Reinforcement learning

COGS-Q355

Current Reinforcement Learning

LETTER

doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

The theory of reinforcement learning provides a normative account[1], deeply rooted in psychological[2] and neuroscientific[3] perspectives on animal behaviour, of how agents may optimize their control of an environment. To use reinforcement learning successfully in situations approaching real-world complexity, however, agents are confronted

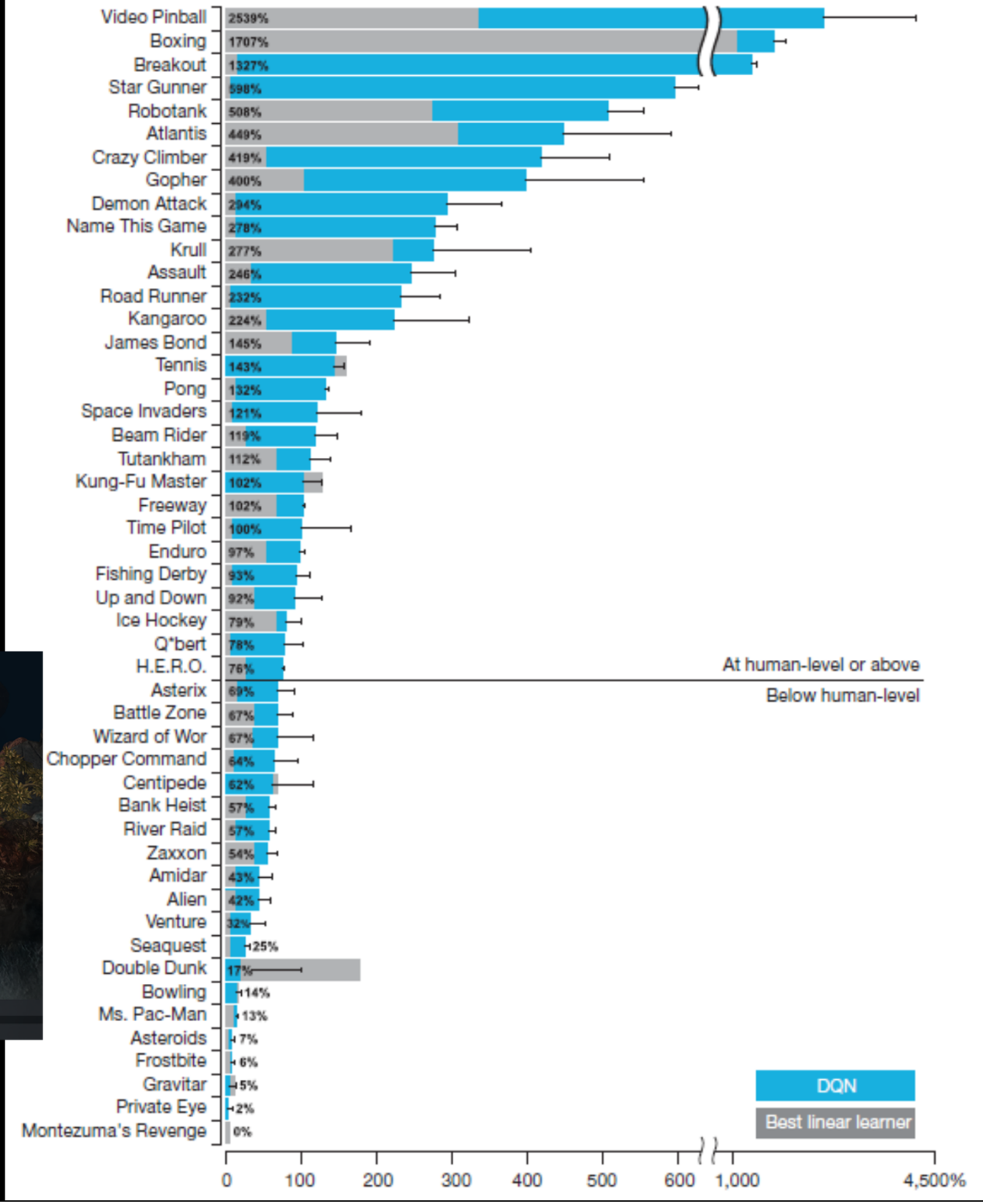
agent is to select actions in a fashion that maximizes cumulative future reward. More formally, we use a deep convolutional neural network to approximate the optimal action-value function

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \mid s_t = s,\ a_t = a,\ \pi\right],$$

Mnih et al. (2015) Nature, doi:10.1038/nature14236

Current Reinforcement Learning

AlphaStar / Starcraft



Thorndike's law

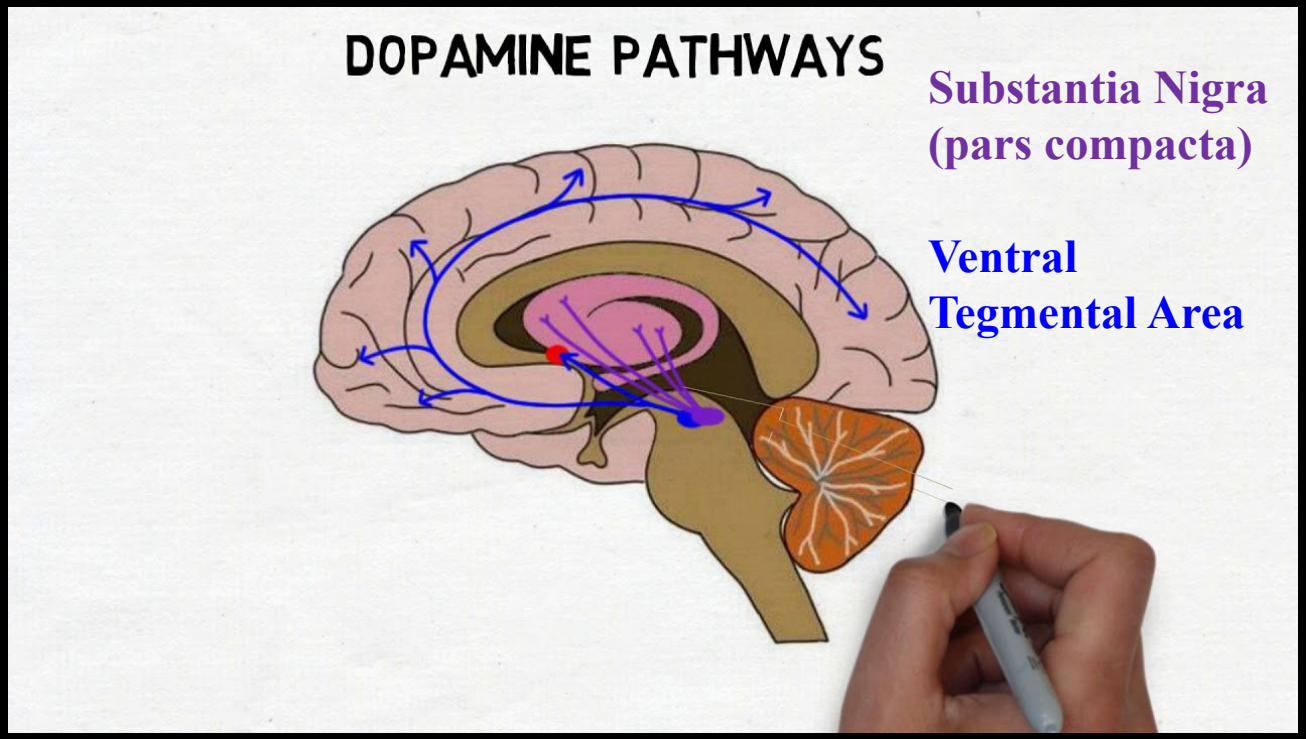
- Behaviors that are rewarded will be more likely to be repeated

