TEMPUS

energy_py

lessons learnt building an energy reinforcement learning library

TEMPUS

4 years energy engineer @ **ENGIE**

graduate & 2 years teaching reinforcement learning @ **Data Science Retreat**

data scientist @ **Tempus Energy**

blog at www.adgefficiency.com

TEMPUS

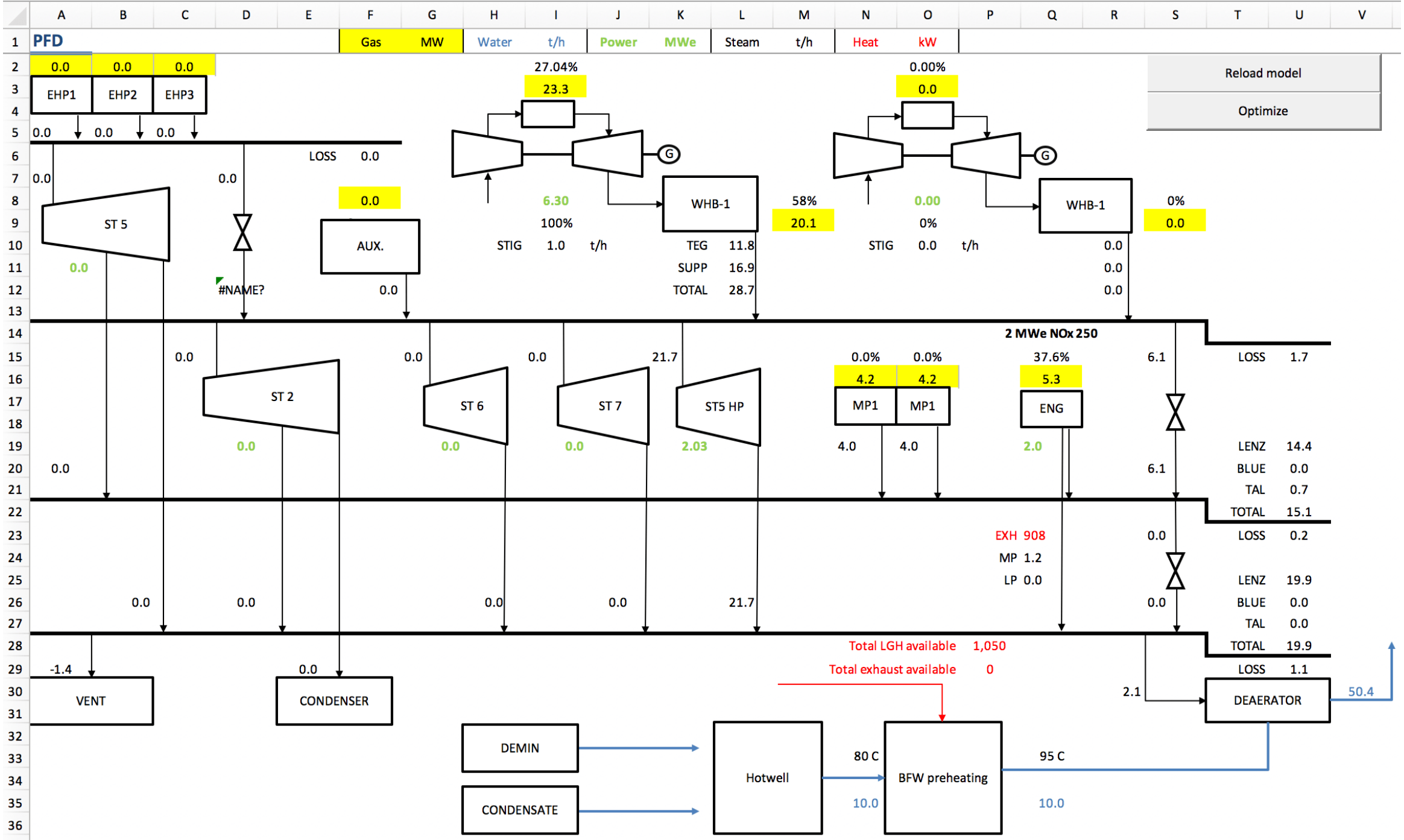
agenda

motivation

API + performance

lessons

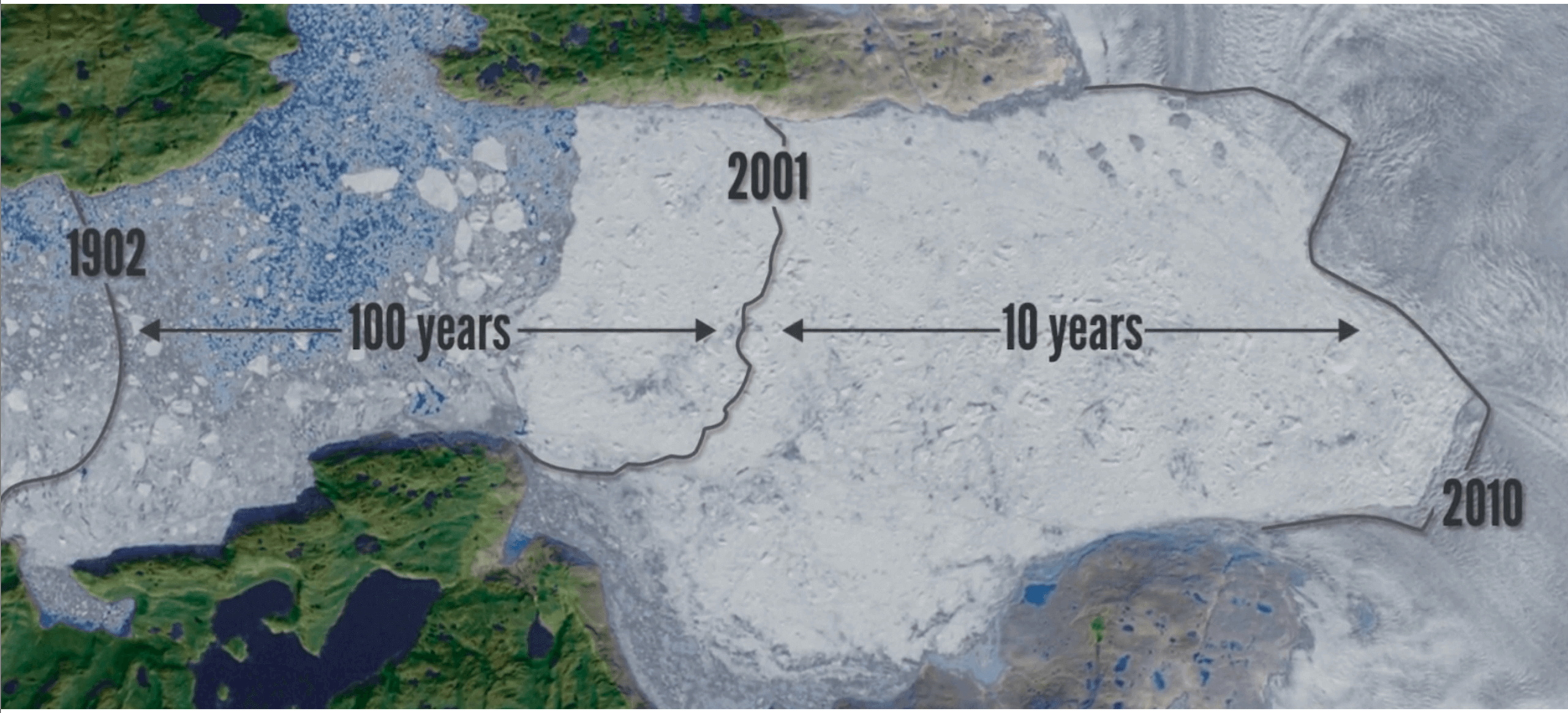
next steps


PFD
Gas MW
Water t/h
Power MWe
Steam t/h
Heat kW
Reload model
Optimize
EHP1
EHP2
EHP3
ST 5
AUX.
#NAME?
LOSS 0.0
27.04%
23.3
6.30
100%
STIG 1.0 t/h
WHB-1
TEG 11.8
SUPP 16.9
TOTAL 28.7
58%
20.1
0.00%
0.0
0.00
0%
STIG 0.0 t/h
WHB-1
0%
0.0
2 MWe NOx 250
ST 2
ST 6
ST 7
ST5 HP
MP1
MP1
ENG
LOSS 1.7
LENZ 14.4
BLUE 0.0
TAL 0.7
TOTAL 15.1
EXH 908
MP 1.2
LP 0.0
LOSS 0.2
LENZ 19.9
BLUE 0.0
TAL 0.0
TOTAL 19.9
LOSS 1.1
VENT
CONDENSER
DEAERATOR
50.4
Total LGH available 1,050
Total exhaust available 0
DEMIN
CONDENSATE
Hotwell
80 C
10.0
BFW preheating
95 C
10.0
LGH dump 0

