Temporal Convolutional Policy Networks

YuXuan Liu **Tony Duan** **Wesley Hsieh**
University of California, Berkeley
{yuxuanliu, tonyduan, wesleyhsieh}@berkeley.edu

Abstract

Many real world tasks can be modeled as partially observed Markov decision processes where the underlying state is not fully observed. In these tasks, policies that condition only on a single observation are usually inadequate for success. Previous work in solving partially observed problems have emphasized memory-based solutions with recurrent networks, such as long-short term memory networks. In this paper, we propose temporal convolutional policy networks for solving partially observed tasks. We compare our models on classical control problems as well as in the Atari domain. We show that our convolutional networks can match if not exceed the performance of recurrent networks for these tasks.

1 Introduction

In many real world problems, we rarely have access to the full state of the system – it is more common to receive observations of the state that are noisy or incomplete. These problems can be modeled as partially observed Markov decision processes, a generalization of the fully observed case. For instance: in a robot manipulation task, we may be given joint angles and images from a camera but not velocities of the objects in the environment. Images such as these are a common source of partial observations, as positions and velocities of objects must be inferred [7].

Approaches to solving Markov decision processes can typically be categorized as either model-based or model-free. In model-based approaches, we maintain a belief distribution over the current state given a series of observations [15]. However, this requires an extensive model of the environment, and can be computationally expensive.

Recent advances in deep reinforcement learning have demonstrated success in model-free approaches. Policy networks and deep q learning, for example, have been able to outperform human experts at playing Atari games [12]. For the partially observed problems we are considering, parametrized policy networks using recurrent memory networks (such as long-short term memory [6]) have been particularly effective. However, recent advances in convolutional models for images [13] and raw audio [14] have shown that convolutional networks can capture spatial and temporal dependencies as well. Inspired by this success, we propose temporal convolutional networks for solving partially observed Markov decision processes.

1.1 Background

In the problem of reinforcement learning (RL), an agent interacts with the environment and tries to optimize for a reward. RL provides a generalized framework for optimal behavior in environments given only a reward signal. When combined with neural networks, deep RL has demonstrated widespread success in robotic manipulation, autonomous helicopter control, Atari, and Go [11] [8] [1] [17].

1.2 Problem Statement

We can model reinforcement learning with a Markov decision process defined over a state space S, a set of actions A that the agent can perform at each time-step, a transition function $T(s', a, s)$ representing the probability of transitioning between states after performing an action, and a reward function $R(s)$ defined over the state space. In most problems we may also have a discount factor γ and an initial state distribution ρ_0. The reinforcement learning problem is therefore the following: find a policy $\pi^*(a|s)$ such that the expected sum of discounted rewards is maximized.

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right] \tag{1}$$

Here, τ is the trajectory of state action pairs when following the policy π. We assume $s_0 \sim p_0, a_t \sim \pi(a_t|s_t), s_t \sim T(s_{t-1}, a_t, s_t)$.

For partially observed Markov decision processes, we no longer have access to the entire state, but rather an observation which is a random function of the current state $o(s)$ (note that for a fully observed MDP, $o(s) = s$). The partially observed problem then is to find a policy $\pi(a_t|o_t, a_{t-1}, o_{t-1}, ..., a_0, o_0)$ that optimizes the sum of expected rewards in equation (1).

2 Approaches

There are two main families of recent approaches in deep reinforcement learning. In value based methods, we approximate the expected sum of discounted rewards from any state s_t after performing action a_t [11]. From this approximation, we can select actions that maximize the expected reward at any state. The second family of approaches are policy optimization methods in which we parametrize and optimize a policy directly. There are various policy optimization techniques including evolutionary methods such as the cross entropy method [2] as well as policy gradient methods.

