# Deep Reinforcement Learning

**- Learning like a baby rather than like a copier -**

Eric Steinberger

# Deep RL via Policy Gradient

$$\nabla_\theta \mathbb{E}_\tau[R] \approx \mathbb{E}_\tau \left[ \sum_{t=0}^{T-1} \nabla_\theta \log(\pi(a_t|s_t,\theta)) A_t \right]$$

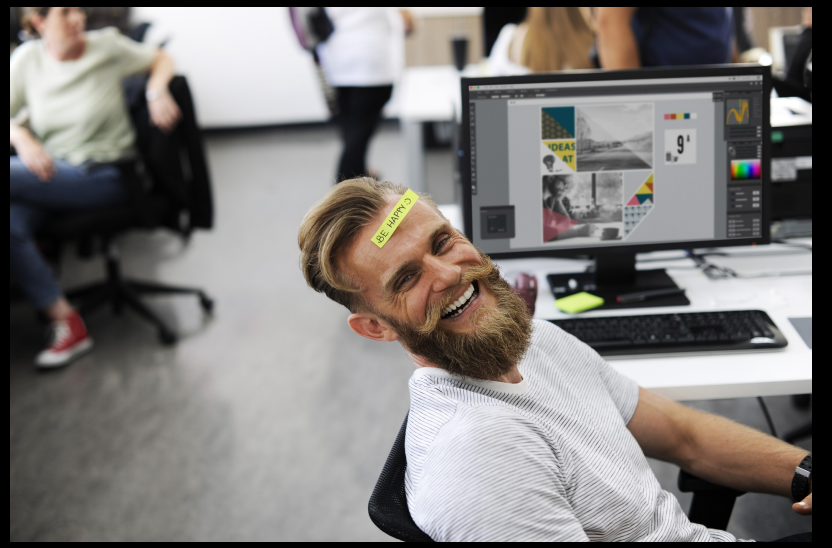
Just kidding… That comes later.

# The Supervised Learning Problem

→ ? → **it‘s a 9**

# The Reinforcement Learning Problem

→ ? →



# The Ethical Question (i.e. Reward Function)

→ ? → ???

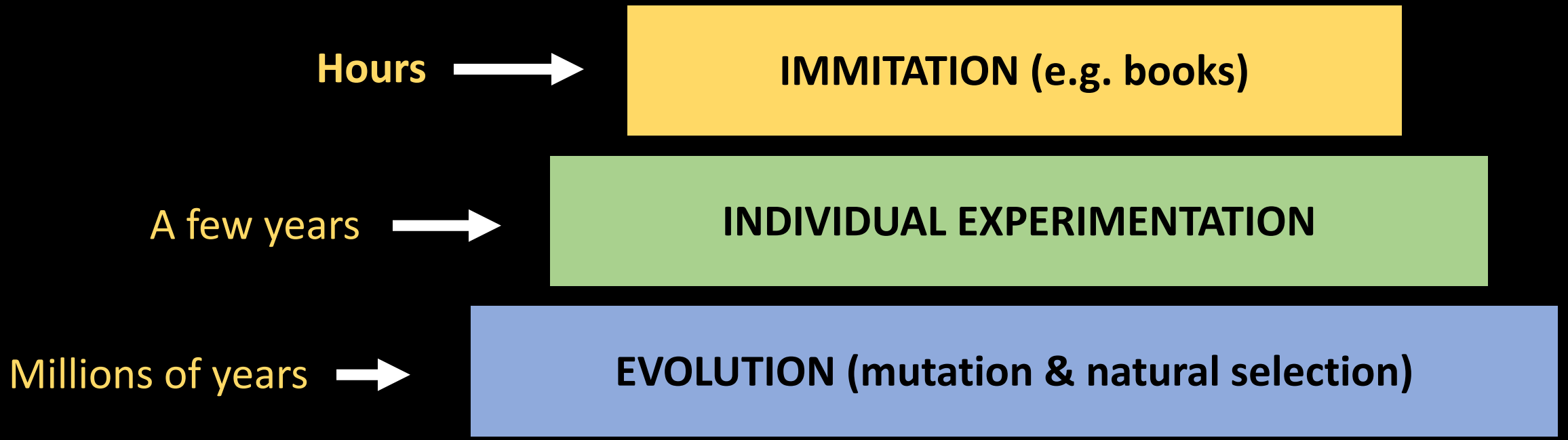
# Life Is An Optimization Problem. How Are We Trying To Solve It?

**IMMITATION (e.g. books)**

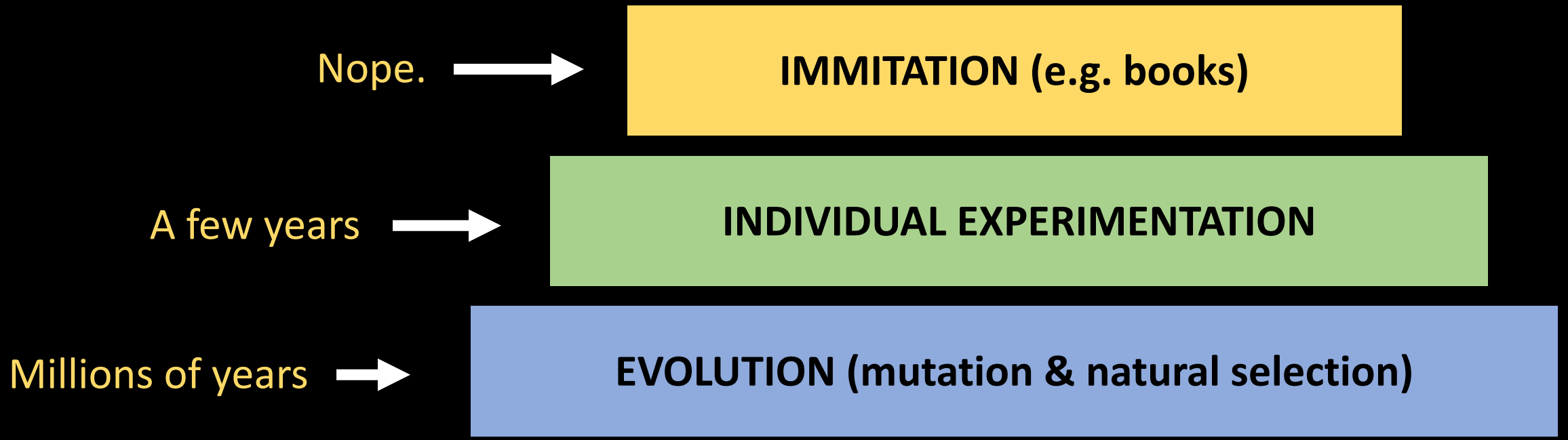
**INDIVIDUAL EXPERIMENTATION**

**EVOLUTION (mutation & natural selection)**

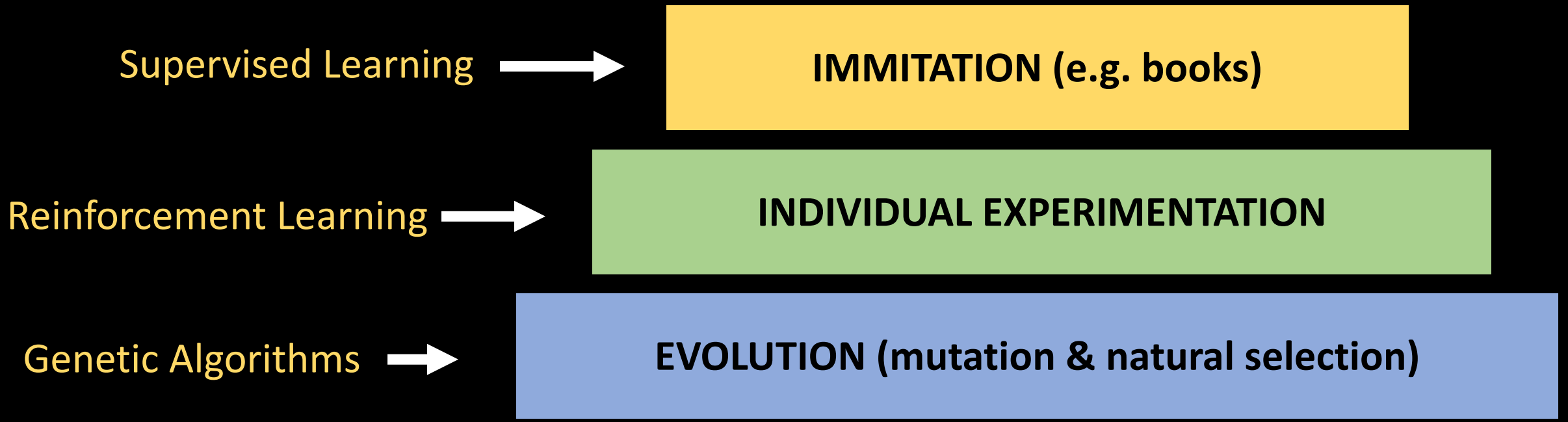
# How Long To Learn/Improve A Lot?

