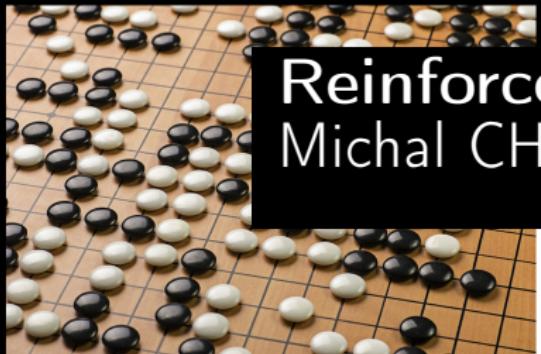


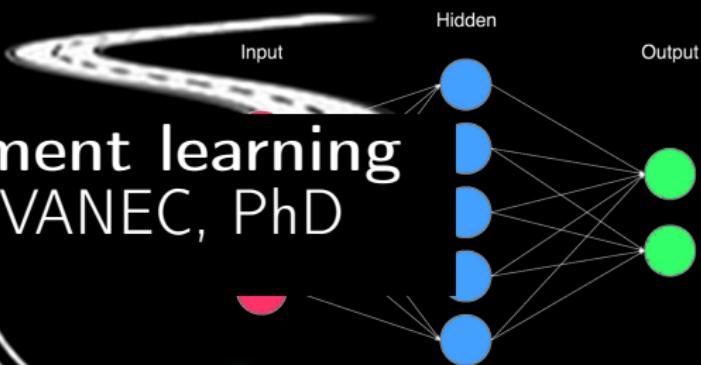
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \lambda \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

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



# Reinforcement learning

Michal CHOVANEC, PhD



# Reinforcement learning

"Master Kihara will listen to noise of fire"



# Reinforcement learning

- learn from punishment and rewards
- learn to play a game with unknown rules



# Reinforcement learning

- ① supervised learning based on human player

$$action = F(observation)$$

$$loss = \left( required\_action - F(observation) \right)^2$$

- ② RL method

$$action = F(observation)$$

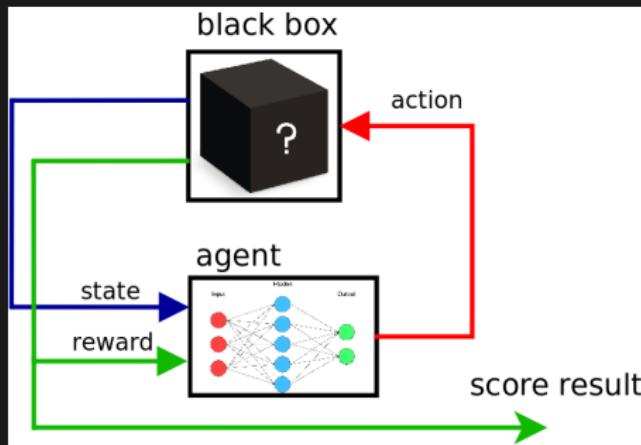
$$loss = \left( required\_action - F(observation) \right)^2$$

**required\_action = ?**

**maximize\_score**

# Reinforcement learning

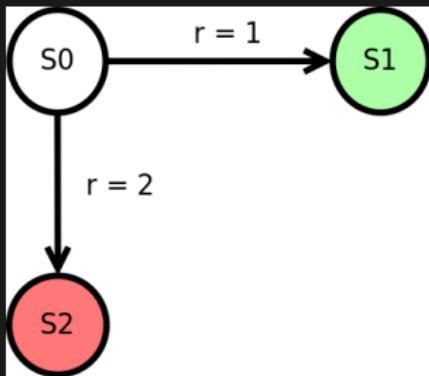
- obtain **state**
- choose **action**
- execute action
- obtain **reward**
- learn from **experiences**



# Making decisions

two possible strategies

- strategy 1 : S0->S1, score = 1.0
- strategy 2 : S0->S2, score = 2.0



$$Q(s, a) = R(s, a)$$

where

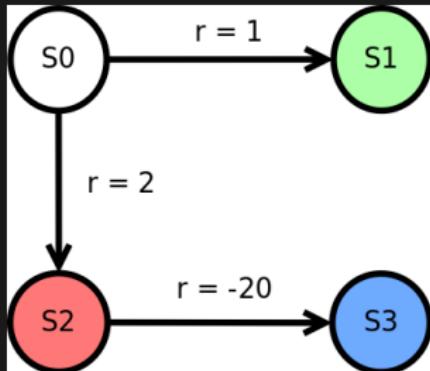
$s$  is state

$a$  is action

# Making decisions

two possible strategies, **greedy = trap**

- strategy 1 : S0->S1, score = 1.0
- strategy 2 : S0->S2->S3, score = 2.0 + (-20.0) = -18

