DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Robotics, Cognition, Intelligence

Anomaly Detection for the behavior of drivers based on Structural Temporal Graph Neural Networks

Hanxi Jiang

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Robotics, Cognition, Intelligence

Anomaly Detection for the behavior of drivers based on Structural Temporal Graph Neural Networks

Erkennung von Anomalien im Verhalten von Autofahrern auf der Grundlage struktureller temporaler neuronaler Netze

Author:	Hanxi Jiang
Supervisor:	Supervisor
Advisor:	Advisor
Submission Date:	Submission date

I confirm that this master's thesis in informatics: robotics, cognition, intelligence is my own work and I have documented all sources and material used.

Munich, Submission date Hanxi Jiang

Acknowledgments

Abstract

Kurzfassung

Contents

1. Introduction

Despite the fact that High Driving Automation(SAE Level 4) is achievable in the foreseeable future[1], the majority of drivers nowadays would still prefer comprehensive control over their own vehicles. Therefore, increasingly more studies have been focusing on driving safety[2, 3]. According to these papers, driver behavior represents the majority cause of accidents while driving. In that case, several methods have been developed to avoid potential danger. Such methods involve either improving monitoring of the vehicle's inside situation, which analyzes the driving parameters as well as the driver him or herself to determine whether there's no abnormality[4], or proposing a vehicle detection and tracking system from an outer view that estimates time-to-collision (TTC) and warn the driver for a possible collision[5].

On the other hand, only few research focus on driving behavior prediction, which may be due to the lack of application for the undetermined decision model in behavior prediction, specifically under the topic of autonomous driving. Scientists have succeeded in years of generating dynamic graphs based on videos. However, there is still a gap in the rational use of these forms for behavioral prediction.

In this work, we would like to focus on constructing a comprehensive behavior prediction model based on graph neural networks (GNNs). A Graph Neural Network is a novel type of neural network architecture that can be applied to graph-like inputs. As we are now expecting to train the behavior model for the participators, the training data would contain interaction between several objects and participators while they are driving. GNN is, therefore, prioritized due to its unique structure. Along with that, we would specifically focus on building dynamic graphs as training datasets, as our model would be trained based on sequences of behaviour descriptions extracted from videos. Besides, to ensure the compatibility of the dataset and training model, we made some adaptions based on JODIE[6], which is the model based on the dynamic evolution of users and items. In order to make the predictions as detailed as possible, we expanded the output catalog to ensure that not only the interaction itself but also the type of it would be described.

In a nutshell, We would first gather all the necessary information with the help of the Large language model (LLM), convert it into dynamic graphs, and insert these graphs into the model we have adapted from model JODIE. These graphs could contain either one specific participant or all the concerned people. By learning how dynamic graphs change under time series, like the appearance and vanish of all these edges and nodes in the graph, the model should be able to absorb the feature behind it and come up with the prediction for behaviour in the future. Therefore, anomaly detection could also be achieved by comparing the predicted graph with the real one and alerting once when any unsuitable behaviour during driving is detected. we believe that this model provides a new perspective for the

prediction of driving behavior detected from videos and allows for more diversified and targeted forecasting.

1.1. contribution

The main contributions of this thesis are summarized as:

Dataset Collecton From the Dataset *drive & act*, we acquire the hierarchical activity labels of given video data and rewrite them in the form of time sequences. We also cluster the behavior types into several categories with the help of the Large Language Model.

Dynamic Graph Construction By reassembling the nodes and edges in the video data, we construct time-sequenced dynamic graphs that could be used as training data for the model. Such graph captures complex dependencies and offers hierarchical representation abstracted from the raw data.

Learning Model Adaption To enrich the diversity of the prediction, we expand the output of the dynamic graph based learning model from binary to catalog description, to ensure that not only the interaction itself but also the type of it would be described.

anormaly detection By comparing the predicted model with the real one, we could detect any unsuitable behavior during the driving and alert the driver. Differ from the research before, our

1.2. Structure

This thesis is structured as follows. In Chapter2 we would introduce and explain concepts and definitions concerned this document. Chapter3 reviews related work in the field of anomaly detection and dynamic link prediction models, the dataset this work refers to and the model we have adapted. Chapter4 describes the methodology of this work, including the dataset collection, dynamic graph construction and model adaption. Chapter5 presents the evaluation of the model and the results of the prediction. Chapter6 discusses the potential future work that could be done based on this work. Chapter7 concludes the thesis and gives a summary of the work done.

2. Background

2.1. Large Language Model

Large Language Models (LLMs) are highly complex artificial intelligence systems that can learn from the vast amounts of available text data[7]. Thanks to the attending of *Transformer*[8], a deep learning architecture, these language models which employed self-supervised pre-training have demonstrated improved efficiency and scalability in many fields.

Based on self-attention mechanisms and feed-forward module, *Transformer* has overwhelming advantages in computing representations and global dependencies.

In concrete terms, the Large Language Models equipped with *Transformer* are capable of diverse tasks raised by Natural Language Processing[9], such as textual entailment, question answering, semantic similarity assessment, and document classification. Take BERT[10] and GPT[7, 11, 12] as two examples, the former utilizes transformer encoder blocks to predict missing words in a given text, and the latter has been enjoying a tremendous reputation for generating diverse and human-like responses, showcasing its potential in various domains.

2.1.1. Text Classification

In our project, a bunch of hierarchical activity labels would be acquired from the dataset *drive & act* to depict every detail of the participant's movements in the driving behavior recorded in the video. These labels, however, are too trivial for the construction of learning data, as considering each activity individually will be tedious in such a vast and complex model training process. Therefore, labels should be classified according to the object on which this behavior operates or the specificity of the moment in which the action takes place. For example, the fastening of a seat belt should occur shortly after entering the vehicle, and all behavior related to eating or drinking should be classified into the same group.

In our task, we use zero-shot text classification. This is a task where a model is trained on a set of labeled examples and then classifies new examples from previously unseen classes. This method, which leverages a pre-trained language model, can be thought of as an instance of transfer learning which generally refers to using a model trained for one task in a different application than what it was originally trained for. This is particularly useful for situations where the amount of labeled data is small, for example, our work with 39 different behaviors to be classified.

The model used in the pipeline is *BART* [13]. According to the paper, BART is trained by corrupting text with an arbitrary noising function and learning a model to reconstruct the original text. It uses a standard Tranformer-based neural machine translation architecture, which, despite its simplicity, can be seen as generalizing BERT (due to the bidirectional

encoder), GPT (with the left-to-right decoder), and many other more recent pre-training schemes. With all these prerequisites, it is quite obvious that *BART* is a very practical model for our classification task.

2.2. Hidden Markov Model

Hidden Markov Model(HMM) [14, 15] is one of the earliest probabilistic models for sequential data analysis. It is a Markov model in which the observations are dependent on a latent (or hidden) Markov process. A detailed introduction could be found in appendix A. HMM do have wide applications in fields such as early speech recognition, part-of-speech tagging, and bioinformatics, where sequences of discrete states and short-term dependencies were dominant. However, limits such as assuming a fixed number of discrete states and being limited by the first-order Markov property (current state only depends on the previous state) could draw back the results when too much feature information is involved. In this work we use HMM as the baseline of our model in order to compare the performance of our model with the traditional one.

2.3. Graph

In graph theory, a graph is a structure made up of a collection of objects, where certain pairs of these objects are connected in a specific way[16]. As a powerful tool for modeling and analyzing complex systems, researchers have combined graph theory with deep learning to solve various problems in different fields, such as social networks[17], complex physic system[18] and Protein-protein interactions (PPIs)[19].

A Graph is $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a pair of sets, where collections of *nodes* $\mathcal{V}$ and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ between pairs of nodes. The nodes here are assumed as the nodes to be endowed with s-dimensional *node feature*, denoted by x_u for all $u \in \mathcal{V}$. Taking social network as an example, nodes represent users, and edges correspond to the friendship relations between them. The features of nodes model user properties such as age, likelihood, etc. Also, some models, including this work, would endow the edges with features. In generic setting $\mathcal{E} \neq 0$, the node features are rows of the $x \times d$ matrix $X = (x_1, \ldots, x_n)^T$. To apply some function to update the features in every node, obtaining the set of latent node feature, we would introduce permutation equivariant function $H = F(X)$, where H is the feature matrix whose uth row is the latent feature of node u.