TEMPUS

computation to help solve the **control** problem

computation to help solve the **climate** problem

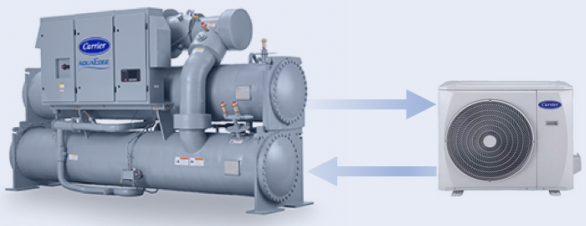

2001
1902
100 years
10 years
2010

TEMPUS

price response flexible demand

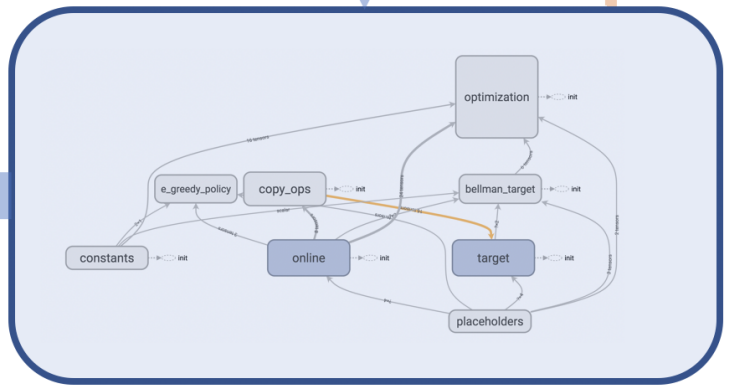
aka the lazy taxi driver

environment



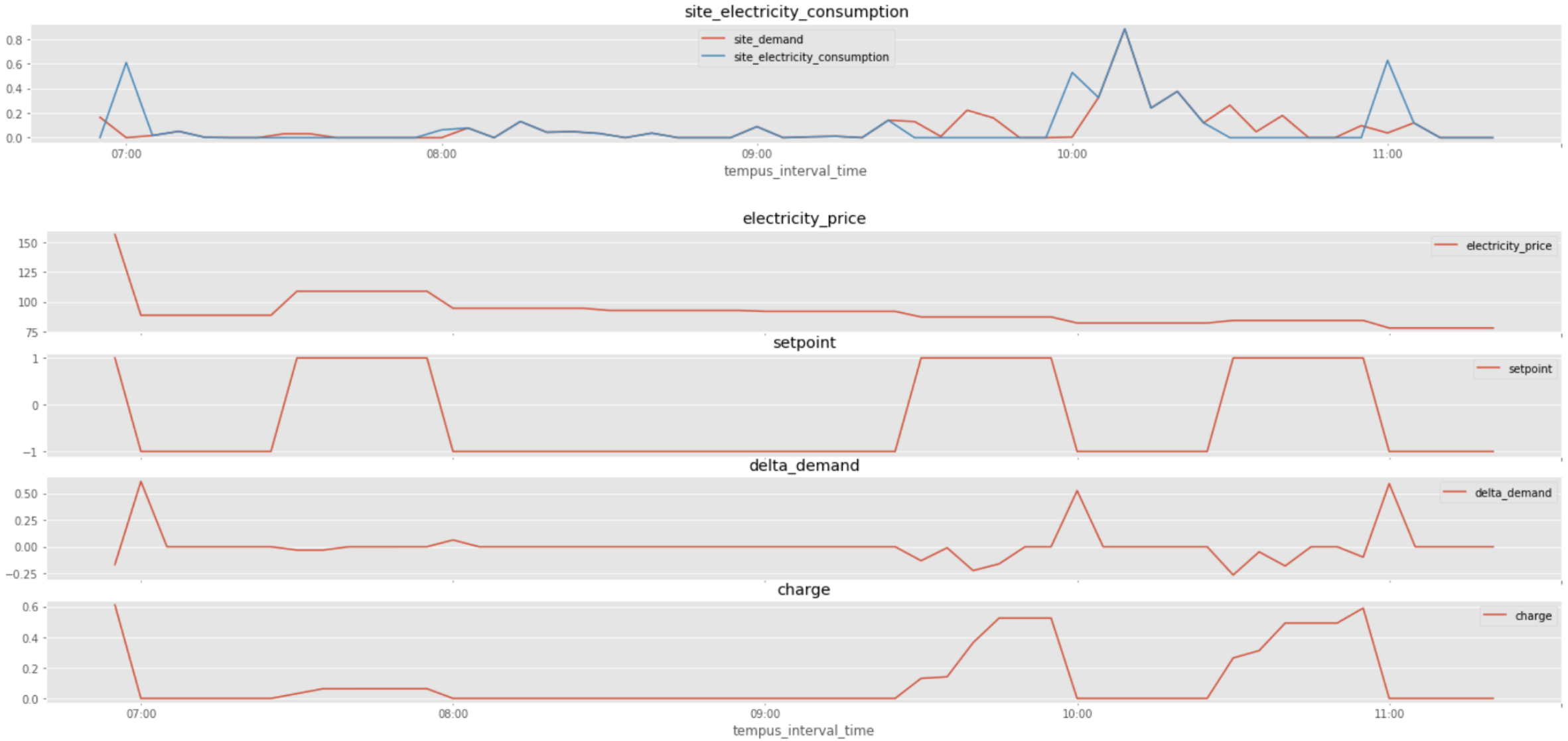
state = cooling demands, electricity prices
reward = electricity consumption * electricity price

