Meta Learning

Miloš Jordanski, PhD student at Faculty of Mathematics

Meta Learning

- Learning quickly:
  - Recognizing objects from only a few examples
  - New skills after just minutes of experience
- Integrate prior experience with a small amount of new information
- Learning to learn
- Applications:
  - Supervised Regression and Classification
  - Reinforcement Learning
  - Unsupervised Learning

How to learn quickly?

- Transfer Learning: train on one task, transfer to a new task:
  - Just try it and hope for the best
  - Diversity: the more varies the training, the more likely transfer is to succeed
  - Fine tune on a new task
  - New architectures suitable for transfer: Progressive Networks
- Multi-task transfer: train on a many tasks, transfer to a new task
  - Requires variety!
- Meta-Learning: learn how to learn many tasks:
  - RNN based meta-learning
  - **Gradient based meta-learning**

Meta-Learning Problem Set-Up

- Model $f: x \rightarrow a$
- Task $T = \{L(x_1, a_1, \dots x_H, a_H), q(x_1), q(x_{t+1}|x_t, a_t)\}$
  - $L(x_1, a_1, \dots x_H, a_H)$ – loss function
  - $q(x_1)$ - distribution over initial observations
  - $q(x_{t+1}|x_t, a_t)$ – transition distribution
  - H – episode length
- $p(T)$ – distribution over tasks

K-shot learning

- Goal: Train a model to learn a new task $T_i \sim p(T)$ from only K samples drawn from q_i and feedback L_{T_i} generated by T_i.
- Meta-training:
  - task $T_j \sim p(T)$
  - Model f is trained with K samples and feedback from the corresponding loss L_{T_j}
  - Test a model f on new samples from T_j
  - Model f is improved by considering how the test error on new data from q_j changes with respect to the parameters.
  - Test error on sampled tasks T_j serves as the training error of the meta-learning process

Meta-Learning Training

- Model represented by a parametrized function f_θ
- When adapting to a new task $T_i \sim p(T)$:

$$\theta_i' = \theta - \alpha \nabla_\theta L_{T_i}(f_\theta)$$

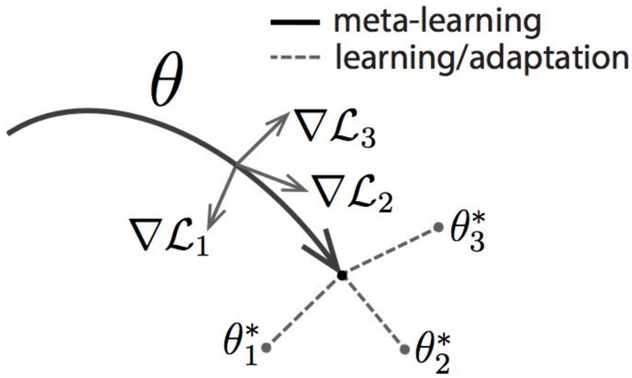
- The model parameters are trained by optimizing for the performance of $f_{\theta_i'}$ with respect to θ across tasks sample $p(T)$:

$$\min_\theta \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta_i'}) = \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta - \alpha \nabla_\theta L_{T_i}(f_\theta)})$$

Meta-Learning Training

- Meta-optimization via stochastic gradient descent (SGD):

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta_i'})$$



$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta - \alpha \nabla_\theta L_{T_i}(f_\theta)})$$

- Meta-gradient update involves a gradient through a gradient
- Computing Hessian is computationally inefficient
- First –order approximation

Model-Agnostic Meta-Learning

- Input: $p(T)$ distribution over tasks; α, β – step size hyperparameters
- Randomly initialize θ
- While not done do:
  - Sample batch of tasks $T_i \sim p(T)$
  - For all T_i do:
    - Evaluate $\nabla_\theta L_{T_i}(f_\theta)$ with respect to K examples
    - Compute adapted parameter with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta L_{T_i}(f_\theta)$
  - End for
  - Update: $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta'})$

