Methods for Training and Evaluating the JEPA Model

Adam Xu
Center for Data Science
New York University
tx543@nyu.edu

Cindy Lin
Center for Data Science
New York University
cyl4981@nyu.edu

Ao Xu
Center for Data Science
New York University
ax2183@nyu.edu

Abstract

This report describes the methods employed for training and evaluating the Joint Embedding Prediction Architecture (JEPA) model in a self-supervised learning setting. The JEPA model, applied to a toy environment, predicts its own representations of future observations, enabling effective modeling of agent trajectories. The focus is on the model's architecture, training pipeline, and evaluation techniques, with special attention to preventing representation collapse and improving generalization.

1 Introduction

Self-supervised learning has emerged as a powerful paradigm for training models without explicit labels. The Joint Embedding Prediction Architecture (JEPA), proposed by LeCun (2022), exemplifies this approach by learning to predict its own representations. In this project, we implement and train a JEPA model to simulate agent trajectories in a toy environment consisting of two rooms separated by a wall and door.

The JEPA model relies on three core components: the encoder, predictor, and target encoder. It minimizes a distance-based energy function to align predicted and target representations. To ensure robustness, techniques such as contrastive loss, regularization, and time consistency are used to prevent representation collapse.

1.1 Dataset

The dataset consists of 2.5 million frames of agent trajectories collected in a toy environment. Each trajectory captures the agent's interaction with the environment, including walls and doors. The dataset includes:

- **States:** Observations are two-channel images of size 64×64, where the first channel represents the agent and the second channel represents walls and borders.
- **Actions:** Vectors of shape $(\Delta x, \Delta y)$ specifying position shifts from the agent's previous global position.

Training data is located in `/scratch/DL24FA/train`, while probing datasets for validation are divided into two categories:

- **Probe Normal:** Validation trajectories similar to the training data.
- **Probe Wall:** Validation trajectories with agents interacting near walls and doors.

The objective is to evaluate the JEPA model's ability to encode agent coordinates (y_1, y_2) while generalizing to novel scenarios.

Preprint. Under review.

2 Architecture

The JEPA model comprises three main components: the *Encoder*, the *Predictor*, and the *Target Encoder*.

2.1 Model Components

- **Encoder:** The encoder is a convolutional neural network (CNN) designed to extract meaningful features from the input observations. It consists of three convolutional layers with ReLU activations, followed by a fully connected layer that maps the extracted features to a latent space. The encoder's output is a compact representation of the input observation, making it suitable for downstream tasks like prediction and probing.
- **Predictor:** The predictor is a fully connected network that predicts the next latent state based on the current latent state and the action taken. It concatenates the current state and action vectors as input and outputs the predicted next state. This component enables the JEPA model to simulate future trajectories in the latent space, a critical capability for self-supervised learning tasks.
- **Target Encoder:** The target encoder serves as a stable reference for the model during training. It is structurally identical to the main encoder but is updated using a moving average of the main encoder's weights. This approach ensures that the target encoder provides consistent target representations, mitigating the risk of representation collapse.

2.2 Training Objective

The JEPA model is trained to minimize the energy $F(\tau)$ of an observation-action sequence τ:

$$F(\tau) = \sum_{n=1}^{N} D(\tilde{s}_n, s'_n),$$

where $\tilde{s}_n$ is the predicted state, s'_n is the target state from the target encoder, and $D(\cdot, \cdot)$ is a distance function, such as Smooth L1 Loss. This objective ensures that the predicted states align closely with the target states. Regularization techniques, including contrastive loss and time consistency, are added to prevent degenerate solutions.

3 Implementation

3.1 Training Pipeline

The training pipeline is designed to handle large-scale data efficiently and ensure robust learning. The key steps include:

1. **Input Preparation:** The training data, consisting of states and actions, is loaded and normalized. States are represented as 4D tensors (B, T, C, H, W), where B is the batch size, T is the trajectory length, C is the number of channels, and $H \times W$ are the spatial dimensions.
2. **Forward Pass:** The encoder processes the initial observations to compute the latent states. The predictor iteratively computes the predicted states for future time steps based on the current latent state and actions.
3. **Loss Computation:** The total loss is computed as a weighted combination of:
   - **Prediction Loss:** Smooth L1 loss between the predicted and target states.
   - **Consistency Loss:** Encourages alignment between the predicted latent states and the current latent states derived from the encoder.
   - **Time Consistency Loss:** Penalizes abrupt changes in predictions over consecutive time steps, ensuring temporal smoothness.
4. **Backpropagation:** Gradients are computed and used to update the model parameters. Gradient clipping is applied to stabilize the training process.

