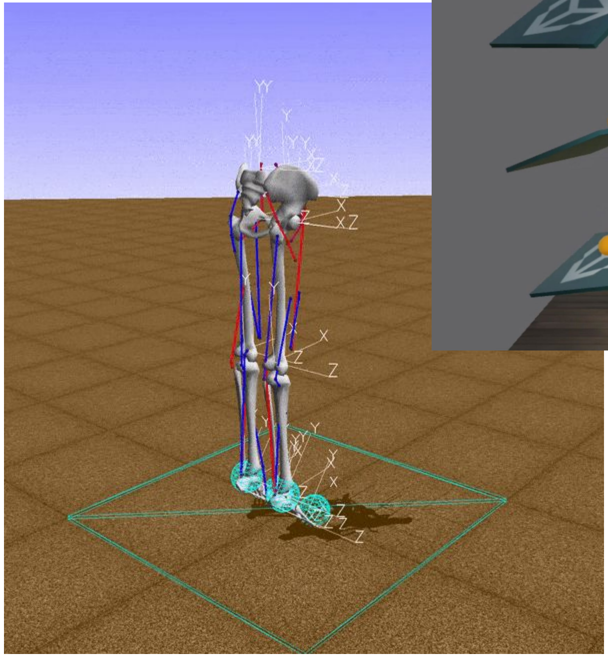
Deep Reinforcement Learning - Environments Tour

