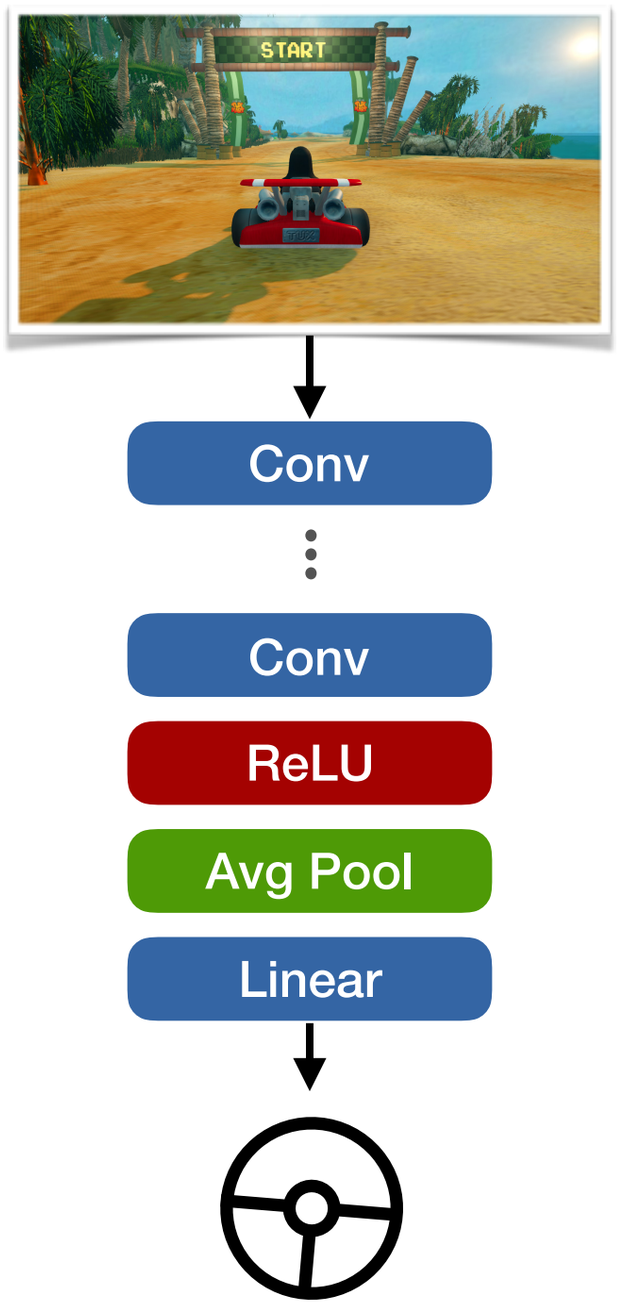
Acting in an environment

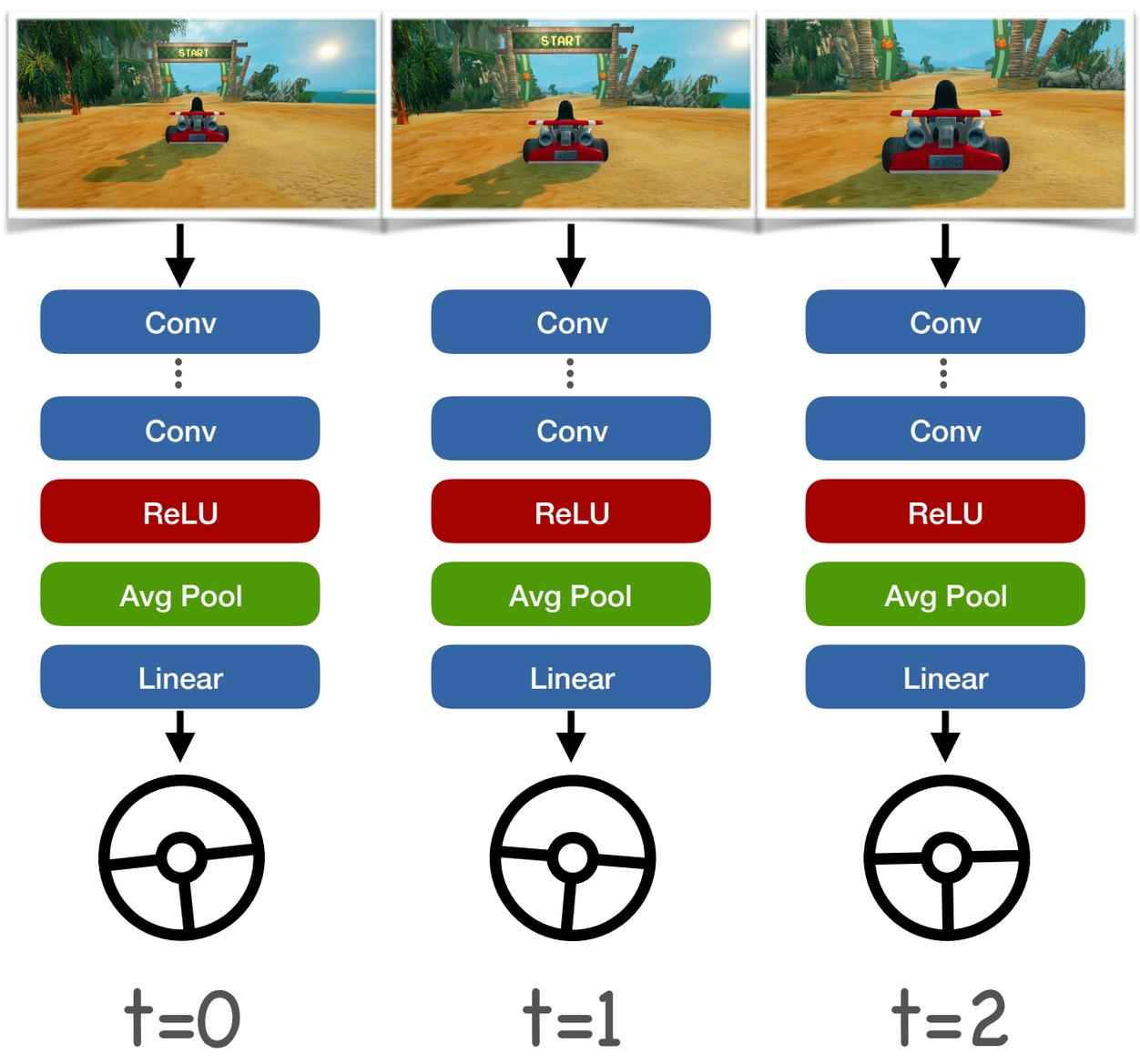
Deep learning for action

- Input
  - Observation
- Output
  - Action



Acting in an environment

- Action changes that state the of the world
- Non-differentiable
- Often non-repeatable
- Long-range dependencies



Acting in an environment

state /
observation

agent

action

left

right

up

jump

down

shoot

environment

Acting in an environment

Video Game

environment

state / observation

Deep Network

agent

action

left

right

up

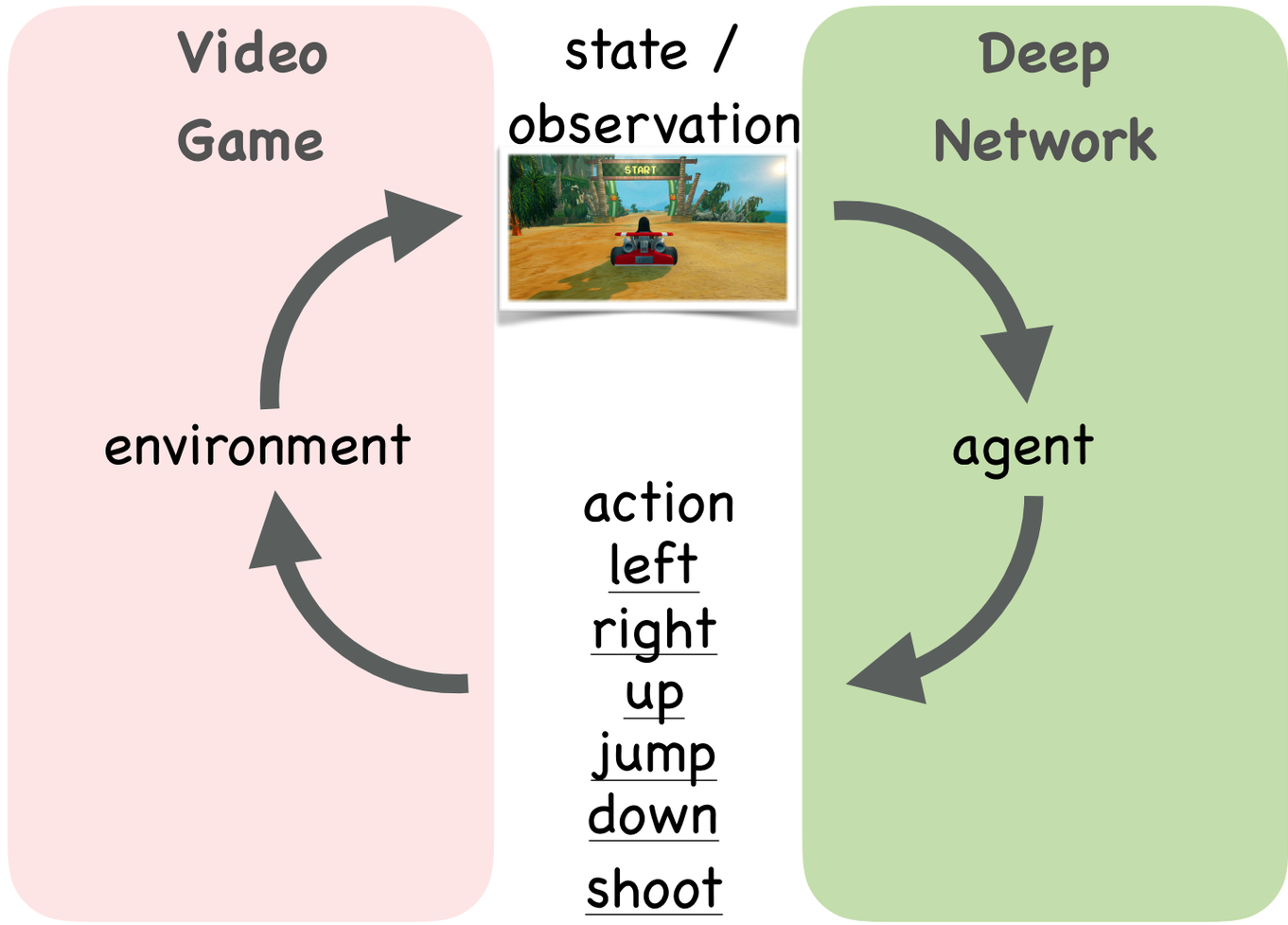
jump

down

shoot

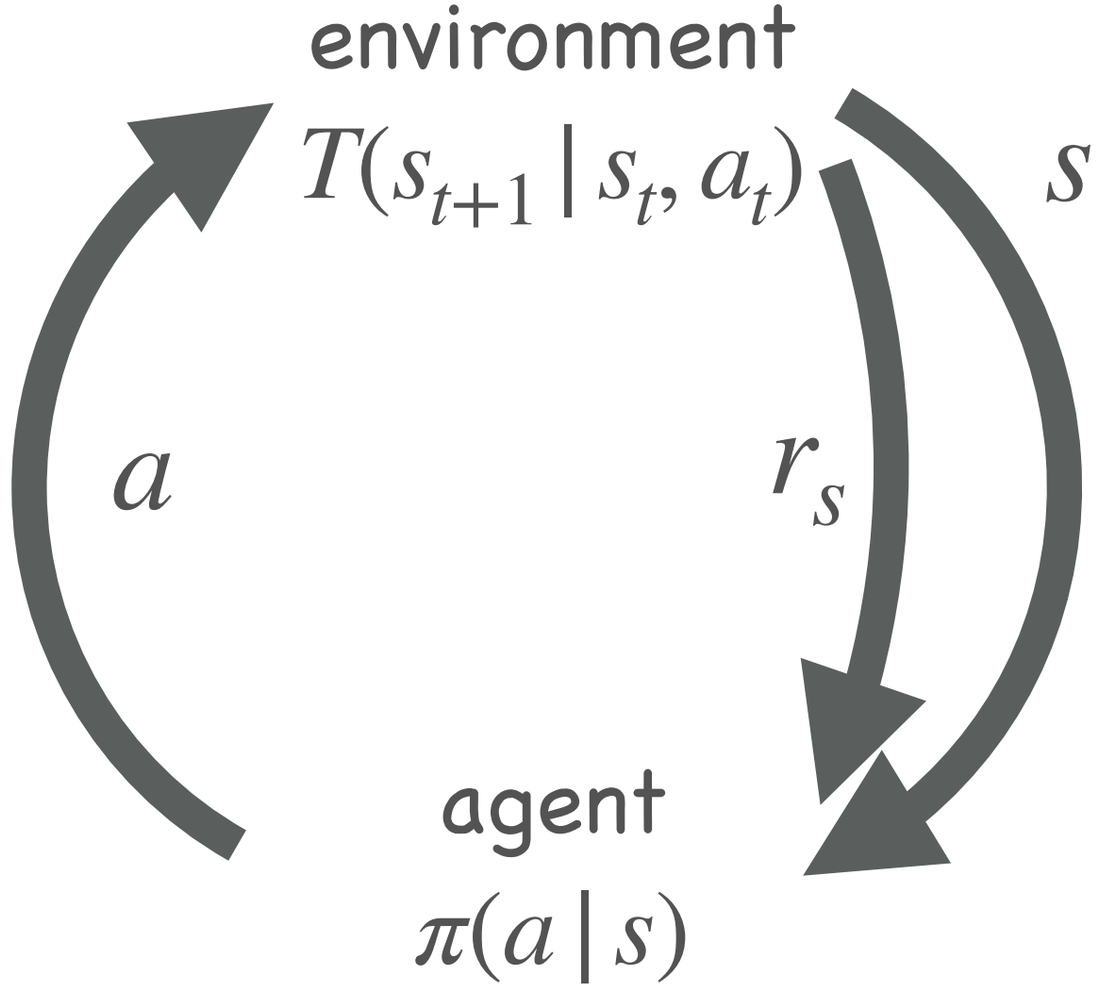
How to train the agent?

- What should the agent learn to do?
  - Minimize loss
    - Reward from environment



Markov decision process (MDP) - Formal definition

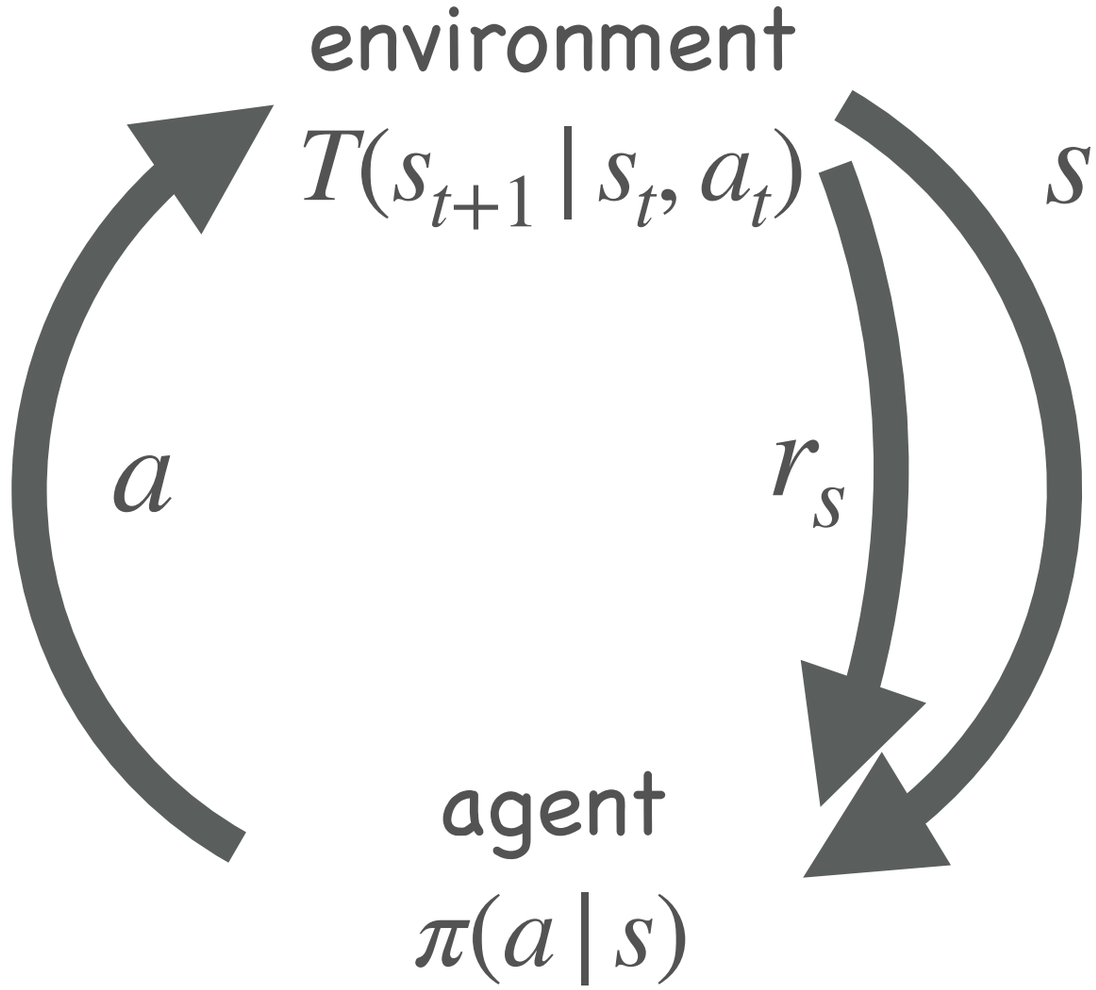
- state $s \in \mathrm{S}$
- action $a \in \mathrm{A}$
- reward $r_s \in \mathbb{R}$
- transition $T(s_{t+1} \mid s_t, a_t)$
- policy $\pi(a \mid s)$



MDP - objective

- Trajectory
  $\tau = \{s_0, a_0, s_1, a_1, \dots\}$

- Return
  $R(\tau) = \sum_t \gamma^t r_{s_t}$

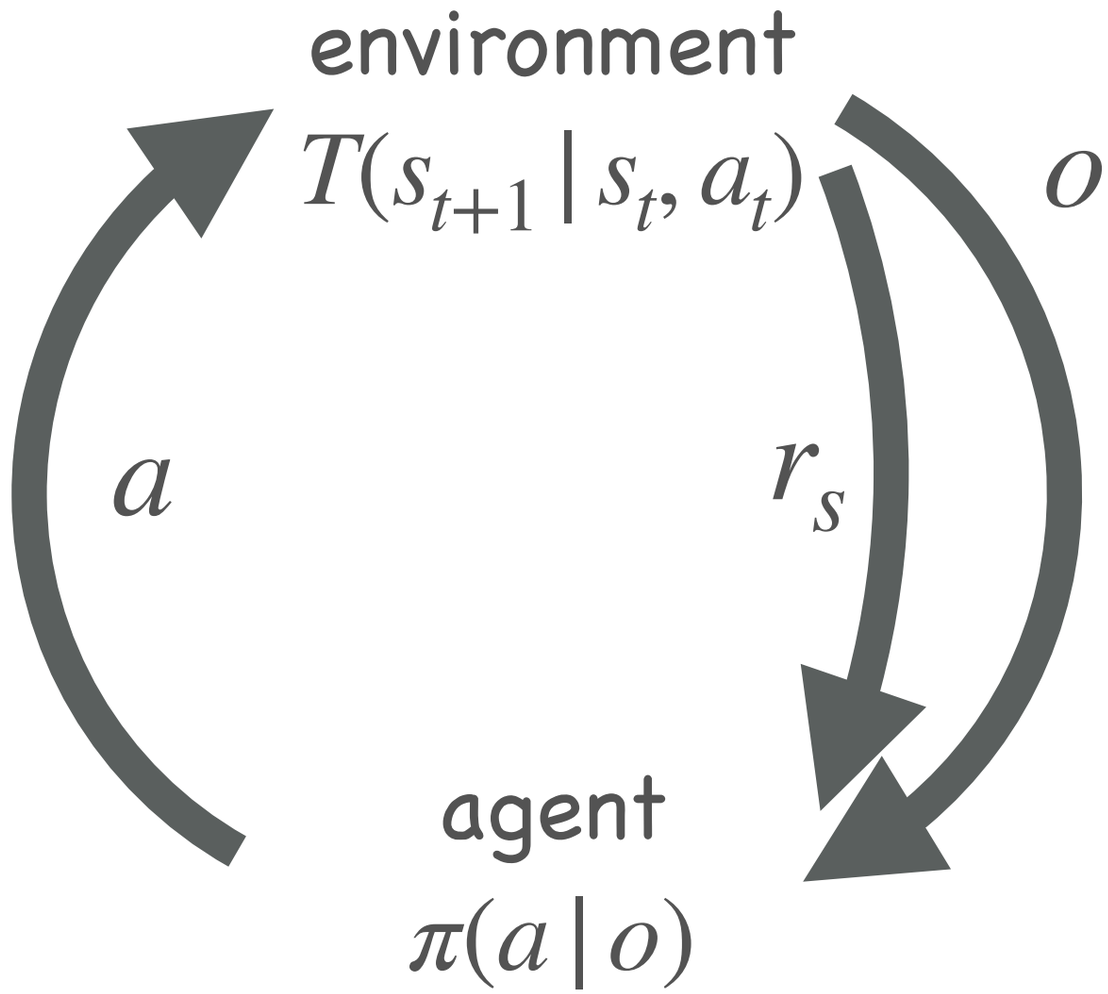


- Objective

  - $\text{maximize}_\pi \; \mathbb{E}_{\tau \sim P_{\pi,T}}[R(\tau)]$

Partially observed Markov decision process (POMDP)

