Markov Decision Process[5]

Markov Property

The future is independent of the past given the present: Once the current state in known, the history of information encountered so far may be thrown away, and that state is a sufficient statistic that gives us the same characterization of the future as if we have all the history.

Markov Decision Process

A Markov decision process is a 4-tuple (S, A, P_a, R_a), where

1. S is a finite set of states,
2. A is a finite set of actions (alternatively, A_s is the finite set of actions available from state s),
3. $P_a(s, s') = Pr(s_{t+1} = s'|s_t = s, a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time $t+1$,
4. $R_a(s, s')$ is the immediate reward(or expected immediate reward) received after transitioning from state s to state s', due to action a

The core task of Markov Decision Process(MDP) is to find a 'policy'(π) for the decision maker. The goal is to choose a policy π that will maximize some cumulative function of the random rewards, typically the expected discounted sum over a potentially infinite horizon:

$$G_t = \sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}), \text{ where } a_t \text{ is given by the policy and } \gamma \text{ is discount factor}$$

Bellman Equation

Value Function[4]: For any policy π, we can define the Value Function

$$V^{\pi}(s) = E\Bigg[\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1})|s_0 = s, \pi\Bigg]$$

$V^{\pi}(s)$ is the expected payoff starting in state s and following policy π.

Bellman Equation[4]:Gives a recursive definition of the above Value Function

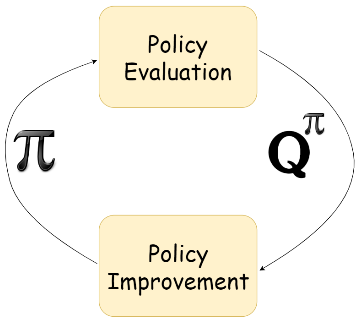
$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{\pi(s)}(s, s') \times V^{\pi}(s')$$

where $R(a)$ is the reward the agent receives when it lands at state s.

Solving MDP

The solution for an MDP is a policy which describes the best action for each state in the MDP, known as the optimal policy.
General Strategy:

- *Dynamic Programming*
  1. Given a fixed policy(π) estimate the value of each state
  2. Given a fixed value function, improve the policy(π)



Loop Till Convergence

Methods[**reinforcementLearningTextbook**]:

- *Policy Iteration*
- *Value Iteration*

Exploration/Exploitation trade off

- *Exploration*: It is finding more information about the environment.
- *Explotation*: It is exploiting known information to maximise the reward

JACKPOT
EXPLOITATION
Playing the machine that (currently) pays out the most.
EXPLORATION
Playing the other machines to see if any pay out more.

[3]

Learning to Learn by Gradient Descent by Gradient Descent[1]

Problem Statement

How the design of an optimization algorithm can be cast as a learning problem, allowing the algorithm to learn to exploit structure in the problems of interest in an automatic way?

Mathematical Formulation

We have to optimize the objective function $f(\theta)$ defined over some domain $\theta \in \Theta$. We define an optimizer g, specified by its own parameters ϕ. This results in the update of the optimizee f of the form

$$\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t, \phi))$$

The update rule g is modeled using RNN.
The final optimizee parameters are defined as a function of ϕ and f, ie $\theta^*(f, \phi)$
Now given a distribution of function f we can write the expected loss as

$$L(\phi) = E_f\left[f\big(\theta^*(f, \phi)\big)\right]$$

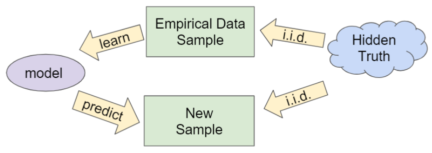
Consider a RNN m, parameterized by ϕ, whose state is denoted by h_t and it outputs the update steps g_t. While the objective function above depends only on the final parameter value, for training the optimizer it is convenient to use an objective function that depends on the entire trajectory of optimization(for some horizon T),

$$\begin{aligned}
L(\phi) &= E_f\left[\sum_{t=1}^{T} w_t f(\theta_t)\right] \text{ where} \\
\theta_{t+1} &= \theta_t + g_t \\
\begin{bmatrix} g_t & h_{t+1} \end{bmatrix} &= m(\nabla_t, h_t, \phi) \text{ and} \\
\nabla_t &= \nabla_\theta f(\theta_t)
\end{aligned}$$

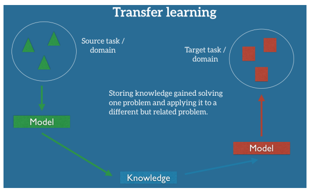
We can minimize $L(\phi)$ using gradient descent on ϕ.

ML Approach Used in the Paper

- ***Casting the problem of transfer learning as generalization***

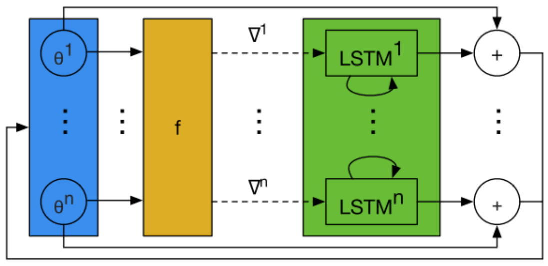


In ordinary statistical learning we have a particular function of interest, which is constrained through a data set. In choosing a model we specify our inductive biases as to how the function should behave at points not observed yet. In this respect, **generalization** refers to the capacity to make predictions about the behaviour of the function at novel points.