Starcraft 2
DeepMind toolset PySc2



OpenSim RL - osim-rl

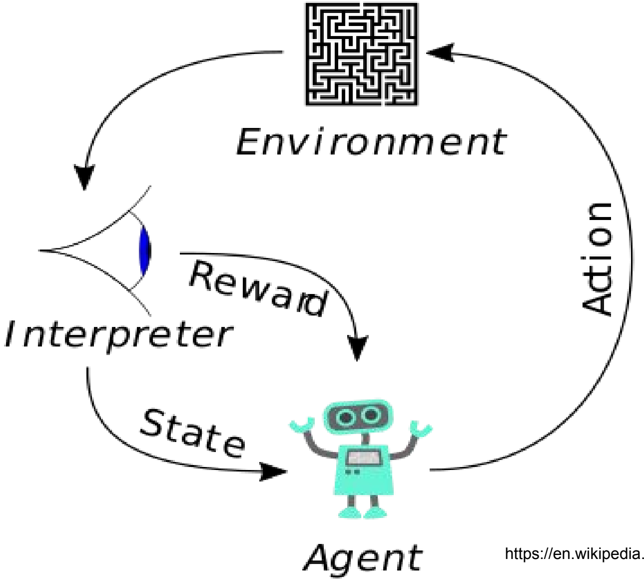
Unity ML-Agents

Concepts behind Reinforcement Learning

Supervised learning = mimic the right answers, based on **many** examples

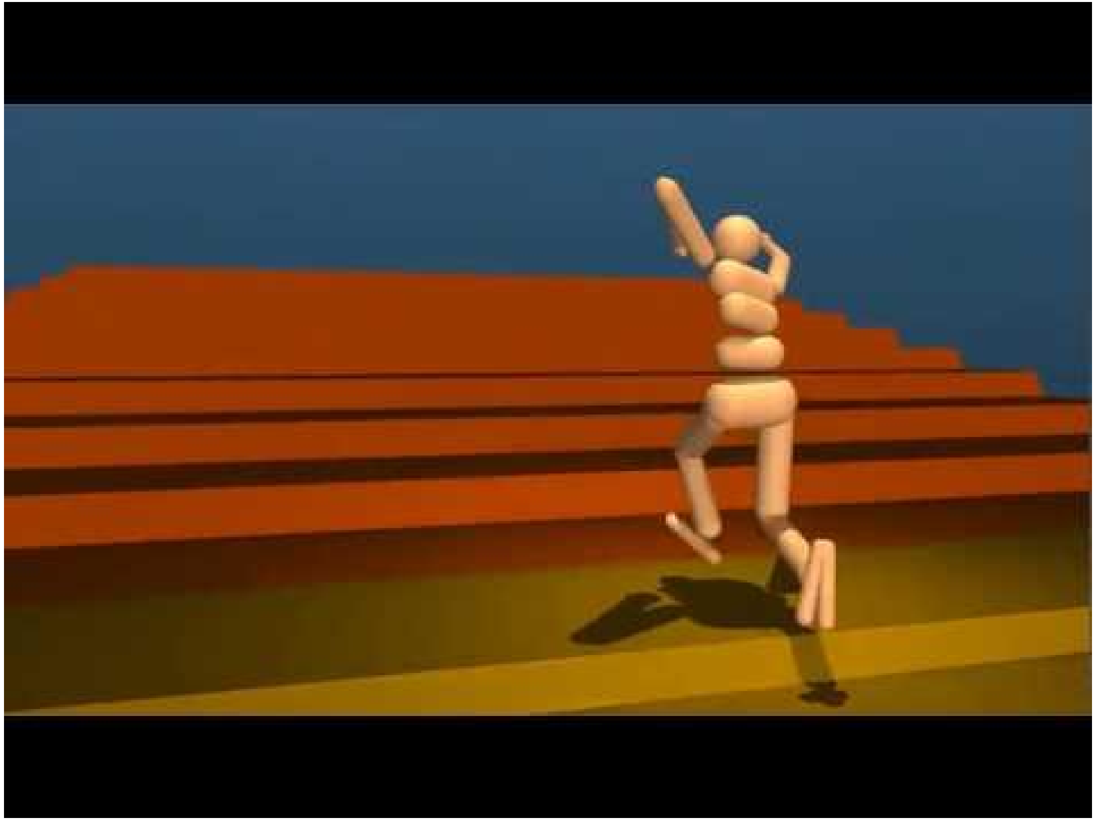
Unsupervised learning = find patterns in data, infer hidden structure without examples

Reinforcement learning = no examples, just the reward function, data could have no hidden structure at all - just do the task very well



