Metric Learning

Jakub Winiarski
University of Warsaw
jj.winiarski@student.uw.edu.pl

June 20, 2025

1 Introduction

I managed to do everything. During implementation I faced several issues, that took me quite a lot of time. At first it was difficult for me to get the whole concept. I implemented the datasets and BroNet without confidence, but later everything started to be more clear.

Also, I heard from my friends, that they had to convert the data to *float16*, because they ran out of memory. Following ths advice, I did the same thing. In my case it turned out to be a mistake. It was a reason for instability of gradient computation. It took me several hours to detect the issue. Fortunately after switching to *float64* everything worked well and I didn't run out of memory as well.

After solving this issue, I didn't face any major problems.

2 Hyperparameters

2.1 Training parameters

For all the trainings I used the same set of hyperparameters to make the results as comparable as possible. It turned out to work decent in each case, so I decided not to change. Below, there is listed a set of parameters that I used:

- *train_steps*: 1000000,
- *n_train_trajectories*: 4000,
- *lr*: 0.0001,
- *batch_size*: 64.

2.2 Model Architectures

For model I used **BRO Net** architecture. I used specified dimensions for each task.

Supervised Learning

- *input_dim*: 800,
- *hidden_dim*: 128,
- *output_dim*: 64 (length of the longest path).

Contrastive Learning

- *input_dim*: 400,
- *hidden_dim*: 128,
- *output_dim*: 64 (arbitrary latent space dimension inspired by notebook).

Stitching

- *input_dim*: 400,
- *hidden_dim*: 128,
- *output_dim*: 64 (arbitrary latent space dimension inspired by notebook).

3 Metrics Implementation

3.1 Correlation

Correlation is computed using *Spearman rank* between the predicted distance and the position in the trajectory. The implementation was ready in the notebook.

Our task was to create a plot representing correleation between distances and actual steps to goal. I decided to select 10 example trajectories and for each trajectory I calculated a predicted distance between first state in the trajectory and each of the following. Later I put the results on the plot. Bolow I present the results for each of training methods.

The predicted distance for *supervised* model is calculated as a classification for 1 of 64 possible distances. For *contrastive* model it is a p_2 norm between latent space vectors.

3.1.1 Supervised Learning

For *supervised learning* the correlation looks quite good. There is a high percentage of trajectories that are predicted well for each distance from the starting point. However, there are several cases where prediction is totally wrong.

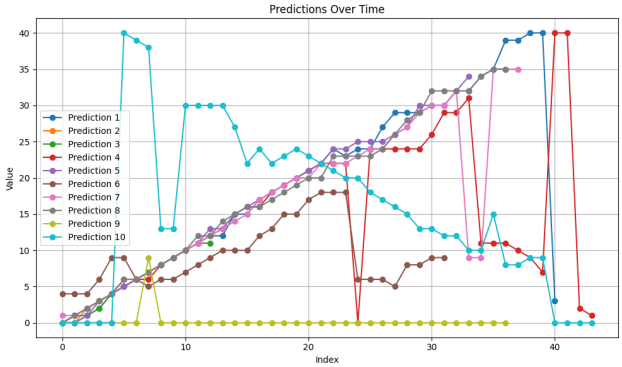


Figure 1: Correlation between predicted and real distances for *supervised learning*.

3.1.2 Contrastive Learning

In case of *contrastive learning*, the results are different. There are no cases where prediction is completely wrong. The predicted distance is proprtional to the real distance. However, it does not correspond to real results 1:1; there is some bias. The predicted distances are a little bit higher than the real ones, especially for shorter distances.

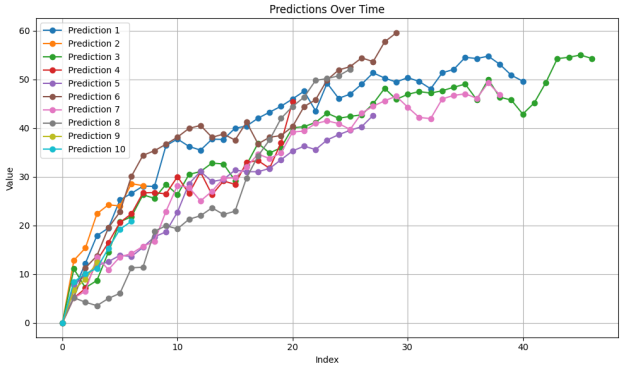


Figure 2: Correlation between predicted and real distances for *contrastive learning*.