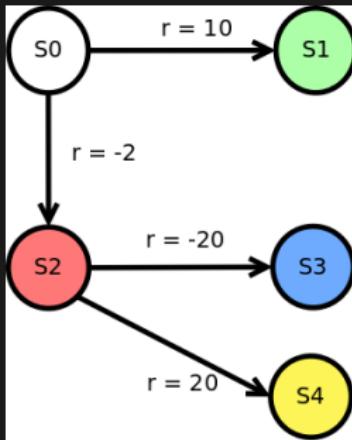


$$Q(s, a) = R(s, a) + \text{FutureReward}$$

# Making decisions

three possible strategies

- strategy 1 : S0->S1, score = 10.0
- strategy 2 : S0->S2->S3, score = -2.0 + (-20.0) = -22
- strategy 3 : S0->S2->S4, score = -2.0 + ( 20.0) = 18



$$Q(s, a) = R(s, a) + \max(\text{FutureRewards})$$

# Q learning

$$Q'(s, a) = R(s, a) + \gamma \max_{a'} Q(s', a')$$

where

$s$  is state

$a$  is action

$s'$  is next state

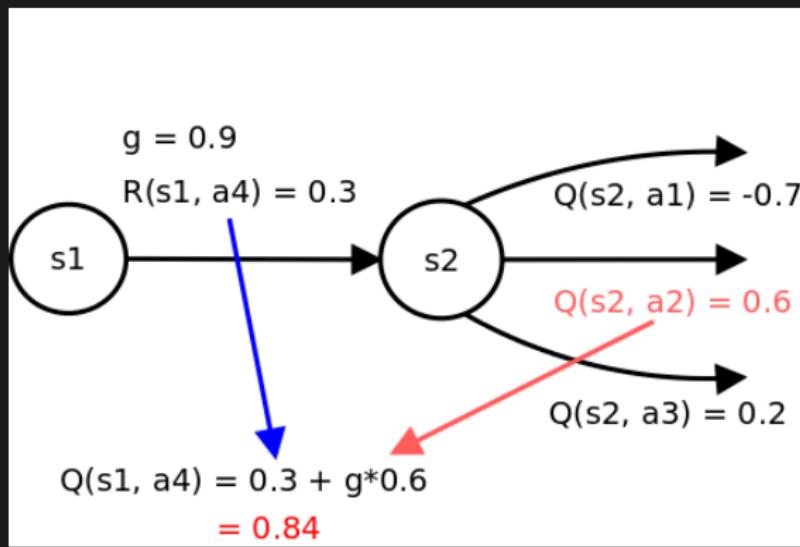
$a'$  is best action in next state

$R(s, a)$  is reward

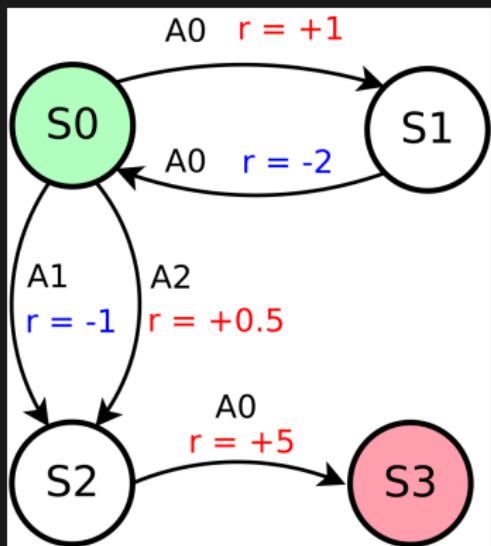
$\gamma \in \langle 0, 1 \rangle$  is discount factor

