Reversing Nearness via Gradient Descent

May 10, 2020

Mathematics

Is gradient descent a viable approach for Reversing Nearness?

Word Count: 3934

Contents

1 Introduction

Gradient descent is an iterative algorithm to optimize[1] a continuous function. It does so by shifting parameters in the direction of opposite of their gradients with respect to (w.r.t.) the function, that is:

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

Where θ is the parameter to optimize to minimize the value of function L, and α is the *learning rate*: an arbitrary scaling factor. Gradient descent can be generalized to multi-variable optimization through the use of partial derivatives within Jacobian matrices. Regardless, it self-evident why continuous functions are required.

The primary goal of this essay is to measure how capable gradient descent is within generalization: whether discrete solutions can be obtained for a discrete loss function by generalizing it into continuous spaces. "Reversing Nearness", a programming contest held by Al Zimmermann, proved to be suitable for this purpose, because to the best of the author's knowledge, had only been approached with non-gradient optimization techniques, such as hill climbing and simulated annealing.[2] For good reason, it is a discrete optimization problem. Yet, it has a relatively simple objective function: given a grid of tokens, "your task is to rearrange the tokens so that pairs of tokens that are near each other become far from each other and those that are far from each other become near."[1] The definitions of "tokens", "grids", and distance will be explained within the essay.

"It will be fun", I said, justifying all of my calculus classes. Hence, my research question: *Is gradient descent a viable approach for* Reversing Nearness*?*

1. **Short answer:** Yes

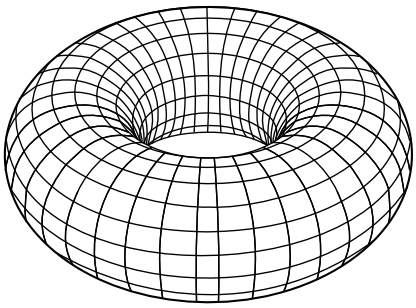
2. **Long answer:**

[1] usually in the context of minimization, hence 'descent'
[2] beyond the scope of this essay

2 Problem Statement

2.1 Toroidal Grid

As per AZsPCs[1] specifications, the initial toroidal grid O is defined as an $N \times N$ grid of unique tokens which "wrap around" the edges. A token is of the form $\boldsymbol{IJ}$, where $\boldsymbol{I}$ and $\boldsymbol{J}$ are alphabetic representations of the indices of the rows and columns respectively, of a token, *within* O,[3] e.g., AC corresponds to the token in row 1, column 3 of O, DF corresponds to the row 4, column 6 of O, etc. Fig. 1b shows O for $N = 4$. Tokens outside of the square grid represent tokens "wrapping around" the edges, resembling a toroidal surface as shown in Fig. 1a.



(a) A Simple Toroid by Yassine Mrabet[2]

	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 is to rearrange the tokens within O to form a new grid X that minimizes a loss function computed with the following procedure:

1. For each unique pair of tokens (e.g., $[AA, BA]$ is equivalent to $[BA, AA]$, so $[BA, AA]$ is omitted), calculate the squared distance between them in X,
2. Multiply each of these by the squared distance between the pair of tokens within O,
3. Sum all of these products,

[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. Subtract a lower-bound corresponding to the value of N (Appendix A)

2.3 Distance Metric

To evaluate the loss function, a distance metric between two tokens must be established, which necessitates a coordinate system. Here, *the coordinate of a token is defined as the indices of the token within* X i.e., any token within row 5, column 3 of X, has coordinates $(5, 3)$. Note that within O, coordinates, indices, and token representations are all equivalent.

Let two two-dimensional coordinates be $s_1 = (x_1, y_1)$ and $s_2 = (x_2, y_2)$. The Euclidean distance d_{euclid} is defined as,

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

On a toroidal surface however, Δx and Δy can each have 2 possible values. A one-dimensional toroidal surface of length N is illustrated in Fig. 2. The corresponding possible values for Δx are as follows,

$$\begin{aligned} \Delta_1(x) &= x_2 - x_1 \\ \Delta_2(x) &= (x_1 - 0) + (N - x_2) = x_1 + N - x_2 \end{aligned} \tag{2}$$

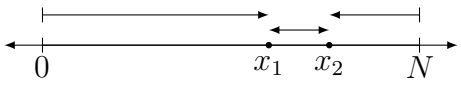


Figure 2: A one-dimensional diagram of toroidal distance, with the two arrows representing two possible distances

To obtain a general equation that works when $x_1 > x_2$:

$$\begin{aligned} \Delta_1(x) &= |x_2 - x_1| \\ \Delta_2(x) &= \min(x_1, x_2) + N - \max(x_1, x_2) \end{aligned} \tag{3}$$

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, that is,

$$\min(a, b) = \begin{cases} a, & \text{if } a \leq b \\ b, & \text{if } a > b \end{cases} \qquad \max(a, b) = \begin{cases} a, & \text{if } a \geq b \\ b, & \text{if } a < b \end{cases} \tag{4}$$

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

One-dimensional toroidal distance is then defined as

$$\Delta x = \min\left(\Delta_1(x), \Delta_2(x)\right) \tag{5}$$

And in two dimensions, the toroidal distance and squared toroidal distance are

$$\begin{aligned} d(s_1, s_2) &= \sqrt{(\Delta x)^2 + (\Delta y)^2} \\ d^2(s_1, s_2) &= (\Delta x)^2 + (\Delta y)^2 \end{aligned} \tag{6}$$

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

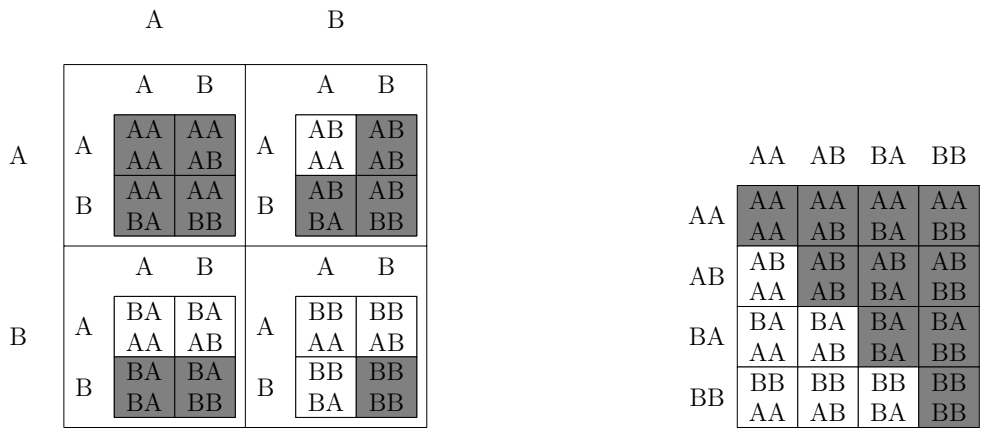
For example, the distance between $s_1 = (1, 4), s_2 = (2, 1)$ and $N = 5$ is calculated as follows:

$$\begin{aligned} \Delta_1(x) &= |2 - 1| = 1 \\ \Delta_2(x) &= \min(1, 2) + 5 - \max(1, 2) = 1 + 5 - 2 = 4 \\ \Delta_1(y) &= |1 - 4| = 3 \\ \Delta_2(y) &= \min(4, 1) + 5 - \max(4, 1) = 1 + 5 - 4 = 2 \\ \Delta(x) &= \min(1, 4) = 1 \\ \Delta(y) &= \min(3, 2) = 2 \\ d^2(s_1, s_2) &= 1^2 + 2^2 = 5 \end{aligned} \tag{7}$$

2.4 Token Comparisons

In order to compute distances between all possible pairs, a corresponding representation is required, i.e., a 4-dimensional matrix (or tensor) with elements being distances between *source* tokens (first 2 dimensions) and *target* tokens (next 2 dimensions). An example of the structure of the comparison for an $N = 2$ grid is shown in Fig. 3a.

The dimensionality of the grid can be reduced from a tensor into a matrix, to reduce complexity (e.g., number of Σs in 2.5) by applying the matrix-flattening scheme in Fig. 4 twice, on the source and target dimensions, reducing $C(X)$ from $N \times N \times N \times N$ to $N^2 \times N^2$. Elements within the 4-dimensional tensor are of form $C(X)_{i,j,k,l}$, whereas the



(a) (b)

Figure 3: Tensors $C(X)$ in the forms $C(X)_{i,j,k,l}$ and $C(X)_{m,n}$ respectively, representing comparison grids of an $N = 2$ toroidal grid, where every element represents the distance between tokens $\boldsymbol{IJ}$ and $\boldsymbol{KL}$ within X. Shaded cells denote unique comparisons.

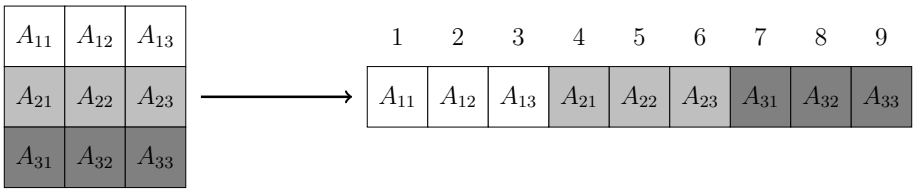


Figure 4: Matrix-flattening

2-dimensional elements are of form $C(X)_{m,n}$. To do this conversion,

$$m = (i-1) \times N + j \implies m \div N = i - 1 \text{ remainder } j$$
$$n = (k-1) \times N + l \implies n \div N = j - 1 \text{ remainder } l \tag{8}$$

Eq. 8 can be understood by realizing that $A_{2,3}$ (Fig. 4) is offset by $(2-1)$ groups of 3 and an additional 3 after flattening.

Note that both $C(X)_{i,j,k,l}$ and $C(X)_{m,n}$ represent the distance between tokens $\boldsymbol{IJ}$ and $\boldsymbol{KL}$ within X as defined within section 2.3, the difference lies only within their dimensionality. The two-dimensional representation of Fig. 3a is shown in Fig. 3b.

From now on, let $C(X)$ refer to the two dimensional comparison grid. Examples are shown in Fig. 5.

AA	AB
BA	BB

(a) O

$$\begin{array}{c} \\ AA \\ AB \\ BA \\ BB \end{array} \begin{array}{c} \begin{array}{cccc} AA & AB & BA & BB \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 2 \\ 1 & 0 & 2 & 1 \\ 1 & 2 & 0 & 1 \\ 2 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

(b) $C(O)$

AA	AB
BB	BA

(c) An example toroidal grid X

$$\begin{array}{c} \\ AA \\ AB \\ BA \\ BB \end{array} \begin{array}{c} \begin{array}{cccc} AA & AB & BA & BB \end{array} \\ \begin{bmatrix} 0 & 1 & 2 & 1 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 \end{bmatrix} \end{array}$$

(d) $C(X)$

Figure 5: Example toroidal grids and comparison grids

2.5 Loss Function in Matrix Form

The computation of products of only unique comparisons is greatly simplified due to the 2-dimensional representation (Fig. 3b) of $C(X)$, which is symmetric along the main diagonal i.e., unique comparisons (shaded cells) along the upper triangle are mirrored along the lower triangle i.e., $\begin{smallmatrix} AA \\ AB \end{smallmatrix}$ is mirrored by $\begin{smallmatrix} AB \\ AA \end{smallmatrix}$. Also, the elements of $C(O)$ and $C(X)$ are zero-valued along the diagonal (i.e., not contributing to the loss function) due to the distance between a point and itself being 0. Therefore, to calculate the loss function a simple factor of $\frac{1}{2}$ can be added, and the loss of X, $L(X)$ can be written as:

$$\begin{aligned} L(X) &= \frac{1}{2}\sum_{m}^{N^2}\sum_{n}^{N^2} C(X)_{m,n}C(O)_{m,n} - c \\ &= \frac{1}{2}\sum_{m}^{N^2}\sum_{n}^{N^2} (C(X) \odot C(O))_{m,n} - c \end{aligned} \tag{9}$$

Where $(\odot)$ denotes element-wise multiplication, that is[4]

$$C = A \odot B \implies C_{i,j} = A_{i,j}B_{i,j}\ \forall i,j \tag{10}$$

The loss function for Fig. 5 is computed as follows:

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

[4]every element within C is equal to the product of the elements of A and B within that position

$$
\begin{aligned}
&= \frac{1}{2}\sum_{m}^{N^2}\sum_{n}^{N^2}\left(\begin{bmatrix}0&1&2&1\\1&0&1&2\\2&1&0&1\\1&2&1&0\end{bmatrix}\odot\begin{bmatrix}0&1&1&2\\1&0&2&1\\1&2&0&1\\2&1&1&0\end{bmatrix}\right)_{m,n} - c \\
&= \frac{1}{2}\sum_{m}^{N^2}\sum_{n}^{N^2}\begin{bmatrix}0&1&2&2\\1&0&2&2\\2&2&0&1\\2&2&1&0\end{bmatrix}_{m,n} - c \\
&= \frac{1}{2}\cdot 20 - c \\
&= 10 - c
\end{aligned} \tag{11}
$$

3 Superposition

3.1 Definition

Since the entries of the toroidal grid are discrete (i.e., discrete tokens with discrete coordinates), it is not yet possible to apply gradient descent. Therefore, relaxing the constraints to enable "superposition" — here defined as having token *fragments* in multiple positions, each of the fragments having its own *weight* — is essential. A fragment here refers to a fraction of a token and weight refers to the literal fraction (i.e. numerical value). Inspiration was taken from the field of physics, where the positions of electrons are not indicated by coordinates, but probability density functions,[5] a concept called *quantum superposition*. This analogy will be taken further within 4.3.

