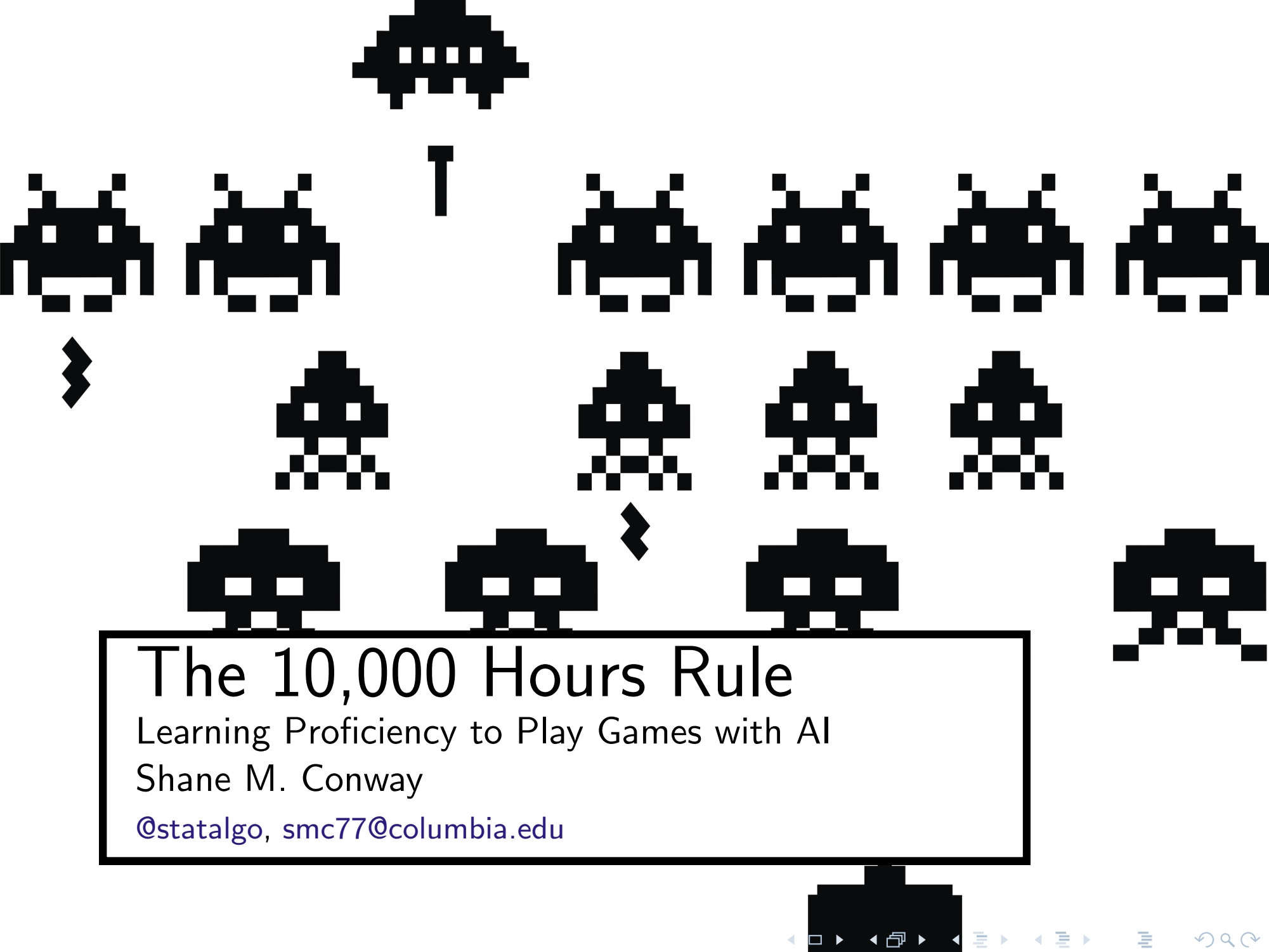


The 10,000 Hours Rule

Learning Proficiency to Play Games with AI

Shane M. Conway

@statalgo, smc77@columbia.edu

"I think we should be very careful about artificial intelligence. If I had to guess at what our biggest existential threat is, it's probably that. So we need to be very careful. I'm increasingly inclined to think that there should be some regulatory oversight, maybe at the national and international level, just to make sure that we don't do something very foolish." -Elon Musk

Outline

Outline

Learning to Learn by Playing Games

Artificial Intelligence

Artificial General Intelligence (AGI) has made significant progress in the last few years.

I want to review some of the latest models:

- Discuss tools from DeepMind and OpenAI.
- Demonstrate models on games.

Artificial Intelligence

Progress in AI has been driven by different advances:

1. Compute (the obvious one: Moore's Law, GPUs, ASICs),
2. Data (in a nice form, not just out there somewhere on the internet - e.g. ImageNet),
3. Algorithms (research and ideas, e.g. backprop, CNN, LSTM), and
4. Infrastructure (software under you - Linux, TCP/IP, Git, ROS, PR2, AWS, AMT, TensorFlow, etc.).

Source: @karpathy

Tools

This talk will highlight a few major tools:

- OpenAI gym and universe
- Google TensorFlow

I will also focus on a few specific models

- DQN
- A3C
- NEC

Game Play

Why games?

Playing games generally involves:

- Very large state spaces.
- A sequence of actions that leads to a reward.
- Adversarial opponents.
- Uncertainty in states.

Outline

Claude Shannon

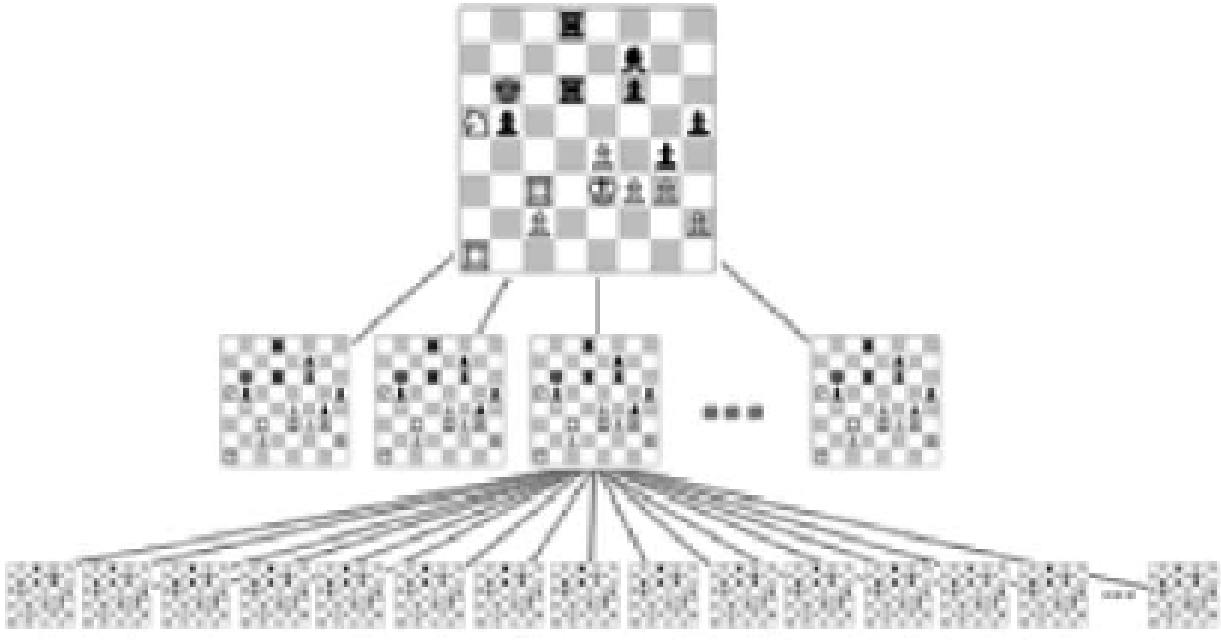
In 1950, Claude Shannon published "Programming a Computer for Playing Chess", introducing the idea of "minimax"

Arthur Samuel

Arthur Samuel (1956) created a program that beat a self-proclaimed expert at Checkers.

Chess

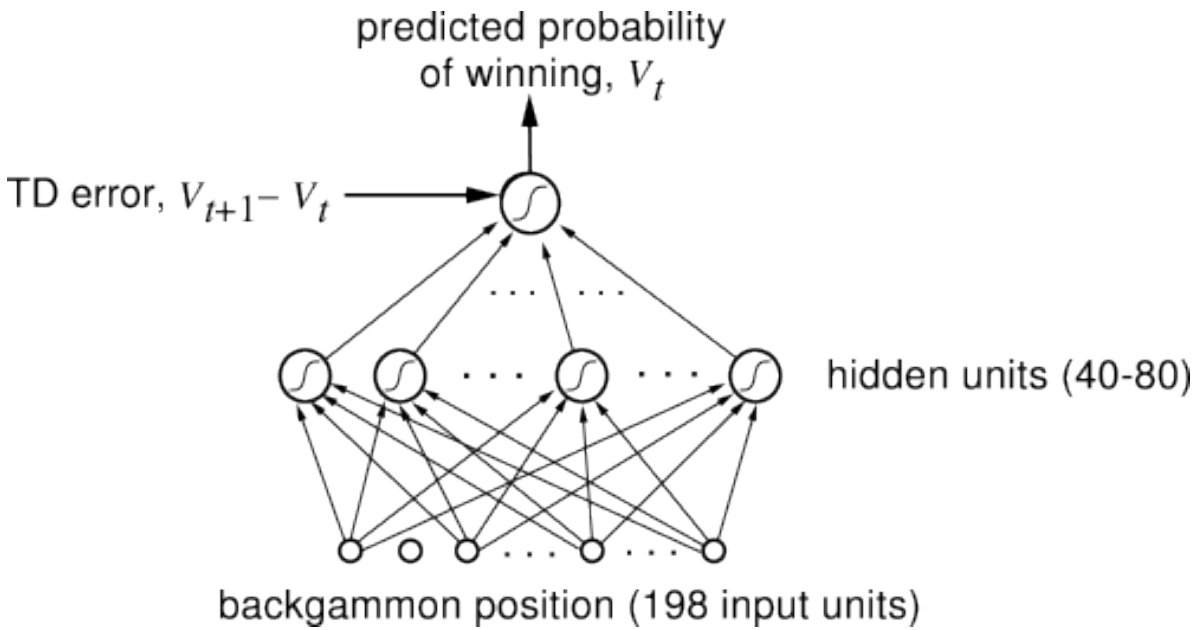
DeepBlue achieved "superhuman" ability in May 1997.