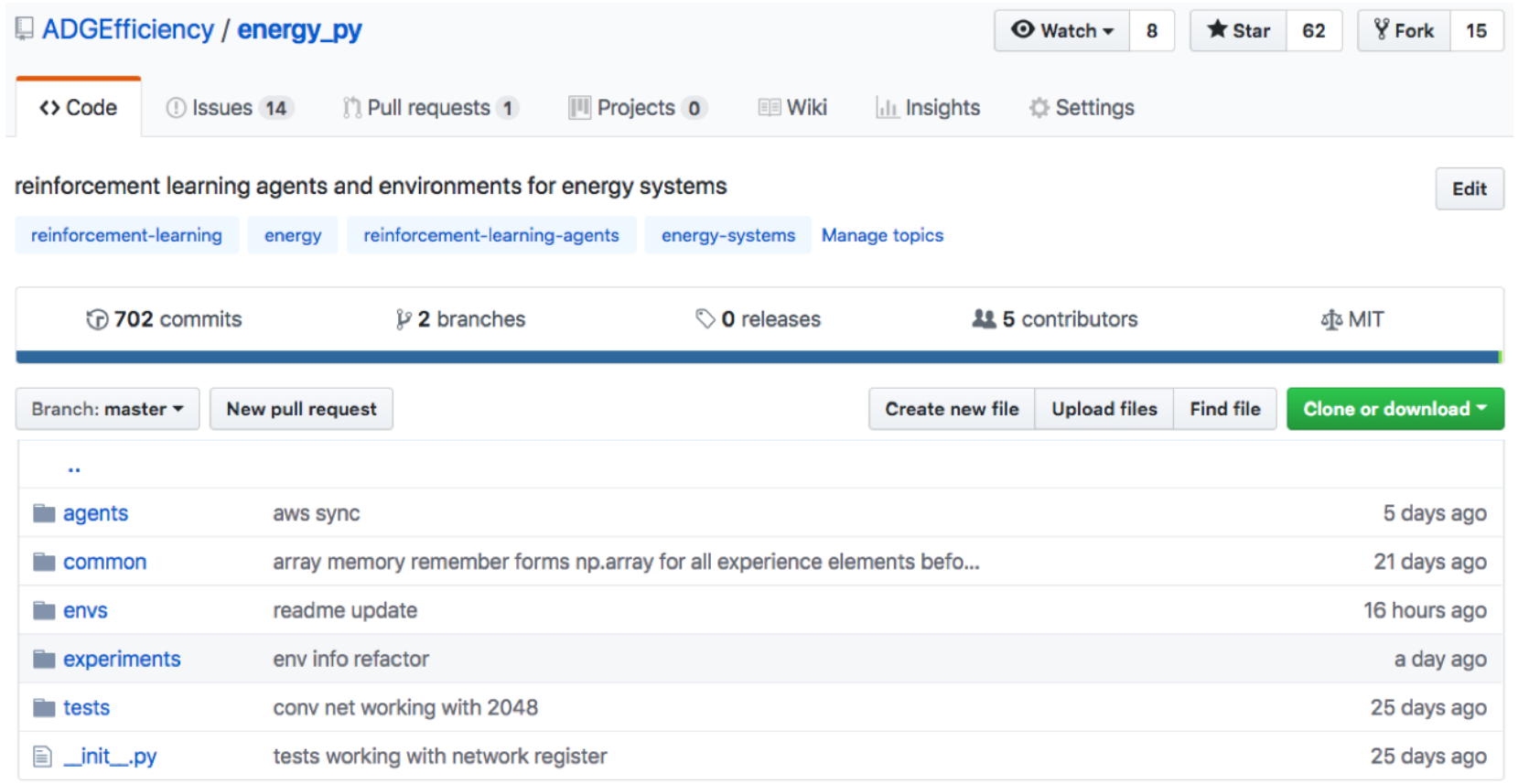
action = temperature setpoint



agent

hyperparameter torturer


site_electricity_consumption
site_demand
site_electricity_consumption
0.8
0.6
0.4
0.2
0.0
07:00
08:00
09:00
10:00
11:00
tempus_interval_time
electricity_price
electricity_price
150
125
100
75
setpoint
setpoint
1
0
−1
delta_demand
delta_demand
0.50
0.25
0.00
−0.25
charge
charge
0.6
0.4
0.2
0.0
07:00
08:00
09:00
10:00
11:00
tempus_interval_time


ADGEfficiency / energy_py
Watch 8
Star 62
Fork 15
Code
Issues 14
Pull requests 1
Projects 0
Wiki
Insights
Settings
reinforcement learning agents and environments for energy systems
Edit
reinforcement-learning
energy
reinforcement-learning-agents
energy-systems
Manage topics
702 commits
2 branches
0 releases
5 contributors
MIT
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
..
agents aws sync 5 days ago
common array memory remember forms np.array for all experience elements befo... 21 days ago
envs readme update 16 hours ago
experiments env info refactor a day ago
tests conv net working with 2048 25 days ago
__init__.py tests working with network register 25 days ago

TEMPUS

DQN + naive agents

energy environments + wrappers around `gym`

tools for experimentation

High level API

```
$ cd energy_py/experiments

$ python experiment.py example dqn

$ tensorboard --logdir='./results/example/tensorboard'
```