The superposition grid S consists can also be visualized as a 4-dimensional matrix with elements representing the weights within all possible *token positions* within O (first 2 dimensions) of all possible *token values* (next 2 dimensions). Using the matrix flattening scheme from Eq. 8, a 2-dimensional representation is possible. The representations are identical to Fig. 3 (but without shaded cells), and in contrast to Fig. 3, the elements of the grid *do not represent distances*, but rather, the element $S_{i,j,k,l}$ or $S_{m,n}$ represent the weight of the fragment within position $\boldsymbol{IJ}$ of token $\boldsymbol{KL}$.

Let S denote the 2-dimensional representation. Note, within S, rows represent the

[5] probability as a function of position

AA	AB
BB	BA

(a) An example toroidal grid X

	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) $S_{m,n}$

Figure 6: Superposition of a toroidal grid. Shaded cells have weight 1, non-shaded cells have weight 0.

possible positions within O and columns represent the token values. An example of superposition for a discrete toroidal grid is shown in Fig. 6.

By doing so, the limitations associated with a discrete grid are sidestepped. All token values are associated with all positions, with continuous (real-valued) fragment weights. Therefore, the loss function can be differentiated w.r.t. the weights — *instead of optimizing X, we optimize its continuous representation, S.* Note that with a superposition grid with entries derived from a toroidal grid, the loss function will simplify into that in 2.5 as demonstrated within the next section.

3.2 Generalization of the Loss Function

Defining the loss function for this formulation requires measuring the distance between *every* token value within *every* position (i.e., all fragments), to *every other* token value (ensuring unique *value* comparisons) within *every* position, scaled by the weights of the fragments. That is, for token value b in position a, compared to the token value d in position c, the product of distances (in O and X) should be scaled by $S_{a,b}S_{c,d}$, because the weights within S reflect the extent to which a fragment should affect the loss (e.g., half a token should impact the loss half as much).

The distance between these two fragments in X is defined as $C(O)_{a,c}$, because a and c correspond to token position, whereas $C(O)_{b,d}$ represents the distance between the 2 tokens within O, since within O, the token values are equal to the token positions (section 2.1).

Alike with section 2.4, duplicate evaluations between values (not positions) must be

prevented, for example: each of $[\begin{smallmatrix}AA\\AA\end{smallmatrix}, \begin{smallmatrix}AB\\AA\end{smallmatrix}, \begin{smallmatrix}BA\\AA\end{smallmatrix}, \begin{smallmatrix}BB\\AA\end{smallmatrix}]$ (token value AA) should be compared to $[\begin{smallmatrix}AA\\AB\end{smallmatrix}, \begin{smallmatrix}AB\\AB\end{smallmatrix}, \begin{smallmatrix}BA\\AB\end{smallmatrix}, \begin{smallmatrix}BB\\AB\end{smallmatrix}]$ (token value AB), but not vice versa, because the value comparisons have already been made.

Therefore, the loss function can be written as

$$L(S) = \frac{1}{2}\sum_{a}^{N^2}\sum_{b}^{N^2}\sum_{c}^{N^2}\sum_{d}^{N^2} S_{a,b}S_{c,d}C(O)_{a,c}C(O)_{b,d} - c \tag{12}$$

Similar to section 2.5, we can simply divide the loss by 2, because duplicate value comparisons are only made when the values of b and d are swapped ($C(O)_{b,d}$ remains the same), and because $C(O)$ is zero-valued along the diagonal, where $b = d$, due to the inclusion of $C(O)_{b,d}$, the distance of a token against itself.

In 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 weight of fragment in position $\boldsymbol{A}$ of value $\boldsymbol{B}$,[6]

$S_{c,d}$ represents the weight of fragment in position $\boldsymbol{C}$ of value $\boldsymbol{D}$,[7]

$C(O)_{a,c}$ represents the distance between source and target positions,

$C(O)_{b,d}$ represents the distance between source and target values within O,

c is the lower bound constant (Appendix A)

In order to calculate the loss of Fig. 6b exhaustively with Eq. 12, $(N^2)^4$ iterations, or 256 iterations for $N = 2$, are required to go over the all the possible combinations of a, b, c, d. For brevity, only non-zero configurations of a, b, c, d (i.e., only if source and target weights are non-zero) are shown. Note that $C(O)$ is taken from Fig. 5b.

[6] expand it by undoing the matrix flattening scheme in Eq. 8

[7] see footnote footnote 6

a	b	c	d	$S_{a,b}$	$S_{c,d}$	$C(O)_{a,c}$	$C(O)_{b,d}$	Product
1	1	1	1	1	1	0	0	0
1	1	2	2	1	1	1	1	1
1	1	3	4	1	1	1	2	2
1	1	4	3	1	1	2	1	2
2	2	1	1	1	1	1	1	1
2	2	2	2	1	1	0	0	0
2	2	3	4	1	1	2	1	2
2	2	4	3	1	1	1	2	2
3	4	1	1	1	1	1	2	2
3	4	2	2	1	1	2	1	2
3	4	3	4	1	1	0	0	0
3	4	4	3	1	1	1	1	1
4	3	1	1	1	1	2	1	2
4	3	2	2	1	1	1	2	2
4	3	3	4	1	1	1	1	1
4	3	4	3	1	1	0	0	0
							Sum	20

Table 1: Superposition loss of Fig. 6b

Therefore, the loss is equal to

$$\begin{aligned} L(S) &= \frac{1}{2} \cdot 20 - c \\ &= 10 - c \end{aligned} \tag{13}$$

This is equivalent to the loss obtained in Eq. 11. Obviously, this loss function can accomodate non-discrete weights within the superposition grid. This example is purposefully simplistic.

