Applied ML Journal Club: Reinforcement Learning

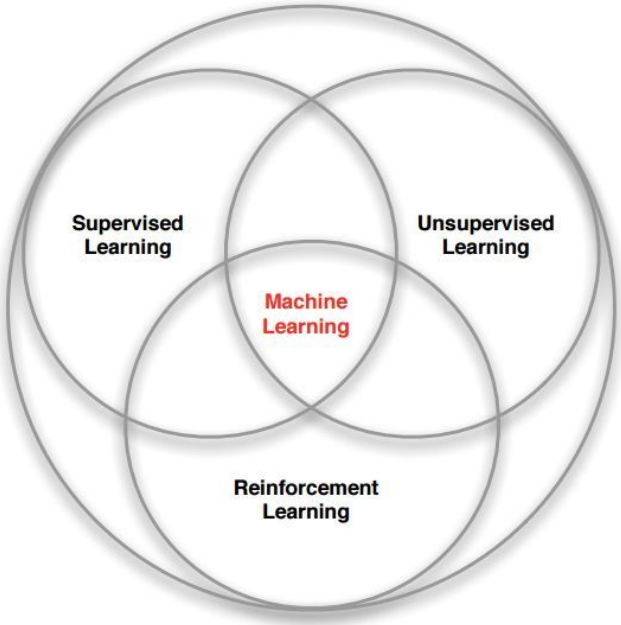
26 August 2020

Anas Abou Allaban
allaban.me

I have a joke about Reinforcement Learning...

...it will start sounding funny after the first 10,000 times you go over it.

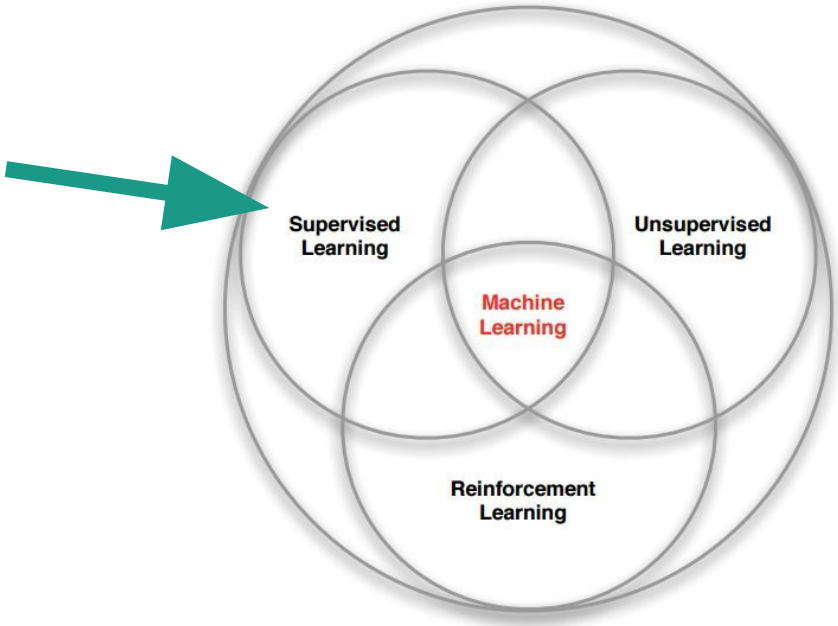
The Domains of Machine Learning



The Domains of Machine Learning

Classification:
-Image Recognition (YOLO, ImageNet)
-Speech Recognition (DeepSpeech, RNN-T)

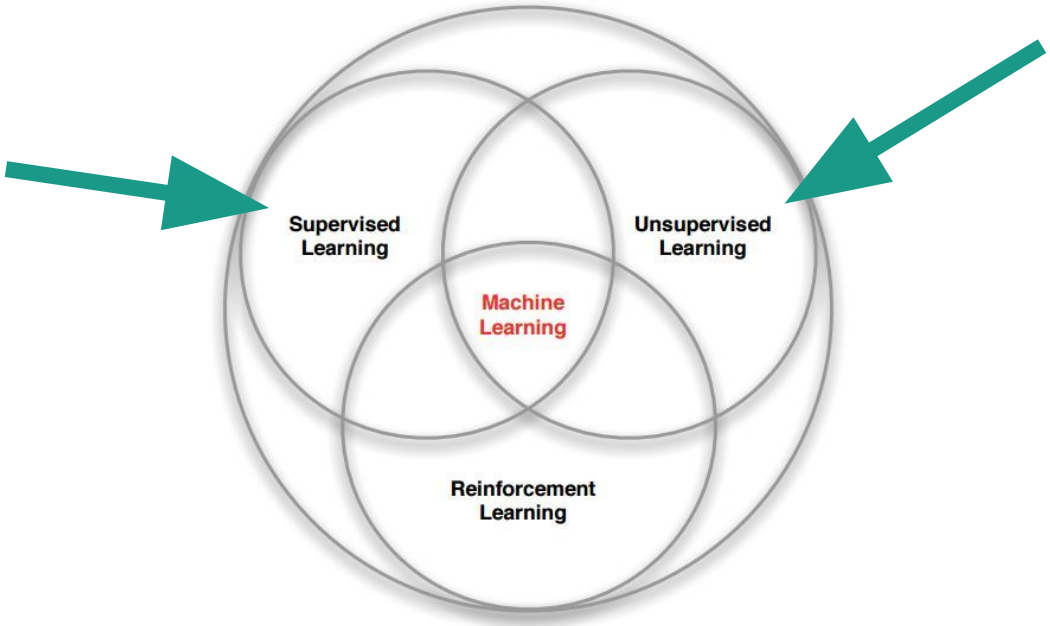
Regression:
-Stock Market Prediction
-User Behavior
-Event Prediction (XGBoost, FF NN)



