Deep Reinforcement Learning applied to Fluid Mechanics

J. Rabault[1,2]
[1]: Norwegian Meteorological Institute, IT department
[2]: University of Oslo, Department of Mathematics

Updated November 2020

Complexity: when our tools break

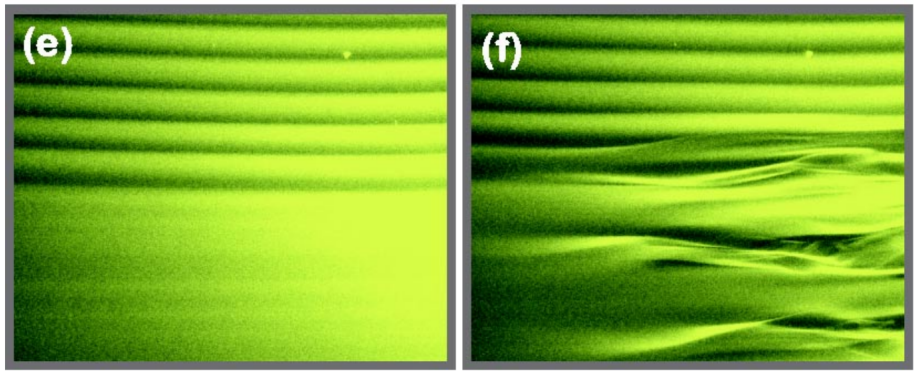
Modern science has much difficulty to handle combination of:

- Non-linearity
- Non-convexity
- High dimensionality

Most analytical tools break; simulation can be expensive. 'Lucky' in Fluid Mechanics to have this problem:

- Proof or dismissal that *"In three space dimensions and time, given an initial velocity field, there exists a vector velocity and a scalar pressure field, which are both smooth and globally defined, that solve the Navier Stokes equations"*, Clay Mathematical Institute / Millenium problems.
- *"I think the next [21st] century will be the century of complexity"*, Stephen W. Hawking (2005).

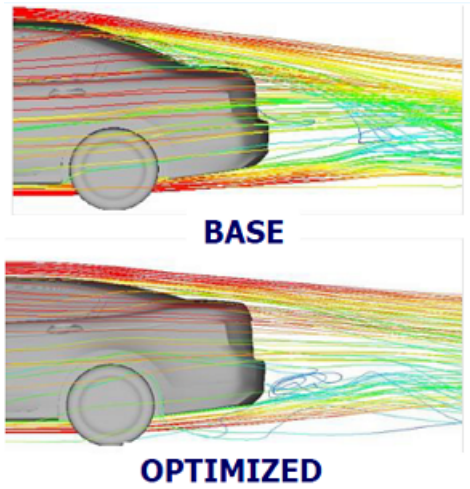
Motivation 1: flow Control and turbulence

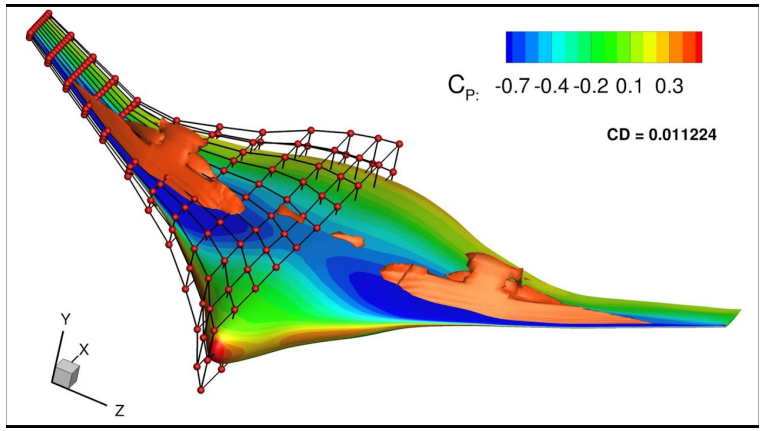


'Delaying transition to turbulence by a passive mechanism', *Fransson et. al.* (2006).

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

Motivation 2: optimization of complex systems





- Shape optimization: everywhere in engineering.
- How to optimize nonlinear problem? Challenge for gradient descent.
- There also, difficult, need user expertise.

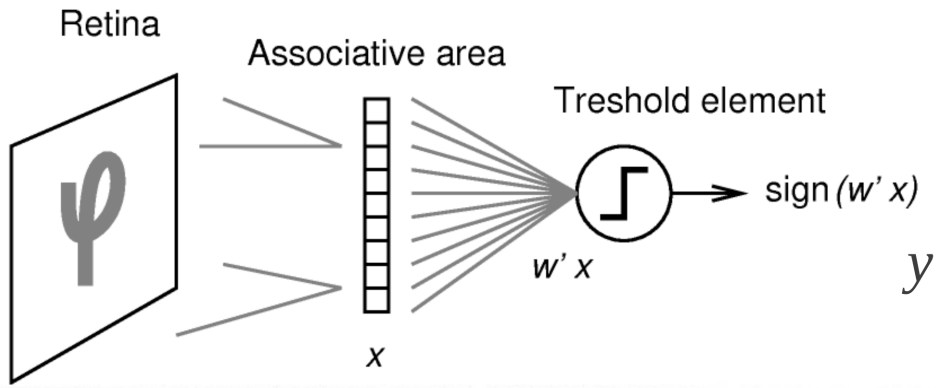
Try to introduce new tools? Recent successes of Deep Reinforcement Learning (DRL) on complex problems is a strong hint (Go, Poker, robotics, control of complex cooling systems).

A short reminder about Artificial Neural Networks and Supervised Learning

Artificial Neural Networks: a bit of history

Idea of ANNs appeared with the first computers:

'The perceptron, a perceiving and recognizing automaton', *Rosenblatt*, Cornell Aeronautical Laboratory (1957).



$$y = sign\left(\sum_{i=1}^{N} W_i X_i + b\right)$$

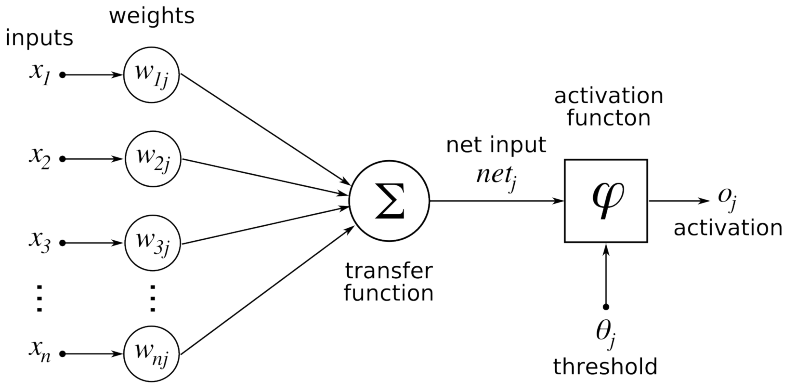
It took some time before useful applications:

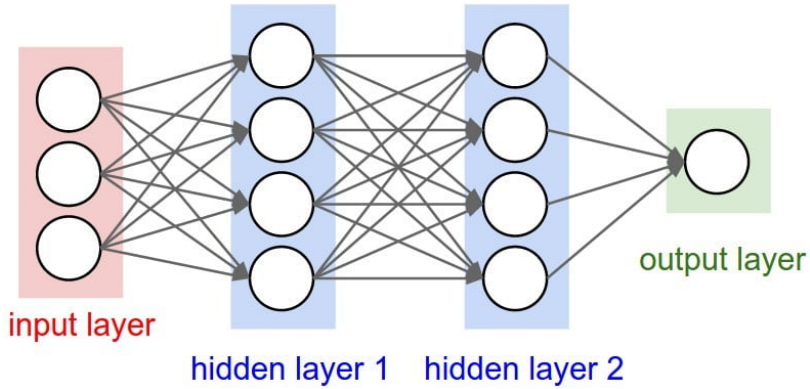
- A few applications during the 70s.
- Large scale application of CNNs in the 90s.
- Recently 'AI spring' since around 2012: Deep Learning.

Succession of 'AI springs' and 'AI winters', but now it is summer...

Deep ANNs: I

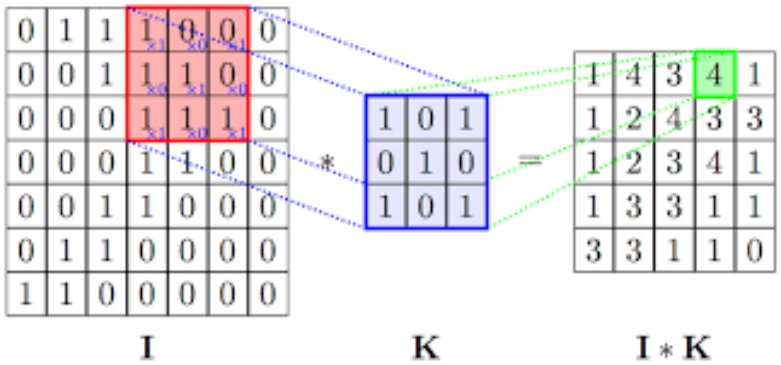
Artificial Neural Networks are simply a set of neurons.





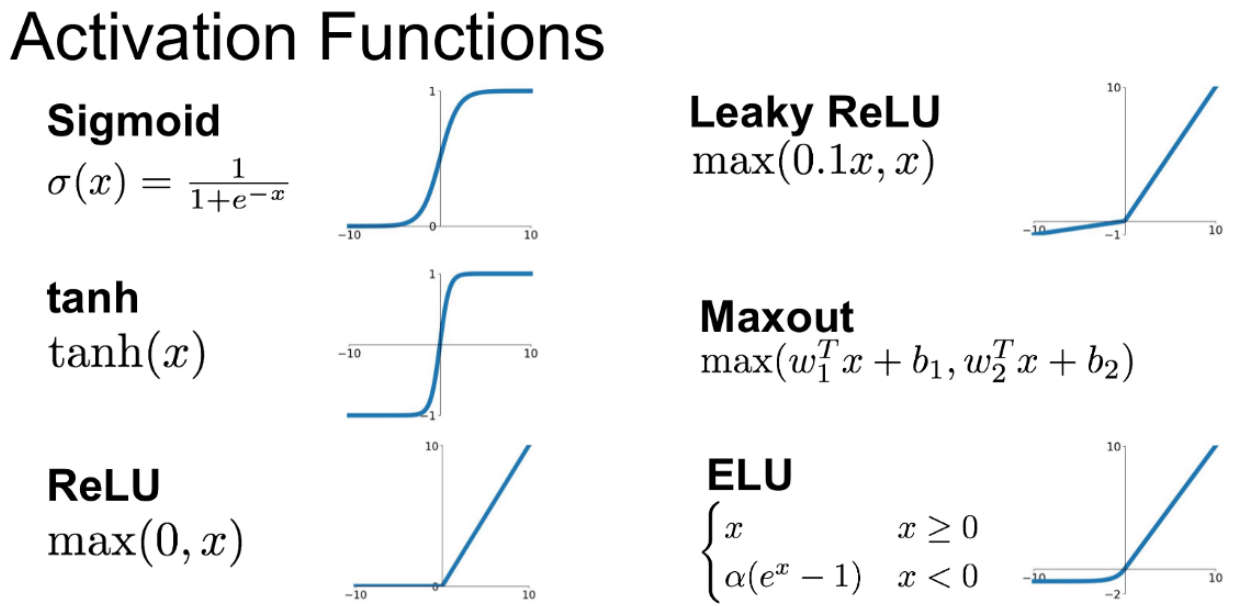
Deep ANNs: ANNs with several hidden layers. Feed one layer in the next: 'Deep Learning', *LeCun et. al.*, Nature (2015).

Convolution layers enforce invariance by translation:



Deep ANNs: II

What activation function to use?



- Linear activation function: only linear network...
- Non linear functions: 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.

Deep ANNs: III

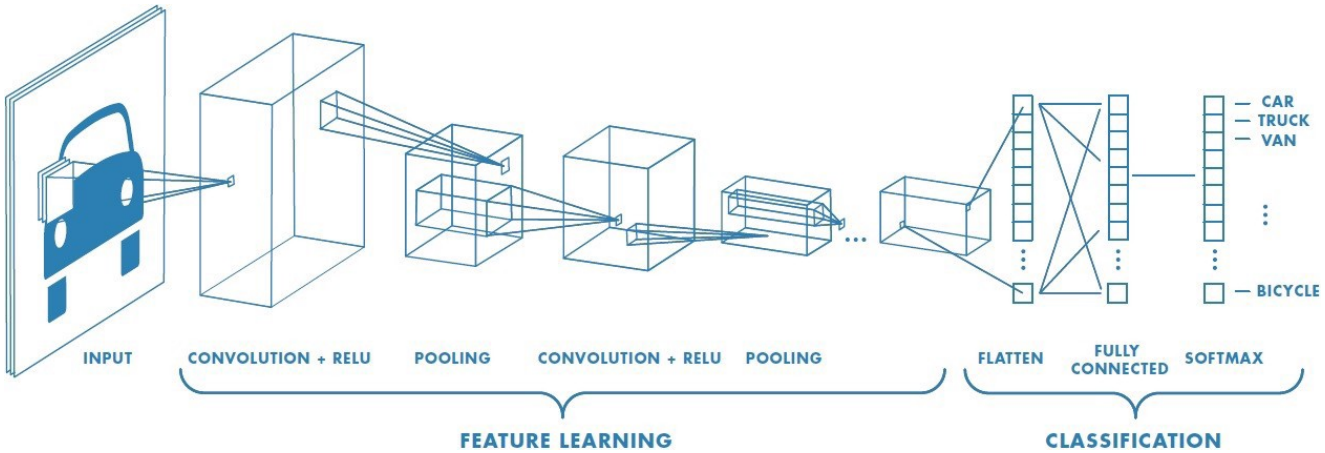
What architecture to use?

- 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:

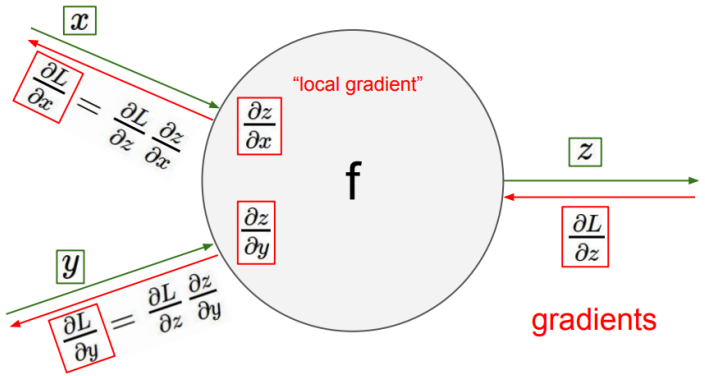


Deep ANNs: IV

Universal approximator, easily parallelizable, effective representation

How to train?

- Naive method: try at random....
- Better method: gradient descent, back propagation of errors: 'Applications of advances in nonlinear sensitivity analysis', *Werbos*, System modeling and optimization (1982).

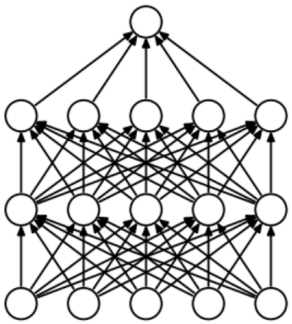


- Feed data in, compute error.
- Back propagate (Leibniz) the error gradient.
- Update the coefficients.

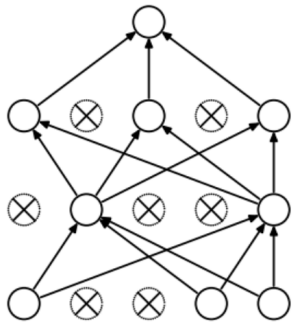
Deep ANNs: V

Still problems training: overfitting, stability of gradient, etc.

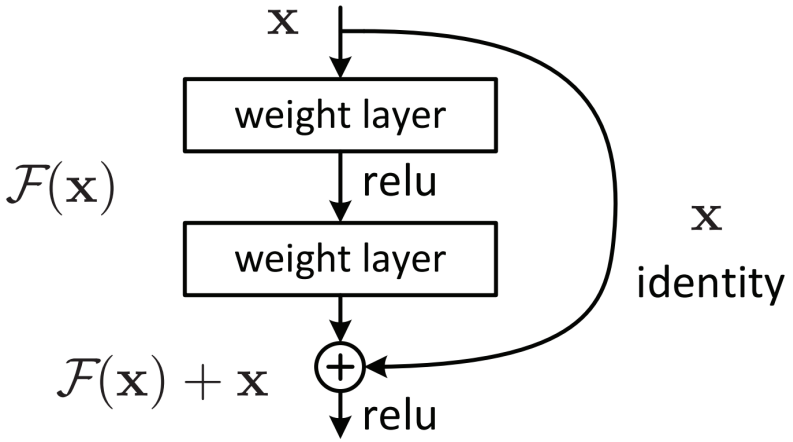
- Large datasets, training vs. validation, data augmentation.
- Dropout layers (drop connections at random).
- Weight initialization for gradient stability.
- Batch renormalization (fix mean and variance in each layer).
- Residual networks (learn changes from identity).



(a) Standard Neural Net



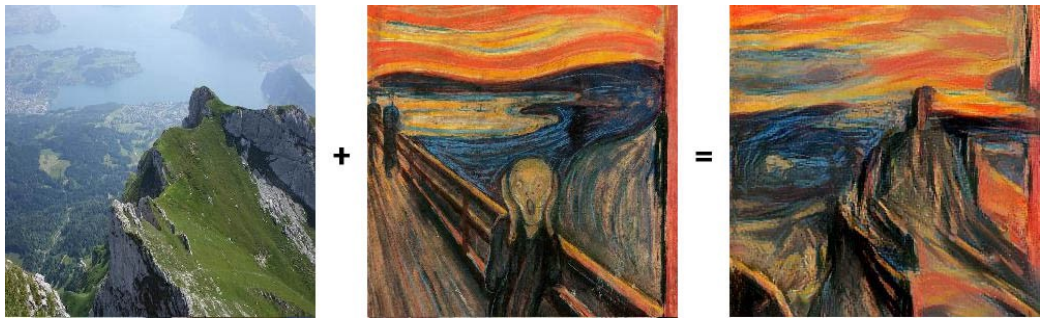
(b) After applying dropout.

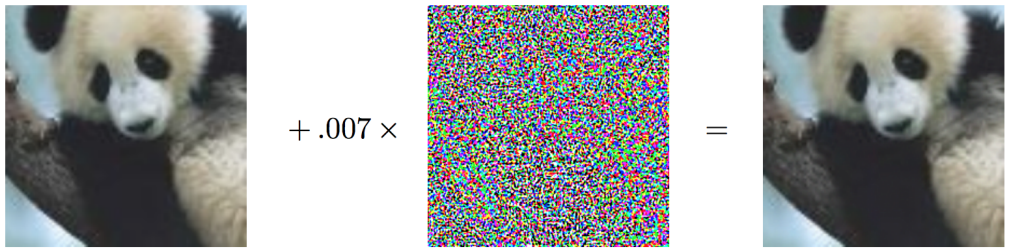


The Deep Learning revolution

- Large amounts of data, powerful GPUs, fast batch training.
- A number of technical tricks, many beautiful / interesting results...

"A Neural Algorithm of Artistic Style", Gatys et. al., ArXiv (2015).
"Explaining and Harnessing Adversarial Examples", Goodfellow et. al., ICLR 2015.



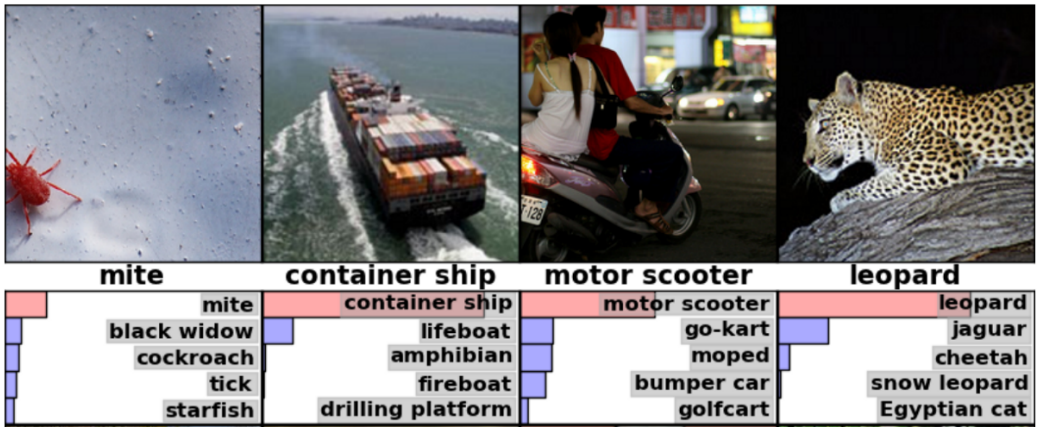


$\boldsymbol{x}$
"panda"
57.7% confidence

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Supervised learning: describe a function from examples



Supervised learning, image analysis work...

Yoshua Bengio

FOLLOW

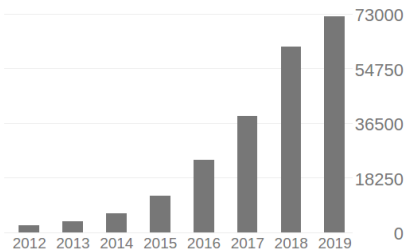
Professor, University of Montreal (Computer Sc. & Op. Res.), Mila, CIFAR, CRM, IVADO, REPARTI, GRSNC
Verified email at mila.quebec - Homepage

Machine learning deep learning artificial intelligence

TITLE	CITED BY	YEAR
Gradient-based learning applied to document recognition Y LeCun, L Bottou, Y Bengio, P Haffner Proceedings of the IEEE 86 (11), 2278-2324	23110	1998
Deep learning Y LeCun, Y Bengio, G Hinton nature 521 (7553), 436-444	21371	2015
Generative adversarial nets I Goodfellow, J Pouget-Abadie, M Mirza, B Xu, D Warde-Farley, S Ozair, ... Advances in neural information processing systems, 2672-2680	14043	2014
Deep learning I Goodfellow, Y Bengio, A Courville MIT press	12039	2016

Cited by VIEW ALL

	All	Since 2014
Citations	238483	218815
h-index	153	141
i10-index	502	443



Co-authors VIEW ALL

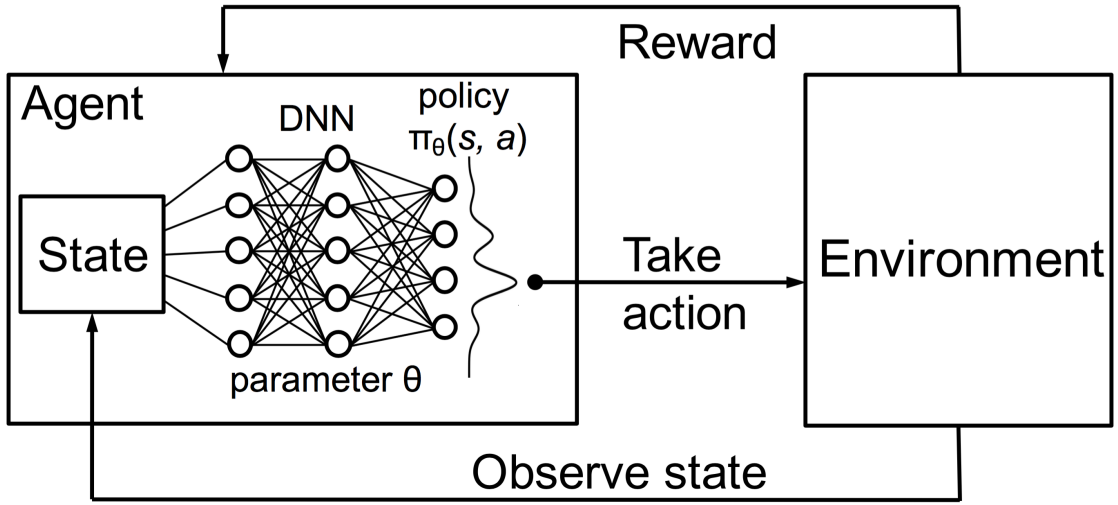
Aaron Courville
Université de Montréal

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., ArXiv (2019).



