Reversing Nearness via Gradient Descent

Jason Ken A

March 29, 2020

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 gradient with respect to (w.r.t.) the function, that is:

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

Here, θ is the parameter to optimize,to minimize the value of function L, and α is an arbitrary scaling factor usually called *learning rate*. 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 the generalization capability of gradient descent, whether a discrete loss function can be generalized to a continuous loss, and therefore whether optimum discrete parameters can be obtained. "Reversing Nearness", a programming contest held by Al Zimmermann, proved to be a suitable testbed for this purpose, because traditionally, it had been approached with non-gradient optimization methods, such as hill climbing and simulated annealing, all of which are beyond the scope of this essay. To the best of my knowledge, a gradient based approach has never been attempted on the competition, for good reason: it is a discrete optimization problem. The objective is deceptively simple, 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 "token", "grid", and distance will be explained within the essay.

"It will be fun", I said, after all, are they not the reason for my calculus classes? Hence, my research question: *Is gradient descent a viable approach for* Reversing Nearness*?*

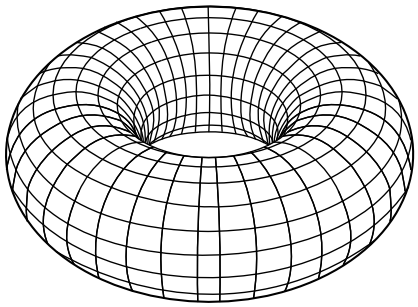
1. **Short answer:** Yes
2. **Long answer:**

[1] usually in the context of minimization, hence 'descent'

2 Problem Statement

2.1 Toroidal Grid

As stated on the AZsPCs website,[1] the initial toroidal grid O is defined as an $N \times N$ grid of unique tokens which "wrap around" (addressed in 2.2.1). For the sake of simplicity (and for reasons explained in 2.2.1), 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,[2] eg. AC corresponds to the token in row 1, column 3 of O, DF corresponds to the row 4, column 6 of O, etc. O for $N = 4$, is shown in Figure 1b. 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[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 (and hence this essay) 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[3] (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 these by the squared distance between the pair of tokens within the original grid,
3. Sum of all of these products
4. Subtract a lower-bound corresponding to the value of N[4]

2.2.1 Distance Metric

To evaluate the loss function, a distance metric between two tokens must be established, which in turn necessitates defining the position of a token within a coordinate system.

The coordinate of a token is defined as the indices of the token within X.

Note that within O, the coordinates, indices, and token representations are all equal.

Let two two-dimensional coordinates $s_1 = (x_1, y_1)$ and $s_2 = (x_2, y_2)$, then 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 Figure 2. The corresponding possible values for Δx are as follows:

$$\Delta_1(x) = x_2 - x_1$$

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

[3] 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

[4] Omitted here for brevity. Can be found at [1]

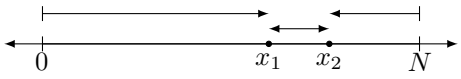


Figure 2: A one-dimensional diagram of toroidal distance, with the two arrows representing two possible distances

$$\Delta_2(x) = (x_1 - 0) + (N - x_2) = x_1 + N - x_2$$

To obtain a general equation which is applicable for $x_1 > x_2$:

$$\Delta_1(x) = |x_2 - x_1|$$
$$\Delta_2(x) = \min(x_1, x_2) + N - \max(x_1, x_2)$$

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:

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

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

One-dimensional toroidal distance is then defined as

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

In two dimensions, the toroidal distance and squared toroidal distance are

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

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

2.2.2 Token Comparisons

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

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

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

There are multiple ways of representing distances between all possible pairs of tokens, one of which is a 4-dimensional matrix (tensor) where every token within X (first 2 dimensions), is compared to every token (next 2 dimensions). An example of the structure of the comparison for an $N = 2$ grid is shown in Figure 3a. The white boxes denote duplicate comparisons (eg. $\begin{smallmatrix} BA \\ AA \end{smallmatrix}$ is identical to $\begin{smallmatrix} AA \\ BA \end{smallmatrix}$).

The dimensionality of the grid can be reduced[5] of the tensor can be reduced into a matrix: from $N \times N \times N \times N$ grid to $N^2 \times N^2$, elements within the 4-dimensional tensor are of form $C(X)_{i,j,k,l}$, the 2-dimensional elements of the form $C(X)_{m,n}$. To do this conversion,

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

-1 is to account for one-based indexing.

