Toroidal Grid Optimization via Gradient Descent

Jason Ken A

March 7, 2020

Contents

1 Introduction

Gradient descent is an iterative algorithm to optimize (to maximize or minimize) a continuous function. It does so by shifting the value of the paramaters in the direction of its gradient with respect to (w.r.t.) the function. In the context of minimization, it is defined as follows

$$\theta' = \theta - \alpha \frac{d}{d\theta} L(\theta)$$

where θ is the paramater to optimize with the goal of minimizing the value of the function L. α is a scaling factor representing how far the paramater should shift. This equation can be generalized to optimize many paramaters by utilizing partial derivatives instead. Anyhow, it is obvious why a continuous function is required: gradients are only defined with continuous functions.

The primary goal of this essay is to test whether a discrete loss function and its corresponding discrete paramaters can be generalized to a continuous loss function and continuous paramaters, with the purpose of finding optimum discrete variables. What better demonstration than its application on a difficult challenge? So, I chose to address this problem within the context of a programming contest called "Reversing Nearness" held by Al Zimmermann, which traditionally, had been approached with algorithms that do not utilize gradients, such as hill climbing (which modifies each paramater sequentially and keeps changes that optimize the function) and simulated annealing (which is essentially probabilistic hill climbing)[1]. But I wanted to do it with gradients! Because what else would my calculus classes be for.

The objective of Al Zimmermann's programming contest, is to rearrange discrete tokens within a discrete grid, to minimize the the a loss function which depends on the distances between these tokens on a toroidal surface. The intricacies of this problem are explained within the essay.

Hence, my research question: *Is gradient descent a viable approach for toroidal grid optimization?*

There are many techniques used within this essay to accomplish this goal. Among them are

1. Generalization of Euclidean distances to accomodate toroidal surfaces
2. Matric permutations to remove duplicate entries
3. "Superposition" to model discrete tokens being within multiple positions, with their respective "probabilities"[2]
4. Enforcement of probabilities (that they should sum to one), using the Sinkhorn-Knopp algorithm[1]
5. Use of Jacobian matrices for gradient descent

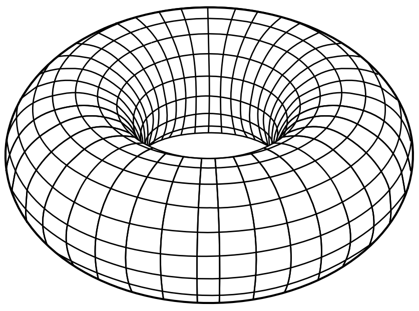
[1]both of which are beyond the scope of this essay

[2]Probabilities here does not refer to the chance of a random event occuring, but rather the "confidence" in which a token is in its position, or the extent in which the position affects the loss function

2 Problem Statement

2.1 Toroidal Grid

According to the problem description on AZsPCs,[2] the initial toroidal grid O is defined as an $N \times N$ grid of unique tokens which "wrap around" (implications are addressed in 2.2.1). These tokens can be arbitrarily defined, as long as they are unique, but for the sake of simplicity (and for reasons explained in 2.2.1), in this essay, a token is of the form by $\boldsymbol{IJ}$, where $\boldsymbol{I}$ and $\boldsymbol{J}$ are alphabetic representations of the indices of the rows and columns respectively, of a token, *within* O[3], eg. AC corresponds to the token in row 1, column 3 of O, DF corresponds to the row 4, column 6 of O, and so on. For $N = 4$, the grid is shown in Figure 1b. The tokens outside of the square grid represent tokens which wrap around the edges, resembling a toroidal surface as shown in Figure 1a.



