Meta-Reinforcement Learning: Learning to Explore

CS224R
Reminders

Homework 3 due **tonight**
(and HW4 out today)

Project milestone due **next Wednesday**

**Following up on high-res feedback:**
- Wanting homeworks to require more conceptual understanding
- Request for summary table of approaches
- Unofficial lecture notes
Why meta-reinforcement learning?

Why are humans good at RL?

People have previous experience.
They have developed representations that facilitate exploration & learning.

Our RL agents start tabula rasa.

Can we allow RL agents to leverage prior experience?

Source: https://i.imgur.com/hJIVfZ5.jpg
Should we be using the same exploration algorithm for:

- Learning to *navigate an environment*
- Learning to *make recommendations to users*
- Learning a policy for *computer system caching*
- Learning to *physically operate a new tool or machine*

This is how we currently approach exploration.
Today’s Lecture

Can we *learn exploration strategies* based on experience from other tasks in that domain?
Outline

Brief Recap on Meta-RL

Algorithms for Learning to Explore
End-to-End Optimization of Exploration Strategies
Alternative Decoupled Exploration Strategies
Decoupled but Consistent Exploration & Exploitation

Case Study: Applying Meta-RL to CS Education
A brief recap of meta-reinforcement learning

Goal:

Collect small amount of experience in new MDP

Collect $\mathcal{D}_{tr} \sim \pi^{exp}$

Learn policy that solves that MDP

$\mathcal{D}_{tr} \rightarrow \pi^{task}$

diagram adapted from Duan et al. ‘17
A brief recap of meta-reinforcement learning

**Meta-Train Time:**
Learn how to efficiently explore & solve many MDPs:

- Meta-train $\pi^{\text{exp}}, \pi^{\text{task}}$

**Meta-Test Time:**
Collect small amount of experience in new MDP

- Collect $\mathcal{D}_{tr} \sim \pi^{\text{exp}}$
- $\mathcal{D}_{tr} \rightarrow \pi^{\text{task}}$

**Key assumption:** Meta-training & meta-testing MDPs come from same distribution.

(so that we can expect generalization)

diagram adapted from Duan et al. ‘17
A brief recap of meta-reinforcement learning

**Common approach:** Implement the learning procedure with a recurrent network.

\[ \nabla_{\theta} J(\theta) = E_{T \sim \pi_{\theta}, T_i} \left[ \left( \sum_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t, D_{i}^{tr}) \right) \left( \sum_t r_i(s_t, a_t) \right) \right] \]

Wang et al. Learning to Reinforcement Learn. 2017; Duan et al. RL\(^2\). 2017
A brief recap of meta-reinforcement learning

Examples of meta-RL tasks

Navigation through different mazes

Locomotion on different terrains, slopes

Object manipulation with different objects, goals

Dialog with different users w/ different preferences
Outline

Brief Recap on Meta-RL

**Algorithms for Learning to Explore**
End-to-End Optimization of Exploration Strategies
Alternative Decoupled Exploration Strategies
Decoupled but Consistent Exploration & Exploitation

Case Study: Applying Meta-RL to CS Education
How Do We Learn to Explore?

Solution #1: Optimize for Exploration & Exploitation *End-to-End* w.r.t. Reward

(Duan et al., 2016, Wang et al., 2016, Mishra et al., 2017, Stadie et al., 2018, Zintgraf et al., 2019, Kamienny et al., 2020)

+ simple
+ leads to optimal strategy in principle

-- challenging optimization when exploration is hard
A simple, running example

Hallway 1

Hallway 2

Hallway N

information on where to go

Different tasks: navigating to the ends of different hallways
How Do We Learn to Explore?

