Model Based Reinforcement Learning for Atari

Łukasz Kaiser [*1] **Mohammad Babaeizadeh** [*2 3] **Piotr Miłoś** [*4 5] **Błażej Osiński** [*4 5 3] **Roy H Campbell** [2] **Konrad Czechowski** [4] **Dumitru Erhan** [1] **Chelsea Finn** [1] **Piotr Kozakowski** [4] **Sergey Levine** [1] **Ryan Sepassi** [1] **George Tucker** [1] **Henryk Michalewski** [4 5]

Abstract

Model-free reinforcement learning (RL) can be used to learn effective policies for complex tasks, such as Atari games, even from image observations. However, this typically requires very large amounts of interaction – substantially more, in fact, than a human would need to learn the same games. How can people learn so quickly? Part of the answer may be that people can learn how the game works and predict which actions will lead to desirable outcomes. In this paper, we explore how video prediction models can similarly enable agents to solve Atari games with orders of magnitude fewer interactions than model-free methods. We describe Simulated Policy Learning (SimPLe), a complete model-based deep RL algorithm based on video prediction models and present a comparison of several model architectures, including a novel architecture that yields the best results in our setting. Our experiments evaluate SimPLe on a range of Atari games and achieve competitive results with only 100K interactions between the agent and the environment (400K frames), which corresponds to about two hours of real-time play.

1. Introduction

Human players can learn to play Atari games in minutes (Tsividis et al., 2017). However, our best model-free reinforcement learning algorithms require tens or hundreds of millions of time steps – the equivalent of several weeks of training in real time. How is it that humans can learn these games so much faster? Perhaps part of the puzzle is that humans possess an intuitive understanding of the physical processes that are represented in the game: we know that planes can fly, balls can roll, and bullets can destroy aliens. We can therefore predict the outcomes of our actions. In this paper, we explore how learned video models can enable learning in the Atari Learning Environment (ALE) benchmark (Bellemare et al., 2015; Machado et al., 2017) with a budget restricted to 100K time steps – roughly to two hours of a play time.

Although prior works have proposed training predictive models for next-frame, future-frame, as well as combined future-frame and reward predictions in Atari games (Oh et al., 2015; Chiappa et al., 2017; Leibfried et al., 2016), no prior work has successfully demonstrated model-based control via such predictive models that achieve results that are competitive with model-free RL. Indeed, in a recent survey by Machado et al. this was formulated as the following challenge: "*So far, there has been no clear demonstration of successful planning with a learned model in the ALE*" (Section 7.2 in Machado et al. (2017)).

Using models of environments, or informally giving the agent ability to predict its future, has a fundamental appeal for reinforcement learning. The spectrum of possible applications is vast, including learning policies from the model (Watter et al., 2015; Finn et al., 2016; Finn & Levine, 2016; Ebert et al., 2017; Hafner et al., 2018; Piergiovanni et al., 2018; Rybkin et al., 2018; Sutton & Barto, 2017, Chapter 8), capturing important details of the scene (Ha & Schmidhuber, 2018), encouraging exploration (Oh et al., 2015), creating intrinsic motivation (Schmidhuber, 2010) or counterfactual reasoning (Buesing et al., 2018). One of the exciting benefits of model-based learning is the promise to substantially improve sample efficiency of deep reinforcement learning (see Chapter 8 in (Sutton & Barto, 2017)).

Our work advances the state-of-the-art in model-based reinforcement learning by introducing a system that, to our knowledge, is the first to successfully handle a variety of challenging games in the ALE benchmark. To that end, we experiment with several stochastic video prediction techniques, including a novel model based on discrete latent variables. We also present an approach, called Simulated Policy Learning (SimPLe), that utilizes these video

[*]Equal contribution, authors listed in random order. [1]Google Brain, Mountain View, CA, USA [2]University of Illinois at Urbana–Champaign, Urbana-Champaign, IL, USA [3]Work partially performed while an intern at Google Brain [4]Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Warsaw, Poland [5]deepsense.ai, Warsaw, Poland. Correspondence to: Błażej Osiński <b.osinski@mimuw.edu.pl>.

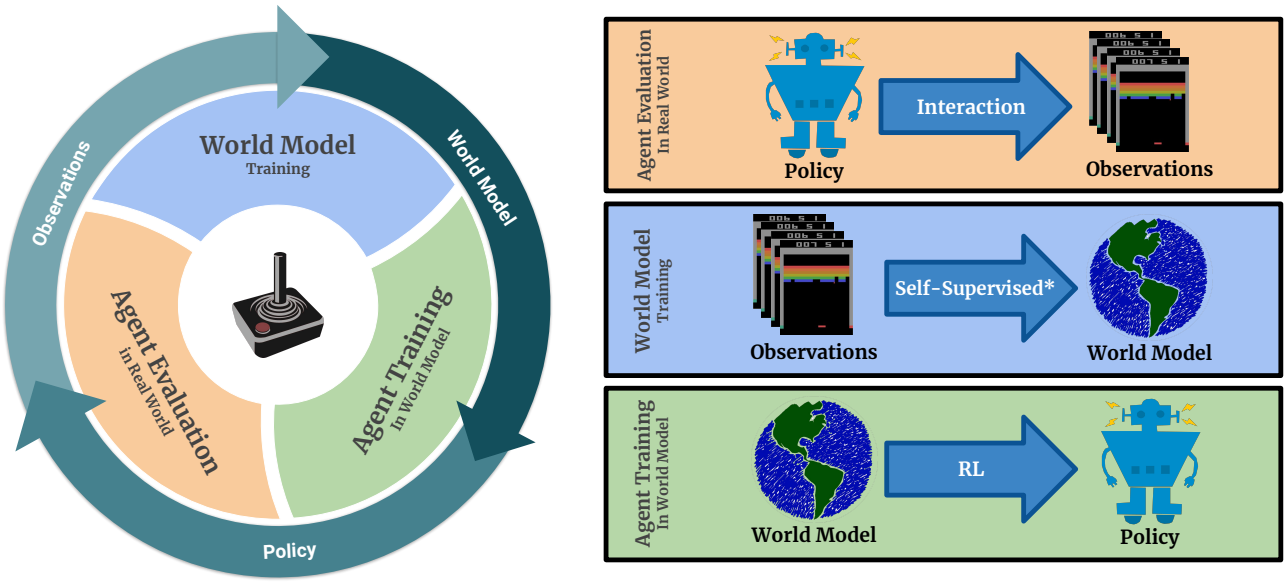

Figure 1: Main loop of SimPLe. 1) the agent starts interacting with the real environment following the latest policy (initialized to random). 2) the collected observations will be used to train (update) the current world model. 3) the agent updates the policy by acting inside the world model. The new policy will be evaluated to measure the performance of the agent as well as collecting more data (back to 1). Note that world model training is self-supervised for the observed states and supervised for the reward.

prediction techniques and can train a policy to play the game within the learned model. With several iterations of dataset aggregation, where the policy is deployed to collect more data in the original game, we can learn a policy that, for many games, can successfully play the game in the real environment (see videos on the project webpage `https://goo.gl/itykP8`).

In our empirical evaluation, we find that SimPLe is significantly more sample-efficient than a highly tuned version of the state-of-the-art Rainbow algorithm (Hessel et al., 2017) on almost all games. In particular, in low data regime of 100k samples, on more than half of the games, our method achieves a score which Rainbow requires at least twice as many samples. In the best case of `Freeway`, our method is more than 10x more sample-efficient, see Figure 3.

2. Related Work

Atari games gained prominence as a popular benchmark for reinforcement learning algorithms with the introduction of the Arcade Learning Environment (ALE) (Bellemare et al., 2015). The combination of reinforcement learning and deep models then enabled RL algorithms to learn to play Atari games directly from images of the game screen, using variants of the DQN algorithm (Mnih et al., 2013; 2015; Hessel et al., 2017) and actor-critic algorithms (Mnih et al., 2016; Schulman et al., 2017; Babaeizadeh et al., 2016; Wu et al., 2017; Espeholt et al., 2018). The most successful methods in this domain remain model-free algorithms (Hessel et al., 2017; Espeholt et al., 2018). Although the sample complexity of these methods has substantially improved in recent years, it remains far higher than the amount of experience required for human players to learn each game (Tsividis et al., 2017). In this work, we aim to learn Atari games with a budget of just 100K agent steps (400K frames), corresponding to about two hours of play time. Prior methods are generally not evaluated in this regime, and we therefore re-optimized Rainbow (Hessel et al., 2017) for optimal performance on 1M steps.

Oh et al. (2015) and Chiappa et al. (2017) show that learning predictive models of Atari 2600 environments is possible using appropriately chosen deep learning architectures. Impressively, in some cases the predictions maintain low L_2 error over timespans of hundreds of steps. As learned simulators of Atari environments are core ingredients of our approach, in many aspects our work is motivated by Oh et al. (2015) and Chiappa et al. (2017), however we focus on using video prediction in the context of learning how to play the game well and positively verify that learned simulators can be used to train a policy useful in original environments. An important step in this direction was made by Leibfried et al. (2016), which extends the work of Oh et al. (2015) by including reward prediction, but does not use the model to learn policies that play the games. Perhaps surprisingly, there is virtually no work on model-based RL in video games from images. Notable exceptions are the works of Oh et al. (2017) and Ha & Schmidhuber (2018). Oh et al. (2017) use a model of rewards to augment model-

free learning with good results on a number of Atari games. However, this method does not actually aim to model or predict future frames, and achieves clear but relatively modest gains in efficiency. Ha & Schmidhuber (2018) present a way to compose a variational autoencoder with a recurrent neural network into an architecture that is successfully evaluated in the VizDoom environment and on a 2D racing game. The training procedure is similar to Algorithm 1, but only one iteration of the loop is needed as the environments are simple enough to be fully explored with random exploration.

Outside of games, model-based reinforcement learning has been investigated at length for applications such as robotics (Deisenroth et al., 2013). Though most of such works do not use image observations, several recent works have incorporated images into real-world (Finn et al., 2016; Finn & Levine, 2016; Babaeizadeh et al., 2017; Ebert et al., 2017; Piergiovanni et al., 2018; Paxton et al., 2018; Rybkin et al., 2018; Ebert et al., 2018) and simulated (Watter et al., 2015; Hafner et al., 2018) robotic control. Our video models of Atari environments described in Section 4 are motivated by models developed in the context of robotics. Another source of inspiration are discrete autoencoders proposed by van den Oord et al. (2017) and Kaiser & Bengio (2018).

