Department of Information Engineering and Computer Science

Master's Degree in
Artificial Intelligence Systems

Final Dissertation

Modular Non-Autoregressive Trajectory Prediction

Supervisors

Enver Sangineto
Andrea Passerini

Student

Mátyás Vincze

Academic year 2022/2023

Acknowledgements

I would like to express my sincere gratitude to my supervisors Enver Sangineto and Andrea Passerini for their invaluable supervision and guidance during the development of this thesis. Their expertise and support have been instrumental in shaping this work and making it a valuable learning experience. I would also like to extend my appreciation to all my friends and family for their support throughout this process. Moltes gràcies Carla.

Contents

Abstract

The safety and effectiveness of numerous systems, ranging from self-driving vehicles and social robots to delivery drones, heavily depend on accurate predictive models that facilitate real-time analysis of the surrounding environment and the movements of nearby entities. To ensure the development of robust systems, it is crucial to achieve scalability without compromising efficiency and sparsity. Social interactions often occur between independent small groups of entities, making sparsity a key consideration. While Graph Neural Networks (GNNs) have been traditionally used to model these interactions, their inherent density and lack of scalability pose limitations.

In this thesis, we propose SI-NPS, a modular non-autoregressive model with a convolutional backbone, which introduces a novel research direction by factorizing knowledge while maintaining scalability. To evaluate the performance of SI-NPS, we conduct a benchmark comparison with 15+ state-of-the-art models. Our results demonstrate that SI-NPS outperforms its predecessors while remaining comparable to models with 50x parameters in terms of Average Displacement Error (ADE) and Final Displacement Error (FDE) when trained and evaluated on the SDD and ETH/UCY datasets. Furthermore, we explore potential enhancements to the model structure and systematically test each segment of the model to gain a deeper understanding of its capabilities and limitations.

Code and datasets are provided in the link: https://github.com/matyas-vincze/SI-NPS

1 Introduction

Trajectory prediction is a crucial skill for autonomous systems navigating in complex and dynamic environments. It involves forcasting the future paths of entities by processing and analyzing their past movements. This capability has utility in various fields of applications, including self-driving cars and surveillance drones. Despite the remarkable progress of computer vision research, particularly in object detection and semantic segmentation, there persists a significant challenge in accurately modeling social behaviour and joint prediction for trajectories of multiple entities.

The learning problem of representations and relations between objects has been historically solved by representing the data as a graph, with edges representing the dependence between the entities. Then Graph Neural Networks (GNNs) can be used to learn about these connections, and they have been successfully applied for trajectory prediction as well. However, scalability and efficiency issues arise when applying GNNs to large, sparse graphs commonly found in complex real-world scenarios. This motivates exploring alternative approaches.

In this thesis, we present an innovative deep learning model SI-NPS that integrates Neural Production Systems (NPS) [9] into state-of-the-art trajectory forecasting techniques. Our primary goal was to create accurate and efficient models capable of real-time performance on edge devices. We placed a strong emphasis on incorporating inductive biases to introduce structured sparsity on both local and global scales. Leveraging the inherent entity-conditional knowledge factorization in the NPS framework, we aim to overcome the limitations of previous Graph Neural Network (GNN)-based methods.

After providing background on the NPS and Social-Implicit [1] modeling approaches, we elaborate on how their combination enables modularity and sparsity well-suited for efficient trajectory prediction. The Social-Implicit model introduces a non-autoregressive framework based on spatial- and temporal-convolutions. Simultaneously, the NPS model's modular architecture affords efficiency benefits through dynamic selective activation.

We evaluate the effectiveness of the SI-NPS model on common benchmark trajectory datasets. The experiments analyze the trade-offs between accuracy and efficiency achieved by varying sparsity hyperparameters. Specifically, we study the performance of the model based on its size, and we present a couple of promising modifications.

In summary, the key contributions are:

- Demonstrating benefits of structured sparsity for efficient trajectory prediction
- Proposing a novel SI-NPS architecture
- Evaluating model performance on trajectory forecasting datasets
- Providing insights on balancing accuracy and efficiency
- Introducing modifications to the NPS model, and show how it effects the evaluation of SI-NPS

This research combines state-of-the-art techniques in a novel way to address limitations of prior works. The results advance efficient and accurate trajectory forecasting for autonomous agents in complex environments, as well as give insights to the research fields of modular deep learning and social learning.

1.1 Problem formulation

The ability to accurately forecast the future positions of entities based on their past trajectories enables safe and efficient navigation, planning, and decision-making. While the earliest approaches relied on

hand-crafted physics-based models, data-driven techniques using deep neural networks have become the dominant paradigm due to their superior performance.

The primary objective is to accurately forecast entity's forthcoming movement patterns. A diverse range of data modalities can be employed for this task, including raw images, coordinates, bounding boxes, and semantically segmented images. In this particular work, we focus on the scenario where the model inputs exclusively the extracted coordinate pairs.

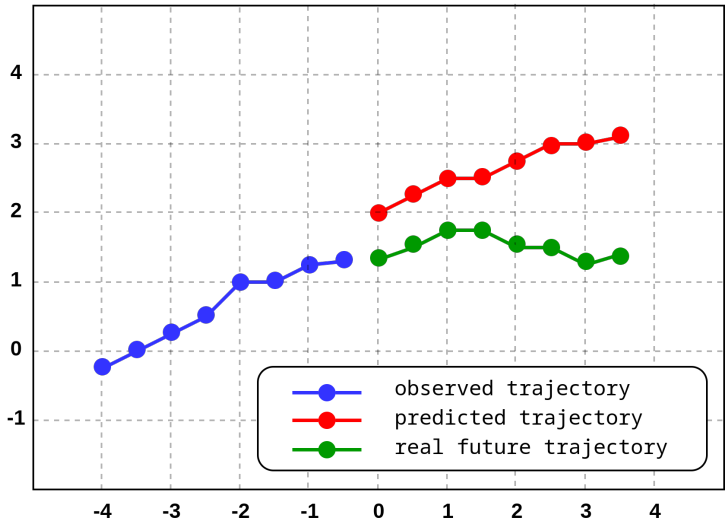


Figure 1.1: Example trajectory.

The data can be represented as a series of historical positions over T_{obs} time-steps. The objective is to predict the continuation of the trajectory for the following $T_{\text{pred}} - T_{\text{obs}}$ time-steps. Each position is defined as a 2-dimensional coordinate-pair (x_t, y_t).

To simplify the learning process and highlight the underlying dynamics, the data is often converted into relative coordinates / velocities, represented as (v_{x_t}, v_{y_t}). This conversion is done by calculating the differences between consecutive coordinates-pairs:

$$(v_{x_t}, v_{y_t}) = (x_t - x_{t-1}, y_t - y_{t-1})$$

This transformation serves two purposes:

1. It standardizes the data, making it suitable for analysis and modeling.
2. It shifts the focus from specific positions to the more crucial aspect of how the data evolves over time.

For simplicity and clarity, we define:

$$X_t := (x_t, y_t), \text{ for } t \in [0, T_{\text{obs}-1}]$$
$$Y_t := (x_t, y_t), \text{ for } t \in [T_{\text{obs}}, T_{\text{pred}-1}]$$

By organizing the input data in this way, we encapsulate the historical trajectory up to time $T_{\text{obs}-1}$, and our goal is to predict the future trajectory from T_{obs} until the time span of $T_{\text{pred}-1}$.

In this thesis, we delve into the intricate world of trajectory prediction, focusing on cases where the model's input is limited to extracted coordinate-pairs. We use this segment of real-world applications of machine learning to benchmark the Neural Production System framework and experiment with our improvements on this model.

By examining the problem and investigating the dynamics of movement patterns, we aim to enhance our understanding of how entities navigate through space and provide valuable insights for improving trajectory prediction performance.

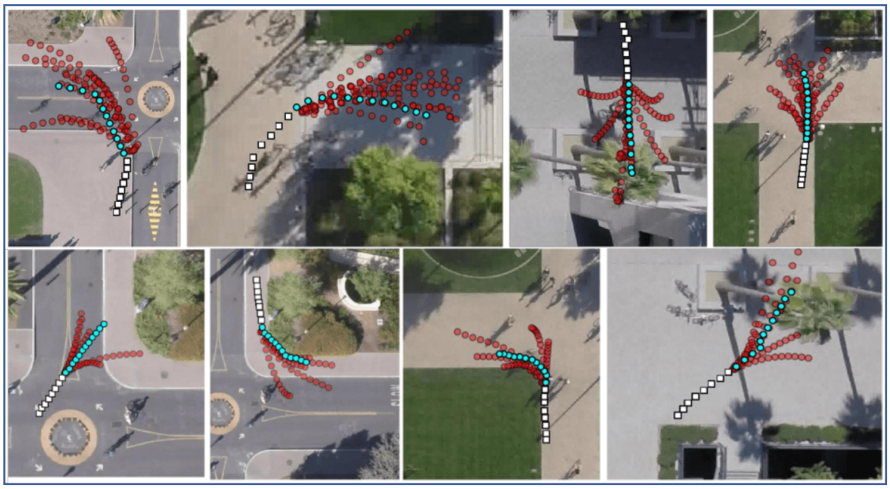


Figure 1.2: Example data from the UCY dataset. White coordinates are the input data, red is the set of predictions, and blue is the ground truth. Image taken from [27]

Through this investigation, we hope to lay the groundwork for more robust and reliable trajectory prediction models that can be applied in various fields such as robotics, autonomous vehicles, and human behavior analysis.

1.2 Applications

The ability to accurately predict the trajectories of multiple interacting entities has become increasingly important in various fields, especially those involving autonomous systems operating in complex, dynamic environments. The applications of multi-agent trajectory forecasting span a diverse range of domains:

Autonomous Vehicles

Self-driving cars rely heavily on understanding intentions and predicting movements of surrounding pedestrians, cyclists and other vehicles. Accurate short and long-term motion forecasts enable safe and efficient navigation in crowded urban settings.

Surveillance

Intelligence agencies and law enforcement leverage AI to track objects of interest in crowded spaces using a network of surveillance cameras. Multi-agent trajectory prediction can help identify normal vs anomalous behavior, as well as forecast potential collisions or unsafe size of gatherings. This assists security professionals in resource allocation and precautionary intervention.

Sports Analytics

Player tracking is used to extract tactics and strategies in team sports. Forecasting player movements and interactions helps quantify individual contributions and team dynamics. This provides coaches and analysts with detailed insights to improve performance. Trajectory prediction also assists in automatically generating game highlights and summaries.

Computer Graphics

Digital humans in films, video games and the metaverse require realistic motion planning to move naturally in virtual environments. Physics-based simulation can be combined with data-driven trajectory forecasting for generating smooth, context-appropriate navigation and interactions.

1.3 Background

The literature on trajectory prediction can be broadly categorized into two main approaches: physics-based models and data-driven models.

Physics-Based Models

Physics-based models treat entities as particles subject to forces from nearby objects and obstacles. One of the earliest approaches in this category is the Social Forces Model [14], which calculates the next entity position based on the sum of these forces. Another example is the BRVO model [19]. However, these models have limitations in capturing complex social interactions.

Data-Driven Models

With the advent of deep learning, data-driven models have gained popularity due to their promising results. The first work to achieve state-of-the-art results using neural networks was Social LSTM [2], which outperforms physics-based approaches by considering not only the past trajectory but also a representation of social information for each entity.

Since 2016, various deep learning architectures have been proposed to improve the state-of-the-art, including:

- Generative Adversarial Networks (GANs) [8]
- Graph Neural Networks [33]
- Integration of attention mechanisms [37]
- Transformer-based architectures [7]

High accuracy of deep learning algorithms makes the relevance of knowledge-based models for predicting local trajectories questionable. However, the ability of deep learning algorithms for large-scale simulation and the description of collective dynamics remains to be demonstrated.

1.4 Purpose

Prior work in trajectory prediction has been dominated by graph neural network (GNN) based approaches, as the data and its social aspect can be easily represented as a graph. However, GNNs face limitations in terms of scalability and efficiency when applied to the large, sparse graphs commonly found in complex real-world scenarios. The inherent density of GNN models often does not match the sparsity of real-life social interactions. GNNs also struggle to model interactions happening in small, independent groups. These issues motivate exploring alternative techniques beyond standard GNN architectures. Opposite to the recent transformer hype, we seek to experiment with different approaches that are more suitable for the application at hand.

Neural Production Systems (NPS), introduced in recent work [9], provide an efficient, versatile and sparse framework compared to GNNs. NPS consists of modular, independent production rules that can be combined to model complex behaviors using these simpler sub-models. The rules are abstract, independent, facilitating knowledge transfer across different domains. Crucially, NPS induces sparsity by matching only a small subset of entities to each rule. This makes NPS well-suited for applications where interactions tend to be sparse, such as trajectory prediction. The use of independent rules make the model scalable, while including stronger inductive bias toward structured sparsity than GNNs.

The purpose of this work is to incorporate NPS into a suitable SOTE architecture used for trajectory prediction, to test how well it does compared to current solutions. In summary, this research proposes a novel combination of state-of-the-art techniques - NPS and Social-Implicit - to advance efficient and accurate trajectory prediction. The goal is developing models capable of balancing accuracy and efficiency through modularity and sparsity, while demonstrating improved generalization over prior approaches. Furthermore we present possible modifications to the NPS framework and

show how they influence the performance. The results provide valuable insights into optimizing this accuracy-efficiency trade-off for real-world deployment of trajectory forecasting systems.

1.5 Outline

Chapter 2 presents a comprehensive review of related work, covering graph neural networks, modular machine learning techniques like Mixture of Experts and Neural Production Systems, as well as state-of-the-art trajectory prediction models.

Chapter 3 describes the model setup, including details on the format of the input data, data augmentation, loss function, as well as the final model architecture.

Chapter 4 explains the experimental setup. This includes the benchmark datasets, optimizers, hyperparameters, evaluation metrics and ablation studies. Results on pedestrian trajectory forecasting tasks are presented and compared to prior work. This chapter critically analyzes the experimental results, discusses limitations of the current approach, and suggests promising directions for future work to further improve efficiency and accuracy.

Chapter 5 summarizes the main contributions of combining modular deep learning with state-of-the-art trajectory prediction techniques. The key takeaways regarding the viability of structured sparsity for efficient and generalizable trajectory forecasting are highlighted.

2 Related work

The development of models for trajectory prediction has seen rapid progress in recent years. However, it remains a challenging problem to effectively leverage contextual information and complex navigation behaviors. In this section, we review relevant literature that has driven advancements in this field and provide context for our own research contributions.

This chapter provides necessary background by reviewing relevant literature on graph neural networks, modular deep learning, and trajectory forecasting models. The discussion lays the groundwork for the proposed integration of Neural Production Systems with Social Implicit in chapter 4.

2.1 Graph Neural Networks

Graphs serve as representations of interconnected entities, finding utility across diverse domains like social networks, biological data analysis, and knowledge graphs. Graph-related tasks can be performed at different levels, including graph, node, or edge level, and involve graph generation, evolution, property prediction, and community detection.

Graphs can be represented either as a collection of edges or in the form of an adjacency matrix. Machine learning techniques applied to graphs primarily aim at generating meaningful representations for nodes, edges, or entire graphs. These representations subsequently serve as the basis for training predictive models for specific tasks. Traditional methods include feature engineering and walk-based approaches, while more recent advancements incorporate Graph Neural Networks (GNNs).

GNNs are expressly designed to maintain invariance under permutations and equivariance, ensuring that the graph's representation and its permutations remain consistent after processing. A fundamental GNN architecture comprises three core elements:

1. Message-passing mechanism that aggregates information from neighboring nodes
2. Node update function that refines node representations based on the aggregated information
3. Readout function that combines node representations to yield the final output

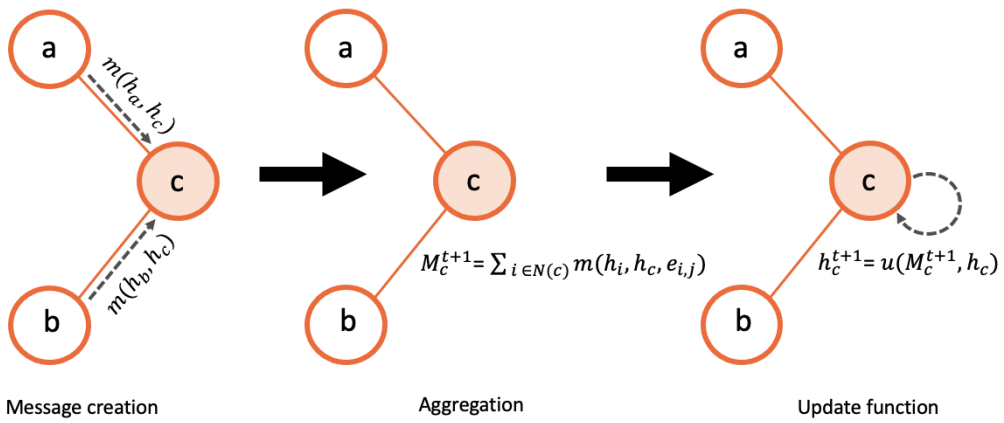


Figure 2.1: Graph Neural Network message-passing iteration.

Various GNN models have been proposed, including Graph Convolutional Networks (GCNs) [21], Graph Attention Networks (GATs) [36], and GraphSAGE [13], each with their unique aggregation and transformation functions.

GNNs have demonstrated their adaptability and effectiveness across a wide array of applications, including traffic prediction, text classification, and drug-target interaction forecasting.

Despite progress in GNN research, there are still open challenges and opportunities for future work, such as developing GNN models for directed graphs, improving computational efficiency and scalability for large graphs, and exploring novel GNN-based solutions for different applications. Addressing these challenges will further advance the field of GNNs and expand their applicability to a wider range of problems and domains.

2.2 Modular machine learning

The field of machine learning research has witnessed a significant trend towards developing increasingly larger models, driven by scaling laws [18] that suggest that more parameters lead to improved performance if we have enough data and the right architecture. These models are trained from scratch through extensive engineering efforts. However, their size presents challenges in fine-tuning, making it a costly process. Despite their large size, these models still exhibit limitations in various domains and applications, for example hallucinations in LLMs.

Large models require extensive high quality datasets and complex tasks to prevent overfitting. Traditionally, scaling up the number of examples has been seen as a way to enhance model performance. However, recent findings suggest that a more targeted and curated approach to training data can yield better results [38].

Modularity offers another avenue for improvement. By modularizing models, it becomes possible to separate fundamental knowledge and reasoning abilities related to language, vision, and other domains from specific task capabilities. Modularity allows for the development of specialized modules that can be combined and integrated into larger models, providing a more flexible and adaptable framework.

Modular deep learning involves computation units implemented as autonomous, parameter-efficient modules, with information conditionally routed to a subset of modules, processed, and aggregated. This enables systematic generalization by separating computation from routing and updating modules locally. Modular neural architectures encourage positive transfer, enable compositionality, and facilitate zero-shot transfer to tasks with new combinations of learned skills or observed features. Modularity also enhances parameter and time efficiency, allowing modules to be added or removed on-the-fly and adjusting model capacity according to task complexity.

Modular deep learning consists of three main components: modules, routing functions, and aggregation functions. Modules are units of computation that can be updated locally and asynchronously. Routing functions control information flow and can be fixed, learned, hard learned, or soft learned.

Algorithm 1: Forward pass of a modular function

Input: example $\mathbf{x}$
$\alpha \leftarrow r(\mathbf{x})$; `// Routing`
for $i \in [1, N]$ **do**
  $h_i \leftarrow f(\mathbf{x}, \theta_\mathbf{i})$; `// Computation`
  $H \leftarrow H \cup h_i$
end
$\mathbf{y} \leftarrow g(\alpha, H)$; `// Aggregation`

Aggregation functions combine the outputs of active modules using attention mechanisms, or other strategies.

- *Fixed Routing*: This method involves making discrete routing decisions based on metadata, such as task identity, before training. Commonly used in function composition techniques like multi-task learning and adapters, fixed routing allows for the selection of different modules based on various aspects of the target setting, such as task and language in natural language processing (NLP) or robot and task in reinforcement learning (RL). This enables models to generalize to unseen scenarios.

