Deep Reinforcement Learning

December 8, 2022

ddebarr@uw.edu

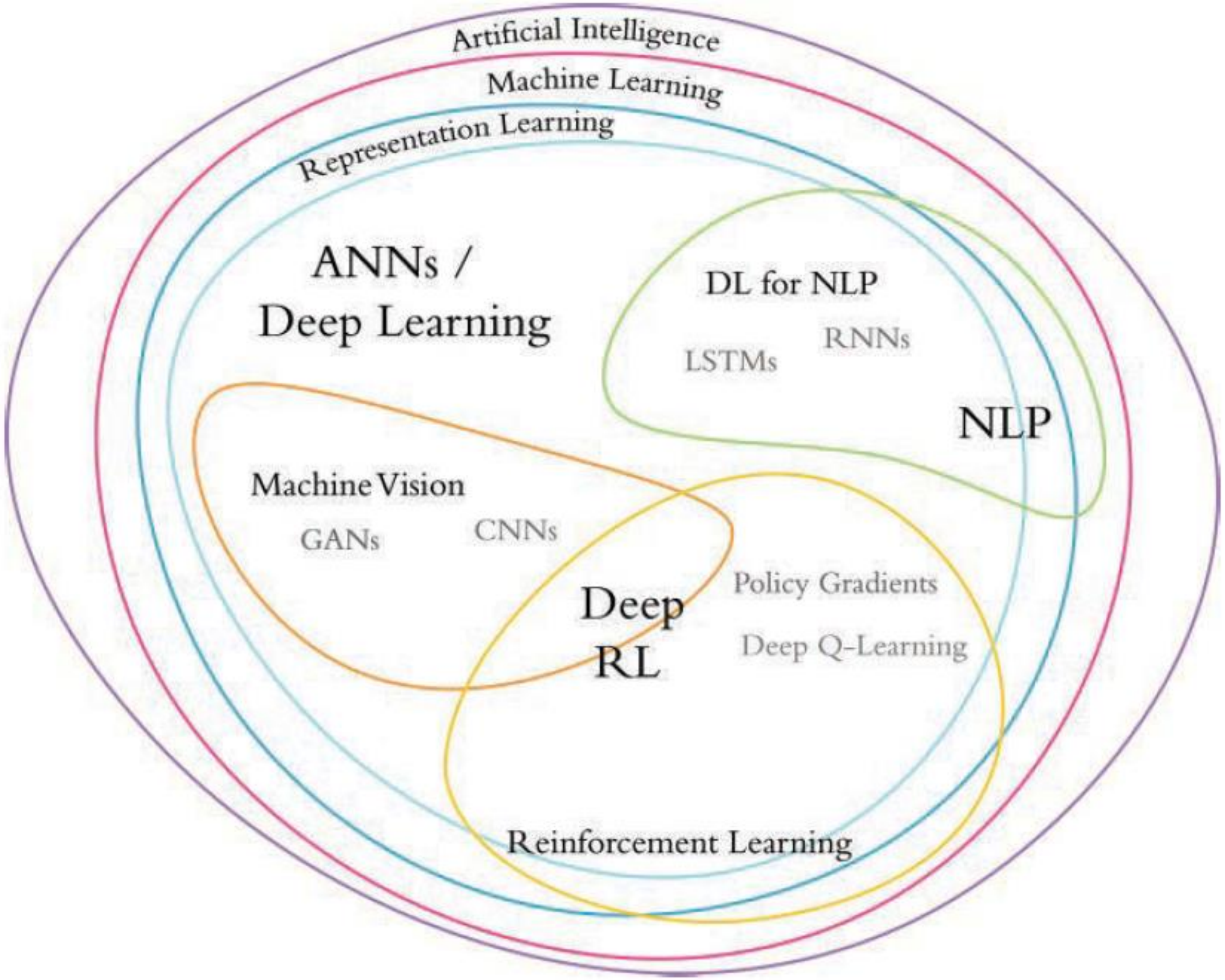
http://cross-entropy.net/ML530/Deep_Learning_7.pdf

[DLI] Game-Playing Machines

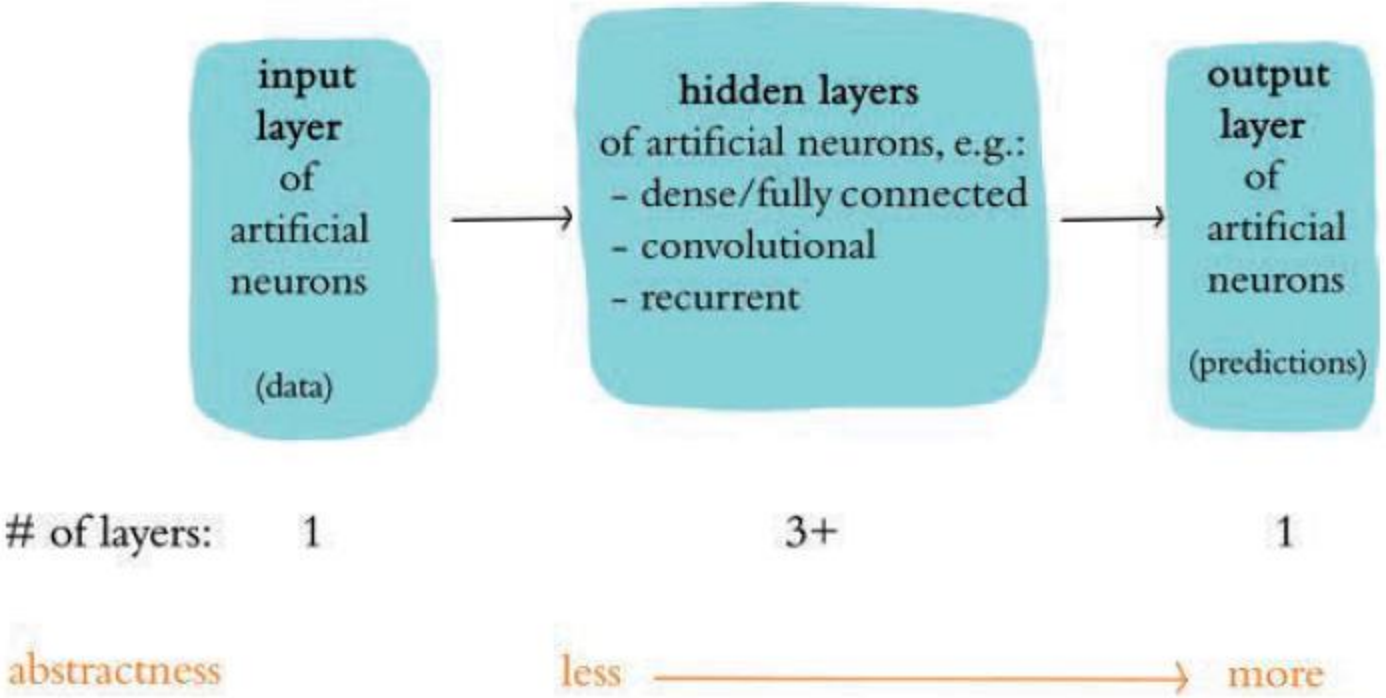
- Deep Learning, AI, and Other Beasts
- Three Categories of Machine Learning Problems
- Deep Reinforcement Learning
- Video Games
- Board Games
- Manipulation of Objects
- Popular Deep Reinforcement Learning Environments
- Three Categories of AI
- Summary

Venn Diagram of Concepts from Textbook

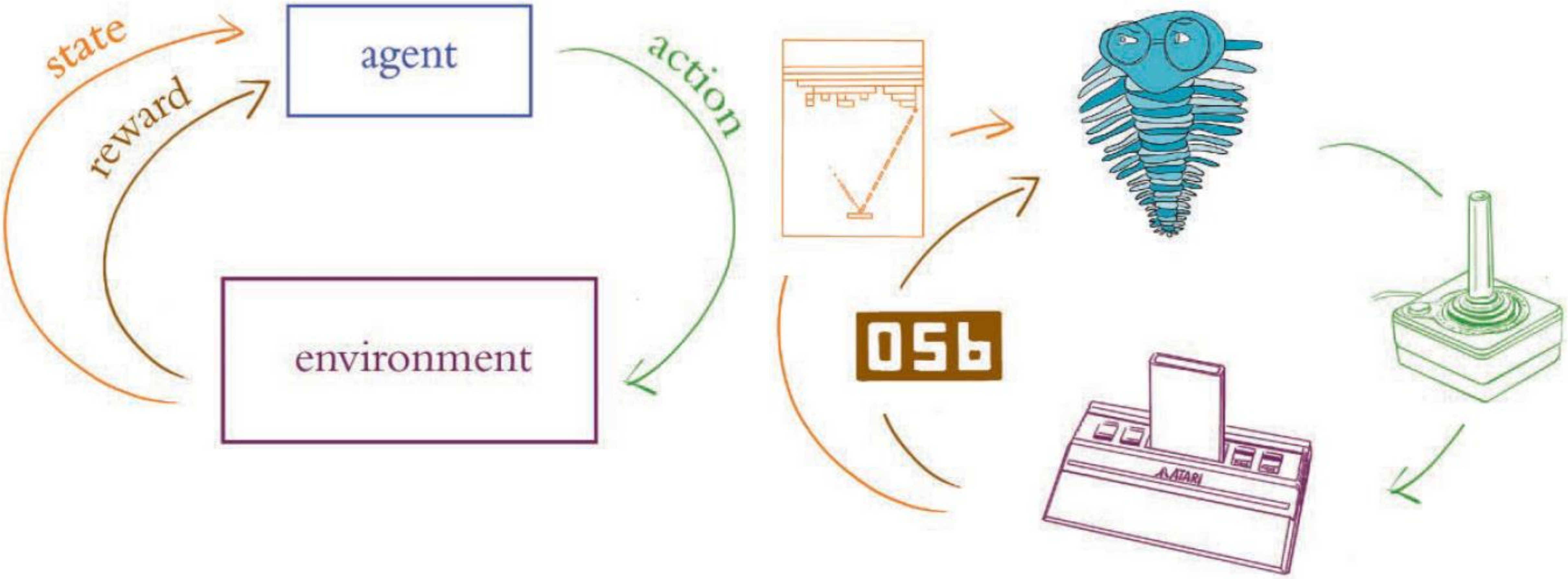
7 boundaries:
Artificial Intelligence
Machine Learning
Representation Learning
Deep Learning
Machine Vision
Natural Language Processing
Reinforcement Learning