3.1.3 Stitching

For *stitching* I used *contrastive learning*, so the results are similar. However, the bias is even higher. Predicted distances can be even two times higher than the real one.

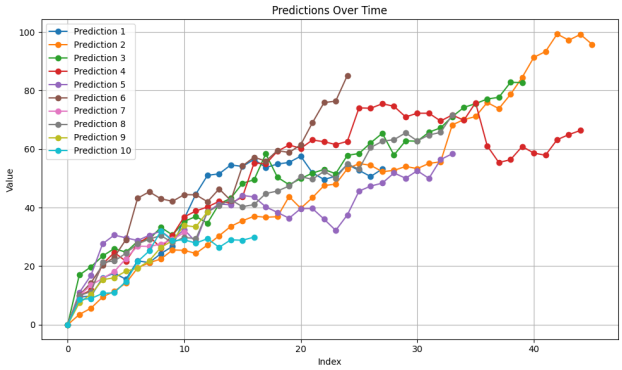


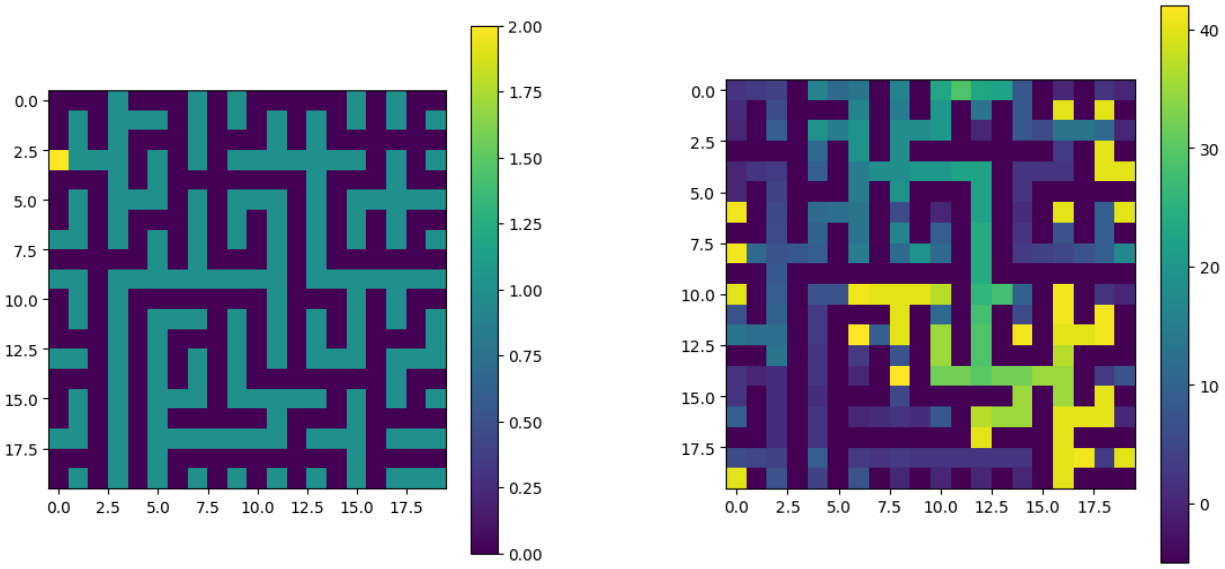
Figure 3: Correlation between predicted and real distances for *stitching*.

3.2 Heatmaps

For each method of training I generated example heatmap of distances. For each section, the first picture is the maze, where the path is in darker blue than the walls. The goal state is marked with yellow. In the heatmeap, the walls are in dark blue and the scale for distances is on the right.

3.2.1 Supervised Learning

In case of supervised learning we can see that predictions are quite accurate. However, a few points have predictions that are far away from ground truth.



