Learning state representations for formal verification

David Kerkkamp

Outline

- Motivation and goal
- State representation learning
- Pipeline
- Pong demo
- Conclusion

Motivation and goal

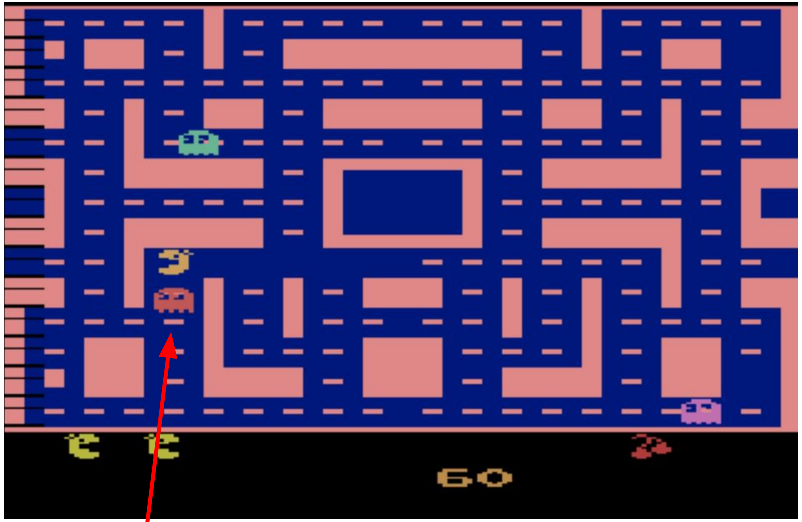
Motivation:

- Verified guarantees against dangerous or fatal situations in systems involving AI/ML
- To apply formal verification, a suitable model of the environment is needed

Goal:

- Learning models of environment suitable for model checking
- Focus on learning state representations of Atari 2600 games

Atari 2600



Enemy near

Enemy

Falling off edge

Plan

Create a pipeline that learns state representations and generates a model:

- State representation learning: based on codebase from paper by Anand et al. [1]
    - Contains techniques for learning representations of Atari games
- Generate a model: Markov Decision Process

[1] A. Anand, E. Racah, S. Ozair, Y. Bengio, M.-A. Côté, and R. D. Hjelm, "Unsupervised state representation learning in atari," *arXiv preprint arXiv:1906.08226*, 2019.

State representation learning

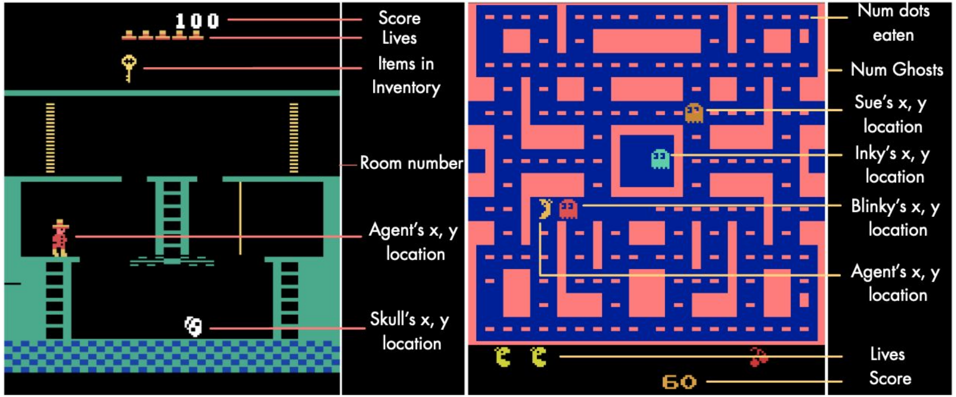
Atari Annotated RAM Interface (AtariARI)

Using OpenAI Gym

Learning representations in two parts:

- Training an encoder
- Predict state labels with linear probing
    - For each label, a 256-way linear classifier is trained

Different encoder architectures are possible



Codebase

Existing code for:

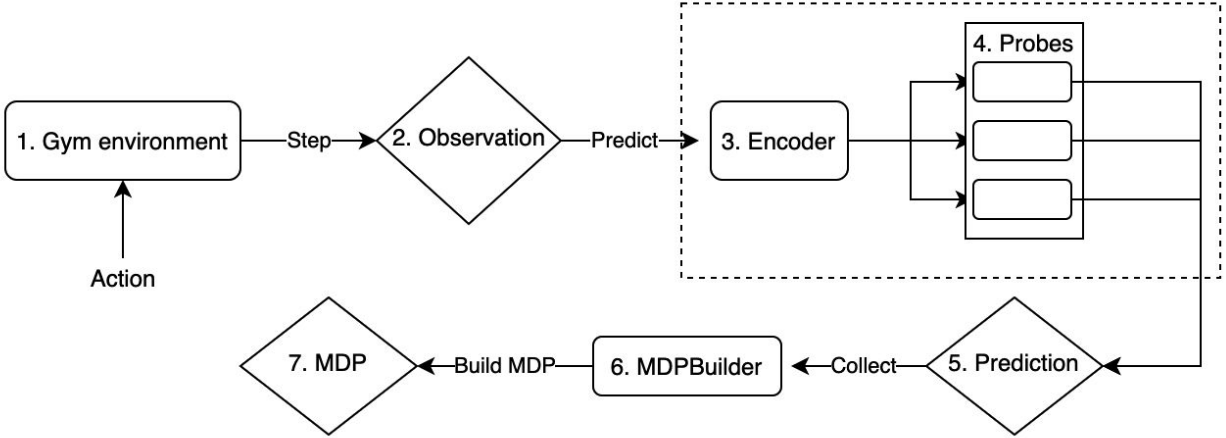
- Encoder/probe training
- Evaluation

Codebase is extended in this project for:

- Obtaining state label predictions
- Building an MDP

Code at: github.com/davidkerkkamp/representation-learning

Pipeline


1. Gym environment
Step
2. Observation
Predict
3. Encoder
4. Probes
Action
7. MDP
Build MDP
6. MDPBuilder
Collect
5. Prediction

Using the encoder

- Code contains class `AtariARIHandler`for:
    - Encoder training
    - Obtaining predictions
- Use handler to obtain Gym environment

```
args.env_name = 'PongNoFrameskip-v4'
handler = AtariARIHandler(args, wandb)
gym_env = handler.get_gym_env()
```

Example: create handler and obtain env for game Pong

Stepping through the Gym environment

- Apply **step** on environment using some **action**
- Each step returns 4 objects:
    - **obs**: 210 x 160 observation matrix representing game screen
    - **reward**
    - **done**: boolean indicating end of episode
    - **info**: dict with ground truth labels of observation

```
for i in range(n):
   obs, reward, done, info = gym_env.step(action)
```

Example: perform `n` steps on environment using some `action`

Obtaining state predictions

- Handler contains `predict` function
- Observation is put through encoder and probes
- Result is a dictionary with labels

```
prediction = handler.predict(obs)
```

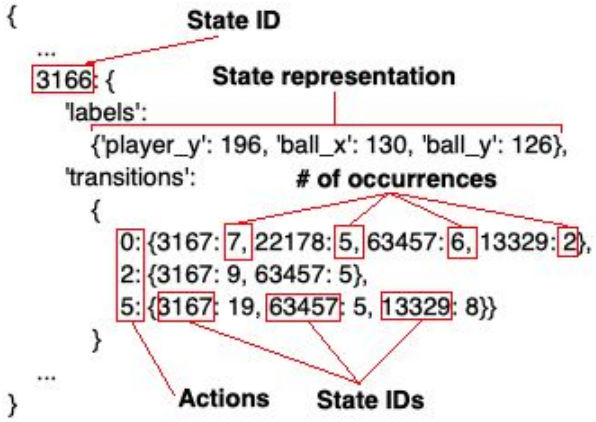
Example: obtain predicted state labels for observation

```
{
    'player_y': 108,
    'enemy_y': 196,
    'ball_x': 156,
    'ball_y': 55,
    'enemy_score': 5,
    'player_score': 0
}
```

