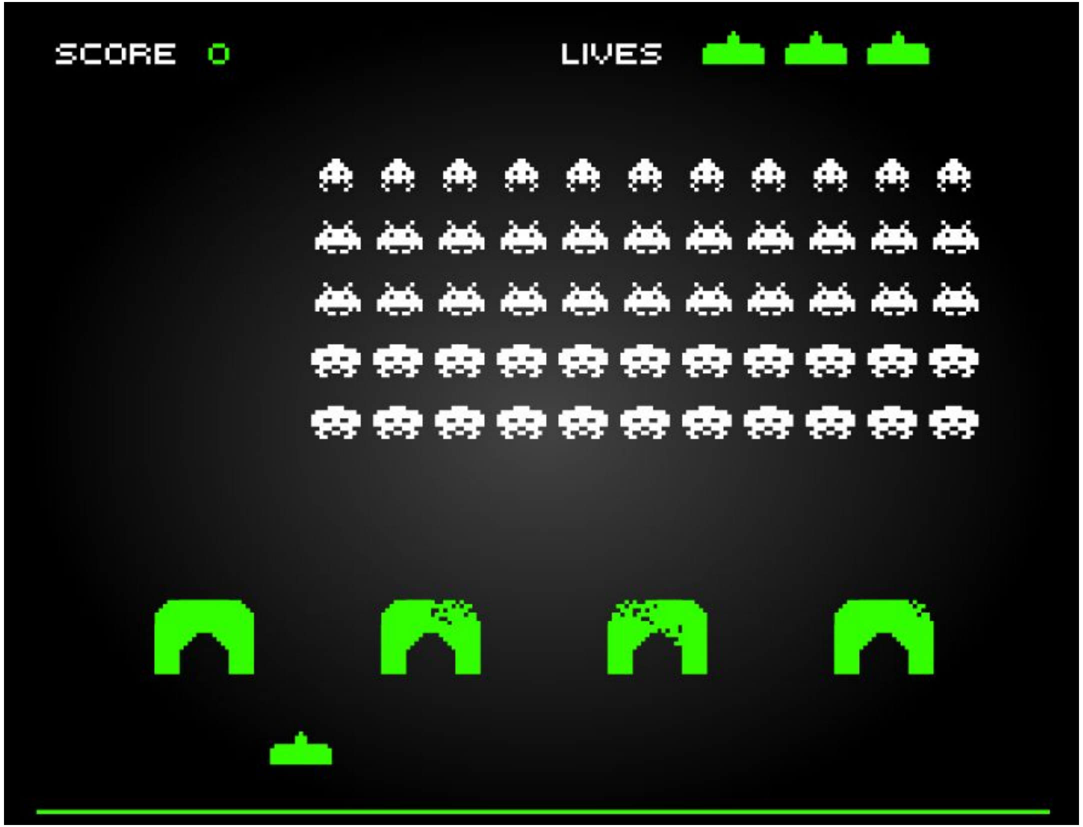
In the previous lecture ...

Markov Decision Process

A **Markov decision process** is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where

- $\mathcal{S}$ is a finite set of states.
- $\mathcal{A}$ is a finite set of actions.
- $\mathcal{P}$ is a state-action transition probability matrix,
  - $\mathcal{P}^a_{s,s'} = \mathbb{P}(S_{t+1} = s \mid S_t = s, A_t = a)$.
- $\mathcal{R}$ is a reward function
  - $\mathcal{R}^a_s = \mathbb{E}(R_{t+1} \mid S_t = s, A_t = a)$
- γ is a **discount factor**, $\gamma \in [0, 1]$.

Let’s define an MPD for space invaders



Policies

A **policy** π is a distribution over actions given states,

$$\pi(a \mid s) := \mathbb{P}(A_t = a \mid S_t = s).$$

- A policy fully defines the behaviour of the agent.
- Policies are stationary and depend only on the current state, not the history.

Recursion equations

As with MRP's, we have similar recursive relations:

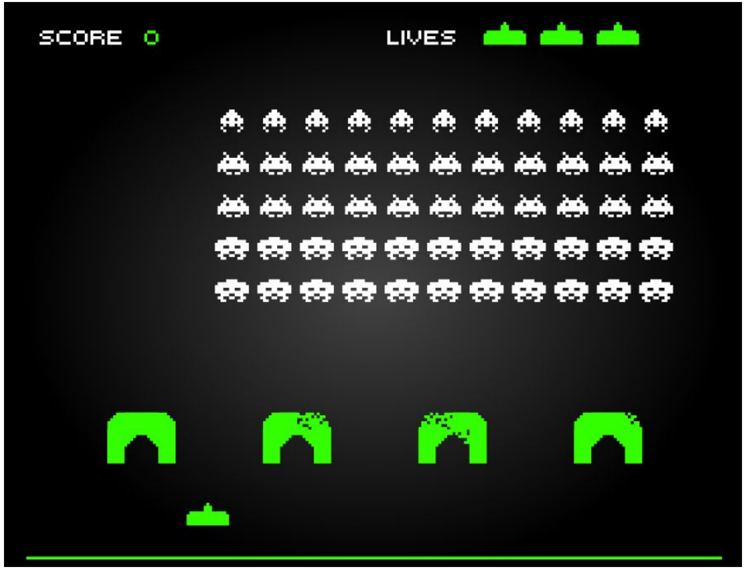
$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi(R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s) \\ q_\pi(s,a) &= \mathbb{E}_\pi(R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a) \end{aligned}$$

In particular, we can, for a given policy π, solve the Bellman equation.

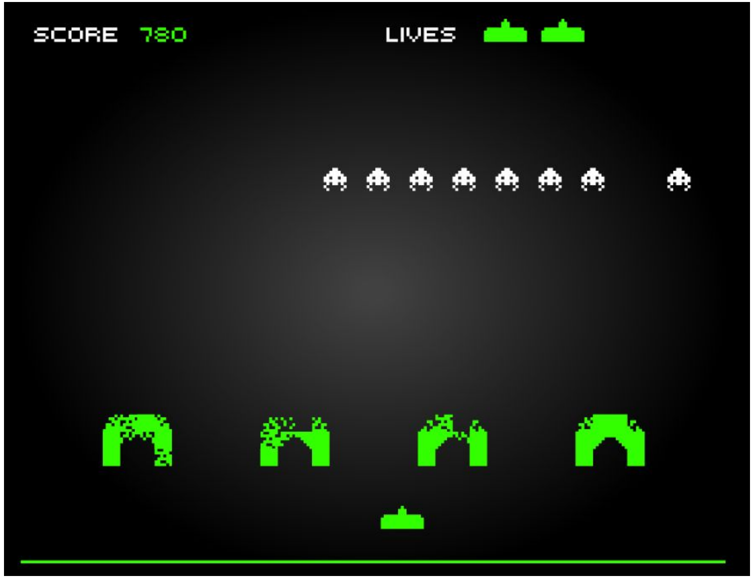
Note also that:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a).$$

S1 =



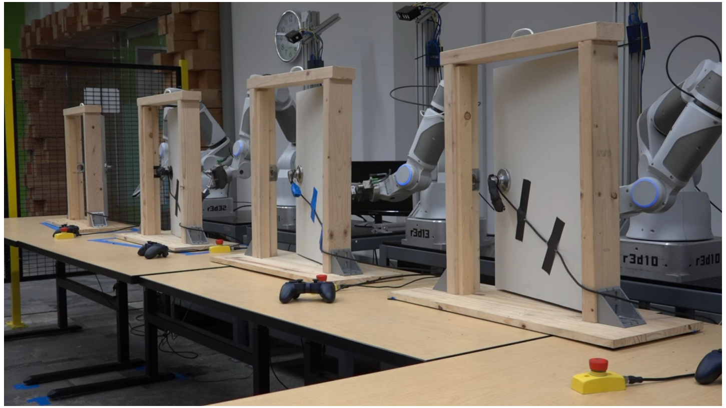
S2 =



Which state has a higher value?