Generalization of Deep Learning Architectures



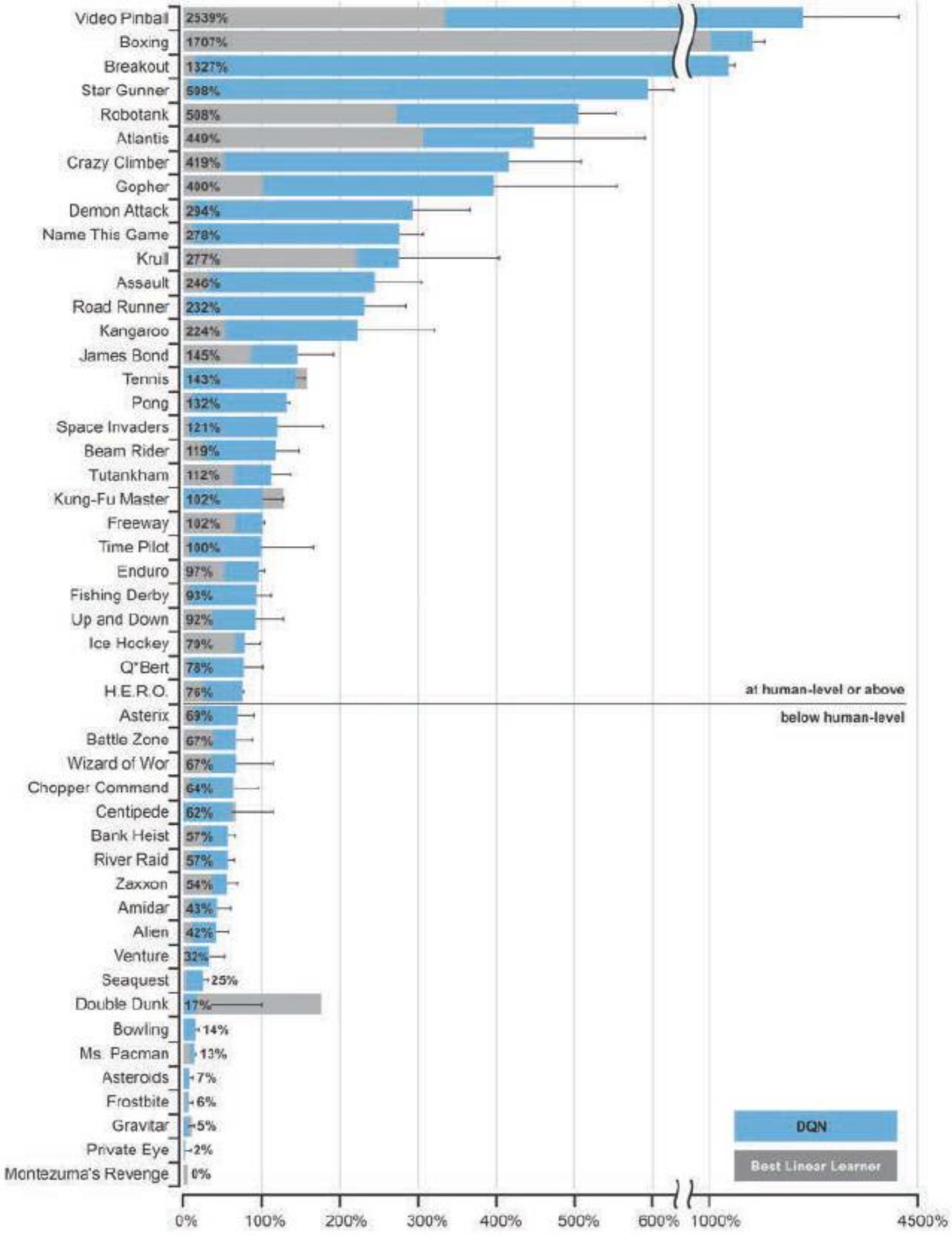
Reinforcement Learning Loop



Demis Hassabis

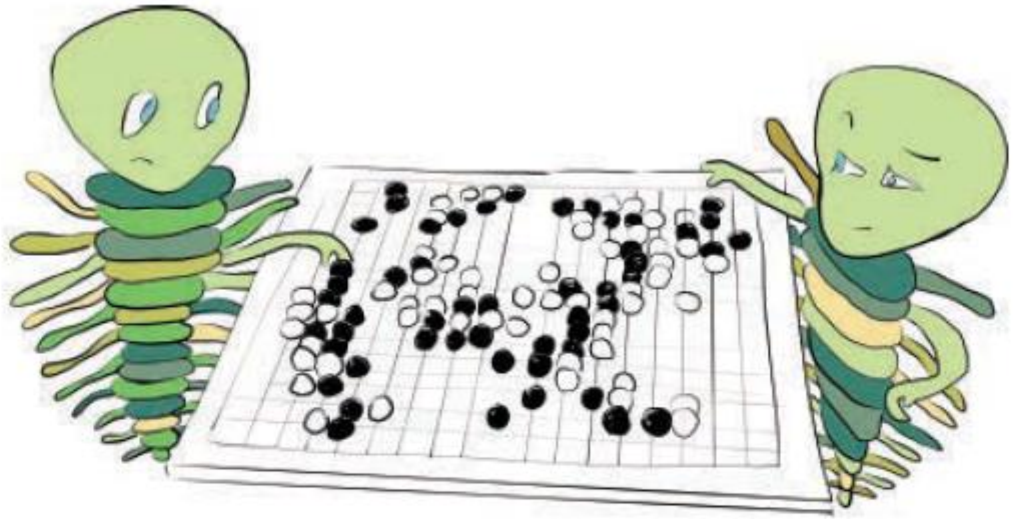
Cofounder of DeepMind

Deep Q(uality) Network Performance on Atari Games [relative to game tester]



Go Game Board

Objective is to encircle your opponent’s stones (capturing them)



David Silver

Researcher at DeepMind

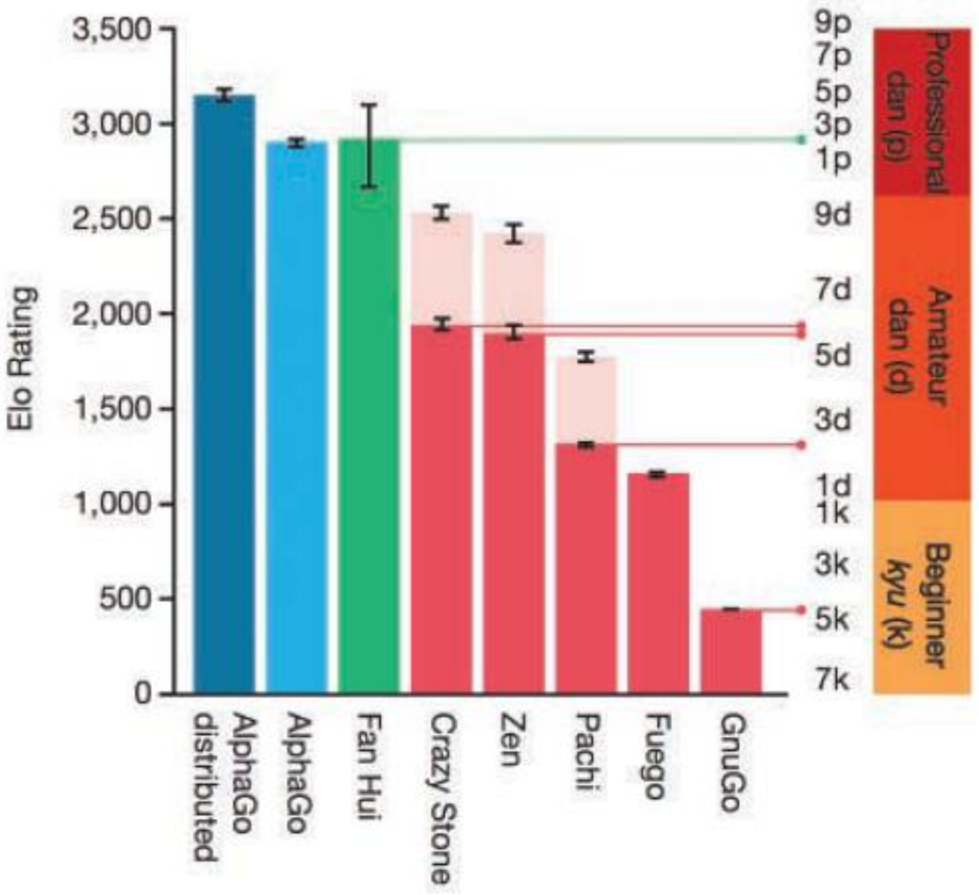
Elo Score of AlphaGo

Example algorithm for Elo Rating:

1. For each win, add your opponent's rating plus 400,
2. For each loss, add your opponent's rating minus 400,
3. And divide this sum by the number of played games.

