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

- Meta-training
- Meta-testing

Supervised learning: \( f(x) \rightarrow y \)
- Input: (e.g., image)
- Output: (e.g., label)

Supervised meta-learning: \( f(D^{tr}, x) \rightarrow y \)
- Training set

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

(few shot) training set

\((x_1, y_1), (x_2, y_2), (x_3, y_3)\)

\(x_{test}\)  
\(y_{test}\)  

Test input  
Test label
What is being “learned”?

(few shot) training set

\((x_1, y_1) \) \((x_2, y_2) \) \((x_3, y_3) \)

supervised meta-learning: \( f(D_{tr}, x) \rightarrow y \)

\( y_{test} \) test label

\( x_{test} \) test input

---

“Generic” learning:

\[ \theta^* = \arg \min_{\theta} \mathcal{L}(\theta, D_{tr}) \]

\[ = f_{\text{learn}}(D_{tr}) \]

“Generic” meta-learning:

\[ \theta^* = \arg \min_{\theta} \sum_{i=1}^{n} \mathcal{L}(\phi_i, D_{ts}^{i}) \]

where \( \phi_i = f_{\theta}(D_{tr}^{i}) \)
What is being “learned”?

“Generic” learning:

\[
\theta^* = \arg \min_{\theta} \mathcal{L}(\theta, \mathcal{D}^{tr})
\]
\[
= f_{\text{learn}}(\mathcal{D}^{tr})
\]

“Generic” meta-learning:

\[
\theta^* = \arg \min_{\theta} \sum_{i=1}^{n} \mathcal{L}(\phi_i, \mathcal{D}_i^{ts})
\]

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

\[\begin{align*}
\text{RNN hidden state} & \quad \text{meta-learned weights} \\
\phi_i &= [h_i, \theta_p] \\
p_{\phi_i}(y|x) & \quad x
\end{align*}\]
Meta-learning methods

**black-box meta-learning**

\[
\begin{align*}
(x_1, y_1) & \quad (x_2, y_2) & \quad (x_3, y_3) & \quad x_{test} \\
\end{align*}
\]

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

---

**non-parametric meta-learning**


---

**gradient-based meta-learning**


---


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!

\[
p_{\text{nearest}}(x_{k}^{\text{tr}} | x_{j}^{\text{ts}}) \propto \exp(\phi(x_{k}^{\text{tr}})^T \phi(x_{j}^{\text{ts}}))
\]
\[
p_{\theta}(y_{j}^{\text{ts}} | x_{j}^{\text{ts}}, D_{i}^{\text{tr}}) = \sum_{k: y_{k}^{\text{tr}} = y_{j}^{\text{ts}}} p_{\text{nearest}}(x_{k}^{\text{tr}} | x_{j}^{\text{ts}})
\]

all training points that have this label

\[
\theta^{*} = \text{arg min}_{\theta} \sum_{i=1}^{n} \mathcal{L}(f_{\theta}(D_{i}^{\text{tr}}), D_{i}^{\text{ts}}) = - \sum_{i=1}^{n} \sum_{j=1}^{m} \log p_{\theta}(y_{j}^{\text{ts}} | x_{j}^{\text{ts}}, D_{i}^{\text{tr}})
\]

learned (soft) nearest neighbor classifier
Matching networks

\[ p_\theta(y_j^{ts} | x_j^{ts}, D_i^{tr}) = \sum_{k: y_k^{tr} = y_j^{ts}} p_{\text{nearest}}(x_k^{tr} | x_j^{ts}) \]

\[ p_{\text{nearest}}(x_k^{tr} | x_j^{ts}) \propto \exp(g(x_k^{tr}, D_i^{tr})^T f(x_j^{ts}, D_i^{tr})) \]

- Different nets to embed \( x^{tr} \) and \( x^{ts} \)
- Both \( f \) and \( g \) conditioned on entire set \( D_i^{ts} \)

Prototypical networks

Two simple ideas compared to matching networks:

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

\[ p_\theta(y|x_j^{ts}, D_i^{tr}) \propto \exp(c_y f(x_j^{ts})) \]

\[ c_y = \frac{1}{N_y} \sum_{k:y_k^{tr}=y} g(x_k^{tr}) \]

2. Get rid of all the complex junk

- bidirectional LSTM embedding
- attentional LSTM embedding

is pretraining a type of meta-learning?
better features = faster learning of new task!
Meta-learning as an optimization problem

\[ \theta^* = \arg\min_\theta \sum_{i=1}^{n} \mathcal{L}(\phi_i, \mathcal{D}_{i}^{ts}) \]

where \( \phi_i = f_\theta(\mathcal{D}_{i}^{tr}) \)

what if \( f_\theta(\mathcal{D}_{i}^{tr}) \) is just a finetuning algorithm?

\[ f_\theta(\mathcal{D}_{i}^{tr}) = \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_{i}^{tr}) \]

(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!