The structure of the model-based RL algorithm that we employ consists of alternating between learning a model, and then using this model to optimize a policy by using model-free reinforcement learning within the model. Variants of this basic algorithm have been proposed in a number of prior works, starting from Dyna (Sutton, 1991) to more recent methods that incorporate deep networks for models and policies (Heess et al., 2015; Feinberg et al., 2018; Kalweit & Boedecker, 2017; Kurutach et al., 2018).

3. Simulated Policy Learning (SimPLe)

Reinforcement learning is formalized in Markov decision processes (MDP). An MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ is a state space, $\mathcal{A}$ is a set of actions available to an agent, P is the unknown transition kernel, r is the reward function and $\gamma \in (0, 1)$ is the discount factor. In this work we refer to MDPs as environments and assume that environments do not provide direct access to the state (i.e., the RAM of Atari 2600 emulator). Instead we use visual observations, typically 210×160 RGB images. A single image does not determine the state. To circumvent this, we stack the four previous frames, using the result as the state. A reinforcement learning agent interacts with the MDP by issuing actions according to a policy. Formally, policy π is a mapping from states to probability distributions over $\mathcal{A}$. The quality of a policy is measured by the value function $\mathbb{E}_\pi\left(\sum_{t=0}^{+\infty} \gamma^t r_{t+1} | s_0 = s\right)$, which for a starting state s estimates the total discounted reward gathered by the agent. In Atari 2600 games our goal is to find a policy which maximizes the value function from the beginning of the game. Crucially, apart from an Atari 2600 emulator environment env we will use *a neural network simulated environment* env' which we call a *world model* and describe in detail in Section 4. The environment env' shares the action space and reward space with env and produces visual observations in the same format, as it will be trained to mimic env. Our principal aim is to train a policy π using a simulated environment env' so that π achieves good performance in the original environment env. In this training process we aim to use as few interactions with env as possible. The initial data to train env' comes from random rollouts of env. As this is unlikely to capture all aspects of the environment, we use the data-aggregation iterative method presented in Algorithm 1.

Algorithm 1: Pseudocode for SimPLe

```
Initialize policy π
Initialize model parameters env'
Initialize empty set D
while not done do
  ▷ collect observations from real env.
  while not enough observations do
    a ← π(s)
    (s', r) ← env(a)
    D ← D ∪ (s, a, r, s')
    s ← s'
  end while
  ▷ update model using collected data.
  θ ← TRAIN_SUPERVISED(env', D)
  ▷ update policy using world model.
  π ← TRAIN_RL(π, θ)
end while
```

4. World Models

A crucial decision in the design of world models is the inclusion of stochasticity. Although Atari is known to be a deterministic environment, it is stochastic given only a limited horizon of past observed frames (in our case 4 frames). The level of stochasticity is game dependent; however, it can be observed in many Atari games. An example of such behavior is the *pause* after a player scores in `Pong`. These pauses are longer than 4 frames, so a model looking at only the past 4 frames does not know when a new round of the game should start and may keep predicting paused frames.

In search for an effective world model we have experimented with various architectures, both new and modified versions of existing ones. In this section, we describe the architectures and the rationale behind our design decisions. In Section 7 we compare the performance of these models.

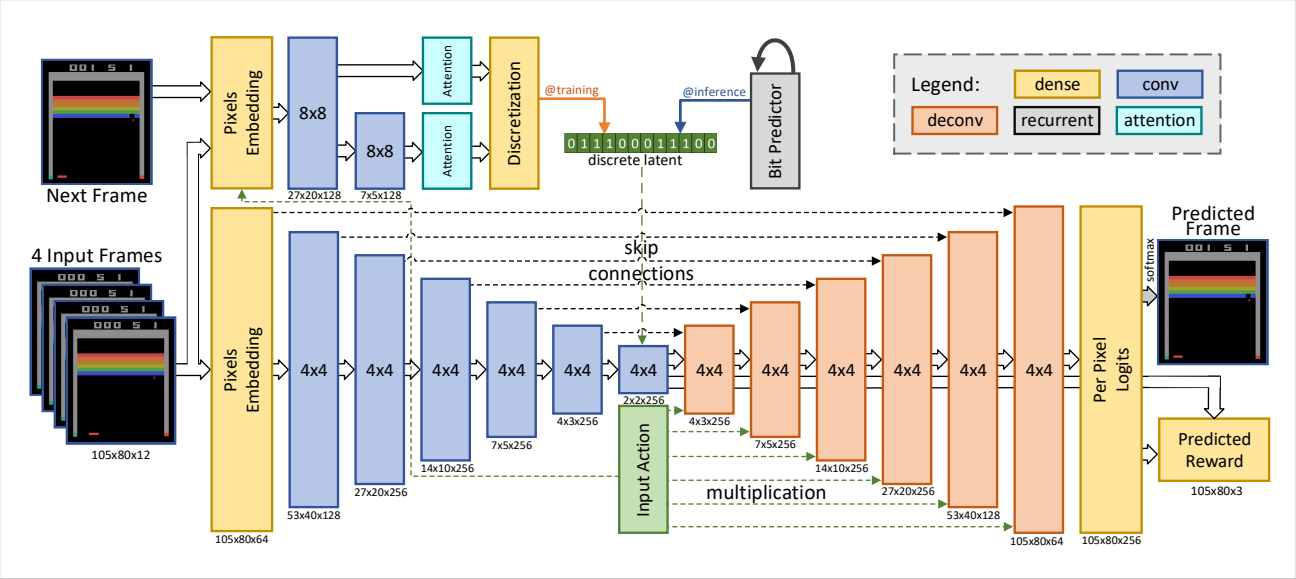

Figure 2: Architecture of the proposed stochastic model with discrete latent. The input to the model is four stacked frames (as well as the action selected by the agent) while the output is the next predicted frame and expected reward. Input pixels and action are embedded using fully connected layers, and there is per-pixel softmax (256 colors) in the output. This model has two main components. First, the bottom part of the network which consists of a skip-connected convolutional encoder and decoder. To condition the output on the actions of the agent, the output of each layer in the decoder is multiplied with the (learned) embedded action. Second part of the model is a convolutional inference network which approximates the posterior given the next frame similar to Babaeizadeh et al. (2017). At training time, the sampled latent values from the approximated posterior will be discretized into bits. To keep the model differentiable, the backpropagation bypasses the discretization following Kaiser & Bengio (2018). A third LSTM based network is trained to approximate each bit given the previous ones. At inference time, the latent bits are predicted auto-regressively using this network. The deterministic model has the same architecture as this figure but without the inference network.

4.1. Deterministic Model

Our basic architecture, presented as part of Figure 2, resembles the convolutional feedforward network from Oh et al. (2015). The input X consists of four consecutive game frames and an action a. Stacked convolution layers process the visual input. The actions are one-hot-encoded and embedded in a vector which is multiplied channel-wise with the output of the convolutional layers. The network outputs the next frame of the game and the value of the reward.

In our experiments, we varied details of the architecture above. In most cases, we use a stack of four convolutional layers with 64 filters followed by three dense layers (the first two have 1024 neurons). The dense layers are concatenated with 64 dimensional vector with a learnable action embedding. Next, three deconvolutional layers of 64 filters follow. An additional deconvolutional layer outputs an image of the original 105×80 size. The number of filters is either 3 or 3×256. In the first case, the output is a real-valued approximation of pixel's RGB value. In the second case, filters are followed by softmax producing a probability distribution on the color space. The reward is predicted by a softmax attached to the last fully connected layer. We used dropout equal to 0.2 and layer normalization.

Loss functions. The visual output of our networks is either one float per pixel/channel or the categorical 256-dimensional softmax. In both cases, we used the *clipped loss* $\max(Loss, C)$ for a constant C. We found that clipping was crucial for improving the prediction power of the models (both measured with the correct reward predictions per sequence metric and successful training using Algorithm 1). We conjecture that the clipping substantially decreases the magnitude of gradients stemming from fine-tuning of big areas of background consequently letting the optimization process concentrate on small but important areas (e.g. the ball in Pong). In our experiments, we set $C = 10$ for L_2 loss on pixel values and to $C = 0.03$ for softmax loss. Note that this means that when the level of confidence about the correct pixel value exceeds 97% (as $-\ln(0.97) \approx 0.03$) we get no gradients from that pixel any longer.

Scheduled sampling. The simulator env' consumes its own predictions from previous steps. Thus, due to compounding errors, the model may drift out of the area of its applicability. Following (Bengio et al., 2015), we mitigate this problem by randomly replacing in training some frames of the input X by the prediction from the previous step. Typically, we linearly increase the mixing probability during training arriving at 100%.

4.2. Stochastic Models

A stochastic model can be used to deal with limited horizon of past observed frames as well as sprites occlusion and flickering which results to higher quality predictions.

Inspired by Babaeizadeh et al. (2017), we used a variational autoencoder (Kingma & Welling, 2013) to model the stochasticity of the environment. In this model, an additional network receives the input frames as well as the future target frame as input and approximates the distribution of the posterior. At each timestep, a latent value z_t is sampled from this distribution and will be passed as input to the original predictive model. At test time, the latent values are sampled from an assumed prior $\mathcal{N}(\mathbf{0}, \mathbf{I})$. To match the assumed prior and the approximate, we use the Kullback–Leibler divergence term as an additional loss term (Babaeizadeh et al., 2017).

Nonetheless, we noticed two major issues with this model. First, the weight of KL divergence loss term is game dependent and has to be optimised as a hyper-parameter which is not practical if one wants to deal with a broad portfolio of Atari games. Second, this weight is usually a very small number in the range of $[e^{-5}, e^{-3}]$ which means that the approximated posterior can diverge significantly from the assumed prior. This can result in previously unseen latent values at inference time that lead to poor predictions. We address these issues by utilizing a discrete latent variable similar to Kaiser & Bengio (2018).

