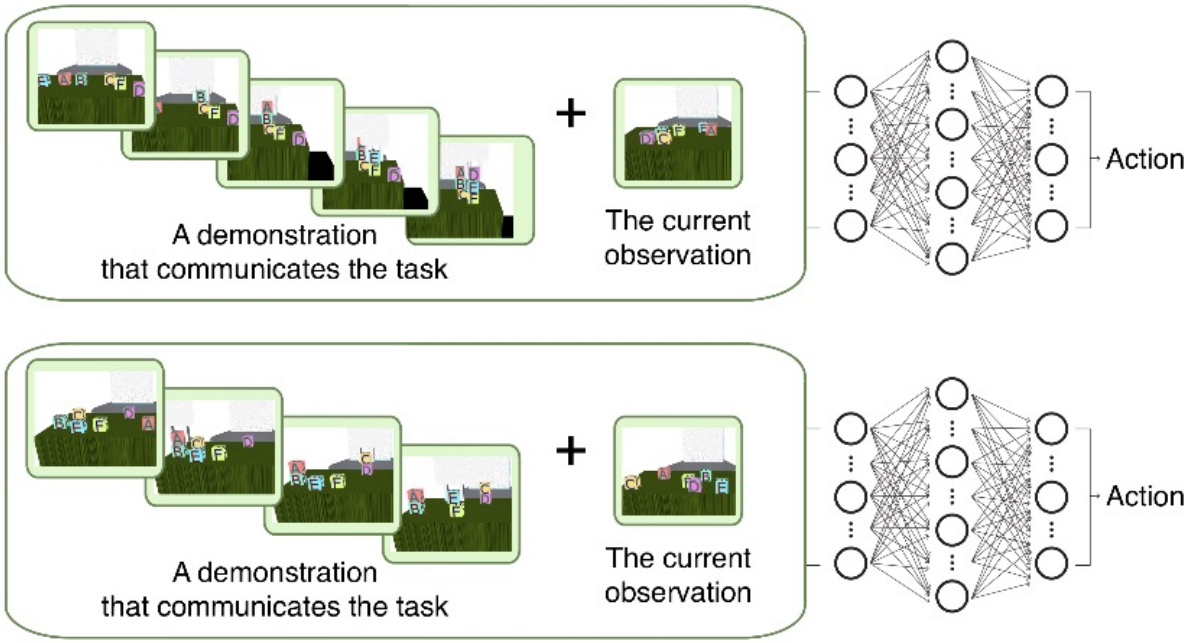
1703.07326 - One-Shot Imitation Learning

- **Yunqiu Xu**
- Other reference:
  - https://zhuanlan.zhihu.com/p/27935902
  - https://blog.openai.com/robots-that-learn/
  - https://github.com/DanielTakeshi/Paper_Notes/blob/master/reinforcement_learning/One-Shot_Imitation_Learning.md

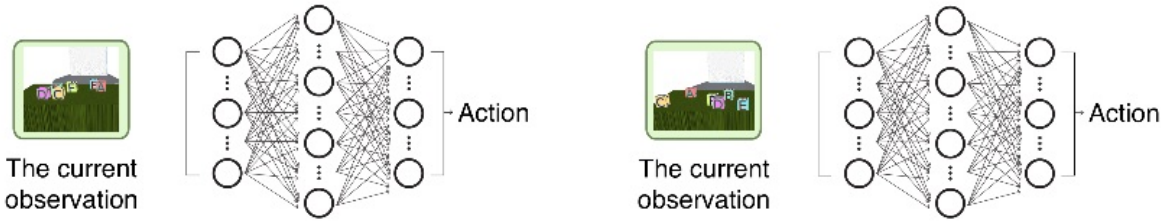
---

1. Introduction

- Challenges of imitation learning
  - careful feature engineering
  - large number of samples
- Our goal: learn from very few demonstrations of some tasks, then generalize to new situations of these tasks without task-specific engineering
- Our method:
  - Train a policy on a large set of tasks
  - Input:
    - current state
    - one demonstration that solves a different instance for the same task
  - Output: current controls
  - Use soft-attention for generalization



One-shot policy. A single policy trained to solve many tasks.



(left) Task-specific policy. This policy is trained to stack blocks into two towers, each of height 3. (right) A separate task-specific policy. This policy is trained to stack blocks into three towers, each of height 2.

