Modeling Conditional Dependencies in Multiagent Trajectories

Yannick Rudolph
Leuphana University of Lüneburg
SAP SE, Machine Learning R&D, Berlin

Ulf Brefeld
Leuphana University of Lüneburg

Abstract

We study modeling joint densities over sets of random variables (next-step movements of multiple agents) which are conditioned on aligned observations (past trajectories). For this setting, we propose an autoregressive approach to model intra-timestep dependencies, where distributions over joint movements are represented by autoregressive factorizations. In our approach, factors are randomly ordered and estimated with a graph neural network to account for permutation equivariance, while a recurrent neural network encodes past trajectories. We further propose a conditional two-stream attention mechanism, to allow for efficient training of random factorizations. We experiment on trajectory data from professional soccer matches and find that we model low frequency trajectories better than variational approaches.

1 INTRODUCTION

This paper deals with modeling joint densities over sets of random variables in cases where the observations we are conditioning on are tightly aligned with the random variables whose distributions are to be modeled. One example for such a case is modeling dependencies in multiagent trajectories *within a single timestep*, conditioned on the past. In this case, we aim to estimate a joint density over next-step movements of several agents while conditioning on their past trajectories. Sets of next-step movements and past trajectories are tightly coupled via acting agents.

Current research on modeling dependencies in multiagent trajectories focuses on the usage of recurrent (RNN) and graph neural networks (GNNs, Scarselli et al., 2008), while latent variables are also often included in recently proposed models (e.g. Yeh et al., 2019; Casas et al., 2020). While GNNs can model interactions in past trajectories and align random variables in a permutation equivariant fashion, variational models involving latent variables are usually argued to help with the multi-modality of distributions over movements (cf. Rudolph et al., 2020).

What is less stressed, is the fact that latent variables, and especially the proposed variational models, in principle enable us to learn intra-timestep dependencies. In theory, they can account for dependencies within the joint distributions over movements, which GNNs alone cannot. This property of latent variable models is theoretically desirable but, as of now, it is not clear whether current latent variable models are actually able to materialize these theoretical benefits in practice. Given that deep autoregressive models have an impressive track record at modeling other kinds of joint distributions (e.g. van den Oord et al., 2016b,a), we instead propose to employ an autoregressive model to estimate joint distributions over next-step movements for multiagent trajectories.

Since we care to maintain the property of permutation equivariance wrt. agents (and thereby trajectories), given an autoregressive factorization we propose to model each factor with a GNN. While autoregressive models usually work wrt. a fixed (sometimes arbitrarily) ordering of the random variables to be estimated (e.g. Larochelle and Murray, 2011), the property of permutation equivariance suggests training over random orderings of intra-timestep agent movements instead. Our approach is thus closely related to training *order-agnostic* autoregressive models such as orderless NADE (Uria et al., 2014, 2016) or XLnet (an autoregressive pretraining method for natural language tasks, Yang et al., 2019). In both cases, models are trained over factorizations wrt. random orderings.

To allow for efficient training of random autoregressive factorizations, we devise conditional two-stream attention (CTSA), a method that extends the two-

Proceedings of the 25th International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151. Copyright 2022 by the author(s).

stream attention mechanism of XLNet to conditional distributions on sets. That is, CTSA allows us to efficiently train autoregressive GNNs or/and transformers (Vaswani et al., 2017) for joint densities over sets of random variables, conditioned on observations that are aligned with the random variables whose distributions we are modeling: With ground truth targets provided via teacher forcing, we only need one forward pass per training instance (or batch) and update. Training the autoregressive model naively or with other training approaches may require as many passes as there are random variables in the set.

Empirically, we experiment on trajectory data from professional soccer matches and compare our proposed model for intra-timestep dependencies to both, a fully-connected GNN in the form of a transformer, and to a variational version thereof. In all models, features of past trajectories are extracted with RNNs and the density over movements is estimated via a mixture density network (MDN, Bishop, 1994) approach. We evaluate models wrt. different discretization frequencies and find that CTSA models low-frequency data better than the methods we compare to.

Specifically, our contributions are: (i) We propose a novel autoregressive approach to learn intra-timestep dependencies in multiagent trajectories (Section 3). (ii) We propose conditional two-stream attention (CTSA, Section 4), to allow for efficient training of conditional autoregressive distributions of random factorizations. (iii) We validate our approach by comparing to GNN and variational GNN models on trajectory data from professional soccer matches (Section 5).

2 PROBLEM SETTING

Suppose we want to model a continuous distribution over movements of several agents on the basis of low-dimensional positional data (i.e. trajectories of xy-coordinates). Further suppose, that there is no natural ordering of those agents, and that we observe different combinations of agents. We might also want to share parts of the model architecture over agents. A reasonable model would thus be autoregressive over time (which is naturally ordered), but permutation equivariant over agents. Hence, combining RNNs with GNNs would be a natural choice. At this point, we want to note that we include position-agnostic transformer architectures when we refer to graph neural networks (for insights into the relationship of graph [neural] networks and transformers cf. Battaglia et al., 2018).