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

Several methods. Main ones:

- Q-learning.
- Policy gradients.

i. 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:

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

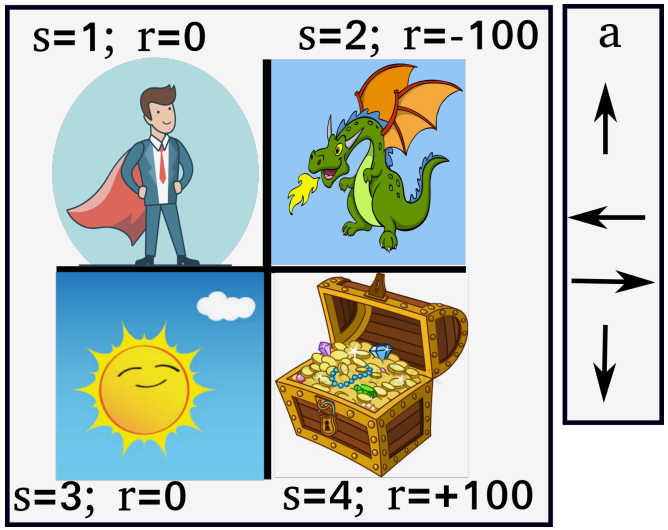
Random action at probability ϵ, arg max action at probability $1 - \epsilon$, and:

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

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

Q-learning on an example

Simple example: a 2 x 2 matrix game.



What does $Q^*(s, a)$ look like? First a bit of initialization:

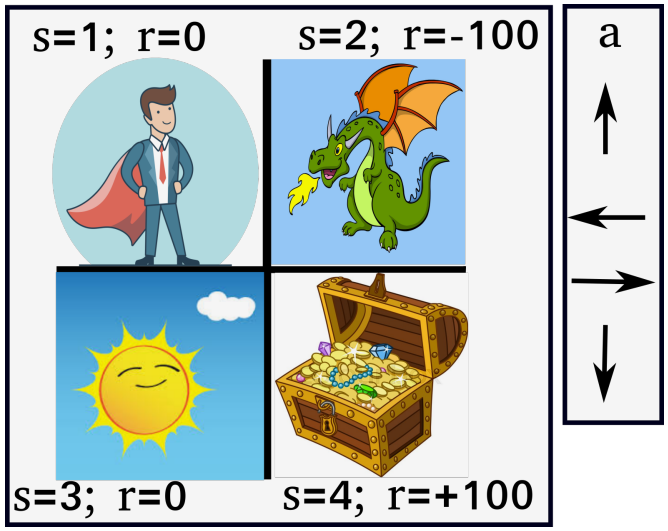
Q(s, a)		s			
		1	2	3	4
a	↑	?	?	?	?
	↓	?	?	?	?
	→	?	?	?	?
	←	?	?	?	?

→

Q(s, a)		s			
		1	2	3	4
a	↑	0	x	0	x
	↓	0	x	0	x
	→	0	x	0	x
	←	0	x	0	x

Q-learning on an example

Simple example: a 2 x 2 matrix game.



What does $Q^*(s, a)$ look like? Now learn, use $\alpha = 0.1$, $\gamma = 0.5$, $\epsilon = 1$: r.

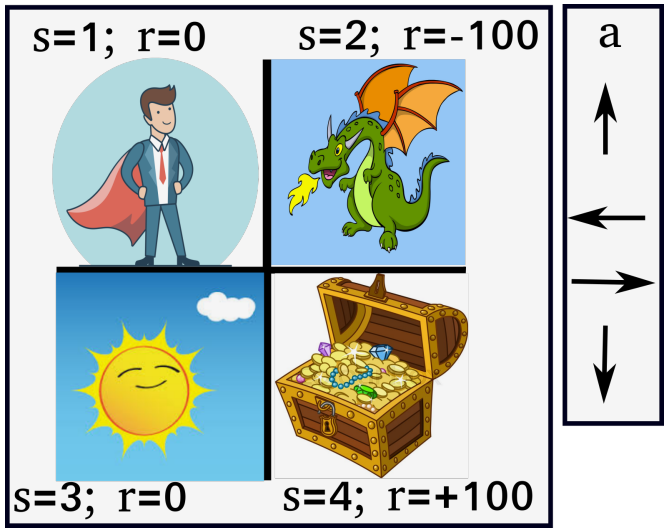
Q(s, a)		s			
		1	2	3	4
a	↑	0	x	0	x
	↓	0	x	0	x
	→	0	x	0	x
	←	0	x	0	x

→

Q(s, a)		s			
		1	2	3	4
a	↑	0	x	0	x
	↓	0	x	0	x
	→	-10	x	0	x
	←	0	x	0	x

Q-learning on an example

Simple example: a 2 x 2 matrix game.



What does $Q^*(s, a)$ look like? Now learn, use $\alpha = 0.1$, $\gamma = 0.5$, $\epsilon = 1$: d;r.

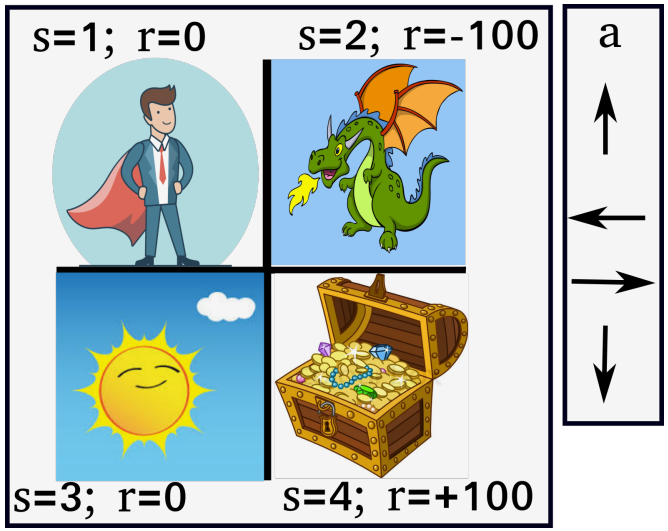
Q(s, a)		s			
		1	2	3	4
a	↑	0	x	0	x
	↓	0	x	0	x
	→	-10	x	0	x
	←	0	x	0	x

→

Q(s, a)		s			
		1	2	3	4
a	↑	0	x	0	x
	↓	0	x	0	x
	→	-10	x	10	x
	←	0	x	0	x

Q-learning on an example

Simple example: a 2 x 2 matrix game.



What does $Q^*(s, a)$ look like? Now learn, use $\alpha = 0.1$, $\gamma = 0.5$, $\epsilon = 1$: d;u;d,r.

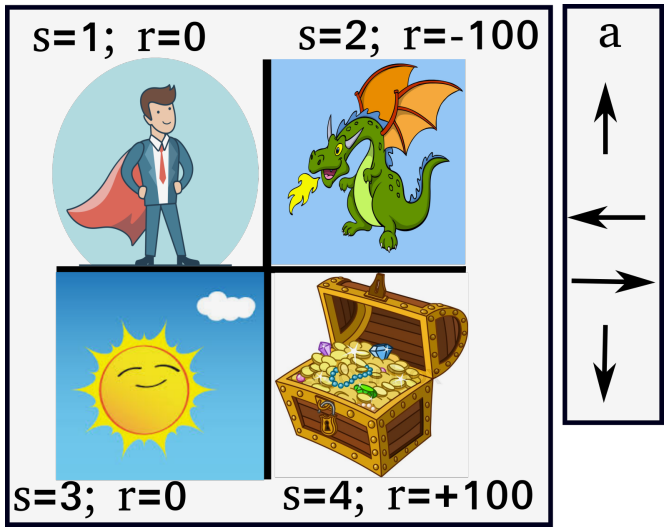
Q(s, a)		s			
		1	2	3	4
a	↑	0	x	0	x
	↓	0	x	0	x
	→	-10	x	10	x
	←	0	x	0	x

→

Q(s, a)		s			
		1	2	3	4
a	↑	0	x	0.025	x
	↓	0.5 ; 0.95	x	0	x
	→	-10	x	19	x
	←	0	x	0	x

Q-learning on an example

Simple example: a 2 x 2 matrix game.



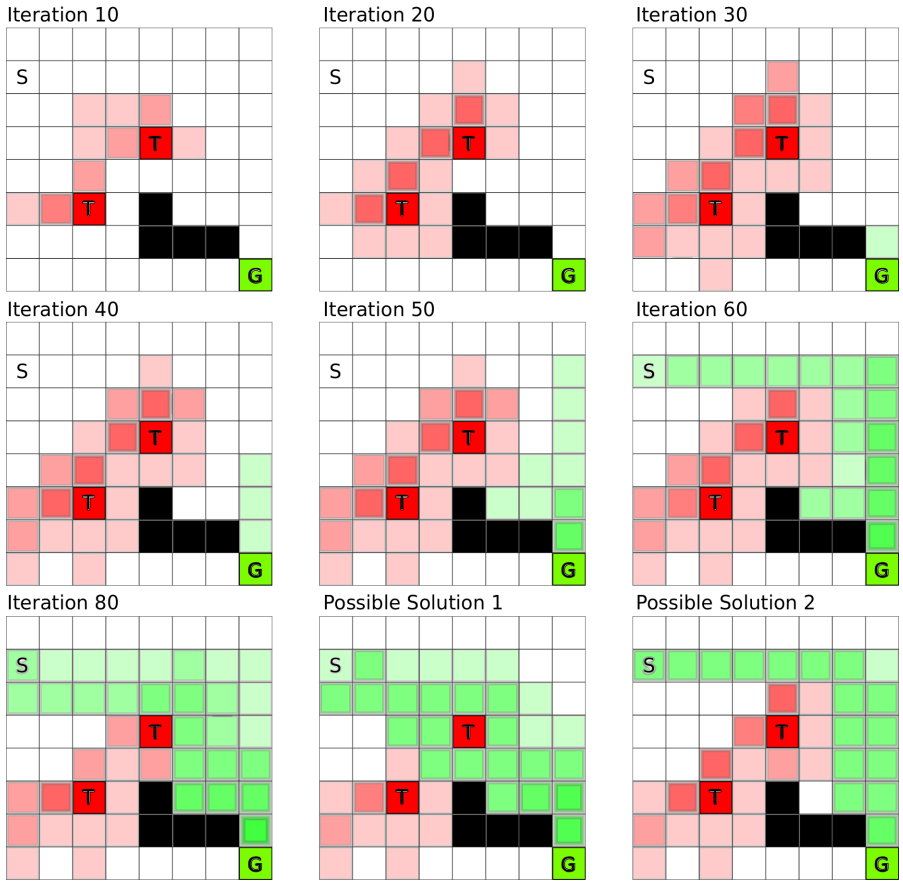
What does $Q^*(s, a)$ look like? After many iterations, converge:

Q(s, a)		s			
		1	2	3	4
a	↑	25	x	25	x
	↓	50	x	50	x
	→	-100	x	100	x
	←	25	x	50	x

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).

ii. Policy Gradient method

Policy Gradient methods

Policy Gradient: directly learn the policy π.

Policy gradient vs Q-Learning:

- Often π simpler than Q.
- Q: need to be able to effectively solve arg max, tricky high dimension.
- Policy gradient: more versatile (continuous learning).
- Q: better exploration (can be taken care of).

Use a stochastic policy: π is a distribution for each state input, 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), ..., (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:

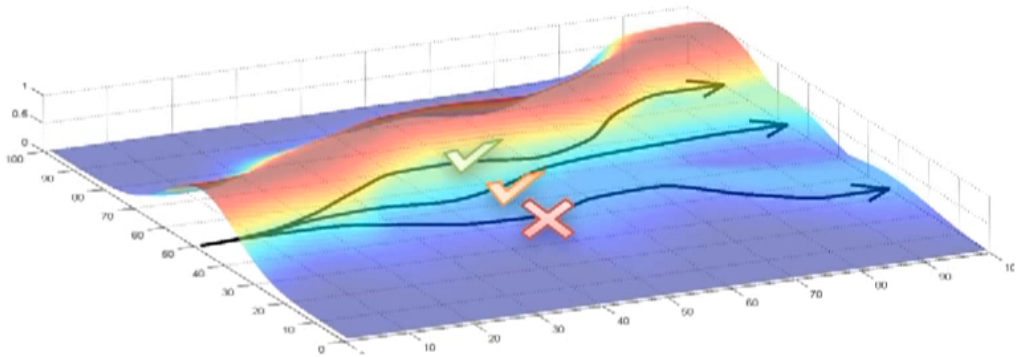
$$\nabla_\Theta V(\Theta) \approx \hat{g} = \frac{1}{M} \sum_{i=1}^{M} \nabla_\Theta \log\left(\mathbb{P}(\tau^{(i)}, \Theta)\right) R(\tau^{(i)})$$

Understanding the policy gradient

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

- Valid for discontinuous / unknown R.
- Works with discrete or continuous state / action.

Works by improving 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:

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

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

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

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

$$\nabla_\Theta V(\Theta) = \mathbb{E}\left[\hat{g}\right],$$

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

Importance sampling and off-policy learning

Derivations so far are 'on-policy':

- Use same policy to explore (i.e. collect samples) and compute the policy gradient.
- Poor sampling efficiency: need new samples each time policy is updated.

Unrealistic in reality. Resort to **importance sampling**:

Use old samples, but compensate for distribution shifts.

This allows to study one distribution, while sampling from another one.

$$\mu = \mathbb{E}_p(f) = \int_D f(x)p(x)dx$$
$$= \int_D \frac{f(x)p(x)}{q(x)} q(x)dx = \mathbb{E}_q\left(\frac{f(X)p(X)}{q(X)}\right)$$

Surrogate objective and KL norm penalization

Re-write value function:

$$V(\theta) = \sum_t \mathbb{E}_{(s_t,a_t)\sim p_\theta(s_t,a_t)}[r_t(s_t,a_t)\gamma^t]$$
$$= \sum_t \mathbb{E}_{s_t\sim p_\theta(s_t)}[\mathbb{E}_{a_t\sim\pi_\theta(a_t|s_t)}[r_t(s_t,a_t)\gamma^t]]$$

so that applying importance sampling:

$$V(\theta') = \sum_t \mathbb{E}_{s_t\sim p_\theta(s_t)}\left[\frac{p_{\theta'}(s_t)}{p_\theta(s_t)}\mathbb{E}_{a_t\sim\pi_\theta(a_t|s_t)}\left[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)}r_t(s_t,a_t)\gamma^t\right]\right]$$

Therefore to allow re-use of previous trajectories:

- Optimize (advantage) surrogate objective: $\max_\theta \left(\mathbb{E}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right]\right)$
- With constraint: $\mathbb{E}\left[KL(\pi_{\theta_{old}}(\cdot|s_t),\pi_\theta(\cdot|s_t))\right] < \delta$

KL = Kullback Leibler divergence ('difference' between 2 distributions).

Many refinements...

Many more refinements than we have time for today.
Read:

- "Advanced Policy Gradient Methods", Joshua Achiam: `http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/lecture_13_advanced_pg.pdf` .
- "Proximal Policy Optimization Algorithms", John Schulman et. al., ArXiv (2017), `https://arxiv.org/pdf/1707.06347.pdf` .

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.

Proximal Policy Optimization (PPO)

State of the art gradient based method:

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

Implement a few tricks:

- Critic network: learn (noisy) actualized reward with a separate ANN.
- Clip gradient (instead of use KL) to discourage unsafe policy changes.

Used in the following; Open Source implementation:

'Tensorforce: a TensorFlow library for applied reinforcement learning', *Kuhnle et. al.*, Github (2017)
https://github.com/tensorforce/tensorforce

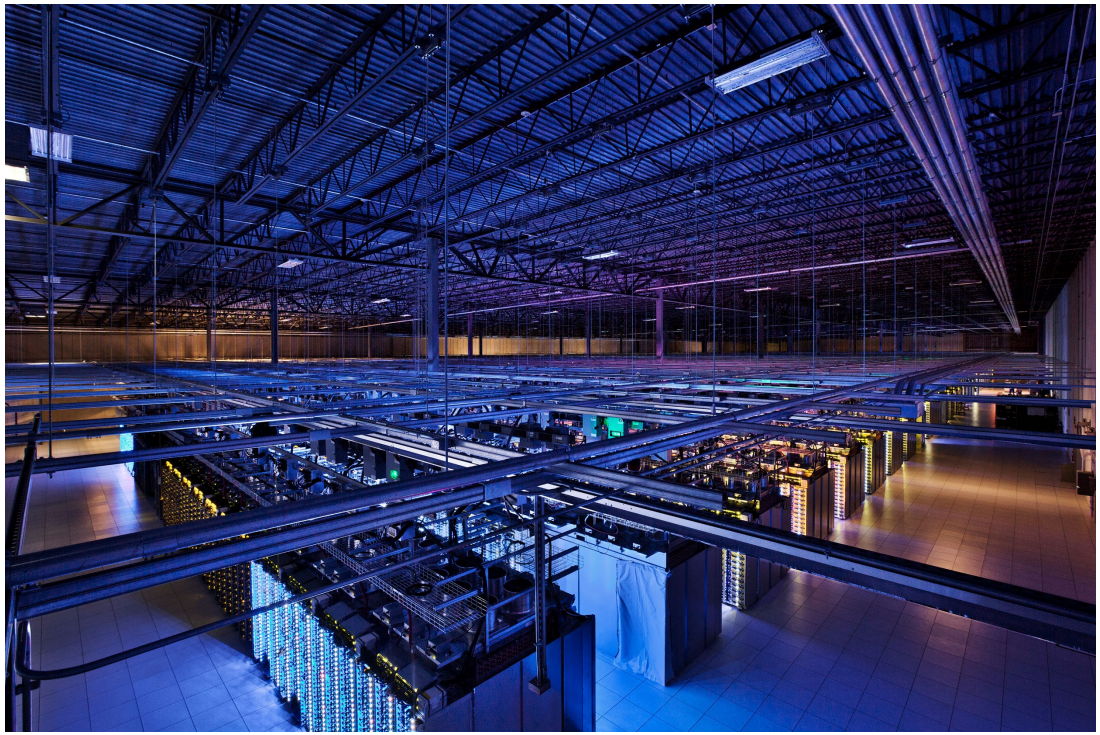
Built as a Tensorflow computational graph for efficiency.

Real world application

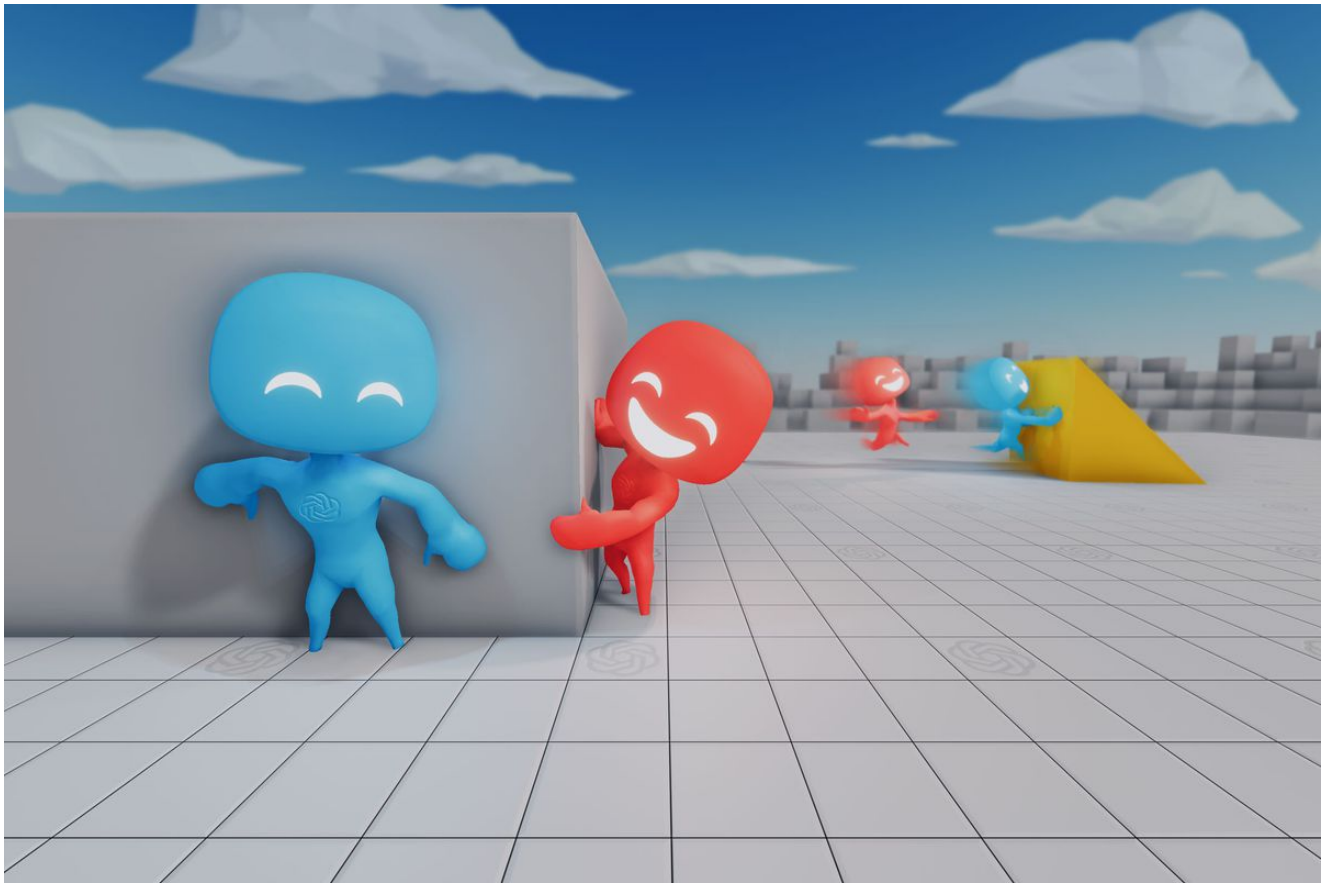
Video games; Go; Poker, Rubik's cube; some recent industry deployments:

"Google just gave control over data center cooling to an AI",
MIT Technology Review, Aug. 2018.

