

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \lambda \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

(The New Action Value = The Old Value) + The Learning Rate \times (The New Information — the Old Information)



advanced RL

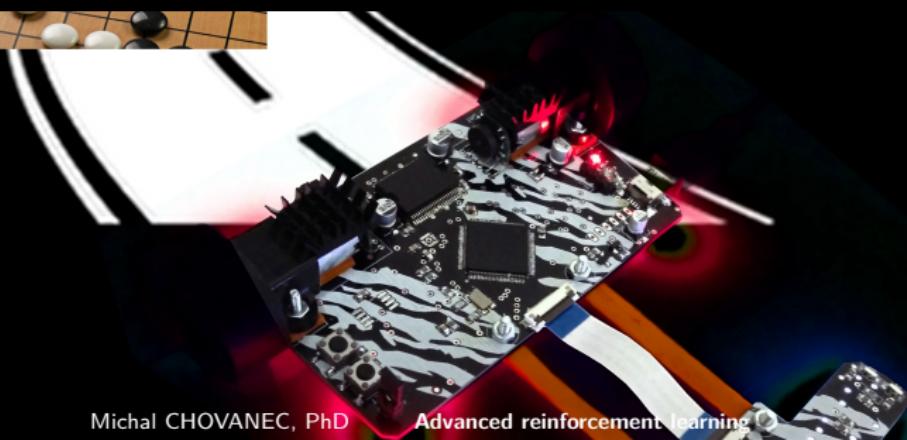
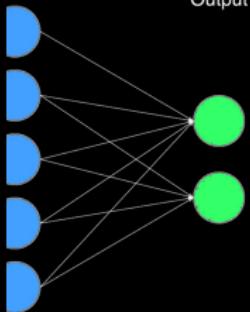
- continuous control
- curiosity
- imagination

Michal CHOVANEC

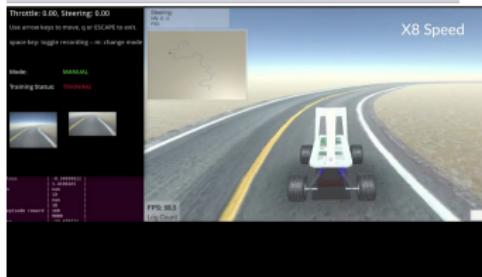
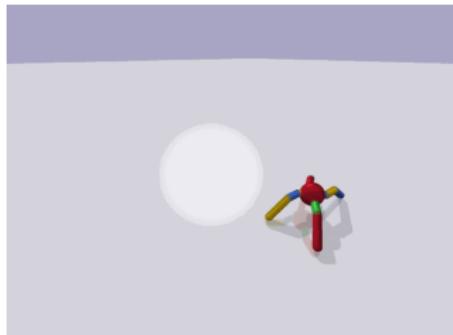


Hidden

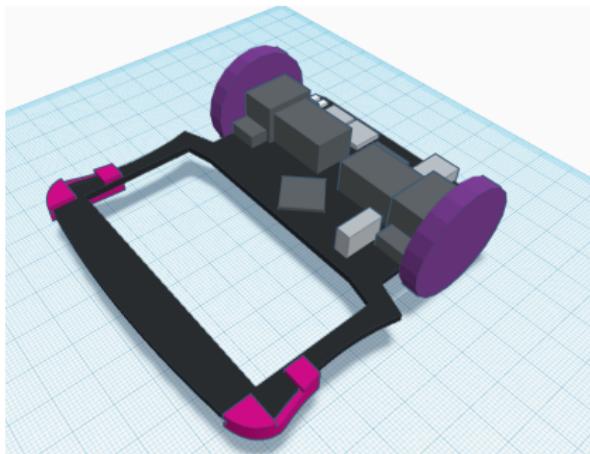
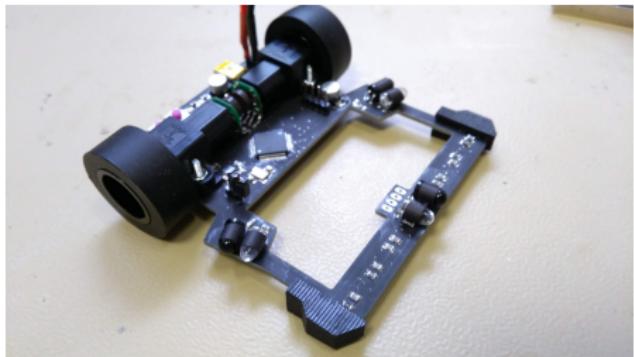
Output