Denoting past observations for agent k at timestep t with $\mathbf{x}_k^{<t}$, let $\mathbf{h}_k^t = f(\mathbf{x}_k^{<t})$ be a representation of $\mathbf{x}_k^{<t}$ encoded with an RNN (optionally the representation can be enriched with the output of additional feature extractors) and let $\mathbf{h}^t = \{\mathbf{h}_1^t, \dots, \mathbf{h}_K^t\}$ be the set of representations for all K agents. Set functions, such as GNNs over fully-connected graphs or position-agnostic transformer architectures, allow us to estimate a conditional joint distribution $p(\Delta\mathbf{x}^t|\mathbf{h}^t)$ over the set of all K movements $\Delta\mathbf{x}_k^t = \mathbf{x}_k^t - \mathbf{x}_k^{t-1}$ at timestep t, which we denote by $\Delta\mathbf{x}^t = \{\Delta\mathbf{x}_1^t, \dots, \Delta\mathbf{x}_K^t\}$. *Assuming conditional independence*, we can thus estimate parameters ψ_k^t of parameterized movement distributions per agent. An appropriate GNN pass would result in the estimation of a set of parameters $\psi^t = \{\psi_1^t, \dots, \psi_K^t\}$, while our joint distribution would factorize as $p(\Delta\mathbf{x}^t|\mathbf{h}^t) = p(\Delta\mathbf{x}^t|\psi^t) = \prod_{k=1}^K p(\Delta\mathbf{x}_k^t|\psi_k^t)$. Due to the permutation equivariance of set functions (cf. Zaheer et al., 2017), this distribution would be invariant to permutation over trajectories.

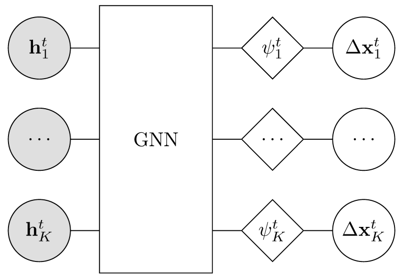


Figure 1: Modeling joint movements $\Delta\mathbf{x}^t$ with a GNN assumes conditional independence (filled/unfilled circles denote observed/unobserved random variables).

At this point, we want to stress the fact that fully-connected GNNs, or transformer architectures for that matter, lead to conditionally independent distributions over movements within a single timestep due to the deterministic nature of these models: The parameters ψ_k^t are deterministically determined given inputs $\mathbf{h}^t$ (compare Figure 1 for a visualization). With an MDN approach, where GNNs output parameters of a Gaussian mixture model (GMM) per agent, we can nevertheless model multimodal continuous distributions per agent. For multiagent trajectories this already is a strong baseline (cf. Rudolph et al., 2020).

However, the performance of these models depends on whether the past trajectories capture all information about potential interactions. If this were to be the case, any stochasticity in the trajectories might reasonably be strictly local and due to the decisions and actions made by each agent. In the case of partial observability, conditionally independent models for intra-timestep movement naturally fail to account for dependencies that are due to unobserved or latent variables. Indeed, incorporating explicit global or local latent variables $\mathbf{z}^t$ or $\mathbf{z}^t = \{\mathbf{z}_1^t, \dots, \mathbf{z}_K^t\}$ into the models is one way to allow for conditional dependencies between movements. Latent variables can induce conditional dependencies into the above factorization of $p(\Delta\mathbf{x}^t|\mathbf{h}^t)$ via pushing a dependence on the latent variables into each factor. For continuous latent variables, we have

$$\begin{aligned} p(\Delta\mathbf{x}^t|\mathbf{h}^t) &= \int_{\mathbf{z}^t} p(\mathbf{z}^t|\mathbf{h}^t)p(\Delta\mathbf{x}^t|\mathbf{h}^t,\mathbf{z}^t)d\mathbf{z}^t \\ &= \int_{\mathbf{z}^t} p(\mathbf{z}^t|\mathbf{h}^t)\prod_{k=1}^{K} p(\Delta\mathbf{x}^t_k|\mathbf{h}^t,\mathbf{z}^t)d\mathbf{z}^t. \end{aligned}$$

These kind of models take on the form of conditional variational autoencoders (CVAEs, Sohn et al., 2015) which can be learned like regular variational autoencoders (VAEs, Kingma and Welling, 2014; Rezende et al., 2014) in principle. For the task of modeling multiagent trajectories, both the encoder as well as the decoder may involve GNNs. We consider the graph variational RNN (GVRNN, Yeh et al., 2019) an example of this model class, even though it comes with more involved modeling choices, such as feeding the latent variables into the RNN, as is prescribed for variational RNNs (Chung et al., 2015).

Even sophisticated dynamical VAEs (cf. Girin et al., 2020) like the GVRNN model might however fail to model certain dependencies due to *posterior collapse*, as VAEs are known to have issues with relatively strong decoders (cf. Bowman et al., 2015; Chen et al., 2017; Zhao et al., 2019). When the dependence of joint next-step movements $\Delta\mathbf{x}^t$ on past trajectories $\mathbf{x}^{<t}$ is relatively high, a decoder that is conditioned on the latter's joint representation $\mathbf{h}^t$ is likely to be a strong decoder. The possibility of this failure mode motivates an autoregressive approach to distribution estimation within each individual timestep. On a technical note, the proposed autoregressive approach further allows for arbitrary conditioning within joint intra-timestep movements, which is not feasible with vanilla latent variable approaches. We would have to make changes to the model and training scheme to allow for such conditioning in latent variable models (for standard VAEs this has been proposed in Ivanov et al., 2019).