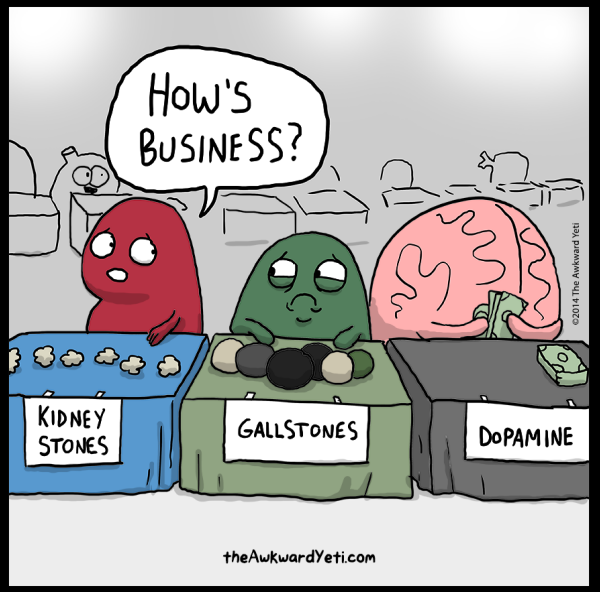
Reinforcement Learning

- Supervised learning: you are told what is the desired output
- Unsupervised learning: you are NOT told what is the desired output
- Reinforcement learning: you are only told whether the output is “good” or “bad”, but you are not told what output would be better

Dopamine

- Is there a signal in the brain that might serve to reinforce behavior? Yes. Dopamine



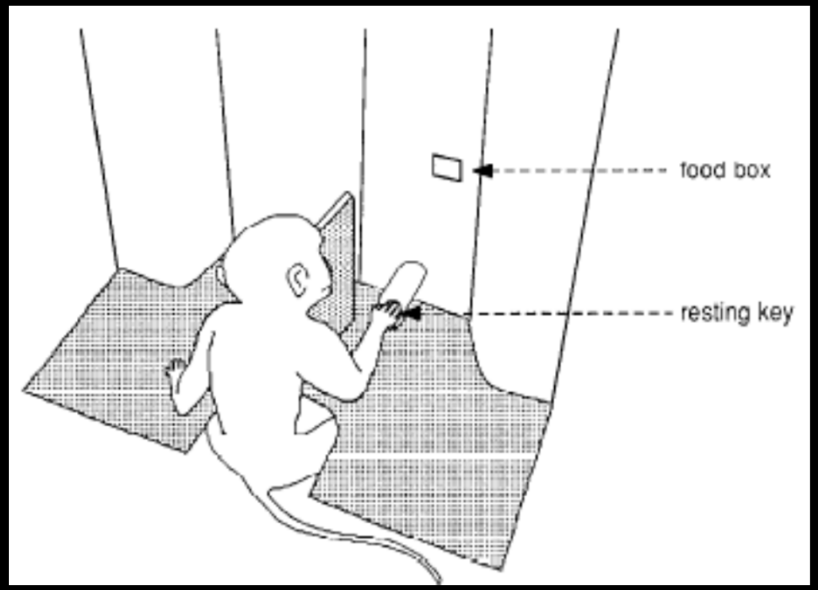


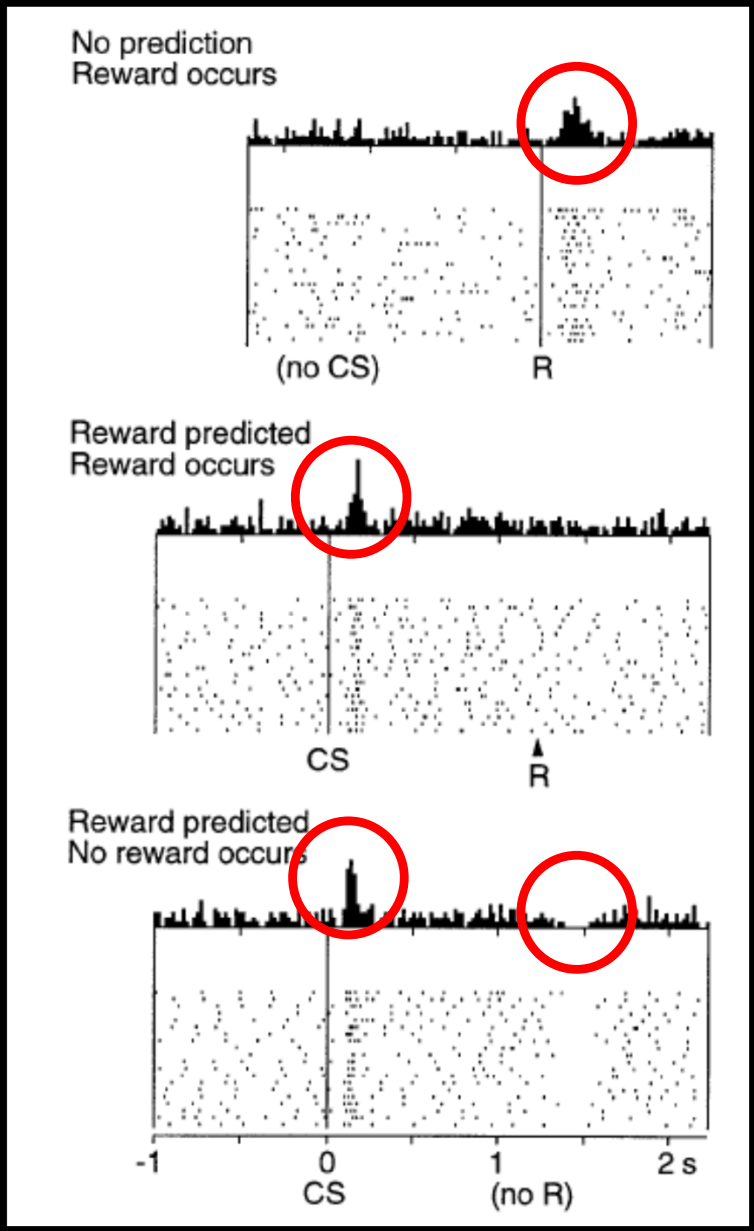
Dopamine

- Released in response to rewarding stimuli
- Released if a cue predicts a *future* rewarding stimulus
- Every addictive drug causes an increase of dopamine release, which reinforces the drug taking behavior

Dopamine

- In the early 1990s, dopamine cells were recorded in behaving monkeys
- Dopamine cells fired a burst of activity when a reward-predicting cue appeared.
- After training, if an expected reward is omitted, cells pause briefly