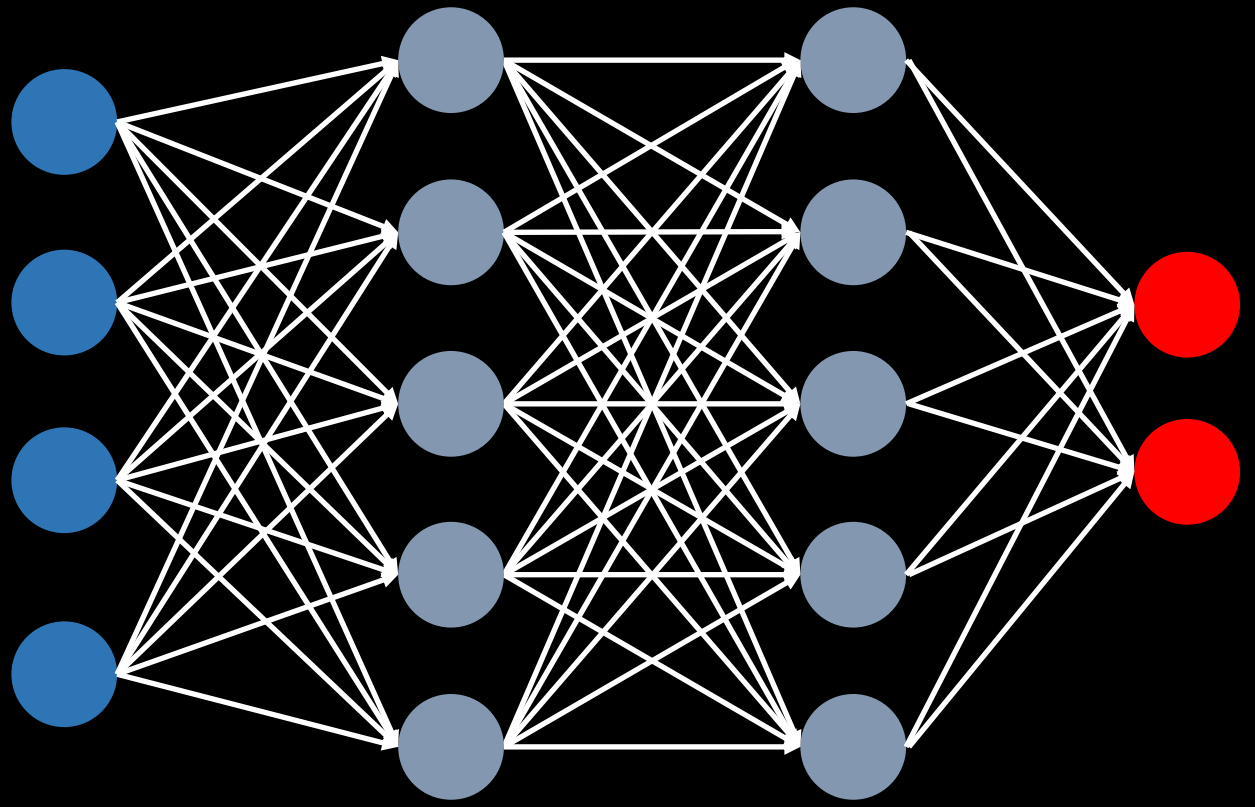


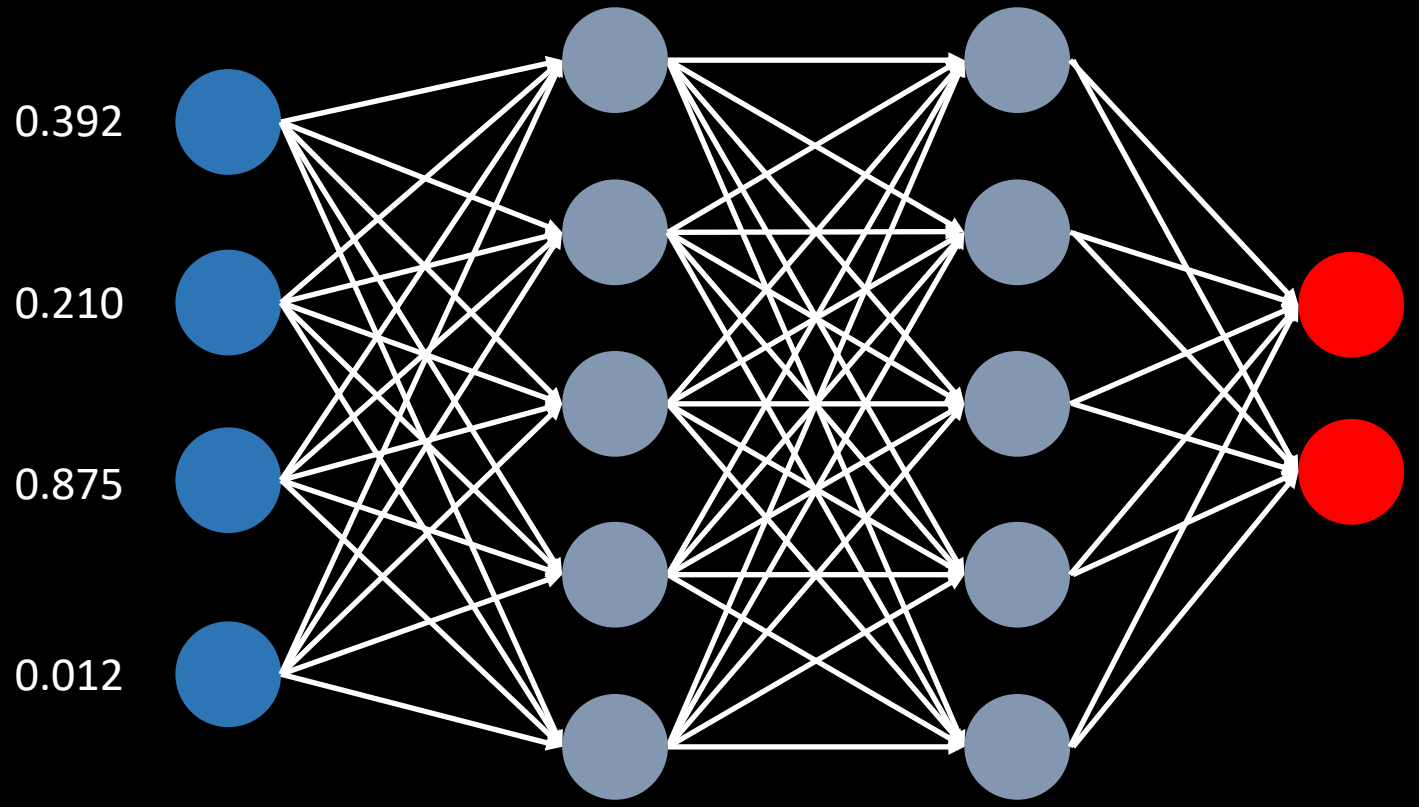
# How Long To Learn Something NEW very well?