- state $s \in \mathrm{S}$
- action $a \in \mathrm{A}$
- reward $r_s \in \mathbb{R}$
- transition $T(s_{t+1} \mid s_t, a_t)$
- **observation** $o \in \mathrm{O}$
- **observation function** $O(o \mid s)$
- **policy** $\pi(a \mid o)$



Examples - Cart-pole

- MDP
- **objective**: balance a pole on movable cart
- **state**: angle, angular velocity, position, velocity
- **action**: force applied
- **reward**: 1 for each time step pole is upright

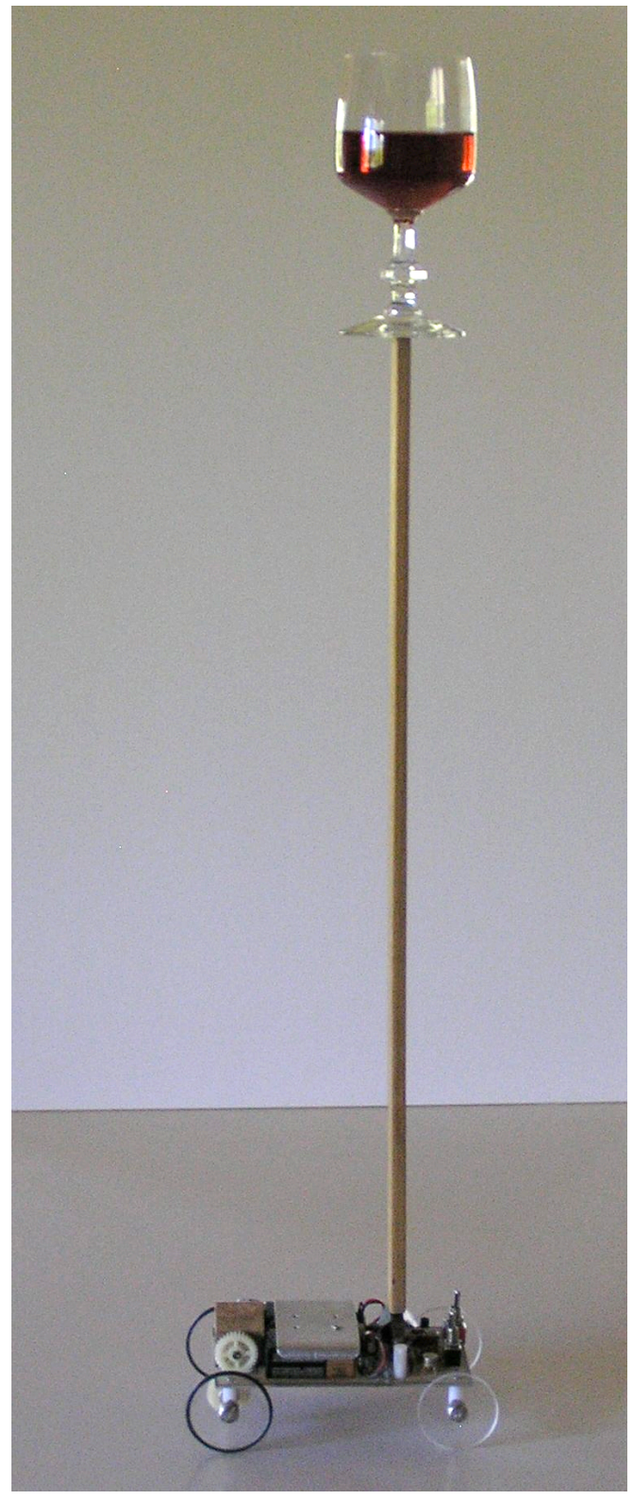
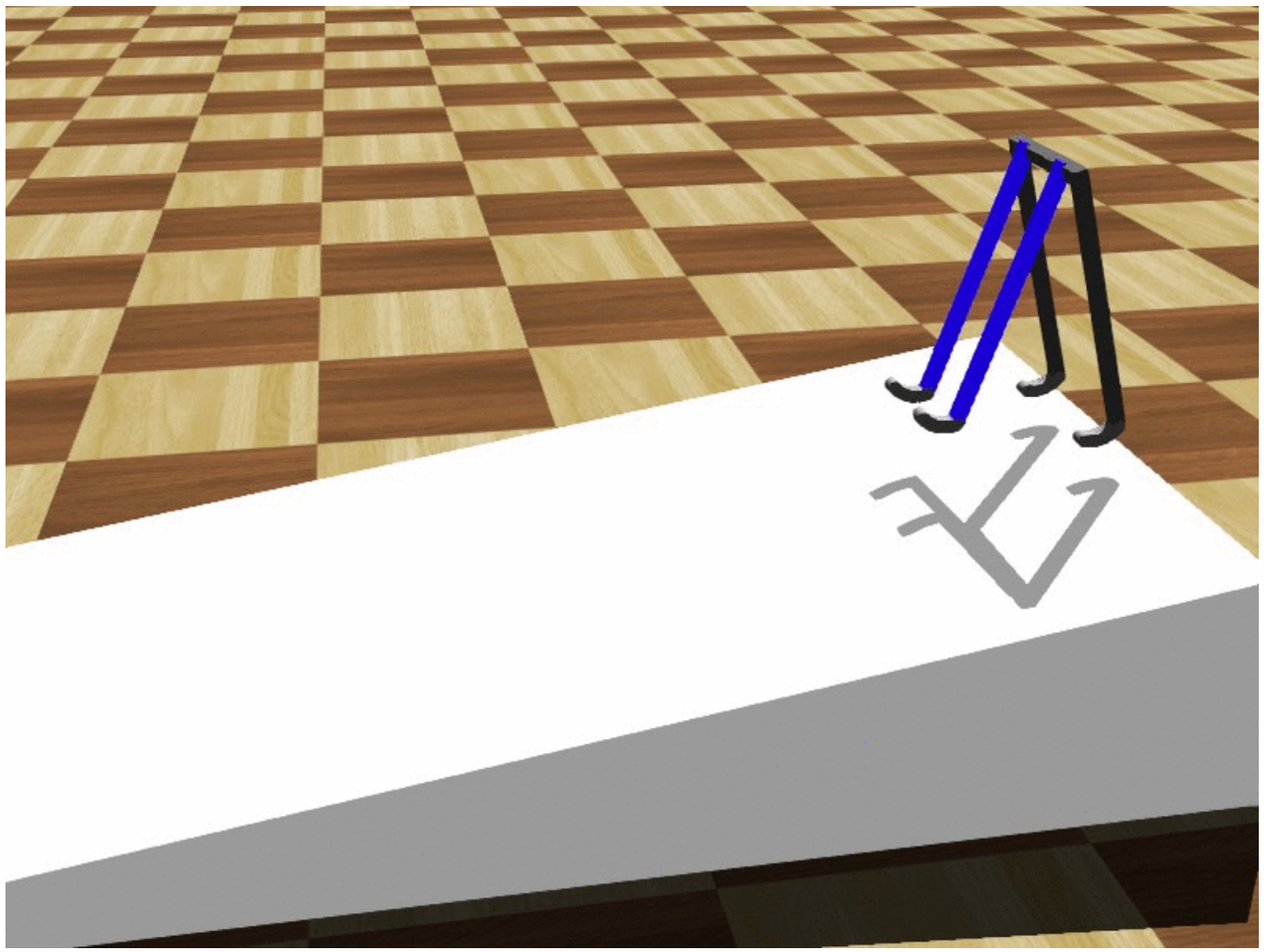


Image source: https://commons.wikimedia.org/wiki/File:Balancer_with_wine_3.JPG

Examples - Robot locomotation

- MDP
- **objective**: make the robot move
- **state**: joint angle and position
- **action**: torques applied to joints
- **reward**: 1 for each time upright + moving



Video source: https://commons.wikimedia.org/wiki/File:Passive_dynamic_walker.gif

Examples - Games

- POMDP
- **objective**: beat the game
- **state**: position, location, state of all objects, agents and world
- **action**: game controls
- **reward**: score increase/ decrease, complete level, die

Video source: SuperTuxKart 1.0 Official Trailer, https://www.youtube.com/watch?v=Lm1TFDBillg

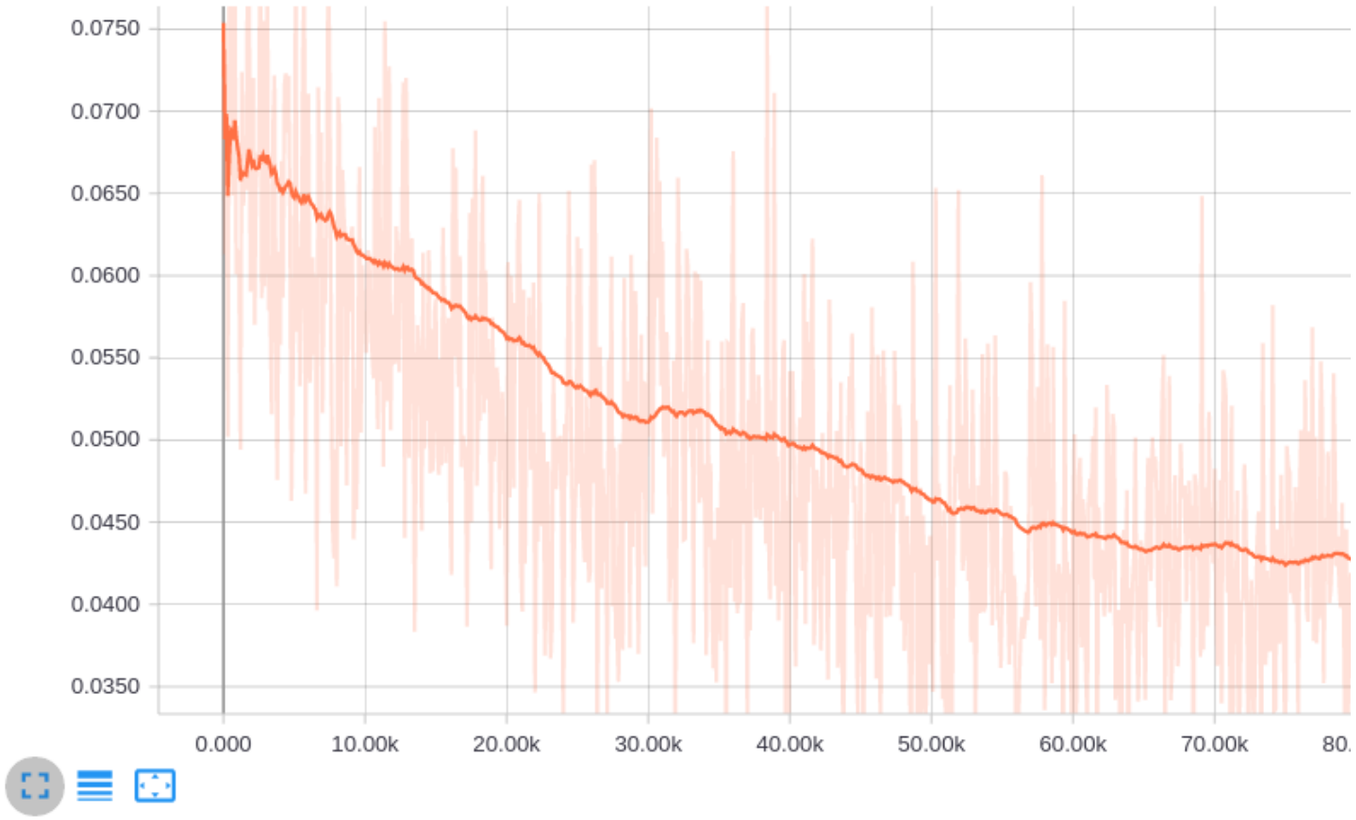
Examples - GO

- MDP
- **objective**: win the game
- **state**: position of pieces
- **action**: next piece
- **reward**: 0 lose, 1 win

Image source: https://en.wikipedia.org/wiki/Go_(game)#/media/File:FloorGoban.JPG

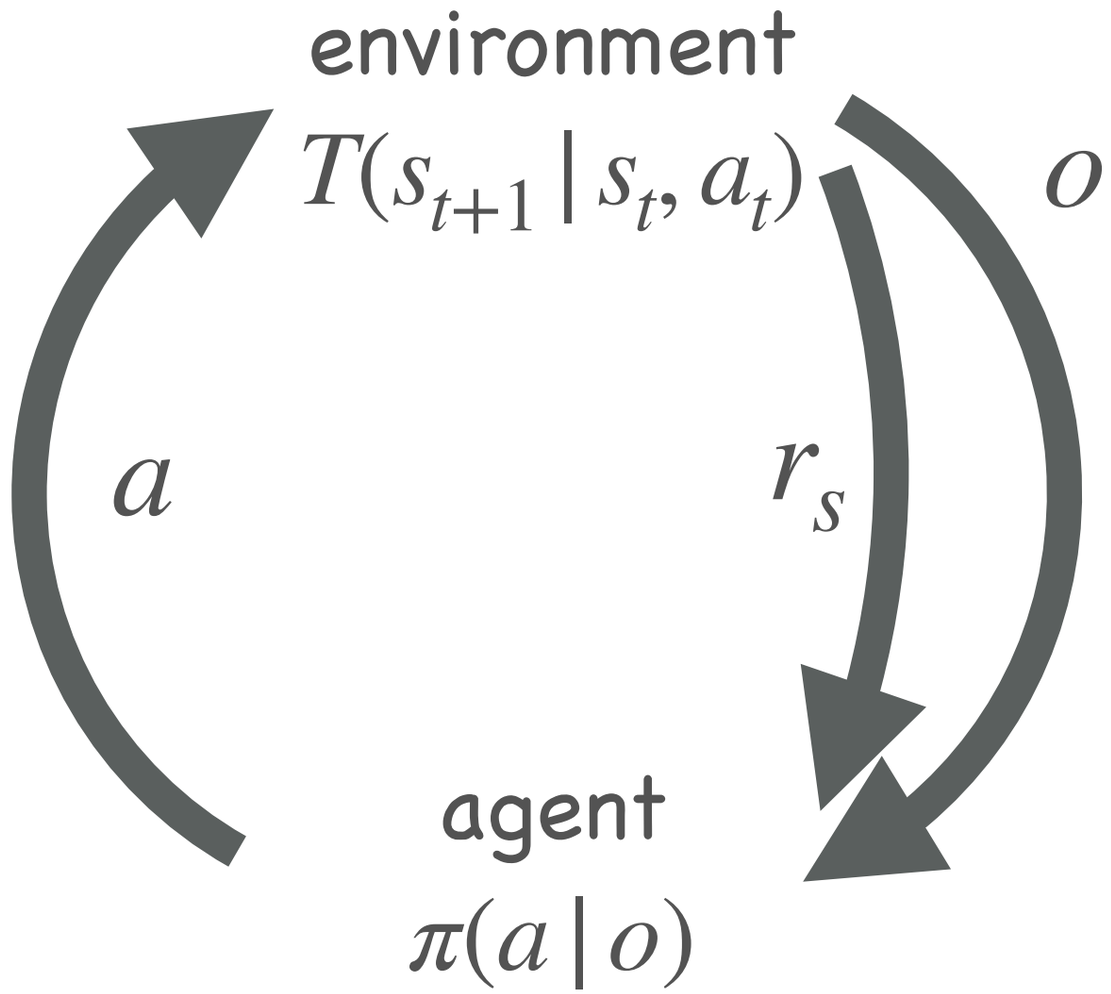
Examples - supervised learning

- MDP
- **objective**: Minimize the training (or validation) loss
- **state**: weights and hyper-parameters
- **action**: gradient update
- **reward**: change in loss



Everything is a (PO)MDP

- Very general concept
  - NP-hard
- Specialized algorithms still work well



01b

January 23, 2024

[1]:
```
%pylab inline
import torch
import sys, os
import pystk
device = torch.device('cuda') if torch.cuda.is_available() else torch.
 ↪device('cpu')
print('device = ', device)
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
device =  cuda
```

[2]:
```
class Rollout:
    def __init__(self, screen_width, screen_height, hd=True,␣
 ↪track='lighthouse', render=True):
        # Init supertuxkart
        if not render:
            config = pystk.GraphicsConfig.none()
        elif hd:
            config = pystk.GraphicsConfig.hd()
        else:
            config = pystk.GraphicsConfig.ld()
        config.screen_width = screen_width
        config.screen_height = screen_height
        pystk.init(config)

        self.render = render
        race_config = pystk.RaceConfig(track=track)
        self.race = pystk.Race(race_config)
        self.race.start()

    def __call__(self, agent, n_steps=200):
        self.race.restart()
        self.race.step()
        data = []
        track_info = pystk.Track()
        track_info.update()
```

```
        for i in range(n_steps):
            world_info = pystk.WorldState()
            world_info.update()

            # Gather world information
            kart_info = world_info.players[0].kart

            agent_data = {'track_info': track_info, 'kart_info': kart_info}
            if self.render:
                agent_data['image'] = np.array(self.race.render_data[0].image)

            # Act
            action = agent(**agent_data)
            agent_data['action'] = action

            # Take a step in the simulation
            self.race.step(action)

            # Save all the relevant data
            data.append(agent_data)
        return data
```

[3]:
```
def dummy_agent(**kwargs):
    action = pystk.Action()
    action.acceleration = 1
    return action
```

[4]:
```
rollout = Rollout(800, 600)

def rollout_many(many_agents, **kwargs):
    for agent in many_agents:
        yield rollout(agent, **kwargs)

data = rollout(dummy_agent)
```

[5]:
```
def show_video(frames, fps=30):
    import imageio
    from IPython.display import Video, display

    imageio.mimwrite('/tmp/test.mp4', frames, fps=fps, bitrate=10000000)
    display(Video('/tmp/test.mp4', width=800, height=600, embed=True))

show_video([d['image'] for d in data])
```

```
IMAGEIO FFMPEG_WRITER WARNING: input image is not divisible by
macro_block_size=16, resizing from (800, 600) to (800, 608) to ensure video
```

```
compatibility with most codecs and players. To prevent resizing, make your input
image divisible by the macro_block_size or set the macro_block_size to 1
(risking incompatibility).
```

```
<IPython.core.display.Video object>
```

[6]:
```
def show_agent(agent, **kwargs):
    data = rollout(agent, **kwargs)
    show_video([d['image'] for d in data])
```

[7]:
```
def less_dummy_agent(**kwargs):
    action = pystk.Action()
    action.acceleration = 1
    action.steer = -0.6
    return action
show_agent(less_dummy_agent)
```

```
IMAGEIO FFMPEG_WRITER WARNING: input image is not divisible by
macro_block_size=16, resizing from (800, 600) to (800, 608) to ensure video
compatibility with most codecs and players. To prevent resizing, make your input
image divisible by the macro_block_size or set the macro_block_size to 1
(risking incompatibility).
```

```
<IPython.core.display.Video object>
```

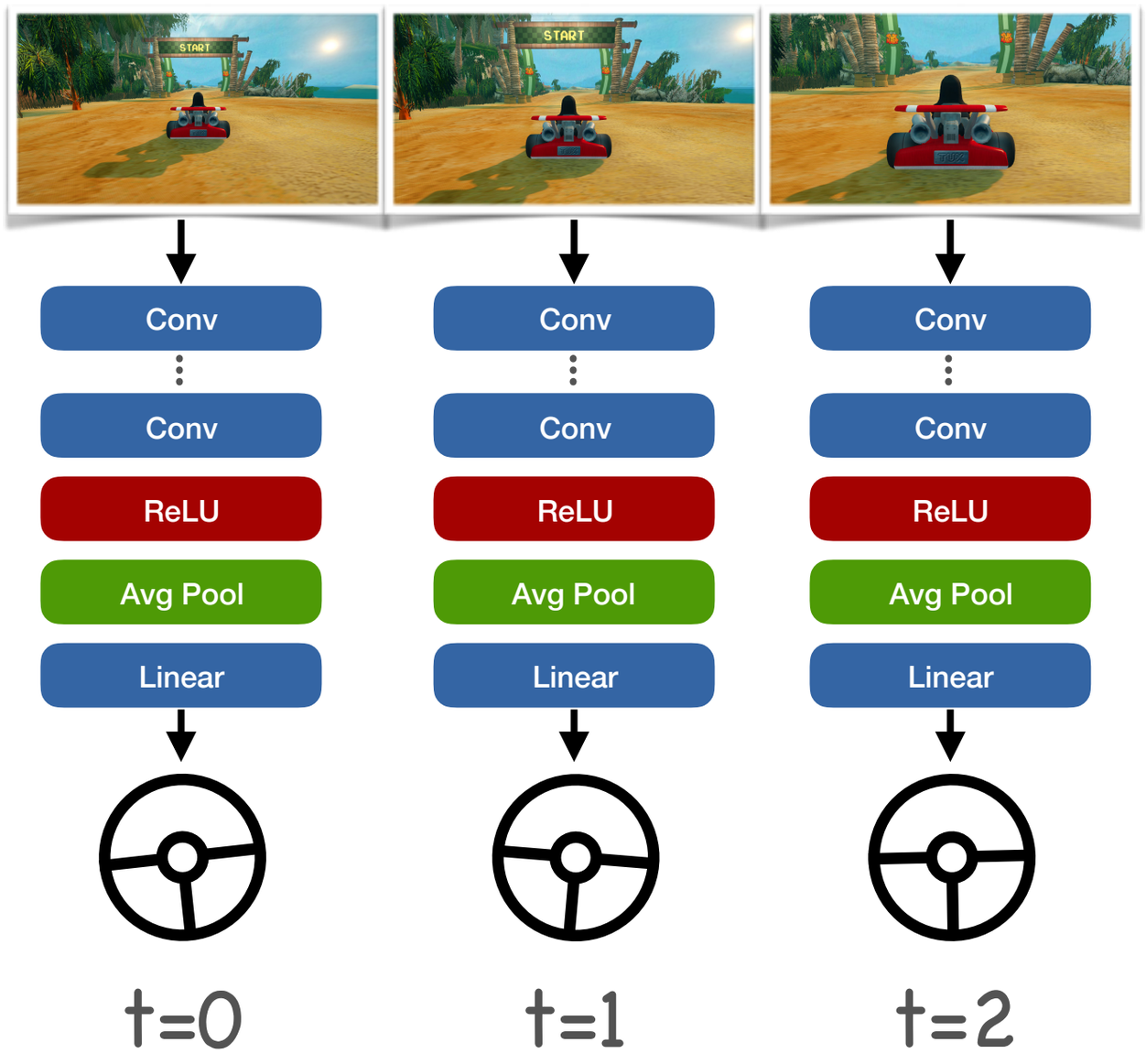
[ ]:
```
# Let's load a fancy auto-pilot. You'll write one yourself in your homework.
from _auto_pilot import auto_pilot

show_agent(auto_pilot, n_steps=600)
```