The Domains of Machine Learning

Classification:
-Image Recognition (YOLO, ImageNet)
-Speech Recognition (DeepSpeech, RNN-T)

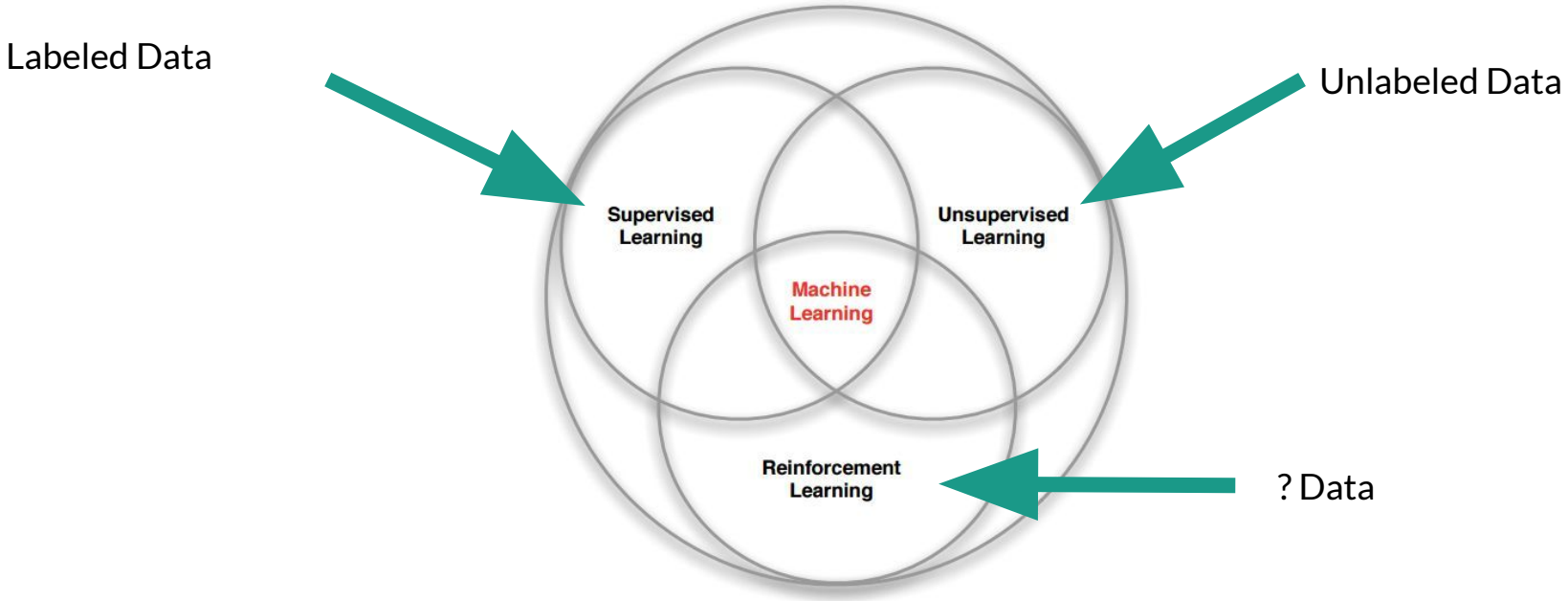
Regression:
-Stock Market Prediction
-User Behavior
-Event Prediction (XGBoost, FF NN)



Clustering:
-Customer Segments
-Dimensionality Reduction (GPT, GAN)