- *Learned Routing*: Typically implemented using a Multi-Layer Perceptron (MLP), this method offers flexibility in selecting modules based on learned weights, allowing the model to adapt to different tasks. However, it introduces additional challenges, such as training instability, and module collapse, which can impact the performance and effectiveness of learned routing methods.
- *Hard Learned routing*: This approach activates a subset of modules using a binary decision for each module. As discrete decisions cannot be learned directly with gradient descent, methods learn hard routing via reinforcement learning, evolutionary algorithms, or stochastic reparametrization.
- *Soft Learned routing*: This method aggregates all modules according to continuous scores, using a weighted sum of the modules. Soft learned routing is often sub-optimal as it under-utilizes and under-specializes modules, but it is the only option available when there is no one-to-one mapping between task and corresponding skill

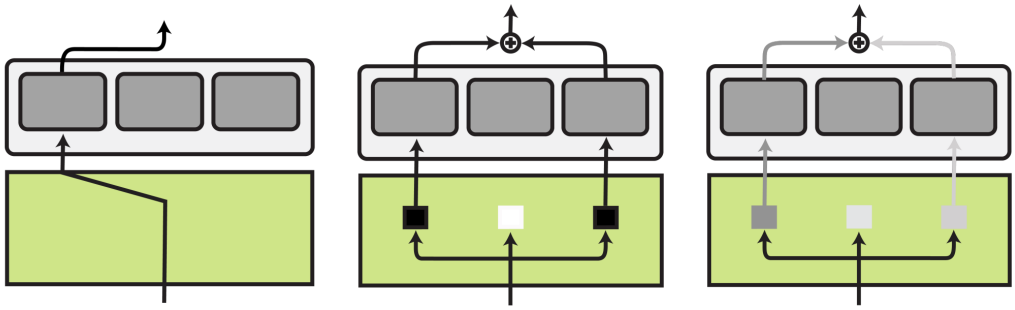


Figure 2.2: Different modular architectures. Image taken from [4]

Modular model architectures have gained popularity in machine learning due to their ability to combine and reuse smaller, independent modules to build complex models. This approach offers several benefits, including improved interpretability, flexibility, and scalability. In the context of deep learning, modularity allows for the separation of fundamental knowledge and reasoning abilities from domain- and task-specific components, leading to better generalization and performance. Modular models can also be more easily adapted to new tasks or domains by updating individual modules without affecting the rest of the network, making them a powerful tool for building flexible and efficient AI systems.

2.2.1 Mixture of Experts (MoE)

MoEs are based on the concept of conditional computation, where different parts of the network are activated on a per-example basis, allowing for a significant increase in model capacity without a corresponding increase in computation.

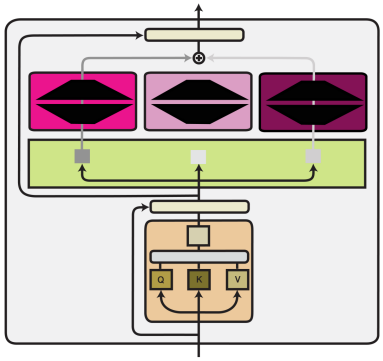


Figure 2.3: Mixture of Experts architecture. Image taken from [4]

In sparsely-activated MoE models, such as the Switch Transformer [6], GLaM [5], and V-MoE [26], a subset of experts is selected on a per-example basis, creating sparsity in the network.

MoE models have demonstrated better scaling in multiple domains and improved retention capability in continual learning settings, such as Expert Gate [3]. However, an ineffective expert routing strategy can lead to under-trained experts, resulting in experts being under or over-specialized.

MoE models have been applied in various domains, including federated learning, multi-modal learning, and bot detection, showcasing their adaptability and effectiveness in handling diverse data and tasks. SOTE models such as GPT4 from OpenAI seems to utilize modularity in order to balance efficiency and performance.

2.2.2 Neural Production Systems (NPS)

NPS [9] claims to offer a new model architecture that is capable of dynamic and efficient learning of social behaviour, while being scalable. It offers to be a replacement of GNNs as well as including modularity.

Although graph neural networks (GNNs) are commonly used to model social interactions, they still face some challenges. One major issue is their inherent density, which does not always match the sparsity found in real-life social connections. Social interactions also tend to happen within smaller, independent groups.

NPS is a modular architecture where each input is updated using the selected module conditioned on a selected subset (context) of the input. These selection procedures are done using the Gumbel-Softmax.

Gumbel-Softmax

Gumbel-Softmax allows us sample from discrete distributions while preserving differentiability, allowing the model to be trained through gradient descent and back-propagation. It was discovered by Jang et al. [17], and has applications in deep learning, particularly when dealing with attention mechanisms. In the following lets assume that we have normalized attention values (using softmax) $\pi_i \rightarrow \sum_i \pi_i = 1$, $\pi_i \geq 0 \; \forall i \in N$

In neural networks, attention is often soft, meaning that it takes a weighted sum of elements rather than focusing on a single or a couple of most important element. Soft attention is easier to backpropagate through, but sometimes hard attention (focusing on a single or a few elements) is more intuitive for the problem at hand.

Gumbel-Softmax combines two techniques: the Gumbel-Max trick and the Softmax function. The Gumbel-Max trick allows for efficient sampling from a categorical distribution by adding a random variable (drawn from a Gumbel distribution) to the log of the probabilities and taking the argmax. This is very similar to the reparametrization trick from variational auto-encoders.

$$z = \text{one\_hot}\left(\arg\max_i [g_i + \log \pi_i]\right)$$

where g_i are i.i.d. sample from a Gumbel distribution.

This reparametrization trick avoids having to backpropagate through the stochastic node, making the process differentiable.

The second step is to replace the argmax with a softmax function allowing gradient flow and helping with end-to-end gradient-based optimization. The softmax function has a temperature parameter, which controls the degree of "hardness" of the softmax and therefore the attention.

$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau}{\sum_{j=1}^{k} \exp((\log(\pi_j) + g_j)/\tau}$$

When the temperature is set to 0, the distribution becomes identical to the categorical one, and the samples are perfectly discrete. As the temperature approaches infinity, both the expectation and the individual samples become uniform.

In practice, the temperature parameter is set to either to 0.5 of 1.0, or annealed between the two using some sort of scheduling to approximate the discrete sampling over time.

A variation of the Gumbel-Softmax trick, called Straight-Through Gumbel-Softmax, allows for hard attention while still estimating gradients during training and using them during inference.

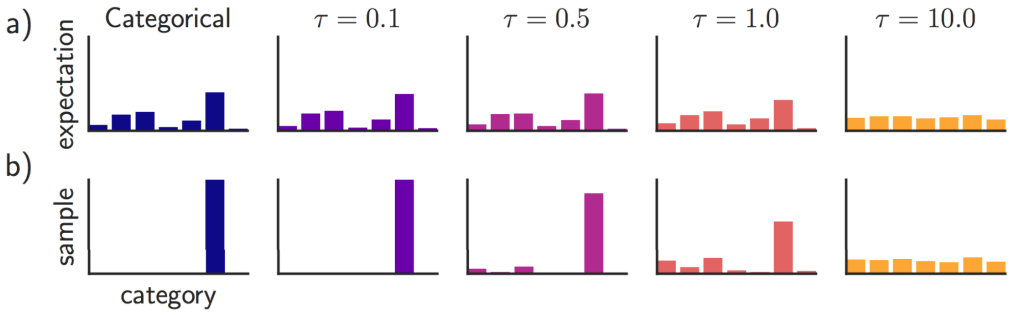


Figure 2.4: Gumbel Softmax τ parameter effect. Image taken from [17]

This is achieved by using the argmax version in the forward pass and the softmax version in the backward pass, approximating the true gradients.

NPS is motivated by the *production system* [15] architecture, that expresses knowledge by condition-action rules. These rules (smaller networks) are paired with entities, using rule conditions.

Production system

A production system consists of a finite set of entities and rules, combined with a rule- and entity selection mechanism. A rule entails details regarding the properties of relevant entities. For example, in case of trajectory prediction, it could separate input entities based on their speed.

The control flow of a production system chooses both the rules and the corresponding bindings between rules and entities, location, or any other feature extracted from the input data. Compared to traditional production systems, the NPS-based model we propose exhibits several shared attributes.

- *Production rules are modular*: they are independent, therefore can be combined to model more complex behaviours using simpler sub-models
- *Production rules are sparse*: only a small subset of entities are matched to each rule
- *Production rules are abstract*: they represent high-level knowledge, therefore capable of extrapolation, allowing the transfer of knowledge across different fields of application

Neural Production System

NPS offers a structural foundation that facilitates the identification and interpretation of object representations in an incoming stream. Additionally, it models guidelines that manage the relationships between objects across time and space.

NPS is made of N rules, $\{\boldsymbol{R}_1, \ldots, \boldsymbol{R}_N\}$. Each rule is make up by two parts, an encoding vector that is used to pair the rule with a set of input entities $\vec{\boldsymbol{R}}$, and the actual network / module that can be a feed-forward network or any other type of neural network based on the task at hand. Furthermore, the NPS contains learnable parameters to calculate the attention embeddings for:

- entities to be changed
- contextual entities to condition this changes on
- rules to make the changes with

For a sequential input $\{x_1, \ldots, x_T\}$, each time-step is encoded into a pre-defined number of *slots* using some sort of encoder. In case of trajectory prediction, this encoding step is skipped and the values are input raw to the network. The slots are made from the coordinates (or velocities) without any modification or computation.

When the slots are created, we select a subset of slots called *primary slots*, for which we perform the following steps:

- *Step 1*: for each primary slot $\boldsymbol{V}_p$, we select a rule $\boldsymbol{R}_p$ to be applied later on.
- *Step 2*: for each primary slot $\boldsymbol{V}_p$, we select a contextual slot $\boldsymbol{V}_c$ to condition the rule application, and therefore represent the social aspect of the model. This is the main difference compared to GNNs, as here the model only considers a dynamically selected single contextual slot to be considered at the rule application.
- *Step 3*: for each primary slot $\boldsymbol{V}_p$, apply the selected rule $\boldsymbol{R}_p$ conditioned on the selected contextual slot $\boldsymbol{V}_c$.

NPS has two versions:

- *Sequential NPS*: A single primary slot is selected at each step, therefore we can control how many slot to update at each time step using a certain number of rule-application steps.
- *Parallel NPS (see Alg.6 in Appendix A)*: All slots are treated as primary slots. In order to allow the model not to change certain slots, a *Null-Rule* is introduced. If this is selected for a slot, then no contextual-slot selected nor any sort of rule-application is done.

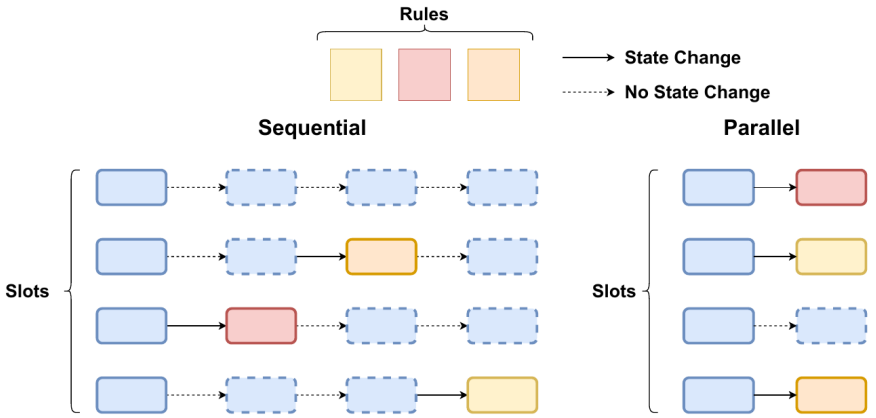


Figure 2.5: Neural Production System sequential and parallel rule application. Image taken from [9]

NPS introduces a scalable (modular), general, and efficient architecture with inductive biases towards sparsity and factorization of knowledge.

In experiments, NPS is shown to learn intuitive rules when applied on simple datasets such as MNIST. Experiments also compare parallel (PNPS) and sequential (SNPS) rule application showcasing when can one or the other be more suitable. This can depend on the application, or the level of sparsity we want to introduce (SNPS updating single slots makes it sparser). Overall, the experiments demonstrate NPS can learn interpretable rules, and the ability to model sparse interactions provides advantages over baselines like graph neural networks. The sparse interactions also enable better generalization as shown on the transfer tasks.

2.3 Trajectory prediction

2.3.1 Baselines

Linear

In order to establish a fundamental baseline, we can consider the most recent position and the calculated velocity derived from the current and previous positions for each entity. By doing so, we make

the assumption that each entity will continue to follow its trajectory from the point (x_t, y_t), with a velocity vector given by $(x_t - x_{t-1}, y_t - y_{t-1})$. Therefore the next points will be given as:

$$\{x_t + n * (x_t - x_{t-1}),\ y_t + n * (y_t - y_{t-1})\} \text{ for } \forall n \in [1, N]$$

The approach, while exhibiting certain limitations, effectively demonstrates the intricate nature of movement dynamics present in the utilized datasets and underscores the significant influence of social factors in the learning process. If we do not beat this baseline by a decent margin, then the social dynamics are not really important for the dataset at hand.

It allows us to understand the complexity of the dataset we are working with as well as serve as a weak benchmark that all proposed models should surpass.

LSTM

Long Short-Term Memory (LSTM) [16] models have gained significant attention in the field of sequential learning research. LSTM is a special type of Recurrent Neural Network (RNN) [39] that addresses the vanishing gradient problem faced by traditional RNNs.

The basic Recurrent Neural Network (RNN) 2.6 is designed to handle sequential input by utilizing recurrent connections. These recurrent connections allow the RNN to process each element of a sequence while taking into account the previous computations.

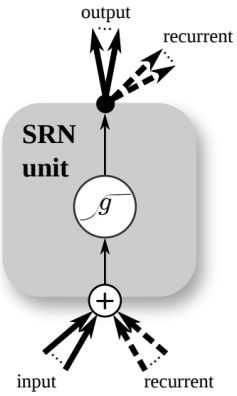


Figure 2.6: RNN. Image taken from [10]

LSTM was designed to overcome the limitations of RNNs in remembering long-term dependencies. While RNNs can remember previous information and use it for processing the current input, they struggle with retaining long-term dependencies due to the vanishing gradient problem. LSTMs are explicitly designed to avoid this issue and allow information to persist over time.

The architecture 2.7 of an LSTM network consists of three new parts: the forget gate, the input gate, and the output gate. These gates control the flow of information in and out of the memory cell or LSTM cell. The forget gate determines whether the information from the previous timestamp should be remembered or forgotten. The input gate quantifies the importance of new information, and the output gate produces the output based on the current hidden state and the long-term memory.

LSTM models have found applications in various domains, including natural language processing, speech recognition, image captioning, handwriting recognition, and time series forecasting. They have proven to be effective in tasks that involve sequential data and long-term dependencies.

Long Short-Term Memory (LSTM) serves as an appropriate baseline model for trajectory prediction tasks, primarily due to its general-purpose architecture. This design choice ensures that the model does not incorporate any problem-specific inductive biases, allowing it to remain neutral with respect to social and spatial aspects of the problem under investigation.

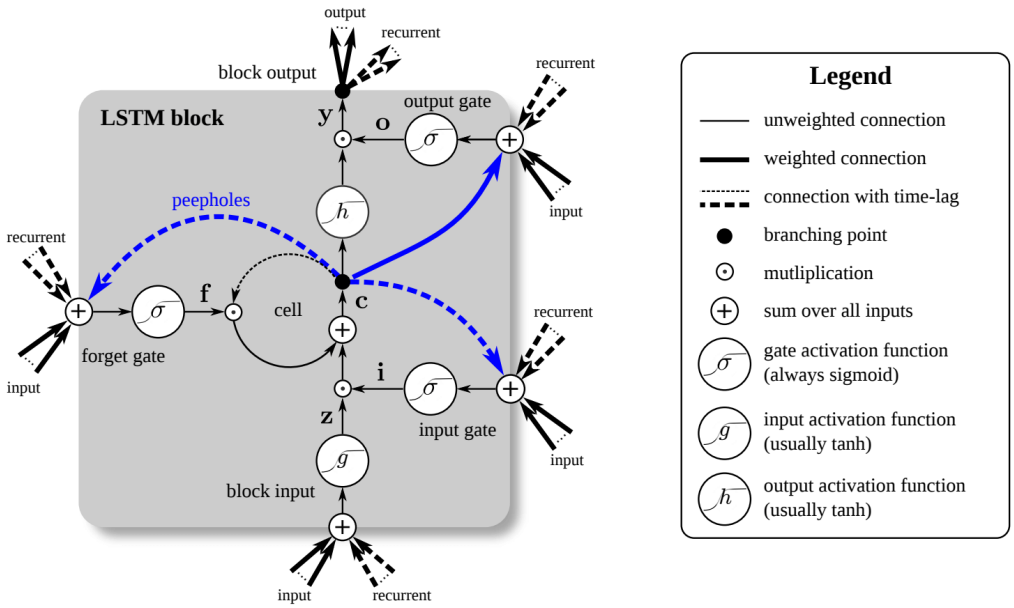


Figure 2.7: LSTM. Image taken from [10]

2.3.2 Social-LSTM

Social-LSTM [2] was a groundbreaking architecture that addressed the limitations found in earlier works on trajectory prediction. It introduced innovative techniques to capture complex interactions and anticipate potential interactions that could occur in the more distant future.

The Social-LSTM architecture is a significant departure from previous methods, which primarily relied on hand-crafted functions to model interactions. These earlier approaches were limited in their ability to capture complex interactions in crowded settings.

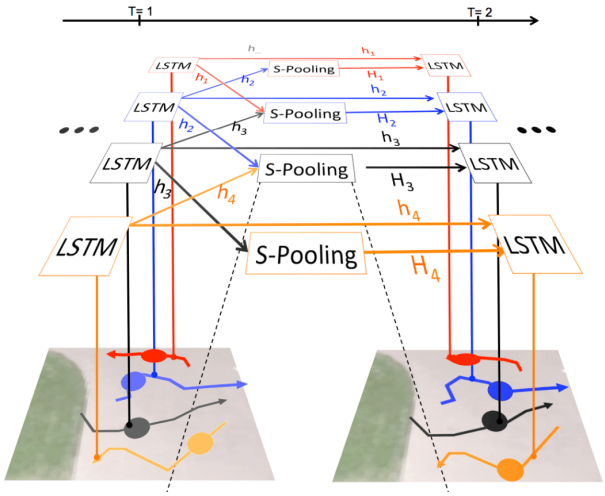


Figure 2.8: Social LSTM. Image taken from [2]

The crucial component of the Social-LSTM architecture is the social pooling layer. This innovative layer enables the LSTMs of spatially proximal sequences to share their hidden states with each other, fostering a more accurate representation of the relationships between nearby trajectories. By incorporating the social pooling layer, the Social-LSTM model can better understand and predict the intricate dynamics that arise from the interactions of multiple entities in a shared environment.

The hidden state of each LSTM is used to predict the distribution of the trajectory position at the next time-step $t + 1$. A bi-variate Gaussian distribution is parametrized my the mean and standard deviation and correlation coefficient contained in a 5-dimensional vector $(\mu_x, \mu_y, \sigma_x, \sigma_y, p)$.

$$(\hat{x}, \hat{y})_t^i \sim \mathcal{N}\left((\mu_x, \mu_y)_t^i, (\sigma_x, \sigma_y)_t^i, p_t^i\right)$$

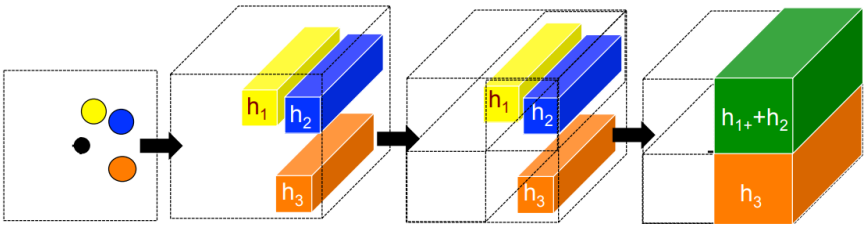


Figure 2.9: Shared hidden states grid. Image taken from [2]

In comparison to other baseline models, such as linear models [34], social force models [14], and LSTM models [16], the Social-LSTM model demonstrates improved performance in trajectory prediction. The model outperforms these baselines by considering complex interactions and leveraging the shared hidden states of nearby sequences.