[ ]:
```
```

Imitation learning

Learning to act

- How do we train our policy?
- Imitation learning
  - Ask an expert



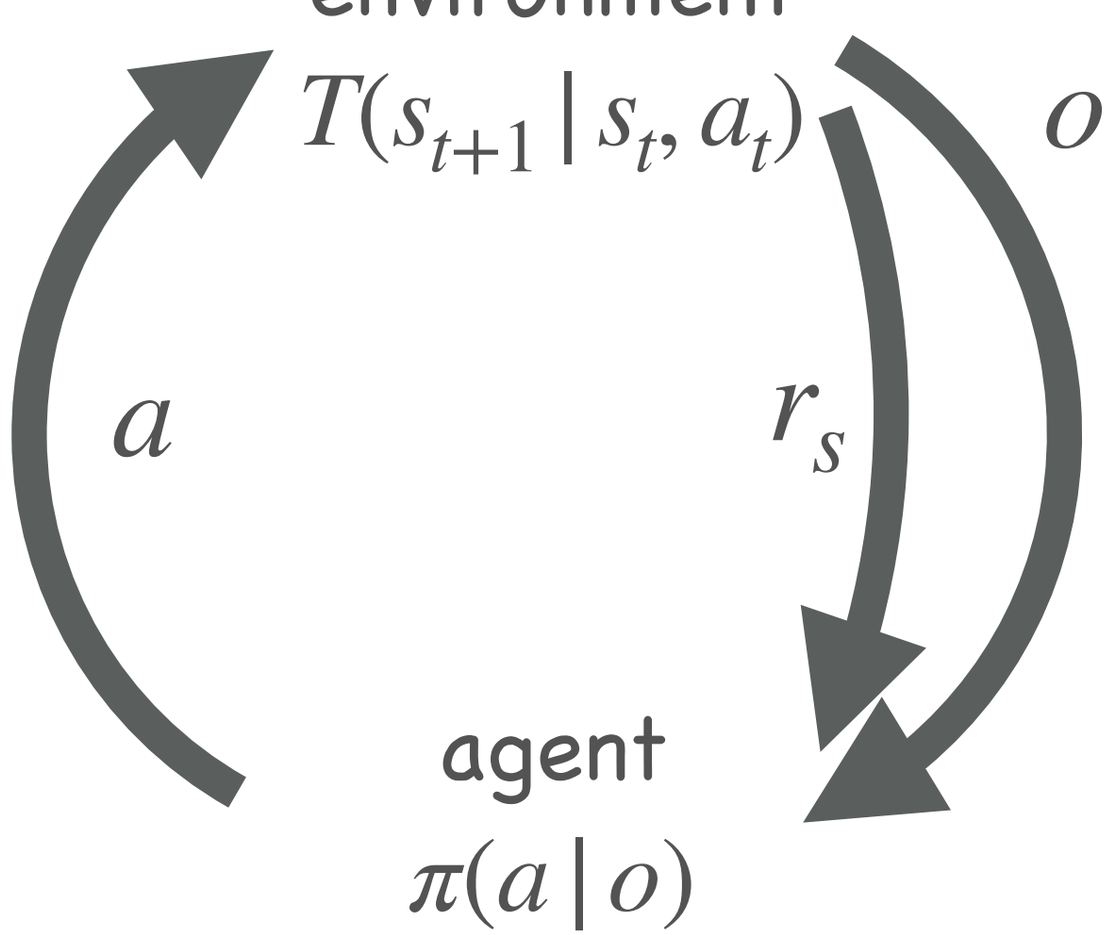
Imitation learning - definition

- Oracle / expert
  - Provides trajectories τ with high return

$$\text{maximize}_{\pi}\ \mathbb{E}_{\tau \sim P_{\pi,T}}[R(\tau)]$$

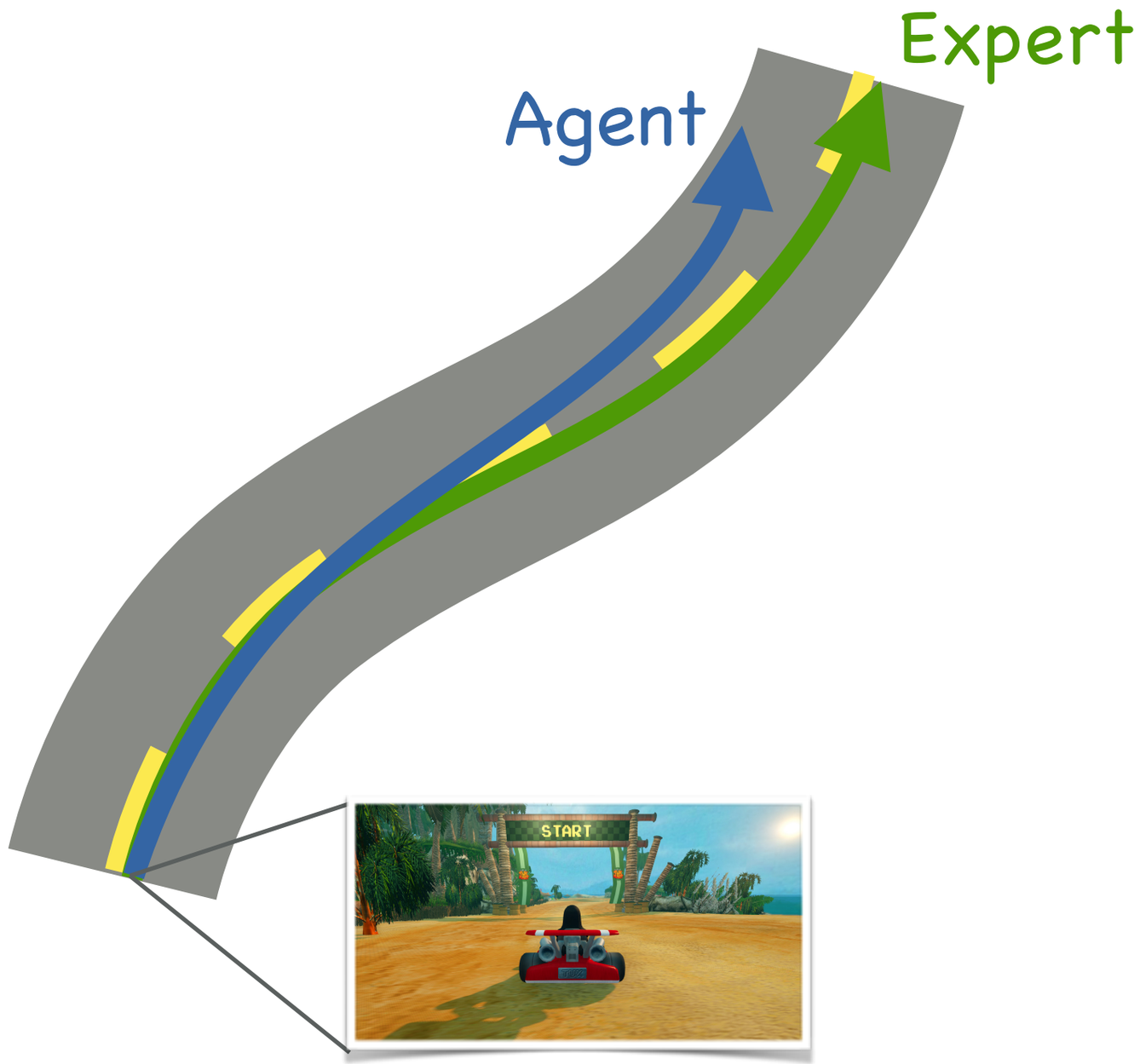
- Policy
  - Supervised learning

$$\text{maximize}_{\pi}\ \mathbb{E}_{\tau}\left[\sum_{t} \log \pi(a_t, s_t)\right]$$



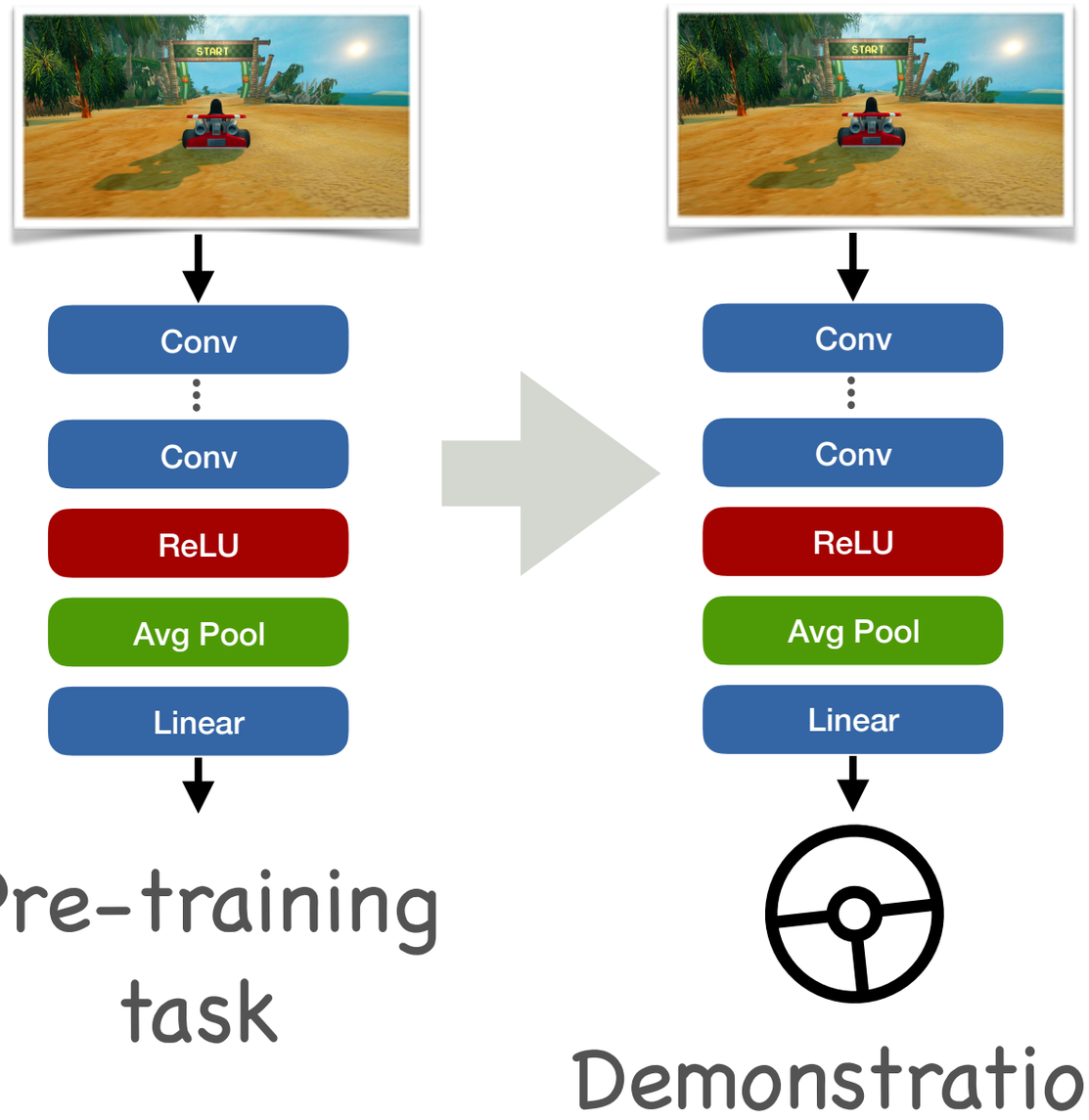
Imitation learning - Issues

- Expert annotations are sometimes expensive
- Drift from expert
  - Distribution mismatch between training and testing



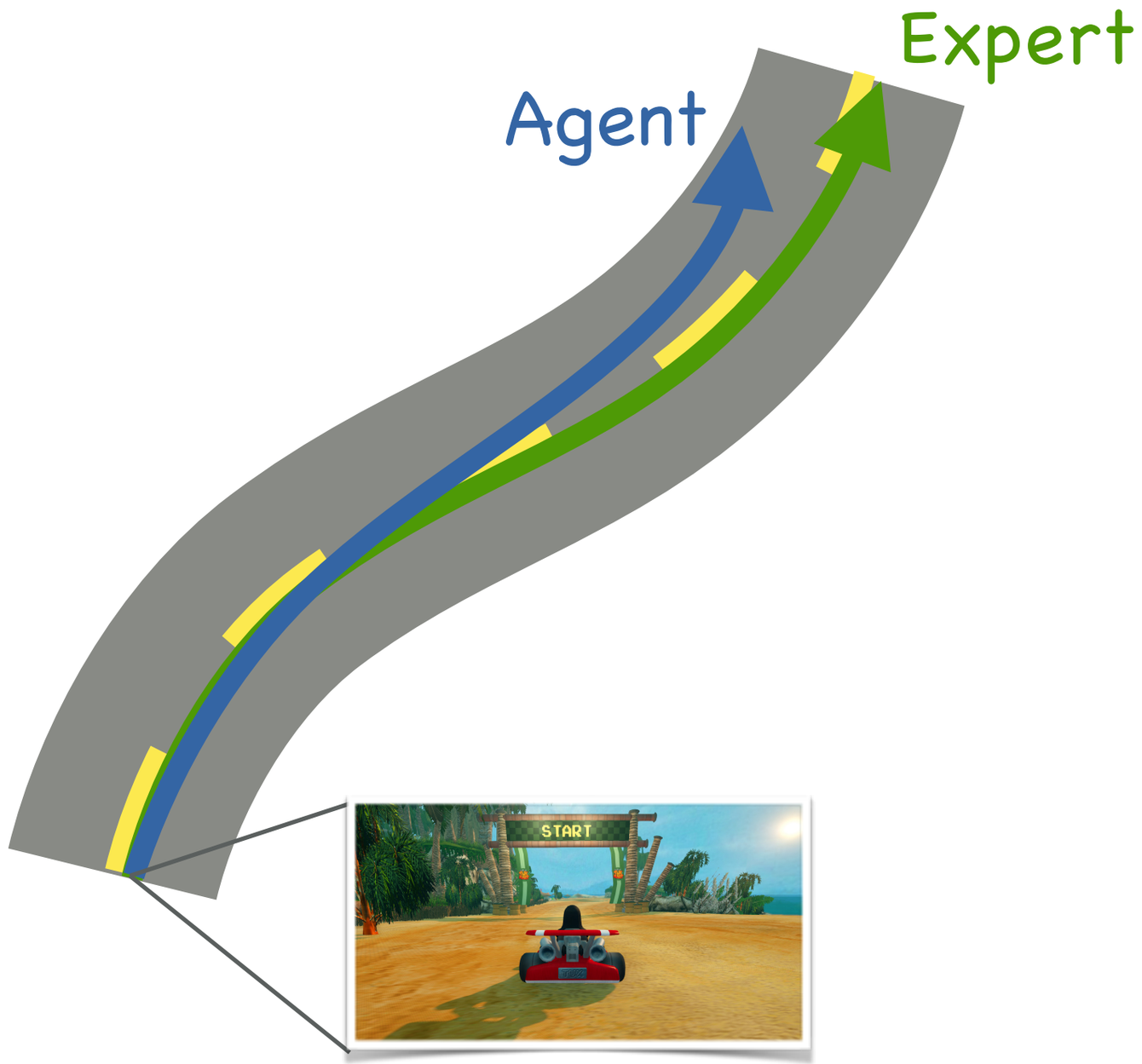
Imitation learning - bag of tricks

- Use pre-trained architecture
- More robust to slight mismatch in observations
- Better generalization



Imitation learning - bag of tricks

- Data augmentation
  - Encourage policy to get back on track



Imitation learning with tricks

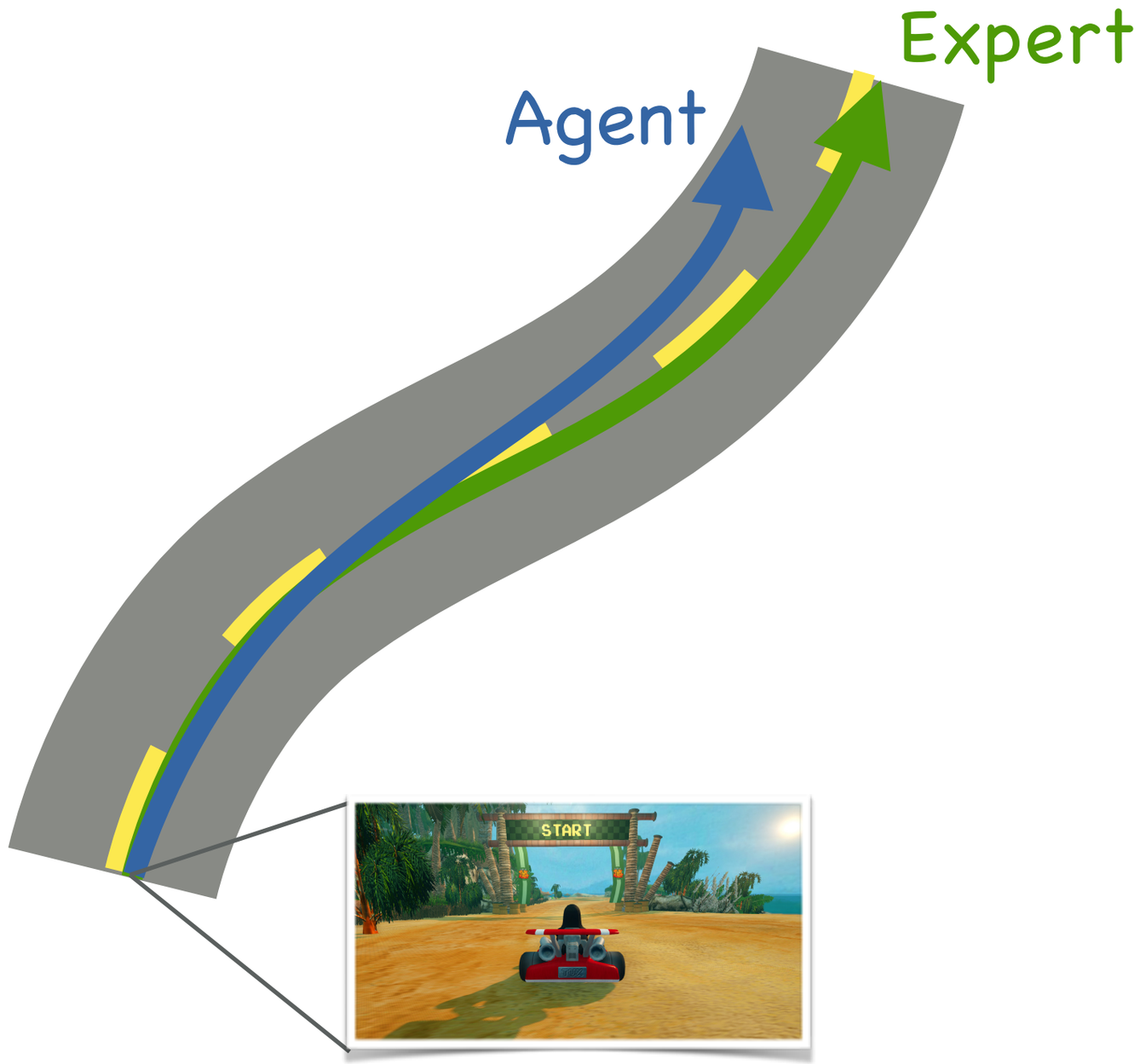
- Easy to train
  - supervised learning
- Sometimes works
  - Major issues with drift / distribution mismatch
- Requires expert annotations

Exploring the Limitations of Behavior Cloning for Autonomous Driving, Codevilla etal. 2019

DAgger

Imitation learning

- Drift
  - Mismatch in training and testing distribution



Imitation learning - Alternative interpretation

- Expert policy π_E
- Agent policy π
- Iterate
  - Take action $a_t^E \sim \pi_E(\cdot \mid s_t)$
  - Imitate $\log \pi(a_t^E \mid s_t)$
  - State update $s_{t+1} \sim T(\cdot \mid a_t^E, s_t)$