Solution #1: Optimize for Exploration & Exploitation *End-to-End* w.r.t. Task Reward

(Duan et al., 2016, Wang et al., 2016, Mishra et al., 2017, Stadie et al., 2018, Zintgraf et al., 2019, Kamienny et al., 2020)

Example episodes during meta-training:

- agent goes to the end of the correct hallway
  - gets positive reward for current task, but $D_i^{tr}$ won’t be different than for any other task
  - +/- provides signal on a *suboptimal* exploration + exploitation strategy
- agent goes to wrong hallway then correct hallway
- agent looks at the instructions
  - good exploratory behavior, but won’t get any reward for this behavior

*It’s hard to learn exploration & exploitation at the same time!*
Another Example of a Hard Exploration Meta-RL Problem

Learned cooking tasks in previous kitchens

**Goal**: Quickly learn tasks in a new kitchen.

meta-training

meta-testing
Why is End-to-End Training Hard in This Example?

**End-to-end approach:** optimize exploration and execution episode behaviors end-to-end to maximize reward of execution

**Ingredient not found** (bad exploration) → **Cannot learn to cook** (bad execution) → **Cannot cook** (bad execution) → Low reward for any exploration

**Coupling problem:** learning exploration and execution depend on each other

—> can lead to poor local optima, poor sample efficiency

Liu, Raghunathan, Liang, Finn. *Decoupling Exploration and Exploitation for Meta-Reinforcement Learning without Sacrifices*. ICML 2021
2a. Use posterior sampling (also called Thompson sampling)

PEARL (Rakelly, Zhou, Quillen, Finn, Levine. ICML ’19)

i. Learn distribution over latent task variable $p(z), q(z | \mathcal{D}_{tr})$ and corresponding task policies $\pi(a | s, z)$

ii. Sample $z$ from current posterior and sample from policy $\pi(a | s, z)$

When might posterior sampling be bad? Eg. Goals far away & sign on wall that tells you the correct goal.
Solution #2: Leverage Alternative Exploration Strategies

2a. Use posterior sampling
(also called Thompson sampling)
   i. Learn distribution over latent task variable \( p(z) \), \( q(z | D_{tr}) \) and corresponding task policies \( \pi(a | s, z) \)
   ii. Sample \( z \) from current posterior and sample from policy \( \pi(a | s, z) \)

2b. Use intrinsic rewards

2c. Task dynamics & reward prediction
   i. Train model \( f(s', r | s, a, D_{train}) \)
   ii. Collect \( D_{train} \) so that model is accurate.

PEARL (Rakelly, Zhou, Quillen, Finn, Levine. ICML ‘19)
MAME (Gurumurthy, Kumar, Sycara. CoRL ‘19)
MetaCURE (Zhang, Wang, Hu, Chen, Fan, Zhang. ‘20)

When might this be bad?
Lots of distractors,
or complex, high-dim state dynamics
Solution #2: Leverage Alternative Exploration Strategies

2a. Use posterior sampling
   (also called Thompson sampling)
   i. Learn distribution over latent task variable $p(z), q(z | \mathcal{D}_{tr})$ and corresponding task policies $\pi(a | s, z)$
   ii. Sample $z$ from current posterior and sample from policy $\pi(a | s, z)$

2b. Use intrinsic rewards

2c. Task dynamics & reward prediction
   i. Train model $f(s', r | s, a, \mathcal{D}_{train})$
   ii. Collect $\mathcal{D}_{train}$ so that model is accurate.

+ easy to optimize
+ many based on principled strategies

-- suboptimal by arbitrarily large amount in some environments.
Can we avoid the chicken-and-egg problem without sacrificing optimality?
(best of both worlds?)
Yes!
Solution #3

Idea from solution #2b: Train model $f(s', r | s, a, D_{tr})$ & collect $D_{tr}$ so that model is accurate.

Do we have to learn a full dynamics & reward model?