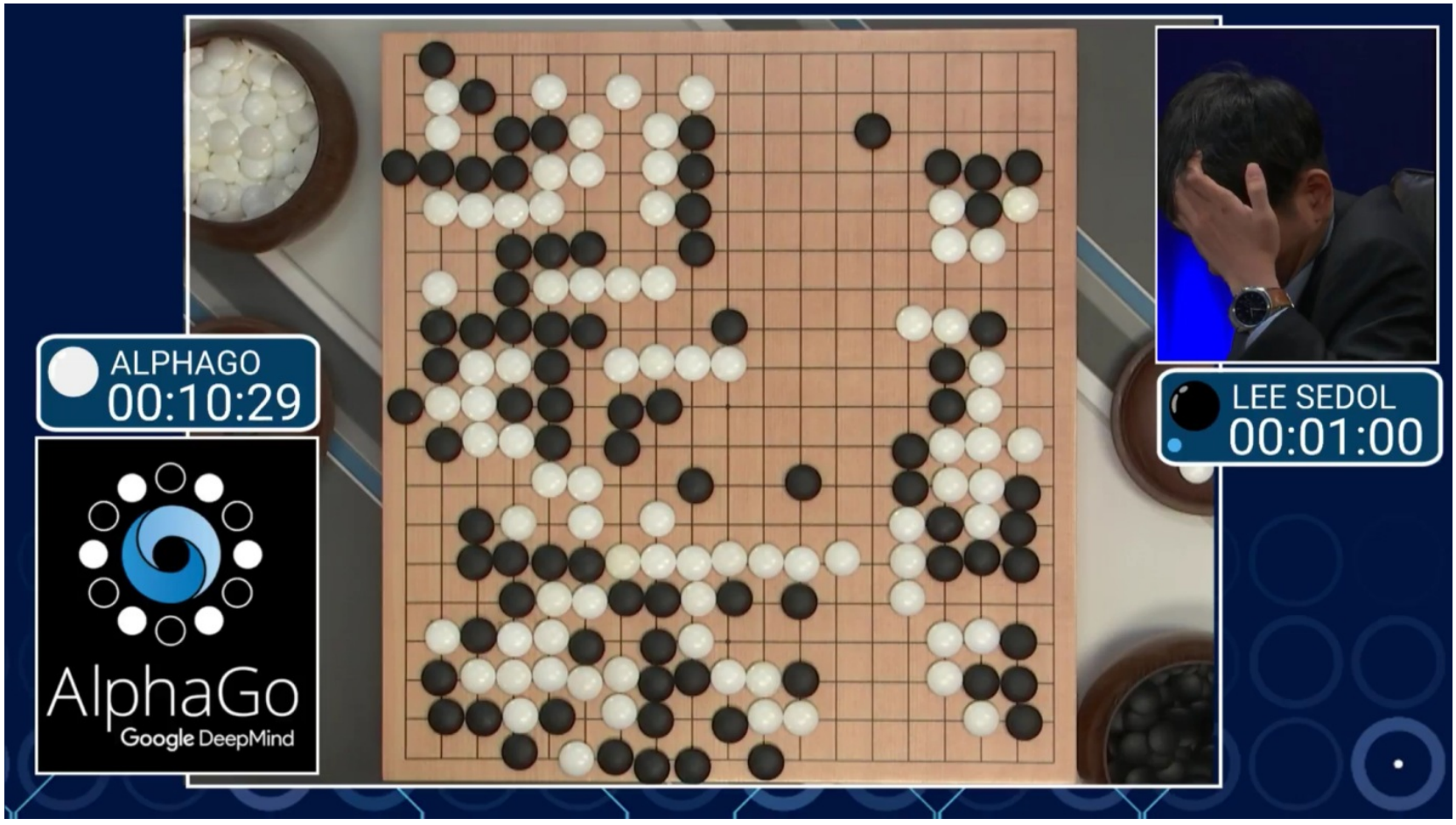
Article about DeepBlue, *General Game Playing course at Stanford*

Backgammon

Tesauro (1995) "Temporal Difference Learning and TD-Gammon" may be the most famous success story for RL, using a combination of the TD(λ) algorithm and nonlinear function approximation using a multilayer neural network trained by backpropagating TD errors.

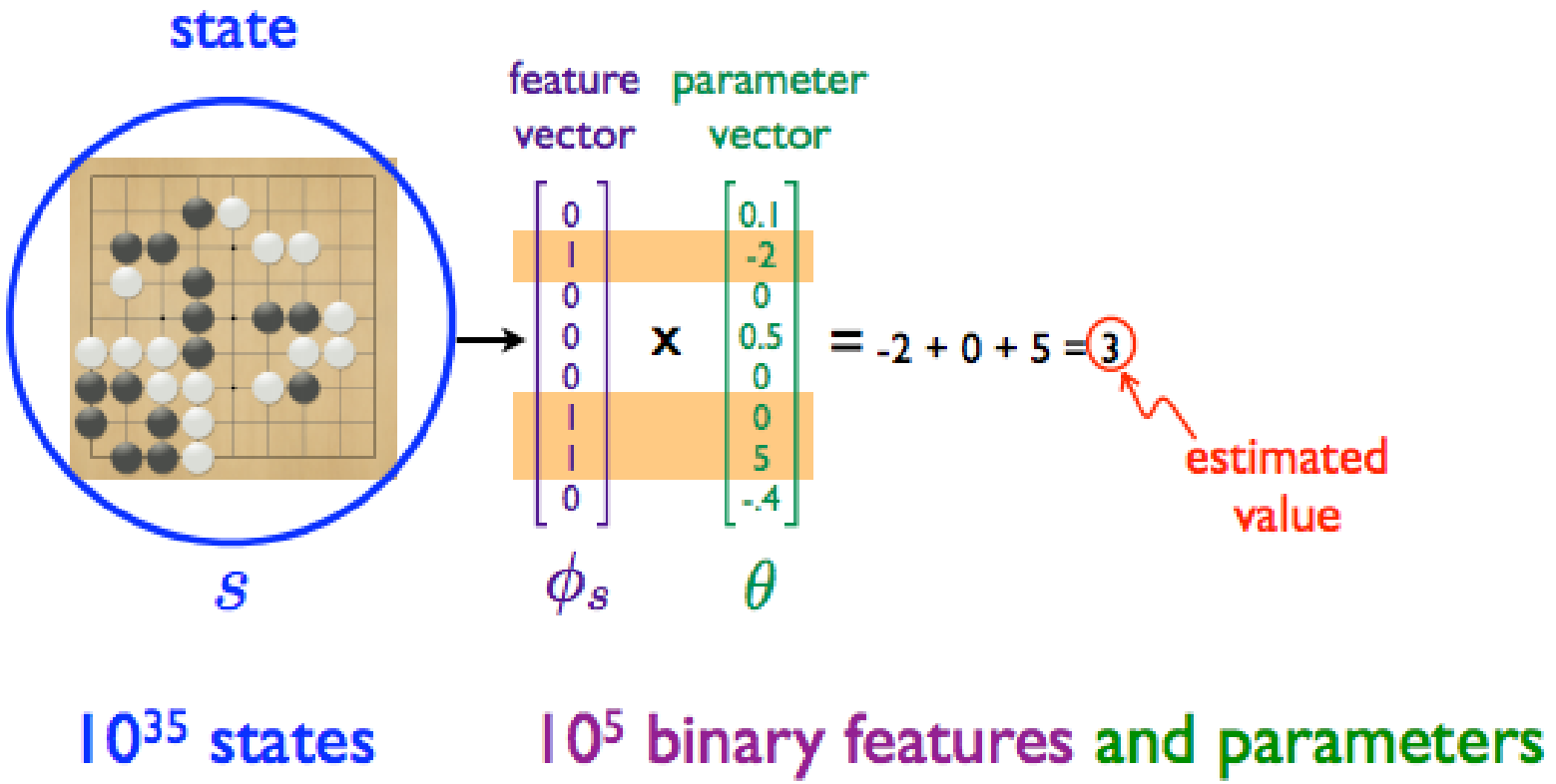


Go



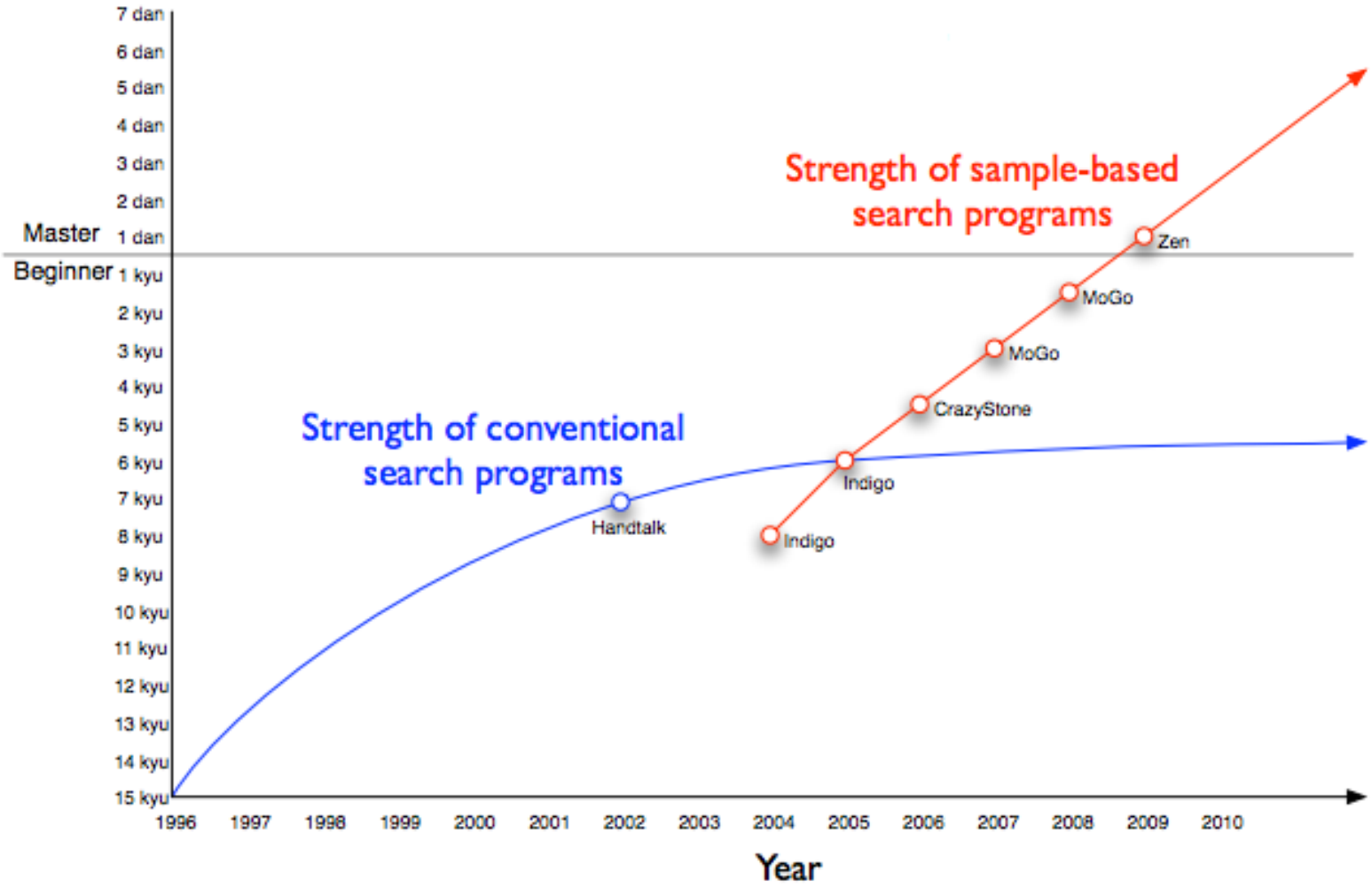
The number of potential legal board positions in go is greater than the number of atoms in the universe.