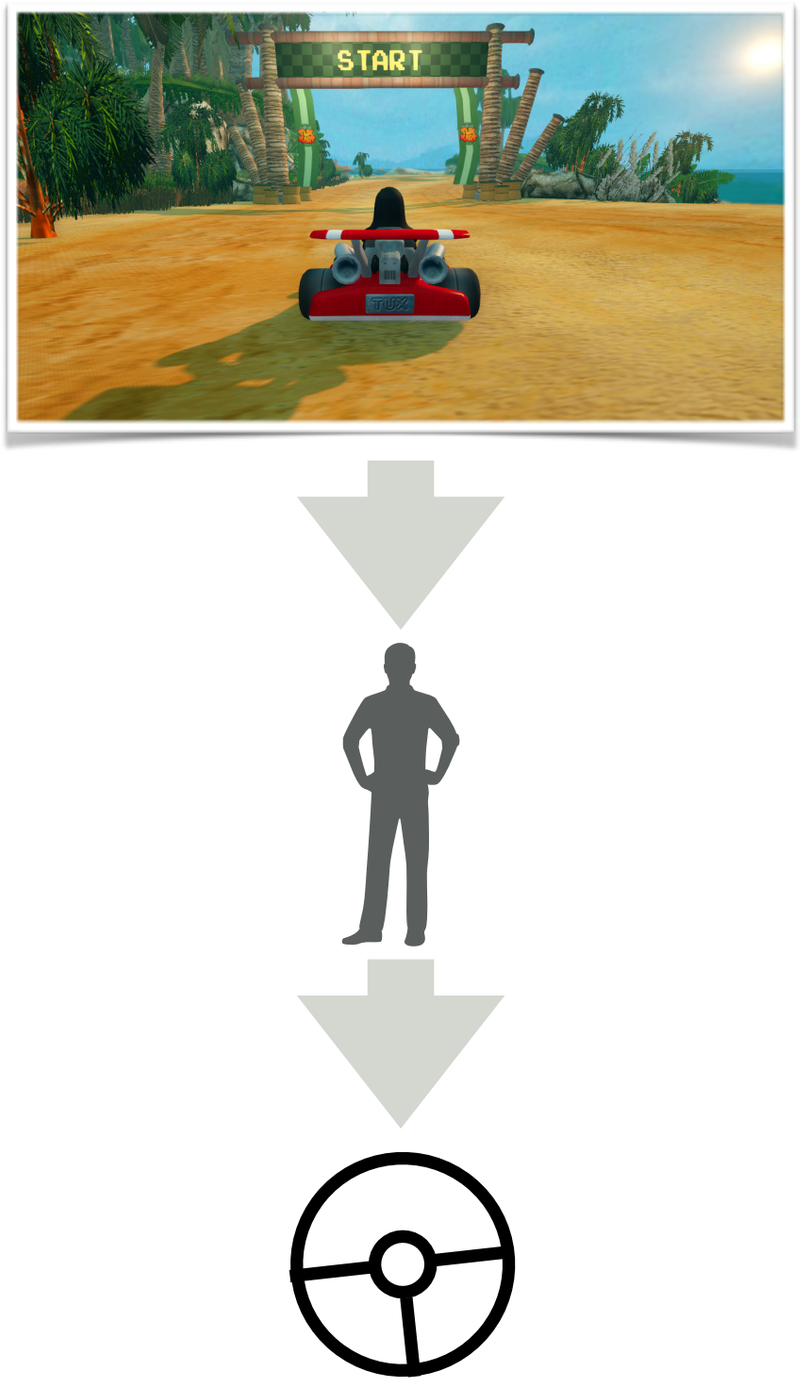
Dataset Aggregation

- Expert policy π_E
- Agent policy π
- Iterate
  - Take action $a_t^E \sim \pi_E(\cdot \mid s_t)$
  - Take action $a_t \sim \pi(\cdot \mid s_t)$
  - Imitate $\log \pi(a_t^E \mid s_t)$
  - State update $s_{t+1} \sim T(\cdot \mid a_t, s_t)$

A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning, Ross et al., AISTATS 2011

DAgger - Issues

- Requires expert oracle
  - Very hard to humans



DAgger

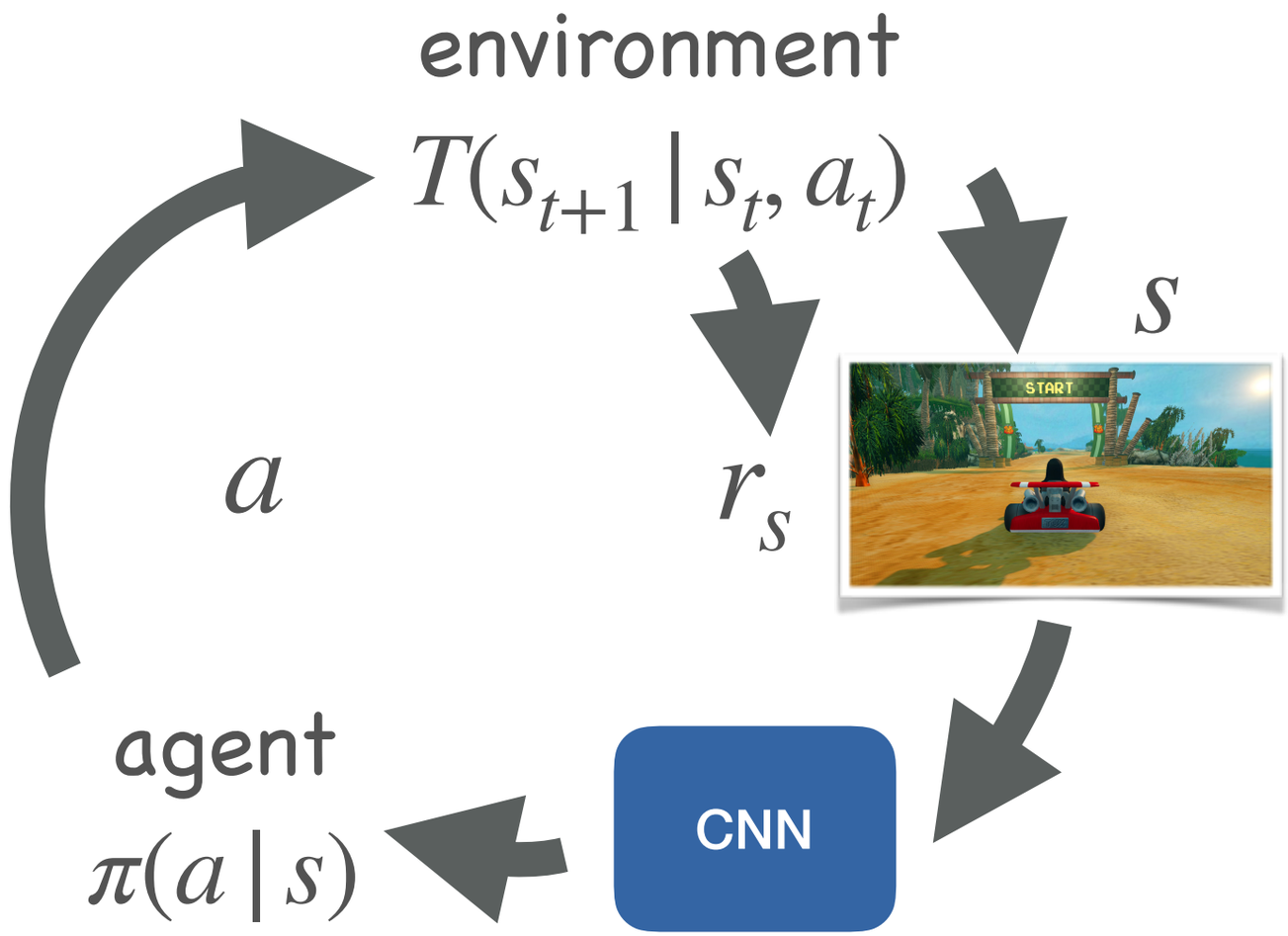
- On-policy imitation learning
- Guaranteed to work for agents with enough capacity and good enough expert

Non-differentiability

Deep learning for action

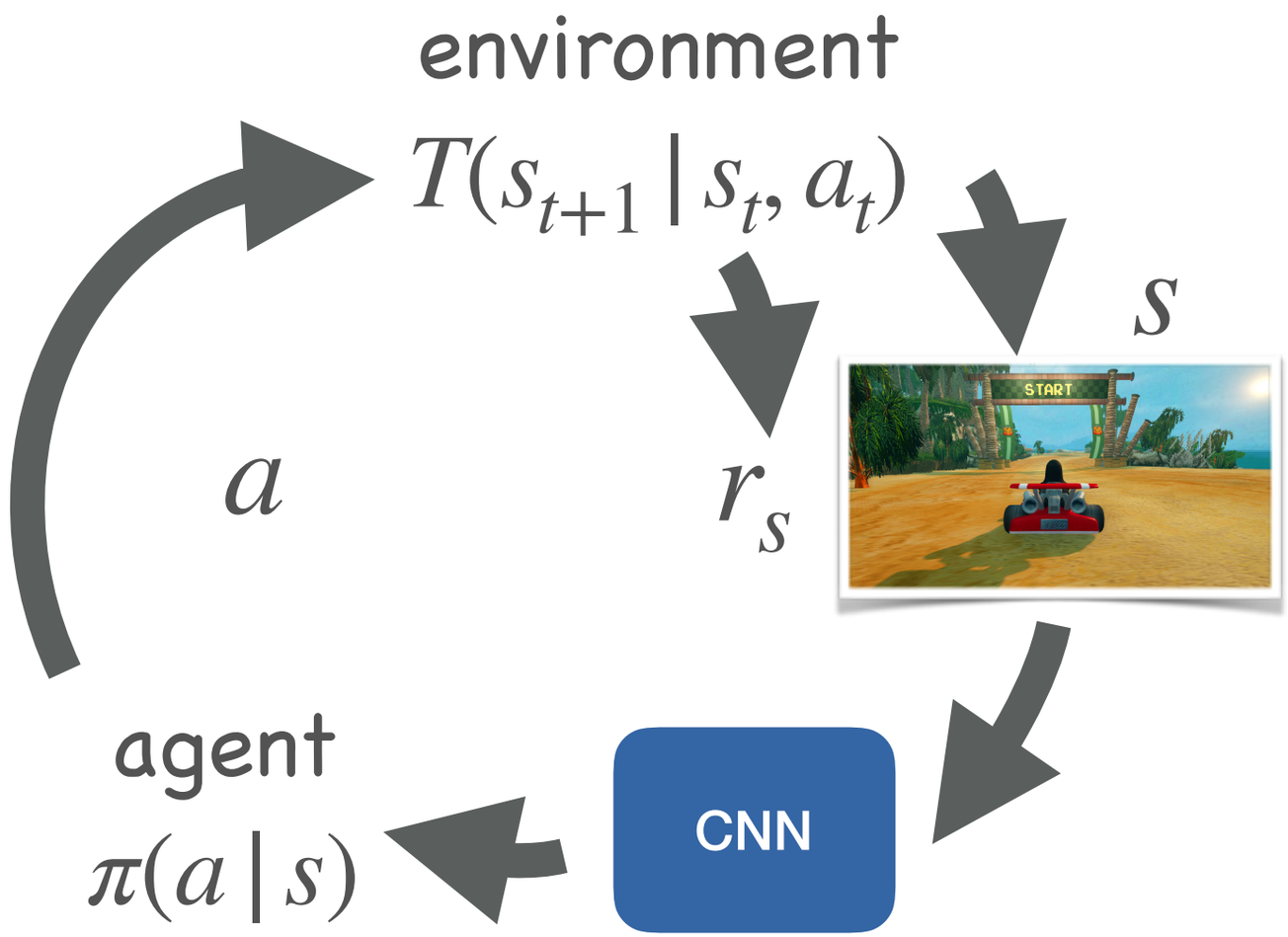
- Why not just learn a policy that maximizes reward?

- Hard to optimize!



Deep learning for action

- Two sources of non-differentiability
  - Sampling
  - Environment



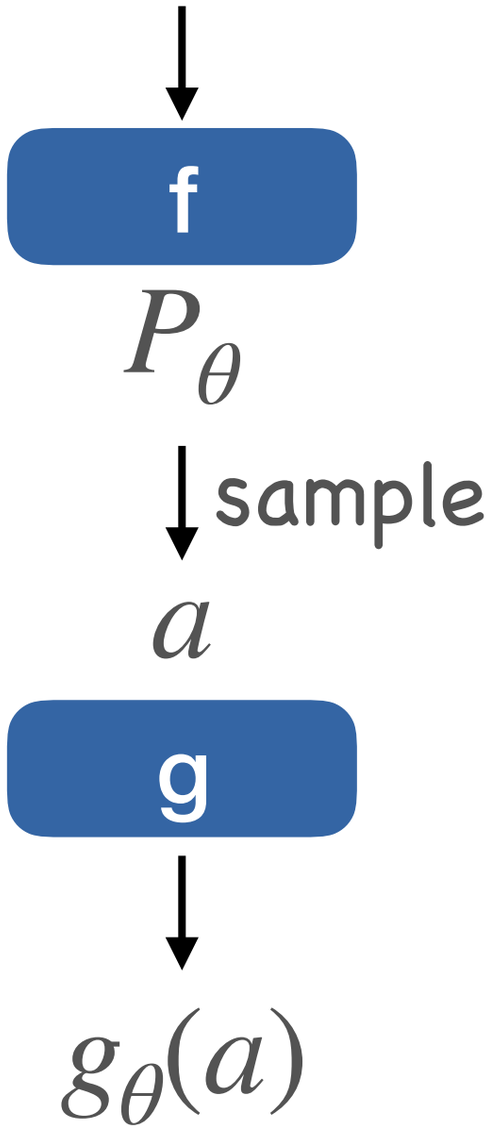
Differentiating sampling

- Compute gradient of

$$\mathbb{E}_{a\sim P_\theta}[g_\theta(a)] = \sum_a P_\theta(a) g_\theta(a)$$

$$\frac{\partial}{\partial\theta}\mathbb{E}_{a\sim P_\theta}[g_\theta(a)] = \sum_a g_\theta(a)\frac{\partial}{\partial\theta}P_\theta(a)$$

- $$+ \sum_a P_\theta(a)\frac{\partial}{\partial\theta}g_\theta(a)$$



Differentiating sampling - Issues

$$\frac{\partial}{\partial\theta}\mathbb{E}_{a\sim P_\theta}[g_\theta(a)] = \sum_a g_\theta(a)\frac{\partial}{\partial\theta}P_\theta(a)$$

$$+\sum_a P_\theta(a)\frac{\partial}{\partial\theta}g_\theta(a)$$

- Large sum over all samples / action
  - Generally intractable

Reparametrization trick

- For continuous distributions
  - Rewrite

$$P_\theta(a) = \frac{1}{\sigma_\theta} P\left(\frac{a - \mu_\theta}{\sigma_\theta}\right)$$

  - e.g. standard normal
  - $\mathbb{E}_{a \sim P_\theta}[g_\theta(a)] = \int_{\tilde{\Omega}} P\left(b\right) g_\theta(b\sigma_\theta + \mu_\theta) db$

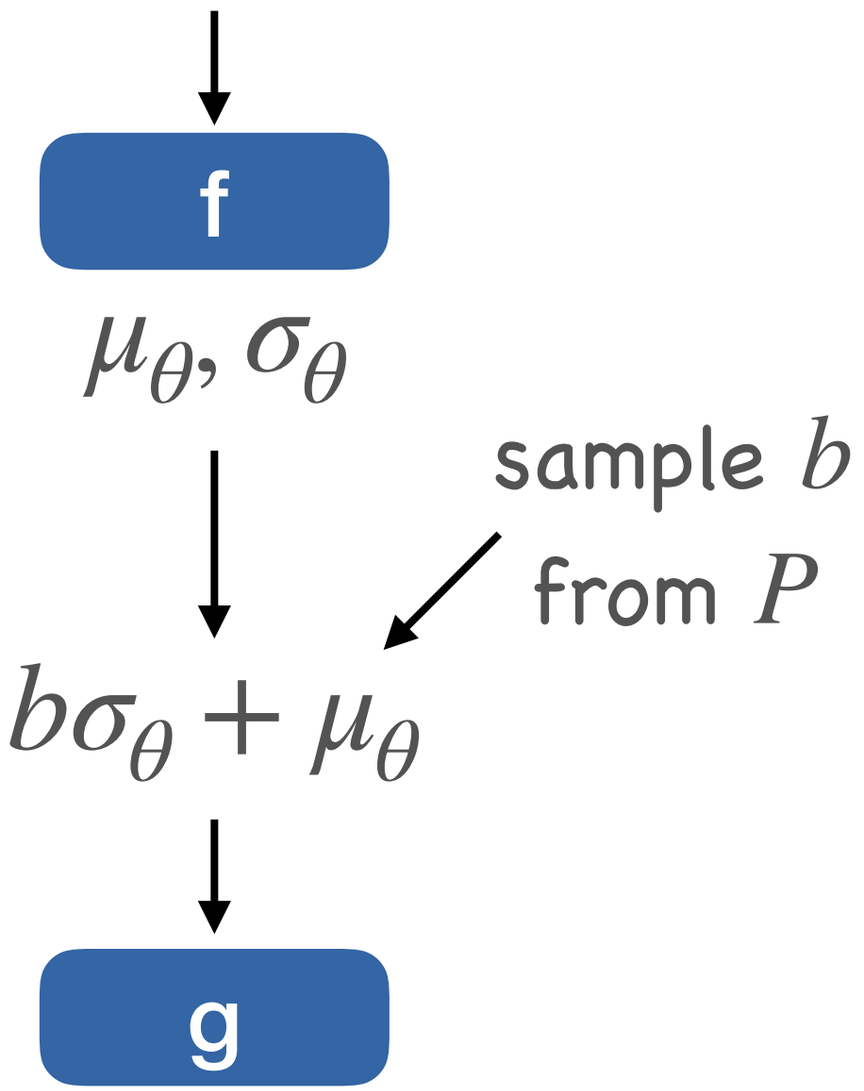
Auto-Encoding Variational Bayes, Kingma and Welling, ICLR 2014

Reparametrization trick

- Compute gradient

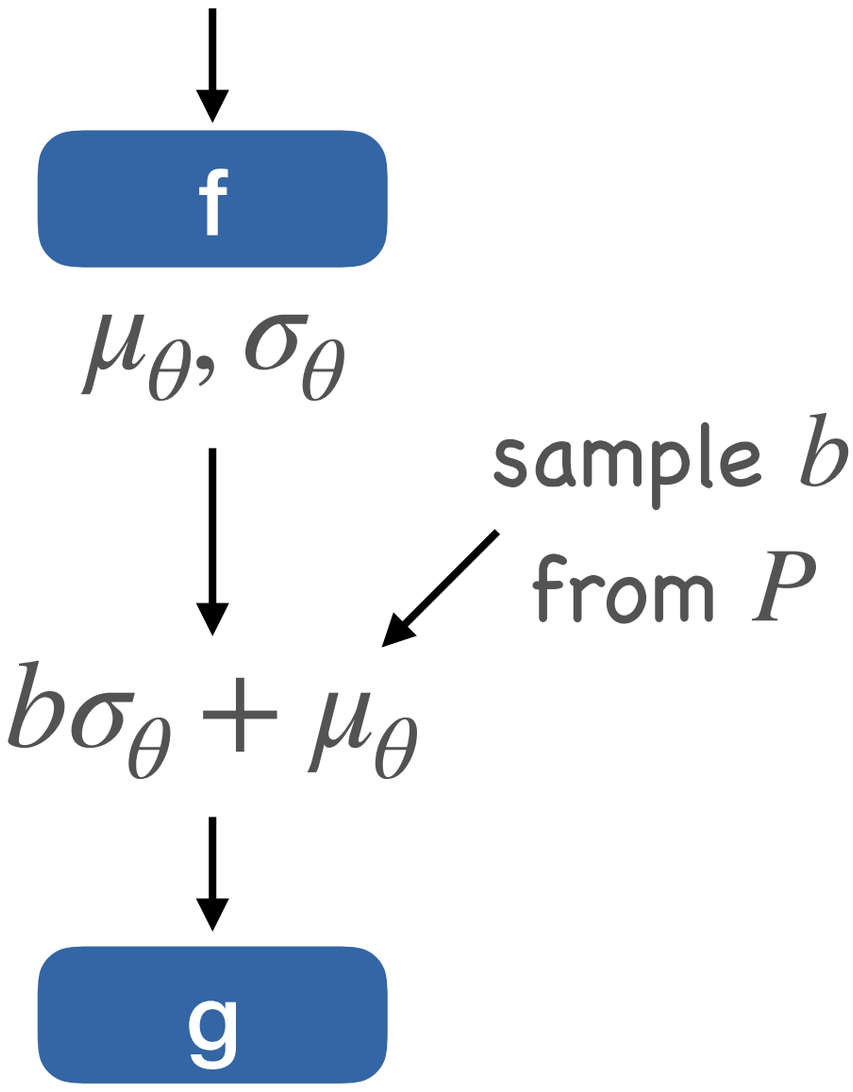
$$\frac{\partial}{\partial\theta}\mathbb{E}_{b\sim P}\left[g_\theta(b\sigma_\theta+\mu_\theta)\right]=\mathbb{E}_{b\sim P}\left[\frac{\partial}{\partial\theta}g_\theta(b\sigma_\theta+\mu_\theta)\right]$$