As visualized in Figure 2, the proposed stochastic model with discrete latent variables discretizes the latent values into bits (zeros and ones) while training an auxiliary LSTM-based (Hochreiter & Schmidhuber, 1997) recurrent network to predict these bits autoregressively. At inference time, the latent bits will be generated by this auxiliary network in contrast to sampling from a prior. To make the predictive model more robust to unseen latent bits, we add uniform noise to approximated latent values before discretization and apply dropout (Srivastava et al., 2014) on bits after discretization.

5. Policy Training

We will now describe the details of SimPLe, outlined in Algorithm 1. In step 6 we use the proximal policy optimization (PPO) algorithm (Schulman et al., 2017) with $\gamma = 0.95$. The algorithm generates rollouts in the simulated environment env' and uses them to improve policy π. The fundamental difficulty lays in imperfections of the model compounding over time. To mitigate this problem we use short rollouts of env'. Typically every $N = 50$ steps we uniformly sample the starting state from the ground-truth buffer D and restart env', see Section 7.2, paragraph **Steps** and paragraph **Random starts** for experimental evaluations.

Using short rollouts may have a degrading effect as the PPO algorithm does not have a way to infer effects longer than the rollout length. To ease this problem, in the last step of a rollout we add to the reward the evaluation of the value function. Note that the value function is learned along with the policy as it serves to calculate the advantage function.

We stress that imperfections in the model are unavoidable, as the model is trained using only data generated with the current policy. Moreover, the training of the world model is prone to overfitting as we try to gather as little data as possible from the real environment. Finally, sometimes the model fails in a catastrophic way by losing semantic coherence (e.g., a disappearing ball in Pong).

The main loop in Algorithm 1 is iterated 15 times. In the first iteration the world model is trained for 45K steps, in each of the following passes the model is trained for 15K steps. Shorter training in later passes does not degrade the performance, because the world model after first iteration captures already part of the game dynamics and only needs to be extended to novel situations.

In the first iteration of the main loop of Algorithm 1 the world model is trained for 45K steps, in each of the following passes the model is trained for 15K steps, we used batches of size 2. In total the main loop in Algorithm 1 is iterated 15 times, see Figure 11 and Section 7.2, paragraph **Model-based iterations**. Shorter training in later passes does not degrade the performance, because the world model after the first iteration captures already part of the game dynamics and only needs to be extended to novel situations.

In each of the iterations, the agent is trained inside the latest world model using PPO. In every PPO epoch we used 16 parallel agents collecting 25,50 or 100 steps from the simulated environment env', see Section 7.2, paragraph **Steps** for ablations with regard to number of steps. The number of PPO epochs is $z \cdot 1000$, where z equals to 1 in all passes except last one (where $z = 3$) and two passes number 8 and 12 (where $z = 2$). This give 800K$\cdot z$ interactions with the simulated environment in each of the loop passes. In the process of training the agent performs 15.2M interactions with the simulated environment env'.

6. Experiments

We evaluate SimPLe on a suite of Atari games from Atari Learning Environment (ALE) benchmark. In our experiments, the training loop is repeated for 15 iterations, with 6400 interactions with the environment collected in each iteration. We apply a standard pre-processing for Atari games: a frame skip equal to 4, that is every action is repeated 4 times and frames are down-scaled by a factor of 2. Because some data is collected before the first iteration of the loop, altogether $6400 \cdot 16 = 102,400$ interactions with the Atari

environment are used during training. This is equivalent to $409,600$ frames from the Atari game (114 minutes in NTCS, 60 FPS). All our code is available as part of the Tensor2Tensor library and it includes instructions on how to run our experiments.[1]

At every iteration, the latest policy trained under the learned model is used to collect data in the real environment `env`. Due to vast difference between number of training data from simulated environment env' and real environment env – 15M vs 100K – we believe that the impact of real data on policy is negligible. We evaluate our method on 26 games selected on the basis of being solvable with existing state-of-the-art model-free deep RL algorithms[2], which in our comparisons are Rainbow (Hessel et al., 2017) and PPO (Schulman et al., 2017). For Rainbow, we used the implementation from the Dopamine package and spent considerable time tuning it for sample efficiency.

For visualization of all experiments see the supplementary website [3], and for a summary see Figures 3 and 4. It can be seen that our method is more sample-efficient than a highly tuned Rainbow baseline on almost all games, requires less than half of the samples on more than half of the games and, on `Freeway` is more than 10x more sample-efficient. In terms of learning speed, our method outperforms PPO by an even larger margin.

Game	SimPLe (ours)	rainbow_100k	ppo_500k	human	random
Alien	405.2	290.6	269.0	7128.0	184.8
Amidar	88.0	20.8	93.2	1720.0	11.8
Assault	369.3	300.3	552.3	742.0	233.7
Asterix	1089.5	285.7	1085.0	8503.0	248.8
BankHeist	8.2	34.5	641.0	753.0	15.0
BattleZone	5184.4	3363.5	14400.0	37188.0	2895.0
Boxing	9.1	0.9	3.5	12.0	0.3
Breakout	12.7	3.3	66.1	30.0	0.9
ChopperCommand	1246.9	776.6	860.0	7388.0	671.0
CrazyClimber	39827.8	12558.3	33420.0	35829.0	7339.5
DemonAttack	169.5	431.6	216.5	1971.0	140.0
Freeway	20.3	0.1	14.0	30.0	0.0
Frostbite	254.7	140.1	214.0	-	74.0
Gopher	771.0	748.3	560.0	2412.0	245.9
Hero	1295.1	2676.3	1824.0	30826.0	224.6
Jamesbond	125.3	61.7	255.0	303.0	29.2
Kangaroo	323.1	38.7	340.0	3035.0	42.0
Krull	4539.9	2978.8	3056.1	2666.0	1543.3
KungFuMaster	17257.2	1019.4	17370.0	22736.0	616.5
MsPacman	762.8	364.3	306.0	6952.0	235.2
Pong	5.2	-19.5	-8.6	15.0	-20.4
PrivateEye	58.3	42.1	20.0	69571.0	26.6
Qbert	559.8	235.6	757.5	13455.0	166.1
RoadRunner	5169.4	524.1	5750.0	7845.0	0.0
Seaquest	370.9	206.3	692.0	42055.0	61.1
UpNDown	2152.6	1346.3	12126.0	11693.0	488.4

Table 1: Mean scores obtained by our method (SimPLe) in comparison with Rainbow trained on 100K steps (400K frames) and PPO trained on 500K steps (2 millions frames). Details and extended numerical results are included in Appendix A.

7. Analysis

In this section, we analyze the results of our experiments. Our goals are to study how much model-based reinforcement learning can improve over the efficiency of current model-free deep reinforcement learning algorithms, analyze the quality of the predictions made by our model, and examine the design decisions in our method. Unless stated otherwise, we assume that SimPLe uses rollouts of length 50 generated with the stochastic discrete model and is trained with $\gamma = 0.95$ (see Section 4 and Section 5).

7.1. Sample Efficiency

The primary evaluation in our experiments studies the sample efficiency of SimPLe, in comparison with state-of-the-art model-free deep RL methods in the literature. To that end, we compare with Rainbow (Hessel et al., 2017; Castro et al., 2018), which represents the state-of-the-art Q-learning method for Atari games, and PPO (Schulman et al., 2017), a model-free policy gradient algorithm. The results of the comparison are presented in Figures 3 and 4. For each game, we plot the number of time steps needed for either Rainbow or PPO to reach the same score that our method reaches after 100K interaction steps. The red line indicates 100K steps: any bar larger than this indicates a game where the model-free method required more steps. SimPLe outperforms the model-free algorithms in terms of learning speed on nearly all of the games, and in the case of a few games, does so by over an order of magnitude. For some games, it reaches the same performance that our PPO implementation reaches at 10M steps. This indicates that model-based reinforcement learning provides an effective approach to learning Atari games, at a fraction of the sample complexity.

The results in these figures and Table 1 are generated by averaging 5 runs for each game. As shown in Table 1, the model-based agent is better than a random policy for all the games except `Bank Heist`. Interestingly, we observed that the best of the 5 runs was often significantly. For 6 of the games, it exceed the average human score (as reported in Table 3 of (Pohlen et al., 2018)). This suggests that further stabilizing model-based RL should improve performance, indicating an important direction for future work. In some cases during training we observed high variance of the results during each step of the loop. There are a number of possible reasons, such as mutual interactions of the policy training and the supervised training or domain mismatch between the model and the real environment. We present detailed numerical results, including best scores and standard deviations, in Appendix A.

[1] `https://github.com/tensorflow/tensor2tensor/tree/master/tensor2tensor/rl`

[2] Specifically, for the final evaluation we selected games which achieved non-random results using our method or the Rainbow algorithm using 100K interactions.

[3] `https://goo.gl/itykP8`

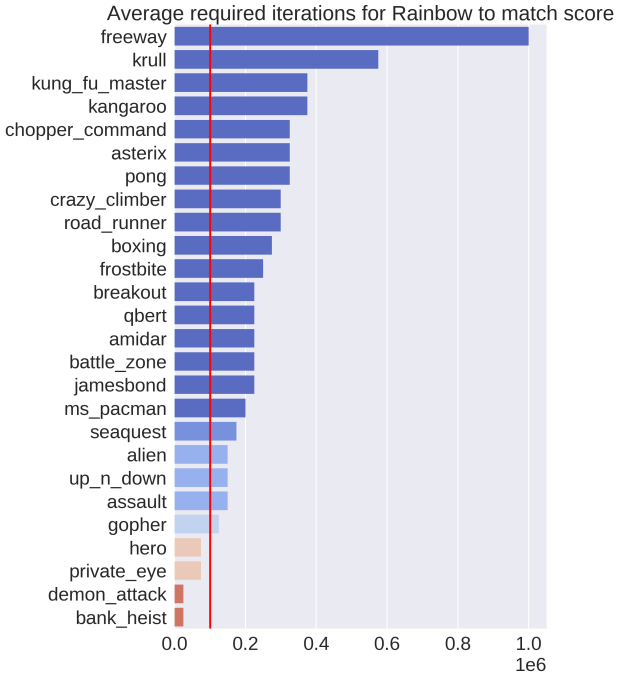

Figure 3: Comparison with Rainbow. Each bar illustrates the number of interactions with environment required by Rainbow to achieve the same score as our method (SimPLe). The red line indicates the 100K interactions threshold which is used by the our method.

