Deep Learning 2024 Final Project Report

Satyam Kumar
Computer Science
New York University
`sk12075@nyu.edu`

Jayesh Chaudhari
Computer Science
New York University
`jsc9903@nyu.edu`

Lucas Wang
Biomedical Engineering
New York University
`jw8269@nyu.edu`

Abstract

Joint embedding prediction architecture, or JEPA, is an energy-based architecture that has been proposed for self-supervised learning. In this project, we aimed to implement JEPA architecture using a dataset of exploratory trajectories, with the goal of predicting the correct location of the agent in each image in the dataset. This problem was challenging primarily due to the need to prevent collapse, which we attempted to address through testing various methods involving combinations of VicReg, Barlow Twins, and BYOL. Ultimately, after multiple rounds of tweaking and modifying our approach, we arrived at a method using a CNN with VicReg combined with the action projection techniques used in MuZero and the moving average approach used for BYOL, and obtained a probe normal loss of 2.54 and a probe wall loss of 5.21.

1 Introduction

Joint embedding prediction architecture (JEPA) is an energy-based architecture used for self-supervised learning. This architecture was first proposed by (1) with the aim of predicting high-level representations, or embeddings, of future observations.

The goal of this project was to implement and train a JEPA architecture on a dataset consisting of 2.5M frames of exploratory trajectories. These exploratory trajectories were collected from a toy environment with an agent (depicted by a dot) is in a space with two rooms separated by a wall with a small gap (the 'door'). The agent cannot travel through the walls, only the door, and there are many different possible environment layouts. The JEPA must be able to perceive the layout of the environment, and it is then evaluated on its ability to capture the true coordinate location of the agent. The tricky part of approaching this problem lies in preventing model collapse, which occurs when all outputs become identical and lack diversity throughout, resulting in uninformative information being passed downstream.

2 Literature Review

To approach this problem, we first investigated the relevant literature on JEPA, VicReg and BYOL.

In (1), JEPA is described as an architecture that focuses on learning a joint embedding space to capture the relationships between context (known information) and target (predicted information) within a dataset. Instead of directly generating data, JEPA learns representations that can then be used for prediction or decision-making.

VicReg, or Variance-Invariance-Covariance Regularization, was first introduced by (2). This approach uses two identical networks to process different augmented views of the same input. The outputs are then compared and regularized to arrive at the ideal results. VicReg also uses invariance, variance, and covariance in its loss term to ensure that the network learns meaningful features, that there is diversity in the learned features, and that the features are non-redundant, respectively, all of which helps to prevent model collapse.

Finally, BYOL, or Bootstrap Your Own Latent, was first introduced by (3), and involves two networks: one as an online network, with an encoder, projector, and predictor, and one as a target network, where encoder and projector are moving average of online network. The objective is to minimize the mean squared error between the predictor output from the online network and the output of the target network. This approach aims to prevent collapse by using the asymmetric architecture of the predictor and target network and the moving average mechanism for the target network. Since the target network moves slowly, this also helps to ensure stability and prevent trivial solutions.

In addition to this provided literature, we also reviewed (4), which introduced the MuZero algorithm. This algorithm learns a model of the environment's dynamics directly from data. In particular, we were interested in the action projection technique outlined in this paper, which utilizes directly broadcasting the one hot action vector into action planes. This is different from typical action projection techniques. Instead, the approach from MuZero uses a learned model of the environment to first predict actions and then project them into action planes.

3 Method

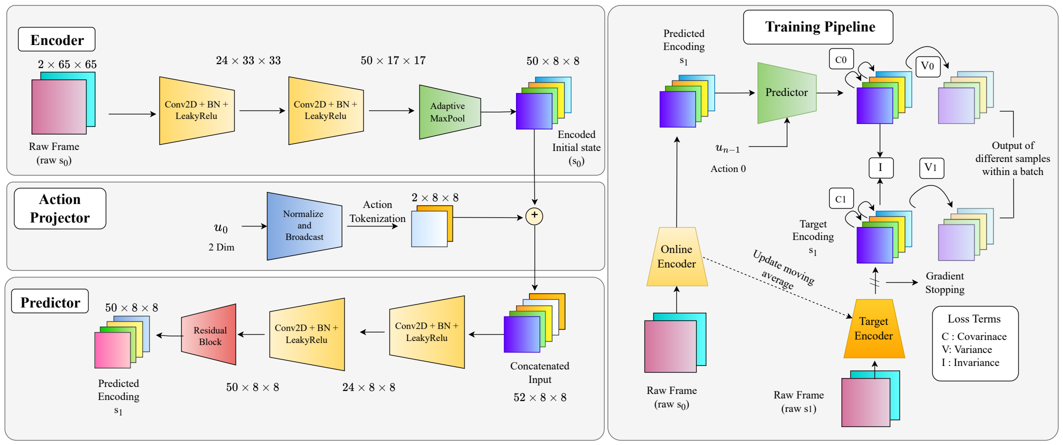

Figure 1: Architecture and training pipeline.