(a) A Simple Toroid by Yassine Mrabet[3]

	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	

(b) A 4×4 *intial* toroidal grid

Figure 1: Representations of a toroidal grid

2.2 Evaluation Function

The goal of the challenge (and hence this essay) is to create a new grid X — with tokens rearranged arbitrarily from the initial grid O — which minimizes a loss function which is computed with the following procedure:

1. For each unique pair of tokens[4] (eg. $[AA, BA]$ is equivalent to $[BA, AA]$, so $[BA, AA]$ is excluded), calculate the squared distance between them in the new grid,
2. Multiply each of them with the squared distance between the pair of tokens within the original grid
3. The loss is the sum of all of these products

2.2.1 Distance Metric

To evaluate the loss function, a distance metric between two tokens must be established. And to do that, we must define the position of a token in a coordinate system. This is where the methodology we used to define the unique tokens comes in handy; the coordinates of the tokens are simply the indices of the token *within the particular grid* X. To avoid confusion between whether the rows or columns correspond to the x-coordinate or y-coordinate, a token within a grid, with indices (i, j) simply has the coordinate (i, j).[5]

With this definition, we can now define the distance metric. If two two-dimensional coordinates are defined as $s_1 = (x_1, y_1)$ and $s_2 = (x_2, y_2)$, the Euclidean distance d_{euclid} is defined as

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

[3]This is important because after rearranging the tokens, the identity of the token depends on its position within O, and not the rearranged position

[4]Comparisons between a token and itself do not affect the loss, since the distance between them is 0, therefore, their inclusion or exclusion does not affect the computation

[5]Note that this means we are not adhering to Cartesian coordinates, where the index (i, j) would have the coordinate (j, i).

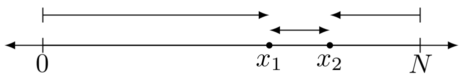


Figure 2: A one-dimensional diagram of toroidal distance, with the two arrows representing two possible distances

and the squared Euclidean distance d^2 evaluates to

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

simplifying much of the computation.

But to generalize Euclidean distance to the distance on a toroidal surface, we must consider several implications. First of all, Δx or Δy can have 2 possible values. Let us first consider the one-dimensional case. A toroidal surface of length N is illustrated in Figure 2. It can be shown that the corresponding possible values for Δx are as follows:

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

To obtain a general equation which works with both $x_1 > x_2$ and $x_1 < x_2$ (as we cannot assume $x_1 < x_2$), 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 between the values of a and b respectively. They can be defined with piece-wise functions as follows:

$$\min(a,b) = \begin{cases} a & \text{if } a \leq b \\ b & \text{if } a > b \end{cases} \qquad \max(a,b) = \begin{cases} a & \text{if } a \geq b \\ b & \text{if } a < b \end{cases}$$

min and max are used to determine the "left-most" and the "right-most" numbers.

Since the distance can only have one value, it is defined as the minimum of the two possible distances:

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

Generalizing it to 2 dimensions, we get

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

From now on, *distance* refers to d^2.

2.2.2 Token Comparisons

Within this essay, a *comparison* is defined as the distance between two tokens within a toroidal grid X. To be able to perform comparisons for every possible pair of tokens, a grid of comparisons on the grid is required. A viable approach would be to represent the comparison in a 4-dimensional grid (or tensor), where every token in the grid (first 2 dimensions), is compared to every token (next 2 dimensions). 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. $\frac{BA}{AA}$ is identical to $\frac{AA}{BA}$), because the distance is invariant to the order of the tokens.

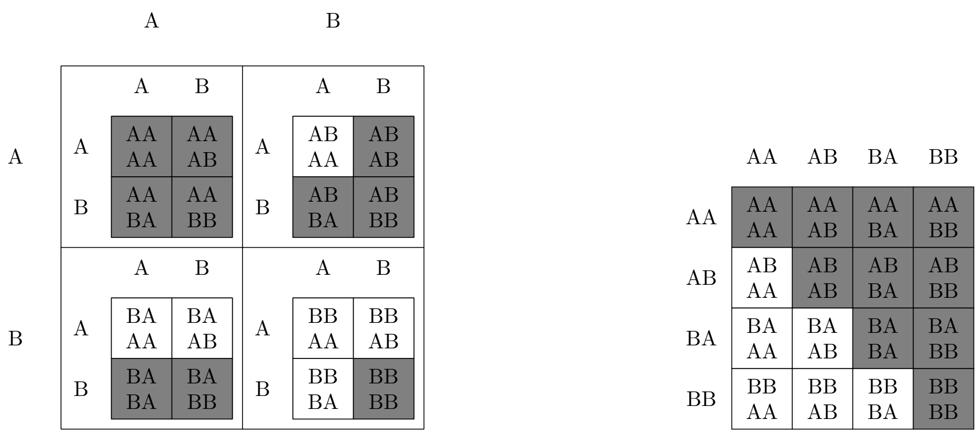
We can reduce the complexity[6] of the comparisons by reducing the dimensionality of the grid: from an $N \times N \times N \times N$ grid to a $N^2 \times N^2$ grid. The elements within the 4-dimensional matrix are in the form $C_{i,j,k,l}$, and the 2-dimensional grid in the form $C_{m,n}$. To do this conversion,