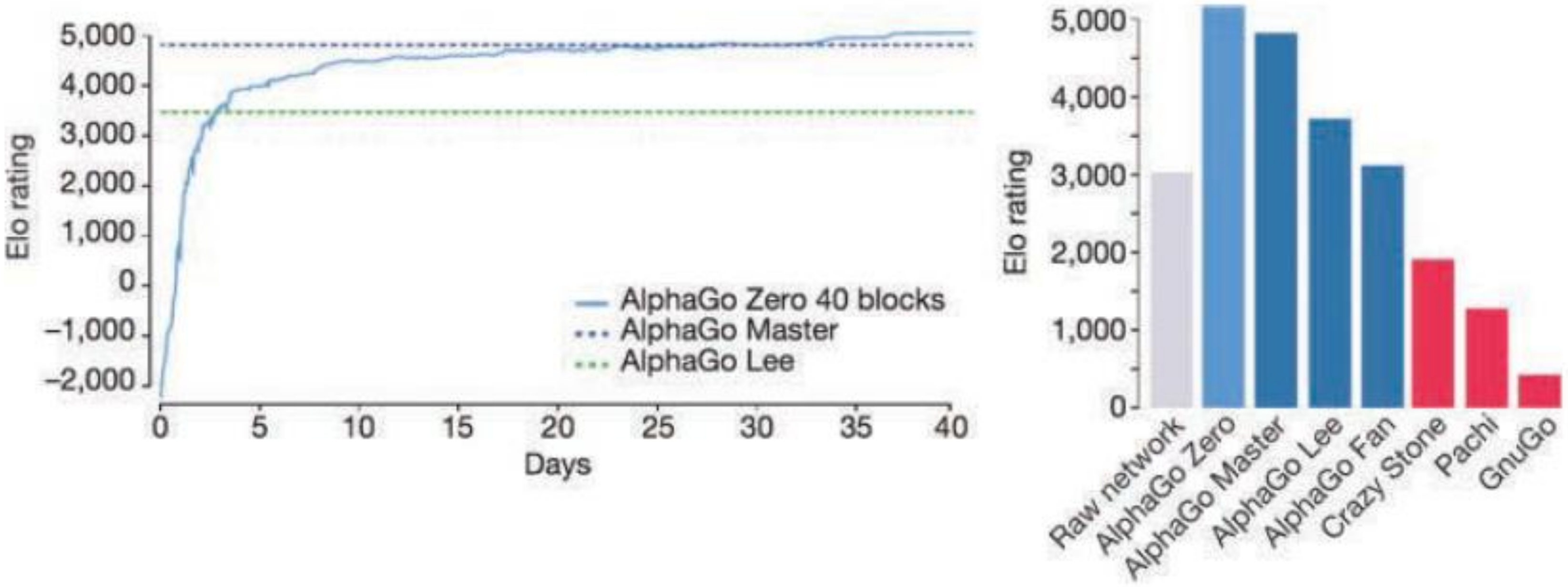
Fan Hui was the European Go Champion

Beginner kyu [kai you]
Amateur dan
Professional dan



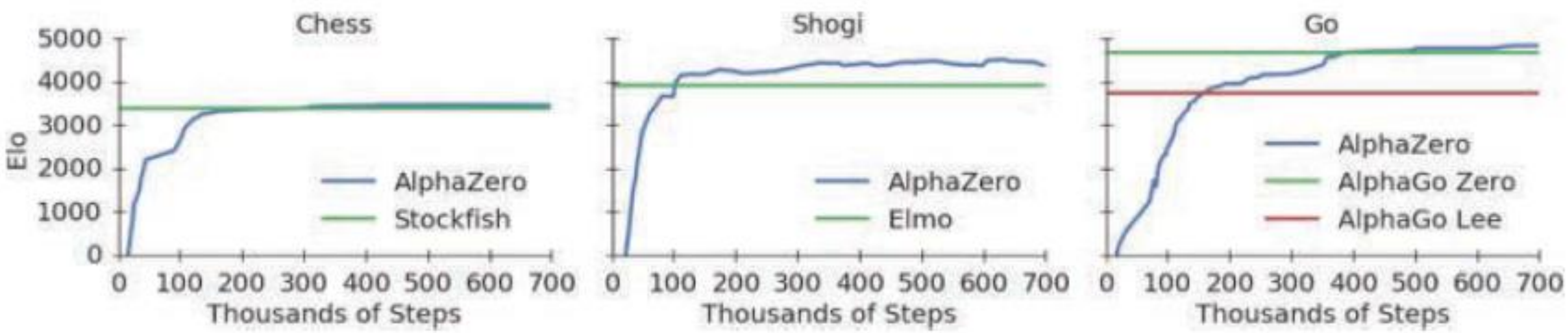
AlphaGo Zero versus AlphaGo

Lee Sedol was a world-champion Go player



Alpha Zero Elo Ratings

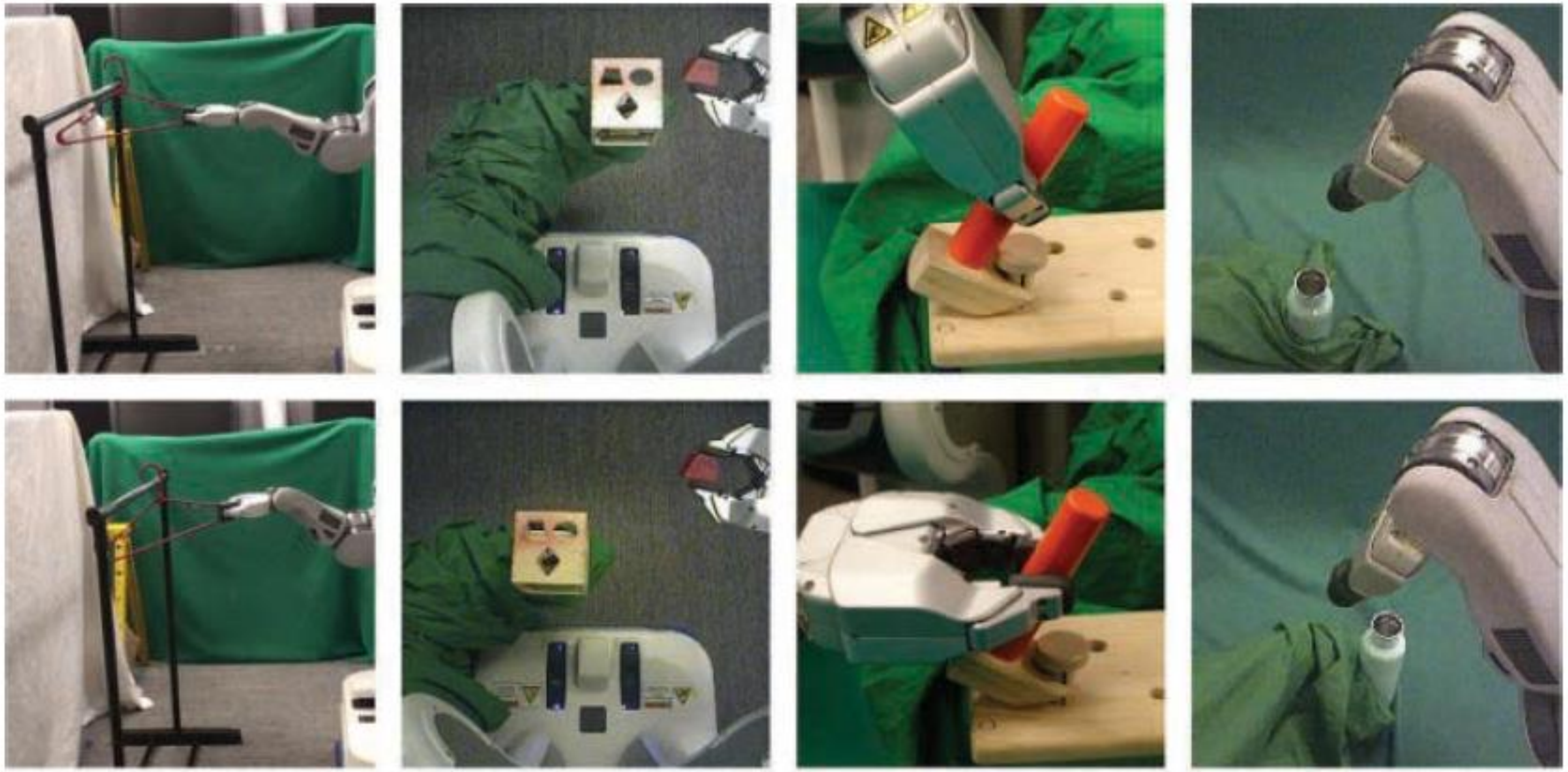
Higher rating than opponent means you're more likely to win