3.2 Evaluation

Evaluation focuses on the quality of the learned latent representations. The process includes:

1. **Probing Tasks:** A two-layer fully connected network (prober) is trained to predict the agent's (y_1, y_2) coordinates from the latent states. This tests how well the latent states capture the underlying spatial information.
2. **Validation Metrics:** The model is evaluated using Mean Squared Error (MSE) on two datasets:
   - **Probe Normal:** Contains trajectories similar to the training set.
   - **Probe Wall:** Includes trajectories where the agent interacts with walls and doors.
3. **Long-Horizon and Generalization Tests:** Hidden validation datasets are used to assess the model's performance on long-horizon predictions and generalization to novel layouts.

The evaluation results include:

- **Normal Loss:** 263.34
- **Wall Loss:** 185.73
- **Wall Other Loss:** 187.77

These metrics evaluates the model's capability to generalize and adapt to diverse scenarios.

4 Training Details

Optimizer: The Adam optimizer is configured with a learning rate of 6×10^{-4} and gradient clipping at a maximum norm of 2.0.

Scheduler: A cosine learning rate scheduler adjusts the learning rate dynamically throughout training, ensuring smooth convergence.

Batch Size: Each training iteration processes a batch of 256 samples, balancing computational efficiency and stability.

Epochs: The model is trained for 15 epochs, allowing sufficient iterations for convergence while preventing overfitting.

Loss Design: The total loss is a combination of:

- Prediction loss (Smooth L1) to align predicted and target states.
- Consistency loss to encourage agreement between representations.
- Time consistency loss to ensure smooth predictions over time.

5 Conclusion

In this project, we implemented and evaluated a JEPA model for self-supervised trajectory prediction in a simplified two-room environment. Despite demonstrating some ability to capture agent dynamics, the model's performance did not meet expectations, particularly for long-horizon and complex scenarios. Our analysis suggests that the current architecture and training approach have several notable limitations:

- **Insufficient Representation Capacity:** Although the encoder–predictor–target encoder pipeline captures basic features, the relatively shallow CNN and fully connected predictor may limit the expressive power needed to handle intricate spatial interactions and long-range dependencies.

- **Hyperparameter Sensitivity:** Our experiments relied on a fixed set of hyperparameters, which may not be optimal. A more systematic hyperparameter search or adaptive weighting strategy could yield better training stability and improved representations.
- **Potential Over-Simplifications in Loss Functions:** While prediction, consistency, and time-smoothness losses were employed to avoid representation collapse, they might not adequately capture the full complexity of state transitions. Additional regularization might help refine latent representations.

Future Work To address these limitations and improve performance, we can consider the following directions for further work:

- **Model Architecture Enhancements:** Incorporate deeper or more advanced architectures to better capture complex spatial and temporal patterns. Exploring attention-based modules may help the model focus on crucial regions of the state space.
- **Data Augmentation and Environment Complexity:** Augment training data by adding stochastic elements, dynamic obstacles, or additional room layouts to promote generalization. Expanding to more realistic or diverse simulation environments could also reveal the model's true capacity.
- **Adaptive Loss Design:** Investigate alternative or additional loss components that encourage a richer understanding of agent dynamics.

By pursuing these improvements, the JEPA model can potentially better capture latent dynamics, avoid representation collapse, and perform robust self-supervised trajectory predictions even in more complex or long-horizon scenarios.

References

- LeCun, Y. (2022). *A Path Towards Autonomous Machine Intelligence*. Published online.
- Vaswani, A., et al. (2017). *Attention is All You Need*. Advances in Neural Information Processing Systems (NeurIPS).
- He, K., et al. (2016). *Deep Residual Learning for Image Recognition*. CVPR.
- Kingma, D. P., and Ba, J. (2015). *Adam: A Method for Stochastic Optimization*. ICLR.