Level Generation for Angry Birds with Sequential VAE and Latent Variable Evolution [8]

Fazilet Cilli
Department of Computer Science
University of California, Irvine
fcilli@uci.edu

Shion Fukuzawa
Department of Computer Science
University of California, Irvine
fukuzaws@uci.edu

Alekhya Pyla
Department of Computer Science
University of California, Irvine
apyla@uci.edu

Vignatha Yenugutala
Department of Computer Science
University of California, Irvine
vyenugut@uci.edu

Abstract

In this project, we re-implement Tanabe et al.'s work on "Level Generation for Angry Birds with Sequential VAE and Latent Variable Evolution"[8] where the authors aim to generate stable, diverse, and playable levels for the game Angry Birds. Video game level generation using deep generative models has been traditionally limited to tile-based level representation. This approach is problematic for games like angry birds that do not have a tiled level representation, leading to the generation of unplayable levels. In the aforementioned paper, this limitation is overcome by adopting a sequential encoding of the level, allowing the processing of the level as text data. The game domain of Angry Birds has been challenging because of its high degrees of freedom as well as the existence and effect of gravity to generate stable and playable levels. The proposed level generator generates stable and diverse levels when compared with the existing approaches.

1 Introduction

Procedural generation of video game content has been increasingly getting attention [4, 5, 7]. Designing and generating new content for video games requires a large amount of knowledge and experience about the game, making it an expensive task. Procedural content generation (PCG) could help circumvent these costs and ideally even allow the game designer to procedurally design levels based on desired constraints such as difficulty and uniqueness relative to existing levels. Machine learning has been demonstrating itself as a promising tool for doing PCG, utilizing a range of ML techniques including graphical probability models, generative adversarial networks, and variational autoencoders. Deep generative models have been advantageous in PCG applications due to their ability to tweak the results using latent variable evolution.

Though Deep Generative Models have been utilized for PCG in many tile-based games, we have focused on tackling the challenge of generating levels for Angry Birds. The game domain of Angry Birds provides unique challenges that didn't exist for tile-based games including freedom for possible locations and orientations of objects, as well as the presence and effect of gravity. Existing level generators struggle to create stable levels for this game domain because of these issues. In this paper, we implemented a proposed LSTM based Sequential VAE to design stable levels with desired features. We have discussed the model used to generate the stable levels, as well as exploration of latent variable evolution beyond what was done in the original paper.

Preprint. Under review.

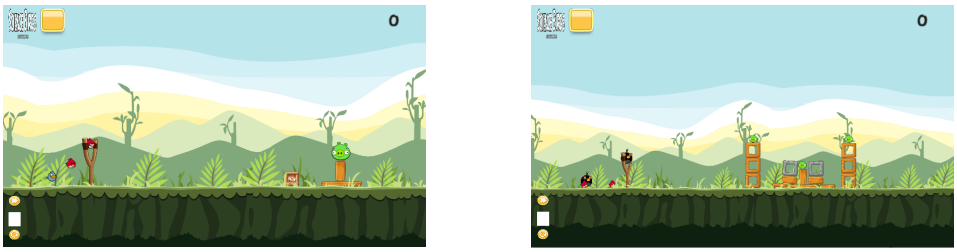

Figure 1: Two example levels from the training set. The meta data such as the type, position, and orientation of the objects are provided in XML files and processed as sentences by our model.

2 Data and Game Domain

The game domain used in this work is a clone of the popular mobile game Angry Birds, called Science Birds [2] used for visualization and simulation of levels. The objective of this game is to destroy all the pigs present at a level by shooting a limited number of colorful birds using a slingshot.

As described in the introduction, the world is not generated using a tile-based map, and instead, for this game, the data is stored in an XML format storing the type, location, and rotation of objects in a scene, as well as a sequence of the birds that will be used for the level. To train the model, we used a data set consisting of 200 levels by IratusAves [6], the winner of the 2018 Level Generation competition, using a 9-1 training-validation split. This is admittedly a very small size of training data but is a realistic number since datasets used in a level generation are usually generated manually. Some examples of levels can be seen in figure 1.

There is no necessary preprocessing step, as the data is already provided in the form of correctly formatted XML files, and we have a protocol for reading level data from these files. For final testing and verification, we used the Unity project as provided by [2] to play and test the stages.