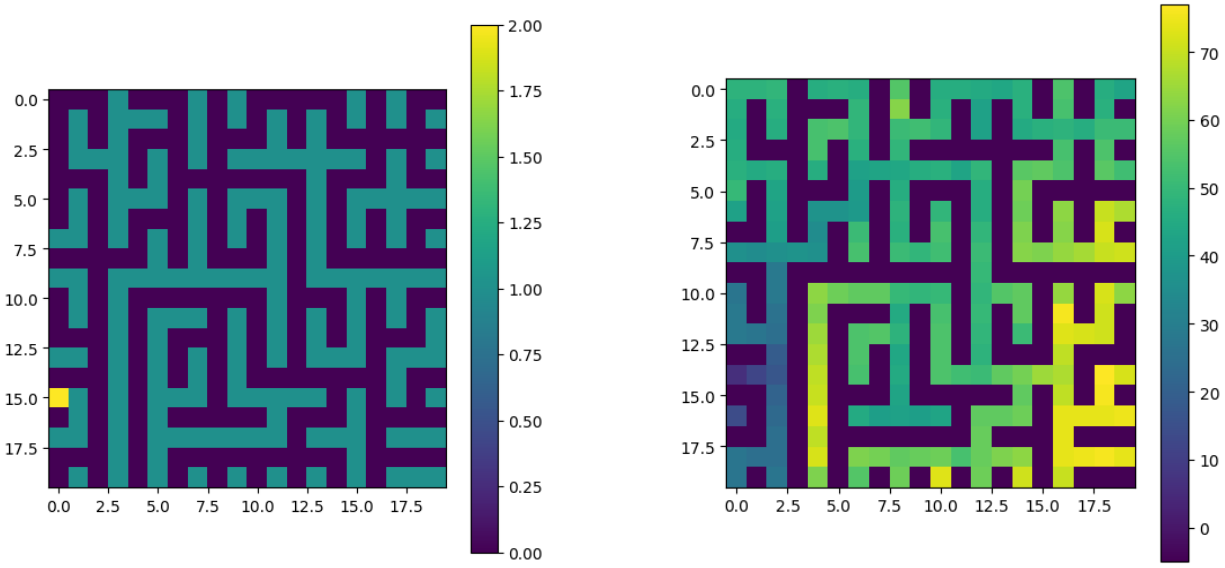
(a) Maze with the goal state

(b) *Supervised* model prediction heatmap

Figure 4: *Supervised* model heatmap

3.2.2 Contrastive learning

Contrastive learning returns a smoother heatmap that looks very reasonable. The only problem is the scale. As mentioned before, there is some bias for the results.



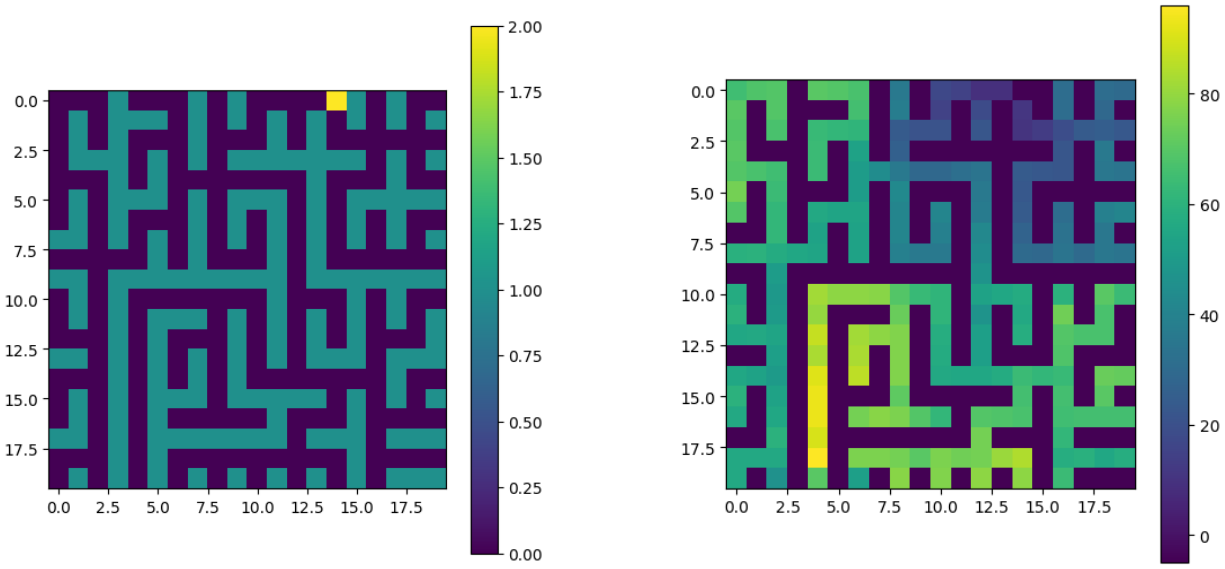
(a) Maze with the goal state

(b) *contrastive* model prediction heatmap

Figure 5: *contrastive* model heatmap

3.2.3 Stitching

Stitching results are similar to contrastive learning, but with higher bias.



