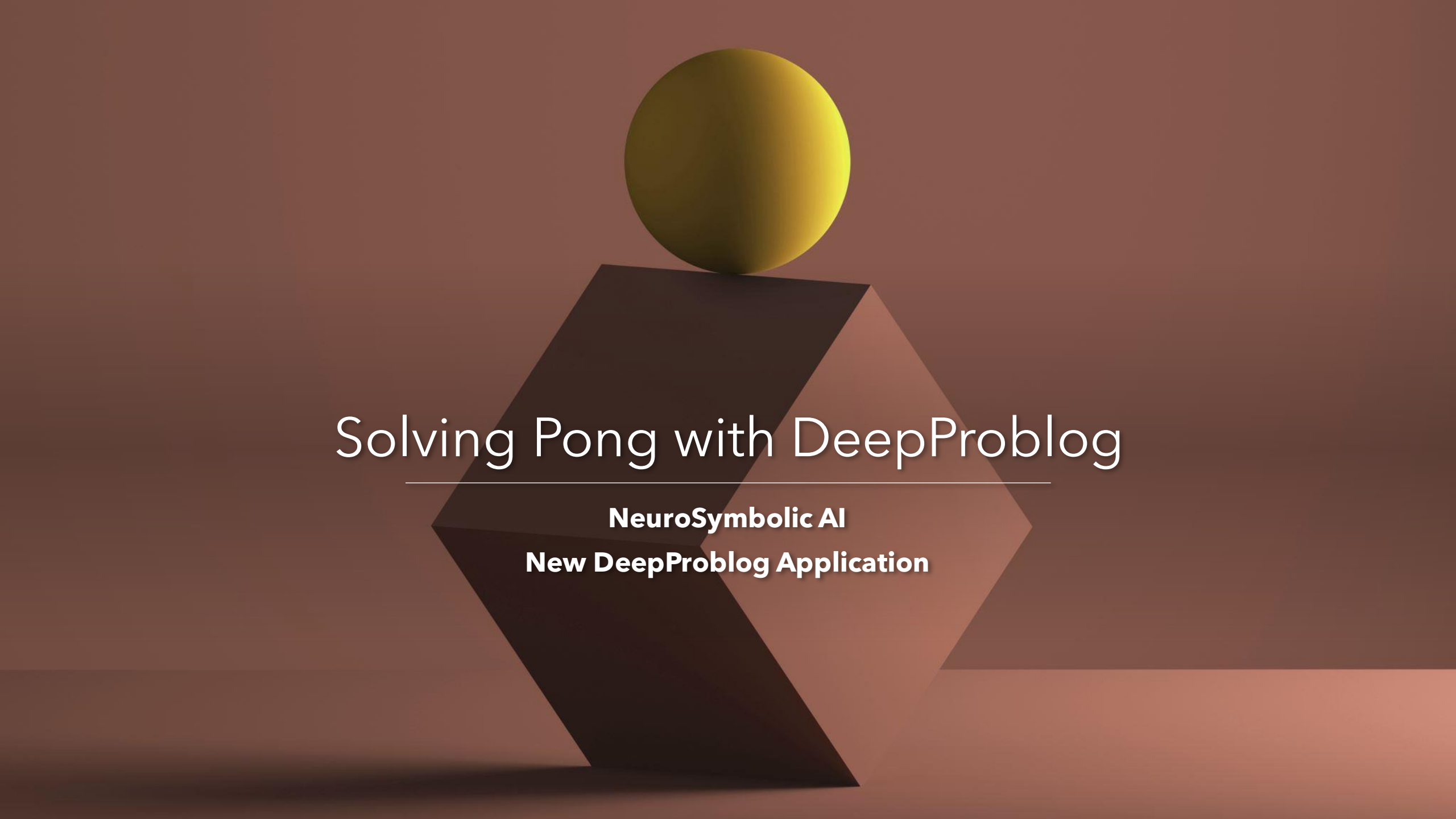


Solving Pong with DeepProblog

NeuroSymbolic AI

New DeepProblog Application

Content

Motivation

Experiments

Discussion

Questions

Motivation

Modelling of Complex Systems

- ProB : create models of increasing complexity

➢ How can we use DeepProblog to minimize the effort needed to adapt an agent to a harder version of the same problem?

