Toroidal Grid Optimization via Gradient Descent

Jason Ken A

February 9, 2020

Contents

1 Problem Statement

1.1 Toroidal Graph

According to AZSPCS, a toroidal graph is defined as an $N \times N$ grid of unique tokens – here presented as IJ, where I and J represent the alphabetic indices (eg. A corresponds to 1) of the rows and columns respectively – whose edges wrap around. For $N = 4$, the grid is shown in Figure 1. The tokens outside of the square grid represent tokens which "wrap around" the grid (hence the toroidal grid).

	DA	DB	DC	DD	
AD	AA	AB	AC	AD	AA
BD	BA	BB	BC	BD	BA
CD	CA	CB	CC	CD	CA
DD	DA	DB	DC	DD	DA
	DA	DB	DC	DD	

Figure 1: A 4×4 toroidal grid

1.2 Evaluation Function

The goal of the challenge (and hence this essay) is to create a new grid – with tokens from the original grid to be arranged in any order – which minimizes a loss function (which quantifies failure) defined as follows:

1. For each pair of tokens, calculate the squared distance between them in the new grid,
2. multiply it with its squared distance within the original grid
3. The sum of these multiplications is the value of the loss

As is with all loss functions, the goal is to minimize the loss function.

1.2.1 Distance Metric

If 2 two-dimensional coordinates are defined as $s_1 = (x_1, y_1)$ and $s_2 = (x_2, y_2)$, the Euclidean distance d is defined as

$$d(s_1, s_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} = \sqrt{(\Delta x)^2 + (\Delta y)^2} \tag{1}$$

and the squared Euclidean distance d^2 evaluates to

$$d^2(s_1, s_2) = (\Delta x)^2 + (\Delta y)^2 \tag{2}$$

simplifying much of the computation.

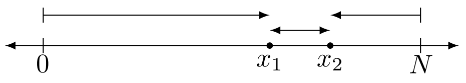


Figure 2: One-dimensional diagram of toroidal distance

But with a toroidal surface, both Δx and Δy have 2 possible values. A one-dimensional case with toroidal surface of length N is shown in Figure 2. These two distances can be written as

$$\begin{aligned}\Delta_1(x) &= x_2 - x_1 \\ \Delta_2(x) &= (x_1 - 0) + (N - x_2) = x_1 + N - x_2\end{aligned}$$

To obtain a general equation which works with both $x_1 > x_2$ and $x_1 < x_2$, we can write

$$\begin{aligned}\Delta_1(x) &= |x_2 - x_1| \\ \Delta_2(x) &= \min(x_1, x_2) + N - \max(x_1, x_2)\end{aligned}$$

where $|a|$ is the absolute value of a, $\min(a, b)$ and $\max(a, b)$ are defined as the minimum and maximum of a and b respectively. min and max are used to determine the "left-most" and the "right-most" numbers.

Since, the distance can only have 1 value, it is defined as

$$\Delta x = \min(\Delta_1(x), \Delta_2(x))$$

Generalizing it to 2 dimensions, we get

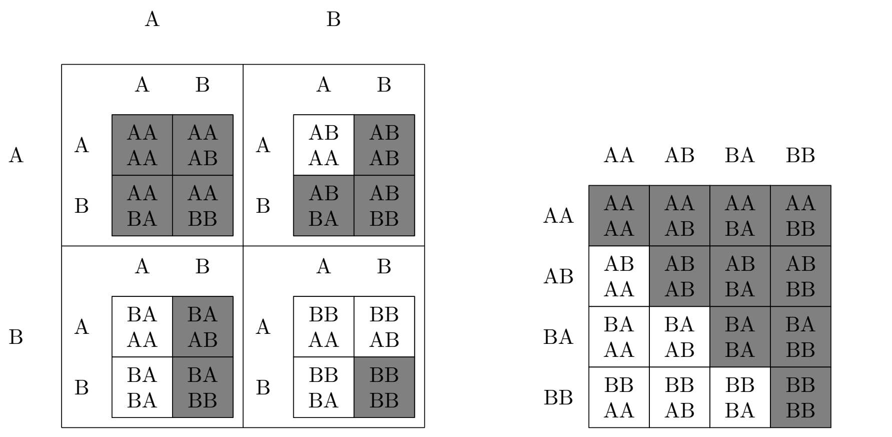
$$\begin{aligned}d(s_1, s_2) &= \sqrt{(\Delta x)^2 + (\Delta y)^2} \\ d^2(s_1, s_2) &= (\Delta x)^2 + (\Delta y)^2\end{aligned}$$

where d^2 is the distance metric to be used in our calculations.