Idea 3.0: Label each training task with a unique ID $\mu$

**Meta training**

Exploration policy: train policy $\pi^{\text{exp}}(a | s)$ and task identification model $q(\mu | D_{tr})$

such that $D_{tr} \sim \pi^{\text{exp}}$ allows accurate task prediction from $f$

Execution policy: train ID-conditioned policy $\pi^{\text{exec}}(a | s, \mu_i)$

**Meta testing**

Explore: $D_{tr} \sim \pi^{\text{exp}}(a | s)$  Infer task: $\hat{\mu} \sim q(\mu | D_{tr})$  Perform task: $\pi^{\text{exec}}(a | s, \hat{\mu})$

+ no longer need to model dynamics, rewards  — may not generalize well for one-hot $\mu$
Solution #3: **Decouple** by acquiring representation of task relevant information

1) Learn execution & identify key information

- MDP identifier $\mu$
- Wall color
- Ingredients
- Decorations

2) Learn to explore by recovering that information

- Exploration policy
- Exploration episode $\mathcal{T}$
- Information recovery reward $\text{MI}(z; \tau)$

---

Liu, Raghunathan, Liang, Finn. *Decoupling Exploration and Exploitation for Meta-Reinforcement Learning without Sacrifices*. ICML 2021
Solution #3: **Decouple** by acquiring representation of task relevant information

1) Learn execution & identify key information

- MDP identifier $\mu$
- Wall color
- Ingredients
- Decorations

2) Learn to explore by recovering that information

- Exploration policy $\pi^{\text{exp}}$
- Exploration episode $\mathcal{T}$
- Information recovery reward $\text{MI}(z; \tau)$

Train $\pi^{\text{exec}}(a | s, z_i)$ and encoder $F(z_i | \mu_i)$ to:

$$\max \sum_i \mathbb{E}_{\pi^{\text{exec}}} [r_i] - D_{\text{KL}} (F(z_i | \mu_i) \| \mathcal{N}(0, 1))$$

Train $\pi^{\text{exp}}$ such that collected $\mathcal{D}_{\text{tr}}$ is predictive of $z_i$.

In practice: (1) and (2) can be trained simultaneously.

Liu, Raghunathan, Liang, Finn. *Decoupling Exploration and Exploitation for Meta-Reinforcement Learning without Sacrifices*. ICML 2021
Solution #3: **Decouple** by acquiring representation of task relevant information

<table>
<thead>
<tr>
<th>Meta-training</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Learn execution &amp; identify key information</td>
</tr>
<tr>
<td>2) Learn to explore by recovering that information</td>
</tr>
</tbody>
</table>

Train $\pi^{\text{exec}}(a | s, z_i)$ and encoder $F(z_i | \mu_i)$ to:

$$\max \sum_i \mathbb{E}_{\pi^{\text{exec}}} [r_i] - D_{KL} (F(z_i | \mu_i) \| \mathcal{N}(0, 1))$$

Train $\pi^{\exp}$ such that collected $\mathcal{D}_{\text{tr}}$ is predictive of $z_i$.

How to formulate the **reward function** for $\pi^{\exp}$?

(a) Train model $q(z_i | \mathcal{D}_{\text{tr}})$
(b) $r_i =$ per-step information gain

$$r_i = \text{prediction error from } \tau_{1:t-1} - \text{prediction error from } \tau_{1:t}$$

**Decoupled Reward-free ExplorAtion and Execution in Meta-Reinforcement Learning (DREAM)**

Liu, Raghunathan, Liang, Finn. *Decoupling Exploration and Exploitation for Meta-Reinforcement Learning without Sacrifices*. ICML 2021
Aside: How can we bottleneck the information in a neural net’s representation?

**V0:** Add noise the representation.

\[ \epsilon \sim \mathcal{N}(0, I) \quad \tilde{z} = z + \epsilon \]

+ will discard information 😊

- if done at test time, my discard good info
- if done during training, model can increase magnitude of \( z \)

**Key ideas:**
1. Add Gaussian noise during training
2. Prevent the model from increasing magnitude

**V1:** Variational information bottleneck

Add noise before passing representation to next layer:

\[ \epsilon \sim \mathcal{N}(0, I) \quad \tilde{z} = z + \epsilon \]

Modify loss term:

\[ L_{tr} + \| z \|^2 \]

\[ \rightarrow \text{equivalent to} \quad D_{KL} \left( F(z | \mu_i) \| \mathcal{N}(0,1) \right). \]

Solution #3: **Decouple** by acquiring representation of task relevant information

(Informal) Theoretical Analysis

(1) **DREAM** objective is *consistent* with end-to-end optimization. 
   -> can in principle recover the optimal exploration strategy [under mild assumptions]

(2) Consider a bandit-like setting with $|\mathcal{A}|$ arms.
   In MDP $i$, arm $i$ yields reward. In all MDPs, arm 0 reveals the rewarding arm.

