Behavioral Cloning in Atari Games Using a Combined Variational Autoencoder and Predictor Model

Brian Chen[§]
Department of Mathematics
University of Michigan
Ann Arbor, USA
chenbri@umich.edu

Siddhant Tandon[§]
Department of Aerospace Engineering
University of Michigan
Ann Arbor, USA
tandons@umich.edu

David Gorsich
Ground Vehicle Systems Center
U.S. Army DEVCOM
Warren, USA

Alex Gorodetsky
Department of Aerospace Engineering
University of Michigan
Ann Arbor, USA

Shravan Veerapaneni
Department of Mathematics
University of Michigan
Ann Arbor, USA

2021 IEEE Congress on Evolutionary Computation (CEC) | 978-1-7281-8393-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/CEC45853.2021.9505001

***Abstract*—We explore an approach to behavioral cloning in video games. We are motivated to pursue a learning architecture that is data efficient and provides opportunity for interpreting player strategies and replicating player actions in unseen situations. To this end, we have developed a generative model that learns latent features of a game that can be used for training an action predictor. Specifically, our architecture combines a Variational Autoencoder with a discriminator mapping the latent space to action predictions (predictor). We compare our model performance to two different behavior cloning architectures: a discriminative model (a Convolutional Neural Network) mapping game states directly to actions, and a Variational Autoencoder with a predictor trained separately. Finally, we demonstrate how we can use the advantage of generative modeling to sample new states from the latent space of the Variational Autoencoder to analyze player actions and provide meaning to certain latent features.**

***Index Terms*—Behavior Cloning, Variational Autoencoder, Predictive Models, Video Games**

I. Introduction

Our motivation for data-efficient and interpretable cloning approaches stems from military applications seeking to understand soldier behavior in gamified environments. Gaming engines have recently been used by the military [1] to determine the operational effectiveness of various technologies for various missions. Large amounts of data is collected with hundreds of soldiers in-the-loop, but it is often hard to analytically assess how these soldiers changed their operations within the environment. It is also hard to determine why some soldiers do better than others without doing surveys and interviews after experiments. On the other hand, it may be possible to extract this information from gaming data using ideas from machine learning (ML). Moreover if their actions can be learned through ML, it may be possible to train autonomous agents to play in the games to mimic these actions and behaviors.

One such research area within ML that utilizes game data is called 'learning from demonstration' (LfD) or better known as imitation learning [2]. A simple form of imitation learning called behavior cloning [3], used primarily in self-driving cars [4], is a unique method that tries to imitate the task behavior of a human player or an artificial agent. It takes an input tuple of states and actions, (s, a), generated by the player and learns to copy the player's behavior. The goal of a behavior cloned model is to determine what a player would do in new and unseen situations, specifically, to generate the same actions that the player would have performed. Similarly, inverse reinforcement learning methods [5] are another class of imitation learning methods, where the approach is to first learn a reward function and derive a policy from that reward function. However, we choose to take a more direct approach and use behavior cloning to directly learn a player's policy.

In general, behavior cloning requires high-quality gameplay data. Data collection efforts must ensure to repeatedly expose players to a variety of environments and conditions, a time and cost-expensive proposition. For example, to collect data on a new set of states S, players would need to be invited to generate the corresponding action set A. Moreover, the amount of scenarios that are possible in complex games may require an exponential growth in data requirements. Ideally, it would be advantageous if we had a model that is able to determine the abstract features a player uses to perform actions. If the number of such features is low, it is potentially

We acknowledge support from the Automotive Research Center (ARC) in accordance with Cooperative Agreement W56HZV-19-2-0001 with U.S. Army DEVCOM Ground Vehicle Systems Center.

[§]Equal contribution from authors, ordered alphabetically.

possible to clone a player with high accuracy and using limited data.

Our aim is to develop a cloning architecture that (1) attempts to derive such abstract features to enable future analysis and predictions and (2) is able to accurately clone a player and recover their actions in unseen environments. Our emphasis is also on how well a behavior cloned model imitates the player, rather than how well the imitating clone plays the game. Indeed, we would like to use the proposed approaches for both expert and non-expert players.

A. Past Work in Behavior Cloning For Games

MineRL [6], Atari Grand Challenge [7] and Atari-HEAD [8] are a few examples of behavior cloning research in games that use human played game data. In Kanervisto *et al.* [9], the authors use the Atari Grand Challenge and Atari-HEAD datasets to train a behavior cloning model. They compare the rewards acquired by cloned models against those of human players, and report an overall poor performance of behavior cloning with the cloned models achieving only a fraction of the human scores across various games. However, the aforementioned behavior cloning techniques have not explicitly investigated how well the cloned model replicates a player's behavior.