2.1 Value Approximation

In value-based methods, we define the values:

$$V_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}...}\left[\sum_{t'=t}^{\infty} \gamma^{t'} R(s_{t'})\right] \tag{2}$$

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}...}\left[\sum_{t'=t}^{\infty} \gamma^{t'} R(s_{t'})\right] \tag{3}$$

where $V_\pi(s_t)$ is the the expected sum of discounted rewards from state s_t and $Q_\pi(s_t, a_t)$ is the expected sum of discounted rewards from state s_t after performing a_t. In finite state-action spaces, approximate dynamic programming methods can solve for V_π and Q_π. For continuous state-action spaces, we approximate V_π and Q_π with a linear combination of features or a neural network over the state/actions.

In recent works, Q-value approximation has been shown to be effective in learning controls for Atari games from raw images [11]. These Q-values are parameterized as neural networks with convolutional layers that feed into full connected layers. In one-step Q-learning, we minimize the L2 error of one (s, a, s') tuple. The loss is defined as

$$L(\theta_i) = \mathbb{E}\left(R + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right)^2 \tag{4}$$

Where θ_{i-1} is the parameters from the previous iteration, and s' is the state after performing action a, and R is the reward obtained.

2.2 Policy Optimization

Another family of reinforcement learning algorithms directly optimize the policy $\pi(a|s)$. In policy optimization, we can parameterize the policy $\pi(a|s;\theta)$ where θ can be linear feature weights or parameters in a neural network. Cross entropy methods use an evolutionary algorithm that selects the top policies sampled from a distribution over policy parameters [2]. Policy gradient methods such as REINFORCE approximate the gradient of the policy parameters from sampling [18].

$$\nabla \mathbb{E}_\tau \left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \right] = \mathbb{E}_\tau \left[\left(\sum_{t=0}^{\infty} \gamma^t R(s_t) \right) \left(\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \right] \tag{5}$$

More stable gradient methods such as Trust Region Policy Optimization [16] optimize the policy subject to a KL divergence bound, preventing the policy from changing too much in each iteration.

2.3 A3C

In policy gradient methods, it is common to reduce variance of the gradient estimate with a baseline function of the state $b(s_t)$

$$\mathbb{E}_\tau \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{\infty} \gamma^{t'} R(s_{t'}) - b(s_t) \right) \right] \tag{6}$$

A near optimal choice of a baseline is the value function $b(s_t) = V(s_t)$, in which case we have an estimate of the advantage $\hat{A} = \sum_{t'=t}^{\infty} \gamma^{t'} R(s_{t'}) - V(s_t)$. We can view this as an "actor-critic" algorithm where the policy π is the actor and the value function $V(s_t)$ is the critic.

In conventional Q learning, we maintain a replay buffer and sample gradient updates from that buffer to decorrelate updates. However, asynchronous methods have been shown to be effective in which multiple agents sample and update the policy on random initializations of the environment. The asynchronous advantage actor-critic method (A3C) maintains a policy $\pi(a_t|s_t;\theta)$ and value function $V(s_t;\theta)$ while performing asynchronous updates [10].

2.4 Temporal Convolutions

In partially observed domains, policies that condition only on the current observation may be insufficient to solve the problem – a way to capture memory over multiple observations is necessary. Previous approaches to partially observed tasks have focused on recurrent networks for capturing sequential dependencies [6]. In this paper, we propose a temporal convolutional network in which we convolve over a series of past observations.

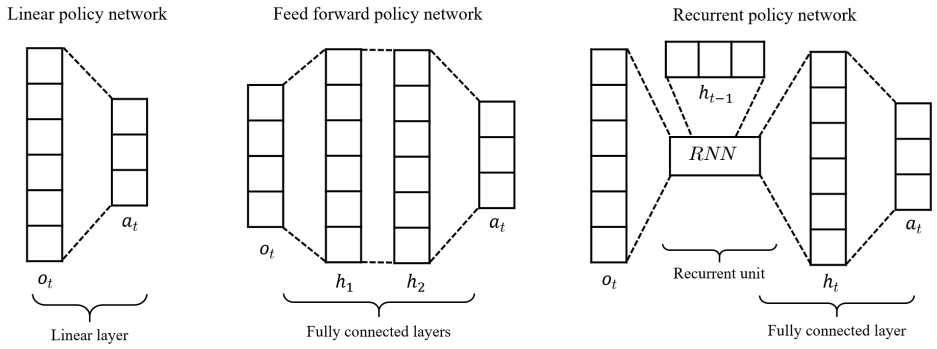

Figure 1: Baseline policy networks: linear, feed forward, and recurrent.