   RL\textsuperscript{2} requires $\Omega(|\mathcal{A}|^2 \log |\mathcal{A}|)$ samples for meta-optimization.
   DREAM requires $\mathcal{O}(|\mathcal{A}| \log |\mathcal{A}|)$ samples for meta-optimization.

   [assuming Q-learning with uniform outer-loop exploration]

Liu, Raghunathan, Liang, Finn. *Decoupling Exploration and Exploitation for Meta-Reinforcement Learning without Sacrifices*. ICML 2021
Empirical Comparison: Sparse Reward 3D Visual Navigation Problem

- Task: go to the (key / block / ball), color specified by the sign
- Agent starts on other side of barrier, must walk around to read the sign
- Pixels observations (80 x 60 RGB)
- Sparse binary reward

More challenging variant of task from Kamienny et al., 2020

Liu, Raghunathan, Liang, Finn. Explore then Execute: Adapting without Rewards via Factorized Meta-RL. ICML 2021
Quantitative Comparison

- End-to-end algorithms ($RL^2$, IMPORT, VARIBAD) perform poorly due to coupling
- **PEARL-UB**: Upper-bound on PEARL: optimal policy and Thompson-Sampling exploration, does not learn the optimal exploration strategy
- **DREAM** achieves near-optimal reward

---

RL$^2$ (Duan et al., 2016), IMPORT (Kamienny et al., 2020), VARIBAD (Zintgraf et al., 2019), PEARL (Rakelly, et. al., 2019), Thompson, 1933

---

Liu, Raghunathan, Liang, Finn. *Explore then Execute: Adapting without Rewards via Factorized Meta-RL*. ICML 2021
Qualitative Results for DREAM

Exploration episode

Execution episode
Goal: Go to key
## How Do We Learn to Explore?

<table>
<thead>
<tr>
<th>End-to-End</th>
<th>Alternative Strategies</th>
<th>Decoupled Exploration &amp; Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ leads to optimal strategy in principle</td>
<td>+ easy to optimize</td>
<td>+ leads to optimal strategy in principle</td>
</tr>
<tr>
<td>- challenging optimization when exploration is hard</td>
<td>+ many based on principled strategies</td>
<td>+ easy to optimize in practice</td>
</tr>
<tr>
<td></td>
<td>- suboptimal by arbitrarily large amount in some environments.</td>
<td>- requires task identifier</td>
</tr>
</tbody>
</table>
Outline

Brief Primer on Meta-RL

Algorithms for Learning to Explore

End-to-End Optimization of Exploration Strategies

Alternative Decoupled Exploration Strategies

Decoupled but Consistent Exploration & Exploitation

Case Study: Applying Meta-RL to CS Education
Problem: Providing Feedback on Interactive Software

Common CS assignment: interactive software

Bounce assignment (Code.org)
**Problem:** Providing Feedback on Interactive Software

Common CS assignment: *interactive software*

Motivating and engaging (fun!)  
⇒ can enrich learning (Pfaffman et al., 2003)
**Problem:** Providing Feedback on Interactive Software

Common CS assignment: **interactive software**

- Bounce assignment (Code.org)
- Motivating and engaging (fun!) ⇒ can enrich learning (Pfaffman et al., 2003)

Increasingly found everywhere

- Harvard CS50
- UC Berkeley CS61B
- UCLA CS32
- Stanford CS106A
- Code.org
- Camp K12
- Tynker
- Google Applied CS Skills
  ...

