Against Deep Learning for Level Generation in Games

Ethan Robison
ethanrobison@u.northwestern.edu

ABSTRACT

Though there is no strong theoretical reason why deep learning is unsuited for the task of procedural content generation—particularly game level generation—the current state of the games world does not afford the volume of data that one would need to train a deep net; nor do our current deep learning systems afford the transparency and ease of customization that a game designer needs when working to create a game.

ACM Reference Format:
Ethan Robison. 2018. Against Deep Learning for Level Generation in Games. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Games require a great deal of human authoring of content. Content is anything from levels to weapons to plot, and encompasses most of games, time-wise. Procedural content generation is the algorithmic production of content with the intent of reducing the amount of human authoring to produce that content [16] [8].

In particular, the process of creating levels as content is difficult for designers. In addition to the dynamic systems implicit in the level—the enemies, the traps, the hitboxes, and otherwise—there is the way in which the player interacts with all of those things, which is as unpredictable as human behavior always is [14]. This means that designers must spend time not merely arranging the levels they care to design but also ensuring that the experience of playing through these levels is sufficient—it must be both manageable and enjoyable. Procedural content generation mitigates the amount of effort needed in both parts, since a hypothetical system can provide both the arrangement of game elements and also fact-check its own work to ensure that the level is playable.[1]

Procedural content generation techniques are not limited to hand-authored systems and include a variety of machine learning techniques [14]. Given the success of deep learning in areas in [7] and outside of games [6], it is tempting to consider how one might apply deep learning to the field of procedural content generation. However, the application of deep learning to the task of general level generation is currently untenable: there are insufficiently many data to gather a corpus of sufficient size to train a deep net, and a hypothetical deep net trained to generate game levels would not afford the level of customization that a game designer requires.

[1]Ensuring that the level is enjoyable is a more daunting task.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 RELATED WORK

Machine learning techniques for level generation have manifested in the form of Markov chains for tile-to-tile transitions. [10] looks at individual tile transitions, and has the advantage of novelty—the levels that this technique produces are often quite unlike anything seen in the training corpus. By contrast, [1] looks at vertical slices of the level. This loses the advantage of novelty held by the technique in [10] but has the advantage of generating levels which are more likely to be playable, although it tends to produce levels more similar to those found in the training corpus. As noted in [13], both of these methods have the issue of lacking "knobs to tweak," i.e., it is difficult to change the kind of levels one gets without editing the training corpus itself. [13] has the advantage that it has a handful (three) of tunable parameters that an end user can adjust to change the way in which the learned model generates levels. [12] takes the approach of representing a given level as a string and training an LSTM to generate the level. This work considers the impact of player path, and so is likely to generate levels which are playable (that is, they can be completed).

The systems above each have the disadvantage of being limited to generating Super Mario Bros. levels. As noted in [15], this is a function of the limited number of games that have been used to generate usable training corpora. [15] attempts to mitigate this problem by providing a broader video game level corpus, although it provides fewer than 1,000 machine-understandable examples. The issue of corpus size is partially addressed in [11]. Both of these, though they do substantially increase the size and number of the available training corpora, do not provide sufficient training data for general machine learning use [14].

There also exists work on the applications of machine learning towards playing games [7], the general playing of perfect information board games [9] and on more general board game playing [2]. It is worth drawing a distinction, however, between the playing of games and the generation of content for them. There is also work on the generation of rulesets for games [5] [4], although these approaches do not take a machine learning perspective and instead focus on search-based on constructive approaches.

3 DEEP LEARNING FOR LEVEL GENERATION

Most procedural content generation is done as part of the process of developing a game, and serves as a stand-in or reduction to the amount of human labor necessary to create content [16][8]. This means that a system that generates a large volume of content for a game which has already been completed is beneficial insofar as the amount of extra human input is limited. However, the typical application of procedural content generation is in reducing the amount of work needed to create a game which is currently in production. In other words, the ideal content generation system is

one which the designer can make and then subsequently solve all of their content creation needs.[2]

This means that the ideal content creation system is one which can be made for a game which does not necessarily have a large volume of content already created (such as one which is midway through production). Furthermore, the ideal system is one which is flexible enough to be adjusted throughout the design process, as the needs of the designer and the game itself change.

In the creation of a system for procedural content generation, one seeks to have the simultaneous features of generality—since it is constructed from the scaffolding of a game whose design characteristics are not necessarily known—and customization—since the design parameters upon which it was initially based are subject to change as the game goes through the development process.