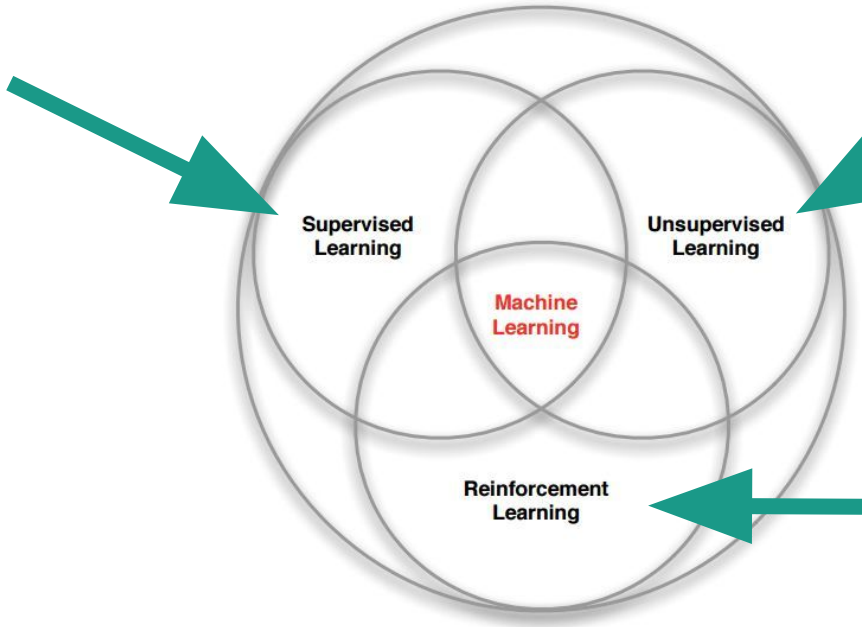
Anomaly Detection

The Domains of Machine Learning



The Domains of Machine Learning

Labeled Data



Unlabeled Data

No Data Collection!

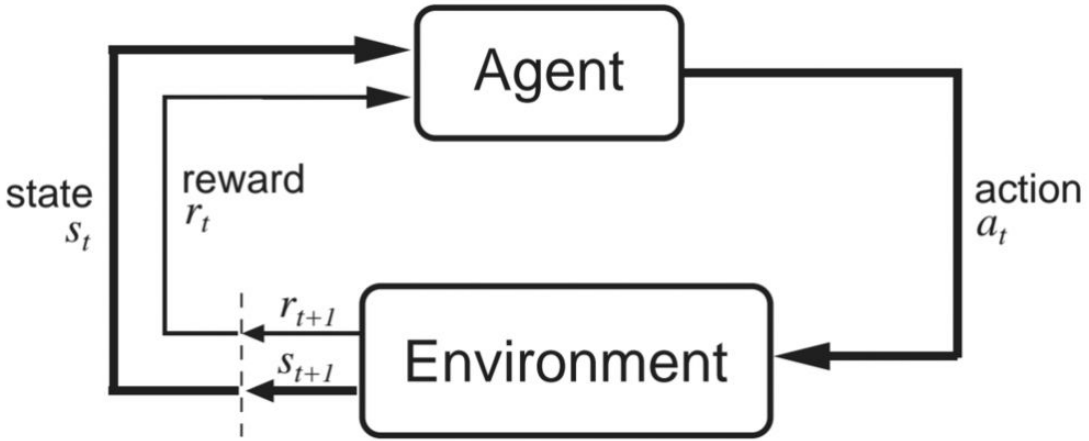
Reinforcement Learning models learn by exploring their environment on their own!

Components of an RL System

1. **State**
The current environment at time *t*

2. **Action**
The action an agent executes
(ex. Move left)

3. **Reward**
Positive (or negative) feedback for action taken



Remind you of something?

Reward (R)

The reward signal defines the goal of the RL problem (ex. Pick up the cup)

A reward signal is sent from the environment at every action that a RL agent takes (referred to as time step).

The RL agent can influence the reward signal directly through its actions and indirectly through altering the environment's state. But *it can't change the problem it's been tasked with*.

State (*s*)

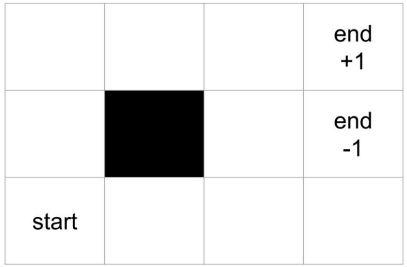
State is how we perceive the environment.

Like images and molecules, we need a way to encode state such that the agent can learn and understand.

The simplest form of state is a 2x2 grid.
States are usually represented as images.

States have value (i.e. I'm in a good situation now!)

$$V(s) = E_{\pi}[R_t|s_t]$$



2x2 grid state

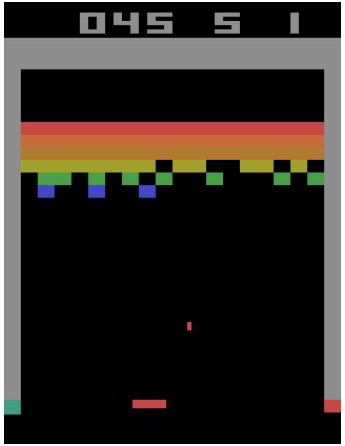


Image state

Actions (*a*)

Domain specific representations of what an agent can perform.

