Goal-Conditioned Generators of Deep Policies

Francesco Faccio (francesco@idsia.ch)
Vincent Herrmann, Aditya Ramesh, Louis Kirsch, and Jürgen Schmidhuber

AAAI-23

Overview of the talk

Learn a single model to evaluate many policies:

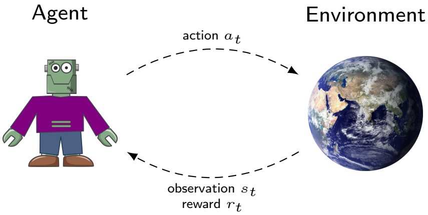
- Faccio, Kirsch, & Schmidhuber (2020). **Parameter-based value functions**. ICLR 2021
- Harb, Schaul, Precup, & Bacon, (2020). **Policy evaluation networks**.
- Faccio, Ramesh, Herrmann, Harb, & Schmidhuber (2022). **General Policy Evaluation and Improvement by Learning to Identify Few But Crucial States**.

Learn a single model to generate many policies:

- Faccio*, Herrmann*, Ramesh, Kirsch, & Schmidhuber (2022). **Goal-Conditioned Generators of Deep Policies**.

The RL Framework

- **Markov Decision Process** (Puterman, 2014; Stratonovich, 1960)
  - $\mathcal{S}$ set of states: $s \in \mathcal{S}$
  - $\mathcal{A}$ set of actions: $a \in \mathcal{A}$
  - $\mathcal{P}(s'|s,a)$ markovian transition matrix
  - $R(s,a)$ reward function
  - γ discount factor
  - μ_0 distribution on initial state



Policy $\pi_\theta : \mathcal{S} \rightarrow \Delta(\mathcal{A})$ inducing a stationary distribution over states $d^{\pi_\theta}(s)$ in the MDP

Value Functions

Traditional Value Functions

(Sutton and Barto, 1998)

- Value functions estimate the return $R_t = \sum_{k=0}^{T-t-1} \gamma^k R(s_{t+k+1}, a_{t+k+1})$ of a policy:
  - State-value function

    $V^{\pi_{\boldsymbol{\theta}}}(s) := \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[R_t | s_t = s]$
  - Action-value function

    $Q^{\pi_{\boldsymbol{\theta}}}(s, a) := \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[R_t | s_t = s, a_t = a]$
- State and action value functions are related by:

$$V^{\pi_{\boldsymbol{\theta}}}(s) = \begin{cases} \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) Q^{\pi_{\boldsymbol{\theta}}}(s, a) \, \mathrm{d}a & \text{if } \pi_{\boldsymbol{\theta}} \text{ is stochastic,} \\ Q^{\pi_{\boldsymbol{\theta}}}(s, \pi_{\boldsymbol{\theta}}(s)) & \text{if } \pi_{\boldsymbol{\theta}} \text{ is deterministic.} \end{cases}$$

PBVFs

Parameter-based Value Functions

(Faccio et al., 2021)

- Parameter-based State-Value Function (**PSVF**)

  $V(s, \boldsymbol{\theta}) := \mathbb{E}[R_t | s_t = s, \boldsymbol{\theta}]$

- Parameter-based Action-Value Function (**PAVF**)

  $Q(s, a, \boldsymbol{\theta}) := \mathbb{E}[R_t | s_t = s, a_t = a, \boldsymbol{\theta}]$

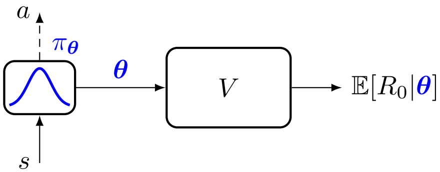
- Parameter-based Start-State-Value Function (**PSSVF**)

  $V(\boldsymbol{\theta}) := \mathbb{E}_{s \sim \mu_0(s)}[V(s, \boldsymbol{\theta})]$

Parameter-based Start-State Value Function

- Stochastic or deterministic policies
- Find the policy $\pi_{\boldsymbol{\theta}}$ maximizing $J(\boldsymbol{\theta})$:

$$J(\boldsymbol{\theta}) = \mathbb{E}[R_0|\boldsymbol{\theta}] = V(\boldsymbol{\theta})$$



