Deep Reinforcement Learning for Active Flow Control - state of the art and perspectives

J. Rabault
with many colleagues over the years!

2023-03-01

Motivation: active flow control

- Flow control: a relevant theoretical and industrial problem.
- Time dependence, nonlinearity, high dimensionality break tools.
- Methods based on linearization often do not work very well.
- Field of active research for a long time, but famously difficult.

Try to introduce new tools? Recent successes of Deep Reinforcement Learning (DRL) on complex problems is a strong hint (Go, Poker, robotics, control of complex cooling systems).

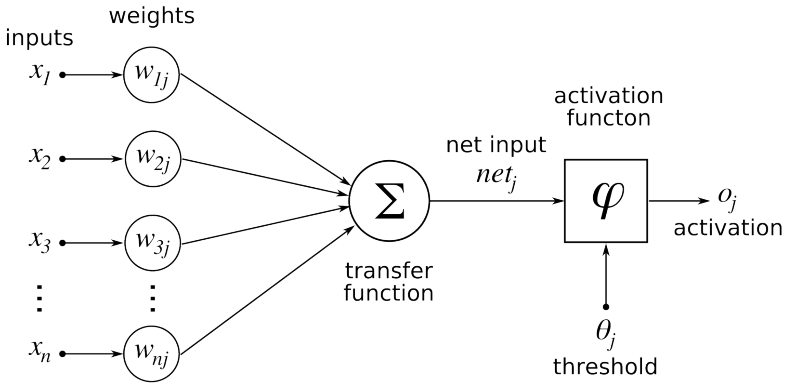
Agenda

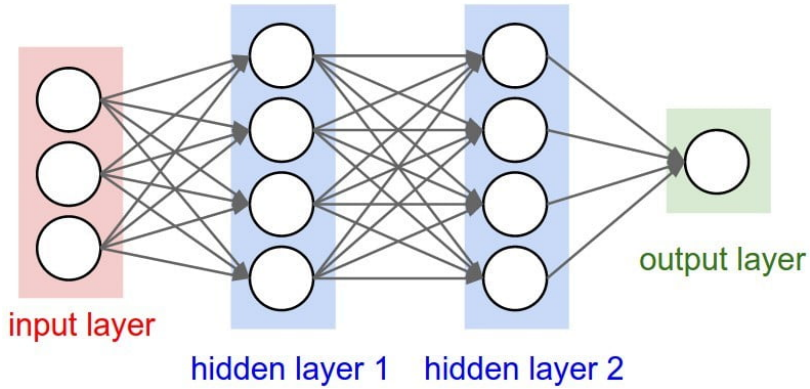
- Short introduction to ANNs and Supervised Learning.
- From Supervised to Reinforcement Learning.
- DRL for Active Flow Control: state of the art and perspectives.

A short reminder about Artificial Neural Networks and Supervised Learning

Deep ANNs: I

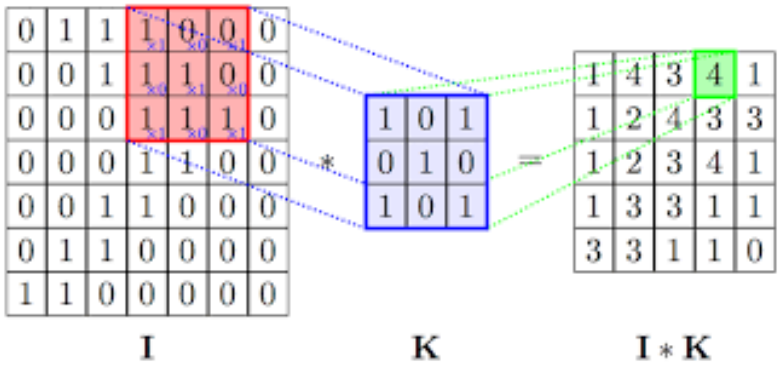
Artificial Neural Networks are simply a set of neurons.





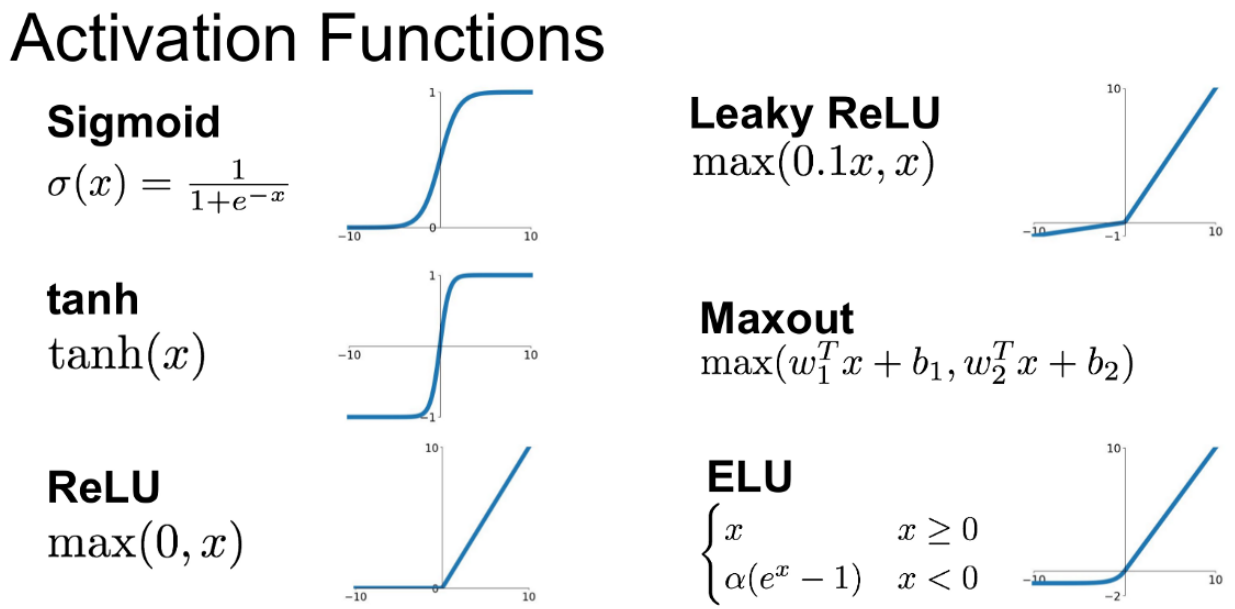
Deep ANNs: ANNs with several hidden layers. Feed one layer in the next: 'Deep Learning', *LeCun et. al.*, Nature (2015).

Convolution layers enforce invariance by translation:



Deep ANNs: II

What activation function to use?



- Linear activation function: boring: only linear network.
- Non linear functions: rich: in practice (leaky) ReLU often best.

Deep ANN + non linear activation function = universal approximator. 'Multilayer feedforward networks are universal approximators', *Hornik et. al.*, Neural Networks (1989). Universal mapping, functional space density.

Deep ANNs: III

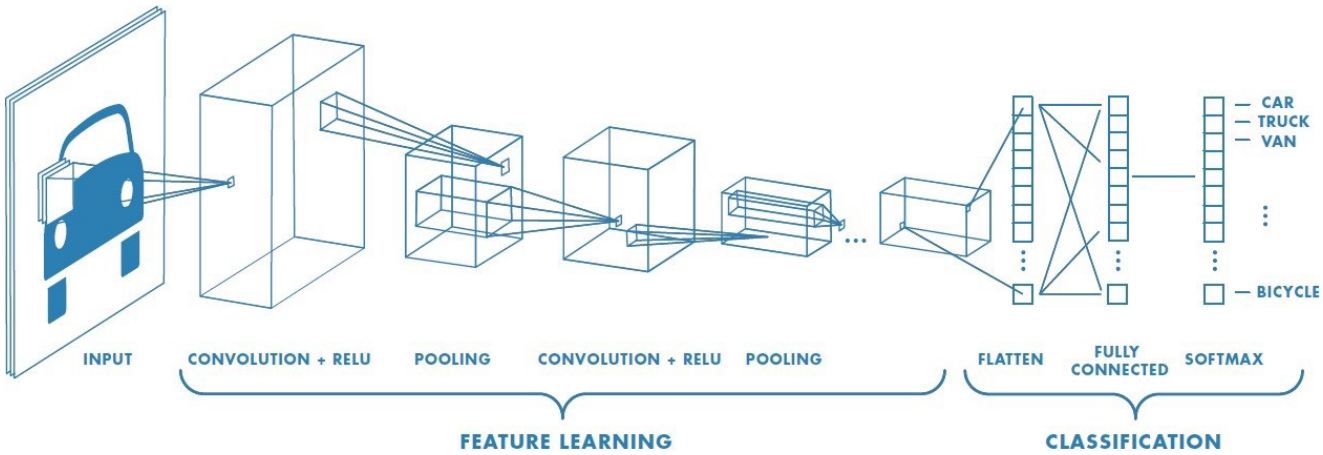
What architecture to use?

- In theory, 1-layer is enough to be universal approximator.
- In practice, advantage of depth, invariance by translation, etc.

Example of XOR function, N inputs:

- Need order 2^N neurons if 1 layer.
- Need order N neurons if $log(N)$ layers.

Example of CNNs for image analysis: re-use the weights:

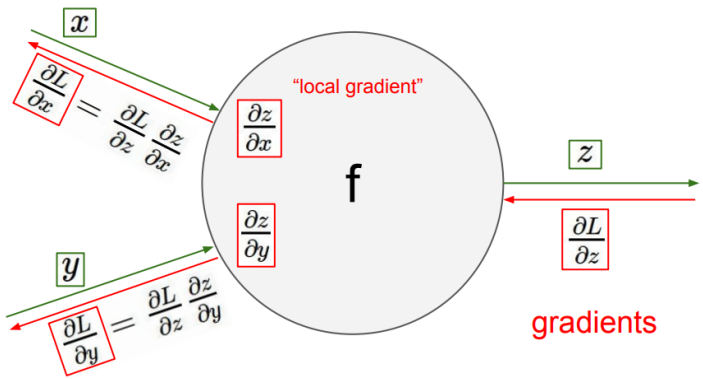


Deep ANNs: IV

Universal approximator, easily parallelizable, effective representation

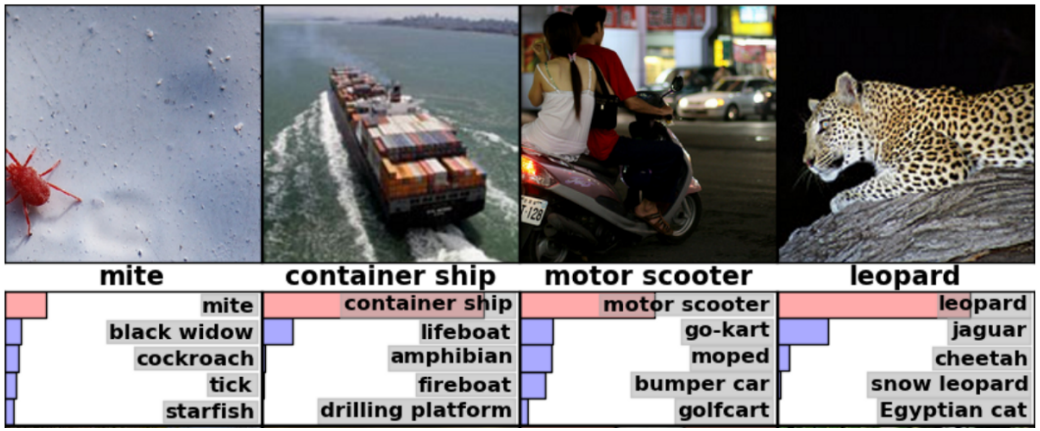
How to train?