3 AN AUTOREGRESSIVE APPROACH

In this section, we propose to model the distribution over joint intra-timestep movements by means of autoregressive factorization, where factors are estimated with a fully-connected GNN. Suppose that the joint movement at timestep t, $\Delta\mathbf{x}^t$, is ordered by a K-tuple o, denoting permutations of integers 1 through K. For any such tuple, the factorization

$$p(\Delta\mathbf{x}^t|\mathbf{h}^t) = \prod_{k=1}^{K} p(\Delta\mathbf{x}^t_{o_k}|\Delta\mathbf{x}^t_{o_{<k}},\mathbf{h}^t)$$

does not impose any constraints on the distribution $p(\Delta\mathbf{x}^t|\mathbf{h}^t)$ (cf. Uria et al., 2014, 2016). Specifically, the factorization does not prescribe any form of conditional independence and we thus in principle can model conditional dependencies inherent in the joint distribution over movements. In accordance with previous publications, we refer to the above factorization as *autoregressive* (AR).

Given a random but fixed ordering o, we can thus introduce a tractable generative model for our task: We first estimate a distribution for the first factor $p(\Delta\mathbf{x}^t_{o_1}|\mathbf{h}^t)$, given features of all past trajectories, with a GNN. We then sample the first movement and encode it into representation $\phi(\Delta\mathbf{x}^t_{o_1})$. Combining representations $\phi(\Delta\mathbf{x}^t_{o_1})$ and $\mathbf{h}^t_{o_1}$ at a graph node, we next estimate the distribution $p(\Delta\mathbf{x}^t_{o_2}|\phi(\Delta\mathbf{x}^t_{o_1}),\mathbf{h}^t)$ and continue to iteratively estimate and sample from $p(\Delta\mathbf{x}^t_{o_k}|\Delta\mathbf{x}^t_{o_{<k}},\mathbf{h}^t)$ in the same fashion until we have generated all joint movements $\Delta\mathbf{x}^t$. See Figure 2 for a sketch of this process. Note that, by combining movement representation $\phi(\Delta\mathbf{x}^t_k)$ of agent k with a representation of its past $\mathbf{h}^t_k$ at the same graph node, we explicitly encode which next-step movement and which past belong to the same agent.

As noted earlier, we assume that there is no natural ordering to the trajectories. Since we care to maintain the property of permutation equivariance wrt. agents (and thereby trajectories), this suggests training over random orderings of intra-timestep agent movements. Assuming the ordering of the trajectories to be random, we are thus training our model according to random factorizations. Note, that this does not result in a model which is equivariant to permutations over different autoregressive factorizations. However, given a model with a *fixed* autoregressive factorization, the proposed method *is* permutation equivariant with respect to past trajectories and conditional densities of individual agent movements. That is, we have to consider two types of equivariance: one regarding random factorizations and one regarding permutations of agents. As in orderless NADE (Uria et al., 2014, 2016), we can take the viewpoint that with respect to random factorizations we are effectively training an ensemble of models with shared parameters: While the model will result in different log-likelihood estimates and/or samples for different orderings, in principle we model and optimize the expected log-likelihood $\mathbb{E}_o[p(\Delta\mathbf{x}^t|\mathbf{h}^t;o)]$, where the expectation is over different orderings o of factors $p(\Delta\mathbf{x}^t_{o_k}|\Delta\mathbf{x}^t_{o_{<k}},\mathbf{h}^t)$, and learning remains straight-forward. We can even eval-

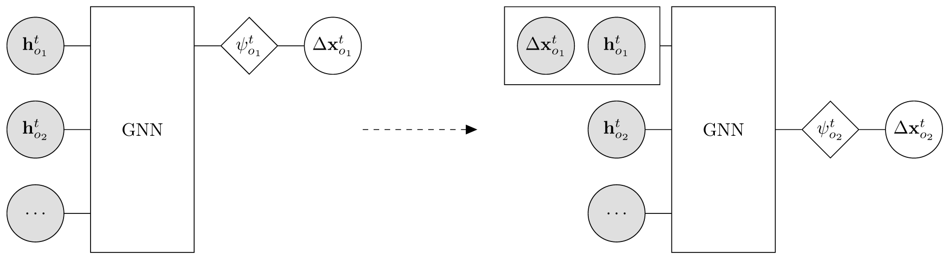


Figure 2: Autoregressive process of generating joint movements $\Delta\mathbf{x}^t$ with a GNN. We depict the first two factors.

uate the log-likelihood with an ensemble over different orderings. Sampling is implemented according to random orderings.

Training the model iteratively wrt. ground truth movements $\Delta\mathbf{x}^t$ (for time t) however is inefficient as we would need K forward passes through the GNN to calculate losses wrt. all K agents. To train more efficiently, we could resort to a masking scheme which has been proposed for training in orderless NADE (Uria et al., 2014, 2016) as well as in Ghazvininejad et al. (2019). However, also with this approach we may not estimate all parameters ψ^t of the joint distribution over movements in parallel (as is also pointed out in Alcorn and Nguyen, 2021b). Standard practice for efficiently training transformer decoders on sequences involves an autoregressive mask, which prevents updates to representations in intermediate and final layers wrt. subsequent nodes in the current ordering (which for language models is usually *causal*, i.e. from left to right, cf. Vaswani et al., 2017). This practice results in a single forward pass through the transformer per sequence and update during training and is thus as efficient wrt. forward passes, as we can get. Naively applying such an autoregressive mask is however not an applicable practice for modeling conditional densities of the form considered in this paper.

