A Machine Learning Approach to Crossword Generation

Julian Booth
Department of Computer Science
Simon Fraser University
Burnaby, BC, V5A 1S6
julius_booth@sfu.ca

Chun-Wei Chen
Department of Computer Science
Simon Fraser University
Burnaby, BC, V5A 1S6
cwc30@sfu.ca

Stefano Macri
Department of Computer Science
Simon Fraser University
Burnaby, BC, V5A 1S6
smacri@sfu.ca

Chad Malla
Department of Computer Science
Simon Fraser University
Burnaby, BC, V5A 1S6
cmalla@sfu.ca

Abstract

Crossword puzzles are ubiquitous, and yet there has been, to our knowledge, no attempts to generate them using machine learning. A valid crossword puzzle is one in which every across and down word is in a lexicon. By making small iterative changes guided by a neural network to a valid "seed" puzzle, we can jump from valid puzzle to valid puzzle until we have a generated completely new puzzle.

1 Introduction

Pick up any random newspaper and you are bound to find a crossword puzzle. The crossword puzzle is an $m \times n$ matrix, where each cell is filled with either a letter or a black square. A valid crossword puzzle is a crossword puzzle in which every horizontal and vertical sequence of letters between black squares and/or the edges of the board are words found in a lexicon L. The American style puzzle (Fig. 1) differs from the British style (Fig. 2) by having very few black squares. Words are interlocked when they share a letter. A fully interlocked puzzle has no black squares. The higher degree of interlocking in American style puzzles means fewer degrees of freedom and thus they are more challenging to generate. We will be focusing on the American style crossword puzzle for our paper.

Figure 1: American (left) and British (right) Crossword Boards

Crosswords puzzles may be ubiquitous in society, but it appears that their generation is still a human-led activity. In the literature, crossword puzzle generation is referred to as "crossword compilation". As far as we know, there are no published crossword puzzle compilers that can create an American style puzzle larger than 4×4 cells within a reasonable amount of time (on 1980's computers). Most of the methods we looked at used a word by word approach [1, 2]. They fill in a puzzle one word at a time until a conflict arises and then they backtrack. They also use heuristics, such as filling in word slots with the fewest number of fitting words first, in order to minimize the amount of backtracking required to enumerate all possible puzzles. A previous attempt at the problem used a similar strategy, but filled in the puzzle letter by letter [3]. The backtracking/enumeration strategy appears to slow down drastically as the puzzle size increases. The papers we looked at used time per solution as a measure of efficiency, but we do not think run-time is a fair way to compare our algorithms to algorithms run in the 1980s. The per solution time of the most recent method increased drastically with size, 1.7 seconds for 3×3, and 1 minute for 4×4 puzzles with complete interlocking [2] and none of the methods provided any 5×5 solutions, therefore if we can create 5×5 or larger solutions, we will consider that an improvement.

Crossword compilation is a hard problem because of how few valid puzzles there are in comparison to the total number of possible puzzles. For a fully interlocked $n \times n$ board there are 26^{n^2} possible puzzles. Let W_n be the number of n letter words in L. We estimate there are only $\frac{W^{2n}}{26^{n^2}}$ valid puzzles per $n \times n$ board. We enumerated the size of the possible puzzle space $P_{n \times n}$ and estimated the size of the valid puzzle space $V_{n \times n}$ in Table 1 for fully interlocked $n \times n$ boards. Interestingly, it is not expected that there are any valid puzzles on a 7×7 (or greater) fully interlocked board. Thus all large puzzles must have some pattern of black squares.