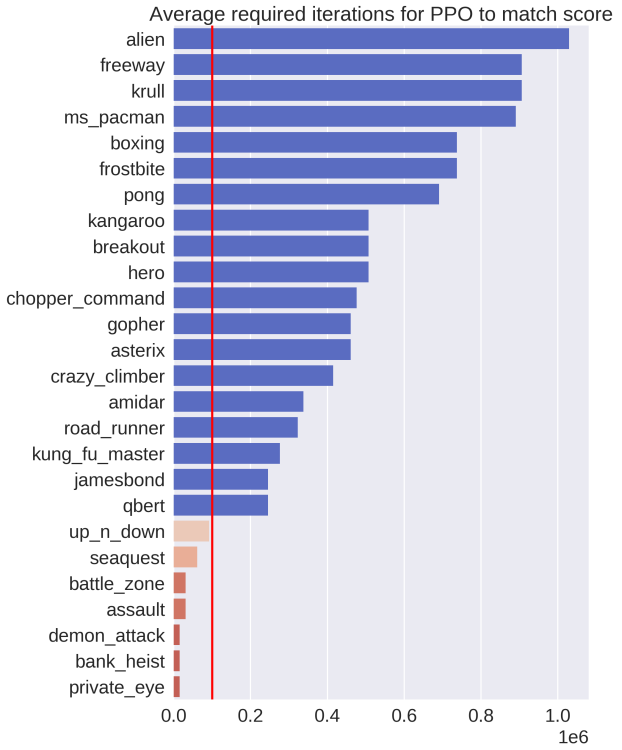

Figure 4: Comparison with PPO. Each bar illustrates the number of interactions with environment required by PPO to achieve the same score as our method (SimPLe). The red line indicates the 100K interactions threshold which is used by the our method.

7.2. Ablations

To evaluate the design of our method, we independently varied a number of the design decisions: the choice of the model, the γ parameter and the length of PPO rollouts. The results for 7 experimental configurations are summarized in the table below.

For each game, a configuration was assigned a score being the mean over 5 experiments. The best and median scores were calculated per game. The columns report the number of games a given configuration achieved the best score or at least the median score, respectively.

Models. To evaluate the model choice, we evaluated the following models: deterministic, deterministic recurrent, and stochastic discrete (see Section 4). As can be seen, our proposed stochastic discrete model performs best. Figures 9 and 10 show the role of stochasticity and recurrence.

Steps. See Figures 7, 8. As described in Section 5 every N steps we reinitialize the simulated environment with ground-truth data. By default we use $N = 50$, in some experiments we set $N = 25$ or $N = 100$. It is clear from the table above and Figure 7 that 100 is a bit worse than either 25 or 50, likely due to compounding model errors, but this effect is much smaller than the effect of model architecture.

Gamma. See Figures 5, 6. We used the discount factor $\gamma = 0.99$ unless specified otherwise. We see that $\gamma = 0.95$ is slightly better than other values, and we hypothesize that it is due to better tolerance to model imperfections. But overall, all three values of γ seem to perform comparably at the same number of steps.

Model-based iterations. The iterative process of training the model, training the policy, and collecting data is crucial for non-trivial tasks where simple random data collection is insufficient. In the game-by-game analysis, we quantified the number of games where the best results were obtained in later iterations of training. In some games, good policies could be learned very early. While this might have been due simply to the high variability of training, it does suggest the possibility that much faster training – in many fewer than 100k steps – could be obtained in future work with more directed exploration policies. We leave this question to future work.

In Figure 11 we present the cumulative distribution plot for the (first) point during learning when the maximum score for the run was achieved in the main training loop of

model	best	at_least_median
deterministic	0	7
det. recurrent	3	13
SD	8	16
SD $\gamma = 0.9$	1	14
default	10	21
SD 100 steps	0	14
SD 25 steps	4	19

Table 2: By default SimPLe uses rollouts of 50 *steps generated with the stochastic discrete model and is trained with* $\gamma = 0.95$.

Algorithm 1.

Random starts. Using short rollouts is crucial to mitigate the compounding errors under the model. To ensure exploration SimPLe starts rollouts from randomly selected states taken from the real data buffer D. In Figure 11 we present a comparison with an experiment without random starts and rollouts of length 1000 on `Seaquest`. These data strongly indicate that ablating random starts substantially deteriorate results.

7.3. Qualitative Analysis

This section provides a qualitative analysis and case studies of individual games. We emphasize that we did not adjust the method nor hyperparameters individually for each game, but we provide specific qualitative analysis to better understand the predictions from the model.[4]

Solved games. The primary goal of our paper was to use model-based methods to achieve good performance within a modest budget of 100k interactions. For two games, `Pong` and `Freeway`, our method, SimPLe, was able to achieve the maximum score.

Exploration. `Freeway` is a particularly interesting game. Though simple, it presents a substantial exploration challenge. The chicken, controlled by the agents, is quite slow to ascend when exploring randomly as it constantly gets bumped down by the cars (see the left video `https://goo.gl/YHbKZ6`). This makes it very unlikely to fully cross the road and obtain a non-zero reward. Nevertheless, SimPLe is able to capture such rare events, internalize them into the predictive model and then successfully learn a successful policy.

However, this good performance did not happen on every run. We conjecture the following scenario in failing cases. If at early stages the entropy of the policy decayed too rapidly

[4]We strongly encourage the reader to watch accompanying videos `https://goo.gl/itykP8`

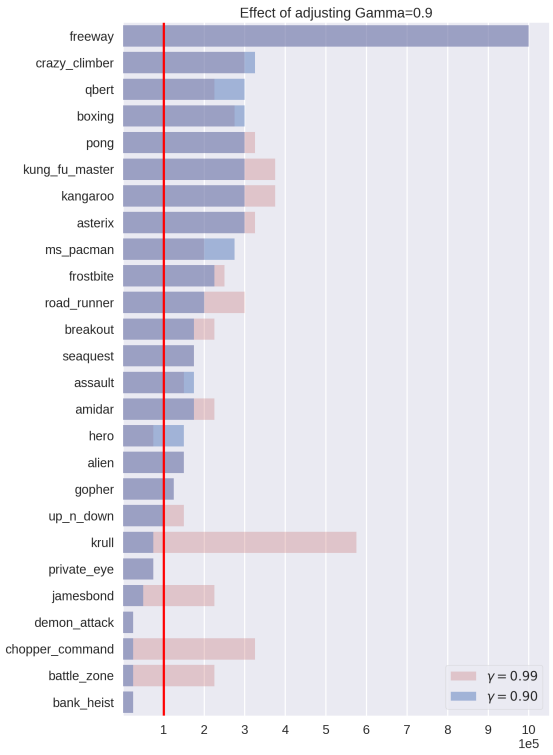

Figure 5: Effect of adjusting γ, 0.9 *vs* 0.99.

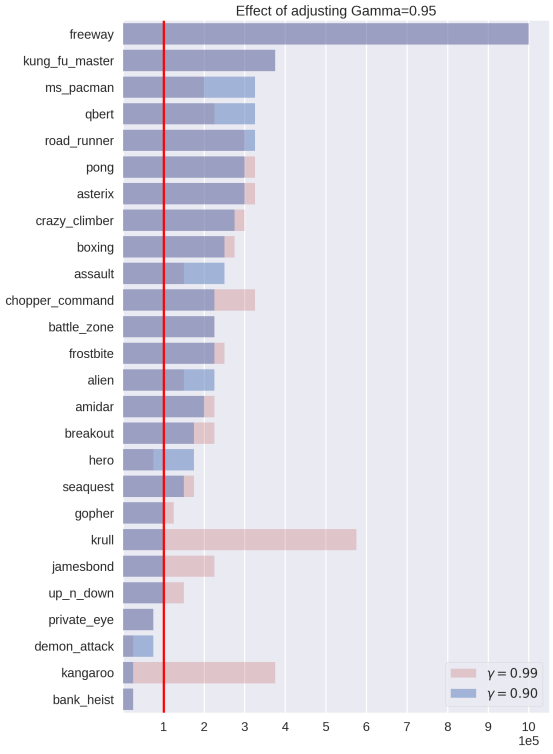

Figure 6: Effect of adjusting γ, 0.95 *vs* 0.99.

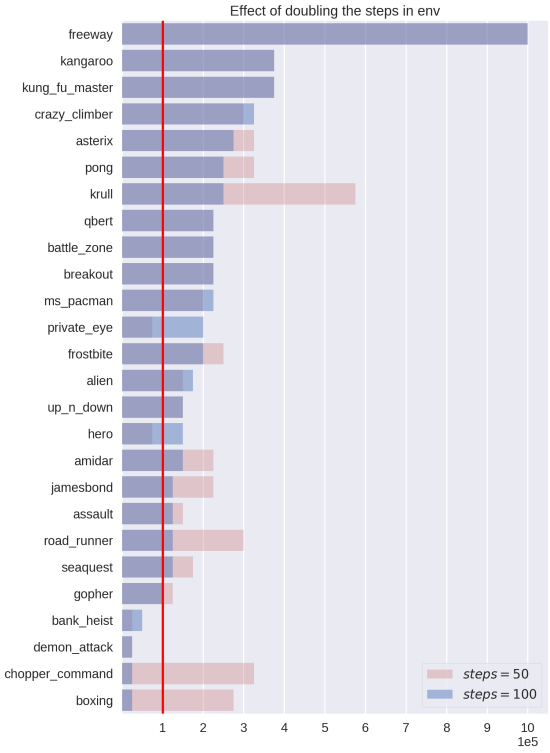

Figure 7: Effect of adjusting number of steps, 50 vs 100.

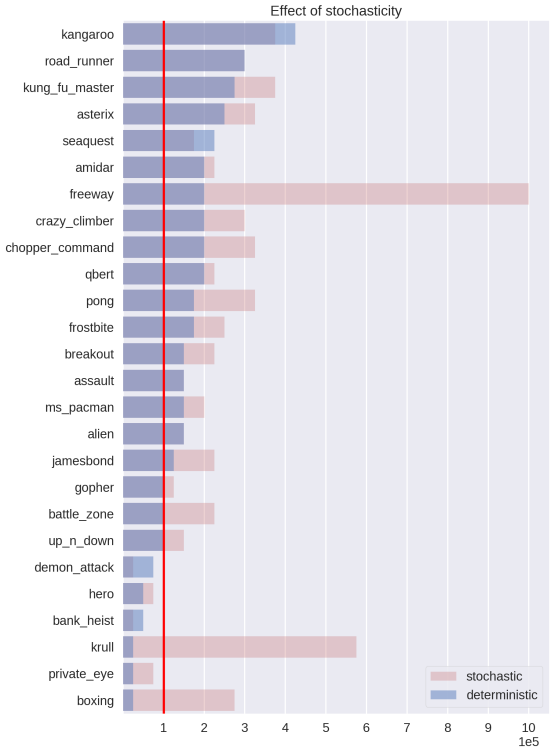

Figure 9: Effect of stochasticity.