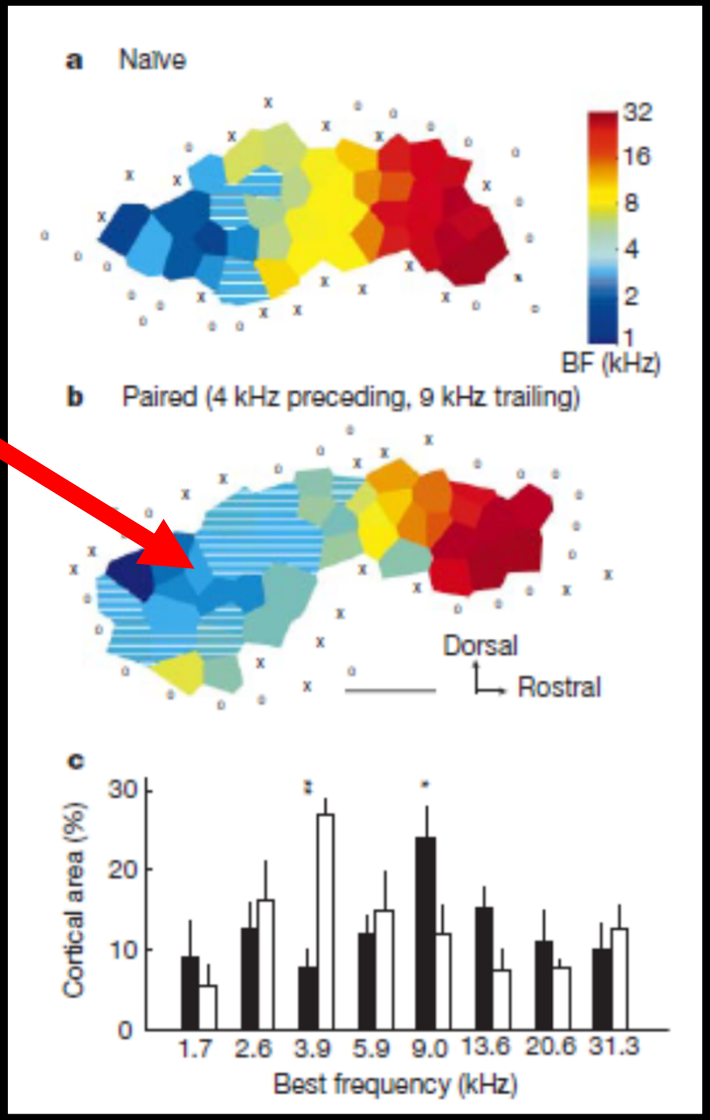
Dopamine

- At the neuron level, dopamine causes synaptic strengths to increase when pre- and post-synaptic cells are active

$$\frac{dW}{dt} = r * x * y$$

Dopamine

- At the cortical level, dopamine increases plasticity
- Sounds followed by VTA dopamine stimulation showed more cortical representational area in rats

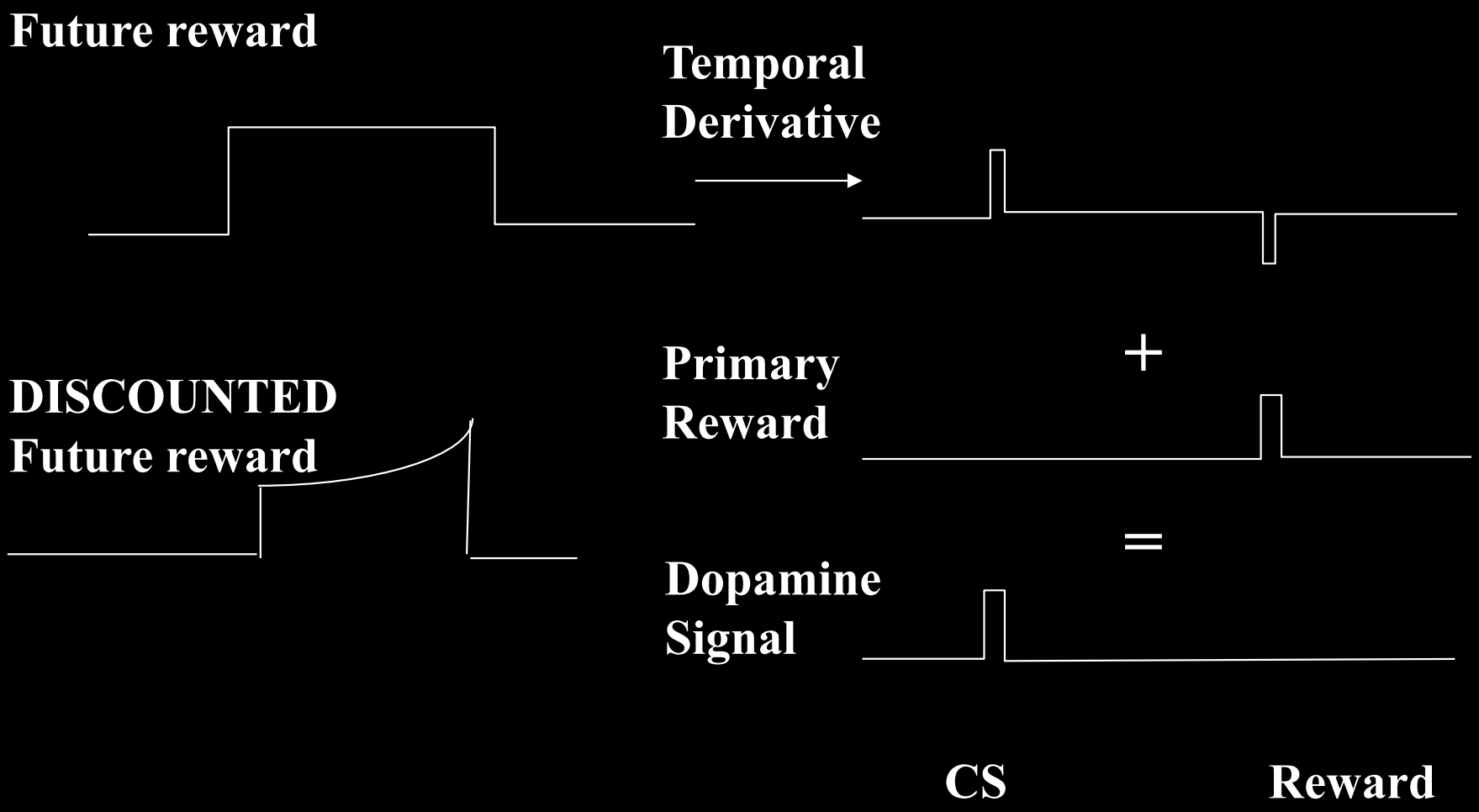


Bao et al., 2001

Reinforcement Learning

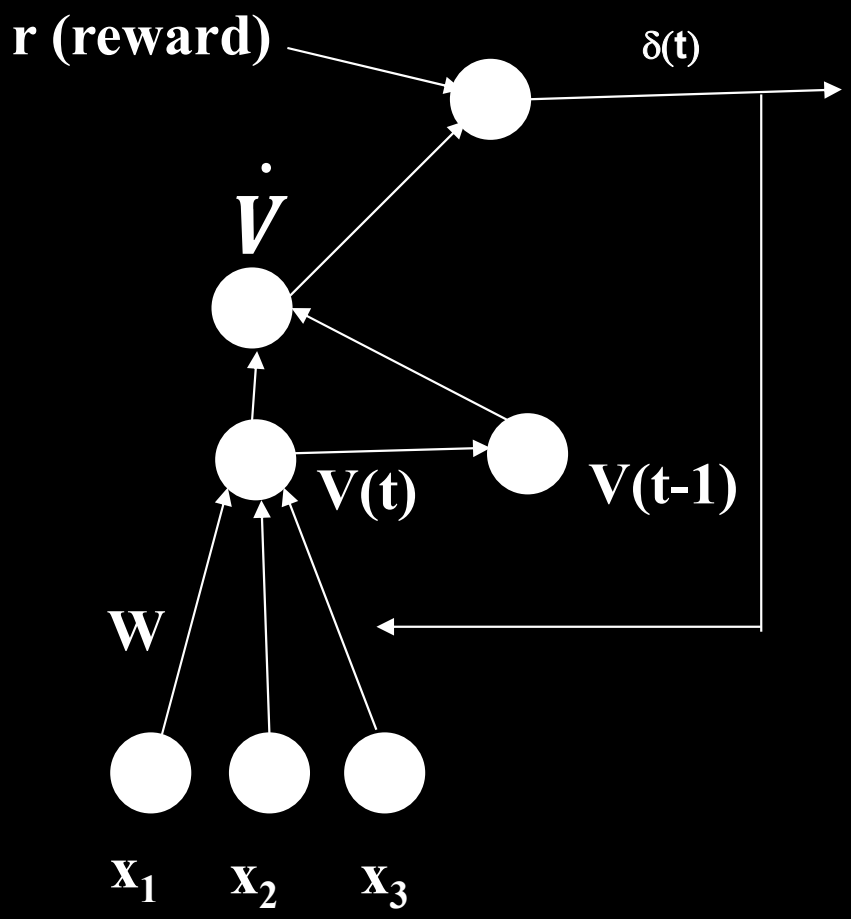
- How do dopamine cells “compute” this signal?
- Various models, but most popular is “temporal difference” model.
- Intuitively, compute a signal representing sum of future reward.
- Dopamine signal is the time derivative of this expected future reward