$$\begin{aligned} m &= (i-1) \times N + j \implies m \div N = i - 1 \text{ remainder } j \\ n &= (k-1) \times N + l \implies n \div N = j - 1 \text{ remainder } l \end{aligned}$$

-1 is to account for the rows index starting at 1.

Regardless, both represent the same thing: the distance between the tokens $\boldsymbol{IJ}$ and $\boldsymbol{KL}$. The two-dimensional representation of Figure 3a is shown in Figure 3b. Let C_{2d} denote a function mapping from the input toroidal grid X and returning the 2d-comparison grid $C_{2d}(X)$.

[6]ie. the number of Σs in 2.2.3



(a) 4-dimensional comparison with elements in the form $C(X)_{i,j,k,l}$

(b) 2-dimensional comparison with elements in the form $C(X)_{m,n}$

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

2.2.3 Loss Function in Matrix Form

The loss function for toroidal grid X is equal to

$$L(X) = \frac{1}{2} \sum_{m}^{N^2} \sum_{n}^{N^2} C(X)_{m,n} C(O)_{m,n}$$

where $C(X)$ and $C(O)$ are matrices of the form $N^2 \times N^2$. Note that the on the diagonal of $C(O)$ and $C(X)$ are comparisons of tokens against themselves, which are all equal to 0, because the distance between a point and itself is equal to 0. Therefore, to take into account only the unique comparisons we can simply divide the product between these 2 grids by 2.

2.2.4 A Comprehensive Example

3 Superposition

Since the entries into the toroidal grid are discrete (ie. discrete elements corresponding to discrete coordinates within the grid), it is not yet possible to optimize the loss function using gradients. Therefore, relaxing the constraints to enable "superposition" — here defined as having a token being in multiple positions, with each of its positions having its own *probability* — 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 position. Let S denote a superposition grid.

		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) $S_{i,j,k,l}$, 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) $S_{m,n}$ a 2-dimensional superposition

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

A simple method of allowing superposition, is by allowing any token to be in any position, which again, can be visualized as a 4-dimensional matrix, and 2-dimensional matrix, as seen in Figure 4, and again, to reduce complexity we choose the 2-dimensional model. But this time, in contrast to Figure 3, the elements of the grid *do not represent comparisons*, but rather, the element $S_{i,j,k,l}$ (in the case of a 4-dimensional superposition) or $S_{m,n}$[7] (in the case of a 2-dimensional superposition), represent the probability of the token $\boldsymbol{KL}$ inhabiting the position the original position (position in O) of token $\boldsymbol{IJ}$. Further constraints to enforce the concept of probabilities will be discussed in 4.1. Note, rows represent various positions within the grid O, and that columns represent the possible token values. An example of superposition for a discrete grid is shown in Figure 5.

By doing so, we are able to remove the limitations associated with a discrete grid. A single token value is now associated with all of the possible token positions, each with a different confidence. So now, the position of a token is fluid, determined by the confidences, we can now differentiate the loss function with respect to the confidences. *Instead of optimizing X, we optimize its continuous representation, S.*

BB	AB
BA	AA

(a) X, 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) $S_{m,n}$ superposition of the grid on the left

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

[7]Using the same conversion as in 2.2.2

3.0.1 Generalization of the Loss Function

Defining the loss function for this formulation requires us to compare *every* value within *every* position, to *every other*[8] value within *every* position. We must do so while taking into account both of the confidences within every comparison (ie. the distance metric should be scaled by their confidence). Therefore, for the token value b in the position a, compared to the token value d in the position c, the distance should be scaled by a factor of $S_{a,b}S_{c,d}$, because the elements within S reflect to what extent the value comparison affects the loss (ie. the distance metric).