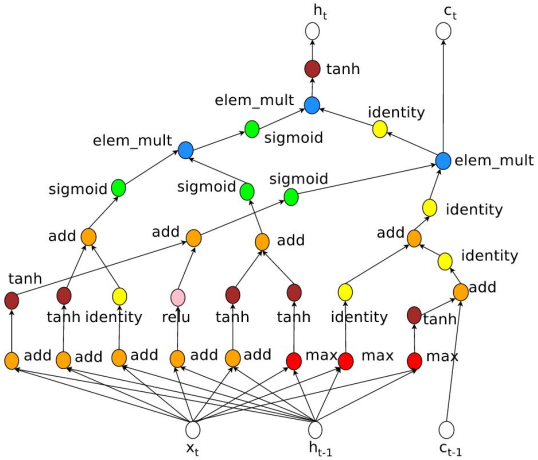
If we play a reasonable policy.

Deep Reinforcement Learning

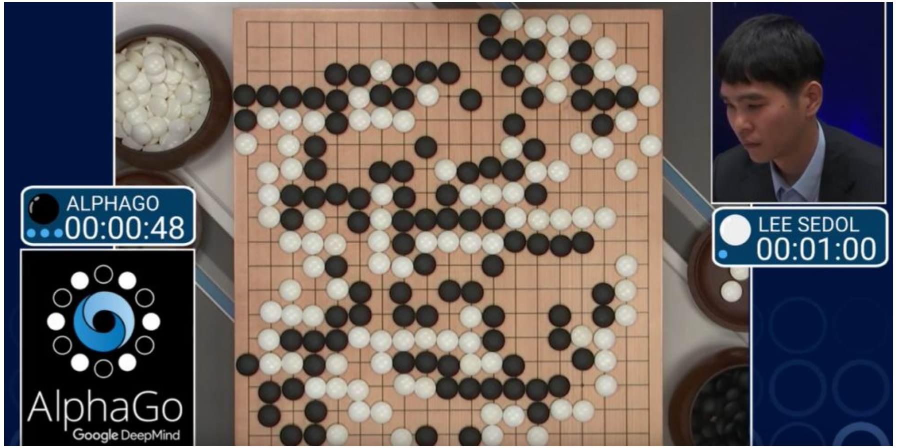


[Collective Robot Reinforcement Learning, Training Phase]

[A Network-based End-to-End Trainable Task-oriented Dialogue System]



[Neural Architecture Search with Reinforcement Learning]



[Mastering the game of Go with deep neural networks and tree search]

Motivation

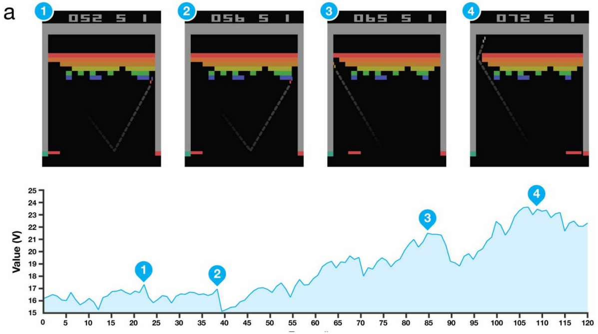
Reinforcement learning prior to 2014:

- handcrafted features, or fully observed, low-dimensional states
- achievements: a super-human Backgammon algorithm, a successful robot soccer agent

[Reinforcement Learning for robot soccer - M.Riedmiller et al. (2009)]

What we want:

- learn directly from high-dimensional sensory input, no feature engineering (end-to-end learning)
- one architecture that can excel at a variety of tasks



[Human-level control through deep reinforcement learning - V.Mnih et al. (2014)]

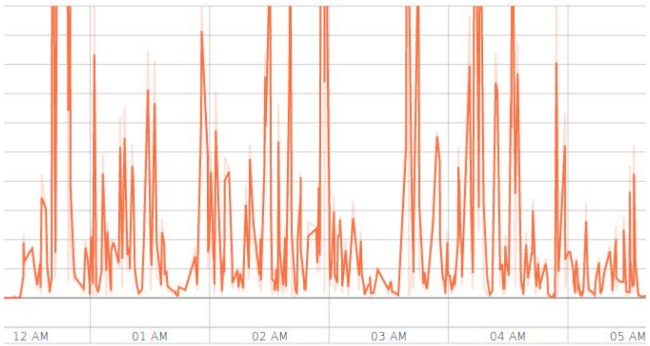
Solution

Deep Learning:

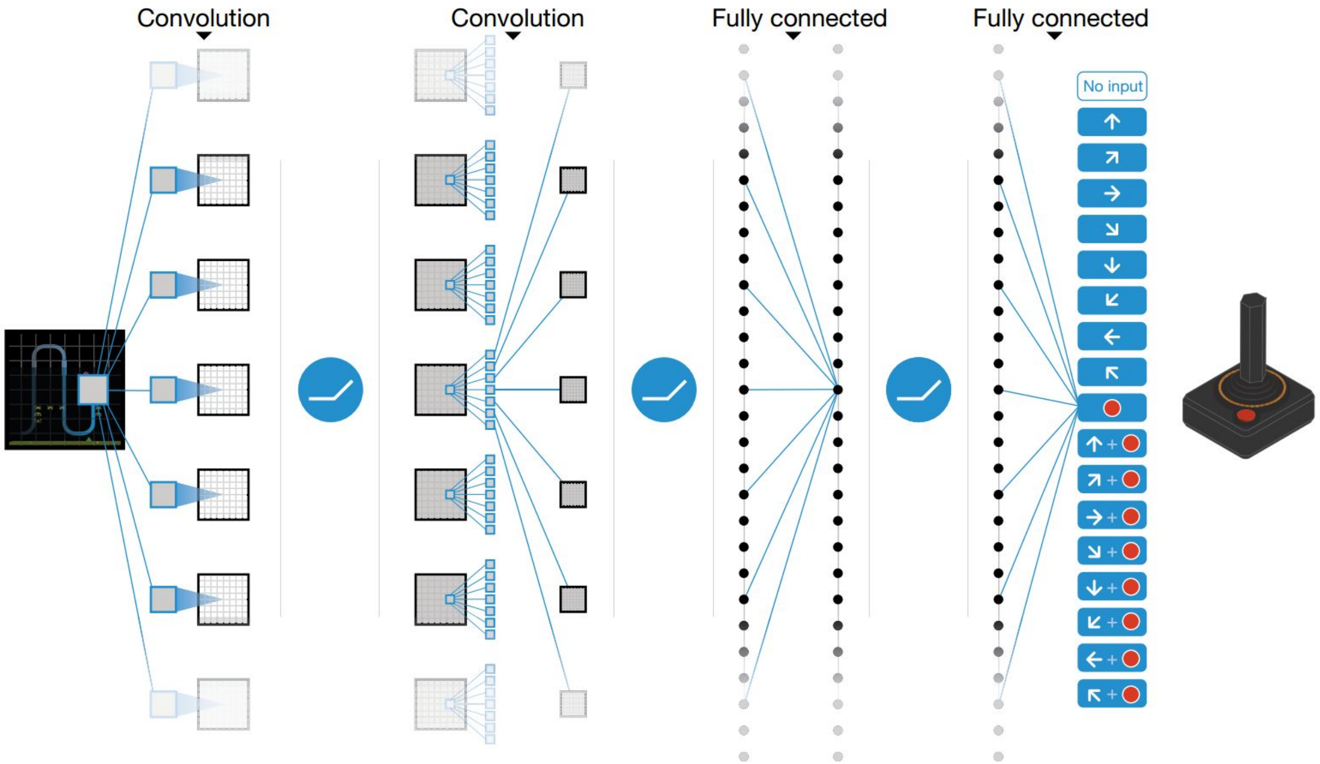
- Deep Neural Networks are capable of processing high-dimensional input and have been successfully applied to a wide range of problems
- we could use a DNN as a approximator for the action-value function (Q function)

Challenges:

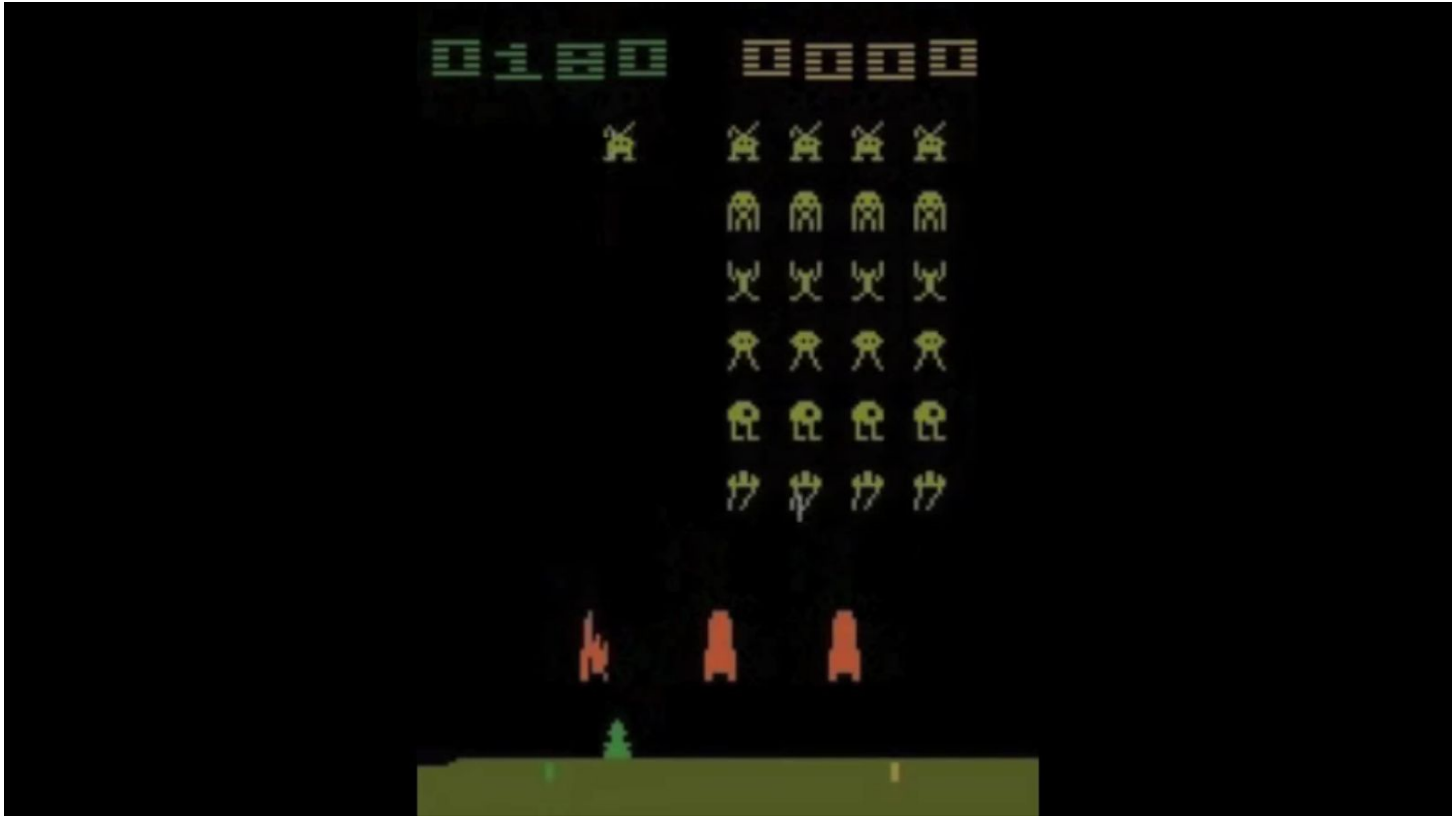
Reinforcement learning is unstable when we use nonlinear function approximators.



Deep Q-Network



DQN Breakout (DeepMind 2016)



DQN Space Invaders (DeepMind 2016)

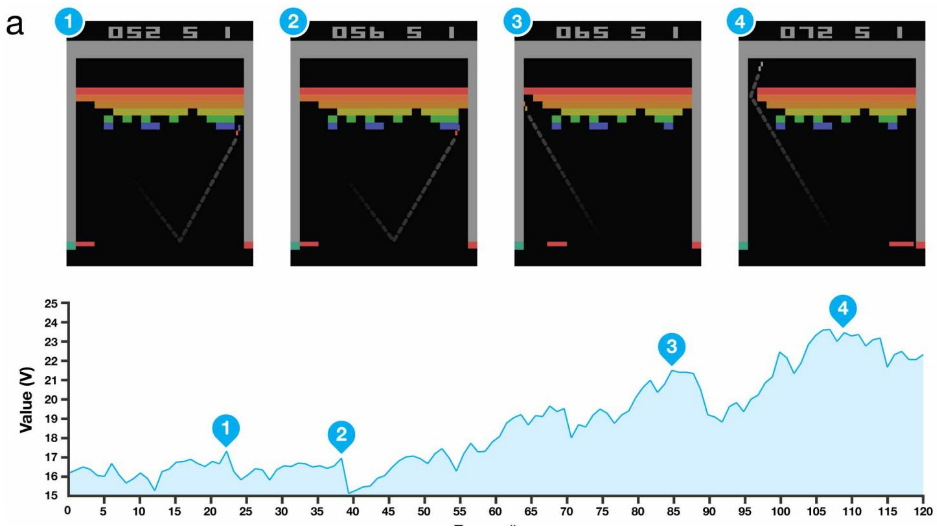
How Deep Q-Network works

- we are modeling the learning task as a Reinforcement Learning problem
- therefore, we are dealing with Markov Decision Processes
- action-value function for an optimal policy

