# Exploring the Solution Space of Aligning Two Sets of 3D Rotation Angles

Javier Alejandro Maroto Morales
Jelena Banjac
*Swiss Federal Institute of Technology in Lausanne (EPFL)*

***Abstract*—The goal of our project is to explore the performance of different algorithms on our non-convex problem: aligning two sets of 3-dimensional rotation angles. The angle alignment implies that there exists a 4-dimensional rotation that would rotate one set of 3-dimensional angles and, if possible, align it with another one. We show that we are dealing with non-convex machine learning objective and explain the methods we use to understand the high-dimensional solution space. The current results obtained are then discussed depending on different combination of settings used and experimental conditions. We find that AdaGrad and FTLR optimizers are the most robust to changing the experimental setting and starting point. The source code of our project is available at: https://github.com/javier-maroto/optML_miniproject.**

## I. Introduction

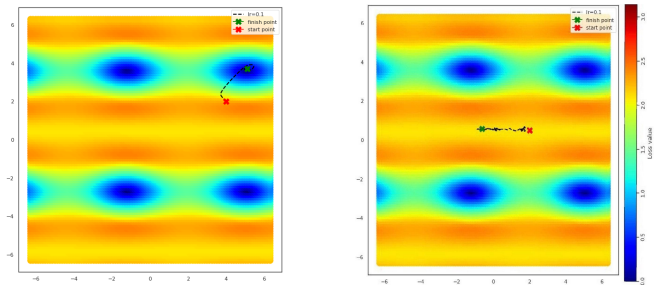
A vast majority of machine learning algorithms train the models by solving an optimization problem. In order to capture the learning and prediction problems accurately, it sometimes happens that the objective itself is a non-convex function. This is especially true for algorithms that operate in a high-dimensional spaces. Some of the things that make non-convex optimization at least NP-hard are potentially many local minima, saddle points, very flat regions, and widely varying curvature. Aligning two sets of 3D rotation angles belongs to the class of non-convex optimization problems and we are not able to solve this problem analytically.

One of the reasons we need to solve this optimization problem in the context of machine learning optimization is due to the fact that there are many feasible solutions (i.e. there are many sequences of rotations that can align two sets of rotations), but we are interested in finding the best solution. Another reason is that we are able to create a loss function that we want to minimize and that way find the optimal parameters (i.e. the optimal rotation). In addition, the training was done in a stochastic setting, where the loss function varies over the batches.

The explanation of the optimization objective and project goal is explained in section II. The results obtained are discussed depending on different combination of approaches used and experimental conditions in section III. Lastly, we summarize the contributions in light of the new results and we discuss the strengths and weaknesses of our approach based on the results in section IV .

## II. Theory

The idea for the optimization comes from the project whose general goal is to find the Cryo Electron Microscope (Cryo-EM)[1] angles at which the projection images are taken of the 3D protein (i.e. how was the microscope rotated when it took an projection image of the 3D protein). The microscope can be rotated around Z1-Y2-Z3[2] axes respectively. The Z1-axis rotation angle is *rotation angle*, the Y2-axis angle is *tilt angle*, and the Z3-axis rotation angle is *in-plane rotation*. Therefore, for every projection image the three angles are predicted. The prediction method used is based on predicting angles based their mutual distance and not absolute location. Due to the prediction method, it can happen that the predicted set of 3 angles is actually rotated ground-truth set of angles. Therefore, pair-wise comparison with ground-truth set will give a big error, even though they are estimated correctly. Therefore, we decided to concentrate on optimization of aligning two sets of 3D rotation angles and explore its solution space.



(a) successful convergence to global minimum (blue): loss is 9e-04 and starting point is (4.0, 2.0, 3.0, 3.06, 2.56, 3.82)

(b) stuck in the local minima (yellow): loss is 1.86 and starting point is (2.0, 0.5, 3.0, 3.06, 2.56, 3.82)

Fig. 1: Loss variation w.r.t. the first two values of 6-dimensional vector (other four values are fixed during the optimization).

One way to check if the set of angles is predicted correctly is the angle alignment optimization, the optimization we developed and study in this project:

$$\hat{R} = \arg\min_{R} \mathcal{L}(R) = \arg\min_{R \in SO(4)} \frac{1}{n} \sum_{i=1} d_q(q_i, R \begin{bmatrix} m & 0 \\ 0 & I \end{bmatrix} \hat{q}_i) \tag{1}$$

where:

- $\mathcal{L}$ is a smooth loss function that is *objective function* of our problem,

- $d_q$ is the distance between two quaternions $d_q : S^3 \times S^3 \rightarrow \mathbb{R}^+$, $d_q(q_1, q_2) = 2\arccos(|\langle q_1, q_2 \rangle|)$. Function $d_q$ is a necessary non-negative function that takes values in range $[0, \pi]$ [3],
- $q_i \in S^3$ is the quaternion[1] of a true rotation angles,
- $\hat{q}_i \in S^3$ is the rotation of the estimated rotation angles,
- $R \in SO(4)$ is $4x4$ orthogonal matrix with determinant 1 that represents a global 4-dimensional rotation [6] and it is created from 6-dimensional vector[2] that is a *variable* of our problem,
- $m \in \{-1, 1\}$ represents a global reflection [8] ($m = 1$ no reflection, $m = -1$ reflection).
- $I$ is a 4x4 identity matrix.

