Deep Variational Autoencoders for NPC Behaviour Classification

Everton Schumacker Soares
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada
schumack@ualberta.ca

Vadim Bulitko
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada
bulitko@ualberta.ca

Abstract—**Procedural content generation (PCG) can create novel, player-specific content in video games, including behaviours of AI-controlled non-playable characters (NPC). Here we present our first results on comparing unsupervised and supervised machine learning for procedurally generated NPC behaviours. Using an artificial life environment as a stand-in for a video game, we run artificial evolution and generate AI agents with various behaviours. We then train deep variational autoencoders on commonly evolved behaviour and measure its efficacy in detecting behaviours unseen during training. As a reference, we use an off-the-shelf deep network trained in a supervised manner to detect behaviours both seen and unseen during its training. Preliminary results demonstrate promising performance that holds even when the training set contains a mixture of several types of behaviours without proper labels.**

Index Terms—**procedural content generation, non-playable-characters, video games, unsupervised machine learning, deep learning, variational autoencoders, anomaly detection.**

I. Introduction

Procedural Content Generation (PCG) [1], [2] can generate video-game levels, narrative, aesthetics and behaviours of non-playable characters (NPCs) [3], [4]. It has the potential to produce novel content and thus enable new types of gameplay in video games (e.g., exploring procedurally generated uncharted alien worlds in *No Man's Sky* [5]). On the down side, procedurally generated content can move beyond the designer's intentions and thus have a negative effect on the player's experience. Thus, it becomes important to automatically classify procedurally generated content – in our case procedurally generated NPC behaviours. Such a classifier can be applied for quality assurance to distinguish expected behavior from bugs or for novelty detection to flag behaviours potentially surprising to the player [6].

In this paper we describe our on-going work on classifying NPC behaviours automatically by training a behaviour classifier. It can then be used to inform PCG techniques and/or game designers when interesting/surprising/unacceptable NPC behaviour emerges. Since we are after detecting previously unseen, novel behaviours we employ unsupervised machine learning in the form of deep variational autoencoders [7]–[9]. In this work in progress we use a simple artificial life simulation [6], [10] in place of an actual video-game engine. This is done for rapid prototyping and allows us to quickly modify both the behaviour generator and the detector. Preliminary results demonstrate promising performance of variational autoencoders that hold even when the training set contains a mixture of several types of behaviours without the corresponding labels.

II. Problem Formulation

The problem is to automatically classify procedurally generated NPC behaviour. We impose the following requirements: (i) the classifier is tuned for a given game automatically via machine learning on training data collected from the game, (ii) the training data contains representatives of only a single class, (iii) the classifier takes easily observable data (e.g., a screen-capture stream) and does not require instrumenting the game engine or accessing its internal states, (iv) the training process is robust to inclusion of trace amounts of data from another class into the training set. The performance of such a classifier is then measured via its accuracy: the percentage of data correctly labeled.

III. Related Work

Player behaviour detection is of interest in on-line games where game modifications, bot use and other anomalous behaviours are considered cheating. For instance, machine learning was used [11] to detect gold farmers in *EverQuest II* [12]. Their approach relies on feature extraction from game logs and may be sensitive to training data pollution (i.e., presence of mislabeled training data).

Detecting anomalous behaviour is a related binary-classification task. Techniques based on distance, clustering and density are common methods for outlier and anomaly detection [13]. Our previous work showed how supervised deep learning can outperform density-based approaches and clustering algorithms (e.g., k-nearest neighbors) when dealing with screen-capture images taken in an A-life environment or a video game [10]. Since a screen-capture image contains positional and morphological information about a population of several NPCs, crafting a distance function to detect outliers poses a challenging problem. For the same reason manually creating descriptive models for data can be challenging. Thus, we here propose to use unsupervised autoencoders as a model-based approach capable of learning a compression that best represents images from a given class. NPC behaviours from a different class can then be detected by measuring how well the learned compression applies to a given image.

Beyond games, a social-force model was used to detect anomalous crowd behaviour in image frames from surveillance cameras [14]. Although this technique is capable of detecting population-level anomalies, it relies on a specific particle motion pattern limiting the set of behaviours it can detect. More recently, autoencoders were used to detect anomalies in crowd behaviours from video footage [8]. However, their anomalies manifested itself in an unusual shape of the individual agents (e.g., a bicycle versus pedestrians). This reliance limits applicability of the approach. For instance, agents may not differ in their shape in the visualization yet behave anomalously [10].