Energy saving in servers cooling, -40% cooling electricity use.



A nice video illustration of DRL effectiveness



https://youtu.be/kopoLzvh5jY

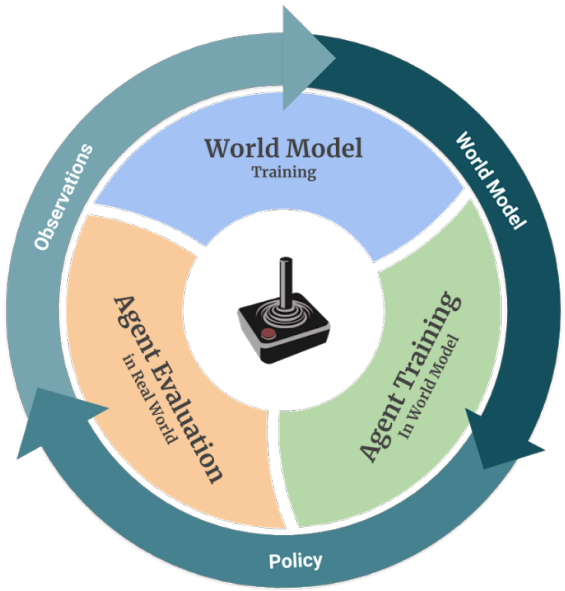
iii. Refinements and amelioration of RL algorithms

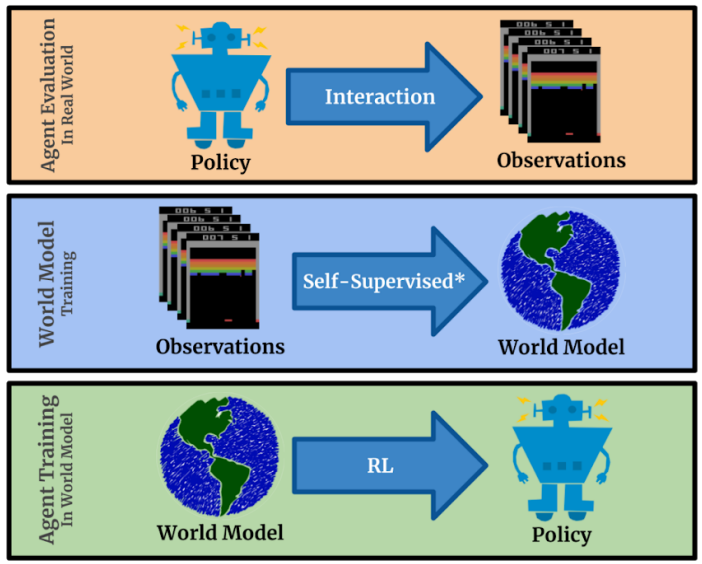
World models

"World Models", Ha and Schmidhuber, ArXiv / NIPS (2018).

- Learn the control and the model at the same time.
- Supervised learning of the model.
- Use it for training without the env.
- May use the same ANN to enrich output.

'Getting ANNs to dream'





Curiosity

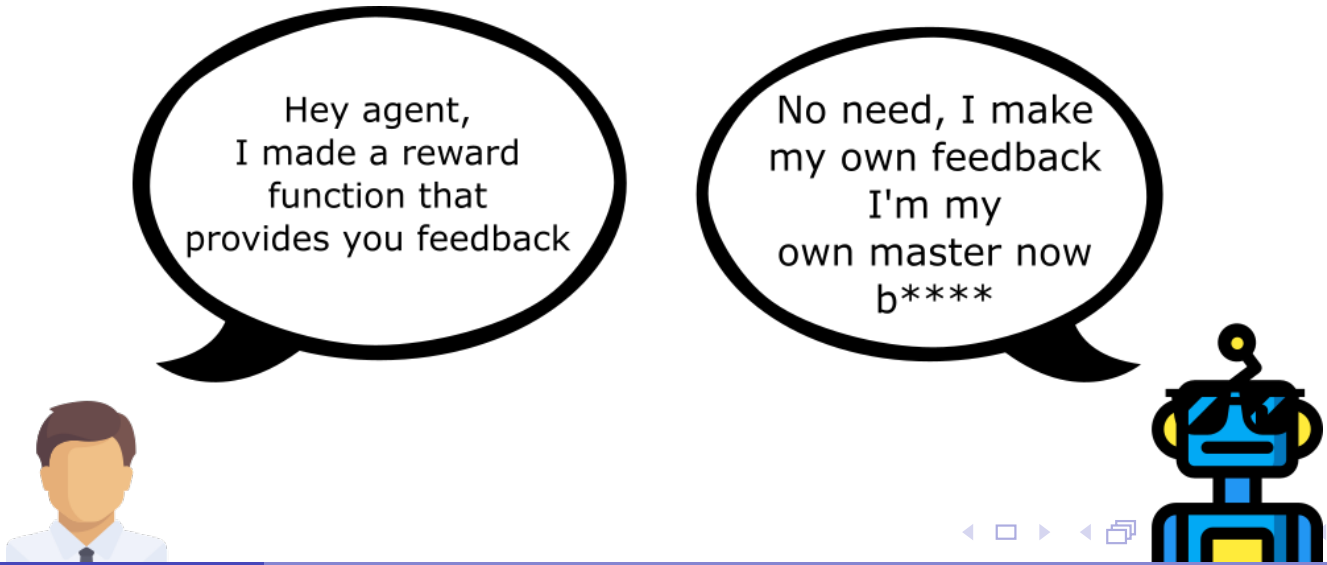
"Surprise-Based Intrinsic Motivation for Deep Reinforcement Learning", J. Achiam, S. Sastry, ArXiv (2017).

- How to favorize exploration?
- How to make sure ANN explores the whole system?

$$\text{reward} = (\text{env reward}) + (\text{surprise reward})$$

Surprise reward:

- The ANN also tries to predict next state.
- Reward based on how different next state vs. prediction.



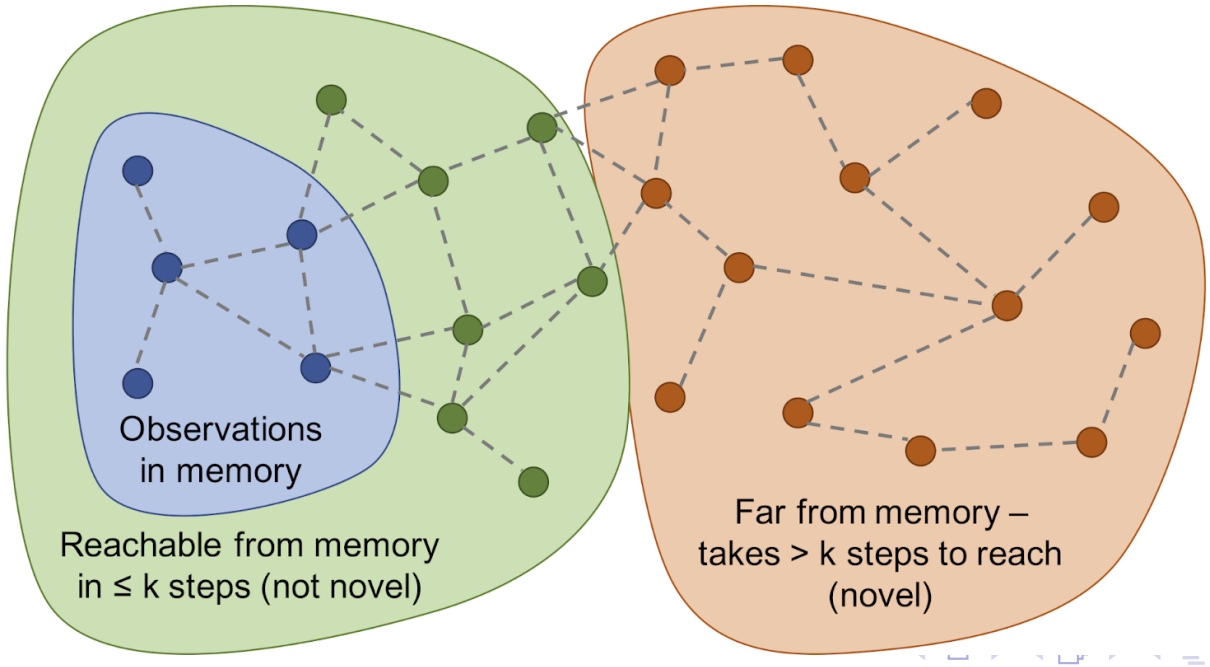
Curiosity through reachability

"Episodic Curiosity through Reachability", N. Savinov et. al., ArXiv (2018).

Problem of vanilla curiosity-driven:

- Procrastination.
- 'couch potato' in front of random TV.

Base 'surprise reward' on distance in terms of number of actions.



Teaching through example

"Learning Montezuma's Revenge from a Single Demonstration", Tim Salimans and Richard Chen , OpenAI blog (2018).

Huge cost in exploration:

- Take a game when need to perform exact sequence of actions.
- Exploration cost exponential in sequence length.
- The 'Moteczuma revenge' benchmark.
- DRL has difficulty with long chains of causal actions and sparse reward.

Solution:

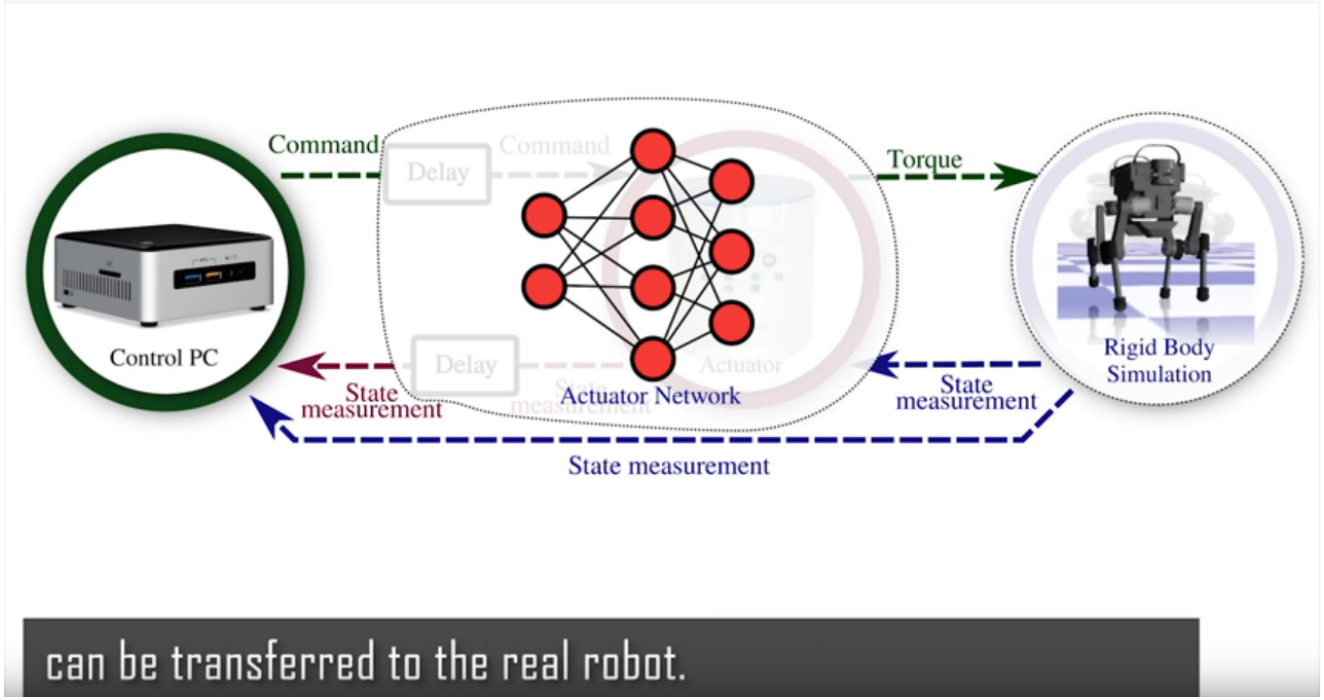
- Provide some example games.
- Start at random position in the example.

Drastically cut the cost of exploration.

Transfer learning simulation / real world, reality gap

Challenges going from simulations to real world: reality gap.
"Learning Agile and Dynamic Motor Skills for Legged Robots", Hwangbo et. al. (2019)

https://youtu.be/aTDkYFZFWug



iv. Frameworks for DRL

Open Source frameworks

Several Open Source frameworks. Among others:

- Tensorforce, `https://github.com/tensorforce/tensorforce`.
- stable-baselines, `https://github.com/hill-a/stable-baselines`.

Workflow:

- Build your application-specific environment by filling in a class.
- Use built-in Agent (PPO, DQN, other) and built-in training functions to write a training script.
- Train.
- Evaluate in deterministic mode.

Moving from a framework to another is very little work (a few copy-pastes), because all what is needed is to implement [state, action, reward, terminal] communication.

Frameworks: environment class in tensorforce

Full specification at:
https://github.com/tensorforce/tensorforce/blob/master/tensorforce/environments/environment.py .

```
from tensorforce.environments import Environment

class MyEnvironment(Environment):
    def states(self):
        """Returns the state space specification.
        """

    def actions(self):
        """Returns the action space specification.
        """

    def reset(self):
        """Resets the environment to start a new episode.
        Returns: dict[state]: Dictionary containing initial state(s) and auxiliary informatic
        """

    def execute(self, actions):
        """Executes the given action(s) and advances the environment by one step.
        Args: actions (dict[action]): Dictionary containing action(s) to be executed
        Returns:
            ((dict[state], bool | 0 | 1 | 2, float)): Dictionary containing next state(s),
            terminal information, and observed reward.
        """
```

Frameworks: environment class in gym

Full specification at:
`https://github.com/openai/gym/blob/master/gym/core.py` .

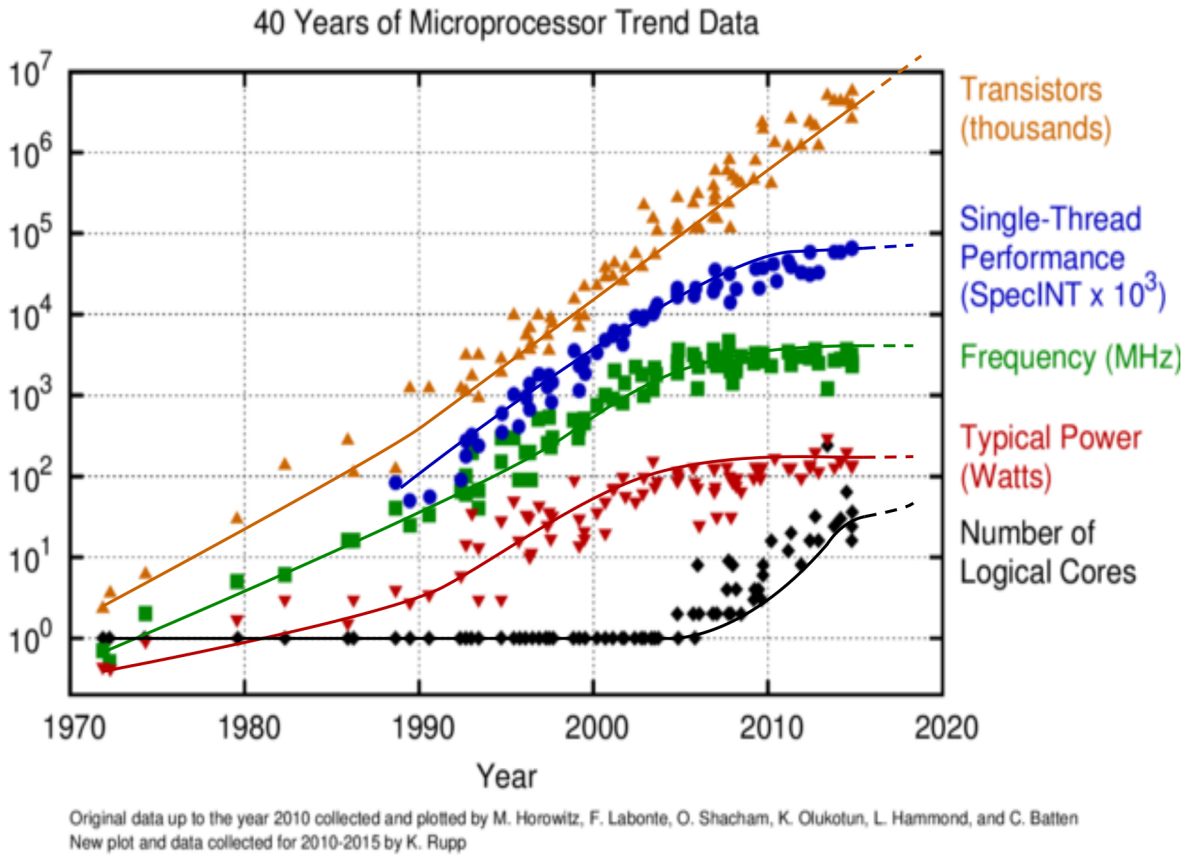
```
import gym

class Env(object):
    """The main OpenAI Gym class. It encapsulates an environment with
    arbitrary behind-the-scenes dynamics. An environment can be
    partially or fully observed.
    The main API methods that users of this class need to know are:
        step
        reset
        render
        close
        seed
    And set the following attributes:
        action_space: The Space object corresponding to valid actions
        observation_space: The Space object corresponding to valid observations
        reward_range: A tuple corresponding to the min and max possible rewards
    Note: a default reward range set to [-inf,+inf] already exists. Set
    it if you want a narrower range.
    The methods are accessed publicly as "step", "reset", etc.. The
    non-underscored versions are wrapper methods to which we may add
    functionality over time.
    """
```

v. Why relevant for future

ANNs, computing power, CPUs, GPUs, Moore's law

Increasingly attractive to use ANNs: the Moore's law is NOT dead yet.
But one thread, free performance updates are dead.

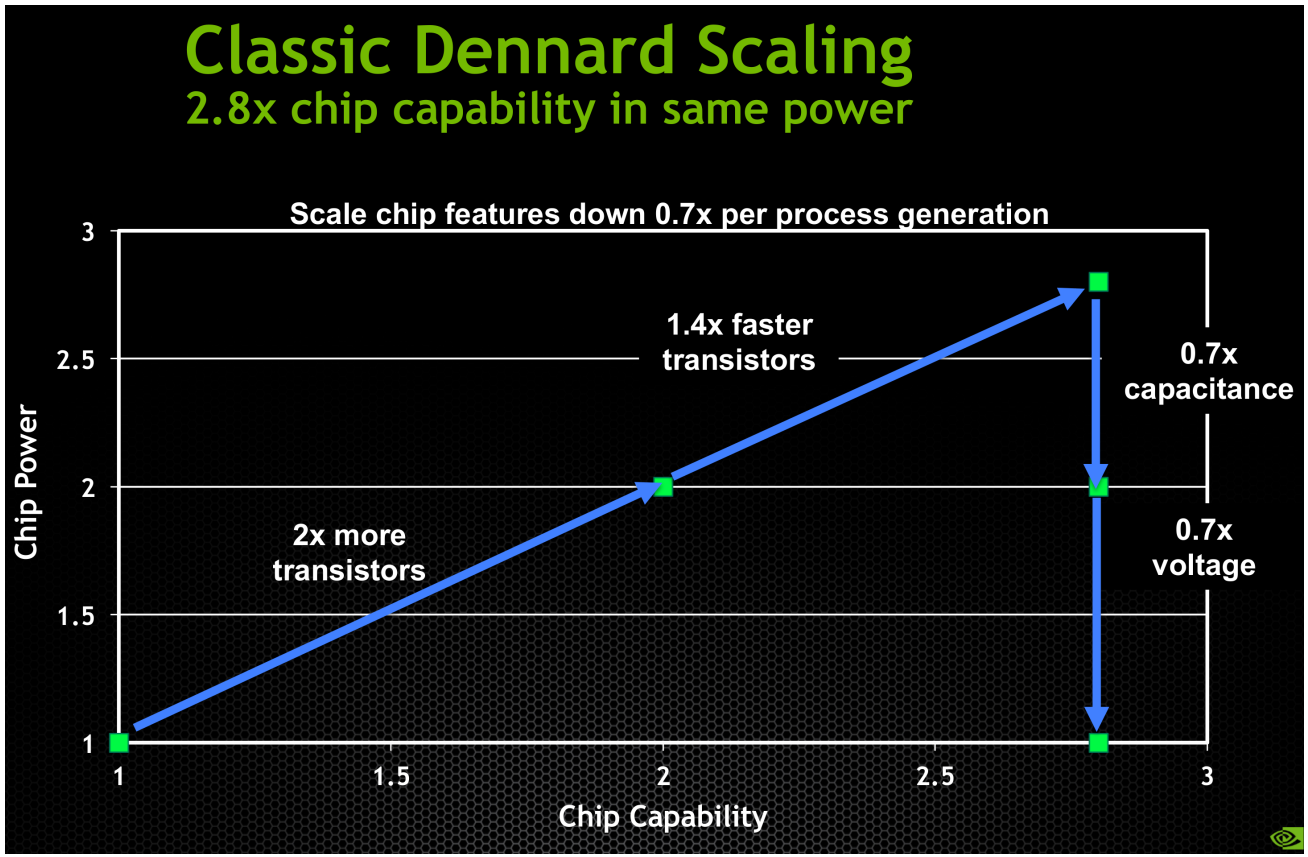


Power limitations is the new wall computing power is hitting.