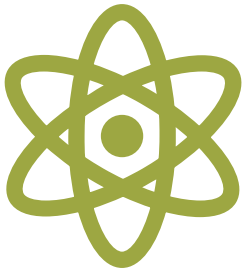
Pong

- Because games are fun…
- Pong is a simple game
- Difficulty increase comes from agent and ball speed variations

Experiments

1. DeepProblog vs PyTorch
2. Increasing agent complexity
3. Robustness to noise

DeepProblog vs PyTorch



Goal

Compare effectiveness and
data efficiency of each method

Measurements

Training Loss over iterations
Testing Accuracy over iterations
Multiple dataset sizes

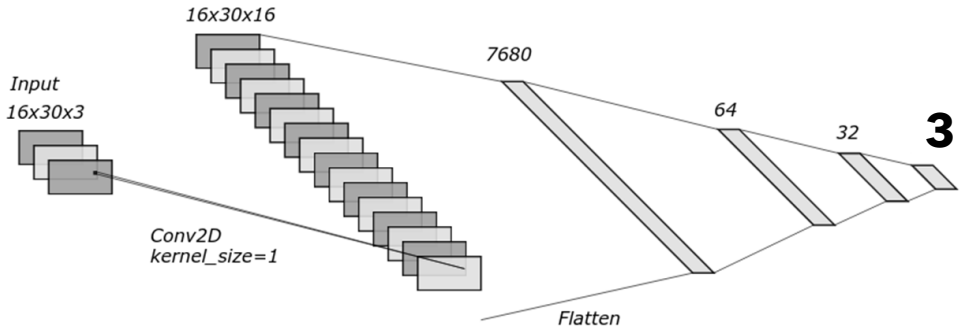
DeepProblog vs PyTorch

Model comparison

- PyTorch CNN

  frame => action

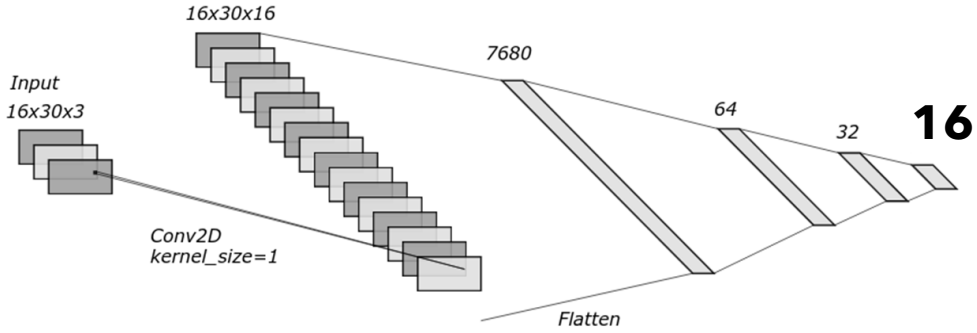
  16x30x3 => down, up, noop



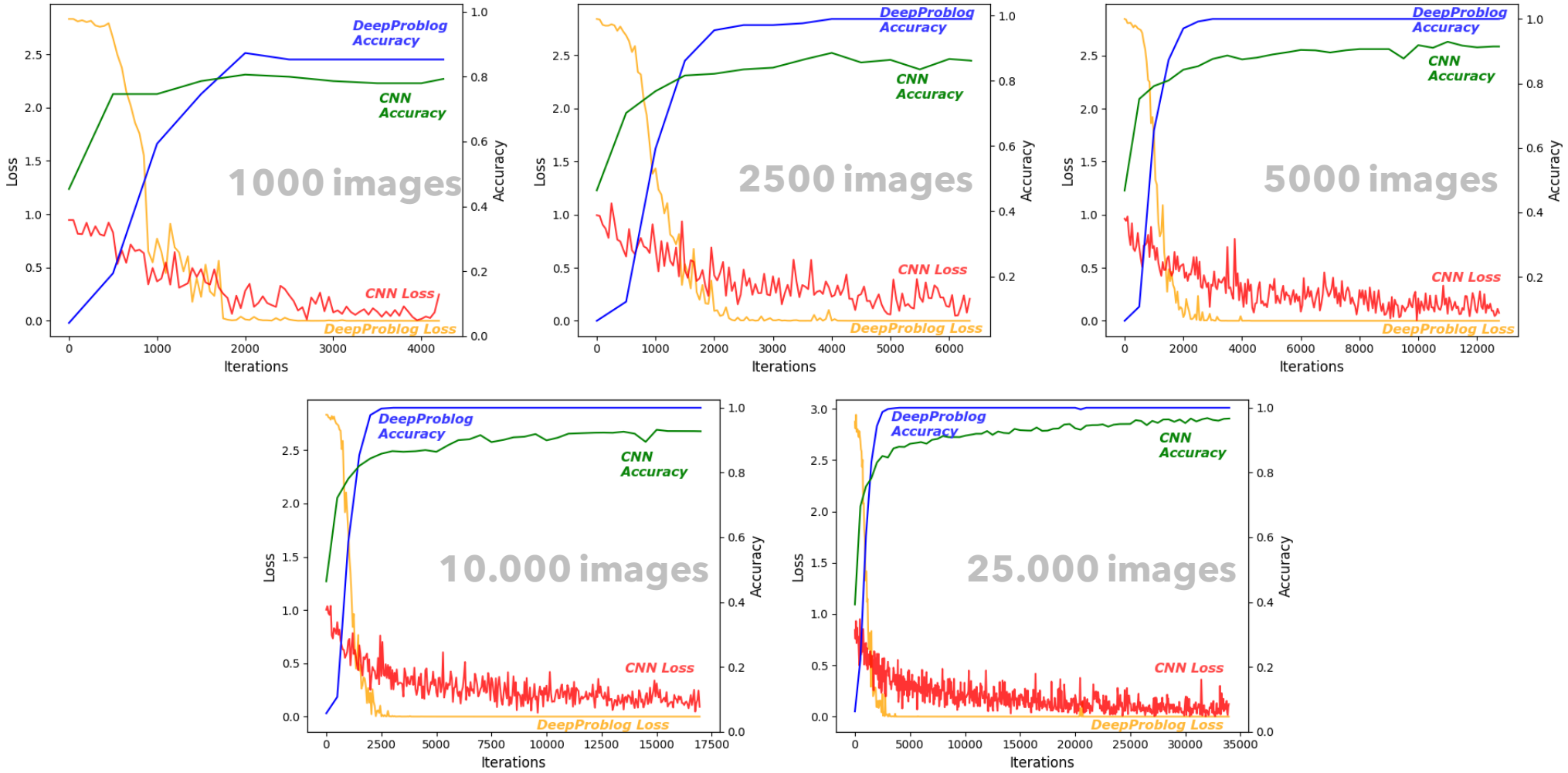
- DeepProblog

  frame => ball y-position