As for the edges $e \in \mathcal{E}$, or the graph connectivity, they can be represented by the $n \times n$ adjacency matrix A defined as

$$a_{uv} = \begin{cases} 1 & (u, v) \in \mathcal{E} \\ 0 & otherwise \end{cases}$$

Here a_{uv} specifies the adjacency information between the nodes described by the uth and the vth rows of X. Most functions acting on graphs can be viewed as the generators for 'local' node-wise output, i.e., whereby the output on node u directly depends on its neighboring

nodes in the graph. It is worthwhile formalizing this constraint explicitly in our model construction by defining what it means for a node to be neighboring another. An (undirected) neighborhood of node u is defined as

$$\mathcal{N}_u = \{v : (u, v) \in \mathcal{E} \text{ or } (v, u) \in \mathcal{E}\}$$

and the neighborhood features as the multiset[20].

$$\mathcal{X}_{\mathcal{N}_u} = x_v : v \in N$$

Thus, the features of a node as well as its neighborhood could be collected simultaneously by a local function $\phi(x_u, \mathcal{X}_{\mathcal{N}_u})$. Then the permutation equivariant function F can be constructed by applying ϕ to every node's neighborhood in isolation (Figure 2.1).

$$F(X, A) = \begin{bmatrix} -- & \phi(x_1, \mathcal{X}_{\mathcal{N}_1}) & -- \\ -- & \phi(x_2, \mathcal{X}_{\mathcal{N}_2}) & -- \\ & \vdots & \\ -- & \phi(x_n, \mathcal{X}_{\mathcal{N}_n}) & -- \end{bmatrix}$$

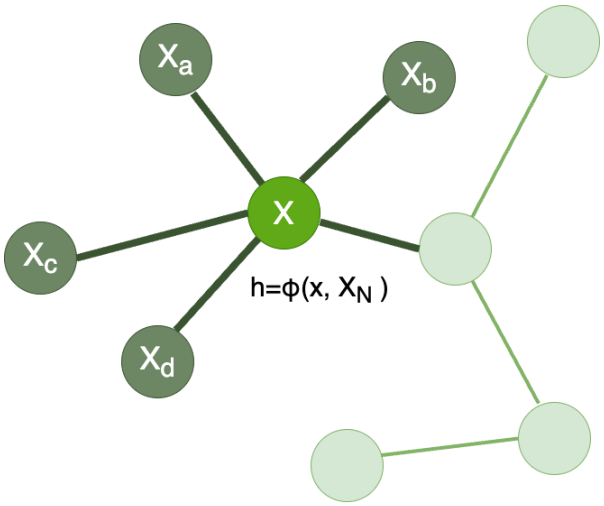

Figure 2.1.: Constructing permutation equivariant functions over graphs.

2.3.1. Graph Neural Networks

The intuitive understanding among **Graph Neural Network (GNN)** is that nodes in a graph represent objects or concepts, and edges represent their relationships. Each concept is naturally defined by its features and the related concepts[21]. GNNs are among the most general class of deep learning architectures currently in existence, and most learning architectures can be understood as a special case of the GNN with additional geometric structure.

Given to the mathematic expression we have settled in section 2.3 we consider a graph to be specified with an adjacency matrix A and node feature X. The GNN architectures

we study arepermutation equivariant functions $F(X, A)$ constructed by applying shared permutation invariant function $\phi(x_u, \mathcal{X}_{\mathcal{N}_u})$ over local neighborhood. This function ψ would be referred to as "diffusion", "propagation", or "message passing", and the all over computation of such of such F as a "GNN layer".

The design and study of GNN layers is one of the most active area of deep learning in the recent past few years. And three vast "flavours" of GNN layers are derived from the vast majority of the literatures. These flavours govern the extent to which ψ transforms the neighbourhood features, allowing for varying degrees of complexity when modelling interactions across the graph(Figure 2.2).

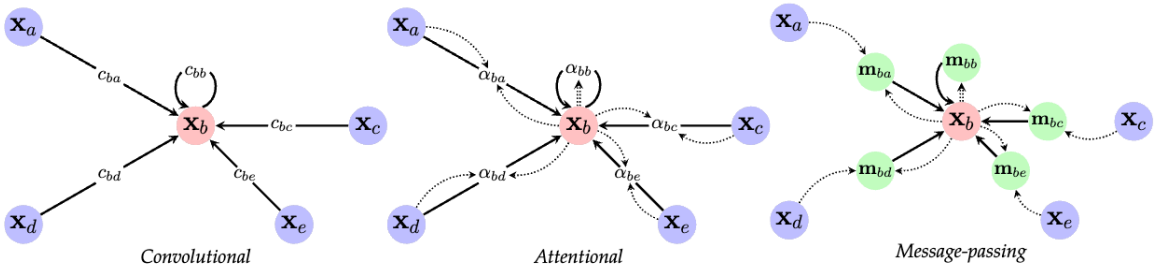

Figure 2.2.: A visualisation of the dataflow for the three flavours of GNNlayers. Reproduced from [20]

In all the three flavours, the feature update function for node u would apply permutation-invariant function $\oplus$ to aggregate features from $\mathcal{X}_{\mathcal{N}_u}$,as well as learnable function ψ. The function $\oplus$ is potentially transformed, by means of some function ψ and itself is usually realized as a nonparametric operation such as sum, mean, or maximum.

In the **convolution flavour**[22, 23, 24]The feature of the neighborhood nodes are directly aggregated with fixed weights c_{uv}, which often directly depends on the entries in A representingthe structure of the graph. The update function is defined as:

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} c_{uv}\psi(\mathbf{x}_v)\right).$$

In the **attention flavour**[25, 26, 27]the weights c_{uv} are replaced by the a learnable self-attention mechanism that computes the importancecoefficients. And the update function is defined as:

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} a(x_u, x_v)\psi(\mathbf{x}_v)\right).$$

And the **message-passing flavour**[28, 29] aims at computing arbitrary vectors of neighborhood $\mathcal{N}_u$ send to u, where vector-based messages are computed based on both the sender

and receiver: $m_{uv} = \psi(x_u, x_v)$.

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} \psi(x_u, x_v)\right).$$

What matters is that a representational containment between these approaches exists: $convolution \in attention \in message - passing$. This, however, does not mean that message-passing is always the most useful choice. Indeed, the vector-based messages always contains most oriented information across the edges, but they are harder to train due to the enormous requirement of memories. And here, in our work, we have such a specific network that none of the approaches above is the best choice.

2.3.2. Dynamic graph

Our discussion so far has focused solely on input that exhibits spatial variations across a given domain; however, the input may also change over time, for instance, video, text or speech. Assume that the input consists of arbitrary steps, and an input signal will be provided at each step t, we represent as $X^{(t)} \in \mathcal{X}(\omega^{(t)})$.

While in many cases the domain in kept fixed across time t, we notice that exceptions also happens. For example in our work, we many find tremendous changes in the videos from our dataset. Such domain as changing over time is referred to as *dynamic graph*[30, 31].

Assume an encoder function $f(X(t))$ offering latent presentation for such dynamic graph learning problem. In the video analysis, ω is a fixed grid, and signals are a sequence of frames. To have a view of the entire frame, one of the options is to implement f as a translation invariant CNN, outputting a k-dimensional representation $z(t) = f(X(t))$ of the frame at time-step t. In our work, however, we use an alternative approaches, which we would introduce in chapter3.

To have a canonical analysis for dynamically aggregate the sequence of vectors $z(t)$, we introduce *Recurrent Neural Network* (RNN)[32] due its advantages in the field of temporal progression of inputs and online arrival of novel data-points. A clear view of analyzing video data with RNN is given in Figure 2.3.

A simple RNN model is defined as the following way: The model holds with an m-dimensional summary vector $h(t)$ of all the input steps up to and including t. This will summarize the current step's features as well as the information collected from the past, through a shared function $R : \mathbb{R}^k \times \mathbb{R}^m \to \mathbb{R}^m$, as follows:

$$h(t) = R(z(t), h(t-1))$$

As both $z(t)$ and $h(t-1)$ are vectors, the function R is usually realized as a simple feed-forward neural network(often known as *Simple RNN*[33, 34]):

$$h(t) = \sigma(W_z z(t) + U h(t-1) + b)$$

Where W_z and U are the weight matrices, b is the bias, and σ is the activation function.