4 Optimization

4.1 Gradient Descent

Gradient descent is an iterative optimization algorithm, utilizing the first derivative of the loss function L with respect to all function parameters θ. To recall, a single iteration of gradient descent is as follows:

$$\theta' = \theta - \alpha \frac{\delta}{\delta \theta} L(\theta) \tag{14}$$

α is the *learning rate*, an arbitrary positive scaling factor, determining the magnitude of the update w.r.t. the gradient.

Fig. 7 is a toy example of gradient descent, but it generalizes to more complicated problems such as Reversing Nearness.

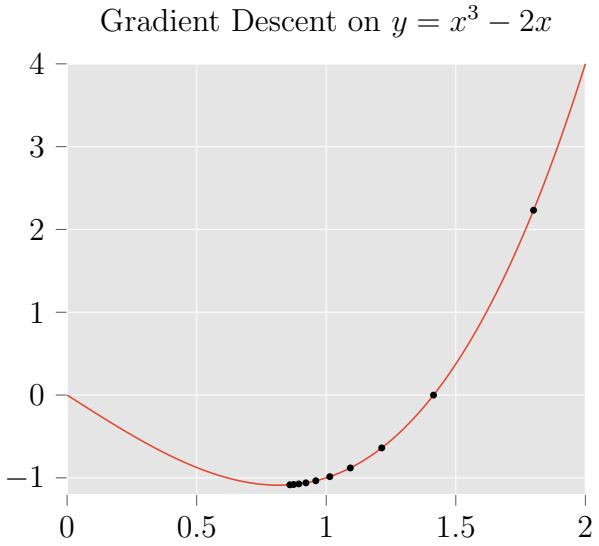


Figure 7: Demonstration of gradient descent convergence: 10 iterations with $\alpha = 5 \times 10^{-2}$ and $\theta_0 = 1.8$.

4.1.1 Partial Derivative of Loss Function

A partial derivative is the derivative of a multi-variable function w.r.t. a single variable, and is denoted by $\frac{\delta}{\delta x}$ instead of $\frac{d}{dx}$. Hence, our goal is to find the Jacobian matrix $\boldsymbol{J}$ of $L(S)$ w.r.t. S, defined as follows:

$$\boldsymbol{J} = \frac{\delta L(S)}{\delta S} = \begin{bmatrix} \frac{\delta L(S)}{\delta S_{1,1}} & \cdots & \frac{\delta L(S)}{\delta S_{1,N^2}} \\ \vdots & \ddots & \vdots \\ \frac{\delta L(S)}{\delta S_{N^2,1}} & \cdots & \frac{\delta L(S)}{\delta S_{N^2,N^2}} \end{bmatrix} \tag{15}$$

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

$$L(S) = \frac{1}{2} \sum_{a}^{N^2} \sum_{b}^{N^2} \sum_{c}^{N^2} \sum_{d}^{N^2} S_{a,b} S_{c,d} C(O)_{a,c} C(O)_{b,d} - c \tag{16}$$

When evaluating $\boldsymbol{J}_{i,j}$, only $S_{i,j}$ is treated as a variable, whereas others are treated as constants, evaluating to 0 after differentiation. Note that $S_{i,j}$ is only included within the loss when $(a, b) = (i, j)$ or $(c, d) = (i, j)$ or both. The derivatives of the terms of for the respective cases are as follows:

$$\begin{aligned} \boldsymbol{A} = \frac{\delta L(S)}{\delta S_{i,j}} &= \frac{\delta}{\delta S_{i,j}} \left(\frac{1}{2} \sum_{c}^{N^2} \sum_{d}^{N^2} S_{i,j} S_{c,d} C(O)_{i,c} C(O)_{j,d} \right) && \text{first case} \\ &= \frac{1}{2} \sum_{c}^{N^2} \sum_{d}^{N^2} S_{c,d} C(O)_{i,c} C(O)_{j,d} \end{aligned}$$

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

$$= \frac{1}{2} \sum_{a}^{N^2} \sum_{b}^{N^2} S_{a,b} C(O)_{a,i} C(O)_{b,j}$$

$$\boldsymbol{C} = \frac{\delta L(S)}{\delta S_{i,j}} = \frac{\delta}{\delta S_{i,j}} \left(\frac{1}{2} S_{i,j} S_{i,j} C(O)_{i,i} C(O)_{j,j} \right) \qquad \text{third case}$$

$$= S_{i,j} C(O)_{i,i} C(O)_{j,j}$$

$$= 0 \tag{17}$$

$\boldsymbol{C}$ is 0 due because it includes the distance of a token against itself. Therefore, $\frac{\delta L(S)}{\delta S_{i,j}}$, taking into account all 3 cases, is equal to $\boldsymbol{A} + \boldsymbol{B} - \boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B}$, but it can be simplified,

$$\frac{\delta L(S)}{\delta S_{i,j}} = \boldsymbol{A} + \boldsymbol{B}$$

$$= \frac{1}{2} \left(\sum_{c}^{N^2} \sum_{d}^{N^2} S_{c,d} C(O)_{i,c} C(O)_{j,d} + \sum_{a}^{N^2} \sum_{b}^{N^2} S_{a,b} C(O)_{a,i} C(O)_{b,j} \right)$$

$$= \frac{1}{2} \left(\sum_{a}^{N^2} \sum_{b}^{N^2} S_{a,b} C(O)_{i,a} C(O)_{j,b} + \sum_{a}^{N^2} \sum_{b}^{N^2} S_{a,b} C(O)_{a,i} C(O)_{b,j} \right) \tag{18}$$

$$= \frac{1}{2} \left(\sum_{a}^{N^2} \sum_{b}^{N^2} S_{a,b} C(O)_{a,i} C(O)_{j,b} + \sum_{a}^{N^2} \sum_{b}^{N^2} S_{a,b} C(O)_{a,i} C(O)_{j,b} \right) \tag{19}$$

$$= \sum_{a}^{N^2} \sum_{b}^{N^2} S_{a,b} C(O)_{a,i} C(O)_{j,b} \tag{20}$$

Eq. 18 utilizes the independence of the 2 summations, equating the summation variables. Eq. 19 utilizes the symmetric nature of $C(O)$, in order to have a and b along the same axes within all terms.