Go



From Sutton (2009) "Deconstructing Reinforcement Learning" ICML

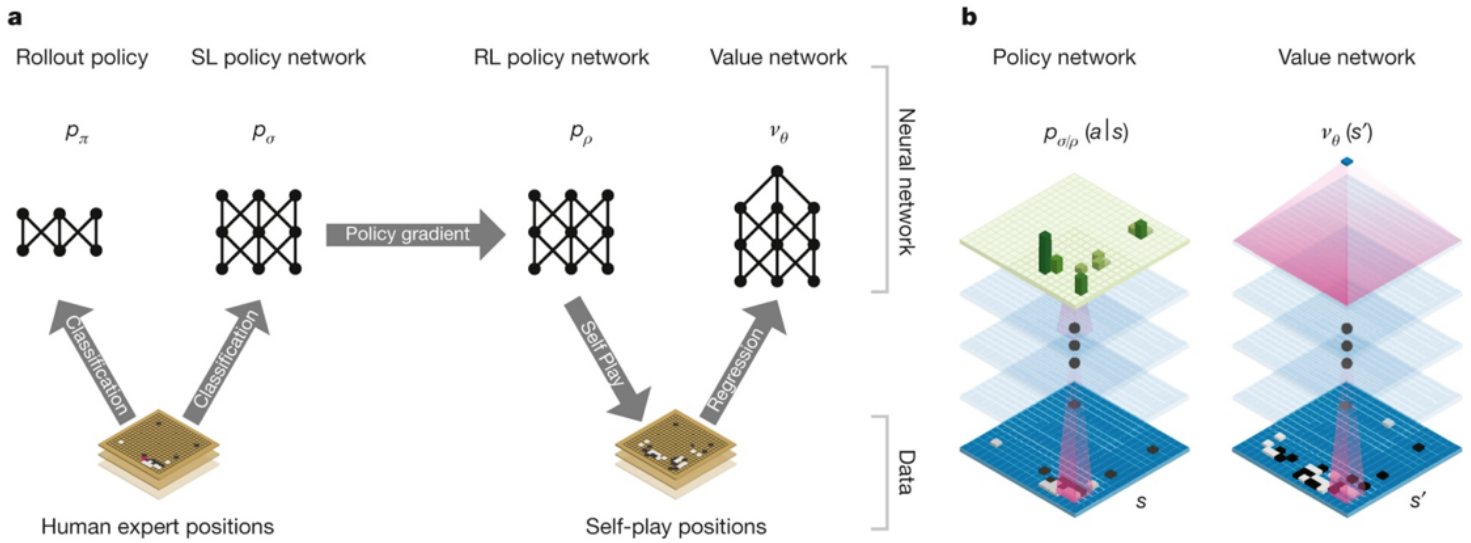
Go



From Sutton (2009) "Fast Gradient-Descent Methods for Temporal-Difference Learning with Linear Function Approximation" ICML

Go

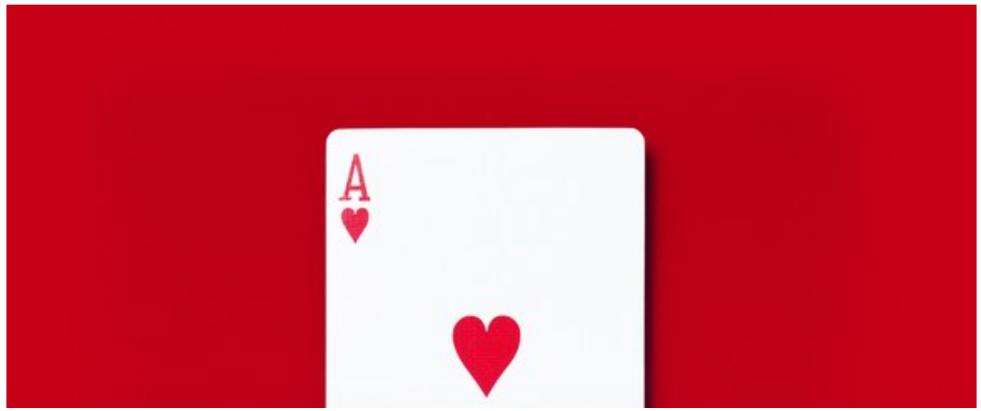
AlphaGo combined supervised learning and reinforcement learning, and made massive improvement through self-play.



Poker

CADE METZ BUSINESS 02.01.17 07:00 AM

INSIDE LIBRATUS, THE POKER AI THAT OUT-BLUFFED THE BEST HUMANS



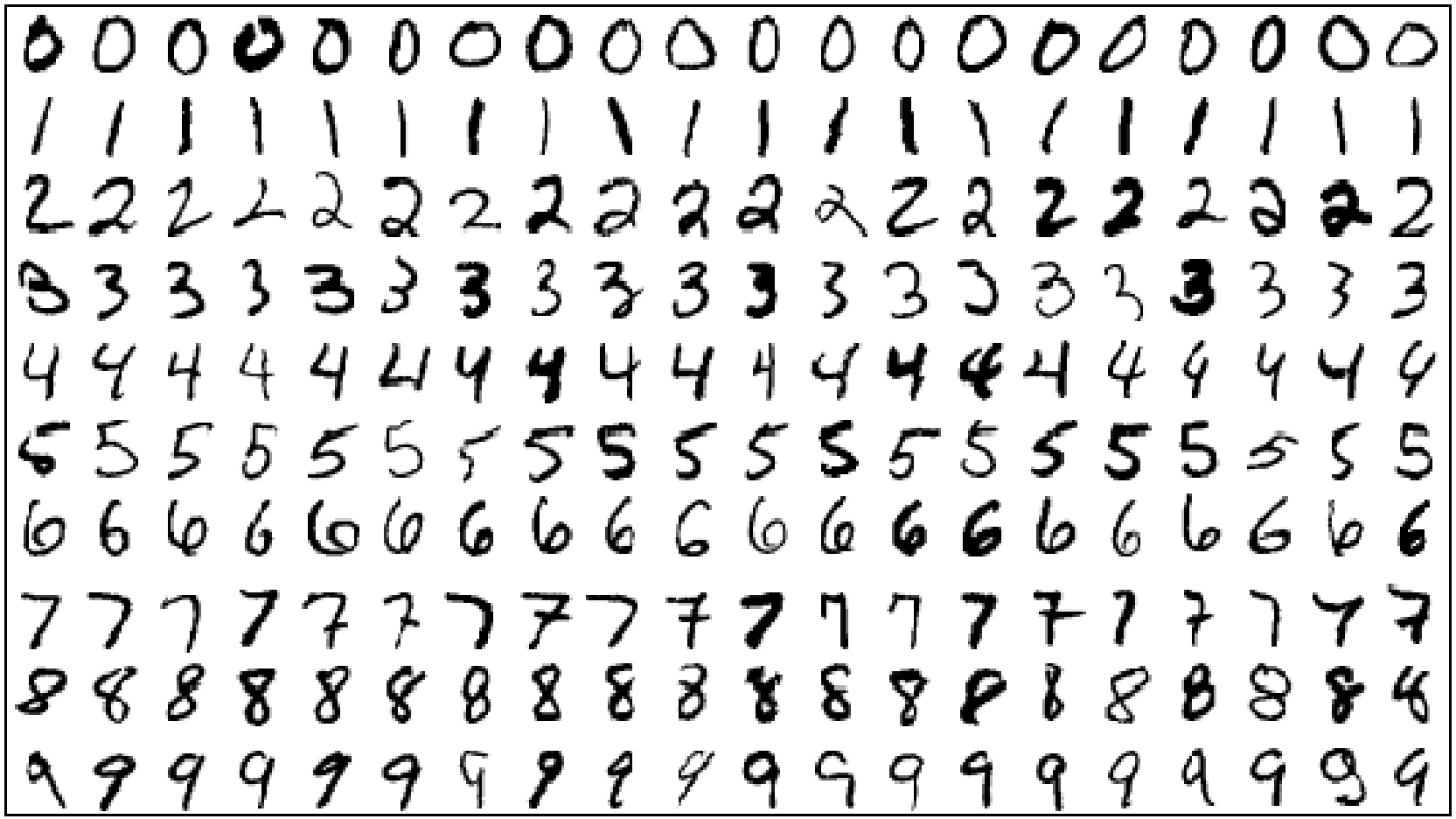
Dota 2

Outline

My Network has more layers than yours...

Benchmarks and progress

MNIST



ImageNet

One of the classic examples of AI benchmarks is ImageNet.

Others: http://deeplearning.net/datasets/
http://image-net.org/challenges/LSVRC/2017/

OpenAI gym

For control problems, there is a growing universe of environments for benchmarking:

- Classic control
- Board games
- Atari 2600
- MuJoCo
- Minecraft
- Soccer
- Doom

Roboschool is intended to provide multi-agent environments.

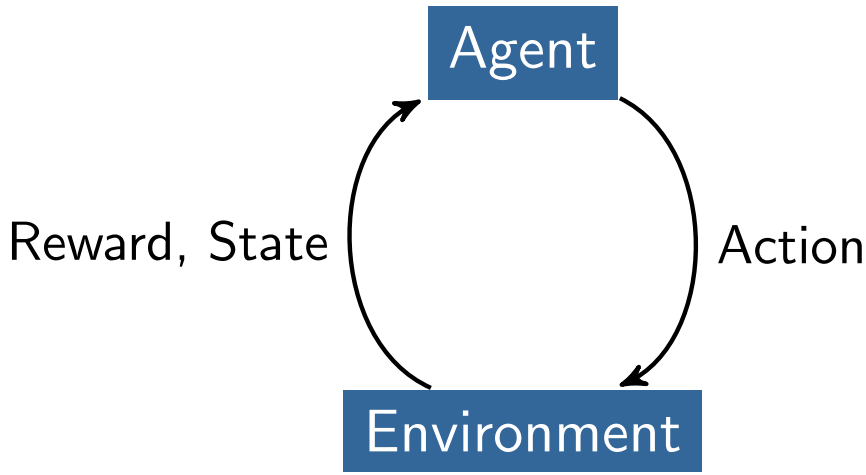
Outline

Try that again

...an again

Reinforcement Learning

In a single agent version, we consider two major components: the *agent* and the *environment*.



The agent takes actions, and receives updates in the form of state/reward pairs.

RL Model

An MDP tranisitons from state s to state s' following an action a, and receiving a reward r as a result of each transition:

$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \ldots \tag{1}$$

MDP Components

- S is a set of states
- A is set of actions
- $R(s)$ is a reward function

In addition we define:

- $T(s'|s, a)$ is a probability transition function
- γ as a discount factor (from 0 to 1)

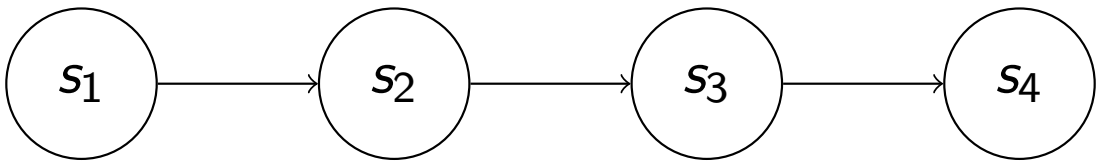
Markov Models

We can extend the markov process to study other models with the same the property.

Markov Models	Are States Observable?	Control Over Transitions?
Markov Chains	Yes	No
MDP	Yes	Yes
HMM	No	No
POMDP	No	Yes

Markov Processes

Markov Processes are very elementary in time series analysis.



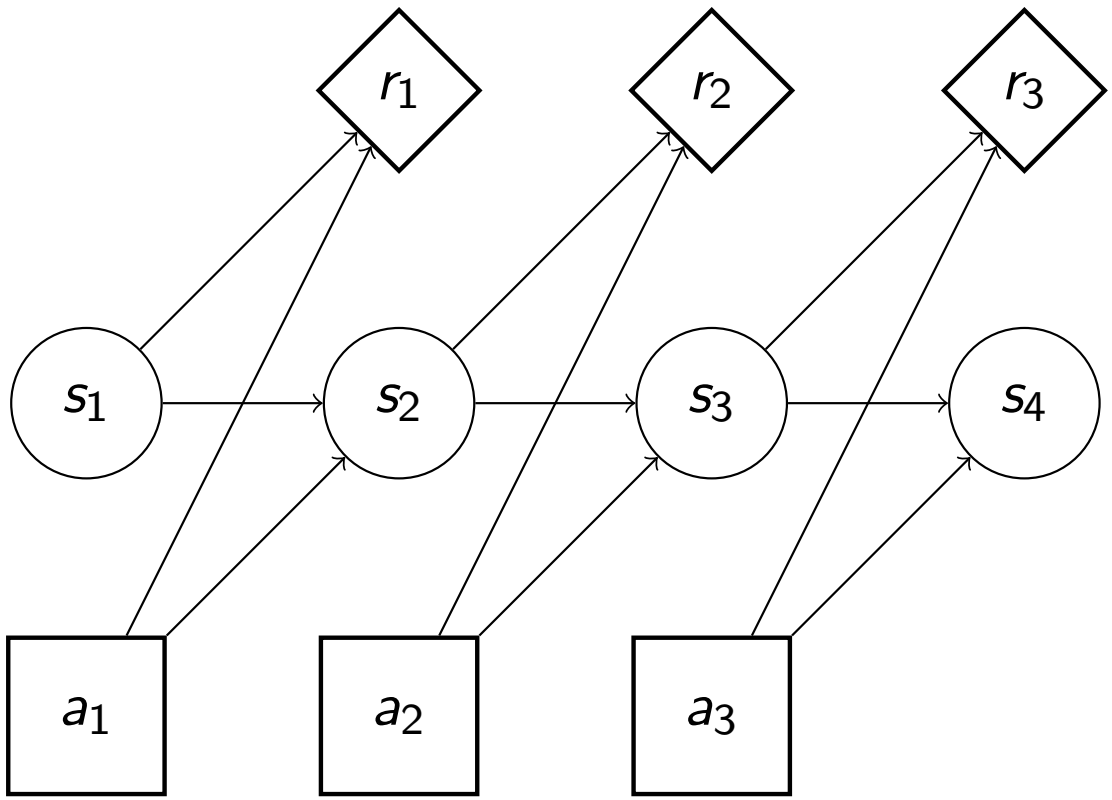
Definition

$$P(s_{t+1}|s_t, ..., s_1) = P(s_{t+1}|s_t) \tag{2}$$

- s_t is the state of the markov process at time t.