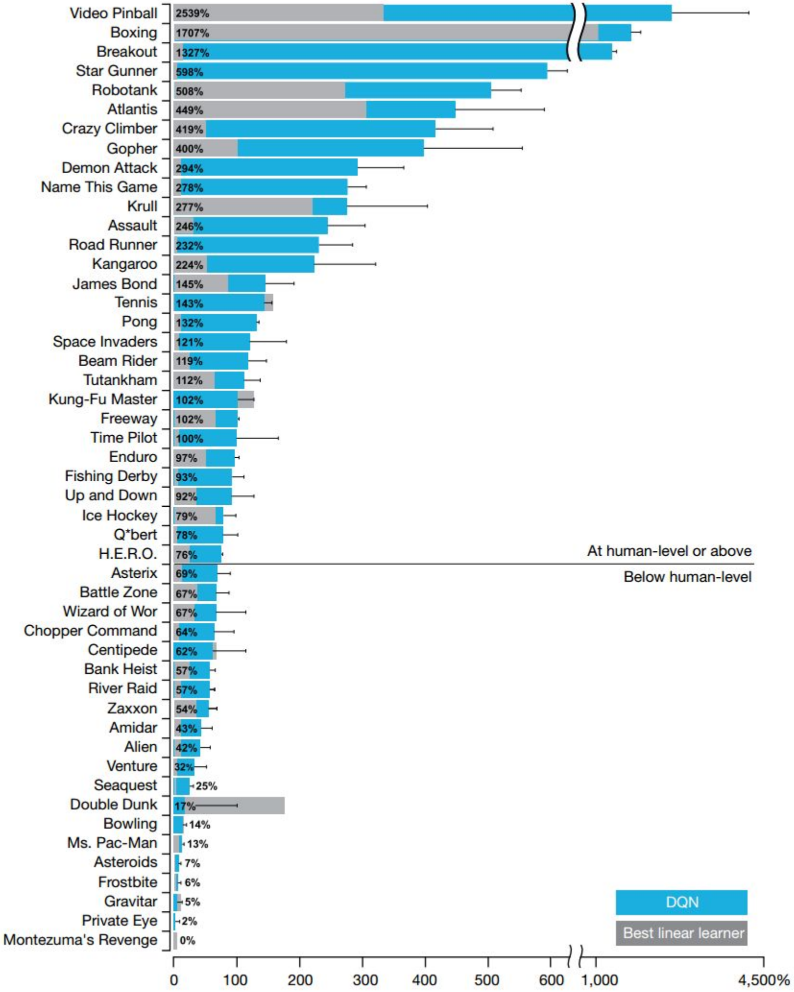
$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s,\ a_t = a,\ \pi\right]$$

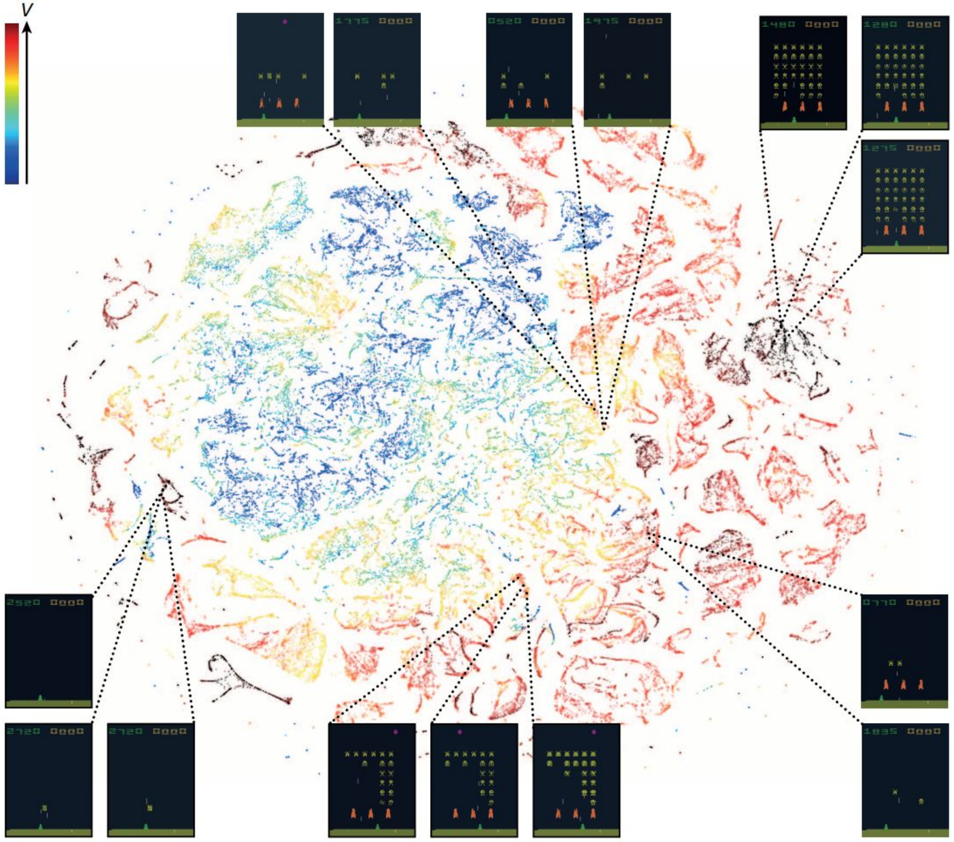
- let's turn it into a loss function for our neural network:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$



[Human-level control through deep reinforcement learning - V.Mnih et al. (2014)]


Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
At human-level or above
Below human-level
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%


V

What about the instability problem?