Gradient descent, back propagation of errors: 'Applications of advances in nonlinear sensitivity analysis', *Werbos*, System modeling and optimization (1982).



- Feed data in, compute error.
- Back propagate (Leibniz) the error gradient.
- Update the coefficients.

Supervised learning: describe a function from examples



Supervised learning, image analysis work 'quite' well.

Yoshua Bengio

FOLLOW

Professor of computer science, University of Montreal, Mila, IVADO, CIFAR
Verified email at umontreal.ca - Homepage

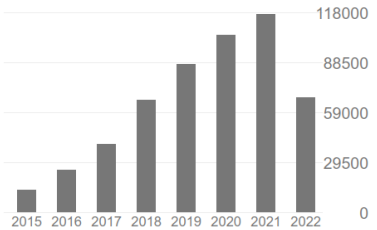
Machine learning deep learning artificial intelligence

GET MY OWN PROFILE

TITLE	CITED BY	YEAR
Deep learning Y LeCun, Y Bengio, G Hinton nature 521 (7553), 436-444	54460	2015
Generative adversarial nets I Goodfellow, J Pouget-Abadie, M Mirza, B Xu, D Warde-Farley, S Ozair, ... Advances in neural information processing systems 27	48154	2014
Gradient-based learning applied to document recognition Y LeCun, L Bottou, Y Bengio, P Haffner Proceedings of the IEEE 86 (11), 2278-2324	47809	1998
Deep learning I Goodfellow, Y Bengio, A Courville MIT press	41823	2016

Cited by VIEW ALL

	All	Since 2017
Citations	555645	486647
h-index	207	188
i10-index	733	654

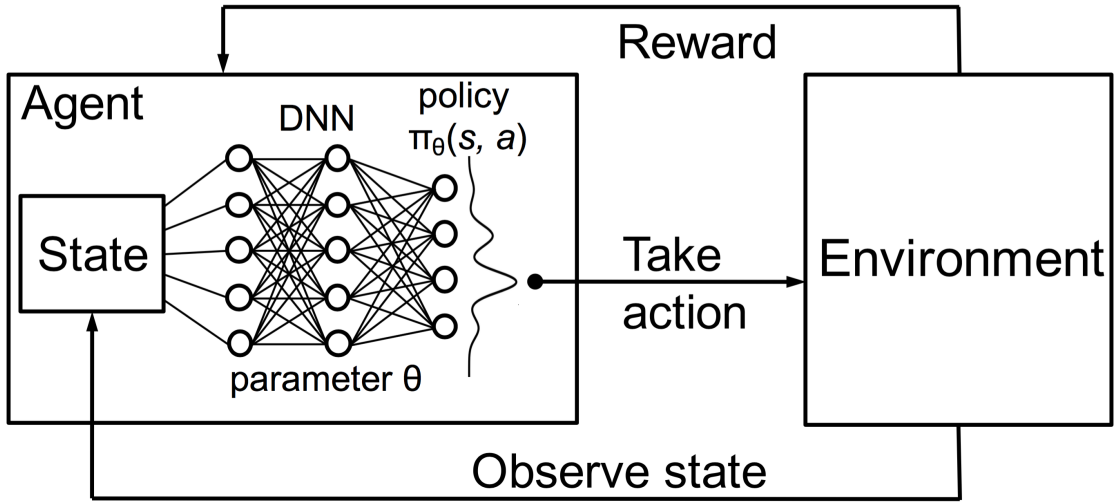


From Supervised to Reinforcement Learning

Reinforcement Learning

How to learn when no solution is known? Learn by trial and error.

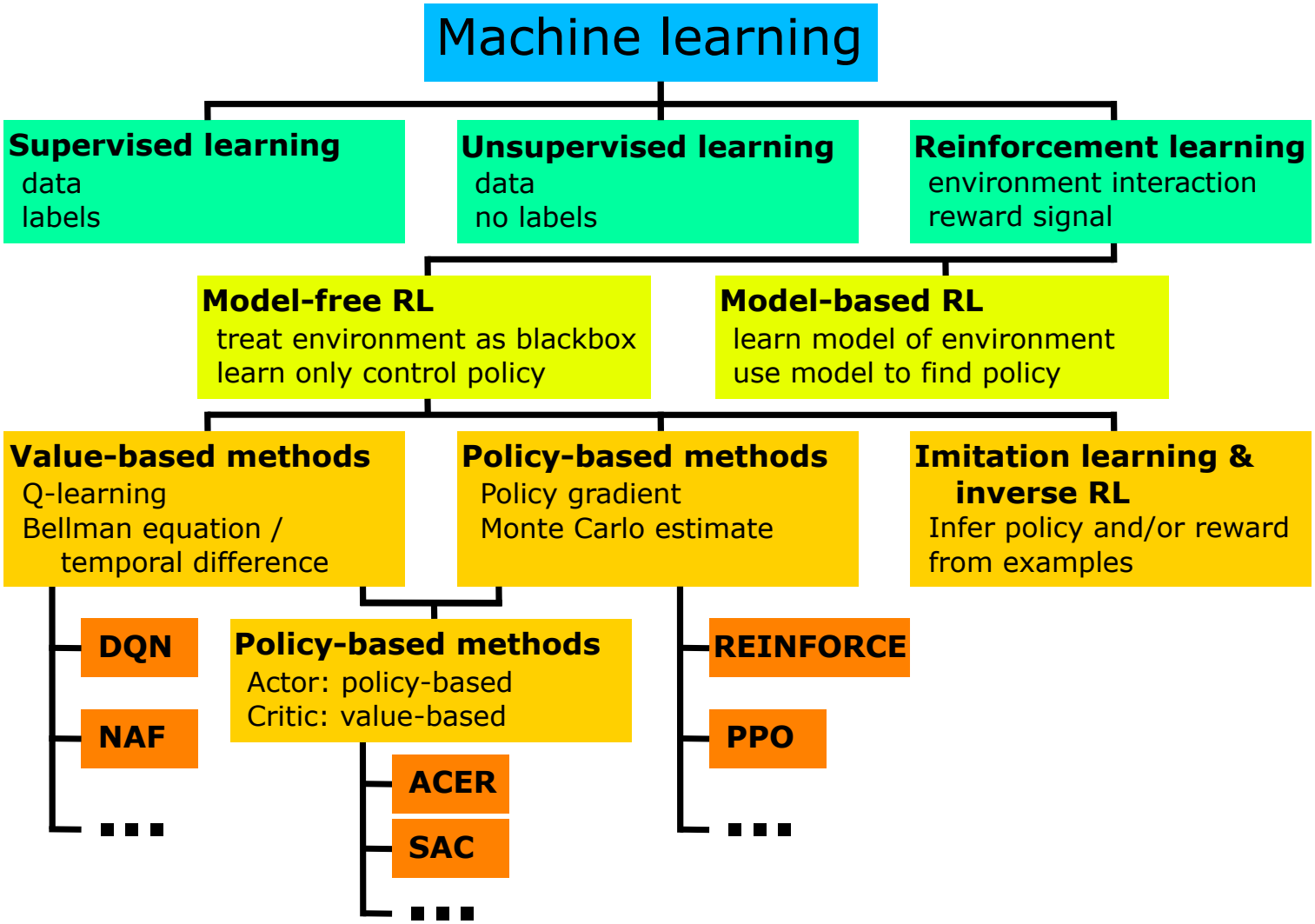
'A review on Deep Reinforcement Learning for Fluid Mechanics', Garnier et. al., Computers and Fluids (2021).



s the state, a action, r reward, policy $\pi(a|s)$ the probability of a in s.

Several methods. Main ones (though boundary is sometimes unclear):

- Q-learning.
- Policy gradients.



See J. Rabault and A. Kuhnle, 'Deep reinforcement learning applied to active flow control' in "Data Driven Fluid Mechanics", 2023, Cambridge University Press. Still a domain of active research...

Q-learning

Q-Learning (I)

'Human-level control through deep reinforcement learning', *Mnih et. al.*, Nature (2015).

Based on older work about Q-Learning:
'Q-learning', *Watkins et. al.*, Machine Learning (1992).

Discounted future reward: $R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... = r_t + \gamma R_{t+1}$.

Given a state s and a policy π define the value function:

$$V^\pi(s) = \mathbb{E}\left(\sum_{t\geq 0} \gamma^t r_t | s, \pi\right),$$

and the optimal value function: $V^*(s) = \max_\pi V^\pi(s)$.

Q-Learning (II)

Introduce the action of the agent: expected return, starting in state s, performing action a, continuing with policy π: $Q^{\pi}(s, a)$.