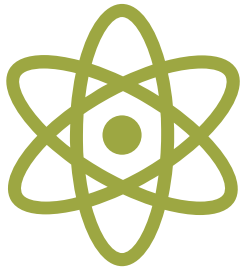
  16x30x3 => 0,1,…,14,15



DeepProblog vs PyTorch



Increasing Agent Complexity



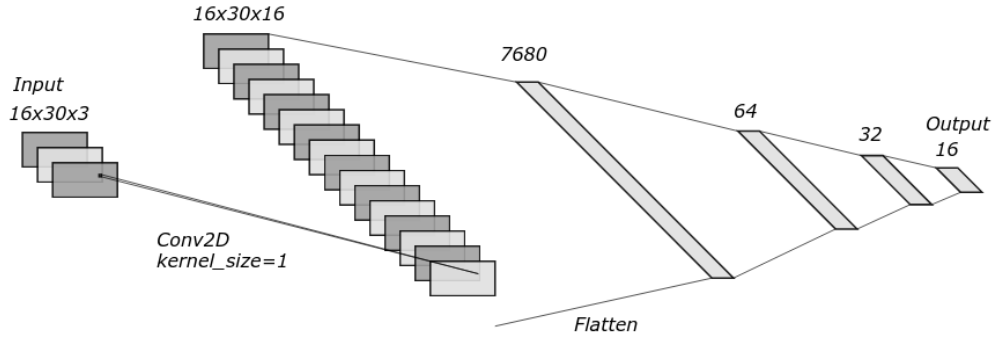
Goal

Creating a more complex agent
for harder game variations

Measurements

Win rate against random agent
@ varying ball and player speed

Agent v1



Follow ball y-position

- Perception: CNN
- Decision: FOL

```
nn(pong_net_y,[Img],BallY,[0,...,15]) :: y_coord(Img,BallY).

distance_y(Img, AIY, D):- y_coord(Img, BallY), D is AIY - BallY.

choose_action(Img, AIY, noop) :- distance_y(Img, AIY, D), D = 0.
choose_action(Img, AIY, up) :- distance_y(Img, AIY, D), D > 0.
choose_action(Img, AIY, down) :- distance_y(Img, AIY, D), D < 0.
```