Problem: Providing Feedback on Interactive Software

Providing feedback / grading is hard

Bounce assignment (Code.org)
Problem: Providing Feedback on Interactive Software

Providing feedback / grading is hard

Bounce assignment (Code.org)

Stochasticity
**Problem:** Providing Feedback on Interactive Software

*Providing feedback / grading is hard*

Bounce assignment (Code.org)

Stochasticity
**Problem:** Providing Feedback on Interactive Software

Providing feedback / grading is hard

Bounce assignment (Code.org)

Stochasticity

Student creativity
Problem: Providing Feedback on Interactive Software

- 3+ min manual grading per assignment
- 70M+ enrolled students

\[ \text{Bounce assignment (Code.org)} \]

\[ \text{300+ years of grading work} \]
**Problem:** Providing Feedback on Interactive Software

Our goal: Automatically provide feedback

- Reduce enormous human grading burden
- Provide *faster* and *iterative* feedback
Setting

Rubric: List of possible errors

- moveError
- whenWall-newBallError
- whenGoal-scoreError
- ...

Setting

Training (~3500 labeled programs)

Program \( \mu \)

Label \( y \): Subset of rubric items present in program

- moveError
- whenGoal - scoreError
- ...

Rubric: List of possible errors

- moveError
- whenWall - newBallError
- whenGoal - scoreError
- ...

Score 2 / 0
Setting

Training (~3500 labeled programs)

Program $\mu$

Label $y$: Subset of rubric items present in program

- moveError
- whenGoal scoreError
- ...

Testing

New student program
Setting

---

**Goal:** Output which bugs are in the program (i.e., predict the label)

---

**Training (~3500 labeled programs)**

- **Program** $\mu$
- **Label $y$:** Subset of rubric items present in program
  - `moveError`
  - `whenGoal scoreError`
  - ...

---

**Testing**

- **New student program**

---
Related Work: Two Paradigms for Automated Feedback

Analyze program code
(Singh et al., ’13, Piech et al., ’15, Bhatia et al., ’16, Rivers et al., ’17, Paasben et al., ’17, Wang et al., ’17, Malik et al., ’19, Wu et al., ’19, Wu et al., ’21)

- Works well for shorter programs (e.g., <50 lines of code)
- Existing methods struggle to scale to longer programs

Analyze program behavior
(King et al., ’76, Godefroid et al., ’08, Zheng et al., ’19, Nie et al., ’21, Gordillo et al., ’21)

- Independent of program length
- Assumes that the program can compile and run

We opt for this approach
The Play-to-Grade Paradigm

Agent interacts with program like human
The Play-to-Grade Paradigm

Score 0 : 0

Agent interacts with program like human
The Play-to-Grade Paradigm

Agent interacts with program like human
The Play-to-Grade Paradigm

Agent interacts with program like human
The Play-to-Grade Paradigm

Agent interacts with program like human
The Play-to-Grade Paradigm

Agent interacts with program like human

Agent outputs feedback
The Play-to-Grade Paradigm

Agent interacts with program like human

Agent outputs feedback

Existing work (Nie et al., '21): Coarse binary feedback

The program is incorrect

Not specific enough for student to learn and correct mistakes
The Play-to-Grade Paradigm

Agent interacts with program like human

Score 0 : 0

The program is incorrect

Existing work (Nie et al., '21):
Coarse binary feedback

Not specific enough for student to learn and correct mistakes

Our goal: **Fine-grained** feedback

The ball incorrectly bounces off the goal

Agent outputs feedback
What makes providing feedback hard?