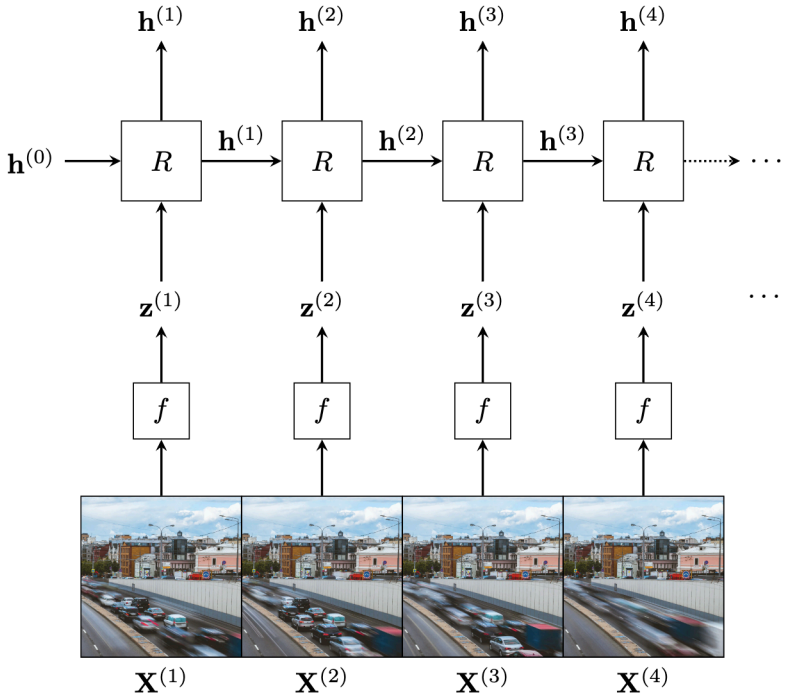

Figure 2.3.: Illustration of processing video input with RNNs. Reproduced from [20]

The summary vectors may then be appropriately leveraged for the down-stream taskif a prediction is required at every step of the sequence, then a shared predictor may be applied to each $h(t)$ individually. For classifying entire sequences, typically the final summary, $h(T)$, is passed to a classifier, just like many works including ours.

2.4. behavior prediction for human driver

The earliest research regarding the prediction of human driver behavior could be dated back to the 1990s, when hidden Markov models are first introduced to model the driver's behavior[35]. An HMM is a statistical model that represents a system with hidden states, where observable outputs are generated by these underlying, unobservable states. Each state transitions to another with a certain probability and emits observations based on specific probability distributions.

The paper is constructed on the ground truth that the intended actions, like to turn or change lanes, are modeled as a sequence of internal states. By observing the temporal patterns of the driver's behavior or even intention. Although the attempt then is to capture and predict driver eye movements in the context of lane keeping/curve negotiation and car following to determine. It still proves that characteristic patterns in driver behavior could be identified and even represent by mathematical models.

With the advancement of various machine learning models, more sophisticated tools have emerged, expanding the range of options available for predicting human driver behavior. In [36], hidden Markov models are adapted for this topic as well, yielding promising results due to the enhanced computational capabilities available today. Another study [37] proposes an LSTM-based trajectory prediction method for human drivers, which facilitates better decision-making for autonomous vehicles, particularly in urban intersection scenarios. Despite these significant achievements, there remains a noticeable gap in driver behavior prediction for driver-oriented dynamic scene graphs. Such an approach could capture individual driving habits, enabling more precise predictions and even anomaly detection tailored to each driver. Our work aims to address this gap.

3. Related Work

3.1. Datasets for driver behavior analysis

As of the time of writing, numerous datasets are available for driver behavior analysis, each with its own specific focus. For instance, *DR(eye)VE Dataset*[38]capturing drivers' gaze patterns in real-world scenarios, includes ego-centric views and car-centric views, which provide data on where drivers look in different driving contexts. *Naturalistic Driving Study (NDS)*[39] contains extensive data collected from naturalistic driving conditions, including in-vehicle driver behavior such as driver movements, eye gaze, and interactions with vehicle controls during different driving scenarios. Detailed information can be found in A

In our work, we utilize the **Drive & Act** dataset[40] as our primary training resource. This dataset includes over twelve hours and 9.6 million frames, capturing individuals engaged in various distractive activities during both manual and automated driving. It is specialized for distinguishing between closely related actions (e.g., opening a bottle vs. closing a bottle) and features a high diversity in action durations and complexities, presenting unique challenges for action recognition models. For instance, brief actions like opening a door from the inside may take less than a second, while prolonged activities such as reading a magazine may last several minutes. This dataset is exceptionally well-suited to our study, as it provides a comprehensive view of driver behavior within the vehicle, with accurately labeled, temporally sequenced annotations for each behavior.

3.2. Dynamic Scene Graph for Video

As is revealed in chapter 2, the most common way to represent the latent information in a video is to use CNNs to directly extract the features of the frames and then use RNNs to model the temporal information. However, this method has its limitations as the excessive high-dimensional abstraction may cause the model to misinterpret human learning intentions; in other words, the black box itself is difficult to interpret. In our work we would like to extract the dynamic scene graph from the video, which is a more intuitive way to represent the video content.

The scene graph is a structured representation of a scene that can clearly express the objects, attributes, and relationships between objects in the scene[41]. Accompanied by the development of computer vision technology, simply detecting and recognizing objects in images no longer satisfies the researchers, as they would expect some higher level of understanding and reasoning for image vision tasks. In this way, an intuitive idea comes up about adding up the relationship between the detected objects (See example in figure3.2). The

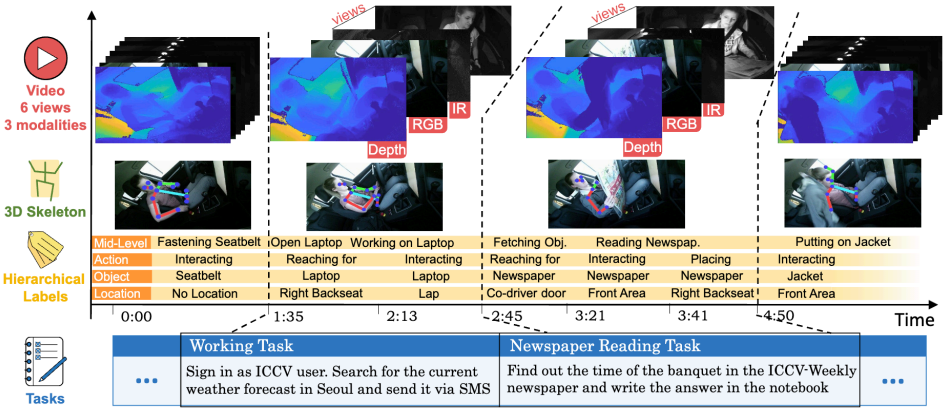

Figure 3.1.: Overview of the Drive&Act dataset for driver behavior recognition. Reproduced from[40]

earliest research could date back to 2017, when some objects and relations of a given image could be inferred, and a scene graph would be produced as a result[42]. Other research like Neural Motifs[43] also shows the possibility of predicting the most frequent relation between object pairs with the given labels and object detections. Later in 2018, videos came into discussion, and both spatial and temporal relations would be concerned in the dynamic graph researchers propose to represent[44]. Meanwhile, the accuracy of Scene Graph Detection tasks has significantly improved thanks to the application of unbiased SGG[44], fueled by the *Detection2* [45], a library that contains various state-of-the-art detection and segmentation algorithms. In a nutshell, representing videos as dynamic scene graphs including the detection of objects and the relations in between has been realized in the past years. And our work would utilize such technique and extract our driver-oriented dynamic scene graphs for further learning.

3.3. Dynamic Link Prediction

After extracting the dynamic scene graph from video data, the subsequent phase involves learning the interactions between elements within the scene to enable accurate behavior prediction. This objective can be conceptualized as a link prediction task within a dynamic scene graph, making the selection of an appropriate model critical to the success of our approach. Leveraging the review provided in *DGB* [47], we have identified and evaluated various link prediction models along with effective evaluation strategies. Below, we introduce each model and provide a theoretical explanation for our choice.

JOINT DYNAMIC USER-ITEMEMBEDDING MODEL(JODIE)

Based on the user-item interaction data, **JODIE**[6] works as a coupled *Recurrent Neural Networks(RNN)*. The update operator alternatively updates the embedding space of suer and an

