```
from fastai.vision.all import *
```

MuZero

Lets define our alphabet soup: * h - calculates latent representation of state s from the past observation (board state, or previous frames) * s - state, latent representation of the environment * f - calculates p(policy) and v(value function) from s(state) * p - policy value for each action * v - value function, based on the reward. For atari n-step reward, final reward for board games. * a - some action, sampled from /p when interacting with the environment, sampled from replay buffer during training. * g - calculates next s(state) and immediate reward(r), recieves previous state and an action as input * r - immediate reward * - policy, approximately p

```
[0.1,0.5,0.6,0.7,...]
```

muzero can learn rules of game, doesn't need to be provided with rules.

```
up,down,left,right
1=(0.2,0.1,0.1,0.7)
```

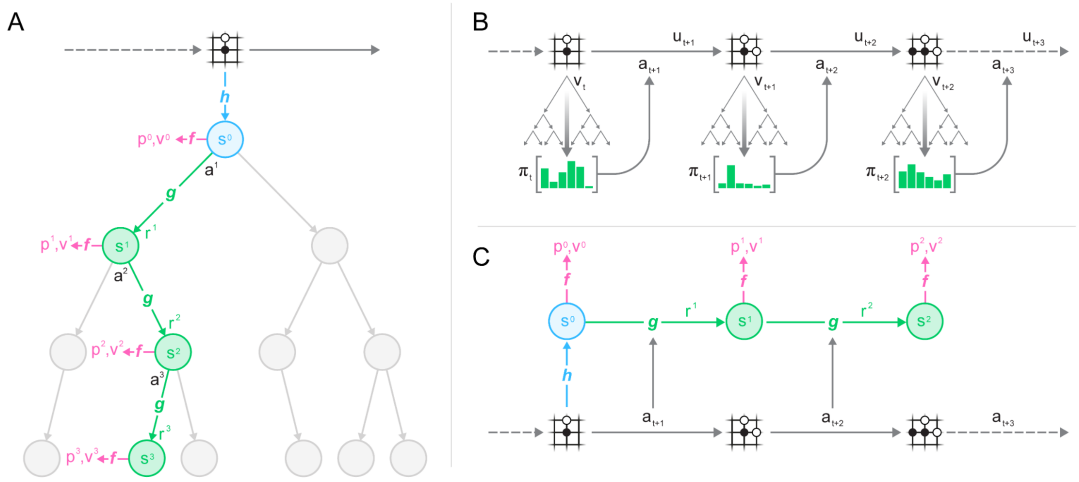


Figure 1: mu.png

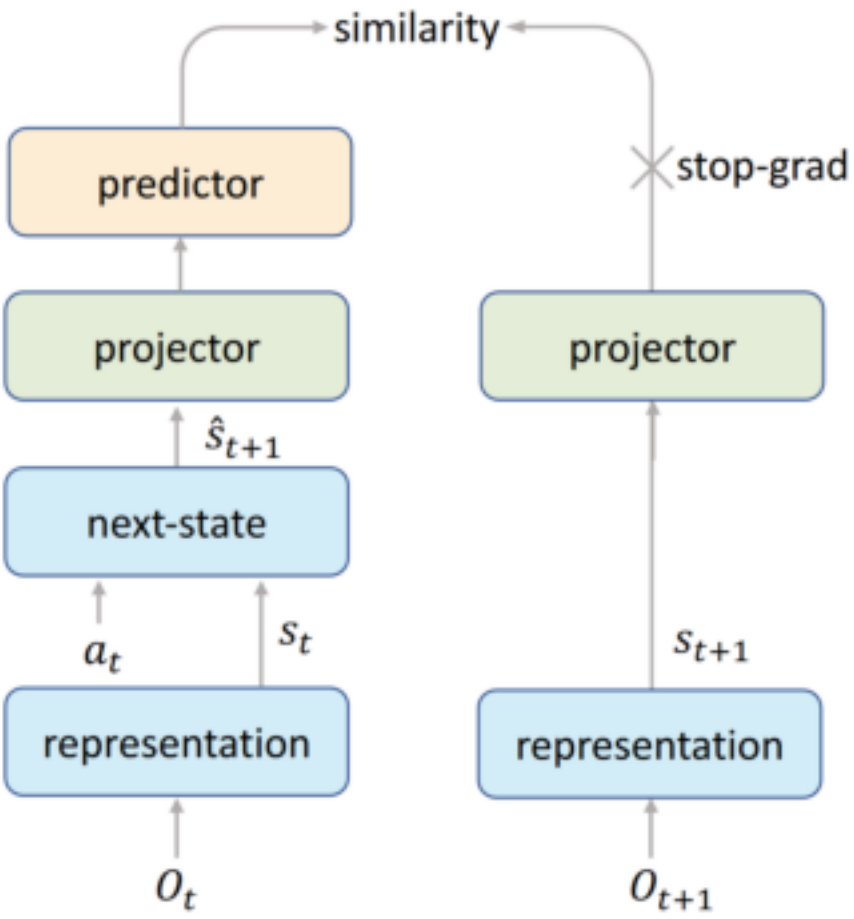
The models are trained to predict p(policy),v(value function),r(immediate reward) from the REPLAY buffer.