B. Contributions

Our goal is to design a behavior cloning algorithm to extract optimal strategies from a player, with the following criterion:

- Train solely off of trajectory (gameplay) data with further interactions only used for evaluation.
- Extract salient features for predicting the actions taken by players.

As such, we propose an architecture combining a Variational Autoencoder (VAE) [10] combined with an action predictor. We show that such an architecture provides reasonable imitation accuracy compared to a discriminative model. Furthermore, as a generative model, one advantage of this architecture is that it allows for the generation of new unseen states as well as the predicted actions for those states, which may be helpful for understanding and interpreting the behavior of the player.

VAEs have been used in other instances of imitation learning, but for different purposes. For instance, Wang *et al.* [11] uses a VAE that takes in full sequences of states and actions and tries to learn a low dimensional policy representation, which can then be used to generate actions. In another example, Brown *et al.* [12] uses a VAE with some additional auxiliary cost functions in order to extract features that are then used to estimate a reward function. Like Brown *et al.*, our VAE takes in stacked frames, but instead of using the latent space to estimate a reward function, we directly use it to predict the actions taken by the player.

Following the discussion above, in this paper we will first introduce the inputs we generate for our behavior cloning model in Section II. After that, in Section III we discuss in detail the architecture of the Variational Autoencoder we use for behavior cloning. Next, we define our training and testing regime in Section IV with comparison against other behavior cloning models and finally, we demonstrate our behavior cloning results in Section V.

II. Data Generation

In this section we describe the generation and structure of the data used for behavior cloning.

As discussed in Kanervisto *et al.* [9], the human datasets available on Atari games are a promising place to start behavior cloning research. However, the available human datasets on Atari games do not provide information on which gameplay trajectories belong to a single human player and provide no method for allowing such classification to occur in the dataset. Our behavioral cloning approach relies on multiple trajectories of actions and processed game states. Since we are cloning single players, these trajectories must also arise from single players' actions. To generate sufficient data to test our approach in this paper, we use an AI player that is trained using the RL Baselines Zoo package [13].* The player is represented as a Deep Q-Network (DQN) [15], which is loaded from the RL Baselines Zoo package and trained on an additional 10^7 samples (states and actions) of each Atari game using the default parameters provided by the package. RL Baselines Zoo provides pre-trained models for the following Atari games: BeamRider, Breakout, Enduro, MsPacman, Pong, Qbert, Seaquest, and Space Invaders. We test our approach on all of these games.

Raw game states are represented as RGB frames of size 210×160. These game states are then processed through five steps: gray scale conversion; downsampling to image sizes of 84×84; taking the component-wise maximum over two consecutive frames; considering only every fourth state and skipping the states in between; and frame stacking. The purpose of taking the maximum over two consecutive frames is to mitigate flickering that occurs in Atari games, while frame skipping is used for data reduction. Prior to the frame stacking pre-processor, the states are gray-scale 84×84 images. We then stack the last four un-skipped states to arrive at an observation of shape $84 \times 84 \times 4$. Examples of single downsampled frames from all the games are shown in Fig. 1.

In addition to pre-processing raw game states, we modify player actions during trajectory generation. Specifically, we incorporate sticky actions, as recommended in M.C. Machado *et al.* [16]. This approach introduces randomness into the action space to increase the range of system states available for training. Randomness is introduced by encoding a 25% chance of repeating the previous action instead of using the chosen action. However, we keep track of both the action selected by player and the taken action, so that during training we can train directly off the selected actions.

Trajectories for the training data are generated beginning from one of three initial conditions: (1) the standard initial condition of the game; (2) after applying 75 initial random actions; or (3) after applying 100 initial random actions. The

*RL Baselines is a package consisting of pre-trained reinforcement learning agents and tuned hyperparameters for further training. This training is done using the Stable Baselines platform [14].

Fig. 1. Downsampled single grayscale frames from the eight Atari games we test on. From left to right, top to bottom, these games are: BeamRider, Breakout, Enduro, Ms.Pacman, Pong, Qbert, Seaquest, and SpaceInvaders.

purpose of these random actions is to initialize the AI player in a wide variety of conditions to more adequately explore the player's actions in wider range of game conditions.