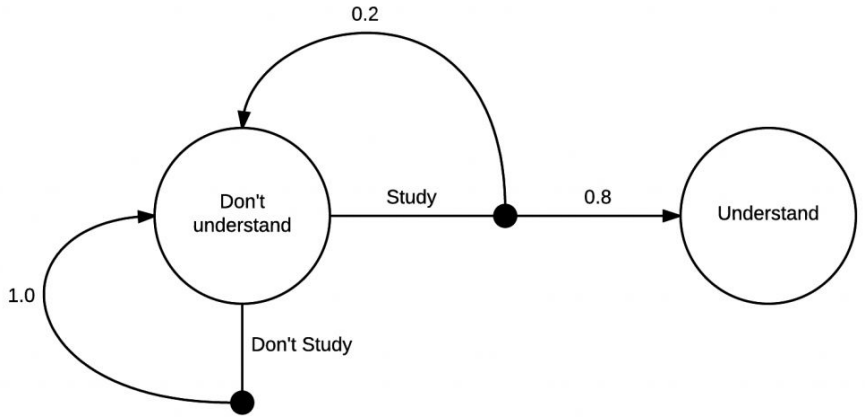
Actions are selected based on a *policy* (Usually denoted with P or π.)

A policy defines the behaviour of an agent and how the agent picks its actions.
It does so by mapping from perceived states of an environment to actions to be taken when in those states.

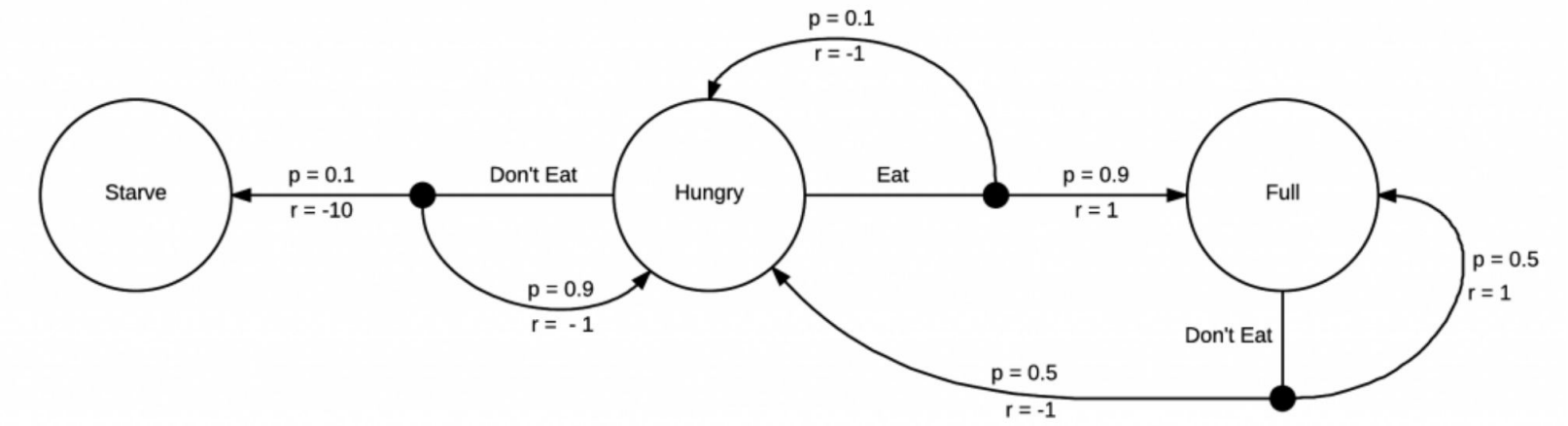
"If we start at state *s* and take action *a* we end up in state with probability" $P(s_{t+1}|s_t, a_t)$

Formalizing the RL Problem

RL is usually formalized as a “Markov Decision Process” (MDP)



Formalizing the RL Problem



Formalizing the RL Problem

If I have a reward…

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

...and I know what reward to expect in certain states....

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[R_t | s_t = s\right]$$

...what's the policy that maximizes my expected reward?

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[R_t | s_t = s, a_t = a\right]$$

We can solve for the optimal policy using the Bellman Equations and Dynamic programming.

$$Q^{\pi}(s,a) = \sum_{s'} \mathcal{P}^{a}_{ss'}\left[\mathcal{R}^{a}_{ss'} + \gamma \sum_{a'} \pi(s',a') Q^{\pi}(s',a')\right]$$

There's nothing "deep" here though….

Formalizing the Deep RL Problem

Deep RL aims to learn the policy $\pi_\theta(a|s)$

"Optimize the parameters either directly by gradient ascent on the performance objective, or indirectly, by maximizing local approximations of the performance objective" $J(\pi_\theta)$

"The corresponding policy is obtained via the connection between Q* and π*: the actions taken by the Q-learning agent are given by:"

$$a(s) = argmax_a Q_\theta(s, a)$$

How much data...?