https://en.wikipedia.org/wiki/File:Reinforcement_learning_diagram.svg

Concepts behind Reinforcement Learning



AlphaGo paper : https://gogameguru.com/i/2016/03/deepmind-mastering-go.pdf

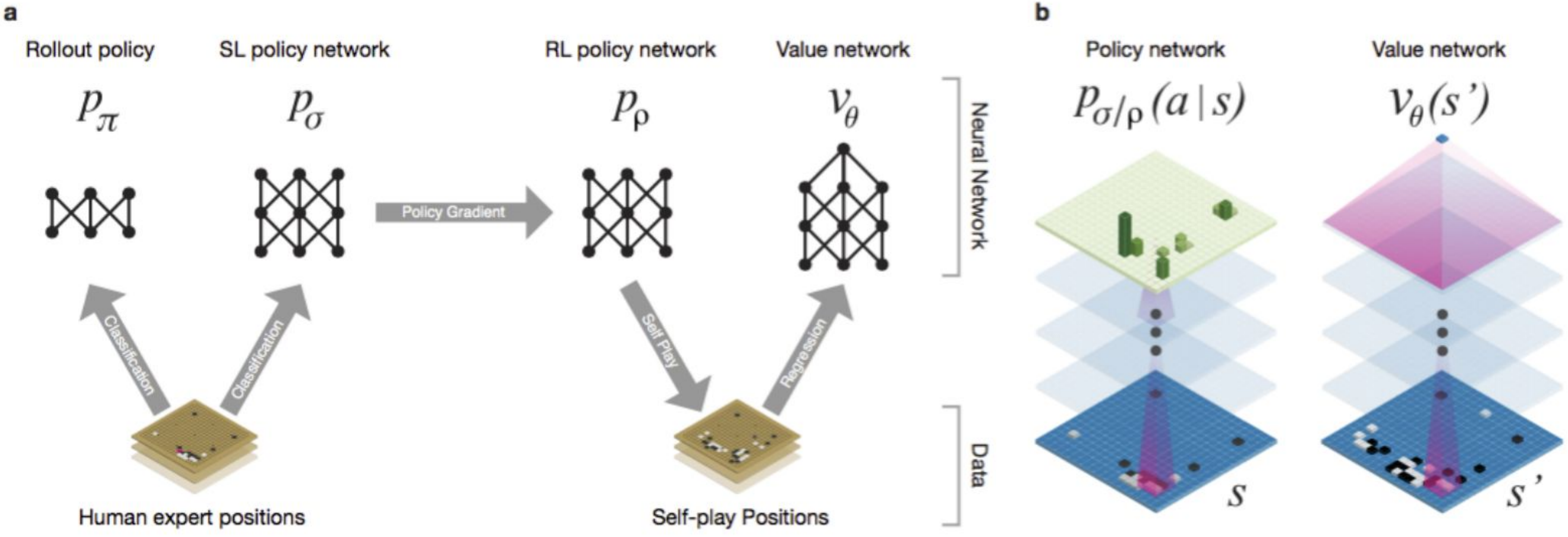


Figure 1: **Neural network training pipeline and architecture.**

Formalism of learning process = **Markov Decision Processes**

A Markov decision process is a tuple $(S, A, \{P_{sa}\}, \gamma, R)$, where:

- S is a set of **states**. (For example, in autonomous helicopter flight, S might be the set of all possible positions and orientations of the helicopter.)
- A is a set of **actions**. (For example, the set of all possible directions in which you can push the helicopter's control sticks.)
- P_{sa} are the state transition probabilities. For each state $s \in S$ and action $a \in A$, P_{sa} is a distribution over the state space. We'll say more about this later, but briefly, P_{sa} gives the distribution over what states we will transition to if we take action a in state s.
- $\gamma \in [0, 1)$ is called the **discount factor**.
- $R : S \times A \mapsto \mathbb{R}$ is the **reward function**. (Rewards are sometimes also written as a function of a state S only, in which case we would have $R : S \mapsto \mathbb{R}$).

/en.wikipedia.org/wiki/File:Reinforcement_learning_diagram.svg

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \ldots$$

Apply discount to know total payoff
(it makes the agent focus more on short-term goals)

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots .$$

$$\gamma \in [0, 1)$$

We can use simpler,
State-only dependant rewards
(but not required)

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots .$$

Our goal in reinforcement learning is to choose actions over time so as to maximize the expected value of the total payoff:

$$\mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots\right]$$

https://en.wikipedia.org/wiki/File:Reinforcement_learning_diagram.svg

A **policy** is any function $\pi : S \mapsto A$ mapping from the states to the actions. We say that we are **executing** some policy π if, whenever we are in state s, we take action $a = \pi(s)$. We also define the **value function** for a policy π according to

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

$V^\pi(s)$ is simply the expected sum of discounted rewards upon starting in state s, and taking actions according to π.[1]

Given a fixed policy π, its value function V^π satisfies the **Bellman equations**:

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^\pi(s').$$

Introduction to RL

Example explanation on Atari games
https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/16_Reinforcement_Learning.ipynb

- Absolutely best materials from this guy:
    a. **TensorFlow Tutorial #16 Reinforcement Learning**
    b. https://github.com/Hvass-Labs/TensorFlow-Tutorials
    c. Great, in-depth, scientific to the bone!
    d. Good for beginners - just click through the notebook

- For Windows, some libraries are missing. Try these
    a. For atari-py -> https://github.com/j8lp/atari-py (involves installing MSYS)

But why DEEP reinforcement learning?

https://www.intelnervana.com/demystifying-deep-reinforcement-learning/

Specific solutions are, for example:

- Q-learning - https://en.wikipedia.org/wiki/Q-learning

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

```
initialize Q[num_states,num_actions] arbitrarily
observe initial state s
repeat
      select and carry out an action a
      observe reward r and new state s'
      Q[s,a] = Q[s,a] + α(r + γ max_a' Q[s',a'] - Q[s,a])
      s = s'
until terminated
```

We describe the state of the game with a set of parameters, specific to the environment - not universal

But why DEEP reinforcement learning? https://www.intelnervana.com/demystifying-deep-reinforcement-learning/

