A collection of notes

Jason Krone
Columbia University

March 25, 2018

Abstract

This is a collection of notes on machine intelligence for personal reference.

Contents

1 Model Agnostic Meta Learning by Finn et al. 2017

1.1 Summary:

Goal: train a model on a set of learning tasks such that it can solve novel learning tasks using a small number of training samples.

Meta-learning should be general to the task and model.

MAML does not introduce more parameters or place constraints on model architecture.

MAML increases the sensitivity loss functions to new tasks i.e. small parameter changes cause large loss improvements.

1.2 Model:

Uses entire tasks as training examples.

Let a model $f_\theta : x \to a$ where x is an observation and a is an output.

A task T is defined as $T = \{L(x_1, a_1, \dots, x_H, a_H), q(x_1), q(x_{t+1} \mid x_t, a_t), H\}$ where L is a loss function, $q(x_1)$ is a distribution over initial observations $q(x_1)$, $q(x_{t+1} \mid x_t, at)$ is a transition distribution and H is the episode length.

Note: this formulation is like a reinforcement learning problem expect we assume to have the transition function?

Let $p(T)$ be a a distribution over tasks we want the model to adapt to.

1.3 Training:

1. A task $T_i \sim p(T)$
2. the model f_θ is trained with K samples using feedback from L_{T_i}
3. f is tested on new samples from T_i
4. f is improved by considering how the test error on new data from q_i changes w.r.t the parameters

In a sense the test error on sampled tasks is used as the training error for meta-learning.

Tasks used in meta-testing are held out during meta-training to remain "unbiased?"

```
**while** not done **do**
    Sample batch of tasks $T_I \sim p(T)$
    **for all** $T_i$ **do**
```

Evaluate $\Delta_\theta L_{T_i}(f_\theta)$ with respect to K examples.
Compute adapted parameters with GD: $\theta_i^{'} = \theta - \alpha \Delta_\theta L_{T_i}(f_\theta)$

The meta learning objective is as follows:

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

Meta-optimization is done as follows:

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

For reinforcement learning focused problems the loss is of the form:

$L_{T_i}(f_\phi) = \mathbb{E}_{x_t, a_t \sim f_\phi, q_{T_i}} \left[\sum_{t=0}^{H} R_i(x_t, a_t) \right]$

This is used in the context of policy gradient methods, which are on policy. Since they are on policy, each additional gradient step requires sampling from the current policy $f_{\theta_i^{'}}$.

1.4 Experiments:

Comparisons are made between MAML, pre-training on all tasks, and an oracle (best possible performance).

1. Tasks: Regression from input to output of sine function where amplitude and phase of sine vary for tasks. Result: Very low MSE for MAML and very high MSE for pre-trained model.
2. Tasks: Few shot classification on Omniglot and MiniImagenet. Result: near perfect accuracy on Omniglot and best MiniImagenet accuracy by 3-5 percent.
3. Tasks: Continuous control environments for reinforcement learning where goal is to run in a particular direction or at a particular velocity. Result: outperforms pre-training by a large margin and almost reaches oracle performance.

1.5 My Conclusions: (concerns, follow up, etc)

What are important applications of MAML:

- Language models? To quickly adapt to different language pairs in translation for instance ?
- Could this be applied Bayesian models also?
- Speech recognition with different accents or languages ?

- Hierarchical reinforcement learning ? Adaption to multiple tasks and come up with a general hierarchical structure?

Concerns:

- Can this be run in real time for robotics application ?
- How much training data do you need for this method? How many tasks / samples per task?
- Is this still helpful when you have a lot of data? For example, as a pre-training method for Imagenet.

Follow up:

- How does the model architecture effect the performance of MAML?
- Is it helpful if the model "remembers" things across tasks?
- How well does this work with recurrent / memory networks?

2 One-Shot Visual Imitation Learning via Meta-Learning by Chelsea Finn et al. 2017

2.1 Summary:

Core Question: How can we leverage information from previous skills to quickly learning new behaviors?

Work forcuses on learning to imiation learn from one demonstration where only video of the demonstration is available.

2.2 Model:

The policy we are trying to learn: $\pi : o \rightarrow \hat{a}$

Each imitation task $T_i = \{\tau = \{o_1, a_1, \ldots, o_T, a_T\} \sim \pi_i^*, L(a_{1:T}, \hat{a}_{1:T}), T\}$ data τ is generated by an expert policy π_i^* and loss function L.

MAML is extended to imitation learning as follows:

Let the demonstration trajectory (images and motor torques) be $\tau := \{o_1, a_1, \ldots, o_T, a_T\}$.