IV. Proposed Approach

Given the constraints of our problem, we use autoencoders [7], [8]. Specifically, we focus on a binary classification task and train a variational autoencoder (VAE) [9] to tell anomalous/unusual/novel behaviours from typical behaviours. While the technique can be applied to a broad variety of game and simulation environments we evaluate it in a simple artificial life (A-life) environment adopted from published studies [6], [10]. This stand-in for an actual video game enables rapid prototyping and evaluation.

PCG in A-life. The environment is a 2D grid (Figure 1, left). The prey agents ("rabbits") eat a re-growing resource ("grass"). The predator agents ("wolves") eat rabbits. An agent's behaviour policy is determined by a set of weights encoded in its genes. On each time step an agent computes utility of each grid cell within its sight radius. The utility is a sum of cell contents (e.g., the amount of grass or the number of wolves in the cell) weighted by the agent's genetic weights (e.g., its affinity to grass or its dislike of the wolves). The agent then moves towards the cell with the highest utility. Agents spend energy moving around and replenish it by eating. They die when their energy drops below a minimum. A sufficiently old agent with enough energy will produce an offspring which inherits the parent's genes perturbed with random noise (mutation). The simulated Darwinian evolution procedurally generates wolves and rabbits. A similar process has been used in recent video games to generate enemy NPCs [3], [4].

On a typical evolution run, rabbits evolve to like grass and dislike wolves while wolves evolve to like rabbits. On some evolution runs, unusually behaving agents emerge. For instance, the leader-follower behaviour is an emergence of a large number of long-living yet grass-disliking rabbits [10]. It can happen when two types of rabbits evolve: those who dislike grass but like other rabbits (followers) and those who like grass (leaders). Since any rabbit automatically eats grass in its current grid cell, grass-disliking rabbits can survive by following grass-liking rabbits to grass-rich areas of the environment. While followers usually appear in small numbers on many runs, an emergence of follower rabbits in a large proportion (e.g., two thirds of the population) happens rarely.

Our behaviour classifier takes a stream of visualization frames produced by the A-life. Temporal patterns are important to define a behaviour. To avoid dealing with time explicitly we capture such patterns in a single image by applying a moving average with an exponential decay to the frames (Figure 1, center). A similar automated process can be applied to a screen-capture stream from a video game.

Variational Autoencoders. Notationally $\boldsymbol{x} \in \mathbb{R}^k$ represents an input datum (i.e., an averaged frame, Figure 1, center). As a subclass of autoencoders, VAE is a neural network composed of an encoder and a decoder trained to reconstruct $\boldsymbol{x}$ as $\boldsymbol{x'}$. The encoder is a function that maps the input image $\boldsymbol{x}$ to two lower-dimensional variables: the mean $\boldsymbol{\mu}(\boldsymbol{x})$ and the variance $\boldsymbol{\sigma}(\boldsymbol{x})$, both vectors in $\mathbb{R}^m$, $m \ll k$. A mixture of convolutional and fully connected layers is used to implemented the encoder. Then the VAE computes the code $\boldsymbol{z}(\boldsymbol{x}) \in \mathbb{R}^m$ by linearly combining $\boldsymbol{\mu}(\boldsymbol{x})$ and $\epsilon \cdot \boldsymbol{\sigma}(\boldsymbol{x})$ where ϵ is a scalar random variable drawn from $\mathcal{N}(0,1)$. The code is then decoded back to $\boldsymbol{x'} \in \mathbb{R}^k$ by the decoder.

A VAE is trained so that its output $\boldsymbol{x'}$ is a close reconstruction of its input $\boldsymbol{x}$. Thus, we use gradient-descent backpropagation with the loss function given by the linear combination: $\mathcal{L}(\boldsymbol{x},\boldsymbol{x'}) + \mathrm{KL}(\mathcal{N}(\mu(\boldsymbol{x}),\sigma(\boldsymbol{x}))|\mathcal{N}(0,1))$ where $\mathrm{KL}(P|Q)$ is the KL-divergence [15] between the probability distributions P and Q. $\mathcal{L}(\boldsymbol{x},\boldsymbol{x'})$ is VAE reconstruction error, here defined as the pixel-wise mean square error: $\mathcal{L}(\boldsymbol{x},\boldsymbol{x'}) = \frac{\sum_{j=1}^{W}\sum_{i=1}^{H}(x_{ij}-x'_{ij})^2}{W \cdot H}$ where x_{ij} is the pixel in row i and column j in the input image $\boldsymbol{x}$ with the width of W pixels and the height of H pixels. Likewise, x'_{ij} is the corresponding pixel in the image $\boldsymbol{x'}$ reconstructed by the VAE.