continuous actions space



continuous actions space

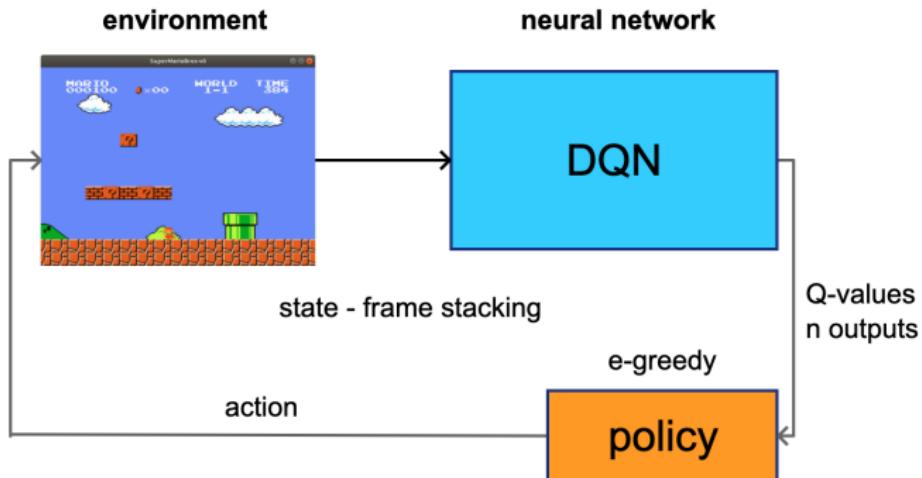


algorithms

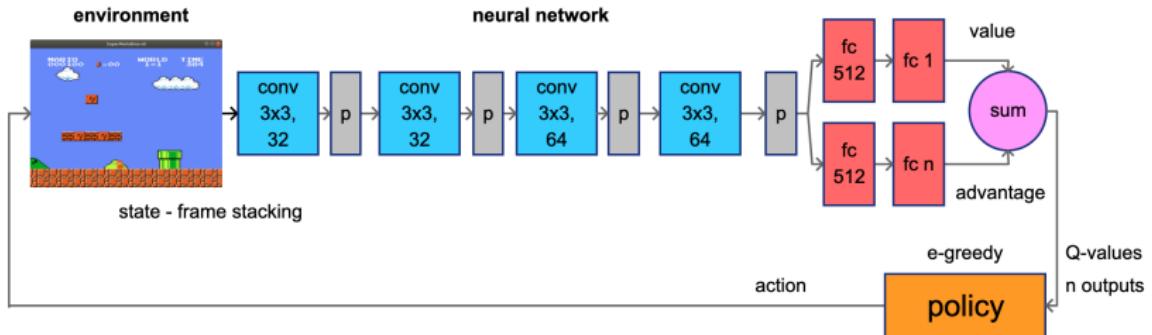
- discrete actions space
 - Deep Q-network, DQN
 - Dueling DQN
 - Rainbow DQN
- continuous actions space
 - Actor Critic
 - Advantage Actor Critic
 - Proximal policy optimization
 - Soft Actor critic
 - Deep deterministic policy gradient
 - D4PG, SDDPG
- model based
 - curiosity
 - world models
 - imagination agents

f.e. SDDPG - sampled DDPG, based on Wasserstein loss : Optimal transport, Cédric Villani, 600+ pages

