Convex Functions for Reinforcement Learning

Siddartha Devic
The University of Texas at Dallas
800 West Campbell Rd
Richardson, Texas 75080

Abstract

Deep Q-learning and its variants yield state-of-the-art methods in many reinforcement learning benchmarks and tasks. However, deep Q-networks have the limitation that, during prediction, they require maximizing a non-convex function. Although recent work attempts to ameliorate this by modifying neural networks to instead approximate convex functions, we instead turn to the field of convex optimization to directly learn functions without the need for neural networks. We introduce a kernel based method to learn a piecewise linear convex function as a Q-value approximator. Due to the convexity of the learned function, we can optimize over the input state to determine the globally optimal action and are currently working to achieve competitive performance on a variety of reinforcement learning tasks.

Related Work

Deep Q-learning is a popular reinforcement learning (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:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a').$$

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 in a given state is non-convex. Motivated by this, (Amos, Xu, and Kolter 2017) 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 and structured prediction experiments done by (Amos, Xu, and Kolter 2017) 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 (Brockman et al. 2016). On the structured prediction side, I implemented the method for facial completion on the Olivetti dataset.

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.

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 convex piecewise linear functions. In particular, we are interested in understanding how well we can fit 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 (Boyd and Vandenberghe 2004). 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.

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. With high probability, points in Gaussian kernel space lie in convex position since there are effectively infinitely many dimensions for the data to be distributed along. Therefore, if we frame our optimization problem in terms of inner products of pairs of training data points, we can compute our functional fit in an arbitrary kernel space.

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. Our algorithm learns a set of points whose convex hull determines the function. However, the convex hull is not determined everywhere in kernel space. When a new point is sampled that lies outside our hull, our function is not well defined. To fix this, we extend our algorithm to learn 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.

Convex Function Fitting for Reinforcement Learning

We use our kernel piecewise convex linear function as a drop in approximator for the Q-value function in deep Q-learning. Similarly to (Amos, Xu, and Kolter 2017), we can explicitly optimize over our maintained function at inference time. Note that 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 the agent attempts to move to the corner of a two dimensional square. 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. I also ran preliminary simulations of our method on a variety of OpenAI Gym reinforcement learning tasks.

Although the initial results are promising, there is room for improvement in a variety of areas. The current implementation of our algorithm is inefficient and has no parallelization. In the coming months, I plan to parallelize the optimization procedures required for each batch Bellman update given to our method by experience replay sampling, a common reinforcement learning technique. 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 reinforcement learning, 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.

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 reinforcement learning that may overcome issues of convergence and correctness that are challenging to establish for current deep Q-learning approaches.

Timeline

- 12/31/19 — Verify correctness and complete parallelization of optimization procedures
- 3/31/20 — Finish preliminary and complex reinforcement learning experiments on more challenging tasks.
- 6/31/20 — Revisit convergence guarantees in arbitrary lifted spaces.

References

Amos, B.; Xu, L.; and Kolter, J. Z. 2017. Input convex neural networks. In Precup, D., and Teh, Y. W., eds., *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, 146–155. International Convention Centre, Sydney, Australia: PMLR.

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

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