1.2.2 Token Comparisons

To be able to perform comparisons for every possible pair of tokens, a grid of comparisons (between two tokens to obtain distances) is required. A viable approach would be to to represent the comparison in a 4-dimensional grid (or tensor), where for every token in the toroidal grid, we compare it to every token. An example of the structure of the comparison for a 2×2 grid is shown in Figure 3a. The white boxes denote duplicate comparisons (eg. $\substack{BA \\ AA}$ is identical to $\substack{AA \\ BA}$), because the distance is invariant to the order of the tokens.

But since removing the irregularly shaped duplicate entries will be a hassle, we can simplify the problem by reducing the dimensionality of the grid: from a 4-dimensional grid A_{ijkl} into a 2-dimensional grid $A_{ij,kl}$. An example is shown in Figure 3b. By doing so, we can easily remove the duplicate entries by multiplying it with the upper triangular matrix, see subsubsection 1.2.3. Let $C(X)$ denote a function mapping, returning this comparison grid for an input toroidal grid X. Note that if X is a grid of the form $N \times N$, $C(X)$ has the form $N^2 \times N^2$



(a) A_{ijkl}, a 4-dimensional comparison (b) $A_{ij,kl}$ a 2-dimensional comparison

Figure 3: Tensors C representing comparison grids of a 2×2 toroidal grid, where every element represents the toroidal distance between T_{ij} and T_{kl}, where T is the toroidal grid. Shaded cells denote unique comparisons.

1.2.3 Loss Function as Matrix Multiplications

Let $\odot$ denote the element-wise matrix multiplication, and O denotes the original grid. The loss function for toroidal grid X is equal to

$$\begin{aligned} L(x) &= \sum_{i}^{N^2} \sum_{j}^{N^2} C(X)_{ij} C(O)_{ij} U_{ij} \\ &= \sum C(X) \odot C(O) \odot U \end{aligned}$$

where $C(X), C(O), U$ are matrices of the form $N^2 \times N^2$, and U_{ij} is defined with the piece-wise defined function:

$$U_{ij} = \begin{cases} 1, & j \geq i \\ 0, & j < i \end{cases} \tag{3}$$

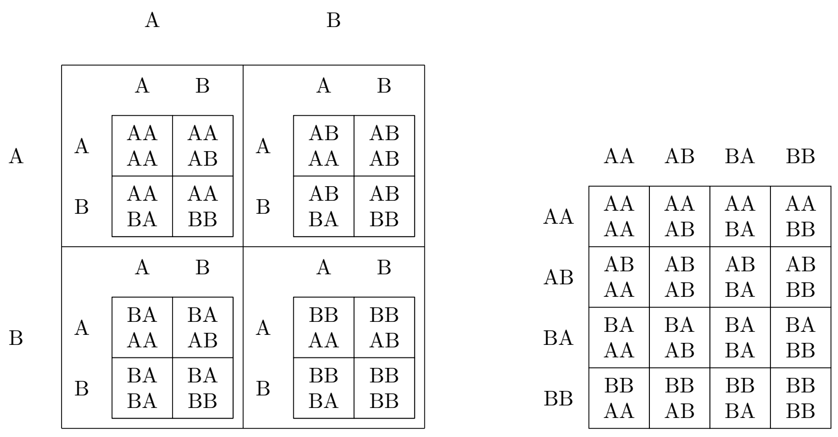
an example the matrix U with shape 4×4 is as follows:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

2 Superposition

Since the entries into the toroidal grid are discrete (eg. AA resolves to discrete coordinates within the grid), it is not possible to optimize the loss function. Therefore, relaxing the constraints to enable superposition – here defined as having a token being in multiple positions at once each with its own "probabilities" – is essential. In this essay, "probability" will not refer to the likeliness of a random event, but rather, the confidence of a token in its posiiton.

A simple method of allowing superposition, is by allowing any position to have any token value, which again, can be visualized as a 4-dimensional matrix, and 2-dimensional matrix, as seen in Figure 4. But this time, the elements of the grid do not represent the distances between tokens. But rather, they represent the probability of a token being placed in a certain position. Further constraints to limit the total probability to 1 will be discussed in a later section. Note that now the $y-$axis represents various positions (the same token in various places), and the $x-$axis represents the possible token values (all of which lie on the same toroid grid cell).



		A		B	
		A	B	A	B
A	A	AA AA	AA AB	AB AA	AB AB
	B	AA BA	AA BB	AB BA	AB BB
B	A	BA AA	BA AB	BB AA	BB AB
	B	BA BA	BA BB	BB BA	BB BB

(a) A_{ijkl}, a 4-dimensional superposition

	AA	AB	BA	BB
AA	AA AA	AA AB	AA BA	AA BB
AB	AB AA	AB AB	AB BA	AB BB
BA	BA AA	BA AB	BA BA	BA BB
BB	BB AA	BB AB	BB BA	BB BB

(b) $A_{ij,kl}$ a 2-dimensional superposition

Figure 4: Tensors S representing superpositions of a 2×2 toroidal grid, where every element S_{ijkl} represents the probability of the token KL being in the of position IJ in the original grid