Two solutions:

1. Experience replay
    - store experience in a replay buffer
    - randomly sample batches of data and train the network on them

2. Target-Value Network
    - target-value network is a clone of our main neural network which we are training
    - use it to calculate the target

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim \mathrm{U}(D)}\left[\left(r + \gamma \max_{a'} \boxed{Q(s',a';\theta_i^-)} - Q(s,a;\theta_i)\right)^2\right]$$

    - create a new clone once a while so that the target-value network is not too different from the trained network

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

Optimal Sepsis Treatment

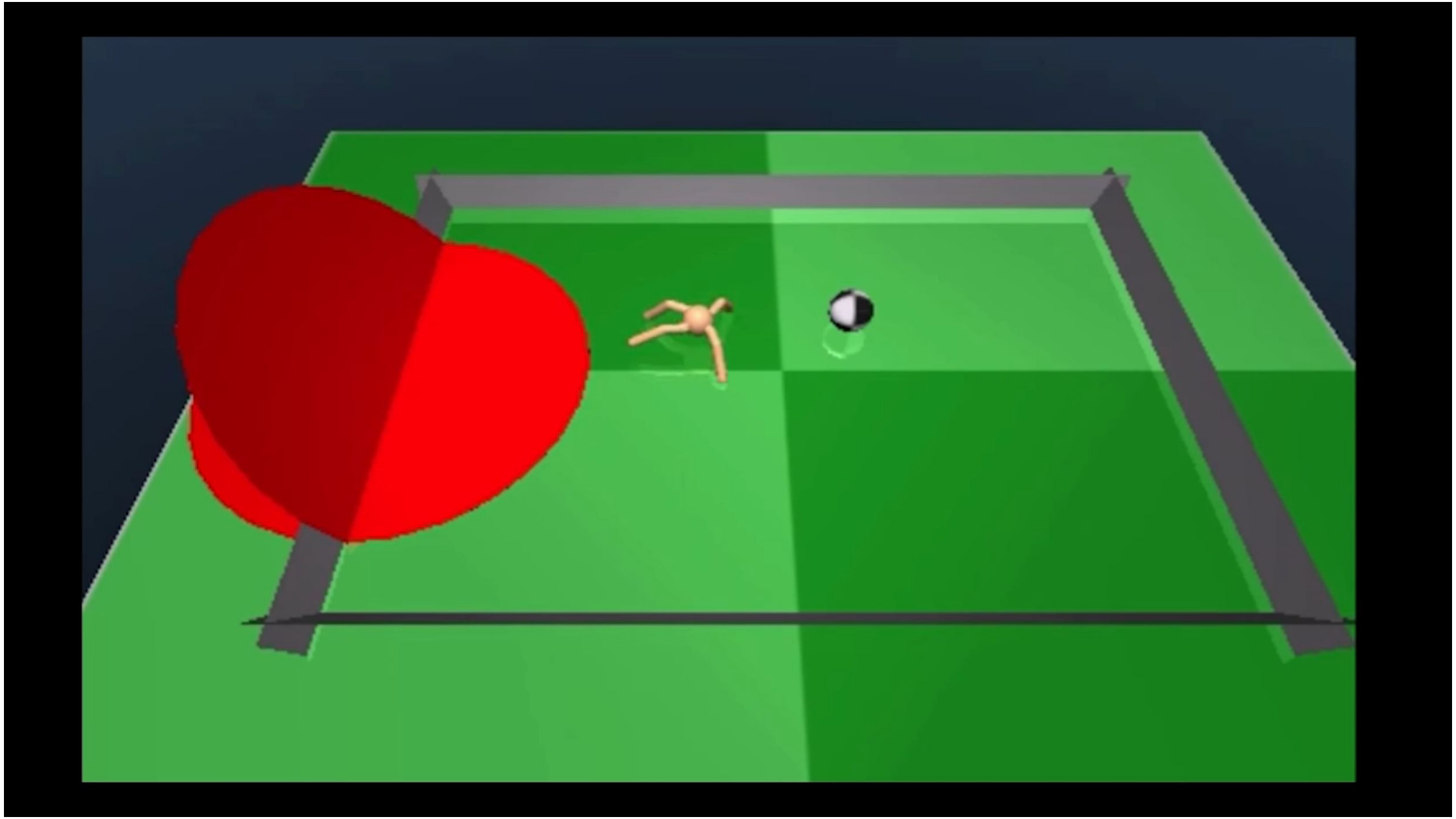
Case study 1

Aniruddh Ragu et al. (2017)

Policy	Expected Return	Estimated Mortality
Physician	9.87	$13.9 \pm 0.5\%$
Normal Q-N	10.16	$12.8 \pm 0.5\%$
Autoencode Q-N	10.73	$11.2 \pm 0.4\%$

Optimal Sepsis Treatment

- training set: 15 500 survivors, 2 300 non-survivors
- physiological parameters including demographics, lab values and vital signs aggregated into windows of 4 hours (47 x 1 state vector for each patient at each timestep)
- 5 x 5 action space covering dosages of two different drugs
- continuous state space, discrete action space
- Deep Q-Network + Double Deep-Q + Dueling Q Network + Prioritized Experience Replay = Dueling Double-Deep Q Network (Dueling DDQN)
- evaluating models offline is difficult



[Ant Soccer (DeepMind 2016)]

Continuous action space

- vanilla Deep Q-Network cannot operate in continuous action space - discretize it?
- on the other hand, we can borrow some tricks introduced with DQN
- use a policy network - an Actor
- best performance when trained together with a Critic => Actor-Critic
- [Continuous Control with Deep Reinforcement Learning - T.P.Lillicrap et al. (2015)]

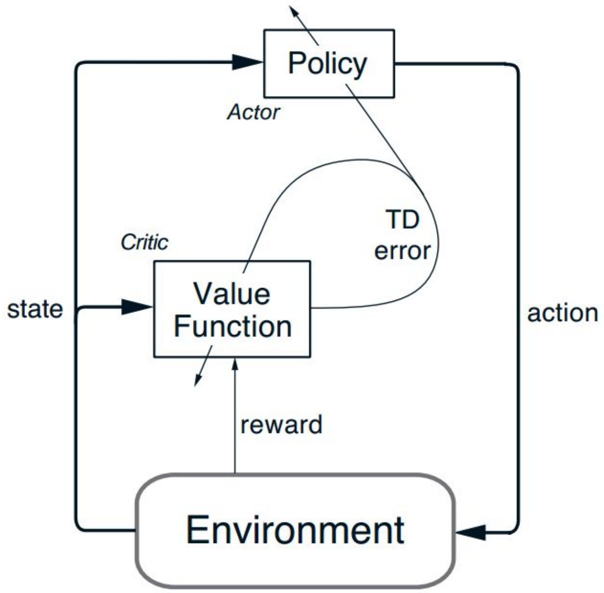


Figure 11.1: The actor–critic architecture.