Supervised Regression and Classification

- Task T_i generate K i.i.d observations x from q_i, H=1
- Regression – MSE:

$$L_{T_i}(f_\theta) = \sum_{x^j, y^j \sim T_i} \left\| f_\theta\left(x^j\right) - y^j \right\|_2^2$$

- Classification – cross entropy:

$$L_{T_i}(f_\theta) = \sum_{x^j, y^j \sim T_i} y^j log f_\theta\left(x^j\right) + (1 - y^j)(1 - log f_\theta\left(x^j\right))$$

MAML For Few-Shot Supervised Learning

- Input: $p(T)$ distribution over tasks; α, β – step size hyperparameters
- Randomly initialize θ
- While not done do:
  - Sample batch of tasks $T_i \sim p(T)$
  - For all T_i do:
    - Sample K datapoints $D = \{x^j, y^j\}$ from T_i
    - Evaluate $\nabla_\theta L_{T_i}(f_\theta)$ using D and L_{T_i}
    - Compute adapted parameter with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta L_{T_i}(f_\theta)$
    - Sample datapoints $D_i' = \{x^j, y^j\}$ from T_i for the meta-update
  - End for
  - Update: $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta'})$ using each D_i' and L_{T_i}

Reinforcement Learning

- Each RL task T_i contains an initial state distribution $q_i(x_1)$, transition distribution $q_i(x_{t+1}|x_t, a_t)$ and loss function L_{Ti}:

$$L_{T_i}(f_\theta) = -E_{x_t, a_t \sim f_\theta, q_{T_i}}\left[\sum_{t=1}^{H} R_t(x_t, a_t)\right]$$

- In K-shopt RL, K rollouts from f_θ and task T_i may be used for adaptation on a new task T_i.

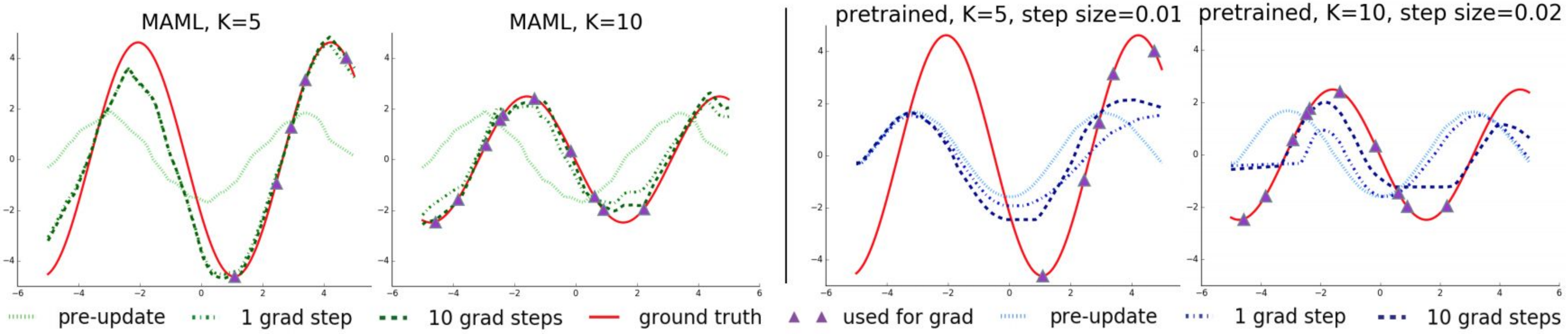
MAML for Reinforcement Learning

- Input: $p(T)$ distribution over tasks; α, β – step size hyperparameters
- Randomly initialize θ
- While not done do:
  - Sample batch of tasks $T_i \sim p(T)$
  - For all T_i do:
    - Sample K trajectories $D = \{(x_1, a_1 \ldots, x_H, a_H)\}$ using f_θ in T_i
    - Evaluate $\nabla_\theta L_{T_i}(f_\theta)$ using D and L_{T_i}
    - Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta L_{T_i}(f_\theta)$
    - Sample trajectories $D_i' = \{(x_1, a_1 \ldots, x_H, a_H)\}$ using f_θ in T_i
  - End for
  - Update: $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta'})$ using each D_i' and L_{T_i}