Classical policy parametrizations to solve control problems include linear, feed forward neural networks, and recurrent neural networks [Figure 1]. We propose two novel temporal convolutional networks: TConv and TConvRNN [Figure 2]. In TConv, we perform a width 2, stride 1 convolution over T previous observations before feeding the convolutional output into a fully connected layer. In

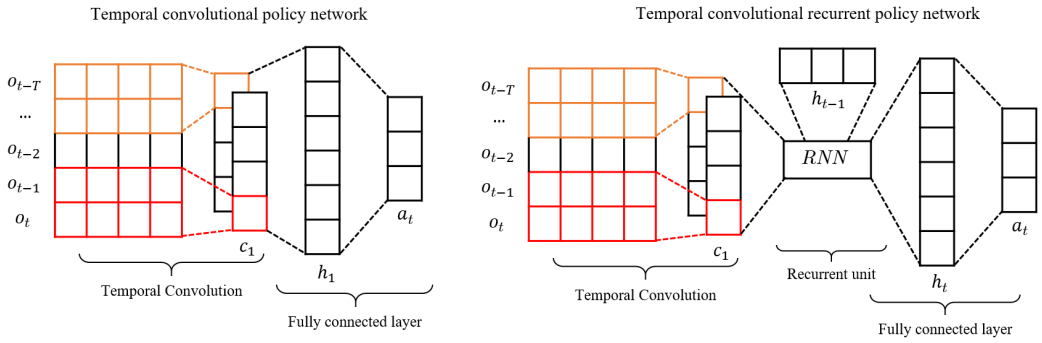

Figure 2: Our proposed temporal convolutional networks: TConv and TConvRNN.

TConvRNN, we use the same TConv architecture but the fully connected output is a 32 unit hidden layer which is used as input into a GRU [4] [Figure 2].

In the original A3C architecture, the policy and value networks share a common sub-network consisting of a series of convolutional and fully connected layers. The output of this sub-network feeds into a softmax layer for the policy and a linear output for the value function [Figure 3]. In recurrent A3C, the last layer of the shared sub-network is a recurrent layer.

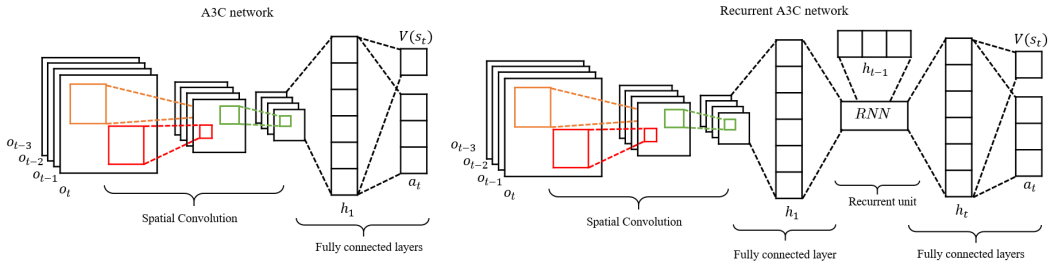

Figure 3: Baseline A3C Networks: Convolution over 4 previous observations followed by RNN or fully connected layers.

We propose two temporal convolutional networks: A3CTConv and A3CTConvRNN [Figure 4]. In A3CTConv, we add a temporal convolution layer after the last convolutional layer. Since each convolution is performed over 4 previous observations, the temporal convolution has a receptive field of $4T$ previous observations. In A3CTConvRNN, we maintain the same A3CTConv architecture; however, the last layer of the sub-network is a recurrent layer.