Chelsea Finn

Professor of CSE at Stanford

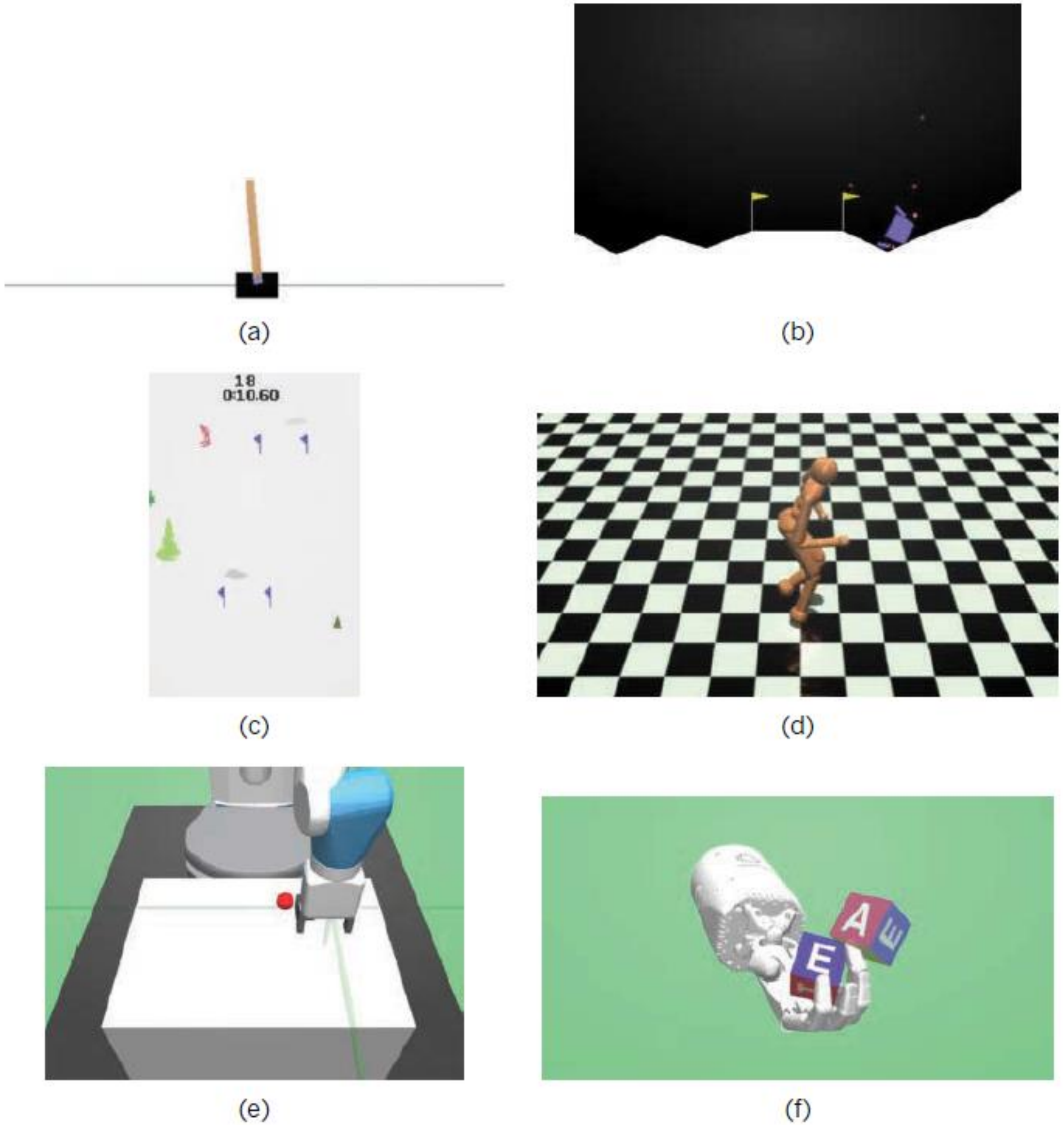
Sample Images from Work of Finn et al



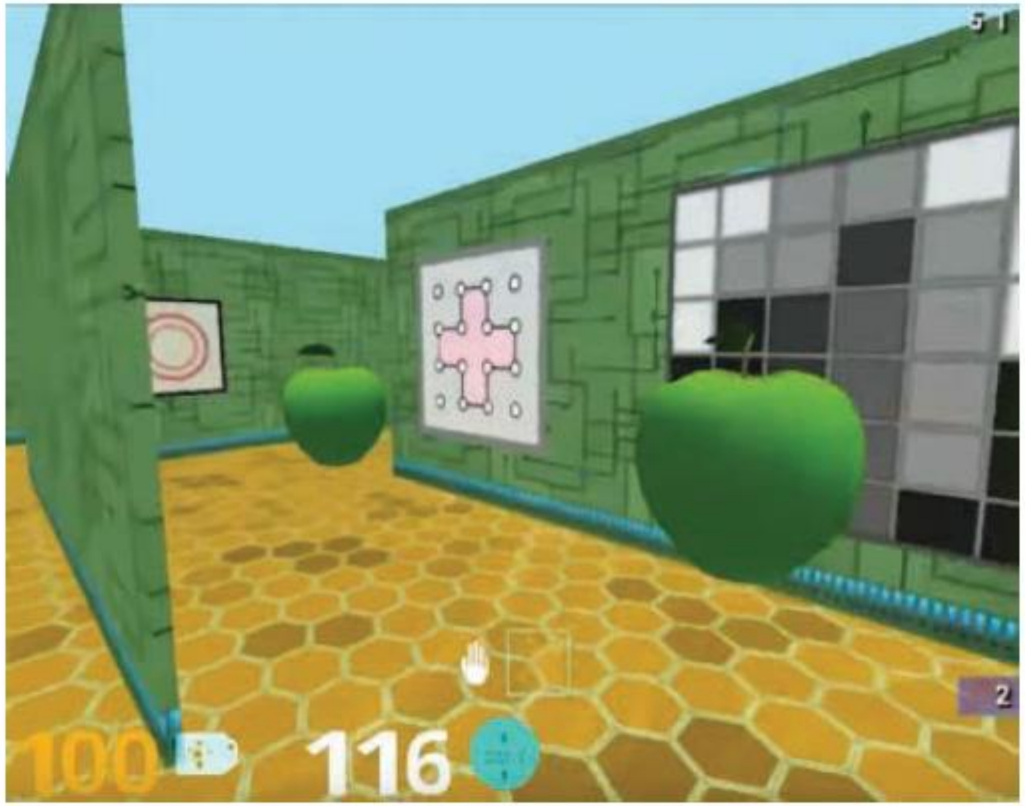
(a) hanger (b) cube (c) hammer (d) bottle

Sample of OpenAI Gym Environments

- CartPole
- LunarLander
- Skiing
- Humanoid
  - MuJoCo
  - Multi Joint dynamics with Contact
- FetchPickAndPlace
- HandManipulateBlock



DeepMind Lab Environment



Three Categories of AI

- ANI: Artificial Narrow Intelligence

  Specifically trained for a task; e.g. text classification, image classification, object detection, machine translation, speech recognition, question answering, games, etc.

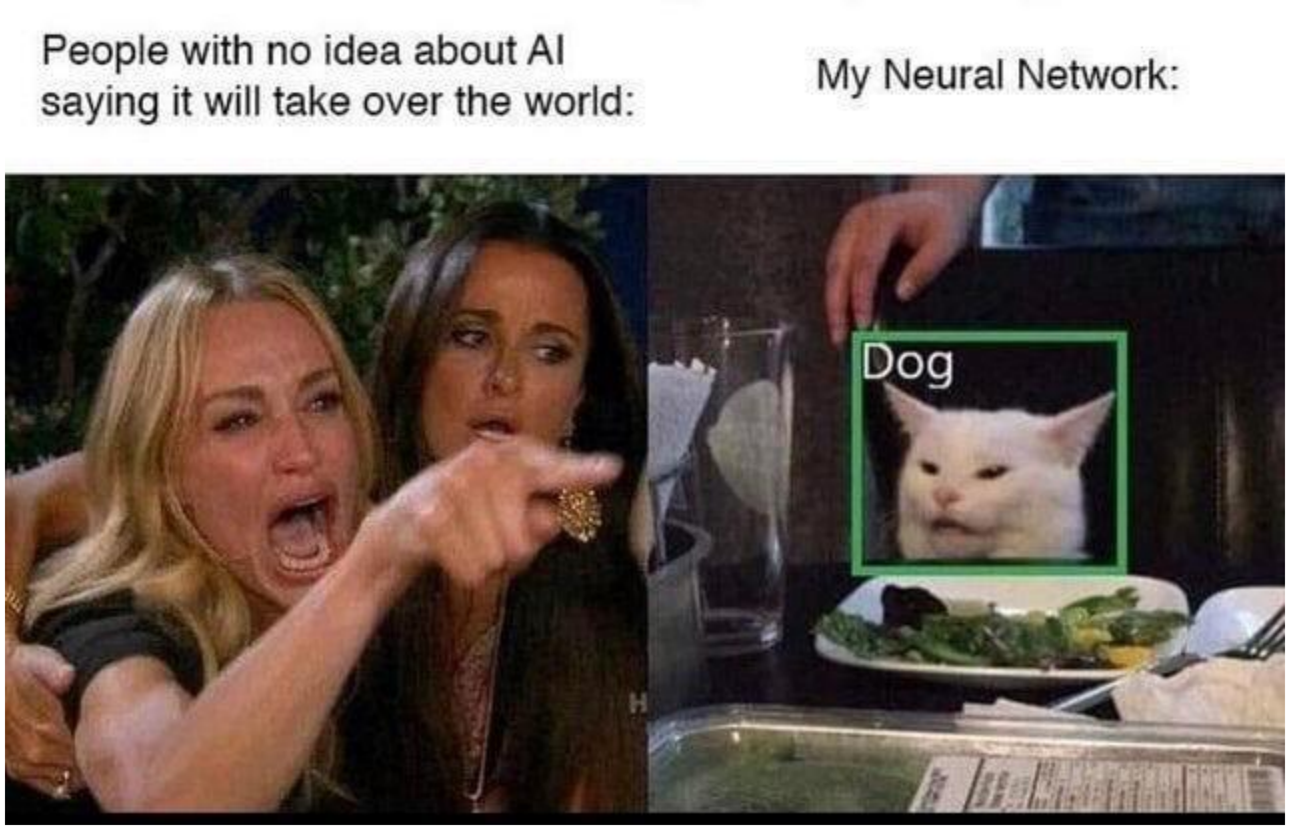
- AGI: Artificial General Intelligence

  Able to generalize; comparable to human intelligence

- ASI: Artificial Super Intelligence

  https://www.nickbostrom.com/papers/survey.pdf

  - “The median estimate of respondents was for a one-in-two chance that high-level machine intelligence will be developed around 2040-2050, rising to a nine-in-ten chance by 2075”
  - “1970s: … it was believed by some of the early pioneers of artificial intelligence and robotics (at places such as MIT, Stanford, and CMU) that solving the 'visual input' problem would be an easy step along the path to solving more difficult problems such as higher-level reasoning and planning“: Richard Szeliski, in “Computer Vision: Algorithms and Applications”
  - Be nice to Alexa … just in case ☺



Source: posted to the Memes/Cartoons channel for NeurIPS 2019

Summary

The chapter began with an overview relating deep learning to the broader field of artificial intelligence. We then detailed deep reinforcement learning, an approach that blends deep learning with the feedback-providing reinforcement learning paradigm. As discussed via real-world examples ranging from the board game Go to the grasping of physical objects, such deep reinforcement learning enables machines to process vast amounts of data and take sensible sequences of actions on complex tasks, associating it with popular conceptions of AI.

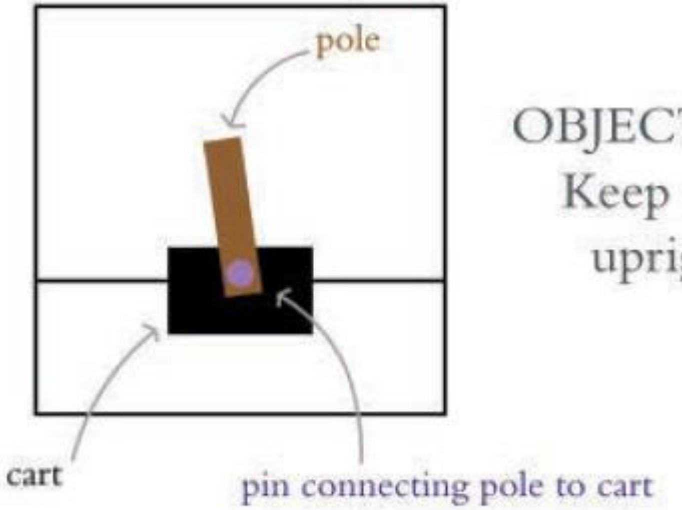
[DLI] Deep Reinforcement Learning

- Essential Theory of Reinforcement Learning
- Essential Theory of Deep Q-Learning Networks
- Defining a DQN Agent
- Interacting with an OpenAI Gym Environment
- Hyperparameter with SLM Lab
- Agents Beyond DQN
- Summary