Then: $Q^* = \max_{\pi} Q^{\pi}(s, a)$, and $V^*(s) = \max_a Q^*(s, a)$. Finally:

$$\pi^*(s) = \arg\max_a Q^*(s, a).$$

Therefore, knowledge of Q^* is knowledge of π^*. Q learning: iterative approximation of Q^* using the Bellman equation:

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

Random action at probability ϵ, arg max action at probability $1 - \epsilon$, and:

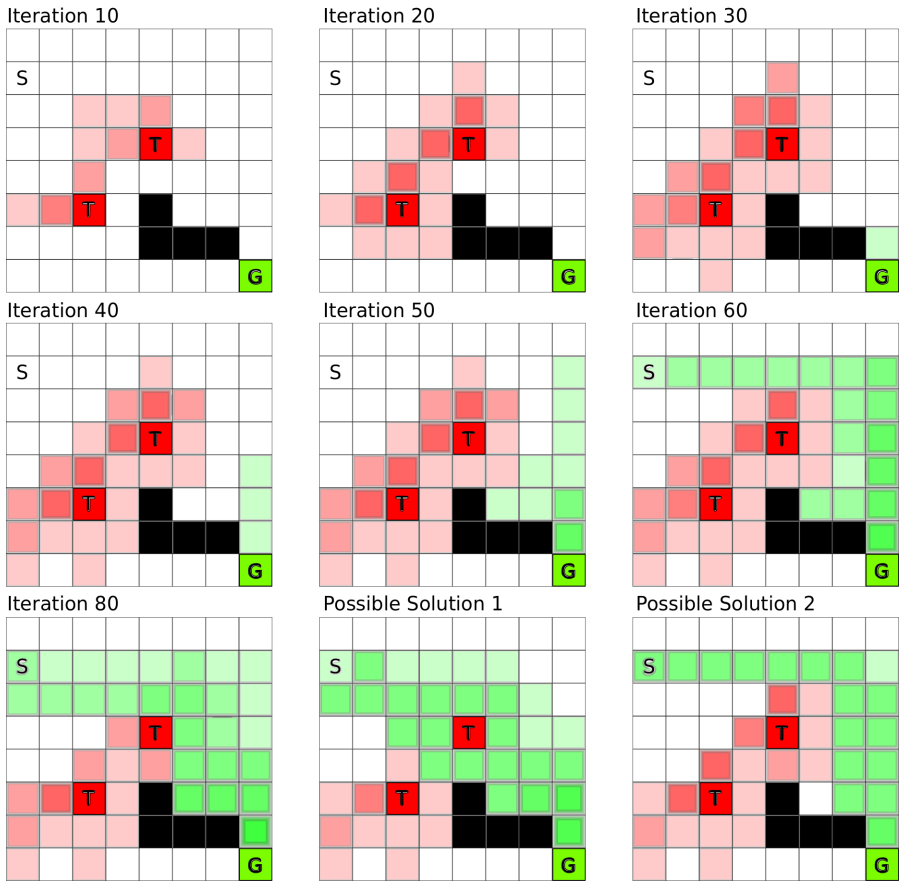
$$Q_{n+1}(s_t, a_t) = Q_n(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q_n(s_{t+1}, a) - Q_n(s_t, a_t)).$$

This kind of techniques also known as 'dynamic programming': break a problem in succession of small sub-problems.

The 'maze' illustration

Good way of thinking about Deep Q Learning: investigate a maze

- For each location, what is best possible output for each action.
- 'Good' probabilities diffuse.



Deep Q-Learning

World is non-linear, high dimensionality, complex...

... use a Deep Neural Network for Q. Therefore 'Deep Q-Learning'.

'Mastering the game of Go without human knowledge', *Silver et. al.*, Nature (2015).

- Parametrize $Q(s, a, \Theta_i)$ using a deep ANN (Θ_i network weights).
- max instead argmax: give the Q-value for each a in different output.
- Loss function: $L_i(\Theta_i) = \mathbb{E}\left[(r + \gamma \max_{a'} Q(s', a', \Theta_i) - Q(s, a, \Theta_i))^2\right]$
- Experience replay buffer inspired from biology.
- Update target value at end of training for stability.

Computers that learn without us knowing a solution. Adapted to non-linear, high dimensional, complex problems. Can make DANNs fight each other (adversarial training). ANN just one solution (but works well).

Policy Gradient method

Policy Gradient methods

Policy Gradient: directly learn the policy π.

policy $\pi(a|s)$: describes the probability of taking action a when in state s.

stochastic policy: π distribution for each state, parametrized by ANN:

- Smooths out actions.
- Train by slightly shifting distribution.
- ANN (weights Θ_i) decides the parameters of a parametric distribution.

Objective: find Θ such that: $\max_\Theta \mathbb{E}\left[\sum_{t=0}^{H} R(s_t)|\pi_\theta\right]$, $R(s_t) = r_t\gamma^t$

Computing the policy gradient

τ: s-a sequence: $(s_0, a_0), (s_1, a_1), \ldots, (s_H, a_H)$, $R(\tau) = \sum_{t=0}^{H} R(s_t, a_t)$

$$\text{Value: } V(\Theta) = \mathbb{E}\left[\sum_{t=0}^{H} R(s_t, a_t)|\pi_\theta\right] = \sum_\tau \mathbb{P}(\tau, \Theta) R(\tau)$$

$$\begin{aligned}
\nabla_\Theta V(\Theta) &= \sum_\tau \nabla_\Theta \mathbb{P}(\tau, \Theta) R(\tau) \\
&= \sum_\tau \frac{\mathbb{P}(\tau, \Theta)}{\mathbb{P}(\tau, \Theta)} \nabla_\Theta \mathbb{P}(\tau, \Theta) R(\tau) \\
&= \sum_\tau \mathbb{P}(\tau, \Theta) \nabla_\Theta \log\left(\mathbb{P}(\tau, \Theta)\right) R(\tau)
\end{aligned}$$

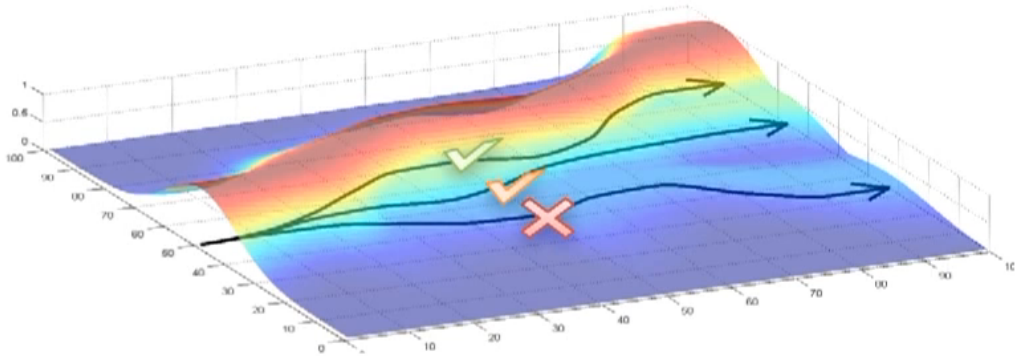
Expected value, approximate with empirical sampling under policy π_θ:

$$\nabla_\Theta V(\Theta) \approx \hat{g} = \frac{1}{M} \sum_{i=1}^{M} \nabla_\Theta \log\left(\mathbb{P}(\tau^{(i)}, \Theta)\right) R(\tau^{(i)})$$

Understanding the policy gradient

$$\nabla_\Theta V(\Theta) \approx \hat{g} = \frac{1}{M} \sum_{i=1}^{M} \nabla_\Theta \log \left(\mathbb{P}(\tau^{(i)}, \Theta) \right) R(\tau^{(i)})$$

Works by increasing the probability of paths with good R (and reducing the probability of paths with bad R).



Computing the gradient of the log probability

Express the log prob as a dynamics model and a policy part:

$$\nabla_\Theta \log\left(\mathbb{P}(\tau^{(i)}, \Theta)\right) = \nabla_\Theta \log\left[\prod_{t=0}^{H} \mathbb{P}(s_{t+1}^{(i)}|s_t^{(i)}, a_t^{(i)})\pi_\theta(a_t^{(i)}|s_t^{(i)})\right]$$

$$= \nabla_\Theta \left[\sum_{t=0}^{H} \log \mathbb{P}(s_{t+1}^{(i)}|s_t^{(i)}, a_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\right]$$

$$= \nabla_\Theta \sum_{t=0}^{H} \log \pi_\theta(a_t^{(i)}|s_t^{(i)})$$

The log prob gradient depends only on the policy. No dynamics model. This method is unbiased.

Algorithm: $\nabla_\Theta V(\Theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{M} \nabla_\Theta \log\left(\mathbb{P}(\tau^{(i)}, \Theta)\right) R(\tau^{(i)})$,
with $\nabla_\Theta \log\left(\mathbb{P}(\tau^{(i)}, \Theta)\right) = \nabla_\Theta \sum_{t=0}^{H} \log \pi_\theta(a_t^{(i)}|s_t^{(i)})$.

Proximal Policy Optimization (PPO)

Among the state of the art gradient based methods:

'Proximal Policy Optimization Algorithms', *Schulman et. al.*, ArXiv (2017).

Implement a few tricks:

- Critic network: learn (noisy) actualized reward with a separate ANN relying on the Bellman equation (i.e. some Q-learning in combination with the policy method...).
- Clip gradient to discourage unsafe policy changes.

Quite complex (and tricky) to implement...

Open Source frameworks

Several Open Source frameworks. Among others:

- Tensorforce, `https://github.com/tensorforce/tensorforce`.
- stable-baselines, `https://github.com/hill-a/stable-baselines`.
- Tianshou, `https://github.com/thu-ml/tianshou`.

Workflow:

- Build your application-specific environment by filling in a class.
- Import available built-in Agent (PPO or other).
- Train.
- Evaluate in deterministic mode.

DRL + fluid mechanics frameworks: DRLinFluids:
`https://github.com/venturi123/DRLinFluids`

Training vs. deterministic runner

Training:

- Generate pdf of action following current ANN policy.
- Randomly sample from the distribution

Therefore, perform exploration, and do not follow the policy believed to be optimal. Visible as exploration noise in control strategy.

Deterministic runner:

- Idem: generate pdf of action following trained ANN policy.
- Pick as action the one 'most likely' to be optimal, i.e. maximum of the distribution.

Can provide smooth policy.

Deep Reinforcement Learning for Active Flow Control

DRL for turbulence control and active flow control?

- DRL is the natural candidate for AFC and turbulence control: need to both find and express a solution.

Several works suggested in 2019 that it is a promising approach:

'Artificial Neural Networks trained through Deep Reinforcement Learning discover control strategies for active flow control', Rabault et. al., JFM (2019).

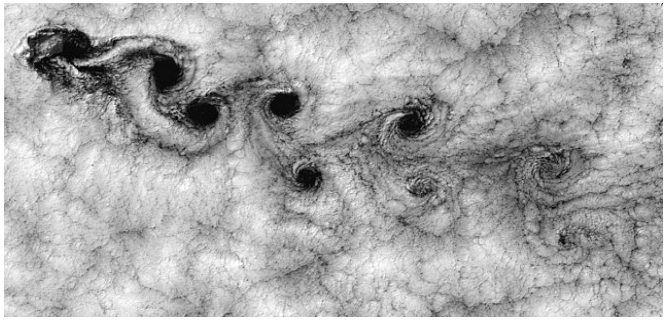
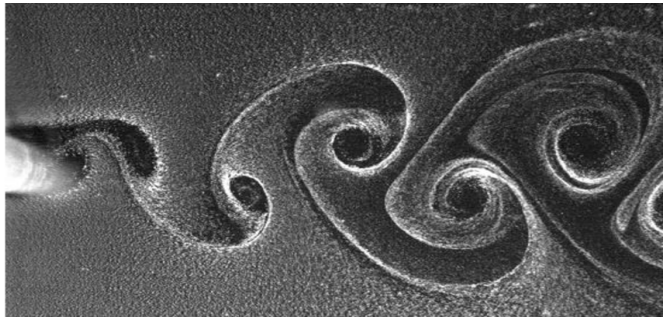
'Accelerating Deep Reinforcement Leaning strategies of Flow Control through a multi-environment approach', Rabault and Kuhnle, PoF (2019).

