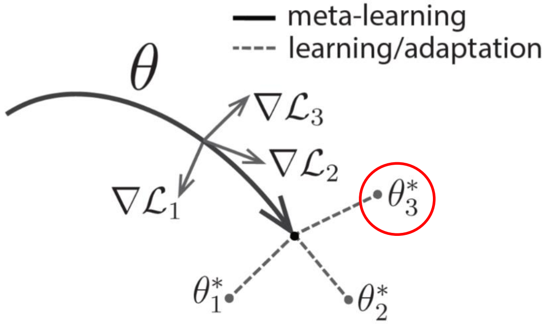




after 1 gradient step

(forward reward)





after 1 gradient step

(backward reward)