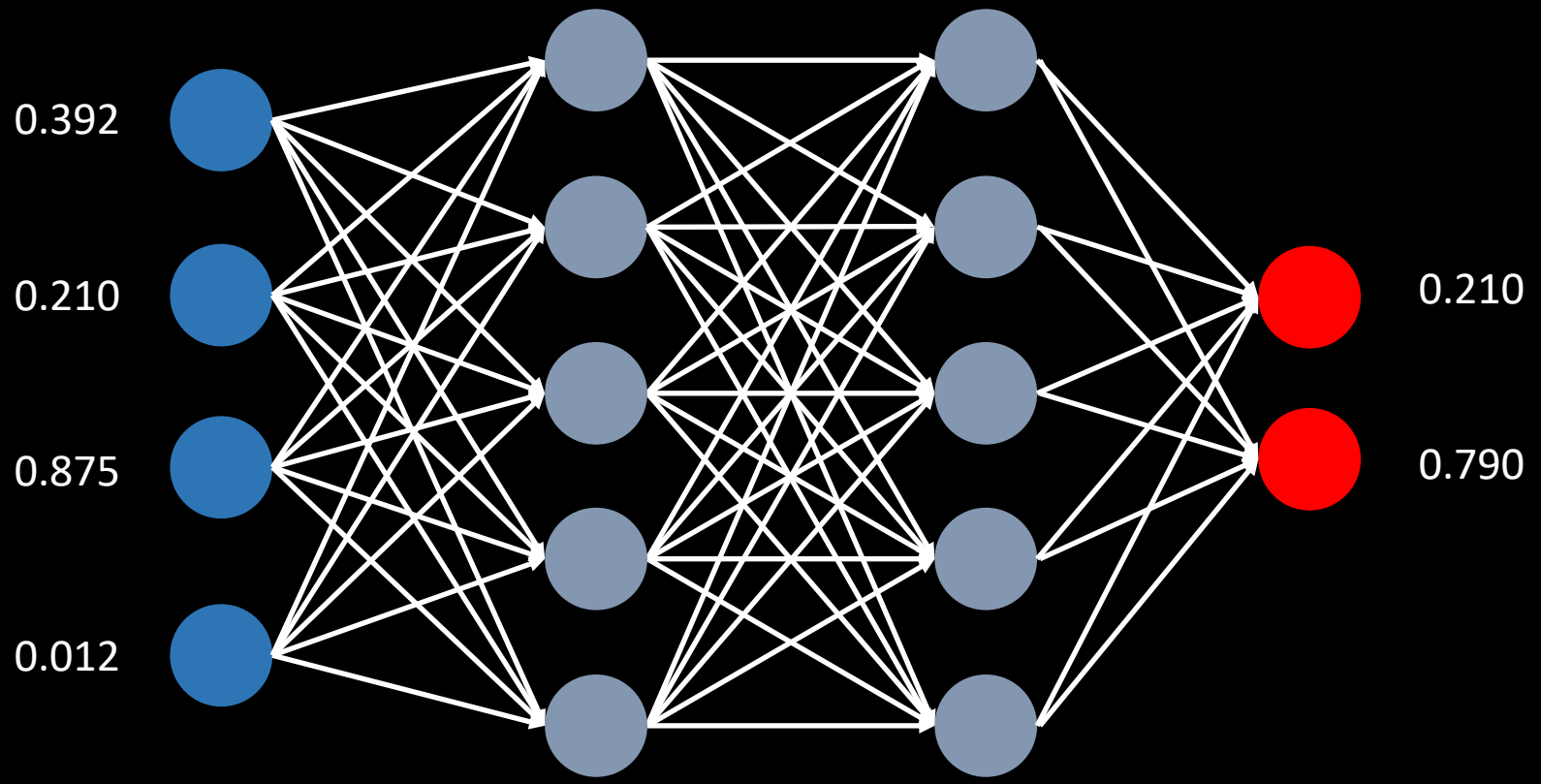


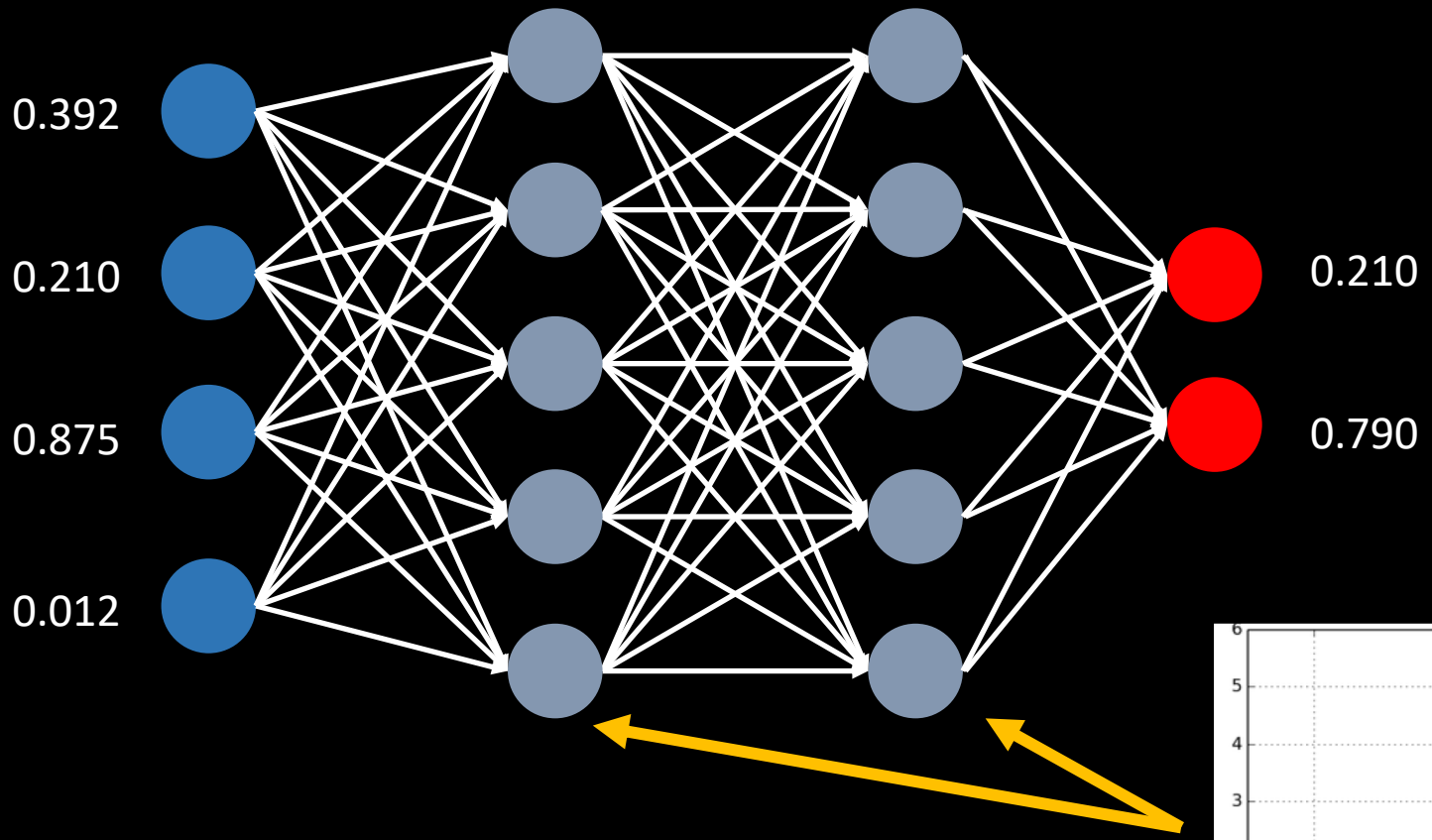
# AI Terms

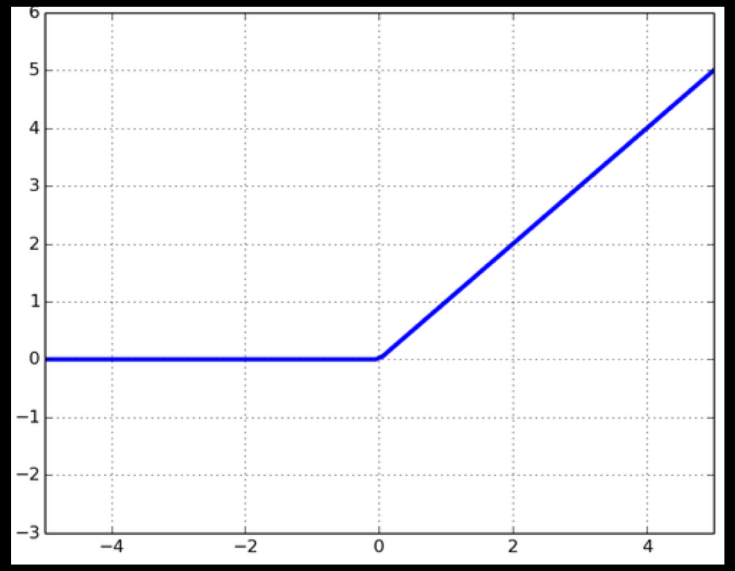

Supervised Learning
IMMITATION (e.g. books)
Reinforcement Learning
INDIVIDUAL EXPERIMENTATION
Genetic Algorithms
EVOLUTION (mutation & natural selection)