The distance between these two probabilities is defined as $C_{a,c}$, because a and c correspond to the token position (and distance does not depend on token value). Whereas $C_{b,d}$ corresponds to the distance between the 2 tokens within O, since within O, the token values are equal to the token positions (2.1).

Alike with the situation in 2.2.2, we must prevent duplicate comparisons between values (not positions), for example: each of $[\frac{AA}{AA}, \frac{AB}{AA}, \frac{BA}{AA}, \frac{BB}{AA}]$ (token value AA) should be compared to $[\frac{AA}{AB}, \frac{AB}{AB}, \frac{BA}{AB}, \frac{BB}{AB}]$ (token value AB), but not vice versa, because the value comparisons have already been made.

Therefore, the loss function can be written as

$$L(S) = \frac{1}{2}\sum_{a}^{N^2}\sum_{b}^{N^2}\sum_{c}^{N^2}\sum_{d}^{N^2} S_{a,b}S_{c,d}C(O)_{a,c}C(O)_{b,d} \tag{3}$$

Similar to 2.2.3, we can simply divide the loss by 2, because all of the diagonal value comparisons: where $b = d$, are equal to 0, because it involves the term $C(O)_{b,d}$, the distance of a token against itself.

As a summary:

$\sum_{a}^{N^2}$	Represents an iteration over source position
$\sum_{b}^{N^2}$	Represents an iteration over source value
$\sum_{c}^{N^2}$	Represents an iteration over target position
$\sum_{d}^{N^2}$	Represents an iteration over target values
$S_{a,b}$	Represents the probability of the source salue in its position
$S_{c,d}$	Represents the probability of the target position and value
$C(O)_{a,c}$	Represents the distance between source and target positions
$C(O)_{b,d}$	Represents the distance between source and target values in O

3.1 A Comprehensive Example

[8] Ensuring unique *token value* comparisons

4 Optimization

4.1 Constraints

How do you minimize the function stated above? First of all, you can let all of the confidences be equal to 0, that way the loss function will be equal to 0. This behavior should be invalid, and hence it is addressed in 4.1.1. What about negative confidences? That way, the losses that result might be negative. Implications are discussed within 4.1.2.

4.1.1 Sinkhorn-Knopp Algorithm

Notice the characteristics of Figure 5b. The sum of its confidences within its rows and columns are all equal to 1. If the sum of the rows were not equal to 1, the probability of being within certain positions, would not be equal 1. And if the columns were not equal to one, that would mean that the probability of having certain token values would not equal to 1. The necessity for this constraint is reflected within the loss function itself. The loss could be zero if the confidences were zero. But this should be impossible, because there exists comparisons between tokens with non-zero values. So these constraints must be enforced.

A matrix that satisfies these constraints (rows and columns summing to one), is called a *doubly stochastic matrix.* A well-known algorithm to convert any non-negative matrix into a doubly stochastic matrix is called the Sinkhorn-Knopp algorithm (also called RAS).[1] There is a proof[4] and several papers[5, 6] analyzing its convergence. Nonetheless, the algorithm itself is simple: iteratively normalizing the rows and columns of a matrix.

Let K be an $n \times n$ non-negative matrix. A single iteration of RAS is defined as follows:

$$K' = \begin{bmatrix} (\Sigma_j^N K_{1,j})^{-1} & & \\ & \ddots & \\ & & (\Sigma_j^N K_{N,j})^{-1} \end{bmatrix} K \qquad \text{normalizing rows}$$

$$K'' = K' \begin{bmatrix} (\Sigma_i^N K'_{i,1})^{-1} & & \\ & \ddots & \\ & & (\Sigma_i^N K'_{i,N})^{-1} \end{bmatrix} \qquad \text{normalizing columns}$$

Blank entries are 0s. Here, normalizing rows simply means dividing each element by the sum of its row, and normalizing columns simply means dividing each element by the sum of its column. Figure 6 demonstrates the effectiveness of RAS. Graphed on the y-axis is the squared error, defined as:

$$E(X) = \sum_i^N \left(\left(\sum_j^N X_{ij} \right) - 1 \right)^2 + \sum_j^N \left(\left(\sum_i^N X_{ij} \right) - 1 \right)^2$$

Although this does not prove the convergence of RAS, it at least demonstrates its effectiveness.