Agents learn on their own by exploring the environment.

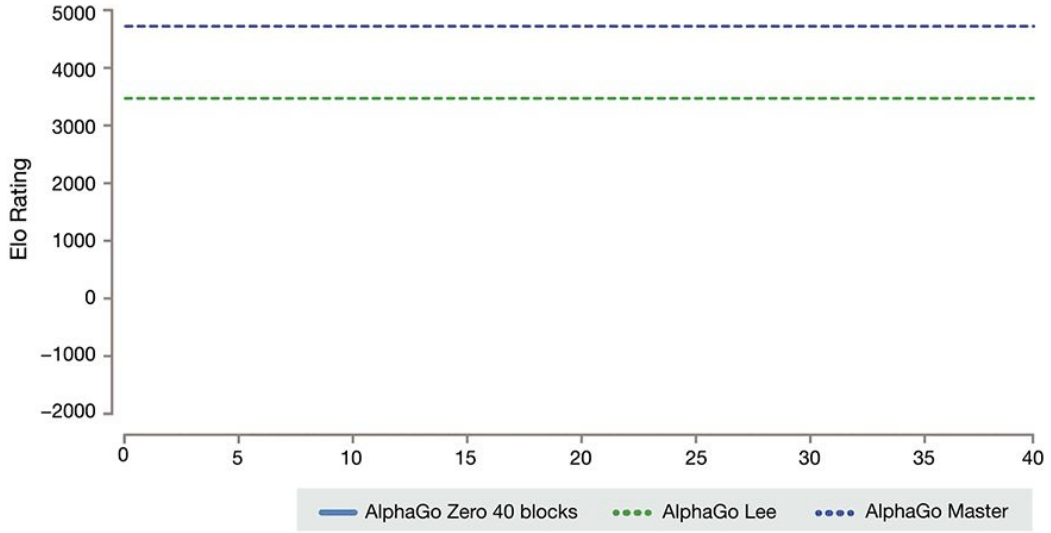
Randomly selecting actions however is *extremely inefficient*. Choosing a good reward function is also hard.

Achieving optimal policies with random sampling can take **100s** of (simulation) years!

Improving sample efficiency and reward engineering (Inverse RL) is an active research area*

Some approaches: Model Predictive Control, Hierarchical RL, and Few Shot Learning

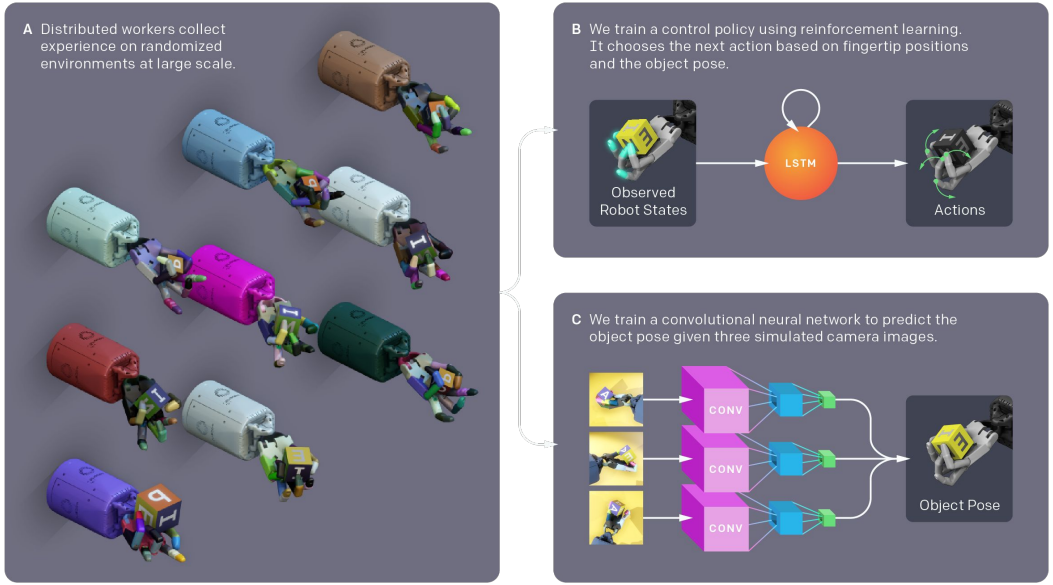
How much data...?



Dactyl

Learn how to move an object into a specific pose or configuration.