2.3.3 Social-GAN

The Social-GAN model [11], short for Socially Acceptable Trajectories with Generative Adversarial Networks, combines tools from sequence prediction and generative adversarial networks to generate socially plausible future trajectories. The model consists of three key components: the Generator (G), Pooling Module (PM), and Discriminator (D) 2.10.

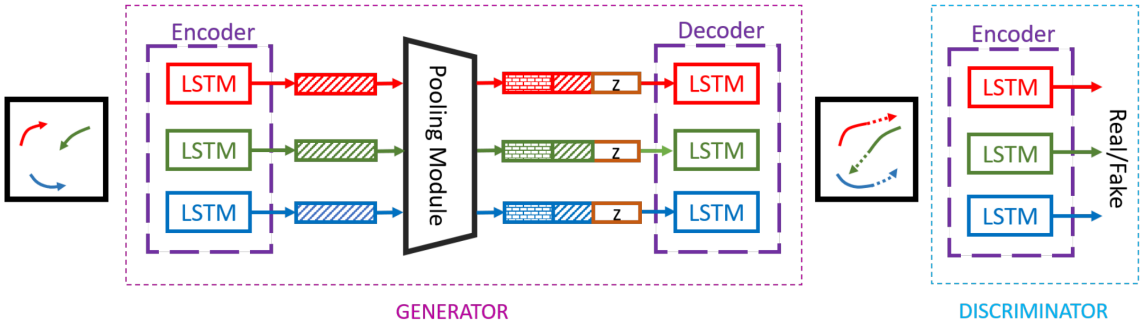


Figure 2.10: Social GAN. Image taken from [11]

The Generator takes past trajectories as input and uses an encoder-decoder framework to predict future behavior. It embeds the location of each entity using a single-layer MLP and feeds them into an LSTM cell for recurrent prediction.

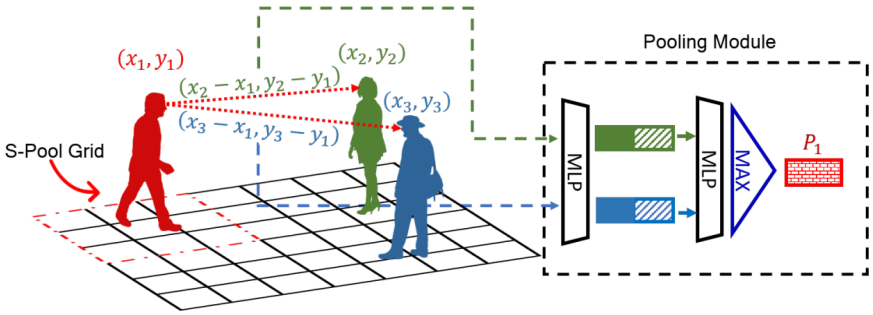


Figure 2.11: Social GAN Pooling scheme compared to Social-LSTM S-Pooling. Image taken from [11]

The Pooling Module aggregates information across people by computing relative positions and concatenating them with each entity's hidden state. This pooled vector captures the subtle cues for

all people involved in the scene.

The Discriminator evaluates the predicted trajectories and classifies them as "real" or "fake". The training procedure follows a minimax game, where the Generator aims to generate trajectories that can fool the Discriminator, and the Discriminator aims to distinguish between real and generated trajectories.

To encourage diversity in the predicted trajectories, the model introduces a variety loss.

$$\mathcal{L}_{\text{variety}} = \min_k \|Y_i - \hat{Y}_i^{(k)}\|$$

This loss, coupled with the pooling layer, encourages the network to produce globally coherent and socially compliant diverse samples.

Experiments on various datasets demonstrate that the Social-GAN model outperforms prior work in terms of accuracy, variety, collision avoidance, and computational complexity. It effectively captures the nature of human motion behavior and generates socially acceptable trajectories in crowded scenes.

2.3.4 Social-Ways

Social-Ways [12] uses a Generative Adversarial Network (GAN) to sample plausible predictions for any entity in the scene. The authors show that the recently proposed Info-GAN allows dramatic improvements in multi-modal pedestrian trajectory prediction to avoid mode collapsing and dropping. The paper also left out L2-loss in training the generator, unlike some previous works, because it causes serious mode collapsing though faster convergence. The authors show through experiments on real and synthetic data that the proposed method leads to generate more diverse samples and to preserve the modes of the predictive distribution.

The architecture adopts a new strategy to produce plausible samples for an entity from the joint predictive distribution of the set of entities.

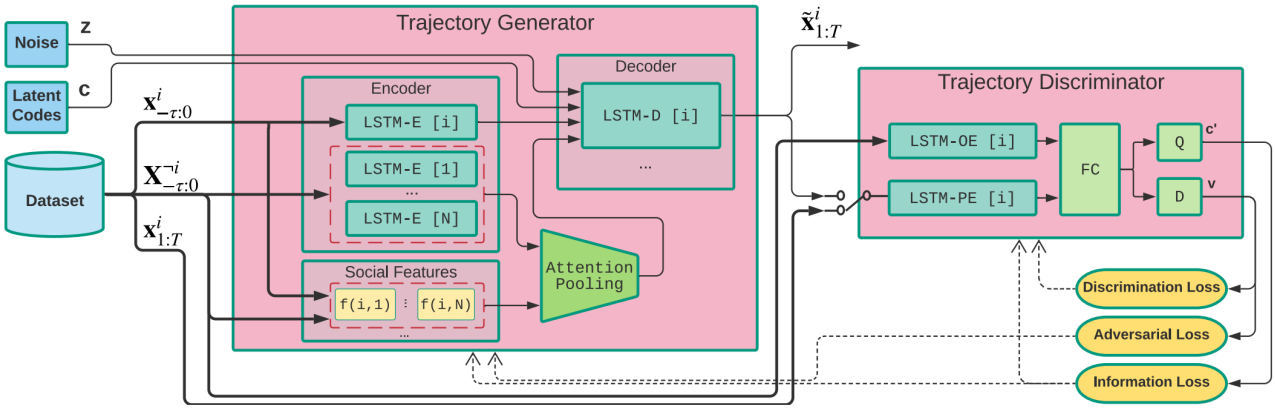


Figure 2.12: Social-Ways architecture. Image taken from [12]

The Sampler is trained to generate plausible predictions for a single entity, given past observations of trajectories for the whole set of the entities. The attention-based pooling scheme relies on a few hand-designed interaction features inspired from the neuroscience/bio-mechanics literature, as a form of prior; the best way to combine them to assess the interaction is learned by the system. The authors also designed a synthetic dataset specifically oriented to the evaluation of the preservation of multi-modality in trajectories predictive distributions.

Social-Ways was SOTA on many benchmarks, improving on previous methods such as the previously introduced Social-LSTM [2], Social-GAN [11] and SoPhie [32].

2.3.5 Spatial-Temporal Graph Attention Network (STGAT)

The proposed model is based on a sequence-to-sequence architecture and consists of three components: Encoder, Spatial-Temporal Graph Attention, and Decoder.

The Encoder component uses two LSTMs to model the motion pattern of each pedestrian. The Spatial-Temporal Graph Attention component captures the spatial interactions among pedestrians

at each time-step and uses an extra LSTM to encode the temporal correlations of interactions. The Decoder component predicts the future trajectories of pedestrians.

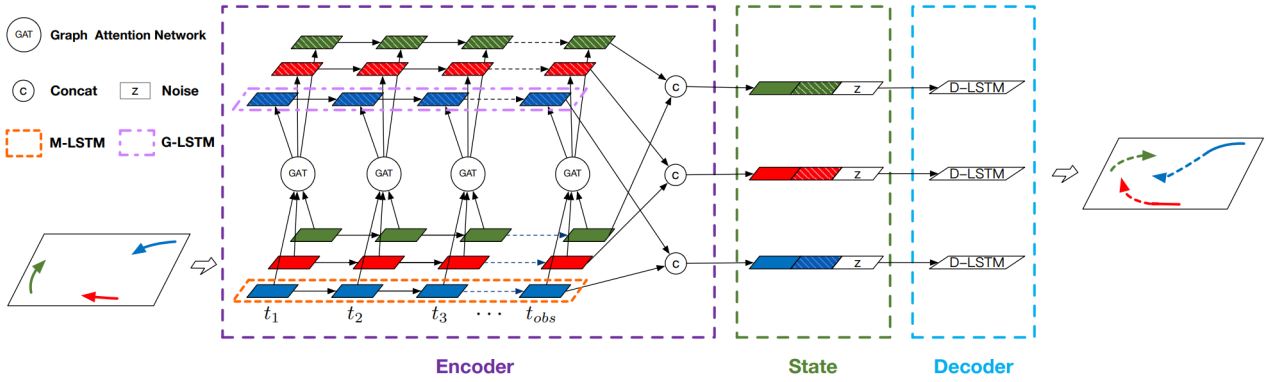


Figure 2.13: STGAT architecture. Image taken from [41]

Their model achieves superior performance and produces more "socially" plausible trajectories for pedestrians. The novelty of the proposed model lies in the combination of graph attention and LSTM to model the spatial and temporal interactions among pedestrians. The authors argue that their model can better capture the complex and diverse interactions among humans in crowded spaces.

2.3.6 DAG-Net

DAG-Net [24] employs a double attention-based graph neural network to collect information about the mutual influences among different entities and to integrate it with data about entities possible future objectives.

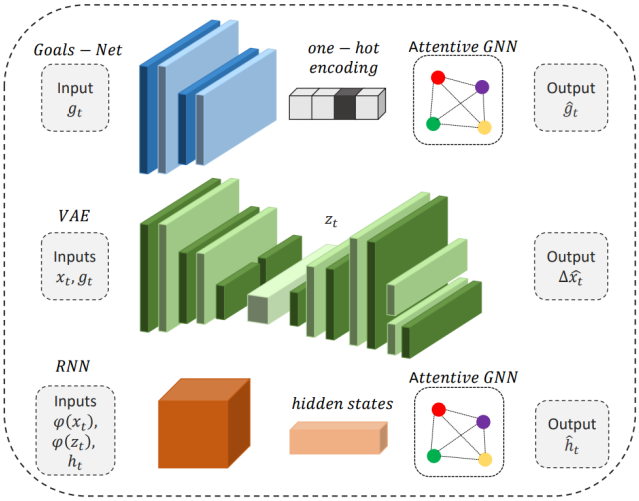


Figure 2.14: DAG-Net architecture. Image taken from [24]

The model treats goals as structured components by exploiting them as graph nodes and accordingly conditions the generation process on the resulting interrelated future objectives. Furthermore, DAG-Net uses an attentive module to associate importance weights to different interactive nodes.

DAG-Net outperformed all previous approaches by a big margin, creating a the new standard for trajectory prediction approaches.

2.3.7 Transformer-based architectures

Transformer networks have achieved significant success in Natural Language Processing (NLP) tasks. They showed that we can eliminate recurrence with the self-attention mechanism, which allows them to model temporal dependencies more effectively than Recurrent Neural Networks (RNNs). Transformers

use multi-head self-attention to jointly attend to different representations at different positions, improving temporal modeling. They also incorporate positional encoding to add positional information to the embeddings.

Although the computational complexity of these models is not optimal, it is very suitable for currently used computational units. This allows for parallelization and therefore to include higher order of complexity by scaling the models using more GPUs.

As transformer-based models have achieved SOTA results across many natural language processing tasks, researchers have begun incorporating them into every sequential learning application. However, it remains an open question whether these complex architectures are best suited for modeling social dynamics, or if their performance gains are simply due to their massive parameter count.

TransformerTF

TransformerTF [7] a multi-agent framework where each entity is modeled by an instance of their transformer network. Each network predicts the future motion of the entity based on their previous motion.

For each entity, the transformer network outputs the predicted future positions by processing their current and prior positions (observations or motion history). The coordinates are embedded using a linear mapping and aggregated with positional encoding. The model uses an encoder-decoder transformer architecture. The encoder processes the input sequence and generates a continuous representation of the sequence, while the decoder generates the output sequence based on the continuous representation.

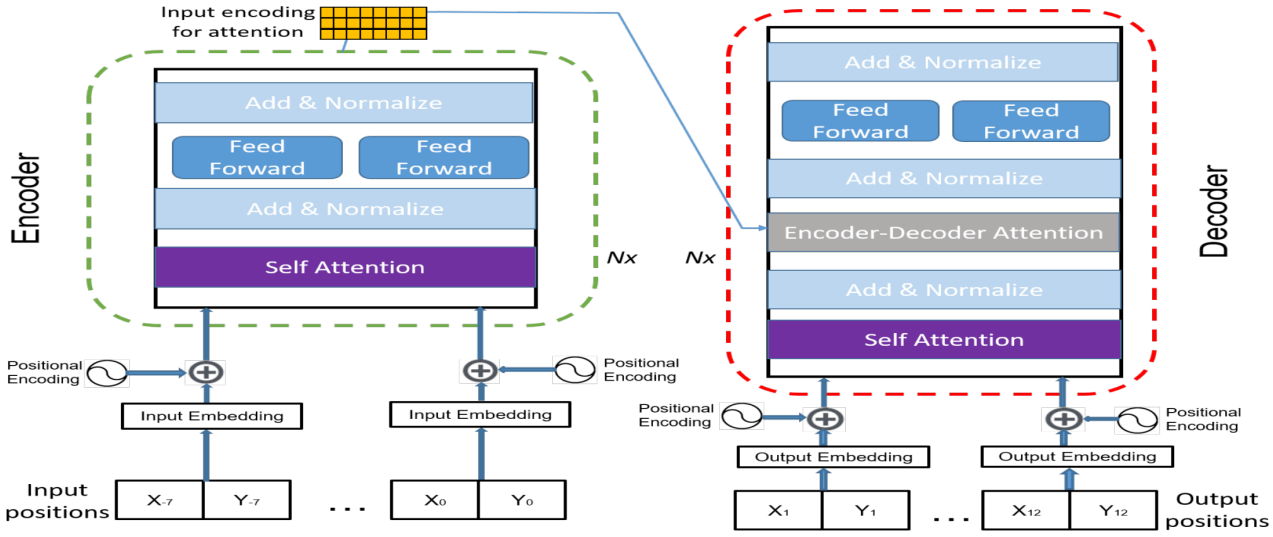


Figure 2.15: TransformerTF. Image taken from [7]

In addition to the original Transformer Network, the authors also consider the Bidirectional Encoder Representations from Transformers (BERT) model for trajectory forecasting. BERT is encoder-only model, meaning it only consists of the encoder part of the Transformer Network. BERT has been pre-trained on large amounts of text data and has achieved state-of-the-art performance on various natural language processing tasks. The BERT model is adapted for trajectory forecasting by fine-tuning the pre-trained model on the trajectory prediction task. The fine-tuning process involves training the model on the trajectory prediction task for a few epochs, allowing the model to adapt its learned representations to the new task.

The fine-tuned BERT achieved good results, but the problem-specific encoder-decoder produced became the new SOTA, and this is what we will use for comparison in the following.

STAR

Graph Neural Networks (GNNs) are powerful deep learning architectures for structured data. Graph convolutions [21] and Graph Attention Networks (GAT) [36] have demonstrated significant improvements in graph machine learning tasks. Temporal graph RNNs allow learning spatio-temporal relationships in graph sequences. STAR [42] builds upon GAT with TGConv, a Transformer-based graph convolution mechanism, to capture complex social interactions.

The STAR model consists of two main components:

- *Temporal Transformer*:

  The Temporal Transformer treats each pedestrian independently and extracts temporal dependencies using a Transformer model. It takes a set of pedestrian trajectory embeddings as input and outputs updated embeddings with temporal dependencies. The structure of a Temporal Transformer block is illustrated in 2.16.

  The Temporal Transformer block uses a self-attention mechanism to learn query, key, and value matrices for each pedestrian. These matrices are computed based on the embedded inputs. The self-attention mechanism allows the model to capture temporal dependencies over long time horizons. The updated embeddings are obtained by applying a fully connected layer with skip connections.

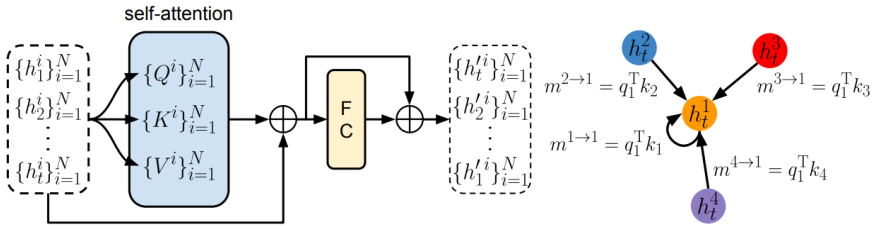
  The equations for the Temporal Transformer are as follows:

$$
\begin{aligned}
Q_i &= fQ(\mathbf{h}_i^{(t-1)}) \\
K_i &= fK(\mathbf{h}_i^{(t-1)}) \\
V_i &= fV(\mathbf{h}_i^{(t-1)})
\end{aligned}
$$

  where $\mathbf{h}_i^{(t-1)}$ represents the trajectory embedding of the i-th pedestrian at time step $t-1$, and fQ, fK, and fV are functions that compute the query, key, and value matrices, respectively.

- *Spatial Transformer*:

  The Spatial Transformer models the crowd as a graph and applies TGConv, a Transformer-based message passing graph convolution, to model social interactions. It treats the crowd as a graph, where each pedestrian is a node, and applies TGConv to capture the interactions between pedestrians. The message from node i to node j is represented by Transformer attention. The Spatial Transformer consists of a pair of spatial and temporal Transformers, which are stacked to extract spatio-temporal interactions. The structure of the Spatial Transformer is shown in 2.16.



(a) Temporal Transformer (b) Spatial Transformer

Figure 2.16: STAR. Image taken from [42]

The STAR model is a novel framework for spatio-temporal trajectory prediction based on attention mechanisms. It combines the power of Transformers with graph convolution to capture complex spatio-temporal interactions in crowd motion dynamics. The model achieves state-of-the-art performance on commonly used real-world pedestrian prediction datasets. This model motivated the research community to explore more transformer-based approaches.

AgentFormer

Simultaneously models the time and social dimensions of multi-agent trajectories. The novelty of AgentFormer [25] lies in its ability to model both dimensions together, rather than separately, as most prior methods do. This is achieved by flattening trajectory features across time and agents, and then using a novel agent-aware attention mechanism that preserves agent identities by attending to elements of the same agent differently than elements of other agents.

The architecture of AgentFormer consists of a time encoder that appends a timestamp feature to each element in the sequence, and an agent-aware attention mechanism that generates two sets of keys and queries via different linear transformations. One set is used for inter-agent attention (agent to agent), while the other set is designated for inter-agent attention (agent to itself).

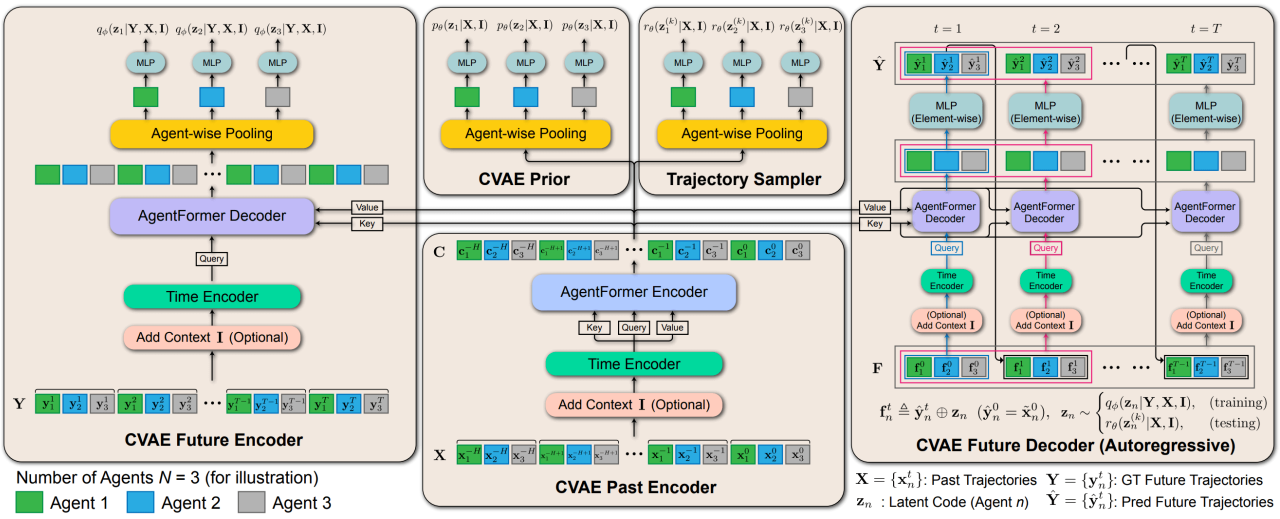


Figure 2.17: AgentFormer. Image taken from [25]

The authors argue that modeling the time and social dimensions separately can be sub-optimal since the independent feature encoding over either the time or social dimension is not informed by features across the other dimension, and the encoded features may not contain the necessary information for modeling the other dimension. To tackle this problem, AgentFormer allows an agent's state at one time to affect another agent's state at a future time directly instead of through intermediate features encoded over one dimension. However, directly applying standard Transformers to these multi-agent sequences will result in a loss of time and agent information since standard attention operations discard the time-step and agent identity associated with each element in the sequence. The authors solve the loss of time information using a time encoder that appends a timestamp feature to each element.