To approach this problem, we iterated over many different stages each with new insights in order to arrive at our final architecture and best results, with the details of these iterations outlined in the next section. Our final architecture, which can be seen in 1, utilized a CNN, with an encoder with two convolutional layers followed by a max-pooling layer, and a predictor with two convolutional layers followed by a residual block. Constraining encoder and predictor to have spatial structure preserved helped the JEPA model to learn relative position between agent and the door opening.

Additionally, projecting actions into action planes where each element is broadcasted from the action vector taking inspiration of action projection technique from (4) to optimize our CNN architecture and guide our embeddings appropriately over each stage.

To address the problem of preventing collapse, we implemented and compared the three methods of VicReg, Barlow Twins, and BYOL. We saw promise in the VicReg method, so we utilized this by calculating the variance, invariance, and covariance with the weighted sum of these being represented as D in the JEPA diagram. Furthermore, we applied BYOL's weight update technique in the encoder, allowing for our loss to improve even more significantly than when using VicReg alone. Our training pipeline is illustrated in 1.

4 Results from each stage

Ultimately, our best results were **2.54** for the probe normal loss, **5.21** for the probe wall loss, **4.96** for the probe wall other loss, and **36.00** for the probe expert loss. These results were the outcome of many iterations of our model, with multiple steps of trial and error and tweaking different aspects of our approach. A summary of the development of our loss values over these iterations is shown in 4.

At first, we tried using a CNN-based encoder to project the input into a 1×512 representation space (the encoder output). The actions (dimension 1×2) were concatenated after being projected to a dimension of 1×512. However, this approach resulted in very poor loss values of 280 for the probe normal loss and 200 for the probe wall loss, which was even worse than the mock model provided in the base code. This was likely due to the fact that naively projecting actions into higher dimension of 512 without spatial dimensions was not very meaningful, and we instead realized that actions needed to be projected into a 2-dimensional space along with a hidden dimension. Since the action vector represents δx and δy information for the agent, we realized that this information must be encoded within spatial constraints in order to result in the best learning.

To improve upon our initial attempt, we modified the projection of actions to project the original 1×2 dimensions into a $2 \times 4 \times 4$ space using a linear layer and then concatenated it with the encoder output (dimension $64 \times 4 \times 4$). Again, we utilized a CNN architecture, using 2 convolutional layers for the encoder and an additional 2 for the projector. This resulted in an improvement in our loss values, with 230 for the probe normal loss and 150 for the probe wall loss, but the model still collapsed even before completing one epoch, despite the MSE loss going down. From this, we realized that our approach only minimized the loss by outputting the same predicted encodings and target encodings to reduce the loss, and explored ways to improve further.

After this, we added regularization terms to try to prevent the collapse, specifically introducing variance and covariance terms as described in (2). The variance loss term used a hinge loss to ensure that the standard deviation (over a batch) of each variable in the embedding remained above a certain threshold, which promoted diversity within batch samples. The covariance loss term reduced covariance (over a batch) between pairs of centered embedding variables, preventing informational collapse by decorrelating the variables. The addition of these terms resulted in significant improvements in our loss values, with 100 for the probe normal loss and 80 for the probe wall loss, but the model still collapsed after two epochs. In particular, we noticed that the introduction of our regularization terms resulted in better generalization, but the evaluation loss was still too high. This was likely due to the fact that our implementation was a naive implementation of action projection and encoder design, which was not adequately capturing the important cues necessary for next-state prediction. Adjusting the coefficients of our terms, such as increasing the variance loss weight, did not resolve the issue, so we decided to move on to optimizing the architecture itself.

Now, we decided to test various different architectures, including CNN, ViT, LSTM, and GRU. Ultimately, we found that the best approach was to use an encoder with two convolutional layers followed by a max-pooling layer, and a predictor with two convolutional layers followed by a residual block. We also utilized the action projection technique outlined in (4), using a 2-dimensional δx and δy for our action, which guides the previous state into the next state. This allowed us to broadcast the values over an action plane without using any linear learned projections, guiding the previous embeddings of each patch into the direction of the action. For 2 actions, we used 2 action planes, each with the shape 8×8. At the end, we received an output in the shape $2 \times 8 \times 8$ as an action token, which we then appended to the previous state's encoding and passed as an input to the predictor's refined network. Ultimately, this resulted in substantial improvements in loss values, with 50 for the probe normal loss and 30 for the probe wall loss.