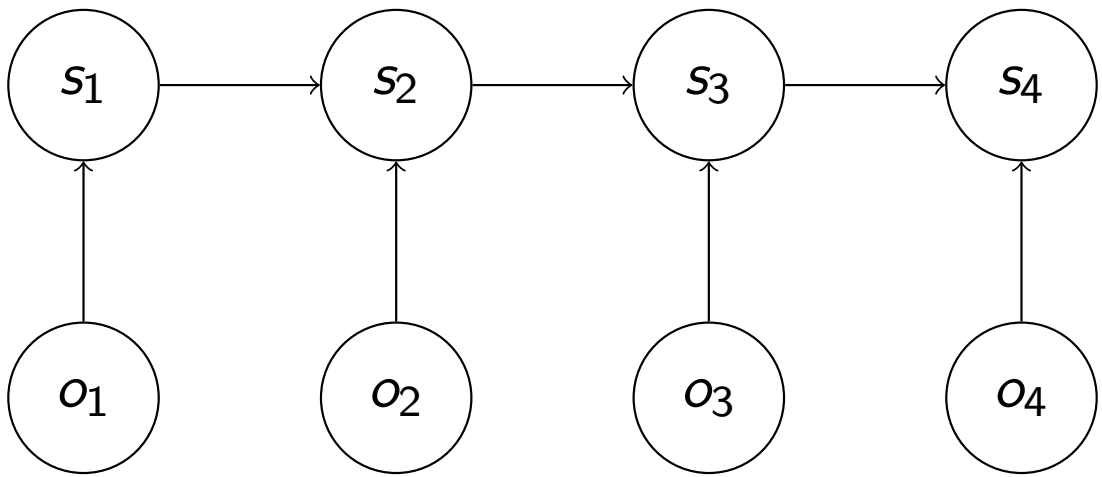
Markov Decision Process (MDP)

A Markov Decision Process (MDP) adds some further structure to the problem.



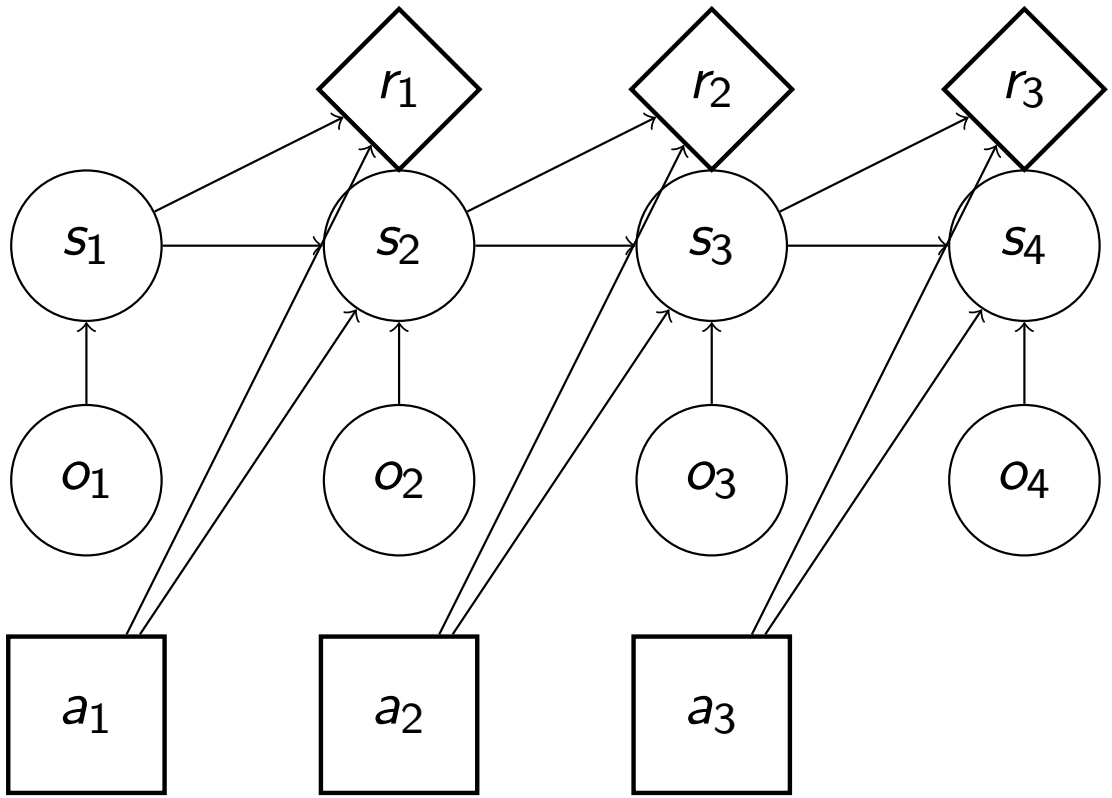
Hidden Markov Model (HMM)

Hidden Markov Models (HMM) provide a mechanism for modeling a hidden (i.e. unobserved) stochastic process by observing a related observed process. HMM have grown increasingly popular following their success in NLP.



Partially Observable Markov Decision Processes (POMDP)

A Partially Observable Markov Decision Processes (POMDP) extends the MDP by assuming partial observability of the states, where the current state is a probability model (a *belief state*).



Value function

We define a *value function* to maximize the expected return:

$$V^{\pi}(s) = E[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots | s_0 = s, \pi]$$

We can rewrite this as a recurrence relation, which is known as the **Bellman Equation**:

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} T(s') V^{\pi}(s')$$

$$Q^{\pi}(s, a) = R(s) + \gamma \sum_{s' \in S} T(s') max_a Q^{\pi}(s', a')$$

Lastly, for policy gradient we can be interested in the advantage function:

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$$

Policy

The objective is to find a policy π that maps actions to states, and will maximize the rewards over time:

$$\pi(s) \rightarrow a$$

The policy can be a table or a model.

Function Approximation

We can use functions to approximate different components of the RL model (value function, policy): generalize from seen states to unseen states.

- Value based: learn the value function, with implicit policy (e.g. ϵ-greedy)
- Policy based: no value function, learn policy
- Actor-Critic: learn value function, learn policy

Policy Search

In *policy search*, we are trying many different policies. We don't need to know the value of each state/action pair.

- Non-gradient based methods (e.g. hill climbing, simplex, genetic algorithms)
- Gradient based methods (e.g. gradient descent, quasi-newton)

Policy gradient theorem:

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)]$$

Outline

I have a rough sense for where I am?

What to do when the state space is too large...

Artificial neural networks (ANN) are learning models that were directly inspired by the structure of biological neural networks.

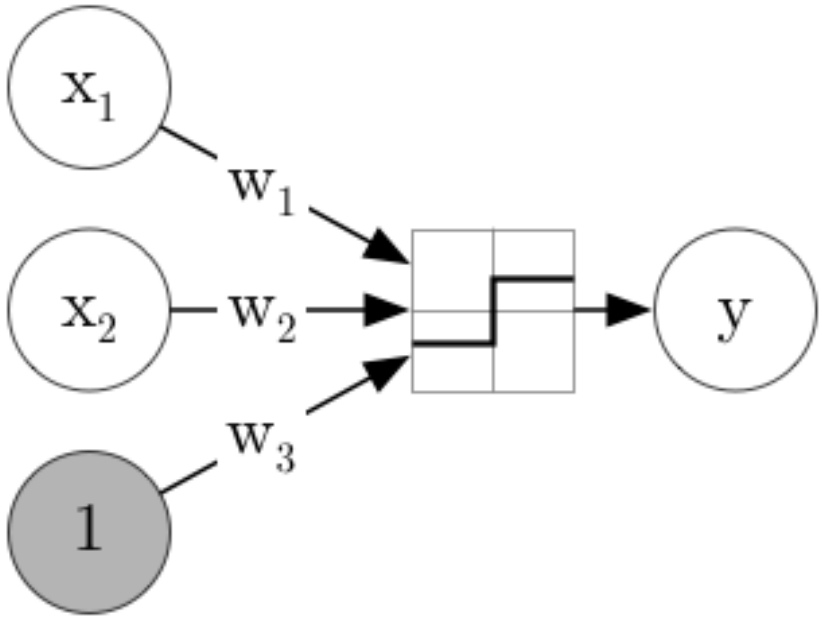


Figure: A perceptron takes inputs, applies weights, and determines the output based on an activation function (such as a sigmoid).