4 CONDITIONAL TWO-STREAM ATTENTION

In this section, we show how efficient training can be addressed by building upon XLNet's (Yang et al., 2019) two-stream attention mechanism and thus by incorporating *query* nodes into our model. We begin our exposition by stating the desiderata that we build upon to implement an autoregressive masking scheme that requires only a single forward pass for our task.

4.1 Desiderata for Masking Scheme

A causal autoregressive mask as discussed above does not transfer to the task where we want to model conditional distributions over next-step movements. In particular, combining representations of next-step movements $\Delta\mathbf{x}^t$ and representations of past trajectories $\mathbf{h}^t$ at individual nodes (as is depicted in Figure 2) cannot result in proper masking. To allow for parallel and efficient training it is important to use separate sets of nodes to represent (i) individual observed movements $\Delta\mathbf{x}^t_{o_k}$ of the agents (which are the targets in our training process) and (ii) individual information about previous timesteps $\mathbf{h}^t_{o_k}$ which we condition on.

These two sets of nodes are however not sufficient, if we aim to not only use shallow graph neural networks in the involved transformations: While we could in principle estimate $\psi^t_{o_k}$ by letting the representation $\mathbf{h}^t_{o_k}$ attend to all of $\mathbf{h}^t$ and appropriately masked targets $\Delta\mathbf{x}^t$, this is only an appropriate masking scheme for a single layer. In any subsequent layer, we cannot allow $\mathbf{h}^t_{o_k}$ to attend to *all* representations of other nodes departing in $\mathbf{h}^t$, as some of these nodes will have been updated with respect to $\Delta\mathbf{x}^t_{o_{\geq k}}$, and thus with information that a node representation resulting in $\psi^t_{o_k}$ is not supposed to have. We thus need to introduce a third set of nodes (iii), which we refer to as *query* nodes and which we denote by $\mathbf{q}^t = \{\mathbf{q}^t_{o_1}, \ldots, \mathbf{q}^t_{o_K}\}$. We can then estimate each $\psi^t_{o_k}$ by transforming its respective query node $\mathbf{q}^t_{o_k}$ appropriately: Each $\mathbf{q}^t_{o_k}$ should be able to attend to *all* representations of $\mathbf{h}^t$ and every preceding target $\Delta\mathbf{x}^t_{o_{<k}}$ according to the ordering o. However, each $\mathbf{q}^t_{o_k}$ should not be allowed to attend either directly or indirectly to any information about $\Delta\mathbf{x}^t_{o_{\geq k}}$.

4.2 Query Nodes as the Solution

In the spirit of XLNet's two-stream attention (Yang et al., 2019), we introduce *conditional* two-stream attention (CTSA), in which we employ *query* nodes that adhere to the above desiderata. Because the specific architecture which we propose constitutes an encoder-decoder like transformer model, we make use of terms from transformer literature. Note however, that the underlying concepts should transfer to other (fully-connected) graph neural networks. With a transformer architecture, updates to our node representation conveniently involve multi-headed attention (Vaswani et al., 2017) and other standard design