$n \times n$	W_n	$P_{n \times n} = 26^{n^2}$	$V_{n \times n} \approx \frac{W_n^{2n}}{26^{n^2}}$	$\frac{V_{n \times n}}{P_{n \times n}}$
1×1	26	26	26	1
2×2	438	4.57×10^5	8.05×10^4	1.76×10^{-1}
3×3	5320	5.43×10^{12}	4.18×10^9	7.69×10^{-4}
4×4	15598	4.36×10^{22}	8.03×10^{10}	1.84×10^{-12}
5×5	30281	2.37×10^{35}	2.74×10^9	1.16×10^{-26}
6×6	44892	8.69×10^{50}	7.71×10^4	8.87×10^{-47}
7×7	58691	2.16×10^{69}	2.67×10^{-3}	1.24×10^{-72}

Table 1: Set Sizes of Valid and Possible Puzzles for Different Sized Boards

It is clear that the letter by letter backtracking method [3] is unreasonable for puzzles even as small as 4×4. The word by word method reduces the search space drastically by reducing the number of n-length character combinations possible in each row and column from 26^n to W. However, because word by word methods had such a difficult time with even 4×4 boards, we do not believe it is a viable method on the road to compiling newspaper sized puzzles (at least 15×15).

Our project used a fundamentally different approach to generate crossword puzzles. Instead of starting with an empty puzzle and filling in the letters or words from scratch, we started with a valid "seed" puzzle and then made incremental, letter by letter changes, guided by a neural network, until we had a completely new puzzle. Compiling refers to producing something from scratch, so when referring to our own method we will use the term "crossword generation". Our technique was able to easily generate 4×4 and even 5×5 puzzles. We were also able to generate larger puzzles with a few, strategically placed black squares. However, we are not able to at this time generate a newspaper sized puzzle. We were unable to find any machine learning approaches to crossword generation in the literature.

We used a larger lexicon than the backtracking methods to increase the size of V. We believe using a larger lexicon is not an unfair advantage over the word by word methods because they would not solely improve with a larger lexicon. For example, backtracking methods may take longer to exhaust the word list before backtracking if stuck on an unpromising board [1,2]. Our lexicon was scraped from the internet and includes the entire dictionary plus many acronyms, names, and phrases that have appeared as crossword answers before.

BACK
BANK
BANE
WANE
WINE

Figure 2: Word Ladder

2 Approach

A 4×4 fully interlocked crossword puzzle can be one hot encoded as a $4 \times 4 \times 26$ matrix. Therefore, the set $P_{4\times4}$ can be seen as a 416 dimensional space. A manifold is a set that is locally homeomorphic to a lower dimensional space. The manifold hypothesis states that many real world data sets lie on low-dimensional manifolds [4]. For example, the set of all cat images may lie on a lower dimensional manifold in the space of all possible images.

The manifold hypothesis is key to the insight that leads to our approach. We believe the set V is distributed throughout P as a series of lower dimensional manifolds. This is intuitively apparent. If each letter in our 4×4 puzzle is randomly chosen, we should expect to see a valid puzzle 1 in 5.43×10^{11} times using our lexicon. But if we start with a valid puzzle and change one letter, such as a to e, we would expect to still see a valid puzzle quite often. We define the function $N(p, d)$ to be the d-neighborhood of a puzzle p. $N(p, d) = p' \in P$ where $||p' - p||_1 \leq 2 \times d$, where d is the number of letters changed (multiplied by 2 because changing a letter flips two bits in the matrix). Within small d-neighborhoods of a valid puzzle v, we should expect to see many p in V, with the percentage decreasing as d increases. Therefore, our approach to crossword generation is to start with a valid "seed" puzzle v_1 and choose a new valid puzzle v_2 from $D(v_1, d)$. We repeat this until a completely new puzzle has been created. This is similar to traversing the manifold like a word ladder (Fig. 3).

We began by randomly sampling $N(p, d)$ on each iteration, resulting in Algorithm 1.