# Q learning

$$Q'(s, a) = R(s, a) + \gamma \max_{a'} Q(s', a')$$

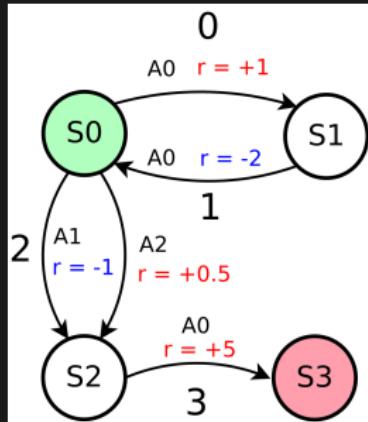


# Q learning - example



S	reward			Q(s,a)		
	A0	A1	A2	A0	A1	A2
0	1	-1	0.5	0	0	0
1	-2	x	x	0	0	0
2	5	x	x	0	0	0
3	x	x	x	0	0	0

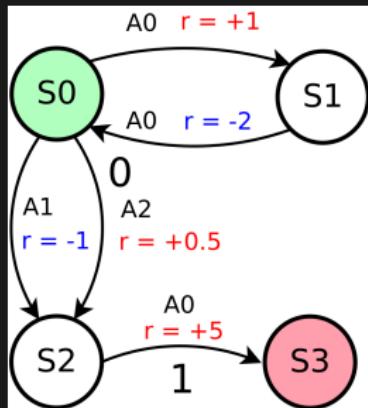
# Q learning - example



S	reward			Q(s,a)		
	A0	A1	A2	A0	A1	A2
0	1	-1	0.5	2.035	3.5	0
1	-2	x	x	1.15	0	0
2	5	x	x	5	0	0
3	x	x	x	0	0	0

state	S0	S1	S0	S2	S3
action	A0	A0	A1	A0	
reward	+1	-2	-1	+5	
Q	$1 + 0.9 * 1.15 =$ 2.035	$-2 + 0.9 * 3.5 =$ 1.15	$-1 + 0.9 * 5 =$ 3.5	5	

# Q learning - example



S	reward			Q(s,a)		
	A0	A1	A2	A0	A1	A2
0	1	-1	0.5	2.035	3.5	5
1	-2	x	x	1.15	0	0
2	5	x	x	5	0	0
3	x	x	x	0	0	0

state	S0	S2	S3
action	A2	A0	
reward	0.5	+5	
Q	$0.5 + 0.9*5 = 5$	5	

# Q learning - stochastic

$$Q'(s, a) = R(s, a) + \gamma \max_{a'} Q(s', a')$$

$$Q'(s, a) = (1 - \alpha)Q(s, a) + \alpha \left( R(s, a) + \gamma \max_{a'} Q(s', a') \right)$$

# Q learning

- obtain reward

```
reward = env.get_reward();
```

- update state

```
state_old = state;
```

```
state = env.get_observation();
```

- select action

```
action_old = action;
```

```
action = select_action(Q(state));
```

- process learning

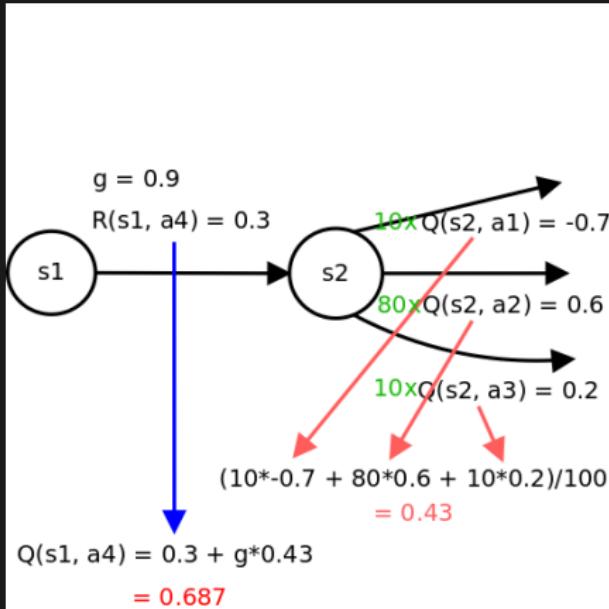
$$Q(\text{state\_old}, \text{action\_old}) += \alpha(\text{reward} + \gamma \max_{a'} Q(\text{state}, a') - Q(\text{state\_old}, \text{action\_old}));$$

- execute action

```
env.action(action);
```

# SARSA learning

$$Q'(s, a) = (1 - \alpha)Q(s, a) + \alpha(R(s, a) + \gamma Q(s', a'))$$



# SARSA learning

- **obtain reward**

```
reward = env.get_reward();
```

- **update state**

```
state_old = state;
```

```
state = env.get_observation();
```

