Final Project Report: Automatic Detection of Interesting Cellular Automata

CS 294-082 – Experimental Design for Multimedia Machine Learning (Graduate) – Fall 2020

|  |  |
| --- | --- |
| Randy Fan, Qitian Liao (Student Authors)  University of California, Berkeley  [randyfan@berkeley.edu](mailto:randyfan@berkeley.edu), liao1024@berkeley.edu | Dr. Gerald Friedland (Faculty Advisor)  Department of EECS, University of California, Berkeley  fractor@eecs.berkeley.edu |

**ABSTRACT**

**The introduction of Machine Learning and Deep Neural Networks has revolutionized a variety of task. Our project explores the potential of using Deep Neural Networks to detect interesting Cellular Automata. A cellular automaton is a collection of colored cells on a grid that evolves according to a set of rules. The rules are applied iteratively for as many time steps as desired.**

One challenge is that Cellular Automata tend to have huge parameter search spaces to find interesting results within. The vast majority of rules within this space will be junk rules with only a small fraction of a percentage being interesting rules. Between the boring rules that either die out tend to chaos, there is a tiny sweet spot of interesting patterns. Manually searching for these patterns would be unrealistic. In this paper, we will discuss our approach to detect the interesting patterns using image processing, Convolutional Neural Networks (CNNs) and ImageNet.

KEYWORDS

Cellular Automata, Image Processing, Convolutional Neural Network, ImageNet, Dynamic patterns

1 INTRODUCTION / PROBLEM

In this paper, we will focus on outer totalistic generations of two-dimensional Cellular Automata. They consist of a two-dimensional grid of cells, which live, die, or are in a “dying” transition date depending on a set of rules. Specifically, the cells are updated according to their previous values, and the sum of the values of the other cells in the neighborhood. We decided to use Moore neighborhood for the update rules, which are the surrounding eight cells of a center cell. An alternative would be Neumann neighborhood, which only includes four cells. Rules are classified as boring if they lead to a pattern which is static noise with no discernable patterns moving across the screen. For this project, we define interesting rules specifically as those that have clear gliders (small patterns that move across the grid) with some individual characteristics. Under most circumstances, it is impossible to tell whether a rule is interesting or boring just by looking at the parameters. A user would have to randomly go through thousands of random rules before finding an interesting one. Even more daunting, under the assumption that we use a Moore neighborhood and a maximum of 10 possible states, there are 29 survival rules, 29 born rules, and 210 states, which leads to a total of 228 combinations of rules. Because of this gigantic number of possible combinations, automatic detection of interesting rules will be very helpful, and it will make sure that users do not have to manually go through the process.

2  BACKGROUND / RELATED WORK

// TODO

3  METHOD

3.1  Data Collection Pipeline

To obtain boring rules, we manually went through random examples, and collected rules that died out immediately or generated static noise or boring non-glider patterns. For interesting rules, we used Cellular Automata Rules Lexicon, and examples provided in Visions of Chaos and only recorded those with gliders. We ended up with 105 boring and 35 interesting set of rules.

We generalized the Cellular Automata generation algorithm as follows:

**Cellular Automata Generation Algorithm**

if center\_cell == max\_state:

for num\_neighbors in survive\_arr:

if total - 1 == num\_neighbors:

return center\_cell

return center\_cell - 1

elif center\_cell != 0 and center\_cell != max\_state:

return center\_cell - 1

else:

for num\_neighbors in born\_arr:

if total == num\_neighbors:

return max\_state

return 0

We used the algorithm above to generate frames and create images. Next, we applied data augmentation due to the limited number of rules that we identified. We reused the boring rules 10 times, and the interesting rules 30 times with different initial configuration. We generated 140 images for each of the patterns.

Our next step is image stitching. We chose to use frames from 100 to 108. Because when we were running examples, we noticed that this is roughly when the patterns were all apparent. Each pattern corresponds to one stitched together 3 by 3 image. We assembled training data with a 50% interesting and 50% boring split.

3.2  Data Training with CNN

We trained the data set with a Convolutional Neural Network with the following architecture.

Table

Description automatically generated

We tuned the hyperparameters and found out that batch normalization and more data inclusion greatly improved the accuracy of the Neural Network. Without batch normalization, the testing accuracy was only 65.08%. After batch normalization and dropout, we were able to achieve 93.44% training accuracy and 84.12% testing accuracy on the testing set with 10% interesting data. The test recall is 100%, indicating every interesting configuration has been correctly labeled as such.

3.3  Feature Extraction

We eventually need to feed the results into Brainome.ai, which requires column data. Therefore, we decided to move on to feature extraction. We used a pretrained NASNet-Large Model, which is a Convolutional Neural Network that is trained on more than a million images from the ImageNet Database. For each of the stitched images, the model returns 1000 selected features.