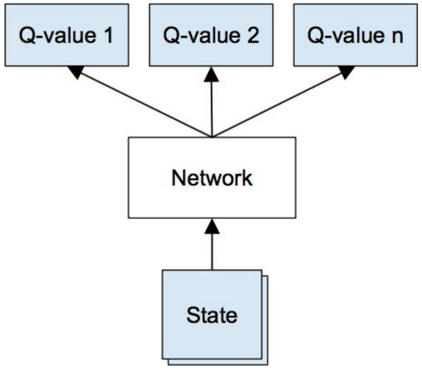
Specific solutions are, for example:

- Deep Q-learning (`baselines.deepq`)
  - A truly universal representation could be just the pixels - we do not care how many parameters are in there, in theory the whole game state can be viewed on screen
  - Run the screen through Convolutional Neural Net -> and get the Q-values

```
initialize replay memory D
initialize action-value function Q with random weights
observe initial state s
repeat
     select an action a
          with probability ε select a random action
          otherwise select a = argmax_a' Q(s,a')
     carry out action a
     observe reward r and new state s'
     store experience <s, a, r, s'> in replay memory D

     sample random transitions <ss, aa, rr, ss'> from replay memory D
     calculate target for each minibatch transition
          if ss' is terminal state then tt = rr
          otherwise tt = rr + γmax_a' Q(ss', aa')
     train the Q network using (tt - Q(ss, aa))^2 as loss

     s = s'
until terminated
```



ε-greedy exploration – with probability ε choose a random action, otherwise go with the "greedy" action with the highest Q-value. Decreases ε over time from 1 to 0.1

But why DEEP reinforcement learning?

https://www.intelnervana.com/demystifying-deep-reinforcement-learning/

Specific solutions are, for example:

- Proximal Policy Optimization (PPO) - https://blog.openai.com/openai-baselines-ppo/
  - https://arxiv.org/pdf/1707.06347.pdf

Google DeepMind + Blizzard = SC2LE

The SC2LE release includes:

- A Machine Learning API developed by Blizzard that gives researchers and developers hooks into the game. This includes the release of tools for Linux for the first time.
- A dataset of anonymised game replays, which will increase from 65k to more than half a million in the coming weeks.
- An open source version of DeepMind's toolset, PySC2, to allow researchers to easily use Blizzard's feature-layer API with their agents.
- A series of simple RL mini-games to allow researchers to test the performance of agents on specific tasks.
- A joint paper that outlines the environment, and reports initial baseline results on the mini-games, supervised learning from replays, and the full 1v1 ladder game against the built-in AI.

Starcraft II RL Tutorial 1 http://chris-chris.ai/2017/08/30/pysc2-tutorial1/

Guide to DeepMinds Starcraft AI Environment https://www.youtube.com/watch?v=URWXG5jRB-A

Building Bots In Starcraft 2 https://gamescapad.es/building-bots-in-starcraft-2-for-psychologists/#installation

https://deepmind.com/blog/deepmind-and-blizzard-open-starcraft-ii-ai-research-environment/

Google DeepMind + Blizzard = SC2LE

- DEMO
-
- Git clone https://github.com/Blizzard/s2client-proto.git
-
- Git clone https://github.com/llSourcell/A-Guide-to-DeepMinds-StarCraft-AI-Environment.git
- `pip install pysc2`
- `pip install baselines`
- `pip install tensorflow`
- Jupyter notebook
- -> A Guide to DeepMind's StarCraft AI Environment.ipynb