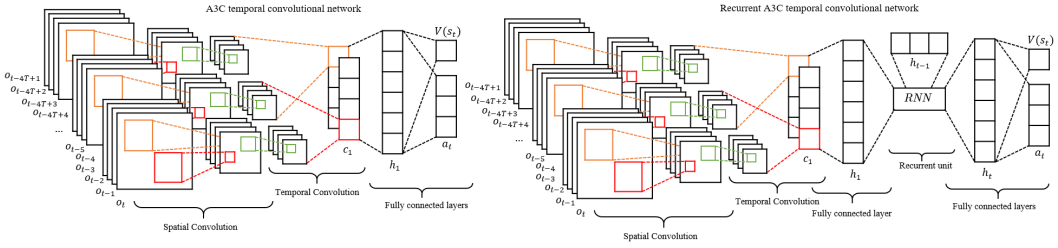

Figure 4: Our proposed temporal convolutional networks: A3CTConv and A3CTConvRNN. We perform temporal convolutional over T previous conv features.

3 Experiments and results

We evaluated our convolutional networks on partially observed environments in both classical control problems and the Pong Atari game.

3.1 Tools

RLLab [5] is a library that provides a framework for building and evaluating reinforcement learning algorithms. It provides an implementation of popular RL algorithms including vanilla policy gradient, the cross entropy method, and trust region policy optimization. It also provides a set of environments across classical control problems, MuJuCo, and OpenAI Gym [3]. We used RLLab extensively for evaluating learning curves and providing environments. We modified the relevant experimental environments in RLLab to become partially observed by removing features (such as velocity) from the state space. All neural networks were implemented in TensorFlow [9].

3.2 Control problems

We experimented on the following control problems in RLLab, modified to be partially observed:

1. CartPole: the goal is to keep a pole balanced using the cart. We removed the velocity of the cart from the observations, observing only position and angle.
2. Mountain Car: the goal is to control a cart to reach the top of the mountain. We removed the velocity of the cart, observing only position.
3. Double Pendulum: the goal is to swing the pendulum to keep it vertical. We removed velocity, observing only position and angle.

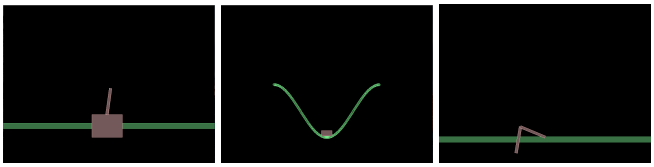

Figure 5: The control problems on which we evaluated our convolutional network. CartPole, Mountain Car, and Double Pendumlum in that order from left to right.

We benchmarked our two new TConv and TConvRNN policy networks against the three baselines: linear, feed forward neural network (FF), and recurrent GRU network (RNN). For consistency, all networks were trained with the Trust Region Policy Optimization algorithm [16].

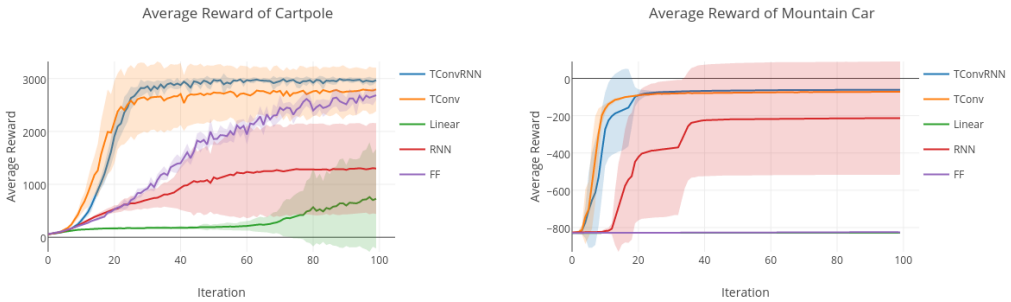

Figure 6: Learning curves for our temporal convolution networks on the CartPole and Mountain Car problems. Our TConv and TConvRNN models tend to have the best performance. In some cases, policies that condition only on the current observation fail to learn any meaningful policy.