Example of a dictionary with predicted labels

Building an MDP

- Class `MDPBuilder` for:
    - Collecting observations
    - Building a Markov Decision Process
- Keeps track of:
    - Distinct observations as states
    - Transitions for all actions
- Export MDP to PRISM file
    - Can be read by model checkers PRISM or Storm



Internal data structure of `MDPBuilder` class

Building an MDP

```
labels_to_use = ['ball_x', 'ball_y', 'player_y']
actions_to_use = [0, 2, 5]
mdp_builder = MDPBuilder(labels_to_use, actions_to_use)

for i in range(n):
    ….
    mdp_builder.add_state_info(prediction, action)
```

Example: create MDPBuilder instance and add observations

```
// Command describing transitions for some action
[] player_y=186 & ball_x=205 & ball_y=0 ->
  0.75 : (player_y'=186) & (ball_x'=205) & (ball_y'=0) +
  0.25 : (player_y'=195) & (ball_x'=205) & (ball_y'=0);

// Transitions from same state as above, different action
[] player_y=186 & ball_x=205 & ball_y=0 ->
  0.33 : (player_y'=186) & (ball_x'=205) & (ball_y'=0) +
  0.33 : (player_y'=187) & (ball_x'=205) & (ball_y'=0) +
  0.17 : (player_y'=181) & (ball_x'=205) & (ball_y'=0) +
  0.17 : (player_y'=190) & (ball_x'=205) & (ball_y'=0);

// Different state
[] player_y=187 & ball_x=205 & ball_y=0 ->
  0.33 : (player_y'=192) & (ball_x'=205) & (ball_y'=0) +
  0.33 : (player_y'=187) & (ball_x'=205) & (ball_y'=0) +
  0.33 : (player_y'=184) & (ball_x'=205) & (ball_y'=0);
```

Example of generated PRISM file

Pong

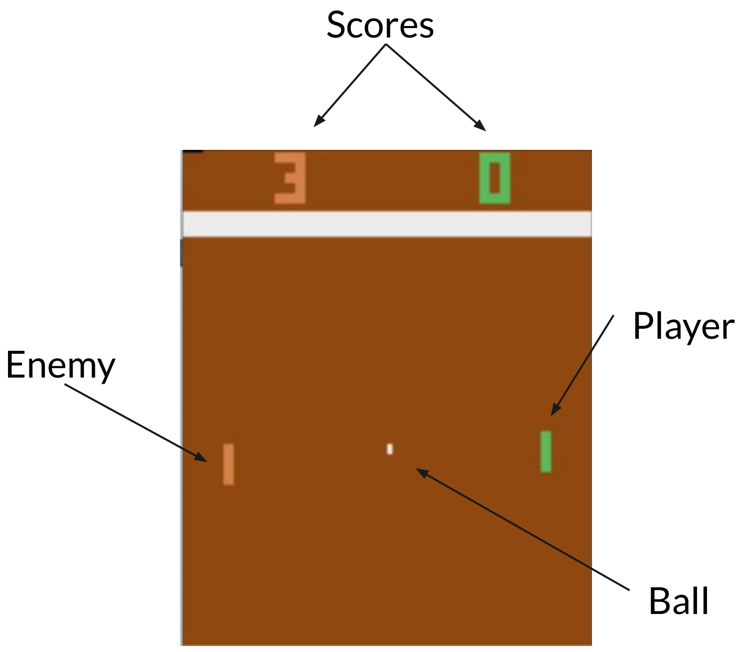
Action space: {no action, up, down}

Available labels:

- player_y, enemy_y
- ball_x, ball_y
- player_score, enemy_score
- player_x and enemy_x are **constants**

Undesired states:

- ball_x > player_x (enemy scores a point)



Pong game screen

Demo

Conclusion

Result:

- Pipeline that takes game observations and creates model of environment
- Model can be used to avoid dangerous situations for the player

Future:

- Other MDP model formats (e.g. explicit)
- Way of collecting observations (now random)
- Better ways to generate MDP (similar transitions, small probabilities, states with 1 transition)
- Applying model checking on generated models

Questions