The loss of agent identity is a more complicated problem: unlike time, there is no innate ordering between agents, and assigning an agent index-based encoding will break the required permutation invariance of agents and create artificial dependencies on agent indices in the model. Instead, the authors propose a novel agent-aware attention mechanism to preserve agent information.

Agent-aware attention generates two sets of keys and queries via different linear transformations; one set of keys and queries is used to compute inter-agent attention (agent to agent) while the other set is designated for inter-agent attention (agent to itself). This design allows agent-aware attention to attend to elements of the same agent differently than elements of other agents, therefore keeping the notion of agent identity. Agent-aware attention can be implemented efficiently via masked operations. Furthermore, AgentFormer can also encode rule-based connectivity between agents (e.g., based on distance) by masking out the attention weights between unconnected agents.

The probabilistic formulation of the model follows the conditional variational auto-encoder (CVAE) where the generative future trajectory distribution is conditioned on context (e.g., past trajectories, semantic maps). The authors introduce a latent code for each agent to represent its latent intent.

To model the social influence of each agent's future behavior (governed by latent intent) on other agents, the latent codes of all agents are jointly inferred from the future trajectories of all agents during

training, and they are also jointly used by a trajectory decoder to output socially-aware multi-agent future trajectories.

To improve the diversity of sampled trajectories and avoid similar samples caused by random sampling, the authors further adopt a multi-agent trajectory sampler that can generate diverse and plausible multi-agent trajectories by mapping context to various configurations of all agents' latent codes.

2.3.8 Recent state-of-the-art approaches

ExpertTraj

Novel approach [43] to predict pedestrian trajectories using goal expertise obtained through a goal-search mechanism on training examples. The authors make three key contributions:

- They develop a framework that exploits nearest examples for high-quality goal position inquiry. This approach naturally considers multi-modality, physical constraints, compatibility with existing methods, and is non-parametric; it therefore does not require additional learning effort typical in goal inference.
- They present an end-to-end trajectory predictor that can efficiently associate goal retrievals to past motion information and dynamically infer possible future trajectories.
- They conduct experiments on two widely explored datasets (SDD and ETH/UCY) and show that their approach surpasses previous state-of-the-art performance by notable margins and reduces the need for additional parameters.

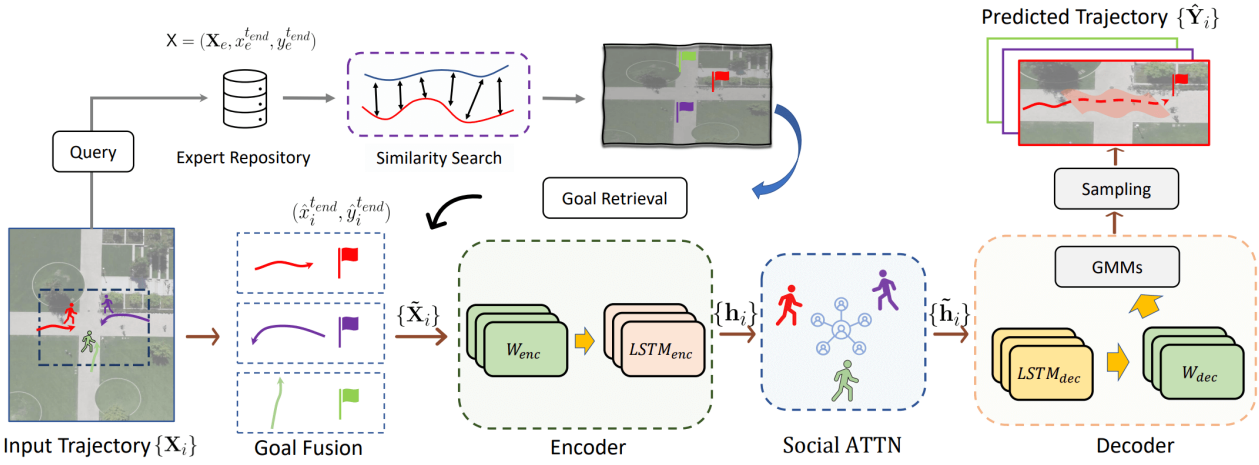


Figure 2.18: ExpertTraj. Image taken from [43]

The proposed architecture consists of two main components: a goal-retrieval algorithm and a trajectory predictor. The goal-retrieval algorithm performs similarity search between partially observed trajectories from a test set and expert examples from a training set, obtaining a small, multi-modal set of candidate goal positions. The trajectory predictor takes the history of trajectory observations and the queried goal results with a novel low overhead data-shift encoding to jointly infer a diverse, yet accurate set of future trajectories.

This approach is the first to explore a non-parametric method for goal inference and demonstrates state-of-the-art performance in pedestrian trajectory prediction. The goal-retrieval algorithm is inspired by techniques seen in expert learning and data-efficient machine learning. It leverages the power of recent advances in data-efficient machine learning, where unlabeled data are self-annotated via metric matching on nearest labeled neighbors.

The trajectory predictor incorporates past observations and queried goal positions to infer diverse and accurate predictions. The authors use a novel data-shift encoding method to incorporate goal

information into feature embedding with zero extra effort. The trajectory prediction is done using a seq2seq generator implemented as two Long Short-Term Memory (LSTM) networks, one for encoding and one for decoding.

SGNet

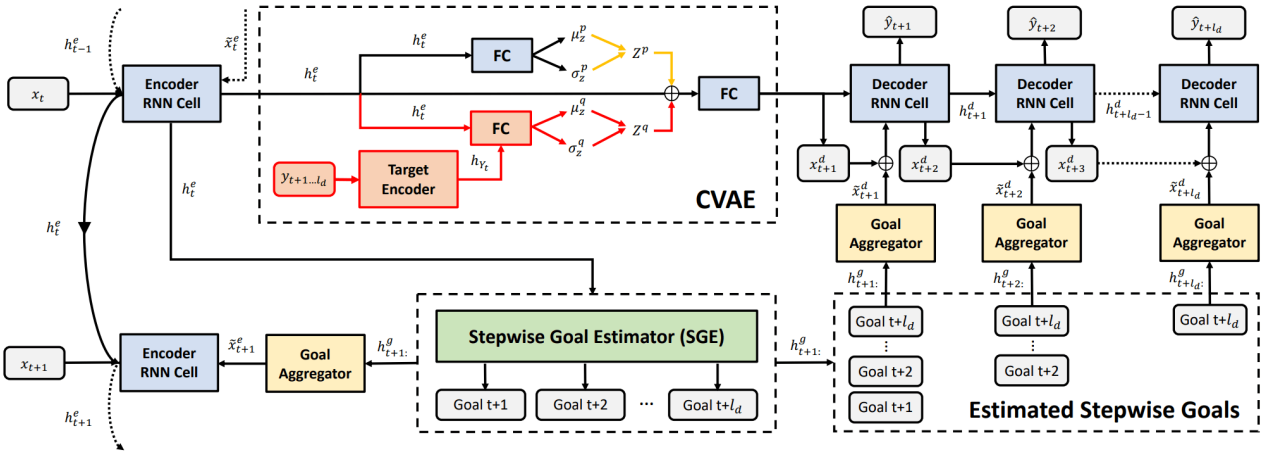


Figure 2.19: SGNet architecture. Arrows in red, yellow, and black indicate connections during training, inference, and both training and inference, respectively. Encoder time evolves vertically, from time t to $t+1$, while decoder time flows horizontally, predicting the trajectory at $t+1$ to $t+l_d$. For deterministic results, they replace CVAE with a non-linear embedding. Image taken from [23]

Predicts the future trajectories of observed entities, such as pedestrians or vehicles, by estimating and using their goals at multiple time scales. The authors argue that the goal of a moving entity may change over time, and modeling goals continuously provides more accurate and detailed information for future trajectory estimation.

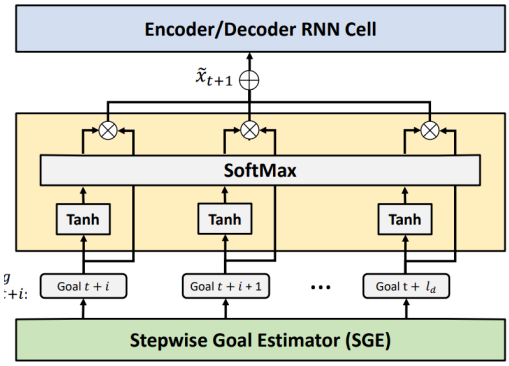


Figure 2.20: SGNet Goal Aggregator. It receives a set of step-wise goals as inputs and calculates the attention weights. Image taken from [23]

The proposed model, called Stepwise Goal-Driven Network (SGNet) [23], estimates and uses goals at multiple temporal scales, unlike prior work that models only a single, long-term goal. SGNet consists of three main components:

- An encoder that captures historical information.
- A step-wise goal estimator that predicts successive goals into the future.
- A decoder that predicts future trajectory.

The step-wise goal estimator (SGE) predicts coarse future goals at multiple temporal scales to encode a comprehensive representation of the intention. To determine the significance of each step-wise goal, a lightweight module with an attention mechanism is used.

The encoder records past data in conjunction with predicted step-wise goals, incorporating a richer hidden representation that aids in predicting the future and creating new step-wise goals for the next time step. The decoder takes advantage of step-wise goals to predict future trajectories.

The authors evaluate their model on multiple first- and third-person datasets, including both vehicles and pedestrians, and compare it to an extensive range of existing work ranging from deterministic to stochastic approaches.

SocialVAE

SocialVAE [40] utilizes a time-wise variational auto-encoder (VAE) architecture combined with a social attention mechanism and a backward posterior approximation to improve the extraction of pedestrian navigation strategies. SocialVAE consists of four key components:

- *Time-wisely generated latent variables (TL)*:

  The VAE architecture incorporates stochastic recurrent neural networks (RNNs) to generate latent variables that condition the hidden dynamics of the RNNs at each time step.

- *Backward posterior approximation (BP)*:

  A backward RNN structure is used for posterior approximation, taking into account the entire trajectory to robustly extract navigation patterns.

- *Neighborhood attention using social features (ATT)*:

  An attention mechanism is employed to encode the states of neighboring entities, considering their observed social features.

- *Optional final position clustering (FPC)*:

  This post-processing technique helps reduce sampling bias and improves prediction quality by clustering final positions.

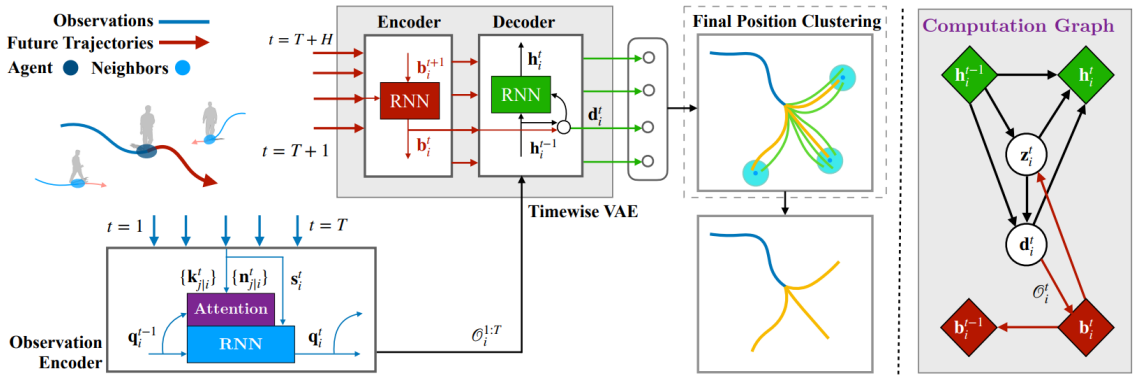


Figure 2.21: SocialVAE. Image taken from [40]

The SocialVAE approach has demonstrated significant improvements over existing trajectory prediction methods. It achieves state-of-the-art performance on several pedestrian trajectory prediction benchmarks, including the ETH/UCY benchmark, Stanford Drone Dataset, and SportVU NBA movement dataset.

The ablation studies conducted on SocialVAE highlight the contributions of its key components. Using the TL scheme or the BP formulation alone reduces the Average Displacement Error (ADE) and Final Displacement Error (FDE) values. The combination of TL and BP leads to an average

improvement of 26% on ADE and 11% on FDE. Adding the ATT mechanism further reduces the prediction error. The application of FPC enhances prediction diversity and leads to a considerable decrease in FDE.

In conclusion, SocialVAE presents a novel approach for human trajectory prediction that effectively captures the uncertainty and multi-modality of human navigation behaviors. It outperforms existing methods on various benchmark datasets.

2.3.9 Social-STGCNN

The complexity of pedestrian trajectory prediction arises from the interactions between pedestrians and their environment, as well as social behaviors exhibited by pedestrians. Previous methods have used recurrent architectures and aggregation mechanisms to model these interactions, but they suffer from limitations in parameter efficiency and modeling accuracy.

The Social Spatio-Temporal Graph Convolutional Neural Network (Social-STGCNN) [28] is a novel approach that overcomes these limitations. It models pedestrian trajectories as a spatio-temporal graph, where the edges represent social interactions between pedestrians. The model utilizes graph convolutional neural networks (CNNs) and temporal CNNs to manipulate the spatio-temporal graph and predict the entire trajectory sequence in a single shot.

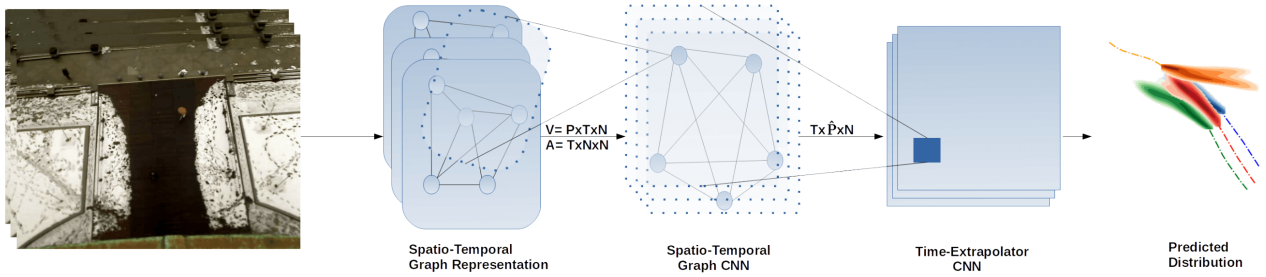


Figure 2.22: Social STGCNN. Image taken from [28]

In Social-STGCNN, the pedestrians' trajectories are represented as a graph from the start, eliminating the need for aggregation layers. The graph edges capture the social interactions between pedestrians. A weighted adjacency matrix is used to quantify the influence between pedestrians, with a kernel function measuring the interaction strength.

To address the limitations of recurrent architectures, Social-STGCNN employs temporal CNNs for modeling over the spatio-temporal graph. This allows for efficient prediction of the entire trajectory sequence. The model outperforms previous methods in terms of prediction accuracy, parameter size, inference speed, and data efficiency.

The Social-STGCNN model demonstrates significant improvements over the state-of-the-art methods. It achieves a 20% improvement in Final Displacement Error (FDE) and up to 48 times faster inference speed, while using 8.5 times fewer parameters. Additionally, the model shows data efficiency by achieving superior performance with only 20% of the training data.

The Social-STGCNN model presents a novel approach to human trajectory prediction by utilizing a social spatio-temporal graph convolutional neural network. It overcomes the limitations of recurrent architectures and aggregation mechanisms, achieving superior performance in terms of accuracy, efficiency, and data utilization. The model's ability to capture social behaviors and interactions between pedestrians makes it a valuable tool for applications such as autonomous driving and surveillance systems.

2.3.10 Social-Implicit

In the field of trajectory prediction, the Social-Implicit [1] model has emerged as a memory-efficient deep learning model that offers competitive results with a relatively small number of parameters. This model has been specifically designed to address the challenges associated with trajectory prediction evaluation.

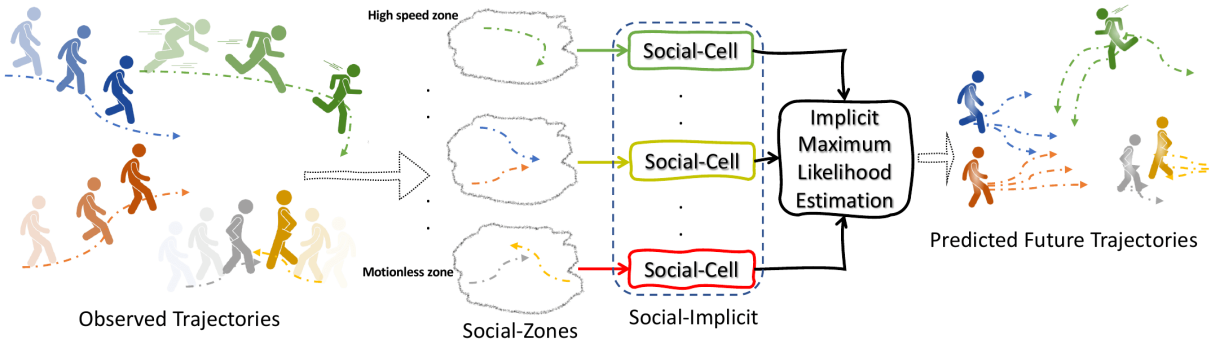


Figure 2.23: Social-Implicit overview. Image taken from [1]

Traditional trajectory prediction models often rely on explicit features, such as velocity and acceleration, to make predictions. However, these models may not fully capture the complex social interactions and implicit cues that influence human behavior.

The Social-Implicit model takes a different approach by incorporating implicit social cues and context into the trajectory prediction process. It leverages deep learning techniques to learn and model the underlying social dynamics in a data-driven manner. With only 5.8K parameters, the Social-Implicit model achieves competitive results while being memory-efficient and capable of real-time execution.

The Social-Implicit model offers several key features and advantages that set it apart from traditional trajectory prediction models:

- *Memory Efficiency*:
  With a small number of parameters, the Social-Implicit model is memory-efficient, making it suitable for resource-constrained environments.

- *Real-Time Execution*:
  The model is designed to run in real-time, with a processing speed of approximately 580Hz. This enables timely trajectory predictions, which are crucial for applications such as autonomous vehicles.

- *Competitive Results*:
  Despite its simplicity and efficiency, the Social-Implicit model achieves competitive results in trajectory prediction tasks. It has been evaluated and compared against existing state-of-the-art models, demonstrating its effectiveness.