Temporal difference model



Montague et al., 1996

- TD (temporal difference) model



Learning signal (time derivative of reward prediction):

$\delta(t)=r(t)+\dot{V}(t)$

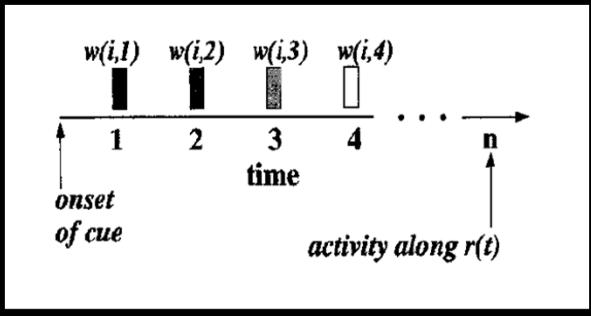
$\delta(t)=r(t)+V(t)-V(t-1)$

Weight update

$$w(i,\ t-1)_{new} = w(i,\ t-1)_{prev} + \eta x(i,\ t-1)\delta(t)$$

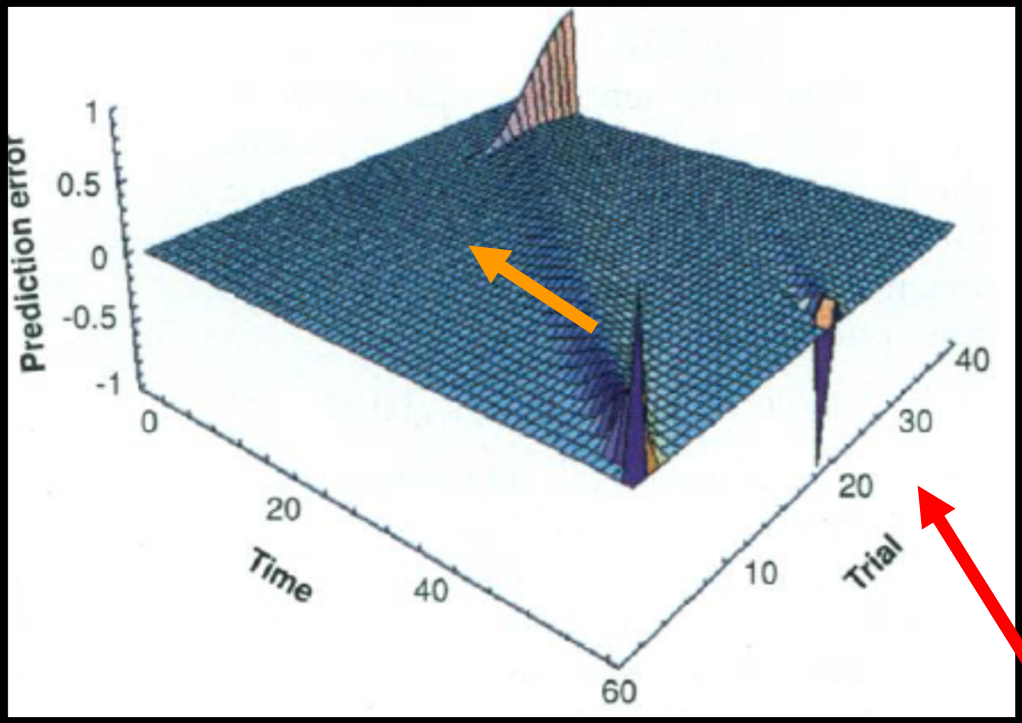
η = learning rate

- δ at time t causes weights to increase, activate V at <u>previous</u> (i.e. t-1) time step
- Next time, greater V at t-1 causes δ signal earlier, which causes V to rise earlier, etc.



Temporal difference model

- TD (temporal difference) model



Prediction travels back in time as learning proceeds across trials

Learning signal (time derivative of reward prediction):

$\delta(t)=r(t)+\dot{V}(t)$

$\delta(t)=r(t)+V(t)-V(t-1)$

Weight update

$$w(i,\ t-1)_{new} = w(i,\ t-1)_{prev} + \eta x(i,\ t-1)\delta(t)$$

η = learning rate

- δ at time t causes weights to increase, activate V at previous (i.e. t-1) time step
- Next time, greater V at t-1 causes δ signal earlier, which causes V to rise earlier, etc.

Schultz et al., 1997

Temporal difference model

- How do you get discounting?

DISCOUNTED
Future reward

Learning signal (time derivative of reward prediction):

Change this:

$$\delta(t)=r(t)+V(t)-V(t-1)$$

To this:

$$\delta(t)=r(t)+\gamma V(t)-V(t-1)$$

Where γ is slightly less than 1

→ Now the V at the previous time step will be trained to reach a value slightly less than the value at V(t), which means V will increase as time progresses

Schultz et al., 1997

Barto et al., 1983

- Actor-critic architecture. The "actor" generates actions, and the "critic" evaluates reward and trains the actor
- Cart-pole problem – how long can the inverted pendulum be kept upright?

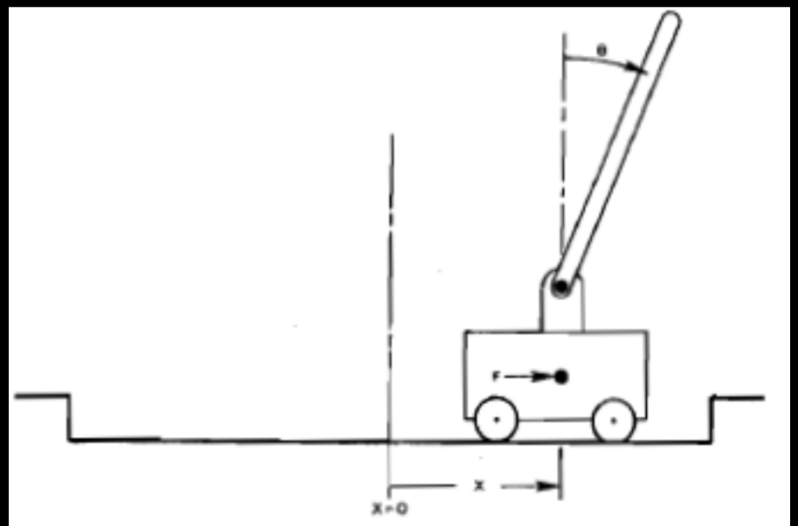


Fig. 1. Cart–pole system to be controlled. Solution of system's equations of motion approximated numerically (see Appendix).

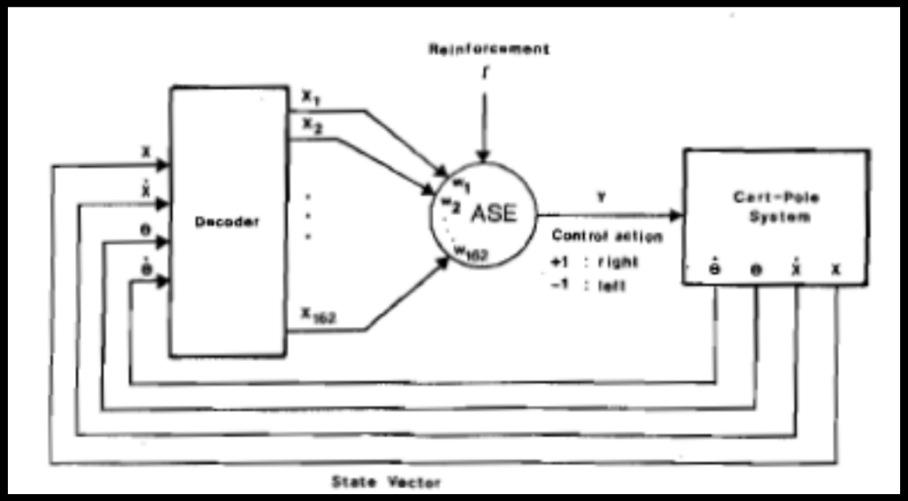
x	position of the cart on the track,
θ	angle of the pole with the vertical,
$\dot{x}$	cart velocity, and
$\dot{\theta}$	rate of change of the angle.