III. Cloning Architecture and Learning Objective

In this section we describe the architecture of our imitator. It consists of a VAE with an action predictor. Our architecture is that of a predictive generative model that aims to both perform dimension reduction as well as minimize errors of our prediction. It consists of

1) A traditional VAE whose encoder has three convolutional layers, one fully connected layer; a latent space of N variables distributed independently as Gaussians with mean $z_{mean} \in \mathbb{R}^N$ and log-variance $z_{logvar} \in \mathbb{R}^N$; and a decoder with a single fully connected layer and three deconvolutional layers.
2) A prediction model mapping N latent variables through two fully connected layers into the set of actions.

The inputs to the autoencoder are the $84 \times 84 \times 4$ processed game states, and the output of the prediction is a vector of probabilities over actions. This architecture is shown in Fig. 2.

We choose a three-component loss function $L(\alpha)$ that complements the two-part architecture by balancing the standard Evidence Lower Bound (ELBO) loss L_{VAE} [10] with a cross entropy loss L_{pred} for the discrete action prediction, plus a L2 regularization term

$$L(\alpha) = L_{VAE} + \alpha L_{pred} + \lambda \|\omega\|_2,$$

where α is a hyperparameter balancing the first two components of our loss function, ω is the vector of weights in our network, and λ is the coefficient on the L2 regularization penalty. For our experiments, we use $\alpha = 1000$ to get L_{VAE} and L_{pred} on roughly the same order of magnitude. The values of λ are shown later, as those values are chosen by a tuning procedure to be described later.

We will refer to this generative model architecture as the 'Combined VAE/Predictor'.

IV. Comparisons, Training Approach, and Accuracy Computation

A. Generative vs Discriminator

We will compare our results to two alternative architectures: A 'Separate VAE/Predictor' training approach and a discriminative model based on a 'Convolutional Neural Network' (CNN).

a) Separate VAE/Predictor: Separate training of the VAE and the predictor has an advantage of simplicity in treating the two loss functions L_{pred} and L_{VAE} separately, without tuning a balancing hyperparameter α. We will show that this approach results in worse prediction accuracy, which we believe is due to the features from the VAE no longer being tailored to action prediction.

b) Convolutional Neural Network (CNN): Our CNN comparison architecture is based off the known neural network architecture used by the AI player (which generates the game data for training) to determine the value of different actions at different states (e.g., the state-action value function) [15]. Since the goal is to learn the action choices made by such a player, we expect this model to perform well for the prediction task. This is, effectively, a heavy advantage for the CNN. However, we will see that it does not yield significantly better predictions than our approach.

Through these comparisons, we also reinforce that our approach uses a generative model because of our motivation to analyze and interpret the actions of players. For instance, we can now generate new system states and cluster predicted player actions in an attempt to extract salient gameplay features that are critical for decision making.

B. Training

Each architecture is trained using the Adam optimizer [17] with cosine decay learning rate schedule (without warm restarts) [18]; for 20 epochs; and using a batch size of 32. For each game, the data set consists of 200 trajectories from the standard initial game state, 50 trajectories starting from 75 random initial actions, and another 50 trajectories starting from 100 random initial actions. 80% of these trajectories are used for training while the rest is kept as a separate validation set. The validation set is used for hyperparameter tuning. The corresponding number of stacked frame / action pairs can be found in Table I.

TABLE I
Number of Stacked Frames Used in Training

Game	*Training*	*Validation*
BeamRider	163583	40768
Breakout	33973	8954
Enduro	816334	207382
MsPacman	93344	21402
Pong	581119	149462
Qbert	60438	15120
Seaquest	134142	35456
SpaceInvaders	95489	20448