Dennard scaling and the energy wall

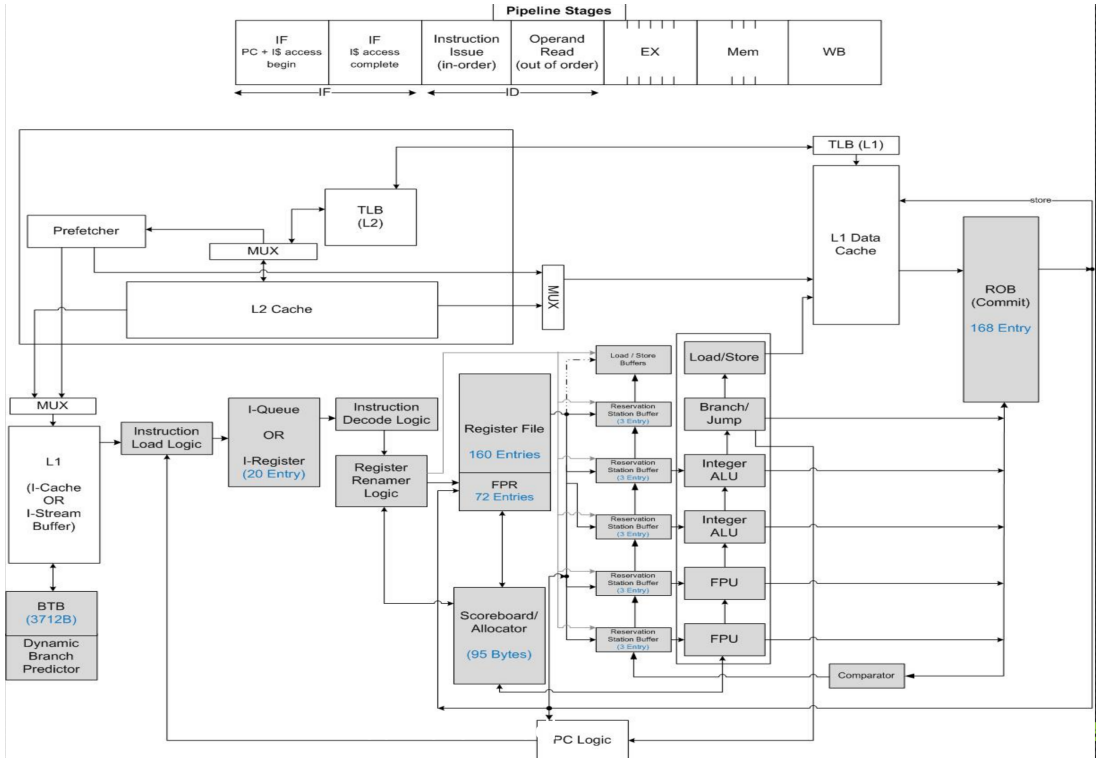
Moore's law holds, Dennard scaling is dead.

"Moore's law gives ut more transistores, Dennard scaling makes it useful",
Bob Colwell, chief engineer P. II, P. III, P. 4, Intel;
Director of the Microsystems Technology Office, DARPA.



Towards massively parallel future

Many problems: power dissipation wall, dark silicon, etc.
CPU: 1700 pJ/Op, GPU: 160 pJ/Op, ASIC: < 10 pJ/Op.



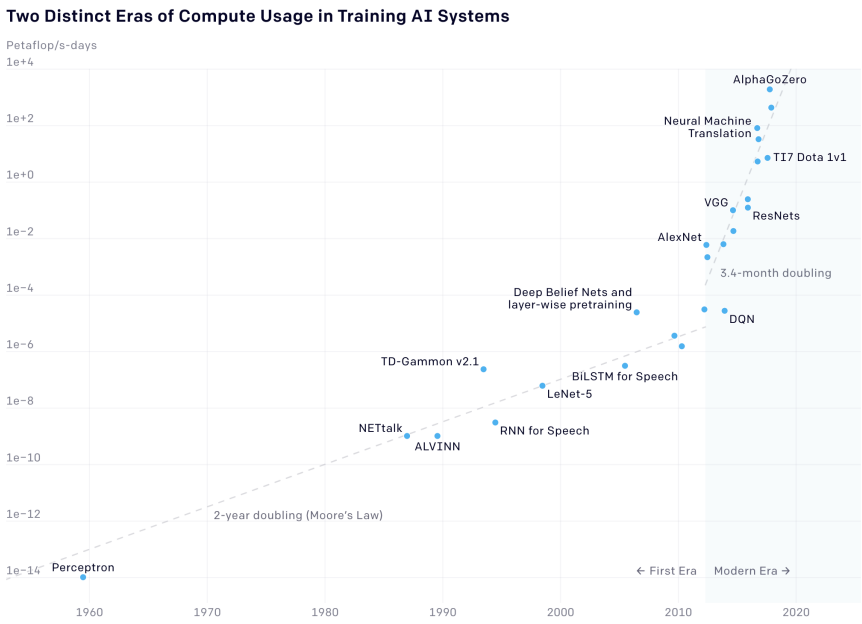
Do like the brain: low frequency, high latency, highly parallel, very large computation power, very energy efficient.

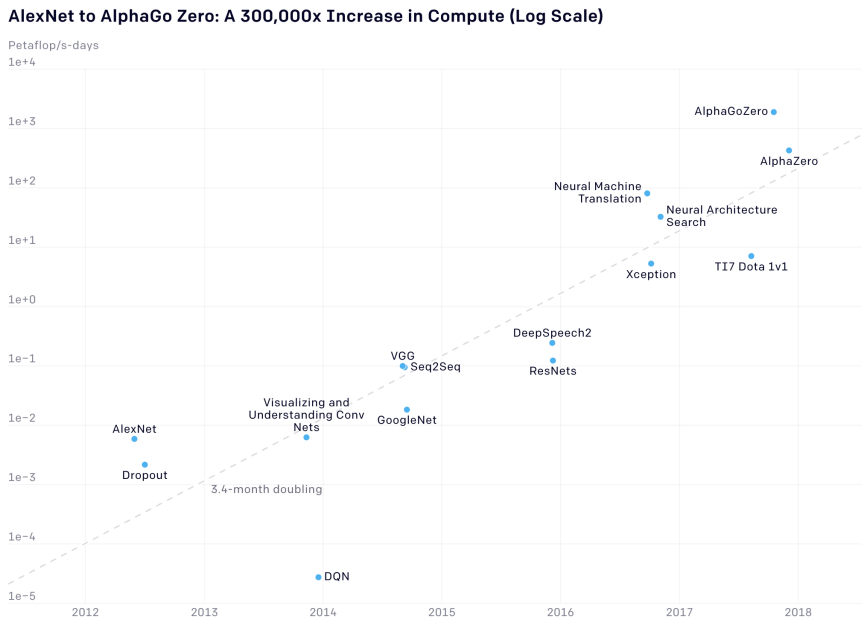
Need adapted algos to take advantage of computing power increases.

AI and compute

"AI and compute", Dario Amodei et. al., OpenAI blog, 2018.

"[...] since 2012, the amount of compute in the largest AI training runs has been increasing exponentially with a 3.4-month doubling time (by comparison, Moore's Law had a 2-year doubling period)".





CPU to GPU to large-scale GPU clusters to large-scale ASIC clusters.

ANNs and computational power

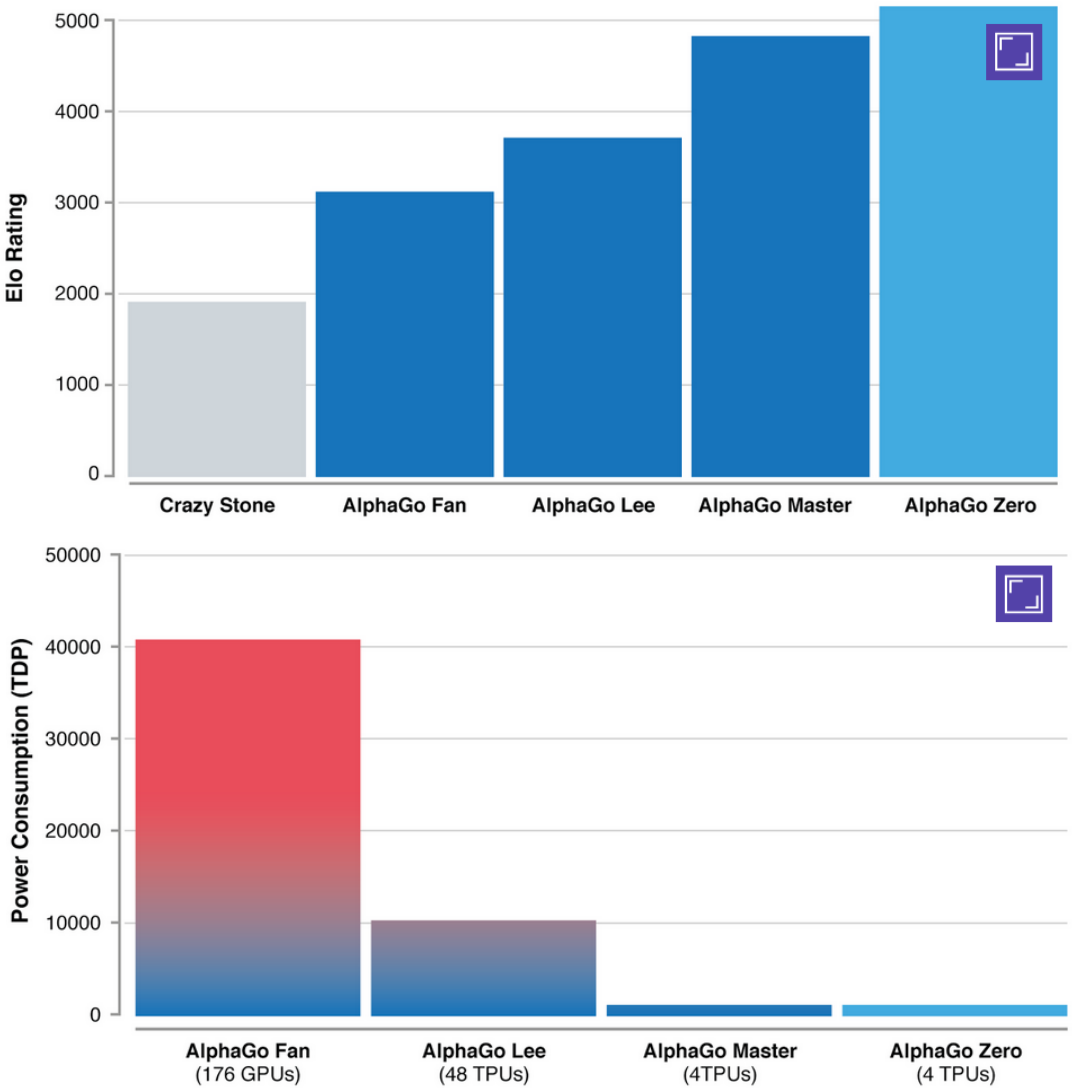
Difficult to take full advantage of large parallel systems:

- Some problems cannot be parallelized.
- Parallelizable problems may be hard to make scale.
- HPC is quite hard.

But ANNs / DRL are naturally well adapted to such architectures.

- Based on small computational units.
- Batch updates is naturally parallel.
- An ASIC ANN design can be used to solve many problems thanks to the generality of ANNs.

Increasing efficiency of learning algorithms



Figures from Google Brain blog.

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 recent works suggest 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).

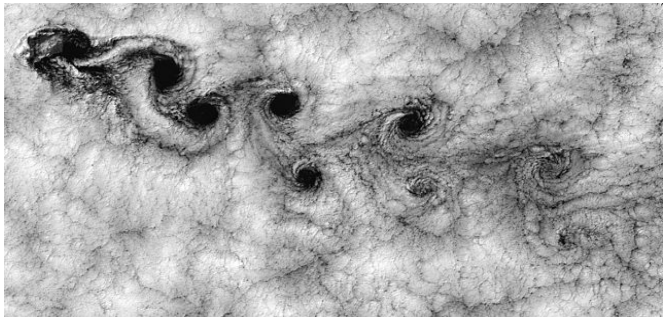
'Accelerating Deep Reinforcement Leaning strategies of Flow Control through a multi-environment approach', Rabault and Kuhnle, PoF (2019).

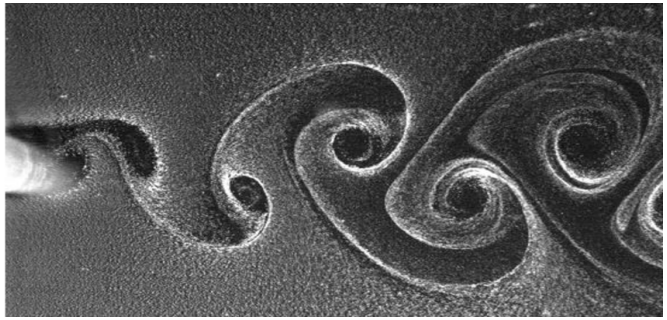
'Control of chaotic systems by Deep Reinforcement Learning', Bucci et. al., PoRS-A (2019).

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

Karman vortex street and the Turek benchmark

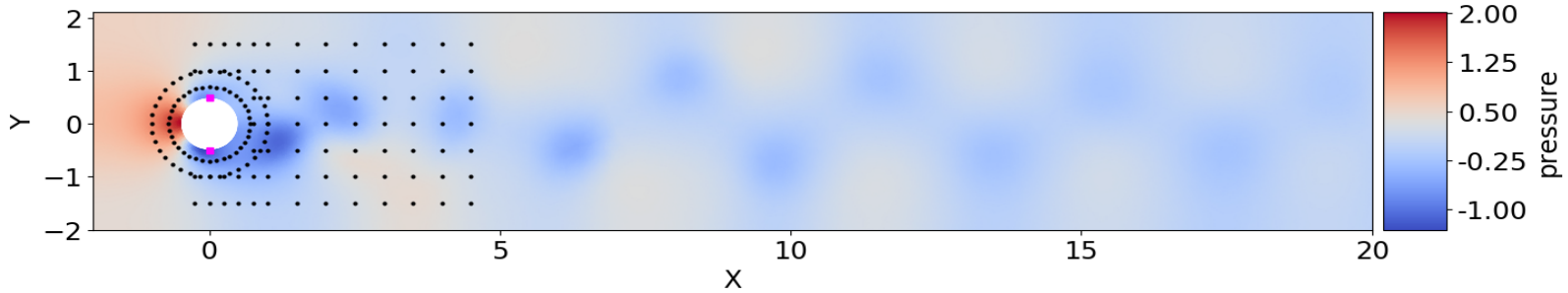
Karman vortex street is emblematic of Fluid Mechanics.





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).



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

Re 100

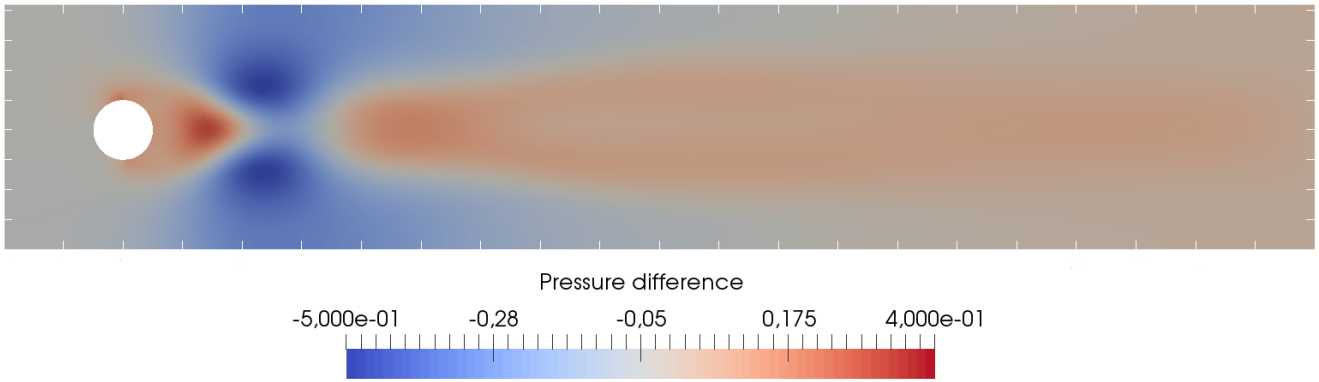
'Artificial Neural Networks trained through Deep Reinforcement Learning discover control strategies for active flow control', Rabault et. al., JFM (2019).

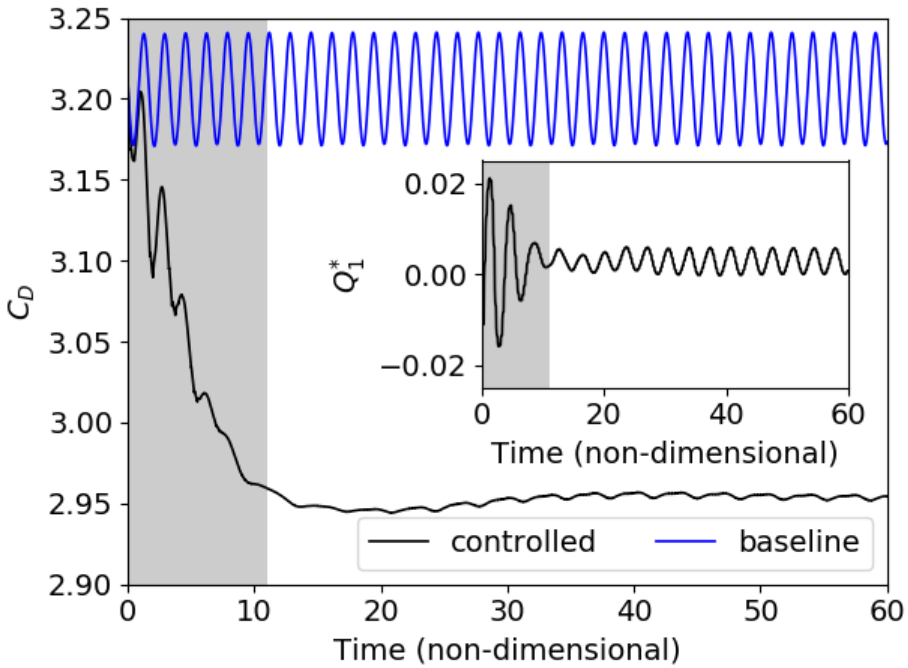
https://youtu.be/0QbKk5SzMws

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

What is happening?

Very small jets allow for 'boat tailing'. Already solving a difficult task!





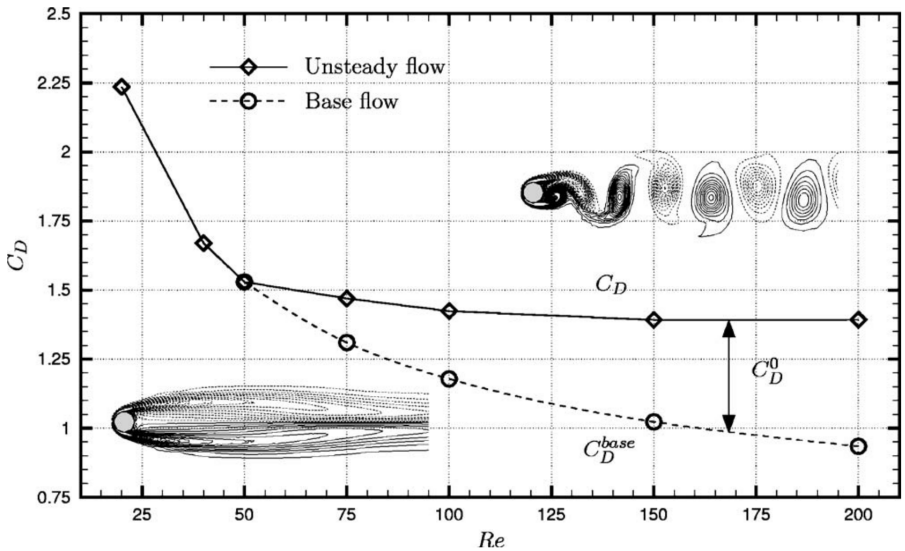
How good is the strategy? (I)

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

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

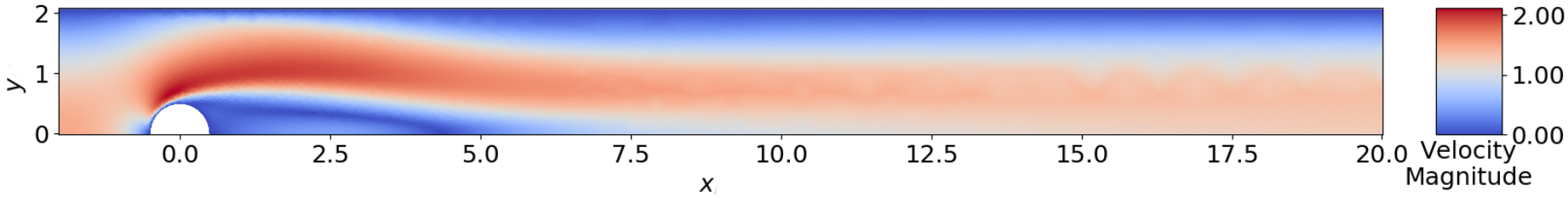
$$C_D = C_D^{baseflow} + C_D^{shedding}$$

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):



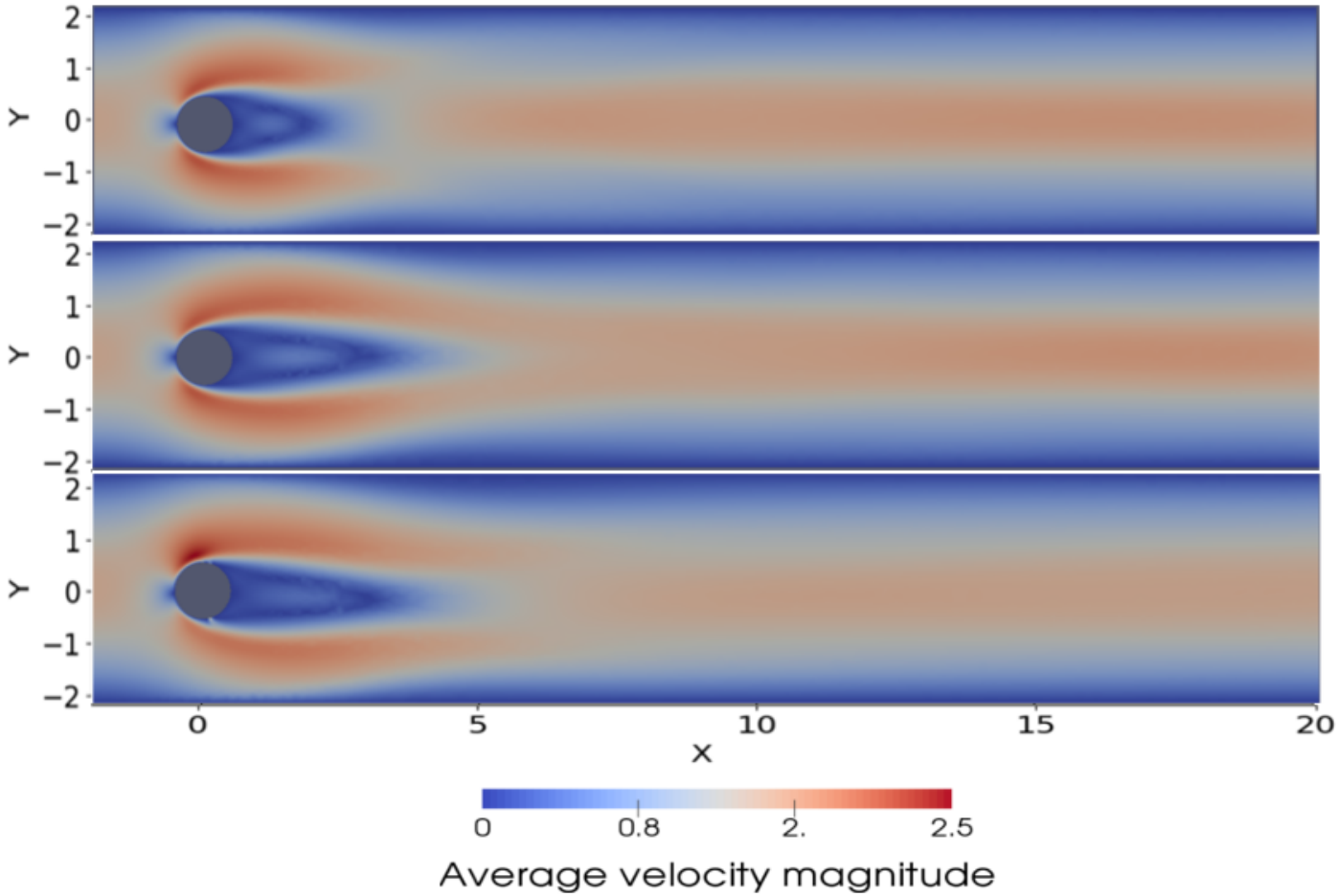
$2C_D^{half-domain} = 2.93, \quad < C_D^{nocontrol} >= 3.205, \quad < C_D^{control} >= 2.95.$

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

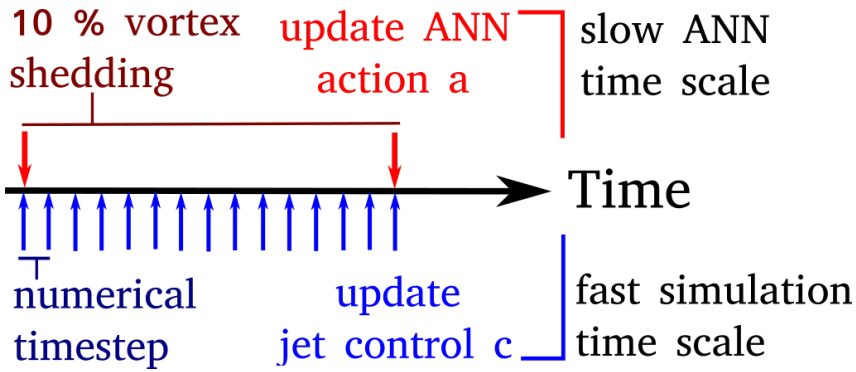
We still need engineers (I)

Choice of the reward function: prevent 'cheating'.