- Gradient computation by sampling



Reparametrization trick - discrete variables

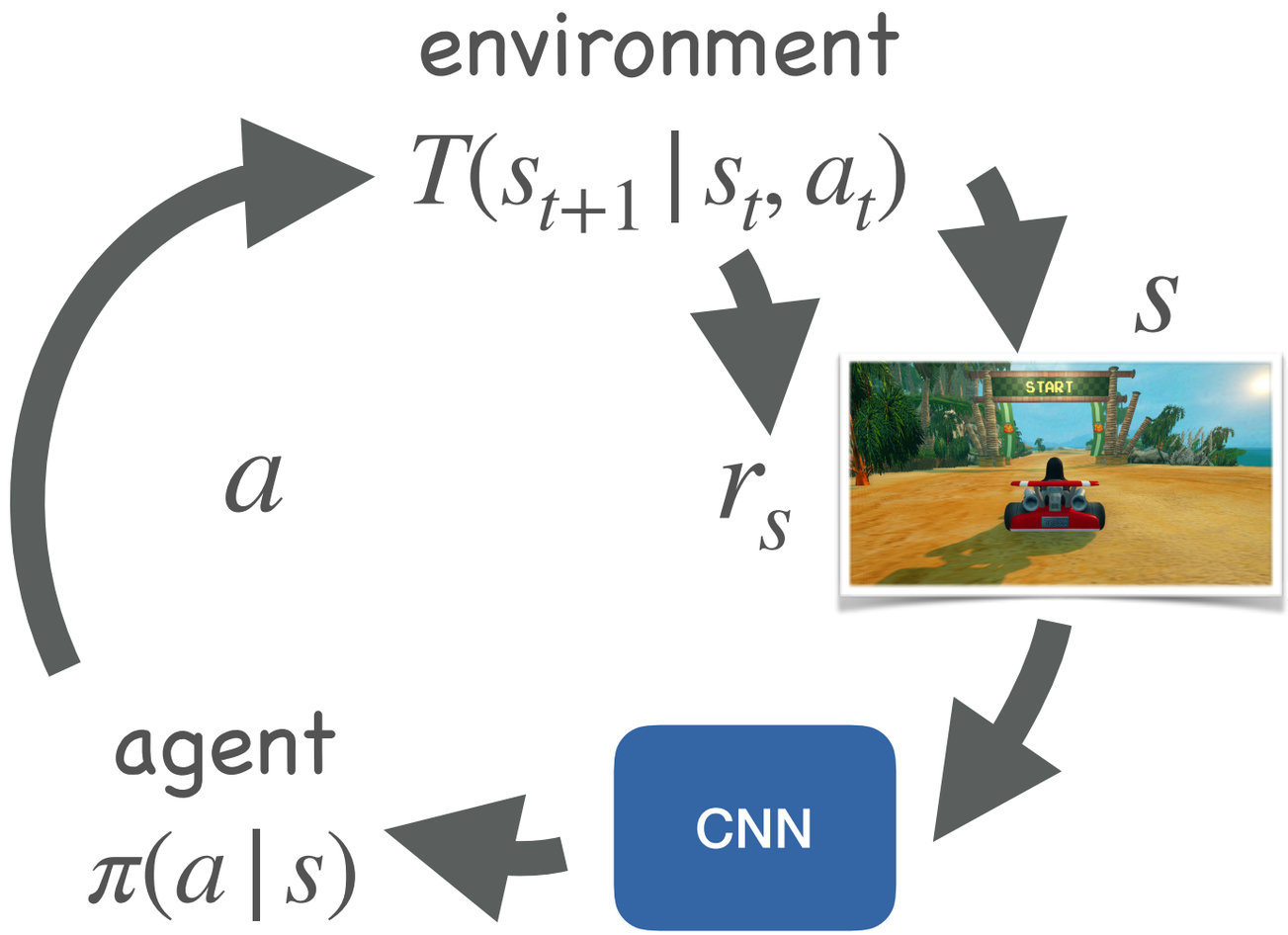
- $\mathbb{E}_{a\sim P_\theta}[g_\theta(a)] = \sum_a P_\theta(a) g_\theta(a)$
- No change of variables
  - No differentiable function that maps to discrete distribution
- Continuous relaxation of one-hot vectors
  - Gumbel softmax



- The Concrete Distribution: a Continuous Relaxation of Discrete Random Variables, Maddison et al., ICLR 2017
- Categorical Reparameterization with Gumbel-Softmax, Jang et al, ICLR 2017

Differentiating the environment

- Quite hard
  - Up next



04

January 23, 2024

[1]:
```
%pylab inline
import torch
import sys, os
import pystk
import ray
device = torch.device('cuda') if torch.cuda.is_available() else torch.
 ↪device('cpu')
print('device = ', device)
ray.init(logging_level=50)
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
device =  cuda
```

[1]:
```
RayContext(dashboard_url='', python_version='3.10.13', ray_version='2.9.1',
ray_commit='cfbf98c315cfb2710c56039a3c96477d196de049', protocol_version=None)
```

[2]:
```
@ray.remote
class Rollout:
    def __init__(self, screen_width, screen_height, hd=True,␣
 ↪track='lighthouse', render=True):
        # Init supertuxkart
        if not render:
            config = pystk.GraphicsConfig.none()
        elif hd:
            config = pystk.GraphicsConfig.hd()
        else:
            config = pystk.GraphicsConfig.ld()
        config.screen_width = screen_width
        config.screen_height = screen_height
        pystk.init(config)

        self.render = render
        race_config = pystk.RaceConfig(track=track)
        self.race = pystk.Race(race_config)
        self.race.start()
```

```
    def __call__(self, agent, n_steps=200):
        torch.set_num_threads(1)
        self.race.restart()
        self.race.step()
        data = []
        track_info = pystk.Track()
        track_info.update()

        for i in range(n_steps):
            world_info = pystk.WorldState()
            world_info.update()

            # Gather world information
            kart_info = world_info.players[0].kart

            agent_data = {'track_info': track_info, 'kart_info': kart_info}
            if self.render:
                agent_data['image'] = np.array(self.race.render_data[0].image)

            # Act
            action = agent(**agent_data)
            agent_data['action'] = action

            # Take a step in the simulation
            self.race.step(action)

            # Save all the relevant data
            data.append(agent_data)
        return data

def show_video(frames, fps=30):
    import imageio
    from IPython.display import Video, display

    imageio.mimwrite('/tmp/test.mp4', frames, fps=fps, bitrate=10000000)
    display(Video('/tmp/test.mp4', width=800, height=600, embed=True))

viz_rollout = Rollout.remote(100, 75)
def show_agent(agent, n_steps=600):
    data = ray.get(viz_rollout.__call__.remote(agent, n_steps=n_steps))
    show_video([d['image'] for d in data])

rollouts = [Rollout.remote(100, 75) for i in range(10)]
def rollout_many(many_agents, **kwargs):
    ray_data = []
    for i, agent in enumerate(many_agents):
```

```
        ray_data.append( rollouts[i % len(rollouts)].__call__.remote(agent,␣
 ↪**kwargs) )
    return ray.get(ray_data)

def dummy_agent(**kwargs):
    action = pystk.Action()
    action.acceleration = 1
    return action

# Let's load a fancy auto-pilot. You'll write one yourself in your homework.
from _auto_pilot import auto_pilot

show_agent(auto_pilot)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[2], line 75
     72     return action
     74 # Let's load a fancy auto-pilot. You'll write one yourself in your␣
 ↪homework.
---> 75 from _auto_pilot import auto_pilot
     77 show_agent(auto_pilot)

ModuleNotFoundError: No module named '_auto_pilot'
```

[3]:
```
from torchvision.transforms import functional as TF

class ActionNet(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.network = torch.nn.Sequential(
            torch.nn.BatchNorm2d(3),
            torch.nn.Conv2d(3, 16, 5, stride=2),
            torch.nn.ReLU(),
            torch.nn.Conv2d(16, 32, 5, stride=2),
            torch.nn.ReLU(),
            torch.nn.Conv2d(32, 32, 5, stride=2),
            torch.nn.ReLU(),
            torch.nn.Conv2d(32, 32, 5, stride=2),
            torch.nn.ReLU()
        )
        self.classifier = torch.nn.Linear(32, 2)

    def forward(self, x):
        f = self.network(x)
        return self.classifier(f.mean(dim=(2,3)))
```

```
class Actor:
    def __init__(self, action_net):
        self.action_net = action_net.cpu().eval()

    def __call__(self, image, **kwargs):
        output = self.action_net(TF.to_tensor(image)[None])[0]

        action = pystk.Action()
        action.acceleration = 1
        action.steer = output[0]
        action.drift = output[1] > 0.5
        return action


def noisy_actor(actor, noise_std=5):
    def act(**kwargs):
        action = actor(**kwargs)
        action.steer += np.random.normal(0, noise_std)
        return action
    return act


action_net = ActionNet()
actor = Actor(action_net)
```

[4]:
```
%load_ext tensorboard
import tempfile
log_dir = 'log_04'
%tensorboard --logdir {log_dir} --reload_interval 1
```

```
<IPython.core.display.HTML object>
```

[5]:
```
import torch.utils.tensorboard as tb
from datetime import datetime

n_epochs = 10
batch_size = 128
n_trajectories = 10

# Create the network
action_net = ActionNet().to(device)

# Create the optimizer
optimizer = torch.optim.Adam(action_net.parameters())

# Create the loss
```

```
loss = torch.nn.MSELoss()

# Collect the data
train_data = []
for data in rollout_many([auto_pilot]*n_trajectories):
    train_data.extend(data)

# Upload to the GPU
train_images = torch.stack([torch.as_tensor(d['image']) for d in train_data]).
 ↪permute(0,3,1,2).to(device).float()/255.
train_labels = torch.stack([torch.as_tensor((d['action'].steer, d['action'].
 ↪drift)) for d in train_data]).to(device).float()

# Start training
global_step = 0
action_net.train().to(device)
logger = tb.SummaryWriter(log_dir+'/'+str(datetime.now()), flush_secs=1)

for epoch in range(n_epochs):
    for iteration in range(0, len(train_data), batch_size):
        batch_ids = torch.randint(0, len(train_data), (batch_size,),␣
 ↪device=device)
        batch_images = train_images[batch_ids]
        batch_labels = train_labels[batch_ids]
        o = action_net(batch_images)
        loss_val = loss(o, batch_labels)

        logger.add_scalar('train/loss', loss_val, global_step)
        global_step += 1

        optimizer.zero_grad()
        loss_val.backward()
        optimizer.step()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[5], line 19
     17 # Collect the data
     18 train_data = []
---> 19 for data in rollout_many([auto_pilot]*n_trajectories):
     20     train_data.extend(data)
     22 # Upload to the GPU

NameError: name 'auto_pilot' is not defined
```

[ ]:
```
show_agent(Actor(action_net), n_steps=600)
```

[8]:
```
# Collect the data
for data in rollout_many([Actor(action_net)]*n_trajectories):
    train_data.extend(data)
# for data in rollout_many([noisy_actor(auto_pilot, noise_std=i//2) for i in␣
 ↪range(n_trajectories)]):
#     train_data.extend(data)

# Supervise using the auto-pilot actions
for d in train_data:
    d['action'] = auto_pilot(**d)

# Upload to the GPU
train_images = torch.stack([torch.as_tensor(d['image']) for d in train_data]).
 ↪permute(0,3,1,2).to(device).float()/255.
train_labels = torch.stack([torch.as_tensor((d['action'].steer, d['action'].
 ↪drift)) for d in train_data]).to(device).float()

# Start training
logger = tb.SummaryWriter(log_dir+'/'+str(datetime.now()), flush_secs=1)
global_step = 0
action_net.train().to(device)

for epoch in range(n_epochs):
    for iteration in range(0, len(train_data), batch_size):
        batch_ids = torch.randint(0, len(train_data), (batch_size,),␣
 ↪device=device)
        batch_images = train_images[batch_ids]
        batch_labels = train_labels[batch_ids]
        o = action_net(batch_images)
        loss_val = loss(o, batch_labels)

        logger.add_scalar('train/loss', loss_val, global_step)
        global_step += 1

        optimizer.zero_grad()
        loss_val.backward()
        optimizer.step()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[8], line 9
      4 # for data in rollout_many([noisy_actor(auto_pilot, noise_std=i//2) for␣
 ↪i in range(n_trajectories)]):
      5 #     train_data.extend(data)
      6
      7 # Supervise using the auto-pilot actions
      8 for d in train_data:
```

```
----> 9     d['action'] = auto_pilot(**d)
     11 # Upload to the GPU
     12 train_images = torch.stack([torch.as_tensor(d['image']) for d in␣
 ↪train_data]).permute(0,3,1,2).to(device).float()/255.

NameError: name 'auto_pilot' is not defined
```

[ ]:
```
show_agent(Actor(action_net), n_steps=600)
```

[ ]:
```

```

05b

January 23, 2024

[1]:
```
%pylab inline
import torch
import sys, os
import pystk
import ray
device = torch.device('cuda') if torch.cuda.is_available() else torch.
 ↪device('cpu')
print('device = ', device)
ray.init(logging_level=50)
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
device =  cuda
```

[1]:
```
RayContext(dashboard_url='', python_version='3.10.13', ray_version='2.9.1',
ray_commit='cfbf98c315cfb2710c56039a3c96477d196de049', protocol_version=None)
```

```
(raylet) [2024-01-23 09:44:28,547 E 7315 7315] (raylet)
node_manager.cc:3022: 2 Workers (tasks / actors) killed due to memory pressure
(OOM), 0 Workers crashed due to other reasons at node (ID:
c390692744c464f918a4ee414045f7d2158877bae42e6c8503ee03be, IP: 10.64.35.146) over
the last time period. To see more information about the Workers killed on this
node, use `ray logs raylet.out -ip 10.64.35.146`
(raylet)
(raylet) Refer to the documentation on how to address the out of memory
issue: https://docs.ray.io/en/latest/ray-core/scheduling/ray-oom-
prevention.html. Consider provisioning more memory on this node or reducing task
parallelism by requesting more CPUs per task. To adjust the kill threshold, set
the environment variable `RAY_memory_usage_threshold` when starting Ray. To
disable worker killing, set the environment variable
`RAY_memory_monitor_refresh_ms` to zero.
```

[2]:
```
@ray.remote
class Rollout:
    def __init__(self, screen_width, screen_height, hd=True,␣
 ↪track='lighthouse', render=True, frame_skip=1):
        # Init supertuxkart
        if not render:
```

```
            config = pystk.GraphicsConfig.none()
        elif hd:
            config = pystk.GraphicsConfig.hd()
        else:
            config = pystk.GraphicsConfig.ld()
        config.screen_width = screen_width
        config.screen_height = screen_height
        pystk.init(config)

        self.frame_skip = frame_skip
        self.render = render
        race_config = pystk.RaceConfig(track=track)
        self.race = pystk.Race(race_config)
        self.race.start()

    def __call__(self, agent, n_steps=200):
        torch.set_num_threads(1)
        self.race.restart()
        self.race.step()
        data = []
        track_info = pystk.Track()
        track_info.update()

        for i in range(n_steps // self.frame_skip):
            world_info = pystk.WorldState()
            world_info.update()

            # Gather world information
            kart_info = world_info.players[0].kart

            agent_data = {'track_info': track_info, 'kart_info': kart_info}
            if self.render:
                agent_data['image'] = np.array(self.race.render_data[0].image)

            # Act
            action = agent(**agent_data)
            agent_data['action'] = action

            # Take a step in the simulation
            for it in range(self.frame_skip):
                self.race.step(action)

            # Save all the relevant data
            data.append(agent_data)
        return data

def show_video(frames, fps=30):
```

```
    import imageio
    from IPython.display import Video, display

    imageio.mimwrite('/tmp/test.mp4', frames, fps=fps, bitrate=1000000)
    display(Video('/tmp/test.mp4', width=800, height=600, embed=True))

viz_rollout = Rollout.remote(400, 300)
def show_agent(agent, n_steps=600):
    data = ray.get(viz_rollout.__call__.remote(agent, n_steps=n_steps))
    show_video([d['image'] for d in data])

rollouts = [Rollout.remote(50, 50, hd=False, render=False, frame_skip=5) for i␣
 ↪in range(10)]
def rollout_many(many_agents, **kwargs):
    ray_data = []
    for i, agent in enumerate(many_agents):
        ray_data.append( rollouts[i % len(rollouts)].__call__.remote(agent,␣
 ↪**kwargs) )
    return ray.get(ray_data)

def dummy_agent(**kwargs):
    action = pystk.Action()
    action.acceleration = 1
    return action
```

[3]:
```
def three_points_on_track(distance, track):
    distance = np.clip(distance, track.path_distance[0,0], track.
 ↪path_distance[-1,1]).astype(np.float32)
    valid_node = (track.path_distance[..., 0] <= distance) & (distance <= track.
 ↪path_distance[..., 1])
    valid_node_idx, = np.where(valid_node)
    node_idx = valid_node_idx[0] # np.random.choice(valid_node_idx)
    d = track.path_distance[node_idx].astype(np.float32)
    x = track.path_nodes[node_idx][:,[0,2]].astype(np.float32) # Ignore the y␣
 ↪coordinate
    w, = track.path_width[node_idx].astype(np.float32)

    t = (distance - d[0]) / (d[1] - d[0])
    mid = x[1] * t + x[0] * (1 - t)
    x10 = (x[1] - x[0]) / np.linalg.norm(x[1]-x[0])
    x10_ortho = np.array([-x10[1],x10[0]], dtype=float32)
    return mid - w / 2 * x10_ortho, mid, mid + w / 2 * x10_ortho


def state_features(track_info, kart_info, absolute=False, **kwargs):
    f = np.concatenate([three_points_on_track(kart_info.distance_down_track +␣
 ↪d, track_info) for d in [0,5,10,15,20]])
```

```
    if absolute:
        return f
    p = np.array(kart_info.location)[[0,2]].astype(np.float32)
    t = np.array(kart_info.front)[[0,2]].astype(np.float32)
    f = f - p[None]
    d = (p-t) / np.linalg.norm(p-t)
    d_o = np.array([-d[1], d[0]], dtype=float32)
    return np.stack([f.dot(d), f.dot(d_o)], axis=1)

# Let's load a fancy auto-pilot. You'll write one yourself in your homework.
from _auto_pilot import auto_pilot
data, = rollout_many([auto_pilot], n_steps=400)

figure()
f = state_features(**data[50])
plot(f[:,1].flat, f[:,0].flat, '*')
axis('equal')
gca().invert_yaxis()

figure()
for d in data:
    f = state_features(**d, absolute=True)
    plot(f[:,1].flat, f[:,0].flat, '*')
axis('equal')
gca().invert_yaxis()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[3], line 29
     26     return np.stack([f.dot(d), f.dot(d_o)], axis=1)
     28 # Let's load a fancy auto-pilot. You'll write one yourself in your␣
 ↪homework.
---> 29 from _auto_pilot import auto_pilot
     30 data, = rollout_many([auto_pilot], n_steps=400)
     32 figure()

ModuleNotFoundError: No module named '_auto_pilot'
```

[4]:
```
from torch.distributions import Bernoulli

def new_action_net():
    return torch.nn.Linear(2*5*3, 1, bias=False)

class Actor:
    def __init__(self, action_net):
        self.action_net = action_net.cpu().eval()
```

```
    def __call__(self, track_info, kart_info, **kwargs):
        f = state_features(track_info, kart_info)
        output = self.action_net(torch.as_tensor(f).view(1,-1))[0]

        action = pystk.Action()
        action.acceleration = 1
        steer_dist = Bernoulli(logits=output[0])
        action.steer = steer_dist.sample()*2-1
        return action

class GreedyActor:
    def __init__(self, action_net):
        self.action_net = action_net.cpu().eval()

    def __call__(self, track_info, kart_info, **kwargs):
        f = state_features(track_info, kart_info)
        output = self.action_net(torch.as_tensor(f).view(1,-1))[0]

        action = pystk.Action()
        action.acceleration = 1
        action.steer = output[0]
        return action
```

[5]:
```
action_net = new_action_net()
show_agent(Actor(action_net))
```

```
IMAGEIO FFMPEG_WRITER WARNING: input image is not divisible by
macro_block_size=16, resizing from (400, 300) to (400, 304) to ensure video
compatibility with most codecs and players. To prevent resizing, make your input
image divisible by the macro_block_size or set the macro_block_size to 1
(risking incompatibility).
```

```
<IPython.core.display.Video object>
```

[ ]:
```
many_action_nets = [new_action_net() for i in range(10)]

data = rollout_many([Actor(action_net) for action_net in many_action_nets],␣
 ↪n_steps=600)

good_initialization = many_action_nets[ np.argmax([d[-1]['kart_info'].
 ↪overall_distance for d in data]) ]
```

[ ]:
```
show_agent(Actor(good_initialization))
```

Recall what we're trying to do in RL: maximize the expected return of a policy π (or in turn minmize a los L)

$$-L = E_{\tau \sim P_\pi}[R(\tau)],$$

where $\tau = \{s_0, a_0, s_1, a_1, ...\}$ is a trajectory of states and actions. The return of a trajectory is then defined as the sum of individual rewards $R(\tau) = \sum_k r(s_k)$ (we won't discount in this assignment).