For example, $\frac{\delta L(S)}{\delta S_{2,3}}$ for Fig. 6b is calculated as follows,

$$\frac{\delta L(S)}{\delta S_{1,1}} = \sum_{a}^{N^2} \sum_{b}^{N^2} S_{a,b} C(O)_{a,i} C(O)_{j,b}$$

$$= \sum_{a}^{N^2} \sum_{b}^{N^2} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \odot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} \odot \begin{bmatrix} 1 & 2 & 0 & 1 \\ 1 & 2 & 0 & 1 \\ 1 & 2 & 0 & 1 \\ 1 & 2 & 0 & 1 \end{bmatrix} \tag{21}$$

$$= \sum_{a}^{N^2} \sum_{b}^{N^2} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$= 3 \tag{22}$$

$C(O)_{a,i}$ and $C(O)_{j,b}$ within Eq. 21 are repeated along the columns and rows respectively, due to i and j being constant w.r.t. a and b respectively. Computing $\boldsymbol{J}$ for Fig. 6b results in,

$$\boldsymbol{J} = \begin{bmatrix} 5 & 5 & 3 & 3 \\ 5 & 5 & 3 & 3 \\ 3 & 3 & 5 & 5 \\ 3 & 3 & 5 & 5 \end{bmatrix} \tag{23}$$

Now, a naive approach would be to apply gradient descent as follows:

$$S' = S - \alpha \boldsymbol{J} \tag{24}$$

But doing so would not take into the constraints involved within discrete optimization (that generalize to continuous optimization).

4.2 Generalization of Discrete Constraints

4.2.1 Doubly Stochastic Matrices

First of all, *negative weights are nonsensical*, since weights should reflect the portion of the token located within a certain position.

In the discrete problem, within X or O, every position has a *single* unique token, and every token has a *single* unique position. Generalizing this to superposition, note that all rows and columns should sum to 1 (e.g., Fig. 6b). Intuitively, a superposition grid divides the weight of each token value, into fragments whose weights sum to 1, with every position containing in total weight 1, albeit from fragments and not *whole* tokens.

Following from these observations, S is said to be doubly stochastic. That is, any matrix A with only non-negative values and

$$\sum_i A_{i,j} = \sum_j A_{i,j} = 1 \tag{25}$$

is doubly stochastic.[3] Therefore, to enforce this constraint, the superposition S must remain doubly stochastic after the gradient descent update, or else, the weights could be set to 0, causing a loss of $0 - c$.

4.2.2 Sinkhorn-Knopp Algorithm

This section explains the algorithm used within section 4.2.3.

A well-known algorithm to convert any non-negative matrix[8] into a doubly stochastic matrix is the Sinkhorn-Knopp algorithm (also named RAS).[4] There is a proof[5] and several papers analyzing its convergence.[6, 7] Nonetheless, the algorithm itself is simple: alternating the normalization of rows and columns of a matrix. Here, "normalization" is defined as forming a sum of 1, by dividing each element within each row or column by the sum of the row or column.

Let K be an $n \times n$ non-negative matrix.[9] A single iteration of RAS is defined as follows:

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

$$\text{RAS}(K) = K'' = K' \begin{bmatrix} (\Sigma_i^N K'_{i,1})^{-1} & & \\ & \ddots & \\ & & (\Sigma_i^N K'_{i,N})^{-1} \end{bmatrix} \quad \text{normalizing columns} \tag{26}$$

The scaling matrices are diagonal (non-diagonal elements are 0s). Let $\text{RAS}^n(K)$ indicate n iterations of RAS on K, i.e., $\text{RAS}^2(K) = \text{RAS}(\text{RAS}(K))$.

Fig. 8 demonstrates the effectiveness of RAS in normalizing randomly sampled matrices. Graphed on the y-axis is the error: the squared distance between the sums of the rows and columns, and 1, defined as,

$$E(X) = \sum_i^N \left(\left(\sum_j^N X_{i,j} \right) - 1 \right)^2 + \sum_j^N \left(\left(\sum_i^N X_{i,j} \right) - 1 \right)^2 \tag{27}$$

An example of a single iteration of RAS:

$$K = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

$$K' = \begin{bmatrix} \frac{1}{6} & 0 & 0 \\ 0 & \frac{1}{9} & 0 \\ 0 & 0 & \frac{1}{12} \end{bmatrix} \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{3} & \frac{2}{3} \\ \frac{1}{9} & \frac{1}{3} & \frac{5}{9} \\ \frac{1}{6} & \frac{1}{3} & \frac{1}{2} \end{bmatrix}$$

[8] having at least 1 positive value in each row and column

[9] see footnote 8

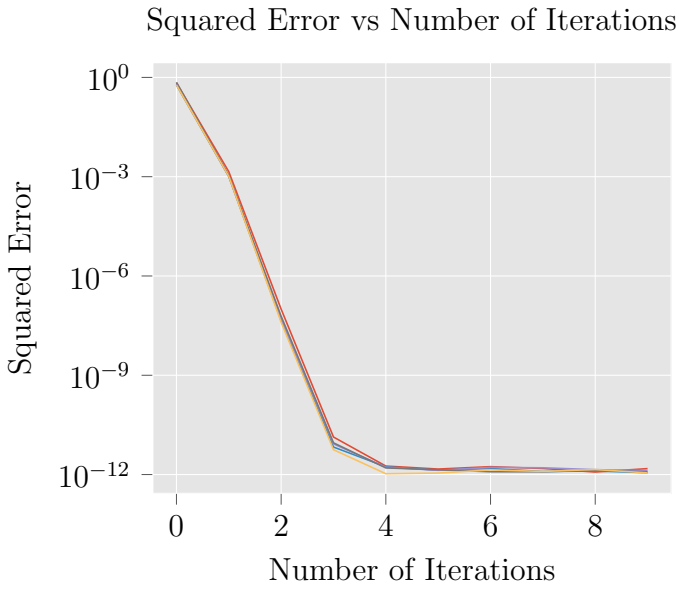


Figure 8: Demonstration of RAS convergence: RAS applied on 5 randomly generated $N \times N$ matrices with $N = 100$, sampled from a uniform distribution $[0, \frac{2}{N})$ (mean $\frac{1}{N}$). Note the logarithmic scale.

$$\text{RAS}(K) = K'' = \begin{bmatrix} 0 & \frac{1}{3} & \frac{2}{3} \\ \frac{1}{9} & \frac{1}{3} & \frac{5}{9} \\ \frac{1}{6} & \frac{1}{3} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} \frac{18}{5} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{18}{31} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{3} & \frac{12}{31} \\ \frac{2}{5} & \frac{1}{3} & \frac{10}{31} \\ \frac{3}{5} & \frac{1}{3} & \frac{9}{31} \end{bmatrix} \tag{28}$$

Note that the sums of rows and columns of RAS(K) are *closer* to 1 than K.