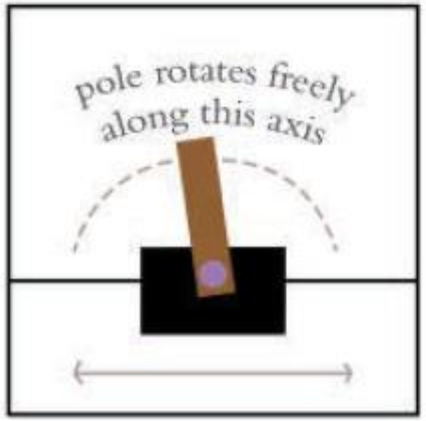
Reinforcement Learning

- An *agent* taking an *action* within an *environment* (let's say the action is taken at some timestep t)
- The environment returning two types of information to the agent:
  - *Reward*: This is a scalar value that provides quantitative feedback on the action that the agent took at timestep t. This could, for example, be 100 points as a reward for acquiring cherries in the video game Pac-Man. The agent's objective is to maximize the rewards it accumulates, and so rewards are what *reinforce* productive behaviors that the agent discovers under particular environmental conditions.
  - *State*: This is how the environment changes in response to an agent's action. During the forthcoming timestep ($t + 1$), these will be the conditions for the agent to choose an action in.
- Repeating the above two steps in a loop until reaching some terminal state. This terminal state could be reached by, for example, attaining the maximum possible reward, attaining some specific desired outcome (such as a self-driving car reaching its programmed destination), running out of allotted time, using up the maximum number of permitted moves in a game, or the agent dying in a game.
- Reinforcement learning problems are sequential decision-making problems
  - Atari video games, such as Pac-Man, Pong, and Breakout
  - Autonomous vehicles, such as self-driving cars and aerial drones
  - Board games, such as Go, chess, and shogi
  - Robot-arm manipulation tasks, such as removing a nail with a hammer

The Cart-Pole Game (control theory problem)

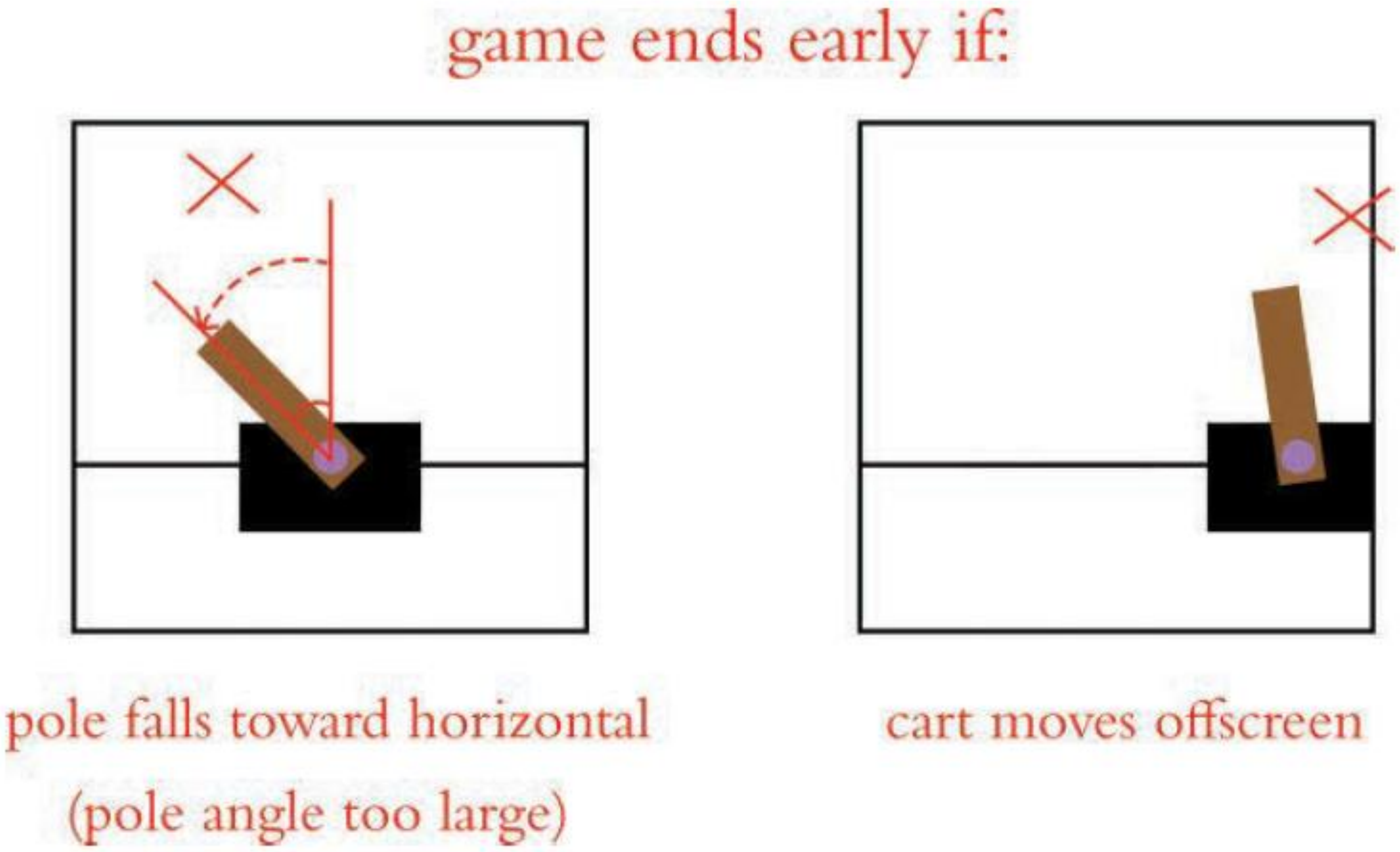


OBJECTIVE:
Keep pole
upright

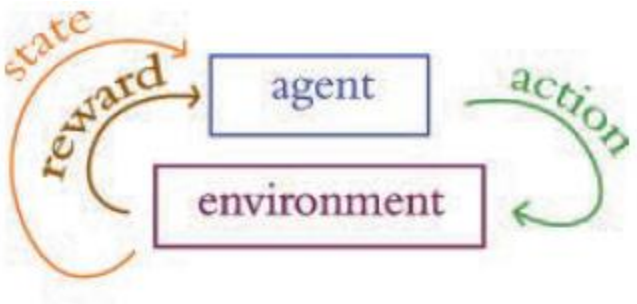


cart is controlled directly by player
(can be moved left or right)

Cart-Pole Game Termination



RL Loop as a Markov Decision Process (MDP)



"Markov Decision Process"

S: all possible states
A: all possible actions
R: reward distribution given (s,a)
P: transition probability to s_{t+1} given (s,a)
γ: discount factor

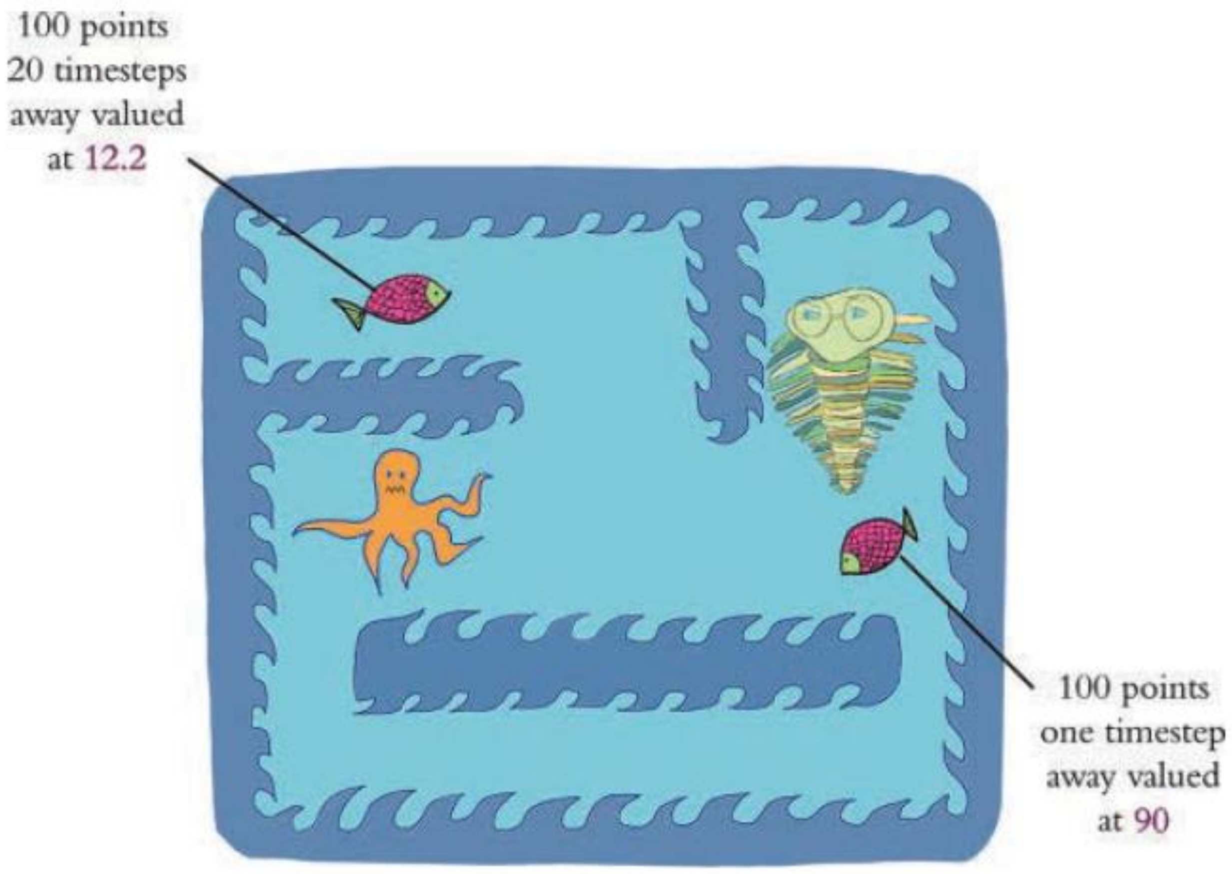
https://github.com/openai/gym/wiki/CartPole-v0

S: {cart position, cart velocity, pole angle, angular velocity for pole tip}
A: { left, right }
R: $P(r|(s_t,a))$ [always 1 for Cart-Pole]
P: $P(s_{t+1}|(s_t,a))$
lower-case gamma: discount [reward * (gamma**timesteps)]
For discount = 0.9:

- 100-point reward 1 time-step away is 100*(0.9**1) = 90 points
- 100-point reward 20 time-steps is 100*(0.9**20) = 12.2 points

Immediate Reward Preferred (gamma = 0.9)

Trilobyte playing the role of pacman; octopus playing the role of ghost; and fish playing the role of cherries