Low level API

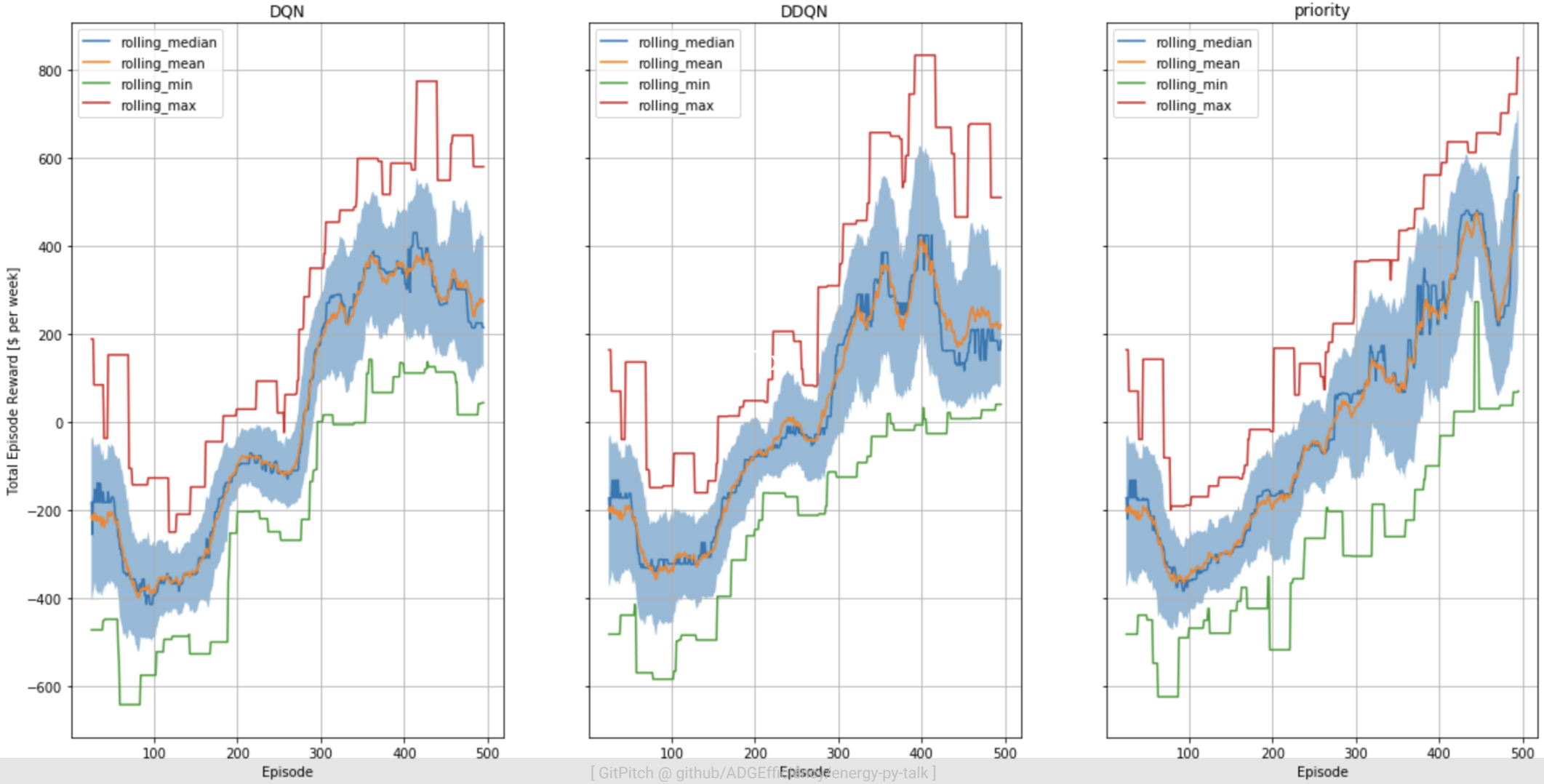
```
import energy_py

with tf.Session() as sess:
    env = energy_py.make_env(
        env_id='battery',
        episode_length=288,
        dataset='example'
    )

    agent = energy_py.make_agent(
        sess=sess,
        agent_id='dqn',
        env=env,
        total_steps=1000000,
        batch_size=1024,
    )
```

TEMPUS

fun with cartpole

battery storage

Average over 5 random seeds. Window length of 25 episodes. Shaded area shows +/- the rolling standard deviation.