4.2.3 Zero Line-Sum Modified Jacobian

To maintain the doubly stochastic nature of the superposition grid within gradient descent update (Eq. 24), $\boldsymbol{J}$ must be modified into a zero line-sum (ZLS) matrix.[8] A ZLS matrix has all rows and columns summing to 0, that is, a matrix A is ZLS if and only if

$$\sum_i A_{i,j} = \sum_j A_{i,j} = 0 \tag{29}$$

Intuitively, this means that the gradient update should not change the sum of the weights of any token value or within a position (to maintain a doubly stochastic S).

As stated within [8], a ZLS can be obtained by taking the difference of 2 doubly stochastic matrices.

Proof. Let A and B be doubly stochastic matrices.

$$\sum_i (A_{i,j} - B_{i,j}) = \sum_i A_{i,j} - \sum_i B_{i,j} = 1 - 1 = 0$$

$$\sum_j (A_{i,j} - B_{i,j}) = \sum_j A_{i,j} - \sum_j B_{i,j} = 1 - 1 = 0 \tag{30}$$

Therefore, $A - B$ is ZLS. □

Hence, $\boldsymbol{J}$ can approximate a doubly stochastic matrix through the RAS algorithm,[10] and by subtracting it by another doubly stochastic matrix, a ZLS-$\boldsymbol{J}$ can be obtained. This second doubly stochastic matrix D can be easily obtained by scaling a ones-matrix (matrix filled with ones). Let D with dimensions $N^2 \times N^2$ be defined as follows:

$$\begin{aligned} D_{i,j} &= \frac{1}{N^2} \\ \sum_i^{N^2} D_{i,j} = \sum_j^{N^2} D_{i,j} &= N^2 \times \frac{1}{N^2} = 1 \qquad\qquad D \text{ is doubly stochastic} \end{aligned} \tag{31}$$

Because D is uniform (all elements have identical values), the relative magnitudes of the elements within $\text{RAS}^n(\boldsymbol{J})$ w.r.t. other elements are preserved, i.e.,

$$\text{RAS}^n(\boldsymbol{J})_{a,b} > \text{RAS}^n(\boldsymbol{J})_{c,d} \iff \text{RAS}^n(\boldsymbol{J})_{a,b} - D_{a,b} > \text{RAS}^n(\boldsymbol{J})_{c,d} - D_{c,d} \tag{32}$$

Let $\boldsymbol{J}' = \text{RAS}^n(\boldsymbol{J}) - D$. For $\boldsymbol{J}$ in Eq. 23, and $n = 1$ it is computed as follows:

$$\begin{aligned} \boldsymbol{J}' &= \text{RAS}^1\left(\begin{bmatrix} 5 & 5 & 3 & 3 \\ 5 & 5 & 3 & 3 \\ 3 & 3 & 5 & 5 \\ 3 & 3 & 5 & 5 \end{bmatrix}\right) - \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \\ &= \frac{1}{16}\begin{bmatrix} 5 & 5 & 3 & 3 \\ 5 & 5 & 3 & 3 \\ 3 & 3 & 5 & 5 \\ 3 & 3 & 5 & 5 \end{bmatrix} - \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \frac{1}{16}\begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \end{aligned} \tag{33}$$

4.2.4 Non-negative Matrices

Assuming S is doubly stochastic, and with ZLS-$\boldsymbol{J}'$, the result of the gradient descent update $S' = S - \alpha\boldsymbol{J}'$, has rows and columns summing to 1. For example, with S and $\boldsymbol{J}$ from Fig. 6b and Eq. 33 respectively, and $\alpha = 1$,

$$S' = S - \alpha\boldsymbol{J}' \tag{34}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} - \frac{1}{16}\begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix} = \frac{1}{16}\begin{bmatrix} 15 & -1 & 1 & 1 \\ -1 & 15 & 1 & 1 \\ 1 & 1 & -1 & 15 \\ 1 & 1 & 15 & -1 \end{bmatrix} \tag{35}$$

[10] the Jacobian of the loss function is non-negative because S and $C(O)$ are non-negative, so RAS is applicable

But S' is not necessarily doubly stochastic, because $S'_{i,j} \geq 0$ is not guaranteed, as shown in Eq. 35. Given that S' has dimension $N^2 \times N^2$ the following procedure corrects negative entries while still maintaining the rows and columns summing to 1,

$$S''_{i,j} = \begin{cases} (S'_{i,j} - \min S') \times \frac{1}{1 - N^2 \min S'}, & \text{if } \exists a, b \quad S'_{a,b} < 0 \\ S'_{i,j}, & \text{if } \nexists a, b \quad S'_{a,b} < 0 \end{cases} \tag{36}$$

$\min S'$ refers to the smallest value within S'.[11] $(S'_{i,j} - \min S')$ removes negative entries, adding $(-N^2 \min S')$ to the sums of the rows and columns; dividing by the new sum restores the doubly stochastic nature of S'. S'' for Eq. 35 is calculated as follows:

$$\begin{aligned} S'' &= \frac{1}{1 + 4(\frac{1}{16})} \left(\frac{1}{16} \begin{bmatrix} 15 & -1 & 1 & 1 \\ -1 & 15 & 1 & 1 \\ 1 & 1 & -1 & 15 \\ 1 & 1 & 15 & -1 \end{bmatrix} + \frac{1}{16} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \right) \\ &= \begin{bmatrix} \frac{4}{5} & 0 & \frac{1}{10} & \frac{1}{10} \\ 0 & \frac{4}{5} & \frac{1}{10} & \frac{1}{10} \\ \frac{1}{10} & \frac{1}{10} & 0 & \frac{4}{5} \\ \frac{1}{10} & \frac{1}{10} & \frac{4}{5} & 0 \end{bmatrix} \end{aligned} \tag{37}$$

With this, S can be optimized by gradient descent.

4.3 Generalization to Discrete Solutions

Note that only the continuous representation S has been optimized, and not the discrete solution X, which has been the goal of the essay. Expanding on the idea of quantum superposition within physics, by observing (or revealing) the position of an electron, the probability density function of the electron collapses into a single point. Within this essay, the probability density function has been represented by the weights of fragments within various positions. For simplicity, instead of sampling from a distribution, revealing a token value will place the entire weight of the token within the position of the fragment with the highest weight (the most likely position).

Taking this analogy even further, the tokens are *entangled*. In physics, if two electrons are entangled, revealing the *spin* of one electron instantaneously reveals the spin of the other: if one is found to be spin up, the other electron has spin down, and vice versa.

