Conditional Level Generation and Game Blending

Anurag Sarkar
Northeastern University

Zhihan Yang
Carleton College

Seth Cooper
Northeastern University
Motivation

- Variational Autoencoders (VAEs) have been used for generating and blending game levels
Motivation

• Variational Autoencoders (VAEs) have been used for generating and blending game levels

• Controllability via latent vector evolution
  • Define objective function
  • Run search in latent space to evolve desired vectors
Motivation

• Variational Autoencoders (VAEs) have been used for generating and blending game levels

• Controllability via latent vector evolution
  • Define objective function
  • Run search in latent space to evolve desired vectors
    --- post-training process independent of the model
  --- sometimes limited controllability
Motivation

- Variational Autoencoders (VAEs) have been used for generating and blending game levels
- Controllability via latent vector evolution
  - Define objective function
  - Run search in latent space to evolve desired vectors
    --- post-training process independent of the model
    --- sometimes limited controllability
- Conditional VAEs enable controllability as part of the model itself
  - Train on labeled data
  - Generation conditioned on input labels
  - Various design affordances
Variational Autoencoder (VAE)

- Autoencoders are neural nets that learn lower-dimensional data representations
  - Encoder $\rightarrow$ input data to latent space
  - Decoder $\rightarrow$ latent space to reconstructed data
- VAEs make latent space model a probability distribution (e.g. Gaussian)
  - Allows learning continuous latent spaces
  - Enables generative abilities similar to those of GANs (sampling, interpolation)

source: jdykeman.github.io/ml/2016/12/21/cvae.html
Variational Autoencoder (VAE)

- Autoencoders are neural nets that learn lower-dimensional data representations
  - Encoder $\rightarrow$ input data to latent space
  - Decoder $\rightarrow$ latent space to reconstructed data
- VAEs make latent space model a probability distribution (e.g. Gaussian)
  - Allows learning continuous latent spaces
  - Enables generative abilities similar to those of GANs (sampling, interpolation)
Conditional VAE (CVAE)

• CVAEs associate input data with labels during training
• Encoder uses label to learn latent encodings of inputs
• Decoder uses same label to learn how to reconstruct input from latent encoding
• Same latent vector can produce different outputs by varying label

source: jdykeman.github.io/ml/2016/12/21/cvae.html
Conditional VAE (CVAE)

- CVAE could inform level design/generation by:
  - Enabling controllable generation by using labels to produce desired content
  - Generate variations of existing content by decoding it using different labels
Approach

- Games:
  - Super Mario Bros.
  - Kid Icarus
  - Mega Man

- Three conditioning approaches:
  - Game elements
  - Mario design patterns
  - Game blending

- For all cases:
  - 16x16 segments
  - Binary-encoded vectors as labels
  - 3 latent dimensions per model (32, 64, 128)
Game Elements

• Unique set of conditioning labels for each game

• Label length $\rightarrow$ number of different elements
  • 5 for SMB/MM, 4 for KI
  • Each unique label corresponds to a unique combination of elements

• Trained separate CVAE for each game

• Labels for training segments determined by checking for the relevant game elements within that segment
  • Present $\rightarrow$ set bit to 1
  • Absent $\rightarrow$ set bit to 0
Game Elements

• Conditioning Accuracy Evaluation:
  • For each game, sampled 1000 latent vectors
  
  • Conditioned generation of each using each possible label (32 for SMB/MM, 16 for KI)
  
  • Compared elements in generated segments with labels used for generation
    
    • Exact → all elements present
    
    • None → none of the elements present
Game Elements

(a) SMB

(b) KI

(c) MM
Design Patterns

- 10 SMB design patterns adapted from Dahlskog and Togelius, “Patterns and Procedural Content Generation: Revisiting Mario in World 1 Level 1”, 2012

- Binary labels of length 10

- Used levels from
  - Super Mario Bros.
  - Super Mario Bros II: The Lost Levels

- Labels assigned manually based on visual inspection

  **Enemy Horde (EH):** group of 2 or more enemies  
  **Gap (G):** 1 or more gaps in the ground  
  **Pipe Valley (PV):** valley created by 2 pipes  
  **Gap Valley (GV):** valley containing a Gap  
  **Null (empty) Valley (NV):** valley with no enemies  
  **Enemy Valley (EV):** valley with 1 or more enemies  
  **Multi-Path (MP):** segment split into multiple parts horizontally by floating platforms  
  **Risk-Reward (RR):** segment containing a collectable guarded by an enemy  
  **Stair Up (SU):** ascending stair case pattern  
  **Stair Down (SD):** descending stair case pattern

  **Mario Design Patterns**
Design Patterns

- More challenging to evaluate
  - Unlike game elements, couldn’t automatically check for design patterns

- Couldn’t automatically determine label matches

- No success in training a classifier due to low amount of data relative to number of unique labels

- Currently, restricted to visual inspection

Mario Design Patterns

- **Enemy Horde (EH):** group of 2 or more enemies
- **Gap (G):** 1 or more gaps in the ground
- **Pipe Valley (PV):** valley created by 2 pipes
- **Gap Valley (GV):** valley containing a **Gap**
- **Null (empty) Valley (NV):** valley with no enemies
- **Enemy Valley (EV):** valley with 1 or more enemies
- **Multi-Path (MP):** segment split into multiple parts horizontally by floating platforms
- **Risk-Reward (RR):** segment containing a collectable guarded by an enemy
- **Stair Up (SU):** ascending stair case pattern
- **Stair Down (SD):** descending stair case pattern
Game Blending