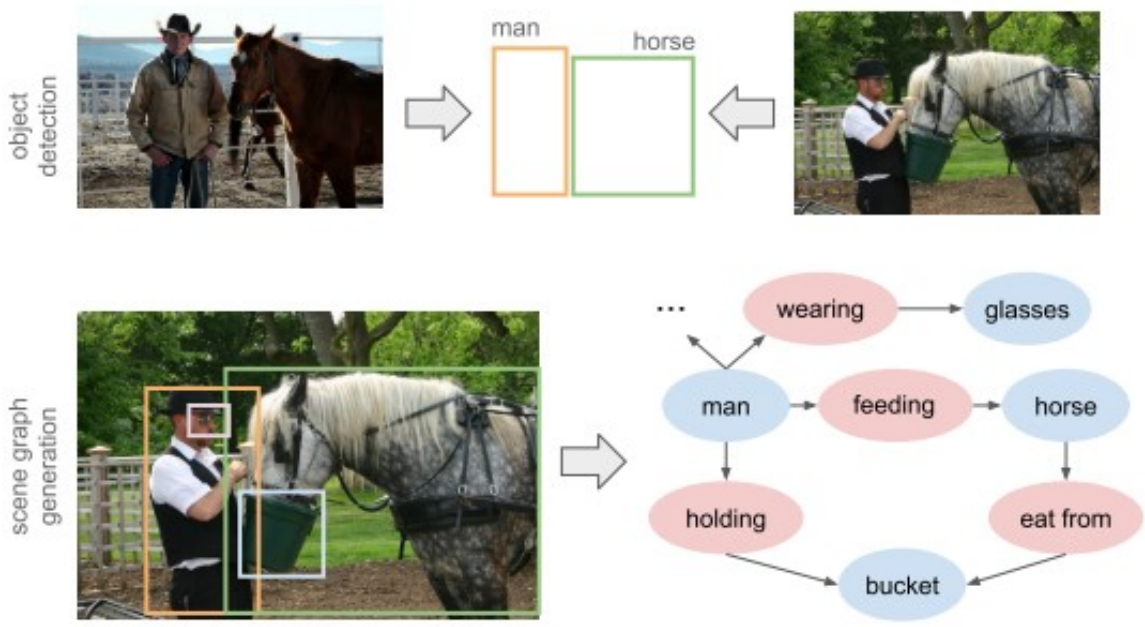

Figure 3.2.: Scene Graph Generation by Iterative Message Passing. Reproduced from[46]

item at each interaction. And the projection operator learns to estimate the embedding of the user at any time in the future, thereby enabling the prediction of forthcoming interactions. The design and functionality of JODIE align well with our objectives, making it a strong candidate for our work.

Representation Learning over Dynamic Graphs(DyRep)

DyRep[48] is an inductive deep representation learning framework that learns a set of functions to efficiently produce low-dimensional node embeddings that evolves over time. The model captures all the new edges appears during the time sequence and updates the embeddings of the nodes accordingly in a custom RNN model. The final output provides a solution to the problems of dynamic link prediction and event time prediction. TGN achieves superior performance over prior models and offers computational efficiency, making it a potential fit for our data, although some constraints exist.

Temporal Graph Network(TGN)

TGN[30] is a generic, efficient framework for deep learning on dynamic graphs represented as sequences of timed events. To train the Memory-related modules, the message matrix would begin with a raw message store and is updated based on interactions which have been processed by the model in the past, which would also propagate to the node embedding as well as the loss function. TGNs significantly outperform previous approaches while being more computationally efficient, Which shows the possibility of applying on our data.

Causal Anonymous Walks Network (CAW-N)

Based on *temporal random walks*, **CAW-N** adopt a novel anonymization strategy that replaces node identities with the hitting counts of the nodes based on a set of sampled walks to keep the method inductive, and simultaneously establish the correlation between motifs. It is specialized in work as automatic retrieval of temporal network motifs to represent network dynamics. Since our dataset has only little similarity with random walking, we didn't take this into consideration at last.

After evaluating each of these models, we selected **JODIE** as the final model for our work. Theoretically, this choice is justified by the structural similarity of our dataset, which features participant-object interactions akin to user-item data. Although **TGN** also offers a sophisticated architecture, its need for extensive training data rendered it impractical for our dataset's constraints.

4. Methodology

This work aims to construct a graph neural network based architecture for predicting, analyzing, and detecting any potentially abnormal behavior regarding the driver during the whole driving process. In particular, The model extracts a description graph, the so-called scene graph, of the driver from the video filmed inside the vehicle and trains itself with these data to learn for future behavior prediction. The result will be used to compare and detect any abnormal behavior. Here we would lay most emphasis on the construction of the training model. To make precious anomaly detection we aim to predict not only if there is a behavior between humans and a specific kind of object but the type of behavior as well, which will cause several adaptions based on existing model *JODIE*.

This chapter will follow the pipeline of the whole work, as shown in figure 4.1. Progressively we would introduce the scene graph generation, the model architecture, and the training process.

4.1. Scene graph generation

The *Drive & Act* dataset comprises 12 hours of video data segmented into 29 lengthy sequences. For each sequence, we aim to generate a unique dynamic graph that temporally represents both the participant's actions and the relationships between objects observed in the video. This process involves several key steps.

First, by sliding through the video at intervals of 1/15 second, we obtain a series of images. Theoretically, each image could yield a corresponding graph with the use of an object detection model [46]. Given that our objective is to predict driver behavior, we concentrate on the driver and the specific objects with which they interact. Consequently, we opt for a streamlined approach by directly extracting data from hierarchical labels within the dataset. Table 4.1and table 4.2 provide an overview of the hierarchical labels. The former one provides the concrete activity descriptions of the driver in specific time frames, while the latter one outlines the objects the driver interacts with, along with the corresponding locations where these interactions occur..

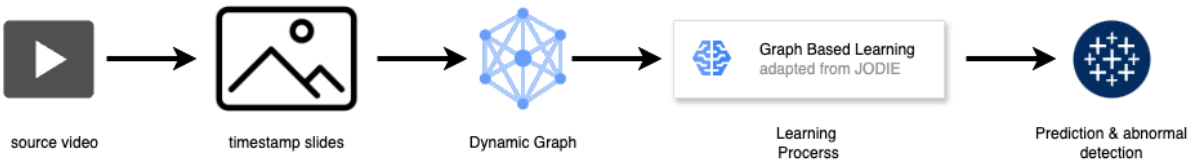

Figure 4.1.: Pipeline of the whole work

file_id	frame_start	frame_end	activity	chunk_id
vp1/run1b.kinect_color	40	58	standing_by_the_door	0
vp1/run1b.kinect_color	58	82	closing_door_outside	0
vp1/run1b.kinect_color	83	102	standing_by_the_door	0
vp1/run1b.kinect_color	102	130	opening_door_outside	0
vp1/run1b.kinect_color	130	156	entering_car	0
vp1/run1b.kinect_color	156	174	closing_door_inside	0

Table 4.1.: Example of the task level hierarchical labels Reproduced from[40]

Start Time	End Time	Action	Object	Location
175	188	reaching_for	seatbelt	left_backseat
189	208	retracting_from	seatbelt	left_backseat
208	230	interacting	seatbelt	no_location
240	258	reaching_for	multimedia_display	center_console_front
3011	3031	reaching_for	automation_button	center_console_back
3031	3056	retracting_from	no_object	center_console_back
3115	3144	reaching_for	multimedia_display	center_console_front
3144	3156	retracting_from	no_object	center_console_front
3311	3357	reaching_for	no_object	right_backseat

Table 4.2.: Example of the object level hierarchical labels Reproduced from[40]

However, these tables alone are insufficient for generating graphs, as object detection at each individual timestamp is required to ensure data completeness. In another word, the row *start time* and *end time* will be replaced by *Timestamp*. To address this, we construct the dynamic graph by processing each timestamp sequentially. Specifically, the participant is designated as the node u, and all relevant objects are represented as nodes i. By analyzing interactions at each timestamp, edges are added between the participant node and the corresponding object nodes to represent the associated actions. Additionally, the features of the edges are enriched with the location where each action occurs. The graph is iteratively updated with each subsequent timestamp, ultimately resulting in a dynamic graph that captures the temporal evolution of interactions.

Another key consideration in graph generation is the classification of activity labels. The raw dataset contains 39 distinct behaviors, which would be computationally intensive and overly too trivial to label directly. To address this, we employ *BART*, a denoising autoencoder designed for pretraining sequence-to-sequence models, to cluster these behaviors into four

categories. This classification is based on two key questions: whether the behavior is related to driving and whether it implies any potential risk, as the ultimate goal is to identify and flag abnormal behaviors. The resulting behavior classifications are presented in Table 4.3.

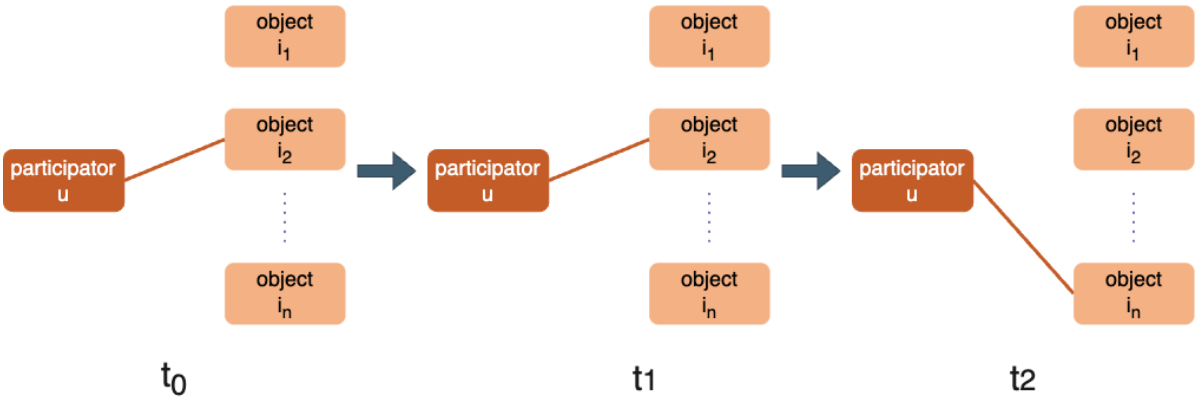

Figure 4.2.: Dynamic Graph Generation for Driver Behavior Recognition

Through the processes outlined above, all dynamic graphs have been successfully generated, as illustrated in Figure 4.2. To streamline subsequent learning steps, these graphs are stored in a tabular format, as shown in Table 4.4.