3.1 Generality

A design goal of a hypothetical level generation system is that it produce levels which fit some notion of "validity" that may be underspecified until late in the design process. A generator for a platformer game is expected to produce valid platformer levels, notwithstanding parameterization on elements such as what the specific building blocks of those levels may be. This holds as well for all genres of game: a level generator must produce valid levels for that set of generic conventions, and it needs to do so early on in the design process.

This means that the content generation system for a given game needs to be based not on the games levels themselves, but on a more general corpus of levels from other games. The issue is not resolved, however, by merely amassing a collection of every game level ever created. As noted in [14], games are unlikely to share an internal representation of levels, so attempting to learn these data structures is not guaranteed to bear fruit. One can mitigate this problem by instead looking at the image produced by playing through the level, as done in [11]. This solves the problem of ambiguity among data structures, since it allows for the more universal representation of a level as a matrix of RGB values.

Universality is limited, though, because what represents a valid level in one game does not necessarily represent a valid level in another. Learning to generate the images of a level in one game is not guaranteed to produce valid levels in another—games do not always share semantic meaning of their game elements. Methods such as representing the level as a string [12] or reducing the level to its more fundamental image components [10][13][1] helps with this problem insofar as it enables the system to learn more abstract concepts such as "platforms" and "enemies" as opposed to color values.

However, these methods of generalizing the representations of levels for a particular game (or class of game) fall short of solving the problem of encoding a Platonic notion of a level—they encode only a level in, e.g., a platformer, or a level which is made up of a finite set of building blocks common to a particular set of generic conventions and mechanics. In other words, even a system which learns to generate general levels for a particular kind of game (meaning one with a specific set of mechanics and conventions) would be unable to generate levels for a game that differed in its mechanics or its gameplay conventions.

So far, this has been a commentary on the needs of a general machine-learning system for generating levels in a game. Specifically, on the importance of its ability to produce levels which adhere to certain generic conventions while at the same time producing levels for the game that the designer is making. The point is that it is inadequate to take the corpus of every level from every game and use that to train your system—it needs to be a corpus whose levels represent valid levels for the kind of game that the designer is trying to make.

This poses a problem for deep learning systems, since the dataset of valid levels is restricted. The largest general video game corpus is only 428 levels [15], and even the entirety of the NES library is only 822 games [14]. A liberal estimate of 100 levels per game would imply that there are something like 10^6 levels to be found in this library alone. Except that the largest series in the library makes up less than 1% of the total corpus [14], so even with the generous 10^6 levels we estimated, there are less than 1,000 for even the best-represented game.

Expanding the sample to include all games in the platformer genre yields 14% [14] of our hypothetical 10^6 or 14,000. However, this substantially larger corpus is still vulnerable to the limitation that the levels within it are not necessarily interchangeable, either in terms of the elements that compose them or of the mechanics governing interaction with those elements. In other words, video game levels from separate games are too distinct from one other to be reasonably combined into a single corpus for use in training a deep net. It is unreasonable to make the claim that those levels represent members of the same class.

3.2 Customization

As noted in [13], a common disadvantage in systems which generate levels for games is the lack of a small and transparent set of tunable parameters. The number of knobs that are available to the designer for tweaking must be manageable, and it needs to be clear to them what each knob is ostensibly doing. Though the point of a content generation system is to reduce the amount of human input in generation, it is also important that the designer working with the system be able to adjust the system if the output that it produces is not what they want it to be [8].

The challenge with using a hypothetical deep learning system to generate levels lies in providing the designer with a useful mental model for how they can adjust the end result (the deep net itself) in order to change the content that it produces. If the list of tunable parameters is large, or if the parameters that are present are not clearly mapped to what they are providing in terms of output, then a designer is going to have difficulty using the system to make content that they care to make.

This is balanced with the notion that a deep net is supposed to be trained on the kind of data the one might want to see in its output—in this case, one would train the net on a corpus of "good levels." The problem with this comes if and when a designer feels disinclined to accept the output of the system as necessarily the kind of content that they want it to produce. Given the nature of

[2]Notwithstanding the amount of work necessary to actually create this system, which is typically substantial.

game design—academic or industrial—this is an inevitability. There is a certain lack of customization present in a level generation system that hinges on a deep net, since the only way to adjust the deep net [3] is to change the training corpus and re-train.

4 CONCLUSION

A hypothetical deep net trained on a vast supply of levels from an already completed game would be insufficient for a designer's needs—the generator needs to be general enough to work with a game which is still in development. However, attempting to train a general deep net to generate levels on the corpus of available data is intractable, due to the limited number of levels in general combined with the uniqueness of the data themselves.