Two step process:
1. Train in simulation
2. Fine tune on the real world



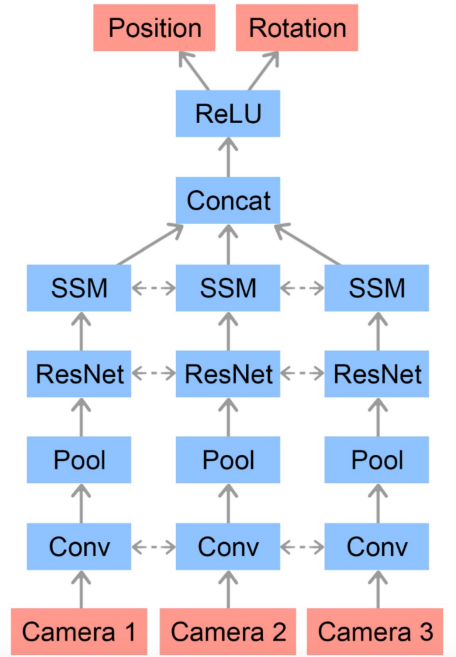
Dactyl: State

The state is the “pose” of the object.

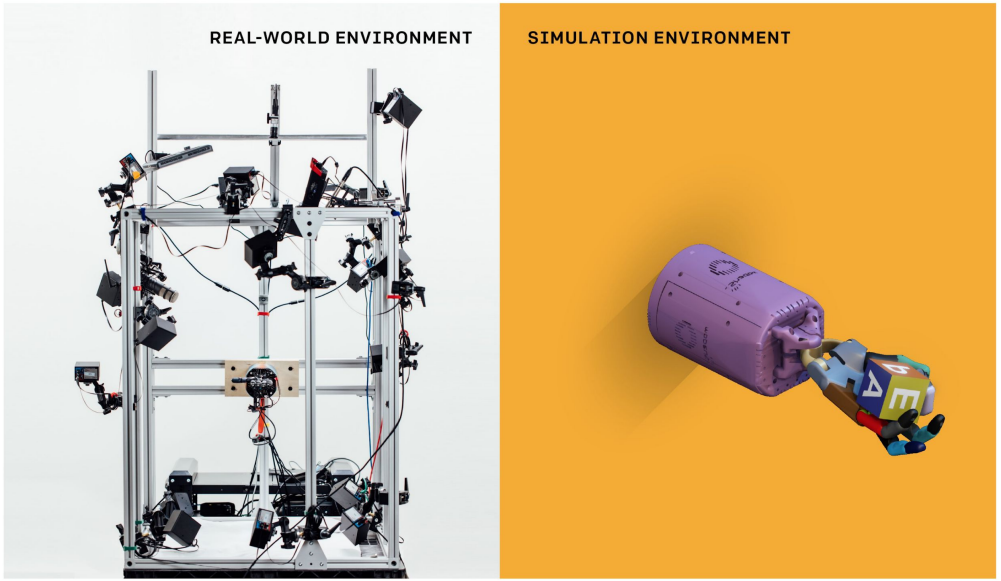
State estimation from vision and motion capture using CNNs.

Used to predict the next state given current object pose

$$P(s_{t+1} | s_t, a_t)$$



Dactyl: State

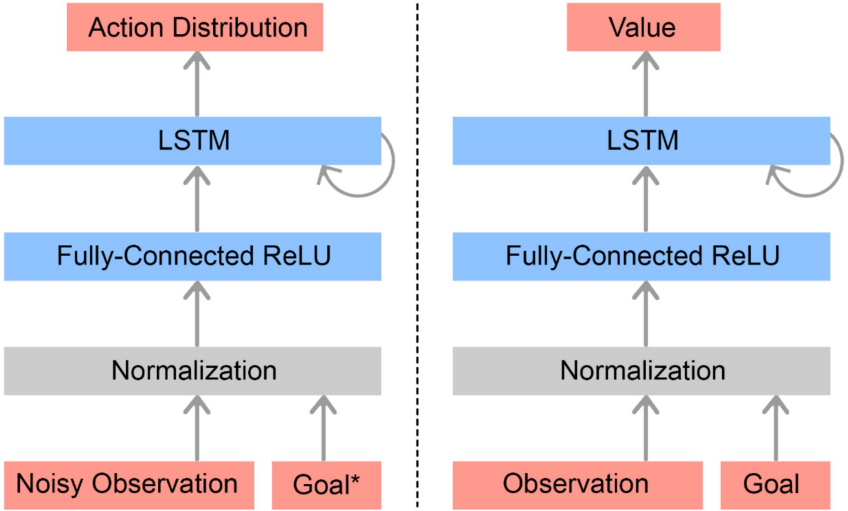


Dactyl: Action

Control Policy is an LSTM with an additional hidden layer with ReLU.