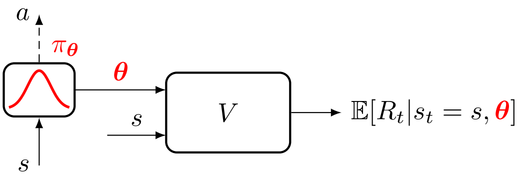
- Taking the gradient of $J(\boldsymbol{\theta})$ we obtain:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} V(\boldsymbol{\theta})$$

Parameter-based State-Value Function

- Stochastic or deterministic policies
- Find the policy $\pi_{\boldsymbol{\theta}}$ maximizing $J(\boldsymbol{\theta})$:

$$J(\boldsymbol{\theta}) = \int_{\mathcal{S}} d^{\pi_b}(s) V(s, \boldsymbol{\theta}) \,\mathrm{d}s$$



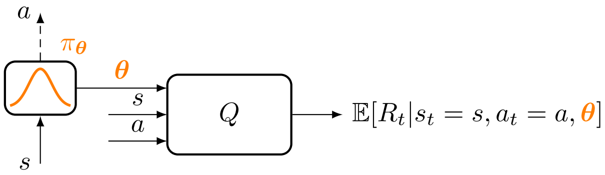
- Taking the gradient of $J(\boldsymbol{\theta})$ we obtain:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{s \sim d^{\pi_b}(s)}[\nabla_{\boldsymbol{\theta}} V(s, \boldsymbol{\theta})]$$

Parameter-based Action-Value Function

- Stochastic policies
- Find the policy π_{θ} maximizing $J(\theta)$:

$$J(\theta) = \int_{\mathcal{S}} d^{\pi_b}(s) \int_{\mathcal{A}} \pi_{\theta}(a|s) Q(s, a, \theta) \, \mathrm{d}a \, \mathrm{d}s$$



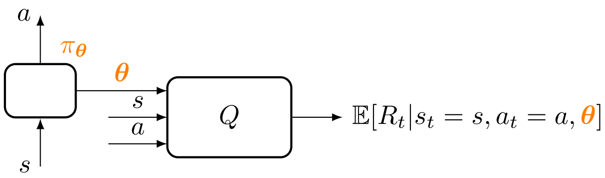
- Taking the gradient of $J(\theta)$ we obtain:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{s \sim d^{\pi_b}(s), a \sim \pi_b(.|s)} \left[\frac{\pi_{\theta}(a|s)}{\pi_b(a|s)} \left(Q(s, a, \theta) \nabla_{\theta} \log \pi_{\theta}(a|s) + \nabla_{\theta} Q(s, a, \theta) \right) \right]$$

Parameter-based Action-Value Function

- Deterministic policies
- Find the policy $\pi_{\boldsymbol{\theta}}$ maximizing $J(\boldsymbol{\theta})$:

$$J(\boldsymbol{\theta}) = \int_{\mathcal{S}} d^{\pi_b}(s) Q(s, \pi_{\boldsymbol{\theta}}(s), \boldsymbol{\theta}) \, \mathrm{d}s$$



- Taking the gradient of $J(\boldsymbol{\theta})$ we obtain:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{s \sim d^{\pi_b}(s)} \left[\nabla_a Q(s, a, \boldsymbol{\theta})|_{a=\pi_{\boldsymbol{\theta}}(s)} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(s) + \nabla_{\boldsymbol{\theta}} Q(s, a, \boldsymbol{\theta})|_{a=\pi_{\boldsymbol{\theta}}(s)} \right]$$

Actor-Critic algorithm

Off-policy actor-critic with PBVFs

Given the behavioral π_b, find $\pi_{\boldsymbol{\theta}}$ **maximizing** $J(\boldsymbol{\theta})$:

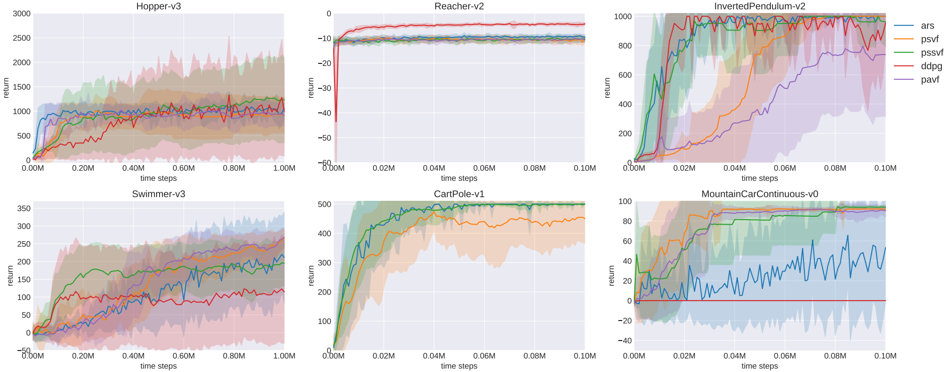
1. Collect data with π_b (expensive in RL)
2. Use data to train $V(\boldsymbol{\theta})$, $V(s, \boldsymbol{\theta})$ or $Q(s, a, \boldsymbol{\theta})$
3. Find $\pi_{\boldsymbol{\theta}}$ following $\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}})$ (offline optimization)
4. Set new behavioral $\pi_{\boldsymbol{\theta}} \leftarrow \pi_b$
5. Repeat until convergence

Demonstration

- PSSVF on LQR using shallow policies

Experiments

- Comparison with DDPG (Lillicrap et al., 2015) and ARS (Mania et al., 2018)



Limitations

Problem:

- The method does not scale well with the number of policy parameters

We must reduce the dimensionality of the policy.

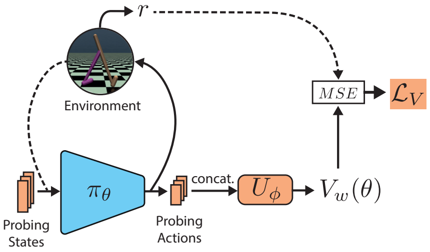
Some desirable properties for policy embedding:

- Differentiability in policy parameters
- Invariances to policy size

Policy Fingerprinting (Harb et al., 2020)

To evaluate a policy π_θ:

- Learn a set of 'probing states' $\{\tilde{s}_k\}_{k=1}^K$ to feed to the policy
- Learn an MLP U_ϕ mapping the 'probing actions' $\{\tilde{a}_k = \pi_\theta(\tilde{s}_k)\}_{k=1}^K$ produced in the probing states to the return r



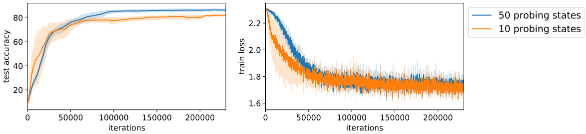
Setting $w = \{\phi, \tilde{s}_1, \dots \tilde{s}_K\}$:

$$\min_w \mathcal{L}_V := \min_w \mathop{\mathbb{E}}_{(\pi_\theta, r)\in B}[(V_w(\theta) - r)^2] = \min_{\phi, \tilde{s}_1, \dots \tilde{s}_K} \mathop{\mathbb{E}}_{(\pi_\theta, r)\in B}[(U_\phi([\pi_\theta(\tilde{s}_1), \dots, \pi_\theta(\tilde{s}_K)]) - r)^2]$$

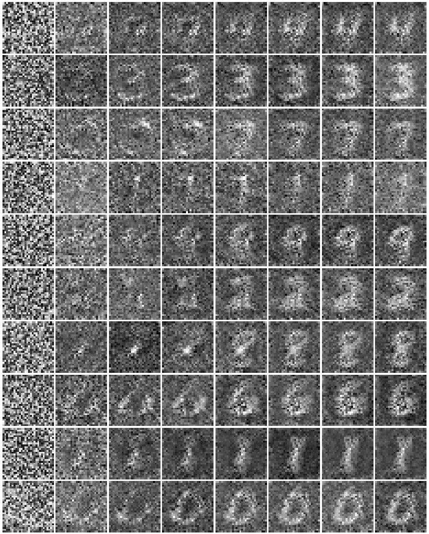
Demonstration: Online MNIST

We start with randomly initialized CNN π_θ and PSSVF $V_w(\theta)$ and iteratively:

- Compute the loss l of π_θ and store (π_θ, l) in the buffer
- Use the data to train $V_w(\theta)$
- Use $V_w(\theta)$ to improve the CNN



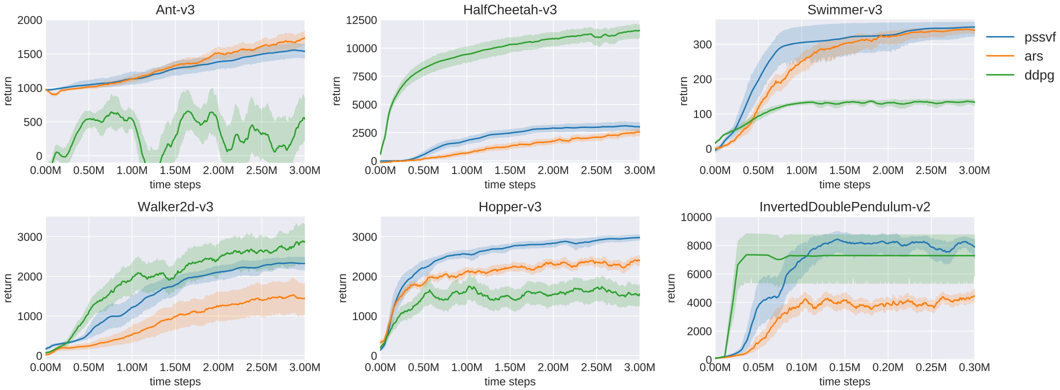
- Learned probing states are digits



Demonstration: Offline MNIST

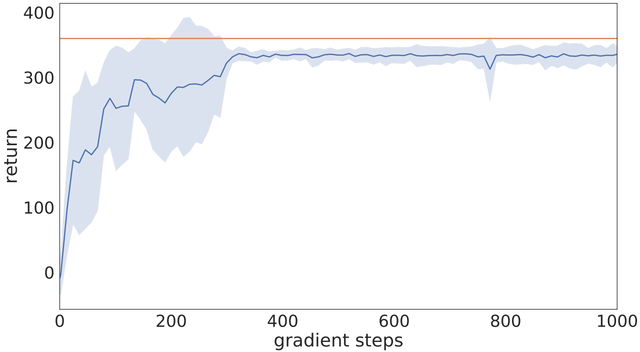
- Given an offline dataset $\{\pi_{\theta_i}, l_i\}_{i=1}^{N}$ of randomly initialized CNNs and their losses (maximum accuracy 12%) on a batch of images, we train V_w to evaluate such CNNs
- $\approx \mathbb{E}[R_0|\theta]$
- We randomly initialize a new CNN and take many steps of gradient ascent through the learned value function, finding $\theta^* = \arg\max_{\theta} V_w(\theta)$

Main Results



Zero-shot learning of new policy architectures

- A PSSVF trained using deep deterministic policies zero-shot learns a linear policy with similar performance in Swimmer



- best deep policy in training
- linear policy zero-shot learned

Learning optimal policies from few crucial states

- Policy fingerprinting learns a set of crucial states that are informative for policy evaluation
- A randomly initialized policy can learn near-optimal behaviors in Swimmer (Hopper) by knowing how to act only in 3 (5) such crucial learned states

Probing States visualization in RL

- More examples of learned probing states:

Limitations and Future Work

Limitations

- Different policies may need different probing states for efficient evaluation
- If there are many probing states, then the concatenated vector of probing actions can be very large
- In some environments a lot of probing states are needed to evaluate a policy

Future work

- Extension to $V(s,\theta)$, $Q(s,a,\theta)$
- Recursive generation of probing states

(Online) Return-Conditioned Reinforcement Learning

A command of the form 'act in the environment and achieve a desired return' is given as input to the policy.