- **select action**

```
action_old = action;
```

```
action = select_action(Q(state));
```

- **process learning**

$$Q(\text{state\_old}, \text{action\_old}) += \alpha(\text{reward} + \gamma Q(\text{state}, \text{action}) - Q(\text{state\_old}, \text{action\_old}))$$

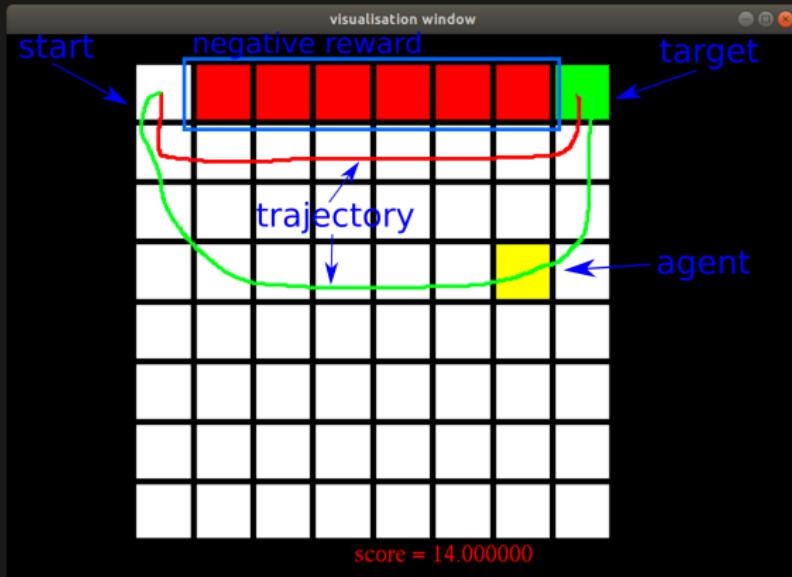
- **execute action**

```
env.action(action);
```

# Q-learning vs SARSA - cliff example

$$Q'(s, a) = (1 - \alpha)Q(s, a) + \alpha \left( R(s, a) + \gamma \max_{a'} Q(s', a') \right)$$

$$Q'(s, a) = (1 - \alpha)Q(s, a) + \alpha \left( R(s, a) + \gamma Q(s', a') \right)$$



# Deep reinforcement learning

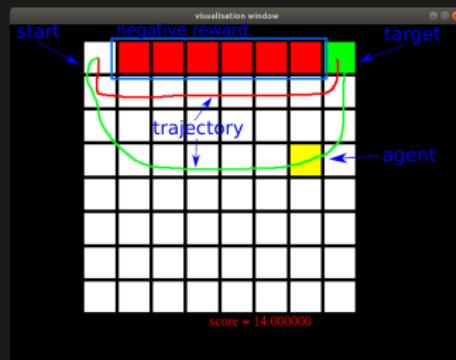
Possible states in some games

- **tic-tac-toe** 26830 states
- **2048** 44096709674720289 states
- **atoms in observable universe**  $10^{82}$
- **chess**  $10^{120}$  states
- **GO**  $10^{170}$  states

storing Q values

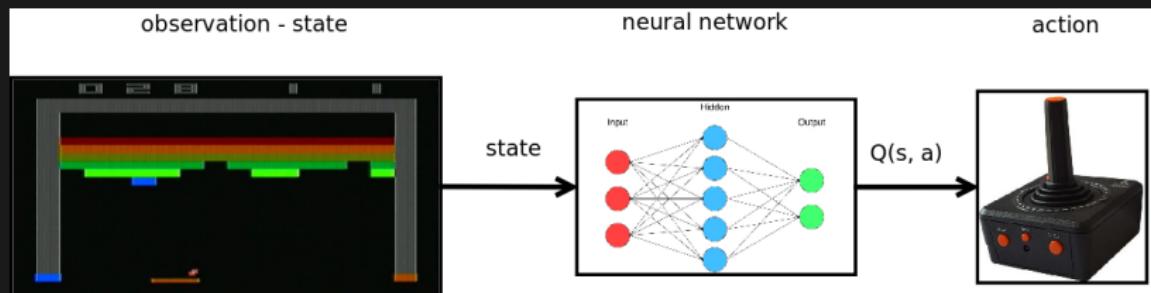
- table
- linear combination of features,  $Q(s, a) = \sum_{i=1}^N w_i F_i(s, a)$
- neural network