Transfer learning
Source task / domain
Target task / domain
Storing knowledge gained solving one problem and applying it to a different but related problem.
Model
Model
Knowledge

In the problem the examples are themselves problem instances, which means generalization corresponds to the ability to transfer knowledge between different problems. This reuse of problem structure is called as **transfer learning**.
By taking the meta-learning perspective, we can cast the problem of **transfer learning** as one the **generalization**.

- ***Coordinatewise LSTM***



Directly applying RNN will lead to the optimization of at least tens of thousands of parameters. This is not feasible as it would require huge hidden state and an enormous number of parameters. To avoid this difficulty we will use an optimizer m which operates coordinatewise on the parameters of the objective function. This network architecture allows us to use a very small network that only looks at a single coordinate to define the optimizer and share optimizer parameters across different parameters of the optimizee.

Upside: It has the effect of making the optimizer invariant to the order of parameters in the network, since the same update rule is used independently on each coordinate.

The update rule for each coordinate is implemented using *LSTM*, using the now-standard forget gate architecture.

Learning to Learn without Gradient Descent by Gradient Descent[2]

Problem Statement

Learning to learn can be used to learn both models and algorithms. In this paper, the goal of meta-learning is to produce an algorithm for global black-box optimization. How to find a global minimizer of an black-box loss function f?

Mathematical Formulation

Consider a black-box loss function f and X as its domain. The function f is not available to the learner in simple closed form at the test time, but can evaluated at query point x in the domain. In other words, we can only observe the function f through unbiased noisy point-wise observations y, ie

$$f(x) = E\big[y|f(x)\big]$$

The optimization algorithm gives the following loop:

1. Given the current state of knowledge h_t propose a query point x_t
2. Observe the response y_t
3. Update internal statistics to produce h_{t+1}

Using a RNN parameterized by θ with combined update and query rule

$$\begin{aligned} h_t, x_t &= RNN_\theta(h_{t-1}, x_{t-1}, y_{t-1}) \\ y_t &= p(y|x_t) \end{aligned}$$

Loss Function:
We have 2 choices for loss function:

$$L_{final}(\theta) = E_{f,y_{1:T-1}}\big[f(x_T)\big], \text{ for some time horizon T}$$
$$\text{or}$$
$$L_{sum}(\theta) = E_{f,y_{1:T-1}}\left[\sum_{t=1}^{T} f(x_t)\right]$$

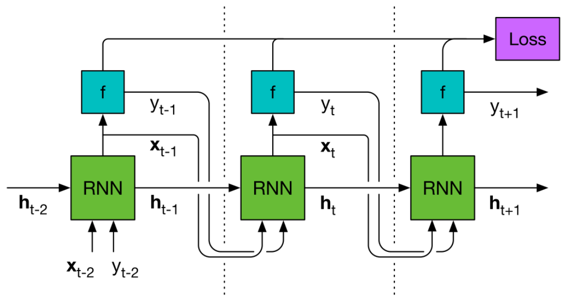
But we will prefer $L_{sum}(\theta)$ because the amount of information conveyed by L_{final} is temporally very sparse. By instead utilizing a sum of losses to train the optimizer we are able to provide information from every step along this trajectory. Note that similar type of loss function is considered for the previous paper - *Learning to Learn by Gradient Descent by Gradient Descent*
Exploration/Exploitation:
We can encourage exploration in the space of optimizers by encoding an exploratory force directly in to the loss function:

$$L_{EI}(\theta) = -E_{f,y_{1:T-1}}\left[\sum_{t=1}^{T} EI(x_t|y_{1:t-1})\right]$$

where $EI(.)$ is the expected posterior improvement of querying x_t given observations up to time t.

ML Approach Used in the Paper

- ***Meta-learning phase***



In the meta-learning phase, we use a large number of differentiable functions generated with a GP to train RNN optimizers by gradient descent. We consider two types of RNN:

- long-short-term memory networks (LSTMs)
- differentiable neural computers (DNCs)

The RNN optimizer learns to use its memory to store information about previous queries and function evaluations, and learns to access its memory to make decisions about which parts of the domain to explore or exploit next. That is, by unrolling the RNN, we generate new candidates for the search process.

References

[1] Marcin Andrychowicz et al. "Learning to learn by gradient descent by gradient descent". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. Curran Associates, Inc., 2016, pp. 3981–3989. URL: `http://papers.nips.cc/paper/6461-learning-to-learn-by-gradient-descent-by-gradient-descent.pdf`.

[2] Yutian Chen et al. "Learning to learn without gradient descent by gradient descent". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 748–756.

[3] *Exploration Exploitation figure*. `https://jeongyoonlee.com/2017/08/09/transition-from-exploration-to-exploitation/`. Accessed 11 March 2019.

[4] Prof. Piyush Rai. *Reinforcement Learning*. Tech. rep. Indian Institute of Technology Kanpur.

[5] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge, 1998.