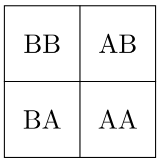
An example is shown in Figure 5

2.0.1 Generalization of the Loss Function

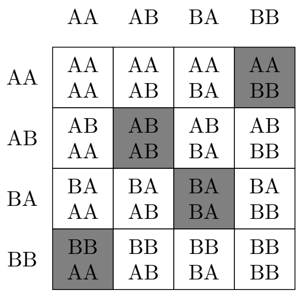
Defining the loss function for this formulation requires us to compare every value, in every position, to every value in every position. And we must do so, while taking both of their confidence's into account (ie. the distance metric should be scaled by their confidence). Therefore, the distance should be scaled with

$$S_{ab}S_{cd} \tag{5}$$

The distance between these two probabilities is defined as $C_{a,c}$, because only



(a) A_{ijkl}, an example toroidal grid



	AA	AB	BA	BB
AA	AA AA	AA AB	AA BA	▨ AA BB
AB	AB AA	▨ AB AB	AB BA	AB BB
BA	BA AA	BA AB	▨ BA BA	BA BB
BB	▨ BB AA	BB AB	BB BA	BB BB

(b) $A_{ij,kl}$ superposition of the grid on the left

Figure 5: Shaded cells represent cells with probability 1, the rest have probability 0

the indices a and c correspond to positions on the grid (thus being important in calculating distance), whereas b and d correspond to only the token itself.

Alike with the situation in subsubsection 1.2.2, we must prevent duplicate comparisons between values (not position), for example: each of $[\frac{AA}{AA}, \frac{AB}{AA}, \frac{BA}{AA}, \frac{BB}{AA}]$ should be compared to $[\frac{AA}{AB}, \frac{AB}{AB}, \frac{BA}{AB}, \frac{BB}{AB}]$, but not vice versa, because the values (not the positions) will have been compared already. Again, to do so, we must construct an upper triangular matrix U, of the shape $N^2 \times N^2$.

Therefore, the loss function can be written as

$$L(X) = \sum_{a}^{N^2} \sum_{b}^{N^2} \sum_{c}^{N^2} \sum_{d}^{N^2} S_{a,b} S_{c,d} C_{a,c} C_{b,d} U_{a,c} \tag{6}$$

$\sum_a^{N^2}$ Represents an iteration over Source Positions

$\sum_b^{N^2}$ Represents an iteration over Source Values

$\sum_c^{N^2}$ Represents an iteration over Target Positions

$\sum_d^{N^2}$ Represents an iteration over Target Values

$S_{a,b}$ Represents the Probability of the Source Position and Value

$S_{c,d}$ Represents the Probability of the Target Position and Value

$C_{a,c}$ Represents the Distance between Source and Target Position

$C_{b,d}$ Represents the Distance between the Source and Target Values in the original grid

$U_{a,c}$ Upper triangular matrix to remove redundant value comparisons

3 Optimization

3.1 Constraints

To be able to properly model probability, the sum of the probabilities of each token in all of its positions must be equal to one. Similarly, the sum of the probabilities of each position, in all of its values must be equal to one. Therefore, the sums of each of the rows and each of the columns must be equal to one, this is called a Doubly Stochastic Matrix. To be able to enforce this constraint, the Sinkhorn-Knopp algorithm was developed.

3.1.1 Sinkhorn-Knopp Algorithm

Given K an $n \times n$ non-negative matrix, by repetitively normalizing the rows and columns, a doubly stochastic matrix can be obtained. A single iteration can be seen through the following equation

$$K' = \left(\sum_{i}^{N} K_{ij} \right)^{-1} K \left(\sum_{j}^{N} K_{ij} \right)^{-1} \tag{7}$$

3.1.2 Non-negative Matrices

Since Sinkhorn-Knopp requires a non-negative matrix, and because negative probabilities make no sense, at every iteration of Sinkhorn-Knopp and Gradient Descent, the values have to be maintained above 0.

$$K'_{ij} = \max(K_{ij}, 0) \tag{8}$$

3.2 Gradient Descent

Gradient Descent is an algorithm to iteratively optimize a convex function, with knowledge of the derivative of the function with respect to all of the function parameters θ. In the case of optimizing a loss function, steps must be taken in the direction opposite to the gradient, therefore, the equation is as follows

$$\theta' = \theta - \alpha \frac{\delta}{\delta\theta} L(\theta) \tag{9}$$

where α is a parameter determining how big of a change there is between iterations of gradient descent.

3.2.1 Derivative of Loss Function coming Soon

4 Evaluation

4.1 Graphs of Loss over Time

4.2 Comparison to State of the Art Lower Bounds

4.3 Limitations of Computational Complexity

4.4 Limitations of How to Discretive Superpositions

4.5 Summary