[11] $\exists a, b \quad S'_{a,b} < 0$ means "exists a, b such that $S'_{a,b}$ is negative"

The system within this essay involves N "electrons" (tokens), and when the position of a token is revealed, fragments of other tokens within the same position are removed — no two tokens should inhabit the same position, adhering to the constraints of discrete optimization. Also, the fragments of revealed tokens (columns of S) and fragments within revealed positions (rows of S) are not subject to optimization since they are *fixed*.

Let R be the set of indices (rows and columns) of revealed fragments, initially empty. The procedure for revealing a token is defined as follows:

$$\begin{aligned} R \leftarrow & R \cup \arg\max_{m,n} S_{m,n} \\ \text{such that } & (\nexists a \ \ (a,m) \in R) \wedge (\nexists b \ \ (n,b) \in R) \end{aligned} \tag{38}$$

$\arg\max_{m,n} S_{m,n}$ returns the indices of the fragment with the highest weight that does not share a token value or token position with any already revealed fragment. If there are multiple fragments with the largest value, any fragment can be chosen. $R \cup \arg\max_{m,n} S_{m,n}$ is the set of already revealed fragments including the newly revealed fragment, and $A \leftarrow B$ indicates a reassignment: the new value of A is B.

Note that the previous procedure only works when $\arg\max$ is defined, *when there are tokens to reveal*, when $|R| < N$, where $|R|$ denotes cardinality, (i.e., the number of indices within R).

For example, with

$$S = \begin{bmatrix} 5 & 4 & 1 & 0 \\ 4 & 1 & 3 & 0 \\ 3 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{39}$$

revealing all tokens sequentially will result in $R = \{(1,1),(2,3),(3,2),(4,4)\}$, revealed in that order. Coordinates can be restored into tokens by applying Eq. 8.

To only perform optimization on non-revealed token values and columns, the revealed rows and columns can be removed from $\boldsymbol{J}$, to obtain $\boldsymbol{J}_{cut}$ and $\boldsymbol{J}'_{cut}$ (with D adjusted to have $D_{i,j} = \frac{1}{N^2-|R|}$), which can then be expanded to its original dimensions $N^2 \times N^2$, by filling the revealed rows and columns with 0s, effectively preventing optimization of revealed rows and columns.

For example, with $\boldsymbol{J}$ from Eq. 23 and $R = \{(1,1)\}$ and $n = 3$,

$$\boldsymbol{J}_{cut} = \begin{bmatrix} 5 & 3 & 3 \\ 3 & 5 & 5 \\ 3 & 5 & 5 \end{bmatrix}$$

$$\begin{aligned} \boldsymbol{J}'_{cut} &= \mathrm{RAS}^n(\boldsymbol{J}_{cut}) - D \\ &= \begin{bmatrix} 0.4884 & 0.2558 & 0.2558 \\ 0.2558 & 0.3721 & 0.3721 \\ 0.2558 & 0.3721 & 0.3721 \end{bmatrix} - \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \\ &\approx \begin{bmatrix} 0.1551 & -0.0775 & -0.0775 \\ -0.0775 & 0.0389 & 0.0389 \\ -0.0775 & 0.0389 & 0.0389 \end{bmatrix} \end{aligned} \tag{40}$$

$$\boldsymbol{J}'_{final} \approx \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0.1551 & -0.0775 & -0.0775 \\ 0 & -0.0775 & 0.0389 & 0.0389 \\ 0 & -0.0775 & 0.0389 & 0.0389 \end{bmatrix} \tag{41}$$

After which the methodology in section 4.2.4 can be used with $\boldsymbol{J}'_{final}$ to obtain S' and S'', since $\boldsymbol{J}'_{final}$ is still ZLS.

4.4 Optimization Procedure

4.4.1 Learning Rate

α could simply be 1. But note that as N increases, the magnitude of each element within $\boldsymbol{J}'$ decreases due to RAS scaling (N^2 non-negative elements have to sum to 1) and ZLS offsetting. The following is an attempt to formulate a "one size fits all" solution for α for all N, without requiring human fine-tuning. Recall Eq. 34, now modified to accomodate revealed tokens,

$$S' = S - \alpha \boldsymbol{J}'_{final} \tag{42}$$

Let A be the set of possible values of α i.e., the α required to result in a 0-valued $S'_{i,j}$, *only if* $\boldsymbol{J}'_{final_{i,j}}$ *is positive* (preventing $\alpha < 0$ or a division by 0). That is,

$$\begin{aligned} S &= \alpha \boldsymbol{J}'_{final} \\ \mathrm{A} &= \left\{ \frac{S_{i,j}}{\boldsymbol{J}'_{final_{i,j}}} \middle| 1 \le i,j \le N^2, \boldsymbol{J}'_{final_{i,j}} > 0 \right\} \end{aligned} \tag{43}$$

α is then simply chosen to be the arithmetic mean of A. Intuitively, this process selects the α that removes about half of the fragments that contribute to the loss negatively, i.e.,

$\boldsymbol{J'}_{final_{i,j}} > 0$). Furthermore, overshooting α, causing $S'_{i,j} < 0$, will be handled by Eq. 36.

For example, for S from Fig. 6b and $\boldsymbol{J'}_{final_{i,j}}$ from Eq. 41,

$$\begin{aligned} \mathrm{A} &= \left\{ \frac{1}{0.1551}, \frac{0}{0.0389}, \frac{1}{0.0389}, \frac{1}{0.0389}, \frac{0}{0.0389} \right\} \\ \alpha &\approx 11.57 \end{aligned} \tag{44}$$

4.4.2 Superposition Initialization

Recall the optimization of S requires S to be doubly stochastic, and the steps in section 3.2 have ensured the gradient descent update maintains the doubly stochastic nature of S, *assuming S is initially doubly stochastic.* This section aims to enforce that.

When no tokens have been revealed, $S = D$ (Eq. 31). To work with an arbitrary amount of revealed tokens, S with dimensions $N^2 \times N^2$ is defined as follows:

$$S_{i,j} = \begin{cases} \frac{1}{N^2 - |R|}, & \text{if } (\nexists a \ \ (a,j) \in R) \wedge (\nexists b \ \ (i,b) \in R) \text{ if neither } i \text{ or } j \text{ have been revealed} \\ 0, & \text{if } (\exists a \ \ (a,j) \in R) \oplus (\exists b \ \ (i,b) \in R) \text{ if only one of } i \text{ or } j \text{ has been revealed} \\ 1, & \text{if } (\exists a \ \ (a,j) \in R) \wedge (\exists b \ \ (i,b) \in R) \text{ if } i,j \text{ is a revealed token} \end{cases} \tag{45}$$

Where $\wedge$ denotes a *logical and* (true if both premises are true), and $\oplus$ is *a logical XOR* (true if *only 1* premise is true). For example, with $R = \{(3,4),(2,1)\}$ and $N = 2$,

$$S = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix} \tag{46}$$

The doubly stochastic nature of S is intuitive, rows and columns which are revealed have only 1 weight, 1, from the revealed token, and the weight of unrevealed rows and columns are shared between fragments without revealed rows and columns.