Behaviour Classification with VAE. As the VAE learns to reconstruct training data by passing a high-dimensional image through a lower dimensional code, it is likely to learn visual patterns in its input images. If such patterns in visualization of normal behaviours (which the training data mostly consists of) are different from patterns specific to visualization of anomalous behaviours (which were mostly absent in the training data) then one may assume that the reconstruction error $\mathcal{L}(\boldsymbol{x},\boldsymbol{x'})$ will be higher for images of anomalous behaviour. Thus, we can turn the VAE into an anomaly detector (i.e., a binary classifier) by comparing its reconstruction error against a threshold θ as follows: the input image $\boldsymbol{x}$ is classified as normal if $\mathcal{L}(\boldsymbol{x},\boldsymbol{x'}) \leq \theta$ and anomalous if $\mathcal{L}(\boldsymbol{x},\boldsymbol{x'}) > \theta$.

The threshold θ has to be tuned to maximize the VAE accuracy. The tuning process is complicated by the fact that labeled anomalous data are assumed unavailable (Section II). Thus we tune θ using automatically synthesized images of a *surrogate* anomaly. To create such a surrogate-anomaly set, we take a separate set of normal images and perturb each image by shuffling its pixels according to a random permutation (Figure 1, right). Note that such synthetic images are likely to be substantially different from naturally occuring anomalies. We then use the two image sets to select the threshold θ for an already trained VAE. The candidate values of θ run the set $\theta_i = Q_3 + \frac{i}{2}\text{IQR}$, $i \in \{0, 1, \ldots, 20\}$. Here $\text{IQR} = Q_3 - Q_1$, Q_1 is the 25th percentile and Q_3 is the 75th percentile [16] of the VAE's reconstruction errors on its training set, computed after its training. We choose the θ_i which maximizes the VAE's accuracy in classifying the surrogate anomalies.

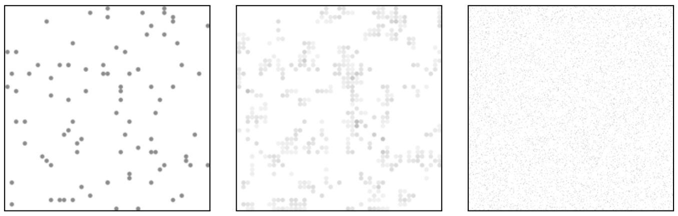
Fig. 1. Representative of a static normal frame from our predator-prey A-life environment (left), its weighted moving average (center), and the corresponding surrogate image (right).

V. Empirical Evaluation

We define three classes of agent behaviours in our A-life as follows. *Leaders* (L) indicates a state of the simulation in which there are at least twice as many leader rabbits as there are follower rabbits. This is the most common state and is considered the normal behaviour. *Followers* (F) is a state of the simulation in which there are at least twice as many followers as there are leaders. This happens much less frequently and thus constitutes an anomaly. Finally, *random* (R) is another type of anomaly where at least half of the agents in the environment have random behaviour policies.[1] Figure 2 shows representative images from the three classes.

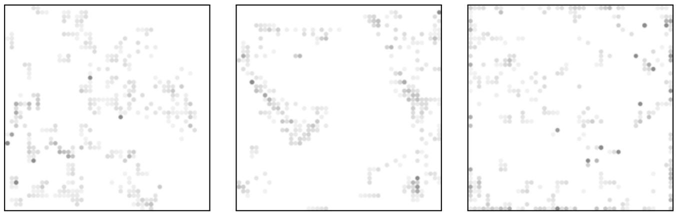
Fig. 2. Representatives of classes L, F and R, respectively from left to right.

Using A-life parameters from prior work [10] we conducted 600 evolutions runs. On each run we detected L, F and R classes and stored only such images, discarding all other images. Of the 600 evolution runs, 281 were used to produce training image sets: L^{train} for the class L (leaders) and F^{train} for the class F (followers). We balanced the image sets by randomly discarding images from larger sets.

[1]For our experiments we needed two different types of anomalous behaviours yet only one anomaly had been observed to date in the A-life environment. So we created the second anomaly artificially. Note that classes L and F emerge naturally in the course of A-life evolution and were previously reported [10]. However, class R is very unlikely to emerge naturally since rabbits with random movement policies do not survive for long. Thus, in order to collect enough data for R we artificially injected hand-built random rabbits into the A-life environment.