Deep Deterministic Policy Gradients

Critic loss:

$$L(\theta^Q) = \mathbb{E}_{s_t \sim \rho^\beta, a_t \sim \beta, r_t \sim E}\left[\left(Q(s_t, a_t|\theta^Q) - y_t\right)^2\right]$$

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q).$$

Actor update:

$$\begin{aligned}\nabla_{\theta^\mu} J &\approx \mathbb{E}_{s_t \sim \rho^\beta}\left[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)}\right] \\ &= \mathbb{E}_{s_t \sim \rho^\beta}\left[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta_\mu} \mu(s|\theta^\mu)|_{s=s_t}\right]\end{aligned}$$

- a gradient points in the direction of the steepest ascent of a function
- gradients of the Critic with respect to the action tell us how to the improve policy network's parameters in order to maximize the Q-value

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

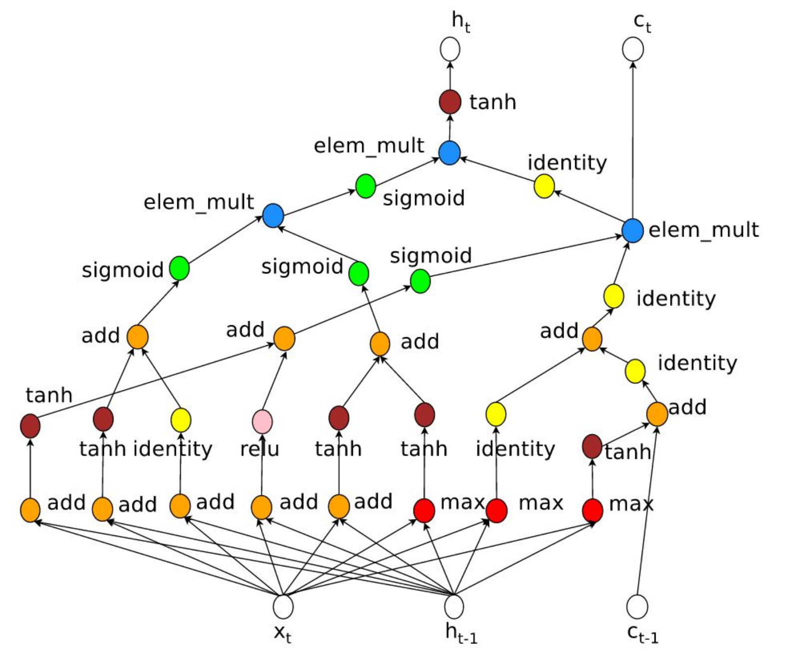
You don't have to understand this algorithm completely. Just the basic idea behind actor-critic.

[DDPG demo]

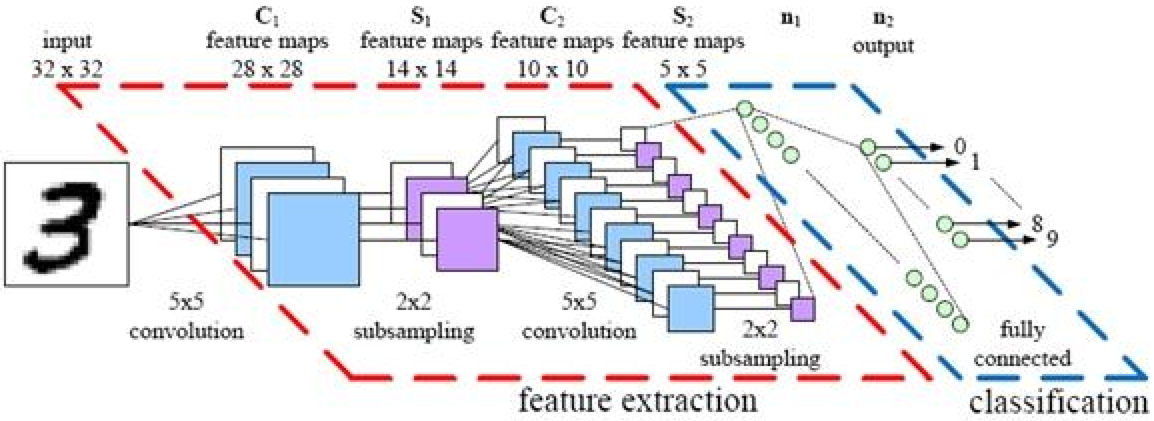
Neural Architecture Search

Case study 2

[Barret Zoph et al. (2017)](#)