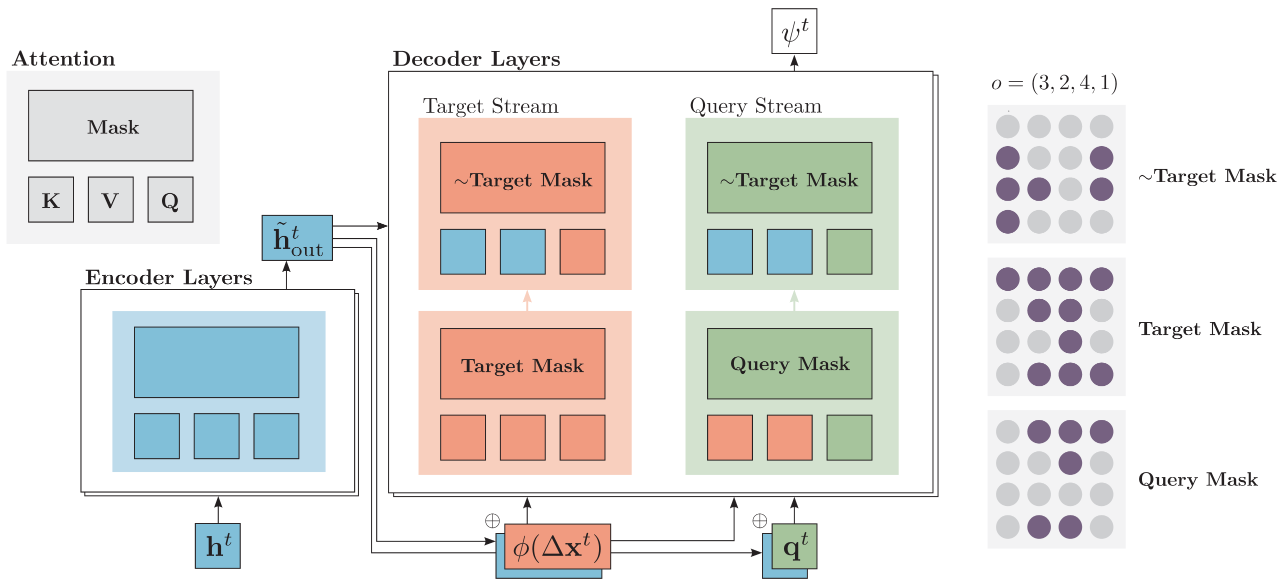


Figure 3: Sketch of the proposed conditional two-stream attention (CTSA) approach: Standard encoder layers model interactions of past trajectories. In decoder layers we employ two attention streams. Example attention masks are shown to the right (a filled dot in row r and column c denotes that node r can attend to position c).

choices, which we assume to be beneficial for the capacity and representative power of the proposed model given recent successes of transformers in areas other than natural language processing (e.g. Carion et al., 2020).

A sketch of the proposed architecture is provided in Figure 3. We assume to have access to some representations of past trajectories $\mathbf{h}^t$, as well as ground truth next-step movements $\Delta\mathbf{x}^t$. We also assume, that we want to train wrt. a specific random ordering o. For the purpose of this exposition, representations in $\mathbf{h}^t$ will be assigned to *context* nodes, while we will call representations based on $\phi(\Delta\mathbf{x}^t)$ *target* nodes (note, that this terminology differs from the terminology used in Yang et al., 2019). We further introduce *query* nodes $\mathbf{q}^t = \{\mathbf{q}^t_{o_1}, \ldots, \mathbf{q}^t_{o_K}\}$, that share a single learnable embedding and whose final representation $\tilde{\mathbf{q}}^t_{\text{out}}$ will be used to estimate parameters ψ^t for agent movements at timestep t (in this notation *tilde* marks updated representations).

We first model any interactions in the context $\mathbf{h}^t$ by means of a standard transformer encoder with unmasked self-attention, giving us an updated context $\tilde{\mathbf{h}}^t_{\text{out}}$ (depicted in the left part of Figure 3). We then add $\tilde{\mathbf{h}}^t_{\text{out}}$ to initial target and query representations $\phi(\Delta\mathbf{x}^t)$ and $\mathbf{q}^t$ (depicted by the $\oplus$ symbols in Figure 3). We propose elementwise addition, but also experimented with concatenation. We then update both target and query representations alike in a transformer decoder, for which we implement two-stream attention: Specifically, we keep updating two *streams* (or strands) of transformer decoder layers: one for the target representations and one for the query representations (depicted in the center of Figure 3). We further ensure appropriate masking (depicted in the right part of Figure 3). The target representations are updated with a self-attention layer, which features an autoregressive mask according to given ordering o, such that any target node can attend to any preceding target and to itself. The query representations are updated with an attention layer, where query representations constitute attention queries (Q), while the targets constitute attention keys (K) and values (V). In principle, we employ the same autoregressive mask here as is used in the self-attention layer for the targets, but exclude the attention of any single query $\tilde{\mathbf{q}}^t_{o_k}$ to its corresponding targets $\tilde{\phi}(\Delta\mathbf{x}^t_{o_k})$ (i.e. the diagonal of the attention mask is turned off). In both decoder strands, what follows is an attention layer attending to $\tilde{\mathbf{h}}^t_{\text{out}}$ (i.e. $\tilde{\mathbf{h}}^t_{\text{out}}$ are used for keys and values) with targets and queries functioning as attention queries (Q) respectively. We decided to employ the inverse of the target mask wrt. this attention layer, although this might be considered an implementation detail.

The important aspect of this architecture is, that we keep updating target and query representations, while making sure that both have access to all information in the context $\tilde{\mathbf{h}}^t_{\text{out}}$, but that any single query representation $\tilde{\mathbf{q}}^t_{o_k}$ will only have had access to information from target representations $\phi(\Delta\mathbf{x}^t_{o_{<k}})$, which precede

the query in the given ordering o. That is: with CTSA we devised a method to estimate all parameters ψ^t of a conditional movement model in parallel while adhering to some random factorization, given ground truth movements are provided during training. Random orderings can be implemented via generating random autoregressive masks. The proposed method is different than the two-stream attention mechanism as introduced for XLNet in various ways. Conceptually most relevant is the following: In XLNet, queries only see target representations which precede these in the given order, and conditioning is only necessary with respect to the positional information, i.e. the query position. For CTSA we are making sure that queries can attend to *all* of the observations $\tilde{\mathbf{h}}^t_{o_1}$ through $\tilde{\mathbf{h}}^t_{o_K}$ which we are *conditioning* on, while also ensuring that the model properly aligns queries with context and targets.

Wrt. the context and the encoder there is nothing to be aligned, as all representations can see all other representations. Within the decoder, we prefer not to attend to information belonging to a single agent k twice (i.e. once to agent k's context and once to agent k's target representation). Our reasoning is, that the model otherwise would have to implicitly align both representations. We hence simply mask attention to the encoder output with the inverse of the target mask. Note, that since we added context information to initial target and query nodes and we have skip connections around attention layers. Each target and query thus always is updated wrt. all (allowed for) information from every other agent, including itself. We refer to Sections B and C of the supplementary material for more implementation details and pseudo code.