Image source: @jaschaephraim

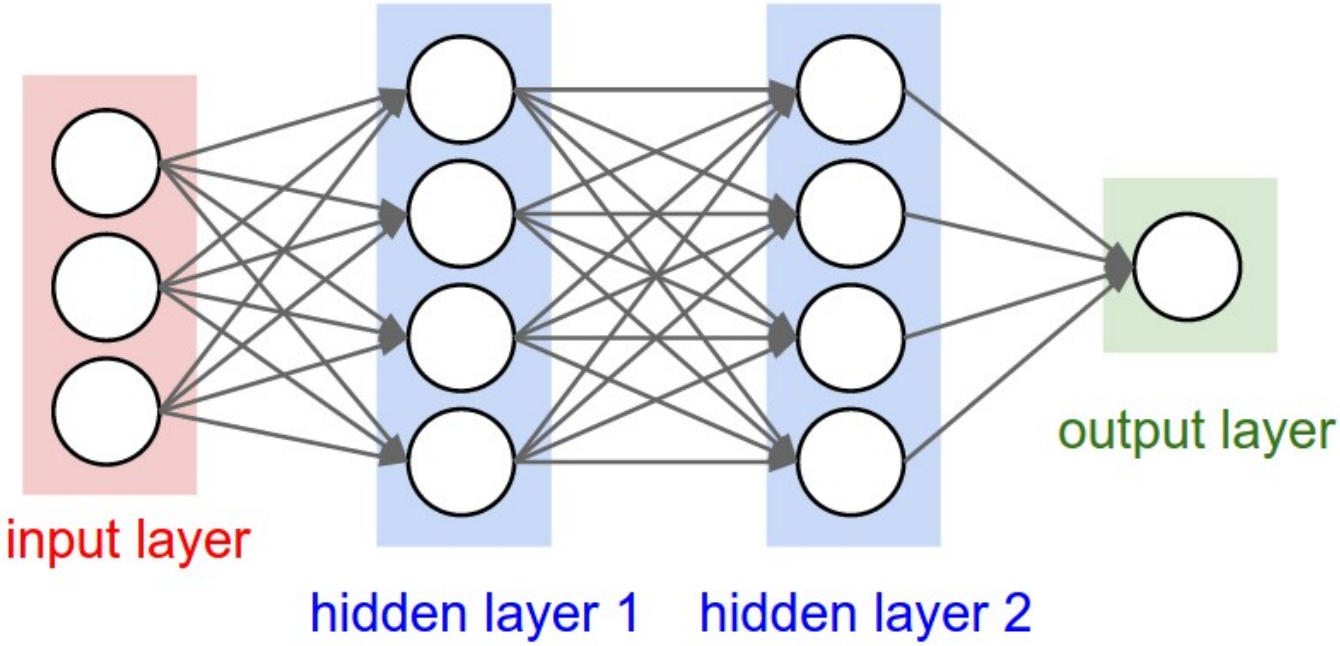
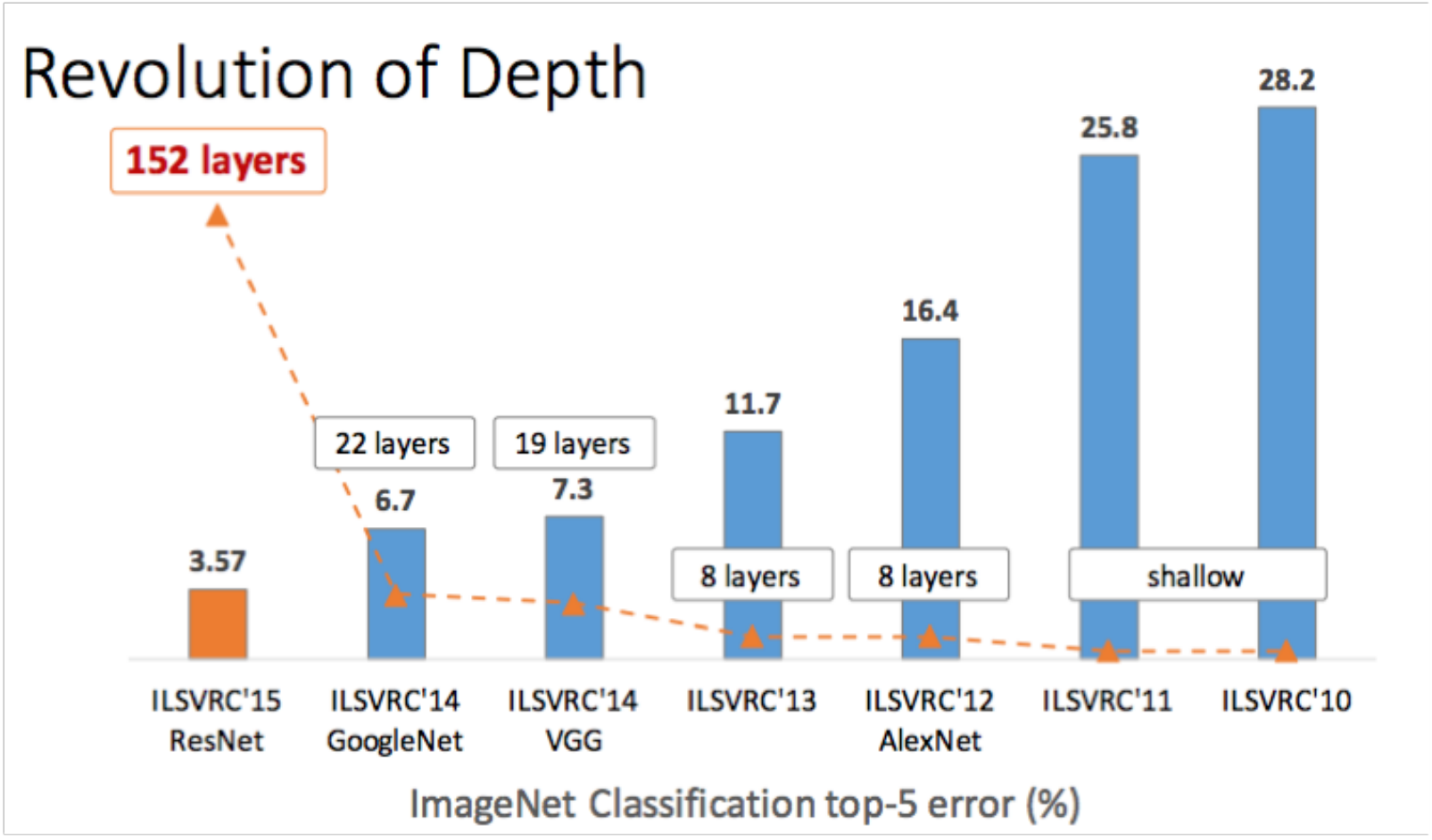


Figure: Multiple layers can be connected together.

Deep Learning

Deep Learning employs multiple levels (hierarchy) of representations, often in the form of a large and wide neural network.



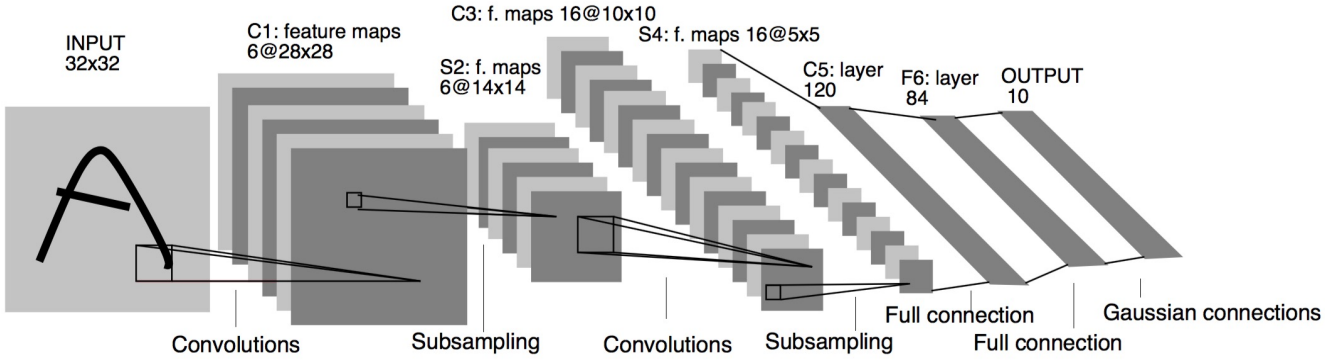


Figure: LeNET (1998), Yann LeCun et. al.

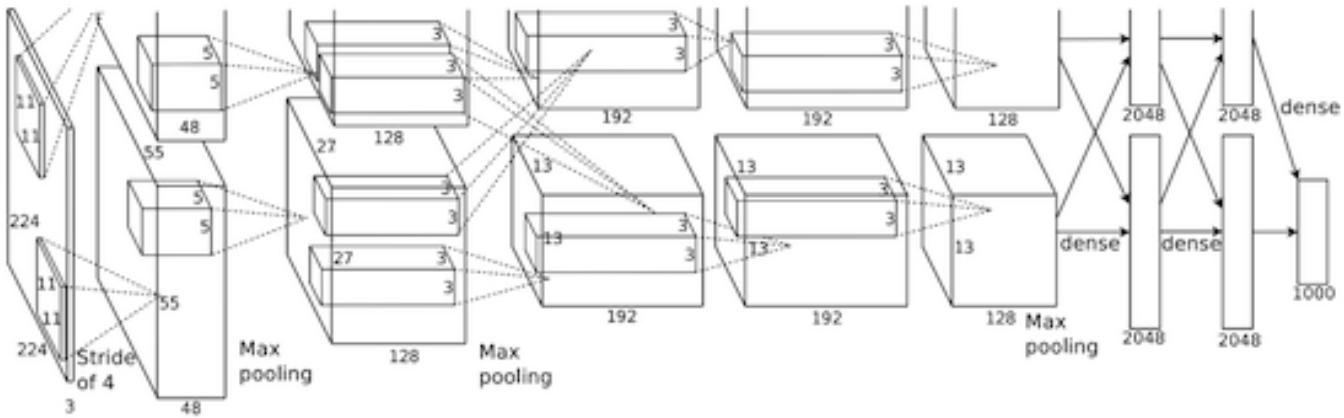


Figure: AlexNET (2012), Alex Krizhevsky, Ilya Sutskever and Geoff Hinton

Source: *Andrej Karpathy*

TensorFlow

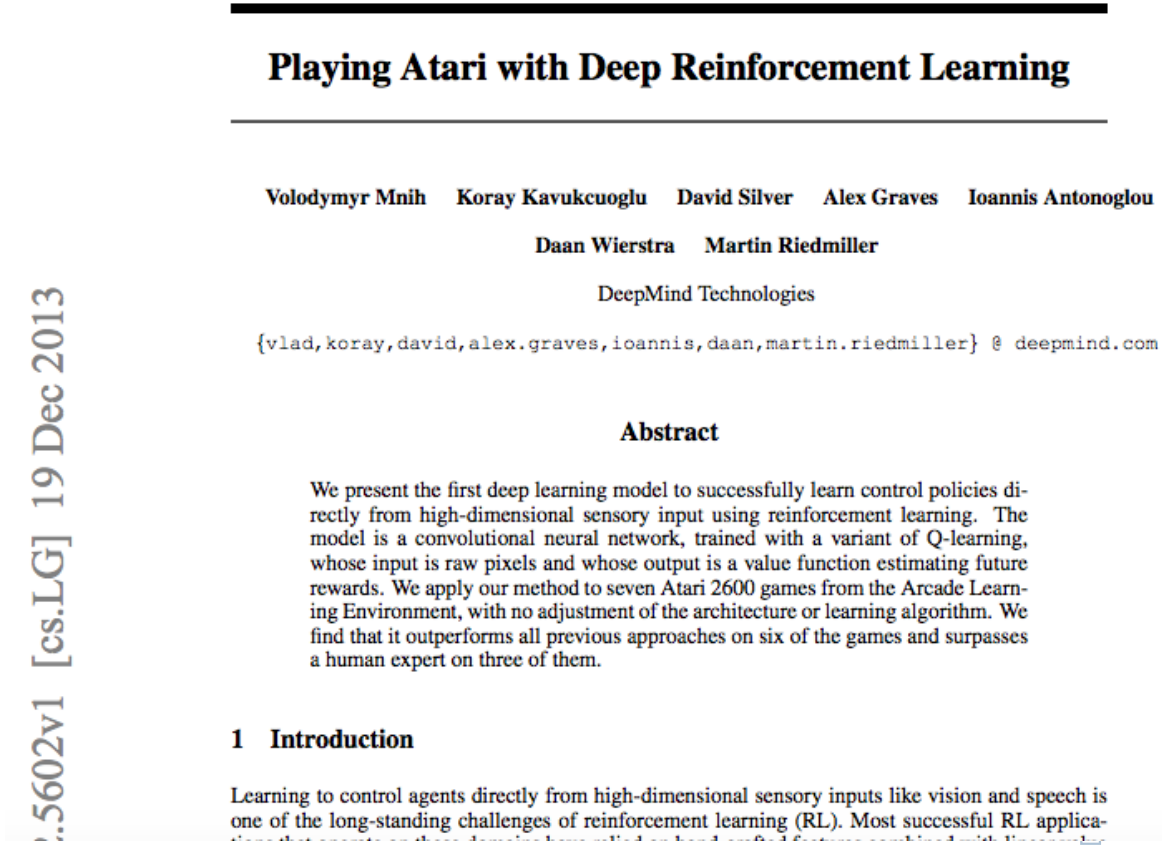
There are a large number of open source deep learning libraries, but TensorFlow is one of the most popular (Theano, Torch, Caffe). Can be coded directly or using a higher level API (Keras).