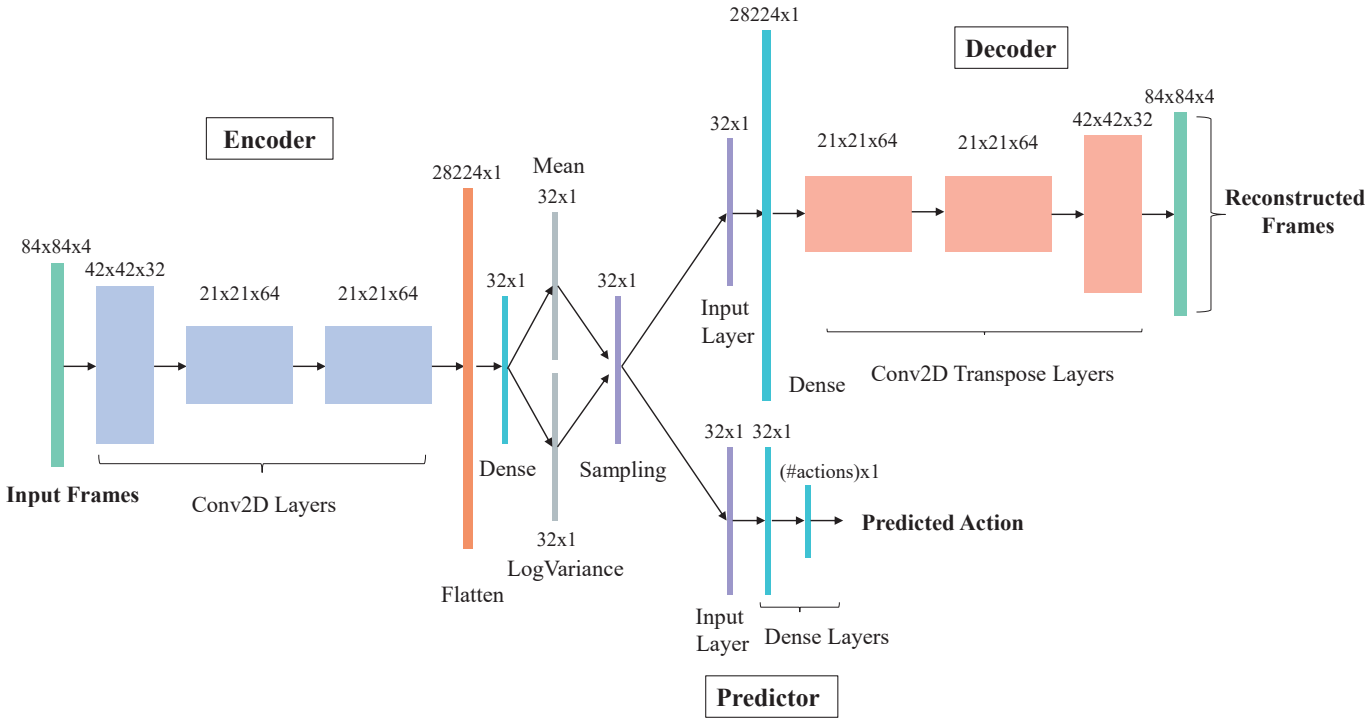


Fig. 2. Proposed Combined VAE/Predictor architecture. The Encoder takes in 4-stack input frames, passes it through a sequence of convolutional layers, a flattening layer, and a dense layer to produce two vectors of length 32, which we call z_{mean} and z_{logvar}. Then, we take a sample from a multivariate normal distribution with the above mean and log variance. This sample is passed separately into both a decoder and a predictor. The decoder consists of a dense layer followed by Transposed Convolution layers to produce a reconstructed frame. The predictor consists of two dense layers to produce a vector of probabilities assigned to each action the agent can take.

The learning rates and regularization penalties can be found in Table II. For the Combined VAE/Predictor and CNN, the learning rates and L2 regularization penalty coefficient λ were chosen after performing a parameter sweep on Ms. Pacman, with a goal of maximizing validation accuracy. These hyperparameters were then applied to the other Atari games. For the Separate VAE/Predictor, we first conducted a parameter sweep for the separate VAE on Ms. Pacman to minimize validation reconstruction loss. Afterwards, the VAE was then used to encode frames, which were then used as inputs into the predictor. Hyperparameters were then chosen for the separate predictor by performing a parameter sweep on Ms. Pacman, with the goal of maximizing validation accuracy. The one exception for this is the CNN for Pong, where the hyperparameters determined above were not suitable and required custom tuning - these are also displayed in Table II.

As mentioned before, our dataset consists of 300 trajectories, where 240 are used as training data. Fig. 3 shows the validation accuracy as a function of the number of training trajectories for Breakout. Validation accuracy increases by about 4% moving from 240 training trajectories to 900. In general, we find that increasing the number of trajectories allows us to converge the imitator for most games. We decided to keep the number of trajectories low, though, since in practice we may only have access to a limited number of demonstrations.

TABLE II
HYPERPARAMETERS

Imitator	*Initial Learning Rate*	λ_{L2}
Combined VAE	10^{-3}	10^{-2}
CNN	10^{-3}	10^{-4}
Separate VAE	10^{-4}	10^{-6}
Separate Predictor	10^{-2}	10^{-6}
Pong (CNN)[a]	10^{-4}	10^{-8}

[a]Hyperparameters for Pong (CNN) determined after additional tuning.

We used the Google Cloud Platform (GCP) to train all the architectures. GCP has many advantages for training machine learning algorithms through its ready to use Virtual Machines (VMs) and flexible compute power options. We utilized the Deep Learning VM, to support training with TensorFlow[†]. The architecture and training guide is maintained on a github repository [‡].