MAML in pictures

\[ \theta \leftarrow \theta - \beta \sum_i \nabla_{\theta} \mathcal{L}(\theta - \alpha \nabla_{\theta} \mathcal{L}(\theta, D_{i}^{tr}), D_{i}^{ts}) \]

\[ \theta \leftarrow \theta - \alpha \nabla_{\theta} \mathcal{L}(\theta, D_{tr}) \]

\[ \theta \leftarrow \theta - \beta \sum_i \nabla_{\theta} \mathcal{L}(\theta, D_{i}^{tr}, D_{i}^{ts}) \]
What did we just do??

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

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

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

\[
f_{\text{MAML}}(D^{tr}, x) = f_{\theta'}(x)
\]

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

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

\[ (x_1, y_1) \quad (x_2, y_2) \quad (x_3, y_3) \]

\[ x_{\text{test}} \quad y_{\text{test}} \]

this implements the “learned learning algorithm”

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

MAML

\[ \theta \quad \nabla_\theta \mathcal{L} \quad \theta' \]

\[ (x_1, y_1) \quad (x_2, y_2) \quad (x_3, y_3) \]

- 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 $f(D_{\text{train}}, x)$

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

+ 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”)

non-parametric meta-learning


+ 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

gradient-based meta-learning


+ 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:

\[
\theta^* = \arg \min_{\theta} \mathcal{L}(\theta, \mathcal{D}^{tr}) \\
= f_{\text{learn}}(\mathcal{D}^{tr})
\]

Reinforcement learning:

\[
\theta^* = \arg \max_{\theta} E_{\pi_{\theta}(\tau)}[R(\tau)] \\
= f_{\text{RL}}(\mathcal{M}) \\
\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, r\}
\]

“Generic” meta-learning:

\[
\theta^* = \arg \min_{\theta} \sum_{i=1}^{n} \mathcal{L}(\phi_i, \mathcal{D}_i^{ts})
\]

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

Meta-reinforcement learning:

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

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

MDP

MDP for task \( i \)
The meta reinforcement learning problem

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

where \( \phi_i = f_\theta(M_i) \)

assumption: \( M_i \sim p(M) \)

meta test-time:

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

\( \{M_1, \ldots, M_n\} \)

\text{meta-training MDPs}
Meta-RL with recurrent policies

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

where \( \phi_i = f_\theta(M_i) \)

main question: how to implement \( f_\theta(M_i) \)?

what should \( f_\theta(M_i) \) do?

1. improve policy with experience from \( M_i \)
   \[ \{(s_1, a_1, s_2, r_1), \ldots, (s_T, a_T, s_{T+1}, r_T)\} \]

2. (new in RL): choose how to interact, i.e. choose \( a_t \)
   meta-RL must also choose how to explore!

pick \( a_t \sim \pi_\theta(a_t|s_t) \)

use \((s_t, a_t, s_{t+1}, r_t)\) to improve \( \pi_\theta \)

RNN hidden state

\( \phi_i = [h_i, \theta_\pi] \)

as before,
Meta-RL with recurrent policies

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

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

2. (new in RL): choose how to interact, i.e. choose $a_t$
   meta-RL must also choose how to explore!

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

optimizing total reward over the entire meta-episode with RNN policy automatically learns to explore!
Meta-RL with recurrent policies

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

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


Architectures for meta-RL

standard RNN (LSTM) architecture


attention + temporal convolution


parallel permutation-invariant context encoder

MAML for RL

\[ \theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta) \]

\[ \theta \leftarrow \theta + \beta \sum_{i} \nabla_{\theta} J_{i}[\theta + \alpha \nabla_{\theta} J_{i}(\theta)] \]
MAML for RL videos

after MAML training

after 1 gradient step
(forward reward)

after 1 gradient step
(backward reward)