Additional 94 runs yielded the image-set pair $(L^{\text{val}}, F^{\text{val}})$ used during VAE training for validation (i.e., to detect when to stop the training process). Then additional 100 evolution runs were used for the image set L^{θ} used to compute the threshold θ for the trained VAE (as described earlier). Then additional 125 runs produced the image-set pair $(L^{\text{test}}, F^{\text{test}})$ used to test trained behaviour detectors. Finally, we ran 10 additional evolutions where, with probability 50%, we replaced each rabbit in the starting population as well as each rabbit born during the evolution with a randomly moving rabbit. These 10 runs yielded the image set R^{test}.

Our VAE implementation uses convolutional, fully connected, batch-normalization, max-pooling and transposed convolutional layers implemented in TensorFlow [17]. While we hand-tunned the architecture to the data at hand, we conjecture that a similar architecture would work for other environments.

Experiments. We first considered the case when the training data set consists entirely of images depicting the normal behaviour (i.e., the class L). Our VAE was initialized with weights and biases drawn from $\mathcal{N}(0, 0.1)$. It was then trained on the image set L^{train}. The learning rate followed a stepwise cooling schedule with initial value of 0.0001 and the decay of 0.8 on each epoch. The training had three stopping criteria: (i) a maximum of 100 epochs (maximum training time), (ii) the UP test [18] with $k = 5, s = 3$ (over-fitting), (iii) the absolute difference in the validation error between consecutive batches does not exceed 0.0001 for five times in a row (plateauing). The validation error was computed on the image set L^{val}. Adam optimizer [19] with the batch size of 50 images was used. As with our VAE architecture, the training parameters were tuned for the problem in hand. Once the training was completed, we selected the threshold θ on the image sets (L^{θ}, S^{θ}). After that we evaluated the VAE on the separate test image set $(L^{\text{test}}, F^{\text{test}})$. Image labels were used only to test our VAE after training. The resulting test accuracy is presented in Table I and discussed below.

To put the test accuracy of our VAE in a context, we also trained GoogLeNet [20] to classify $(L^{\text{test}}, F^{\text{test}})$. Being a supervised learning approach its training required both classes to be present and labeled in the training data. Thus, while our VAE was trained only on L^{train}, GoogLeNet was trained on both L^{train} and F^{train}, all accurately labeled. We used a version of GoogLeNet included in the Machine Learning toolbox with MATLAB R2018a. This network was trained with Adam optimizer from the MATLAB toolbox using the same learning schedule as we used for the VAE and the batch size of 50 images. The training stopped either when 100 epochs where reached or when the validation error computed every 20 batches on $(L^{\text{val}}, F^{\text{val}})$ did not improve over the previously smallest validation error three times in a row. Additionally, since GoogLeNet included with MATLAB had 1000 neurons in its output layer and we have only two classes, we replaced the output layer with a new layer of two neurons. To compensate for the lack of ImageNet-based pre-training for the weights leading to the new layer, we increased the layer's

TABLE I
ACCURACY OF BEHAVIOUR CLASSIFIERS ON TEST DATA.

Classifier	**Training set**	**Threshold-selection set**	**Validation set**	$(L^{\text{test}}, F^{\text{test}})$	$(L^{\text{test}}, R^{\text{test}})$
VAE	L^{train}	(L^{θ}, S^{θ})	L^{val}	$86.0 \pm 3.0\%$	$\mathbf{70.2 \pm 4.8\%}$
GoogLeNet	$(L^{\text{train}}, F^{\text{train}})$		$(L^{\text{val}}, F^{\text{val}})$	$\mathbf{97.0 \pm 2.4\%}$	$49.7 \pm 0.4\%$

learning rate ten folds. A version of GoogLeNet initialized with random weights on all layers yielded similar results.

Results. The means and the standard deviations of test accuracy were measured over four trials and are listed in Table I. The trials differed in the initialization of the VAE's weights and biases, GoogLeNet weights for the last layer and the randomization of the training process using Adam optimizer. However all image sets stayed the same on all trials — a limitation to be addressed in the future work.

GoogLeNet had a mean test accuracy of 97% when detecting the follower anomaly (class F). This confirms that supervised learning can be a viable behaviour detector when labeled samples of anomalous behaviour are available during training [10]. However, when tested on the random anomaly (class R) missing from the training data, GoogLeNet had the chance-level test accuracy of 49.7%, failing to recognize novel behaviours not seen during its training. On the other hand, the VAE did not require seeing anomalous behaviours during training and thus could detect the F anomaly 86% of the time and the R anomaly 70.2% of the time. We suspect the difference in the test accuracies is due to similarities and dissimilarities between images in L, S, F, R classes. The results suggest viability of VAE as a novel behaviour detector.