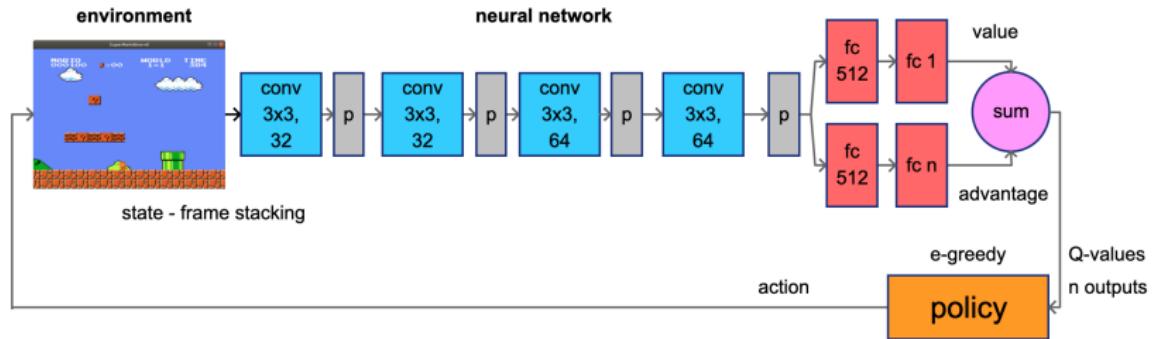
from DQN to DDPG



from DQN to DDPG

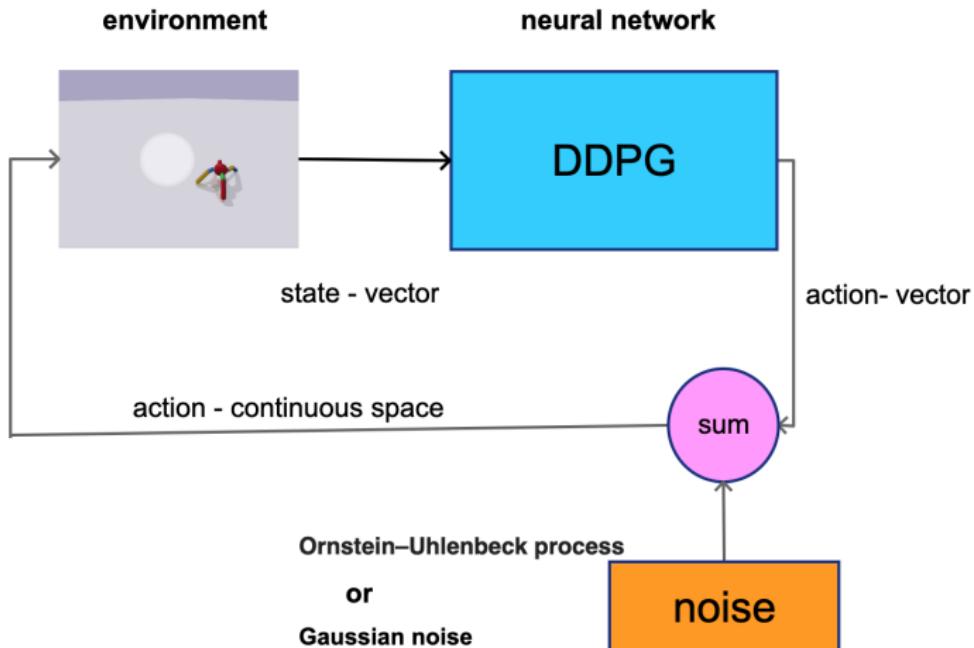


from DQN to DDPG

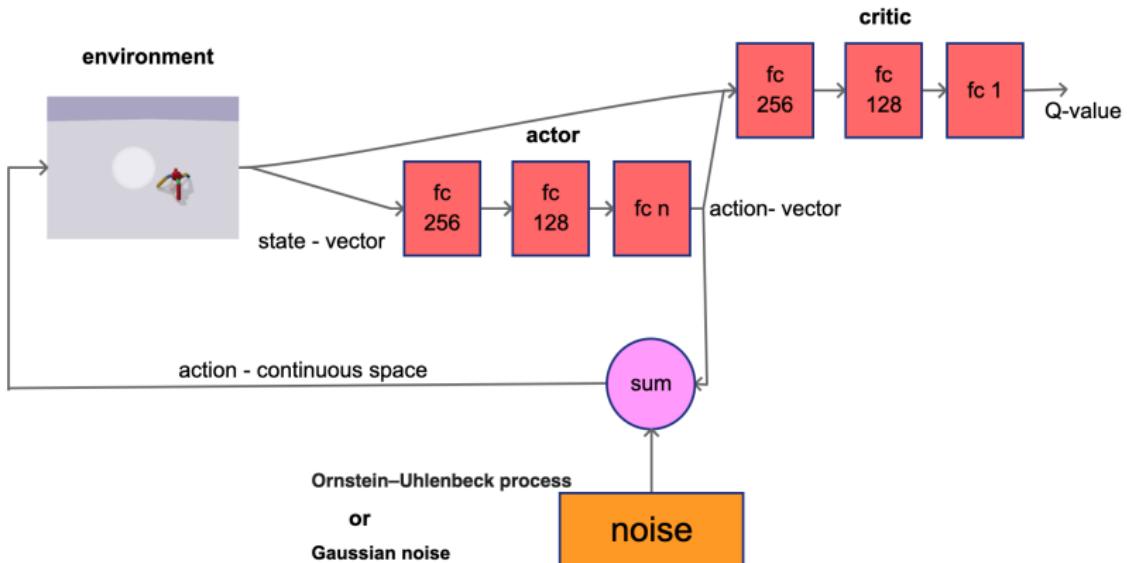


$$\mathcal{L}(\theta) = \left(R + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2$$