Provides many functions to deal with network architecture.

```
convolution_layer = tf.contrib.layers.convolution2d()
```

DQN

DeepMind first introduced Deep Q-Networks (DQN). DQN introduced several important innovations: deep convolution network, experience replay, and a second target network. Has since been extended in many ways including Double DQN and Dueling DQN.



arXiv:1312.5602v1 [cs.LG] 19 Dec 2013

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**
Daan Wierstra **Martin Riedmiller**
DeepMind Technologies
{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

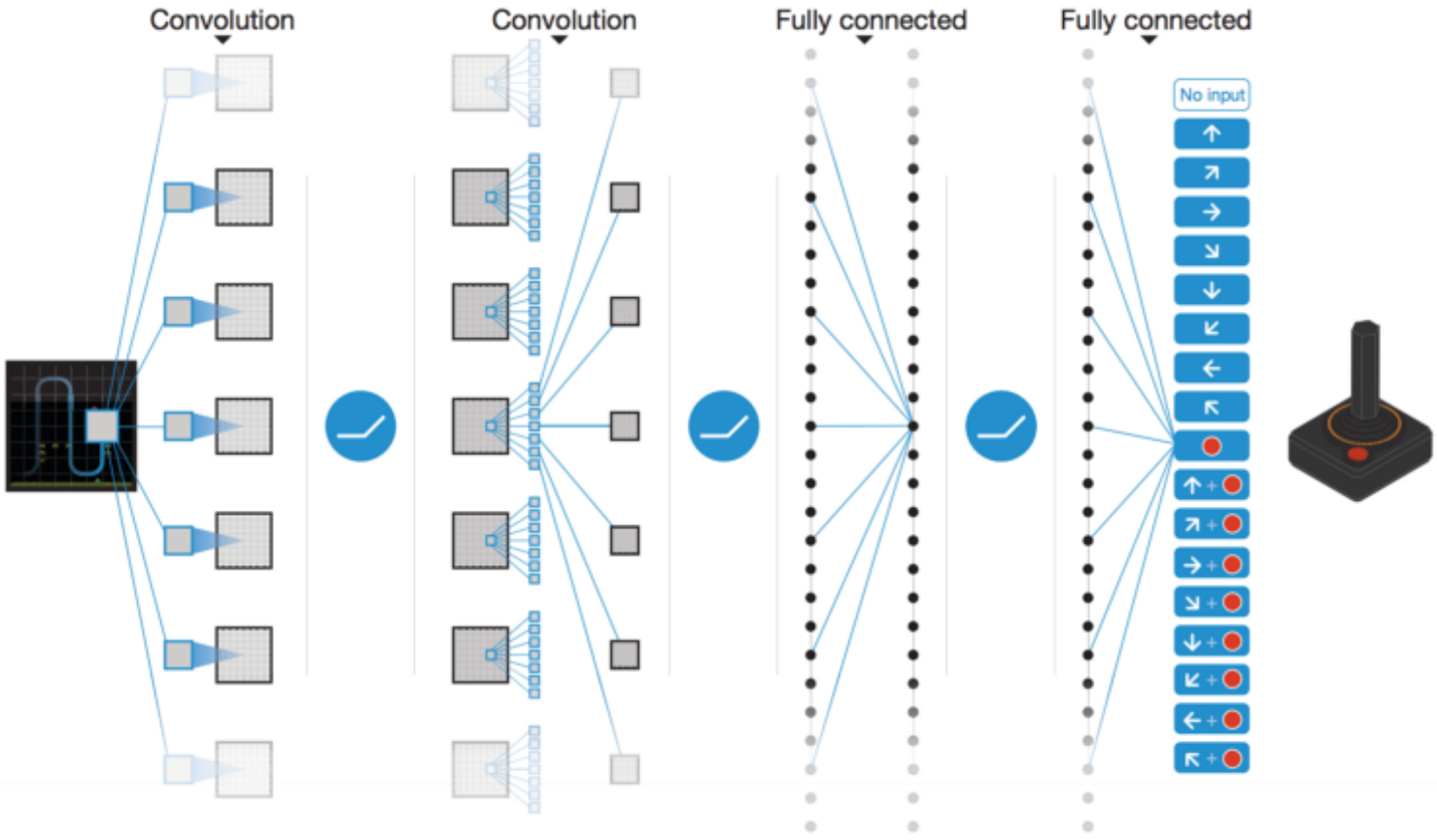
Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

1 Introduction

Learning to control agents directly from high-dimensional sensory inputs like vision and speech is one of the long-standing challenges of reinforcement learning (RL). Most successful RL applica-

DQN Network



Source code from DeepMind

A3C

Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih[1] VMNIH@GOOGLE.COM
Adrià Puigdomènech Badia[1] ADRIAP@GOOGLE.COM
Mehdi Mirza[1,2] MIRZAMOM@IRO.UMONTREAL.CA
Alex Graves[1] GRAVESA@GOOGLE.COM
Tim Harley[1] THARLEY@GOOGLE.COM
Timothy P. Lillicrap[1] COUNTZERO@GOOGLE.COM
David Silver[1] DAVIDSILVER@GOOGLE.COM
Koray Kavukcuoglu [1] KORAYK@GOOGLE.COM

[1] Google DeepMind
[2] Montreal Institute for Learning Algorithms (MILA), University of Montreal

1783v2 [cs.LG] 16 Jun 2016

Abstract

We propose a conceptually simple and lightweight framework for deep reinforcement learning that uses asynchronous gradient descent for optimization of deep neural network controllers. We present asynchronous variants of four standard reinforcement learning algorithms and show that parallel actor-learners have a stabilizing effect on training allowing all four methods to successfully train neural network controllers. The best performing method, an asynchronous variant of actor-critic, surpasses the current state-of-the-art on the Atari domain while training for half the time on a single

line RL updates are strongly correlated. By storing the agent's data in an experience replay memory, the data can be batched (Riedmiller, 2005; Schulman et al., 2015a) or randomly sampled (Mnih et al., 2013; 2015; Van Hasselt et al., 2015) from different time-steps. Aggregating over memory in this way reduces non-stationarity and decorrelates updates, but at the same time limits the methods to off-policy reinforcement learning algorithms.

Deep RL algorithms based on experience replay have achieved unprecedented success in challenging domains such as Atari 2600. However, experience replay has several drawbacks: it uses more memory and computation per real interaction; and it requires off-policy learning algorithms that can update from data generated by an older policy.

Advantage Actor Critic

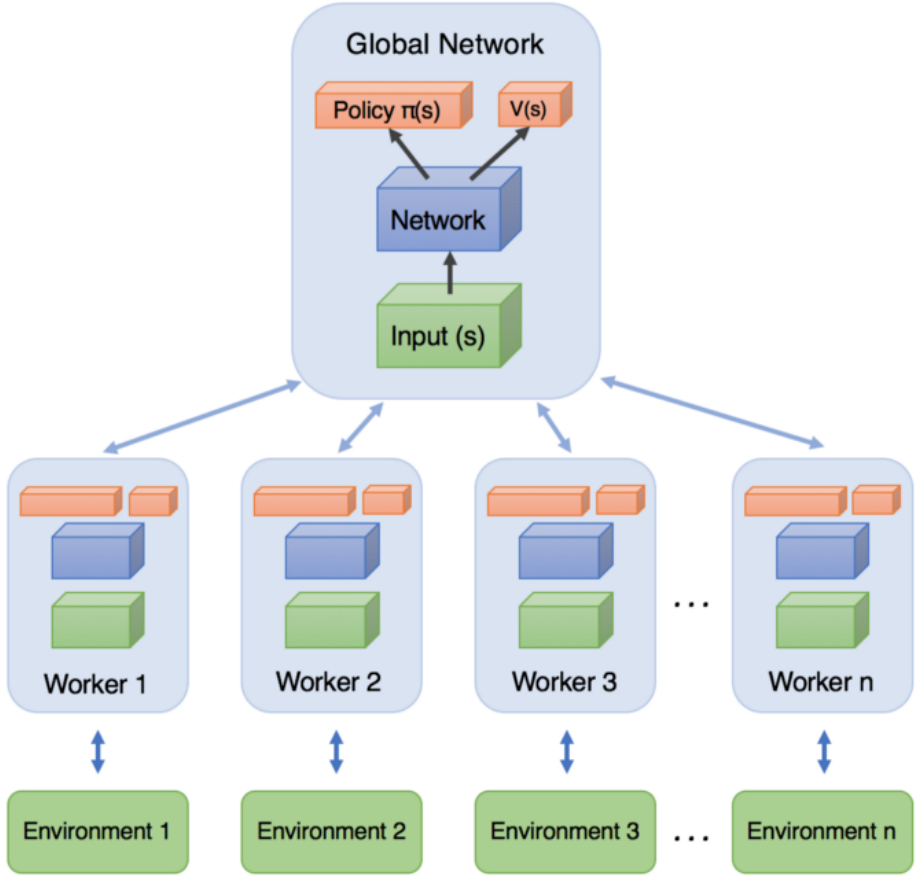
The policy gradient has many different forms:

- REINFORCE: $\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log \pi_\theta(s, a) v_t]$
- Q Actor-Critic: $\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log \pi_\theta(s, a) Q^w(s, a)]$
- Advantage Actor-Critic: $\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log \pi_\theta(s, a) A^w(s, a)]$

A3C uses an Advantage Actor-Critic model, using neural networks to learn both the policy and the advantage function $A^w(s, a)]$.

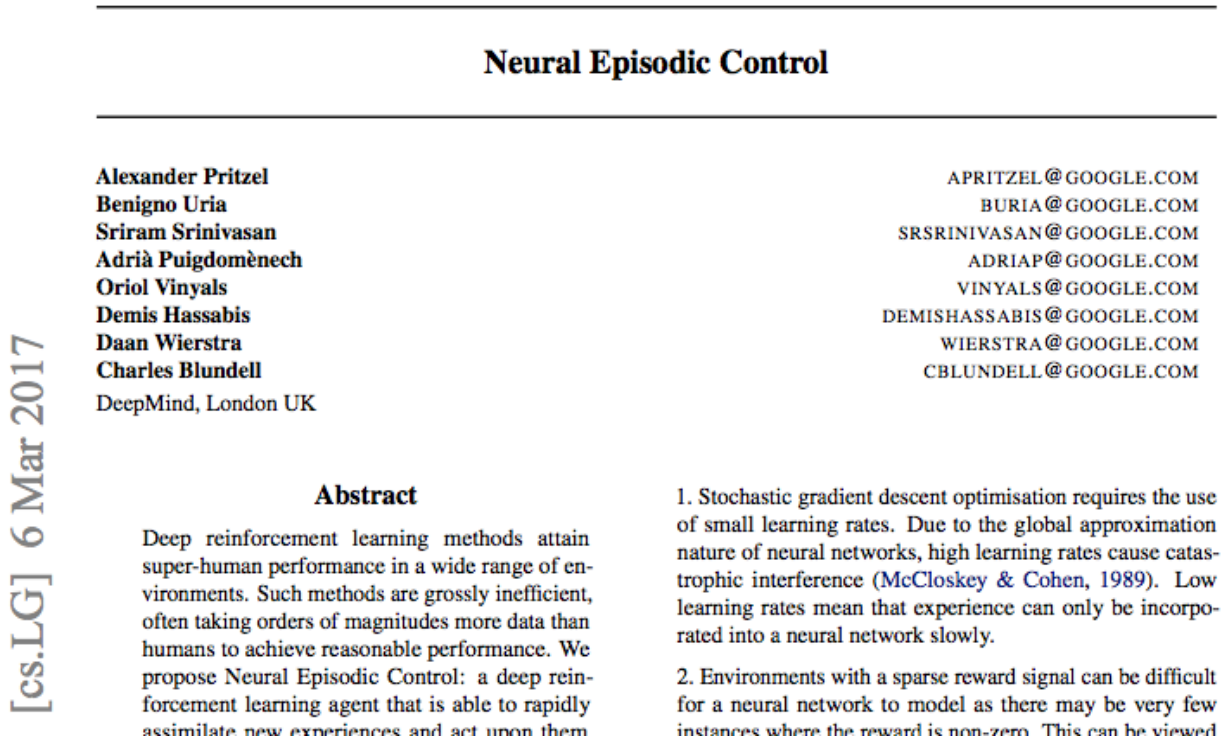
A3C Algorithm

The A3C algorithm parallelizes single episodes, and then aggregates learning to a global network.