From there, we introduced moving averages to the encoder, inspired by the techniques outlined in (3). This allowed gradient flow to only occur through Enc_θ (the online encoder in (3)), whereas the rest of the encoders were a moving average of the first encoder. This resulted in even greater improvements in our loss values, with 4 for the probe normal loss and 10 for the probe wall loss.

This approach ensured the target network parameters were updated as a slow-moving average of the online network's parameters, introducing an asymmetry in the learning process. This asymmetry ensures that the two networks do not prematurely converge to identical representations, which helps to prevent trivial solutions where all outputs collapse to a constant vector. The moving averages we implemented also acts as a method of regularization, stabilizing the overall training dynamics. Consequently, the target network can evolve gradually, smoothing out erratic updates in the online network and ensuring that the learning signal remains consistent throughout. Additionally, while VicReg loss terms (from (2)) attempt to move towards minimizing variance and covariance, our approach forces two constraints together for both variance and covariance, emphasizing this phenomenon in a much greater way.

Finally, we settled on fine-tuning our hyperparameters, experimenting with various initializations, activation functions, learning rates, and so on. In particular, we found that changing ReLU to Leaky ReLU resulted in greater improvements in our results, with our best results ending at 2.5416 for the probe normal loss, 5.2102 for the probe wall loss, 4.9613 for the probe wall other loss, and 36.0063 for the probe expert loss. These improvements are likely due to the fact that Leaky ReLU provides the benefit of avoiding dead neurons by maintaining a small gradient even for negative inputs, conserving small variations in activations and ensuring better gradient flow overall. This also helped to support our model in preventing collapse by maintaining diversity in representations throughout the network.

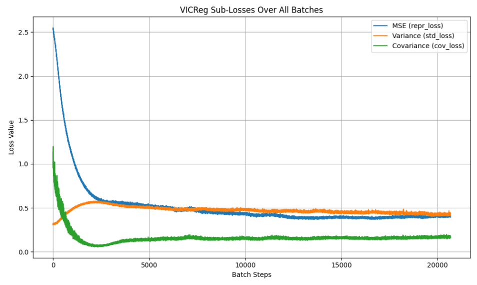

Figure 2: Training Loss Visualization
Green represents Covariance Loss, Orange represents Variance Loss and Blue represents MSE Loss

Improvements at Each Stage	**Probe Normal Loss**	**Probe Wall Loss**
Initial Attempt	280	200
Action Projection	230	150
Reg. Terms	100	80
CNN Arch	50	30
Moving Average	4	10
Fine-Tuning	**2.54**	**5.21**

Table: Loss Values Over Various Stages

5 Future Directions

Beyond our attempts, we also recognize that our approach could be further improved using various methods. Our method shows great generalization on probe wall, probe normal and probe wall other but fails on the probe expert. After visualizing the probe expert data and understanding the complexity of the trajectories our model is not able to generalize on these difficult scenarios thus yielding 36 eval loss for probe expert. For starters, we could try applying BYOL to the predictor network to see if it would result in improvements in our loss values. We could also try to obtain further improvements on the probe expert loss in particular by introducing more constraints on loss terms, using techniques similar to those outlined in Barlow Twins (5). Rather than using a fixed trajectory length of 17 in training , we could implement a curriculum where we first train the model to only predict next 4 states then to next 8 and then to next 17. This form of curriculum could help the model to learn effectively and methodically. Additionally, after viewing the final presentations from the top-ranked teams, we also identified techniques they used that could have led to improvements in our results, such as decoupling in the encoder instead of concatenating, or using 3D embeddings instead of 2D.

References

[1] Y. LeCun, "A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27," *Open Review*, vol. 62, no. 1, pp. 1–62, 2022.

[2] A. Bardes, J. Ponce, and Y. LeCun, "Vicreg: Variance-invariance-covariance regularization for self-supervised learning," *arXiv preprint arXiv:2105.04906*, 2021.

[3] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, *et al.*, "Bootstrap your own latent-a new approach to self-supervised learning," *Advances in neural information processing systems*, vol. 33, pp. 21271–21284, 2020.

[4] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, p. 604–609, Dec. 2020.

[5] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, "Barlow twins: Self-supervised learning via redundancy reduction," in *International conference on machine learning*, pp. 12310–12320, PMLR, 2021.