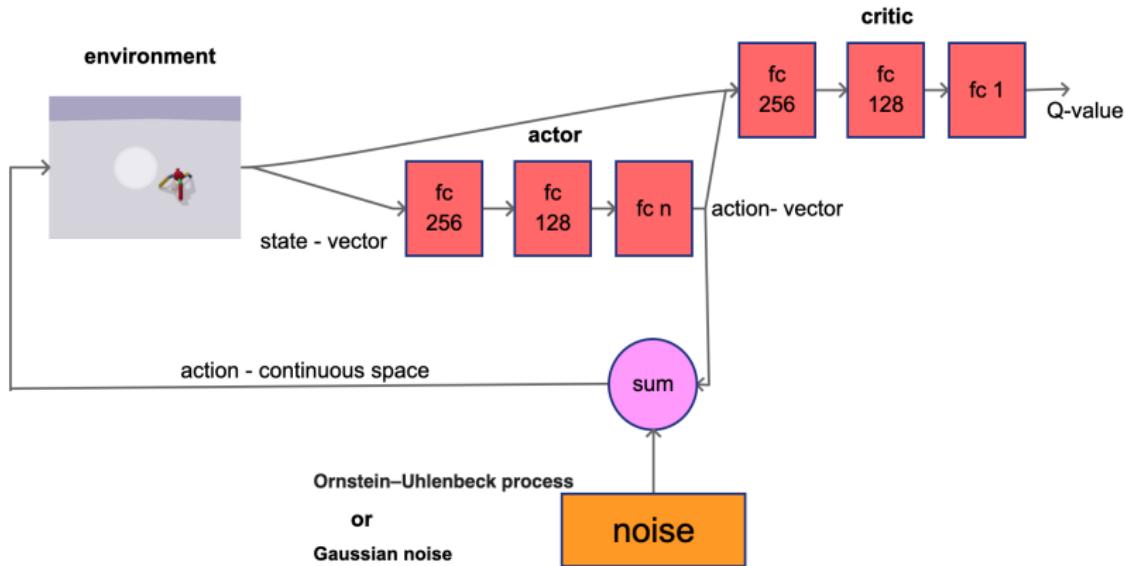
from DQN to DDPG



from DQN to DDPG



from DQN to DDPG



critic loss:

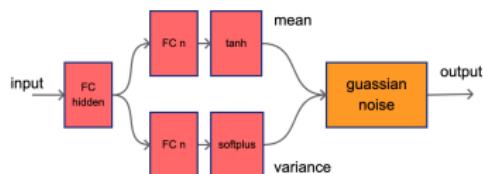
$$\mathcal{L}_c(\theta) = (R + \gamma Q(s', a'; \theta^-, \phi^-) - Q(s, a; \theta, \phi))^2$$

actor loss :

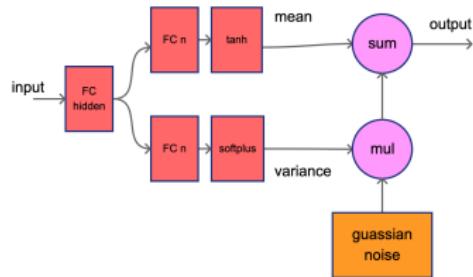
$$\mathcal{L}_a(\phi) = -Q(s, a; \theta, \phi)$$

reparametrization trick

non-differentiable



differentiable



$$\frac{\partial y}{\partial f_{noise}(x)} = \frac{\partial y}{\partial f_{noise}} \frac{\partial f_{noise}}{\partial x}$$

wise Wizard's DDPG spell chart

- **neurons count** on 1st layer = 10x state vector size
- **neurons count** on 2nd layer = 0.5x neurons on 1st layer
- **weight init** for hidden layers : use Xavier
- **weight init** actor output : use uniform $\langle -0.3, 0.3 \rangle$
- **weight init** critic output : use uniform $\langle -0.003, 0.003 \rangle$
- **gaussian noise** : linear decay variance, from 1 to 0.3, for 1M steps
- use **soft** target network update, $\tau = 0.001$
- actor learning rate $\eta_a = 0.0001$
- critic learning rate $\eta_c = 0.0002$

DDPG critic

```
class Model(torch.nn.Module):
    def __init__(self, input_shape, outputs_count, hidden_count = 256):
        super(Model, self).__init__()

        self.device = "cpu"

        self.layers = [
            nn.Linear(input_shape[0] + outputs_count, hidden_count),
            nn.ReLU(),
            nn.Linear(hidden_count, hidden_count//2),
            nn.ReLU(),
            nn.Linear(hidden_count//2, 1)
        ]

        torch.nn.init.xavier_uniform_(self.layers[0].weight)
        torch.nn.init.xavier_uniform_(self.layers[2].weight)
        torch.nn.init.uniform_(self.layers[4].weight, -0.003, 0.003)

        self.model = nn.Sequential(*self.layers)
        self.model.to(self.device)

        print(self.model)

    def forward(self, state, action):
        x = torch.cat([state, action], dim = 1)
        return self.model(x)
```

