COMP6247 Final Assignment: Learning Controller

Wei Chien Teoh (Eugene)
wct1c16@soton.ac.uk

4 June 2021

1 Introduction

This report presents the findings and results for the final assignment of COMP6247 [1]. The code implementation is stored in a Github repository [2].

2 Radial Basis Functions

This section explores the use of Radial Basis Functions (RBF) on Reinforcement and Online Learning problems.

First, RBF is used to solve a non-linear regression problem. The Airfoil Self-noise Dataset [3] from the UCI repository of machine learning is used as the problem of interest for regression. The dataset consists of samples of aerodynamic and acoustic tests of airfoil blades conducted in a wind tunnel. The features given include:

1. Frequency, in Hertz
2. Angle of attack, in degrees
3. Chord length, in meters
4. Free-stream velocity, in metres per second
5. Suction side displacement thickness, in metres

The output to be predicted is the scaled sound pressure level, in decibels.

First, we attempt to solve the problem with a linear regression in closed form, $\boldsymbol{W} = (X^T X)^{-1} X^T \boldsymbol{y}$. We then proceed to apply an RBF kernel with Gaussian function. Three approaches were used to solve for the RBF regression parameters:

- Closed form, $\boldsymbol{W} = (U^T U)^{-1} U^T \boldsymbol{y}$
- Gradient Descent (GD), epochs = 50000
- Mini-batch Stochastic Gradient Descent (SGD), batch size = 64, epochs = 10000

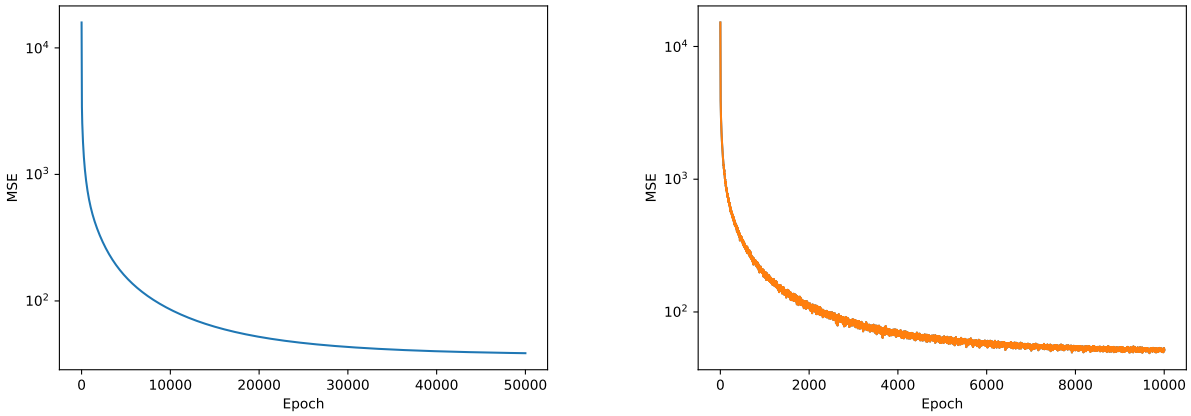
The Mean Squared Error (MSE) results are shown in Table 1. Linear regression is shown to be insufficient in solving this problem, as the MSE calculated is substantially higher than RBF regression solutions. The RBF closed-form solution provided the optimal solution. Both gradient descent methods show convergence of the loss function and also provided similar results, as illustrated in Fig. 1. Also, SGD shows more fluctuations during the training, it requires fewer epochs before convergence. The loss of the gradient descent methods can be further optimised by lowering the learning rate or using adaptive methods.

Table 1: Mean squared error for regression problem.

Model	MSE
Linear	915.288358
RBF Closed Form	33.754672
RBF GD	49.234992
RBF SGD	48.822703

3 Mountain Car Learning Controller

In this section, we explore the mountain car problem [4]. More specifically, we will attempt to learn a controller for the car to drive up a steep hill. We will explore several Temporal Difference (TD) learning algorithms and investigate the performance of them.



(a) Gradient Descent

(b) Stochastic Gradient Descent

Figure 1: Log-scaled loss curves for regression using gradient descent approaches.