Agent v1

Player = Ball = 1

- ➢ 100% winrate
- ❖ Fails at higher speeds

Player	Ball	win rate	Player	Ball	win rate
1	1	100%	1	2	46.6%
2	2	63.5%	1	3	48.5%
3	3	62.3%	2	3	47.2%
4	4	55.5%	2	4	56.4%
			3	4	51.6%

Agent v2

Predict ball y-position

- Perception + x-pos
- Decision: simulation

```
nn(pong_net_y,[Img],BallY,[0,...,15]) :: y_coord(Img,BallY).
nn(pong_net_x,[Img],BallX,[1,...,28]) :: x_coord(Img,BallX).

%used to train the pong_net_x network
distance_x(Img, D) :- x_coord(Img, X), D is 29 - X.
coordinates(Img, X, Y):- x_coord(Img,X), y_coord(Img, Y).

choose_action(Img0, Img1, AIY, Action):-
    coordinates(Img0, X0, Y0),
    coordinates(Img1, X1, Y1),
    choose_action(AIY, X0, Y0, X1, Y1, Action).

%going away from agent
choose_action(AIY, X0, _, X1, _, noop):-
    direction_x(BallPrevX, BallCurX, -1),
    AIY = 8.
choose_action(AIY, X0, _, X1, _, up):-
    direction_x(BallPrevX, BallCurX, -1),
    AIY > 8.
choose_action(AIY, X0, _, X1, _, down):-
    direction_x(BallPrevX, BallCurX, -1),
    AIY < 8.

%going towards agent
choose_action(AIY, X0, Y0, X1, Y1, noop):-
    direction_x(X0, X1, 1),
    intersect_y(X0, Y0, X1, Y1, YFuture),
    AIY = YFuture.
choose_action(AIY, X0, Y0, X1, Y1, up):-
    direction_x(X0, X1, 1),
    intersect_y(X0, Y0, X1, Y1, YFuture),
    AIY > YFuture.
choose_action(AIY, X0, Y0, X1, Y1, down):-
    direction_x(X0, X1, 1),
    intersect_y(X0, Y0, X1, Y1, YFuture),
    AIY < YFuture.

%to be safe: if x doesn't change (wrong perception)
% => assume towards agent
direction_x(X0, X1, 1):- PrevX =< CurX.
direction_x(X0, X1, -1):- PrevX > CurX.
```

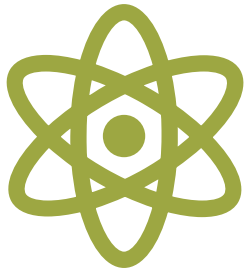
```
speed(Prev, Cur, Speed):-
    (Cur > Prev; Cur < Prev),
    Speed is Cur - Prev.

intersect_y(X0, Y0, X1, Y1, YFuture):-
    speed(X0, X1, SpeedX),
    speed(Y0, Y1, SpeedY),
    DistX is 29 - X1,
    FramesLeft is DistX / SpeedX,
    intersect_y_iter(SpeedY, FramesLeft, Y1, YFuture).

intersect_y_iter(_, FramesLeft, Y, Y):-
    FramesLeft =< 0.
intersect_y_iter(SpeedY, FramesLeft, TempY, Y):-
    FramesLeft > 0,
    NewFramesLeft is FramesLeft - 1,
    bounded_y_calc(TempY, SpeedY, NewTempY, NewSpeedY),
    intersect_y_iter(NewSpeedY, NewFramesLeft, NewTempY, Y).

%calculates the next y position and updates yspeed if there was a bounce
bounded_y_calc(CurY, SpeedY, NewY, NewSpeedY):-
    Steps is abs(SpeedY),
    DirY is sign(SpeedY),
    bounded_y_calc_iter(CurY, DirY, NewY, NewDirY, Steps),
    NewSpeedY is Steps * NewDirY.

bounded_y_calc_iter(CurY, DirY, CurY, DirY, 0).
bounded_y_calc_iter(CurY, DirY, FinalY, FinalDirY, Steps):-
    Steps > 0,
    (CurY = 0; CurY = 15), %if next step would hit top or bottom of screen
    NewDirY is -1*DirY,
    TargetY is CurY + NewDirY,
    NewSteps is Steps - 1,
    bounded_y_calc_iter(TargetY, NewDirY, FinalY, FinalDirY, NewSteps).
bounded_y_calc_iter(CurY, DirY, FinalY, FinalDirY, Steps):-
    Steps > 0,
    CurY > 0,
    CurY < 15,
    TargetY is CurY + DirY,
    NewSteps is Steps - 1,
    bounded_y_calc_iter(TargetY, DirY, FinalY, FinalDirY, NewSteps).
```