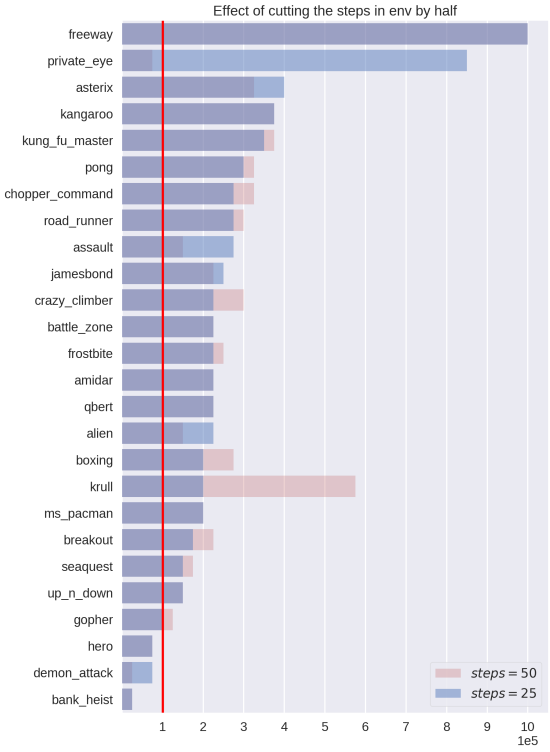

Figure 8: Effect of adjusting number of steps, 25 vs 50.

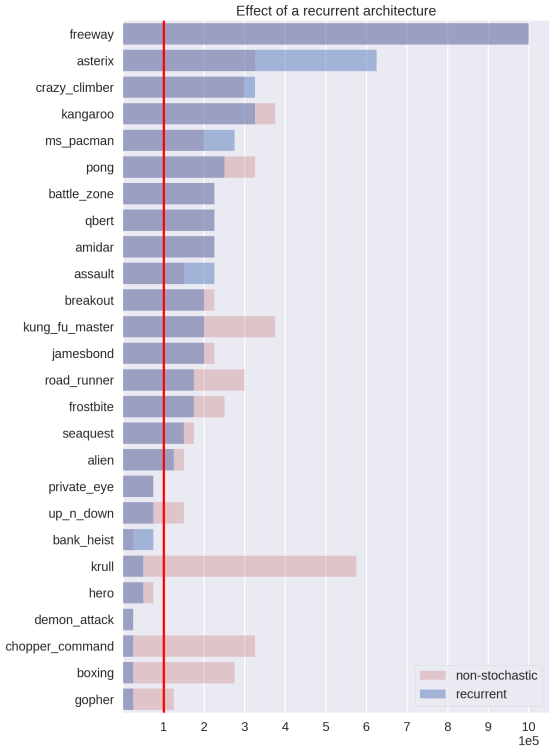

Figure 10: Comparison with a recurrent model.

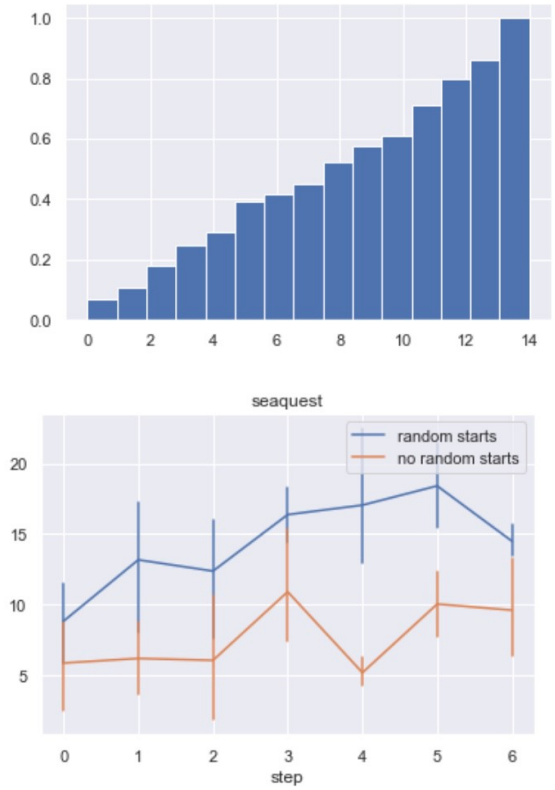

Figure 11: (up) CDF of the number of iterations to acquire maximum score. The vertical axis represents the fraction of all games. (down) Comparison of random starts vs no random starts on `Seaquest` *(for better readability we clip game rewards to* $\{-1, 0, 1\}$*). The vertical axis shows a mean reward and the horizontal axis the number of iterations of Algorithm 1.*

the collected experience stayed limited leading to a poor world model, which was not powerful enough to support exploration (e.g. the chicken disappears when moving to high). In one of our experiments, we observed that the final policy was that the chicken moved up only to the second lane and stayed waiting to be hit by the car and so on so forth.

Pixel-perfect games. In some cases (for `Pong`, `Freeway`, `Breakout`) our models were able to predict the future perfectly, down to every pixel. This property holds for rather short time intervals, we observed episodes lasting up to 50 time-steps. Extending it to long sequences would be a very exciting research direction. See videos `https://goo.gl/uyfNnW`.

Benign errors. Despite the aforementioned positive examples, accurate models are difficult to acquire for some games, especially at early stages of learning. However, model-based RL should be tolerant to modest model errors. Interestingly, in some cases our models differed from the original games in a way that was harmless or only mildly harmful for policy training.

For example, in `Bowling` and `Pong`, the ball sometimes splits into two. While nonphysical, seemingly these errors did not distort much the objective of the game, see Figure 12 and also `https://goo.gl/JPi7rB`.

Figure 12: Frames from the `Pong` *environment.*

In `Kung Fu Master` our model's predictions deviate from the real game by spawning a different number of opponents, see Figure 13. In `Crazy Climber` we observed the bird appearing earlier in the game. These cases are probably to be attributed to the stochasticity in the model. Though not aligned with the true environment, the predicted behaviors are plausible, and the resulting policy can still play the original game.

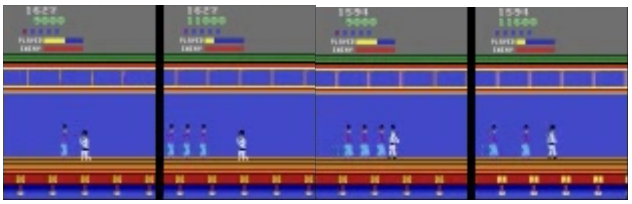

Figure 13: Frames from the `Kung Fu Master` *environment (left) and its model (right).*

Failures on hard games. On some of the games, our models simply failed to produce useful predictions. We believe that listing such errors may be helpful in designing better training protocols and building better models. The most common failure was due to the presence of very small but highly relevant objects. For example, in `Atlantis` and `Battle Zone` bullets are so small that they tend to disappear. Interestingly, `Battle Zone` has pseudo-3D graphics, which may have added to the difficulty. See videos `https://goo.gl/uiccKU`.

Another interesting example comes from `Private Eye` in which the agent traverses different scenes, teleporting from one to the other. We found that our model generally struggled to capture such large global changes.

8. Conclusions and Future Work

We presented, SimPLe, a model-based reinforcement learning approach that can operate directly on raw pixel observations, and can learn effective policies to play games in the Atari Learning Environment. Our experiments demonstrate that SimPLe can learn to play many of the games with just 100K transitions, corresponding to 2 hours of play time. In many cases, the number of samples required for prior methods to learn to reach the same reward value is several times larger.

Our predictive model has stochastic latent variables thus, hopefully, can be applied to truly stochastic domains. Studying such domains is an exciting direction for future work and we expect that effective probabilistic modeling of dynamics would be effective in such settings as well. Another interesting direction for future work is to study other ways in which the predictive neural network model could be used. Our approach utilizes the model as a learned simulator and then directly uses model-free policy search methods to acquire the behavior policy. However, since neural network models are differentiable, the additional information contained in the dynamics gradients could itself be incorporated into the reinforcement learning process. Finally, the representation learned by the predictive model is likely be more meaningful by itself than the raw pixel observations from the environment, and incorporating this representation into the policy could further accelerate and improve the reinforcement learning process.

While SimPLe is able to learn much more quickly than model-free methods, it does have a number of limitations. First, the final scores are on the whole substantially lower than the best state-of-the-art model-free methods. This is generally common with model-based RL algorithms, which excel more in learning efficiency rather than final performance, but suggests an important direction for improvement in future work. Another, less obvious limitation is that the performance of our method generally varied substantially between different runs on the same game. The complex interactions between the model, policy, and data collection were likely responsible for this: at a fundamental level, the model makes guesses when it extrapolates the behavior of the game under a new policy. When these guesses are correct, the resulting policy performs well in the final game. In future work, models that capture uncertainty via Bayesian parameter posteriors or ensembles may further improve robustness (Kurutach et al., 2018; Chua et al., 2018).

As a long-term challenge, we believe that model-based reinforcement learning based on stochastic predictive models represents a promising and highly efficient alternative to model-free RL. Applications of such approaches to both high-fidelity simulated environments and real-world data represent an exciting direction for future work that can enable highly efficient learning of behaviors from raw sensory inputs in domains such as robotics and autonomous driving.

Acknowledgments

We thank Marc Bellemare and Pablo Castro for their help with Rainbow and Dopamine. The work of Konrad Czechowski, Piotr Kozakowski and Piotr Miłoś was supported by the Polish National Science Center grants UMO-2017/26/E/ST6/00622. The work of Henryk Michalewski was supported by the Polish National Science Center grant UMO-2018/29/B/ST6/02959. This research was supported by the PL-Grid Infrastructure. In particular, Konrad Czechowski, Piotr Kozakowski, Henryk Michalewski, Piotr Miłoś and Błażej Osiński extensively used the Prometheus supercomputer, located in the Academic Computer Center Cyfronet in the AGH University of Science and Technology in Kraków, Poland.