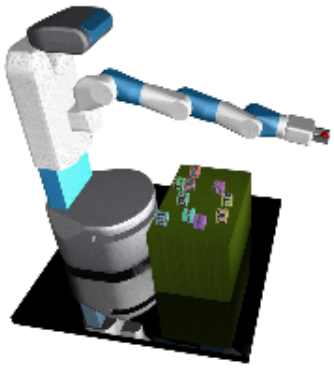
2. Related Work

- Main lines of imitation learning:
  - Behavioral cloning: performs supervised learning from observations to actions
  - Inverse reinforcement learning: reward function is estimated that explains the demonstrations as (near) optimal behavior
  - These 2 methods consider each skill separately $\rightarrow$ learn one skill can not accelerate the learning to imitate another skill
- One-shot and few-shot learning:
  - Many of the afore-mentioned approaches are a form of meta-learning

  - Learn the algorithm itself $\rightarrow$ learning to learn
- Another related work is 1703.01703 - Third-Person Imitation Learning , but not good enough: learning uses one demonstration on whatever new task is being considered
- Our method relies on "soft attention" and sequence-to-sequence model
  - an attention model over the demonstration
  - an attention model over the current observation

3. One Shot Imitation Learning

- $t \in T$: a task in the distribution of tasks
- $d \in D(t)$
  - $D(t)$ is the distribution of demonstrations of task t
  - d is a demonstration in $D(t)$
  - $d = [(o_1, a_1), (o_2, a_2), \ldots, (o_T, a_T)]$: a demonstration is a sequence of observations and actions
  - We assume T is given $\rightarrow$ for each $t \in T$ we can get demonstrations from $D(t)$
- $\pi_\theta(a|o, d)$: policy
- $R_t(d)$: evaluation function
- Objective: maximize the expected performance of the policy, where the expectation is taken over tasks $t \in T$, and demonstrations $d \in D(t)$
- An example: block stacking



An example of the initial state, where blocks are randomly placed on the table.

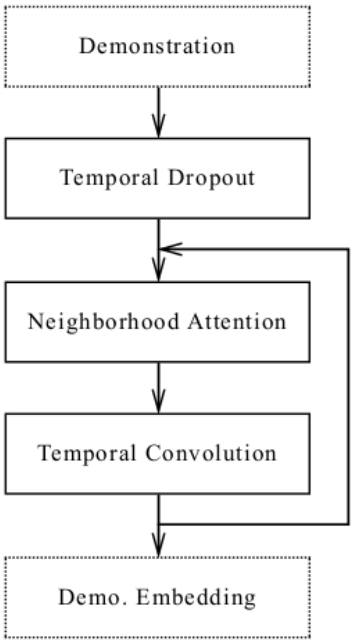
Trajectory of stacking 4 towers of height 2 each, where block A is on top of block B, block C is on top of block D, block E is on top of block F, and block G is on top of block H. This task is identified as `ab cd ef gh`.

Trajectory of stacking 1 block tower of height 4, where block G is on top of block H, block H is on top of block I, and block I is on top of block J. This task is identified as `ghij`.

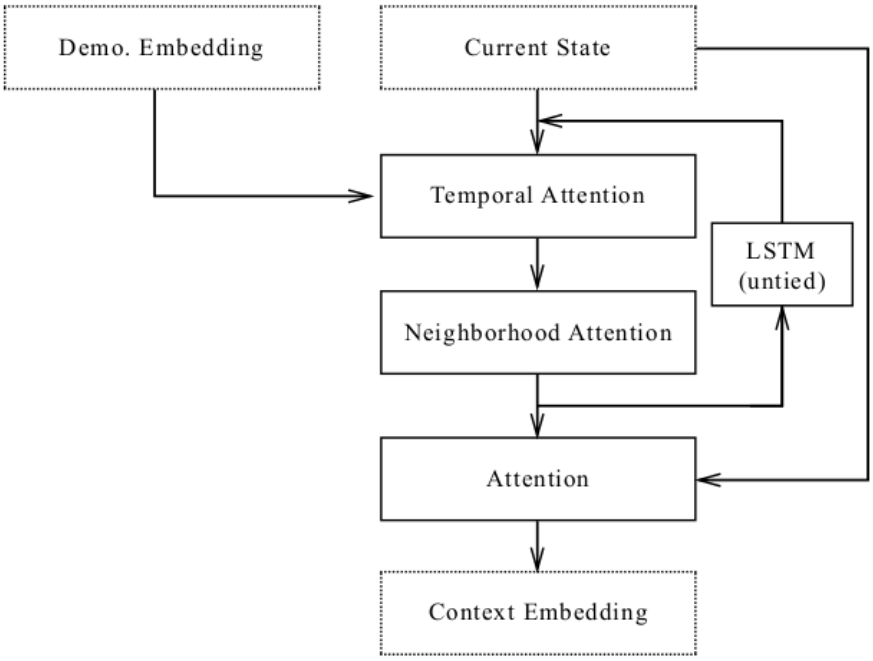
Figure 3. The tasks are to control a Fetch robotic arm to stack blocks into various layouts. Entire episode takes up to several thousand time-steps. We define a *stage* as a single operation of stacking one block on top of another. The first task shown above has 4 stages, whereas the second task has 3.