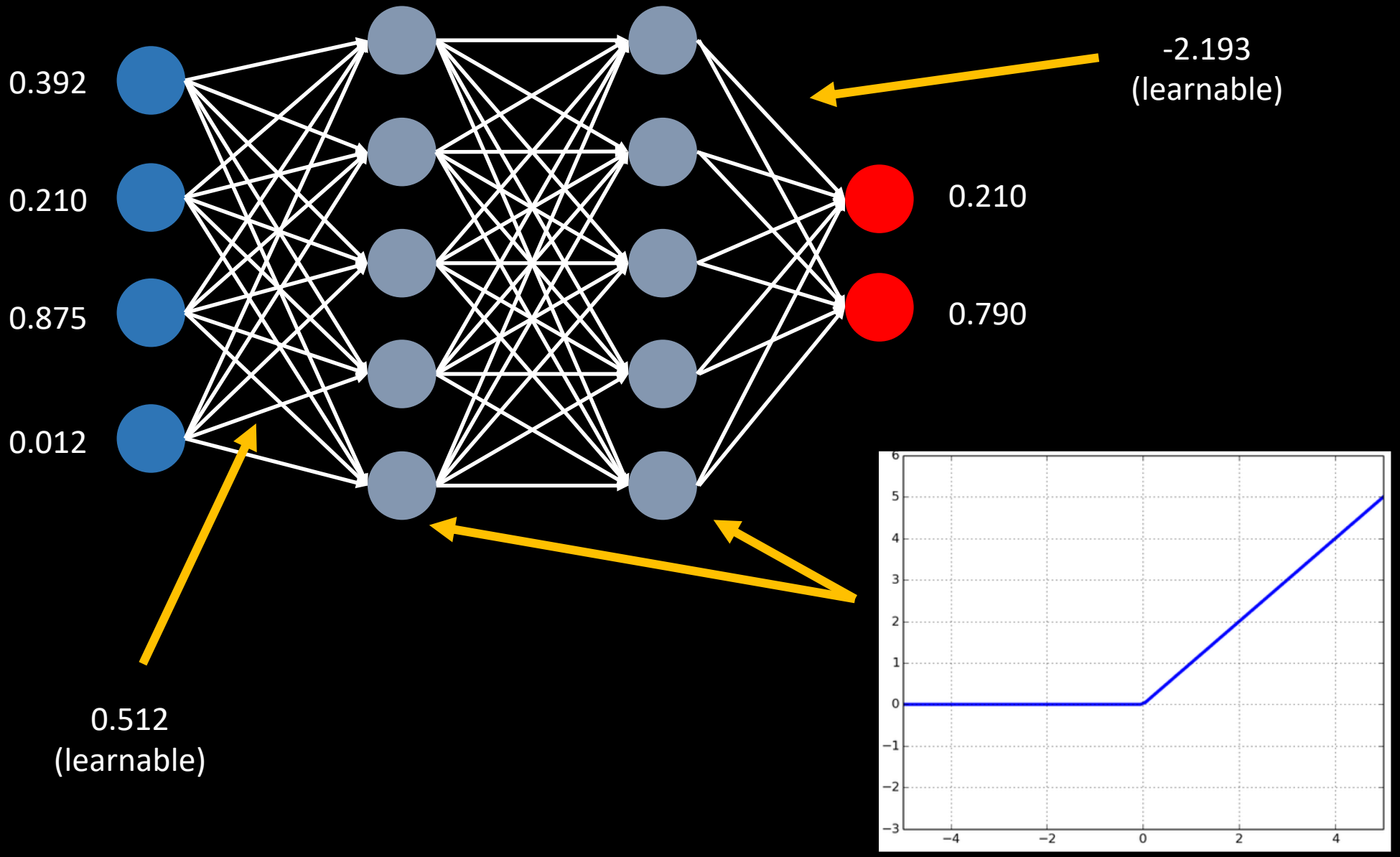



0.392
0.210
0.875
0.012

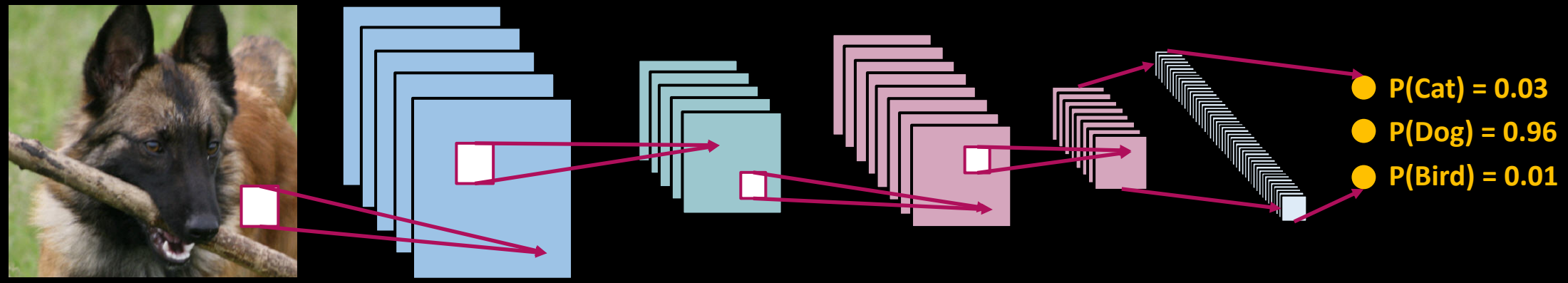

0.392
0.210
0.875
0.012
0.210
0.790


0.392
0.210
0.875
0.012
0.210
0.790

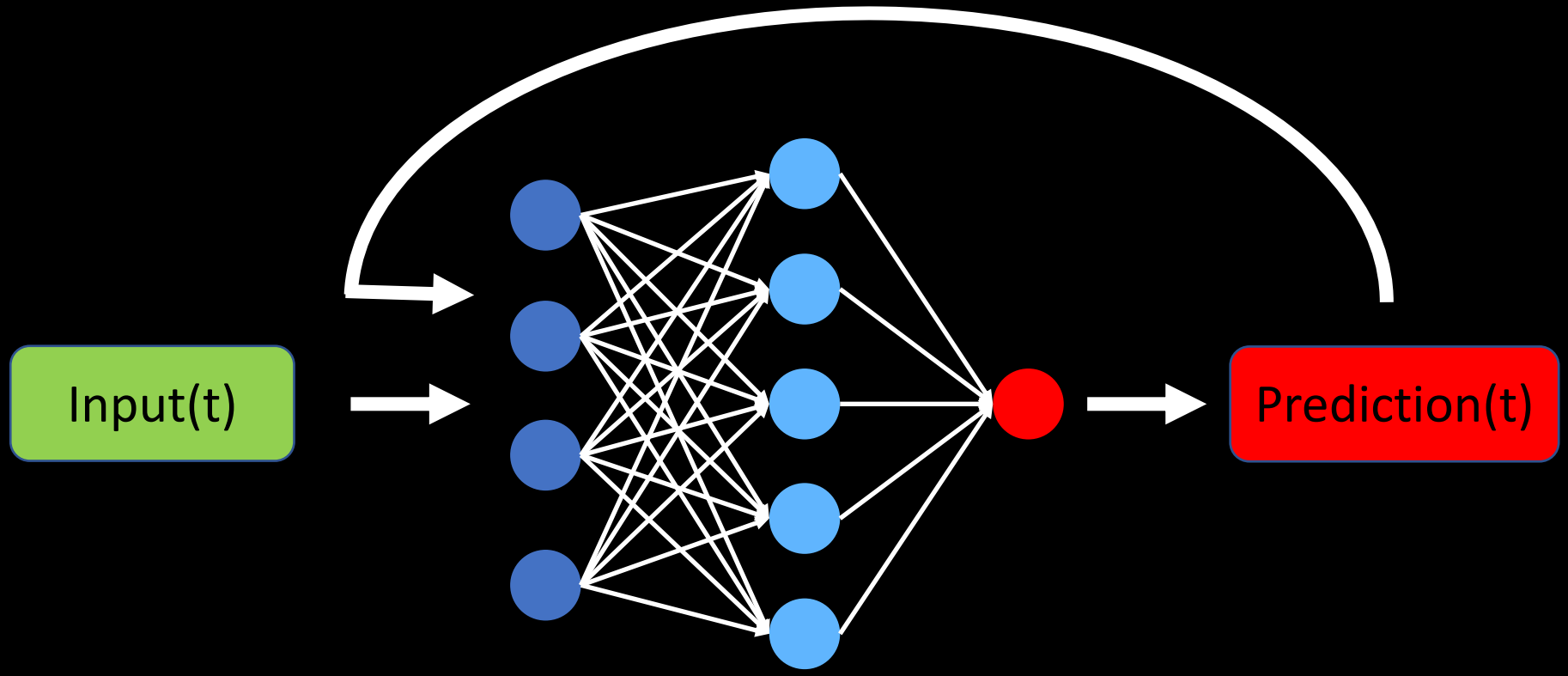

6
5
4
3
2
1
0
−1
−2
−3
−4
−2
0
2
4

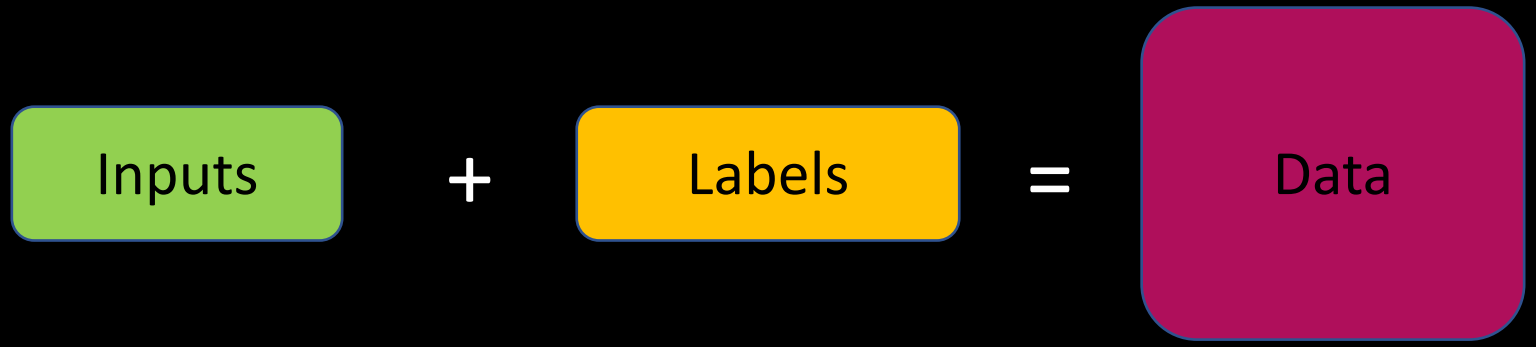

0.392
0.210
0.875
0.012
-2.193
(learnable)
0.210
0.790
0.512
(learnable)