5 MODELING MULTIAGENT TRAJECTORIES

5.1 Data and Evaluation

We evaluate our proposed approach on proprietary trajectory data from professional soccer matches.[1] The data which we use consists of xy-coordinates for all players and xyz-coordinates for the ball, recorded at 25 Hz (i.e. frames) per second. In total, we experiment wrt. 95 matches, from which we extract roughly 92,000 sequences of five seconds each, where (i) all 22 players and the ball are on the pitch, (ii) the ball remains in play and (iii) one team retains ball possession over the whole five seconds. Consecutive sequences in general overlap by two seconds. We experiment wrt. five data folds over matches, always training on 3/5 of the data, validating on 1/5 and testing the models with the best validation log-likelihoods on the last 1/5 of the data.

We transform the data as such, that the team in possession of the ball always plays from left to right. Since our models process the data as sets of trajectories, we provide type embeddings which encode whether a trajectory belongs to (i) the ball, (ii) the keeper belonging to the team with ball possession, (iii) a field player belonging to the team with ball possession, (iv) the keeper belonging to the team without ball possession and (v) the field players belonging to the team without ball possession. For different experiments, we downsample the sequences to either 5 Hz, 2.5 Hz or 1 Hz. Note that this results in sequences with 25, 13 and 5 frames respectively. For every timestep t we encode trajectories into representations $\mathbf{h}^t$ by means of an RNN with GRU cells (Cho et al., 2014). We further provide the models with current positions $\mathbf{x}^{t-1}$, as well as last time differences $\Delta\mathbf{x}^{t-1}$, on both of which we apply simple feed-forward networks for feature extraction. Positions are normalized such that the pitch (which is of the same size for all our instances) ranges from -1 to 1 in both, x and y dimension. Time differences are standardized wrt. the training data. Output distributions $p(\Delta\mathbf{x}^t_k|\ldots)$ are modeled with a mixture density network approach with ten mixture components.

In all experiments, we only model the ten trajectories $\mathbf{x}_{1:10}$ of the field players belonging to the defending team. At every timestep, we condition on the ground truth past of the ball and other players, but neither on their next movement nor on any other future information. That is, at every timestep we model $p(\Delta\mathbf{x}^t_{1:10}|\mathbf{x}^{<t}_{1:23})$, were $\mathbf{x}_{11:23}$ refer to the trajectories of the ball and of other players. When we generate trajectories, we update $\mathbf{x}^t_{11:23}$ with ground truth data after sampling $\mathbf{x}^t_{1:10}$ via next-step movements $\Delta\mathbf{x}^t_{1:10}$.

We report log-likelihood values, respectively importance sampled estimates thereof for the variational model (cf. Rezende et al., 2014; Burda et al., 2015), that relate to the data at an unnormalized scale. We provide mean values over trajectories, agents and timesteps. As a proxy for the quality of the different models, we further measure the L_2 deviation from generated $\tilde{\mathbf{x}}$'s to ground truth data $\mathbf{x}$. Specifically, we employ the following measures:

$$\frac{1}{SKT}\sum_{s=1}^{S}\sum_{k=1}^{K}\sum_{t=1}^{T}\left\|\tilde{\mathbf{x}}^t_{k,s}-\mathbf{x}^t_k\right\|_2\,, \qquad (1)$$

$$\text{and} \min_{s\in\{1,\ldots,S\}}\frac{1}{KT}\sum_{k=1}^{K}\sum_{t=1}^{T}\left\|\tilde{\mathbf{x}}^t_{k,s}-\mathbf{x}^t_k\right\|_2\,, \qquad (2)$$

where $t=1$ denotes the first and T the last prediction

[1]All matches are taken from season 2017/18 of the German Bundesliga and can be acquired from the German league (DFL). Only rather small datasets of similar kind are publicly available, for example the *soccer video and player position dataset* (Pettersen et al., 2014).

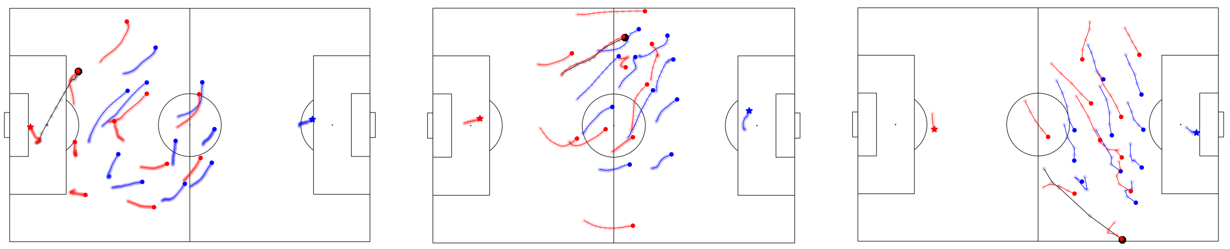

(a) 5 seconds at 5 Hz (b) 5 seconds at 2.5 Hz (c) 5 seconds at 1 Hz

Figure 4: Movement generation with CTSA jointly modeling the ten blue field players (last three seconds).

timestep. With s, we index independent trajectories rolled out with the models, and fix S to 5 in our experiments. As we only average over the trajectories which we model, K is set to 10. In results Table 1, *Avg.* L_2 refers to mean deviations over samples (1), while *Avg.* L_2 *(best)* refers to the mean deviation within the best sample wrt. a single trajectory (2). *Fin.* and *Fin. (best)* refer to the measures evaluated only at the final step $t = T$. We further average within the testset, while mean and standard errors are wrt. datasplits.