4.1.2 Non-negative Matrices

Both because Sinkhorn-Knopp requires a non-negative matrix and because negative probabilities do not make sense, we have to ensure that the entries within S are above 0. To do that, we can simply take the absolute value of all the entries, before applying RAS.

$$K'_{ij} = |K_{ij}| \tag{4}$$

4.2 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

$$\theta' = \theta - \alpha \frac{\delta}{\delta\theta} L(\theta) \tag{5}$$

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

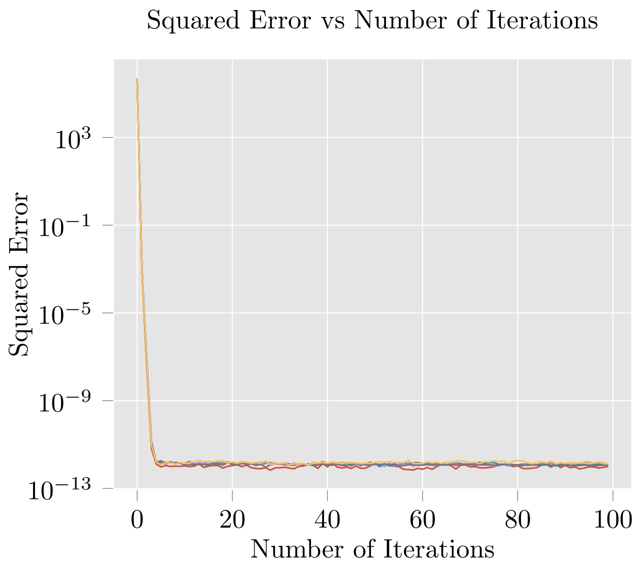


Figure 6: Demonstration of RAS convergence: 5 samples with $N = 100$, randomly generated from a uniform distribution $[0, 1)$. Error is equal to the sum of squared errors of the sums of the rows and columns from 1. Note the logarithmic scale.

4.2.1 Partial Derivative of Loss Function

The derivative of a multi-variable function w.r.t. a single variable, is called a partial-derivative, and is denoted by $\frac{\delta}{\delta x}$ instead of $\frac{d}{dx}$. Since our goal is to update each of the paramaters within S, we need to find a matrix which contains partial derivatives of the loss funtion against all of the elements.[9] That is, we are trying to find:

$$\frac{\delta L(S)}{\delta S} = \begin{bmatrix} \frac{\delta L(S)}{\delta S_{1,1}} & \cdots & \frac{\delta L(S)}{\delta S_{1,N^2}} \\ \vdots & \ddots & \vdots \\ \frac{\delta L(S)}{\delta S_{N^2,1}} & \cdots & \frac{\delta L(S)}{\delta S_{N^2,N^2}} \end{bmatrix}$$

To do that, we need to find the general solution to the partial derivative: $\frac{\delta L(S)}{\delta S_{i,j}}$. Recall that the loss function is defined as

$$L(S) = \frac{1}{2} \sum_a^{N^2} \sum_b^{N^2} \sum_c^{N^2} \sum_d^{N^2} S_{a,b} S_{c,d} C(O)_{a,c} C(O)_{b,d}$$