'Control of chaotic systems by Deep Reinforcement Learning', Bucci et. al., PoRS-A (2019).

'Reinforcement learning versus linear control of Rayleigh Benard convection', G. Beintema et. al., ETC2019.

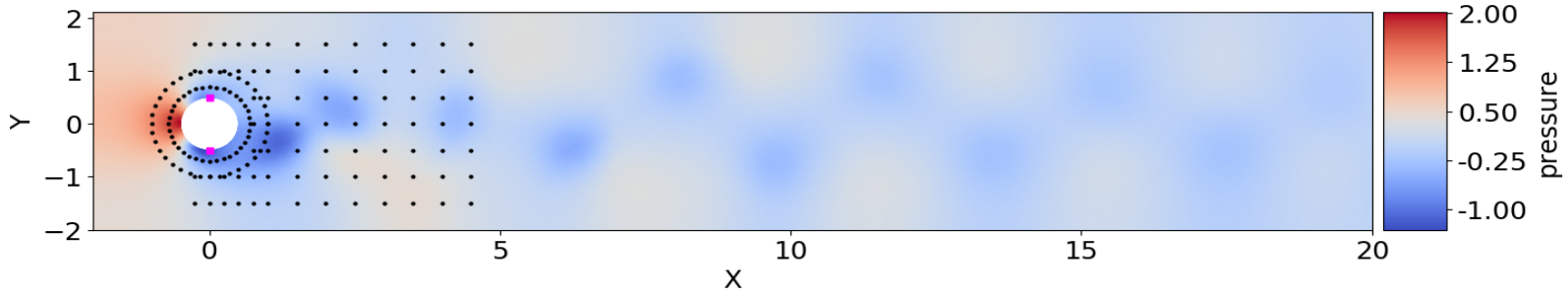
Karman vortex street and the Turek benchmark

Karman vortex street is emblematic of Fluid Mechanics.

Thereafter: 2D FEniCS at $Re = \bar{U}L/\nu = 100$: laminar unsteady, as in:

'Benchmark Computations of Laminar Flow Around a Cylinder', *Schäfer and Turek*, Flow Sims. with High-Perf. Computers (1996).



Velocity probes (s_t), normal jets (a_t), want reduce $C_D = \frac{D}{1/2\rho U^2 D_C}$ (r_t).

Re 100

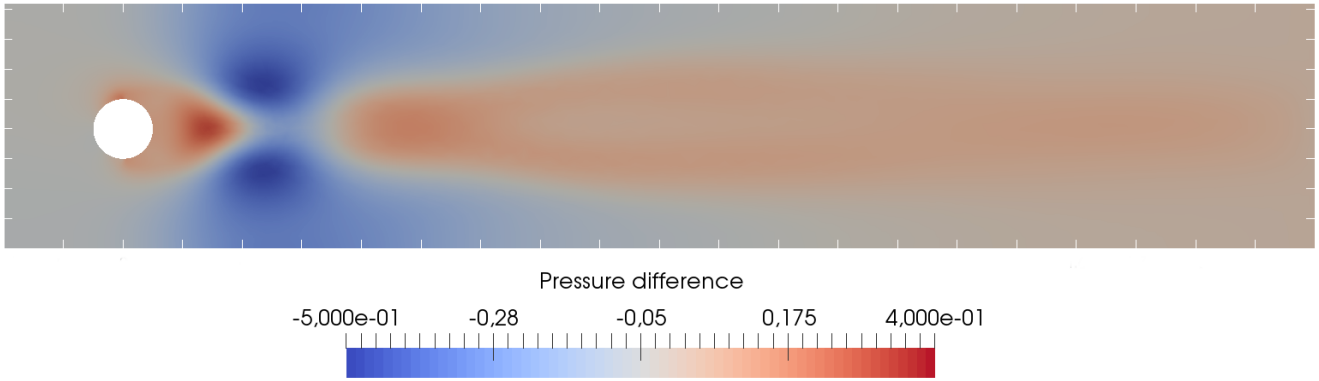
'Artificial Neural Networks trained through Deep Reinforcement Learning discover control strategies for active flow control', Rabault et. al., JFM (2019).

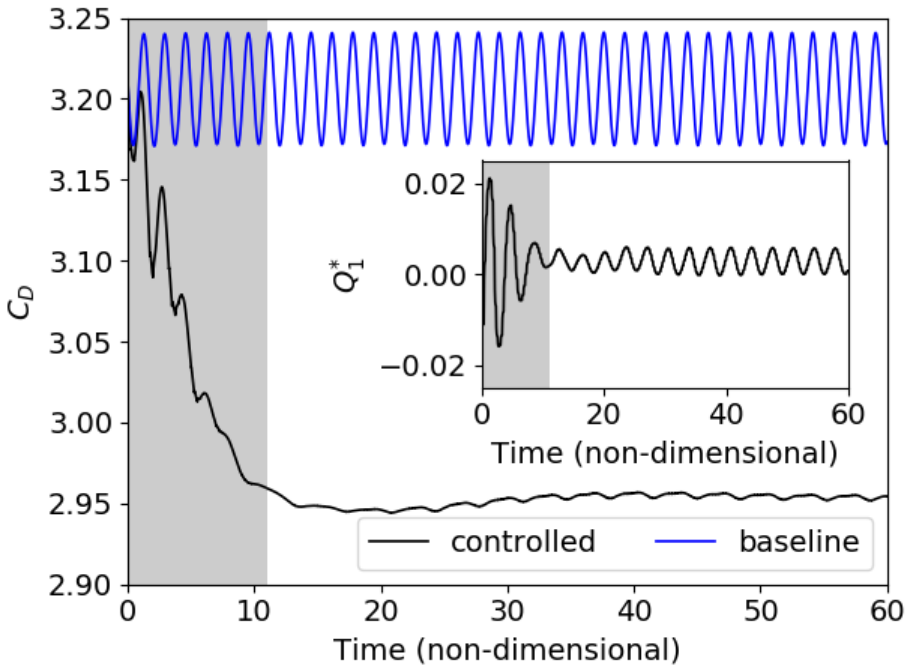
https://youtu.be/0QbKk5SzMws

Full code on github:
`https://github.com/jerabaul29/Cylinder2DFlowControlDRL`

What is happening?

Small jets allow for 'boat tailing'.





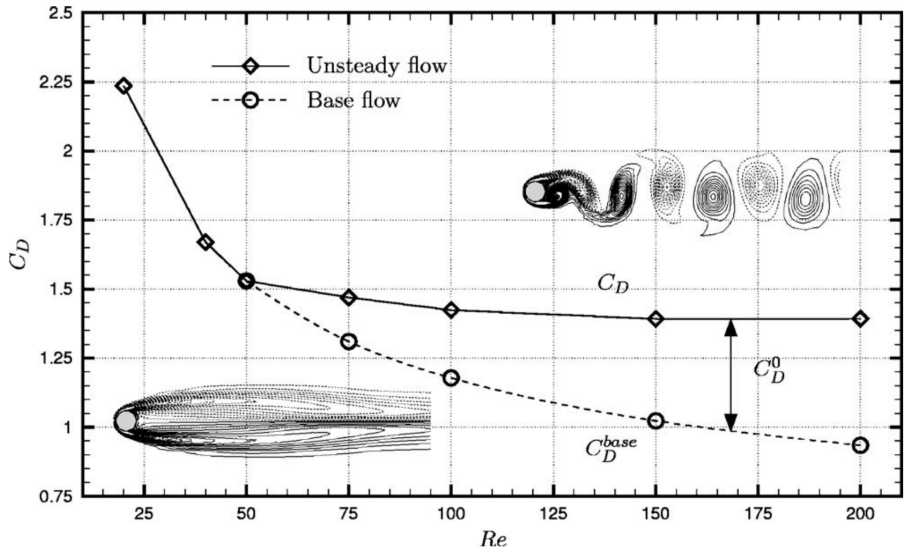
How good is the strategy? (I)

Get about 8% drag reduction. How good is that?

Drag is due to the 'base flow' and 'vortex shedding' components:

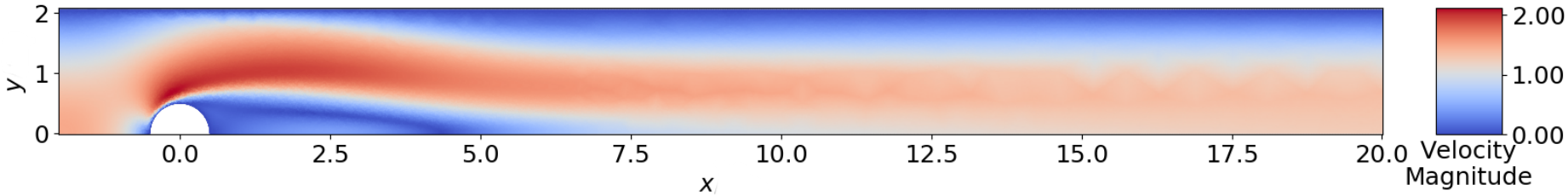
$$C_D = C_D^{baseflow} + C_D^{shedding}$$

Control can only affect $C_D^{shedding}$ ("Optimal rotary control of the cylinder wake using proper orthogonal decomposition reduced-order model", Bergmann et. al., PoF, 2005):



How good is the strategy? (II)

Estimate $C_D^{baseflow}$ in our case from an axisymmetric simulation (symmetric BC on lower border):



$2C_D^{half-domain} = 2.93, \quad < C_D^{nocontrol} >= 3.205, \quad < C_D^{control} >= 2.95.$

Suppressed 93 % of the vortex-shedding-induced drag.

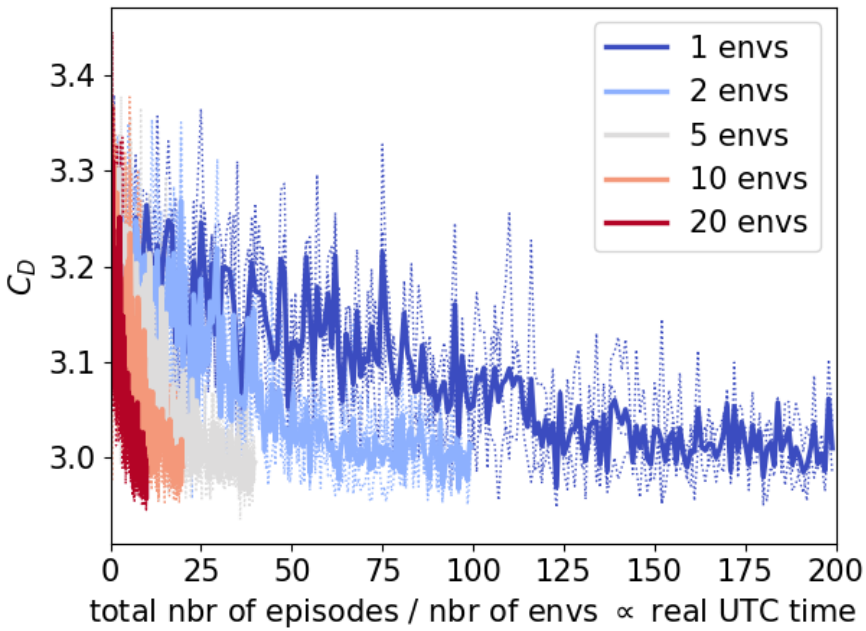
How to progress further?