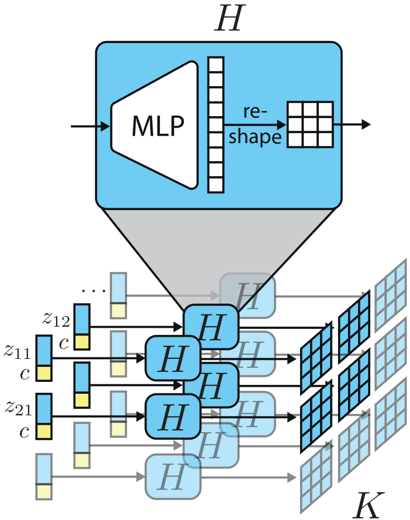
General framework (Schmidhuber, 2019; Srivastava et al., 2019):

- Act in the environment with a given return command
- Use the data to learn a map from return commands and states into actions
- Ask for higher return commands at next environment interaction
- Most methods are based on the idea of hindsight learning: the agent's behavior is optimal if it had had the achiever return as input command

Here we propose instead to learn a generator $G_\rho : \mathbb{R} \longrightarrow \Theta$ such that if $\theta = G_\rho(c)$, then $\mathbb{E}[R_0|\theta] = c$.

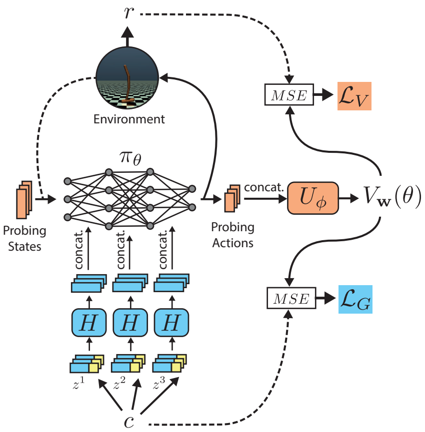
Fast Weights / Hypernetworks (Schmidhuber, 1992; Ha et al., 2016)

- Each weight matrix of the policy is split into slices
- For each slices we learn an embedding z
- A shared MLP H receives as input a z and outputs a slice
- further context information can be given to H in form of an additional conditioning input c
- The policy weights are finally combined concatenating the generated slices
- The parameters of the generator G_ρ are the parameters of H and all the embeddings z