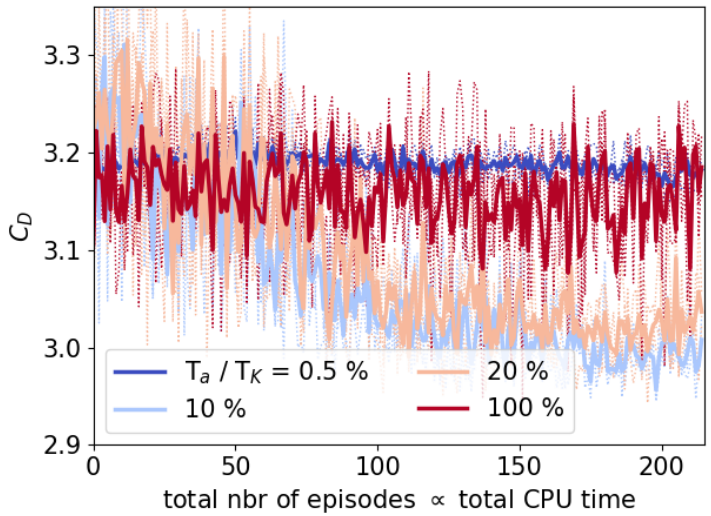
$$r = \langle D \rangle_S - \alpha |\langle L \rangle_S|, \tag{1}$$



We still need engineers (II)



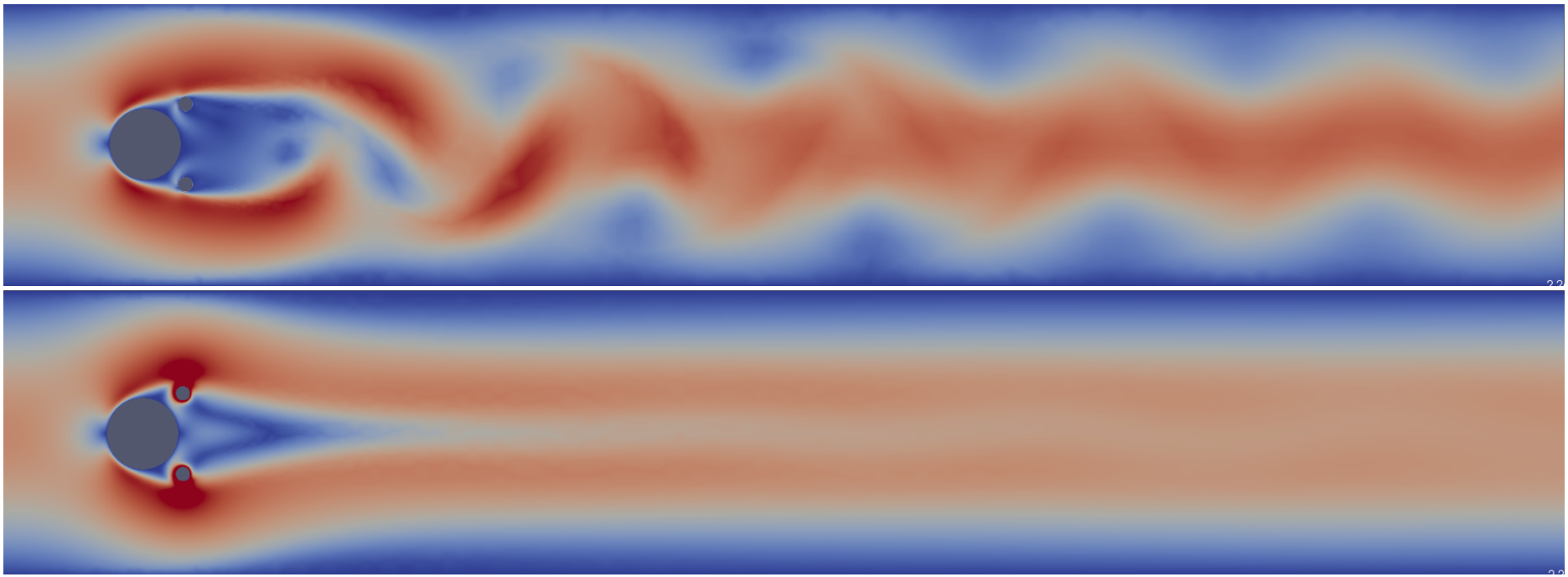
Choose the right characteristic timescale!



Of course, can predict action duration...

Flexible method

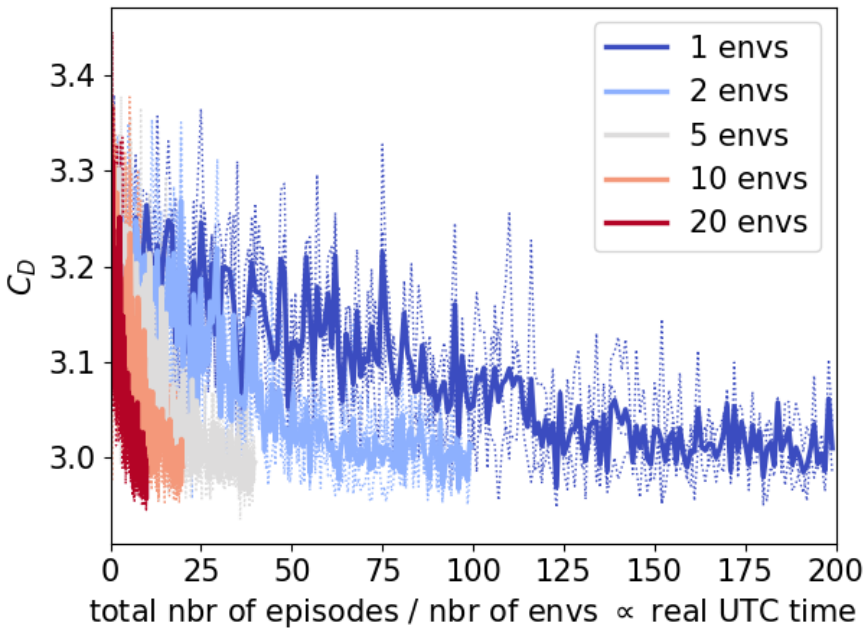
Flexible to look at different cases: small control cylinders (credit: Wei Zhang).

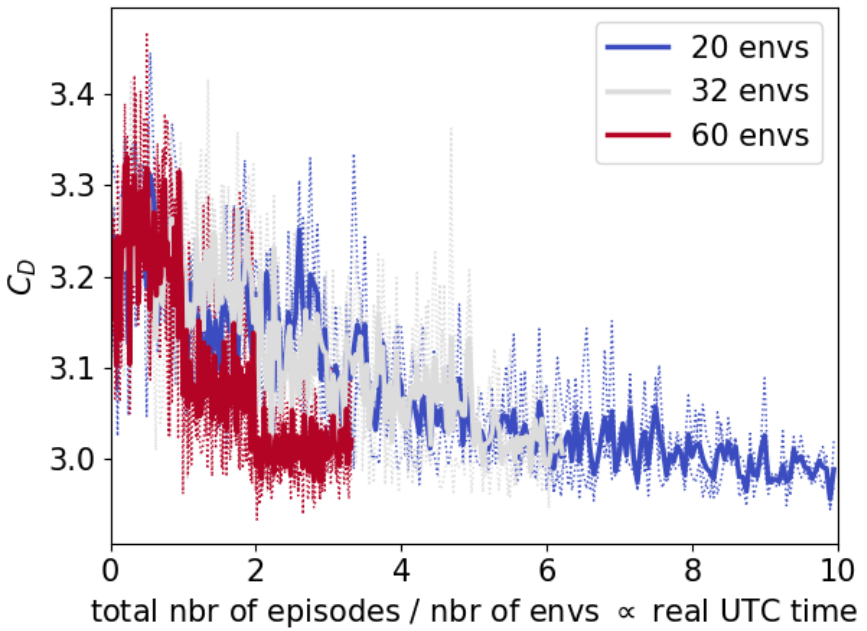


Challenge I. Towards higher Re: faster learning

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`

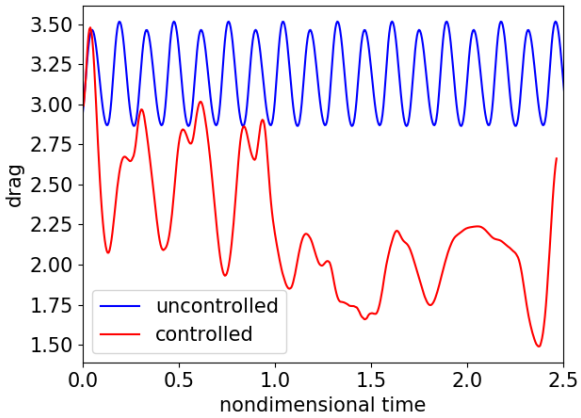
Controlling more challenging flows: Re 500

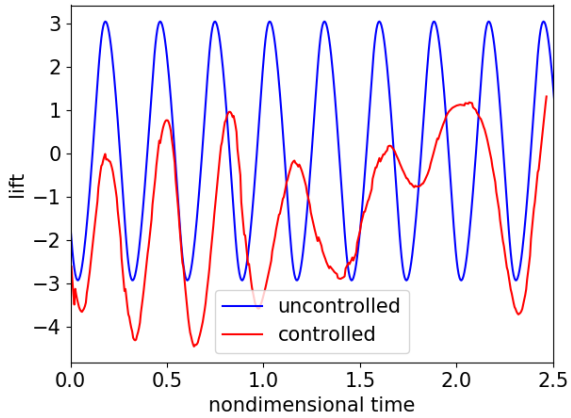
Ongoing work, collaboration with Nanjing Univ. of Aero. and Astro.
Much stronger separation, so stronger jets and more unstable - but some promising preliminary results.

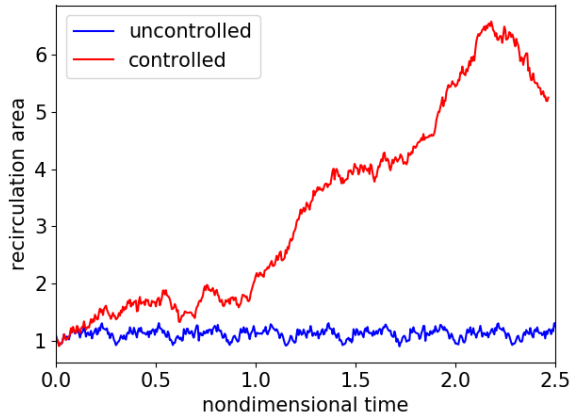
https://youtu.be/PDyr6CHIwVk

Complex dynamics at Re 500

Already at Re 500, the dynamics start to be more complex:





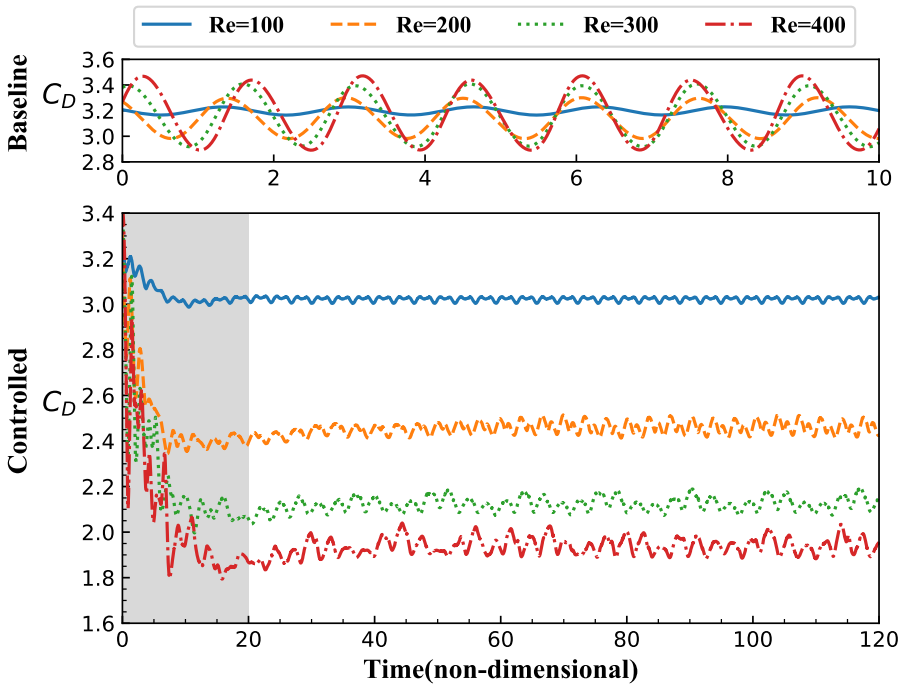


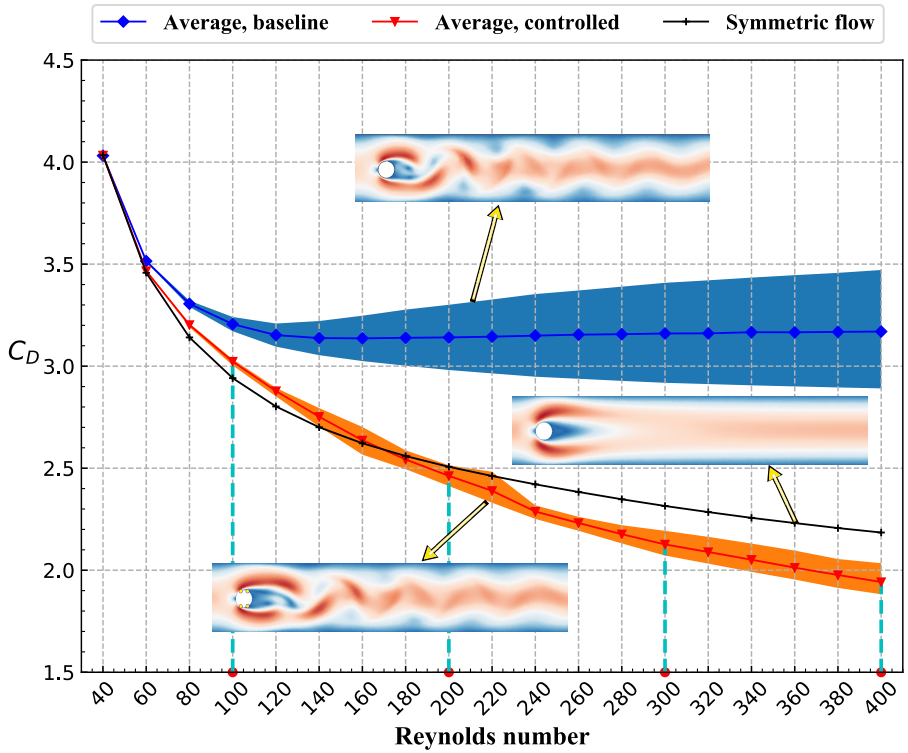
- Very large changes in recirculation area.
- Non pseudo-periodic behavior, large frequency shift.
- Violent instability and collapse of the wake.

Currently working on keeping stable control also for later times. Going to higher Re / 3D flows will require another solver.

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.

Using another solver for higher Res: LBM

Collaboration Hong Kong Polytechnic Univ.
R. Feng, J. Rabault, H. Tang (under writing).

Combine parallelism at simulation level and faster CFD:

- Reproduce (much quicker) Re 100 results.
- Extend to higher Re: 1000 (but in 2D).
- Strongly pseudo-chaotic.

HiFiCoMa 2019

THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學
DEPARTMENT OF MECHANICAL ENGINEERING
機械工程學系

Active Flow Control of Flow Past a Circular Cylinder at Moderate Reynolds Number Using Deep Reinforcement Learning

Feng REN[1, *], Jean Rabault[2], Hui Tang[1, #]

* feng.ren@polyu.edu.hk ; #h.tang@polyu.edu.hk

[1]Department of Mechanical Engineering, The Hong Kong Polytechnic University

[2]Department of Mathematics, University of Oslo

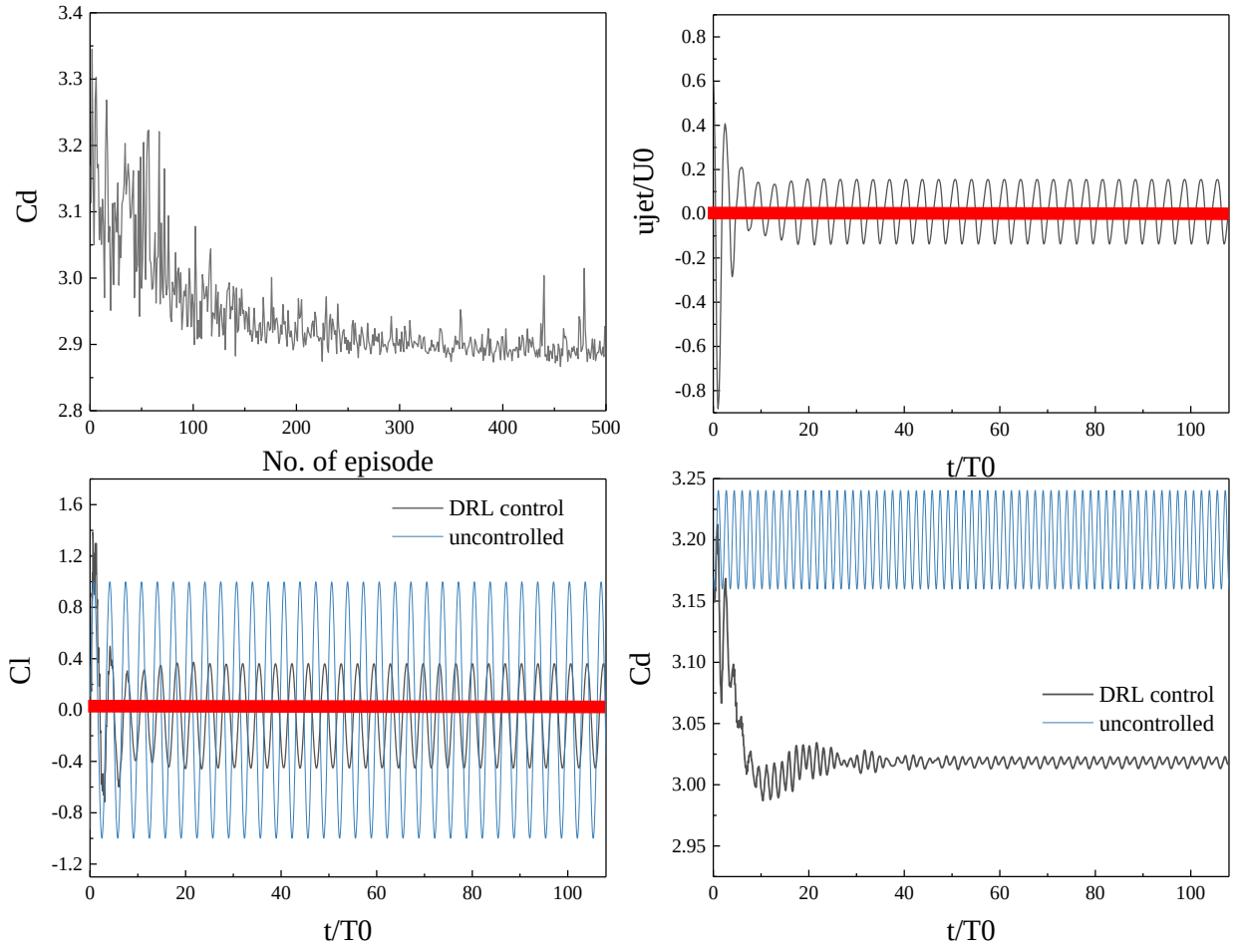
Dec 15, 2019 Shanghai

2D Flow at Re=100

To improve the control, we adjusted the lift penalization coefficient, i.e., r_t=-<C_d>-|<C_l>|

Lift bias is eliminated greatly.