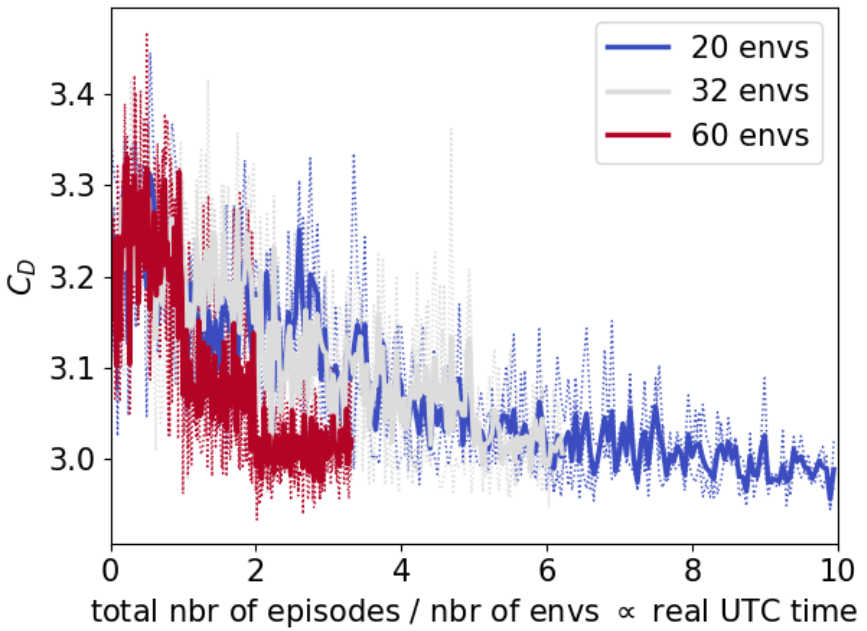
- Go up in complexity / Re.
- Increase the dimensionality of the control space.
- Look at more challenging configurations.

Towards higher Re: faster learning

Parallelize collection of trajectories in the phase space during training.

'Accelerating Deep Reinforcement Leaning strategies of Flow Control through a multi-environment approach', Rabault and Kuhnle, PoF (2019).



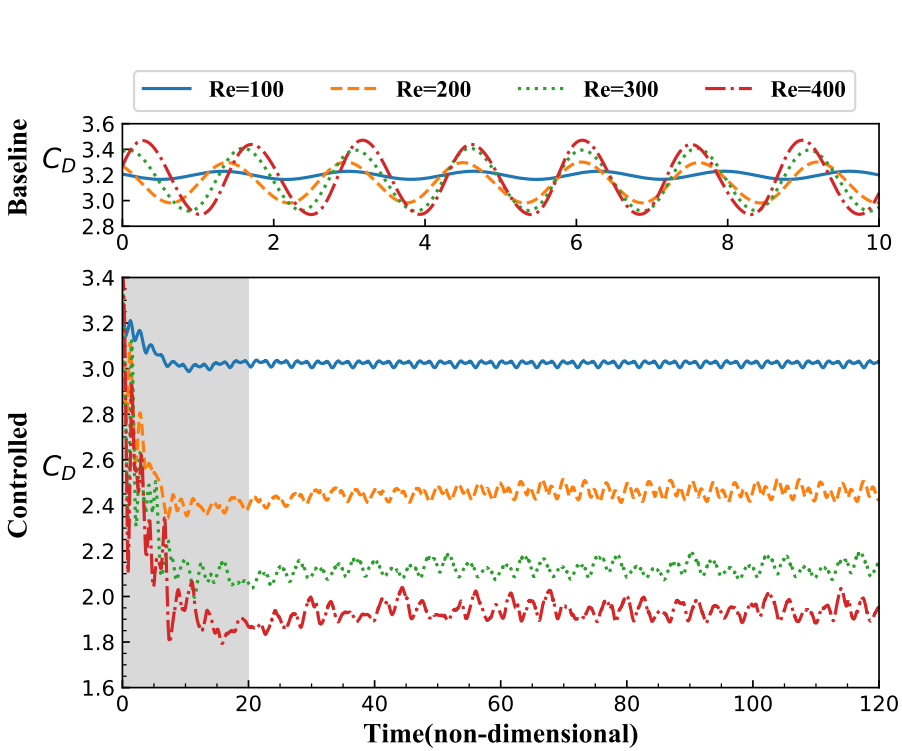


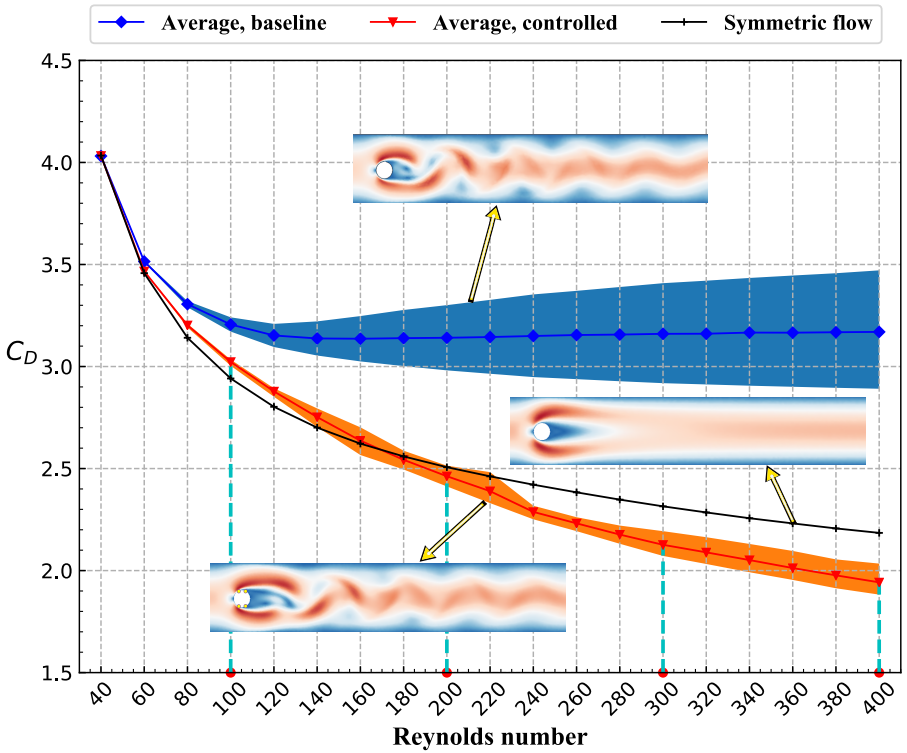
Speedup x60. Code fully available on github:

`https://github.com/jerabaul29/Cylinder2DFlowControlDRLParallel`

Learning 'global' control policies

'Robust active flow control over a range of Reynolds numbers using an artificial neural network trained through deep reinforcement learning', Tang et. al., PoF (2020).





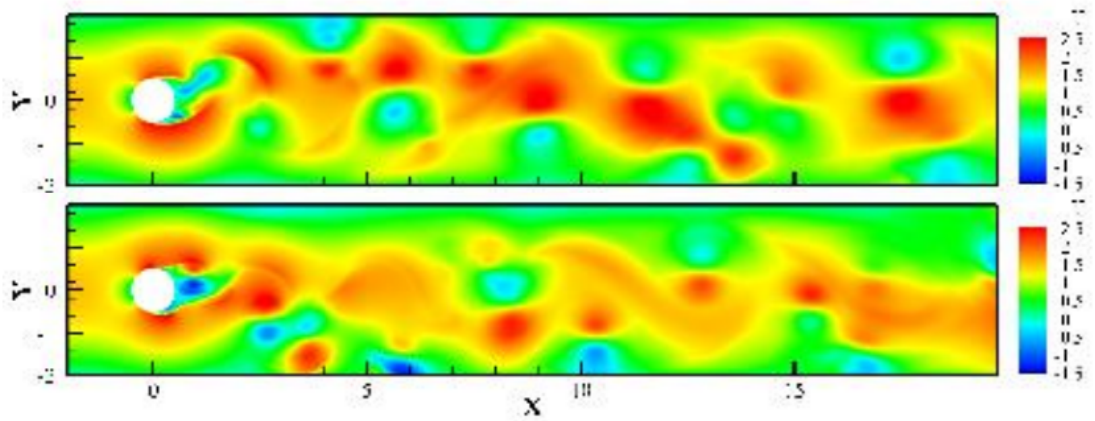
1 ANN, control at all Res within training range efficiently.

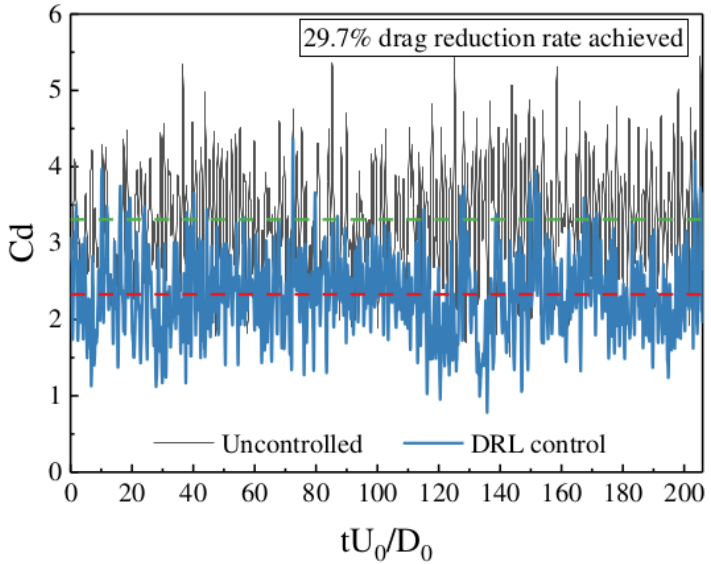
`https://github.com/thw1021/Cylinder2DFlowControlGeneral`

Pushing for higher Res and 3D simulations

Combine parallelism at simulation level and faster CFD using the LBM method on GPUs:
'Applying deep reinforcement learning to active flow control in weakly turbulent conditions', Ren et. al., Physics of Fluids (2021).