We also investigated training the VAE on data that includes anomalous data (without proper labels identifying them as such). We are unable to report the results in detail due to space limitations. Briefly, with 1% of anomalous images mixed in with the normal images, the VAE test accuracy is 81.1% for the F anomaly and 70.5% for the R anomaly.

VI. Future Work & Conclusions

The results presented above are promising but can likely be improved by further tuning VAE architecture and its training parameters (e.g., via neuroevolution [21], [22]). We hand-tuned our approach to maximize classification accuracy and not precision, recall and rank power often used to assess anomaly detection [13]. Our performance on these measures is currently poor. Our previous work ported supervised classification from an A-life environment to a commercial video game [10]. Future work will do so for autoencoders. In particular, we will investigate how well VAE can cope with complex screen images in a video game.

In summary, as progressively more game content is generated procedurally, classifying the resulting content becomes more critical. We presented an approach to classifying NPC behaviours from readily available screen captures. By using unsupervised machine learning we trained such a behaviour classifier automatically and without the need for human-labeled training data. In doing so we extended our previous work [10] and showed that deep autoencoders can detect procedurally generated NPC behaviours not seen during training.

VII. Acknowledgments

We appreciate contributions from the NSERC and Nvidia.

References

[1] J. Togelius, A. J. Champandard, P. L. Lanzi, M. Mateas, A. Paiva, M. Preuss, and K. O. Stanley, "Procedural content generation: Goals, challenges and actionable steps," in *Dagstuhl Follow-Ups*, vol. 6, 2013.
[2] S. Risi and J. Togelius, "Neuroevolution in games: State of the art and open challenges," *TCIAIG*, 2017.
[3] T. Soule, S. Heck, T. E. Haynes, N. Wood, and B. D. Robison, "Darwin's Demons: Does Evolution Improve the Game?" in *European Conference on the Applications of Evolutionary Computation*, 2017.
[4] Polymorphic Games, "Project Hastur," 2018.
[5] Hello Games, "No Man's Sky Next," 2018.
[6] V. Bulitko, S. Carleton, D. Cormier, D. Sigurdson, and J. Simpson, "Towards Positively Surprising Non-Player Characters in Video Games," in *EXAG/AAAI*, 2017, pp. 34–40.
[7] E. Protopapadakis, A. Voulodimos, A. Doulamis, N. Doulamis, D. Dres, and M. Bimpas, "Stacked autoencoders for outlier detection in over-the-horizon radar signals," *Comp. intelligence and neuroscience*, 2017.
[8] M. Ribeiro, A. E. Lazzaretti, and H. S. Lopes, "A study of deep convolutional auto-encoders for anomaly detection in videos," *Pattern Recognition Letters*, vol. 105, pp. 13–22, 2018.
[9] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," *arXiv preprint arXiv:1312.6114*, 2013.
[10] E. S. Soares, V. Bulitko, K. Doucet, M. Cselinacz, T. Soule, S. Heck, and L. Wright, "Learning to recognize A-Life behaviours," in *ACS*, 2018.
[11] M. A. Ahmad, B. Keegan, J. Srivastava, D. Williams, and N. Contractor, "Mining for gold farmers: Automatic detection of deviant players in mmogs," in *Int. Conference on Comp. Science and Engineering*, 2009.
[12] Daybreak Game Company, "EverQuest II," 2004.
[13] K. G. Mehrotra, C. K. Mohan, and H. Huang, *Anomaly Detection Principles and Algorithms*. Springer, 2017.
[14] R. Mehran, A. Oyama, and M. Shah, "Abnormal crowd behavior detection using social force model," in *CVPR*, 2009.
[15] S. Kullback and R. A. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.
[16] Y. Zhao, B. Lehman, R. Ball, J. Mosesian, and J.-F. de Palma, "Outlier detection rules for fault detection in solar photovoltaic arrays," in *APEC*, 2013.
[17] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: http://tensorflow.org/
[18] L. Prechelt, "Automatic early stopping using cross validation: quantifying the criteria," *Neural Networks*, vol. 11, no. 4, pp. 761–767, 1998.
[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[20] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.
[21] A. Rawal and R. Miikkulainen, "From nodes to networks: Evolving recurrent neural networks," *arXiv preprint arXiv:1803.04439*, 2018.
[22] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy *et al.*, "Evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, 2019.