Algorithm 1 Random Sampling

```
root = starting puzzle
for i in iterations do
    sample = random p ∈ N(root, d)
    root = sample if sample ∈ V else root
end
return root
```

Just by randomly sampling, we were able to beat all previous methods, by quickly generating 4×4, 5×5, and some larger, nearly interlocked puzzles. However, for newspaper sized puzzles, we were unable to attain completely new puzzles. We believe this is because long words act to separate manifolds. We define a valid puzzle v to be on the same d-resolution manifold (DRM) in the discrete space P as another valid puzzle v' if v is reachable from v' using Algorithm 1 with a search distance of d. All 2 letter words in our lexicon were reachable within 1 letter change, and therefore $V_{2\times2}$ is distributed as one DRM in $P_{2\times2}$ using $d \geq 1$. However, as the puzzle size increases and longer words are added, d must increase to keep V distributed in P as one DRM. The arg-max-minimum Hamming distance of a word of length n, is the word that is the most number of changes away from any other word of length n. The number of letters changed in any iteration of algorithm 1 must be at least as large as the largest word in the puzzles' max-minimum Hamming distance to keep V distributed as one manifold. This number can be almost as large as the word itself. For example, in our lexicon, the word JGSWPZZL has a Hamming distance of at least 5 between it and every other 8 letter word. The word EPIDIDYMODEFERENTIAL is at least 15 letter changes from any other 20 letter word. Therefore, for there to be one DRM in a puzzle space with a 20 letter word, d must be ≥ 15.

Unfortunately, as expected, increasing d also drastically decreases the percentage of valid puzzles v found in the d-neighborhood of v'. With a search depth of 3 in a 4×4 puzzle, Algorithm 1 found

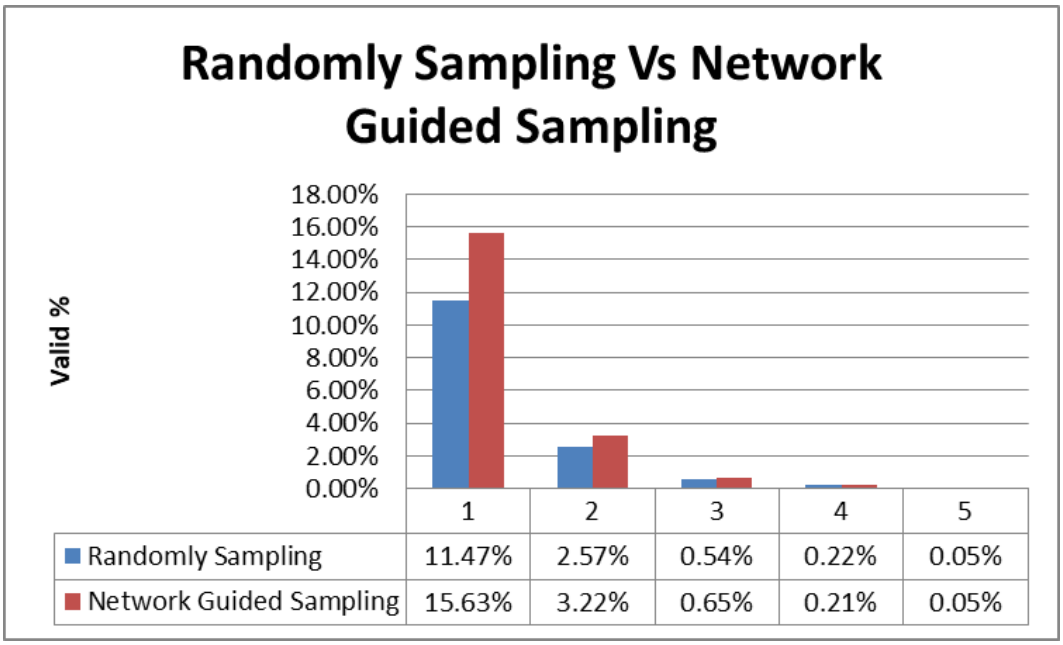

Figure 3: Success Rate vs Search Depth, Algorithm 1 vs Algorithm 2

a valid puzzle in fewer than 1% of iterations. Our solution was to train a neural network to choose where to sample in $N(v, d)$, guiding our search and allowing us to increase d without drastically increasing the runtime of our algorithm.