(a) Maze with the goal state

(b) *stitching* model prediction heatmap

Figure 6: *stitching* model heatmap

3.3 Solved Rate

The pseudocode of a function calculating *Solved Rate* is presented in Algorithm 1. It uses a heap as a priority queue with nodes that might be visited. The model serves as a heuristic function that predicts the distance. You visit the nodes that have the lowest prediction first. You visit only unseen nodes - when you visit the node you mark it as seen.

Number of expanded nodes is the number of seen nodes. For each method, the mean expanded nodes per env is calculated.

When visiting a node, you add the unseen neighbors of the node to the queue. In version with search you add all the unseen neighbors. In version without search you only add the one with lowest predicted distance.

Solved rate is the rate of envs that were solved by the algorithm (i.e. the algorithm has visited the goal state node at some point). I tested each method on 1000 trajectories.

The results are presented below.

3.3.1 Supervised Learning

Version with search

- *solved rate*: 0.995,
- *average expanded nodes*: 43.824.

Version without search

- *solved rate*: 0.285,
- *average expanded nodes*: 11.63.

Algorithm 1 CheckSolvedRate

Require: Value estimator $\mathcal{V}$, number of instances n, flag `do_search`

Initialize `solved_count` $\leftarrow 0$, `expansions` $\leftarrow [\,]$
for $i \leftarrow 1$ to n **do**
  Initialize maze environment $\mathcal{M}_i$
  Priority queue $Q \leftarrow \{(0, \mathcal{M}_i.\text{current\_state})\}$
  `seen` $\leftarrow \emptyset$, `expanded` $\leftarrow 0$
  while Q not empty and not solved **do**
    Pop (p, s) from Q
    if s equals $\mathcal{M}_i$.solved_state **then**
      `solved_count++`
      break
    end if
    `expanded++`
    Generate neighbors $\{(p_j, s_j)\}$ of s using $\mathcal{V}$
    for all (p_j, s_j) not in `seen` **do**
      Push (p_j, s_j) to Q, mark s_j as seen
      if `do_search` **then**
        break
      end if
    end for
  end while
  Append `expanded` to `expansions`
end for
Output: Solved rate = `solved_count`$/n$
Output: Average expansions = mean(`expansions`)

3.3.2 Contrastive Learning

Version with search

- *solved rate*: 0.994,
- *average expanded nodes*: 27.491.

Version without search

- *solved rate*: 0.561,
- *average expanded nodes*: 16.728.

3.3.3 Stitching

Version with search

- *solved rate*: 0.997,
- *average expanded nodes*: 33.607.

Version without search

- *solved rate*: 0.391,
- *average expanded nodes*: 14.767.

As we can see, *contrastive* learning, even though has high bias, better predicts the relations between two states (i.e. which one is closer to the goal state). As a result it has higher solved rates for version without search and lower average number of expanded nodes for both methods. The solved rate for version with search is close to one, as the method should always find a solution. The only problem is when the start and goal state are the same. I decided not to fix this issue, because it has a low impact on results.

4 Results

4.1 Supervised Training

Training *supervised* model requires a lot of training steps. For some time it probably oscilates around a local minimum of the loss function. After around 50000 steps, the loss function started decreasing. It dropped to around zero, but it does not look to overfit, because the loss decreases monotonously. The training curve is presented below.

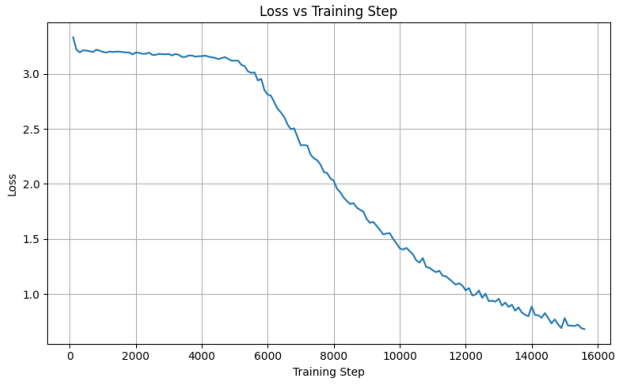


Figure 7: Loss during *supervised* training.