Policy

Policy function (pi) maps a state to an action

$$S \xrightarrow{\pi} A$$

$$J(\pi^*) = \max_{\pi} J(\pi) = \max_{\pi} \mathbb{E}\left[\sum_{t>0} \gamma^t r_t\right]$$

The objective of an optimal policy is to maximize the expected discounted future reward

Value and Quality-Value (Q-Value) Functions

- V^{π} (s): maps a state to expected discounted future reward
- Q^{π} (s,a): maps a state and an action to expected discounted future reward
  - A Deep 'Q' Network (DQN) is a deep network that estimates Q^{π} (s,a): the quality of an action
  - Exploitation: we select the action with the maximum expected discounted future reward
  - Exploration: we randomly select an action

Richard Sutton

Professor at the University of Alberta [also research scientist at DeepMind]

http://bit.ly/SuttonBarto

DQN Agent Dependencies

```
import random
import gym
import numpy as np
from collections import import deque
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
import os
```

Cart-Pole DQN Hyperparameters

```
env = gym.make('CartPole-v0')
state_size = env.observation_space.shape[0]
action_size = env.action_space.n
batch_size = 32
n_episodes = 1000
output_dir = 'model_output/cartpole/'
if not os.path.exists(output_dir):
os.makedirs(output_dir)
```

DQN Agent: Part 1

```
class DQNAgent:
    def __init__(self, state_size, action_size):
        self.state_size = state_size
        self.action_size = action_size
        self.memory = deque(maxlen=2000)
        self.gamma = 0.95
        self.epsilon = 1.0
        self.epsilon_decay = 0.995
        self.epsilon_min = 0.01
        self.learning_rate = 0.001
        self.model = self._build_model()
```

```
def _build_model(self):
    model = Sequential()
    model.add(Dense(32, activation='relu',
        input_dim=self.state_size))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(self.action_size, activation='linear'))
    model.compile(loss='mse',
    optimizer=Adam(lr=self.learning_rate))
    return model
def remember(self, state, action, reward, next_state, done):
    self.memory.append((state, action,
        reward, next_state, done))
```

DQN Agent: Part 2

```
def train(self, batch_size):
    minibatch = random.sample(self.memory, batch_size)
    for state, action, reward, next_state, done in minibatch:
        target = reward # if done
        if not done:
            target = (reward +
                self.gamma *
                np.amax(self.model.predict(next_state)[0]))
        target_f = self.model.predict(state)
        target_f[0][action] = target
        self.model.fit(state, target_f, epochs=1, verbose=0)
    if self.epsilon > self.epsilon_min:
        self.epsilon *= self.epsilon_decay
```

```
def act(self, state):
    if np.random.rand() <= self.epsilon:
    return random.randrange(self.action_size)
    act_values = self.model.predict(state)
    return np.argmax(act_values[0])
  def save(self, name):
    self.model.save_weights(name)
  def load(self, name):
    self.model.load_weights(name)
```

target for action taken: reward plus discounted predicted value for next state

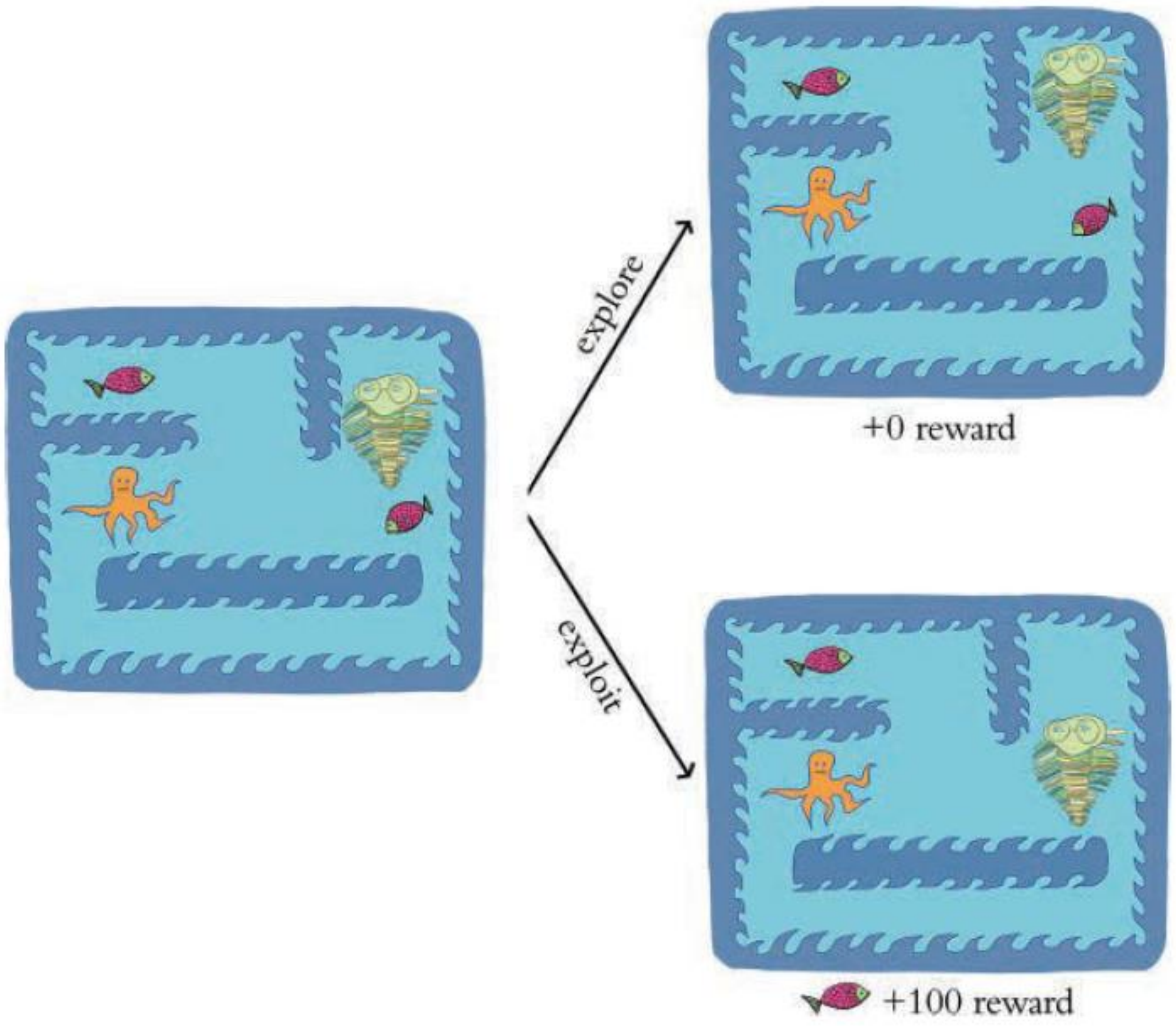
Exploration versus Exploitation

Exploration:

- Choose random action

Exploitation:

- Choose “best” action



Exploration versus Exploitation

- Epsilon Greedy Strategy
  - Exploration: for a fixed probability ϵ [lowercase epsilon], we choose a move at random
  - Exploitation: for a fixed probability $1 - \epsilon$, we choose the best move based on what we've learned so far
- Epsilon Decreasing Strategy
  - It is common to decrease epsilon over time

```
self.epsilon = 1.0
self.epsilon_decay = 0.995
self.epsilon_min = 0.01
if self.epsilon > self.epsilon_min:
    self.epsilon *= self.epsilon_decay
```

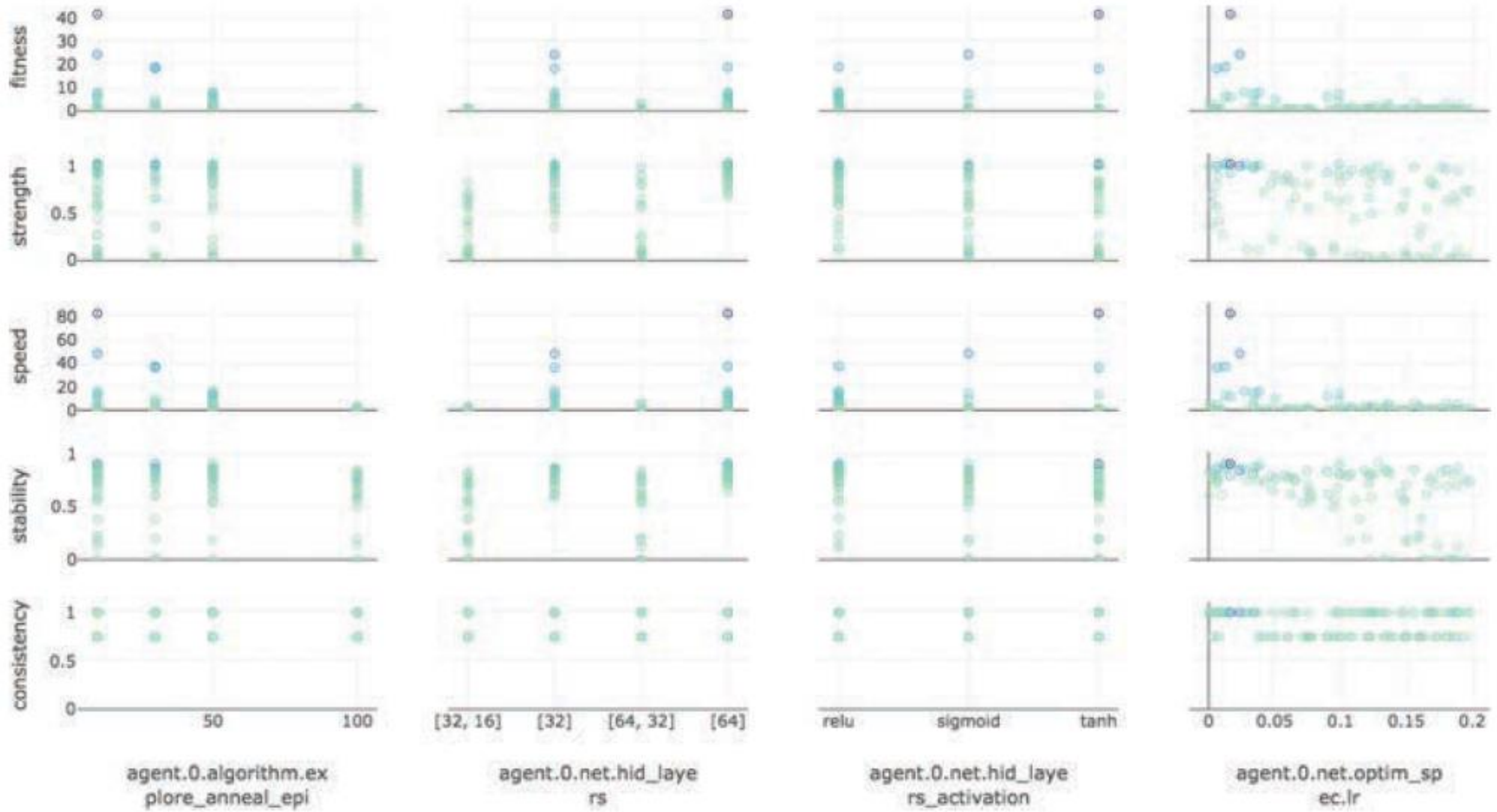
  - Exploration-Exploitation Tradeoff: our choice of exploration probability can have a large effect on how fast we learn the game [e.g. sacrificing our score for the current game can help us to improve our later scores]