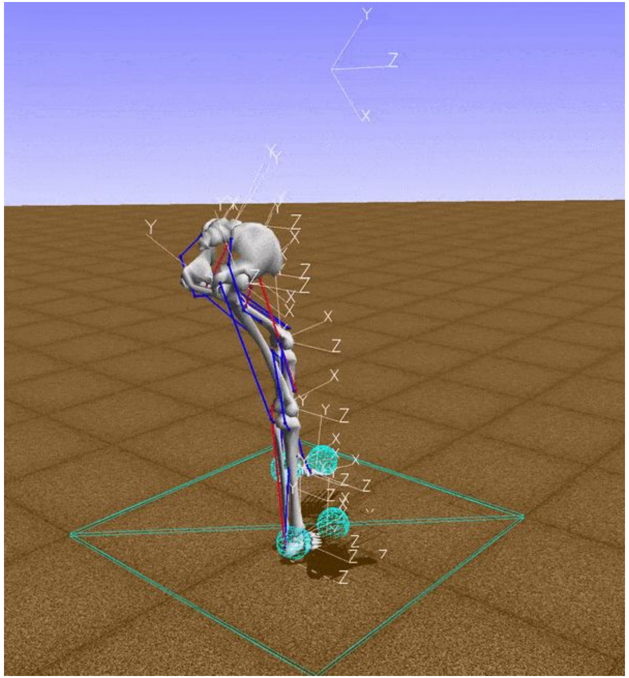
https://deepmind.com/blog/deepmind-and-blizzard-open-starcraft-ii-ai-research-environment/

OpenSim Core + Opensim-RL + keras + keras-RL = Creepy Skeletons

- https://github.com/opensim-org/opensim-core
  a. http://opensim.stanford.edu/

https://github.com/stanfordnmbl/osim-rl

https://github.com/matthiasplappert/keras-rl
BTW - keras-rl is passing development to community!
HELP WANTED



OpenSim Core + Opensim-RL + keras + keras-RL = Creepy Skeletons

- https://github.com/matthiasplappert/keras-rl
  a. Deep Q Learning (DQN) [1], [2]
  b. Double DQN [3]
  c. Deep Deterministic Policy Gradient (DDPG) [4]
  d. Continuous DQN (CDQN or NAF) [6]
  e. Cross-Entropy Method (CEM) [7], [8]
  f. Dueling network DQN (Dueling DQN) [9]
  g. Deep SARSA [10]

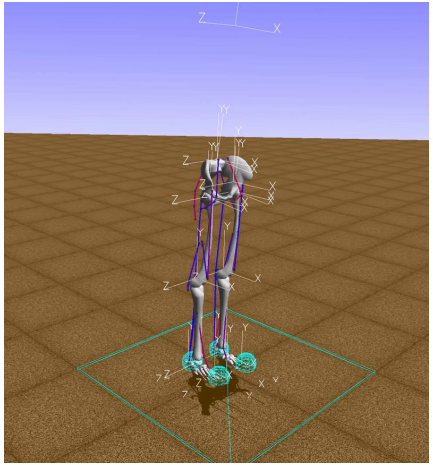
OpenSim Core + Opensim-RL + keras + keras-RL = Creepy Skeletons

- https://github.com/stanfordnmbl/osim-rl -> tutorial
- DEMO
- Install Anaconda
- Create Anaconda Python environment
  a. `conda create -n opensim-rl -c kidzik opensim git python=2.7`
     `activate opensim-rl`
  b. `conda install -c conda-forge lapack git`
  c. `pip install git+https://github.com/stanfordnmbl/osim-rl.git`
  d. Check import `python -c "import opensim"`
  e. Run this python snippet:

```python
from osim.env import RunEnv

env = RunEnv(visualize=True)
observation = env.reset(difficulty = 0)
for i in range(200):
    observation, reward, done, info = env.step(env.action_space.sample())
```

Random Activation Vector



- WHAT IS THE STATE?
  a. Current positions
  b. Velocities of joints (angular velocities)
  c. Accelerations of joints (angular accelerations)

- Substitute random activation with invocation of your own controller

```
total_reward = 0.0
for i in range(200):
    # make a step given by the controller and record the state and the reward
    observation, reward, done, info = env.step(my_controller(observation))
    total_reward += reward
    if done:
        break
```

OpenSim Core + Opensim-RL + keras + keras-RL = Creepy Skeletons

Tutorial for Deep Deterministic Policy Gradients.

```
conda install keras -c conda-forge
```

```
pip install git+https://github.com/matthiasplappert/keras-rl.git
```