C. Imitator Accuracy

To determine the accuracy of our imitators, we play through the Atari games three hundred times, where we take actions

[†]https://www.tensorflow.org/
[‡]https://github.com/bchen0/vae_behavior_cloning

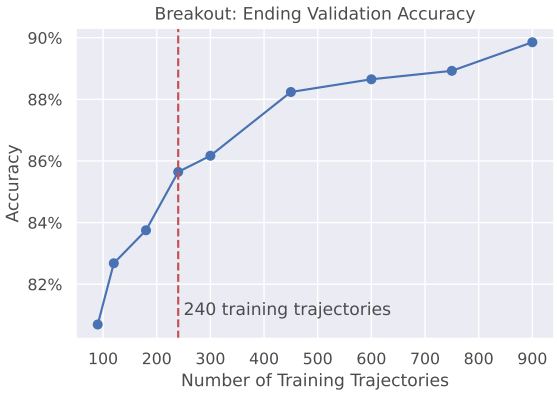


Fig. 3. Ending validation accuracy of a Combined VAE/Predictor trained on Breakout, where the number of trajectories in the training set is changed. Note that we measure the validation accuracy for these models on the same validation set. The dotted red line corresponds to the number of trajectories we use in the rest of our experiments. We note that increasing the number of training trajectories improves the performance of the Combined VAE/Predictor but that the accuracy starts to converge. For our experiments, we decided to keep the number of training trajectories fixed at 240.

according to the AI player. At each time step, we compare the action chosen by the AI player to the actions that would have been chosen by the imitators. We then compute the percentage of time these actions agree for each game, and average the accuracies over the three hundred trajectories. Note that in gameplay, we still implement sticky actions, but the imitator actions are compared to the actions the AI player would have taken (instead of the actually taken action). More details can be found in Algorithm 1.

We perform this testing procedure over two conditions:

- Gameplay where two-thirds (200) of the games start at the default starting state, one-sixth (50) start after 75 random initial actions, and one-sixth (50) start after 100 random actions. This matches the starting conditions for the training data. We will refer to these results as "Interpolatory".
- Gameplay where all of the games (300) start after 150 random initial actions. Because these conditions were not encountered in the training data, we will refer to these as "Extrapolatory".

V. Results and Discussions

In this section, we evaluate and compare the performance of our proposed cloning approach with the alternatives described earlier. Table III summarizes our findings.

We note that in most games, the CNN performs the best in terms of accuracy, with the Combined VAE/Predictor trailing slightly. However, as mentioned before, the CNN is based off the architecture used to generate the player's action preferences; as such, it is not surprising that the CNN achieved higher accuracies than the Separate VAE/Predictor. The fact that the differences are so small allows us to gain the advantage of obtaining a generative model without sacrificing accuracy.

Algorithm 1 Accuracy Evaluation

Inputs: Player and Imitator models, which take states s in state space S and returns actions a in action space A, Game environment, K = number of trajectories, M = number of initial random actions, p = sticky action probability

accuracylist = [ ] {initialize empty list to store accuracies over each trajectory}
for $i = 1$ **to** K **do**
  Take M random actions and let $s \leftarrow$ state of game after these random actions
  $a_{\text{prev}} \leftarrow$ None {stores previous action}
  while trajectory is not done **do**
    $a_{\text{player}} \leftarrow \text{player}(s)$ {player's chosen action}
    $a_{\text{imitator}} \leftarrow \text{imitator}(s)$ {imitator's chosen action}
    if a_{prev} is not None, with probability p: **then**
      $a_{\text{next}} \leftarrow a_{\text{prev}}$ {a_{next} is the next action}
    else
      $a_{\text{next}} \leftarrow a_{\text{player}}$
    end if
    Take action a_{next} and let $s \leftarrow$ state after the action
    $a_{\text{prev}} \leftarrow a_{\text{next}}$
  end while
  accuracy $\leftarrow$ percent of time $a_{\text{player}} == a_{\text{imitator}}$
  Append accuracy to accuracylist
end for
return Mean of accuracylist

The Separate VAE/Predictor generally results in quite poor accuracy. We hypothesize that this is due to the VAE extracting features that are useful for reconstruction but not necessarily useful for action prediction, once the VAE and predictor are separated.

The accuracy achieved by both the Combined VAE and CNN is generally high, although it varies by some amount between games. It is likely that training on a larger number of training trajectories can improve the accuracy of the predictors. As seen in Fig. 3, increasing the number of training trajectories for Breakout from 240 to 900 led to a roughly 4% increase in accuracy.