3 Proposed Model

For the Procedural content generation concerning the Angry Birds game, we propose to generate new levels using a generative model implemented using Word Embeddings and Sequential VAE as shown in 2

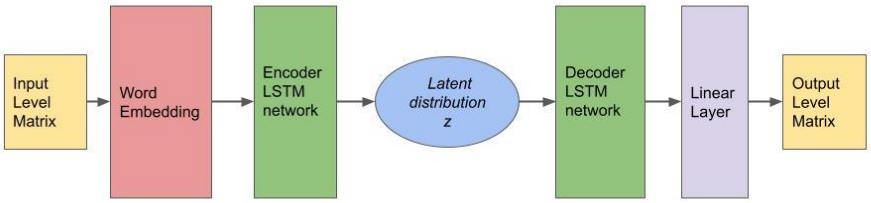

Figure 2: Model Architecture

3.1 Sequential Encoding and Word-based Representation

The level is encoded as sequential data $S = (s_1, \ldots, s_T)$. We refer to this matrix S as the level matrix. The time step in this sequential data indicates the order of dropping objects, while the type and horizontal positions of the dropping objects are specified by s_t. This encoding scheme naturally captures the constraints of gravity and avoids generating any objects overlapping with each other, as well as guaranteeing objects to be placed on the floor or other objects. In our experiments, we set $T = 30$ which is the same as the maximal length in the training data.

The next key step that was used in this work is to process the level matrix S as if it is a natural language sentence. This is done by processing each s_1 as if it were a word that spans the entire width of the world. This word is subdivided into 94 columns that the objects can be placed in, and each of these columns is regarded as a letter in the word. The level matrix is then mapped to a sequence $W = \left(w^{(t)}\right)_{t=1}^{T}$ of one-hot vectors representing a word in the vocabulary, which is just the set of all words that show up in the training dataset. The authors argue that word-based representation leads to better performance than character-based modeling since the correlation between words is easier to process than the correlation between characters.

3.2 Word Embedding

Word embedding is used to map the words from high dimensional space into a vector representation. This high dimensionality of words poses a challenge to the training of the Variational AutoEncoder. Therefore using standard embedding techniques of word2vec with the Continuous Bag-of-Words model, we obtain the low dimensional representation of words. The Bag-of-Words model consists of two linear layers ϵ and τ , one to map the corresponding word to the embedding and next to obtain probability on top of it using soft-max. X = ϵ(W) is represented as the embedding of the word.

3.3 Sequential Variational Autoencoder and Level Generation

The game levels are generated as a sequence W using Sequential Variational Auto-Encoders. Auto-Encoders are one of the most popular and effective Deep generative models. The Auto-Encoders consists of two components, named encoder and decoder. The Encoder transforms data from a high-dimensional space to a lower-dimensional space, called the latent space. In Variational Auto-Encoders, the latent space is a distribution of data, represented as p(x) with x being the input. This latent space distribution is used to retrieve the data back using a model named decoder. In the current work, both encoder and decoder networks are designed from Long Short Term Memory(LSTM) models. This is because LSTM models are widely known for their effectiveness in modeling long-term dependencies in sequence data.

The decoder on the latent space distribution helps in content generation. To facilitate this, the encoding distribution is regularised during the training to ensure that its latent space has good properties allowing us to generate some new data. The encoder is represented as $q_\phi(z/X)$ where x would be the input from the embedding output. The encoded distributions are chosen to be that of normal so that the encoder can be trained to return the mean and the covariance matrix that describes these Gaussians. The decoder models the conditional probability $p_\theta(W/z)$ of sequence W , given latent representations z.