3.1 Tabular Q Learning

Learning a controller involves learning an optimal policy. In Q learning, the action-value function is learned instead. The optimal policy and its associated value function can be further derived as:

$$\begin{aligned} \pi^* &= \arg\max_a Q(s,a) \\ V^* &= \max_a Q(s,a) \end{aligned} \tag{1}$$

The mountain car problem can be discretised into grids, where each block of the grid describes a state of which the car/agent is present. In this experiment, the problem is discretised into 40 states per state dimension (position and velocity). The Q function is tabularized with a dimension of $\mathbb{R}^{40\times40\times3}$, of which there exist 40×40 states and 3 actions, $a \in -1, 0, 1$ (left, neutral, right). The algorithm is performed for 1000 episodes, where the episode length is illustrated in Fig. 2. Note that the reward is described as -1 for every time step, thus the episode length can also be interpreted as the negative reward. For all experiments in this report, the ϵ-greedy policy will be used during training, where ϵ is set to 0.05. The plot shows evidence of the convergence towards an optimal policy that maximises reward, synonymous with minimising episode length.

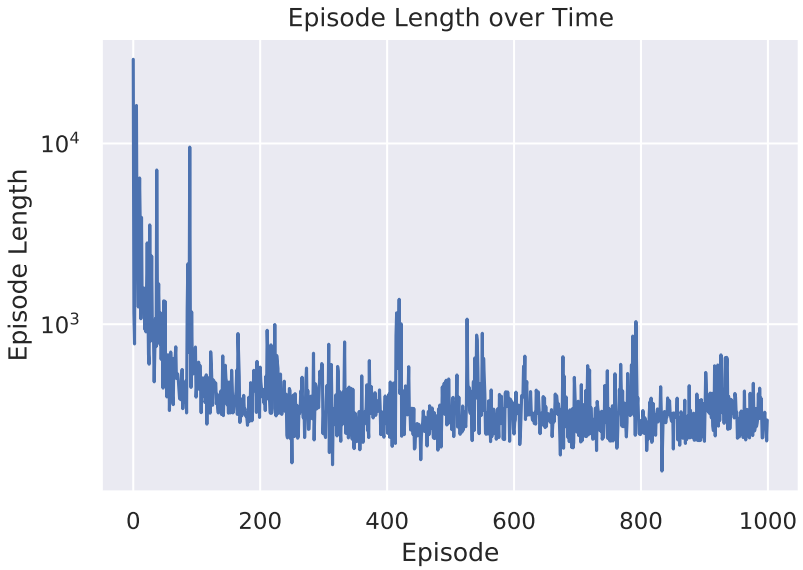


Figure 2: Log-scaled episode length over time. Final episode length = 154.

Following Eq. (1), the optimal value function can be obtained by taking the maximum of the corresponding

action with maximises the reward.

After obtaining the optimal value function, we attempt to approximate it using RBF regressors previously explored in Section 2. However, an RBF sampler using the Random Kitchen Sink [5] algorithm will be used as an approximation to the RBF kernel. The RBF sampler approximates the RBF by using Monte Carlo random sampling of its Fourier transform. The number of basis functions, J is varied and the centroids are randomly initialised. The scaling factor, σ_j is set to 1.0. Scaling does not provide significant effects on the results and thus is not further explored. Note that setting a global σ_j will allow the RBF network to have universal approximation capability [6]. The RBF regressors are solved using the closed-form solution.

Fig. 3 shows both the true and several examples of approximated value functions with different basis functions. A lesser number of RBF basis functions causes the approximated value function landscape to be less refined, whereas a greater number of basis functions produce landscapes that are visually different from the true landscape. The optimal approximation using RBF regression is with 20 basis functions, with an average reward of -128.4 and an MSE of 59 compared to the true value function. The approximated action-value functions are further reused as the optimal policy for the mountain car environment, of which the episode lengths are evaluated. The episode lengths and the MSE illustrated in Fig. 4 is shown to reflect the phenomenon shown in the value function landscapes. The episode length of lesser than 5 and greater than 25 basis functions do not converge even after 5000 steps. However, there are selected few (around 41 basis functions) that still provide good performances relative to the optimal value function (20 basis functions). The anomaly is potentially due to its basis functions having a similar shape to the true value function, even when its MSE is shown to exponentially diverge from the true value function after 25 basis functions, illustrated in Fig. 4b.