Both $C(X)_{i,j,k,l}$ and $C(X)_{m,n}$ represent the distance between tokens $\boldsymbol{IJ}$ and $\boldsymbol{KL}$ within X. The two-dimensional representation of Figure 3a is shown in Figure 3b.

From now on, let $C(X)$ refer to the two dimensional distance matrix.

2.2.3 Loss Function in Matrix Form

Therefore the loss function for toroidal grid X is equal to

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

c is the aforementioned lower-bound constant. Note that the diagonals of $C(O)$ and $C(X)$ are 0-valued, because they measure the distance between a token against itself. Therefore, to take into account only the unique distance measurements (ie. shaded cells) the factor of $\frac{1}{2}$ is added.

3 Superposition

Since the entries into the toroidal grid are discrete (ie. 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. Fragment here refers to a fraction of a token, and weight refers to the fraction of the fraction itself. Inspiration was taken from the field of physics, with electrons quantum superposition, with the position of the electron being represented with a probability density function.[6] This analogy will be taken further within 4.3.

		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: Grids S representing superpositions of an $N = 2$ toroidal grid, where every element represents the probability of the token KL being in the of position IJ in O

The superposition grid S consists of the weights of all the fragments of all the tokens. Again, this can be visualized as a 4-dimensional matrix, with the first 2 dimensions representing token positions (within O) and the next 2 dimensions representing the tokens. Using the indices conversions from 2.2.2 a

[5] ie. to reduce the number of Σs in 2.2.3

[6] function of probability over position

2-dimensional representation of this matrix is possible. Both are shown in Figure 4. Note that, in contrast to Figure 3, the elements of the grid *do not represent distances*, but rather, the element $S_{i,j,k,l}$ or $S_{m,n}$ represents the weight of the fragment of token $\boldsymbol{KL}$ within position $\boldsymbol{IJ}$. Let S denote the 2-dimensional representation. Note, within S rows represent the positions within O, and columns represent the token values. An example of superposition for a discrete toroidal grid is shown in Figure 5.

Further constraints to enforce the concept of weights will be discussed in 4.2.

By doing so, the limitations associated with a discrete grid are sidestepped. Now, all token values are associated with all of positions, with continuous[7] weights. Therefore, we can differentiate the loss function w.r.t. the weights, *instead of optimizing X, we optimize its continuous representation, S.* Note that with a discrete superposition grid (entries containing only 0s and 1s), the loss function is equivalent to that in 2.2.3.

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 have weight 1, non-shaded cells have weight 0

3.0.1 Generalization of the Loss Function

Defining the loss function for this formulation requires us to measure distance between *every* token value within *every* position (ie. all fragments), to *every other*[8] token value within *every* position, scaled by the weights the fragments. That is, for token value b in position a, compared to the token value d in position c, the distance[9] should be scaled by $S_{a,b}S_{c,d}$, because the weights within S reflect the extent a fragment should affect the loss.

The distance between these two fragments 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 (2.1).

Alike with 2.2.2, 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

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

Similar to 2.2.3, we can simply divide the loss by 2, because diagonal value evaluations: where $b = d$, are equal to 0, due to $C(O)_{b,d}$, 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

[7] as in non-discrete

[8] Ensuring unique *token value* comparisons

[9] squared toroidal distance in X multiplied by squared toroidal distance in O

$S_{a,b}$ represents the weight of fragment $\boldsymbol{AB}$

$S_{c,d}$ represents the weight of fragment $\boldsymbol{CD}$

$C(O)_{a,c}$ 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

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:

$$\theta' = \theta - \alpha \frac{\delta}{\delta\theta} L(\theta) \tag{3}$$

α is an arbitrary scaling factor usually called *learning rate*.

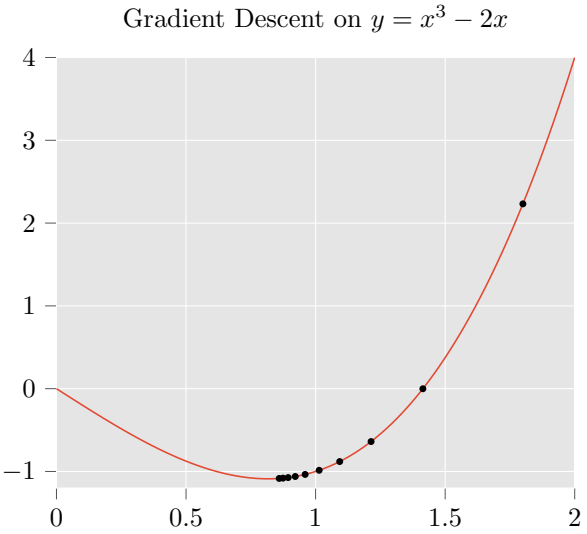


Figure 6: Demonstration of gradient descent convergence: 10 iterations with $\alpha = 5 \times 10^{-2}$ and $\theta_0 = 1.8$.