TEMPUS

details - space design

```
#  create an action space of two dimensions
action_space = GlobalSpace('action').from_spaces(
    [ContinuousSpace(0, 100), DiscreteSpace(3)],
    ['acceleration', 'gear']
)

#  randomly sample an action
action = action_space.sample()

#  check the action is valid
assert action_space.contains(action)

#  discretize the space
discrete_spaces = action_space.discretize(20)

#  randomly sample a discrete action
action = action_space.sample_discrete()
```

```
#  load a state or observation space from a dataset
state_space = GlobalSpace('state').from_dataset('example')

# we can sample an episode from the state
episode = state_space.sample_episode(start=0, end=100)

# sample from the current episode by calling the space
state = state_space(steps=0)
```

TEMPUS

rl today

hard

unstable

sample inefficient

exciting

TEMPUS

the environment model problem / opportunity


state
action
environment model
next_state
reward

In a similar vein, you can easily outperform DQN in Atari with off-the-shelf Monte Carlo Tree Search. Here are baseline numbers from Guo et al, NIPS 2014. They compare the scores of a trained DQN to the scores of a UCT agent (where UCT is the standard version of MCTS used today.)

Agent	*B.Rider*	*Breakout*	*Enduro*	*Pong*	*Q*bert*	*Seaquest*	*S.Invaders*
DQN	4092	168	470	20	1952	1705	581
-best	5184	225	661	21	4500	1740	1075

Agent	*B.Rider*	*Breakout*	*Enduro*	*Pong*	*Q*bert*	*Seaquest*	*S.Invaders*
UCT	7233	406	788	21	18850	3257	2354

```
@misc{rlblogpost,
    title={Deep Reinforcement Learning Doesn't Work Yet},
    author={Irpan, Alex},
    howpublished={\url{https://www.alexirpan.com/2018/02/14/rl-hard.html}},
    year={2018}
}
```

backwards induction

```
previous_state_payoffs = {state: 0 for state in model.states}

for step in steps[::-1]:
    viable_transitions = [get_viable_transitions(step, next_state, model)
                          for next_state in model.states]

    new_state_payoffs = defaultdict(list)

    for t in viable_transitions:
        payoff = t.reward + previous_state_payoffs[t.next_state]
        new_state_payoffs[t.state].append(payoff)

    for k, v in new_state_payoffs.items():
        new_state_payoffs[k] = np.max(v)

    previous_state_payoffs = new_state_payoffs
```