Targeted exploration

Adaptive exploration
**Approach:** DREAMGRADE

**Explanation policy** $\pi$:
- Takes program $\mu$ and produces trajectories $\tau$

**Feedback classifier** $g$:
- Takes trajectories $\tau$ and predicts label $y$

- $\text{whenWall-newBallError}$
- $\text{whenWall-scoreError}$
- $\ldots$
**Approach:** DREAMGRADER

**Exploration policy** $\pi$: Takes program $\mu$ and produces trajectories $\tau$

**Feedback classifier** $g$: Takes trajectories $\tau$ and predicts label $y$

Maximize probability of correct label

$$J(\pi, g) = \mathbb{E}_{\mu \sim p(\mu), \tau \sim \pi(\mu)} \left[ g(y \mid \tau) \right]$$

Sample a program and roll out exploration policy
**Approach:** DREAMGRADER

**Exploration policy** $\pi$:
Takes program $\mu$ and produces trajectories $\tau$

**Feedback classifier** $g$:
Takes trajectories $\tau$ and predicts label $y$

Naive approach:
Treat this as end-of-episode reward

$$J(\pi, g) = \mathbb{E}_{\mu \sim p(\mu), \tau \sim \pi(\mu)} \left[ g(y \mid \tau) \right]$$

Sample a program and roll out exploration policy
**Approach:** DREAMGRADER
Approach: DREAMGRADER
**Approach:** DREAMGRADER
Approach: DREAMGRADER
**Approach:** DREAMGRADER
**Approach:** DREAMGRADER

Naive approach

\[ J(\pi, g) = \mathbb{E}_{\mu \sim p(\mu), \tau \sim \pi(\mu)} [g(y | \tau)] \]

End-of-episode reward
**Approach:** DREAMGRADER

\[ \mathcal{J}(\pi, g) = \mathbb{E}_{\mu \sim \rho(\mu), \tau \sim \pi(\mu)} [g(y | \tau)] \]

**Naive approach**

End-of-episode reward

Reward given at 4 but bug discovered at 3
Approach: DREAMGRADER

Naive approach

\[ \mathcal{J}(\pi, g) = \mathbb{E}_{\mu \sim p(\mu), \tau \sim \pi(\mu)} [g(y | \tau)] \]

End-of-episode reward

Reward given at 4 but bug discovered at 3

Instead, use DREAM (Liu et al., '21) to provide credit at 3
**Approach:** DREAMGRADER

**Naive approach**

\[ J(\pi, g) = \mathbb{E}_{\mu \sim p(\mu), \tau \sim \pi(\mu)} [g(y | \tau)] \]

End-of-episode reward

Reward given at 4 but bug discovered at 3

Instead, use DREAM (Liu et al., '21) to provide credit at 3

Intuition: maximize information gain

\[ r_t = \log \frac{g(y | \tau_{t+1})}{g(y | \tau_t)} \]
**Approach:** DREAMGRADER

Why does the DREAM meta-RL algorithm apply here?

**Few-shot meta-RL:**

1) Agent is given new task
2) Agent gets to explore for a few episodes
3) Agent uses exploration to maximize returns on new episode
**Approach:** DREAMGRADER

Why does the DREAM meta-RL algorithm apply here?

**Few-shot meta-RL:**

1) Agent is given new task
2) Agent gets to explore for a few episodes
3) Agent uses exploration to maximize returns on new episode

Agent is given new program
Approach: DREAMGRADER

Why does the DREAM meta-RL algorithm apply here?

Few-shot meta-RL:

1) Agent is given new task
2) Agent gets to explore for a few episodes
3) Agent uses exploration to maximize returns on new episode

Agent is given new program

Exploration policy runs to find bugs
**Approach:** **DREAMGRADER**

Why does the DREAM meta-RL algorithm apply here?

**Few-shot meta-RL:**

1) Agent is given new task

2) Agent gets to explore for a few episodes

3) Agent uses exploration to maximize returns on new episode

**Agent is given new program**

**Exploration policy runs to find bugs**

**Feedback classifier uses exploration to predict label**
Experiments: Questions

How feasible is automated feedback generation?

Can automated feedback generation handle student creativity?
Experiments: Questions

How feasible is automated feedback generation?

Can automated feedback generation handle student creativity?

Bounce programming assignment from Code.org
Dataset of ~700K real student submissions, released by Nie et al., ‘21
Train systems on 3,500 programs — hold out the rest
**Experiments: Questions**

How feasible is automated feedback generation?