**Image Feature Extraction Algorithm**

model\_name="nasnetalarge"

model=pretrainedmodels.\_\_dict\_\_[model\_name](num\_classes=1000, pretrained='imagenet')

model.eval()

load\_img = utils.LoadImage()

tf\_img = utils.TransformImage(model)

features\_file = open("file.csv", "ab")

feature\_data = []

for i in range(len(image\_paths)):

input\_img = load\_img(image\_paths[i])

input\_tensor = tf\_img(input\_img)

input\_tensor = input\_tensor.unsqueeze(0)

input = torch.autograd.Variable(input\_tensor, requires\_grad=False)

output\_logits = model(input)

output\_features = model.features(input)

output\_logits = model.logits(output\_features)

output\_logits = output\_logits[0].detach().numpy()

row\_data = np.append(output\_logits, labels[i])

feature\_data = np.append(feature\_data, row\_data)

We fed the data into Brainome.ai and obtained the following information about Decision Trees and Neural Networks. The decision tree has 1026 parameters, and the estimated memory equivalent capacity for neural networks is 11034 parameters. Expected generalization using Decision Tree is 2.05 bits/bit and using a Neural Network is 0.19 bits/bit.

3.4  Entropy Evaluation

// TODO

4  CONTRIBUTIONS AND FUTURE WORK

// TODO

(Custom Feature Engineering)

5  CLASS QUESTIONNAIRE

Here the paper will answer the eight questions posted by Professor Friedland to evaluate machine learners.

**Q1: What is the variable the machine learner is supposed to predict? How accurate is the labeling? What is the annotator agreement (measured)?**

Given a set of rules and an initial configuration, the machine learner aims to predict whether the generated Cellular Automata pattern will be interesting or boring. CNN gives a training accuracy of 93.44% and testing accuracy of 84.12%.

**Q2: What is the required accuracy metric for success? How much data do we have to train the prediction of the variable? Are the classes balanced? How many modalities could be exploited in the data? Is there temporal information? How much noise are we expecting? Do we expect bias?**

The classes are balanced because the data contains exactly 50% interesting rules and 50% boring rules. There is temporal information because we use sequences of frames, which record the state of the particles with respect to time. Noise is possible but very rare according to empirical rules. They occur because random initial configuration may lead to results different as expected. For instance, a supposedly boring set of rules might generate glider patterns given a particular initial configuration.

**Q3: What is the Memory Equivalent Capacity for the data (as a dictionary). What is the expected Memory Equivalent Capacity for a neural network?**

We have 2000 images in total, and the classification of each image requires 1 bit since there are two classes. Therefore, the Memory Equivalent Capacity for the data is 2000 bits. The expected Memory Equivalent Capacity for a neural network is computed as (log2(thresholds + 1) \* d), which is 986985 bits.

**Q4: What is the expected generalization in bits/bit and as a consequence the average resilience in dB? Is that resilience enough for the task? How bad can adversarial examples be? Do we expect data drift?**

// TODO

**Q5: Is there enough data? How does the capacity progression look like?**

// TODO

**Q6: Train your machine learner for accuracy at memory equivalent capacity. Can you reach near 100% memorization? If not, why (diagnose)?**

Yes, we can increase the number of layers and neurons, which will allow us to reach memory equivalent capacity easily.

**Q7: Train your machine learner for generalization: Plot the accuracy/capacity curve. What is the expected accuracy and generalization ratio at the point you decided to stop? Do you need to try a different machine learner? How well did your generalization prediction hold on the independent test data? Explain results. How confident are you in the results?**

// TODO

**Q8: Comment on any other quality assurance measures possible to take/the authors should have taken. Are there application-specific ones?**

// TODO

ACKNOWLEDGMENTS

Randy Fan, Qitian Liao

REFERENCES

|  |  |
| --- | --- |
| [1] |  |
| [2] | Bowman, M., Debray, S. K., and Peterson, L. L. 1993. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst.* 15, 5 (Nov. 1993), 795-825. DOI= http://doi.acm.org/10.1145/161468.16147.dy of Positive Youth Development. Journal of Early Adolescence 25 (1): 17-71. |
| [3] | . |
| [4] | . |
| [5] | . |

Context for Reviewers

The second page, which will be for the reviewers only, and not included in the proceedings, should provide more context for reviewers. You may format this page as you prefer. Examples of content you might provide on this page include: description of tentative layout and content of the poster, description of any handouts that may be available, brief description of author expertise and background as it relates to the poster (remember not to include actual author names or institutions in your anonymous submission). You should also justify how your poster will engage participants in discussion.