Why LSTM?
Learn to generalize dynamics of the world from multiple observations.

$$a(s) = argmax_a Q_\theta(s, a)$$



Dactyl: Action

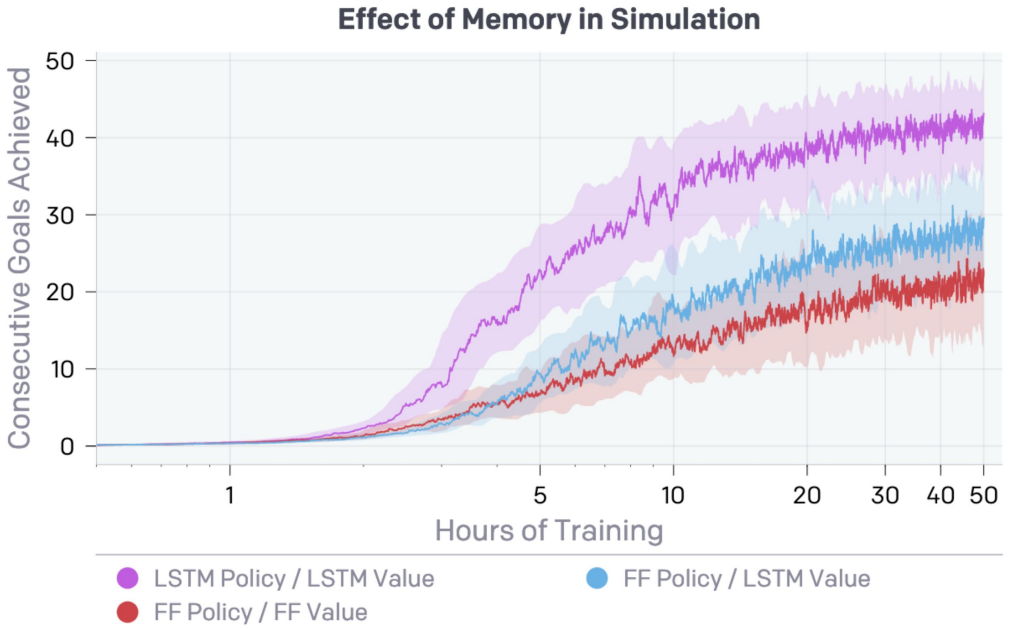
Table 2: Observations of the policy and value networks, respectively.

Input	**Dimensionality**	**Policy network**	**Value network**
fingertip positions	15D	✓	✓
object position	3D	✓	✓
object orientation	4D (quaternion)	×[4]	✓
target orientation	4D (quaternion)	×	✓
relative target orientation	4D (quaternion)	✓	✓
hand joints angles	24D	×	✓
hand joints velocities	24D	×	✓
object velocity	3D	×	✓
object angular velocity	4D (quaternion)	×	✓

Dactyl: Action

Policy uses memory to adapt to current environment.

Better performance than simple Feed-forward NN.