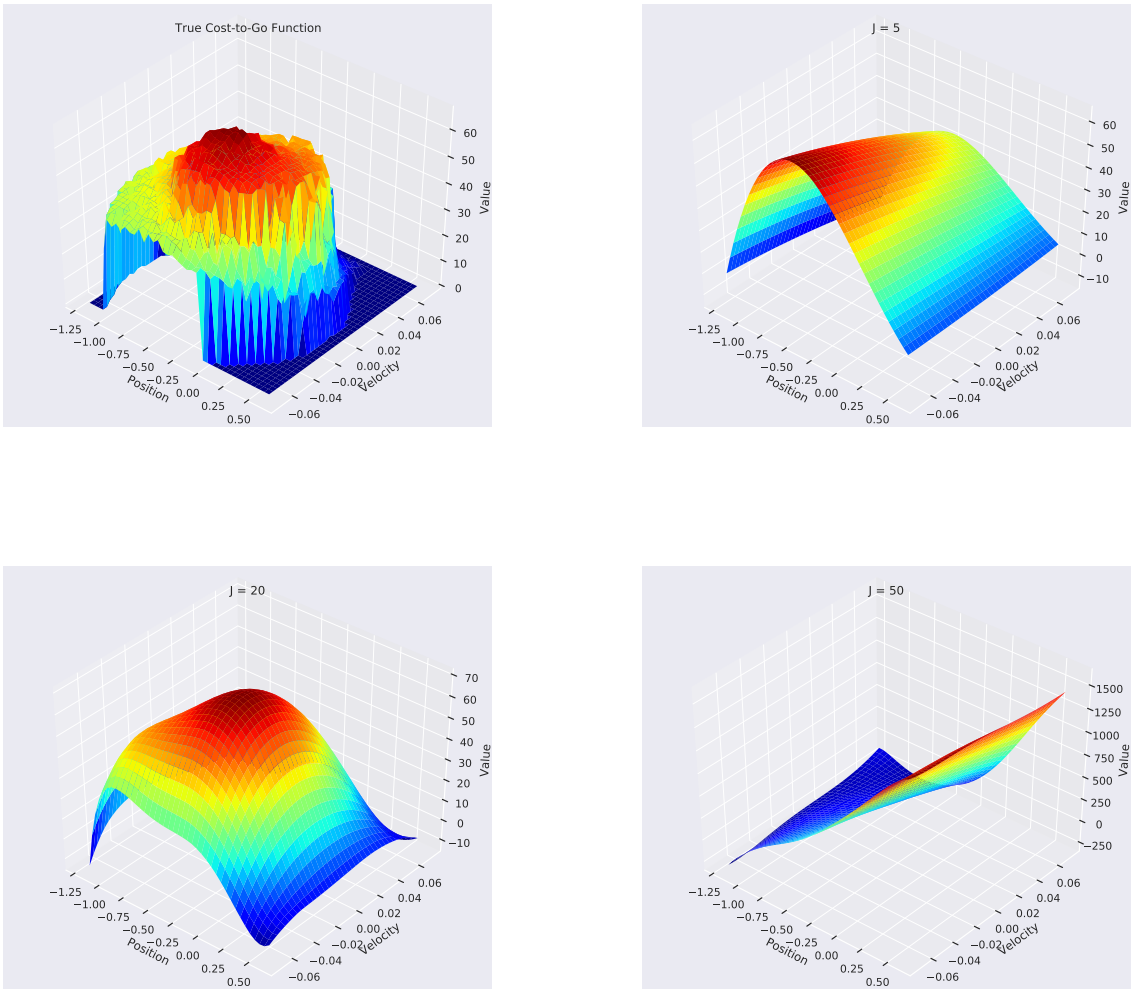
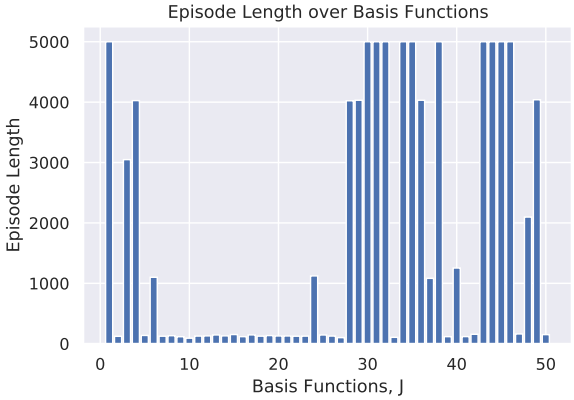
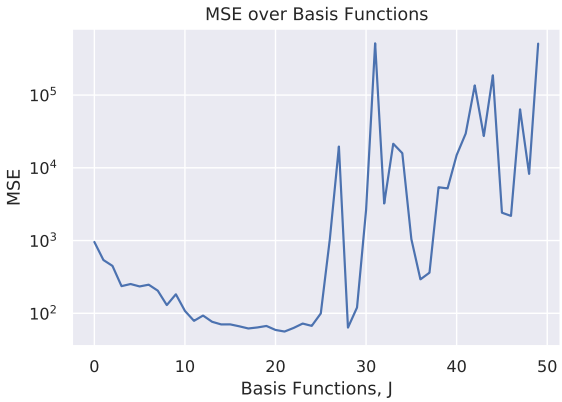