An additional point of note is the difference between interpolatory accuracy and extrapolatory accuracy. As described earlier, the extrapolatory accuracy results are measured on gameplay where the first 150 actions made are random, which is a starting condition that is not encountered in the training data. For some games, like Enduro, Pong, and BeamRider, the difference between interpolatory and extrapolatory accuracy is minimal. On the other hand, the drop off is much larger in games like SpaceInvaders and Qbert. A potential explanation for this is differences in game complexity, where there may be more substantial gameplay-relevant differences between the interpolatory and extrapolatory datasets for certain games (e.g., we may expect Pong to only have a few salient features for gameplay whereas SpaceInvaders has more moving parts as well as progression as the enemies move down the screen).

TABLE III
IMITATION ACCURACY OVER THREE HUNDRED TRAJECTORIES.

	Interpolatory Accuracy			**Extrapolatory Accuracy**			**Number of Stacked Frames**	
	CombinedVAE	CNN	Separate VAE	CombinedVAE	CNN	Separate VAE	Interpolatory	Extrapolatory
BeamRider	$76.4 \pm 0.5\%$[a]	$\mathbf{77.9 \pm 0.5\%}$[b]	$53.0 \pm 0.7\%$	$72.0 \pm 0.5\%$	$\mathbf{73.8 \pm 0.5\%}$	$47.4 \pm 0.7\%$	211681	179692
Breakout	$85.8 \pm 1.1\%$	$85.9 \pm 1.0\%$	$54.3 \pm 1.9\%$	$77.8 \pm 1.6\%$	$\mathbf{80.1 \pm 1.5\%}$	$38.1 \pm 3.2\%$	42371	32179
Enduro	$88.2 \pm 0.2\%$	$88.3 \pm 0.2\%$	$67.1 \pm 0.4\%$	$88.5 \pm 0.1\%$	$\mathbf{88.7 \pm 0.1\%}$	$67.8 \pm 0.3\%$	1061197	958378
MsPacman	80.3 ± 1.9 %	$80.5 \pm 1.9\%$	$59.0 \pm 1.6\%$	$72.6 \pm 2.8\%$	$71.0 \pm 3.0\%$	$58.3 \pm 3.7\%$	120429	26056
Pong	$88.2 \pm 0.2\%$	$\mathbf{91.6 \pm 0.2\%}$	$37.2 \pm 0.3\%$	$87.5 \pm 0.2\%$	$\mathbf{90.9 \pm 0.2\%}$	$36.7 \pm 0.4\%$	737625	696298
Qbert	$89.9 \pm 1.7\%$	$90.9 \pm 1.8\%$	$53.0 \pm 2.0\%$	$61.0 \pm 3.1\%$	$58.1 \pm 3.3\%$	$37.3 \pm 2.8\%$	75292	27498
Seaquest	$64.2 \pm 1.5\%$	$\mathbf{67.7 \pm 1.4\%}$	$34.8 \pm 1.4\%$	$53.2 \pm 1.9\%$	$\mathbf{58.3 \pm 1.8\%}$	$24.0 \pm 1.5\%$	174766	174352
SpaceInvaders	$71.0 \pm 2.1\%$	$\mathbf{73.4 \pm 2.0\%}$	$57.2 \pm 2.1\%$	$45.9 \pm 3.2\%$	$45.1 \pm 3.2\%$	$39.1 \pm 3.0\%$	111867	87305

[a]The confidence intervals are +/- two standard errors.
[b]Cases where the accuracy for one model is greater than the two other models by more than two standard errors are bolded.

Investigating this difference and improving the extrapolatory accuracy for these games is left to future work.

A. Generating Synthetic Frames

Despite the slightly lower prediction accuracies, an advantage of our proposed Combined VAE/Predictor methodology versus discriminative models like the CNN is the ability to generate frames as well as predictions for those frames. The ability may be helpful in interpreting the actions of a player, especially when data is limited or it is not possible to generate new trajectories from that player.

To generate synthetic frames, we sample vectors from the latent space, according to a multivariate normal distribution. The regularization term in the VAE is meant to keep the distribution close to a multivariate $N(0, I)$ distribution. However, we found that using the observed means and covariances (while still assuming normality) generally led to better reconstructions. In other words, we found the empirical mean μ and empirical covariance matrix Σ for the latent space and sampled from $N(\mu, \Sigma)$.