4. Architecture

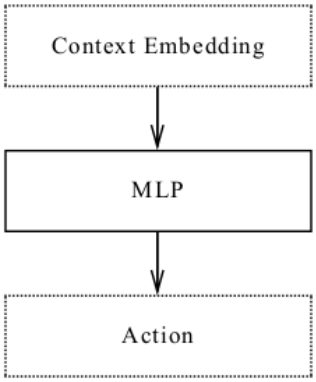
- Particle reaching: 3 candidates
  - Plain LSTM
  - LSTM with attention
  - Final state with attention
- Block stacking: 3 modules
  - Demonstration network:
    - Input: the entire demonstration
    - Output: some embedding
    - This demonstration can be long, so they need to subsample for timestep



  - Context network:
    - Inpput: embedding from demonstration, and current state,
    - Output: context embedding
    - the demonstration embedding is going to be fixed, so the input at time t is the concatenated vector (d, x_t)

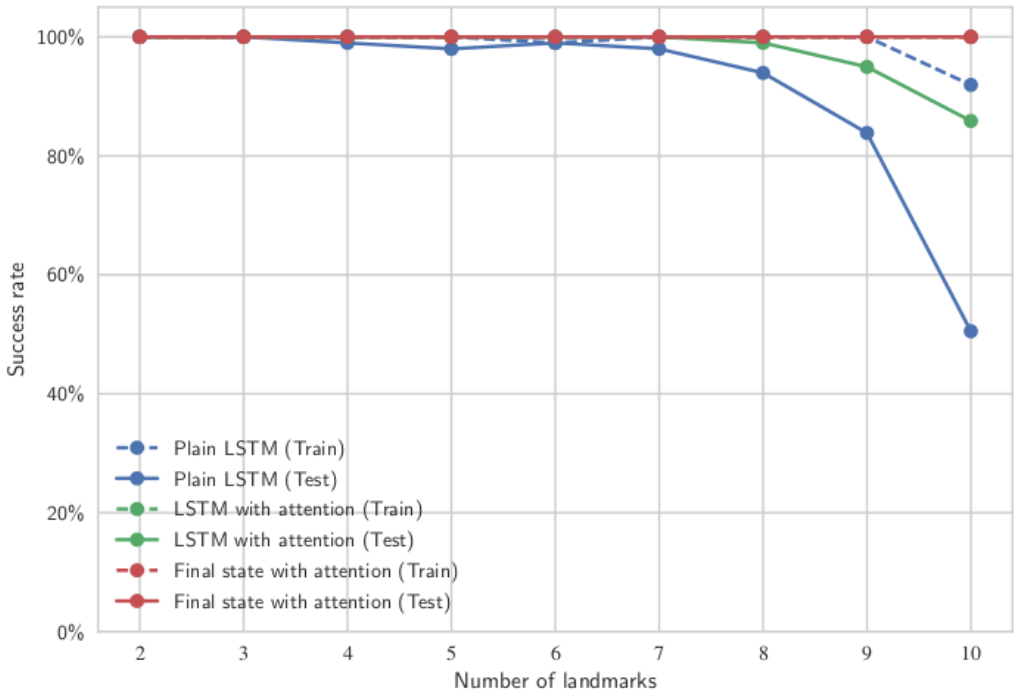


- Manipulation network:
  - Input: embedding from context network
  - Output: the action to be taken



5. Experiments

- Particle Reaching



- Block stacking: Figure 9 - Figure 13