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:

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

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

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

When evaluating $J_{i,j}$, only $S_{i,j}$ is treated as a variable, others are treated as constants. $S_{i,j}$ is only included within the loss when $(a,b) = (i,j)$ or $(c,d) = (i,j)$ or both. The derivatives for the respective cases are as follows:

$$\begin{aligned}
\boldsymbol{A} = \frac{\delta L(S)}{\delta S_{i,j}} &= \frac{\delta}{\delta S_{i,j}} \left(\frac{1}{2} \sum_{c}^{N^2} \sum_{d}^{N^2} S_{i,j} S_{c,d} C(O)_{i,c} C(O)_{j,d} \right) && \text{first case} \\
&= \frac{1}{2} \sum_{c}^{N^2} \sum_{d}^{N^2} S_{c,d} C(O)_{i,c} C(O)_{j,d} \\
\boldsymbol{B} = \frac{\delta L(S)}{\delta S_{i,j}} &= \frac{\delta}{\delta S_{i,j}} \left(\frac{1}{2} \sum_{a}^{N^2} \sum_{b}^{N^2} S_{a,b} S_{i,j} C(O)_{a,i} C(O)_{b,j} \right) && \text{second case} \\
&= \frac{1}{2} \sum_{a}^{N^2} \sum_{b}^{N^2} S_{a,b} C(O)_{a,i} C(O)_{b,j} \\
\boldsymbol{C} = \frac{\delta L(S)}{\delta S_{i,j}} &= \frac{\delta}{\delta S_{i,j}} \left(\frac{1}{2} S_{a,b} S_{a,b} C(O)_{a,b} C(O)_{a,b} \right) && \text{third case} \\
&= \frac{1}{2} \sum_{a}^{N^2} \sum_{b}^{N^2} S_{a,b} C(O)_{a,a} C(O)_{b,b} \\
&= 0
\end{aligned}$$

$\boldsymbol{C}$ is 0 due because it includes the distance of a token against itself. The partial derivative of $L(S)$ w.r.t. $S_{i,j}$ is therefore equal to $\boldsymbol{A} + \boldsymbol{B} - \boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B}$, because they encapsulate all of the cases where $S_{i,j}$ is a part of a term. A naive approach would be to update the entries as follows:

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

4.2 Generalization of Discrete Constraints

4.2.1 Doubly Stochastic Matrices

First of all, regardless of whether discrete grids or superposition grids are involved, *negative weights do not make sense*, as weights represent what portion of the token is located within a certain position.

Within X or O, every position has a *single* unique token, and every token has a *single* unique position. To put it another way, no token does not have a position, and no position does not have a token. Note the superposition of a discrete grid in 5b, the sums of all the rows and columns are equal to 1. In other words, the sum of all the weights within every position is 1, and the sum of all the weights of every token is 1. That is, a matrix A with non-negative values and

$$\sum_{i} A_{i,j} = \sum_{j} A_{i,j} = 1$$

is called a doubly stochastic matrix.[3] Therefore, the superposition S must always be doubly stochastic.

4.2.2 Sinkhorn-Knopp Algorithm

This section explains the algorithm used within 4.2.3. A well-known algorithm to convert any non-negative matrix into a doubly stochastic matrix is called the Sinkhorn-Knopp algorithm (also named RAS).[4] There is a proof[5] and several papers[6, 7] analyzing its convergence. Nonetheless, the algorithm itself is simple: repetitively normalizing the rows and columns of a matrix.