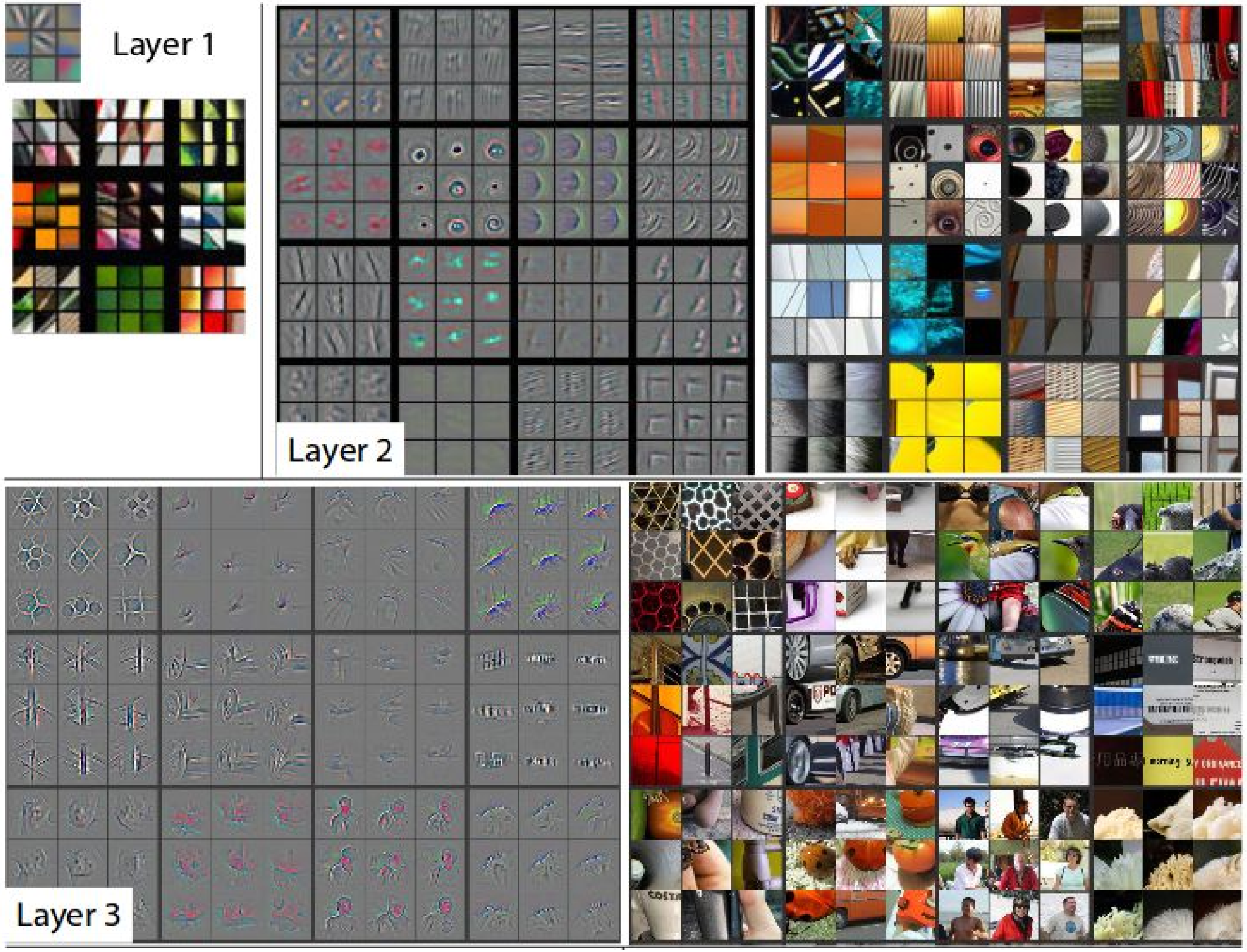


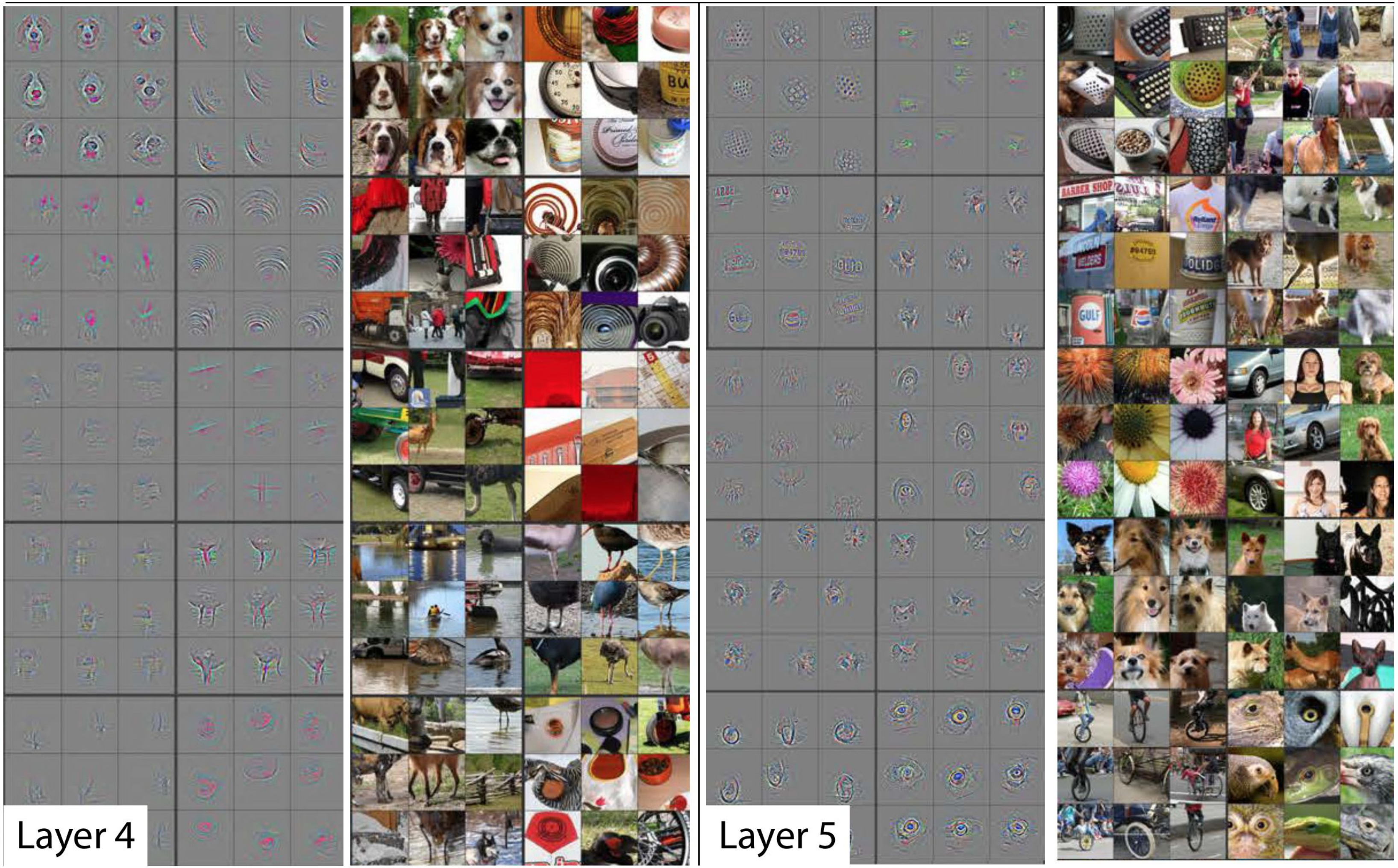
Convolutional Neural Networks



input
32 x 32
C1
feature maps
28 x 28
S1
feature maps
14 x 14
C2
feature maps
10 x 10
S2
feature maps
5 x 5
n1
n2
output
0
1
8
9
5x5
convolution
2x2
subsampling
5x5
convolution
2x2
subsampling
fully
connected
feature extraction
classification

Source


Layer 1
Layer 2
Layer 3


Layer 4
Layer 5

Neural Architecture Search

- train a RNN that generates hyperparameters for neural networks

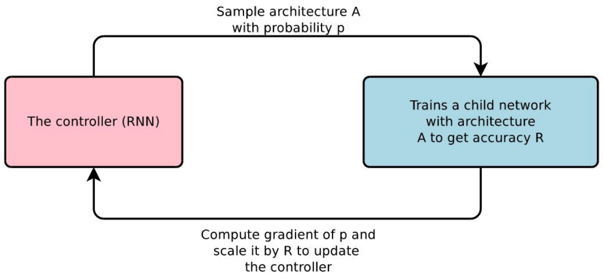


Figure 1: An overview of Neural Architecture Search.

- reward: the test accuracy of the generated network

- update: $\nabla_{\theta_c} J(\theta_c) = \sum_{t=1}^{T} E_{P(a_{1:T};\theta_c)} \big[\nabla_{\theta_c} \log P(a_t | a_{(t-1):1}; \theta_c) R \big]$