# Q table example

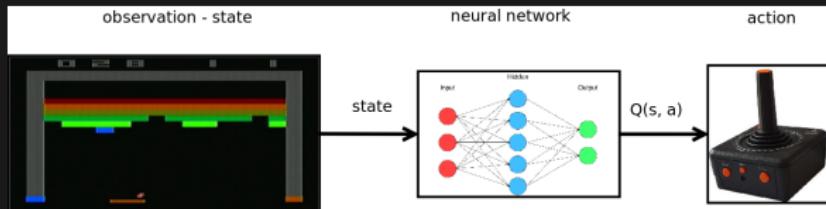


State	A0 UP	A1 LEFT	A2 DOWN	A3 RIGHT
0			0.43	-1
1, 2, 3, 4, 5, 6	T	T	T	T
7	T	T	T	T
8				0.49
9	-1			0.53
10	-1			0.59
11	-1			0.65
12	-1			0.72
13	-1			0.81
14	-1			0.9
15	1			

# Neural network for $Q(s, a)$ approximation - Q networks



# Neural network for $Q(s, a)$ approximation - Q networks



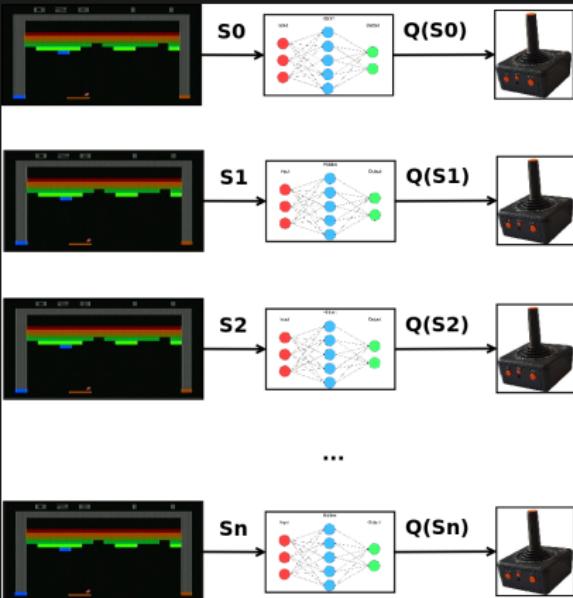
$$SARSA : Q'(s, a) = (1 - \alpha)Q(s, a) + \alpha \left( R(s, a) + \gamma Q(s', a') \right)$$

$$\text{loss} = \sum_{i=1}^N \left( \text{target\_value}_i - \text{resulted\_value}_i \right)^2$$

$$L = \sum_{i=1}^N \left( Q(s_i, a_i) - \hat{Q}(s_i, a_i) \right)^2$$

$$L = \sum_{i=1}^N \left( R(s_i, a_i) + \gamma Q(s'_i, a'_i) - \hat{Q}(s_i, a_i) \right)^2$$

# Training network



$$\Delta \hat{Q}(s_0, a_0) = \alpha \left( R(s_0, a_0) + \gamma \hat{Q}(s_1, a_1) \right)$$

$$\Delta \hat{Q}(s_1, a_1) = \alpha \left( R(s_1, a_1) + \gamma \hat{Q}(s_2, a_2) \right)$$

$$\Delta \hat{Q}(s_2, a_2) = \alpha \left( R(s_2, a_2) + \gamma \hat{Q}(s_3, a_3) \right)$$

...

$$\Delta \hat{Q}(s_n, a_n) = \alpha \left( R(s_n, a_n) + \gamma \hat{Q}(s_{n+1}, a_{n+1}) \right)$$