As illustration of heads of this table is as following:

u The action initiator or generator, in our work typically representing the participant or driver.

i The object or target that is acted upon or influenced by the action.

timestamp The specific moment or time frame at which the action or interaction occurs.

stable label A stable vector that characterizes the consistent attributes or categorical label associated with the edge.

feature A weight vector representing the proportion of occurrences for a particular attribute (in this case, the location column), reflecting the relative frequency or appearance percentage of this attribute over time.

4.2. Baseline: Hidden Markov Model (HMM)

The Hidden Markov Model (HMM) is a suitable baseline for our driver behavior prediction task due to its effectiveness in modeling temporal sequences with underlying hidden states, allowing it to capture the evolving patterns of driver behavior over time. HMMs provide a probabilistic framework that handles uncertainty in sequential data, making them a widely adopted baseline for tasks involving sequence prediction, including applications in graph

No.	State	Behaviors
1	**Driving concerned behaviors**	standing by the door, closing door outside, opening door outside, entering car, closing door inside, fastening seat belt, moving towards door, unfastening seat belt, opening door inside, exiting car
2	**Independent behaviors**	sitting still, looking or moving around (e.g. searching), looking back left shoulder, looking back right shoulder
3	**Eat or drink concerned behaviors**	preparing food, eating, opening bottle, drinking, closing bottle
4	**Other object concerned behaviors**	using multimedia display, sitting still, using multimedia display, pressing automation button, fetching an object, opening laptop, working on laptop, interacting with phone, closing laptop, placing an object, putting on jacket, taking off sunglasses, putting on sunglasses, reading newspaper, writing, talking on phone, reading magazine, taking off jacket, opening backpack, closing backpack, putting laptop into backpack, taking laptop from backpack

Table 4.3.: Behavior classification

u	i	timestamp	state label	feature
1	2	127	0	0.27

Table 4.4.: Storage of dynamic graph

neural network (GNN) models. In our work, this baseline serves as a foundation for comparison, particularly given that our hybrid model does not align directly with more complex modern models, which may lack sufficient parallels for a direct comparison.

To perform the prediction task, we first extract the temporal sequence from the dynamic graph data. In this context, the state represents a classification of similar behaviors observed within the dataset, while the features encapsulate consequential representations that may influence future observations. Thus, we set the value *state label* as the observation sequence and the *feature* as the hidden state, see in figure4.3.

With the help of the library *hmmlearn*, we can train the model with the extracted data and predict the state of the next timestamp. Comparing the predicted state with the actual state, we can evaluate the performance of HMM. And the result will be used as a baseline for the following models.

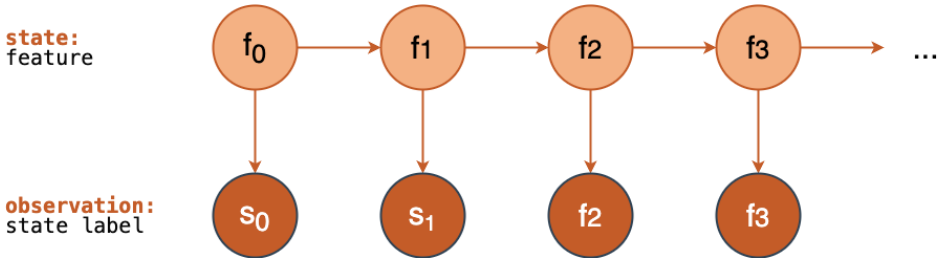

Figure 4.3.: HMM Model Architecture

4.3. Training Model Architecture

4.3.1. Introduction to JODIE

As is mentioned in chapter 3, the model *JODIE* is a coupled recurrent neural network model that learns the embedding trajectories of users and items. Before we start our own contribution to the model, some introduction to the original model is necessary.

The model *JODIE* is aimed at learning the embedding trajectories of users, denoted as $\mathbf{u}(\mathbf{t}) \in \mathbb{R}^n$ for each $u \in \mathcal{U}$ and item, denoted as $\mathbf{i}(\mathbf{t}) \in \mathbb{R}^n$ for each $i \in \mathcal{I}$, over a time interval $\forall t \in [0, T]$. These embeddings are derived from an ordered sequence of temporal user-item interactions $S_r = (u_r, i_r, t_r, r)$, which indicates that an interaction S_r involving a user $u_r \in \mathcal{U}$ and an item $i_r \in \mathcal{I}$ at time $t_r \in \mathbb{R}^+$, in the period $0 < t_1 < t_2 < \cdots < T$. In our study, the *user* corresponds to the driver, and the *item* to the object of which the action will take place. The process of interaction thus represents the drivers behavior toward the object over time.

To encode both long-term stationary properties and the dynamic evolution of user-item interactions, the embedding contains **static and dynamic embeddings**. Static embeddings $\overline{\mathbf{u}} \in \mathbb{R}^d \ \forall u \in \mathcal{U}$ and $\overline{\mathbf{i}} \in \mathbb{R}^d \ \forall i \in \mathcal{I}$ keep constant over time, with one-hot vectors representation similar in LSTM[49] and TimeAware-LSTM[50]. In contrast, the dynamic embeddings are designed to evolve with each interaction: each user and item is associated with a dynamic embedding, $\mathbf{u}(\mathbf{t})$ and $\mathbf{i}(\mathbf{t})$, respectively. These dynamic embeddings change over time, producing what is referred to as an *embedding trajectory* that reflects the historical sequence of interactions.

4.3.2. Core Components of JODIE

The model could be separated into two parts: the **update operation** and the **projection operation**. The former update the embeddings of the user and the item, while the latter uses the previous observed state and the elapsed time to predict the future embedding of the user. The figure 4.4 illustrates the architecture of the model.

In the **update operation**, the interaction $S = (u, i, t, f)$ between a user u and item i at time t is used to generated their dynamic embeddings $\mathbf{u}(\mathbf{t})$ and $\mathbf{i}(\mathbf{t})$. Two recurrent neural networks RNNs are involved in this process, named RNN_U and RNN_I. They recursively update the embeddings of the user and the item. This structure offers significant advantages over traditional one-hot vector representations, as the dynamic embedding reflects the items

current state. This leads to more meaningful user embeddings and streamlines the training process. The dynamic embeddings are updated as follows:

$$\mathbf{u}(\mathbf{t}) = \sigma(W_1^u \mathbf{u}(\mathbf{t}^-) + W_2^u \mathbf{i}(\mathbf{t}^-) + W_3^u \mathbf{f} + W_4^u \Delta_u)$$
$$\mathbf{i}(\mathbf{t}) = \sigma(W_1^i \mathbf{i}(\mathbf{t}^-) + W_2^i \mathbf{u}(\mathbf{t}^-) + W_3^i \mathbf{f} + W_4^i \Delta_i)$$

where Δ_u denotes the time since the last interaction of user u and Δ_i denotes the time since the last interaction of item i. $\mathbf{f}$ is the representation of the interaction, in another word, the feature vector. All the matrices $W_1^u \dots W_4^u$ and $W_1^i \dots W_4^i$ are parameters belong to RNN_U and RNN_I respectably, they will be trained to predict the embedding of the item at u's next interaction. σ is a sigmoid function to introduce non-linearity.

In the **projection operation**, the future trajectory of the user embedding is predicted. This projection occurs at a specified time interval, δ after the last interaction at time t, when the user's embedding is update to $u(t + \Delta)$. In order to effectively embed time-sequence data, the method *LatentCross*[51] is introduced, which creates a temporal attention vector. Firstly, Δ needs to convert into a time-context vector $\mathbf{w} \in \mathbb{R}^n$ using a linear layer $\mathbf{w} = W_p\Delta$, and W_p is a learnable vector initialized from a zero-mean Gaussian distribution. The projected embedding is then derived by taking the element-wise product of the time-context vector and the previous embedding as follows:

$$\hat{u}(t + \Delta) = (1 + \mathbf{w}) * \mathbf{u}(\mathbf{t})$$

Here the vector $1 + \mathbf{w}$ is a temporal attention to scale the past user embedding. According to the original *LatentCross* work, a linear layer is optimal for projecting the embedding, as introducing non-linearity can distort the embedding. Consequently, we retain this linear layer in our implementation.

