CS230

Deep Reinforcement Learning for Unsteady Flow Control

Anthony Corso
Department of Aeronautics and Astronautics
Stanford University
acorso@stanford.edu

Abstract

Unsteady flow control remains a difficult task for even the most advanced control techniques, and if solved, could benefit many industrial applications. This work demonstrates that deep reinforcement learning can be used to discover control policies for active flow control. Vortex shedding behind a 2D circular cylinder is studied and a functional control policy is learned for a Reynolds number of 50. Multiple reinforcement learning algorithms and architectures are compared and contrasted. Additional analysis reveals that the near-wake flow-field is most important when designing a control policy.

1 Introduction

Unsteady flow control remains a difficult task for even the most advanced control techniques. Control in this domain is challenging because the dynamics of an unsteady fluid system are nonlinear, high-dimensional, and continuous. Solving the active flow control problem, however, could have a big payoff for many industrial applications including energy generation, transportation, climate control, and chemical processing.

This work focuses on a canonical flow-control problem: the suppression of vortices behind a cylinder in a 2D simulated fluid flow (see fig. 3 for an example of such vortices). The appearance of vortex shedding is governed by a flow parameter called the Reynolds number, given by $\mathcal{R} = \frac{\rho U_\infty D}{\mu}$ where U_∞ is the freestream velocity, ρ is the fluid density, μ is the fluid viscosity, and D is the diameter of the cylinder. When the Reynolds number is larger than a critical threshold (in this case $\mathcal{R} > 45$), then small perturbations in a steady flow field can grow into oscillating vortices that a shed into the wake of the cylinder. Vortex shedding leads to increased drag on the cylinder, and is generally undesirable when considering the performance of aerodynamic systems. These vortices can be disrupted and suppressed by injecting momentum and energy into the flow field in a highly specific manner. The two most common ways of suppressing vortices are by rotating the cylinder to use frictional forces to add energy to the flow, or by using jets to inject momentum and energy directly.

The best control policy for when and how to inject momentum into the fluid is difficult to discern. This work attempts to cast the problem of unsteady flow control as a Markov Decision Process (MDP) and apply deep reinforcement learning (deep RL or DRL) techniques to automatically find a viable control policy. Deep reinforcement learning uses a Deep Neural Network (DNN) to encode a mapping from system state to control action, and optimizes that mapping to achieve good performance on a desired task. For the problem of vortex shedding, the state (or input to the reinforcement learning algorithm) is a snapshot of the flow-field around the circular cylinder. In this case, the snapshot contains density, x-velocity, y-velocity, and energy values on a 256×128 grid. The output of the DRL control policy will be the angular velocity of the cylinder. If the action space is continuous then this will correspond to a single real value, but if it is discrete then the angular velocity will be binned into a user-specified range of values.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

This paper is organized as follows: section 2 gives a brief overview of past work to give context to this problem. Then `gym-pyfr` is presented in section 3 as a new framework for solving flow control problems using RL methods. The methods applied to the vortex suppression problem are discussed in section 4 and the results of several experiments are given in section 5. Lastly, section 6 concludes and gives directions for future work.

2 Related work

Traditionally the problem of vortex suppression behind a circular cylinder has been studied experimentally such as Bearman & Branković (2004) or in a traditional control framework such as Homescu et al. (2002). Experimentation is costly and can be unprincipled, while the use of traditional control techniques can be challenging for a system as complicated as the unsteady Navier-Stokes equations. Homescu et al. (2002) used optimal control strategies to solve for the optimal control policy for vortex suppression, but needed to solve a carefully crafted and complicated set of equations to do so, making it difficult to extend this methodology to other problem formulations.

There has recently been some work applying Deep Neural Networks (DNNs) to solve the vortex suppression problem. Morton et al. (2018) used a DNN to learn a linearization of the dynamics of the system and then applied model predictive control to suppress the vortex shedding (using cylinder rotations) at a Reynolds number of 50. This work also demonstrated that flow control (at $\mathcal{R} = 50$) can be achieved using a proportional controller based on the y-velocity of the fluid at a specific point in the wake (0.4 times the diameter of the cylinder). The appropriate gain was found to 0.88. Rabault et al. (2018, 2019) used the proximal policy optimization algorithm (PPO) Schulman et al. (2017) to learn a control policy for sucking and blowing jets on the surface of a cylinder. Traditional PPO was insufficient to solve the control problem, however, and the authors used several tricks to get the policy to be continuous and smooth which sped up the learning process. Positive results were achieved for a Reynolds number of $\mathcal{R} = 100$. The reward function that was used in this case was a function of drag and lift on the cylinder.

