1 Introduction

Reinforcement learning (RL) is a subfield of machine learning (ML) wherein an agent learns to optimally interact with a simulated environment by attempting to maximize a reward signal. RL has been used to learn the dynamics of many difficult problems including scheduling for radiation therapy in cancer treatment, optimizing chemical reactions, and achieving superhuman performance in complex games like "Go". Unlike "supervised learning" tasks, tasks such as locating and identifying cancerous cells in images, RL does not require the expensive procedure of collecting and labeling large amounts of data. Therefore, creating practical RL algorithms with provable convergence guarantees is of great interest to the general scientific community wishing to apply ML to tasks where data collection is expensive and performance critical.

However, current popular deep RL techniques suffer from a lack of theoretical convergence guarantees. We propose a method which utilizes a lifted convex function as a drop in replacement for a deep neural network, and are working towards showing provable convergence guarantees within a popular RL framework. We successfully implement our method and show it's viability within a simple RL environment.

2 Related Work

Problems that fit into the RL framework can all be described by Markov Decision Processes, which specify the dynamics of an environment through sequences of *states* and *actions* the simulated agent takes to transition between these states. Deep Q-learning is a popular RL paradigm where neural networks are utilized to approximate the Q-value of a state-action pair, $Q(s, a)$. The neural network is utilized as a function approximator in an attempt to learn the value, i.e., the optimal reward, of state-action pairs. During training, Q-values are updated according to the Bellman optimality condition:

$$Q^*(s, a) = r + \gamma \max_{a'} Q^*(s', a') \qquad (1)$$

In practice, it is not clear that using neural networks as Q-value function approximators is optimal since the maximization procedure for choosing the best possible action a' in a given state s' is non-convex. Motivated by this, [1] introduce *input convex* neural networks (ICNNs). Since ICNNs are defined as the composition of convex activation functions and non-negative affine functions, they are convex in the input. The authors utilize this to find the best action given a fixed state, ensuring that they arrive at the maximal Q-value during training and inference.

I implemented and replicated the RL experiments done by [1] for fully and partially input convex neural networks (ICNNs & PICNNs). ICNNs specify the network weights to be positive along paths from both the input state and action vector, while PICNNs need positive weights only from the action vector input, given that the action is what we are attempting to optimize over. I utilized Tensorflow to create the ICNN DQNs, with a projection based method to learn the weights of the neural network. I evaluated both methods on a variety of RL tasks from OpenAI Gym [3].

Importantly, I obtained empirical evidence that training ICNNs as Q-value function approximators in the context of RL is practically challenging and does not have convergence guarantees: we observed that the learned fully ICNN Q-value function approximator can diverge to arbitrarily bad fits depending on choices of the hyperparameters. These results motivated us to seek a new convex function approximation scheme independent of ICNNs.

3 Fitting Convex Functions

ICNNs are simply the composition of positive affine functions and convex activation functions. If we fix the activation function to be any piecewise linear function, e.g., ReLUs, then ICNNs reduce to a composition of positive piecewise linear functions. Therefore, to directly learn a convex function similar to ICNNs, it is natural to study fitting convex piecewise linear functions of the form:

$$f(x) = \max\{a_1^T x + b_1, \ldots, a_L^T x + b_L\}.$$

Finding the best piecewise linear function that approximates a finite point set is framed as a convex optimization problem in [2]. I implemented this in Python with the package cvxpy by solving a quadratic program.

However, using simple two layer neural networks as function approximators easily outperforms the best convex piecewise linear function on simple regression datasets. Therefore, I extended the algorithm and implemented methods to compute the difference of an arbitrary number of convex functions. With this new algorithm, I was able to learn a set of convex piecewise linear functions whose difference could outperform simple neural networks on a variety of UCI regression datasets. Although practically effective, learning a difference of piecewise functions is intractable for many real-time applications we may be interested in, such as RL. Furthermore, since a difference of convex functions is not necessarily convex, we have not achieved our goal of allowing for efficient global minimization.