4.3.3. Training Process of JODIE

Let u interact with item i at time j and then with item j at time $t + \Delta$. Right before $t + \Delta$, can we predict which item u will interact with? We use this task to train the update and projection operations in JODIE. We train JODIE to make this prediction using us projected embedding $\hat{u}(t + \Delta)$. Rather than computing the interaction probability between each user u and item i, *JODIE* directly outputs an item embedding vector $\tilde{j}(t + \Delta)$. This approach allows *JODIE* to make predictions with a single forward pass of the prediction layer, significantly reducing computational demands during inference by bypassing additional neural network passes. Using Locality Sensitive Hashing (LSH) techniques [52], we can then retrieve the item with the closest embedding in near-constant time.

Thus, J is trained to minimize the L_2 difference between the predicted item embedding $\tilde{j}(t + \Delta)$ and the real item embedding $[\bar{j}, j(t + \Delta^-)]$. The latter is a concatencation of the static embedding $\bar{j}$ and the dynamic embedding $j(t + \Delta^-)$ that immediately before time $t + \Delta$. The loss function is defined as follows:

$$\|\tilde{j}(t + \Delta) - [\bar{j}, j(t + \Delta^-)]\|_2$$

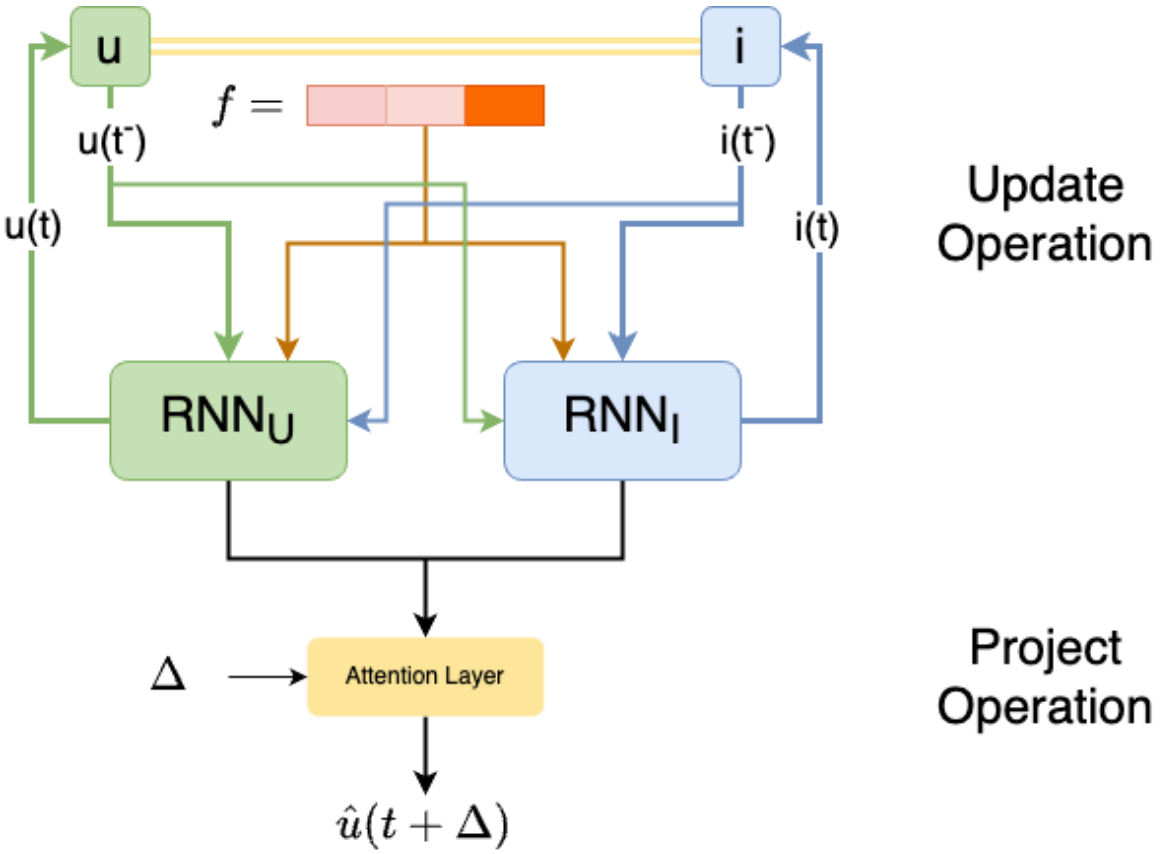

Figure 4.4.: JODIE Model Architecture

As for the prediction, it is mad using a fully connected linear layer as follows:

$$\tilde{j}(t+\Delta) = W_1\hat{\mathbf{u}}(\mathbf{t}+\boldsymbol{\Delta}) + W_2\bar{\mathbf{u}} + W_3\mathbf{i}(\mathbf{t}+\boldsymbol{\Delta}^-) + W_4\bar{\mathbf{i}} + B$$

This prediction leverages the projected user embedding, $\hat{u}(t+\delta)$ nd the embedding of the item from u's most recent interaction immediately before $t+\Delta$,denoted as $i(t+\Delta^-)$. The inclusion of $i(t+\Delta^-)$ is crucial, as users frequently interact with the same item consecutively (i.e.,$i = j$) , and incorporating the item embedding simplifies the prediction. This consideration is especially important in our work, where the scenario is based on the assumption that the driver interacts with the same object throughout the driving process. The matrices W_1, W_2, W_3, W_4 and B are learnable parameters, and the loss function is defined as the mean squared error between the predicted and actual item embeddings. Additionally, static embeddings are included to capture long-term properties of both the user and the item, ensuring a comprehensive prediction.

Finally, the total loss function is calculated as following:

$$\text{Loss} = \sum_{(u,i,t,f)\in S} \left\| \tilde{j}(t) - \left[\bar{i}, i(t^-)\right] \right\|_2 + \lambda_U \left\| \mathbf{u}(t) - \mathbf{u}(t^-) \right\|_2 + \lambda_I \left\| \mathbf{i}(t) - \mathbf{i}(t^-) \right\|_2 \tag{4.1}$$

The first term minimizes the L_2 distance between the predicted item embedding and the ground truth item embedding at each interaction. The last two terms serve as regularization, preventing excessive variation in consecutive dynamic embeddings of the user and item. Scaling parameters λ_U and λ_U ensure the losses are within the same range. Notably, negative sampling is not used during training to avoid overfitting.

In the original paper, the authors also propose an extension of JODIE for prediction tasks with categorical outputs rather than binary ones. In such cases, additional training labels are required, and the prediction function is redefined as $\mathbb{R}^{n+d} \rightarrow C$ enabling label prediction using the user embedding after an interaction. Here, the cross-entropy loss is calculated for categorical labels and incorporated into the overall loss function with an additional scaling parameter. The following section will discuss how this adaptation is implemented in our work.

4.3.4. Adapting JODIE to Enhance Interaction State Prediction

Returning to our initial goal, the primary objective is to predict the behavior of the driver. Therefore, it is essential to account for both the object the driver interacts with and the drivers behavior toward that object. In the original *JODIE* model, only the object is predicted as the node to which the edge will connect in the subsequent timestamp. However, in our approach, the behavior is represented as the classification of each edge connecting the user and item nodes in the dynamic graph, referred to as the *state* of each edge. Incorporating the *state* into the embedding space and modifying the loss function accordingly enables the prediction of behavior. The proposed adaptations are illustrated in Figure 4.5. To accomplish this, we explicitly add the interaction state into the embeddings of both the user and the item:

$$\mathbf{u}(\mathbf{t}) = \sigma(W_1^u \mathbf{u}(\mathbf{t}^-) + W_2^u \mathbf{i}(\mathbf{t}^-) + W_3^u \mathbf{f} + W_4^u s + W_5^u \Delta_u)$$

$$\mathbf{i}(\mathbf{t}) = \sigma(W_1^i \mathbf{i}(\mathbf{t}^-) + W_2^i \mathbf{u}(\mathbf{t}^-) + W_3^i \mathbf{f} + W_4^i s + W_5^u \Delta_i)$$

Similar to other matrices, W_4^u and W_4^i are also learnable parameters. Their values, denoted as s, have been enumerated in the previous steps and undergo a linear transformation before being incorporated into the embedding space. This adaptation enhances the representation of the embedding spaces for both the user u and the item i, providing additional information crucial for predicting the behavior state in future interactions. This modification is then integrated into the prediction function, which is updated as follows:

$$\tilde{j}(t+\Delta) = W_1 \hat{u}(t+\delta) + W_2 \bar{u} + W_3 i(t+\Delta^-) + W_4 \bar{i} + W_5 s + B$$