Furthermore, such a deep net would lack the transparency and customization that the designer of an in-development game would necessarily need to have, since the design constraints of the project are subject to change over time. The only general way to tweak a deep net is to adjust and reconstruct the net, which is not a tenable method for anyone attempting to be nimble in their development process.

Though there are applications of deep learning in gaming [3] [7] they are currently limited to applications which are analogous to real-world applications but are mapped onto the digital space. This is not to say that there is not a future for deep learning in areas such as narrative or procedural content generation, rather that current technologies and abstract representations of data do not afford the application of deep nets to problem areas which are more content focused.

REFERENCES

[1] Steve Dahlskog, Julian Togelius, and Mark J Nelson. [n. d.]. Linear levels through n-grams. ([n. d.]). http://www.kmjn.org/publications/NgramLevels_MT14.pdf

[2] Fernando de Mesentier Silva, Scoo Lee, Julian Togelius, and Andy Nealen. 2017. AI-based Playtesting of Contemporary Board Games. (2017). https://doi.org/10.1145/3102071.3102105

[3] Astrid Ensslin, Tejasvi Goorimoorthee, Shelby Carleton, Vadim Bulitko, and Sergio Poo Hernandez. [n. d.]. Deep Learning for Speech Accent Detection in Videogames. ([n. d.]). https://aaai.org/ocs/index.php/AIIDE/AIIDE17/paper/viewFile/15861/15222

[4] Ahmed Khalifa, Michael Cerny Green, Diego Perez-Liebana, and Julian Togelius. [n. d.]. General Video Game Rule Generation. ([n. d.]). http://julian.togelius.com/Khalifa2017General.pdf

[5] Ahmed Khalifa and Julian Togelius. [n. d.]. Marahel: A Language for Constructive Level Generation. ([n. d.]). http://julian.togelius.com/Khalifa2017Marahel.pdf

[6] Li-Jia Li, Kai Li, Jia Deng, Wei Dong, Richard Socher, and Li Fei-Fei. 2009. ImageNet: a Large-Scale Hierarchical Image Database. (2009). https://doi.org/10.1109/CVPR.2009.5206848

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. [n. d.]. Playing Atari with Deep Reinforcement Learning. ([n. d.]). https://arxiv.org/pdf/1312.5602.pdf)

[8] Noor Shaker, Julian Togelius, and Mark J. Nelson. 2016. *Procedural Content Generation in Games*. Springer International Publishing.

[9] David Silver, Julian Schrittwieser, Karen Simonyan, ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas baker, Matthew Lai, Adrian bolton, Yutian chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature Publishing Group* 550 (2017). https://doi.org/10.1038/nature24270

[10] Sam Snodgrass and Santiago Ontã. 2013. Generating Maps Using Markov Chains. (2013). https://pdfs.semanticscholar.org/379f/a1bb03257804bef9a0b186c6d128509f9696.pdf

[11] Adam Summerville, Matthew Guzdial, Michael Mateas, and Mark O Riedl. [n. d.]. Learning Player Tailored Content From Observation: Platformer Level Generation from Video Traces Using LSTMs. ([n. d.]). https://www.aaai.org/ocs/index.php/AIIDE/AIIDE16/paper/viewFile/14069/13620

[12] Adam Summerville and Michael Mateas. 2016. Super Mario as a String: Platformer Level Generation Via LSTMs. (3 2016). http://arxiv.org/abs/1603.00930

[13] Adam Summerville, Shweta Philip, and Michael Mateas. 2015. MCMCTS PCG 4 SMB: Monte Carlo Tree Search to Guide Platformer Level Generation. (2015). https://www.aaai.org/ocs/index.php/AIIDE/AIIDE15/paper/viewFile/11569/11396

[14] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. [n. d.]. Procedural Content Generation via Machine Learning (PCGML). ([n. d.]). https://arxiv.org/pdf/1702.00539.pdf

[15] Adam James Summerville, Sam Snodgrass, Michael Mateas, and Santiago Ontã Nón. [n. d.]. The VGLC: The Video Game Level Corpus. ([n. d.]). https://arxiv.org/pdf/1606.07487.pdf

[16] Julian Togelius, Emil Kastbjerg, David Schedl, and Georgios N. Yannakakis. 2011. What is procedural content generation?. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games - PCGames '11*. ACM Press, New York, New York, USA, 1–6. https://doi.org/10.1145/2000919.2000922

[3] Other than manually tuning the weights, which is absurd.