When trying to reconstruct the original data, there would be a reconstruction loss between the decoder output and the actual data. A higher reconstruction loss indicates a higher loss in the amount of information transfer from encoder and decoder. Comparing the generated data with input and back-propagating the error, the model improves on its reconstruction loss. But, there is a higher chance of over-fitting in this case, which would not be beneficial to generate a good latent space distribution for a generation. To address this, regularization is added to the encoder network to map it close to standard Gaussian, determined by a loss called KL divergence. Thus, the sequential VAE loss is composed of the weighted sum of KL loss and the reconstruction loss, called β-VAE as shown in 3

$$L_{rec} = -\frac{1}{\mathcal{D}} \sum_{W \in \mathcal{D}} \ln p_\theta(W|z_W), z_W \sim q_\phi(.|\epsilon(W)) \tag{1}$$

$$L_{\mathcal{KL}} = -\frac{1}{\mathcal{D}} \sum_{W \in \mathcal{D}} D_{\mathcal{KL}}\ q_\phi(z|\epsilon(W))||\ \mathcal{N}(0, I) \tag{2}$$

$$L_{\beta-VAE} = -L_{rec}(\mathcal{D}) + \beta * L_{\mathcal{KL}}(\mathcal{D}) \tag{3}$$

The techniques of KL annealing and word drop are adopted while training the model to address the issue in which the trained decoder ignores the latent vector.

The level generator is defined as $\mathcal{G} : z \rightarrow W$, which maps between the latent space and W. The levels generated are expected to follow the constraints on training data of being playable and stable. From the generated levels, the desired levels are chosen using Latent Variable Evolution. The objective function to be optimized in this case can be represented as minimizing over z for $f(\mathcal{G}(z))\ where\ z \in \mathcal{Z}$, denoting the search space.

4 Latent Variable Evolution

The first latent variable evolution (LVE) approach was proposed by [1]. In their work, the authors train a GAN on a set of real fingerprint images and then apply evolutionary search to find latent vectors matching subjects in the dataset. Although, existing studies on sequential VAE have reported that the latent vector tends to be ignored, and the variation of outputs is created by the randomness of the generator itself, Tanabe and others tried to incorporate LVE into the sequential VAE. In the above experiments, we tried to follow the same procedure explained in the paper. We have worked on a Python implementation using the library pycma [3], which is a stochastic derivative-free numerical optimization algorithm for difficult optimization problems in continuous search spaces. Two objective functions were replicated to decide on the inherent characteristics of the levels generated.

Equation 4 and 5 are the two objective functions to measure the difficulty of a level and the aesthetics of a level respectively.

$$max(5 - N_{Birds}, N_{RemBirds}) * 10 + N_{Blocks} \tag{4}$$

$$max(60 - N_{Blocks}, N_{Blocks} - N_{RemBlocks}) * 10 - N_{Pigs} \tag{5}$$

5 Experiments and Results

Experiments were performed to generate levels using the model described in the previous section for the training set of 200 levels over 500 iterations. The plot of the total loss function, including Reconstruction Loss and KL Loss, can be seen in figure **??**. The word drop rate was set to 0.3, and to implement KL annealing, the KL loss was ignored for the first 250 iterations, justifying the 0 loss of KL initially. The final training and validation loss observed on the model is 0.06 and 0.5 respectively.

Given the initial learning curve for the GCP, we have decided to use a shared colab project which was sufficient to experiment on the current idea. We have trained the model on 200 example levels and generated 20 levels using GPU as the runtime accelerator in colab. The model was successful in generating all the 20 levels as stable and playable. Several examples of levels generated in the final iteration are shown in figure 4.

The generated levels were validated to be playable by loading the levels into Unity Project and testing for the arranged objects' stability under gravity.

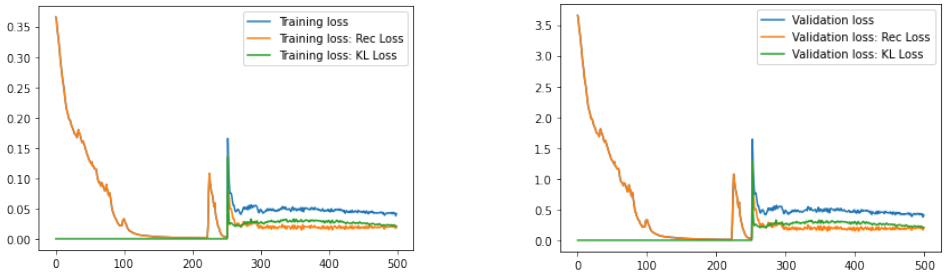

Figure 3: The plots of the loss function on training and validation samples over 500 iterations of the algorithm.