To meet the final goal of predicting the behavior state, the loss function's output of potential edge appearance should be converted from a binary classification(if the user will interact

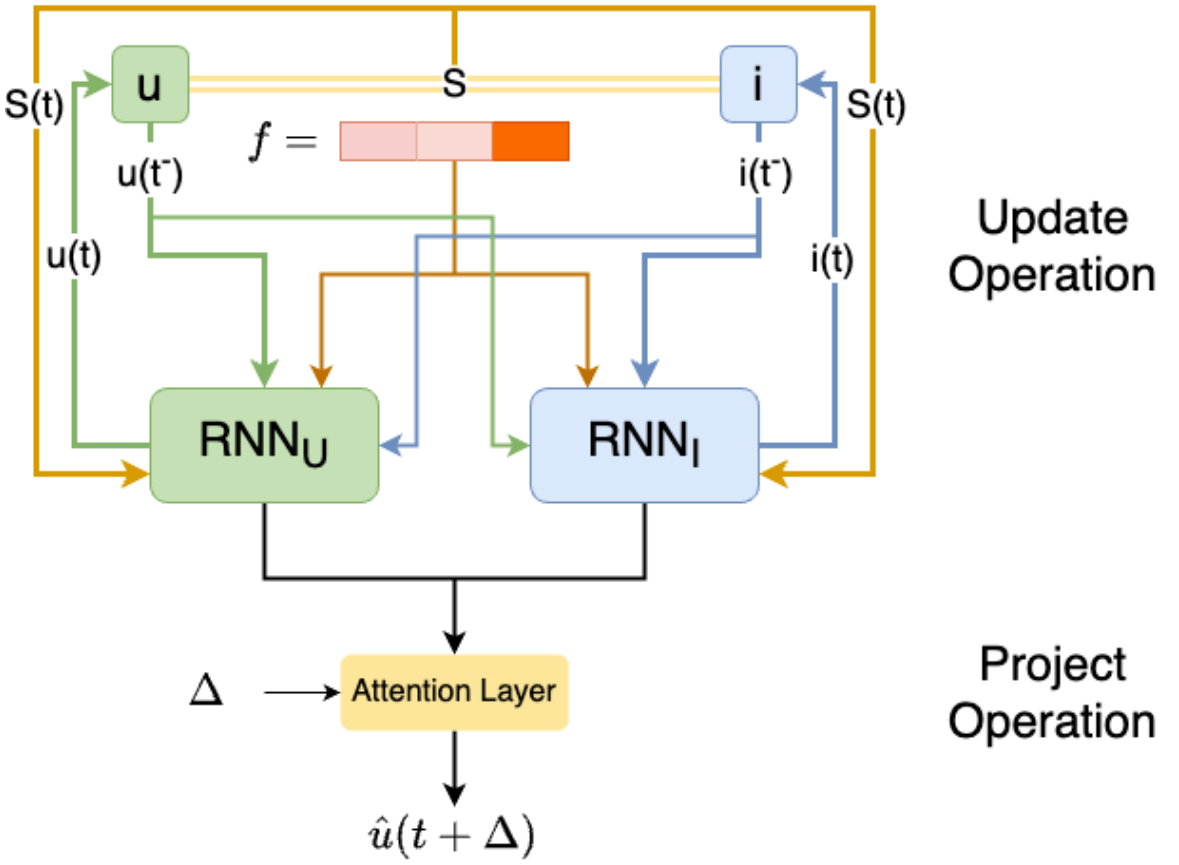

Figure 4.5.: JODIE Model Architecture with State Prediction

with the item) to a multi-class classification (either the user won't interact with this item, or one kind of interaction would be taken place here). In a mathematic representation, the domain of the output would convert from a binary set $\{0, 1\}$ to a set of n classes $0, s_1, s_2, s_3, s_4$. In this case, the cross-entropy loss function is used to calculate the difference between the predicted and actual behavior states.

Several steps are required to adapt the model for this task:

update the prediction Mechanism In the original model, predictions are made using a fully connected linear layer. However, in our case, the requirement is to determine the correct relationship between two nodes, $u \in \mathcal{U}$ and $i \in \mathcal{I}$. This relationship falls into one of five categories: no connection, or connections labeled with s_1 through s_4. Consequently, the embedding space needs to be transformed into a 5-dimensional vector. To extract the predicted probability $p(t)$, the output of the linear layer is passed through a softmax function, obtaining a probability distribution over the behavior states.

$$p(t) = softmax(W\tilde{j}(t) + b)$$

Here W is a learnable parameter matrix and b is a bias term, the probability distribution $p \in \mathbb{R}^{5\times n}$ to match the dimension of the output.

Cross-Entropy loss The Cross-Entropy loss for a single interaction is expressed as:

$$\text{Loss}_{CE} = -\sum_{k=1}^{5} y_k \log p_k(t)$$

where y_k is the ground truth label of the behavior state, and $p_k(t)$ is the predicted probability of the behavior state. The total loss is the sum of the losses for all interactions in the dataset.

Integrate into the overall loss function Incorporating this into the overall loss function, the updated objective becomes:

$$\text{Loss} = \sum_{(u,i,t,f)\in S} \text{Loss}_{CE}(t) + \lambda_U \left\|\mathbf{u}(t) - \mathbf{u}(t^-)\right\|_2 + \lambda_I \left\|\mathbf{i}(t) - \mathbf{i}(t^-)\right\|_2 \tag{4.2}$$

where $\sum_{(u,i,t,f)\in S}$ ensures accurate classification of the behavior, and the regularization terms, scaled by parameters λ_U and λ_I control the smooth evolution of the user and item embeddings over time.

Though simply conducting L_2 **loss function** for 5 times may also be a choice, however, L_2 only works well or regression tasks or similarity-based optimization (as in the original *JODIE*), and may ignore the corresponding relation in our task when several classes are involved. Therefore, we choose **Cross-Entropy loss** to optimize the model.

4.4. Abnormal detection

After adequate training, our model could competent behavior prediction for individual driver in most situations. The next step is to apply our model to alert any potential emergency by detecting abnormal behavior. The process is illustrated in figure 4.6.

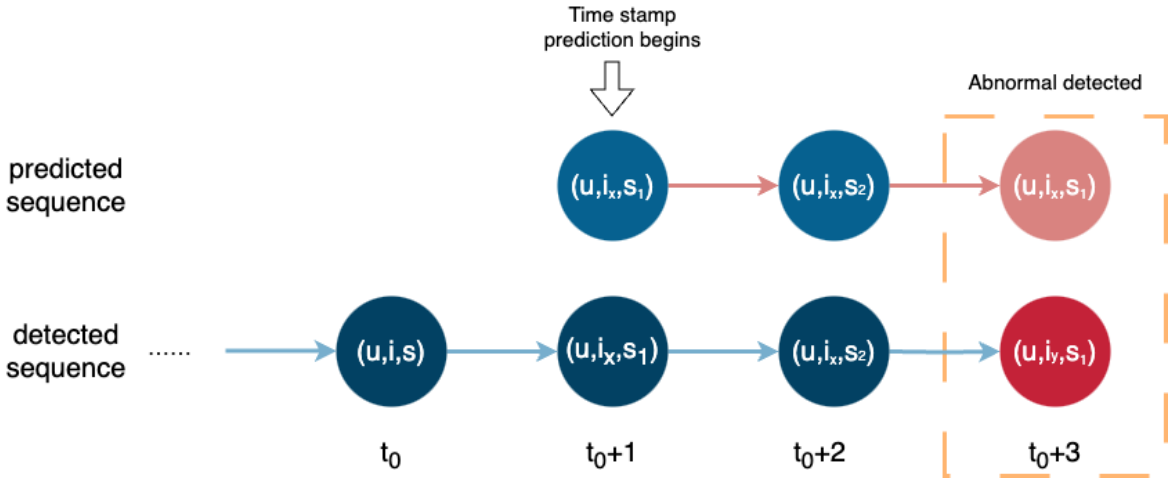

Figure 4.6.: Abnormal Detection Process

Suppose the vehicle equipped with our model also has a camera to film the driver's behavior. The video will be processed to generate the dynamic graph, and the model will predict the behavior state of the driver. Starting at the point when the prediction begins, a comparison will take place in each timestamp afterwords. Once the predicted behavior state is different from the actual behavior state, the model will draw attention to it immediately. Once this anomaly accumulates to some threshold, the model will alert the driver or the system to take action.

5. Evaluation

This chapter would

5.1.