Policy gradient computes the gradient of the loss L using the log-derivative trick

$$\nabla_\pi L = -E_{\tau \sim P_\pi}[\sum_k r(s_k) \nabla_\pi \sum_i \log \pi(a_i|s_i)].$$

Since the return $r(s_k)$ only depends on action a_i in the past $i < k$ we can further simplify the above equation:

$$\nabla_\pi L = -E_{\tau \sim P_\pi} \left[\sum_i \left(\nabla_\pi \log \pi(a_i|s_i) \right) \left(\sum_{k=i}^{i+T} r(s_k) \right) \right].$$

We will implement an estimator for this objective below. There are a few steps that we need to follow:

- The expectation $E_{\tau \sim P_\pi}$ are rollouts of our policy
- The log probability $\log \pi(a_i|s_i)$ uses the `Categorical.log_prob`
- Gradient computation uses the `.backward()` function
- The gradient $\nabla_\pi L$ is then used in a standard optimizer

[ ]:
```
import copy

n_epochs = 20
n_trajectories = 10
n_iterations =50
batch_size = 128
T = 20

action_net = copy.deepcopy(good_initialization)

best_action_net = copy.deepcopy(action_net)

optim = torch.optim.Adam(action_net.parameters(), lr=1e-3)


for epoch in range(n_epochs):
    eps = 1e-2

    # Roll out the policy, compute the Expectation
    trajectories = rollout_many([Actor(action_net)]*n_trajectories, n_steps=600)
    print('epoch = %d   best_dist = '%epoch, np.max([t[-1]['kart_info'].
 ↪overall_distance for t in trajectories]))

    # Compute all the reqired quantities to update the policy
    features = []
    returns = []
    actions = []
    for trajectory in trajectories:
        for i in range(len(trajectory)):
```

```
            # Compute the returns
            returns.append( trajectory[min(i+T,␣
↪len(trajectory)-1)]['kart_info'].overall_distance -
                            trajectory[i]['kart_info'].overall_distance )
            # Compute the features
            features.append( torch.as_tensor(state_features(**trajectory[i]),␣
↪dtype=torch.float32).cuda().view(-1) )
            # Store the action that we took
            actions.append( trajectory[i]['action'].steer > 0 )

    # Upload everything to the GPU
    returns = torch.as_tensor(returns, dtype=torch.float32).cuda()
    actions = torch.as_tensor(actions, dtype=torch.float32).cuda()
    features = torch.stack(features).cuda()

    returns = (returns - returns.mean()) / returns.std()

    action_net.train().cuda()
    avg_expected_log_return = []
    for it in range(n_iterations):
        batch_ids = torch.randint(0, len(returns), (batch_size,), device=device)
        batch_returns = returns[batch_ids]
        batch_actions = actions[batch_ids]
        batch_features = features[batch_ids]

        output = action_net(batch_features)
        pi = Bernoulli(logits=output[:,0])

        expected_log_return = (pi.log_prob(batch_actions)*batch_returns).mean()
        optim.zero_grad()
        (-expected_log_return).backward()
        optim.step()
        avg_expected_log_return.append(float(expected_log_return))

    best_performance, current_performance =␣
↪rollout_many([GreedyActor(best_action_net), GreedyActor(action_net)],␣
↪n_steps=600)
    if best_performance[-1]['kart_info'].overall_distance <␣
↪current_performance[-1]['kart_info'].overall_distance:
        best_action_net = copy.deepcopy(action_net)
```

[ ]:
```
show_agent(GreedyActor(best_action_net))
```

[ ]:
```

```

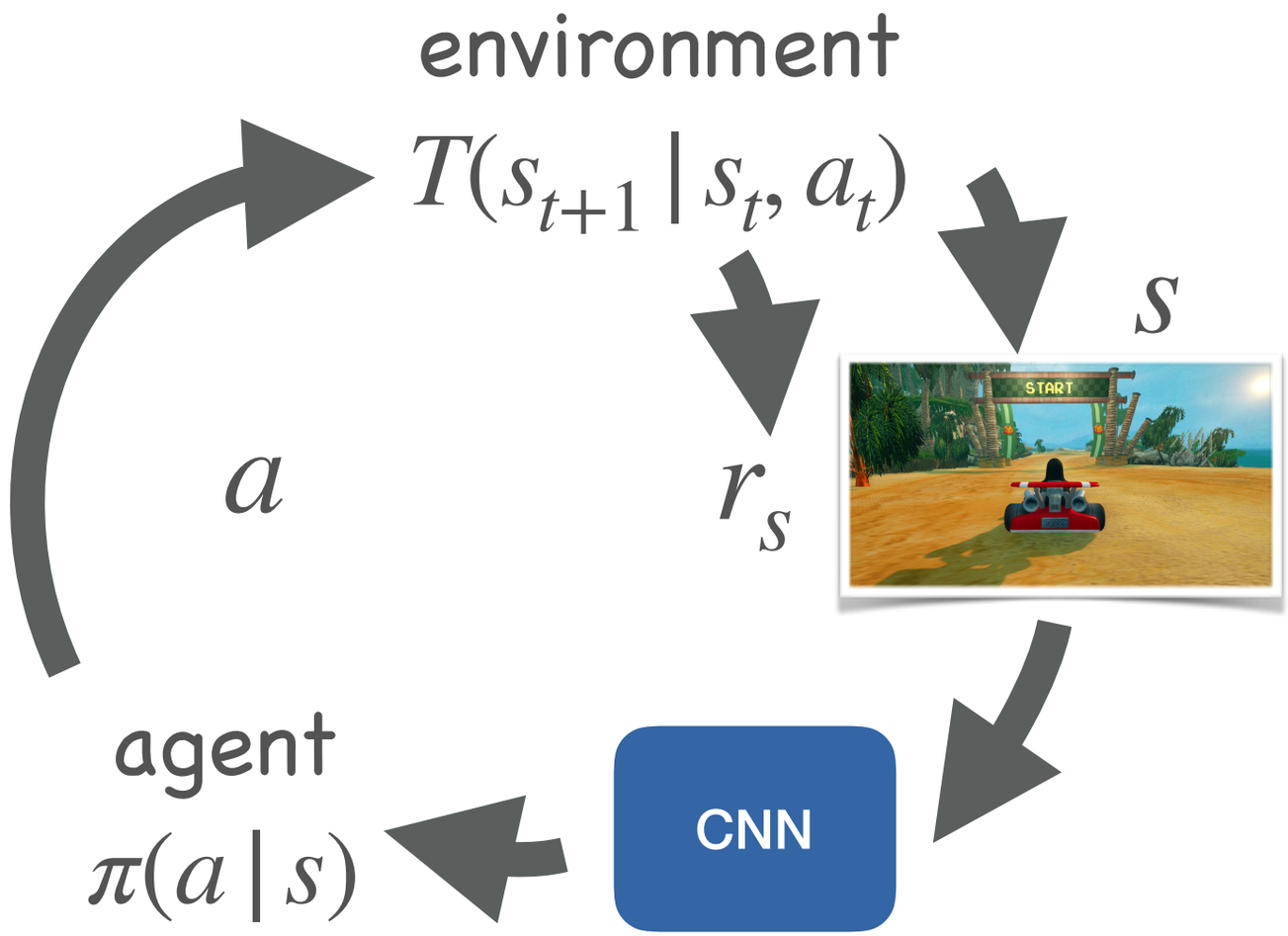
REINFORCE

Non-differentiability

- Compute gradient of

$$\mathbb{E}_{\tau \sim P_{\pi,T}}[R(\tau)]$$

$$= \sum_{\tau} P_{\pi,T}(\tau) R(\tau)$$



The log-derivative trick

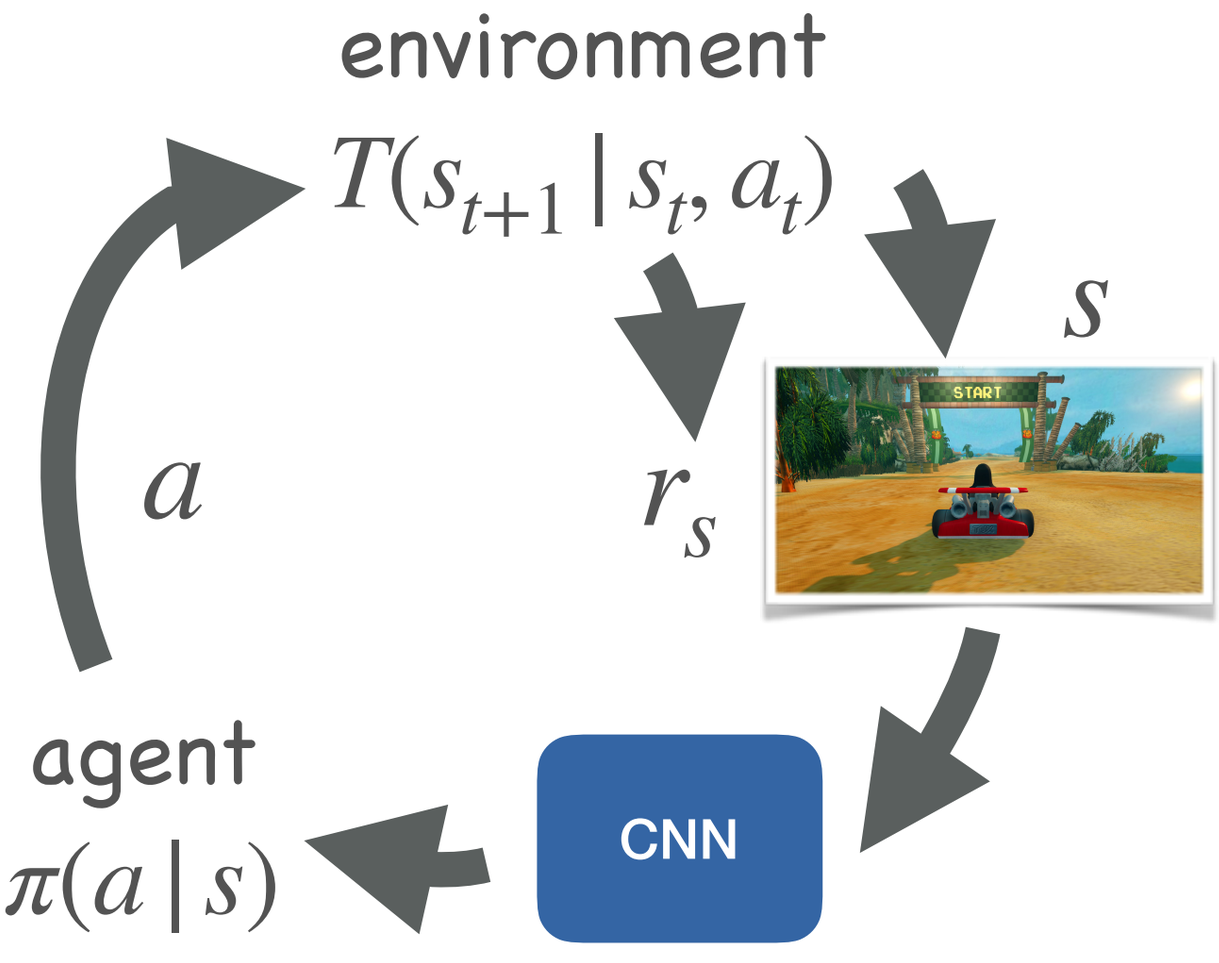
- Simple chain rule

$$\nabla_\theta p_\theta(x) = p_\theta(x) \nabla_\theta \log p_\theta(x)$$

- Gradient of expected return

$$\nabla \mathbb{E}_{\tau \sim P_{\pi,T}}[R(\tau)] = \sum_\tau P_{\pi,T}(\tau) R(a) \nabla \log P_{\pi,T}(\tau)$$

$$= \mathbb{E}_{\tau \sim P_{\pi,T}} \left[R(\tau) \nabla \log P_{\pi,T}(\tau) \right]$$



REINFORCE

- Compute gradient using Monte Carlo sampling

$$\mathbb{E}_{\tau \sim P_{\pi,T}} \left[R(\tau) \nabla \log P_{\pi,T}(\tau) \right]$$

$$\approx \frac{1}{N} \sum_{\tau \sim P_{\pi,T}} \left[R(\tau) \nabla \log P_{\pi,T}(\tau) \right]$$

Simple statistical gradient-following algorithms for connectionist reinforcement learning, Williams, Machine learning 1992

REINFORCE issues

$$\frac{1}{N}\sum_{\tau\sim P_{\pi,T}}\left[R(\tau)\,\nabla\log P_{\pi,T}(\tau)\right]$$

- Needs lots of samples for a good gradient

- High-variance gradient estimator

- Cannot reuse rollouts (τ)

Policy gradient

Policy gradient

$$\frac{1}{N}\sum_{\tau\sim P_{\pi,T}} R(\tau)\,\nabla \log P_{\pi,T}(\tau)$$

- REINFORCE on steroids
  - lower variance
    - baseline
  - off-policy
    - reuse rollouts

Vanilla policy gradient algorithm

$$\frac{1}{N}\sum_{\tau\sim P_{\pi,T}} R(\tau)\,\nabla \log P_{\pi,T}(\tau)$$

- For i iterations
  - Collect rollouts
  - Estimate the sample gradient
  - Take a gradient step

Variance of REINFORCE

$$\frac{1}{N}\sum_{\tau\sim P_{\pi,T}} R(\tau)\,\nabla \log P_{\pi,T}(\tau)$$

- What happens if all rewards are positive?
  - Only learn to do "more" things in τ
  - SGD zig-zags
- RL worst best of we have positive and negative returns

Baselines

- Gradient for constant return is zero

$$\frac{1}{N}\sum_{\tau\sim P_{\pi,T}} R(\tau)\,\nabla \log P_{\pi,T}(\tau)$$

  - $\mathbb{E}_{\tau\sim P_{\pi,T}}[b\,\nabla \log P_{\pi,T}(\tau)] = 0$

- Reduces variance

  - Positive and negative returns

- Unbiased gradient estimate

On- vs off-policy

- REINFORCE is on-policy

$$\frac{1}{N}\sum_{\tau\sim P_{\pi,T}} R(\tau)\,\nabla \log P_{\pi,T}(\tau)$$

  - Trajectories (rollouts) need to come from current policy

  - No reuse of trajectories between gradient update

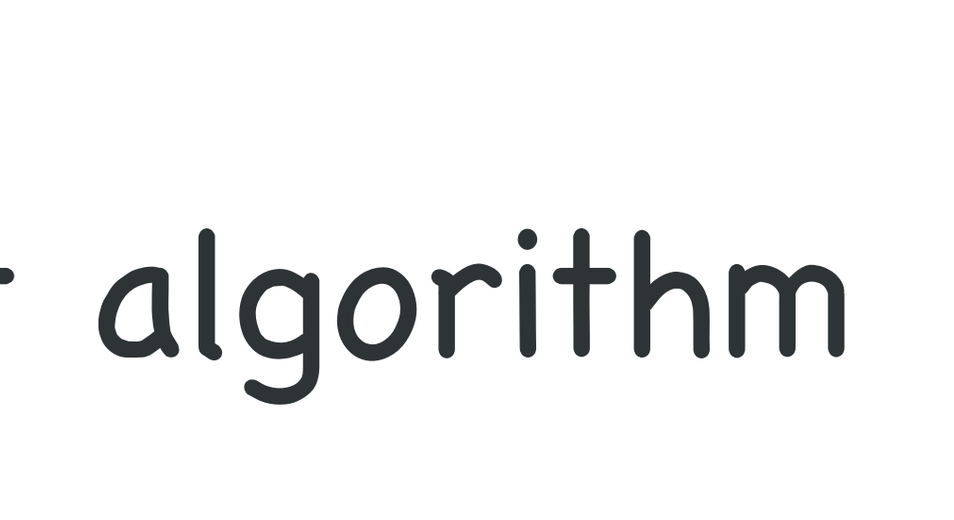
Off-policy

$$\frac{1}{N}\sum_{\tau\sim Q}\frac{P_{\pi,T}(\tau)}{Q(\tau)}R(\tau)\nabla\log P_{\pi,T}(\tau)$$

- Importance sampling
  - Many variants

Policy gradient algorithm

- For i iterations
  - Collect rollouts
    - Add to **replay buffer**
- Update baseline network
- For j batches
  - Estimate the sample gradient on **replay buffer**
  - Take a gradient step



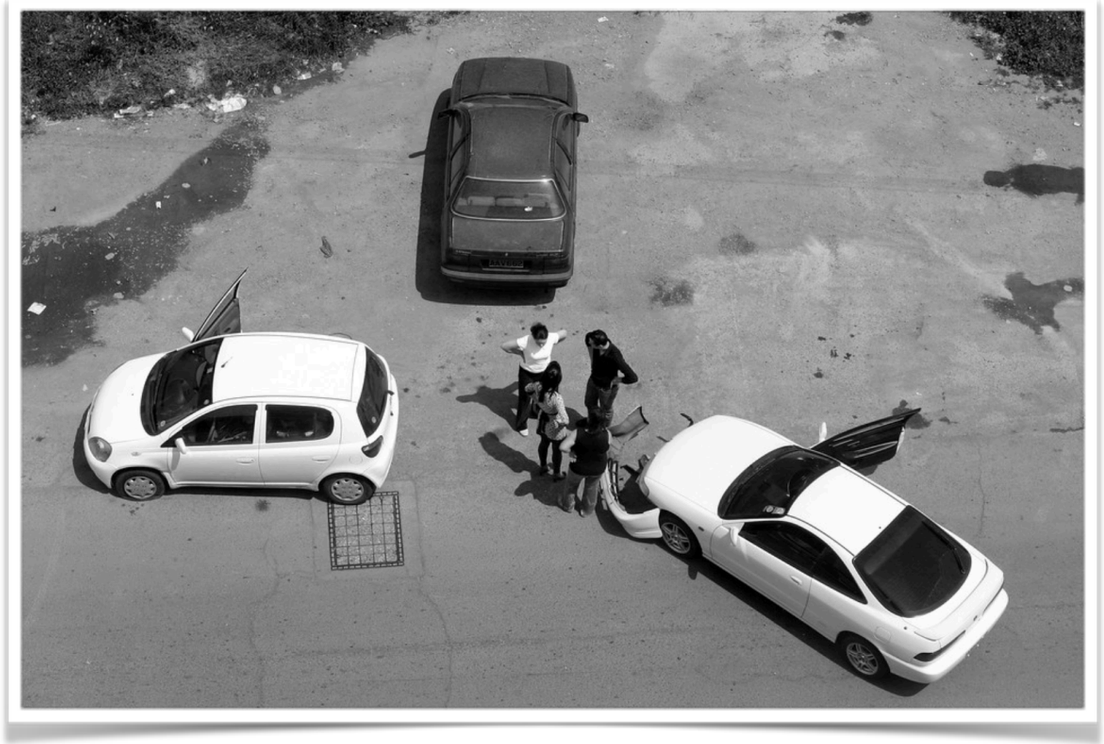
Policy gradient

- REINFORCE with many tricks

$$\frac{1}{N}\sum_{\tau\sim P_{\pi,T}} R(\tau)\,\nabla \log P_{\pi,T}(\tau)$$

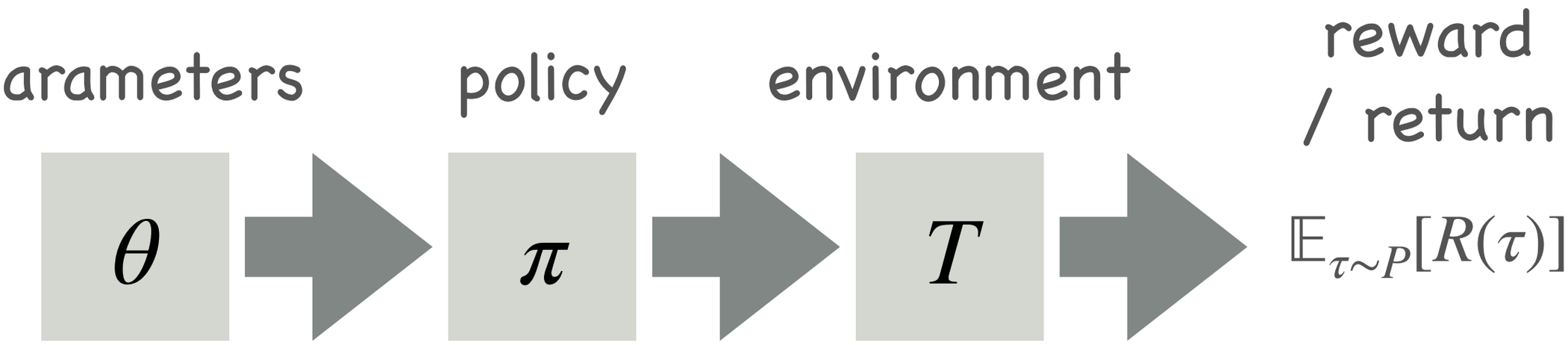
- Not very sample efficient

- Gradient estimate by sampling from an exponential trajectory space



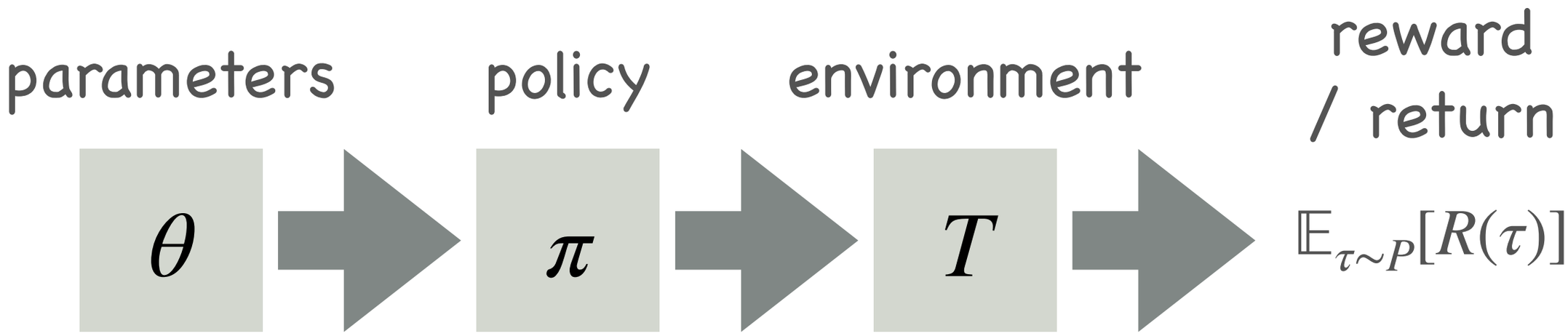
Gradient free optimization

Why do we need a gradient?



getting good gradients is hard!

Why do we need a gradient?



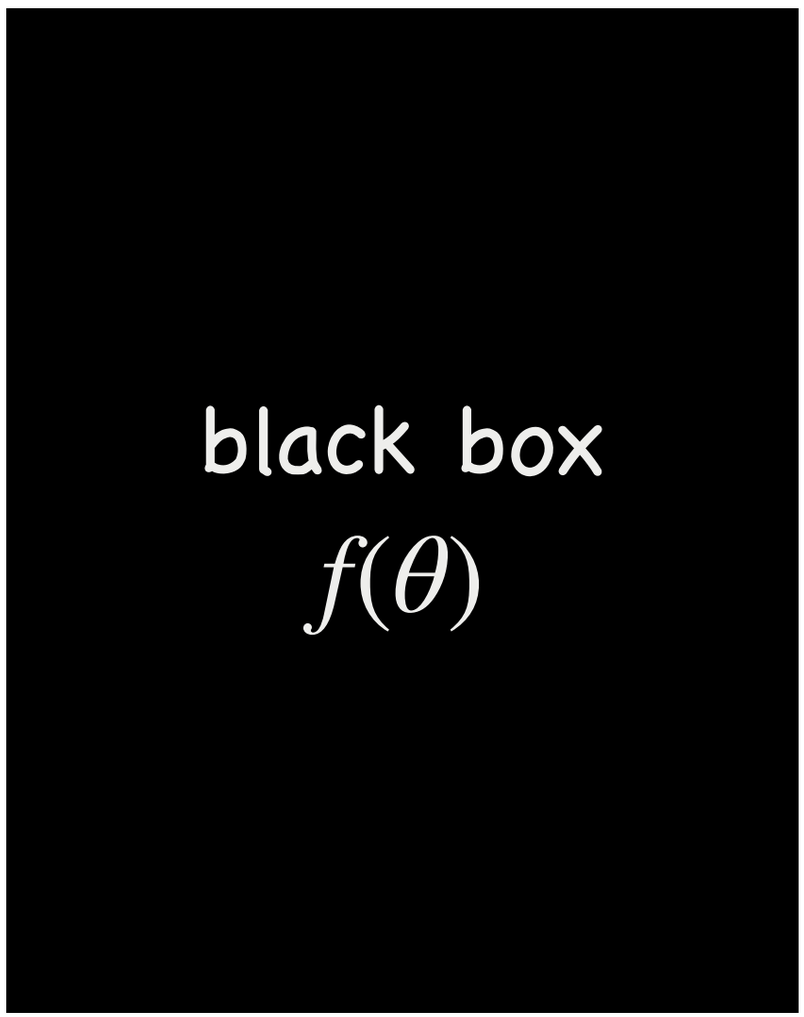
is the forward pass hard?