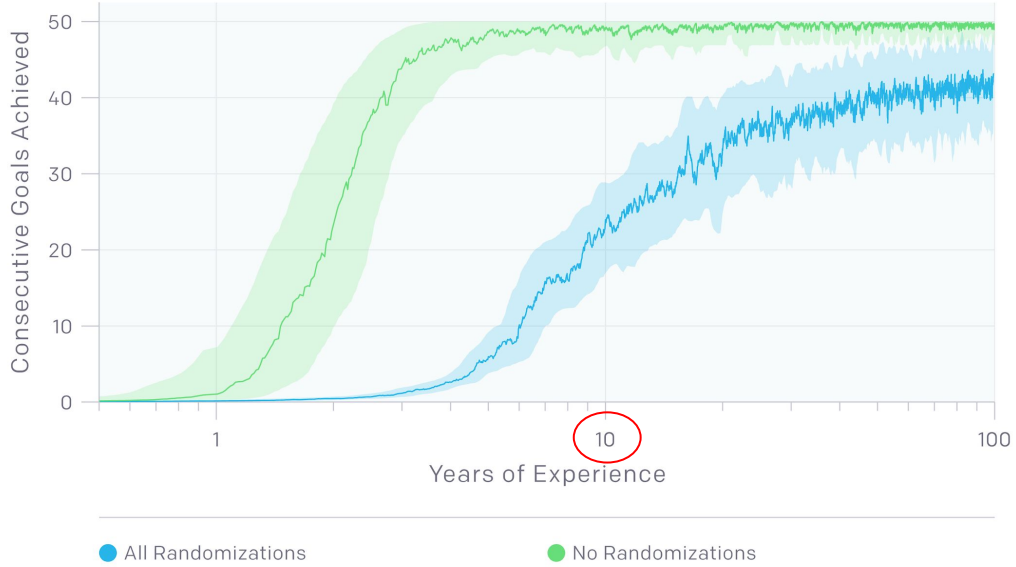


Dactyl: Reward

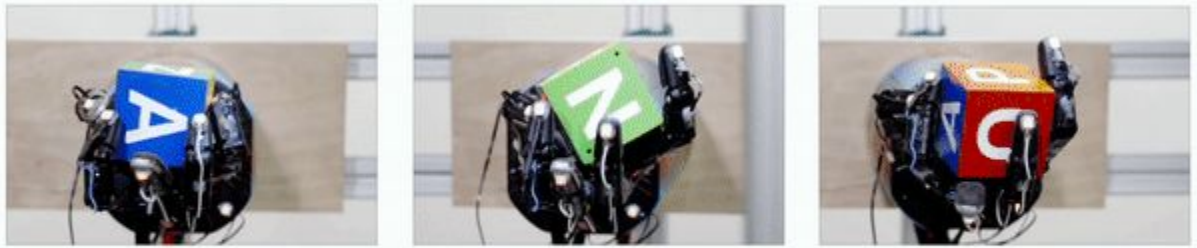
Use Proximal Policy Optimization (PPO) to boost exploration*

TL;DR -
1. Collect a small batch of experiences interacting with the environment.
2. Use that batch to update decision-making policy.
3. Once the policy is updated with this batch, the experiences are thrown away and a newer batch is collected with the newly updated policy.

Learning Progress



Emergent behaviors

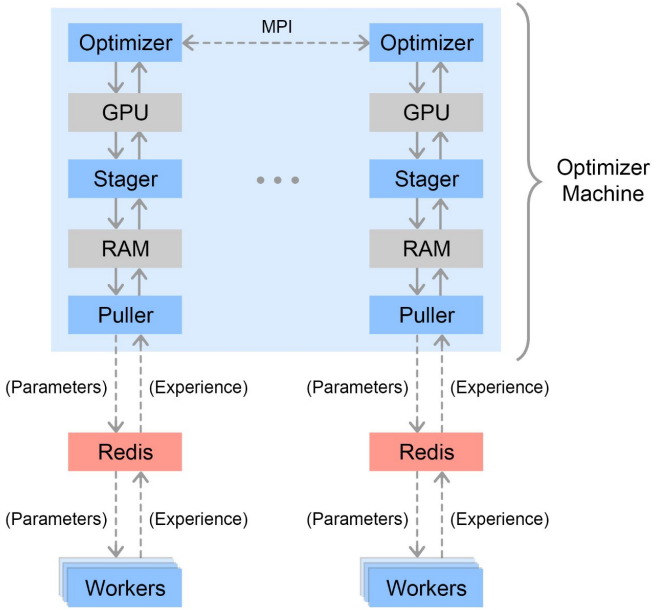


Engineering

384 worker machines, each with 16 CPU cores
Optimization performed on a single machine with 8 V100 GPUs

Generates ~2 years of simulated experience per hour

“Workers download the newest policy parameters from the optimizer at the beginning of every epoch, generate training episodes, and send the generated episodes back to the optimizer.”



I have a joke about reinforcement learning...

...but I won't get a reward for telling it.

Open Questions

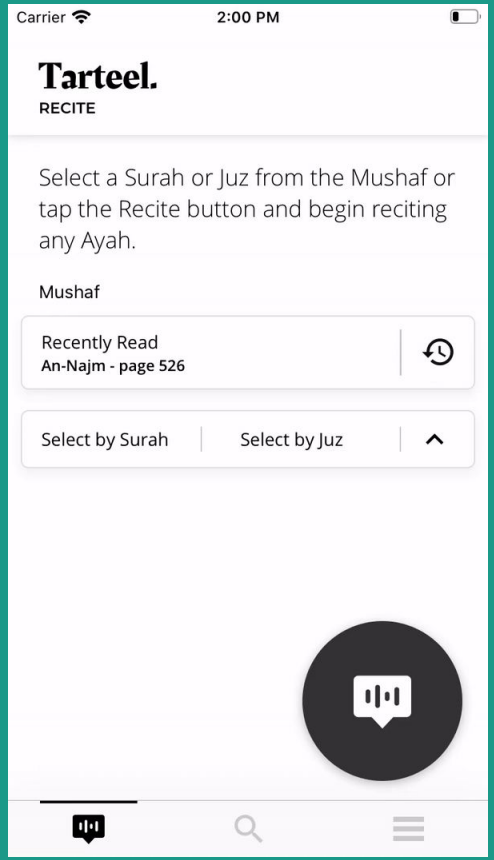
1. How to generalize to other tasks? (Meta-learning, Few shot learning, Evolutionary strategies?)
2. Why do we hit a limit of 50 consecutive tasks? (Does LSTM *really* have long term memory?)
    a. Did it overfit?
3. What model of the world did the agent *actually* learn?
   (What happens if we add grease or peanut butter to the cube? :))

Excited? Get started! https://spinningup.openai.com/en/latest/

Questions?

Shameless Plug: Checkout **Tarteel!**

https://www.download.tarteel.io



Resources

https://joshgreaves.com/reinforcement-learning/introduction-to-reinforcement-learning/
https://bitsandatoms.co/primer-reinforcement-learning/
https://openai.com/blog/learning-dexterity/