We had to generate our own data to train our network. As a proof of concept, we decided to start small, with a network that would analyze 4×4 fully interlocked boards and output the most promising letter changes. The inputs to the network were $4 \times 4 \times 26$ one-hot encoded valid puzzles generated by algorithm 1. We generated 1000 4×4 puzzles, 900 of which were used for training and 100 for validation. The training output values were created using a script that created a new $4 \times 4 \times 26$ matrix for every input matrix. Output matrices had a 1 at every (i, j, k) where changing cell (i, j) in puzzle v to letter k and leaving all other letters untouched led to a new valid puzzle v'. In essence, the output matrix represents the choices that can be made to reach all $v' \in N(v, 1)$. Our network architecture is very basic. A flattened 416 dimensional input is fed into 3 fully connected hidden layers with 400 nodes each, all with rectified linear unit (ReLU) activation functions. The output is another 416 dimensional vector with an element-wise sigmoid activation function, this is used to force each output value into a $[0, 1]$ range. The output vector is reshaped into a $4 \times 4 \times 26$ matrix. A value of 0.8 at $(1, 1, 10)$ indicates an 80% probability that changing the top left cell to a j will result in a valid puzzle. Since each probability is independent, an element-wise sigmoid function was used as opposed to a softmax function. We used a cross-entropy loss function and trained with stochastic gradient descent. Our learning rate was set to 0.001, our batch size was 50 and we ran for 3 epochs. We also trained a convolutional network and attained the same results and therefore chose the simpler architecture.

Using our network to guide our search we were able to modify Algorithm 1 to Algorithm 2.

Algorithm 2 Network Guided Sampling

```
root = starting puzzle
for i in iterations do
    sample = network chosen p ∈ N(root, d)
    root = sample if sample ∈ V else root
end
return root
```

3 Experiments

Algorithm 1 was sufficient to outperform all previous published attempts at crossword compilation. We were able to generate many 4×4, 5×5, and larger, nearly interlocked puzzles from seed puzzles (Fig. 4).

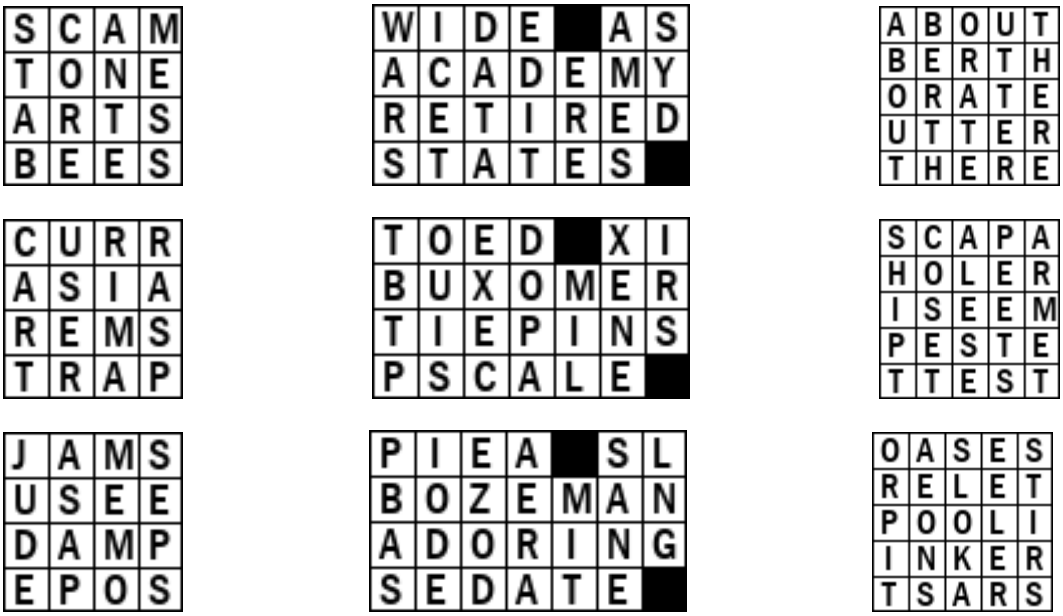

Figure 4: Seed Crosswords (top row) and Generated Crosswords (bottom two rows)

However, we were unable to generate newspaper sized puzzles because long words prevented us from reaching distant regions of P (Fig. 5). Some of the long words are highlighted. These particular words may be many letter changes from the nearest word of the same length. For example, the closest word to BEEFWELLINGTON is BESTSELLINGBOO which is 5 changes away. The closest word to VIENNASAUSAGES is POLISHSAUSAGES, which is 6 changes away. This puzzle requires $d \geq 6$ to reach every puzzle in P, and may in reality, require an even higher search depth than 6.