What if we had an oracle?

- Given policies π_A and π_B
- which one is better?
  - rollout and compute return

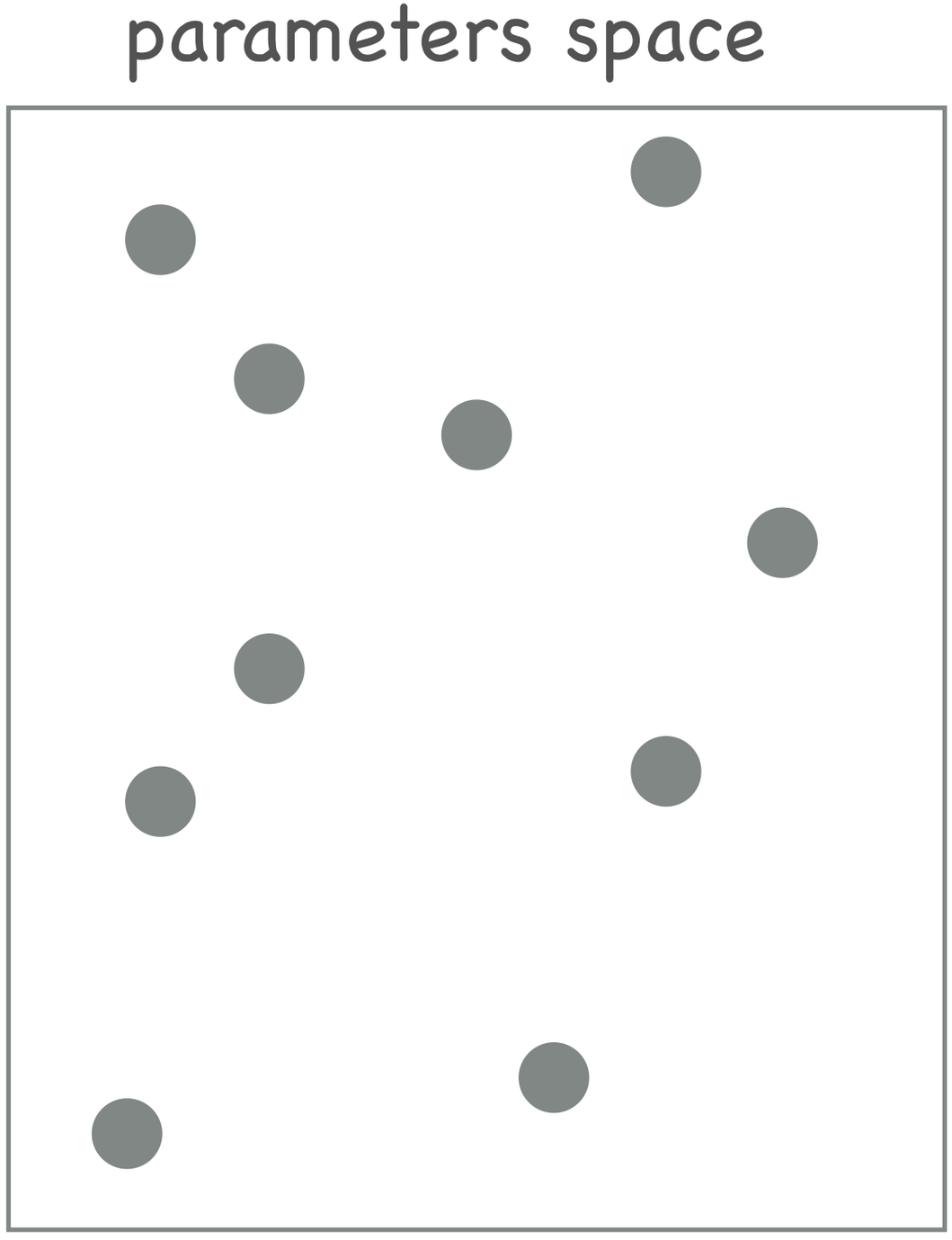
Gradient Free Optimization

- maximize $f(\theta)$ w.r.t. θ
  - can only evaluate function value
- no gradients
- f is smooth
  - similar θ produce similar $f(\theta)$



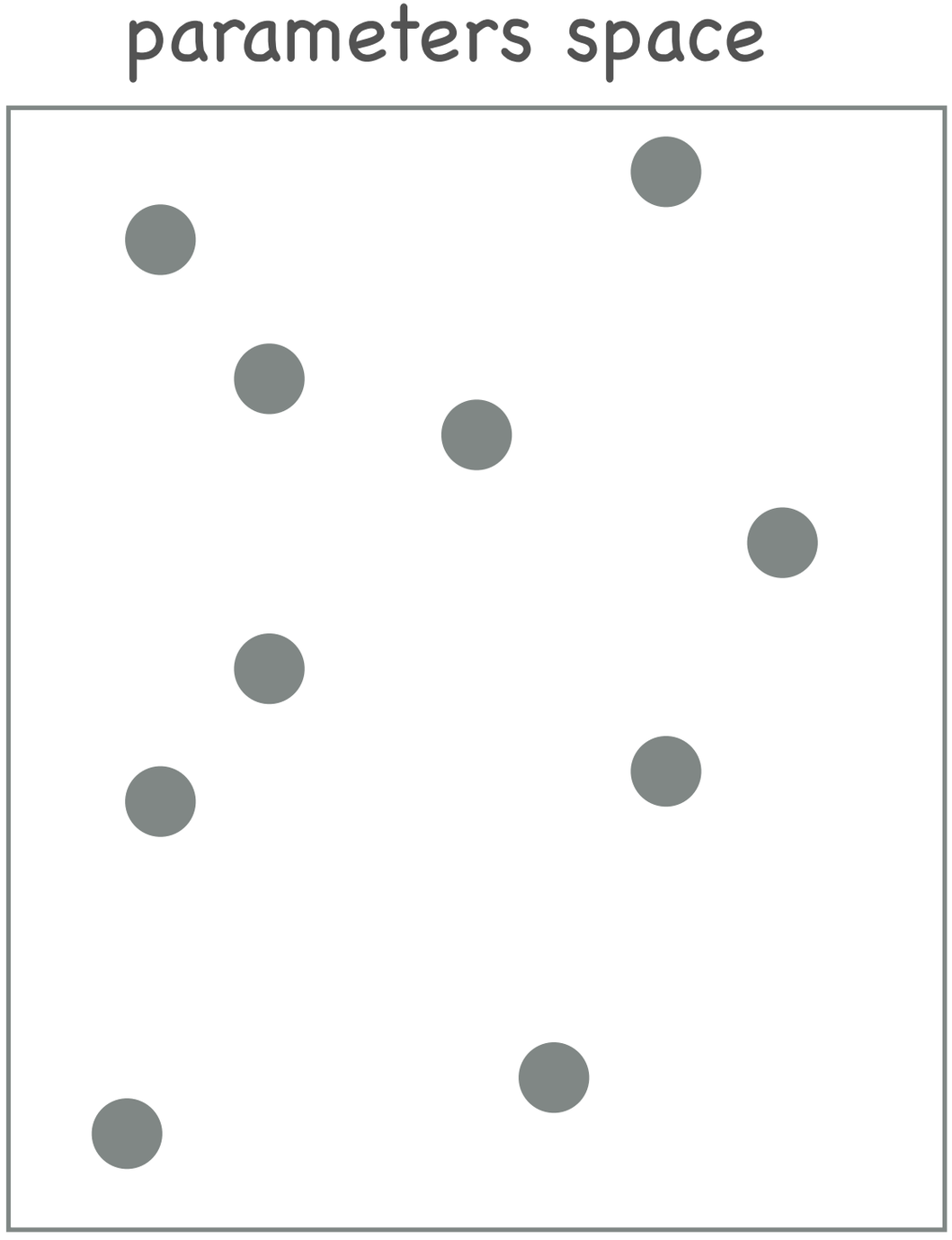
Random Search

- Randomly sample θ
- pick highest $f(\theta)$



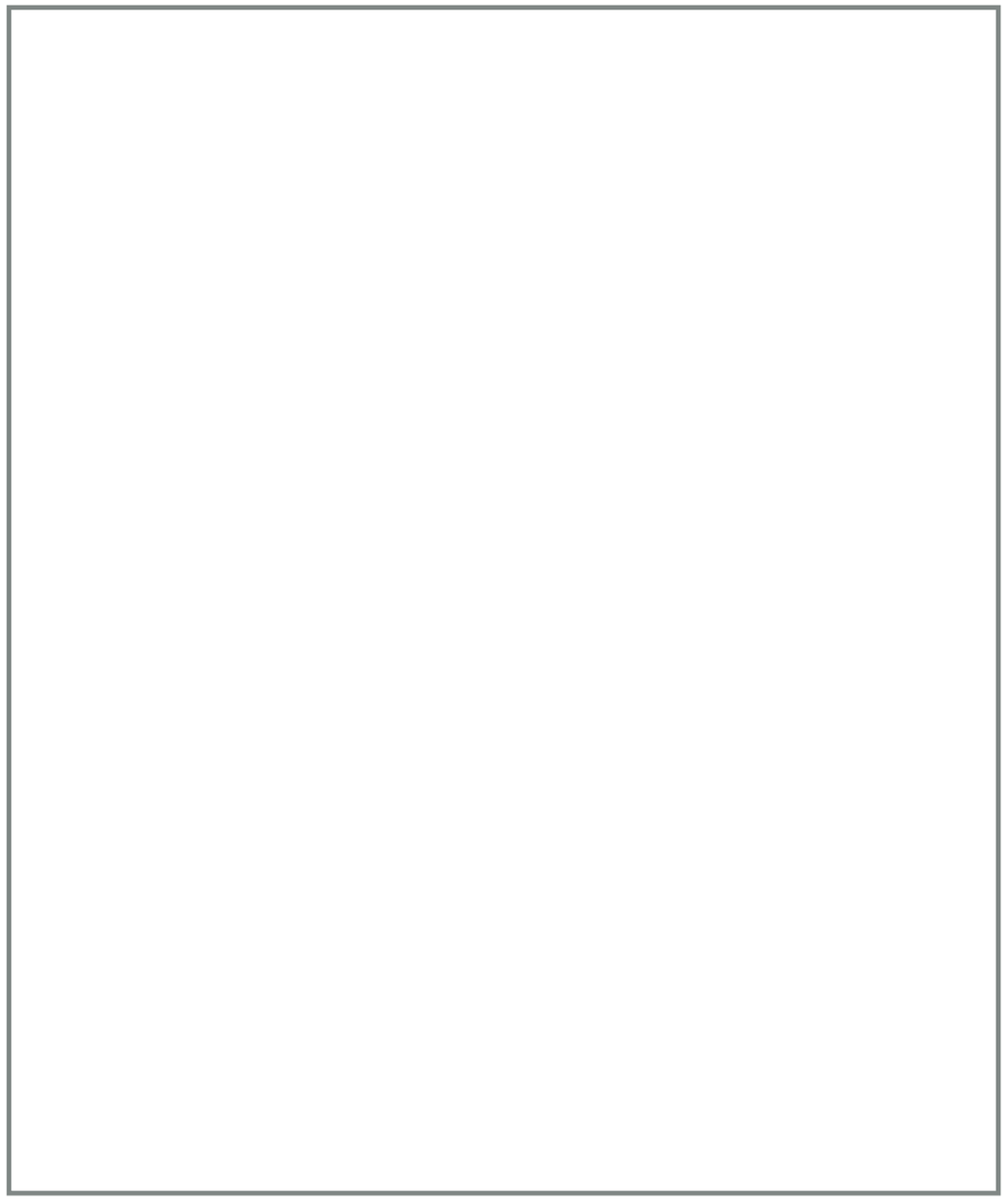
Iterative Random Search

- Randomly sample θ
- pick highest $f(\theta)$
- Sample more points around maxima
- repeat



Cross entropy method

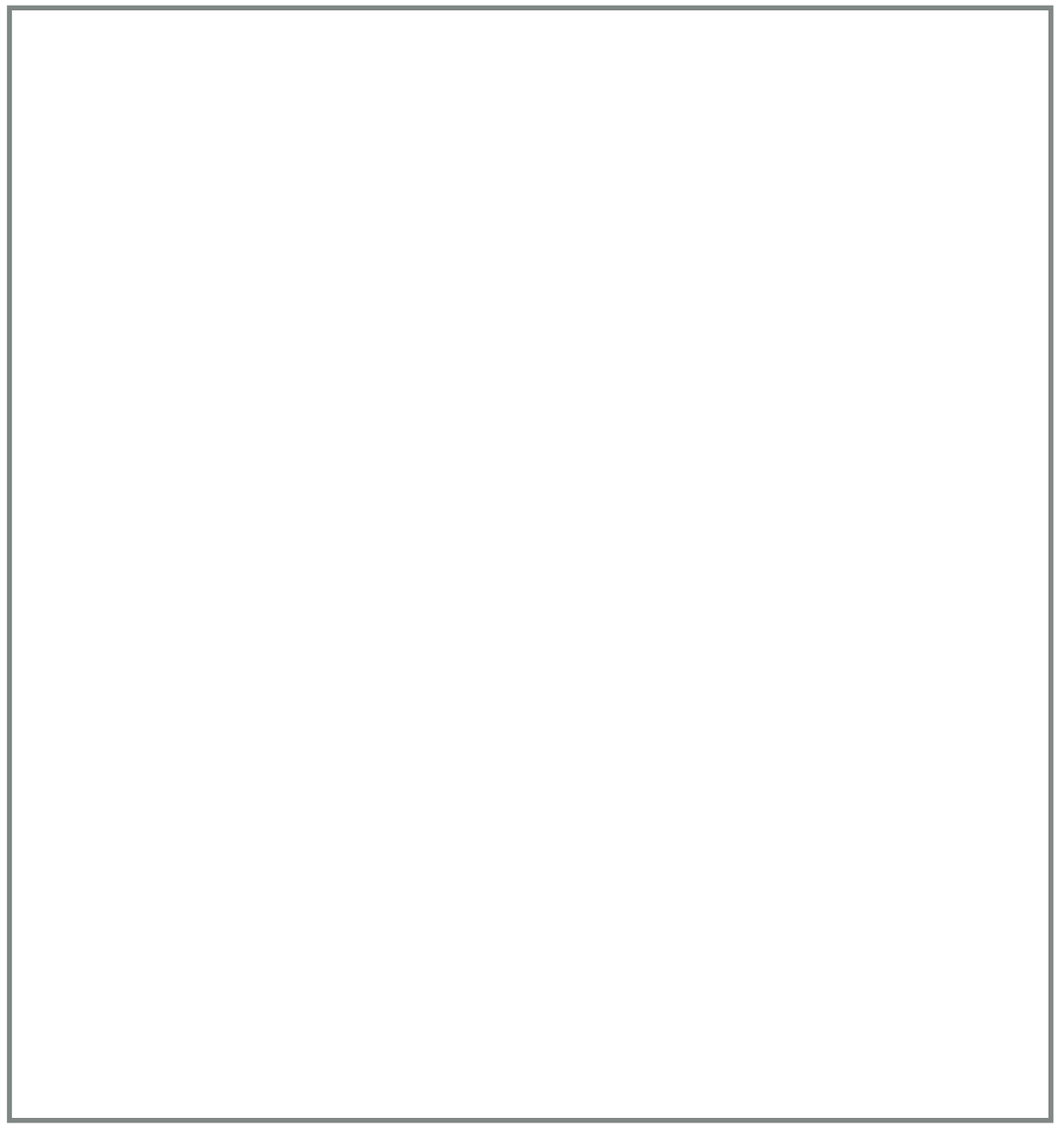
parameters space



- Initialize μ, σ
  - sample $\theta \sim \mathcal{N}(\mu, \sigma^2)$
  - compute reward $f(\theta)$
  - select top $p\,\%$ $(p = 20)$
  - fit Gaussian for new μ, σ
  - repeat

Evolution Strategies

parameters space



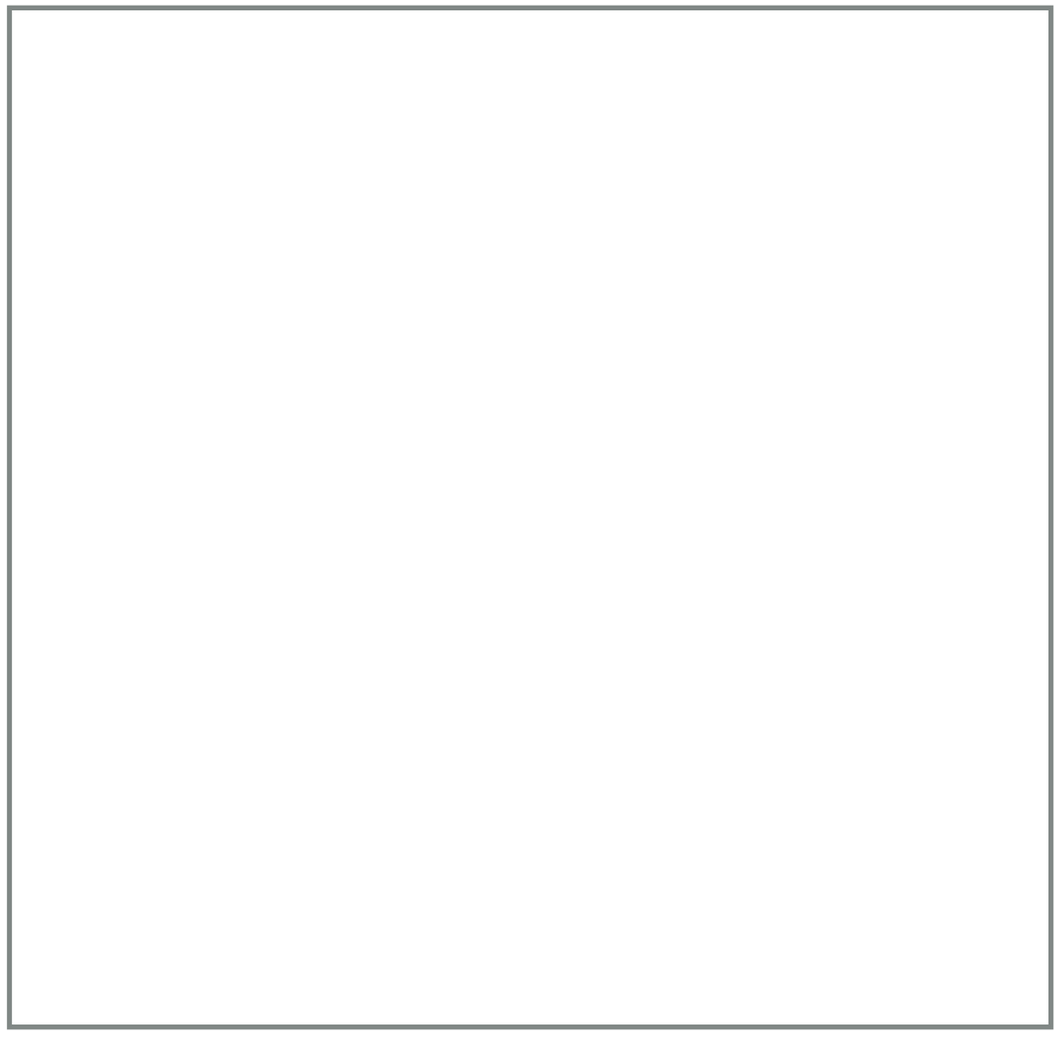
- Initialize θ
- Iterate
  - Sample $\epsilon_0, \epsilon_1, \ldots, \epsilon_n \sim \mathcal{N}(0,I)$
  - Compute returns $F_i = R(\theta + \sigma\epsilon_i)$
  - Normalize $\tilde{F}_i = \dfrac{F_i - \mu_F}{\sigma_F}$
  - Update $w := w + \dfrac{\alpha}{\sigma n} \sum_{i=1}^{n} \tilde{F}_i \epsilon_i$

Evolution strategies as a scalable alternative to reinforcement learning, Salimans et al., arXiv 2017

Augmented random search

- Initialize θ
- Iterate
  - Sample $\epsilon_0, \epsilon_1, \ldots, \epsilon_n \sim \mathcal{N}(0,I)$
  - Compute returns $F_i^+ = R(\theta + \sigma\epsilon_i)$, $F_i^- = R(\theta - \sigma\epsilon_i)$
  - Update