NEC

Neural Episodic Control (NEC) addresses the problem that RL algorithms requires a very large number of interactions to learn, by trying to learn from single examples.



[cs.LG] 6 Mar 2017

Neural Episodic Control

Alexander Pritzel APRITZEL@GOOGLE.COM
Benigno Uria BURIA@GOOGLE.COM
Sriram Srinivasan SRSRINIVASAN@GOOGLE.COM
Adrià Puigdomènech ADRIAP@GOOGLE.COM
Oriol Vinyals VINYALS@GOOGLE.COM
Demis Hassabis DEMISHASSABIS@GOOGLE.COM
Daan Wierstra WIERSTRA@GOOGLE.COM
Charles Blundell CBLUNDELL@GOOGLE.COM
DeepMind, London UK

Abstract

Deep reinforcement learning methods attain super-human performance in a wide range of environments. Such methods are grossly inefficient, often taking orders of magnitudes more data than humans to achieve reasonable performance. We propose Neural Episodic Control: a deep reinforcement learning agent that is able to rapidly assimilate new experiences and act upon them

1. Stochastic gradient descent optimisation requires the use of small learning rates. Due to the global approximation nature of neural networks, high learning rates cause catastrophic interference (McCloskey & Cohen, 1989). Low learning rates mean that experience can only be incorporated into a neural network slowly.

2. Environments with a sparse reward signal can be difficult for a neural network to model as there may be very few instances where the reward is non-zero. This can be viewed

Example code: [1], [2]

Outline

OpenAI

OpenAI has released a number of tools:

- gym
- universe
- roboschool
- baselines

See https://openai.com/systems/ for complete details.

gym

```
import gym
from gym import wrappers

env = gym.make("FrozenLake-v0")
env = wrappers.Monitor(env, "/tmp/gym-results")
observation = env.reset()
for _ in range(1000):
    env.render()
    action = env.action_space.sample() # your agent here (t
    observation, reward, done, info = env.step(action)
    if done:
        env.reset()

env.close()
gym.upload("/tmp/gym-results", api_key="YOUR_API_KEY")
```

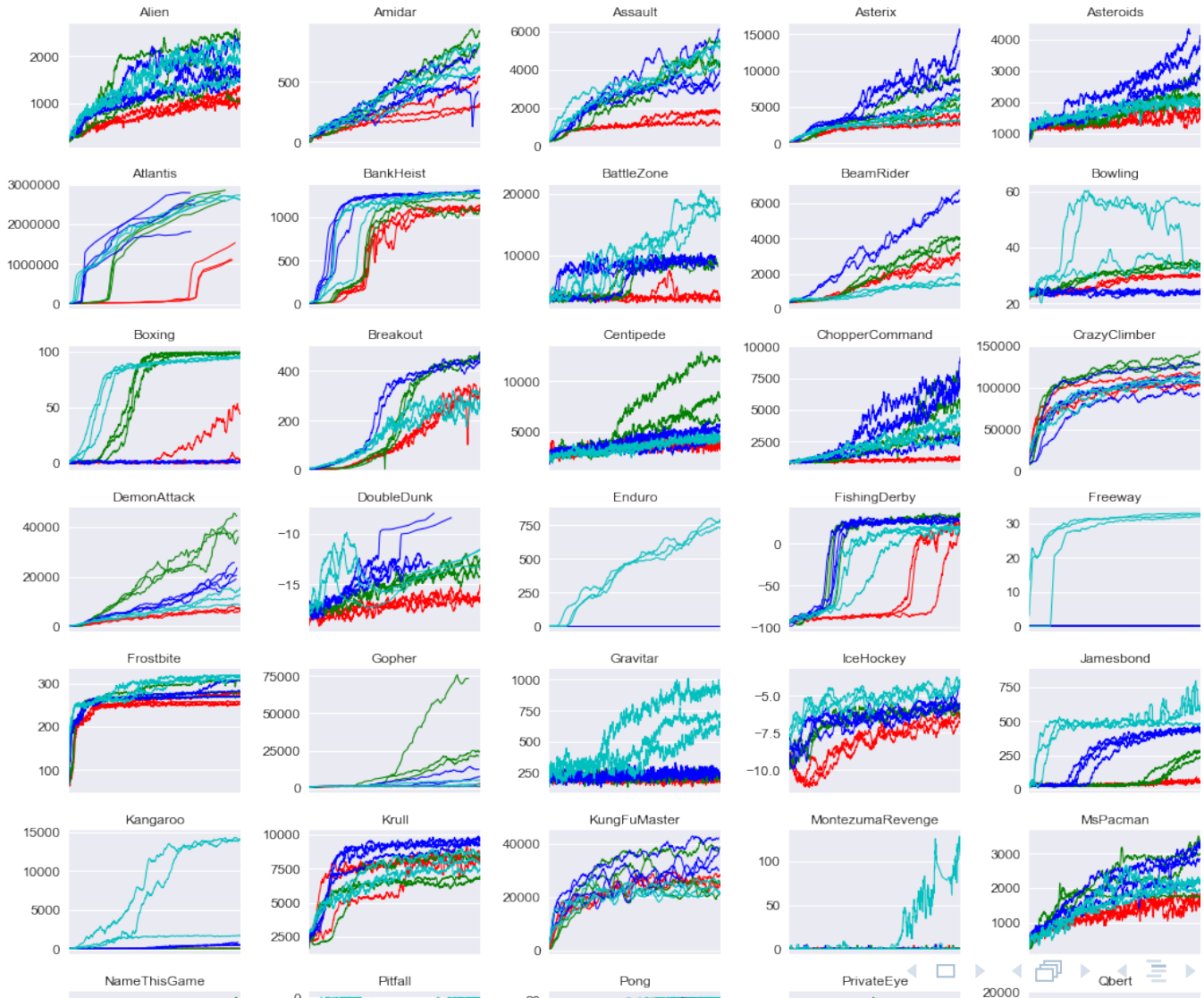
Baselines

OpenAI recent started a new project called Baselines, which provides good implementations of state-of-the-art (SOTA) agents.

- Source code: https://github.com/openai/baselines
- DQN discussion: https://blog.openai.com/openai-baselines-dqn/
- A2C and ACKTR discussion: https://blog.openai.com/baselines-acktr-a2c/

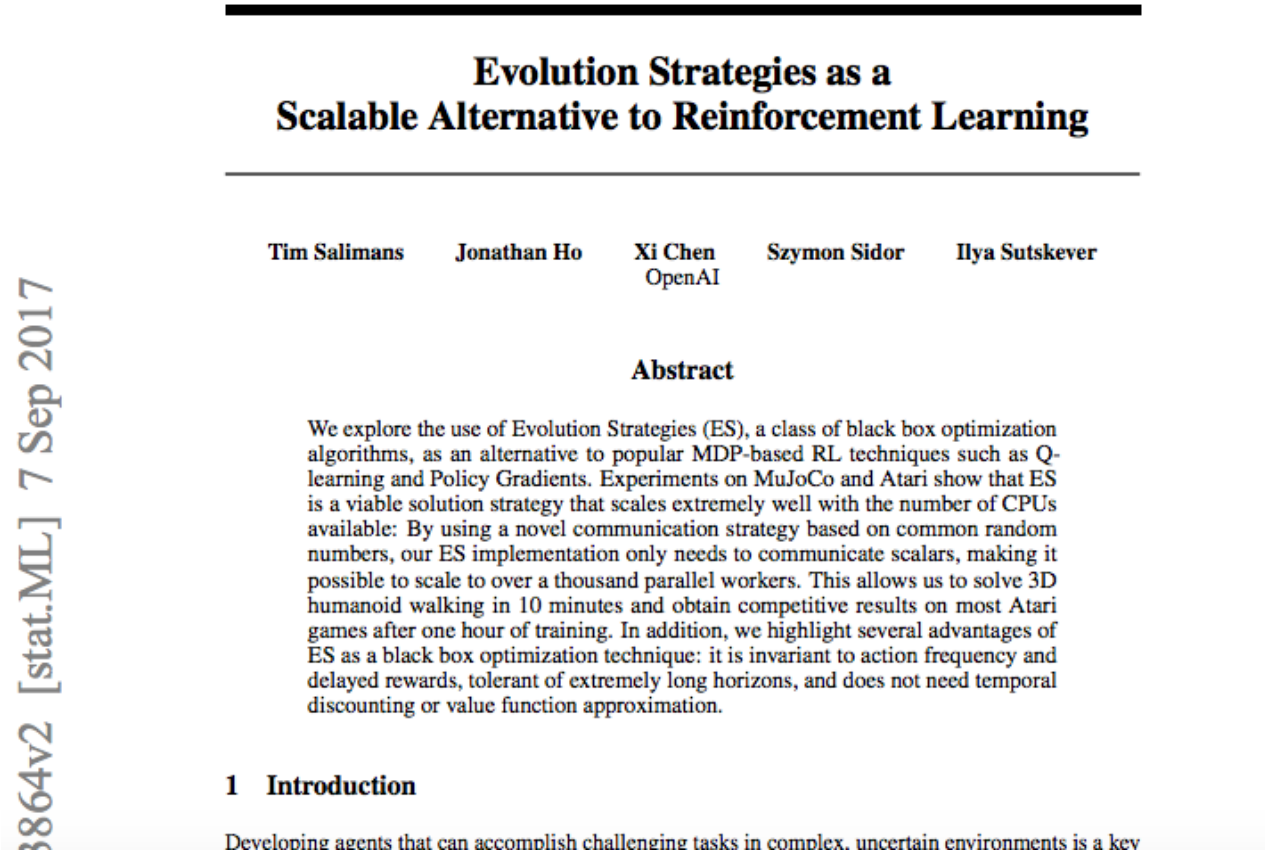
Baselines

Baselines provides examples of comparing the performance of different agents across many environments:



Evolutionary Strategies

There are alternatives to solving these problems, including Evolutionary Strategies.



Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Tim Salimans **Jonathan Ho** **Xi Chen** **Szymon Sidor** **Ilya Sutskever**
OpenAI

3864v2 [stat.ML] 7 Sep 2017

Abstract

We explore the use of Evolution Strategies (ES), a class of black box optimization algorithms, as an alternative to popular MDP-based RL techniques such as Q-learning and Policy Gradients. Experiments on MuJoCo and Atari show that ES is a viable solution strategy that scales extremely well with the number of CPUs available: By using a novel communication strategy based on common random numbers, our ES implementation only needs to communicate scalars, making it possible to scale to over a thousand parallel workers. This allows us to solve 3D humanoid walking in 10 minutes and obtain competitive results on most Atari games after one hour of training. In addition, we highlight several advantages of ES as a black box optimization technique: it is invariant to action frequency and delayed rewards, tolerant of extremely long horizons, and does not need temporal discounting or value function approximation.