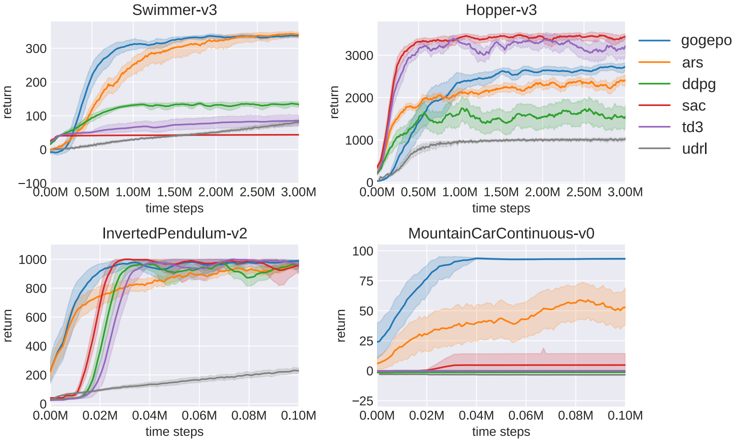


Goal-Conditioned Generators of Deep Policies

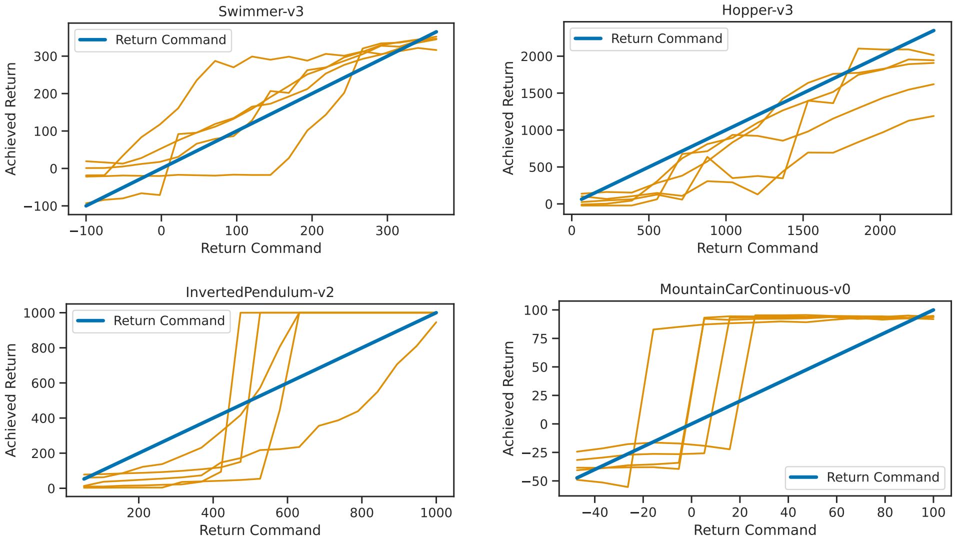
1. Given command c, generate $\theta = G_\rho(c)$
2. Simulate $\pi_{\boldsymbol{\theta}}$ and obtain return r
3. Use data to train $V_{\mathbf{w}}(\boldsymbol{\theta})$
4. Minimize $\mathcal{L}_G(\rho) = \underset{c \in D}{\mathbb{E}}[(V_{\mathbf{w}}(G_\rho(c)) - c)^2]$ to learn the parameters ρ of the generator
5. Choose new command c
6. Repeat until convergence



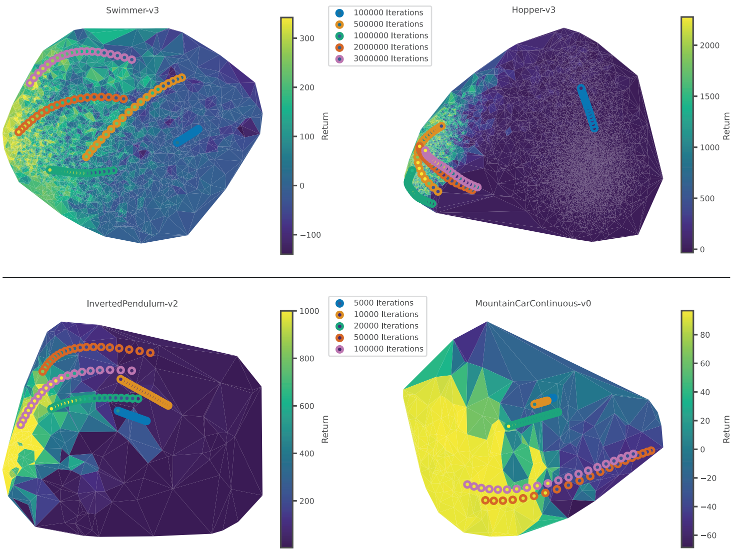
Experiments



Testing the Generator



Testing the Generator



Limitations and Future Work

Limitations

- Policies created by an untrained generator might have weights that are far from typical initialization schemes
- The method is based on the episodic return signal
- Different policies may need different probing states for efficient evaluation
- In some environments a lot of probing states are needed to evaluate a policy

Future work

- Extension to state-based evaluation and generation of policies
- Context commands different than desired return
- Richer policy generation through VAEs

Thank You for Your Attention!

References

Faccio, F., Kirsch, L., and Schmidhuber, J. (2021). Parameter-based value functions. *arXiv preprint arXiv:2006.09226*.

Ha, D., Dai, A., and Le, Q. V. (2016). HyperNetworks. In *International Conference on Learning Representations*.

Harb, J., Schaul, T., Precup, D., and Bacon, P.-L. (2020). Policy evaluation networks. *arXiv preprint arXiv:2002.11833*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Mania, H., Guy, A., and Recht, B. (2018). Simple random search of static linear policies is competitive for reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1800–1809.

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Schmidhuber, J. (1992). Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139.

Schmidhuber, J. (2019). Reinforcement Learning Upside Down: Don't Predict Rewards–Just Map Them to Actions. *arXiv:1912.02875*.

Srivastava, R. K., Shyam, P., Mutz, F., Jaśkowski, W., and Schmidhuber, J. (2019). Training Agents Using Upside-down Reinforcement Learning. In *NeurIPS Deep RL Workshop*.

Stratonovich, R. (1960). Conditional Markov processes. *Theory of Probability And Its Applications*, 5(2):156–178.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.