Recall that in partial derivatives, only the variable in question (ie. $S_{i,j}$) is treated as a variable, the rest are treated as constants. We can see that the term $S_{i,j}$ is included within the loss when $(a, b) = (i, j)$ or $(c, d) = (i, j)$ or both. The derivatives are as following:

$$\begin{aligned} \boldsymbol{A} = \frac{\delta L(S)}{\delta S_{i,j}} &= \frac{\delta}{\delta S_{i,j}} \left(\frac{1}{2} \sum_c^{N^2} \sum_d^{N^2} S_{i,j} S_{c,d} C(O)_{i,c} C(O)_{j,d} \right) && \text{First case} \\ &= \frac{1}{2} \sum_c^{N^2} \sum_d^{N^2} S_{c,d} C(O)_{i,c} C(O)_{j,d} \\ \boldsymbol{B} = \frac{\delta L(S)}{\delta S_{i,j}} &= \frac{\delta}{\delta S_{i,j}} \left(\frac{1}{2} \sum_a^{N^2} \sum_b^{N^2} S_{a,b} S_{i,j} C(O)_{a,i} C(O)_{b,j} \right) && \text{Second case} \\ &= \frac{1}{2} \sum_a^{N^2} \sum_b^{N^2} S_{a,b} C(O)_{a,i} C(O)_{b,j} \end{aligned}$$

[9]This is very similar to Jacobian matrices; except that the Jacobian takes the partial derivatives of a vector against a vector, but we are taking partial derivatives of a scalar against a matrix.

$$\boldsymbol{C} = \frac{\delta L(S)}{\delta S_{i,j}} = \frac{\delta}{\delta S_{i,j}} \left(\frac{1}{2} S_{a,b} S_{a,b} C(O)_{a,b} C(O)_{a,b} \right) \qquad \text{Third case}$$
$$= \frac{1}{2} \sum_{a}^{N^2} \sum_{b}^{N^2} S_{a,b} C(O)_{a,a} C(O)_{b,b}$$
$$= 0$$

$\boldsymbol{C}$ is 0 because of the comparisons of tokens against themselves. The partial derivative of $L(S)$ against $S_{i,j}$ is equal to $\boldsymbol{A} + \boldsymbol{B} - \boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B}$, because $\boldsymbol{A}$ and $\boldsymbol{B}$ represent all the cases in which $S_{i,j}$ is part of the loss function. Now, we can optimize S with the Gradient Descent:

$$S'_{i,j} = S_{i,j} - \alpha \left(\boldsymbol{A} + \boldsymbol{B} \right)$$

4.3 Optimization Procedure

With all of the steps above, we can now optimize S. The procedure used within this essay is as following:[10]

1. Randomly initialize S from a Uniform Distribution (equal probabilities for all values) on the interval $[0, 1)$
2. Repeat until convergence
   (a) Ensure S is a non-negative matrix
   (b) Single iteration of RAS
   (c) Single iteration of Gradient Descent

[10]The procedure is by no means optimal. The purpose of this essay is not to be optimal, but to provide insight on what is possible.

5 Evaluation

5.1 Graphs of Loss over Time

5.2 Comparison to State of the Art Lower Bounds

5.3 Limitations of Computational Complexity

5.4 Limitations of How to Discretive Superpositions

5.5 Summary

6 Acknowledgements

I would like to thank Emmerich Education Center for kindly providing computational resources used within this Extended Essay.

References

[1] Richard Sinkhorn and Paul Knopp. “Concerning nonnegative matrices and doubly stochastic matrices”. In: *Pacific Journal of Mathematics* 21.2 (1967), pp. 343–348.

[2] Al Zimmermann. *Reversing Nearness*. URL: `http://azspcs.com/Contest/Nearness` (visited on 02/25/2020).

[3] Yassine Mrabet. *A simple Torus*. 2007. URL: `https://upload.wikimedia.org/wikipedia/commons/c/c6/Simple_Torus.svg` (visited on 02/25/2020). License: Creative Commons BY-SA.

[4] Alberto Borobia and Rafael Cantó. “Matrix scaling: A geometric proof of Sinkhorn’s theorem”. In: *Linear algebra and its applications* 268 (1998), pp. 1–8.

[5] Deeparnab Chakrabarty and Sanjeev Khanna. “Better and simpler error analysis of the sinkhorn-knopp algorithm for matrix scaling”. In: *arXiv preprint arXiv:1801.02790* (2018).

[6] Philip A Knight. “The Sinkhorn–Knopp algorithm: convergence and applications”. In: *SIAM Journal on Matrix Analysis and Applications* 30.1 (2008), pp. 261–275.