3 RL Framework for Flow Control (Data Generation)

There are several limitations of many modern fluid flow simulations that make them difficult to use with existing reinforcement learning frameworks. First, simulations are often setup via a configuration file or other non-programmatic setup (like a GUI), making it difficult to dynamically generate many configurations. Second, fluid simulations are notoriously slow due to the difficulty of accurately solving the Navier-Stokes equations. Lastly, simulators often have interfaces that do not allow for interaction with the simulation while it is running. To address these limitations and to help bring RL to the fluids community, this work presents `gym-pyfr` a new, open-source, framework for solving flow control problems with RL. The framework is built on top of two outstanding modules, `OpenAI gym` (Brockman et al. (2016)) and `PyFR` (Witherden et al. (2014)).

`PyFR` is an open-source flow simulation package written in python that achieves good simulation performance using GPU computing. To accelerate the performance for the cylinder problem for this work, the cylinder mesh was made as coarse as possible (through trial and error) while still retaining solution accuracy. The original mesh used for rotational flow control had 3234 elements while the newly created mesh has only 653 and runs about $10\times$ faster.

`OpenAI gym` is a python module that provides a common interface for solving RL problems. To bring `PyFR` into a gym environment, a lot of work had to be done to modularize the running of `PyFR` and allow simulations to be setup programmatically. A gym environment must provide three main functions to be used with most existing reinforcement learning algorithms: `init`, `step`, and `reset`. Below is a description of what each of these functions do in the context of a `gym-pyfr` environment.

- `init(options)` - Initializes the observation (state) space to be $256 \times 128 \times 4$ and the action space to be either 1D continuous (the rotational velocity of the cylinder) or discretized into a desired number of bins. A `PyFR` object (see appendix for details) is created based on the user-specified options such as the desired $\mathcal{R}$ and episode length parameters.
- `step`(a) - This function first sets the cylinder rotation based on the specified action, a. Then the simulation is advanced by one timestep and the reward at the new state is observed.

Lastly, the timestep is checked to see if the end of the episode has been reached. The function returns the new observation, the reward, and a flag indicating episode completion.

- `reset()` - This function re-initializes the `PyFR` object back to its starting state and returns the state of the flow field.

4 Methods

This sections outlines the formulation of the problem statement in more detail and describes the specific algorithms and architectures that are applied to it.

4.1 Markov Decision Process Formulation

In order for this problem to be solved using common DRL algorithms, we formulate it as a Markov Decision process which is a 5-tuple of the form ($\mathcal{S}$, A, P, γ, R). $\mathcal{S}$ is the state space, A is the action space, P is a the transition probability, γ is the discount factor, and R is the reward function (which is detailed in the following subsection). For this problem, the following are defined as

- $\mathcal{S}$ is a (128,256,4) tensor that stores the flow-field information at each gridpoint
- $\mathcal{A}$ is a continuous or discrete variable a that represents the angular velocity of the cylinder
- $\mathcal{P}(s' \mid s, a)$ is a deterministic function governed by the Navier-Stokes equations
- The discount factor will be set to $\gamma = 1$ since episode lengths are finite

The goal of a reinforcement learning algorithm is to learn an optimal mapping given by $p(a) = \pi^*(s)$. The optimal policy $\pi^*(s)$ returns a probability distribution over the best action to take given the current state s. Some RL algorithms seek to learn π^* directly while others try to learn another function known as the optimal state-action value function $Q^*(s, a)$ which returns the expected sum of future rewards when following an optimal policy. The optimal policy itself can then be extracted from Q^* via

$$\pi^*(s) = \max_a Q^*(s, a) \tag{1}$$

As detailed further is section 4.3 both approaches will be attempted.

4.2 Reward Function

A reward function is chosen to guide the RL algorithm into producing steady flow fields in the wake of the cylinder. In this work, we take the approach of creating a baseline flow-field that is steady, and then giving a reward based on how close the current flow-field is to the baseline. To make this more concrete, we can define the reward r to be

$$r(s, a) = - \left\| x - x_b \right\|_2 \tag{2}$$

where x is a vector containing each degree of freedom for each point in the flow field, and x_b is the baseline flow state.

The baseline flow fields were found by running the fluid simulation with vortex shedding for 210 timesteps and then time-averaging the solution to get a pseudo-steady flow solution. Although this approach does not give the true steady-state profile, it is close enough to promote vortex suppression.