# Convolutional Neural Networks (CNNs)



# Recurrent Neural Networks (RNNs)




Inputs
+
Labels
=
Data

# Reinforcement Learning (RL)

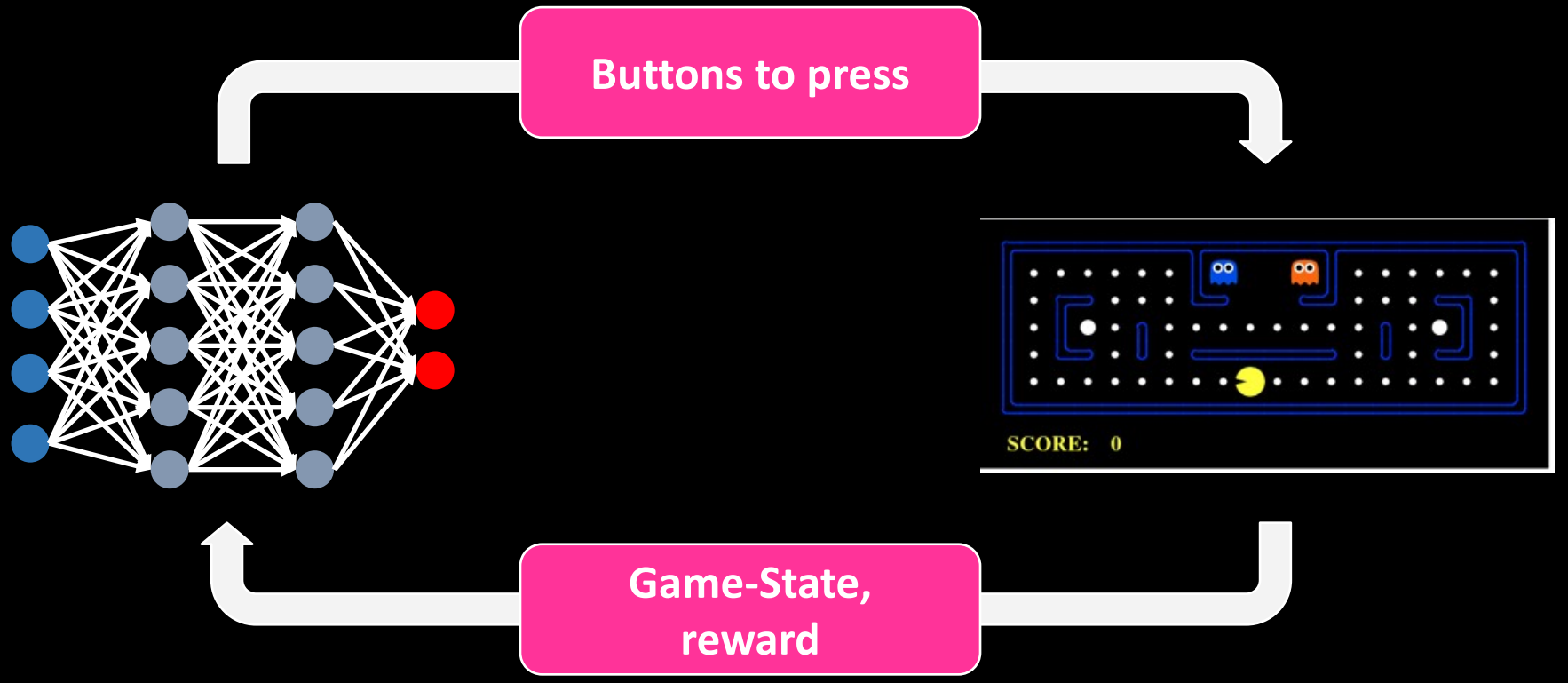



Image PacMan: http://ai.berkeley.edu

# But… How?

# What is a game? (Game Theory)

# Is The Agent Alone?

- single- / multi-agent
  - Single-agent: Pacman (1 vs. Game)
  - Multi-agent: Chess (1 vs. 1); all players learn

# Environment Format

- continuous / discrete observations
  - Continuous: real world, Starcraft II, Dota 2
  - Discrete: Chess, Go
- continuous / discrete action-space
  - Continuous: Steering wheel angle in racing game
  - Discrete: Chess

# What Can The Agent Actually See?

- Perfect / imperfect information games
  - Perfect information: Chess
  - Imperfect information: Battleships, Poker, financial market
- Perfect information problems:
  - The optimal policy is always deterministic
- Imperfect information problems
  - The optimal policy is often stochastic

# Major Types Of RL:

- Value-Function based (i.e. predict reward)
- Policy Gradient (i.e. do highly rewarding actions more often)
- Their combination: Actor-Critic

# Definitions

- s ... State at time t
- s‘ … State at time t+1
- a … action
- r(s) … the reward we empircally got in just that timestep
- R(s) … sum of expected rewards from state *s* until the end of the game

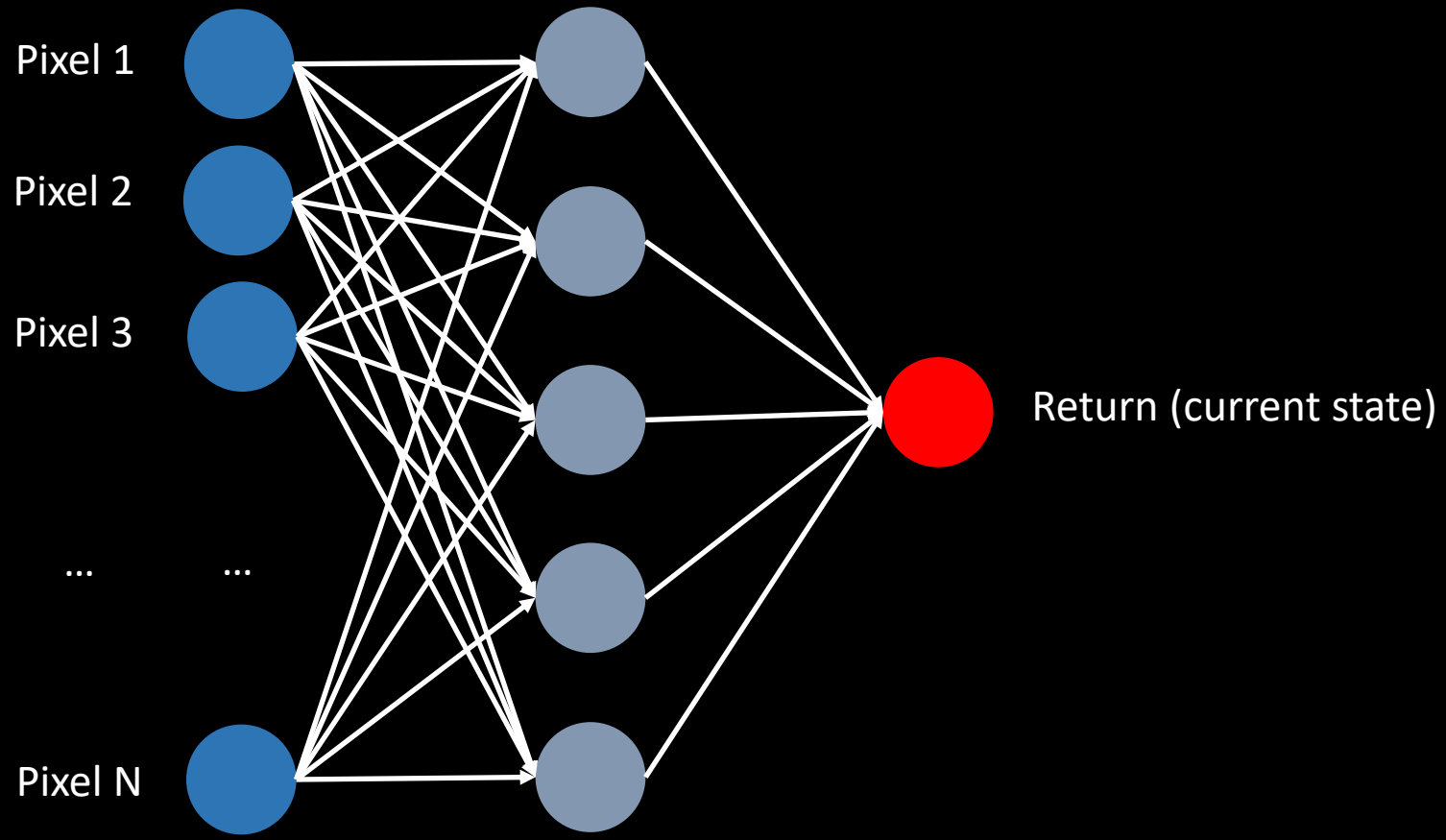
- V*(s) … exact expected return in state s
- V* … a function mapping state to expected reward
- V … an approximation to V* (usually Neural Net)
- A => B means “A approaches B”

# Learning State-Value Functions

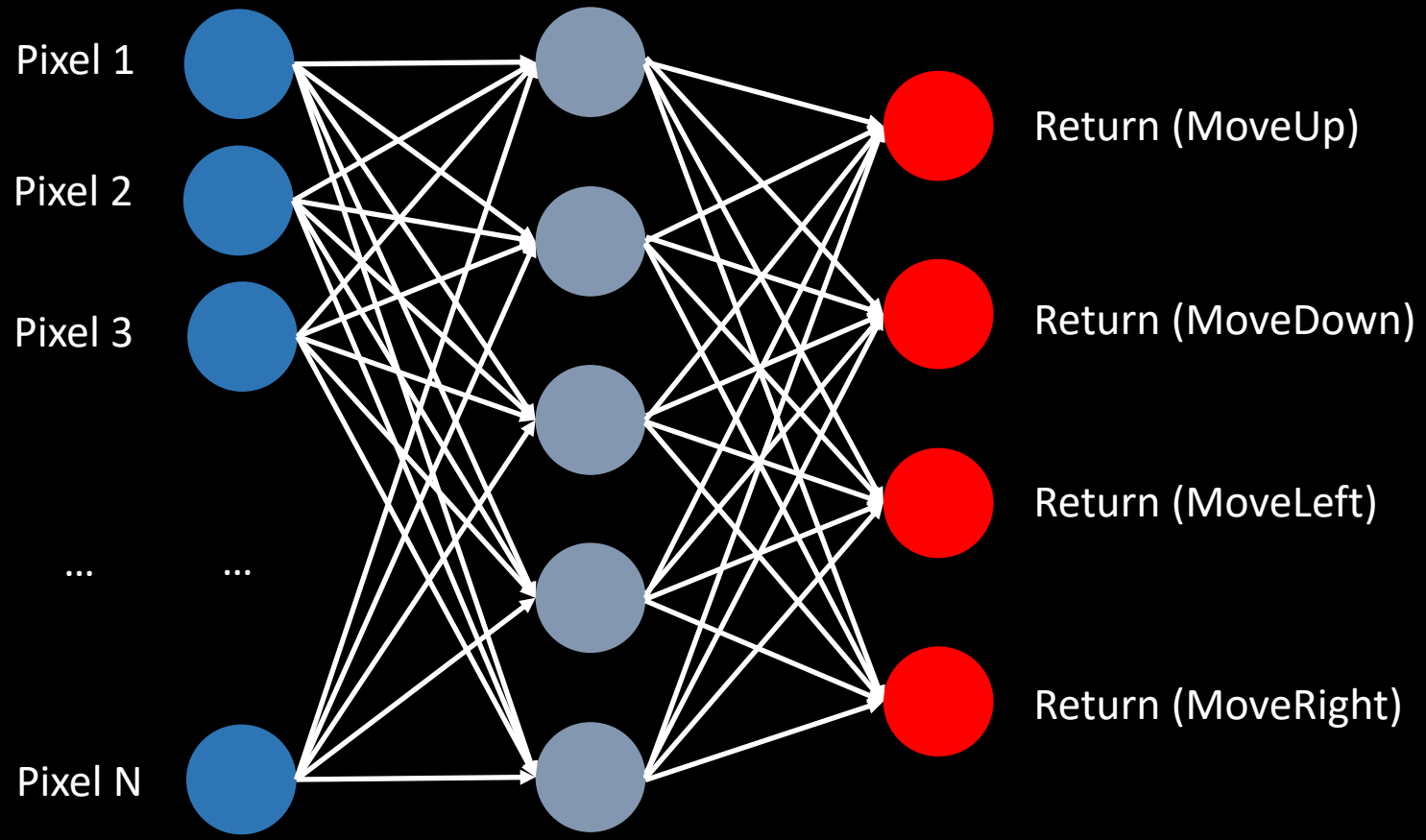
- What we want to achieve:
  - V => V*
  - Given a policy that chooses actions…
- How?
  - Iteratively update V with: V(s) => r(s) + V(s‘)