Barto et al., 1983

- Representation of state: the “boxes” method.
- Divide 4-dimensional space into 162 discrete little “boxes” (3x3x6x3)
- State boxes are assigned one per input neuron (x_i), so that only one state box is active at a given time
- y(t) = control output (force left or right)

$$y(t) = f\left[\sum_{i=1}^{n} w_i(t)x_i(t) + \text{noise}\,(t)\right]$$

$$f(x) = \begin{cases} +1, & \text{if } x \geqslant 0 \quad \text{(control action right)} \\ -1, & \text{if } x < 0 \quad \text{(control action left).} \end{cases}$$



Barto et al., 1983

- So how do we train the weights $w_i(t)$?

$$w_i(t+1) = w_i(t) + \alpha r(t) e_i(t) \qquad (2)$$

where

α	positive constant determining the rate of change of w_i,
$r(t)$	real-valued *reinforcement* at time t, and
$e_i(t)$	*eligibility* at time t of input pathway i.

- Ok, then what are "r" and "e"??

  r = 0 as long as the pole is upright. r= -1 when it falls.

  e is an eligibility trace, i.e. a record of recent activity

- (Note "δ" below is a decay rate, NOT dopamine as before

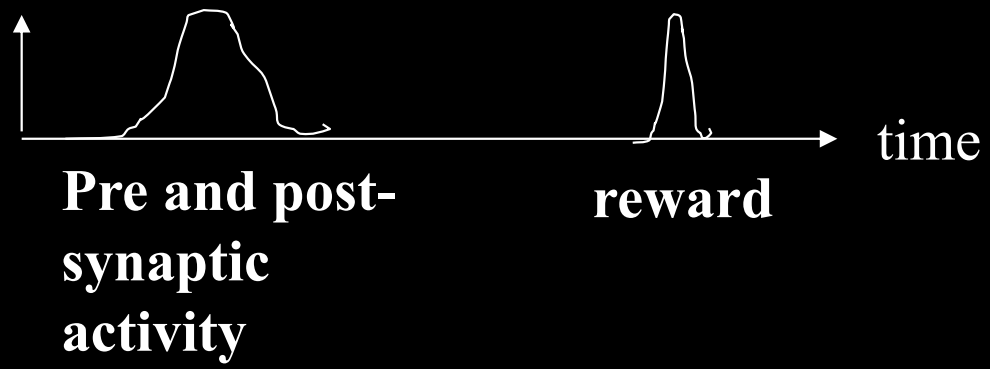
$$e_i(t+1) = \delta e_i(t) + (1-\delta) y(t) x_i(t),$$

where $\delta, 0 \leqslant \delta < 1$, determines the trace decay rate.

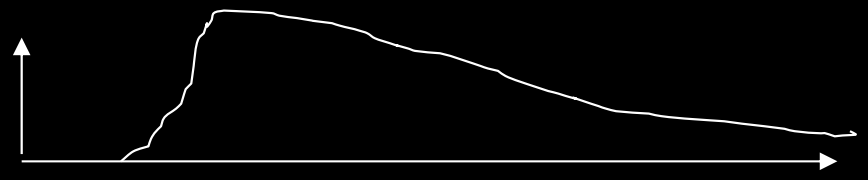
Eligibility traces

- Why eligibility traces? Often there is a DELAY between an action and a reward.
- If we used pre- and post-synaptic activity as a measure of what should be strengthened by a reward signal, then the activity will have died away by the time the reward comes

How to bride the time gap??



The eligibility trace “e” creates activity that persists

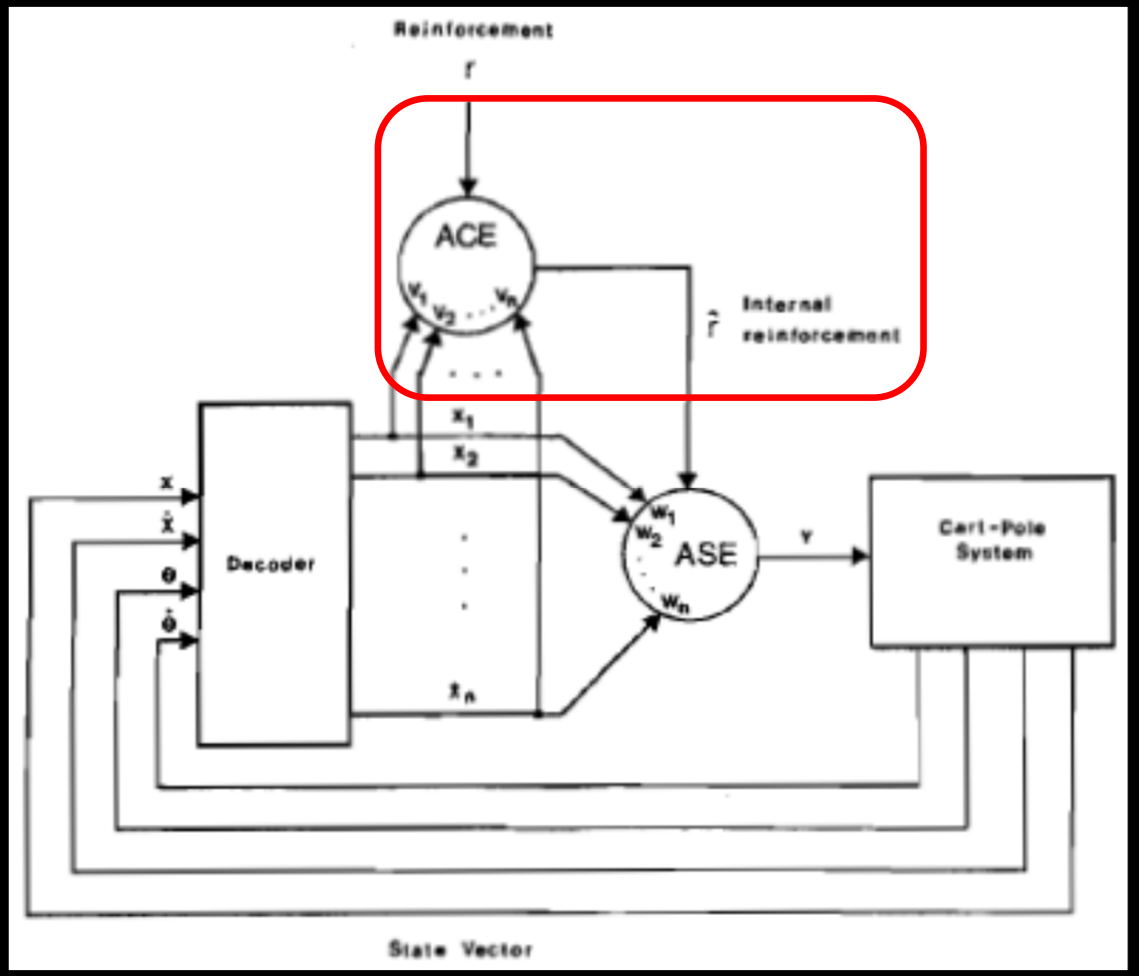


Barto et al., 1983

- We can come up with a better value for r, namely r that anticipates the future reinforcement, beyond simply measuring the present. This reduces uncertainty about how the actions are working out.
- The mechanism to do this is the Adaptive Critic Element (ACE).
- The ACE tries to adjust the “reward” signal now to include predictions about how the current action will affect reward later.