## III. Experiments

In this section, we are exploring the solution space and answering the questions regarding the behaviour observed. We also compare the performance of different optimizers and learning rates.

**Datasets:** We are given two datasets:

- **Inputs:** Set of 5000 *predicted angles* of size 3 converted to set of quaternions ($\hat{q}_i$ in equation 1): shape of predicted set is 5000 x 4.
- **Outputs:** Set of 5000 *ground-truth angles* of size 3 converted to set of quaternions ($q_i$ in equation 1): shape of ground-truth set is 5000 x 4, and

**Optimization:** We use optimization with the goal to minimize the pair-wise quaternion distance $d_q$ between ground-truth quaternions and rotated predicted quaternion and to find a 4x4 rotation matrix $R$ (i.e. 4-dimensional rotation) that will align the initial set of quaternions $\hat{q}_i$ and the set of ground-truth quaternions $q_i$. This 4x4 rotation matrix is created from a 6-dimensional vector, therefore, we are estimating six values.

### A. Experiment 1: Convergence to the optima

The setting for this experiment was:

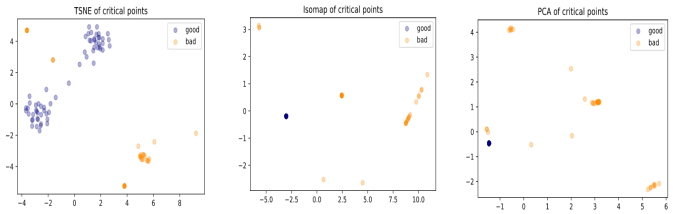
- Number of gradient steps: 300,
- Batch size: 256,
- Learning rate: 0.1,
- Optimizer: Adam,
- Starting point: random initialization.

We ran the experiment 100 times and we observed that it manages to converge to global minima 63% of the cases. To understand the reason, we explored the solution subspace that can be seen in figure 1.

From this experiment, we observe that the convergence depends on the starting point. Therefore, the optimization problem we are dealing with is a non-convex problem with a non-convex objective.

[1]Quaternions are extension of complex numbers. Generally they are represented in the form: $q = a + bi + cj + dk$. In this project we use them to represent a rotation. More information about the quaternions can be found in [4], [5]

[2]Taking that a 4-dimensional space is represented with (X, Y, Z, W) axes, then each value of this 6-dimensional vector represents an angle of rotation about ZW, YW, YZ, XW, XZ, XY-axis respectively[7]




(a) TSNE (b) Isomap (c) PCA

Fig. 2: The 2-dimensional solution vectors embeddings from TSNE, Isomap, and PCA algorithms that are colored based on whether they converged to the optimum (purple) or not (orange)

### B. Experiment 2: Dimensionality reduction of the objective variable

In section III-A we performed the experiment by varying only the first two values of the 6-dimensional vector, but we are interested to see what is happening on the full solution space. To do that, we tested several dimensionality reduction techniques to decrease the 6-dimensional vector to a 2-dimensional vector. The dimensionality reduction algorithms we tested include: PCA, MDS, TSNE, and Isomap[9].

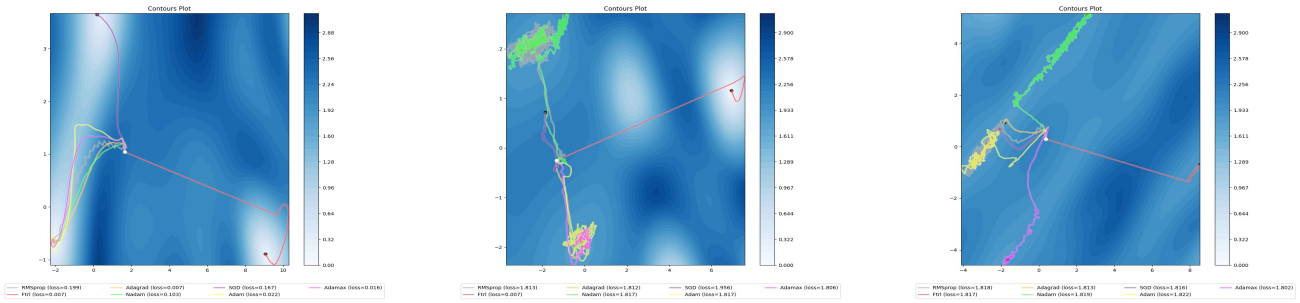
Observing the embeddings, we noticed that Isomap is performing better than other algorithms (comparison between TSNE, Isomap, and PCA can be seen in Figure 2), i.e. Isomap is embedding the successful convergences in one place as well as the PCA. However, the grouping of unsuccessful convergences is better in Isomap. One of the reasons is that Isomap is a non-linear dimensionality algorithm that extends the MDS by incorporating the geodesic distances. According to [3] which compares a number of metrics, the geodesic distance of the normed quaternions is a similar option for a distance and the Euclidean distance is not appropriate here. Whereas, PCA is not as good for these critical points, but it is good for finding the inversion (from 2D to 6D vector) that will be later used in the plots to visualize the solution plane.