Training

```
agent = DQNAgent(state_size, action_size)
for e in range(n_episodes):
    state = env.reset()
    state = np.reshape(state, [1, state_size])
    done = False
    time = 0
    while not done:
        # env.render()
        action = agent.act(state)
        next_state, reward, done, _ = env.step(action)
        reward = reward if not done else -10
        next_state = np.reshape(next_state,
                                [1, state_size])
        agent.remember(state, action, reward,
                       next_state, done)
        state = next_state
        if done:
            print("episode: {}/{}, score: {}, e: {:.2}"
                  .format(e, n_episodes-1, time,
                          agent.epsilon))
        time += 1
    if len(agent.memory) > batch_size:
        agent.train(batch_size)
    if e % 50 == 0:
        agent.save(output_dir + "weights_"
                   + '{:04d}'.format(e) + ".hdf5")
```

An Experiment Run with SLM Lab

Strange Loop Machine (SLM) is an homage to Douglas Hofstadter's exploration of human consciousness

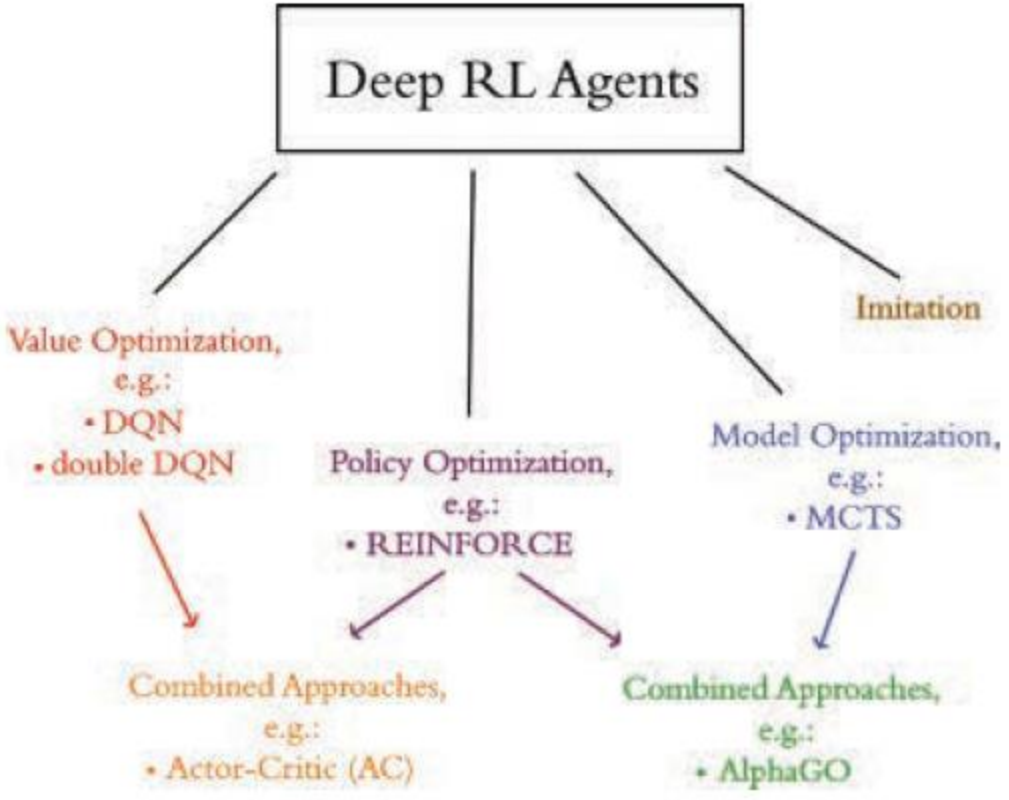


SLM Lab Results

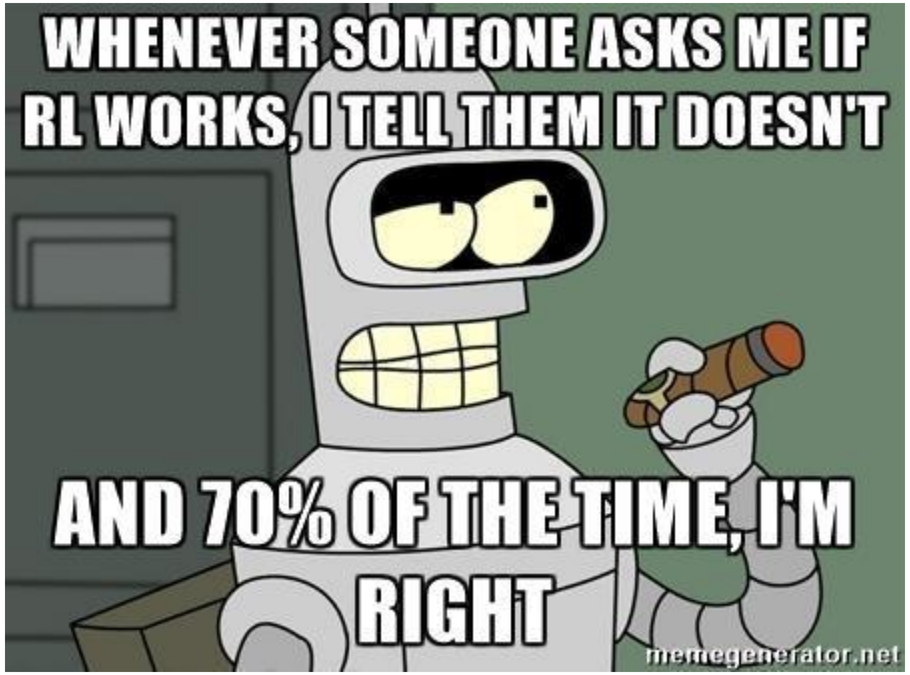
- Fitness: an overall summary metric that takes into account the other four metrics simultaneously
- Strength: a measure of the cumulative reward attained by the agent
- Speed: This is how quickly (i.e., over how many episodes) the agent was able to reach its strength
- Stability: a measure of how well it retained its solution over subsequent episodes (after the agent solved how to perform well in the environment)
- Consistency: a measure of how reproducible the performance of the agent was across trials that had identical hyperparameter settings

- Hyperparameters
  - explore_anneal_epi: number of episodes to decay epsilon from 1.0 to 0.01
  - net_hidden_layers
  - hidden_layers_activation
  - optim_spec_lr (learning rate)
- Selected configuration
  - a single hidden-layer architecture, with 64 neurons in that layer
  - the tanh activation function
  - a learning rate of ~0.02
  - trials with an exploration rate that anneals over 10 episodes outperform trials that anneal over 50 or 100 episodes

Categories of Deep RL Agents



Beware of
"Faulty Reward Functions in the Wild"



"Despite repeatedly catching on fire, crashing into other boats, and going the wrong way on the track, our agent manages to achieve a higher score using this strategy [repeatedly visiting waypoints with rewards] than is possible by completing the course in the normal way."

https://openai.com/blog/faulty-reward-functions/

https://www.alexirpan.com/2018/02/14/rl-hard.html

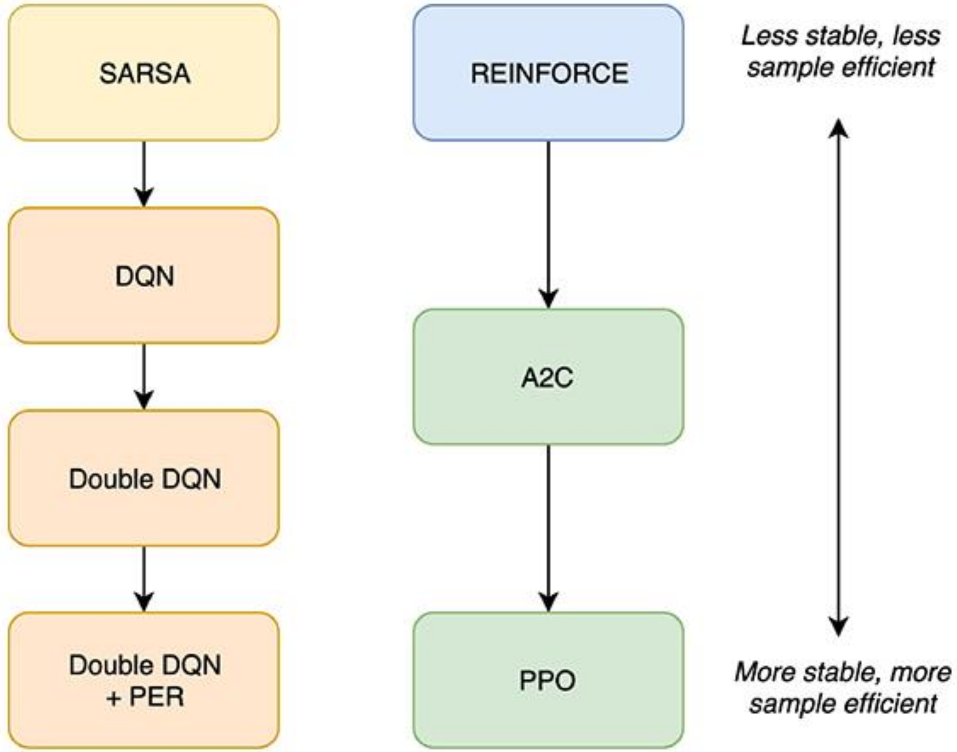
Deep RL Book

From the authors of SLM Lab: a deep reinforcement learning framework for PyTorch …