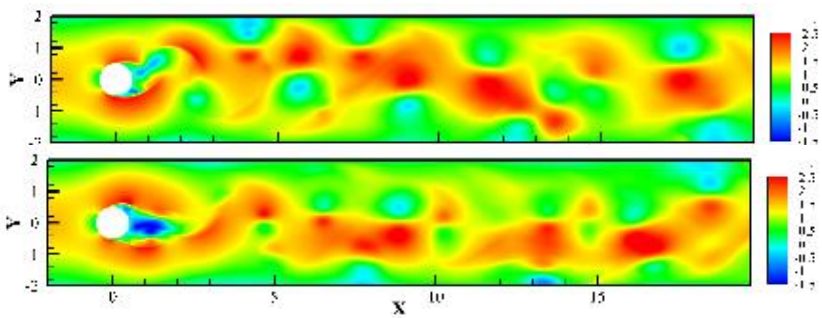
Note that the control frequency is about 10% smaller than that of the uncontrolled vortex shedding frequency



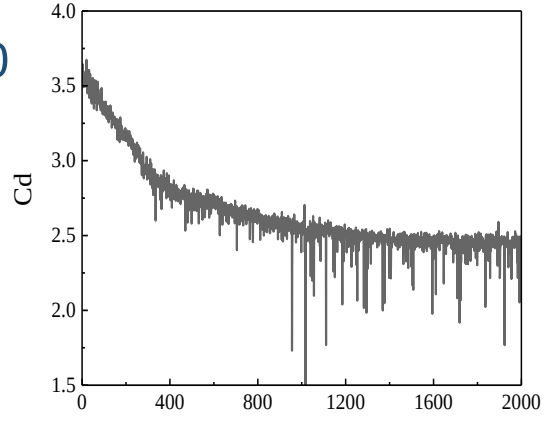
2D Flow at Re=1,000

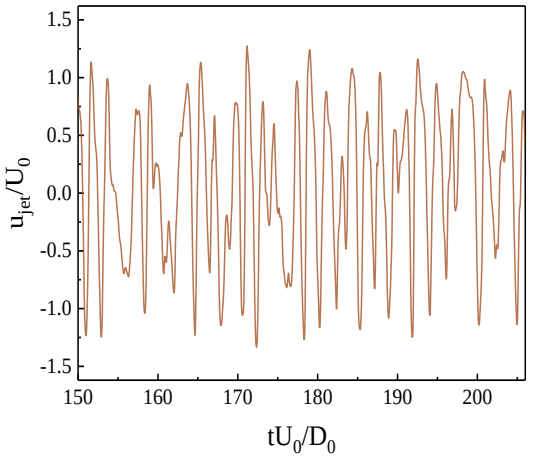
Training 2:

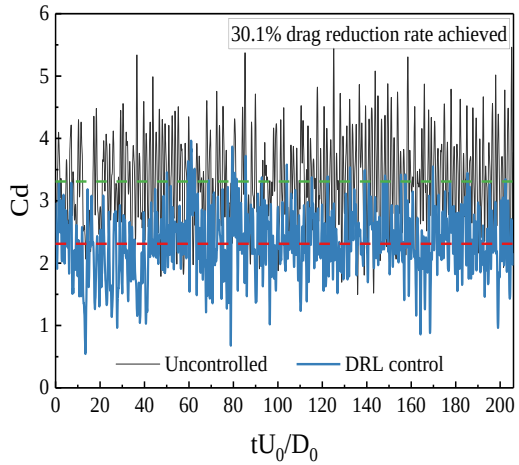
- Start from trained policy at Re = 100
- Best control is found at Ep=1289
- Cd is reduced by 30.1%
- Cl fluctuation is reduced by 29.4%
- Mean lift bias is small, i.e., -0.12

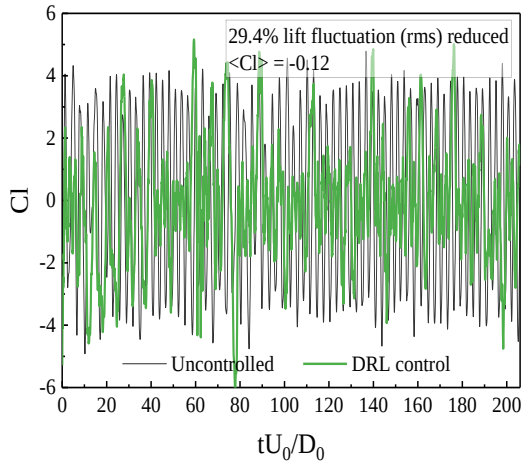


Instantaneous streamwise velocity field for uncontrolled (up) and DRL-controlled (down) case







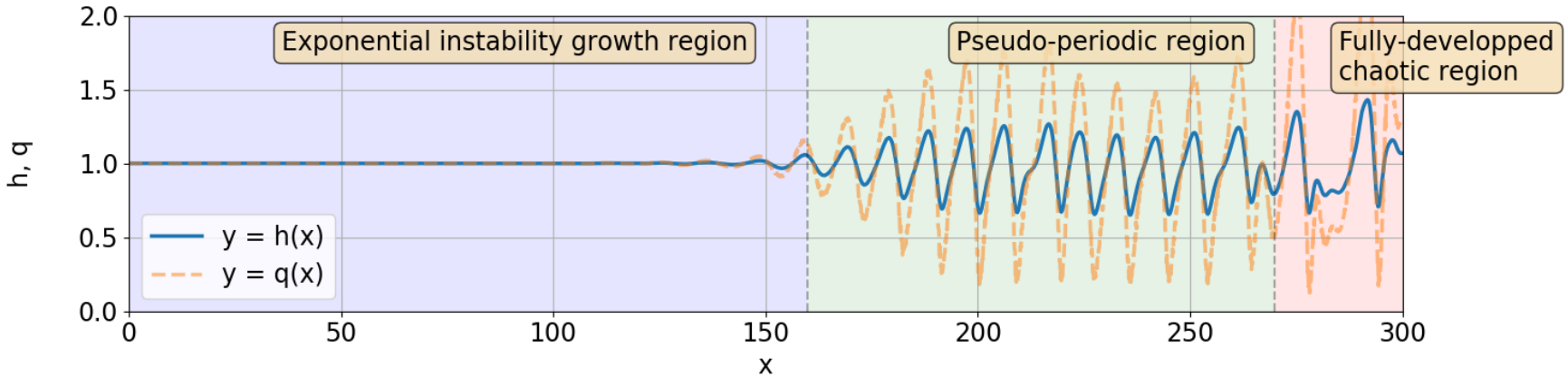


Challenge II. How to increase the number of controls? Example of falling fluid film.

"Exploiting locality and physical invariants to design effective Deep Reinforcement Learning control of the unstable falling liquid film", Belus et. al., ArXiv (2019), accepted AIP-A.
Use simulation from "An ensemble method for sensor optimisation applied to falling liquid films", Z. Che et. al., IJMFF (2014):

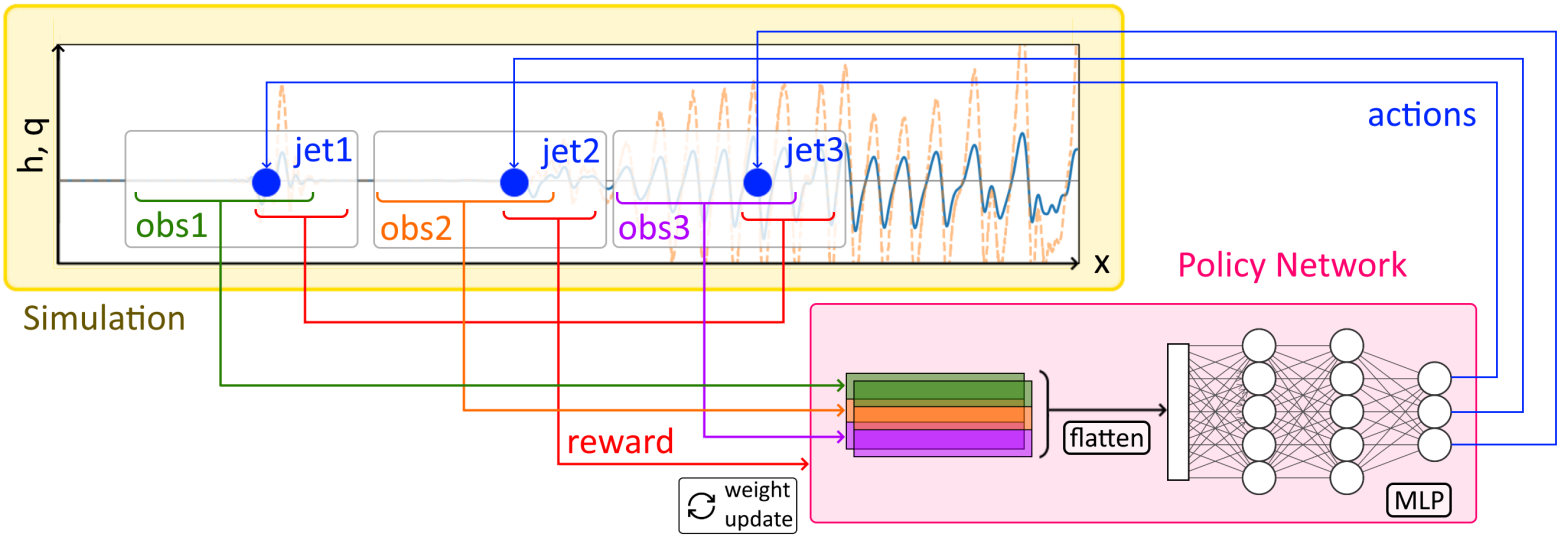
$$\frac{\partial h}{\partial t}+\frac{\partial q}{\partial x}=0,\frac{\partial q}{\partial t}+\frac{6}{5}\frac{\partial}{\partial x}\left(\frac{q^2}{h}\right)=\frac{1}{5\delta}\left(h\frac{\partial^3 h}{\partial x^3}+h-\frac{q}{h^2}\right),$$

with $\delta=(\rho H_c^{11}g^4/\sigma)^{1/3}/45\nu^2$.



How to insert several jets? (M1)

Naive approach: large inputs and outputs.



Curse of dimensionality on the number of jets:

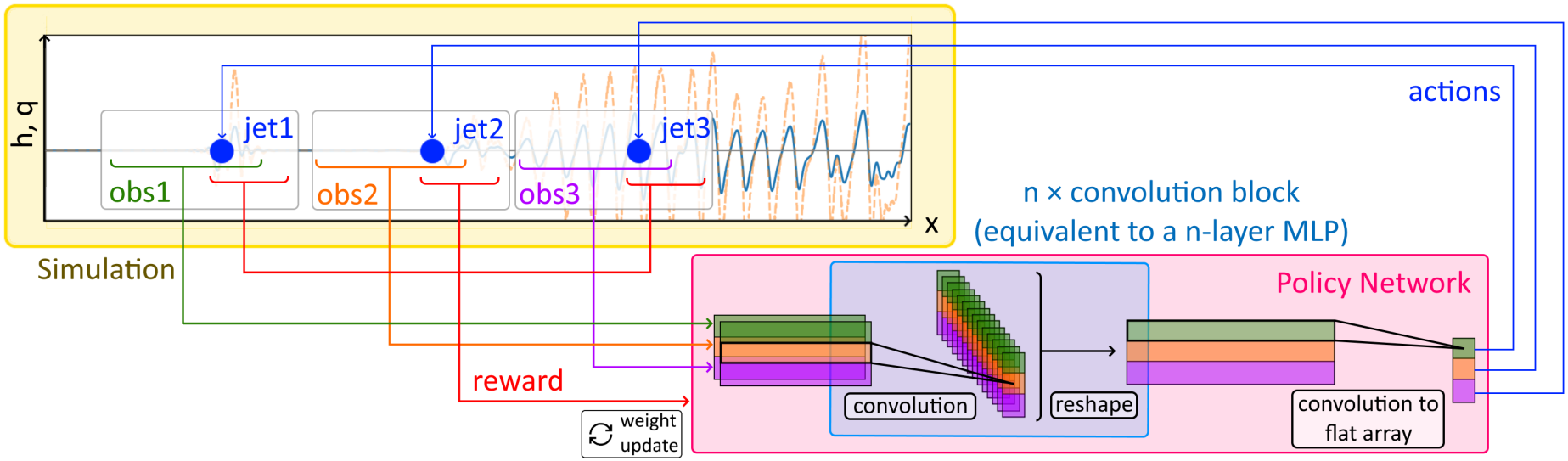
- For thought experiment, p possible jet strengths.
- Use N jets, and a fully connected ANN.
- Then, need to fully explore:

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

Of course, we can do much better...

How to insert several jets? (M2)

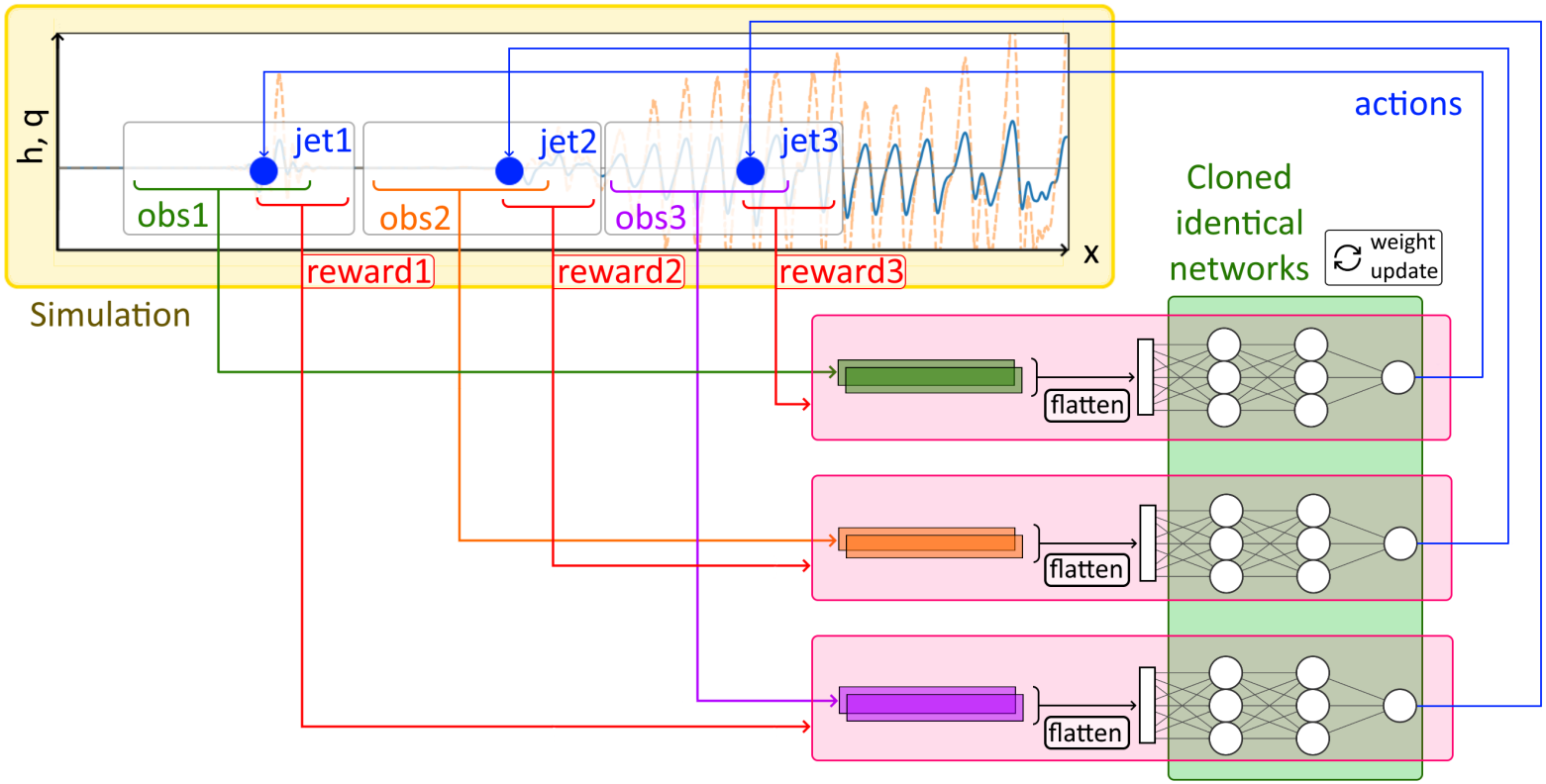
Smarter approach: use a fully convolutional network.
Respect invariance by translation.



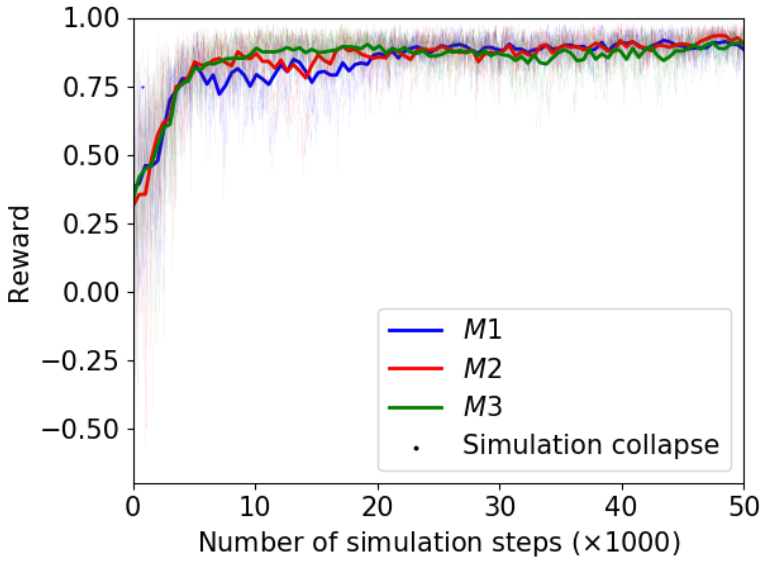
This way, truly learn only 1 set of weights. Take advantage of translational invariance.

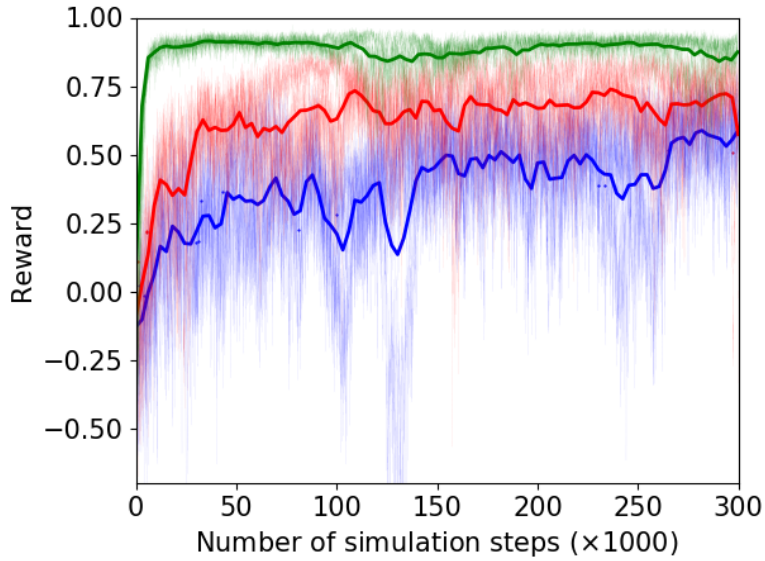
How to insert several jets? (M3)

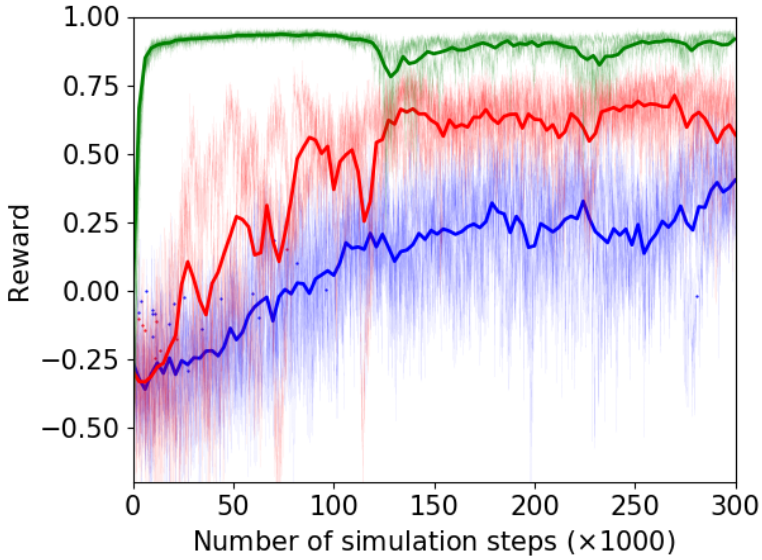
Even better: split the simulation into cloned environments.
This way, provide both invariance and more reward signal.

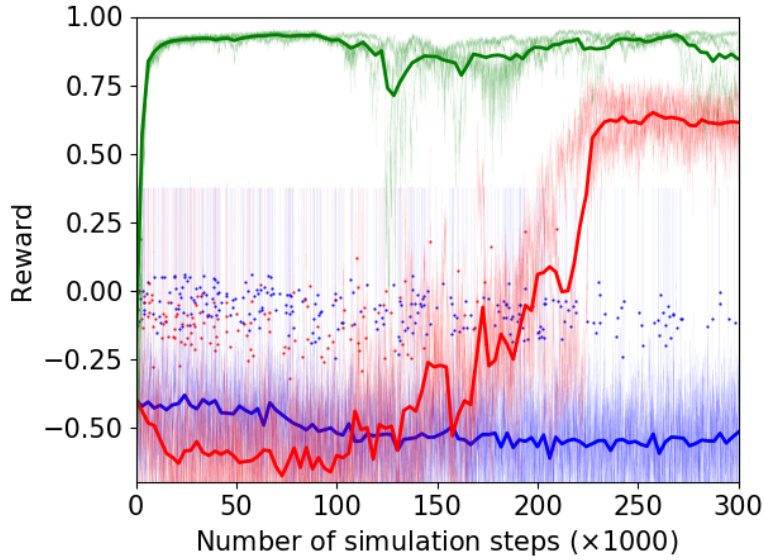


Comparing M1, M2, M3





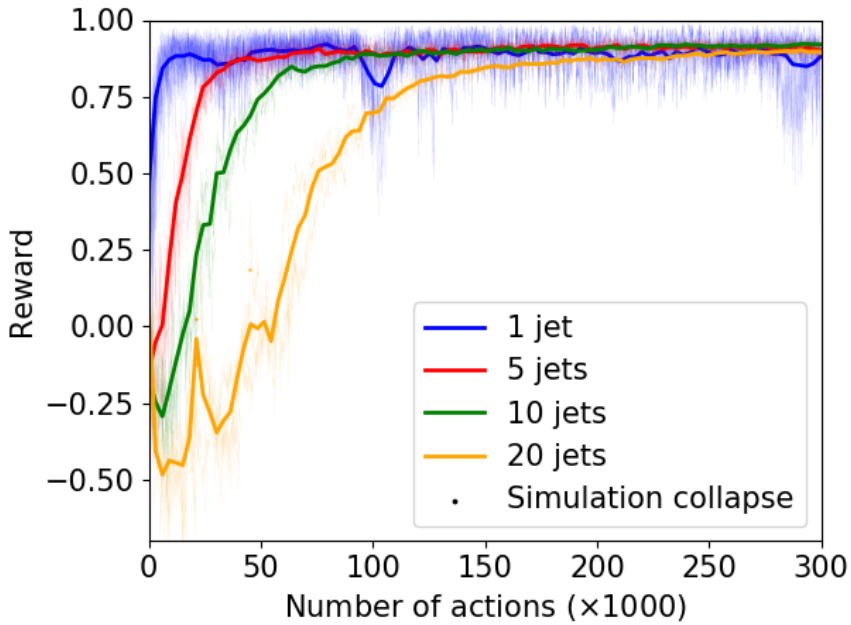


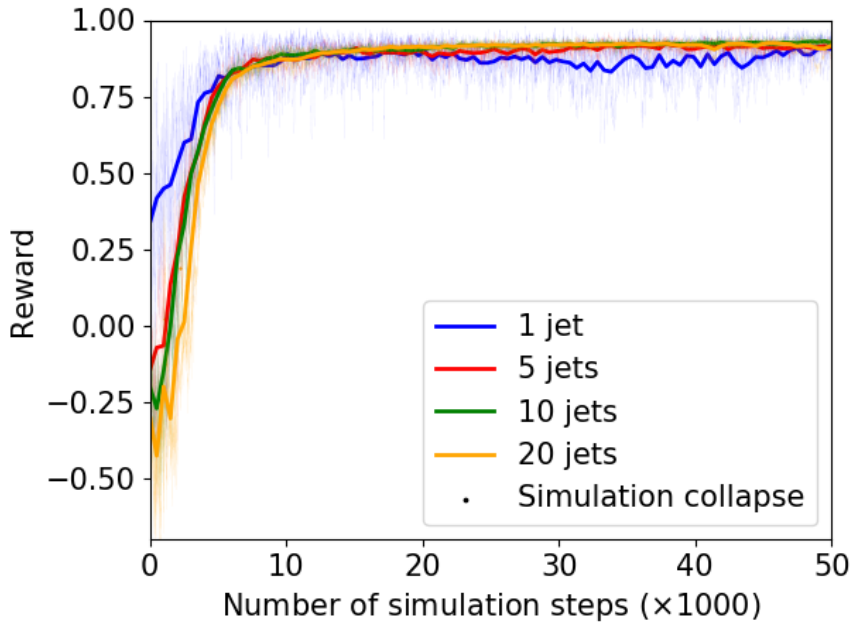


M3 can handle arbitrary number of jets

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].