Results - Regression

- Sine wave with amplitude varies within [0.1, 5.0] and phase varies within [0, π]
- x uniformly from [-5, 5]
- Loss: mean-squared error
- Neural Network model with 2 hidden layers of size 40 with ReLU
- One update with K sample, α = 0.01
- Baselines:
  - pretraining on all of the tasks and fine-tuning with gradient descent on the K provided points
  - true amplitude and phase

Results - Regression



Results - Classification

- Omniglot dataset: 20 instances of 1623 characters from 50 different alphabets. Each instance was drawn by a different person
- MiniImagenet dataset: involves 64 training classes, 12 validation classes, and 24 test classes
- 4 modules with a 3×3 convolutions and 64 filters, followed by batch normalization, a ReLU nonlinearity, and 2×2 max-pooling
- For Omniglot, strided convolutions instead of max-pooling.

Omniglot results

Omniglot (Lake et al., 2011)	5-way Accuracy 1-shot	5-way Accuracy 5-shot	20-way Accuracy 1-shot	20-way Accuracy 5-shot
MANN, no conv (Santoro et al., 2016)	82.8%	94.9%	–	–
MAML, no conv (ours)	$\mathbf{89.7 \pm 1.1\%}$	$\mathbf{97.5 \pm 0.6\%}$	–	–
Siamese nets (Koch, 2015)	97.3%	98.4%	88.2%	97.0%
matching nets (Vinyals et al., 2016)	98.1%	98.9%	93.8%	98.5%
neural statistician (Edwards & Storkey, 2017)	98.1%	99.5%	93.2%	98.1%
memory mod. (Kaiser et al., 2017)	98.4%	99.6%	95.0%	98.6%
MAML (ours)	$\mathbf{98.7 \pm 0.4\%}$	$\mathbf{99.9 \pm 0.1\%}$	$\mathbf{95.8 \pm 0.3\%}$	$\mathbf{98.9 \pm 0.2\%}$