Let K be an $n \times n$ non-negative matrix.[10] A single iteration of RAS is defined as follows:

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

[10] every row and column should contain at least 1 positive value

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

The scaling matrices are diagonal (non-diagonal elements are 0s). Here, "normalizing" is defined as forming a sum of 1. Figure 7 demonstrates the effectiveness of RAS. Graphed on the y-axis is the squared error, defined as:

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

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

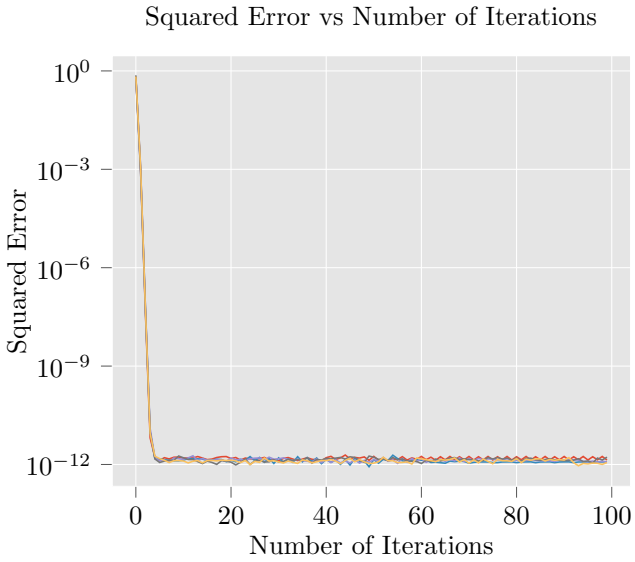


Figure 7: Demonstration of RAS convergence: 5 samples with $N = 100$, randomly generated from a uniform distribution $[0, \frac{2}{N})$ (mean $\frac{1}{N}$). Error is equal to the sum of squared errors of the sums of the rows and columns from 1. Note the logarithmic scale.

Another example of the RAS algorithm:

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

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

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

4.2.3 Zero Line-Sum Modified Jacobian

To maintain the doubly stochastic nature of the superposition grid within gradient descent update (in 4.1.1), the Jacobian matrix must be modified into a zero line-sum (ZLS) matrix.[8] A ZLS matrix has all rows and columns summing to 1, that is, a matrix A is ZLS iff.

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

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.

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

Therefore, $A - B$ is doubly stochastic. □

Hence, a Jacobian can be converted into a doubly stochastic matrix through the RAS algorithm,[11] and by subtracting it by another doubly stochastic matrix, a ZLS-Jacobian can be obtained. This second doubly stochastic matrix can be obtained by scaling a ones-matrix (matrix filled with ones). Let this second doubly stochastic matrix be D with dimension $N \times N$. D is defined as follows:

$$D_{i,j} = \frac{1}{N}$$
$$\sum_i D_{i,j} = \sum_j D_{i,j} = N \times \frac{1}{N} = 1 \qquad D \text{ is doubly stochastic}$$

Because D is uniform (all elements have identical values), the magnitudes of the elements doubly stochastic Jacobian elements w.r.t. other elements are preserved, that is $\text{RAS}(J)_{a,b} > \text{RAS}(J)_{c,d} \iff \text{RAS}(J)_{a,b} - D_{a,b} > \text{RAS}(J)_{c,d} - D_{c,d}$

Let $J' = \text{RAS}(J) - D$.

4.2.4 Non-negative Matrices

Assuming S is doubly stochastic, and with ZLS-J', the result of the gradient descent update $S' = S - \alpha J'$, has rows and columns summing to 1. *But S' is not necessarily doubly stochastic, because S' might not be non-negative.* 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, effectively ensuring that S' is doubly stochastic:

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

$\min S'$ refers to the smallest value within S'. $S'_{i,j} + \min S$ effectively creates a non-negative matrix.[12]

Proof. Let S' be a matrix with rows and columns summing to 1.

$$\sum_i^{N^2} S''_{i,j} = \sum_i^{N^2} (S'_{i,j} + \min S) \times \frac{1}{1 + N^2 \min S} = \frac{1}{1 + N^2 \min S} \sum_i^{N^2} (S'_{i,j} + \min S)$$
$$= \frac{1}{1 + N^2 \min S} (\sum_i^{N^2} (S'_{i,j}) + N^2 \min S) = \frac{1}{1 + N^2 \min S} (1 + N^2 \min S)$$
$$= 1$$

Replace $\sum_i^{N^2}$ with $\sum_j^{N^2}$ and repeat. Therefore, S'' is doubly stochastic. □

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 position collapses, being reduced to a single point. And because it relates with probabilities, the results are random, they are *probabilistic*. Within this essay, the probability density function has been represented

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

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

by the weights of fragments within various positions. But instead of sampling from a distribution, by revealing a token, it will be placed in the position of the fragment with the highest weight, ie. the most likely position.