5.2 Models and Results

Within this setting, we compare three models for the task of estimating conditional distributions over joint next-step movements: Specifically, we compare an autoregressive model trained via conditional two-stream attention (which we refer to as CTSA) to both, a fully-connected GNN in the form of a standard transformer encoder, as well as to a conditional variational autoencoder. In the latter, we associate local latent variables with each agent. The variational posterior and decoder are both modeled with transformer encoder layers. The prior over latent variables is taken to be unconditional and standard normal (while this departes from the most standard CVAE formulation, assuming an uncoditional prior is a valid alteration to the model; cf. Sohn et al., 2015). We refer to this graph variational model as GVNN. All three models have been designed to have roughly 4.5 million parameters and were trained with Adam (Kingma and Ba, 2014) at a learning rate of 0.001 and standard parameters otherwise. For all models we fairly tweaked architectures and experimented with different levels of regularization via dropout (cf. Srivastava et al., 2014) in the transformer layers, as well as with different learning rates in preliminary experiments. Detailed model architectures are provided in Section B of the supplementary material. We also explored annealing the KL divergence (cf. Sønderby et al., 2016) for the GVNN. However, this did not improve results but rather resulted in unstable training and KL divergences remained very low for experiments at all three frequencies. We thus credit the low KL values to a form of posterior collapse that annealing alone is not able to fix. In line with the low KL values, importance sampling (with 100 samples) improves on the ELBO only marginally.

Results for experiments on trajectories at 5 Hz, 2.5 Hz and 1 Hz data can be found in Table 1. We would like to stress that the empirical results for different Hz rates are *not* comparable. When generating trajectories from the models, we condition on two seconds of data and generate the final three seconds always. Nevertheless, *intra-model* performances relate to different tasks since we condition on different pieces of information given different data frequencies. Regarding the results, we especially find that CTSA performs best wrt. log-likelihood and L_2 deviations when comparing models at 1 Hz. At 2.5 and 5 Hz both GNN and GVNN are performing better than CTSA wrt. L_2 deviations. Log-likelihood values (respectively importance sampled estimates thereof) for these two experiments are very close. Generated trajectories from the CTSA model are depicted in Figure 4. We conjecture, that there are more conditional dependencies in the joint distribution over movements in the 1 Hz data than at higher frequencies. This is a reasonable conjecture, because there are strictly more unknowns in the 1 Hz data. And it implies, that while CTSA does not provide benefits at higher frequencies, it models conditional dependencies better than other models.

See Table 2 for an ablation study wrt. CTSA model choices conducted on the 1 Hz data. We compare to both, a model without attention mask for context attention, as well as to a model in which we do not include context attention at all. Note, that even the model without context attention should have strictly more power than a vanilla GNN, since we initialize queries and targets with the context representation as updated by a transformer encoder in the first place. The results do support our architectural choices. Importantly, the difference of CTSA to the model without context attention is statistically significant, as per a paired t-test with a p-value of 0.006. The advantage of CTSA over a model without context mask is

Table 1: Results for experiments with trajectories at different frequencies. Notably, CTSA is best at modeling the 1 Hz data, for which we suspect more conditional dependencies in joint movements. Note, that metrics are *not* comparable over experiments at different Hz. For the velocity baseline, we continued trajectories with last known time-differences $\Delta \mathbf{x}^t$. Further: LL = log-likelihood, SE = standard error, and L_2 deviations are in meter.

		LL		Avg. L_2		Avg. L_2 (best)		Fin. L_2		Fin. L_2 (best)	
Hz	Step model	Mean	SE	Mean	SE	Mean	SE	Mean	SE	Mean	SE
1	Velocity			3.35	±0.01			5.58	±0.01		
1	CTSA	**−1.49**	±0.01	**2.24**	±0.01	**1.81**	±0.01	**3.31**	±0.02	**2.61**	±0.01
1	GVNN	−1.65	±0.00	2.29	±0.01	1.90	±0.01	3.39	±0.02	2.74	±0.01
1	GNN	−1.65	±0.00	2.29	±0.02	1.90	±0.02	3.39	±0.03	2.74	±0.03
2.5	Velocity			2.58	±0.01			5.51	±0.01		
2.5	CTSA	**1.45**	±0.00	1.60	±0.01	1.32	±0.01	2.97	±0.01	2.39	±0.01
2.5	GVNN	1.44	±0.01	1.58	±0.01	1.31	±0.00	2.93	±0.01	2.36	±0.01
2.5	GNN	1.43	±0.01	**1.56**	±0.00	**1.30**	±0.00	**2.90**	±0.01	**2.34**	±0.01
5	Velocity			2.06	±0.00			4.82	±0.01		
5	CTSA	**3.82**	±0.02	1.37	±0.01	1.14	±0.01	2.82	±0.02	2.28	±0.02
5	GVNN	**3.83**	±0.02	**1.33**	±0.00	**1.10**	±0.00	**2.71**	±0.01	**2.19**	±0.01
5	GNN	**3.82**	±0.02	**1.33**	±0.01	1.11	±0.00	2.73	±0.01	**2.20**	±0.01