4.2 Contrastive Training

Contrastive training is smoother. It requires less steps to reach the goal, but after 1000000 steps it was trained on, it probably overfits.

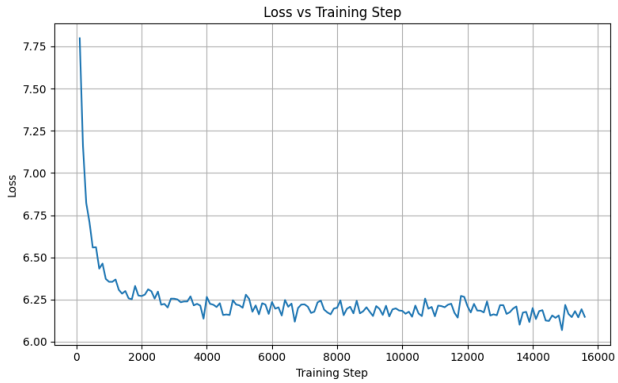


Figure 8: Loss during *contrastive* training.

5 Analysis

5.1 Overall Performance

Based on the metrics implemented, all three models (*supervised*, *contrastive*, and *stitching*) achieved decent performance. However, the models differed significantly in their performance in solving the maze task without search and in both *solved rate* the *average number of expanded nodes*.

The *contrastive* model demonstrated the most efficient search behavior, achieving a better trade-off between solved rate and number of expansions. This suggests that the contrastive latent space better reflects relative proximity to the goal, even if absolute distances are biased. The *supervised* model struggled in the version without search, indicating that its predictions are less reliable for ranking states by proximity.

Heatmap visualizations further support this: the contrastive and stitching models produce smoother gradients across space, while the supervised model sometimes gives locally inconsistent predictions. Correlation plots align with this analysis: although supervised learning performs well in many cases, contrastive learning shows more consistent, monotonic alignment between latent distance and actual path distance.

5.2 Supervised vs. Contrastive Comparison

The supervised model is trained to directly predict the number of steps to the goal as a classification task, which leads to high accuracy when the model generalizes well. However, it can make large mistakes due to its discrete output space and lack of structural constraints. In contrast, the contrastive model learns a representation space where distance corresponds to temporal distance. While this approach introduces bias, especially for shorter distances, it tends to better preserve ordering and neighborhood relationships.

The contrastive model also leads to more effective planning in the search algorithm. With a lower number of expanded nodes and a higher solved rate without exhaustive search, contrastive learning proves more useful for practical decision-making tasks.

5.3 Stitching Capabilities

To evaluate stitching, I tested whether the model could generalize to novel trajectories by combining fragments of previously seen ones. The model was trained on partial trajectories and then evaluated on longer paths.

The stitching model shares its architecture and training procedure with the *contrastive* model but is trained on a different dataset. It performed comparably to the contrastive model in terms of correlation and heatmap consistency, although the bias in predicted distances increased slightly. This is expected, as stitching introduces more complex generalization challenges. Still, its solved rate and efficiency in planning were better than supervised learning and only slightly worse than the original contrastive model, which demonstrates the success of the stitching mechanism in generalizing over composite trajectories.

6 Conclusion

In this project, I implemented and analyzed three metric learning approaches for estimating distances in a maze-like environment: supervised learning, contrastive learning, and stitching-based contrastive learning.

Supervised learning showed good results when the model generalized correctly but suffered from discrete misclassifications and poor planning performance without search. Contrastive learning provided smoother latent representations that preserved relative distances, enabling more efficient planning and higher success rates, especially without exhaustive search. Stitching, though introducing more bias, retained these advantages and successfully generalized over composed trajectories.

Overall, contrastive methods demonstrated better alignment with the goals of metric learning—providing useful distances for planning—while supervised learning may still be suitable when precise distance classification is needed. Future improvements could involve combining both approaches or applying additional regularization to reduce bias in contrastive embeddings.