Barto et al., 1983

- Define variable "p", which is prediction of future reinforcement based on "x", the state, with weights "v"



$$p(t) = \sum_{i=1}^{n} v_i(t) x_i(t)$$

p is like "V" in Montague et al. 1996

Barto et al., 1983

$$p(t) = \sum_{i=1}^{n} v_i(t) x_i(t)$$

- Now we need a way to update the weights "v":

$$v_i(t+1) = v_i(t) + \beta[r(t) + \gamma p(t) - p(t-1)]\,\bar{x}_i(t),$$

- This is a "discounted" temporal difference (by γ), $\hat{r}(t) = r(t) + \gamma p(t) - p(t-1).$
- with a trace of input $\bar{x}$

$$\bar{x}_i(t+1) = \lambda \bar{x}_i(t) + (1-\lambda) x_i(t),$$

- Typically γ = about 0.95

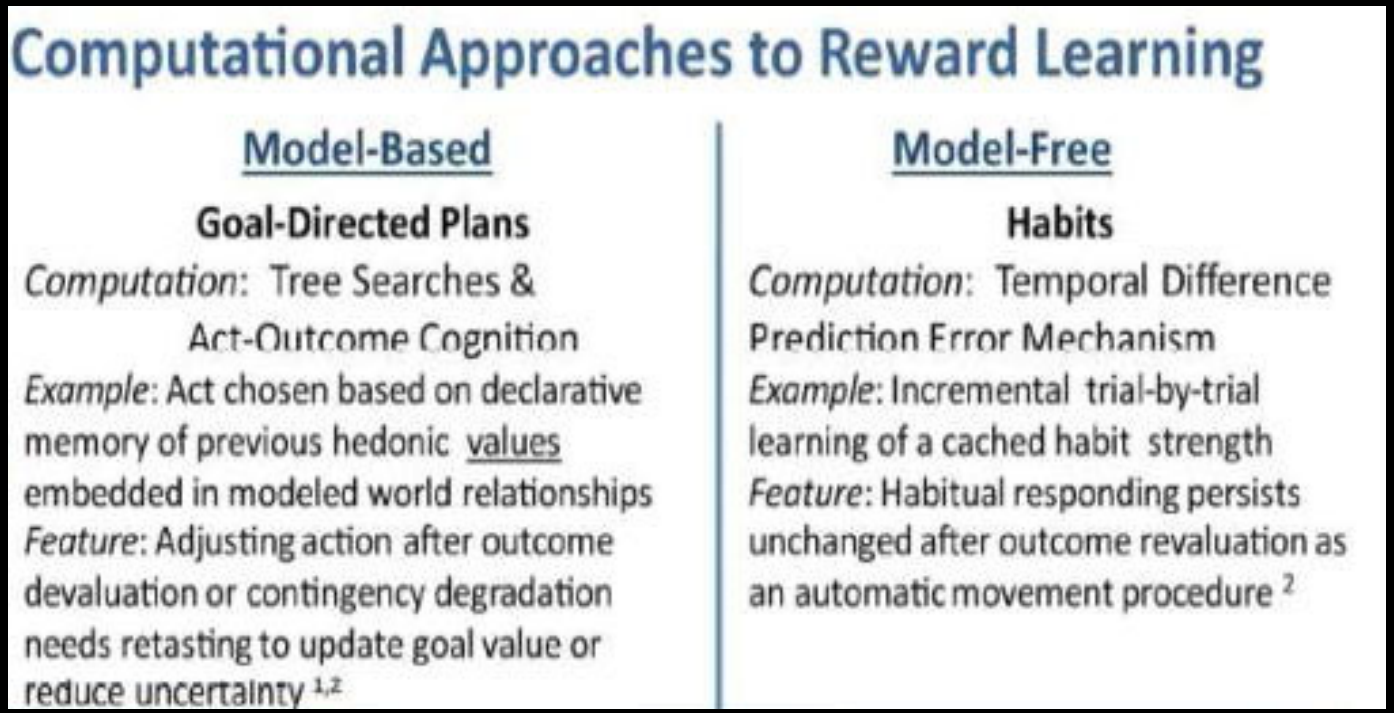
Bellman equation

Reinforcement Learning

- Barto et al. was a landmark, but it has limitations
  - Representation of “boxes” in state space doesn’t scale well at all.
  - No internal model of the world
- RL can be further developed into model-free and model-based categories
- Model-free RL: habits
- Model-based RL: goal-directed plans

Reinforcement Learning

- Model-based RL: agent has a model that can predict the next state and reward before taking the action
- Barto et al., 1983 is model free RL



https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4074442/

Some terms

- Action (A): All the possible moves that the agent can take
- State (S): Current situation returned by the environment.
- Reward (R): An immediate return send back from the environment to evaluate the last action.
- Policy (π): The strategy that the agent employs to determine next action based on the current state.
- Value (V): The expected long-term return with discount, as opposed to the short-term reward R. $V\pi(s)$ is defined as the expected long-term return of the current state sunder policy π.
- Q-value or action-value (Q): Q-value is similar to Value, except that it takes an extra parameter, the current action *a*. $Q\pi(s, a)$ refers to the long-term return of the current state *s*, taking action *a* under policy π.

https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287

Reinforcement Learning

- The "model" in model-based RL:
  - $p(s_{t+1} \mid s_t, a_t)$
  - Probably that the next state is "s" given that action "a" was taken from previous state "s"
  - Learn p by exploring, making actions from a given state and seeing what happens

Q-learning

- The "Q" value Q(s,a) is the discounted value of future reward associated with a state and action (like "p" in Barto et al., 1983)

```
Initialize Q(s, a) arbitrarily
Repeat (for each episode):
   Initialize s
   Repeat (for each step of episode):
      Choose a from s using policy derived from Q
         (e.g., ε-greedy)
      Take action a, observe r, s'
      Q(s, a) <-- Q(s, a) + α [r + γ max_α,Q(s', a') - Q(s, a)]
      s <-- s';
   until s is terminal
```

- α is learning rate, γ is discount parameter
- (What happens if the reward moves to a different state? Have to re-train all the Q values!!)

https://www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html

SARSA

- The “Q” value for SARSA reflects updates from a quintuple of numbers – s, a, r, s’, a’.
- I.e. taking action a from state s lead to reward r in state s’, and subsequent action a’.

- SARSA is like Q learning, but where Q learning assumes the a’ action is the one that maximizes reward, the SARSA update rule allows learning from other, perhaps less optimal actions a’. This can be useful when the reward maximizing action carries unacceptably high risk.