Humans  Naive approach of direct maximization  Existing state-of-the-art approach (Nie et al., '21)

Can automated feedback generation handle student creativity?

Bounce programming assignment from Code.org

Dataset of ~700K real student submissions, released by Nie et al., '21

Train systems on 3,500 programs — hold out the rest
**Experiments:** How Feasible is Automated Feedback?

Humans are accurate, but infeasible: Requires ~4 years to grade the dataset.

![Graph showing feedback accuracy and F1 metrics for different methods.](image)
**Experiments:** How Feasible is Automated Feedback?

Humans are accurate, but **infeasible**: Requires ~4 years to grade the dataset.

DREAMGRADER achieves within 1.5% of human accuracy.

---

**Feedback Accuracy and F1**

- **Accuracy**
- **F1**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Human</th>
<th>Naive approach</th>
<th>DREAMGRADER</th>
<th>Nie et al., 2021</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>95 ± 2</td>
<td>90 ± 3</td>
<td>97 ± 1</td>
<td>96 ± 2</td>
</tr>
<tr>
<td>F1</td>
<td>85 ± 2</td>
<td>80 ± 3</td>
<td>87 ± 1</td>
<td>86 ± 2</td>
</tr>
</tbody>
</table>
Humans are accurate, but infeasible: Requires ~4 years to grade the dataset.

DREAMGRADER achieves within 1.5% of human accuracy.

Appropriate credit assignment is critical for learning effective exploration.

### Experiments: How Feasible is Automated Feedback?

![Feedback Accuracy and F1](image.png)

- **Accuracy**
  - Human
  - DREAMGRADER
  - Naive approach
  - Nie et al., 2021

- **F1**
  - Human
  - DREAMGRADER
  - Naive approach
  - Nie et al., 2021
Humans are accurate, but infeasible: Requires ~4 years to grade the dataset.

DREAMGRADER achieves within 1.5% of human accuracy.

Appropriate credit assignment is critical for learning effective exploration.

Improves on existing methods by 18.8%.

**Experiments:** How Feasible is Automated Feedback?
**Experiments:** Learned Exploration Behavior

What happens when...

- the ball hits the goal?
- the ball hits the floor?
- the ball hits the wall?

Main gap with humans appears in these sorts of programs with many balls.
Experiments: Can We Handle Some Student Creativity?

One type of student creativity in the dataset: ball and paddle speed

- Test handling student creativity by evaluating on held out ball and paddle speeds

<table>
<thead>
<tr>
<th></th>
<th>Both held out</th>
<th>Held out ball speed</th>
<th>Held out paddle speed</th>
<th>Neither held out</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>88.0%</td>
<td>88.8%</td>
<td>88.2%</td>
<td>88.4%</td>
</tr>
<tr>
<td>Precision</td>
<td>38.8%</td>
<td>41.6%</td>
<td>44.9%</td>
<td>38.6%</td>
</tr>
<tr>
<td>Recall</td>
<td>82.1%</td>
<td>87.2%</td>
<td>91.4%</td>
<td>85.6%</td>
</tr>
<tr>
<td>F1</td>
<td>52.8%</td>
<td>56.3%</td>
<td>60.2%</td>
<td>53.2%</td>
</tr>
</tbody>
</table>

Performance on held out speeds roughly matches speeds seen during training
**Bonus Experiment:** Beyond Code.org bounce game?

- Stanford CS106A: Students program Breakout in homework assignment
- Ball “skewering” bug: common mistake, most difficult to detect/grade

Learned exploration policy
Outline

Brief Recap on Meta-RL

Algorithms for Learning to Explore
End-to-End Optimization of Exploration Strategies
Alternative Decoupled Exploration Strategies
Decoupled but Consistent Exploration & Exploitation

Case Study: Applying Meta-RL to CS Education
Reminders

Homework 3 due **tonight**
(and HW4 out today)

Project milestone due **next Wednesday**

**Next week:** Can we make reinforcement learning more autonomous? Can RL agents discover skills themselves? Can we do hierarchical RL?