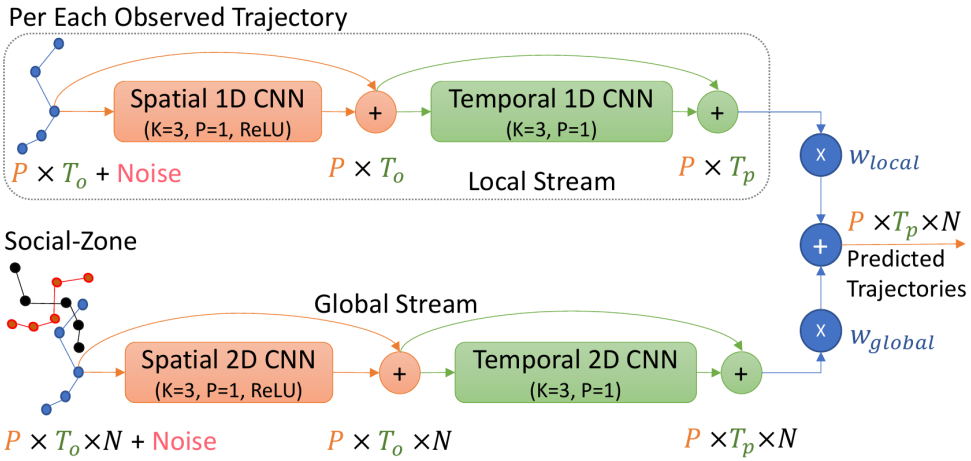


Figure 2.24: Social-Implicit model. Image taken from [1]

Overall, the Social-Implicit model offers a fresh perspective on trajectory prediction evaluation and highlights the importance of considering implicit social cues in understanding and predicting human behavior.

3 Methodology

In this study, we present a novel approach that seamlessly integrates the Neural Production System (NPS) within the Social-Implicit model, with the primary objective of enhancing trajectory prediction capabilities in the realm of modular machine learning. By incorporating the NPS framework, our method aims to introduce sparsity into an already high-performing model, consequently increasing its generality and efficiency.

The integration of NPS into the Social-Implicit model is designed to improve the overall performance of the system while requiring minimal modifications to the existing architecture. This ensures that the benefits of the original model are preserved, while simultaneously leveraging the advantages offered by the NPS framework.

In addition to the integration of NPS, we also explore various modifications to the NPS model itself, specifically focusing on the rule- and context-selection procedures. The author's of NPS handle these selections completely independently, while we believe that there are better alternatives that can offer improved performance without the loss of sparsity and factorization of knowledge. By examining these aspects, we aim to gain a deeper understanding of the underlying mechanisms that govern the model's behavior and identify potential areas for further optimization.

To evaluate the effectiveness of our proposed approach, we employ trajectory prediction as a benchmark application. This allows us to measure the impact of the modifications made to the NPS model, as well as the overall performance of the integrated system. Through rigorous experimentation and analysis, we hope to demonstrate the potential of our approach in advancing the field of modular machine learning and contributing to the development of more efficient, general, and robust models for trajectory prediction and other applications where scalability and sparse social interactions are existent.

To clarify, we're using Social-Implicit models instead of MLPs in the rule heads of NPS for our trajectory prediction model. While Social-Implicit has already proven to be effective in this domain, it's worth noting that any other model could be used in its place. Our focus is not on analyzing the capabilities of Social-Implicit or addressing its limitations, such as its training instability which introduces noise into our experiments. Nonetheless, we believe that our work will provide valuable insights into the capabilities of NPS.

3.1 Input data

As previously stated, our methodology exclusively depends on the utilization of extracted coordinate pairs, excluding any image information from the learning process. Although semantic maps and other types of additional input data can provide useful information to achieve good performance, our primary goal is to analyze the social dynamics and conduct controlled experiments measuring how far we can progress using only coordinate inputs. While we acknowledge prior works demonstrating the benefits of incorporating image data in the model architecture, we aim to first thoroughly investigate the social dynamics encoded in raw coordinates before integrating other data sources. Once this foundation is established, we can then augment our model with semantic maps and other supplementary data sources to further enhance performance.

Coordinate pre-processing

Considering the potential for substantial variation in the scale of raw coordinates across different datasets, we implement well-established normalization techniques to effectively tackle this challenge.

In our research, we explore a diverse range of normalization techniques to determine the most effective approach for managing the variations in coordinate scales across different datasets.

- *Absolute coordinates*
  Absolute coordinates refer to the simplest method available, which involves using the raw coordinates from the datasets without any modification. However, this approach may not be practical, as each scene has a different scale that is irrelevant to the optimization problem at hand. Consequently, the neural network might struggle to generate meaningful outcomes during testing when exposed to a new scene with a different origin compared to the training set.

- *Batch normalization*
  Batch normalization involves normalizing each input batch using the mean and standard deviation computed along entities and all past time-steps. This transformation is applied during both training and evaluation to ensure that the input data has a similar scale for each input.

- *Re-scale to origin at last input*
  By subtracting the last known position of each entity from all past coordinate-pairs of that entity, we obtain a natural and commonly used normalization technique that is computationally efficient and works well in most cases.

- *Re-scale to origin at first input*
  This method follows the same idea as the previous one but uses the first known position for normalization instead of the last input.

Given the lack of a universally agreed-upon best approach among these normalization techniques, we will provide a comprehensive analysis of the results obtained using each method in the subsequent sections. This comparative evaluation will offer valuable insights into the relative performance and effectiveness of the various normalization strategies in the context of our research.

Velocities

In the majority of scenarios, the model is fed computed velocities as input, as the primary objective is to make the model comprehend the evolving dynamics of the trajectories over time. It is important to note that velocities are calculated prior to normalizing the coordinates, as this computation can be regarded as a normalization technique in its own. Integrating it with another method could potentially yield exceedingly small values, leading to diminished performance in the learning process. The normalized coordinates will be only used during the context selection part of the models, as the absolute distance between trajectories the influence they have on each other, but afterwards the velocities contain all the information we need to determine their future joint behaviour. Augmentation techniques are applied prior to normalization.

3.2 Data augmentation

We adopt the same data augmentation strategy employed during the training of the Social-Implicit model to address data imbalance issues within the training sets. This approach includes:

- *Random rotation*
  Rotate the entire scene by several degrees to improve generalization across different orientations.

- *Flip x, y locations*
  A simple technique that generates new, socially acceptable trajectories by flipping the x and y coordinates.

- *Jitter coordinates*
  Add small noise to the locations to enhance the model's robustness during learning.

- *Increase the number of the nodes in the scene*
  Combine multiple scenes to increase the complexity of the task at hand.

- *Changing the speed of pedestrians*
  Adjust the velocity to address the scale issue in the training set.

- *Reverse trajectory*
  Another straightforward method for generating new, realistic data points for learning.

- *Identity*
  No changes to the data.

For each input scene, we randomly select one of these augmentation techniques to be applied. To ensure that the generated data remains realistic and valuable for learning purposes, we refrain from combining multiple augmentations simultaneously. This cautious approach helps maintain the integrity and relevance of the augmented data for our model.

3.3 Implicit Maximum Likelihood Estimation (IMLE)

In the field of trajectory prediction, a variety of evaluation metrics have been proposed in the literature, with many of them relying on stochastic models. These stochasticity-based metrics typically involve sampling 20 predictions for each input and selecting the best predicted trajectory as a measure of the overall prediction quality. Given the diverse range of approaches in the literature, including both stochastic and deterministic methods, it is crucial to establish a means of comparing these different techniques.

The Social-Implicit model, for instance, is deterministic by nature. To make it compatible with stochastic evaluation metrics, the authors opted to employ Implicit Maximum Likelihood Estimation (IMLE). Several alternative solutions have been proposed to address this issue, such as having the model predict the parameters of a multivariate Gaussian distribution instead of the actual trajectory, or replicating the last layer of the model n times to generate n distinct outcomes rather than a single one.

IMLE offers a versatile and adaptable solution, as it can be applied to any deterministic model without requiring modifications to the model architecture. By introducing n noisy inputs to the model, IMLE only extends both the training and inference time, although this increase can be mitigated through parallelization since the noised inputs are independent of one another. To generate the noise, values from a standard normal distribution are sampled and subsequently multiplied by a learned scalar weight, ensuring that the noise is appropriately scaled to match the specific dataset being used.

Algorithm 2: Implicit Maximum Likelihood Estimation for Trajectory Prediction

Input: input trajectory $X = (x_1, \ldots, x_n)$,
noise distribution σ,
deterministic model Θ
learnable parameter θ

for $e \in [0, \mathit{Epochs}]$ **do**

***step* 1**: draw n random samples from σ

- $\hat{X}_i = (x_1 + \sigma_i, \ldots, x_n + \sigma_i)$
- multiply $\hat{X}_i$ by θ to scale the noised input

***step* 2**: forward each $\hat{X}_i$ to the model for prediction
***step* 3**: calculate BestOfN Loss given the stochastic predictions

end

3.4 Loss function

In order to ensure a fair comparison with the *Social-Implicit* paper, the same loss function, referred to as the *Social-Loss*, is employed. This loss function consists of four distinct components, each designed to address specific aspects of trajectory prediction.

- *IMLE LOSS*
  The Implicit Maximum Likelihood Estimation (IMLE) Loss is a standard BestOfN loss that measures the distance between the best prediction and the ground-truth trajectory. The L_1 norm is used to calculate this distance:

$$\mathcal{L}_{\text{IMLE}} = \|d_p - \tilde{d}_p^1\|_1$$

  Here, d_p represents the ground-truth, while $\tilde{d}_p^m$ denotes the m-th best prediction.

- *Triplet loss*
  To encourage all predictions to be close to the ground-truth, the *Triplet Loss* measures the variation of the predictions. This is achieved by calculating the difference between the distance of the best and second-best predictions, and the distance of the best and worst predictions:

$$\mathcal{L}_{\text{Triplet}} = \|\tilde{d}_p^1 - \tilde{d}_p^2\|_1 - \|\tilde{d}_p^1 - \tilde{d}_p^{-1}\|_1$$

- *Geometric distance*
  This component ensures that the distance between trajectory points in the predictions matches the distance in the ground-truth:

$$\mathcal{L}_{\text{Geom-dist}} = \frac{1}{\frac{T_p(T_p-1)}{2}} \sum_{t \in T_p} \sum_{j \in T_p, j>t} \|\|p_t - p_j\|_2 - \|\tilde{p}_t - \tilde{p}_j\|_2\|_1$$

- *Gemetric angle*
  Similarly, the *Geometric Angle* component ensures that the angles between trajectory points in the predictions match those in the ground-truth:

$$\mathcal{L}_{\text{Geom-angle}} = \frac{1}{\frac{T_p(T_p-1)}{2}} \sum_{t \in T_p} \sum_{j \in T_p, j>t} \|\angle(p_t, p_j) - \angle(\tilde{p}_t, \tilde{p}_j)\|_1$$

By combining these four components, the *Social-Loss* is defined, with each component weighted using a pre-defined hyperparameter:

$$\mathcal{L}_{\text{Social}} = \mathcal{L}_{\text{IMLE}} + \alpha_1 * \mathcal{L}_{\text{Triplet}} + \alpha_2 * \mathcal{L}_{\text{Geom-dist}} + \alpha_3 * \mathcal{L}_{\text{Geom-angle}}$$

The hyperparameters are set as follows: $\alpha_1 = 1e-4, \alpha_2 = 1e-4, \alpha_3 = 1e-5$.

This loss function is very important to transform our originally deterministic model into a stochastic one, as it makes sure that the predictions not only do well on the stochastic metrics but are suitable for real-world applications. It encourages the model to predict concise trajectories, and not only predict a huge area that likely includes a random prediction close to the ground truth.

3.5 Social-Implicit + Neural Production Systems

The Social-Implicit paper presents a novel model for trajectory prediction that shares several similarities with Neural Production Systems (NPS). In this model, cells are utilized, which can be likened to the rules found in NPS. However, the routing to these cells is hand-crafted and, as a result, highly data-specific.

We propose the use of Gumbel-Softmax to select the trajectories that will be forwarded into each cell or rule. This approach is more general, aligns better with the modular machine learning framework, and has significant potential to yield improved results.

Simultaneously, the Social-Implicit model processes only a subset of trajectories using each cell, but the social aspect of learning is dense within each individual cell. This is achieved through the use of 2D convolutional layers that take into account all routed trajectories along the spatial and temporal dimensions. We argue that this process is not only dense but also results in a loss of information due

to the exclusive consideration of similar velocity trajectories as influential. It is worth noting that the Social-Implicit model does not use absolute coordinates for predicting future velocities at all.

To address these limitations, we propose using Gumbel-Softmax once more to select context velocities based on past velocities and absolute coordinates. This approach should enable the model to handle more complex social scenarios. By default, we select only a single context trajectory for each input trajectory, rendering the 2D convolution irrelevant. Consequently, we modify the 2D convolution to a simpler 1D convolution by aggregating the primary and context trajectories along the spatial dimension. This change reduces the number of parameters in the model while allowing for more efficient social-dynamics modeling. This approach is suitable as we only consider a single context trajectory, otherwise the permutation invariance property known from GNNs would become more important.

In the context of trajectory prediction, the Parallel Neural Production System (NPS) [9] proves to be more suitable, as it enables simultaneous joint predictions. To optimize this version of NPS for the specific application, we introduce two key modifications.

First, we eliminate the Null-Rule, and consequently, the entire Step 2 of the algorithm. We chose to do this as every trajectory should receive a prediction, rendering the Null-Rule irrelevant. In case many entities have almost no change in their position in the training dataset, the model has the capacity to use a rule to predict future trajectories for these type of entities.

Second, we adjust the computation of keys and queries for the rule embeddings. Instead of using a weight matrix to generate keys and queries, we propose leveraging the rule-embeddings themselves, which are already learned parameters for this purpose. This approach not only reduces unnecessary complexity but also minimizes the total number of parameters without compromising the model's effectiveness. We are able to make this modification, as we define the rule-embedding to have the exact dimensionality that the attention requires it to be. In case we would embed the input trajectories, this change should be reverted.

Algorithm 3: Updated Parallel NPS

Input: sequential data $\{x^1, x^2, \dots, x^T\}$,
set of embeddings for rule-matching $\vec{\boldsymbol{R}}_i$,
set of rule-heads $\boldsymbol{R}_{1\dots N}$

for $t \in [1, T]$ **do**

***step* 1**: select a rule for each slot

- $\boldsymbol{k}_i = \vec{\boldsymbol{R}}_i \quad \forall i \in \{1, \dots, N\}$
- $\boldsymbol{q}_p = \boldsymbol{V}_p^t \boldsymbol{W}^q \quad \forall p \in \{1, \dots, M\}$
- $\boldsymbol{r}_p = \text{argmax}_i(\boldsymbol{q}_p \boldsymbol{k}_i + \gamma) \quad \forall i \in \{1, \dots, N\}\ \forall p \in \{1, \dots, M\}, \quad \text{where} \quad \gamma \sim \text{Gumbel}(0, 1)\}$

***step* 2**: select a contextual slot for each slot

- $\boldsymbol{k}_j = \boldsymbol{V}_j^t \tilde{\boldsymbol{W}}^k \quad \forall j \in \{1, \dots, M\}$
- $\boldsymbol{q}_p = \boldsymbol{V}_p^t \tilde{\boldsymbol{W}}^q \quad \forall p \in \{1, \dots, M\}$
- $c_p = \text{argmax}_j(\boldsymbol{q}_p \boldsymbol{k}_j + \gamma) \quad \forall i \in \{1, \dots, N\}\ \forall p \in \{1, \dots, M\}, \quad \text{where} \quad \gamma \sim \text{Gumbel}(0, 1)\}$

***step* 3**: apply selected rule to each slot conditioned on the selected contextual slot

- $\boldsymbol{R}_p = \text{RuleHead}_{\boldsymbol{r}_p}(\text{Concat}([\boldsymbol{V}_p^t, \boldsymbol{V}_{c_p}^t])) \quad \forall p \in \{1, \dots, M\}$
- $\boldsymbol{V}_p^{t+1} = \boldsymbol{V}_p^t + \boldsymbol{R}_p \quad \forall p \in \{1, \dots, M\}$

end

3.6 Proposed model

Augmented trajectories are forwarded through the model.

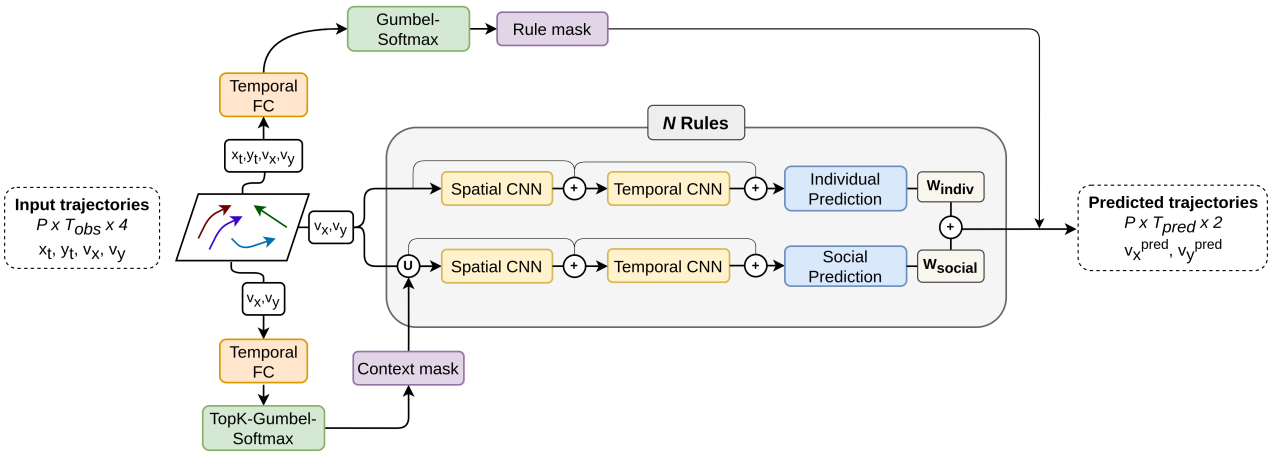


Figure 3.1: Proposed architecture.

Both absolute coordinates and velocities are used.

1. *Rule selection*
   For each input trajectory, a rule is selected based on the absolute coordinates and velocities, using Gumbel-Softmax.

2. *Context selection*
   For each input trajectory, a context trajectory is selected based on the velocities, using Gumbel-Softmax.

3. *Rule-heads*
   Each trajectory is processed using a sequence of spatial- and a temporal CNNs. On both the individual, and social path, a 1D CNN is used. The temporal CNN has 8 input and 12 output channels, making the model non-autoregressive. The individual and social predictions are aggregated using a learned weighted-sum.

In order to smoothly approximate the discrete sampling using the Gumbel-Softmax, we anneal the temperature parameter using the schedule

$$\tau_i = \tau_{\max} * \exp(-\log_e(\tau_{\min}/\tau_{\max})/\text{num\_epochs}) * i \quad \text{for } i \in [0, \text{num\_epochs}]$$

Using this schedule, the temperature parameter is annealed from 1.0 to 0.5 over the training.

This model serves as a generalization of the Social-Implicit model, and in the following we prove that it is capable to outperform the original model, while having no dataset-specific handcrafted routing. We also demonstrate some of the important aspects of this new model and explore different parameter settings and their effect on the performance.

As our model is very small, it is very easy to over-fit the training dataset, therefore the choice of optimizer, learning rate and seed becomes very important, but this comes from the original instability of the Social-Implicit model.

4 Experiments

4.1 Datasets

In order to thoroughly evaluate the performance of our trajectory prediction model, we have employed widely recognized datasets and metrics to benchmark our approach. This comprehensive evaluation allows us to compare different variations of our proposed model and contrast our results with those reported in the existing literature.

We have chosen to utilize two well-established datasets in the trajectory prediction domain for our experiments: ETH/UCY Dataset and the Stanford Drone Dataset. Afterwards, the two commonly used metrics are introduced, Average Displacement Error and Final Displacement Error.

By leveraging these datasets and evaluation metrics, we can effectively assess the performance of our proposed model in the trajectory prediction regime. This rigorous evaluation process enables us to identify the strengths and weaknesses of our approach, compare it with state-of-the-art methods, and ultimately contribute to the advancement of modular machine learning and attention-based trajectory prediction research.

4.1.1 ETH / UCY

Pedestrian trajectory forecasting is a critical component of numerous computer vision applications, including traffic analysis, crowd management, and autonomous driving systems. Accurate prediction of pedestrian movements can improve safety, optimize traffic flow, and enhance the overall efficiency of urban environments. To facilitate research in this area, several publicly available datasets have been created, such as the ETH and UCY datasets. These datasets are widely used for benchmarking various methods due to their high quality.

ETH and UCY datasets

The ETH dataset [29] was collected by ETH Zurich University and consists of two distinct scenes: ETH and Hotel. Both scenes were captured from a bird's eye view using a fixed camera, providing a comprehensive perspective of pedestrian movements. The dataset contains approximately 750 pedestrian trajectories annotated at a frequency of 2.5 frames per second (FPS).