As a proof of principle, we show in Fig. 4 synthetic frames from Enduro, where the imitator has a strong preference to move either left ("LEFT", "DOWNLEFT", "LEFTFIRE") or right ("RIGHT", "DOWNRIGHT", "RIGHTFIRE"). As can be seen, the frames where left is predicted tend to have the vehicle on the right side of the road and/or with a left curving road up ahead. We generally see the opposite (left side of road and/or right curving road) for frames where right is predicted, but this is not always the case, as can be seen in the first and third image in Fig. 4b.

We also note that not all reconstructions are perfect. For example, Fig. 5 provides an example of an imperfect reconstruction for Enduro. Furthermore, we do not see all of the important details of the game, such as other cars, in the case of Enduro. In future work, we will try to improve the quality of our sampling procedure in order to better understand the actions being taken by a player.

VI. CONCLUSION

In this paper, we discussed the possibilities of improving behavior cloning performance by using deep generative models like the Variational Autoencoder. We designed a Combined VAE/Predictor to clone gameplay data generated by an AI player, where we use sticky actions and random initial actions to increase the variety of states observed by the player. We compare its performance against two other architectures, namely, a different generative model (Separate VAE/Predictor) and a discriminator (CNN). Through our results, we observed that the Combined VAE/Predictor performs similarly to the CNN in both interpolatory and extrapolatory accuracies. In particular, we find that the Combined VAE/Predictor achieves interpolatory and extrapolatory accuracy within $\pm 3\%$ of the CNN imitator in six of the eight games.

An advantage of the generative model is the ability to sample synthetic frames and predict actions for those frames for aiding interpretability. We demonstrate this proof-of-principle example for Enduro, where it appears that sampled frames that have high predictions for "left" differ qualitatively from those with high predictions for "right". In future work we will explore whether this architecture can be applied to a human dataset. Furthermore, we will use these cloned models to see if there is an explainable behavior that can be extracted from different players (of varying skill levels), in order to learn the strategies they use.

REFERENCES

[1] D. J. Gorsich, "The use of gaming engines for design requirements," *Proceedings of the Design Society: DESIGN Conference*, vol. 1, p. 141–146, 2020.

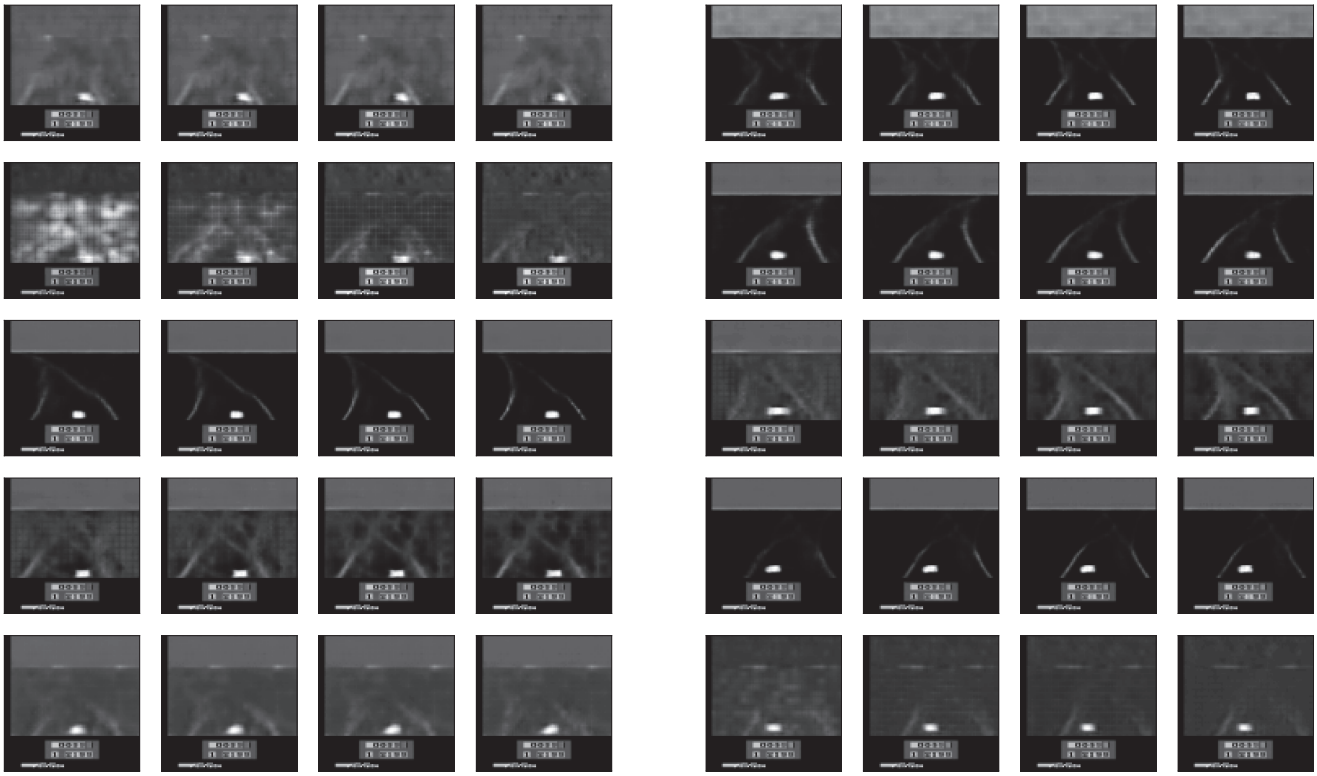
[2] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Comput. Surv.*, vol. 50, no. 2, Apr. 2017. [Online]. Available: https://doi.org/10.1145/3054912