- How could we choose an action?
  - E.g.: argmax([V(get_new_state(a)) for a in all_actions]) awkward…

# State-Value Network



# Action-Value Network

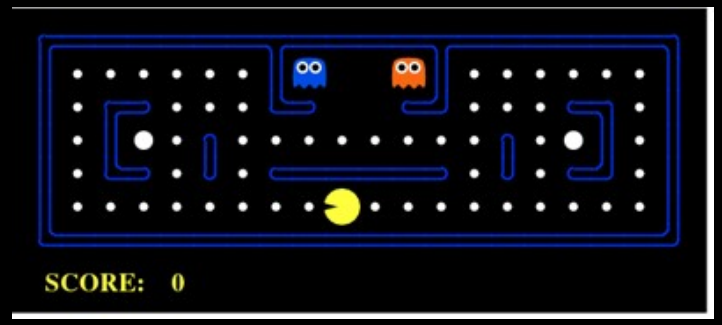


# Learning Action-Value Functions

- What we want to achieve:
  - Q => Q*
- How?
  - DQN agent:
    - Iteratively update Q with: Q(s, a) => r(s, a) + max(Q(s‘))
  - SARSA agent:
    - Iteratively update Q with: Q(s, a) => r(s, a) + Q(s‘, a‘)
- How do we choose an action?

  a = argmax(Q(s))

# Choose The Action With Argmax for DQN



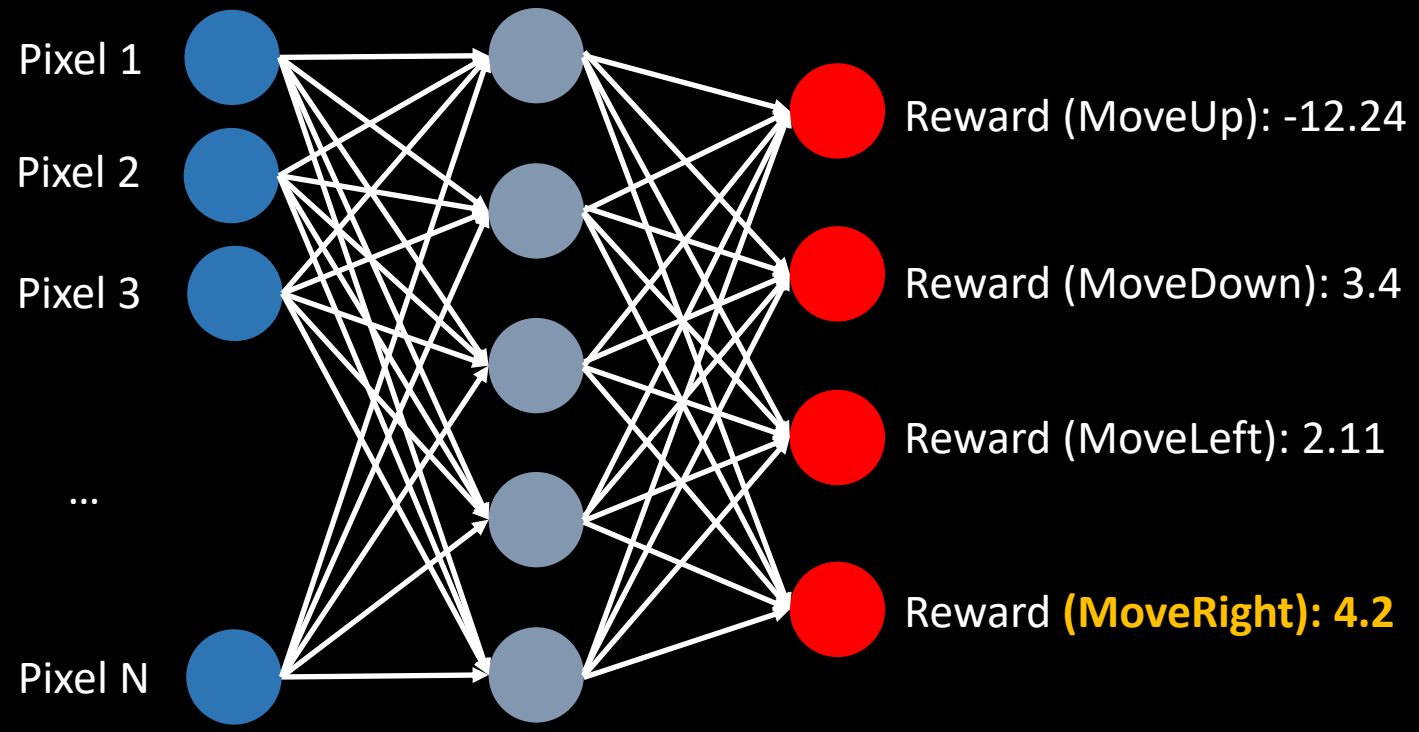
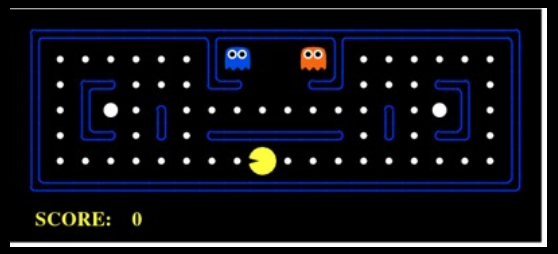
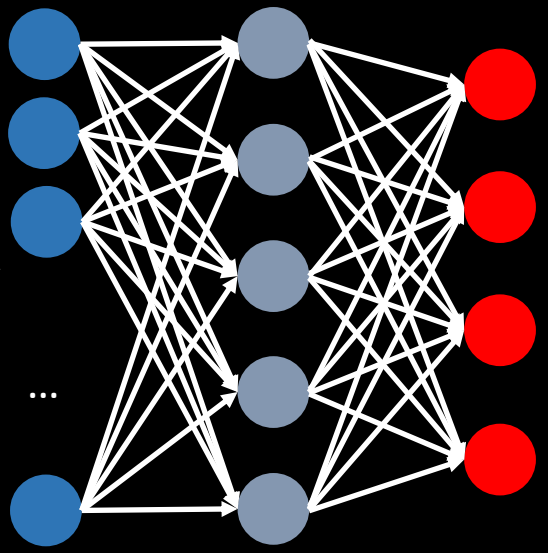




Image PacMan: http://ai.berkeley.edu

# Train DQN




s



...

R(s)_pred = 0.5

Prediction of Reward

r(s)_emp = 0.04

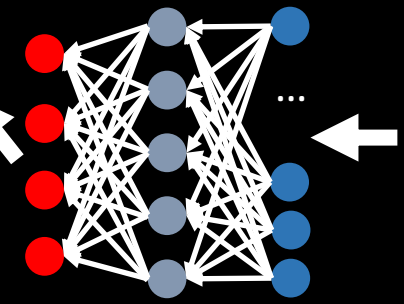
Observed Reward from playing action *a*

Mean Squared Error

Err = $(0.5 - 0.4)^2$

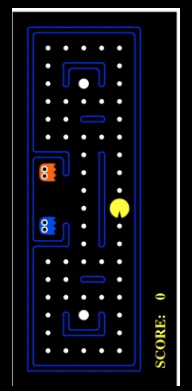
Backprop only for *a*

+



...