EfficientZero

Improvement 1

```
show_image(Image.open('efficient_similarity.png'),figsize=[7,7])
```

```
<AxesSubplot:>
```



The general idea for this one is from this paper: https://arxiv.org/abs/2011.10566

1. The "representation" is generated using h from before.

2. g then creates s_(t+1) using s_t and a_t for “next state”
3. The “projector” and “predictor” seem to be both thrown away. The projector mapping s_t+1 to a lower deminsional space. This seems to be because we want the predictor to have very few parameters. (lower deminsional space just has less numbers)
4. The “predictor” seems to “bridge” the gap between both branches and allows for smaller batch size training and more stable training. “Exploring Simple Siamese Representation Learning” suggests that the predictor should model the Expected value of what was before the projector, including the difference between O_t and O_(t+1)
5. Since the projector and representation is on both sides, and the predictor is sufficiently small, The hidden states for the two s_(t+1) must be similar.

Improvement 2

Predicting when a player will lose a point at a particular timestep is hard. Though it is much easier to predict “if” a player will lose a point.

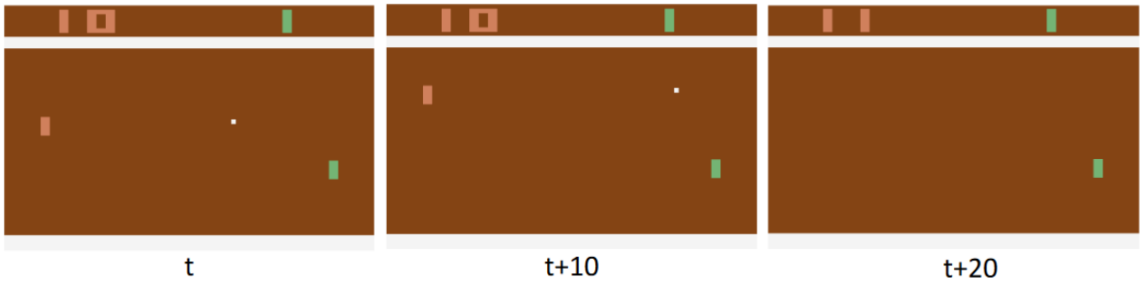


Figure 2: efficient_future.png

Predicting the immediate reward(r) is hard, how do we know 20 steps ahead of time, exactly which time step we will get the reward? Instead they use a LSTM to predict the total reward up until the current time, and train to predict that value instead.

Improvement 3

To use or not to use the replay buffer? - Green, MuZero uses the replayer buffer as is - Yellow, on more recent examples, Efficient Zero will use its policy to predict 1 state for training. - Blue, on less recent examples, Efficient Zero will us its policy for part of the future states. - Red, Efficient Zero can dream up the majority of the policy if observations in the replay buffer are very old.