They use a MSE loss function on the policy parameters ϕ:

$$L_{T_i}(f_\phi) = \sum_{\tau^{(j)} \sim T_i} \sum_t \left\| f_\phi(o_t^{(j)}) - a_t^{(j)} \right\|_2^2$$

2.2.1 Two head architecture:

The architecture uses two heads:

1. pre-update head whose parameters are not used in post-update policy. (think this is θ)
2. post-update head which is not updated using the demonstration (think this is θ')

2.3 Training:

Meta-training loop:

1. Sample a batch of tasks and two demonstrations per task
2. Using one of the demonstrations $\tau^{(1)} \sim T_i$ compute $\theta_i^{'}$ for each task T_i
3. Using the second demonstration $\tau^{(2)} \sim T_i$ compute the gradient of the meta-objective

4. Update θ using the gradient of the meta objective $\Delta_\theta \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta'_i})$ with the MSE loss given above

2.4 Experiments:

1. Simulated reaching
2. Simulated pushing
3. Real-World Placing

2.5 My Conclusions: (concerns, follow up, etc)

Follow up:

- How could you adversarially attack MAML? Are there any good opertunities here?
- Could you somehow use MAML objective as a fitness function in evolutionary algos?

3 Reptile: a Scalable Meta Learning Algorithm by Alex Nichol and John Schulman, 2018

3.1 Motivation

Non-gradient based approaches (e.g. using LSTM to predict weights) sometimes suffer from lack of generalization (overfitting). While MAML achieves better generalization given it relies on SGD, it requires differentiation within the optimization process, which is costly if we take a large number of gradient steps at test time.

3.2 Algorithm

Let $SGD(L, \phi, k)$ denote the function that takes k steps of SGD on loss L starting with parameters ϕ and returns the final parameter vector.

Initalize model parameters ϕ For $i = 1, 2, \ldots$:

- Sample task τ from task distribution D_τ, corresponding to loss L_τ on weight vectors W
- Compute $W = SGD(L_\tau, \phi, k)$
- $\phi = \phi + \epsilon(W - \phi)$

There is also a batched version of Reptile that works as follows:

Initialize ϕ For $i = 1, 2, \ldots$

- Sample n tasks $\tau_1, \tau_2, \ldots, \tau_n$ from D_τ
- For $j = 1, 2, \ldots, n$
  - $W_j = SGD(L_j, \tau_j, k)$
- $\phi = \phi + \epsilon\frac{1}{k}(\sum_{j=1}^{n} W_j - \phi)$

Note that when $k = 1$ then Reptile is the same as updating ϕ by $\Delta_\phi \mathbb{E}_\tau(L_\tau(f_\phi))$. However when $k > 1$ then the Reptile update is different than taking the derivative of the expected loss.

3.3 Experiments

3.3.1 Sign Wave Regression:

The task is to predict y value given x for a sign wave τ_i with certain amplitude and phase. Where $L_\tau = \int_{-5}^{5} dx \mid f(x) - f_\tau(x) \mid^2$, which is evaluated using 50 equally spaced points. Training on the expected loss $\mathbb{E}_\tau[L_\tau]$ would not work because this loss is minimized by $f(x) = 0, \forall x$. Therefore, it is promising that after training reptile converges instead to a sign wave that is close to the function f_τ.

3.4 Why it works

4 Hindsight Experience Replay by Marcin Andrychowicz et al. 2018

4.1 Summary:

Hindsight Experience Replay (HER) is a technique related to reward shaping, which "re-plays" a trajectory with a set of "goals" that may differ from the actual goal in order to improve learning efficiency. A design choice that must be made when using her is what additional goals to use.

4.2 Model:

In this framework, each goal g corresponds to a reward function $r_g : S \times A \to R$. In addition, the agent receives as input the goal at each time-step. HER parameterizes the action-value function Q with a goal g i.e. $Q^{\pi}(s_t, a_t, g) = \mathbb{E}_{\pi, p(s_0)}[G_t \mid s_t, a_t, g]$.

4.3 Training:

HER can be trained using any off-policy reinforcement learning algorithm such as DQN, DDPG, NAF, and SDQN. Training proceeds as follows for each episode:

- Sample a goal g, initial state s_0, and roll out policy
- Store trajectory transitions $(s_t \mid g, a_t, r_t, s_{t+1} \mid g)$ with original goal g and other goals g' in replay buffer
- Sample a mini batch B from replay buffer
- Perform one step of optimization using algorithm A and mini batch B

4.4 Experiments:

This paper evaluated the model on the following three tasks using the Mujoco simulator with a 7-DOF Fetch robotics arm:

1. Pushing
2. Sliding
3. Pick-and-place

4.5 My Conclusions: (concerns, follow up, etc)

- The actions used in the paper are 4-dimensional and specify the desired position of the gripper at the next time step. I am unsure if the gripper will deterministically move to this position at the next time step or not. If so, this seems a bit simplistic. Can the algorithm accommodate higher dimensional action spaces or must you relay on a method to get to a specific position ?
- What does the "future" version (where g' is from the same episode being replayed and is a state observed after it) of the algorithm really do? Does it refer to the current state?
- How could this interact with imitation learning? Could this augment imitation learning somehow?
- When else is it useful to parameterize the Q function with goals?