R(s')_pred = 0.36




s'

Image PacMan: http://ai.berkeley.edu

# Let‘s Take A Step Back…

- HOW DOES THAT EVEN WORK?!
  - The data distribution changes every batch because the policy changes!
  - How does the network know which action to blame? (Credit assignment)
  - What if success in the game requires a specific long sequence of actions?
- Memory buffer (to stabilize the data distribution)
  - Sample from last N environment interactions
  - Typical sizes are 50k to 1M samples in games like Atari!
- Exploration (to make the agent consider new paths)
  - E.g. With some probability ε act randomly
  - E.g. Parameter noise
  - E.g. Noisy DQN

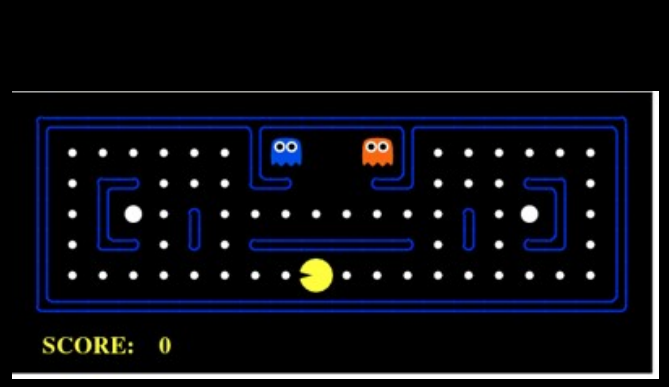
# Improved Variants

- Prioritized Replay Buffer [Schaul et al]
  - Don’t sample uniformly rand from buffer
- Double DQN [van Hasselt et al]
  - solve overestimation bias
- Dueling DQN [Wang et al]
  - Learn normalized advantage function
- Distributional DQN [Bellemare et al]
  - Learn distribution over rewards, not expected
- Noisy DQN [Fortunato et al]
  - exploration through parameter noise
- Rainbow [Hessel et al]
  - combines all of the above

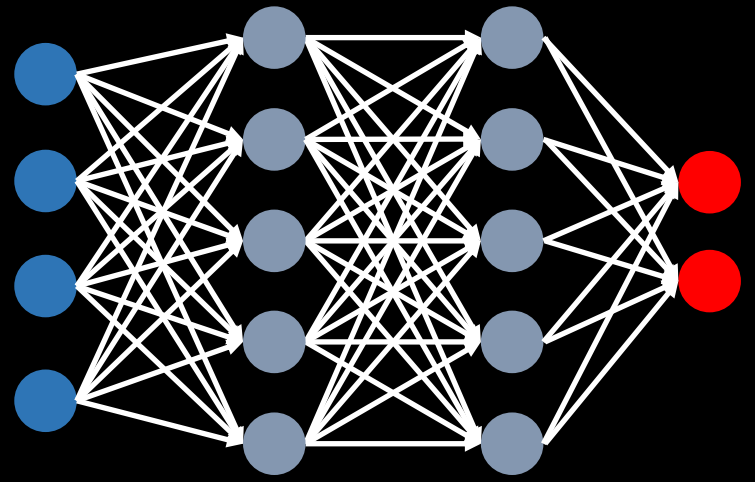
# Problems with DQN alone

- Deterministic policy… -> doesn‘t work with imperfect information
- We learn the policy only indirectly
- The value function might be super complex, while the optimal policy might be very simple.

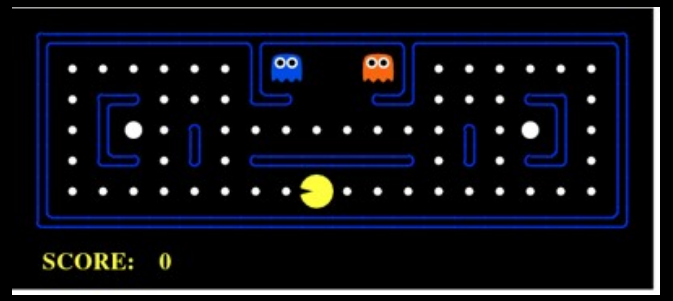
# Policy Gradient




screen



action




reward

INCREASE PROBABILITY OF REWARDING ACTIONS

Image PacMan: http://ai.berkeley.edu

# How to „make rewarding actions more likely“?

1. Output of $\pi$_θ is softmax over all possible actions
2. We want: ∇ R w.r.t. θ
3. Cause then we can: θ = θ + α ∇R($\pi$ , s, a)
4. We can: Sample N trajectories using $\pi$ and store (*s, a, r*) tuples
5. We can: Compute (s_t, R_t) from that
6. We get:

$$\nabla_\theta \mathbb{E}_\tau[R] \approx \mathbb{E}_\tau \left[ \sum_{t=0}^{T-1} \nabla_\theta \log(\pi(a_t|s_t, \theta)) R_\tau \right]$$

# The Policy Gradient

$$\nabla_\theta \mathbb{E}_\tau[R] \approx \mathbb{E}_\tau \left[ \sum_{t=0}^{T-1} \nabla_\theta \log(\pi(a_t|s_t, \theta)) R_\tau \right]$$

Where…

Tau = a trajectory (sequence of state-action-state-action…)

t = timestep t in the trajectory

π = current policy

R_Tau = Empirical return of trajectory Tau

E_Tau [R] = Estimate of the return of trajectory Tau

π(a|s) = probability of policy π performing action *a* in state *s*

# Policy Gradient with Advantage Estimation

$$\nabla_\theta \mathbb{E}_\tau[R] \approx \mathbb{E}_\tau \left[ \sum_{t=0}^{T-1} \nabla_\theta \log(\pi(a_t|s_t, \theta)) A_t \right]$$

Where…

Tau = a trajectory (sequence of state-action-state-action…)

t = timestep t in the trajectory

π = current policy

A_t = action advantage estimate (i.e. A(a) = Q(s, π, a) - mean(Q(s, π) for all a))

E_Tau [R] = Estimate of the return of trajectory Tau

π(a|s) = probability of policy π performing action *a* in state *s*

# Policy Gradient Methods

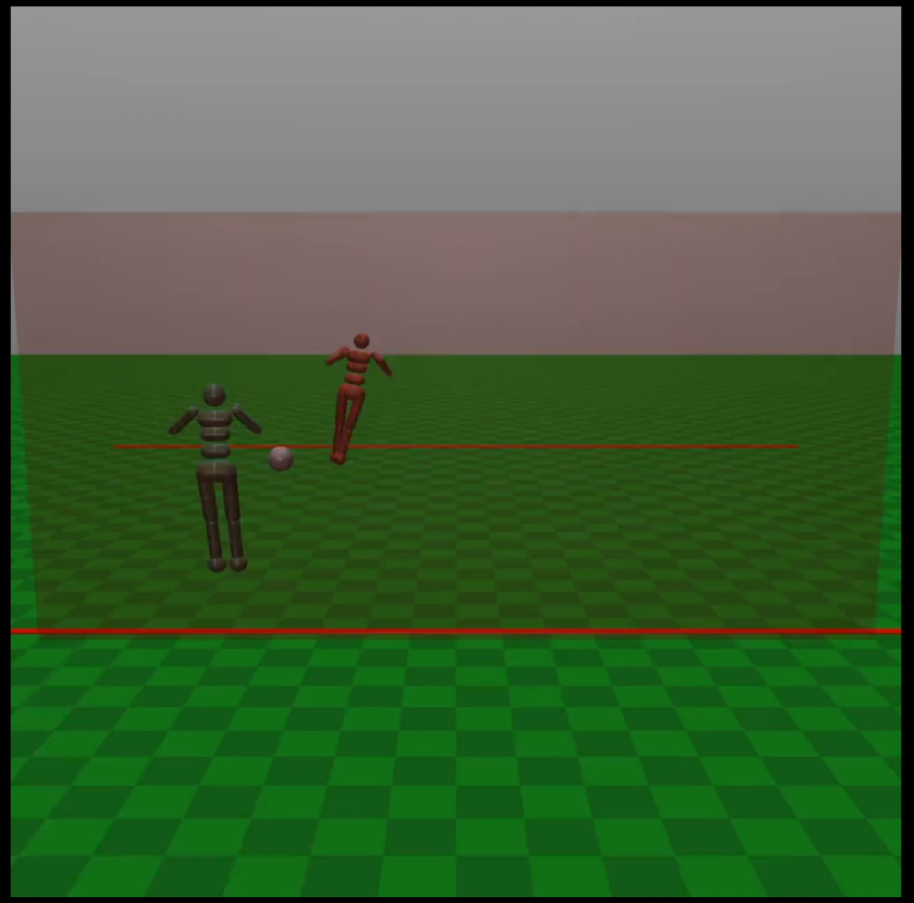
- Output an action-probability distribution (non-deterministic strategy) Problem:
  - Gradient Decent: “For step-size S, I treat the error manifold as if it was linear”
  - Large step size -> huge change in strategy –> no granularity
  - small step size -> super slow convergence

- Some PG algorithms: PPO, ACER, TRPO

# That Was Loooots Of Info – Summary so far:

- SL just maps x -> y while RL finds *strategies.*
- Every DNN needs input-label type data (doesn‘t matter if SL, USL, RL)
- We can use the ***empirical*** reward gained by sampling to learn either a
  - ***Value Function V*** or a
  - ***Action-value Function Q*** or a
  - ***Policy π*** directly

# Self-Play In Multi-Agent Environments



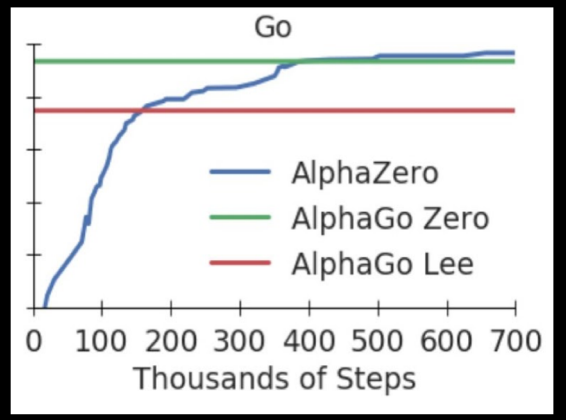
Emergent Complexity via Multi-Agent Competition [Bansal et al., 2018]