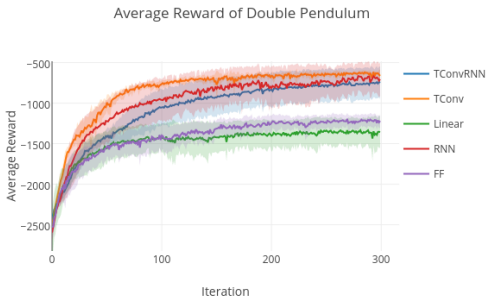

Figure 7: Learning curves for the Double Pendulum environment. Our TConv and TCounvRNN policy networks have comparable performance with the RNN policy.

Notice that all three tasks are made substantially more difficult when velocity is removed from the state. The linear policy and feed-forward network are unable to receive any reward in the Mountain Car environment because they cannot infer velocity over time.

3.3 Atari Pong

We evaluated our models for the Atari domain as well, which has become a popular method to benchmark RL algorithms. We trained our networks on raw 84x84 images using A3C with 12 asynchronous processes. Since we use raw images, the problem is inherently partially observed – a single static image confers no notion of velocity. We compared the learning curves of our convolutional models (A3CTConv and A3CTConvRNN) against the baseline A3C feed forward network (A3C) as well as a recurrent A3C network (A3CRNN).

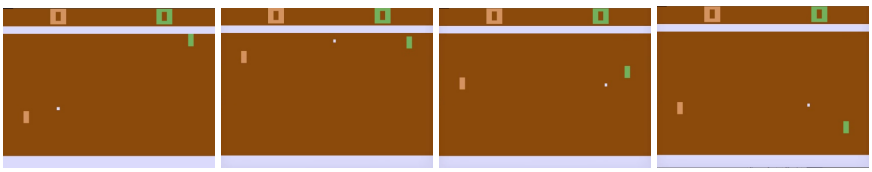

Figure 8: A sequence of 4 pong frames. Notice that velocity of the ball cannot be inferred from any single frame and requires a sequence of frames.

We confirmed that our temporal convolutional networks can learn an optimal policy with far fewer samples than the baseline models. Between the temporal networks themselves, an interesting observation is that the A3CTConvRNN model learned at a slower rate than the non-recurrent A3CTConv model, but was eventually able to achieve a more consistent and higher average score.

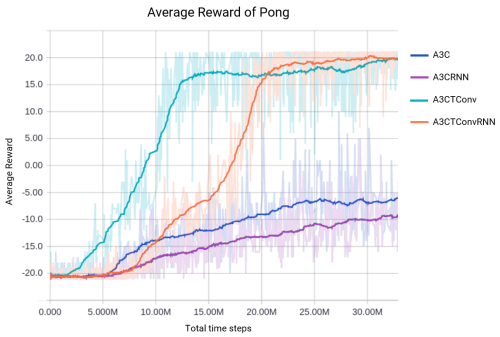

Figure 9: Learning curves for Pong. Our temporal convolutional networks (A3CTConv and A3CTConvRNN) can learn much faster than the baseline feed forward (A3C) and recurrent (A3CRNN) networks.

4 Conclusions

We have found that temporal convolutional networks outperform or match performance of recurrent neural networks and vanilla feed-forward neural networks for many partially observed Markov decision processes. The improvement in performance is more pronounced in more complex environments such as Pong; our temporal convolutional networks tend to both learn faster and attain a higher average score.

Due to the noise and restricted set of observations available in partially observed environments, it is unsurprising that vanilla single-state learners are unable to fully solve the problems. Recurrent models have traditionally been employed to retain memory from the state history. We have introduced temporal convolutional networks to the task, and shown that they can outperform recurrent networks, at least in learning speed. Our combined convolutional and recurrent model tends to perform the best, learning policies that attain both high score and low variance.

5 Future Work

There are many interesting directions in which we con continue our work.