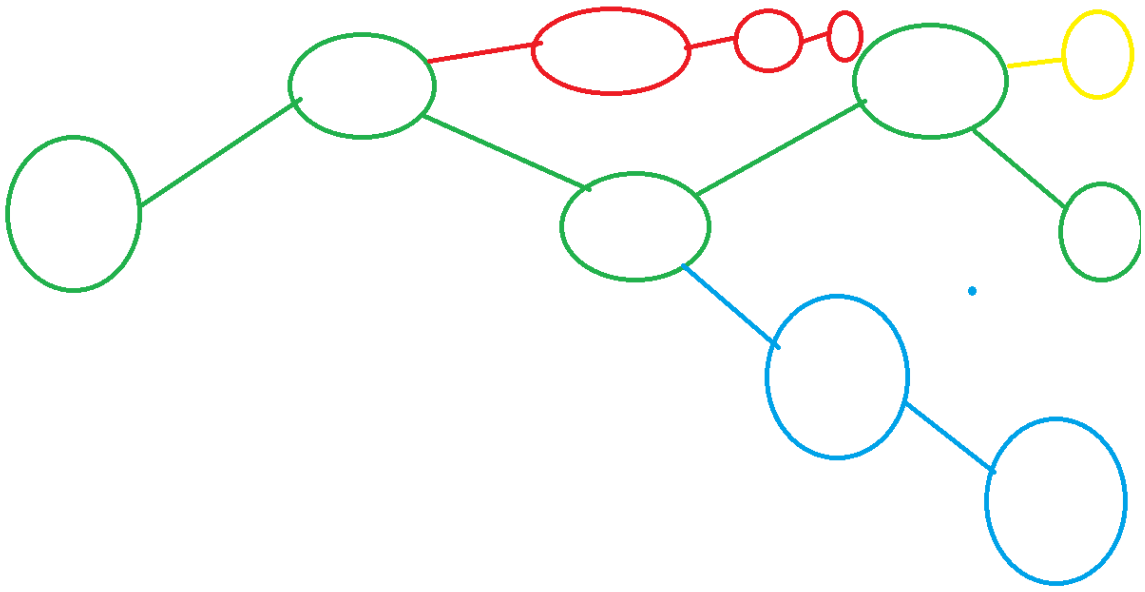


Figure 3: efficient_replay.png

Results

Results where inconsistent across different applications, so it may be better to think of this as a handful of techniques as opposed to something to always use together. More abalations in the other environments should help to determine what techniques generalize outside of atari and board games.

References

- Mu zero
- Efficient Zero
- Simple Siamese Representations
- Yanic Muzero
- Yanic Efficient Zero
- Images are from their respective papers.

Game	Full	w.o. consistency	w.o. value prefix	w.o. off-policy correction
Alien	808.5	**961.3**	558	619.4
Amidar	148.6	32.2	31.0	**256.3**
Assault	**1263.1**	572.9	955.0	1190.4
Asterix	**25557.8**	2065.6	7330.0	13525.0
Bank Heist	**351.0**	165.6	273.0	297.5
BattleZone	13871.2	14063.0	9900.0	**16125.0**
Boxing	52.7	6.1	**60.2**	30.5
Breakout	**414.1**	237.4	379.2	400.3
ChopperCommand	1117.3	1138.0	1280	**1487.5**
Crazy Climber	83940.2	75550.0	**106090.0**	70681.0
Demon Attack	**13003.9**	5973.8	6818.5	8640.6
Freeway	**21.8**	21.8	21.8	21.8
Frostbite	**296.3**	248.8	235.2	227.5
Gopher	**3260.3**	1155	2792.0	2275.0
Hero	**9315.9**	5824.4	3167.5	9053.0
Jamesbond	**517.0**	154.7	380.0	356.3
Kangaroo	**724.1**	375.0	200.0	687.5
Krull	**5663.3**	4178.625	4527.6	3635.6
Kung Fu Master	**30944.8**	19312.5	25980.0	25025.0
Ms Pacman	1281.2	1090.0	**1475.0**	1297.2
Pong	**20.1**	-1.5	16.8	19.5
Private Eye	96.7	**100.0**	100.0	100.0
Qbert	**13781.9**	5340.7	6360.0	13637.5
Road Runner	**17751.3**	2700.0	3010.0	9856.0
Seaquest	**1100.2**	460.0	468.0	843.8
Up N Down	**17264.2**	3040.0	7656.0	4897.2
Normed Mean	**1.943**	0.881	1.482	1.475
Normed Median	**1.090**	0.340	0.552	0.836

Figure 4: efficient_inconsistent.png