Figure 3: Cost-to-Go functions. (Top left) True value function. (Others) RBF approximation of value function with different number of basis functions, J.



(a) Episode length over number of basis functions, averaged over 5 runs. The environment is terminated after 5000 steps.



(b) Log-scaled MSE over number of basis functions.

Figure 4: Relevant metrics over different number of basis functions.

3.2 TD Learning with Function Approximation

We previously attempted to solve the mountain car problem by tabular discretisation of Q Learning. We then explored the use of RBF Networks to approximate the tabular Q function. One could combine these two concepts by directly learning the weights of the function approximator using online learning methods such as stochastic gradient or Bayesian inference. This section will explore both on-policy and off-policy Temporal Difference learning methods, namely Q Learning and SARSA, of which the Q function will be approximated by RBF networks.

$$Q(s,a) \approx \hat{q}(s,a) = \sum_{j=1}^{J} w_j \phi(\|\boldsymbol{x} - \boldsymbol{m}_j\| / \boldsymbol{\sigma}_j) \tag{2}$$

There are several advantages to function approximation of the Q function:

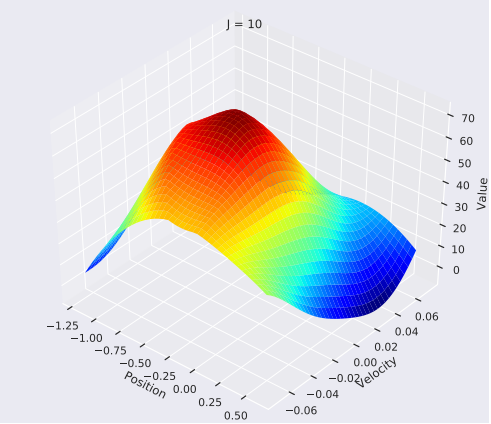
- As the number of state spaces become large, it is infeasible to tabulate values of each state space due to computational constraints. This also applies to continuous state space.
- Function approximation allows generalisation to unseen state spaces. Hence, it is not needed to explore every possible state to obtain an optimal policy or value function.
- Raw and unstructured data such as images can be directly be fed into the function approximator using state-of-the-art approaches such as Convolutional Neural Networks.

There are several approaches to the parameterization of the function approximator. Since Q takes both the states and actions as inputs, one could build a regressor for each action. However, it requires linear time to obtain the max or argmax of the action values. Alternatively, the RBF can be represented as a shallow neural network with multiple output nodes for each action [7]. Computing the max or argmax would then only result in taking the maximum value of the outputs. Since it is a neural network, the weights, centres, and scaling parameters can also be learned through backpropagation. Both the neural network and the backpropagation approaches were attempted but were unable to obtain a stable convergence of the Q function. This is potentially due to a large number of learnable parameters (including centres and scaling) and the stochasticity of the gradients in comparison to the small number of states and actions. Thus, the original linear time approach will be used in this experiment.