MiniImage results

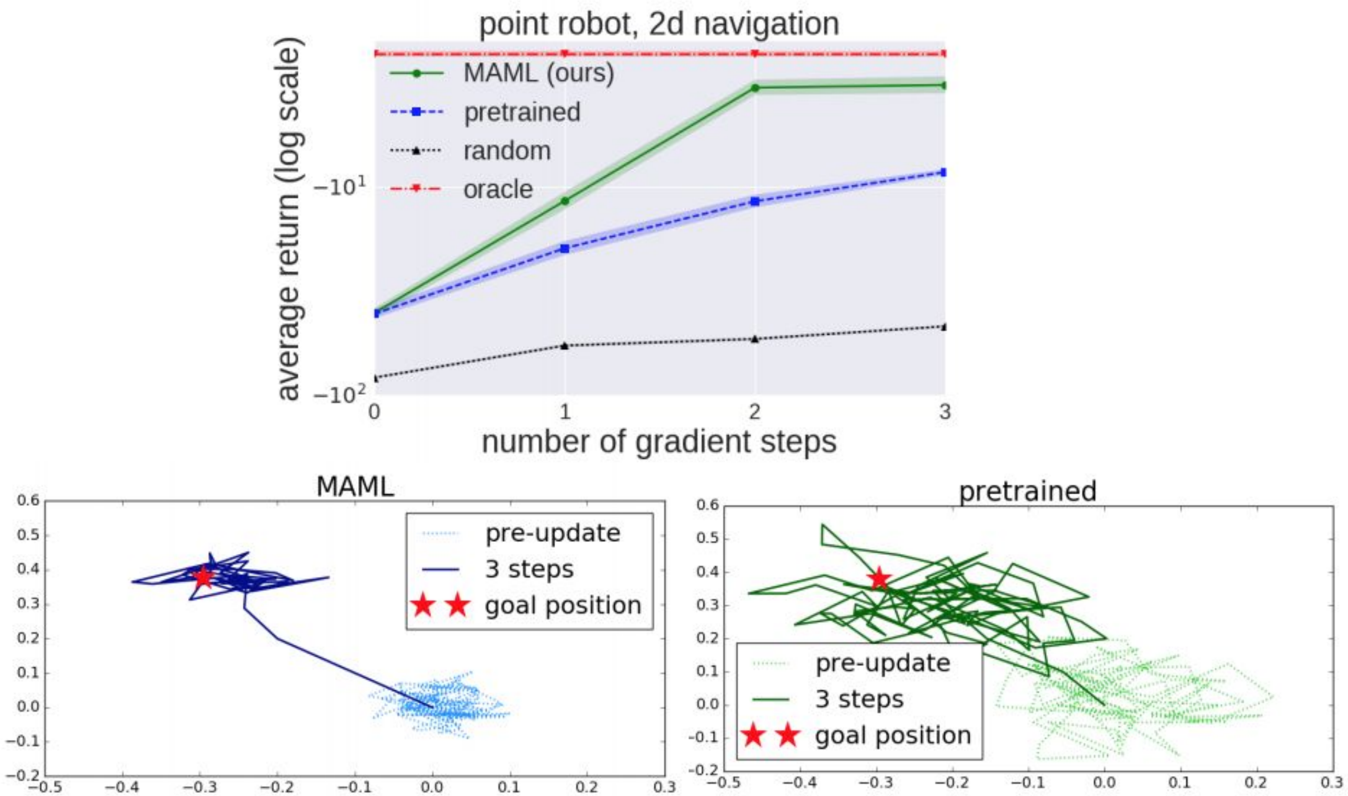
MiniImagenet (Ravi & Larochelle, 2017)	5-way Accuracy 1-shot	5-way Accuracy 5-shot
fine-tuning baseline	$28.86 \pm 0.54\%$	$49.79 \pm 0.79\%$
nearest neighbor baseline	$41.08 \pm 0.70\%$	$51.04 \pm 0.65\%$
matching nets (Vinyals et al., 2016)	$43.56 \pm 0.84\%$	$55.31 \pm 0.73\%$
meta-learner LSTM (Ravi & Larochelle, 2017)	$43.44 \pm 0.77\%$	$60.60 \pm 0.71\%$
MAML, first order approx. (ours)	$\mathbf{48.07 \pm 1.75\%}$	$\mathbf{63.15 \pm 0.91\%}$
MAML (ours)	$\mathbf{48.70 \pm 1.84\%}$	$\mathbf{63.11 \pm 0.92\%}$

Results - Reinforcement Learning

- Neural network policy with two hidden layers of size 100, with ReLU nonlinearities
- Planar cheetah and a 3D quadruped
- Baselines:
  - pretraining one policy on all of the tasks and then fine-tuning
  - training a policy from randomly initialized weights
  - an oracle policy

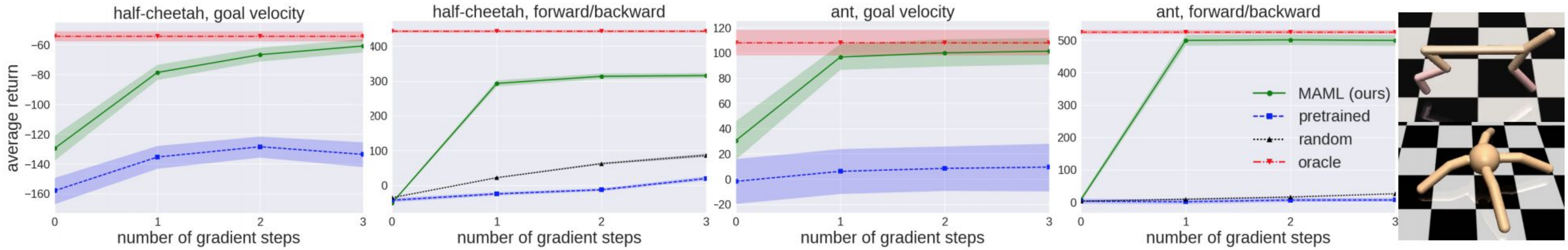
Results - Reinforcement Learning

- 2D Navigation



Results - Reinforcement Learning

- Locomotion - high-dimensional locomotion tasks with the MuJoCo simulator
  - planar cheetah
  - a 3D quadruped



THANKS!