Thanks to this experiment, we can now visualize and potentially better understand the effects of changing the learning rates and optimizers.

### C. Experiment 3: Choosing the optimizer

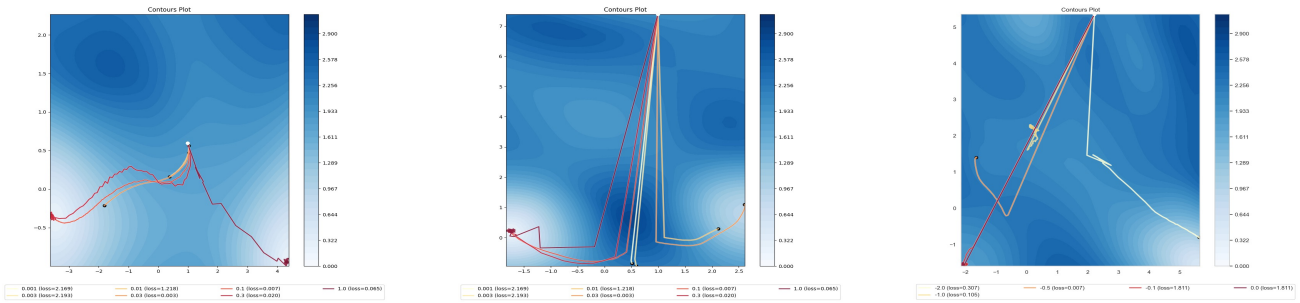
The optimizers we decided to test for this problem are: RMSprop, Adagrad, SGD, Adamax, Ftrl, Nadam, and Adam. We evaluated their performance with different hyperparameter setting and the ones showing the best perfomance can be found in Appendix V.

**RMSprop**. In Figure 3(a), it manages to converge to minimum with loss 0.199 where majority of optimizers managed to converge, even though with a smaller loss value than RMSprop. However, its gradient descent trajectory line has zig-zag shape probably due to a higher learning rate in that region. In Figure 3(b) it did not converge to global minimum but got stuck in a local minimum with loss 1.813. In Figure




(a) Seed 10 (b) Seed 55 (c) Seed 67

Fig. 3: The performance of different optimizers with different starting points (controlled by seed value)




(a) AdaGrad (b) FTLR (c) FTLR (varying LR power)

Fig. 4: The performance of different learning rates for two optimizers

3(c) it also did not converge to global minimum, neither did all the other optimizers.

**Adagrad**. In Figure 3(a), it manages to converge to minimum with a very good loss of 0.007 where majority of optimizers managed to converge. In Figure 3(b) it did not converge to global minimum but got stuck in a local minimum with loss 1.812. In Figure 3(c) it also did not converge to global minimum.

**SGD**. In Figure 3(a), it manages to converge to minimum with a loss of 0.167 in a location that wasn't discovered by other optimizers. In Figure 3(b) it did not converge to global minimum but got stuck in a local minimum with loss 1.956. In Figure 3(c) it also did not converge to global minimum.

**Adamax**. In Figure 3(a), it manages to converge to minimum with a loss of 0.016 in a location found by majority of optimizers. In Figure 3(b) it did not converge to global minimum but got stuck in a local minimum with loss 1.806. In Figure 3(c) it also did not converge to global minimum.

**FTRL**. In Figure 3(a), it manages to converge to minimum with a loss of 0.007 in a location different from the one found by majority of optimizers. In Figure 3(b) it is the only optimizer that manages to find the global minimum. In Figure 3(c) it also did not converge to global minimum.

**Nadam**. In Figure 3(a), it manages to converge to minimum with a loss of 0.103. In Figure 3(b) and (c) it did not converge to global minimum but got stuck in a local minimum.

**Adam**. In Figure 3(a), it manages to converge to minimum with a loss of 0.022. However, it seems it had a high momentum which led the gradient descent trajectory to closest plateau where later it managed to converge to global minimum. This was caused by smaller exponential decay weight of first and second momentum. In Figure 3(b) and (c) it did not converge to global minimum but got stuck in a local minimum.