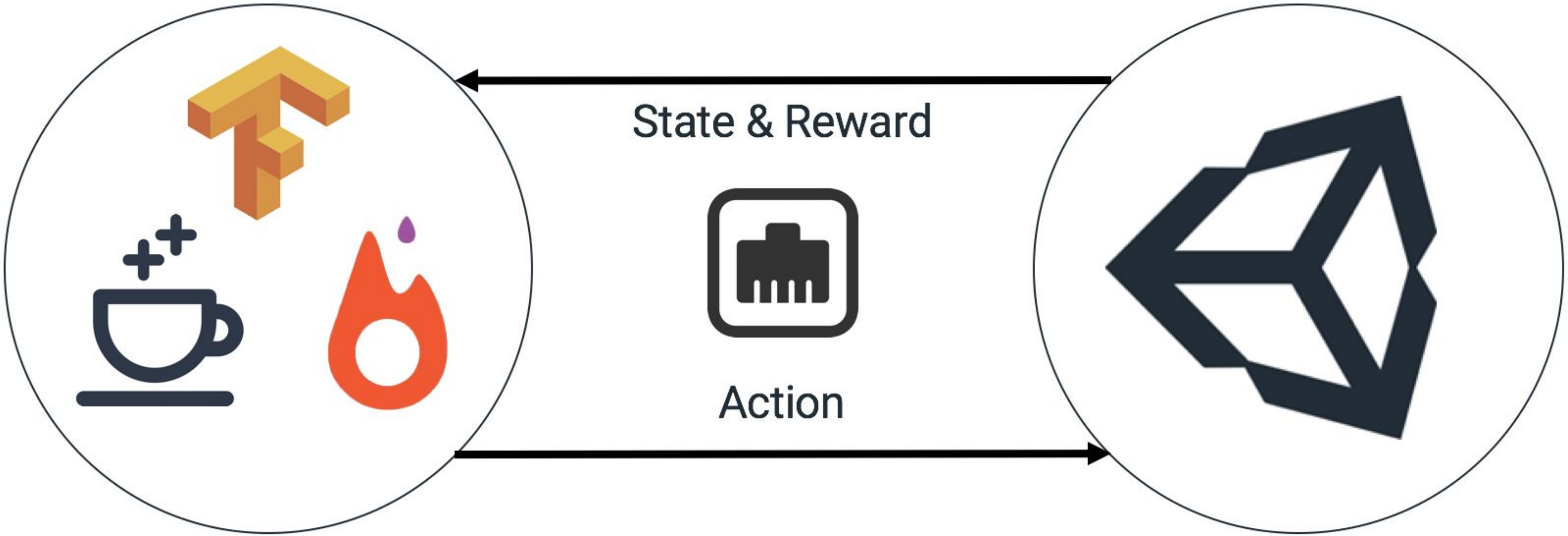
```
git clone http://github.com/stanfordnmbl/osim-rl.git
```

```
cd osim-rl/scripts
```

```
python example.py --visualize --train --model sample
```

```
python example.py --visualize --test --model sample  # walk as far as possible
```

Unity Machine Learning Agents



https://blogs.unity3d.com/2017/09/19/introducing-unity-machine-learning-agents/

Unity Machine Learning Agents

https://github.com/Unity-Technologies/ml-agents
https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Getting-Started-with-Balance-Ball.md -> **Building Unity Environment**

https://github.com/Unity-Technologies/ml-agents/tree/master/unity-environment

Tutorial = https://github.com/Unity-Technologies/ml-agents/tree/master/python
Git clone…
Cd python
Pip install -r requirements.txt
Jupyter notebook
Navigate to web browser URL=localhost:8888
-> Basics.ipynb (launching and interfacing with Unity)
-> PPO.ipynb (training agents)

Tensorboard --logdir='./summaries'
Navigate to web browser URL=localhost:6006 # to monitor training

For the impatient people - training on AWS UNITY + AWS

Materials

Tutorials & Courses on Reinforcement Learning:

- Berkeley Deep RL course by Sergey Levine
- Intro to RL on Karpathy's blog
- Intro to RL by Tambet Matiisen
- Deep RL course of David Silver
- A comprehensive list of deep RL resources

Frameworks and implementations of algorithms:

- RLLAB
- modular_rl
- keras-rl

OpenSim and Biomechanics:

- OpenSim Documentation
- Muscle models
- Publication describing OpenSim
- Publication describing Simbody (multibody dynamics engine)