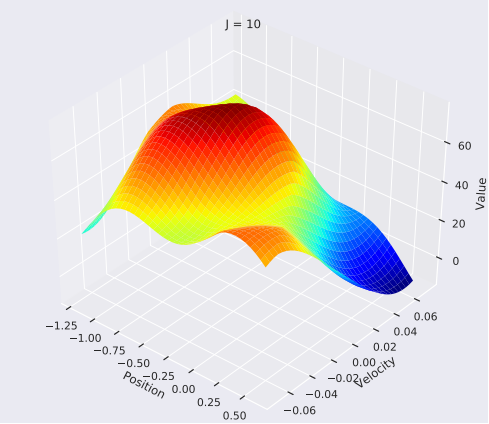
Again, the RBF kernel was approximated using the Random Kitchen Sink algorithm with σ_j as 1.0. As before, σ_j will not be tuned or modified in the experiments, as the objective in this section is to attempt to successfully utilise function approximation to further improve on previous methods. Three regressors were built for each action. The regressors were trained for 500 episodes. It should be noted that it required more time, approximately 4 times on average to train the function approximators. Five different number of basis functions, $J = (5, 10, 15, 20, 50)$ was used, and finally, the one with maximum reward is selected and illustrated in Fig. 5. Both plots did not achieve approximations as well as those in Fig. 3. This is potentially due to the stochasticity and the insufficient episodes due to time constraints. The stochasticity is caused by updating the gradient descent only one sample at a time. *Experience replay* can be introduced to solve this, where the

experiences at each time step, $e_t = (s_t, a_t, r_t, s_{t+1})$ is stored over many episodes to perform mini-batch updates of gradient descent. Despite the approximations, both Q Learning and SARSA with function approximation provided comparative rewards to the tabular method.

From the discussion and analysis above, it can be concluded that both tabular and function approximation approaches to TD learning were able to successfully solve the mountain car problem with decent rewards. As the scale of the problem is small, the tabular discretisation method is proven to be the better choice, exceeding the function approximation method in the number of hyperparameters to tune, training time and expected reward.



(a) (Q Learning) Average reward: -162.2
MSE: 245.1



(b) (SARSA) Average reward: -167.2
MSE: 590.2

Figure 5: Optimal Approximated Cost-to-Go Function. Optimal policy was performed for 5 runs, with average rewards stated above. MSE against the true value function is also stated.

3.3 Resource-Allocating Network

In previous sections, the number of basis functions, J was arbitrarily tuned by sweeping multiple different values. Such a tuning method is often impractical due to a large amount of time required. A Resource-Allocating Network [8] (RAN) allow the RBF network to grow in its number of basis functions in an online setting, using stochastic gradient descent. RAN can find the optimal number of basis functions by learning from the presented data. RAN grows its network by considering two conditions:

$$\begin{aligned} \|\boldsymbol{x}_n - \boldsymbol{u}_{nr}\| &> \zeta_n \\ e_n = y_n - f(\boldsymbol{x}_n) &> e_{\min} \end{aligned} \tag{3}$$

where:

- an observation $(\boldsymbol{x}_n, y_n)$ is given
- $\zeta_n = \max(\zeta_{\max}\gamma^n, \zeta_{\min})$
- $\boldsymbol{u}_{nr}$ is the nearest centroid to $\boldsymbol{x}_n$

Such an approach can be extended to TD learning methods, where RAN can be combined with function approximation. An example of Q learning with RAN is shown in Algorithm 1. The algorithm can also be modified to work with SARSA by changing the TD error to using the same policy (e.g. ϵ-greedy).

Suppose the model is non-linear, which an RBF network is of that, one can further extend the RAN algorithm by replacing the Least Mean Squares (LMS) algorithm (or SGD) with non-linear state estimation methods. The EKF is shown to work relatively well [9]. If the model is highly non-linear, Bayesian inference methods such as Sequential Importance Sampling is also feasible.

Algorithm 1: Q Learning with RAN Function Approximation.