TEMPUS

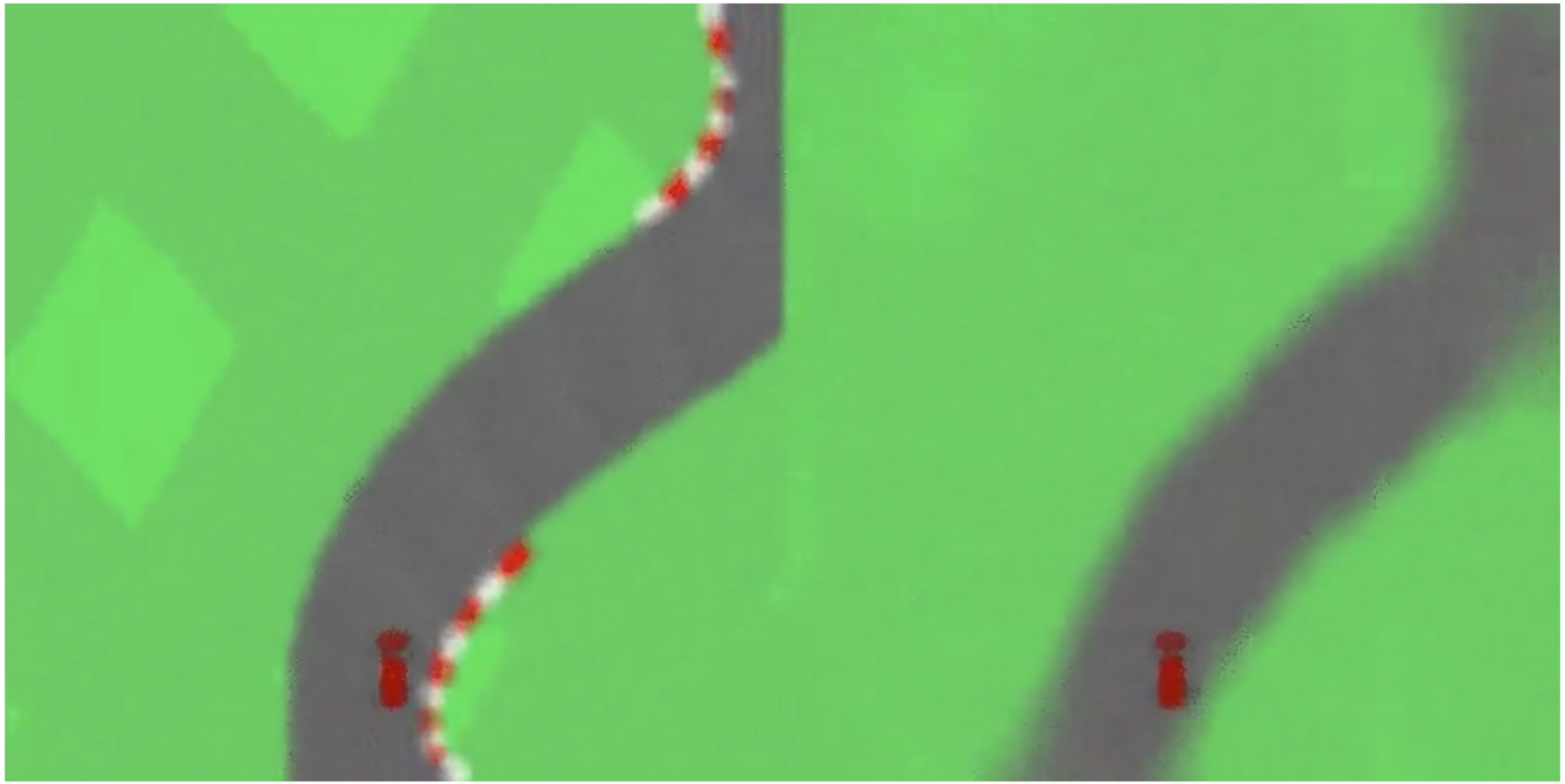
synthetic data - aka poor mans GANS

orld Models

vid Ha, Jürgen Schmidhuber

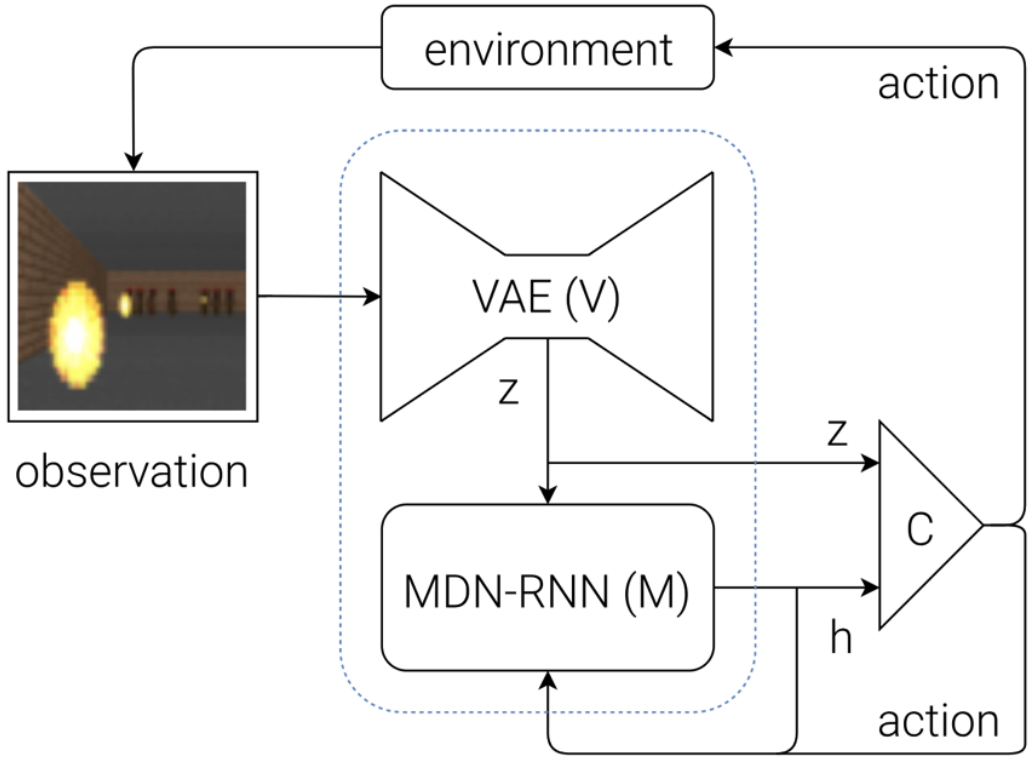
bmitted on 27 Mar 2018 (v1), last revised 9 May 2018 (this version, v4))

We explore building generative neural network models of popular reinforcement learning environments. Our wor model can be trained quickly in an unsupervised manner to learn a compressed spatial and temporal representation of the environment. By using features extracted from the world model as inputs to an agent, we can train a very compact and simple policy that can solve the required task. We can even train our agent entirely inside of its own hallucinated dream generated by its world model, and transfer this policy back into the actual environment.

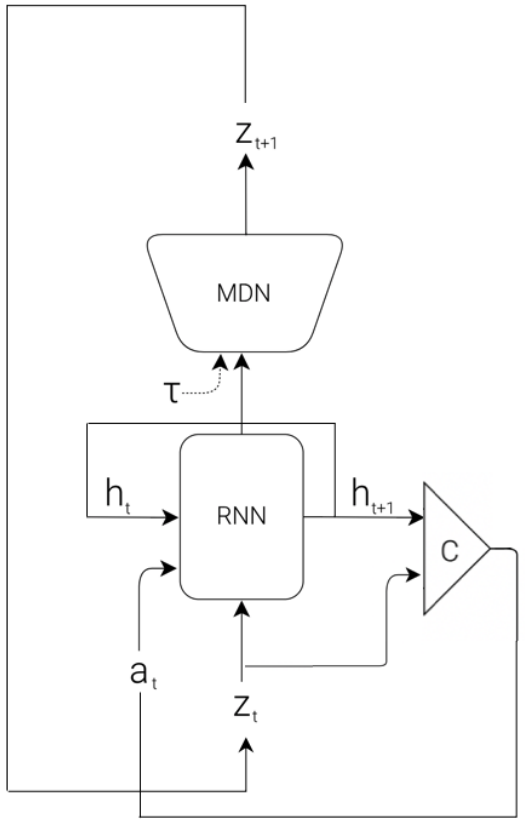


Actual observations from the environment.