Similarly, the UCY dataset [22] was compiled by the University of Cyprus and features three scenes: Univ, Zara1, and Zara2. These scenes were also recorded from a bird's eye perspective using a static camera. The UCY dataset comprises roughly 786 pedestrian tracks labeled at 2.5 FPS.

ETH-UCY combined dataset

Although the ETH and UCY datasets were not gathered concurrently or by the same group, they share similar attributes and are often used together. Consequently, they are generally referred to as the ETH-UCY dataset.

Evaluation methodology

To assess the performance of pedestrian trajectory prediction models, the ETH-UCY dataset is typically divided into the five scenes: ETH, Hotel, Univ, Zara1, and Zara2. This distribution allows for us to train in a cross-validation fashion. Researchers employ a leave-one-out methodology for evaluation, which involves training the model on four scenes and testing it on the remaining scene. This process is repeated five times, with each scene serving as the test set once. By averaging the results of these five evaluations, researchers can obtain a more reliable and comprehensive understanding of a model's performance, without compromising the evaluation by using unbalanced testing datasets.

4.1.2 Stanford Drone Dataset (SDD)

The Stanford Drone Dataset (SDD) [30] is a large-scale, real-world dataset for research on human trajectory forecasting and interactions. The key characteristics of this dataset include:

- Data Source:
  The SDD contains annotated aerial videos captured by drones flown over various locations at the Stanford University campus.
- Types of Entities:
  The dataset includes trajectories of pedestrians, bicyclists, skateboarders, carts, cars, buses and other entities in outdoor scenes.
- Number of Scenes:
  There are over 100 distinct static scenes such as plazas, sidewalks, and hallways.
- Dataset Size:
  The SDD has approximately 20,000 tracked targets with 11,000 pedestrians, 6,400 bicyclists, and 1,300 cars.
- Annotations:
  The data provides timestamped positions of all entities obtained by tracking the drone footage.
- Applications:
  The SDD has been extensively used for benchmarking trajectory forecasting models and analyzing human interactions.

The diversity of scenes, entity types and complex interactions make the SDD a rich dataset for research on trajectory prediction and modeling social behavior. Standardized data splits and evaluation protocols enable systematic comparison of different deep learning architectures on this data.

In this work, we follow the evaluation protocol of the Social-Implicit model, which is based on the setup of DAG-Net. This differs from the scaling used in some other models, but allows direct comparison to the Social-Implicit model results, which is our primary goal.

4.2 Optimizers and scheduling

Stochastic gradient descent

Stochastic gradient descent (SGD) is an optimization algorithm commonly used to minimize cost functions and train machine learning models. It is a variant of the gradient descent algorithm that approximates the true gradient by calculating it from a randomly selected subset of data points.

In standard gradient descent, the gradient is computed on the entire training dataset for each update to the model parameters. This becomes very slow and computationally expensive for large datasets with millions of data points, as the gradient calculation has to be performed on the full dataset in each iteration.

SGD overcomes this limitation by estimating the gradient from a small randomly selected batch of data points (mini-batch) in each iteration. The gradient estimate will be noisy compared to the full gradient, but tends to point in the right direction on average. By using mini-batches, SGD can make progress in the weight space in a faster and less expensive way. The stochasticity also helps escape local minima and saddles, allowing SGD to converge to an optimal set of parameters. The step size or learning rate is a key hyperparameter in SGD that controls how far the parameters move in the direction of the estimated gradient. An appropriate learning rate schedule (fixed, decaying, adaptive etc.) is important for good convergence. Momentum, Nesterov acceleration, and other techniques are often used along with SGD to improve its speed and stability.

Social-Implicit uses SGD with a high learning rate of 1.0 with batch size 128 and decay the initial learning to 0.1 after for the last 10% of the training.

We use the Social-Implicit setting for fair comparison, but experiment with different optimizers and schedulers to fully explore the potential of our proposed model.

Algorithm 4: Stochastic Gradient Descent

for training sample i in the randomly sampled batch of the full dataset **do**
  for for each weight j **do**
    $w_j \mathrel{+}= \Delta w_j$, where: $\Delta w_j = \eta(\text{target}^{(i)} - \text{output}^{(i)})x_j^{(i)}$
  end
end

AdamW

AdamW is a variant of the Adam optimizer that decouples weight decay from the adaptive learning rate based gradient updates. This allows us to apply weight decay regularization through the optimizer, while still benefiting from Adam's adaptive learning rates and momentum.

Algorithm 5: Adam with decoupled weight decay (AdamW)

for each randomly sampled batch **do**
  $t \mathrel{+}= 1$
  $\nabla f_t(\theta_{t-1}) =$ calculate gradient for θ_{t-1}
  $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\nabla f_t(\theta_{t-1})$
  $\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)(\nabla f_t(\theta_{t-1}))^2$
  $\hat{\mathbf{m}}_t = \mathbf{m}_t/(1-\beta_1^t)$
  $\hat{\mathbf{v}}_t = \mathbf{v}_t/(1-\beta_2^t)$
  $\theta_t = \theta_{t-1} - \eta_t \left(\alpha \hat{\mathbf{m}}_t/(\sqrt{\hat{\mathbf{v}}_t} + \epsilon) + \lambda \theta_{\mathbf{t-1}}\right)$
end

Cosine annealing with linear warm-up

Instead of the step-wise scheduling, we try out the commonly used cosine annealing learning rate scheduler, along with a linear warm-up period.

Cosine annealing gradually decreases the learning rate from a maximum value to a minimum value over each cycle, following a cosine curve shape. This allows for more gradual and smooth annealing of the learning rate compared to step-wise scheduling. After each cycle, the learning rate is reset to the maximum value, simulating a restart or "warm restart" of the training process. We use a single cycle in our experiments.

Before the cosine scheduler comes into play, we start with a so-called warm-up period in which the learning rate increases linearly from 0 to the maximum value over a fixed number of epochs. This warm-up helps stabilize the initial training and avoids sudden jumps in the learning rate at the start.

4.3 Implementation details

The pre-processed Stanford Drone Dataset (SDD) and ETH/UCY datasets were utilized for this research, following the data augmentation and sampling strategies outlined in the Social-Implicit project.

Using the same datasets and data processing as Social-Implicit is crucial, as it provides a fair baseline for comparison to evaluate how much the inclusion of Neural Production Systems (NPS) improves the prediction model. The rest of the analysis regarding the capabilities and detailed results of our model are presented in the following sections.

4.4 Hyperparameters

The stochastic gradient descent (SGD) optimizer used an initial learning rate of 1.0, following the setup in the Social-Implicit paper. In contrast, the AdamW optimizer computes adaptive learning rates for each parameter, requiring more tuning to find the optimal initial value. Through experimentation, we

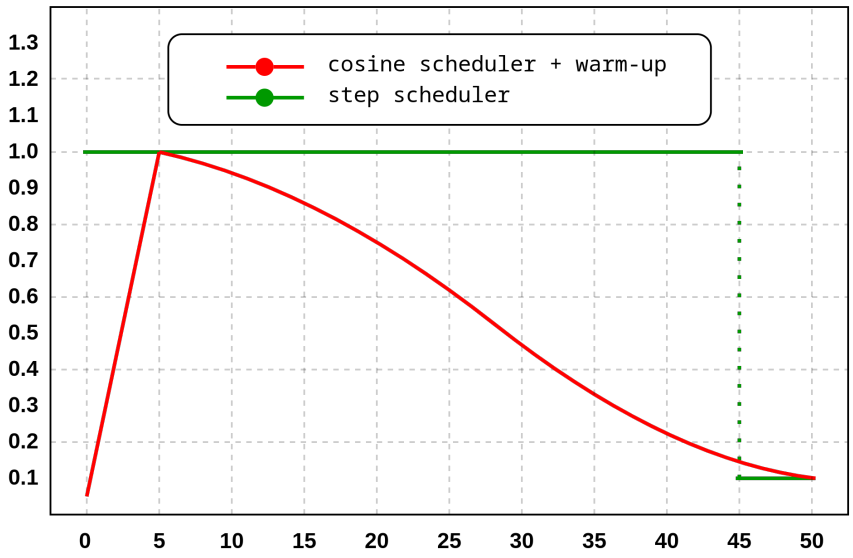


Figure 4.1: Step- and cosine scheduling.

determined an initial learning rate of $1e-2$ resulted in the fastest reduction in training loss without divergence.

For learning rate scheduling, StepLR followed the Social-Implicit paper. For all datasets except ETH, we used a step size of 45 epochs and a gamma of 0.1, balancing training speed and performance. For ETH, we used a step size of 10 to help with overfitting.

With cosine annealing, we smoothly decreased the learning rate from its initial value to 1/10th over training. We allocated 10% of epochs for warm-up starting from 0.0 before the cosine annealing schedule.

The model itself has few hyperparameters. We used 4 rules to match the 4 cells in Social-Implicit. A single context is chosen using Gumbel-Softmax, although later we show k contexts can be used. The rest of the model structure is fixed, requiring no tuning.

We trained each model for 50 epochs with a batch size of 128. The initially high learning rate quickly guides the model into a local minimum, so more epochs do not improve results.

4.5 Evaluation metrics

Evaluating the performance of machine learning models is a critical part of the model development process. Appropriate evaluation metrics need to be selected to quantify the model's capabilities on the desired tasks. For my research focusing on trajectory prediction, we utilized two key evaluation metrics - final displacement error and average displacement error - to evaluate my model's ability to accurately predict trajectories over time. The training of Social-Implicit is far from stable, the high learning rate and limited capacity makes it difficult to find the right hyperparameters for the inclusion of NPS. We tried our best to find the optimal settings, but the results presented can be noisy.

4.5.1 Average Displacement Error (ADE)

The primary metric used to assess the performance of trajectory prediction models is the Average Displacement Error (ADE). ADE quantifies the accuracy of a model's predictions by calculating the mean Euclidean (L2) distance between the predicted points and the actual points in a trajectory.

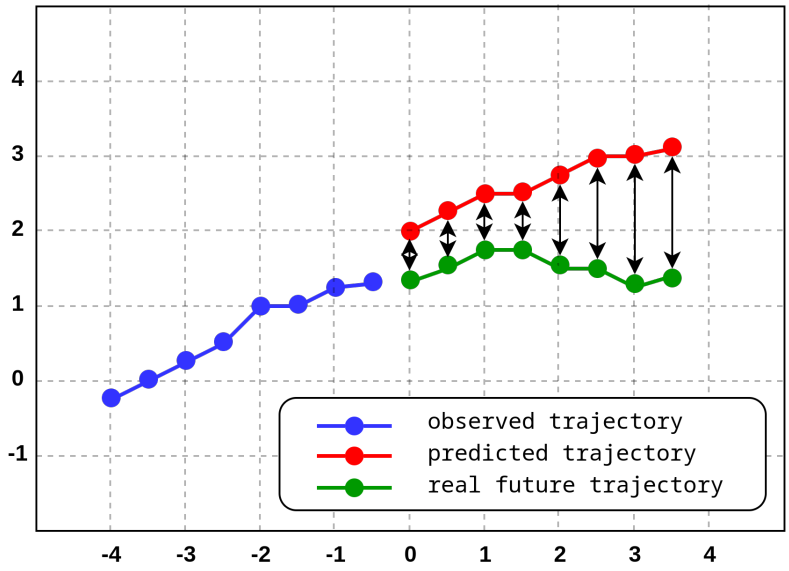


Figure 4.2: Average Displacement Error is the average distance between predicted and real future trajectories, therefore the average length of the black arrows.

The formula for ADE is as follows:

$$\text{ADE} = \frac{\sum_{i=1}^{n} \sum_{t=T_{obs}}^{T_{pred-1}} \|\hat{Y}_t^i - Y_t^i\|_2}{n(T_{pred} - T_{obs})}$$

Here, the variables represent the following:

- n: total number of trajectories
- T_{obs}: observation time horizon
- T_{pred}: prediction time horizon
- $\hat{Y}_t^i$: predicted position of the i-th trajectory at time t
- Y_t^i: ground truth position of the i-th trajectory at time t

A lower ADE value indicates better accuracy in predicting trajectories, making it a valuable tool for comparing the performance of different models in the field of trajectory prediction.

4.5.2 Final Displacement Error (FDE)

The Final Displacement Error (FDE) is another key metric for evaluating the accuracy of trajectory prediction models. FDE measures the mean Euclidean (L2) distance between the last predicted point and the last ground truth point for each trajectory. The formula for FDE is:

$$\text{FDE} = \frac{\sum_{i=1}^{n} \|\hat{Y}_{T_{pred-1}}^i - Y_{T_{pred-1}}^i\|_2}{n}$$

FDE focuses on the long-term accuracy of trajectory predictions, as it only considers the error at the final time step. A lower FDE value indicates better long-term accuracy in predicting trajectories.

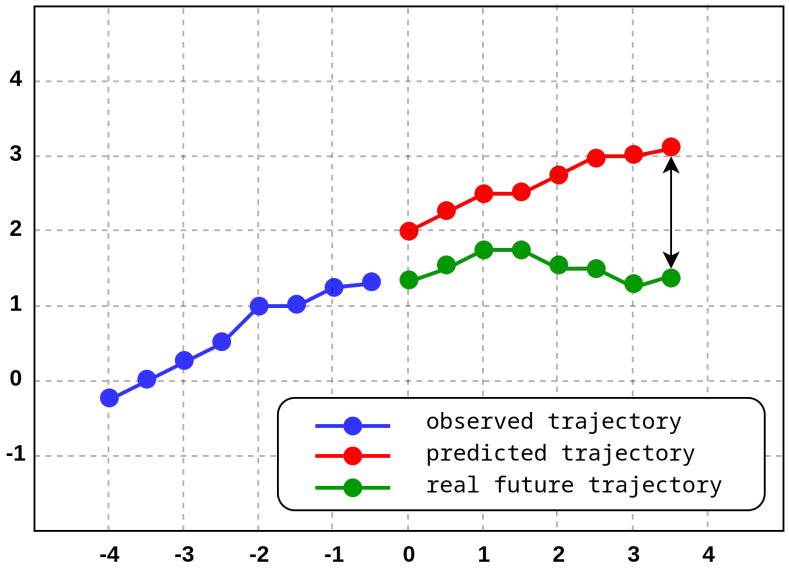


Figure 4.3: Final Displacement Error is the distance between final point of the predicted and real future trajectories, therefore the lenght of the black arrow.

4.5.3 Stochastic model evaluation

The current state-of-the-art methods for trajectory prediction are predominantly generative models, which means they predict a probability distribution over future trajectories rather than exact coordinate predictions. While generative modeling has advantages, it also poses challenges for evaluation on benchmark datasets, which require single point predictions. The common evaluation protocol is to generate 20 sample predictions from the distribution output by the model, and select the sample with the lowest error compared to the ground truth future trajectory.

As discussed in depth in [1], this sampling evaluation methodology has many drawbacks. Most importantly, it rewards models that produce high-variance distributions to increase chances of getting close samples purely by chance, rather than models that produce accurate, low-variance predictions. Despite the flaws, this sampling evaluation procedure remains the standard in the literature.

In our work, we adopt this same sampling evaluation protocol in order to compare our proposed model against existing state-of-the-art methods on benchmark datasets. However, we believe that assessing performance through continual evaluation and other methods described in [1] would greatly decrease the noise in this segment of machine learning research.

4.6 Quantitative results

In this section, we present a comprehensive comparison of our solution with various benchmark models and state-of-the-art approaches in pedestrian trajectory prediction. Our aim is to demonstrate the effectiveness of our proposed model, which combines Neural Production System (NPS) with Social-Implicit, in improving the accuracy and long-term prediction of multi-entity trajectories.

To evaluate the performance of each model, we conduct experiments on two widely used datasets in pedestrian trajectory prediction, ETH and UCY, and one newer dataset, SDD.

Using the ETH/UCY datasets, we compare our solution with a total of 14 models, including 2 benchmark models (Linear and LSTM), 4 historically important models (Social-LSTM, Social-GAN, Social-Ways, and STGAT), 3 transformer-based architectures (TransformerTF, STAR, and AgentFormer), and 3 recent state-of-the-art (SOTA) methods (SGNet, SocialVAE, and ExpertTraj). Additionally, we compare our solution with Social-Implicit and the author's previous work, Social-STGCNN.

When evaluating our solution on the SDD dataset, we compare it to 4 models (Social-Ways, STGAT, DAG-Net, and Social-Implicit) due to the pre-processing method used.

This comprehensive comparison allows us to assess the performance of our model in various scenarios and against a wide range of existing approaches. By comparing our solution with these benchmark models, historically important models, transformer-based architectures, and recent SOTA methods, we can provide a thorough analysis of its strengths and weaknesses. This analysis will not only demonstrate the effectiveness of our proposed model but also contribute to the existing body of knowledge in pedestrian trajectory prediction.

Table 1 presents the experimental results obtained for each model in five scenarios included in the ETH and UCY datasets, and the average results for each pedestrian trajectory prediction method are given in the last column. **Bold** values highlight the best result across all models, while **purple** values highlight the best result between Social-Implicit and our solution.

Table 4.1: Evaluation results for next 12 time-step prediction on the ETH and UCY datasets. The numbers represent the minimum ADE and FDE of 20 predicted samples. The Linear model is deterministic and is listed as a baseline. **Bold** numbers indicate the lowest values, while **purple** numbers indicate the lowest values between the Social-Implicit and Our model.

Results: ETH/UCY Datasets

MODEL	ETH		HOTEL		UNIV		ZARA1		ZARA2		AVG	
	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE
Linear	1.33	2.94	0.39	0.72	0.82	1.59	0.62	1.21	0.77	1.48	0.79	1.59
LSTM	1.09	2.41	0.86	1.91	0.61	1.31	0.41	0.88	0.52	1.11	0.70	1.52
Social-LSTM	1.09	2.35	0.79	1.76	0.67	1.40	0.47	1.00	0.56	1.17	0.72	1.54
Social-GAN	0.81	1.52	0.72	1.61	0.60	1.26	0.34	0.69	0.42	0.84	0.58	1.18
Social-Ways	0.39	0.64	0.39	0.66	0.55	1.31	0.44	0.64	0.51	0.92	0.46	0.83
STGAT	0.65	1.12	0.35	0.66	0.52	1.10	0.34	0.69	0.29	0.60	0.43	0.83
TransformerTF	0.61	1.12	0.18	0.30	0.35	0.65	0.22	0.38	0.17	0.32	0.31	0.55
STAR	0.36	0.65	0.17	0.36	0.31	0.62	0.26	0.55	0.22	0.46	0.26	0.53
AgentFormer	0.45	0.75	0.14	0.22	0.25	0.45	0.18	**0.30**	0.14	0.24	0.23	0.39
SGNet	0.47	0.77	0.21	0.44	0.33	0.62	0.18	0.32	0.15	0.28	0.27	0.49
SocialVAE	0.47	0.76	0.14	0.22	0.25	0.47	0.20	0.37	0.14	0.28	0.24	0.42
ExpertTraj	**0.30**	**0.62**	**0.09**	**0.15**	**0.19**	**0.44**	**0.15**	0.31	**0.12**	**0.24**	**0.17**	**0.35**
Social-STGCNN	0.64	1.11	0.49	0.85	0.44	0.79	0.34	0.53	0.30	0.48	0.44	0.75
Social-Implicit	0.66	1.44	0.20	0.36	**0.31**	**0.60**	0.25	0.50	0.22	**0.43**	0.33	0.67
Ours (Social-Implicit+NPS)	**0.50**	**0.79**	**0.16**	**0.28**	0.32	0.62	**0.24**	**0.46**	**0.22**	0.44	**0.29**	**0.52**

Table 2 shows similar results, but obtained on the SDD dataset. This table contains less information due to the lack of models to compare to, but as we followed the code-setup of the Social-Implicit paper, we had to use the numbers from their paper and we had no resources to implement all models from Table 1 and run them on this version of SDD. Table 2 is still important as we can draw conclusions about the performance gain of our solution, and during the experiments of the improvements we propose.

Our experimental results demonstrate that our proposed model outperforms all other comparable models on both datasets in terms of ADE and FDE metrics. Specifically, on the ETH-UCY datasets, we observe a 12.12% decrease on the ADE metric and a 22.39% decrease on the FDE metric. On the SDD dataset, we observe a 12.77% decrease on the ADE metric and a 20.22% decrease on the FDE metric.