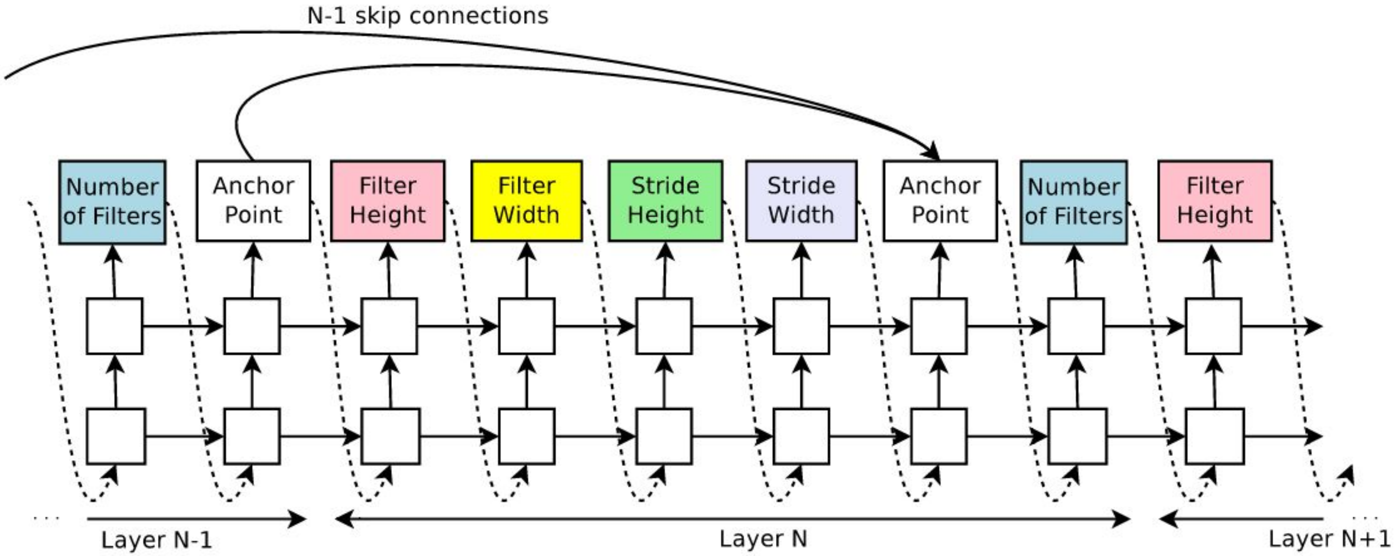


Figure 4: The controller uses anchor points, and set-selection attention to form skip connections.

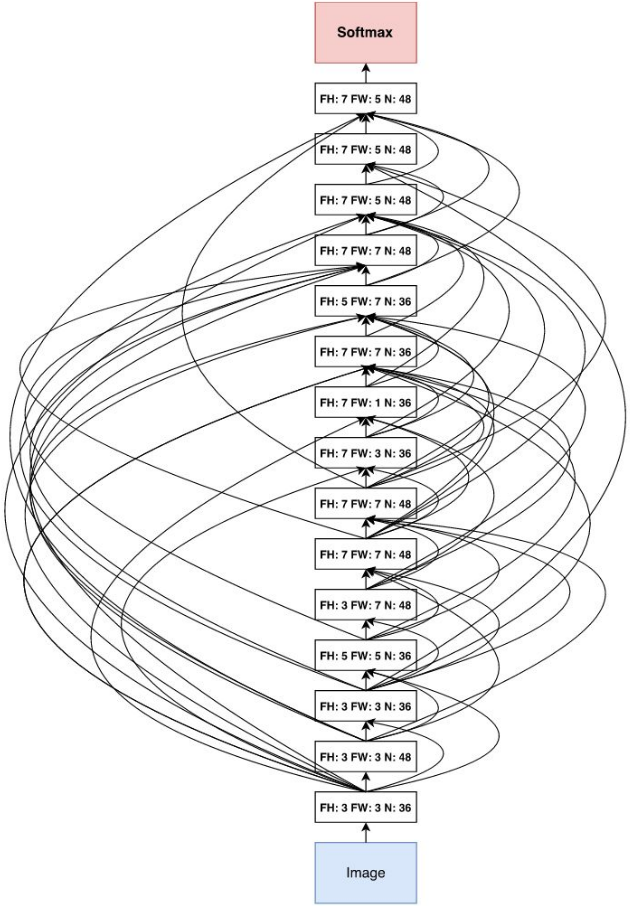


Figure 7: Convolutional architecture discovered by our method, when the search space does not have strides or pooling layers. FH is filter height, FW is filter width and N is number of filters. Note that the skip connections are not residual connections. If one layer has many input layers then all input layers are concatenated in the depth dimension.

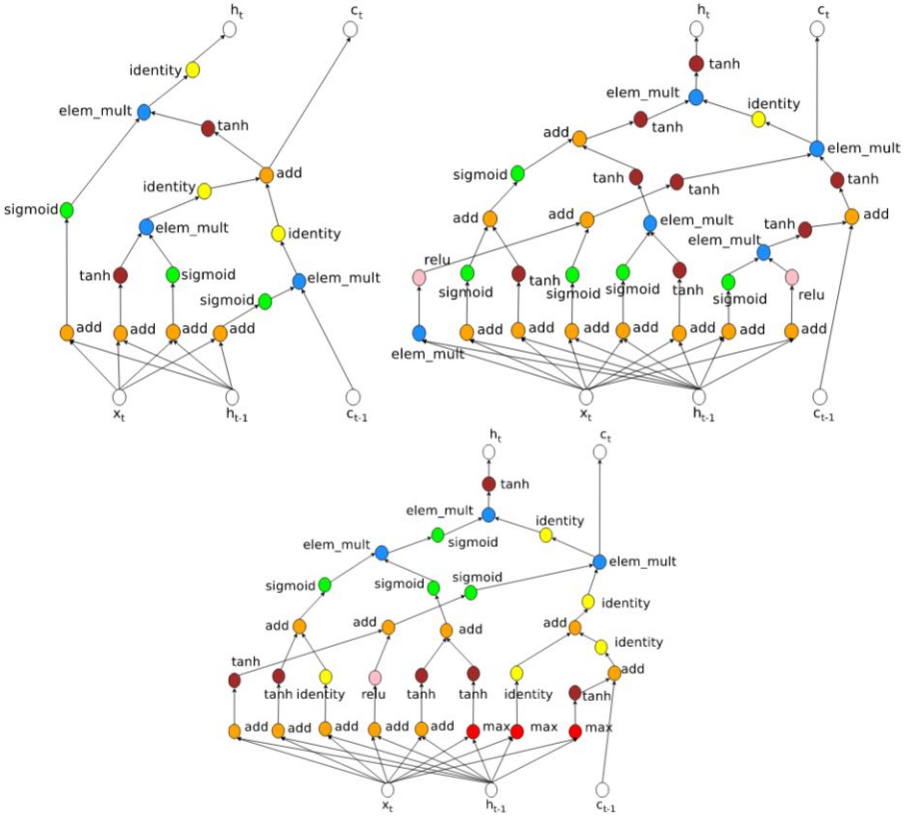


Figure 8: A comparison of the original LSTM cell vs. two good cells our model found. Top left: LSTM cell. Top right: Cell found by our model when the search space does not include max and sin. Bottom: Cell found by our model when the search space includes max and sin (the controller did not choose to use the sin function).

Summary

- we’ve seen how to combine neural networks with reinforcement learning
- Deep Q-Network is one the first architectures that made this combination work
- for continuous states, we can use an Actor or a combination of an Actor and a Critic => Actor-Critic (for example the DDPG algorithm)
- there are many use cases of Deep Reinforcement Learning outside playing video games; on the other hand, video games are a great benchmark

The future of Deep Reinforcement Learning

- we’ve seen a lot of advancements in the Atari games environment but not many applications for real world problems; why?

- move towards general agents that are easy to deploy in many settings
- this could be accomplished by training agents in a modular fashion
- e.g. a module for vision, planning, motion and so on
- similar to how modern software is built