• Trained on segments from all 3 games taken together

• 3-element labels indicating which game a segment belonged to

• Blending by conditioning generation using blended labels
  • \(<110> \rightarrow \text{SMB + KI}\)
  • \(<011> \rightarrow \text{KI + MM}\)
  • \(<101> \rightarrow \text{SMB + MM}\)
Game Blending

- Label accuracy evaluation issues:
  - Hard to automatically detect blending
  - No ground truth for blended levels
Game Blending

• Label accuracy evaluation issues:
  • Hard to automatically detect blending
  • No ground truth for blended levels

• Proxy evaluation:
  • Train a classifier on original segments to predict which game they belong to
  • Test to see how predictions on CVAE-generated segments change with different conditioning labels
Game Blending

• Label accuracy evaluation issues:
  • Hard to automatically detect blending
  • No ground truth for blended levels

• Proxy evaluation:
  • Train a classifier on original segments to predict which game they belong to
  • Test to see how predictions on CVAE-generated segments change with different conditioning labels
  • Sample 1000 latent vectors
  • Condition generation of each using each of 8 possible conditioning labels
  • For each, compute % of generated segments predicted as SMB, KI or MM by classifier
Game Blending

- Expectations
  - Conditioning with an original game label (<100>, <010>, <001>)
    --- e.g. using <100> $\rightarrow$ very high % of SMB predictions
  - Conditioning with blended game label (e.g. <110>, <101>)
    --- more variance among predictions
    --- e.g. using <101> $\rightarrow$ moderately high % for both SMB/MM, but not too high, low % for KI
Game Blending

• Expectations
  • Conditioning with an original game label (<100>, <010>, <001>)
    --- e.g. using <100> \( \rightarrow \) very high % of SMB predictions
  • Conditioning with blended game label (e.g. <110>, <101>)
    --- more variance among predictions
    --- e.g. using <101> \( \rightarrow \) moderately high % for both SMB/MM, but not too high, low % for KI

• Results
  • True to expectations
  • <100>, <010>, <001> \( \rightarrow \) high% for SMB, KI, MM respectively
  • More variance among labels with multiple 1s (i.e. blended)
  • Most variance using <000> and <111>

<table>
<thead>
<tr>
<th>Label</th>
<th>SMB</th>
<th>KI</th>
<th>MM</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;000&gt;</td>
<td>38.7</td>
<td>18.1</td>
<td>43.2</td>
</tr>
<tr>
<td>&lt;001&gt;</td>
<td>3.8</td>
<td>2.4</td>
<td>93.8</td>
</tr>
<tr>
<td>&lt;010&gt;</td>
<td>0.7</td>
<td>95.5</td>
<td>3.8</td>
</tr>
<tr>
<td>&lt;011&gt;</td>
<td>6.8</td>
<td>22.9</td>
<td>70.3</td>
</tr>
<tr>
<td>&lt;100&gt;</td>
<td>97.6</td>
<td>1.4</td>
<td>1</td>
</tr>
<tr>
<td>&lt;101&gt;</td>
<td>71.9</td>
<td>2.9</td>
<td>25.2</td>
</tr>
<tr>
<td>&lt;110&gt;</td>
<td>86.5</td>
<td>11.8</td>
<td>1.7</td>
</tr>
<tr>
<td>&lt;111&gt;</td>
<td>56.7</td>
<td>10.3</td>
<td>33</td>
</tr>
</tbody>
</table>

Blending Classification
Game Blending

• Further evaluation:
  • Compare distributions of levels obtained using each label with original game distributions
  • Generated 1000 segments using each blend label
  • Computed E-distance between each set of 1000 vs. each of SMB, KI and MM
  • Lower the E-distance between 2 distributions, more similar they are
  • Used 4 tile-based metrics – Density, Leniency, Nonlinearity, Interestingness
Game Blending

- Further evaluation:
  - Compare distributions of levels obtained using each label with original game distributions
  - Generated 1000 segments using each blend label
  - Computed E-distance between each set of 1000 vs. each of SMB, KI and MM
  - Lower the E-distance between 2 distributions, more similar they are
  - Used 4 tile-based metrics – Density, Leniency, Nonlinearity, Interestingness
Game Blending

SMB

KI

MM

Random

000

001

010

011

100

101

110

111
Conclusion

• Explored the use of conditional VAEs for PCGML

• Enable controllable level generation and blending

• Editing and producing novel variations of existing levels
Future Work

• Combine with evolutionary search for further controllability

• Blending – improve quality, more controllability

• More thorough focus on design patterns, more robust evaluations (user-study, playability)

• Combine with our sequential model for enabling conditional generation of whole levels

• Incorporate into co-creative tools
Future Work

• Combine with evolutionary search for further controllability

• Blending – improve quality, more controllability

• More thorough focus on design patterns, more robust evaluations (user-study, playability)

• Combine with our sequential model for enabling conditional generation of whole levels

• Incorporate into co-creative tools

Contact
Anurag Sarkar
Northeastern University
sarkar.an@northeastern.edu