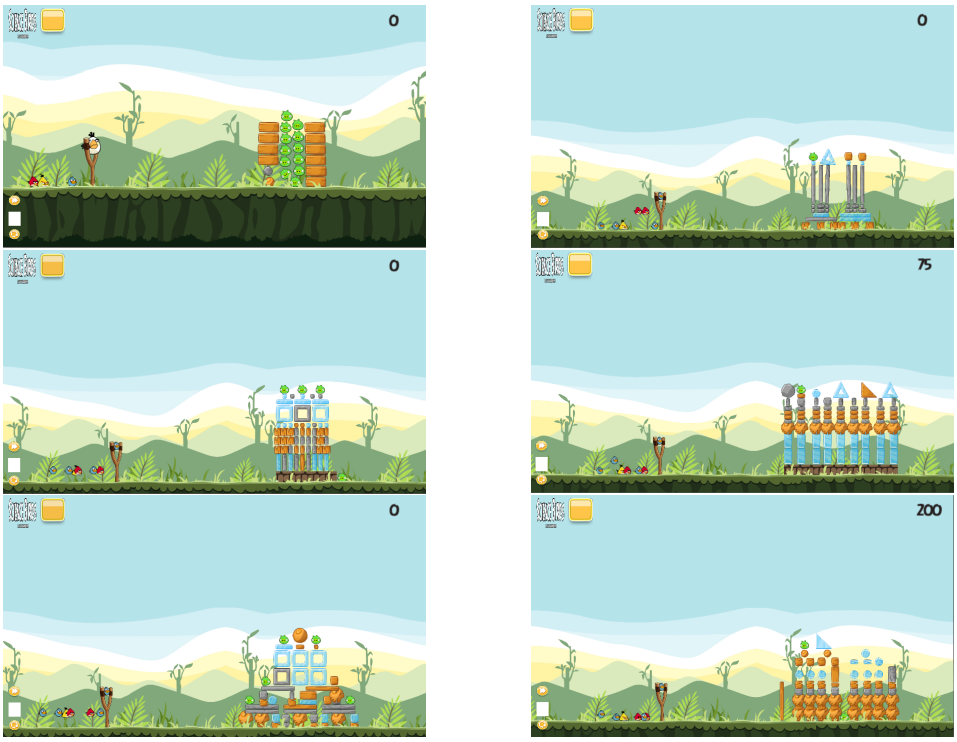

Figure 4: Six example levels that were generated after the end of 500 training iterations.

6 Conclusion

In this study, we proposed a level generator for Angry Birds where it is difficult to generate levels that are diverse and stable under gravity using the existing Deep Generative Models. We encoded each level of an object as a sequence and trained a sequential VAE for level generation rather than considering them as images. We expect that this idea can be applied to other game domains with gravity at a similar level.

A possible future work could be to develop a methodology to evaluate the level in a human-in-the-loop manner as it is difficult to design an objective function to consider qualitative measures such as aesthetics. Furthermore, it can be extended for improving the robustness of AI agents based on reinforcement learning and LVE, by understanding the weakness of the agent in levels and improving on the poor levels.

Work was split equivalently throughout the project amongst the four members.

References

[1] Philip Bontrager, Aditi Roy, Julian Togelius, Nasir Memon, and Arun Ross. Deepmasterprints: Generating masterprints for dictionary attacks via latent variable evolution, 2018.

[2] Lucas Ferreira and Claudio Toledo. A search-based approach for generating Angry Birds levels. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8, Dortmund, Germany, August 2014. IEEE.

[3] Nikolaus Hansen, Youhei Akimoto, and Petr Baudis. CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634, February 2019.

[4] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 9(1):1–22, February 2013.

[5] Jialin Liu, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N. Yannakakis, and Julian Togelius. Deep learning for procedural content generation. *Neural Computing and Applications*, 33(1):19–37, January 2021.

[6] Matthew Stephenson and Jochen Renz. Generating varied, stable and solvable levels for angry birds style physics games. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 288–295, New York, NY, USA, August 2017. IEEE.

[7] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. Procedural Content Generation via Machine Learning (PCGML). *arXiv:1702.00539 [cs]*, May 2018.

[8] Takumi Tanabe, Kazuto Fukuchi, Jun Sakuma, and Youhei Akimoto. Level Generation for Angry Birds with Sequential VAE and Latent Variable Evolution. *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1052–1060, June 2021.