References

Babaeizadeh, M., Frosio, I., Tyree, S., Clemons, J., and Kautz, J. Reinforcement learning through asynchronous advantage actor-critic on a gpu. *arXiv preprint arXiv:1611.06256*, 2016.

Babaeizadeh, M., Finn, C., Erhan, D., Campbell, R. H., and Levine, S. Stochastic variational video prediction. *ICLR*, 2017.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents (extended abstract). In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pp. 4148–4152, 2015.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 1171–1179, 2015.

Buesing, L., Weber, T., Zwols, Y., Racanière, S., Guez, A., Lespiau, J., and Heess, N. Woulda, coulda, shoulda: Counterfactually-guided policy search. *CoRR*, abs/1811.06272, 2018.

Castro, P. S., Moitra, S., Gelada, C., Kumar, S., and Bellemare, M. G. Dopamine: A research framework for deep reinforcement learning. *CoRR*, abs/1812.06110, 2018.

Chiappa, S., Racanière, S., Wierstra, D., and Mohamed, S. Recurrent environment simulators. *CoRR*, abs/1704.02254, 2017.

Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pp. 4759–4770, 2018.

Deisenroth, M. P., Neumann, G., and Peters, J. A survey

on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2), 2013.

Ebert, F., Finn, C., Lee, A. X., and Levine, S. Self-supervised visual planning with temporal skip connections. *CoRR*, abs/1710.05268, 2017.

Ebert, F., Finn, C., Dasari, S., Xie, A., Lee, A., and Levine, S. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, pp. 1406–1415, 2018.

Feinberg, V., Wan, A., Stoica, I., Jordan, M. I., Gonzalez, J. E., and Levine, S. Model-based value estimation for efficient model-free reinforcement learning. *CoRR*, abs/1803.00101, 2018.

Finn, C. and Levine, S. Deep visual foresight for planning robot motion. *CoRR*, abs/1610.00696, 2016.

Finn, C., Tan, X. Y., Duan, Y., Darrell, T., Levine, S., and Abbeel, P. Deep spatial autoencoders for visuomotor learning. In *IEEE International Conference on Robotics and Automation, ICRA*, pp. 512–519, 2016.

Ha, D. and Schmidhuber, J. World models. *CoRR*, abs/1803.10122, 2018.

Hafner, D., Lillicrap, T. P., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. *CoRR*, abs/1811.04551, 2018.

Heess, N., Wayne, G., Silver, D., Lillicrap, T. P., Tassa, Y., and Erez, T. Learning continuous control policies by stochastic value gradients. *CoRR*, abs/1510.09142, 2015.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298, 2017.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Kaiser, L. and Bengio, S. Discrete autoencoders for sequence models. *CoRR*, abs/1801.09797, 2018.

Kalweit, G. and Boedecker, J. Uncertainty-driven imagination for continuous deep reinforcement learning. In Levine, S., Vanhoucke, V., and Goldberg, K. (eds.), *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pp. 195–206. PMLR, 13–15 Nov 2017.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. Model-ensemble trust-region policy optimization. *CoRR*, abs/1802.10592, 2018.

Leibfried, F., Kushman, N., and Hofmann, K. A deep learning approach for joint video frame and reward prediction in Atari games. *CoRR*, abs/1611.07078, 2016.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M. J., and Bowling, M. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *CoRR*, abs/1709.06009, 2017.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML*, pp. 1928–1937, 2016.

Oh, J., Guo, X., Lee, H., Lewis, R. L., and Singh, S. P. Action-conditional video prediction using deep networks in atari games. In *NIPS*, pp. 2863–2871, 2015.

Oh, J., Singh, S., and Lee, H. Value prediction network. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 6118–6128. Curran Associates, Inc., 2017.

Paxton, C., Barnoy, Y., Katyal, K. D., Arora, R., and Hager, G. D. Visual robot task planning. *CoRR*, abs/1804.00062, 2018.

Piergiovanni, A. J., Wu, A., and Ryoo, M. S. Learning real-world robot policies by dreaming. *CoRR*, abs/1805.07813, 2018.

Pohlen, T., Piot, B., Hester, T., Azar, M. G., Horgan, D., Budden, D., Barth-Maron, G., van Hasselt, H., Quan, J., Vecerík, M., Hessel, M., Munos, R., and Pietquin, O. Observe and look further: Achieving consistent performance on atari. *CoRR*, abs/1805.11593, 2018.

Rybkin, O., Pertsch, K., Jaegle, A., Derpanis, K. G., and Daniilidis, K. Unsupervised learning of sensorimotor affordances by stochastic future prediction. *CoRR*, abs/1806.09655, 2018.

Schmidhuber, J. Formal theory of creativity, fun, and intrinsic motivation (1990-2010). *IEEE Trans. Autonomous Mental Development*, 2(3):230–247, 2010.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I.,

and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Sutton, R. S. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, July 1991.

Sutton, R. S. and Barto, A. G. *Reinforcement learning - an introduction, 2nd edition (work in progress)*. Adaptive computation and machine learning. MIT Press, 2017.

Tsividis, P., Pouncy, T., Xu, J. L., Tenenbaum, J. B., and Gershman, S. J. Human learning in atari. In *2017 AAAI Spring Symposia, Stanford University, Palo Alto, California, USA, March 27-29, 2017*, 2017.

van den Oord, A., Vinyals, O., and Kavukcuoglu, K. Neural discrete representation learning. *CoRR*, abs/1711.00937, 2017.

Watter, M., Springenberg, J. T., Boedecker, J., and Riedmiller, M. A. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems*, pp. 2746–2754, 2015.

Wu, Y., Mansimov, E., Liao, S., Grosse, R. B., and Ba, J. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *CoRR*, abs/1708.05144, 2017.

A. Numerical results

Below we present numerical results of our experiments. We tested SimPLe on 7 configurations (see description in Section 7.2). For each configuration we run 5 experiments. For the evaluation of the i-th experiments we used the policy given by softmax(logits$(\pi_i)/T$), where π_i is the final learnt policy in the experiment and T is the temperature parameter. We found empirically that $T = 0.5$ worked best in most cases. A tentative explanation is that polices with temperatures smaller than 1 are less stochastic and thus more stable. However, going down to $T = 0$ proved to be detrimental in many cases as, possibly, it makes policies more prone to imperfections of models.

In Table 3 we present the mean and standard deviation of the 5 experiments. We observed that the median behaves rather similarly, which is reported it in Table 5. In this table we also show maximal scores over 5 runs. Interestingly, in many cases they turned out to be much higher. This, we hope, indicates that our methods has a further potential of reaching these higher scores.

Human scores are "Avg. Human" from Table 3 in (Pohlen et al., 2018).