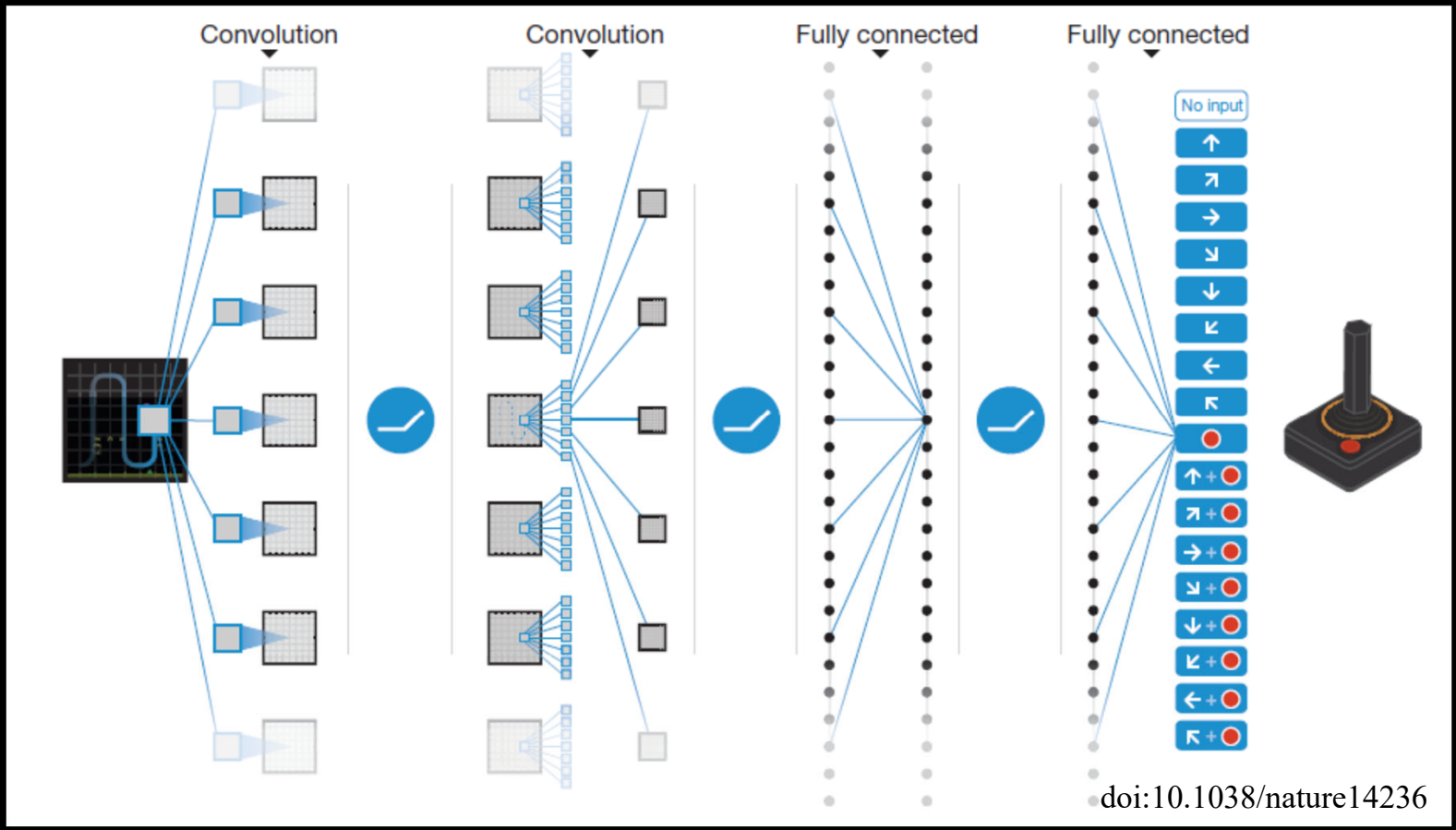
https://www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html

Deep Q Learning

- RL with discrete states *s* doesn't scale well. Too many states → algorithm intractable.
- Instead of discrete state representations, use function approximation to model state/action transition probabilities (e.g. via neural net)
  - $x_{t+1} = f(x_t, a_t)$ (where x is a state vector)
- Can also use deep network to learn Q values of states, e.g. $Q = f(x_t, a_t)$

Deep Q Learning

- Mnih et al. (2015) – use deep network to approximate the Q values
- State (s) is the visual state of the board, i.e. visual input
- Actions (a) – there are 18 possible actions
- Reward (r) = change in game score



Deep Q Learning

- Mnih et al. (2015) – train deep Q network on Atari games to approximate Q values as Q(s,a; θ)
- θ are the deep network weights
- Learning (model free) proceeds as:
  - 1) Use the DQN to predict the Q value of all actions a' (and associated next states s'), given current states s and action a
  - 2) Pick the action a' associated with the best Q value and execute it, i.e. choose a' to maximize

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') | s,a\right]$$

This is the network "output"

  - This gives a predicted next state Q value estimate y (before acting):

$$y = r + \gamma \max_{a'} Q(s',a';\theta_i^-)$$

  - 3) After the action, the system has transitioned to some state s', so re-evaluate Q(s,a; θ) at the next state, to get the actual next state Q value from the deep network:
  - The loss function is then L(θ) = (predicted – actual)^2

$$L_i(\theta_i) = \mathbb{E}_{s,a,r}\left[(\mathbb{E}_{s'}[y|s,a] - Q(s,a;\theta_i))^2\right]$$

  - So do gradient descent on L to update θ

Deep Q Learning

- Mnih et al. (2015) – details
- Model free – doesn't estimate reward and transition probabilities directly
- Reward discounting γ set to 0.99
- Exploration is important to find potentially more valuable options, so network uses "off policy" (ε-greedy) learning – selects random actions with probability ε (annealed linearly from 1.0 to 0.1 over first 1 million frames, then 0.1 thereafter)

Deep Q Learning

- 84x84x2 input, with max of RGB and max luminance across 4 frames
- Convolutional neural network, as discussed before
- Each of the 49 Atari games was trained separately
- Train with RMSprop. Use every 4^{th} video frame as input
- Training is slow, so the last up to 1,000,000 video frames and output were stored and played back repeatedly to the network for training

OpenAI gym

- Source of Atari games
- “We provide the environment; you provide the algorithm”
- Pole-cart, “classic control” examples
- Algorithmic tasks, like learning multiplication by RL
- Atari games
- 2D and 3D robot control