1 Introduction

Developing agents that can accomplish challenging tasks in complex, uncertain environments is a key

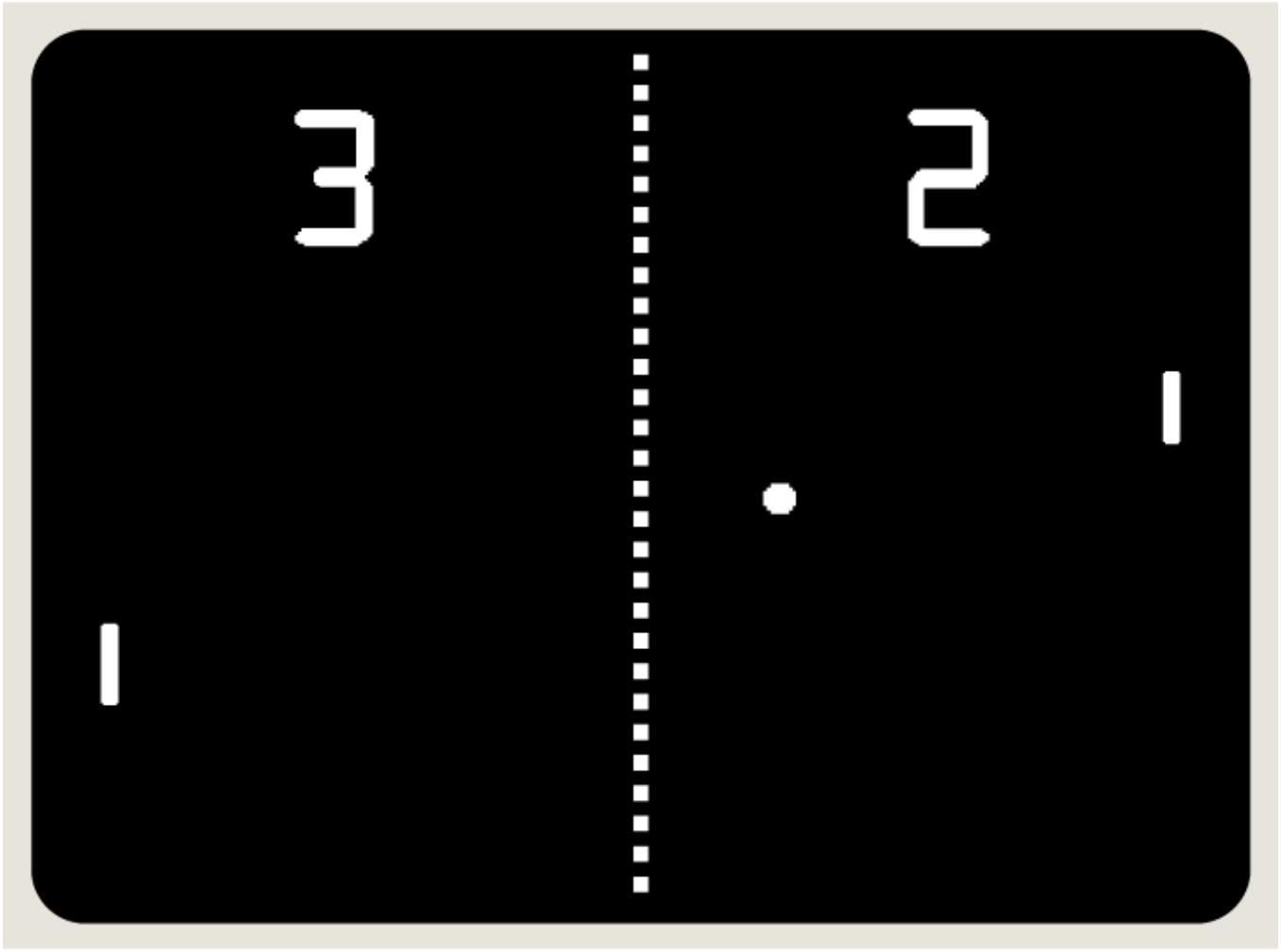
Example code: [1], [2]

Outline

Pixels in...winning out!

Pong

Pong (1972) was the first commercially successful arcade game.

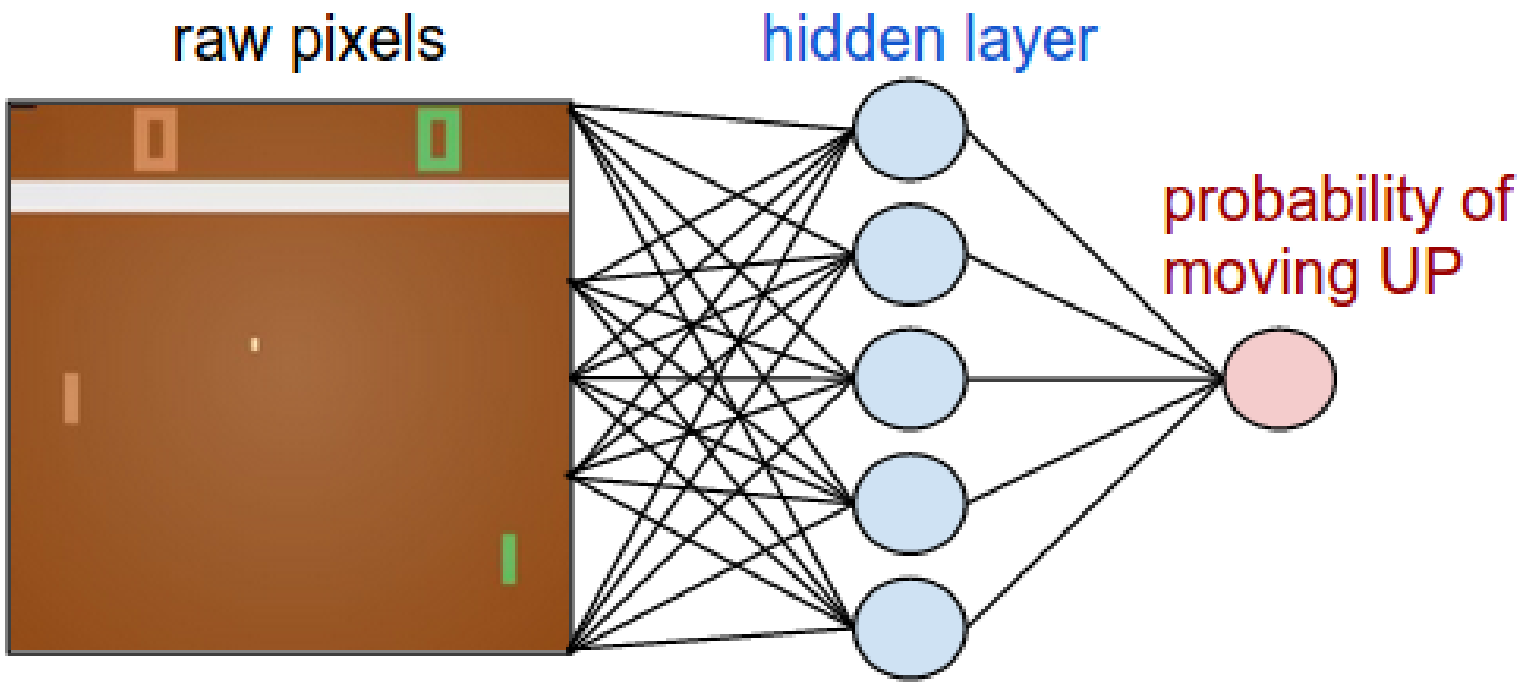


RL Structure

The game structure is fairly simple:

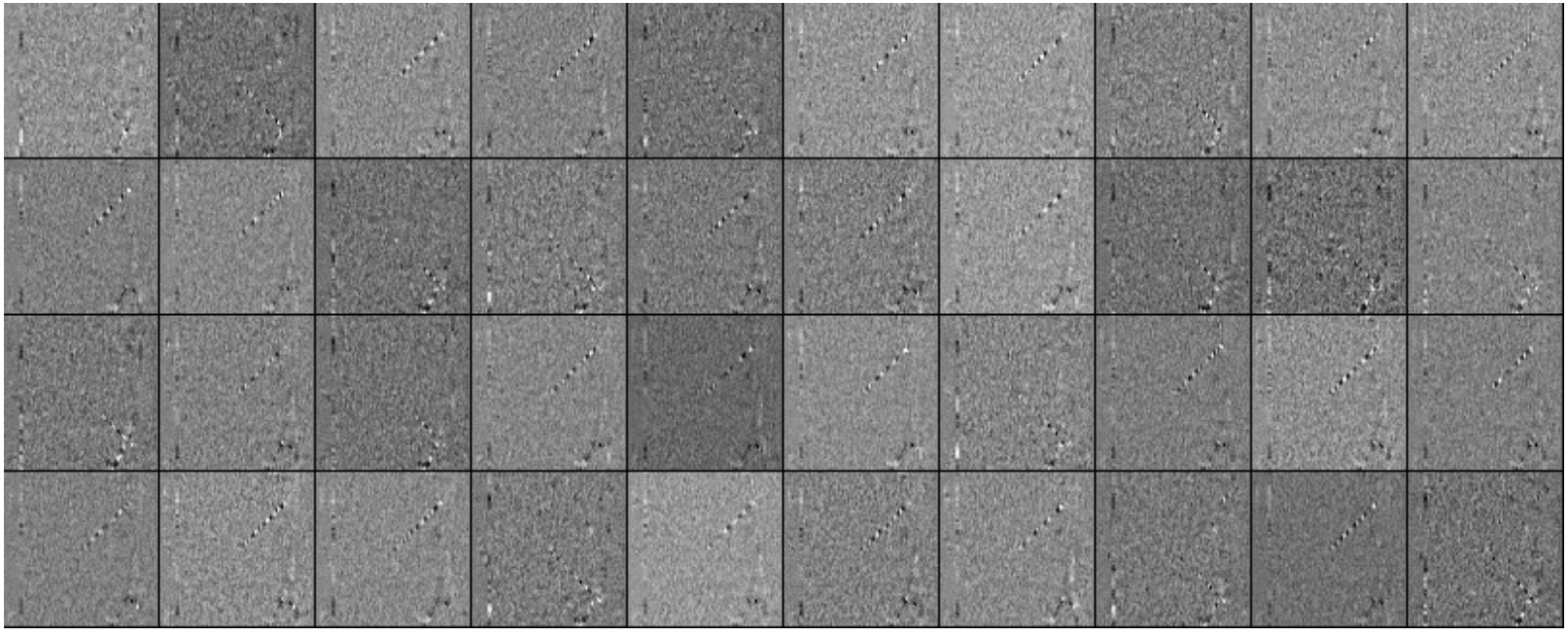
- We get a *reward* of $+1$ for every game win and -1 for every loss. An episode is defined as when the game ends (first to 21 points).
- The *actions* are limited to up and down.
- The *state* (and features) characterizes the entire board, including our paddle position.
- Before we take an action, we reduce the state space by elminating unimportant aspects of the pixels.

Policy network



Source: @karpathy

Network Weights



Source: @karpathy

Outline

Resources: Teams

Reinforcement learning research is being conducted by a combination of academic and industrial groups, all of which support open source/publishing.

- University of Alberta: http://spaces.facsci.ualberta.ca/rlai/
- OpenAI: https://openai.com/
- DeepMind: https://deepmind.com/

Resources: Courses/Tutorials

A number of recent courses and tutorials provide more detail on topics that we discussed:

- David Silver (UCL): "Reinforcement Learning"
- Sergey Levine, John Schulman, Chelsea Finn (Berkeley): "Deep Reinforcement Learning"
- Katerina Fragkiadaki, Ruslan Satakhutdinov (CMU): "Deep Reinforcement Learning and Control"

- David Silver: Deep Reinforcement Learning tutorial at ICML 2016
- John Shulman: Deep Reinforcement Learning tutorial at the Deep Learning School 2016

Resources: Code

The OpenAI code is written in Python:
https://openai.com/systems/

There are also wrappers for Juila and R:

- R: https://cran.r-project.org/web/packages/gym/index.html
- Julia: https://github.com/JuliaML/OpenAIGym.jl

Resources: Reading

There are many great materials online. Sutton/Barto wrote the classic text, which is now fully available:

- *Sutton/Barto "Reinforcement Learning: An Introduction"*
- Andrej Karpathy: Deep Reinforcement Learning: Pong from Pixels
- Arthur Juliani: 8-Part Series on "Reinforcement Learning with Tensorflow"
- Denny Britz: Learning Reinforcement Learning (with Code, Exercises and Solutions)