These results confirm that the inclusion of NPS instead of handcrafted cells significantly improves the performance of the model, even with including significantly less context trajectories on the social aspect of the model.

This efficient solution not only leads to better performance but also improves long-term prediction significantly, as the FDE metric is decreased almost twice as much as the ADE metric.

Table 4.2: Evaluation results for next 12 time-step prediction on the SDD dataset. The numbers represent the minimum ADE and FDE of 20 predicted samples. **Bold** numbers indicate the lowest values.

Results: SDD Datasets		
MODEL	**SDD**	
	ADE	FDE
Social-Ways	0.62	1.16
STGAT	0.58	1.11
DAG-Net	0.53	1.04
Social-Implicit	0.47	0.89
Ours (Social-Implicit+NPS)	**0.41**	**0.71**

We acknowledge, that significantly bigger models can still outperform our solution, but the difference is less significant when we compare the performance with the number of necessary parameters.

In the following sections, we provide a detailed analysis of the performance gain achieved by our proposed model and highlight the key features that make it superior to other models in the field of pedestrian trajectory prediction.

Prediction visualizations can be found in Appendix B.

4.7 Ablation studies

In this chapter, we present a series of ablation studies conducted on the proposed trajectory prediction model to quantify the importance of key architectural components. Specifically, we focus on the number of the rules being used, auto-regressiveness and NPS variants. For each ablation experiment, we detail the modified model architecture, training methodology, and evaluation results on the trajectory forecasting tasks.

The findings from these studies demonstrate the significance of each component and validate their inclusion in the full model. Altogether, these ablation experiments provide a granular analysis into the inner workings of our model and the rationality behind its design. The remainder of this chapter dives into the setup, results, and insights gleaned from each ablation study.

4.7.1 Optimizer and scheduling

We compare the optimizer + scheduler of Social-Implicit to a more recently used solution. AdamW solves the commonly-know issue of weight-decay in Adam, that convinced people to use SGD in order to get the best results possible.

Cosine annealing and warm-up helps to stabilize the convergence and is the chosen methodology in recent influential papers such as [35].

Results: ETH/UCY (AVG) and SDD Datasets				
Optimizer - Scheduler	**ETH/UCY**		**SDD**	
	ADE	FDE	ADE	FDE
SGD - StepLR	**0.29**	**0.52**	0.45	**0.78**
AdamW - Linear Warm-up + Cosine Annealing	0	0	**0.43**	0.80

Table 4.3: Effect of the optimizer and scheduler on the performance.

When using the SDD, AdamW produces a lower ADE and higher FDE score, but we can conclude that the difference in both cases is irrelevant and therefore both approaches could be utilized.

TODO ETH/UCY analysis if results are in.

4.7.2 Data Normalization

As we mentioned earlier, there are different ways to normalize the input trajectories. In order to determine, which option is most suitable, we experimented with all options.

Table 4.4: Different data normalization options

Results: ETH/UCY (AVG) and SDD Datasets				
Data normalization	**ETH/UCY**		**SDD**	
	ADE	FDE	ADE	FDE
Absolute coordinates	0.29	0.52	0.41	0.71
Batch normalization (original)	**0.29**	**0.52**	0.45	0.78
Re-scale to origin at last time-step	0.30	0.53	0.41	**0.71**
Re-scale to origin at first time-step	0.29	0.52	**0.41**	0.72

We can see from Table 4.4 that there is no significant difference between the data normalization techniques. Based on early experiments, we decided to use batch normalization, although it performs a bit worse on the SDD dataset. Nevertheless, it still outperforms Social-Implicit by a strong margin.

We conclude that the normalization technique is not a major factor when assessing the performance of the model, and that our model person well even without any normalization, just as the Social-Implicit model.

4.7.3 Number of rules

The rule-selection procedure is a crucial aspect of our model. Gumbel-softmax allows for the rule-selection procedure to be learned using back-propagation end-to-end. This means that the number of rules utilized can be treated as a hyperparameter, which can be optimized for each dataset.

To understand the optimal number of rules to be used for each dataset, we conducted experiments with different number of rules. This allowed us to see how far we could scale the model simply by using more rules.

Our findings were consistent with previous research in the field. In both the Neural Production Systems (NPS) paper and the Mixture of Experts (MoE) literature, it is recommended to use less than 10 modules. This is because using too many modules can make it difficult for the model to figure out how to use all of them effectively, resulting in a fuzzy solution where modules are not really independent and therefore the whole model is less efficient.

As per our experiments, we found that the optimal number of rules for each dataset varied. ETH-UCY worked well with 4 or 5 rules, while SDD worked significantly better with only 4 rules. However, as we can see from [REF], using 4 rules seems to be quite adequate for both datasets.

Our experiments also showed that using a few more or less rules did not significantly impact the performance of the model. This suggests that the model is able to effectively utilize a range of different numbers of rules, providing flexibility in the design of the model.

Overall, our findings suggest that the number of rules used should be carefully selected based on the specific dataset and problem at hand. However, using a learned rule-selection procedure can provide flexibility in the number of rules utilized, allowing for optimization of the model's performance.

It is important note that in this version of the training, nothing forces the model to balance the load on the modules.

Table 4.5: Effect of the number of rules being used on the performance of the model

Results: ETH/UCY (AVG) and SDD Datasets					
Number of rules	**Parameter Count**	**ETH/UCY**		**SDD**	
		ADE	FDE	ADE	FDE
1	3275	0.34	0.68	0.45	0.78
2	4166	0.32	0.63	0..49	0.90
3	5057	0.33	0.64	0.49	0.86
4	5948	**0.29**	0.52	**0.41**	**0.71**
5	6839	0.29	**0.52**	0.44	0.80
10	11294	0.32	0.63	0.50	0.89

This means that even if 10 rules are available, the model still can learn to only use a single rule and not forward any input to the other 9. This could be improved with additional auxiliary losses, but its effectiveness would remain questionable [20].

4.7.4 Autoregressive variation

To determine if the model's performance is constrained by its non-autoregressive nature, we can test a variation where the model recursively predicts one time step into the future per forward pass. The model's own prediction is then fed back as input to predict the next time step. This autoregressive approach has some key aspects to consider:

Number of channels in the last convolutional layer

We reduced the number of channels in the final convolutional layer from 12 to 8 in order to force the model to predict only a single time step into the future per input sequence. With 12 output channels, the model was designed to predict 12 future time steps simultaneously in a non-autoregressive manner. By reducing this to 8 channels, the model's output dimension is constrained to match the input sequence length. We can then compare the first 7 predicted values to the last 7 values from the input sequence. This allows us to evaluate the model's one-step-ahead prediction accuracy. The final 8th predicted value can be fed back as input to recursively generate multi-step forecasts. At each iteration, the model is limited to predicting a single incremental time step based on the previous iterative predictions.

We also considered reducing the number of output channels to 1 instead of 8. However, this extreme reduction would eliminate many of the benefits provided by using a convolutional neural network model. Reducing the output channels to 1 would also limit our ability to effectively select from the 20 sampled predictions for the autoregressive feedback, which can be easily done using the other 7 channels.

IMLE noise generation

Since we generate 20 samples per input, we must select which sample to use for the autoregressive feedback. A straightforward approach is to choose based on proximity to the input trajectory. Specifically, we can measure the distance between:

1. last $T_{\text{pred}} - 1$ items of the last input trajectory along the temporal axis
2. first $T_{\text{pred}} - 1$ items of each predicted trajectory

Given these 20 distance values, we can choose to use the prediction for the autoregression, that minimizes the first term of our loss function:

$$\mathcal{L}_{\text{IMLE}} = \|d_p - \tilde{d}_p\|_1$$

Using this nearest neighbor selection provides a simple and intuitive way to pick appropriate predictions for feeding back into the model recursively. The sample most consistent with the input acts as the starting point for the next iteration's forecast.

Training time and inference cost

Adopting an autoregressive approach comes with additional computational expenses both during training and inference. For training, the model must learn to recursively refine its predictions over multiple time steps. This requires unrolling the model through longer sequences, increasing training time per batch. At inference, the sequential nature means multiple passes are necessary to generate a full output sequence. This reduces computational parallelism and throughput compared to one-shot non-autoregressive prediction. Exact costs depend on model architecture details, sequence lengths, and hardware acceleration support. But in general, both training and inference stages become less efficient as more sequential iterations are required per sample.

Table 4.6: Autoregressive / Non-Autoregressive comparison on the SDD dataset. Bold numbers show the best results.

Results: SDD Datasets

METHOD	ADE	FDE	Parameter Count	Training time per epoch	Inference time validation set	Inference time test set
Non-Autoregressive (4 rules)	**0.42**	**0.73**	5948	**6 (sec)**	**<1 (sec)**	**∼1 (sec)**
Autoregressive (4 rules)	0.49	0.81	**4860**	50 (sec)	∼1 (sec)	∼3 (sec)
Autoregressive (6 rules)	0.47	0.87	6098	82 (sec)	∼2 (sec)	∼12 (sec)

We can see that non-autoregressivness not only improves the results, but increases the training and inference-speed by a big margin.

4.7.5 NPS variants

The Neural Production Systems (NPS) model offers a fresh perspective on the rule-and context-selection process in machine learning. However, the complete independence of the rule-and context-selection can be seen as an unnecessary factorization, which can make the optimization problem harder than necessary.

To address this issue, we have two possibilities. The first is to choose the rules first and condition the context-selection on the chosen rules. The second possibility is to do the exact opposite, by choosing the context first and conditioning the rule-selection on the chosen context. These approaches can simplify the optimization problem by reducing the number of possible combinations of rules and contexts.

Rule-specific attention

To condition the context-selection on the result of the rule-selection, we can define one attention matrix used for the context selection for each rule. This way, after the rule-selection is done, we can use the appropriate attention matrix to select the contexts. This variation of NPS solves the issue of complete independence between the two selection procedures, providing a more efficient and effective solution.

However, it is important to note that this approach is quite brute-force and comes with a significant increase in parameters. By defining one attention matrix for each rule, we are effectively increasing

the number of parameters in the model, which can make it more complex and harder to optimize, without the appropriate increase in capacity.

The results show that this modification of NPS provides a slight improvement on the SDD dataset, but not on the ETH/UCY.

Table 4.7: Effect of the rule-conditioned context selection on the performance.

Results: ETH/UCY (AVG) and SDD Datasets					
Model	**Parameter Count**	**ETH/UCY**		**SDD**	
		ADE	FDE	ADE	FDE
Original	5948	**0.29**	**0.52**	0.45	0.78
Rule-conditioned context selection	12248	0.33	0.65	**0.43**	**0.77**

As the size of the model almost triples, we can conclude that this modification does not contribute much to the overall effectiveness of the model.

The reason why the ETH/UCY dataset shows worse result can be due to the Social-Implicit rule-head, as the ETH dataset tends to easily over-fit.

Context conditioned rule selection

To condition the rule-selection on the context selection procedure, we can simply aggregate each trajectory with its context trajectory, resulting in a new 4-dimensional trajectory that can be used in the rule-selection procedure. This approach is simpler than defining one attention matrix for each rule, and comes with fewer changes in the algorithm.

By aggregating each trajectory with its context trajectory, we are effectively conditioning the rule-selection on the context selection procedure. This allows us to take into account the context information when selecting the rules.

Table 4.8: Effect of the context-conditioned rule selection on the performance.

Results: ETH/UCY (AVG) and SDD Datasets					
Model	**Parameter Count**	**ETH/UCY**		**SDD**	
		ADE	FDE	ADE	FDE
Original	5948	**0.29**	**0.52**	0.45	**0.78**
Context-conditioned rule selection	6204	0.33	0.66	**0.44**	0.80

Similar to the previous variation of NPS, conditioning the rule selection on the selected contexts without any embedding does improve the performance of the model. It is possible that with even a minimal encoder, forcing the model to learn about the connection between the rule- and context-selection would become easier as it could encode

4.8 Future work

Gumbel-Softmax-based context selection and modularization seems promising for trajectory prediction, but there are several ways to improve its current form in the future. These changes have the possibility to increase the model's capacity as well as its training robustness.

4.8.1 Top-k context selection

We can easily extend the Gumbel-Softmax trick to hard select not only 1 element, but TopK based on [31]. This allows us to include more than a single context trajectory, but we have different options to include this new information into our framework. This modification increases the complexity our model can handle, and introduces an important application-dependent hyperparameter to the model, effectively allowing us to balance the sparsity of social interaction based on the dataset.

In the following we introduce how we could incorporate the selected k context entities into the model to efficiently learn about their influence.

High dimensional coordinate space

We can keep using the methodology, where the TopK selected context trajectory is concatenated to the primary trajectory along the spatial dimension. The main issue with this solution, is that the ordering of these TopK context trajectories play a role in the learning process of our model, while in reality this ordering has nothing to do with the information at hand.

We can possibly help this issue by ordering the selected context trajectories based on their attention score.

Another problem is that if the social dynamic are quite dense, then we might want to use a higher value of K, which would significantly increase the dimension of the concatenated tensor, and increase the size of the model.

2D-convolution

Another solution is to bring back the 2D convolutions from the Social-Implicit model, which helps with the permutation invariance.

This approach solves the problem of high dimensions and increased model size, but does not satisfy the permutation invariance requirement.

GNN motivated permutation invariant context aggregation

Lastly we can use the aggregation step from GNNs, therefore using some permutation invariant aggregation to merge the context trajectories, and the concatenate this aggregated version into the primary trajectory along the spatial dimension.

Simple aggregation functions such as SUM, MEAN, MIN, MAX could be used, but in this case some sort of embedding prior to the rule-heads and aggregation would be useful.

4.8.2 Balanced load for each module

This model currently does not have any inductive bias towards balancing the load for the modules. MoE literature often uses some sort of auxiliary loss function to help this issue. This loss could penalize over- and under-loaded modules, and therefore force the model to avoid module-collapse.

5 Conclusions

This thesis has successfully explored the combination of modular deep learning with state-of-the-art trajectory prediction techniques, with a specific focus on the viability of structured sparsity for efficient and generalizable trajectory forecasting.

The primary objective was to move away from commercial transformer- or GNN-based models and instead investigate the adequacy of Gumbel-Softmax for introducing sparsity and modularity. By selecting a single context and forwarding it to a single module, a highly efficient framework was created. This framework was then combined with Social-Implicit to leverage its non-autoregressive nature, resulting in a small, quick, and well-performing model.

The experimental setup, as presented in Chapter 4, included benchmark datasets, optimizers, hyperparameters, evaluation metrics, and ablation studies. The results on pedestrian trajectory forecasting tasks were compared to prior work, and the chapter critically analyzed the experimental results, discussed the limitations of the current approach, and suggested promising directions for future work to further improve efficiency and accuracy. The findings demonstrated that SI-NPS outperforms its predecessors, improving both the Average Displacement Error (ADE) and the Final Displacement Error (FDE) metrics on both datasets.

Furthermore, we conducted experiments to explore additional improvements to the NPS model and modified it specifically for trajectory prediction. This research opens up new possibilities for the integration of modular deep learning and trajectory forecasting, paving the way for more efficient and accurate models in the future.

Bibliography

[1] Alexandre Alahi, Kshitij Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social-Implicit: Learning social Etiquette With Implicit Signals and Maximum Entropy Markov Modeling. *European Conference on Computer Vision*, 2014.

[2] Alexandre Alahi, Kshitij Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social LSTM: Human Trajectory Prediction in Crowded Spaces. *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[3] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. *arXiv preprint arXiv:1904.09081*, 2019.

[4] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Modular deep learning. *arXiv preprint arXiv:1606.04658*, 2016.

[5] Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. Glam: Efficient scaling of language models with mixture-of-experts. *arXiv preprint arXiv:2112.06905*, 2022.

[6] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformer: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

[7] Francesco Giuliari, Irtiza Hasan, Marco Cristani, and Fabio Galasso. Transformer networks for trajectory forecasting. *arXiv preprint arXiv:2003.08111*, 2020.

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 2020.

[9] Anirudh Goyal, Ingmar Kanitscheider, Kelvin Xu, Yoshua Bengio, Sergey Levine, Dale Schuurmans, Aaron Courville, and Samy Bengio. Neural production systems: Learning rule-governed visual dynamics. *arXiv preprint arXiv:2103.01937*, 2021.

[10] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 2017.

[11] Abhishek Gupta, Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Social GAN: Socially acceptable trajectories with generative adversarial networks. *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[12] Abhishek Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social-Ways: Learning Social Norms from Human Trajectories. *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[13] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 2017.

[14] Dirk Helbing and Peter Molnar. Social Forces: An Approach to Social Dynamics. *European Physical Journal B*, 1995.

[15] Christoph Herrmann, Christopher Klinkmüller, Stefan Schönig, Christian Schön, and Dominik Schön. A framework for production system design: Insights from industrial experience. *Procedia CIRP*, 2019.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[17] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[18] Jared Kaplan, Sam McCandlish, Tom Henighan, Lasse Espeholt, Ankur Chaurasia, Minhao Li, Ofir Bansal, Pieter Abbeel, and Ilya Sutskever. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[19] Sujeong Kim, Stephen J. Guy, Wenxi Liu, David Wilkie, Rynson W.H. Lau, Ming C. Lin, and Dinesh Manocha. Brvo: Predicting pedestrian trajectories using velocity-space reasoning. *International Journal of Robotics Research*, 2015.

[20] Sujeong Kim, Stephen J. Guy, Wenxi Liu, David Wilkie, Rynson W.H. Lau, Ming C. Lin, and Dinesh Manocha. Mixture-of-experts with expert choice routing. *arXiv preprint arXiv:2202.09368*, 2022.

[21] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[22] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. Crowds by example. *Computer Graphics Forum*, 2007.

[23] Yaguang Li, Jianming Zhang, and Kaiqi Zhang. SgNet: Semantic Graph Network for Trajectory Prediction. *IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[24] Yaguang Li, Jianming Zhang, and Kaiqi Zhang. Dag-Net: Double Attention Guided Network for Trajectory Prediction. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[25] Yujia Li, Yuxuan Wu, Yuchen Zhang, and Yizhou Wang. AgentFormer: Agent-Based Transformer for Trajectory Prediction. *IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

[26] Michael Maire, Pablo Arbelaez, Jonathan T. Barron, Lubomir Bourdev, Ross Girshick, Saurabh Gupta, Bharath Hariharan, Kaiming He, Dinesh Jayaraman, Sergey Karayev, et al. Scaling vision with sparse mixture of experts. *arXiv preprint arXiv:2103.07579*, 2021.

[27] Karttikeya Mangalam, Harshayu Girase, Shreyas Agarwal, Kuan-Hui Lee, Ehsan Adeli, Jitendra Malik, and Adrien Gaidon. It is not the journey but the destination: Endpoint conditioned trajectory prediction. *European Conference on Computer Vision*, 2020.

[28] Abduallah Mohamed, Kun Qian, Mohamed Elhoseiny, and Christian Claudel. Social-stgcnn: A social spatio-temporal graph convolutional neural network for human trajectory prediction. *Proceedings of the IEEE/CVF International Conference on Computer Vision and Pattern Recognition*, 2020.

[29] Stefano Pellegrini, Andreas Ess, and Luc Van Gool. You'll never walk alone: Modeling social behavior for multi-target tracking. *2009 IEEE 12th International Conference on Computer Vision*, 2009.

[30] Alexandre Robicquet, Amir Sadeghian, Alexandre Alahi, and Silvio Savarese. Learning social etiquette: Human trajectory understanding in crowded scenes. *European Conference on Computer Vision*, 2016.

[31] Alexey Romanov, Anna Rumshisky, and Michael Collins. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

[32] Amir Sadeghian, Vineet Kosaraju, Ali Sadeghian, Noriaki Hirose, S. Hamid Rezatofighi, and Silvio Savarese. Sophie: An attentive gan for predicting paths compliant to social and physical constraints. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[33] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 2009.

[34] Clemens Schöller and Vincent Aravantinos. What the constant velocity model can teach us about pedestrian motion prediction. *arXiv preprint arXiv:1903.07933*, 2019.

[35] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2302.13971*, 2023.

[36] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2018.

[37] Anirudh Vemula, Katharina Muelling, and Jean Oh. Social attention: Modeling attention in human crowds. *Proceedings of IEEE Conference on Robotics and Automation (ICRA)*, 2018.