Taking this analogy even further, the tokens are *entangled*. In physics, if two electrons are entangled, when the *spin*[13] of an electron is revealed, the other is also instantaneously revealed. If one is found to be spin up, the other electron has spin down, vice versa. The system within this essay involves N "electrons" (tokens), and when the position of this is revealed, the fragments from other tokens within the same position are removed, because no two tokens should inhabit the same position.

Also, the fragments of revealed tokens (columns of S) and fragments within revealed positions (rows of S) should be optimized. Since revealed tokens and positions are *fixed*.

The procedure for revealing a token is defined as follows Let $R = \varnothing$ be a set of indices (rows and columns) of revealed fragments.

$$R \leftarrow R \cup \arg\max_{(m,n)\notin R} S_{m,n}$$

$$S_{i,j} = \Big\{ S_{ij}$$

$\arg\max_{(m,n)\notin R} S_{m,n}$ returns the indices of the fragment with the highest weight, $a \leftarrow b$ indicates a reassignment: the new value of a is b.

To only perform optimization on non-revealed token fragments and positions, the revealed rows and columns can be removed from J, the appropriate procedures can be performed to obtain J', and the J' can 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.

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

4.4 Optimization Procedure

4.4.1 Superposition Initialization

Recall that for the optimization of S, S must be doubly stochastic. When no tokens have been revealed, $S = D$ (from 4.2.3). To work with an arbitrary amount of revealed tokens, S with dimensions $N^2 \times N^2$ is defined as follows:

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

Proof. upcoming proof of double stochasticity □

The following is the optimization procedure

1. Initialize S
   (a) For every N:
      i. For n_{optim} steps:
         A. Calculate J, ensure ZLS J'
         B. Perform gradient descent update
         C. Ensure doubly stochastic S''
      ii. Reveal token

5 Evaluation

5.1 Graphs of Loss over Time

From the following images, we can see that Gradient Descent is in fact viable for Toroidal Grid Optimization *in its continuous representation*, and that its success does not translate well over to the discretized version