# Training network

- ① compute temporary  $Q_t$  batch

$$Q_t(s_0, a_0) = R(s_0, a_0) + \gamma Q_t(s_1, a_1)$$

$$Q_t(s_1, a_1) = R(s_1, a_1) + \gamma Q_t(s_2, a_2)$$

$$Q_t(s_2, a_2) = R(s_2, a_2) + \gamma Q_t(s_3, a_3)$$

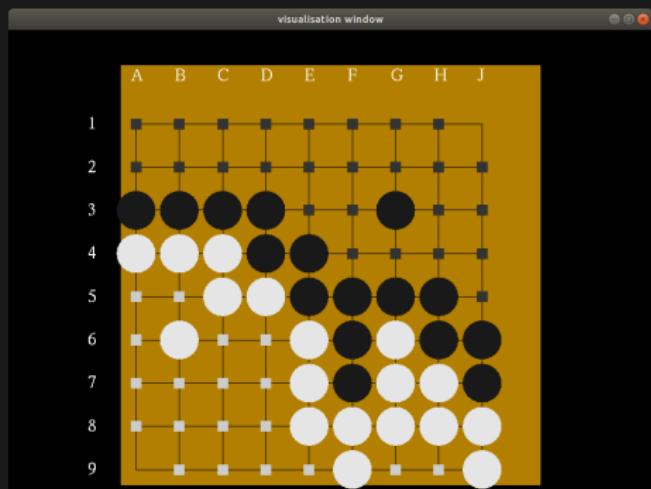
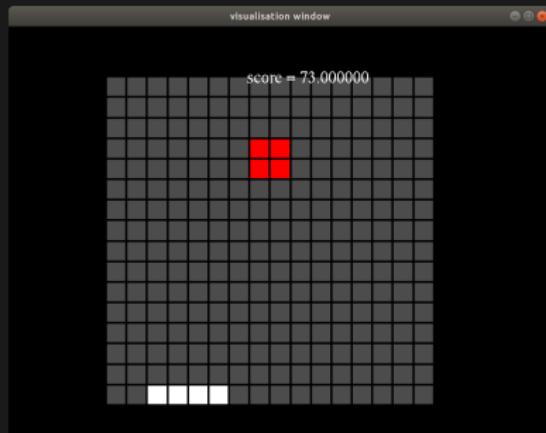
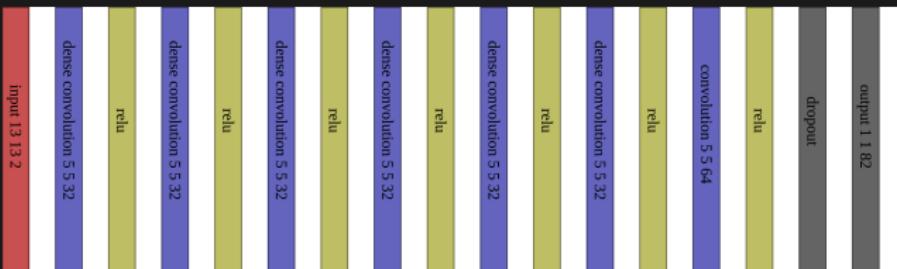
...

$$Q_t(s_n, a_n) = R(s_{n+1}, a_{n+1})$$

- ② train neural network using

$$Q_{new}(s_i, a_i) = (1 - \alpha)Q_{old}(s_i, a_i) + \alpha Q_t(s_i, a_i)$$

## Examples



# Usefull links

-  **Andrej Karpathy : Pong from pixels**  
<http://karpathy.github.io/2016/05/31/r1/>
-  **Richard S. Sutton : Reinforcement Learning: An Introduction**  
https://www.amazon.com/Reinforcement-Learning-Introduction-Adaptive-Computation/dp/0262193981
-  **Maxim Lapan : Deep reinforcement learning**  
<https://www.amazon.com/Practical-Reinforcement-Learning-Maxim-Lapan/dp/1788834240>
-  **Mohit Sewak : Practical Convolutional Neural Networks**  
<https://www.amazon.com/Practical-Convolutional-Neural-Networks-Implement/dp/1788392302>
-  **CHRISTOPHER J.C.H. WATKINS : Q-learning**  
<http://www.gatsby.ucl.ac.uk/~dayan/papers/cjch.pdf>
-  **Densely Connected Convolutional Networks**  
<https://arxiv.org/pdf/1608.06993.pdf>
-  **Mastering the Game of Go without Human Knowledge**  
[https://deepmind.com/documents/119/agz\\_unformatted\\_nature.pdf](https://deepmind.com/documents/119/agz_unformatted_nature.pdf)

# Q&A



michal chovanec (michal.nand@gmail.com)  
[www.youtube.com/channel/UCzVvP2ou8v3afNiVrPAHQGg](https://www.youtube.com/channel/UCzVvP2ou8v3afNiVrPAHQGg)  
github <https://github.com/michalnand>