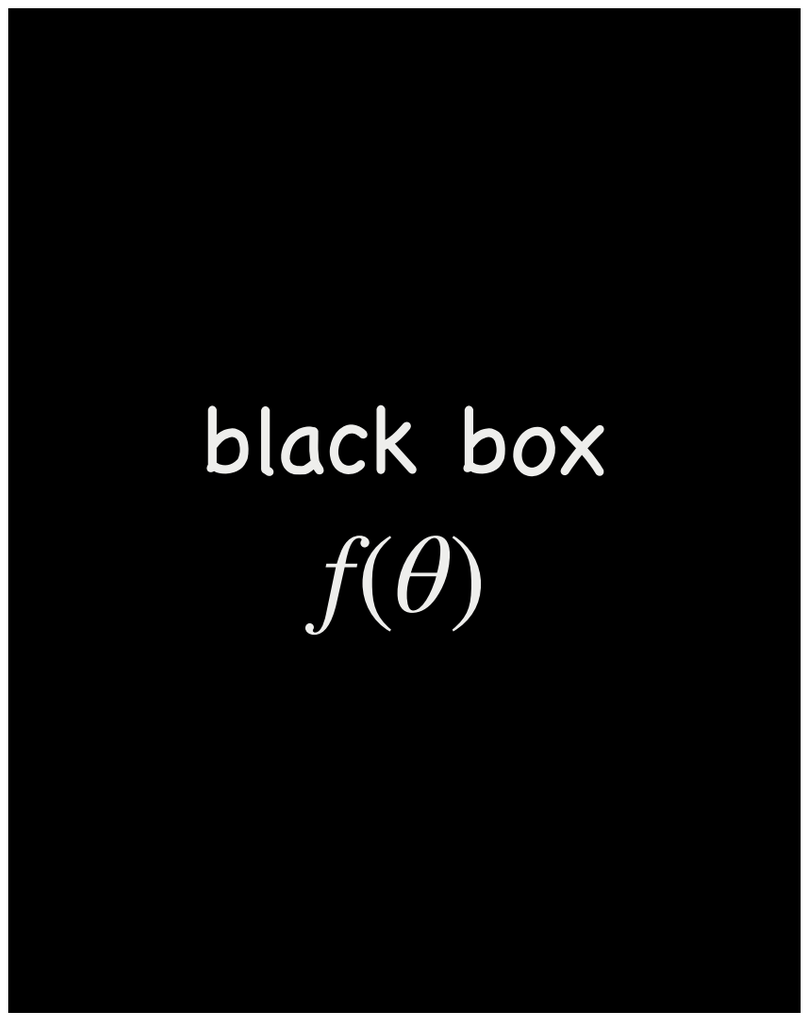
$$w := w + \frac{\alpha}{\sigma n} \sum_{i=1}^{n} (F_i^+ - F_i^-)\epsilon_i$$



- Simple random search provides a competitive approach to reinforcement learning, Mania et al., NIPS 2018
- Multivariate Stochastic Approximation Using a Simultaneous Perturbation Gradient Approximation, Spall, *Automatic Control, 1992*

Gradient free optimization

- Exponential in parameter space
- works better if
  - parameter space small
  - parameters correlate with expected return



Open problem: Structure vs data

Vision and action

- In humans and animals
  - vision developed as a side product for action
  - no explicit supervision for vision
  - emerges from model structure (connections)

Image source: https://en.wikipedia.org/wiki/Cambrian_explosion#/media/File:Opabinia_BW2.jpg

Vision and action

- In computer vision
  - lots of data and labels
    - explicit supervision
    - emerges from data
- Classical robotics
  - Planing after computer vision

example 0: ( , Zebra)

example 1: ( , Zebra)

example 2: ( , Zebra)

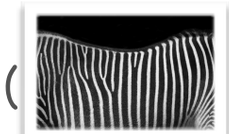
⋮

example 999: ( , Zebra)

Deep Network

Open problem

- Why this disconnect?

example 0: ( , Zebra)

example 1: ( , Zebra)

example 2: ( , Zebra)

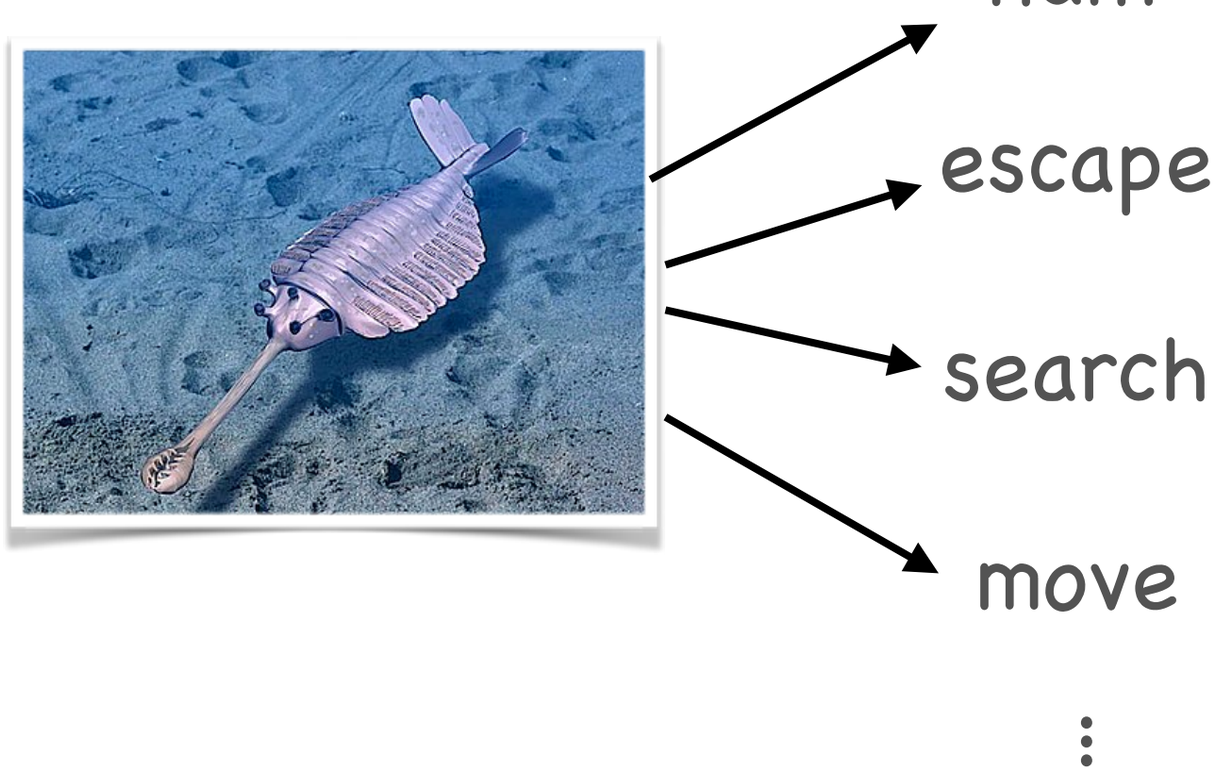
⋮

example 999: ( , Zebra)

Deep Network

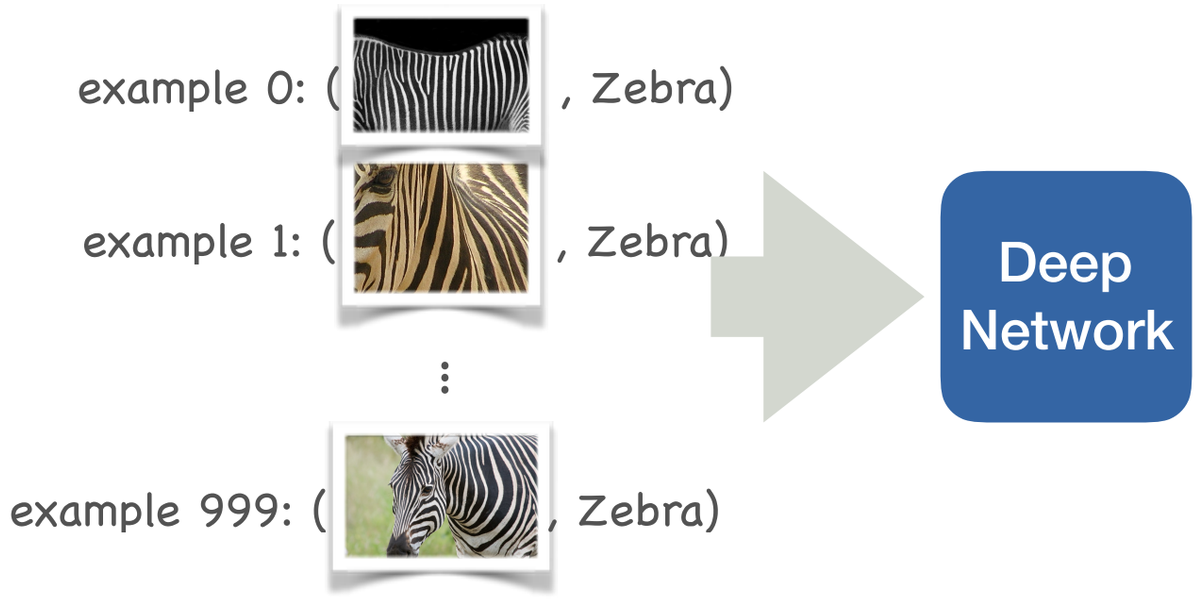
Hypothesis 1 – Too narrow tasks

- On single (narrow) task
  - Data and labels always win
- On multiple tasks
  - Generalization between tasks creates visual representation



Hypothesis 2 - Wrong models and algorithms

- Backprop + SGD biased
- Doesn't work on all tasks and architectures equally well

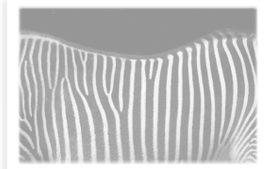


Hypothesis 3 - No evolution

- Insufficient optimization of models in outer loop
  - Meta-learning can find visual representations without much supervision
    - supervision: acting well → survival

Implications

- If any of above hypothesis are right
  - We are wasting time with labeled data

example 0: ( , Zebra)

example 1: ( , Zebra)

example 2: ( , Zebra)

example 3: ( , Zebra)

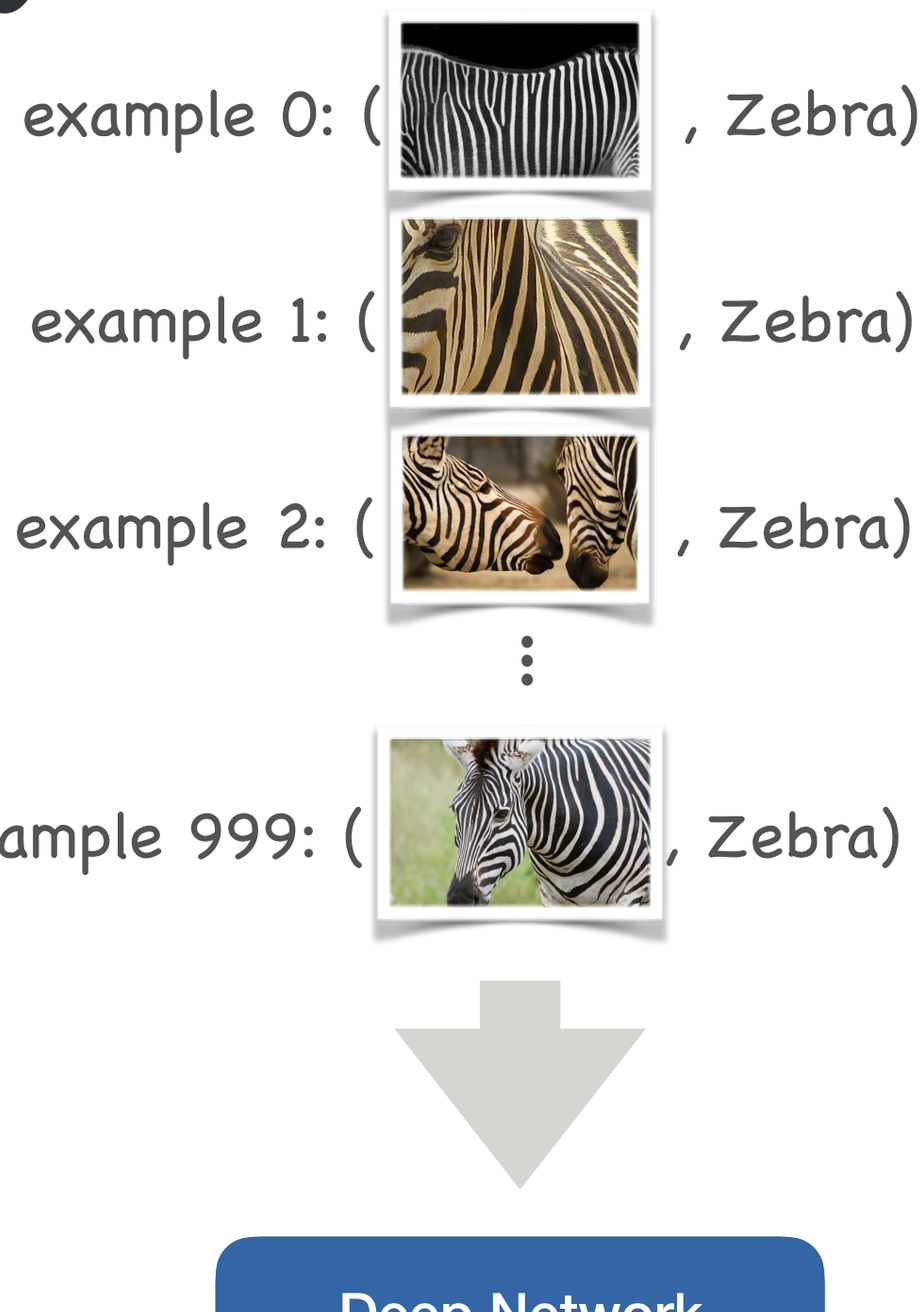
example 4: ( , Zebra)

⋮

example 999: ( , Zebra)

Hypothesis 4 - Value of labels

- Our current approach is fine
- labeled data provides abstract representation without need for evolution and massive optimization



09

January 23, 2024

[1]:
```
%pylab inline
import torch
import sys, os
import pystk
import ray
device = torch.device('cuda') if torch.cuda.is_available() else torch.
 ↪device('cpu')
print('device = ', device)
ray.init(logging_level=50)
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
device =  cuda
```

[1]:
```
RayContext(dashboard_url='', python_version='3.10.13', ray_version='2.9.1',
ray_commit='cfbf98c315cfb2710c56039a3c96477d196de049', protocol_version=None)
```

[2]:
```
@ray.remote
class Rollout:
    def __init__(self, screen_width, screen_height, hd=True,␣
 ↪track='lighthouse', render=True, frame_skip=1):
        # Init supertuxkart
        if not render:
            config = pystk.GraphicsConfig.none()
        elif hd:
            config = pystk.GraphicsConfig.hd()
        else:
            config = pystk.GraphicsConfig.ld()
        config.screen_width = screen_width
        config.screen_height = screen_height
        pystk.init(config)

        self.frame_skip = frame_skip
        self.render = render
        race_config = pystk.RaceConfig(track=track)
        self.race = pystk.Race(race_config)
        self.race.start()
```

```
    def __call__(self, agent, n_steps=200):
        torch.set_num_threads(1)
        self.race.restart()
        self.race.step()
        data = []
        track_info = pystk.Track()
        track_info.update()

        for i in range(n_steps // self.frame_skip):
            world_info = pystk.WorldState()
            world_info.update()

            # Gather world information
            kart_info = world_info.players[0].kart

            agent_data = {'track_info': track_info, 'kart_info': kart_info}
            if self.render:
                agent_data['image'] = np.array(self.race.render_data[0].image)

            # Act
            action = agent(**agent_data)
            agent_data['action'] = action

            # Take a step in the simulation
            for it in range(self.frame_skip):
                self.race.step(action)

            # Save all the relevant data
            data.append(agent_data)
        return data

def show_video(frames, fps=30):
    import imageio
    from IPython.display import Video, display

    imageio.mimwrite('/tmp/test.mp4', frames, fps=fps, bitrate=1000000)
    display(Video('/tmp/test.mp4', width=800, height=600, embed=True))

viz_rollout = Rollout.remote(400, 300)
def show_agent(agent, n_steps=600):
    data = ray.get(viz_rollout.__call__.remote(agent, n_steps=n_steps))
    show_video([d['image'] for d in data])

rollouts = [Rollout.remote(50, 50, hd=False, render=False, frame_skip=5) for i␣
 ↪in range(10)]
def rollout_many(many_agents, **kwargs):
```

```
    ray_data = []
    for i, agent in enumerate(many_agents):
        ray_data.append( rollouts[i % len(rollouts)].__call__.remote(agent,␣
 ↪**kwargs) )
    return ray.get(ray_data)

def dummy_agent(**kwargs):
    action = pystk.Action()
    action.acceleration = 1
    return action
```

[3]:
```
def three_points_on_track(distance, track):
    distance = np.clip(distance, track.path_distance[0,0], track.
 ↪path_distance[-1,1]).astype(np.float32)
    valid_node = (track.path_distance[..., 0] <= distance) & (distance <= track.
 ↪path_distance[..., 1])
    valid_node_idx, = np.where(valid_node)
    node_idx = valid_node_idx[0] # np.random.choice(valid_node_idx)
    d = track.path_distance[node_idx].astype(np.float32)
    x = track.path_nodes[node_idx][:,[0,2]].astype(np.float32) # Ignore the y␣
 ↪coordinate
    w, = track.path_width[node_idx].astype(np.float32)

    t = (distance - d[0]) / (d[1] - d[0])
    mid = x[1] * t + x[0] * (1 - t)
    x10 = (x[1] - x[0]) / np.linalg.norm(x[1]-x[0])
    x10_ortho = np.array([-x10[1],x10[0]], dtype=float32)
    return mid - w / 2 * x10_ortho, mid, mid + w / 2 * x10_ortho


def state_features(track_info, kart_info, absolute=False, **kwargs):
    f = np.concatenate([three_points_on_track(kart_info.distance_down_track +␣
 ↪d, track_info) for d in [0,5,10,15,20]])
    if absolute:
        return f
    p = np.array(kart_info.location)[[0,2]].astype(np.float32)
    t = np.array(kart_info.front)[[0,2]].astype(np.float32)
    f = f - p[None]
    d = (p-t) / np.linalg.norm(p-t)
    d_o = np.array([-d[1], d[0]], dtype=float32)
    return np.stack([f.dot(d), f.dot(d_o)], axis=1)

# Let's load a fancy auto-pilot. You'll write one yourself in your homework.
from _auto_pilot import auto_pilot
data, = rollout_many([auto_pilot], n_steps=400)

figure()
```

```
f = state_features(**data[50])
plot(f[:,1].flat, f[:,0].flat, '*')
axis('equal')
gca().invert_yaxis()

figure()
for d in data:
    f = state_features(**d, absolute=True)
    plot(f[:,1].flat, f[:,0].flat, '*')
axis('equal')
gca().invert_yaxis()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[3], line 29
     26     return np.stack([f.dot(d), f.dot(d_o)], axis=1)
     28 # Let's load a fancy auto-pilot. You'll write one yourself in your␣
 ↪homework.
---> 29 from _auto_pilot import auto_pilot
     30 data, = rollout_many([auto_pilot], n_steps=400)
     32 figure()

ModuleNotFoundError: No module named '_auto_pilot'
```

[4]:
```
def new_action_net():
    return torch.nn.Linear(2*5*3, 1, bias=False)

class Actor:
    def __init__(self, action_net):
        self.action_net = action_net.cpu().eval()

    def __call__(self, track_info, kart_info, **kwargs):
        f = state_features(track_info, kart_info)
        output = self.action_net(torch.as_tensor(f).view(1,-1))[0]

        action = pystk.Action()
        action.acceleration = 1
        action.steer = output[0]
        return action
```

[5]:
```
action_net = new_action_net()
show_agent(Actor(action_net))
```

```
IMAGEIO FFMPEG_WRITER WARNING: input image is not divisible by
macro_block_size=16, resizing from (400, 300) to (400, 304) to ensure video
compatibility with most codecs and players. To prevent resizing, make your input
image divisible by the macro_block_size or set the macro_block_size to 1
```

```
(risking incompatibility).
```

```
<IPython.core.display.Video object>
```

[6]:
```
many_action_nets = [new_action_net() for i in range(10)]

data = rollout_many([Actor(action_net) for action_net in many_action_nets],␣
 ↪n_steps=600)

good_initialization = many_action_nets[ np.argmax([d[-1]['kart_info'].
 ↪overall_distance for d in data]) ]
```

[7]:
```
show_agent(Actor(good_initialization))
```

```
IMAGEIO FFMPEG_WRITER WARNING: input image is not divisible by
macro_block_size=16, resizing from (400, 300) to (400, 304) to ensure video
compatibility with most codecs and players. To prevent resizing, make your input
image divisible by the macro_block_size or set the macro_block_size to 1
(risking incompatibility).
```

```
<IPython.core.display.Video object>
```

[8]:
```
import copy

n_epochs = 20
n_step = 20

action_net = copy.deepcopy(good_initialization)
best_action_net = copy.deepcopy(good_initialization)
best_dist = 0

for epoch in range(n_epochs):
    eps = 1e-2

    w = 1*action_net.weight.data

    networks = []
    ray_data = []
    for i in range(n_step):
        dp = torch.randn(w.shape) * eps

        # Try positive
        action_net_dp = copy.deepcopy(action_net)
        action_net_dp.weight.data[:] += dp

        networks.append(action_net_dp)
    data = rollout_many([Actor(network) for network in networks], n_steps=600)
    distances = [d[-1]['kart_info'].overall_distance for d in data]
```

```
    print(np.max(distances))
    action_net = networks[np.argmax(distances)]
    if np.max(distances) > best_dist:
        best_dist = np.max(distances)
        best_action_net = action_net
```

```
584.630615234375
785.595458984375
804.114013671875
988.268798828125
982.7676391601562
1018.0629272460938
1017.465576171875
1021.9312133789062
1033.591064453125
1016.3307495117188
1039.4539794921875
1018.5470581054688
1050.0584716796875
1026.889404296875
1048.7305908203125
1060.83740234375
1045.785400390625
1063.168212890625
1004.17138671875
1028.7884521484375
```

[9]:
```
show_agent(Actor(action_net))
```

```
IMAGEIO FFMPEG_WRITER WARNING: input image is not divisible by
macro_block_size=16, resizing from (400, 300) to (400, 304) to ensure video
compatibility with most codecs and players. To prevent resizing, make your input
image divisible by the macro_block_size or set the macro_block_size to 1
(risking incompatibility).
```

```
<IPython.core.display.Video object>
```

[ ]:
```
```

Summary

Summary

- Learning to act using deep networks
  - Option 1: Imitate expert
  - Option 2: Policy gradient
  - Option 3: Gradient free

Which one should I chose?

- Is it easy for human to perform the tasks?
  - Imitation
- Do we want to do better than humans?
  - Gradient free
- Can I use existing code?
  - Policy gradient