[13] a quantum property beyond the scope of this essay

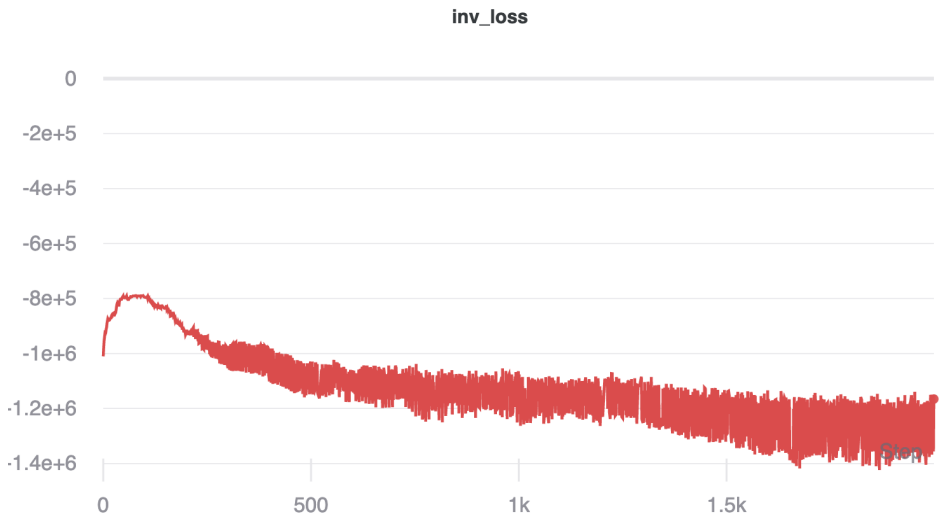


Figure 8: Negative of Maximum Deviation vs Iterations

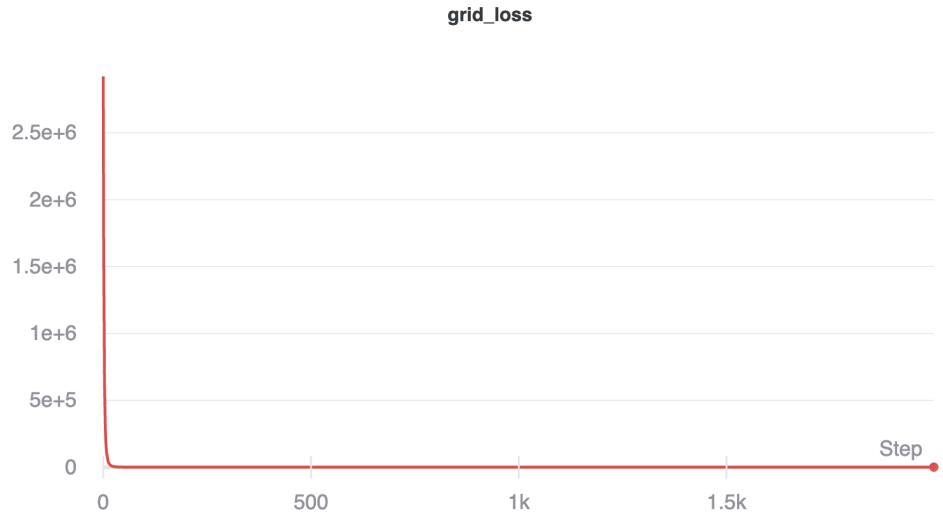


Figure 9: Superposition Loss vs Iterations

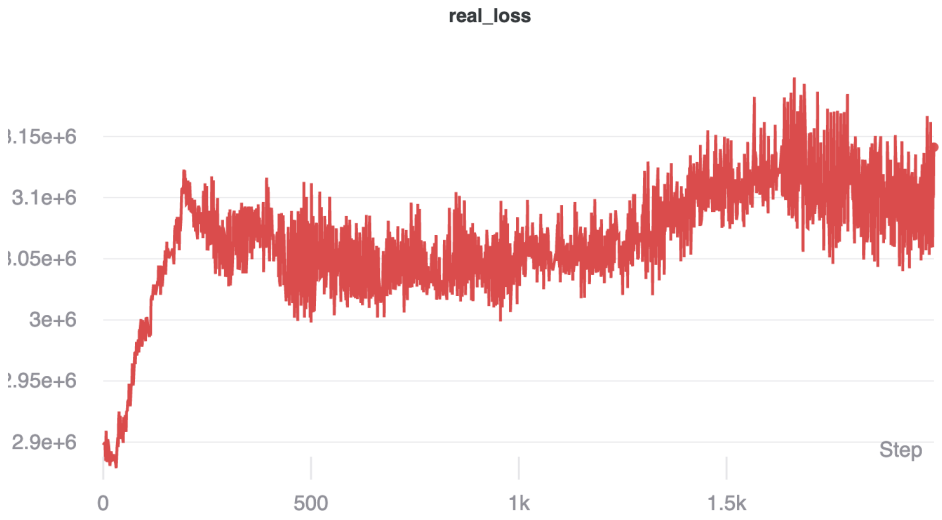


Figure 10: Discretized Grid Loss vs Iterations

of the grid. This is reflected by the fact that the discretized loss after the optimization is worse off than it was in the beginning, during random initialization. This reflects a clear discrepancy between continuous and discrete optimization. The non-smoothness of 2 of the graphs can be traced to the non-discrete natures of both of those losses.

5.2 Limitations

A multi-objective loss function is difficult to optimize by nature. How much each component affects the overall loss function is to be determined arbitrarily, and these hyper-parameters[14], have to be tested arbitrarily. In this essay, only a single set of hyperparameters such as learning rate, proportion of negative maximum deviation and toroidal grid loss is used. This essay by no means provides a comprehensive evaluation of the possible optimization strategies, and therefore, may lack to optimize the function because of the wrong tuning of these hyperparameters and not a mistaken problem formulation.

The second possible reason is that I have not properly enforced the idea of discreteness within the optimization. There are a plentiful amount of methods to attempt this, including the Sinkhorn-Knopp algorithm for doubly stochastic matrices, and normal standard deviation. But through my trials, all of them work just about the same, which does not say a lot.

5.3 Summary

Gradient descent is truly an amazing optimization algorithm. That is, if your loss function reflects the objective. But in the case of toroidal grid optimization, that does not seem to be the case. The stark difference between the continuous and the discrete nature of the problem of toroidal grid optimization makes gradient descent not a viable choice for this problem, according to the experiments laid out within this essay. But this does not hinder the effectiveness of Gradient Descent, but rather it highlights how it should be used properly. With a continuous objective.

[14] Parameters that cannot be learnt

6 Acknowledgements

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

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.