Agent v2

- ➢ Better performance at higher speeds
- ❖ Logic errors in speed calculation @ bounces

Player	Ball	win rate	Player	Ball	win rate
1	1	100%	1	3	95.9%
2	2	100%	1	4	73.7%
3	3	79.8%	2	3	79.8%
4	4	77.9%	2	4	69.7%
1	2	100%	3	4	65.6%

Noise Robustness



Goal

Particle effects ≈ noise

How much is too much?

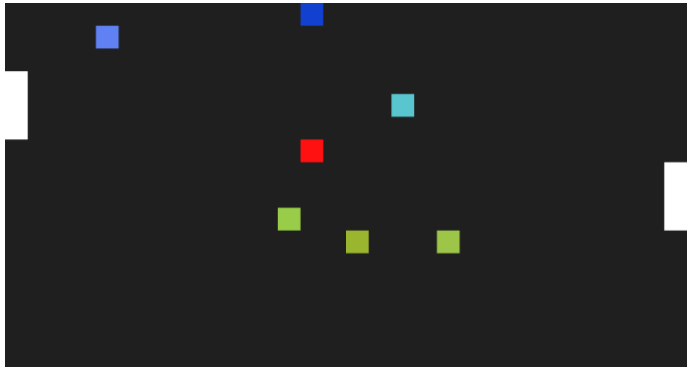
Measurements

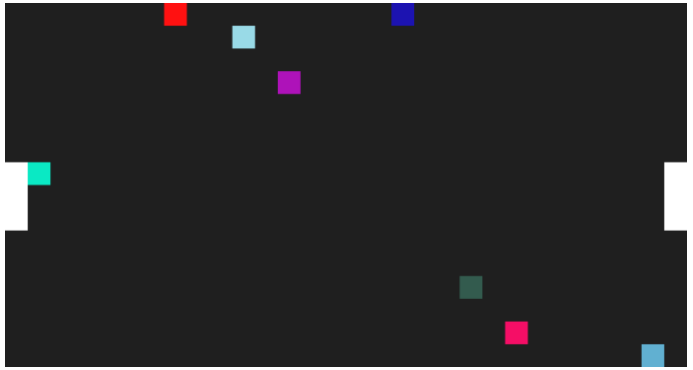
Win rate against random agent

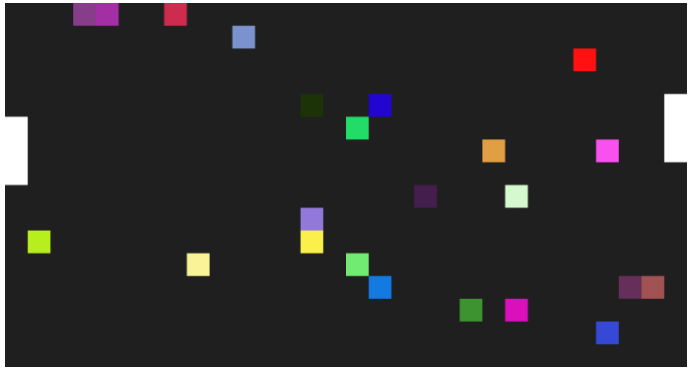
@ varying noise levels

Noise Robustness

- Agent v2
- Player Speed = 1
- Ball Speed = 2







Noise	win rate
0%	100%
1%	89.2%
2.5%	85.2%
5%	62.5%

Discussion

- Is DeepProblog data efficient and effective?
- Is DeepProblog useful to model levels of complexity?

Is DeepProblog data efficient and effective?

+ Less data is needed
+ Higher accuracy is reached
+ Accuracy is reached in less iterations
+ More intuitive outputs

- Training iterations are slower
- Comparison isn’t entirely fair

Is DeepProblog useful to model levels of complexity?

- Increasing perception complexity
  - \+ Know what each network does
  - \+ Reusable building blocks
  - \+ Easy training FOL feedback clauses
- Increasing decision making complexity
  - \+ Understandable AI
  - ± Feature Engineering
  - \- Prone to Human Error (edge cases)


Questions