DDPG actor

```
class Model(torch.nn.Module):
    def __init__(self, input_shape, outputs_count, hidden_count = 256):
        super(Model, self).__init__()
        self.device = "cpu"

        self.layers = [
            nn.Linear(input_shape[0], hidden_count),
            nn.ReLU(),
            nn.Linear(hidden_count, hidden_count//2),
            nn.ReLU(),
            nn.Linear(hidden_count//2, outputs_count),
            nn.Tanh()
        ]

        torch.nn.init.xavier_uniform_(self.layers[0].weight)
        torch.nn.init.xavier_uniform_(self.layers[2].weight)
        torch.nn.init.uniform_(self.layers[4].weight, -0.3, 0.3)

        self.model = nn.Sequential(*self.layers)
        self.model.to(self.device)

        print(self.model)

    def forward(self, state):
        return self.model(state)
```

wise Wizard's magic staff

- fully connected nets (robotic envs) **train on CPU** - AMD Ryzen
- convolutional nets (visual inputs envs) **train on GPU**
 - NVIDIA GTX1080+
- use fast CPU - envs are slow
- 32GB of RAM is enough
- for small visual envs (Atari, DOOM, Nec) - GTX1080ti



model based RL



- curiosity
- world models
- imagination

curiosity in RL

Curiosity-driven Exploration by Self-supervised Prediction

Deepak Pathak¹ Pulkit Agrawal¹ Alexei A. Efros¹ Trevor Darrell¹

- Curiosity-driven Exploration by Self-supervised Prediction, 2017, Deepak Pathak
- old idea : adaptive control (dynamic system identification, adaptive filtering)
- **main idea** : curiosity = squared model prediction error

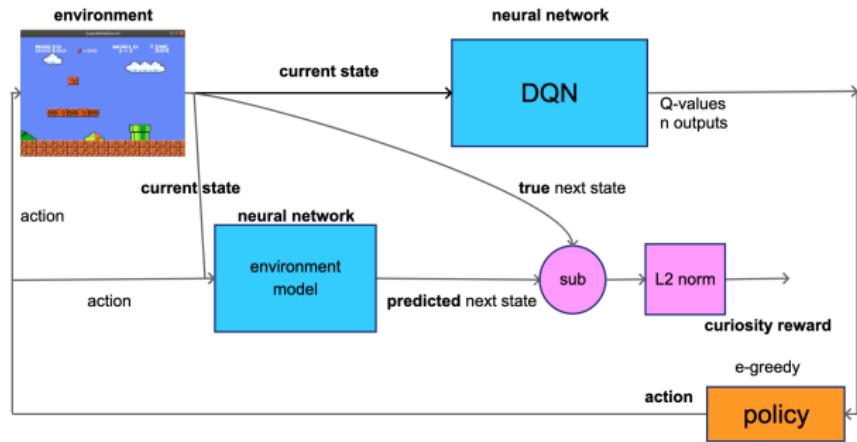


(a) learn to explore in Level-1



(b) explore faster in Level-2

curiosity in RL

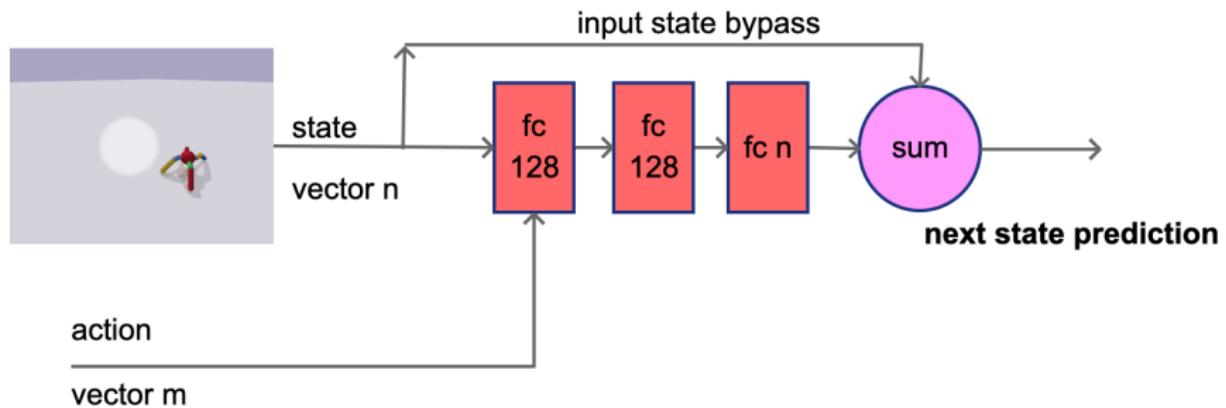


$$Q'(s, a; \theta) = R + \beta C(s, s', a; \phi) + \gamma \max_{a'} Q(s', a'; \theta^-)$$