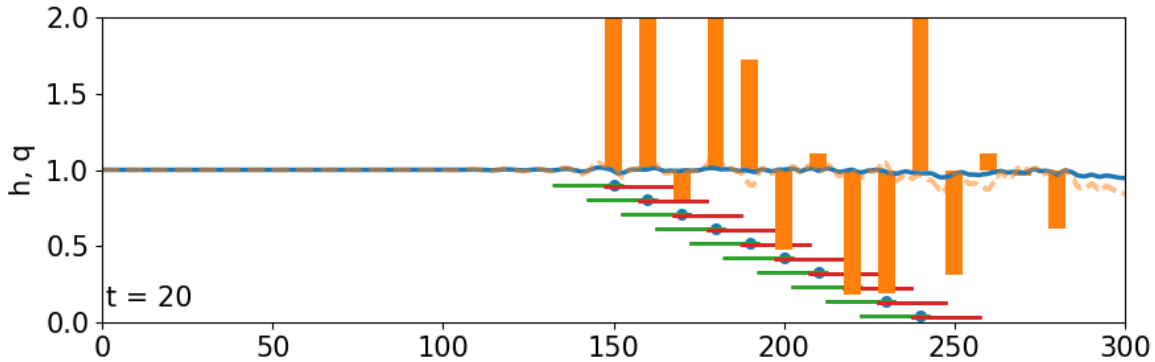




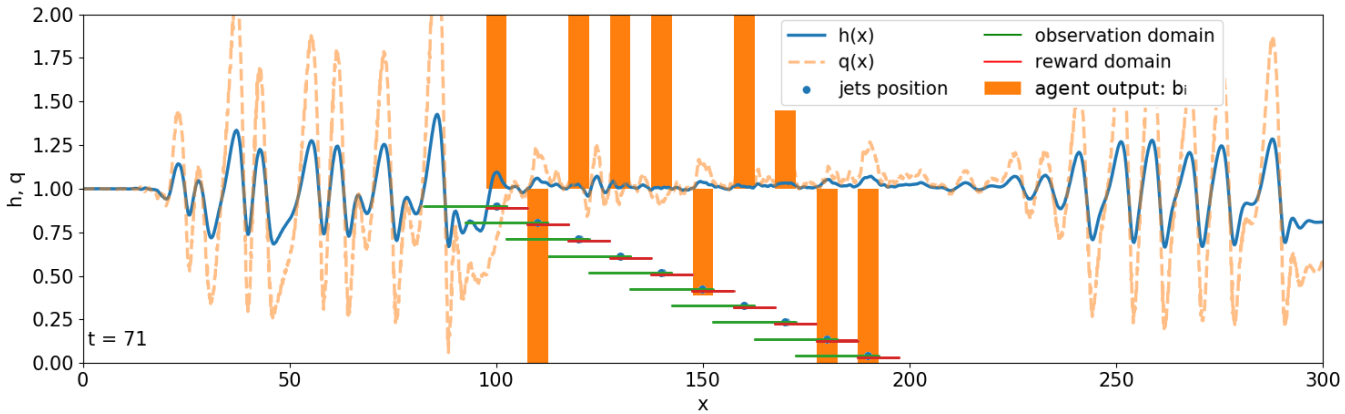
Constant cost of learning for any number of jets. Makes sense physically.

Satisfactory control

Control from the start location of instability development:



Control in an area where large interfacial waves are present:



Confronting some of the difficulties of turbulence: multi-modality with weak coupling

Difficulties with turbulence: multi-modality with weak coupling.
Nonlinear crosstalks system of 'Machine Learning Control-Taming Nonlinear Dynamics and Turbulence', Duriez, Brunton and Noack (2016):

$$\frac{da_1}{dt} = \sigma a_1 - a_2 \tag{2}$$

$$\frac{da_2}{dt} = \sigma a_2 + a_1 \tag{3}$$

$$\frac{da_3}{dt} = -0.1a_3 - 10a_4 \tag{4}$$

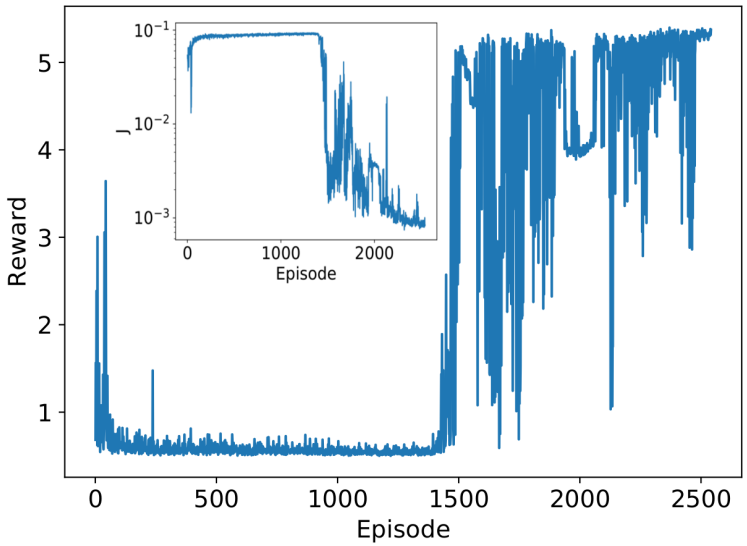
$$\frac{da_4}{dt} = -0.1a_4 + 10a_3 + b \tag{5}$$

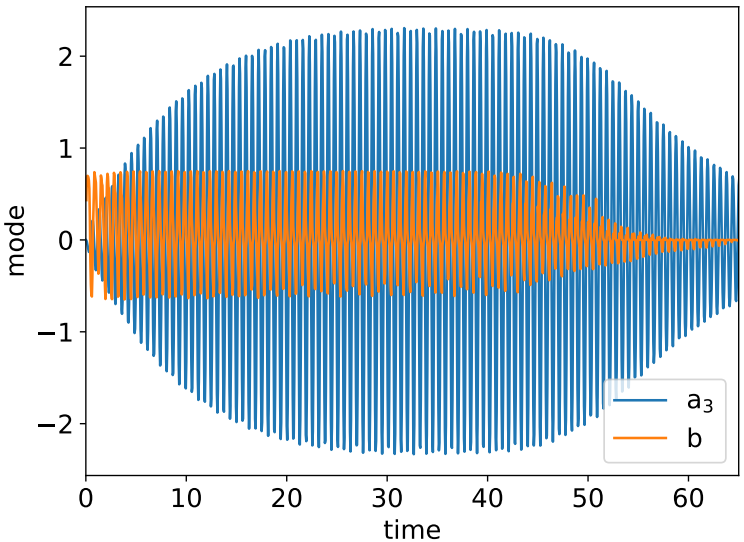
$$\sigma = 0.1 - a_1^2 - a_2^2 - a_3^2 - a_4^2, \tag{6}$$

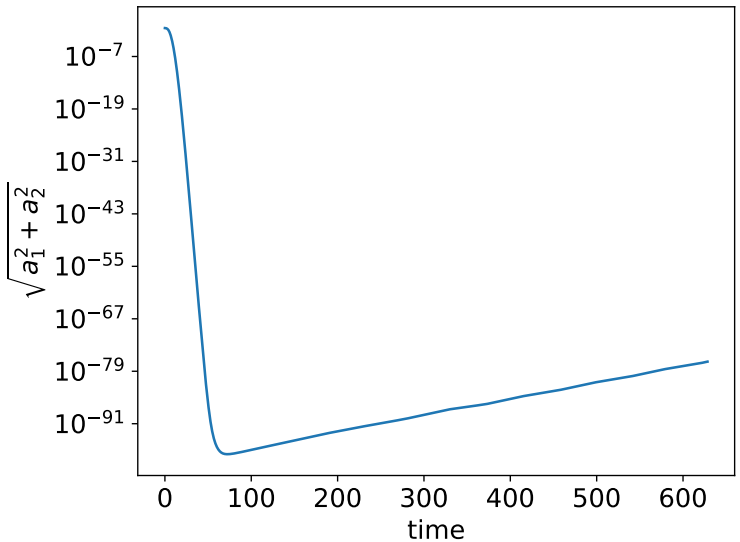
Coupling disappears if linearize in a state where initially $a_3 = a_4 = 0$.
Measure quality of control: $J = \overline{a_1^2 + a_2^2} + \gamma \overline{b^2}$.

Results

Manage to control with DRL:







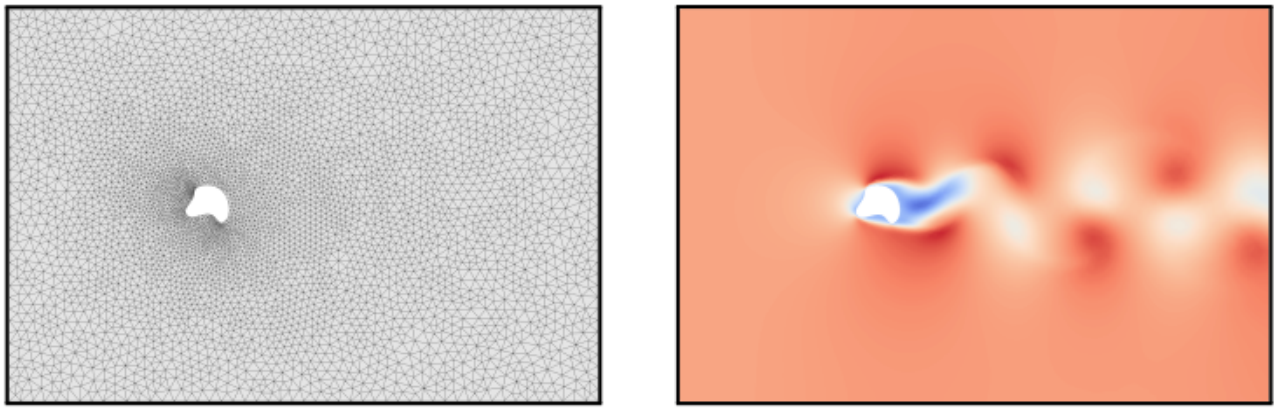
DRL for shape optimization

Can we learn to produce lift in a simulation?

We take wings for granted, but took a long time to 'discover' / 'invent'. Do we really know that a wing is the best solution?

Use DRL to produce lift.

J. Viquerat, J. Rabault, et. al. "Direct shape optimization through Deep Reinforcement Learning"



(a) Mesh of the computational domain

(b) Computed v_x velocity field at $Re = 200$

Figure 2: **Mesh and computed v_x velocity field at $Re = 100$** behind a random shape. The recirculation area behind the obstacle is clearly visible, followed by a well-established Von Karman vortex alley. The velocity field is scaled in the $[-1, 1]$ range for displaying purposes.

Shape parametrization

- Bezier curves.
- Start from a cylinder.
- Each point allowed to be moved in a torus domain.
- Punish 'bad', i.e. non meshing, shapes.

Figure 3: **Example of generated shape with geometrical constraints.** The colored dots indicate the control points generated by the agent, that are then joined using Bézier curves. Each point the agent suggests is restricted by construction in radius (inner and outer circles of radii $r_{\min}$ and $r_{\max}$) and in azimuth (diverging white lines).

Reward and normalization

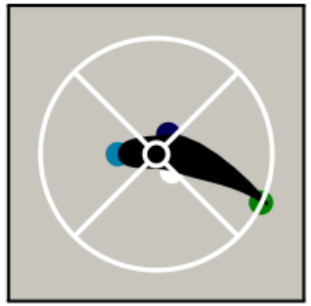
- Need a 'normalized' reward to avoid simple blowing up of the shape.
- Normalize inputs / outputs to keep ANN inputs / outputs in [-1, 1].

$$r_t = \left\langle \frac{C_l}{|C_d|} \right\rangle - \left\langle \frac{C_l}{|C_d|} \right\rangle_{\text{cyl}}, \tag{7}$$

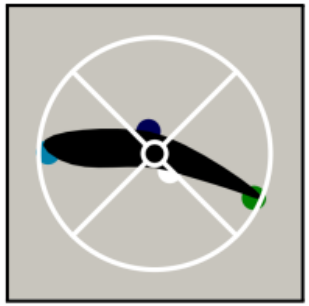
$$\begin{cases} r = r_{\max} \max\left(|p|, r_{\min}\right), \\ \theta = \dfrac{\pi}{n}\left(i + \dfrac{q}{2}\right), \\ x = r\cos\left(\theta\right), \\ y = r\sin\left(\theta\right), \\ e = \dfrac{1}{2}\left(1 + r\right). \end{cases} \tag{8}$$

Results

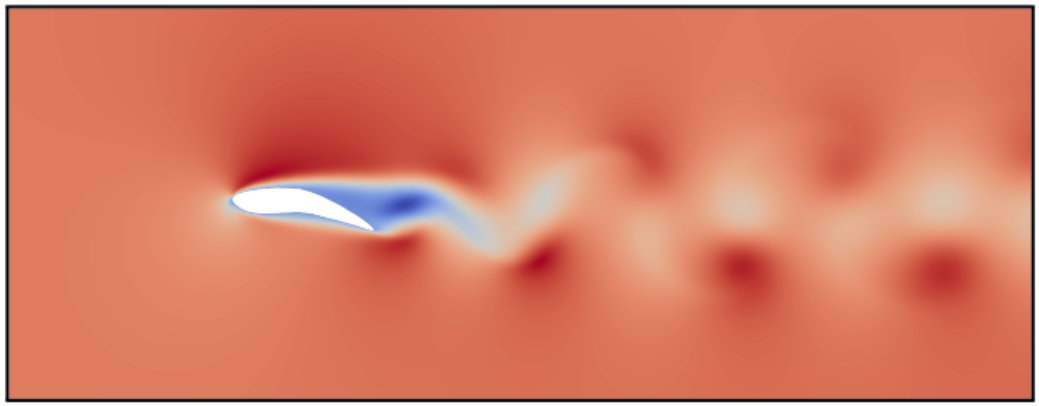
(a) Best shape with 4 points, 1 free point (3 d.o.f.s)



(b) Best shape with 4 points, 3 free points (9 d.o.f.s)



(c) Best shape with 4 points, 4 free points (12 d.o.f.s)



(d) Computed v_x velocity field at $Re \sim 600$ around shape 5c (the domain is cropped).

Figure 5: **Results of the baseline shape optimization process**. Best shapes obtained using 1, 3 and 4 free points are shown in subfigures 5a, 5b and 5c respectively. In subfigure 5a, the left, top and bottom points are fixed to their initial position (*i.e.* that of the reference cylinder), while the rightmost one is free to move. In subfigure 5b, only the left point is fixed, while in subfigure 5c, all four points are free to move. The velocity field corresponding to shape 5c is shown in subfigure 5d.

Future improvements

- Use a level-set line to deform mesh.
- ANN sees many configurations and optimize.

We want to use the ANN to produce an 'intuition'. Think about the drag reduction on a truck:

- Strongly nonlinear: changing the front affects the flow at the back.
- Gradient descent adjoint method do not work: too many local extrema.
- For now: human experts with years of experience.

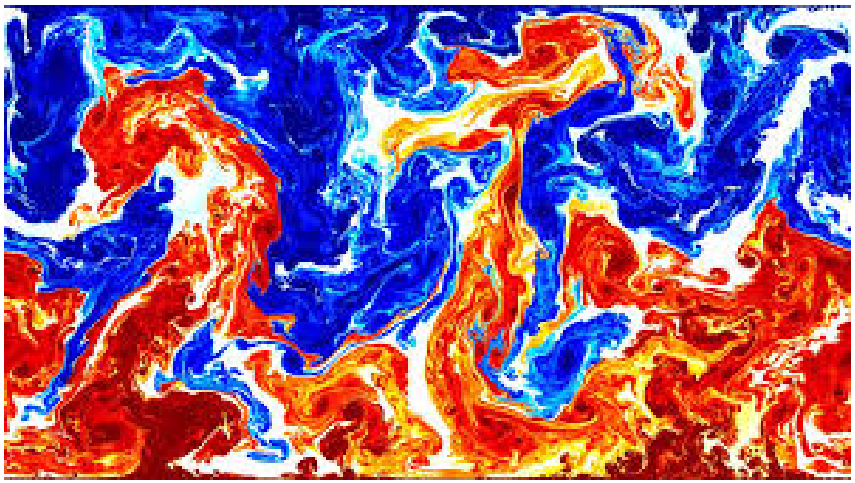
DRL would scale better than human experts.

Other applications in the literature

i. Rayleigh-Benard instability control

Rayleigh-Benard instability and manufacturing of monochristals

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



https://www.youtube.com/watch?v=Ju7JyFgvqFc

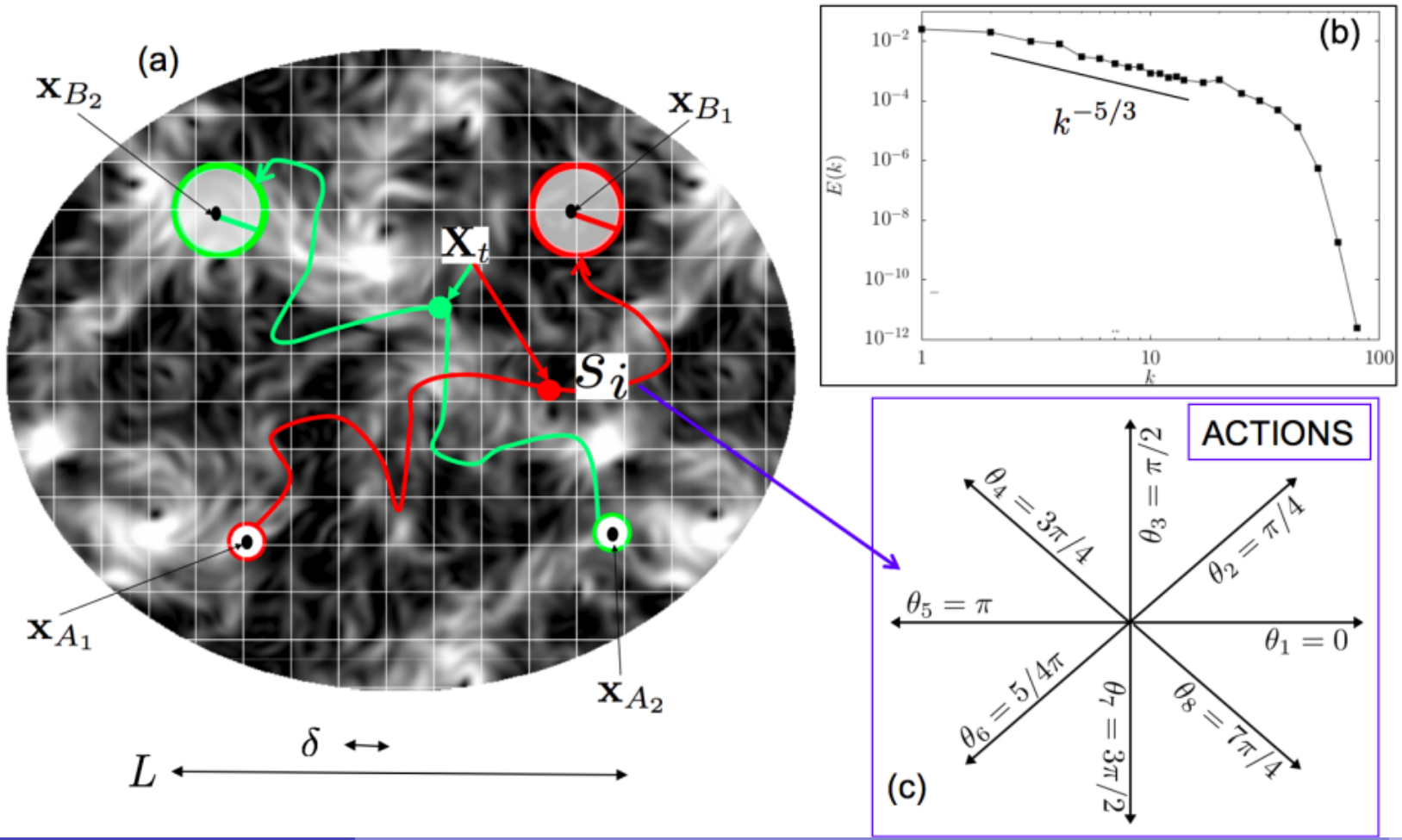
https://www.youtube.com/watch?v=UjGd1if99jg

Successful control while traditional methods do not work.