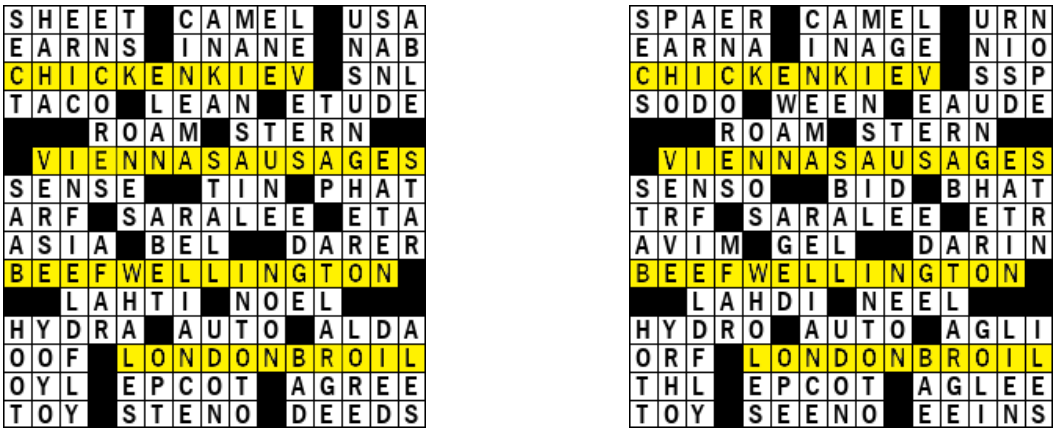

Figure 5: 15x15 Seed Puzzle (left) and Generated Puzzle (right)

By increasing our search radius, we believe we can reduce the number of DRM in P. However, increasing d drastically reduces the % of valid puzzles in $N(v, d)$, effectively limiting us to $d \leq 4$ (Fig. 3)