Game	Ours, deterministic		Ours, det. recurrent		Ours, SD		Ours, SD $\gamma = 0.90$		Ours, SD $\gamma = 0.95$		Ours, SD 100 steps		Ours, SD 25 steps		random	human
Alien	378.3	(85.5)	321.7	(50.7)	405.2	(130.8)	413.0	(89.7)	**590.2**	(57.8)	435.6	(78.9)	534.8	(166.2)	184.8	7128.0
Amidar	62.4	(15.2)	86.7	(18.8)	**88.0**	(23.8)	50.3	(11.7)	78.3	(18.8)	37.7	(15.1)	82.2	(43.0)	11.8	1720.0
Assault	361.4	(166.6)	490.5	(143.6)	369.3	(107.8)	406.7	(118.7)	549.0	(127.9)	311.7	(88.2)	**664.5**	(298.2)	233.7	742.0
Asterix	668.0	(294.1)	**1853.0**	(391.8)	1089.5	(335.3)	855.0	(176.4)	921.6	(114.2)	777.0	(200.4)	1340.6	(627.5)	248.8	8503.0
Asteroids	743.7	(92.2)	821.7	(115.6)	731.0	(165.3)	882.0	(24.7)	**886.8**	(45.2)	821.9	(93.8)	644.5	(110.6)	649.0	47389.0
Atlantis	14623.4	(2122.5)	12584.4	(5823.6)	14481.6	(2436.9)	**18444.1**	(4616.0)	14055.6	(6226.1)	14139.7	(2500.9)	11641.2	(3385.0)	16492.0	29028.0
BankHeist	13.8	(2.5)	**15.1**	(2.2)	8.2	(4.4)	11.9	(2.5)	12.0	(1.4)	13.1	(3.2)	12.7	(4.7)	15.0	753.0
BattleZone	3306.2	(794.1)	4665.6	(2799.4)	**5184.4**	(1347.5)	2781.2	(661.7)	4000.0	(788.9)	4068.8	(2912.1)	3746.9	(1426.8)	2895.0	37188.0
BeamRider	**463.8**	(29.2)	358.9	(87.4)	422.7	(103.6)	456.2	(160.8)	415.4	(103.4)	456.0	(60.9)	386.6	(264.4)	372.1	16926.0
Bowling	25.3	(10.4)	22.3	(17.0)	**34.4**	(16.3)	27.7	(5.2)	23.9	(3.3)	29.3	(7.5)	33.2	(15.5)	24.2	161.0
Boxing	-9.3	(10.9)	-3.1	(14.1)	9.1	(8.8)	**11.6**	(12.6)	5.1	(10.0)	-2.1	(5.0)	1.6	(14.7)	0.3	12.0
Breakout	6.1	(2.8)	10.2	(5.1)	**12.7**	(3.8)	7.3	(2.4)	8.8	(5.1)	11.4	(3.7)	7.8	(4.1)	0.9	30.0
ChopperCommand	906.9	(210.2)	709.1	(174.1)	**1246.9**	(392.0)	725.6	(204.2)	946.6	(49.9)	729.1	(185.1)	1047.2	(221.6)	671.0	7388.0
CrazyClimber	19380.0	(6138.8)	**54700.3**	(14480.5)	39827.8	(22582.6)	49840.9	(11920.9)	34353.1	(33547.2)	48651.2	(14903.5)	25612.2	(14037.5)	7339.5	35829.0
DemonAttack	191.9	(86.3)	120.3	(38.3)	169.5	(41.8)	187.5	(68.6)	194.9	(89.6)	170.1	(42.4)	**202.2**	(134.0)	140.0	1971.0
FishingDerby	-94.5	(3.0)	-96.9	(1.7)	-91.5	(2.8)	-91.0	(4.1)	-92.6	(3.2)	**-90.0**	(2.7)	-94.5	(2.5)	-93.6	-39.0
Freeway	5.9	(13.1)	23.7	(13.5)	20.3	(18.5)	18.9	(17.2)	**27.7**	(13.3)	19.1	(16.7)	27.3	(5.8)	0.0	30.0
Frostbite	196.4	(4.4)	219.6	(21.4)	**254.7**	(4.9)	234.6	(26.8)	239.2	(19.1)	226.8	(16.9)	252.1	(54.4)	74.0	-
Gopher	510.2	(158.4)	225.2	(105.7)	771.0	(160.2)	**845.6**	(230.3)	612.6	(273.9)	698.4	(213.9)	509.7	(273.4)	245.9	2412.0
Gravitar	**237.0**	(73.1)	213.8	(57.4)	198.3	(39.9)	219.4	(7.8)	213.0	(37.3)	188.9	(27.6)	116.4	(84.0)	227.2	3351.0
Hero	621.5	(1281.3)	558.3	(1143.3)	1295.1	(1600.1)	2853.9	(539.5)	**3503.5**	(892.9)	3052.7	(169.3)	1484.8	(1671.7)	224.6	30826.0
IceHockey	-12.6	(2.1)	-14.0	(1.8)	**-10.5**	(2.2)	-12.2	(2.9)	-11.9	(1.2)	-13.5	(3.0)	-13.9	(3.9)	-9.7	1.0
Jamesbond	68.8	(37.2)	100.5	(69.8)	125.3	(112.5)	28.9	(12.7)	50.5	(21.3)	68.9	(42.7)	**163.4**	(81.8)	29.2	303.0
Kangaroo	**481.9**	(313.2)	191.9	(301.0)	323.1	(359.8)	148.1	(121.5)	37.5	(8.0)	301.2	(593.4)	340.0	(470.4)	42.0	3035.0
Krull	834.9	(166.3)	1778.5	(906.9)	**4539.9**	(2470.4)	2396.5	(962.0)	2620.9	(856.2)	3559.0	(1896.7)	3320.6	(2410.1)	1543.3	2666.0
KungFuMaster	10340.9	(8835.7)	4086.6	(3384.5)	**17257.2**	(5502.6)	12587.8	(6810.0)	16926.6	(6598.3)	17121.2	(7211.6)	15541.2	(5086.1)	616.5	22736.0
MsPacman	560.6	(172.2)	1098.1	(450.9)	762.8	(331.5)	1197.1	(544.6)	**1273.3**	(59.5)	921.0	(306.0)	805.8	(261.1)	235.2	6952.0
NameThisGame	1512.1	(408.3)	2007.9	(367.0)	1990.4	(284.7)	2058.1	(103.7)	**2114.8**	(387.4)	2067.2	(304.8)	1805.3	(453.4)	2136.8	8049.0
Pong	-17.4	(5.2)	-11.6	(15.9)	**5.2**	(9.7)	-2.9	(7.3)	-2.5	(15.4)	-13.9	(7.7)	-1.0	(14.9)	-20.4	15.0
PrivateEye	16.4	(46.7)	50.8	(43.2)	58.3	(45.4)	54.4	(49.0)	67.8	(26.4)	88.3	(19.0)	**1334.3**	(1794.5)	26.6	69571.0
Qbert	480.4	(158.8)	603.7	(150.3)	559.8	(183.8)	899.3	(474.3)	**1120.2**	(697.1)	534.4	(162.5)	603.4	(138.2)	166.1	13455.0
Riverraid	1285.6	(604.6)	1740.7	(458.1)	1587.0	(818.0)	1977.4	(332.7)	**2115.1**	(106.2)	1318.7	(540.4)	1426.0	(374.0)	1451.0	17118.0
RoadRunner	5724.4	(3093.1)	1228.8	(1025.9)	5169.4	(3939.0)	1586.2	(1574.1)	**8414.1**	(4542.8)	722.2	(627.2)	4366.2	(3867.8)	0.0	7845.0
Seaquest	**419.5**	(236.2)	289.6	(110.4)	370.9	(128.2)	364.6	(138.6)	337.8	(79.0)	247.8	(72.4)	350.0	(136.8)	61.1	42055.0
UpNDown	1329.3	(495.3)	926.7	(335.7)	**2152.6**	(1192.4)	1291.2	(324.6)	1250.6	(493.0)	1828.4	(688.3)	2136.5	(2095.0)	488.4	11693.0
YarsRevenge	3014.9	(397.4)	3291.4	(1097.3)	2980.2	(778.6)	2934.2	(459.2)	3366.6	(493.0)	2673.7	(216.8)	**4666.1**	(1889.4)	3121.2	54577.0

Table 3: Models comparison. Mean scores and standard deviations over five training runs. Right most columns presents score for random agent and human.

Game	Ours, SD		PPO_100k		PPO_500k		PPO_1m		Rainbow_100k		Rainbow_500k		Rainbow_1m		random	human
Alien	405.2	(130.8)	291.0	(40.3)	269.0	(203.4)	362.0	(102.0)	290.6	(14.8)	828.6	(54.2)	945.0	(85.0)	184.8	7128.0
Amidar	88.0	(23.8)	56.5	(20.8)	93.2	(36.7)	123.8	(19.7)	20.8	(2.3)	194.0	(34.9)	275.8	(66.7)	11.8	1720.0
Assault	369.3	(107.8)	424.2	(55.8)	552.3	(110.4)	1134.4	(798.8)	300.3	(14.6)	1041.5	(92.1)	1581.8	(207.8)	233.7	742.0
Asterix	1089.5	(335.3)	385.0	(104.4)	1085.0	(354.8)	2185.0	(931.6)	285.7	(9.3)	1702.7	(162.8)	2151.6	(202.6)	248.8	8503.0
Asteroids	731.0	(165.3)	1134.0	(326.9)	1053.0	(433.3)	1251.0	(377.9)	912.3	(62.7)	895.9	(82.0)	1071.5	(91.7)	649.0	47389.0
Atlantis	14481.6	(2436.9)	34316.7	(5703.8)	4836416.7	(6218247.3)	-	(-)	17881.8	(617.6)	79541.0	(25393.4)	848800.0	(37533.1)	16492.0	29028.0
BankHeist	8.2	(4.4)	16.0	(12.4)	641.0	(352.8)	856.0	(376.7)	34.5	(2.0)	727.3	(198.3)	1053.3	(22.9)	15.0	753.0
BattleZone	5184.4	(1347.5)	5300.0	(3655.1)	14400.0	(6476.1)	19000.0	(4571.7)	3363.5	(523.8)	19507.1	(3193.3)	22391.4	(7708.9)	2895.0	37188.0
BeamRider	422.7	(103.6)	563.6	(189.4)	497.6	(103.5)	684.0	(168.8)	365.6	(29.8)	5890.0	(525.6)	6945.3	(1390.8)	372.1	16926.0
Bowling	34.4	(16.3)	17.7	(11.2)	28.5	(3.4)	35.8	(6.2)	24.7	(0.8)	31.0	(1.9)	30.6	(6.2)	24.2	161.0
Boxing	9.1	(8.8)	-3.9	(6.4)	3.5	(3.5)	19.6	(20.9)	0.9	(1.7)	58.2	(16.5)	80.3	(5.6)	0.3	12.0
Breakout	12.7	(3.8)	5.9	(3.3)	66.1	(114.3)	128.0	(153.3)	3.3	(0.1)	26.7	(2.4)	38.7	(3.4)	0.9	30.0
ChopperCommand	1246.9	(392.0)	730.0	(199.0)	860.0	(285.3)	970.0	(201.5)	776.6	(59.0)	1765.2	(280.7)	2474.0	(504.5)	671.0	7388.0
CrazyClimber	39827.8	(22582.6)	18400.0	(5275.1)	33420.0	(3628.3)	58000.0	(16994.6)	12558.3	(674.6)	75655.1	(9439.6)	97088.1	(9975.4)	7339.5	35829.0
DemonAttack	169.5	(41.8)	192.5	(83.1)	216.5	(96.2)	241.0	(135.0)	431.6	(79.5)	3642.1	(478.2)	5478.6	(297.9)	140.0	1971.0
FishingDerby	-91.5	(2.8)	-95.6	(4.3)	-87.2	(5.3)	-88.8	(4.0)	-91.1	(2.1)	-66.7	(6.0)	-23.2	(22.3)	-93.6	-39.0
Freeway	20.3	(18.5)	8.0	(9.8)	14.0	(11.5)	20.8	(11.1)	0.1	(0.1)	12.6	(15.4)	13.0	(15.9)	0.0	30.0
Frostbite	254.7	(4.9)	174.0	(40.7)	214.0	(10.2)	229.0	(20.6)	140.1	(2.7)	1386.1	(321.7)	2972.3	(284.9)	74.0	-
Gopher	771.0	(160.2)	246.0	(103.3)	560.0	(118.8)	696.0	(279.3)	748.3	(105.4)	1640.5	(105.6)	1905.0	(211.1)	245.9	2412.0
Gravitar	198.3	(39.9)	235.0	(197.2)	235.0	(134.7)	325.0	(85.1)	231.4	(50.7)	214.9	(27.6)	260.0	(22.7)	227.2	3351.0
Hero	1295.1	(1600.1)	569.0	(1100.9)	1824.0	(1461.2)	3719.0	(1306.0)	2676.3	(93.7)	10664.3	(1060.5)	13295.5	(261.2)	224.6	30826.0
IceHockey	-10.5	(2.2)	-10.0	(2.1)	-6.6	(1.6)	-5.3	(1.7)	-9.5	(0.8)	-9.7	(0.8)	-6.5	(0.5)	-9.7	1.0
Jamesbond	125.3	(112.5)	65.0	(46.4)	255.0	(101.7)	310.0	(129.0)	61.7	(8.8)	429.7	(27.9)	692.6	(316.2)	29.2	303.0
Kangaroo	323.1	(359.8)	140.0	(102.0)	340.0	(407.9)	840.0	(806.5)	38.7	(9.3)	970.9	(501.9)	4084.6	(1954.1)	42.0	3035.0
Krull	4539.9	(2470.4)	3750.4	(3071.9)	3056.1	(1155.5)	5061.8	(1333.4)	2978.8	(148.4)	4139.4	(336.2)	4971.1	(360.3)	1543.3	2666.0
KungFuMaster	17257.2	(5502.6)	4820.0	(983.2)	17370.0	(10707.6)	13780.0	(3971.6)	1019.4	(149.6)	19346.1	(3274.4)	21258.6	(3210.2)	616.5	22736.0
MsPacman	762.8	(331.5)	496.0	(379.8)	306.0	(70.2)	594.0	(247.9)	364.3	(20.4)	1558.0	(248.9)	1881.4	(112.0)	235.2	6952.0
NameThisGame	1990.4	(284.7)	2225.0	(423.7)	2106.0	(898.8)	2311.0	(547.6)	2368.2	(318.3)	4886.5	(583.1)	4454.2	(338.3)	2136.8	8049.0
Pong	5.2	(9.7)	-20.5	(0.6)	-8.6	(14.9)	14.7	(5.1)	-19.5	(0.2)	19.9	(0.4)	20.6	(0.2)	-20.4	15.0
PrivateEye	58.3	(45.4)	10.0	(20.0)	20.0	(40.0)	20.0	(40.0)	42.1	(53.8)	-6.2	(89.8)	2336.7	(4732.6)	26.6	69571.0
Qbert	559.8	(183.8)	362.5	(117.8)	757.5	(78.9)	2675.0	(1701.1)	235.6	(12.9)	4241.7	(193.1)	8885.2	(1690.9)	166.1	13455.0
Riverraid	1587.0	(818.0)	1398.0	(513.8)	2865.0	(327.1)	2887.0	(807.0)	1904.2	(44.2)	5068.6	(292.6)	7018.9	(334.2)	1451.0	17118.0
RoadRunner	5169.4	(3939.0)	1430.0	(760.0)	5750.0	(5259.9)	8930.0	(4304.0)	524.1	(147.5)	18415.4	(5280.0)	31379.7	(3225.8)	0.0	7845.0
Seaquest	370.9	(128.2)	370.0	(103.3)	692.0	(48.3)	882.0	(122.7)	206.3	(17.1)	1558.7	(221.2)	3279.9	(683.9)	61.1	42055.0
UpNDown	2152.6	(1192.4)	2874.0	(1105.8)	12126.0	(1389.5)	13777.0	(6766.3)	1346.3	(95.1)	6120.7	(356.8)	8010.9	(907.0)	488.4	11693.0
YarsRevenge	2980.2	(778.6)	5182.0	(1209.3)	8064.8	(2859.8)	9495.0	(2638.3)	3649.0	(168.6)	7005.7	(394.2)	8225.1	(957.9)	3121.2	54577.0