4.3 Algorithms and Network architecture

Two Deep RL algorithms were used and compared, Deep Q-Learning (DQN) from Mnih et al. (2013) and Proximal Policy Optimization (PPO) from Schulman et al. (2017). DQN is a value-function approximation method which means that it uses a DNN to approximate the optimal state-action value function Q^*. It uses an experience replay buffer (limited to 5000 states by system memory) to store and reuse previous data points to improve sample efficiency. PPO is a policy gradient method which means that is uses a DNN to approximate the mapping $\pi^*(s)$ directly. Implementations of these two algorithms were used from the python module `stable-baselines` Hill et al. (2019).

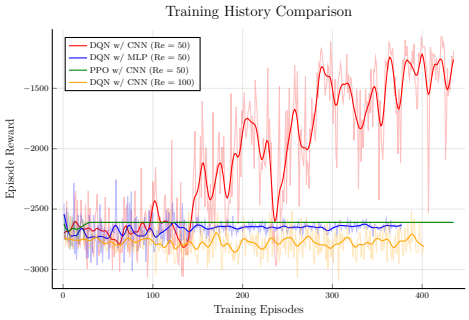


Figure 1: Learning curves of different policy/algorithm combinations

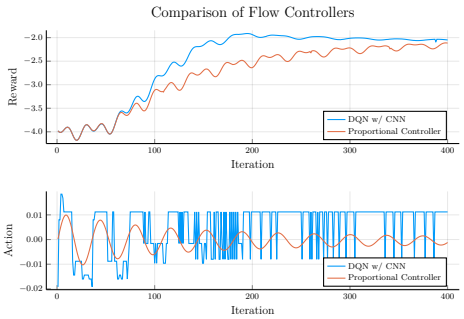


Figure 2: Performance of DQN-trained CNN policy

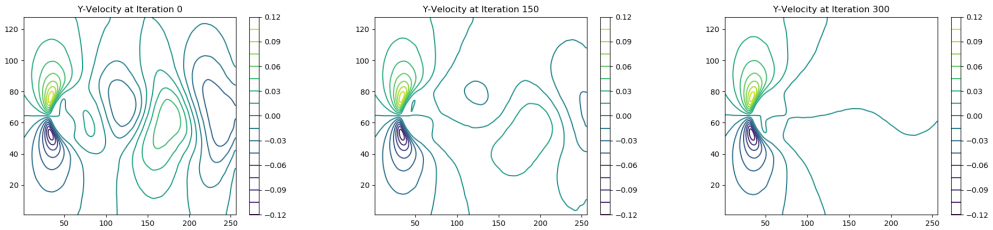
Two different networks will be referenced in the experimentation section. The first is a basic multi-layer perception (MLP) that has two hidden layers, each with 64 units and a ReLU activation for each layer. The second is a convolutional neural network (CNN) that has the same architecture as Mnih et al. (2015) which involves three convolutional layers (32 (8×8) filters with stride 4, 64 (4×4) filters with stride 2, and 64 (3×3) filters with stride 1, respectively). Then a fully connected hidden layer with 512 ReLUs. The output layer is fully connected with an output for each action in the action space (50 outputs were used).

5 Experiments

The goals of the experiments were three fold: 1) Compare different network architectures and algorithms to see which is the most promising for continued research, then 2) find a control policy using the most promising approach that could outperform an existing proportional controller (for $\mathcal{R} = 50$), and 3) analyze that controller to gain insight into the problem of vortex shedding.

To achieve the first goal, we trained two different RL algorithms, tried two different networks, and used two different Reynolds numbers. Experimentation was limited because each training run took between 30 - 48 hours (7 minutes per episode) on a K520 GPU on AWS with the Adam optimization method. The results of training are shown in fig. 1. The first comparison was between DQN and PPO, both with a CNN network. The DQN algorithm started to learn after about 150 episodes and reached high performance after about 300. PPO on the other hand did not learn at all, and actually got stuck applying the same action at each timestep. It is unclear why PPO experienced this problem and it is possible that with additional hyperparameter tuning better performance could be achieved. The next comparison was using DQN to compare the MLP network and the CNN network. The MLP also failed to learn any useful polices in the training time allotted. A CNN might be a better network choice because it requires fewer parameters to extract features in a large state-space. Lastly, we took the best architecture (DQN with CNN network) and applied it to the more difficult task of flow control at a higher Reynolds number ($\mathcal{R} = 100$). In this case, the algorithm was not able to achieve a good policy in the training time allotted. Note that the reward is actually much lower for $\mathcal{R} = 100$ shedding due to the larger difference between the steady and unsteady solutions so 6600 was added to the reward to bring it to a similar scale as the other experiments.