5 A Simple Neural Attentive Meta-Learner by Nikhil Mishra et al. 2018

5.1 Summary:

Claim: most meta-learning approaches are hand crafted to a degree, either in terms of the architecture or algorithm. To solve this issue the authors propose a Simple Neural Attentive Meta-Learner (SNAIL), which uses temporal convolutions and soft attention. Methods that are less hand designed are likely to perform better because the optimal strategy will very over task distributions and therefore you must be able to adapt the strategy.

5.2 Notation:

Each task T_i is defined by inputs x_t, outputs a_t, a loss function $L_i(x_t, a_t)$, a transition distribution $(P_i(x_t \mid x_{t-1}, a_{t-1})$, and an episode length H_i. The aim is to minimize the expected loss over tasks with respect to θ (the model parameters) i.e.

$$\min_{\theta} \mathbb{E}_{T_i \sim T} \left[\sum_{t=0}^{H_i} L_i(x_t, a_t) \right]$$

where

$$x_t \sim P_i(x_t \mid x_{t-1}, a_{t-1}), a_t \sim \pi(a_t \mid x_1, \ldots, x_t; \theta)$$

The meta-learner is trained on a mini-batch of tasks sampled from T, and it is evaluated on a held out set of tasks $\hat{T}$ that is similar to the training tasks distribution.

5.3 Model:

- Temporal Convolutions: allow for aggregation of contextual information from past experience. In this context causal means **only influenced by past timesteps and not future ones**. A current deficiency with temporal convolutions is that the "to scale to long sequences, the dilation rates generally increase exponentially, so the required layers scale logarithmically with the sequence length ... their bounded capacity and positional dependence can be undesirable."
- Soft Attention: allow it to focus on pieces of information within that context. However, soft attention suffers from a lack of positional dependence, which is an even larger problem for reinforcement learning where in observations, actions, and rewards are sequential.

5.4 Training:

5.5 Experiments:

5.6 My Conclusions: (concerns, follow up, etc)

- Not sure I agree with their claim that the current meta-learning approaches are hand-designed. TODO: think about this in context of previous work.
- TODO: read about temporal convolutions

6 Dynamic Routing Between Capsules by Sabour et al. 2017

6.1 Summary:

Capsules aim to solve to problem of routing information in neural networks, currently done poorly by max-pooling, which throws away most information.

6.2 Model:

The model is a parse tree that is carved out of a fixed multi-layer neural network, which is divided in to capsules. Each capsule represents propoerties of the object, such as class, position, albedo, texture, etc, seen in a single fixation. Specifically, the length of the vector (norm) indicates the existance of the entity.

All but the last layer of capsules are convolutional - to ensure that knowledge about good weight values is shared between positions. Low level capsules "place coded" information i.e. the capsule that fires indicates properities and higher level capsules are "rate coded" i.e. the scalar values indicate higher level properties.

The following "squashing" function is used to determine the "length" of a vector. Note that we want short vectors to have length near 0 and long vectors to have length near 1.

$$v_j = \frac{\| s_j \|^2}{1+ \| s_j \|^2} \frac{s_j}{\| s_j \|}$$

here v_j is the vector output of capsule j and s_j is its total input.

For every layer after the first, the input s_j is a weighted sum of "prediction vectors" from the capsules in the previous layers. The prediction vectors are obtained by multiplying the output u_i of a capsule by W_{ij}. Namely, "prediction vector" $\hat{u}_{j|i} = W_{ij} u_i$. These "prediction vectors" are weighted by coupling coefficients c_{ij} that are determined by the dynamic routing process to create the input to capsule j i.e. $s_j = \sum_i c_{ij} \hat{u}_{j|i}$

6.3 Training:

6.4 Experiments:

6.5 My Conclusions: (concerns, follow up, etc)

7 PAPER TITLE by AUTHOR et al. YEAR

7.1 Summary:

7.2 Model:

7.3 Training:

7.4 Experiments:

7.5 My Conclusions: (concerns, follow up, etc)

References