Input: step size $\eta \in (0, 1]$, small $\epsilon > 0$, $\zeta_{\min} > 0$, $e_{\min} > 0$, $\zeta_{\max} > e_{\min}$, $0 < \gamma > 1$
$\hat{q}(s, a, \boldsymbol{w}) = w_0 + \sum_{j=1}^{J} w_j \phi_j(s, a)$, where $\phi(s, a)$ is an RBF kernel with centroid u_j and scaling factor σ_j
Initialize weights $\boldsymbol{w}$ arbitrarily (e.g., $\boldsymbol{w} = \boldsymbol{0}$)
$\zeta_n = \zeta_{\max}$, $w_0 = y_0$
for *each episode* **do**
  $S, A \leftarrow$ initial state and action of episode (e.g. ϵ-greedy)
  for *each step of episode* **do**
    `/* TD error */`
    Take action A, observe R, S'
    $\zeta_n = \max(\zeta_{\max}\gamma^n, \zeta_{\min})$
    if S' *is terminal* **then**
      $e_n = R - \hat{q}(S, A, \boldsymbol{w}^{(n)})$
    end
    else
      $e_n = R + \arg\max_a \hat{q}(S', a, \boldsymbol{w}^{(n)}) - \hat{q}(S, A, \boldsymbol{w}^{(n)})$
    end
    if $e_n > e_{\min}$ *and* $\|\boldsymbol{x}_n - \boldsymbol{u}_{nr}\| > \zeta_n$ **then**
      `/* Allocate a new hidden unit */`
      $\alpha_{K+1} = e_n$
      $\boldsymbol{u}_{K+1} = \boldsymbol{w}_n$
      $\sigma_{K+1} = \kappa\|\boldsymbol{w}_n - \boldsymbol{u}_{nr}\|$
    end
    else
      $\boldsymbol{w}^{(n+1)} = \boldsymbol{w}^{(n)} + \eta e_n \boldsymbol{w}_n$
    end
    $S \leftarrow S'$
  end
end

4 Learning to Optimize Neural Nets: Review

The paper "Learning to Optimize Neural Nets" [10] proposes a framework for learning optimization algorithms using reinforcement learning. Learning to optimize is a difficult problem as there exist many different types of objective functions, input data, and learning algorithms. Previous work from Bengio et al. [11] proposed a way to learn the formula used to update the model parameters during learning. However, the learned formula does not take the objective function into account, thus generalisation is an issue. Maclaurin et al. [12] proposed to learn regularization hyperparameters and learning rate scheduler. However, the generalization of hyperparameters to other objective functions was not considered. More recently, Andrychowicz et al. [13] proposed a method to learn from the objective function through backpropagation on a particular task using supervised learning. Again, the method fails to generalise to unseen objective functions, thus only allowing it to be applicable in specific tasks the model is trained on. The method which this paper proposes attempts to learn to optimize task-independent objective functions. The problem can be represented as a reinforcement learning problem, where an optimal policy is learned by rewarding those that converge quickly and penalizing those that do not. It will not depend on the data it is given but rather only the environment it interacts with. The author claims this approach could solve the generalisation problem of learning to optimize.

Rather than maximizing rewards, the optimal policy to be will be minimizing the expectation over a sequence of states and actions of the summation of the cost c. The cost in this case is the true objective value $f(x^{(t)})$.

$$\pi^* = \arg\min_{\pi} \mathbb{E}_{s_0, a_0, s_1, \cdots, s_T}[\sum_{t=0}^{T} c(s_t)] \tag{4}$$

In Eq. (4), there involves an expectation value of a joint distribution over states and actions. Expectations typically involve integrals, which are often intractable. The author then proposed the use of function approximators to approximate $\pi(a_t|o_t, t) = \mathbb{N}(\mu^\pi(o_t), \Sigma^\pi(o_t))$, where the functions $\mu^\pi(\cdot)$ and $\Sigma^\pi(\cdot)$ are learned.

The state s_t include:

- $x^{(t)}$, the current iteration loss
- $\phi(x^{(1)}, \cdots, x^{(t)})$, features depending on the previous iterations
- $\nabla\hat{f}(x^{(1)}), \cdots, \nabla\hat{f}(x^{(t)})$, gradients
- $\hat{f}(x^{(1)}), \cdots, \hat{f}(x^{(t)})$, values from the objective function

The action a_t is the update to the iteration Δx.

With the optimization problem defined above, it can then be solved with any reinforcement learning algorithm which can obtain the optimal policy π^*. The author proposed the use of Guided Policy Search (GPS). GPS introduces a new policy, ψ. The algorithm performs policy optimization on ψ, then uses it to train π. The constrained optimization problem of GPS is defined in Eq. (5). The constraints can further be relaxed by equating its expectations. The problem can then be solved using constrained optimization algorithms, such as ADMM.