https://gym.openai.com/

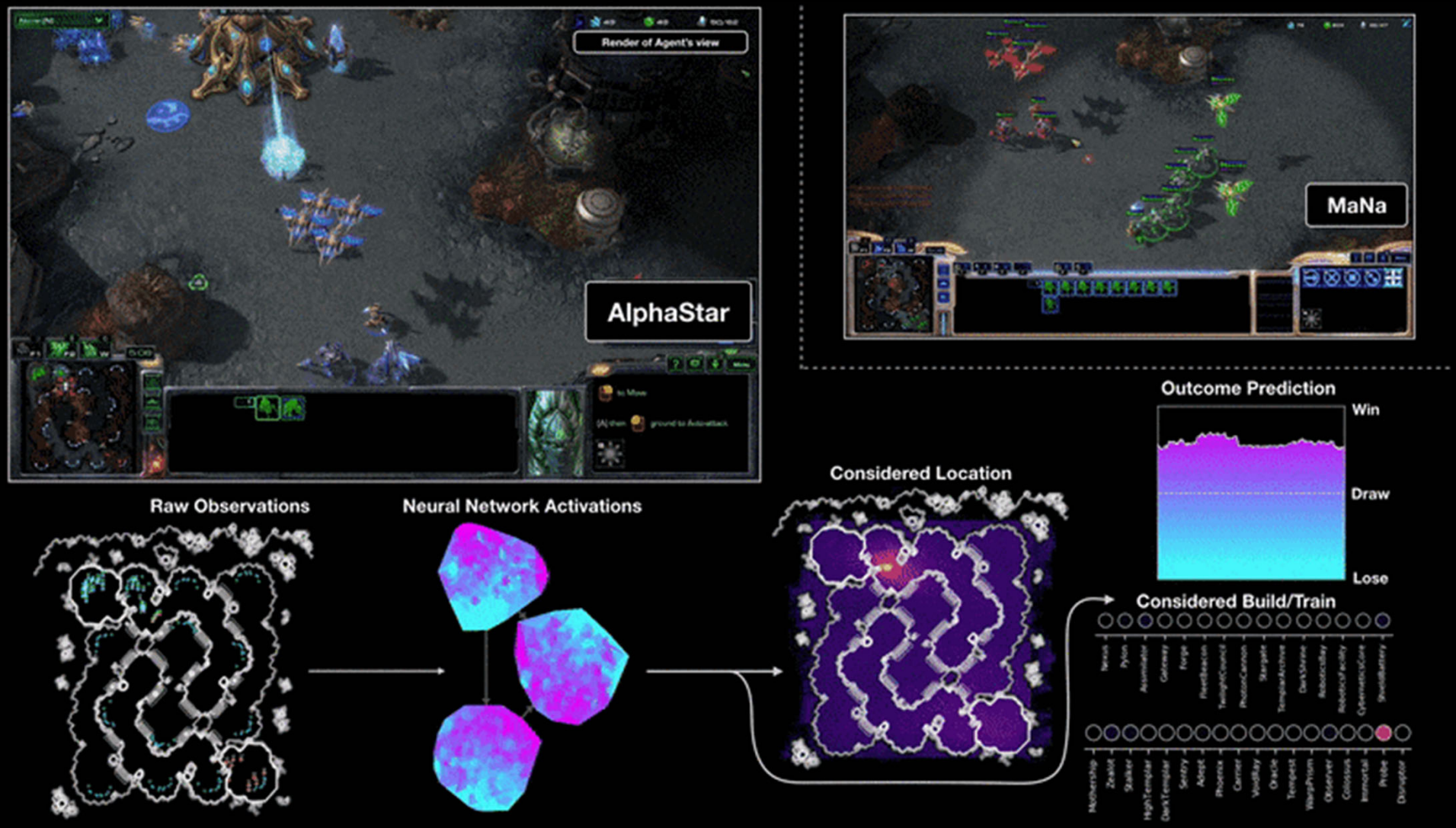
OpenAI gym

- Easy to use:
- git clone https://github.com/openai/gym
- cd gym
- pip install -e .

```
import gym
env = gym.make('CartPole-v0')
env.reset()
for _ in range(1000):
        env.render()
        env.step(env.action_space.sample()) # take a random action
env.close()
```


https://gym.openai.com/

AlphaStar



https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/#gif-264

AlphaStar

- Recently won 5-0 against one of worlds 10 strongest StarCraft II players
- “I was impressed to see AlphaStar pull off advanced moves and different strategies across almost every game, using a very human style of gameplay I wouldn’t have expected,” he said. “I’ve realised how much my gameplay relies on forcing mistakes and being able to exploit human reactions, so this has put the game in a whole new light for me. We’re all excited to see what comes next.”

AlphaStar

- Go is hard, with about 10^{800} possible game states
- AlphaStar not published yet, but consider alphaGo Zero (Silver, 2017):
- Learns *tabula rasa* to defeat DeepMind's previous best AlphaGo, 100-0
- How? Generative adversarial learning, i.e. have the network play against itself
- Monte Carlo Tree Search (MCTS) – starting with the current state of play, simulate running the game to completion 1600 times. Select the move that most often leads to a win

AlphaGoZero

- Monte Carlo Tree Search

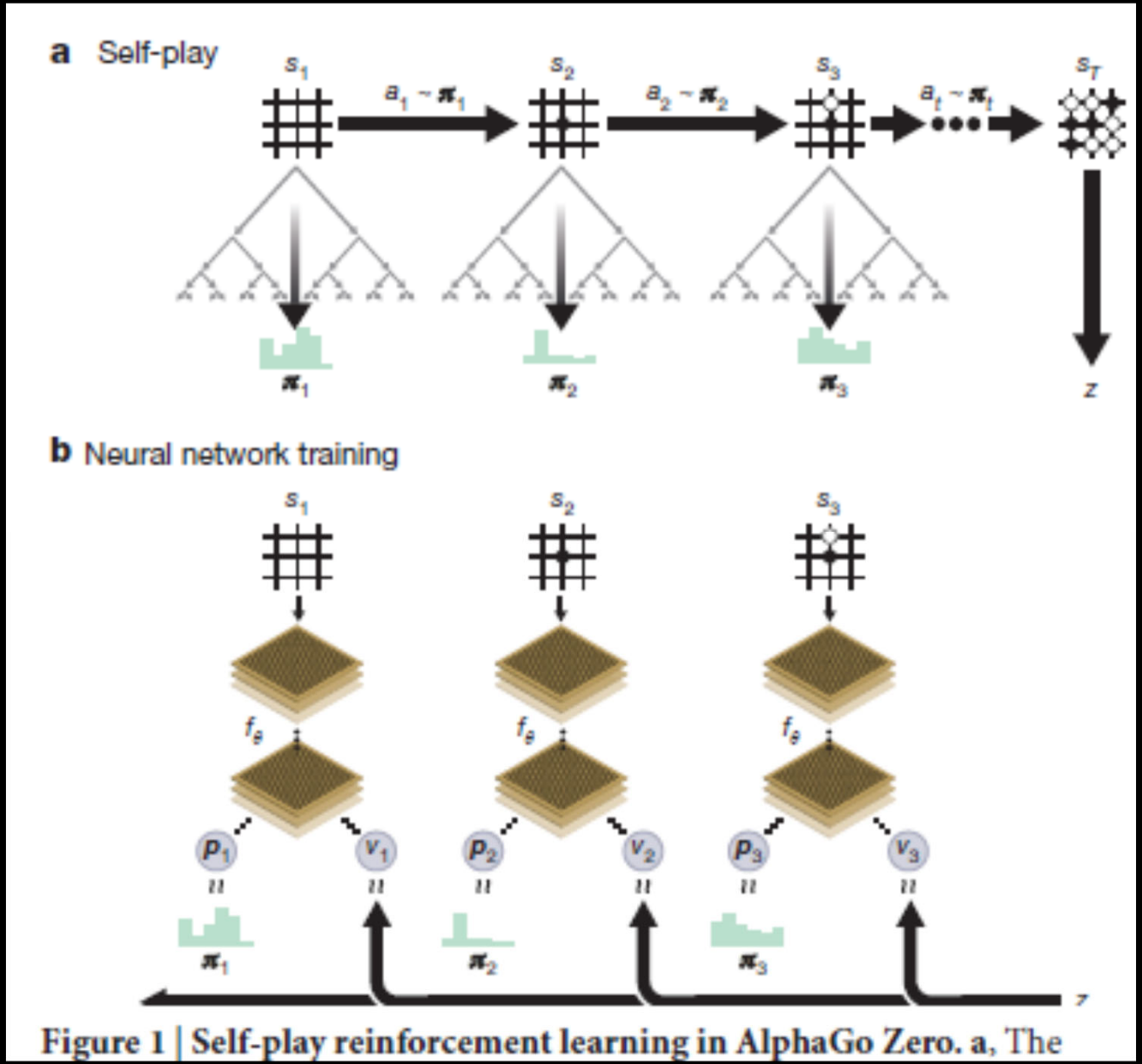


Figure 1 | Self-play reinforcement learning in AlphaGo Zero. a, The

AlphaGoZero

- The probability of a board move is $(p,v)=f_\theta(s)$, where p is the probability of a move, v is the probability of winning, and s is the current state of play. F is a deep neural network
- The MCTS generates an actual probability of moves (π) and actual probability of winning (z)
- The network is updated by gradient descent, with a loss function:

$$(p, v) = f_\theta(s) \text{ and } l = (z - v)^2 - \pi^{\mathrm{T}} \log p + c\|\theta\|^2$$

- Note SSE loss function for value but entropy loss function for probability. C prevents overfitting by regularizing parameters

- During performance of a game, 1600 games are simulated for each move, with the network playing against itself. For the first 30 moves, action selection is proportional to the probability of the move. After that, the most probable move is always chosen.

AlphaGoZero

- Completely smokes the competition, and only needs 1 machine with 4 TPUs (AlphaGo required 48 TPUs)