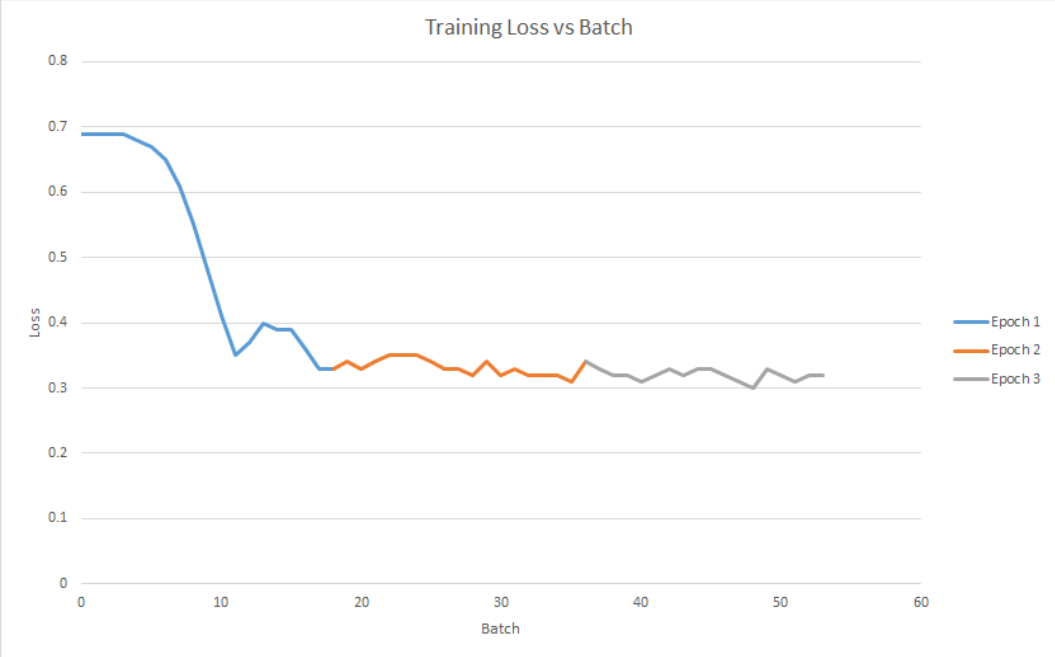

Figure 6: Cross-Entopy Loss vs Batch #

We tried to train a neural network to choose which letters to change to make sampling more efficient. The network trained very quickly (Fig. 6). We believe that the network learned to identify letter frequencies and letter positions, both relative to the board and other letters. For example, $p(i, j, 17)$ was almost always less than 0.01, regardless of i and j, because q's very infrequently make a valid word. On the other hand $p(i, j, 5)$ was almost always above 0.1. However $p(1, 1, 5)$ was usually much lower than $p(4, 4, 5)$ because e's are rarely the first letter in four letter words, but are frequently the last letter.

With Algorithm 2, we were able to sample valid puzzles more often than with random sampling for low values of d. However, for larger values of d, we were still unable to find valid puzzles at a reliable rate. There was no difference in neural network guided sampling and random sampling for $d \geq 4$ (Fig. 3).

This is not surprising, as the network was only trained to find valid puzzles in the 1-neighborhood of v. Training data, using the same technique as before, for multiple letter changes would require an output size of $O(N^d)$, where $N = n \times n \times 26$. Training a larger network also means needing to generate more data, however, the time needed to generate training data outputs also takes $O(N^d)$ time. Clearly a different strategy is needed for better neural network search guidance.

4 Conclusion

We were able to outperform all previous attempts at crossword compilation by generating larger American style puzzles. However, we were not able to accomplish our goal of generating a 15×15 American style puzzle. This is because we need to make many changes per iteration to change longer words, but making many changes at once seldom results in a valid puzzle.

Using a neural network to guide our search for a new valid puzzle in the d-neighborhood of the current puzzle did not work for larger values of d because the network was only trained with $d = 1$. Our training method is not suitable to be scaled up to higher values of d and it is not immediately clear to us how it could be.

For search depth d, using the current strategy, network size must be $O(N^d)$ and the time needed to enumerate which letter combinations result in a valid puzzle is also $O(N^d)$. An idea we had, would be to use a recurrent neural network to reduce the output size back to $O(N)$. The network could simply output for d steps, remembering the previous letter changes it had made. Instead of generating labeled training outputs with every possible valid combination of d changes, we could use a reinforcement learning approach. The network would simply explore the puzzle space, proposing a set of actions at every iteration, getting rewarded for actions that keep it on the valid puzzle manifold. Unfortunately, these ideas were too ambitious for us to implement with the time that we had.

5 Contributions

Julian Booth: Came up with the ideas. Implemented search and network. Wrote some of the paper.

Chun-Wei Chen: Wrote helper/utility functions. Formatted and edited the poster and the paper.

Stefano Macri: Wrote helper/utility functions. Got the poster printed. Helped make data tables and charts.

Chad Malla: Wrote function to get word counts of length n. Proofread and revise the paper.

6 References

[1] Smith, P. D. & Steen, S. Y. (1981) A prototype crossword compiler. *The Computer Journal, 24(2)*, pp. 107-111. Oxford University Press.

[2] Berghel, H. (1987) Crossword compilation with Horn clauses. *The Computer Journal, 30(2)*, pp. 183-188. Oxford University Press.

[3] Mazlack, L. J. (1976) Machine selection of elements in crossword puzzles: an application of computational linguistics. *SIAM Journal on Computing, 5(1)*, pp. 51-72. SIAM.

[4] Cayton, L. (2005) Algorithms for manifold learning. *Univ. of California at San Diego Tech. Rep 12, no. 1-17*, pp. 1.