Instead, we explored learning the best convex piecewise linear function using an iterative method. We begin with the lower convex envelope of the data, then shift the anchors up through an efficient gradient descent update scheme which maintains convexity of the function. I debugged and tested an implementation of this method on a variety of regression datasets, comparing it against the previous methods. After experimenting, I found that this method learns a function that can oftentimes outperform the other methods on a held out test set due to neural networks overfitting and multi-function fitting being computationally inefficient. However, since we are in fact learning the best *convex* piecewise affine function, our solution can only be considered "good" if we assume that the data we are attempting to fit lies in some convex position. In datasets where the data clearly lie in non-convex positions, neural networks usually perform the best given that they are universal function approximators which can learn arbitrary functions.

4 Kernel Methods and Hyperplanes

Since the restriction that our training data points lie in convex position is quite strict, we extend our algorithm to work with kernels. Kernels are functions which map input data to higher dimensions where simpler functions can be considered more expressive. With high probability, points in Gaussian kernel space–a special kernel which maps pairs of input points to arbitrary dimensional space—lie in convex position since there are effectively infinitely many dimensions for the data to be distributed along.

Preliminary results indicate that we can in fact learn complicated and potentially non-convex functions in the input space due to the expressivity of the kernel. Furthermore, we can find global optima of these functions since our procedure learns a convex function in the "lifted" space. However, our algorithm really learns a set of points whose convex hull determines the function, and is not determined everywhere in kernel space. When a new point is sampled that lies outside the hull, our function is not well defined. To fix this, we extend our algorithm to learn

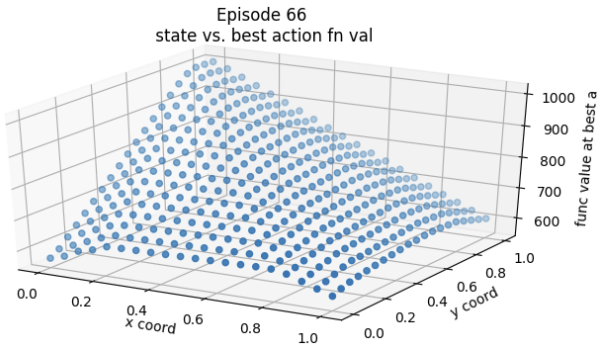

Figure 1: 2-D state vs. Q-value of best action within the state. Learned "envelope" function is concave and maximized at top left corner.

an associated hyperplane with each point in our lower envelope set. The convex function is then defined as the pointwise maximum of the set of hyperplanes across each anchor of our lower convex envelope.

5 Convex Function Fitting for RL

We use our kernel piecewise convex linear function as a drop in approximator for the Q-value function in deep Q-learning. Similarly to [1], we can explicitly optimize over our maintained function at inference time. Since we are attempting to choose the action which maximizes the Q-value given a fixed state, we assume the Q-value function is concave and instead fit our convex function to the negative Q-value function.

I ran simulations of our method on a simple RL environment where a randomly placed agent is rewarded for moving to the top left corner of a unit square by taking actions which correspond to moving a fixed distance in any direction. After working with our collaborators on debugging our method, we were able to learn a piecewise convex function in kernel space that correctly represents the Q-value function for the problem (Fig. 1). Our function correctly indicates that the max Q-value occurs at the top left corner since the agent receives 1000 units of reward for any action taken within that state.

6 Conclusions and Future Work

Although the initial results are promising, there is room for improvement in a variety of areas. Our current implementation is inefficient and has no parallelization. In the coming weeks, I plan to parallelize the optimization procedures required for each batch update through multiprocessing.

In parallel, our collaborators have proved the convergence of our method in fixed finite dimensions under somewhat restrictive assumptions. With more theoretical work, we hope to obtain convergence/correctness guarantees when our method is used as a drop-in function approximator for RL, unlike in deep Q-learning or in ICNNs. However, it is not clear that we can obtain the same convergence guarantees when working in, for example, Gaussian kernel space. I plan to investigate convexity in arbitrary dimensional spaces, attempting to prove convergence guarantees of our method.

Obtaining provable performance in RL is important for adoption in many, more critical applications within medicine and the hard sciences. Although our algorithm still needs improvements regarding runtime and performance, it is a promising first step in applying more traditional methods from convex optimization to RL that may overcome issues of convergence and correctness that are challenging to establish for current deep Q-learning approaches.

References

[1] B. Amos, L. Xu, and J. Z. Kolter. Input convex neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *PMLR*, pages 146–155, 2017.

[2] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[3] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.