[38] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Textbooks are all you need ii: phi-1.5 technical report. *arXiv preprint arXiv:2004.10195*, 2020.

[39] Paul J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 1990.

[40] Pei Xu, Yifei Zhang, Yifan Wang, and David J. Crandall. Socialvae: Human trajectory prediction using timewise latents. *European Conference on Computer Vision*, 2022.

[41] Huaxiu Yao, Fei Wu, and Jintao Ke. Stgat: Modeling spatial-temporal interactions for human. *AAAI Conference on Artificial Intelligence*, 2020.

[42] Changqian Yu, Xiaojun Ma, Jie Ren, Haifeng Zhao, and Shuai Yi. Spatio-temporal graph transformer networks for pedestrian trajectory prediction. *European Conference on Computer Vision*, 2020.

[43] He Zhao and Richard P. Wildes. Where are you heading? dynamic trajectory prediction with expert goal examples. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.

Appendix A Parallel NPS pseudo-code

Pseudo-code for Parallel NPS. We used the modified version throughout out experiments.

Algorithm 6: Parallel NPS

Input: sequential data $\{x^1, x^2, \ldots, x^T\}$,
set of embeddings for rule-matching $\vec{\boldsymbol{R}}_i$,
set of rule-heads $\boldsymbol{R}_{1\ldots N}$,
embedding for the null-rule $\vec{R}_{\text{Null}}$

for $t \in [1, T]$ **do**

***step* 1**: infer the entity *state* in each slot j, from the previous state and the current input

***step* 2**: select the set of primary slots $\boldsymbol{P}$

- $\boldsymbol{R}_{\text{cat}} = \text{Concat}([\vec{R}_i \mid \forall i \in \{1, \ldots, N\}])\boldsymbol{W}^{R_a}$
- $\boldsymbol{k}_j = \boldsymbol{V}_j^t \hat{\boldsymbol{W}}^k \quad \forall j \in \{1, \ldots, M\}$
- $\boldsymbol{P} = \{j \text{ if } \boldsymbol{R}_{\text{cat}}\boldsymbol{k}_j + \gamma > \boldsymbol{R}_{\text{Null}}\boldsymbol{k}_j + \gamma \text{ where } j \in \{1, \ldots, M\} \text{ and } \gamma \sim \text{Gumbel}(0, 1)\}$

***step* 3**: select a rule for each primary slot in $\boldsymbol{P}$

- $\boldsymbol{k}_i = \vec{\boldsymbol{R}}_i \boldsymbol{W}^k \quad \forall i \in \{1, \ldots, N\}$
- $\boldsymbol{q}_p = \boldsymbol{V}_p^t \boldsymbol{W}^q \quad \forall p \in \boldsymbol{P}$
- $\boldsymbol{r}_p = \text{argmax}_i(\boldsymbol{q}_p \boldsymbol{k}_i + \gamma) \quad \forall i \in \{1, \ldots, N\} \quad \forall \boldsymbol{p} \in \boldsymbol{P}, \quad \text{where} \quad \gamma \sim \text{Gumbel}(0, 1)\}$

***step* 4**: select a contextual slot for each primary slot in $\boldsymbol{P}$

- $\boldsymbol{k}_j = \boldsymbol{V}_j^t \tilde{\boldsymbol{W}}^k \quad \forall j \in \{1, \ldots, M\}$
- $\boldsymbol{q}_p = \boldsymbol{V}_p^t \tilde{\boldsymbol{W}}^q \quad \forall p \in \boldsymbol{P}$
- $c_p = \text{argmax}_j(\boldsymbol{q}_p \boldsymbol{k}_j + \gamma) \quad \forall i \in \{1, \ldots, N\} \quad \forall \boldsymbol{p} \in \boldsymbol{P}, \quad \text{where} \quad \gamma \sim \text{Gumbel}(0, 1)\}$

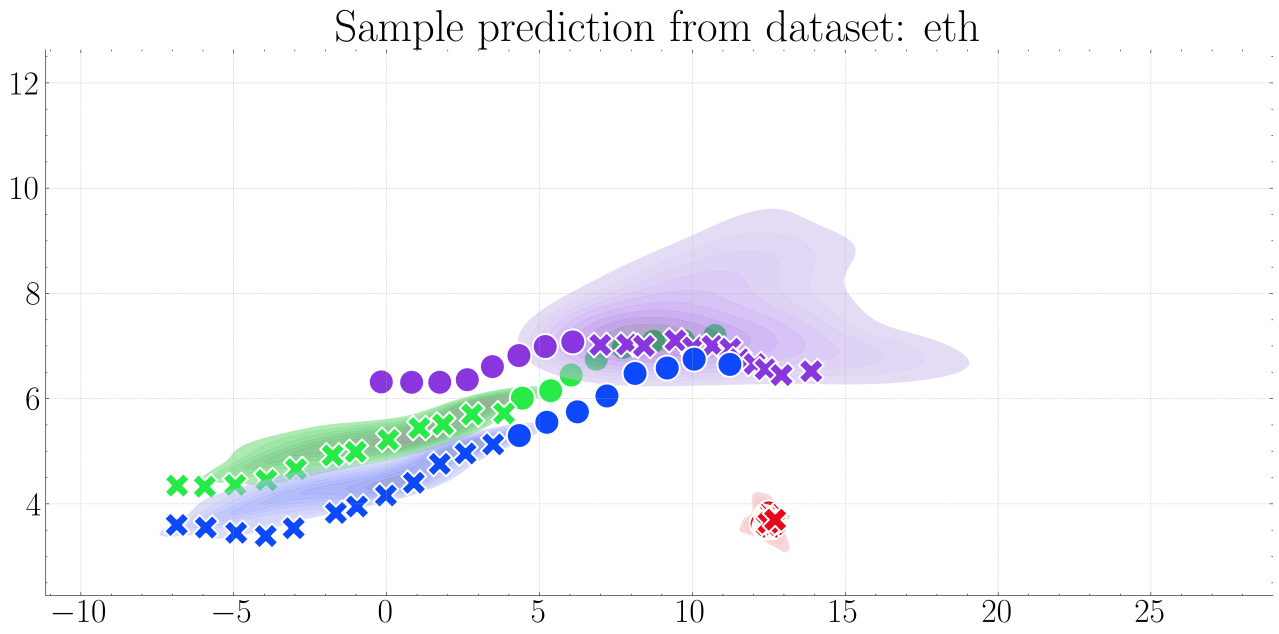
***step* 5**: apply selected rule to each primary slot conditioned on the contextual slot

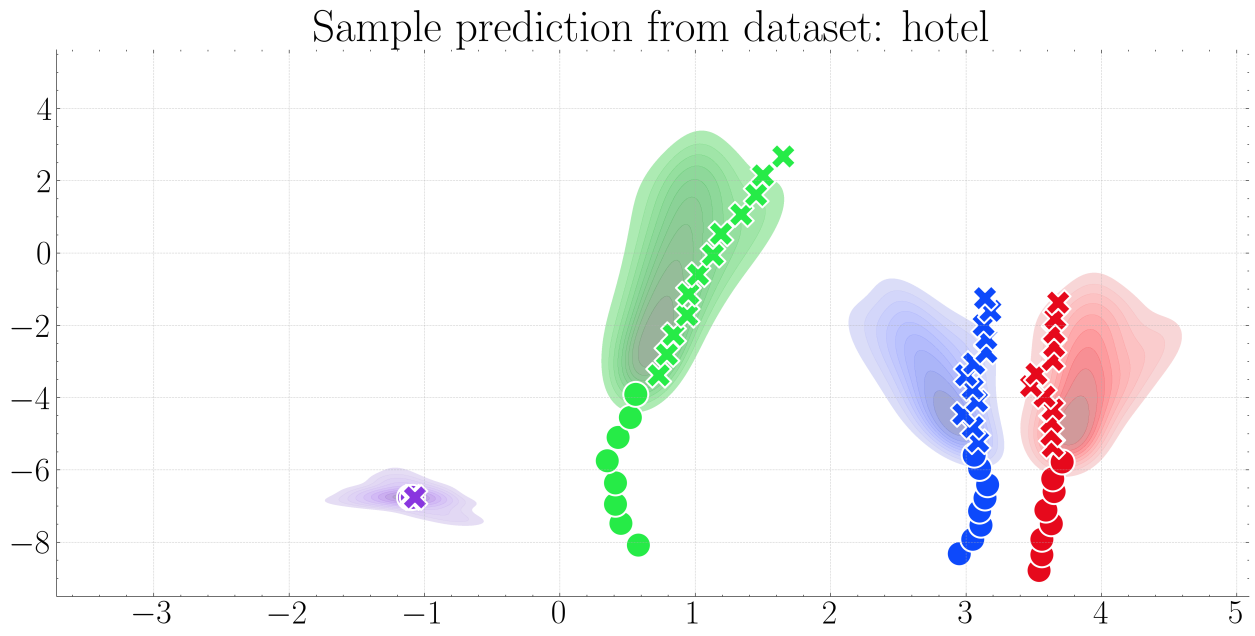
- $\tilde{\boldsymbol{R}}_p = \text{RuleHead}_{\boldsymbol{r}_p}(\text{Concat}([\boldsymbol{V}_p^t, \boldsymbol{V}_{c_p}^t])) \quad \forall \boldsymbol{p} \in \boldsymbol{P}$
- $\boldsymbol{V}_p^{t+1} = \boldsymbol{V}_p^t + \tilde{\boldsymbol{R}}_p \quad \forall \boldsymbol{p} \in \boldsymbol{P}$

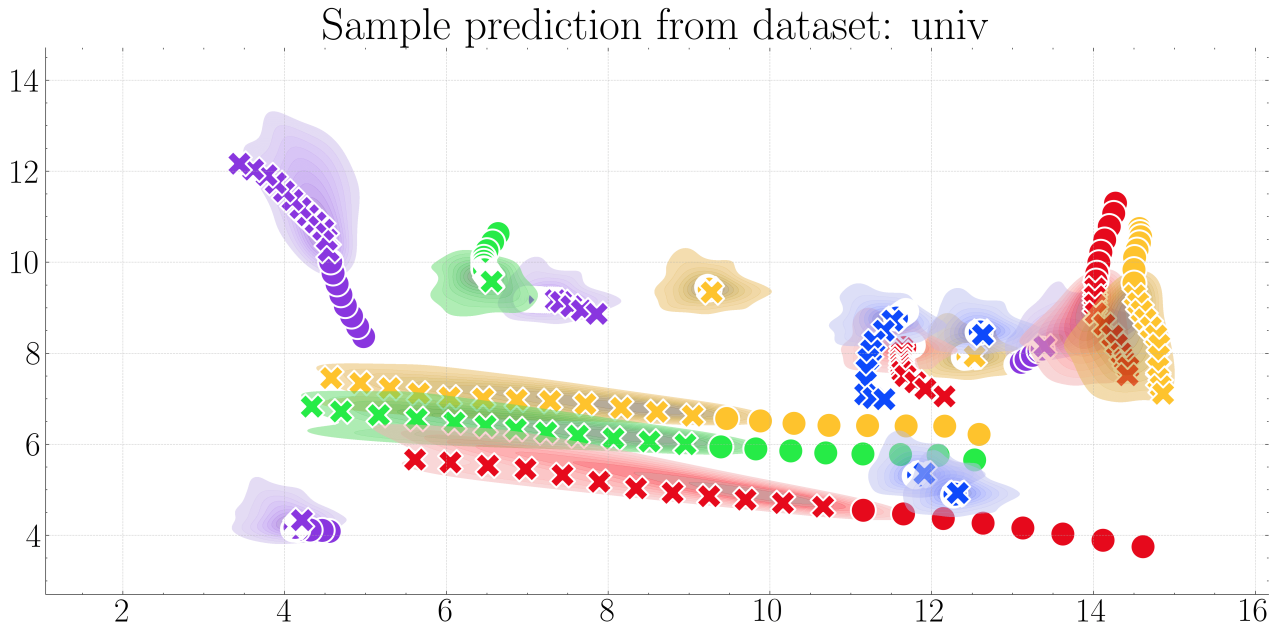
end

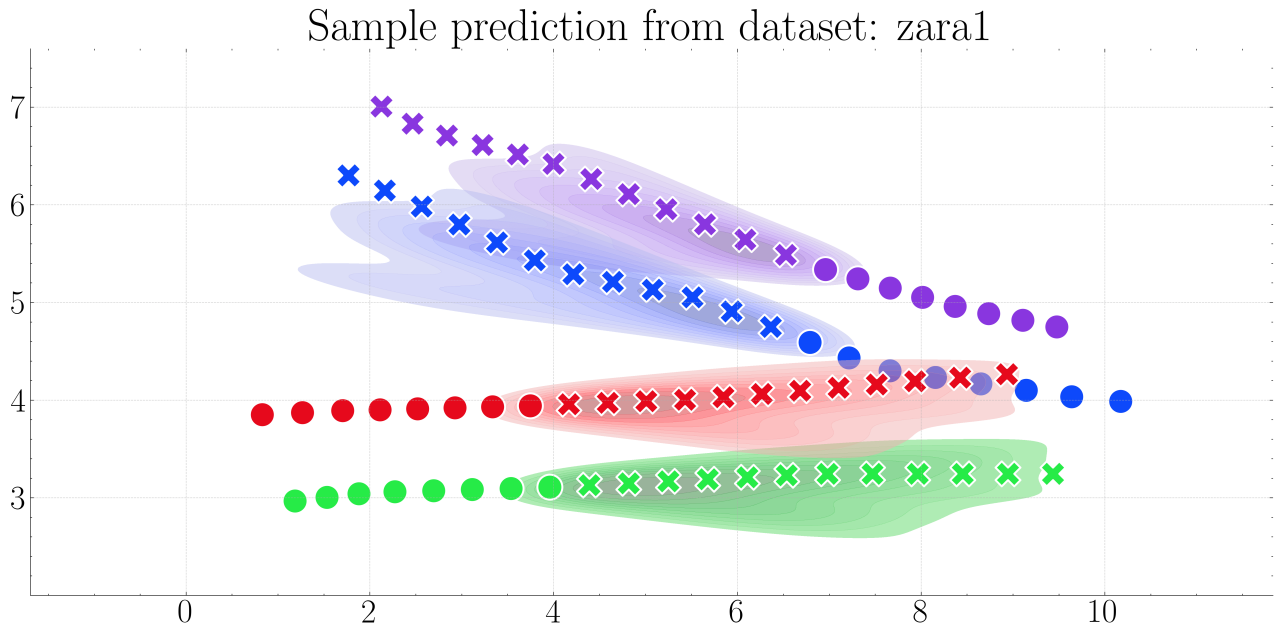
Appendix B Prediction visualization

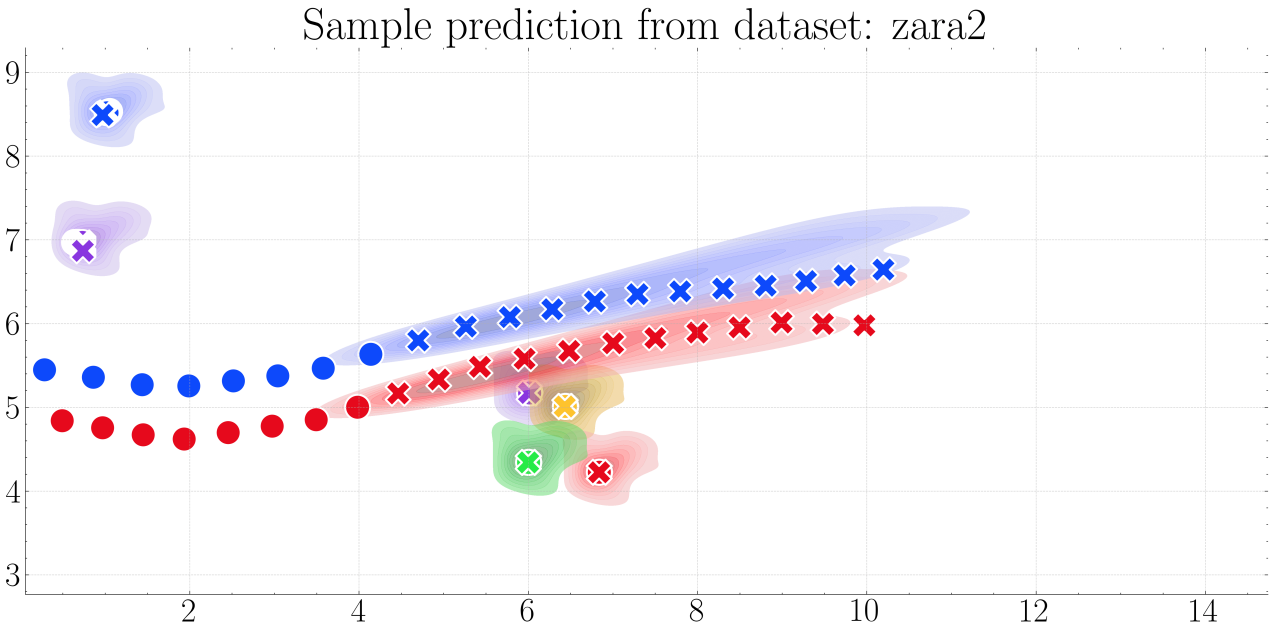
Additional prediction samples for each dataset. We visualize the 20 predictions as a distribution, while the golden trajectory as a scatter plot.

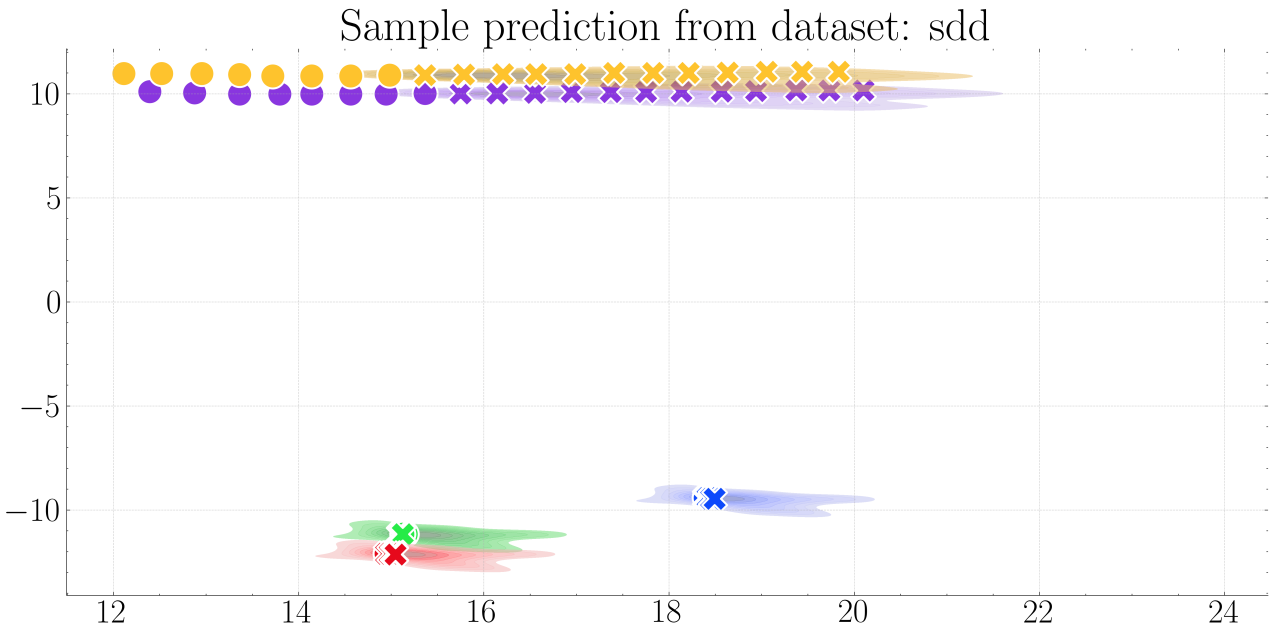





Sample prediction from dataset: univ
14
12
10
8
6
4
2
4
6
8
10
12
14
16


Sample prediction from dataset: zara1
7
6
5
4
3
0
2
4
6
8
10


Sample prediction from dataset: zara2
9
8
7
6
5
4
3
2
4
6
8
10
12
14


Sample prediction from dataset: sdd
10
5
0
−5
−10
12
14
16
18
20
22
24