6. Future Work

7. Conclusion

A. appendix

If there are several additions you want to add, but they do not fit into the thesis itself, they belong here.

A.1. Drive & act dataset

A.2. Hidden Markov Model

Even sections are possible, but usually only used for several elements in, e.g. tables, images, etc.

B. Figures

B.1. Example 1

✓

B.2. Example 2

✗

List of Figures

List of Tables

Bibliography

[1] T. Inagaki and T. B. Sheridan. "A critique of the SAE conditional driving automation definition, and analyses of options for improvement". In: *Cognition, technology & work* 21 (2019), pp. 569–578.

[2] J. D. Lee. "Driving safety". In: *Reviews of human factors and ergonomics* 1.1 (2005), pp. 172–218.

[3] J. D. Lee. "Fifty years of driving safety research". In: *Human factors* 50.3 (2008), pp. 521–528.

[4] M. Karrouchi, I. Nasri, M. Rhiat, I. Atmane, K. Hirech, A. Messaoudi, M. Melhaoui, and K. Kassmi. "Driving behavior assessment: a practical study and technique for detecting a driver's condition and driving style". In: *Transportation Engineering* 14 (2023), p. 100217.

[5] B. Aytekin and E. Altu. "Increasing driving safety with a multiple vehicle detection and tracking system using ongoing vehicle shadow information". In: *2010 IEEE International Conference on Systems, Man and Cybernetics*. IEEE. 2010, pp. 3650–3656.

[6] S. Kumar, X. Zhang, and J. Leskovec. "Predicting dynamic embedding trajectory in temporal interaction networks". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 1269–1278.

[7] A. Radford. "Improving language understanding by generative pre-training". In: (2018).

[8] A. Vaswani. "Attention is all you need". In: *Advances in Neural Information Processing Systems* (2017).

[9] K. Chowdhary and K. Chowdhary. "Natural language processing". In: *Fundamentals of artificial intelligence* (2020), pp. 603–649.

[10] S. Alaparthi and M. Mishra. "Bidirectional Encoder Representations from Transformers (BERT): A sentiment analysis odyssey". In: *arXiv preprint arXiv:2007.01127* (2020).

[11] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8 (2019), p. 9.

[12] T. B. Brown. "Language models are few-shot learners". In: *arXiv preprint arXiv:2005.14165* (2020).

[13] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. 2019. arXiv: 1910.13461 [cs.CL]. URL: https://arxiv.org/abs/1910.13461.

[14] L. R. Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286.

[15] L. E. Baum et al. "An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes". In: *Inequalities* 3.1 (1972), pp. 1–8.

[16] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. "Graph neural networks: A review of methods and applications". In: *AI open* 1 (2020), pp. 57–81.

[17] Y. Wu, D. Lian, Y. Xu, L. Wu, and E. Chen. "Graph convolutional networks with markov random field reasoning for social spammer detection". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 01. 2020, pp. 1054–1061.

[18] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. "Graph networks as learnable physics engines for inference and control". In: *International conference on machine learning*. PMLR. 2018, pp. 4470–4479.

[19] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur. "Protein Interface Prediction using Graph Convolutional Networks". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/f507783927f2ec2737ba40afbd17efb5-Paper.pdf.

[20] M. M. Bronstein, J. Bruna, T. Cohen, and P. Velikovi. "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges". In: *arXiv* (2021). DOI: 10.48550/arxiv.2104.13478. eprint: 2104.13478.

[21] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80. ISSN: 1045-9227. DOI: 10.1109/tnn.2008.2005605.

[22] T. N. Kipf and M. Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[23] M. Defferrard, X. Bresson, and P. Vandergheynst. "Convolutional neural networks on graphs with fast localized spectral filtering". In: *Advances in neural information processing systems* 29 (2016).

[24] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger. "Simplifying graph convolutional networks". In: *International conference on machine learning*. PMLR. 2019, pp. 6861–6871.

[25] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, et al. "Graph attention networks". In: *stat* 1050.20 (2017), pp. 10–48550.

[26] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. "Geometric deep learning on graphs and manifolds using mixture model cnns". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 5115–5124.

[27] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung. "Gaan: Gated attention networks for learning on large and spatiotemporal graphs". In: *arXiv preprint arXiv:1803.07294* (2018).

[28] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. "Neural message passing for quantum chemistry". In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[29] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. "Relational inductive biases, deep learning, and graph networks". In: *arXiv preprint arXiv:1806.01261* (2018).

[30] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein. "Temporal graph networks for deep learning on dynamic graphs. arXiv 2020". In: *arXiv preprint arXiv:2006.10637* (2020).

[31] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan. "Inductive representation learning on temporal graphs". In: *arXiv preprint arXiv:2002.07962* (2020).

[32] R. M. Schmidt. "Recurrent neural networks (rnns): A gentle introduction and overview". In: *arXiv preprint arXiv:1912.05911* (2019).

[33] J. L. Elman. "Finding structure in time". In: *Cognitive science* 14.2 (1990), pp. 179–211.

[34] M. I. Jordan. "Serial order: A parallel distributed processing approach". In: *Advances in psychology*. Vol. 121. Elsevier, 1997, pp. 471–495.

[35] A. Pentland and A. Liu. "Modeling and Prediction of Human Behavior". In: *Neural Computation* 11.1 (1999), pp. 229–242. ISSN: 0899-7667. DOI: 10.1162/089976699300016890.

[36] Q. Deng, J. Wang, and D. Soffker. "Prediction of human driver behaviors based on an improved HMM approach". In: *2018 IEEE Intelligent Vehicles Symposium (IV)* 00 (2018), pp. 2066–2071. DOI: 10.1109/ivs.2018.8500717.

[37] Z. Qiao, J. Zhao, J. Zhu, Z. Tyree, P. Mudalige, J. Schneider, and J. M. Dolan. "Human Driver Behavior Prediction based on UrbanFlow*". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)* 00 (2020), pp. 10570–10576. DOI: 10.1109/icra40945.2020.9196918.

[38] A. Palazzi, D. Abati, F. Solera, R. Cucchiara, et al. "Predicting the driver's focus of attention: the dr (eye) ve project". In: *IEEE transactions on pattern analysis and machine intelligence* 41.7 (2018), pp. 1720–1733.

[39] M. Regan, A. Williamson, R. Grzebieta, and L. Tao. "Naturalistic driving studies: literature review and planning for the Australian naturalistic driving study". In: *Proc. Australasian College Of Road Safety Conference, A Safe System: Expanding The Reach, Sydney*. 2012.

[40] M. Martin, A. Roitberg, M. Haurilet, M. Horne, S. ReiSS, M. Voit, and R. Stiefelhagen. "Drive Act: A Multi-Modal Dataset for Fine-Grained Driver Behavior Recognition in Autonomous Vehicles". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 2801–2810. DOI: 10.1109/ICCV.2019.00289.

[41] X. Chang, P. Ren, P. Xu, Z. Li, X. Chen, and A. Hauptmann. "A Comprehensive Survey of Scene Graphs: Generation and Application". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.1 (2023), pp. 1–26. DOI: 10.1109/TPAMI.2021.3137605.

[42] D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei. "Scene graph generation by iterative message passing". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 5410–5419.

[43] R. Zellers, M. Yatskar, S. Thomson, and Y. Choi. "Neural motifs: Scene graph parsing with global context". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 5831–5840.

[44] X. Wang and A. Gupta. "Videos as space-time region graphs". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 399–417.

[45] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

[46] K. Tang, Y. Niu, J. Huang, J. Shi, and H. Zhang. "Unbiased scene graph generation from biased training". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 3716–3725.

[47] F. Poursafaei, S. Huang, K. Pelrine, and R. Rabbany. "Towards better evaluation for dynamic link prediction". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 32928–32941.

[48] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha. "Representation learning over dynamic graphs". In: *arXiv preprint arXiv:1803.04051* (2018).

[49] Y. Zhu, H. Li, Y. Liao, B. Wang, Z. Guan, H. Liu, and D. Cai. "What to do next: Modeling user behaviors by time-LSTM." In.

[50] I. M. Baytas, C. Xiao, X. Zhang, F. Wang, A. K. Jain, and J. Zhou. "Patient subtyping via time-aware LSTM networks". In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 2017, pp. 65–74.

[51] A. Beutel, P. Covington, S. Jain, C. Xu, J. Li, V. Gatto, and E. H. Chi. "Latent Cross: Making Use of Context in Recurrent Recommender Systems". In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. WSDM '18. Marina Del Rey, CA, USA: Association for Computing Machinery, 2018, pp. 46–54. ISBN: 9781450355810. DOI: 10.1145/3159652.3159727. URL: https://doi.org/10.1145/3159652.3159727.

[52] A. Rajaraman. *Mining of massive datasets*. 2011.