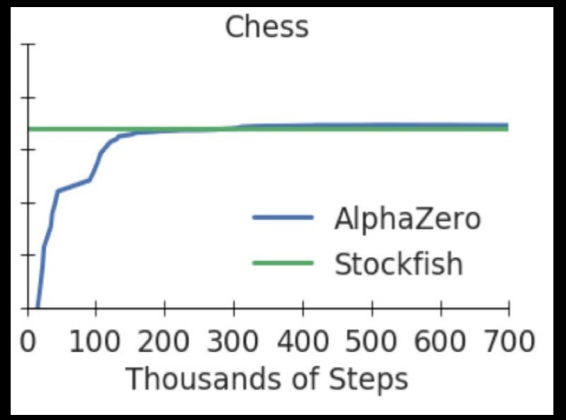
# Self-Play RL > Human Expert Data

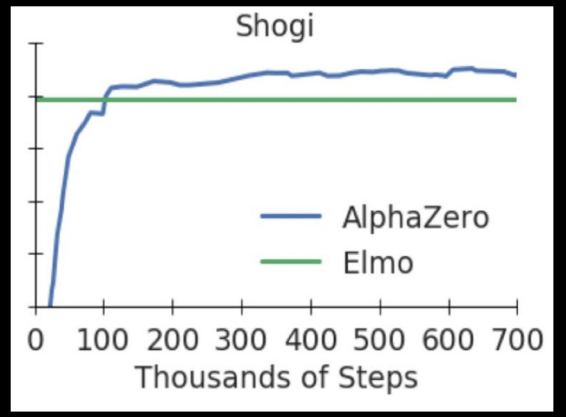
**AlphaGo Lee**
Google DeepMind
March 2016

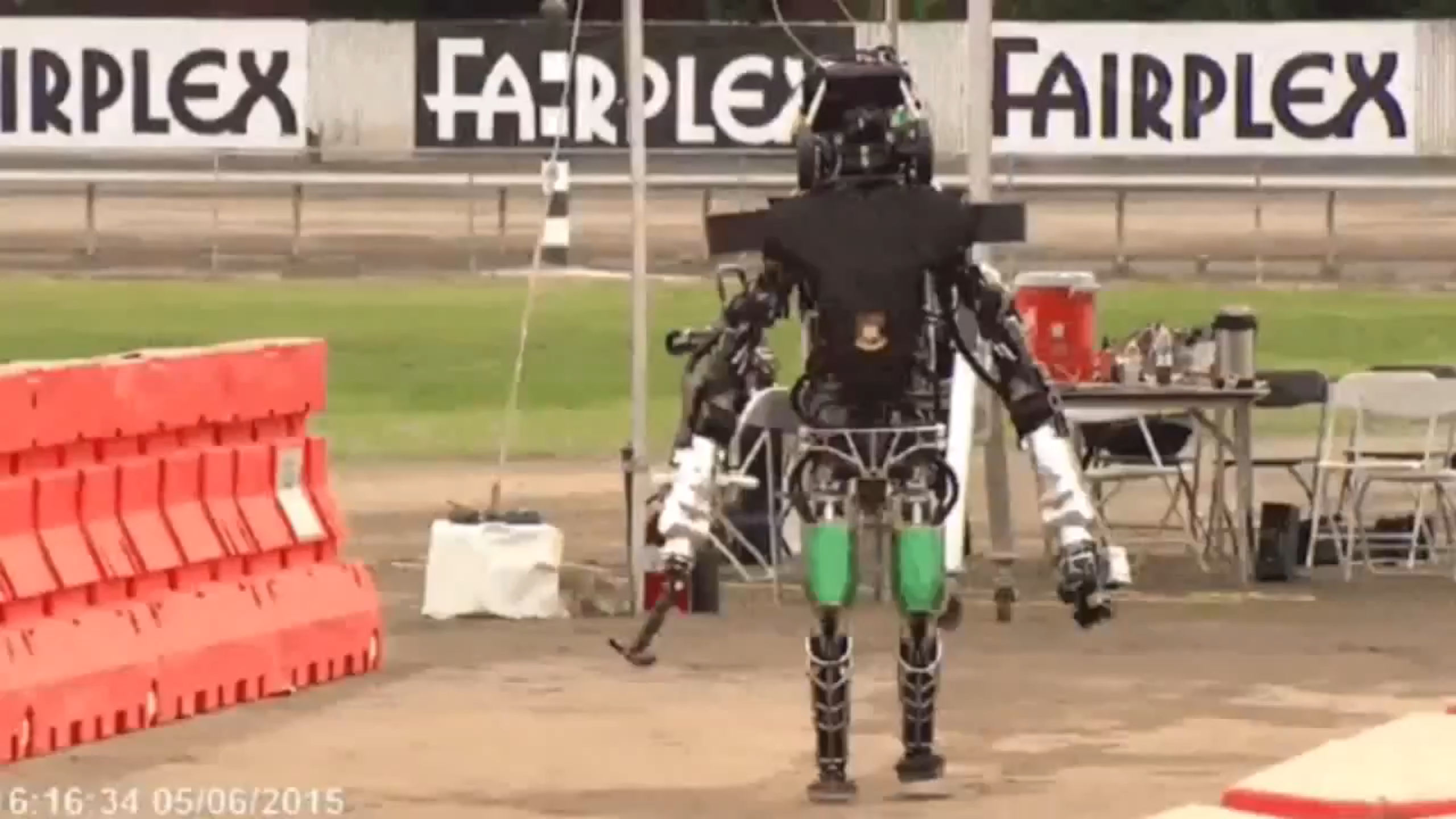
**AlphaGo Zero**
Google DeepMind
October 2017

**Alpha Zero**
Google DeepMind
December 2017

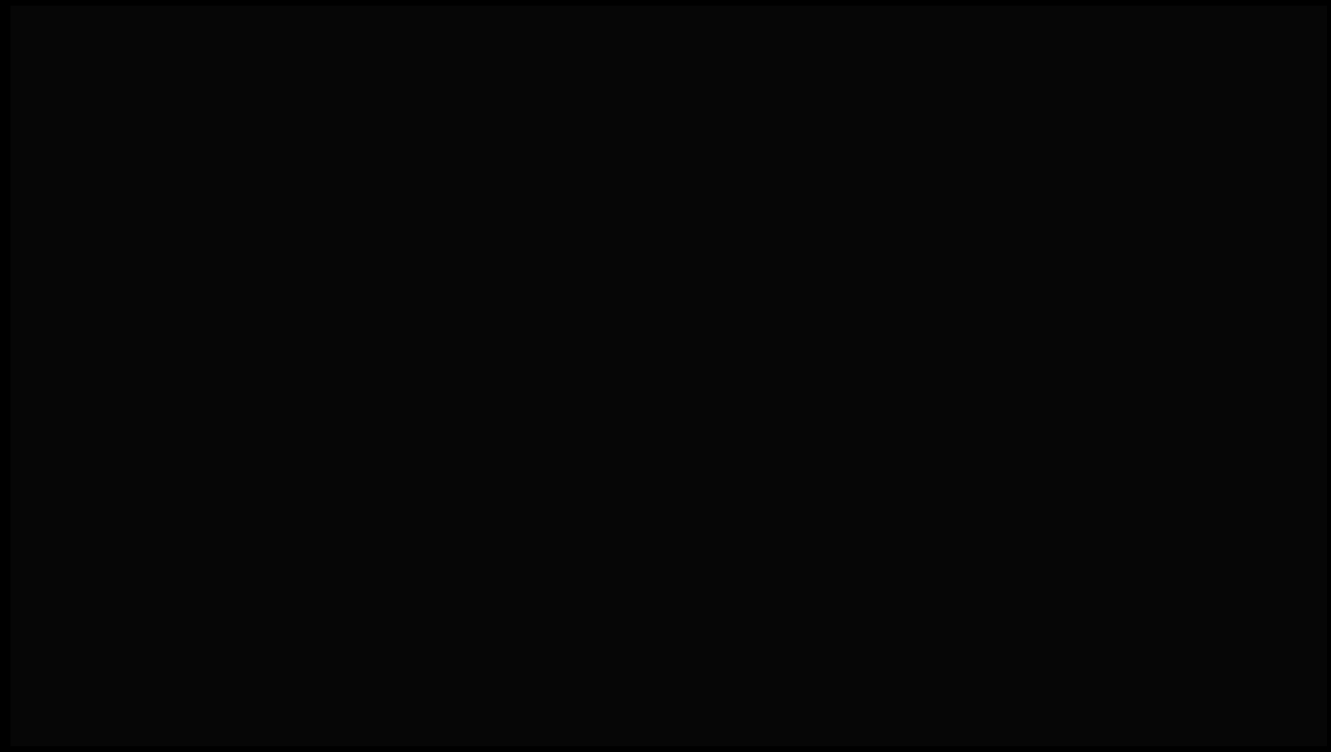



FAIRPLEX
FAIRPLEX
FAIRPLEX
6:16:34 05/06/2015

# Randomization Helps To Generalize



OpenAI Dexterity [OpenAI et al., 2018]

# Discussion & Questions

- Deep Reinforcement Learning -


**Eric Steinberger**

**Contact: eric@steinberger-ai.com**
**Twitter: @EricSteinb**
**Linkedin.com/in/ericsteinbergerai**