- Reproduce (much quicker) Re 100 results.
- Extend to higher Re: 1000 (but in 2D).
- Strongly pseudo-chaotic conditions.
- Get around 30% drag reduction in average.





Different physics at higher Re: energizing the BL

"Deep Reinforcement Learning for Flow Control Exploits Different Physics for Increasing Reynolds Number Regimes", Varela et. al., Actuators 2022.

https://youtu.be/8R3adCQmeEA

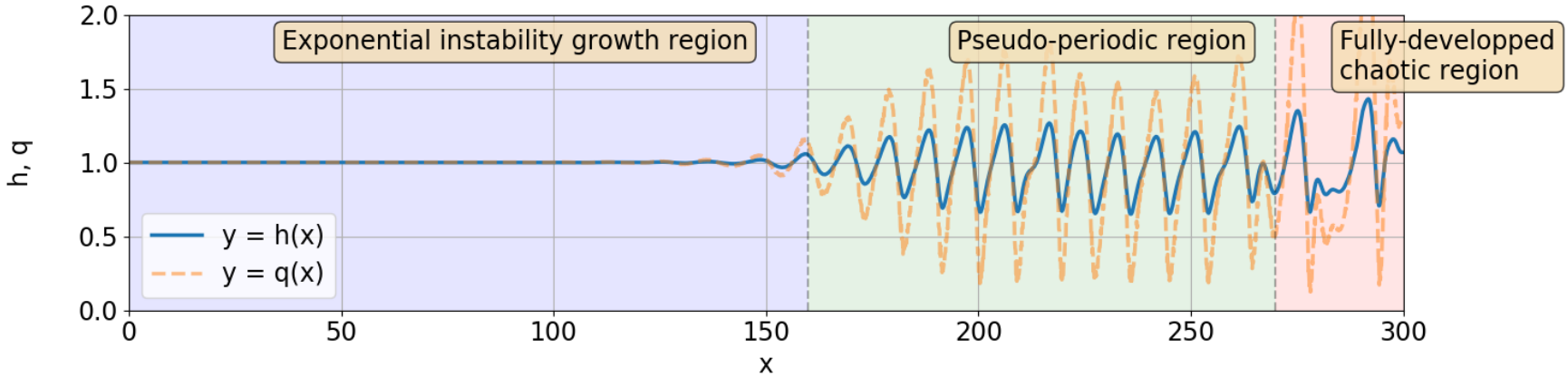
At higher Re (2D, Re 2000), the DRL agent reflects the change of physics: energize the BL instead of doing wake opposition control.

Falling fluid film: how to control a multiple input-output system?

"Exploiting locality and physical invariants to design effective Deep Reinforcement Learning control of the unstable falling liquid film", Belus et. al., AIP-A (2019).

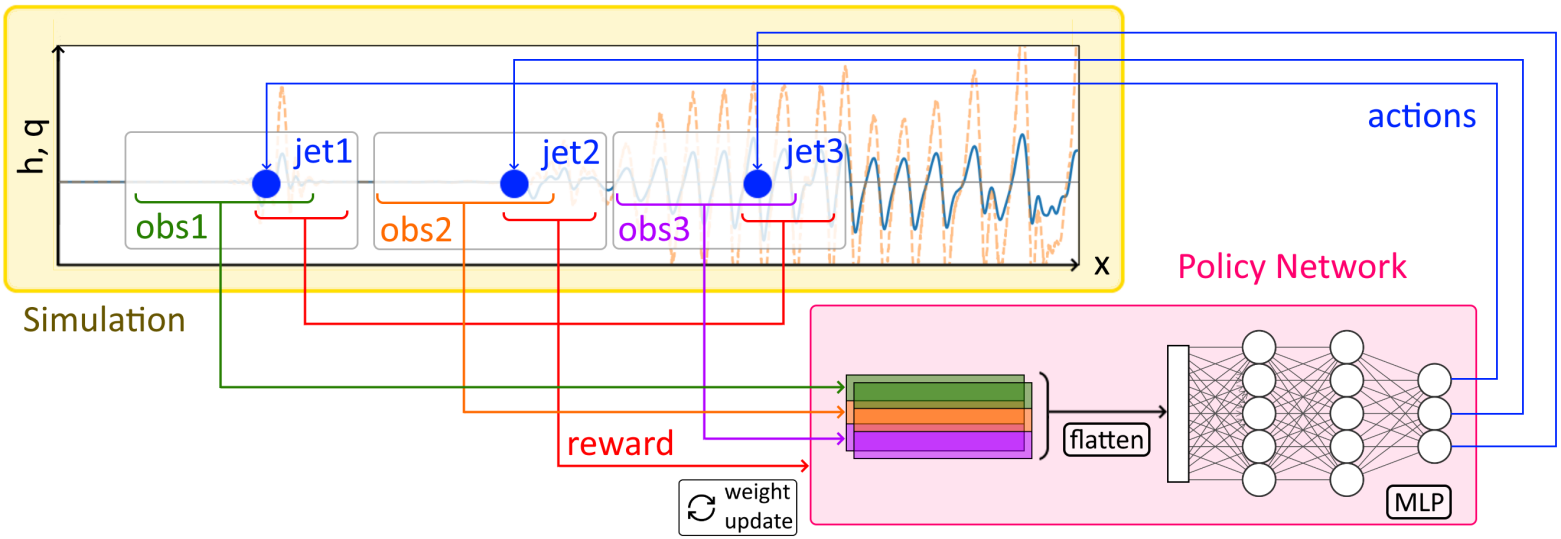
$$\frac{\partial h}{\partial t}+\frac{\partial q}{\partial x}=0,\frac{\partial q}{\partial t}+\frac{6}{5}\frac{\partial}{\partial x}\left(\frac{q^2}{h}\right)=\frac{1}{5\delta}\left(h\frac{\partial^3 h}{\partial x^3}+h-\frac{q}{h^2}\right).$$

`https://github.com/jerabaul29/docker_image_falling_fluid_film`



How to insert several jets? (M1)

Naive approach: large inputs and outputs.



Curse of dimensionality on the number of jets:

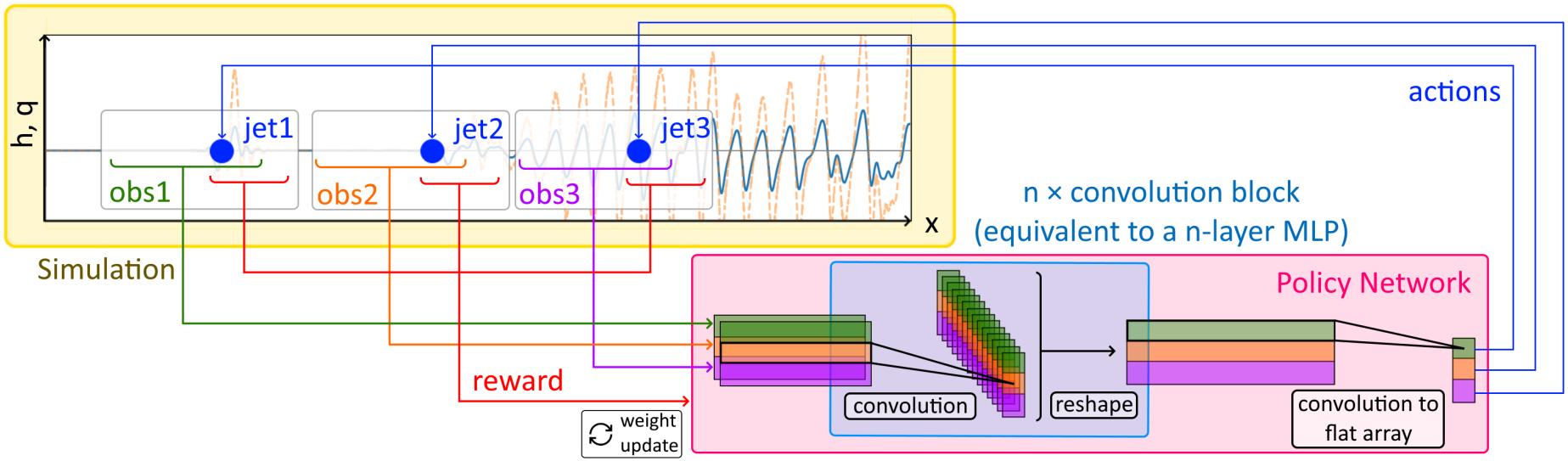
- For thought experiment, p possible jet strengths.
- Use N jets, and a fully connected ANN.
- Then, need to fully explore:

$$C = p^N \text{ combinations...}$$

'Of course', we can do much better.

How to insert several jets? (M2)

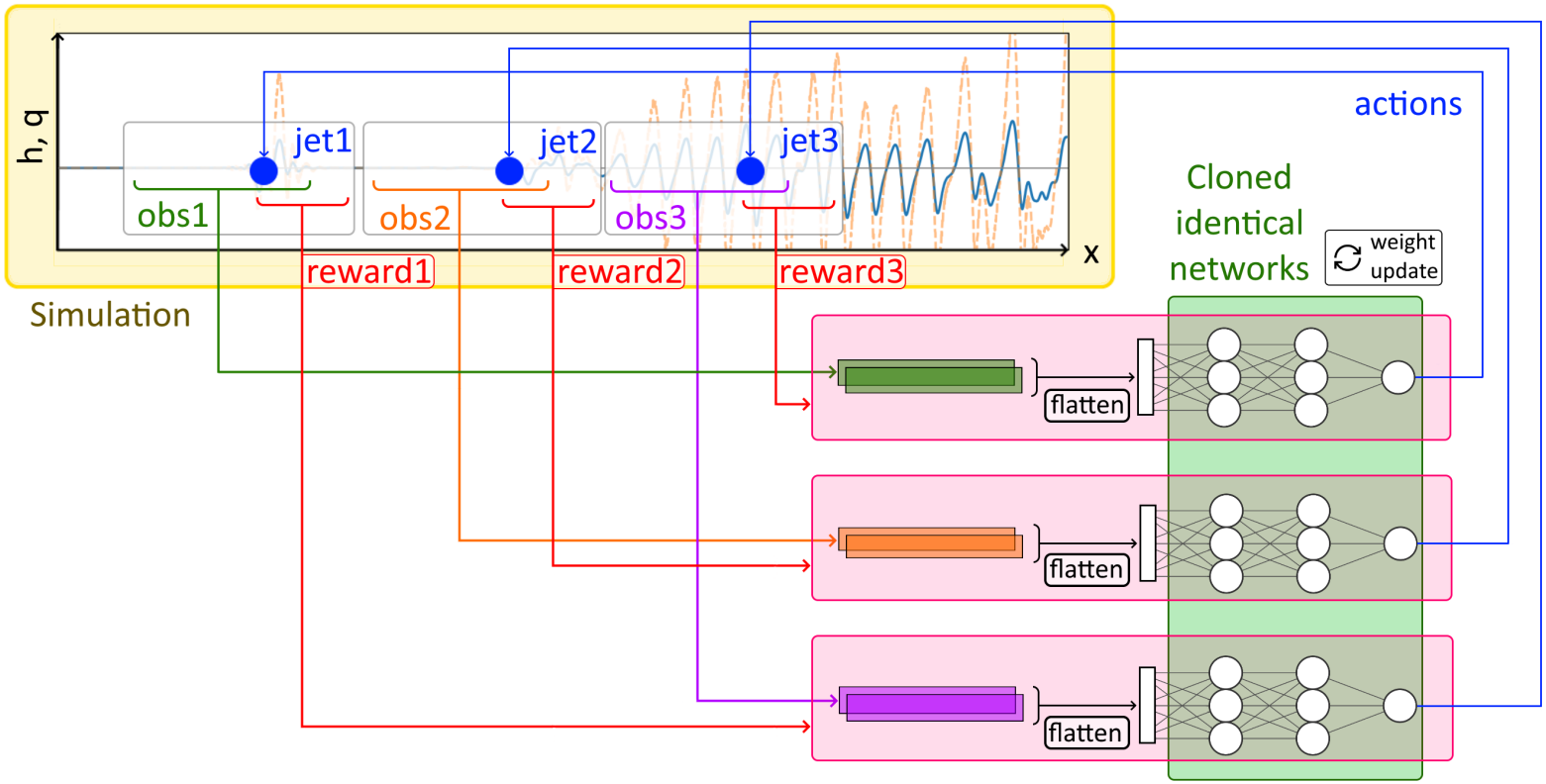
Smarter approach: use a fully convolutional network.
Respect invariance by translation.