4.4.3 Optimization Loop

The following is a summary of the optimization procedure:

1. For each element in $\{1, 2, \ldots, N^2\}$ (revealing every token):
   (a) Initialize S (Eq. 45)

(b) For n_{optim} steps:

i. Calculate $L(S)$ (Eq. 12)

ii. Calculate $\boldsymbol{J}$ (Eq. 15) and $\boldsymbol{J}_{cut}$ (Eq. 40)

iii. Calculate α (Eq. 44)

iv. Calculate ZLS-$\boldsymbol{J}'_{cut}$ (Eq. 33) and $\boldsymbol{J}'_{final}$ (Eq. 41)

v. Obtain S' from gradient descent update (Eq. 42)

vi. Ensure doubly stochastic S'' (Eq. 36)

(c) Reveal token (Eq. 38)

2. Obtain final X from R using Eq. 8

5 Evaluation

5.1 Graphs of Loss over Time

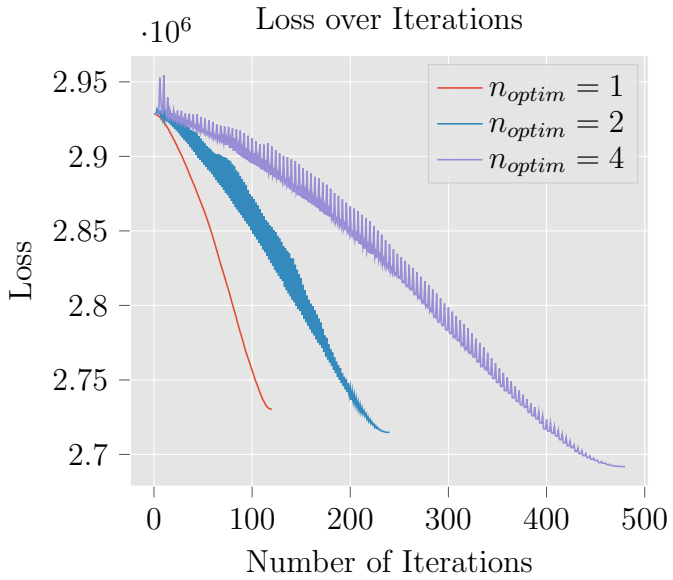


Figure 9: Loss over number of gradient descent iterations for $N = 11$ for various n_{optim}, and $n = 5$. *Lower bound constant has not been subtracted yet.*

Fig. 9 shows the loss over time for $N = 11$ for various n_{optim} with $n = 5$ (number of RAS iterations). All of the tested values of N from $[6, 30]$ had similar curves. The sharp discontinuities within the curves when $n_{optim} > 1$ are attributed to the reinitialization of

S, after revealing each of the N^2 tokens. Furthermore, it can be observed that a higher n_{optim} leads to a lower final loss.

5.2 Comparison to State of the Art

N	Loss	N	Loss	N	Loss
6	5526	14	5 457 848	22	203 877 974
7	17 779	15	9 164 700	23	286 960 950
8	57 152	16	15 891 088	24	409 173 438
9	144 459	17	25 152 826	25	560 363 762
10	362 950	18	40 901 354	26	776 271 362
11	740 798	19	61 784 724	27	1 039 341 134
12	1 585 264	20	95 115 180	28	1 404 785 310
13	2 888 120	21	138 133 813	29	1 843 328 926
				30	2 439 441 116

Table 2: SoTA results on Reversing Nearness obtained from [1]

N	n_{optim}	Loss	N	n_{optim}	Loss	N	n_{optim}	Loss
6	4	6088	14	4	5 986 496	22	2	223 649 376
7	4	19 050	15	4	9 860 722	23	2	308 724 928
8	4	63 972	16	2	17 411 568	24	2	446 677 760
9	4	156 871	17	2	28 262 956	25	2	605 336 704
10	4	408 960	18	2	44 567 096	26	2	840 422 400
11	4	808 639	19	2	67 792 800	27	2	1 129 623 808
12	4	1 741 120	20	2	103 479 680	28	2	1 539 258 368
13	4	3 081 512	21	2	147 573 376	29	2	2 009 468 416
						30	2	2 667 657 728

Table 3: Gradient descent results on Reversing Nearness

The gradient descent results consistently maintained a loss within 13% above SoTA, despite having significantly reduced computation times: $N = 30$ with $n_{optim} = 2$ took under 15 minutes using a graphical processing unit (GPU). The results here could be improved trivially, by increasing n_{optim} and requiring only more time.

5.3 Conclusion

Gradient descent has proved to be a generalizable algorithm, applicable to discrete problems such as Reversing Nearness, by the use of superposition. Furthermore, the computational complexity of gradient descent have proved to be viable for at least up to $N = 30$, despite requiring N^4 elements in S, a feat of its own.

References

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

[2] 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.

[3] Eric W. Weisstein. *Doubly Stochastic Matrix. From MathWorld–A Wolfram Web Resource*. URL: http://mathworld.wolfram.com/Tree.html (visited on 03/29/2020).

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

[5] 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.

[6] 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).

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

[8] Manfred Weis (https://mathoverflow.net/users/31310/manfred-weis). *Properties of Zero Line-Sum Matrices*. MathOverflow. eprint: https://mathoverflow.net/q/293024. URL: https://mathoverflow.net/q/293024.

A Lower Bound Constants

Table of lower bound constants copied from [1].

N	Lower Bound	N	Lower Bound	N	Lower Bound
1	0	11	1 883 244	21	325 763 172
2	10	12	3 813 912	22	474 261 920
3	72	13	7 103 408	23	673 706 892
4	816	14	12 958 148	24	949 783 680
5	3800	15	22 225 500	25	1 311 600 000
6	16 902	16	37 474 816	26	1 799 572 164
7	52 528	17	60 291 180	27	2 425 939 956
8	155 840	18	95 730 984	28	3 252 444 776
9	381 672	19	146 469 252	29	4 294 801 980
10	902 550	20	221 736 200	30	5 643 997 650