$$\begin{aligned} &\min_{\theta,\eta} \mathbb{E}_\psi[\sum_{t=0}^{T} c(s_t)] \text{ s.t. } \psi(a_t|s_t,t;\eta) = \pi(a_t|s_t;\theta)\ \forall a_t, s_t, t \\ &\implies \min_{\theta,\eta} \mathbb{E}_\psi[\sum_{t=0}^{T} c(s_t)] \text{ s.t. } \mathbb{E}_\psi[a_t] = \mathbb{E}_\psi[\mathbb{E}_\pi[a_t|s_t]]\ \forall t \end{aligned} \tag{5}$$

In the experiments, the author uses shallow neural nets to train on the MNIST dataset and then tested on TFD, CIFAR-10 and CIFAR-100. The training set only consisted of one objective function, the cross-entropy loss. The experimental results have shown to work better than existing popular optimization algorithms on both the training and test datasets, especially compared to Maclaurin et al.'s work. The author successfully showed that it scales to larger neural net architectures, noisy gradients and longer time horizons.

However, there are several caveats with the experiments. The experiments only involve datasets of images, trained and tested on only one objective function. Thus, it does not fully reflect on their claim on generalising to different objective functions and tasks. Although the author attempted on larger models, it is nowhere the scale of state-of-the-art architectures such as ResNets. Moving forward, more experiments on state-of-the-art and larger models should be included, along with different data types.

Stemming from the paper, several open challenges can further improve this work. First, one can improve on the algorithm to learn the intractable expectation in Eq. (4). Current reinforcement learning methods have high variance and are highly susceptible to vanishing and exploding gradient problems. Secondly, one can improve on the generalisation of optimization algorithm trained on low-dimensional problems to high-dimensional problems.

References

[1] Mahesan Niranjan and Christine Evers. *COMP6247(2020/21): Reinforcement and Online Learning - Final Assignment*. School of Electronics and Computer Science, University of Southampton.

[2] Eugene Teoh. *Eugeneteoh/COMP6247-Reinforcement-Online-Learning*. May 7, 2021. URL: `https://github.com/eugeneteoh/COMP6247-Reinforcement-Online-Learning` (visited on 05/27/2021).

[3] *UCI Machine Learning Repository: Airfoil Self-Noise Data Set*. URL: `https://archive.ics.uci.edu/ml/datasets/Airfoil+Self-Noise` (visited on 05/27/2021).

[4] Andrew William Moore. *Efficient Memory-Based Learning for Robot Control*. 1990.

[5] Ali Rahimi and Ben Recht. "Random Features for Large-Scale Kernel Machines". In: (), p. 8.

[6] Yue Wu et al. "Using Radial Basis Function Networks for Function Approximation and Classification". In: *ISRN Applied Mathematics* 2012 (Mar. 6, 2012), e324194. DOI: `10.5402/2012/324194`. URL: `https://www.hindawi.com/journals/isrn/2012/324194/` (visited on 06/02/2021).

[7] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. Dec. 19, 2013. arXiv: `1312.5602 [cs]`. URL: `http://arxiv.org/abs/1312.5602` (visited on 06/03/2021).

[8] John Platt. "A Resource-Allocating Network for Function Interpolation". In: *Neural Computation* (1991).

[9] Visakan Kadirkamanathan and Mahesan Niranjan. "A Function Estimation Approach to Sequential Learning with Neural Networks". In: *Neural Computation* 5 (Nov. 1, 1993), pp. 954–975. DOI: `10.1162/neco.1993.5.6.954`.

[10] Ke Li and Jitendra Malik. *Learning to Optimize Neural Nets*. Nov. 30, 2017. arXiv: `1703.00441 [cs, math, stat]`. URL: `http://arxiv.org/abs/1703.00441` (visited on 06/04/2021).

[11] *Learning a Synaptic Learning Rule — Semantic Scholar*. URL: https://www.semanticscholar.org/paper/Learning-a-synaptic-learning-rule-Bengio-Bengio/d130325c41947a41a55a4431e9e8e15be89da8ea (visited on 06/04/2021).

[12] Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. *Gradient-Based Hyperparameter Optimization through Reversible Learning*. Apr. 2, 2015. arXiv: 1502.03492 [cs, stat]. URL: http://arxiv.org/abs/1502.03492 (visited on 06/04/2021).

[13] Marcin Andrychowicz et al. *Learning to Learn by Gradient Descent by Gradient Descent*. Nov. 30, 2016. arXiv: 1606.04474 [cs]. URL: http://arxiv.org/abs/1606.04474 (visited on 06/04/2021).