Foundations of Deep Reinforcement Learning: Theory and Practice in Python: https://www.amazon.com/dp/0135172381

Deep RL Book Algorithms

		environment		functions learned		
Algorithm	**on/off-policy**	**discrete**	**continuous**	V^π	Q^π	**policy** π
REINFORCE	on-policy	✔	✔			✔
SARSA	on-policy	✔			✔	
DQN	off-policy	✔			✔	
Double DQN + PER	off-policy	✔			✔	
A2C	on-policy	✔	✔	✔		✔
PPO	on-policy	✔	✔	✔		✔



Policy-based Example:
https://github.com/pytorch/examples/blob/master/reinforcement_learning/reinforce.py

Advantage (for A2C):

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

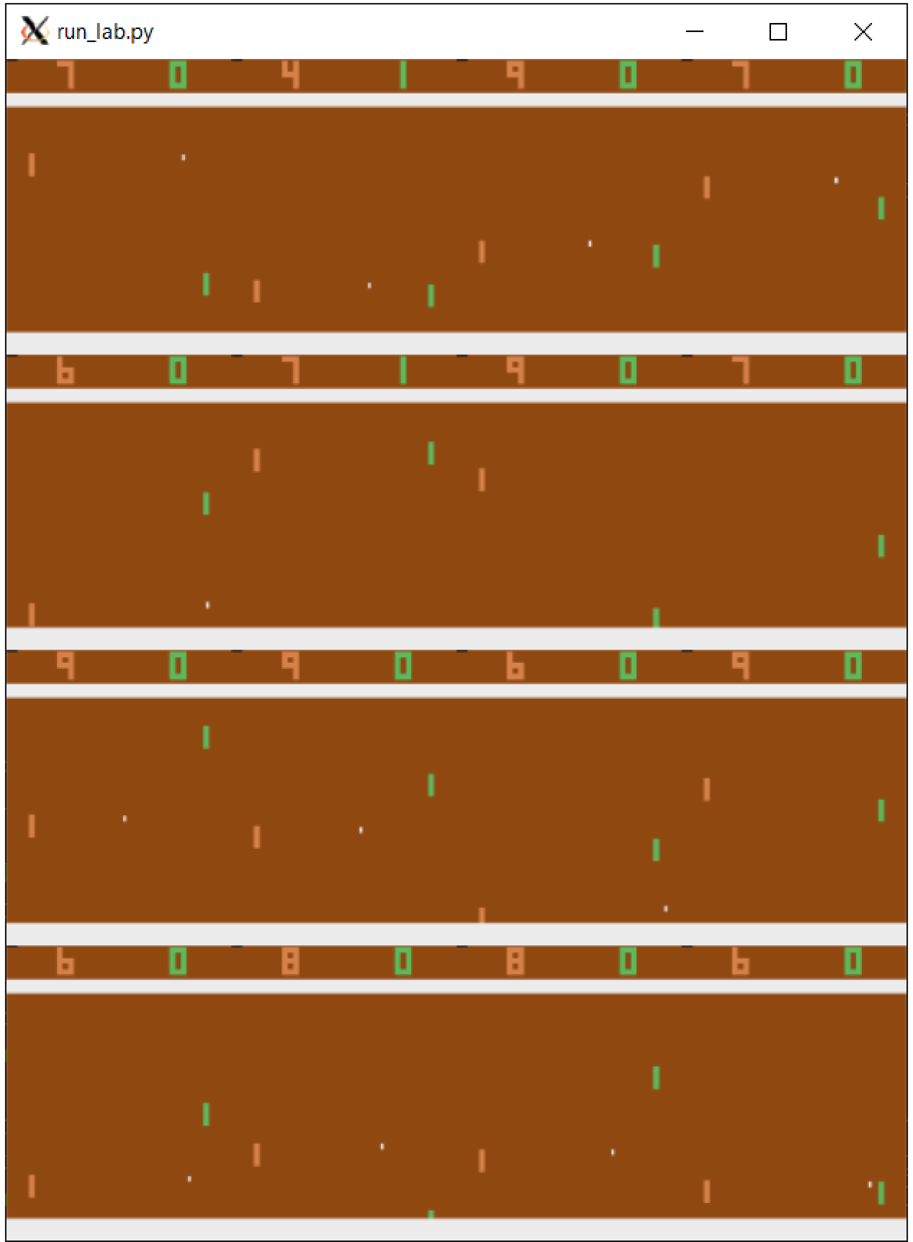
SLM-Lab

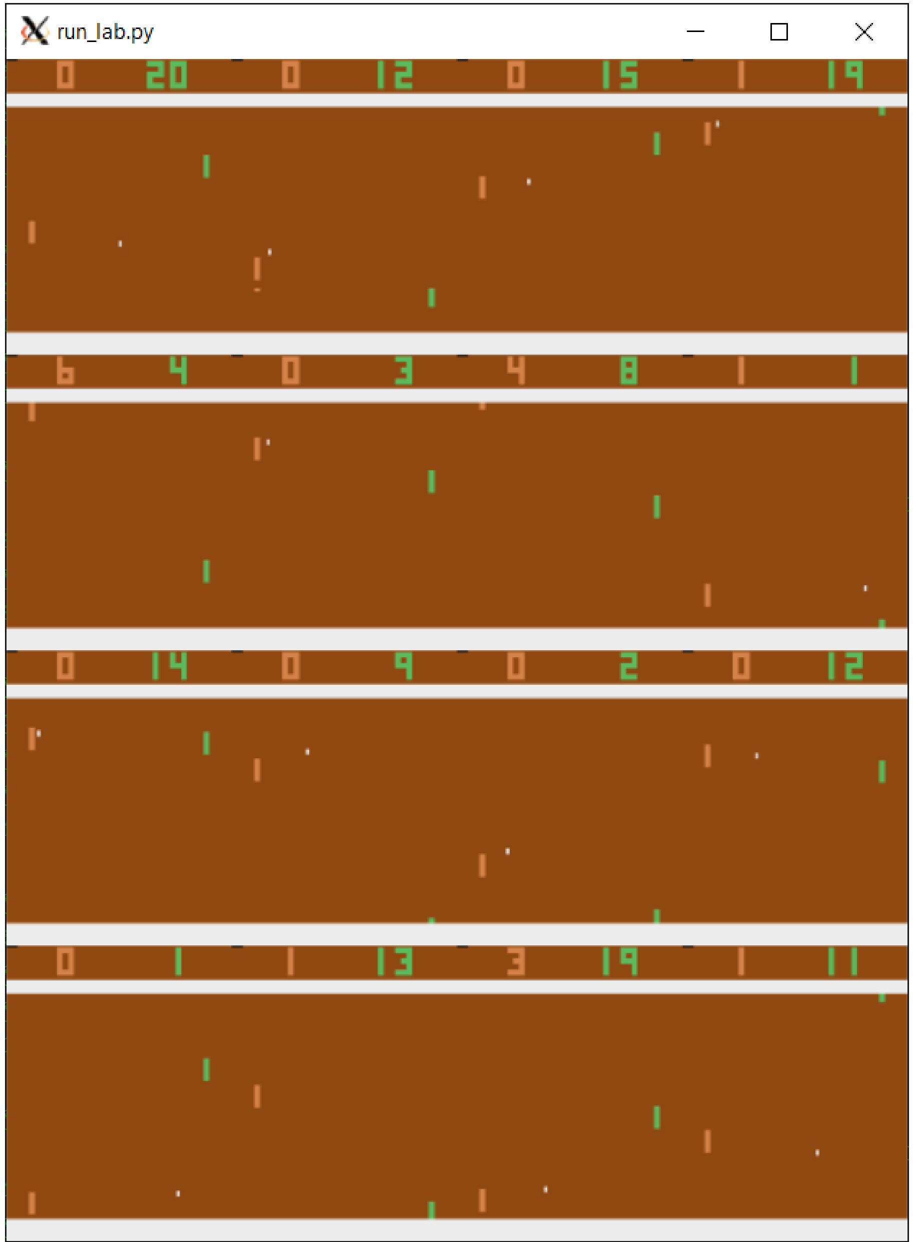
```
git clone https://github.com/kengz/SLM-Lab.git
cd SLM-Lab/
./bin/setup
conda activate lab
python run_lab.py slm_lab/spec/benchmark/dqn/dqn_pong.json dqn_pong dev
```

https://slm-lab.gitbook.io/slm-lab/analyzing-results/performance-metrics [time steps are called frames]

pong state: (210, 160, 3) RGB image
16 environments
agent has green paddle

Early:
lower
scores





Later:
higher
scores

Summary

In this chapter, we covered the essential theory of reinforcement learning, including Markov decision processes. We leveraged that information to build a deep Q-learning agent that solved the Cart-Pole environment. To wrap up, we introduced deep RL algorithms beyond DQN such as REINFORCE and actor-critic. We also described SLM Lab—a deep RL framework with existing algorithm implementations as well as tools for optimizing agent hyperparameters.

Key Concepts

Listed here are the key concepts from across this book. The final concept—covered in the current chapter—is highlighted in purple.

- parameters:
  - weight w
  - bias b
- activation a
- artificial neurons:
  - sigmoid
  - tanh
  - ReLU
  - linear
- input layer
- hidden layer
- output layer
- layer types:
  - dense (fully connected)
  - softmax
  - convolutional
  - de-convolutional
  - max-pooling
  - upsampling
  - flatten
  - embedding
  - RNN
  - (bidirectional-)LSTM
  - concatenate
- cost (loss) functions:
  - quadratic (mean squared error)
  - cross-entropy
- forward propagation
- backpropagation
- unstable (especially vanishing) gradients
- Glorot weight initialization
- batch normalization
- dropout
- optimizers:
  - stochastic gradient descent
  - Adam
- optimizer hyperparameters:
  - learning rate η
  - batch size
- word2vec
- GAN components:
  - discriminator network
  - generator network
  - adversarial network
- deep Q-learning

Datasets Review

- Images (PIL)
  - MNIST Digit Classification
  - Fashion Accessory Classification
  - CIFAR10 Image Classification
  - Tiny ImageNet Classification (MAP)
- Text (spaCy, NLTK)
  - Reuters MultiLabel Classification (Macro Averaged ROC AUC)
  - Newsgroups Classification
  - IMDB Review Sentiment Classification
  - Penn TreeBank Language Modeling (Perplexity)
- Speech (libROSA)
  - Google Commands