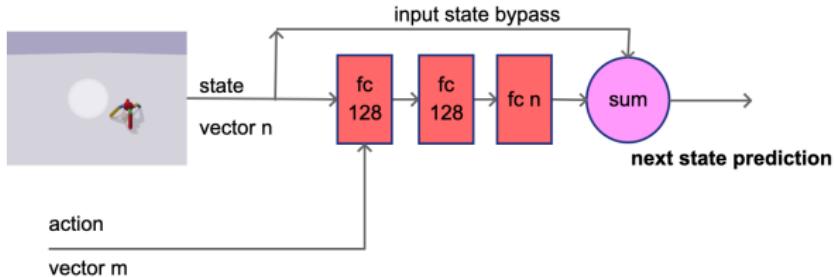
$$\mathcal{L}_{em}(\phi) = (s' - EM(s, a; \phi))^2$$

$$C(s, s', a) = \|s' - EM(s, a; \phi)\|_2^2$$

environment model - vector input



environment model - vector input



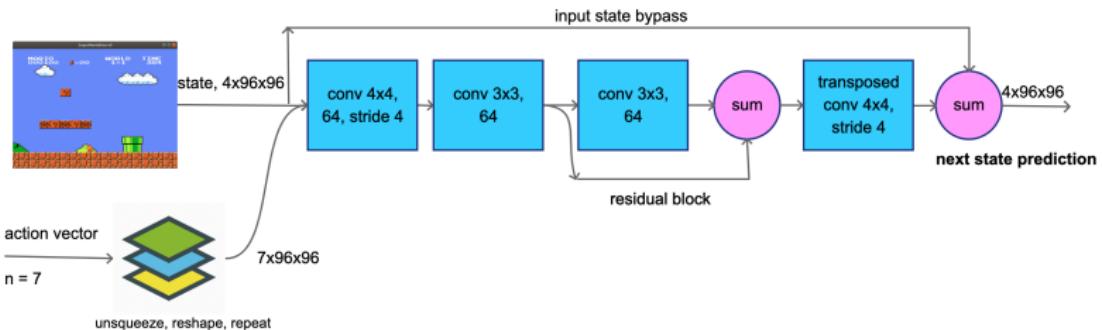
```
def forward(self, state, action):
    x = torch.cat([state, action], dim = 1)

    features = self.model_features(x)

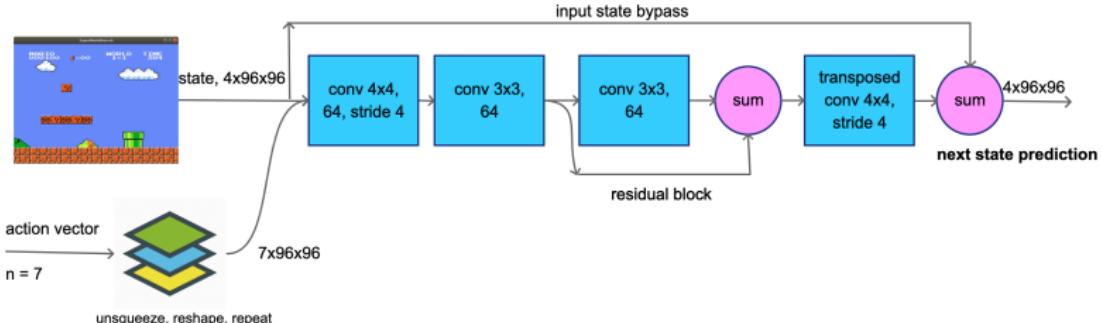
    state_prediction = self.model_state(features) + state.detach()

    return state_prediction, self.model_reward(features)
```

environment model - visual input



environment model - visual input



```
def forward(self, state, action):
    action_ = action.unsqueeze(1).unsqueeze(1).transpose(3, 1).repeat((1, 1, self.input_shape[1], self.input_shape[2]))

    model_input      = torch.cat([state, action_], dim = 1)
    conv0_output     = self.conv0(model_input)
    conv1_output     = self.conv1(conv0_output)

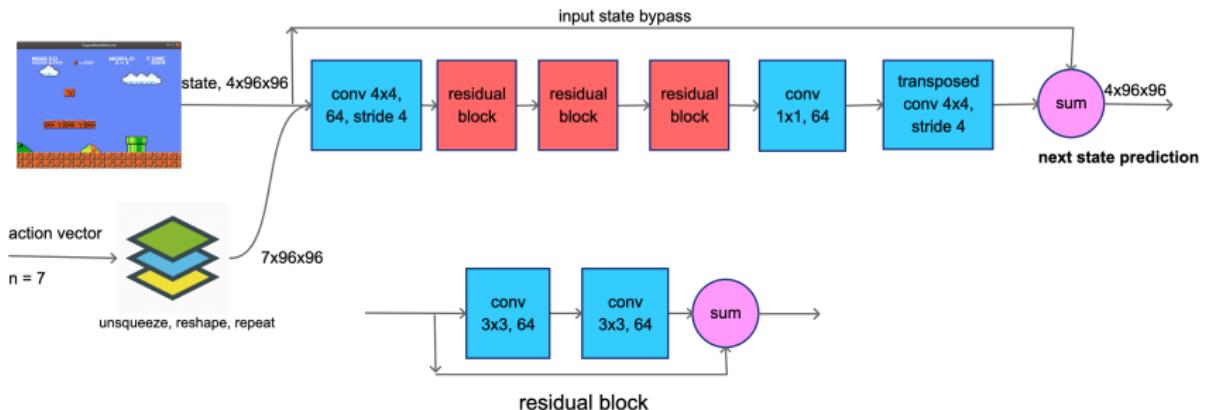
    tmp = conv0_output + conv1_output

    frame_prediction      = self.deconv0(tmp)
    reward_prediction     = self.reward(tmp)

    frames_count          = state.shape[1]
    state_tmp              = torch.narrow(state, 1, 0, frames_count-1)
    frame_prediction      = frame_prediction + torch.narrow(state, 1, 0, 1)
    observation_prediction = torch.cat([frame_prediction, state_tmp], dim = 1)

    return observation_prediction, reward_prediction
```