This way, truly learn only 1 set of weights. Take advantage of translational invariance.

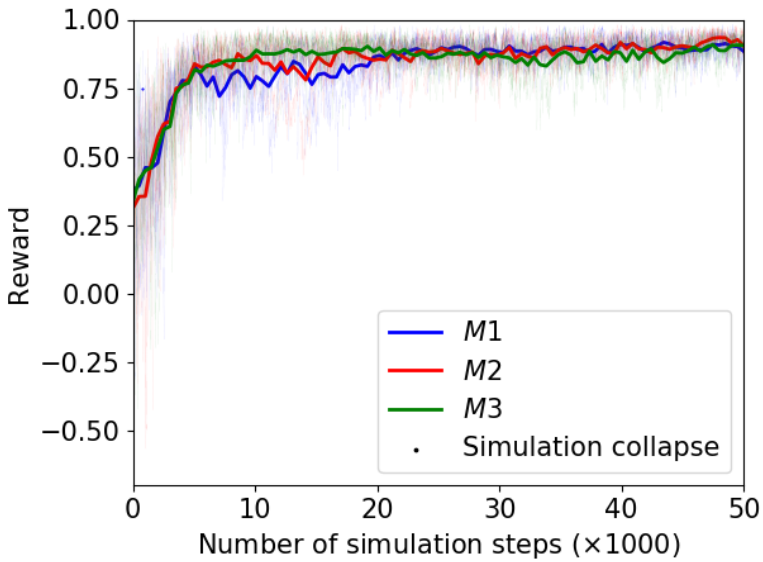
How to insert several jets? (M3)

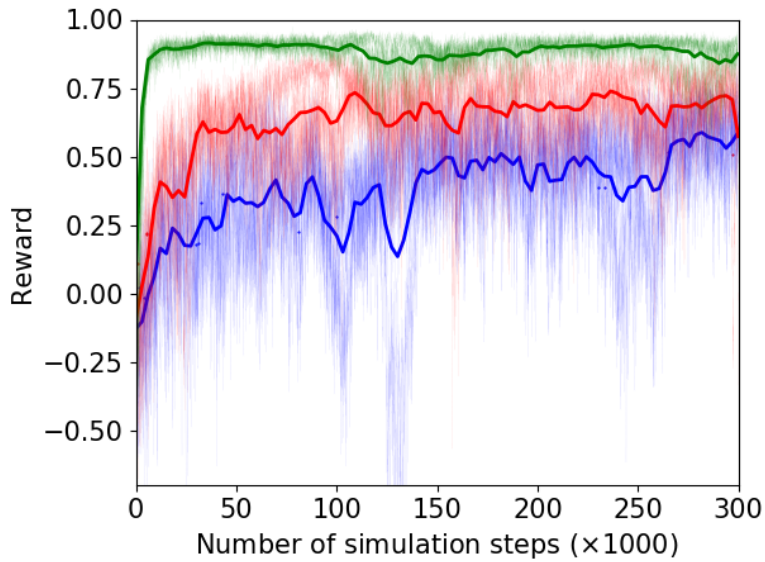
Even better: split the simulation into cloned environments.
This way, provide both invariance and more reward signal.

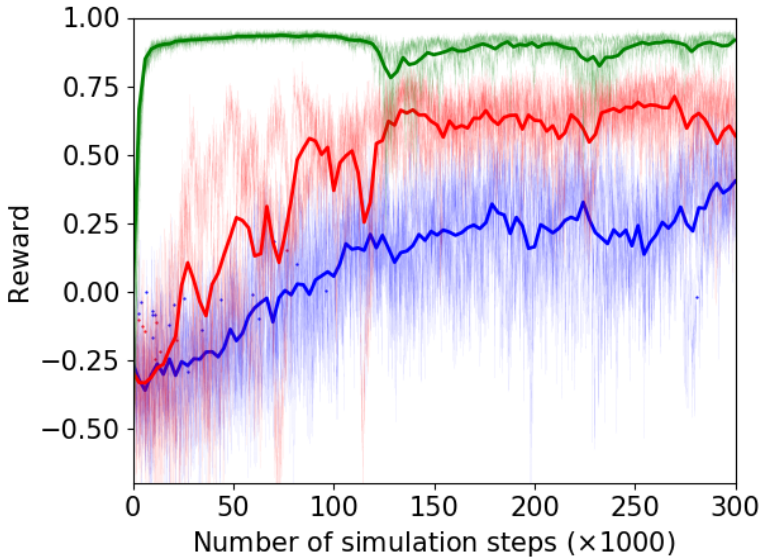


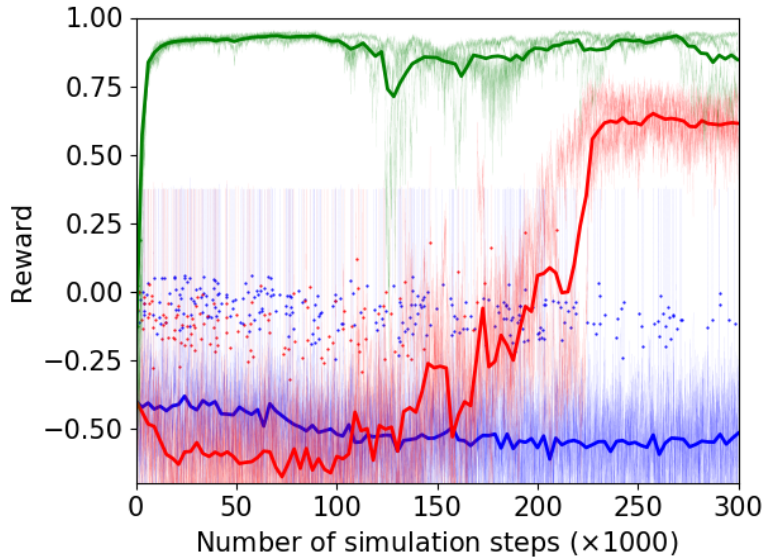
Self-collaborative control strategy. Possibility to mix the local and global rewards.

Comparing M1, M2, M3: 1, 5, 10, 20 jets





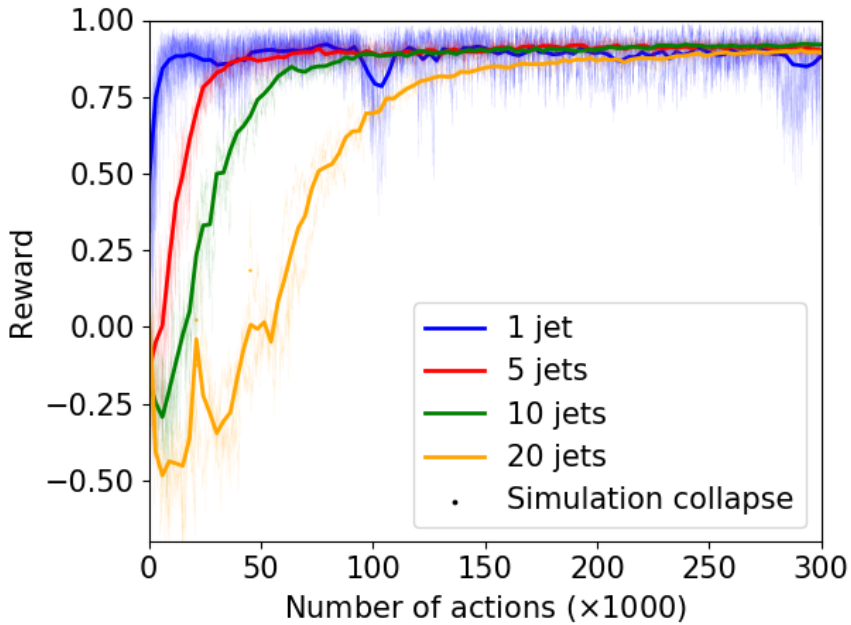


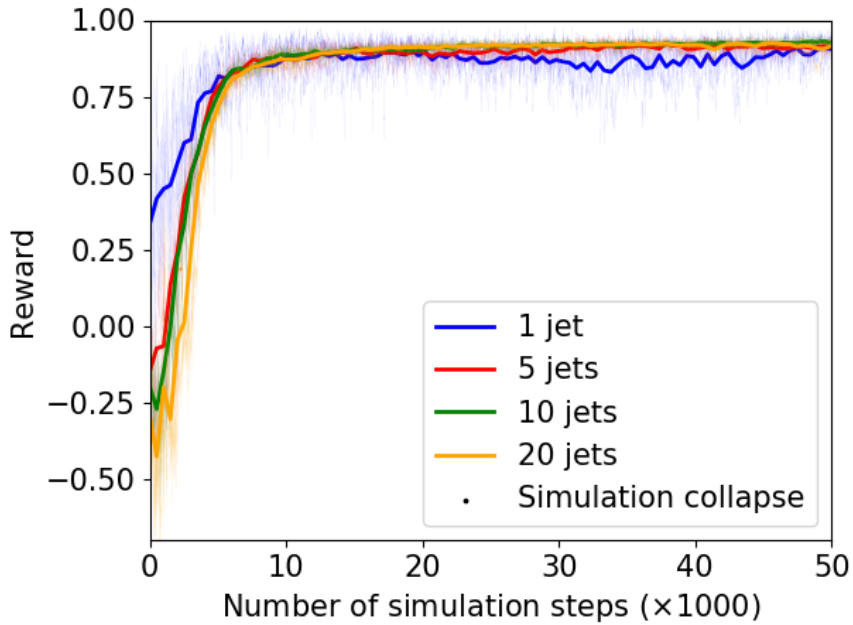


M3 can handle arbitrary number of jets

With M3, for N jets, perform N actions per simulation advance.