Learning rate fine tuning can be found in Appendix VI.

## IV. Conclusion

To conclude, in this project we managed to align two sets of 3D rotations using a 4D rotation. We tested and evaluated the performance of different optimizers and their hyper-parameters and singled out the ones that perform the best. Adam is usually the first and default optimizer when starting with the optimization problem due to its robustness to different problems that can be noisy. Since here we are dealing with the problem that is well defined and has the possibility to converge to zero loss values, we discover that AdaGrad and FTLR are the ones that perform the best, especially AdaGrad. Unlike noted in [10] that Adagrad struggles in a non-convex setting, we discovered that for our optimization problem it is very stable and fast in comparison to other optimizers and has a nice smooth line towards the global minimum.

## References


[1] Cryo-electron microscopy - an overview | ScienceDirect topics. [Online]. Available: https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/cryo-electron-microscopy

[2] "Euler angles," page Version ID: 957791539. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Euler_angles&oldid=957791539

[3] D. Q. Huynh, "Metrics for 3d rotations: Comparison and analysis," vol. 35, no. 2, pp. 155–164. [Online]. Available: http://link.springer.com/10.1007/s10851-009-0161-2

[4] W. R. Hamilton, "ON QUATERNIONS, OR ON a NEW SYSTEM OF IMAGINARIES IN ALGEBRA," p. 92.

[5] "Quaternion," page Version ID: 959215339. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Quaternion&oldid=959215339

[6] "Rotations in 4-dimensional euclidean space," page Version ID: 950831829. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Rotations_in_4-dimensional_Euclidean_space&oldid=950831829

[7] quaternions - rotation in 4d? Library Catalog: math.stackexchange.com. [Online]. Available: https://math.stackexchange.com/questions/1402362/rotation-in-4d

[8] "Reflection (mathematics)," page Version ID: 942253197. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Reflection_(mathematics)&oldid=942253197

[9] V. de Silva and J. B. Tenenbaum, "Global versus local methods in nonlinear dimensionality reduction," p. 8.

[10] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," p. 39.

[11] tf.keras.optimizers.RMSprop | TensorFlow core v2.2.0. Library Catalog: www.tensorflow.org. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/RMSprop

[12] tf.keras.optimizers.adagrad | TensorFlow core v2.2.0. Library Catalog: www.tensorflow.org. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adagrad

[13] tf.keras.optimizers.SGD | TensorFlow core v2.2.0. Library Catalog: www.tensorflow.org. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/SGD

[14] tf.keras.optimizers.adamax | TensorFlow core v2.2.0. Library Catalog: www.tensorflow.org. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adamax

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization." [Online]. Available: http://arxiv.org/abs/1412.6980

[16] tf.keras.optimizers.ftrl | TensorFlow core v2.2.0. Library Catalog: www.tensorflow.org. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Ftrl

[17] H. B. McMahan, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos, J. Kubica, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, and E. Davydov, "Ad click prediction: a view from the trenches," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '13*. ACM Press, p. 1222. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2487575.2488200

[18] tf.keras.optimizers.nadam | TensorFlow core v2.2.0. Library Catalog: www.tensorflow.org. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Nadam

[19] T. Dozat, "Incorporating nesterov momentum into adam," p. 6.

[20] tf.keras.optimizers.adam | TensorFlow core v2.2.0. Library Catalog: www.tensorflow.org. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam


## V. Appendix: Optimizer hyperparameters

**RMSprop**. We have used the following parameters: $\gamma = 0.1$, $\rho = 0.9$, $\epsilon = 1e-07$, without momentum. More information in [11].

**Adagrad**. We have used the following parameters: $\gamma = 0.1$, $\epsilon = 1e-07$, and initial accumilator value for gradients is 0.1. More information in [10], [12].

**SGD**. We have used the following parameters: $\gamma = 0.1$, without momentum. More information in [13].

**Adamax**. We have used the following parameters: $\gamma = 0.1$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e-07$. More information in [14], [15].

**FTRL**. We have used the following parameters: $\gamma = 0.1$, learning rate decrease by power of 2 (value -0.5), initial accumulator value is 0.1, $L_1$ regularization strength is 0, and $L_2$ regularization strenght is 0. More information in [16], [17].

**Nadam**. We have used the following parameters: $\gamma = 0.1$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e-07$. More information in [18], [19].

**Adam**. We have used the following parameters: $\gamma = 0.01$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e-07$. More information in [20], [15].

## VI. Appendix: Experiment 4: choosing the learning rate

From the previous experiment in section III-C, we decided to pursue using Adagrad and FTRL optimizers with varying learning rates. Both of them are ran with the following values for the learning rate $\gamma = \{0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1.0\}$, the results can be seen in Figure 4(a) and (b). Changing the learning rate power of the FTLR algorithm we managed to change the size of the first step of the trajectory.