environment model - visual input



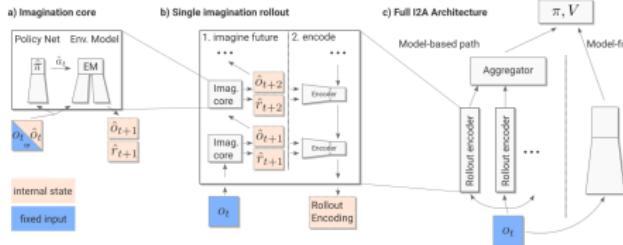
imagination augmented agents

Imagination-Augmented Agents for Deep Reinforcement Learning

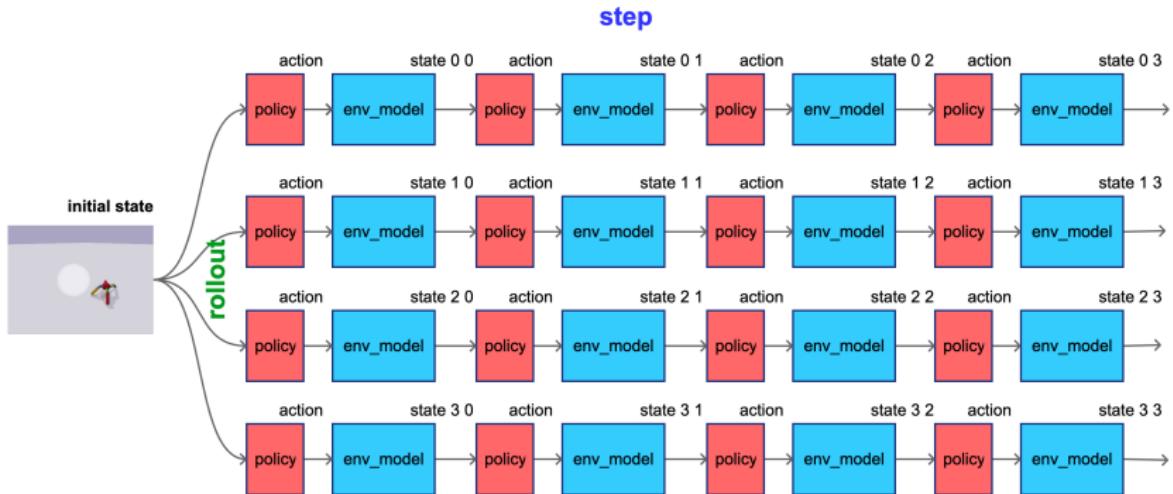
Théophane Weber* Sébastien Racanière* David P. Reichert* Lars Buesing
Arthur Guez Danilo Rezende Adria Puigdomènech Badia Oriol Vinyals
Nicolas Heess Yujia Li Razvan Pascanu Peter Battaglia
Demis Hassabis David Silver Daan Wierstra
DeepMind

- imagination-Augmented Agents for Deep Reinforcement Learning, 2018, Theeophane Weber
- **main idea** : exploration and action selection in imagined trajectories first