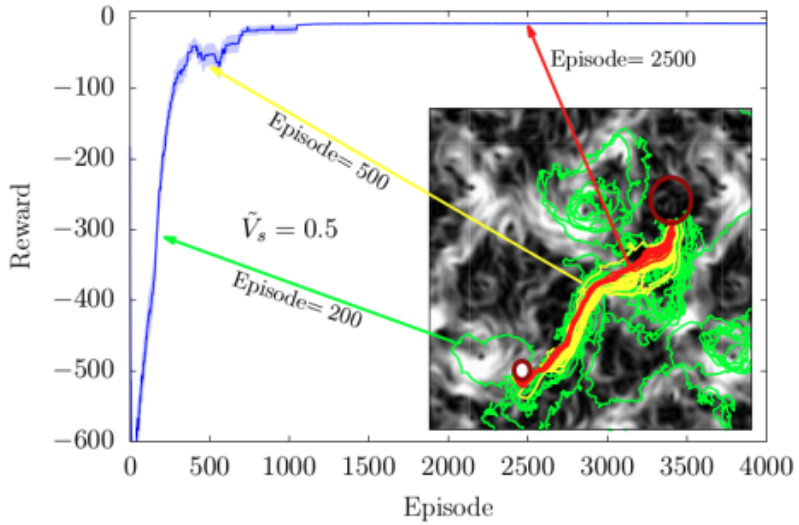
ii. Optimal navigation: the Zemelo problem

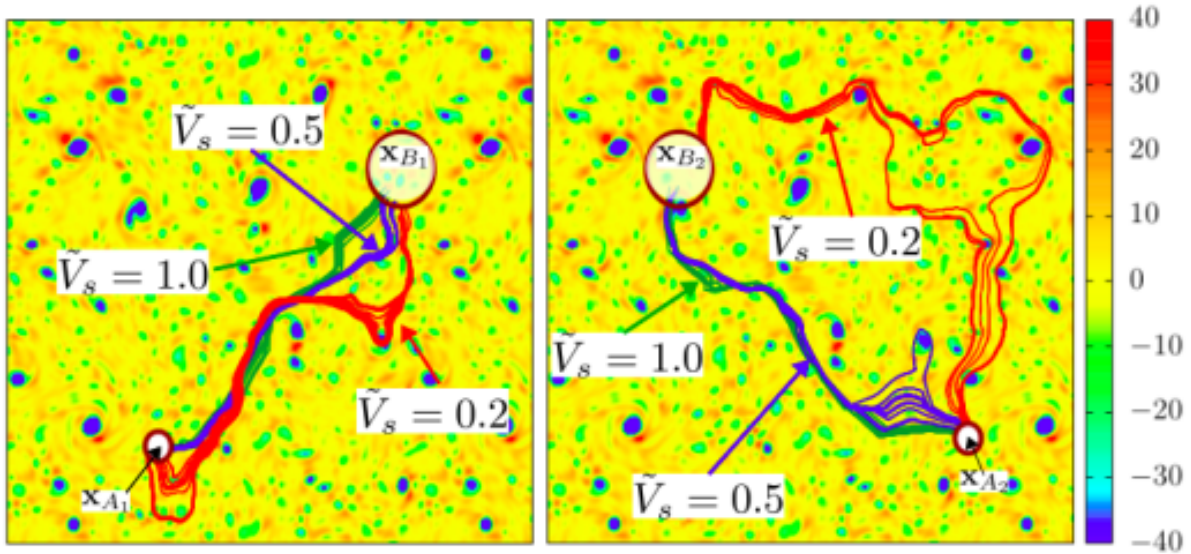
Optimal traveling

"Zermelo problem: Optimal point-to-point navigation in 2D turbulent flows using Reinforcement Learning", Biferale et. al., Chaos: Interdisciplinary Journal of Nonlinear Science 29:10 (2019).



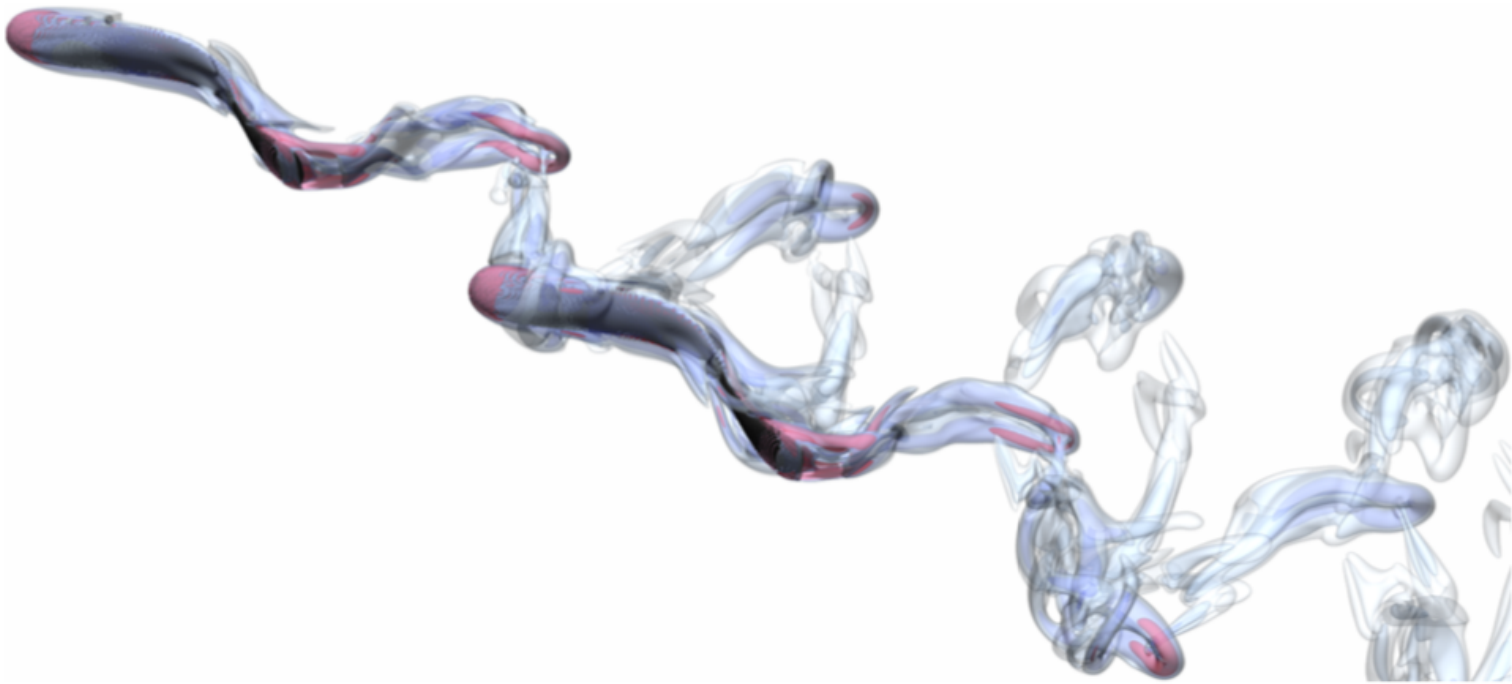
Optimal traveling





iii. Optimal swimming of fishes

Group swimming



"Efficient collective swimming by harnessing vortices through deep reinforcement learning", Verma et al., PNAS (2018)

iv. Control of chaotic systems

Control of chaotic systems

"Control of chaotic systems by deep reinforcement learning", Bucci et. al., PoRSA (2019).
Control the 1D Kuramoto-Sivashinsky model using DDPG.

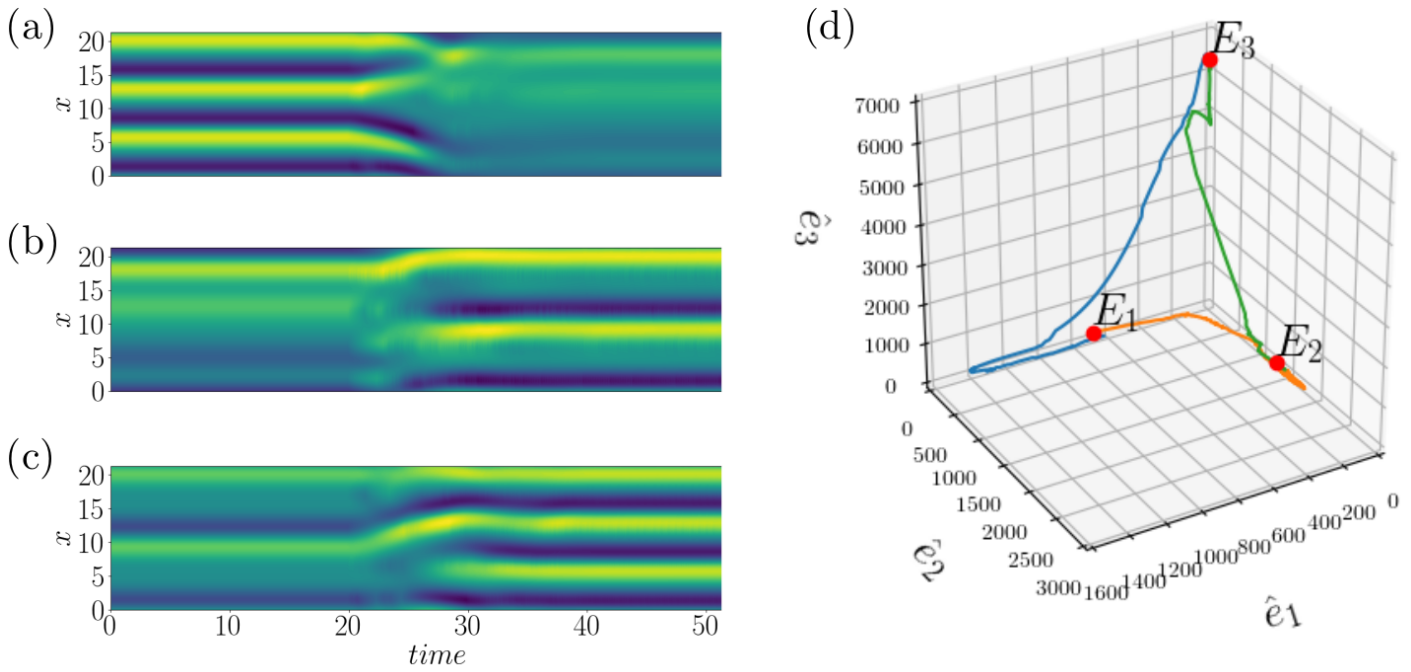


Figure 4: Closed-loop dynamics for the three control test cases: (a) $E_3 \to E_1$, (b) $E_1 \to E_2$ and (c) $E_2 \to E_3$. The controller is switched on at $t = 20$. The control strategy used in each test case is the optimal RL policy. The trajectories are shown in the Fourier phase-space in (d).

v. Control of AUVs

Attitude control of drones

"Deep Reinforcement Learning Attitude Control of Fixed-Wing UAVs Using Proximal Policy Optimization", Bohn et. al., 2019 International Conference on Unmanned Aircraft Systems (ICUAS). IEEE (2019).

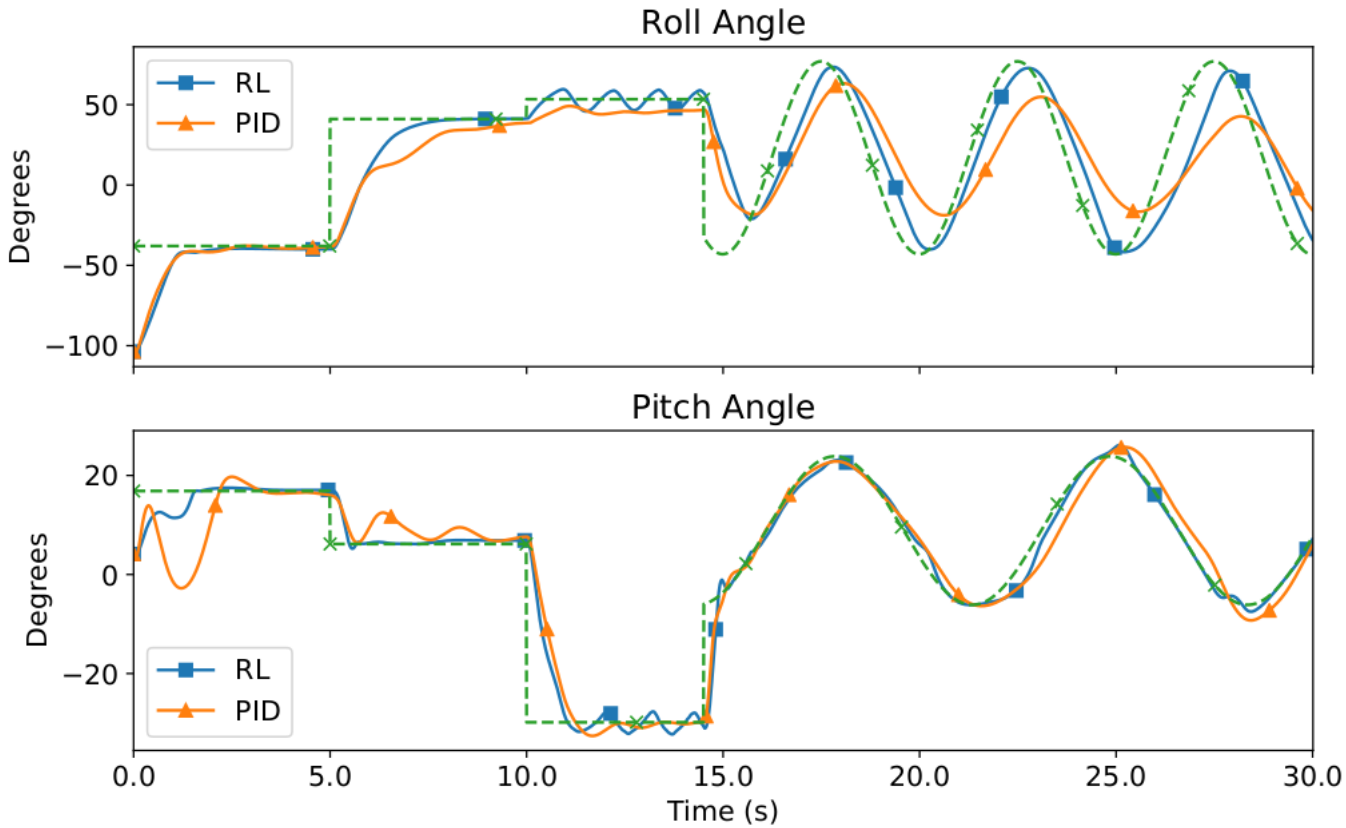


Fig. 3: Comparison of the PID and RL controllers tasked with tracking the dashed green line.

vi. Control of glider

Glider soaring

"Glider soaring via reinforcement learning in the field", Reddy et. al., Nature (2018).

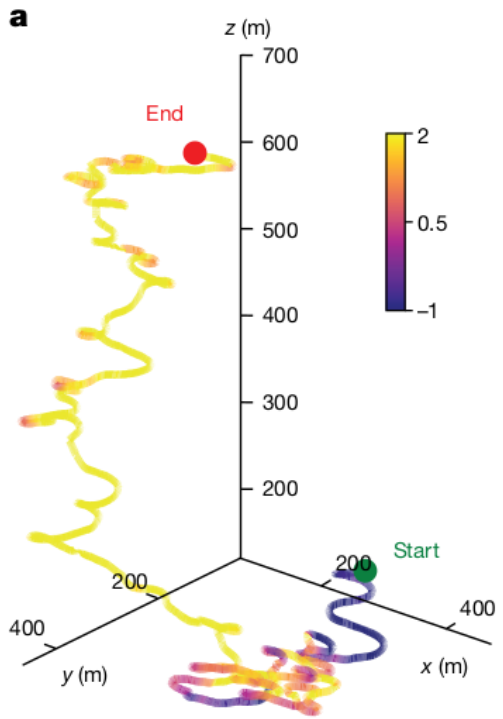
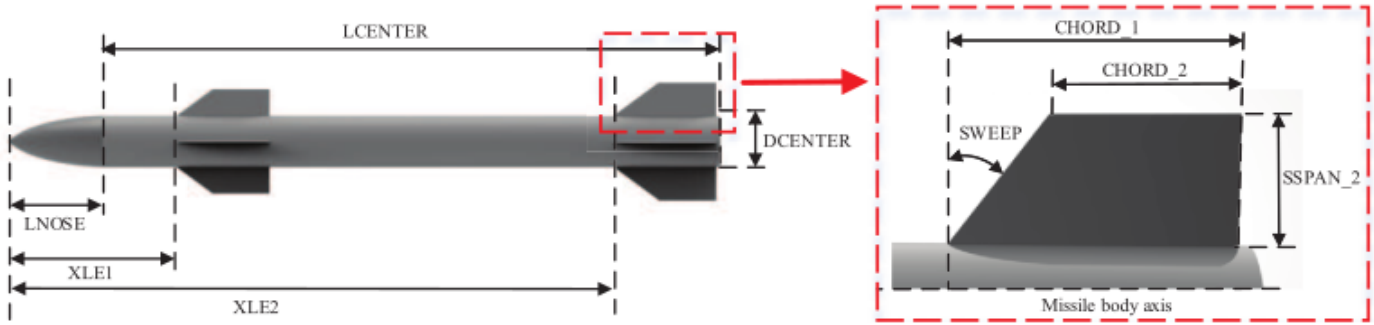


Fig. 3 | Performance of the learned strategy and its dependence on the wingspan. a, A 12-min-long trajectory of the glider executing the learned strategy for navigating thermals in the field, coloured according to the vertical ground velocity at each instant. **b**, Experimentally measured climb

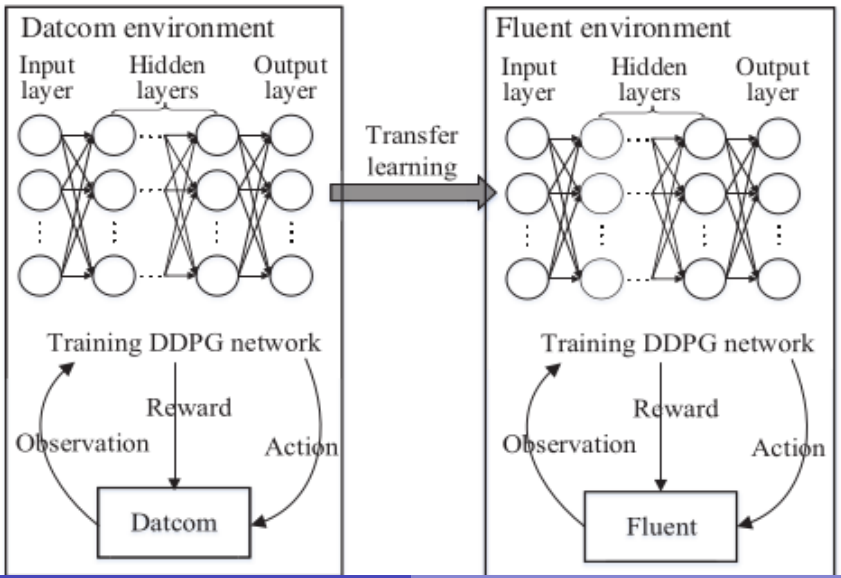
vii. Optimal design of missiles

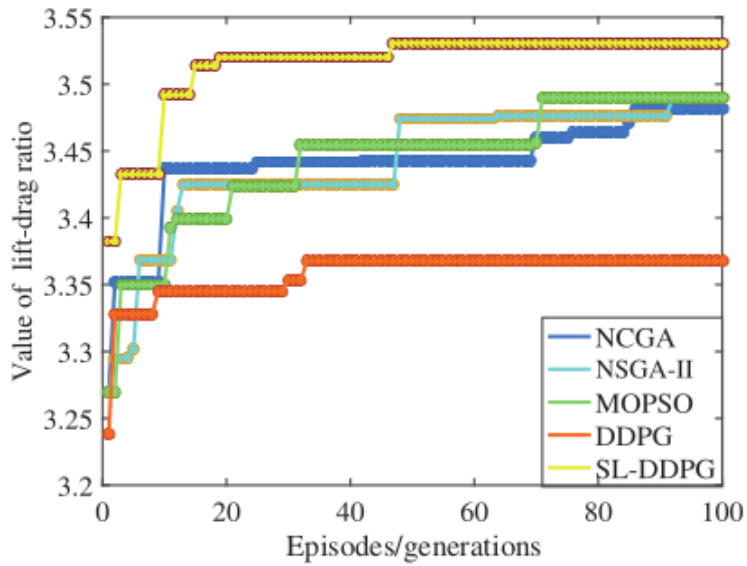
Missiles and RL

"Aerodynamic shape optimization using a novel optimizer based on machine learning techniques", X. Yan et. al.



(a)





viii. Growing interest

APS - DFD

2018: 1 abstract about DRL (mine).

2019: 10 abstracts (details: `http://meetings.aps.org/Meeting/DFD19`):

- Airfoil Shape Optimization using Deep Q - Network
- Airfoil control with Proximal Policy Optimization
- Zermelo's problem: Optimal point-to-point navigation in 2D turbulent flows using Reinforcement Learning
- Suppressing flow separation over a flat plate using machine learning
- Reinforcement Learning for a Bio-Inspired Vehicle with Undulating Fin Propulsion.
- Reinforcement learning enabled control of chaotic dynamics
- Elimination of velocity deficit behind a cylinder using reinforcement learning
- Deep Reinforcement Learning for Flow Control
- Reinforcement learning enabled control of chaotic dynamics
- Predator-Prey Interactions using Deep Reinforcement Learning

Conclusion

Summary of the results and general considerations

- Effective for control of non-linear, high dimensional systems.
- Many advantages, compared with for example adjoint method: train once use many times, closed loop, partial observation, transfer learning, extrapolation, 'trivially' parallel training algorithm.

More generally DRL is a nonlinear, non convex, general purpose optimizer: CFD engineer workflow can be seen as DRL.

(b) Best shape with 4 points, 3 free points (9 d.o.f.s)

Finally a tool that could fully use actuations and design possibilities?

Strengths and weaknesses of DRL

Weaknesses:

- Needs lots of data.
- No guarantee of convergence / stability.
- Not (easily) explainable.

Strengths:

- Seems to work quite well in practise.
- No need for full access to the system.
- Can (try to) extrapolate.
- Seems robust.

Sometimes unfair criticisms from reviewers / others:

- "this is only gradient descent": true but false.
- "not explainable = no value": false, just one (experimental) tool.

Future and perspectives

Control may be for the time being the most promising avenue.
Recent results:

- Proof-of-concept.
- Parallelization / speedup.
- Application to large systems / invariants.

Minor remaining questions; well adapted for MSc thesis, extend released code:

- Robustness? (some preliminary work, mostly remains to be done).
- Interpolation / extrapolation of policies? (already done, to be released).
- Reality gap / experiment?

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

- 3D systems.
- Higher Re, possibly through experiences.
- Example I: sphere in 3D at moderate Re (already well advanced).
- Example II: BL and BL friction.

Additional resources

- Book chapter about DRL in fluid mechanics: https://www.researchgate.net/publication/343934046_DEEP_REINFORCE
- DRL control at Re 100: https://github.com/jerabaul29/Cylinder2DFlowControlDRL
- DRL control in parallel: https://github.com/jerabaul29/Cylinder2DFlowControlDRLParallel
- Robust DRL control over range of Res: https://github.com/thw1021/Cylinder2DFlowControlGeneral
- DRL control with invariants: https://github.com/vbelus/drl-fluid-film-notebook