Once the best architecture and algorithm was determined, it could be tested on a flow problem and compared to the proportional controller outlined in Morton et al. (2018). The results of this comparison are shown in fig. 2. In the top graph we can see that the CNN policy was able to suppress the vortex shedding more quickly than the proportional controller, obtaining a larger total reward. In the bottom graph we can compare the control law found by the RL algorithm to the control law of a proportional controller. In the first part of episode (up to iteration 125), both control laws look somewhat similar, especially when you compare the sign of the control input. Once the vortices have mostly been suppressed, the two algorithms had different control strategies for maintaining the suppression. The RL algorithm spends most of its control effort with a positive rotation while the proportional controller continues to oscillate at low amplitude. Snapshots of the y-velocity of the state space are shown in fig. 3 for different iterations during the suppression process.



(a) Iteration 0 (b) Iteration 150 (c) Iteration 300

Figure 3: Vortex shedding suppression at $\mathcal{R} = 50$ using a DQN-trained CNN state-action value function

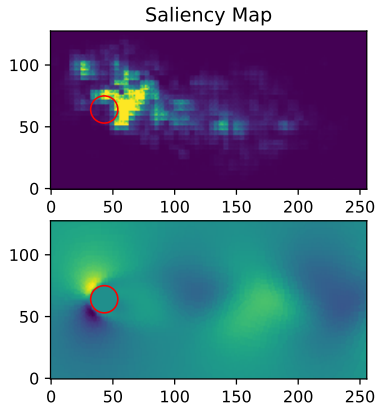


Figure 4: Saliency map for taken action on first iteration

Lastly, to visualize what the network is looking at when making action decisions, a saliency map was generated using the module from dsmilkov et al. (2019). On iteration 0, the flow state is fed to the value network and the gradient was backpropogated from the selected action (ω = -0.02) to the input. The magnitude of the gradient was then plotted in fig. 4 where the red circle shows where the cylinder is located. We can see from the saliency map that the network is primarily focused on the near-wake of the cylinder as the most important region. Regions in front of of the cylinder and outside of the wake are not deemed important at all. Additionally we can observe an asymmetry in the saliency for this action: the network is more dependent on the upper region of the wake than the lower region.

6 Conclusion and Future Work

The results of these experiments demonstrate that it is feasible to using deep reinforcement learning to discover a control policy for vortex suppression behind a circular cylinder via rotation. The best algorithm seems to be DQN with a CNN value network, but further experimentation should be conducted. This method has only been effectively demonstrated at lower Reynolds number and will likely require modifications or longer training times for larger Reynolds numbers.

Future work on this topic could include

- Using a different reward function such as the drag or vorticity of the flow field.
- Using transfer learning from a linearization network Morton et al. (2018) to accelerate the feature extraction process of the learning.
- Replacing the compressible flow solver with an incompressible solver that can run much more quickly.
- Implementing the capabilities of sucking and blowing jets to compare the efficacy of the approaches.

References

Bearman, P. and Branković, M. Experimental studies of passive control of vortex-induced vibration. *European Journal of Mechanics-B/Fluids*, 23(1):9–15, 2004.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

dsmilkov, ruthcfong, nsthorat, and nalinimsingh. saliency. `https://github.com/PAIR-code/saliency`, 2019.

Hill, A., Raffin, A., and Ernestus, M. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2019.

Homescu, C., Navon, I., and Li, Z. Suppression of vortex shedding for flow around a circular cylinder using optimal control. *International Journal for Numerical Methods in Fluids*, 38(1):43–69, 2002. URL `https://web.math.fsu.edu/~aluffi/archive/paper132.pdf`.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Morton, J., Jameson, A., Kochenderfer, M. J., and Witherden, F. Deep dynamical modeling and control of unsteady fluid flows. In *Advances in Neural Information Processing Systems*, pp. 9278–9288, 2018.

Rabault, J., Reglade, U., Cerardi, N., Kuchta, M., and Jensen, A. Deep reinforcement learning achieves flow control of the 2d karman vortex street. *arXiv preprint arXiv:1808.10754*, 2018.

Rabault, J., Kuchta, M., Jensen, A., Réglade, U., and Cerardi, N. Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. *Journal of Fluid Mechanics*, 865:281–302, 2019.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Witherden, F. D., Farrington, A. M., and Vincent, P. E. Pyfr: An open source framework for solving advection–diffusion type problems on streaming architectures using the flux reconstruction approach. *Computer Physics Communications*, 185(11):3028–3040, 2014.