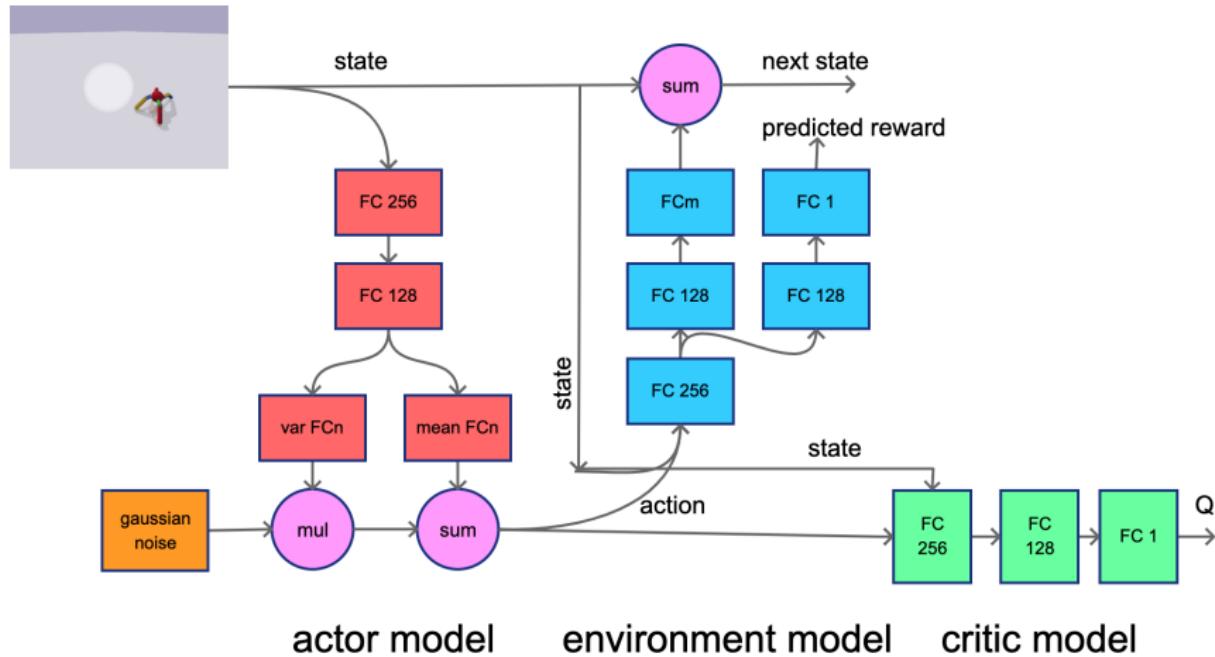
2 The I2A architecture



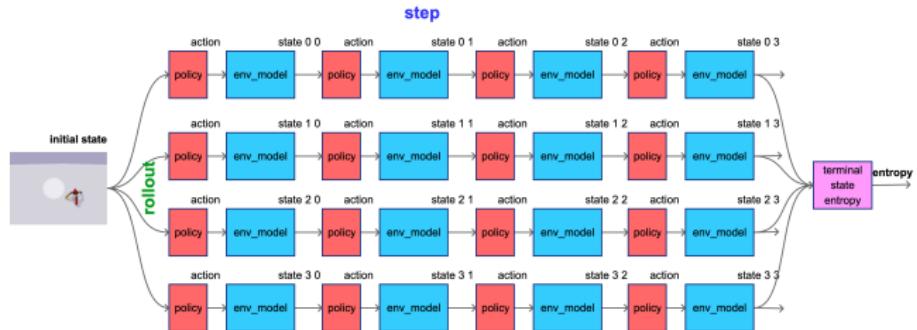
imagination in RL



imagination in RL - model detail with meta-actor



imagination in RL - model detail with meta-actor

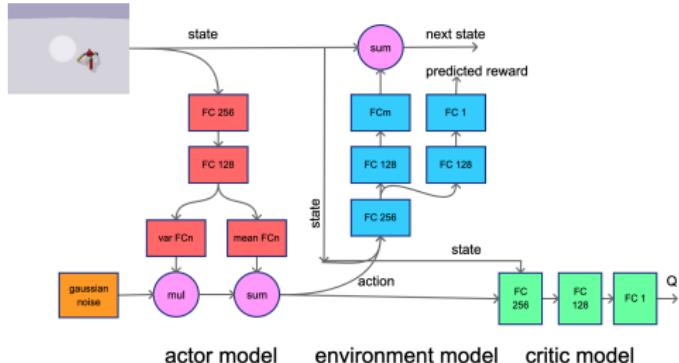


$$\mathcal{L}_{\text{entropy}}(\phi) = -\frac{1}{2} \ln(2\pi e \sigma(\cdot; \phi)^2)$$

note : maximum entropy defined as

$$H(x) = \int_{-\infty}^{\infty} x \ln(x) dx$$

imagination in RL - model detail with meta-actor



$$\mathcal{L}_c(\theta) = (R + \gamma Q(s', a'; \theta^-, \phi^-) - Q(s, a; \theta, \phi))^2$$

$$\mathcal{L}_a(\phi) = -Q(s, a; \theta, \phi) - \beta H(\{s_0^{im}..s_k^{im}\}; \phi)$$

$$\mathcal{L}_m(\kappa) = (s' - EM(s, a; \kappa))^2$$

books to read

- Maxim Lapan, 2020, Deep Reinforcement Learning Hands-On second edition
- Maxim Lapan, 2018, Deep Reinforcement Learning Hands-On
- Praveen Palanisamy, 2018, Hands-On Intelligent Agents with OpenAI Gym
- Andrea Lonza, 2019, Reinforcement Learning Algorithms with Python
- Rajalingappa Shanmugamani, 2019, Python Reinforcement Learning
- Micheal Lanham, 2019, Hands-On Deep Learning for Games

Q&A



michal.nand@gmail.com

https://github.com/michalnand/imagination_reinforcement_learning