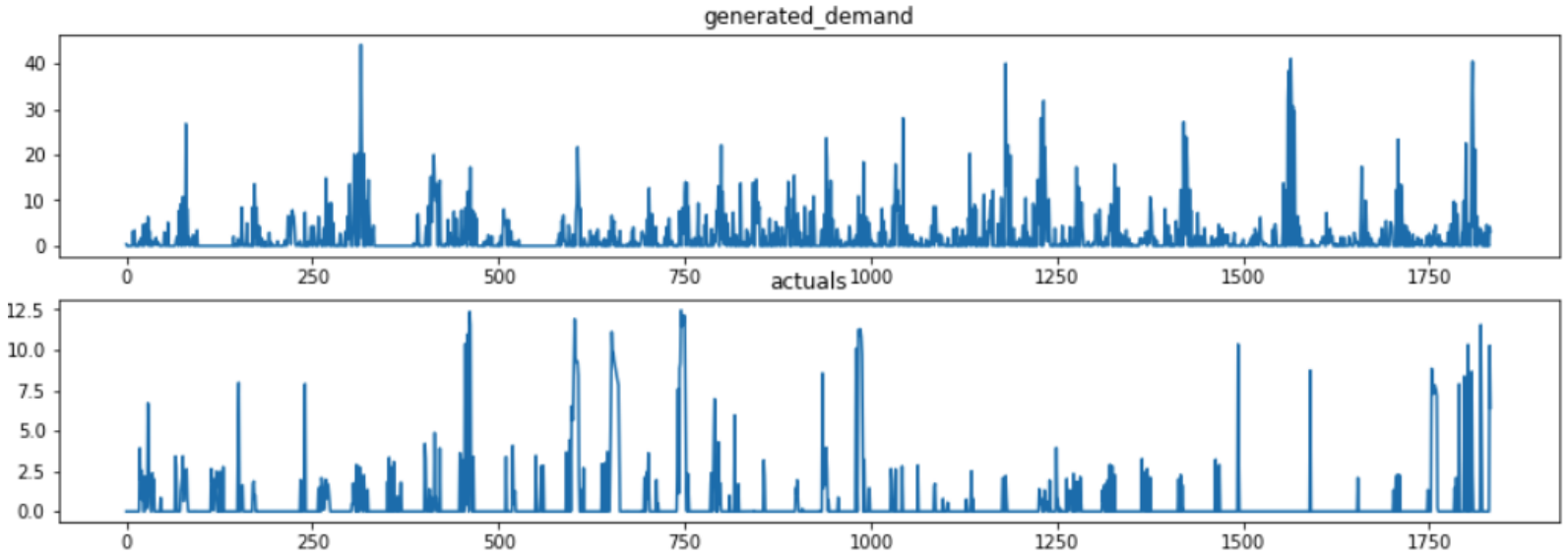
What gets encoded into z_t.



Learning using the real environment



Learning inside a dream


generated_demand
40
30
20
10
0
0
250
500
750
1000
1250
1500
1750
actuals
12.5
10.0
7.5
5.0
2.5
0.0
0
250
500
750
1000
1250
1500
1750

TEMPUS

long term work

a fourth complete rebuild

wrapping other environments

model based methods

integrating with `google/dopamine`

Dopamine

Dopamine is a research framework for fast prototyping of reinforcement learning algorithms. It aims to fill the need for a small, easily grokked codebase in which users can freely experiment with wild ideas (speculative research).

Our design principles are:

- *Easy experimentation*: Make it easy for new users to run benchmark experiments.
- *Flexible development*: Make it easy for new users to try out research ideas.
- *Compact and reliable*: Provide implementations for a few, battle-tested algorithms.
- *Reproducible*: Facilitate reproducibility in results.

In the spirit of these principles, this first version focuses on supporting the state-of-the-art, single-GPU *Rainbow* agent (Hessel et al., 2018) applied to Atari 2600 game-playing (Bellemare et al., 2013). Specifically, our Rainbow agent implements the three components identified as most important by Hessel et al.:

- n-step Bellman updates (see e.g. Mnih et al., 2016)
- Prioritized experience replay (Schaul et al., 2015)
- Distributional reinforcement learning (C51; Bellemare et al., 2017)

TEMPUS

thank you