On the model side, we can attempt to capture more long term dependencies with hierarchical and dilated convolutions. For experimentation and evaluation, we can apply our networks to more complex partially observed environments; tasks that require inferring information from state history beyond only velocity. This would allow us to assess how the benefit of temporal convolutional layers scales with task complexity.

6 Team Contributions

YuXuan Liu (41.11%): I implemented the temporal convolutional networks in both Rllab and with A3C. I ran and created graphs for all experiments. I helped put together parts of the poster and presentation and contributed to the Experiments section. I wrote the Abstract, Introduction, and Approaches sections of the paper and created diagrams for baseline and our models.

Tony Duan (29.44%): I implemented partially observed environments for control problems. I helped put together parts of our presentation, drafted our poster and diagrams, wrote the Experiments and results section of our paper, and contributed to the Introduction and Conclusion as well.

Wesley Hsieh (29.44%): I implemented partial observations and state history features in the experimental problem environments. I helped put together parts of the poster and presentation I wrote the Conclusion and Future Work sections of the paper.

References

[1] Pieter Abbeel, Adam Coates, and Andrew Y. Ng. "Autonomous Helicopter Aerobatics Through Apprenticeship Learning". In: *Int. J. Rob. Res.* 29.13 (Nov. 2010), pp. 1608–1639. ISSN: 0278-3649. DOI: 10.1177/0278364910371999. URL: http://dx.doi.org/10.1177/0278364910371999.

[2] P. T. De Boer et al. "A Tutorial on the Cross-Entropy Method". In: *Annals of Operations Research* 134 (2002).

[3] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[4] Junyoung Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *CoRR* abs/1412.3555 (2014). URL: http://arxiv.org/abs/1412.3555.

[5] Yan Duan et al. "Benchmarking Deep Reinforcement Learning for Continuous Control". In: *CoRR* abs/1604.06778 (2016). URL: http://arxiv.org/abs/1604.06778.

[6] Nicolas Heess et al. "Memory-based control with recurrent neural networks". In: *CoRR* abs/1512.04455 (2015). URL: http://arxiv.org/abs/1512.04455.

[7] Sergey Levine and Vladlen Koltun. "Guided policy search". In: *Proceedings of The 30th International Conference on Machine Learning*. 2013.

[8] Sergey Levine et al. “End-to-end Training of Deep Visuomotor Policies”. In: *J. Mach. Learn. Res.* 17.1 (Jan. 2016), pp. 1334–1373. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2946645.2946684.

[9] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[10] Volodymyr Mnih et al. “Asynchronous Methods for Deep Reinforcement Learning”. In: *CoRR* abs/1602.01783 (2016). URL: http://arxiv.org/abs/1602.01783.

[11] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. URL: http://dx.doi.org/10.1038/nature14236.

[12] Volodymyr Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: *CoRR* abs/1312.5602 (2013).

[13] Aäron van den Oord et al. “Conditional Image Generation with PixelCNN Decoders”. In: *CoRR* abs/1606.05328 (2016). URL: http://arxiv.org/abs/1606.05328.

[14] Aäron van den Oord et al. “WaveNet: A Generative Model for Raw Audio”. In: *CoRR* abs/1609.03499 (2016). URL: http://arxiv.org/abs/1609.03499.

[15] Andres C. Rodriguez, Ronald Parr, and Daphne Koller. “Reinforcement Learning Using Approximate Belief States”. In: *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*. 1999, pp. 1036–1042. URL: http://papers.nips.cc/paper/1667-reinforcement-learning-using-approximate-belief-states.

[16] John Schulman et al. “Trust Region Policy Optimization”. In: *International Conference on Machine Learning (ICML)*. 2015.

[17] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529 (2016), pp. 484–503. URL: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

[18] Ronald J. Williams. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning”. In: *Mach. Learn.* 8.3-4 (May 1992), pp. 229–256. ISSN: 0885-6125. DOI: 10.1007/BF00992696. URL: http://dx.doi.org/10.1007/BF00992696.