[3] I. Bratko, T. Urbančič, and C. Sammut, "Behavioural cloning: Phenomena, results and problems," *IFAC Proceedings Volumes*, vol. 28, no. 21, pp. 143 – 149, 1995, 5th IFAC Symposium on Automated Systems Based on Human Skill (Joint Design of Technology and Organisation), Berlin, Germany, 26-28 September. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667017467164

[4] U. Sumanth, N. S. Punn, S. K. Sonbhadra, and S. Agarwal, "Enhanced behavioral cloning based self-driving car using transfer learning," 2020.

[5] A. Y. Ng, S. J. Russell *et al.*, "Algorithms for inverse reinforcement learning." in *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000, pp. 663–670.

[6] W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. Veloso, and R. Salakhutdinov, "Minerl: A large-scale dataset of minecraft demonstrations," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International



(a) VAE chooses left

(b) VAE chooses right

Fig. 4. Above are examples of frames for Enduro that are generated using our CombinedVAE, where the predictor has a strong preference towards "left" or towards "right". In particular, we generated 1000 samples and chose the top five stack of frames that had the highest prediction probability on moves containing left ("LEFT", "DOWNLEFT", "LEFTFIRE") and five stacks that had the highest probability on moves containing right ("RIGHT", "DOWNRIGHT", "RIGHTFIRE". Note that one frame is removed from the "Right" group since it did not produce a clear image - this frame is shown in Fig. 5. The generation process of these frames is described in Section V-A

Fig. 5. Example of poor reconstruction in Enduro.

Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 2442–2448. [Online]. Available: https://doi.org/10.24963/ijcai.2019/339

[7] V. Kurin, S. Nowozin, K. Hofmann, L. Beyer, and B. Leibe, "The atari grand challenge dataset," 2017. [Online]. Available: http://arxiv.org/abs/1705.10998

[8] R. Zhang, C. Walshe, Z. Liu, L. Guan, K. Muller, J. Whritner, L. Zhang, M. Hayhoe, and D. Ballard, "Atari-head: Atari human eye-tracking and demonstration dataset," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, pp. 6811–6820, Apr. 2020. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/6161

[9] A. Kanervisto, J. Pussinen, and V. Hautamäki, "Benchmarking end-to-end behavioural cloning on video games," in *2020 IEEE Conference on Games (CoG)*. IEEE, 2020, pp. 558–565.

[10] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2014. [Online]. Available: http://arxiv.org/abs/1312.6114

[11] Z. Wang, J. S. Merel, S. E. Reed, N. de Freitas, G. Wayne, and N. Heess, "Robust imitation of diverse behaviors," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/044a23cadb567653eb51d4eb40acaa88-Paper.pdf

[12] D. Brown, R. Coleman, R. Srinivasan, and S. Niekum, "Safe imitation learning via fast bayesian reward inference from preferences," in *International Conference on Machine Learning*. PMLR, 2020, pp. 1165–1177.

[13] A. Raffin, "Rl baselines zoo," https://github.com/araffin/rl-baselines-zoo, 2018.

[14] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

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

[16] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling, "Revisiting the arcade learning environment: Evalua-

tion protocols and open problems for general agents," *J. Artif. Int. Res.*, vol. 61, no. 1, p. 523–562, Jan. 2018.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[18] I. Loshchilov and F. Hutter, "SGDR: stochastic gradient descent with warm restarts," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=Skq89Scxx