Table 4: Comparison of our method (SimPLe) with model-free benchmarks - PPO and Rainbow, trained with 100 thousands/500 thousands/1 million steps. (1 step equals 4 frames)

Game	Ours, deterministic		Ours, det. recurrent		Ours, SD		Ours, SD $\gamma = 0.90$		Ours, SD $\gamma = 0.95$		SD 100 steps		Ours, SD 25 steps		random	human
Alien	354.4	516.6	299.2	381.1	409.2	586.9	411.9	530.5	567.3	682.7	399.5	522.3	525.5	792.8	184.8	7128.0
Amidar	58.0	84.8	82.7	118.4	85.1	114.0	55.1	58.9	84.3	101.4	45.2	47.5	93.1	137.7	11.8	1720.0
Assault	334.4	560.1	566.6	627.2	355.7	527.9	369.1	614.4	508.4	722.5	322.9	391.1	701.4	1060.3	233.7	742.0
Asterix	529.7	1087.5	1798.4	2282.0	1158.6	1393.8	805.5	1159.4	923.4	1034.4	813.3	1000.0	1128.1	2313.3	248.8	8503.0
Asteroids	727.3	854.7	827.7	919.8	671.2	962.0	885.5	909.1	886.1	949.5	813.8	962.2	657.5	752.7	649.0	47389.0
Atlantis	15587.5	16545.3	15939.1	17778.1	13645.3	18396.9	19367.2	23046.9	12981.2	23579.7	15020.3	16790.6	12196.9	15728.1	16492.0	29028.0
BankHeist	14.4	16.2	14.7	18.8	8.9	13.9	12.3	14.5	12.3	13.1	12.8	17.2	14.1	17.0	15.0	753.0
BattleZone	3312.5	4140.6	4515.6	9312.5	5390.6	7093.8	2937.5	3343.8	4421.9	4703.1	3500.0	8906.2	3859.4	5734.4	2895.0	37188.0
BeamRider	453.1	515.5	351.4	470.2	433.9	512.6	393.5	682.8	446.6	519.2	447.1	544.6	385.7	741.9	372.1	16926.0
Bowling	27.0	36.2	28.4	43.7	24.9	55.0	27.7	34.9	22.6	28.6	28.4	39.9	37.0	54.7	24.2	161.0
Boxing	-7.1	0.2	3.5	5.0	8.3	21.5	6.4	31.5	2.5	15.0	-0.7	2.2	-0.9	20.8	0.3	12.0
Breakout	5.5	9.8	12.5	13.9	11.0	19.5	7.4	10.4	10.2	14.1	10.5	16.7	6.9	13.0	0.9	30.0
ChopperCommand	942.2	1167.2	748.4	957.8	1139.1	1909.4	682.8	1045.3	954.7	1010.9	751.6	989.1	1031.2	1329.7	671.0	7388.0
CrazyClimber	20754.7	23831.2	49854.7	80156.2	41396.9	67250.0	56875.0	58979.7	19448.4	84070.3	53406.2	64196.9	19345.3	43179.7	7339.5	35829.0
DemonAttack	219.2	263.0	135.8	148.4	182.4	223.9	160.3	293.8	204.1	312.8	164.4	222.6	187.5	424.8	140.0	1971.0
FishingDerby	-94.3	-90.2	-97.3	-94.2	-91.6	-88.6	-90.0	-85.7	-92.0	-88.8	-90.6	-85.4	-95.0	-90.7	-93.6	-39.0
Freeway	0.0	29.3	29.3	32.2	33.5	34.0	31.1	32.0	33.5	33.8	30.0	32.3	29.9	33.5	0.0	30.0
Frostbite	194.5	203.9	213.4	256.2	253.1	262.8	246.7	261.7	250.0	255.9	215.8	247.7	249.4	337.5	74.0	-
Gopher	514.7	740.6	270.3	320.9	856.9	934.4	874.1	1167.2	604.1	1001.6	726.9	891.6	526.2	845.0	245.9	2412.0
Gravitar	232.8	310.2	219.5	300.0	202.3	252.3	223.4	225.8	228.1	243.8	193.8	218.0	93.0	240.6	227.2	3351.0
Hero	71.5	2913.0	75.0	2601.5	237.5	3133.8	3135.0	3147.5	3066.2	5092.0	3067.3	3256.9	1487.2	2964.8	224.6	30826.0
IceHockey	-12.4	-9.9	-14.8	-11.8	-10.0	-7.7	-11.8	-8.5	-11.6	-10.7	-12.9	-10.0	-12.2	-11.0	-9.7	1.0
Jamesbond	64.8	128.9	64.8	219.5	87.5	323.4	25.0	46.9	58.6	69.5	61.7	139.1	139.8	261.7	29.2	303.0
Kangaroo	500.0	828.1	68.8	728.1	215.6	909.4	103.1	334.4	34.4	50.0	43.8	1362.5	56.2	1128.1	42.0	3035.0
Krull	852.2	1014.3	1783.6	2943.6	4264.3	7163.2	1874.8	3554.5	2254.0	3827.1	3142.8	6315.2	3198.2	6833.4	1543.3	2666.0
KungFuMaster	7575.0	20450.0	4848.4	8065.6	17448.4	21943.8	12964.1	21956.2	20195.3	23690.6	19718.8	25375.0	18025.0	20365.6	616.5	22736.0
MsPacman	557.3	818.0	1178.8	1685.9	751.2	1146.1	1410.5	1538.9	1277.3	1354.5	866.2	1401.9	777.2	1227.8	235.2	6952.0
NameThisGame	1468.1	1992.7	1826.7	2614.5	1919.8	2377.7	2087.3	2155.2	1994.8	2570.3	2153.4	2471.9	1964.2	2314.8	2136.8	8049.0
Pong	-19.6	-8.5	-17.3	16.7	1.4	21.0	-2.0	6.6	3.8	14.2	-17.9	-2.0	-10.1	21.0	-20.4	15.0
PrivateEye	0.0	98.9	75.0	82.8	76.6	100.0	75.0	96.9	60.9	100.0	96.9	99.3	100.0	4038.7	26.6	69571.0
Qbert	476.6	702.7	555.9	869.9	508.6	802.7	802.3	1721.9	974.6	2322.3	475.0	812.5	668.8	747.3	166.1	13455.0
Riverraid	1416.1	1929.4	1784.4	2274.5	1799.4	2158.4	2053.8	2307.5	2143.6	2221.2	1387.8	1759.8	1345.5	1923.4	1451.0	17118.0
RoadRunner	5901.6	8484.4	781.2	2857.8	2804.7	10676.6	1620.3	4104.7	7032.8	14978.1	857.8	1342.2	2717.2	8560.9	0.0	7845.0
Seaquest	414.4	768.1	236.9	470.6	386.9	497.2	330.9	551.2	332.8	460.9	274.1	317.2	366.9	527.2	61.1	42055.0
UpNDown	1195.9	2071.1	1007.5	1315.2	2389.5	3798.3	1433.3	1622.0	1248.6	1999.4	1670.3	2728.0	1825.2	5193.1	488.4	11693.0
YarsRevenge	3047.0	3380.5	3416.3	4230.8	2435.5	3914.1	2955.9	3314.5	3434.8	3896.3	2745.3	2848.1	4276.3	6673.1	3121.2	54577.0

Table 5: Models comparison. Scores of median (left) and best (right) models out of five training runs. Right most columns presents score for random agent and human.