Cost of learning $\propto$ [number of simulation advances] rather than to [number of actions].





Constant cost of learning for any number of jets. Makes sense physically.

Controlling turbulence in 3D: turbulent channel flow

First applications to 3D turbulent channel:

"Reinforcement Learning of Control Strategies for Reducing Skin Friction Drag in a Fully Developed Channel Flow", Sonoda et. al., ArXiv 2022.

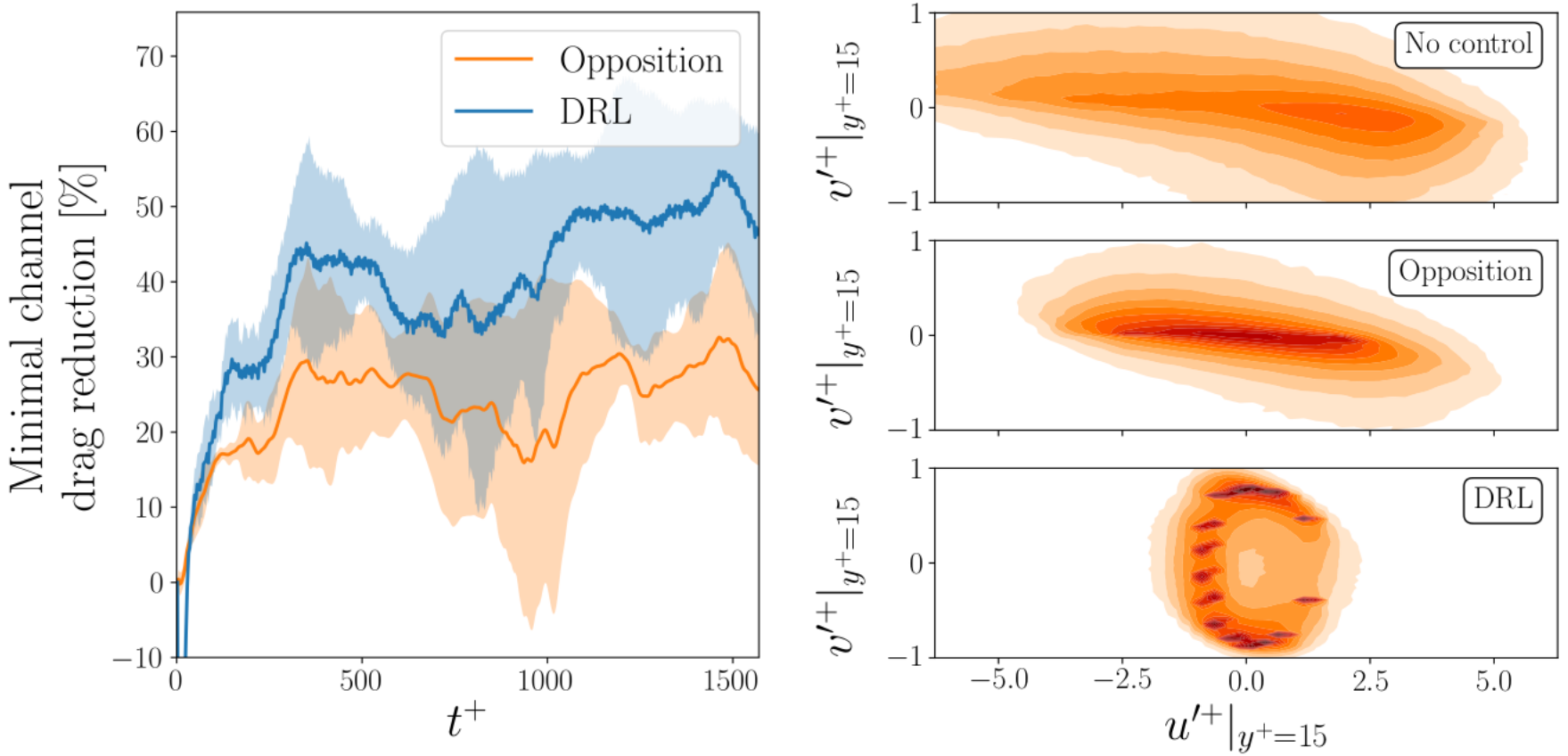
"Deep reinforcement learning for turbulent drag reduction in channel flows, L. Guastoni et. al., EPJE 2023.

- train in minimal turbulent channel Re_τ 180
- wall suction / blowing (0 net mass flow rate) at lower wall
- goal: minimize drag / Re_τ
- multi-agent reinforcement learning (M3)
- compare to opposition control from literature

`https://github.com/KTH-FlowAI/MARL-drag-reduction-in-wall-bounded-flows`

Controlling turbulence in 3D: turbulent channel flow

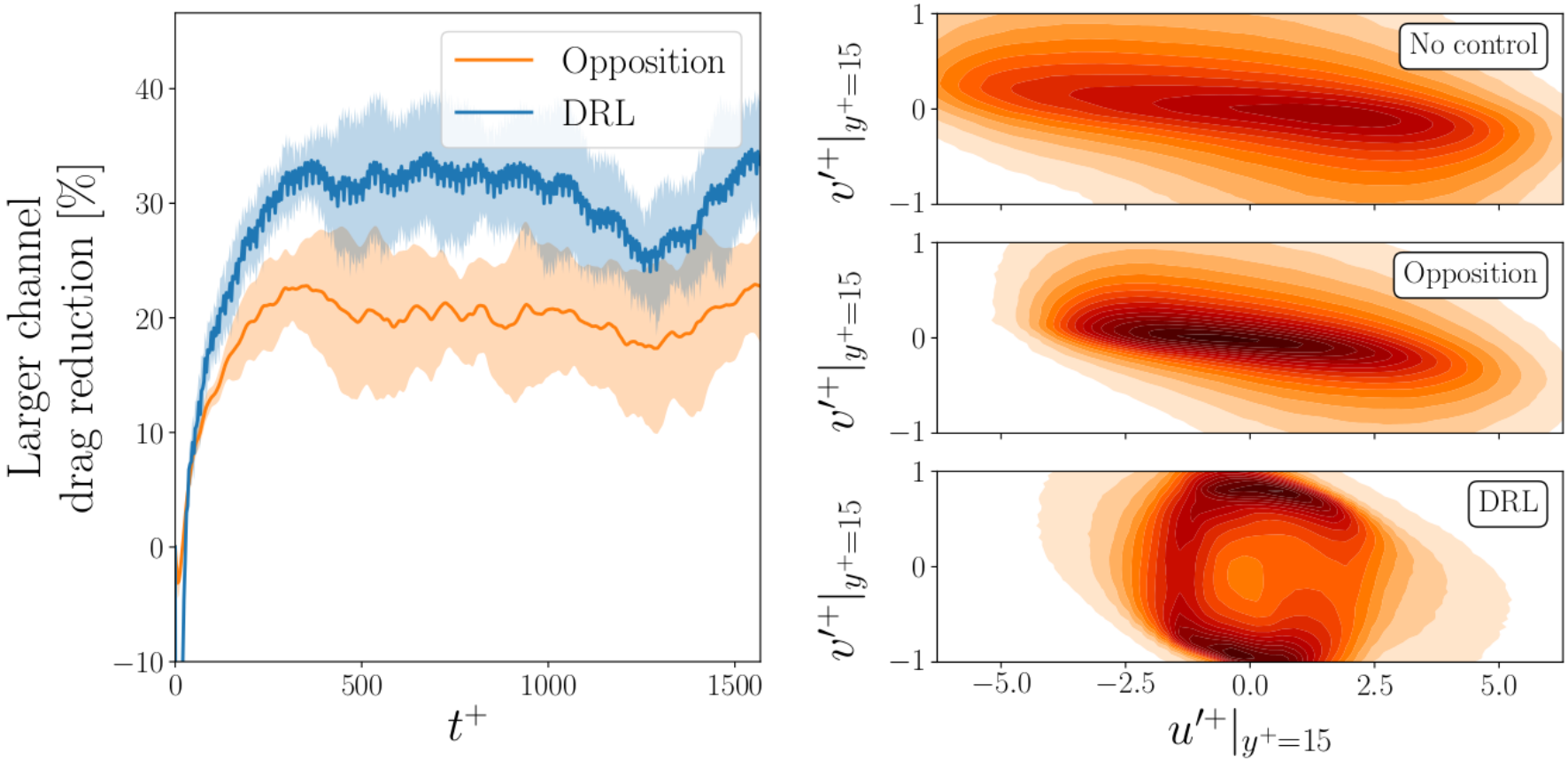
Results from Guastoni et. al., minimal channel.



Sweeps and ejections in the BL vanish; DRL learns a control that changes the flow physics compared with no control / opposition control.

Controlling turbulence in 3D: turbulent channel flow

Results from Guastoni et. al., full channel.



A bit of sweeps / ejections back; because trained in minimal channel but evaluate in full channel? - different physics? What if train in full channel (CPU time...)?

Conclusion

Future and perspectives

DRL is currently a promising avenue for active flow control.

Fundamentaly, exploration in a 'DRL-like' way may be the only possible approach with complex, non-linear, high-dimensionality systems.

Recent results:

- Proof-of-concept / parallelization / speedups / robustness.
- Application to large systems / invariants.

Major remaining works; need solvers, (many) CPUs, man-years:

- 3D systems of increasing complexity and realism.
- Higher Re, possibly through experiences (build FOSS framework?).

Progressively increase complexity

Share code / implementations to speed up progress,
enforce reproducibility, and split tasks between groups.

Relation to other black-box optimization methods?

Conceptually quite similar to e.g. Genetic Programming:

- Learn through trial and error
- No assumptions about the system
- Gradient descent vs. individual selection ("noisy individual reproduction" drift to the minimum analogous to gradient descent?)
- Difference: reward at each timestep vs. once per trajectory

Discussion of different black box optimizations methods: "Comparative analysis of machine learning methods for active flow control", Pino et. al., JFM, 2023.