Table 2: Ablation study for CTSA at 1 Hz

Step model	Mean LL	SE
CTSA (as proposed)	**−1.493**	±0.006
CTSA w/o ctx. mask	−1.497	±0.004
CTSA w/o ctx. attn.	−1.503	±0.005

not statistically significant (p-value of 0.375) though. Whether the latter result relates to the rather shallow CTSA architecture (with only two CTSA decoder layers) or whether this might be an indication that attention does not need to be guided explicitly regarding alignment of past trajectories and targets remain open questions. We also explored the option of concatenating the context to target and query representations (including necessary transforms), however this did not result in any performance improvements over a model with elementwise addition.

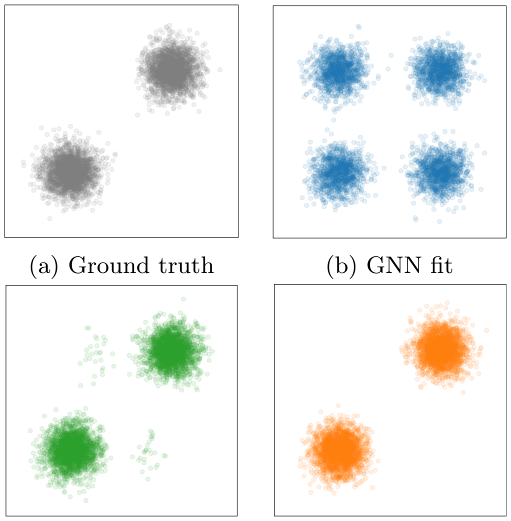
5.3 Fitting Synthetic One-dimensional Movements of Two Coordinated Agents

To highlight the capacities of the GNN, GVNN and CTSA respectively, we further experiment on a simple synthetic dataset. We assume two agents are moving a single coordinated step on the real line; one on the x-axis and one on the y-axis. Each agent has information wrt. the location of different two-dimensional Gaussians, which they combine in a mixture model to coordinate their movements.

We use downsized models to fit the data in form of local agent information and samples from the joint movement distribution. Notably, we employ four mixture components in the Gaussian mixture output distribution of the MDN architecture, which still amounts to strictly more capacity than is needed to fit the data. We show resulting distribution estimates by sampling from fitted models in Figure 5. While both the GVNN and CTSA can fit the data reasonably well, the GNN fails to model the data as is to be expected, since it assumes independent distributions per agent. We want to point out, that fitting the data with the GVNN took several approaches, and that we still observe artifacts of wrongly placed Gaussians in Figure 5c. Arguably, it is easier to fit a GVNN with a single Gaussian output distribution to the data.

6 RELATED WORK

Generative models for soccer player trajectories have recently been argued to bear potential for in-depth game analysis, especially considering the study of counterfactual movements (Tuyls et al., 2021). While early work on modeling multiagent trajectories involved heuristics to account for player ordering (Le et al., 2017; Zhan et al., 2019), the use of GNNs (cf. Battaglia et al., 2018) with attention (Xu et al., 2015; Bahdanau et al., 2015) or in the form of transformers (Vaswani et al., 2017) to model agent interaction is widespread. Notable applications to sports include Hoshen (2017) and Kipf et al. (2018). Variational autoencoding approaches towards sports trajectories in-



(a) Ground truth (b) GNN fit

(c) GVNN fit (d) CTSA fit

Figure 5: Synthetic data with 2 agents jointly moving on the x- and y-axis respectively. Samples from the ground truth distribution as well as from fitted models.

clude Felsen et al. (2018), Zhan et al. (2019) and Yeh et al. (2019). A recently proposed autoregressive approach is Alcorn and Nguyen (2021a). Fadel et al. (2021) propose to model agent movements conditioned on other agents via a combination of GNNs and conditional flows (Lu and Huang, 2020), however they do not consider joint movement distributions.

The closest approach to our proposed method, we consider to be Alcorn and Nguyen (2021a), who employ a standard transformer encoder with autoregressive mask over both the timestep and agent dimension to model basketball trajectory data in a discretized output space. Abstracting from initial locations, for each timestep and agent, positional representations are augmented with target representations, which are inserted sequentially to the right. While this allows to use a standard autoregressive mask for subsequent representations, this in effect doubles the number of representations in which attention is quadratic. Redundancies are introduced, as each previous position is attend to twice (once as query representation and once as target representation). While we also introduce additional nodes, we do not explicitly attend over different timesteps, since we employ an RNN architecture to encode past trajectories. Most importantly, the model proposed in Alcorn and Nguyen (2021a) requires attention to implicitly align all representations belonging to the same trajectory, while we propose to align trajectories more explicitly in CTSA.

While we chose to model past trajectories individually, others (e.g. Dick et al., 2021; Yeh et al., 2019) successfully model interactions in trajectories within the RNN. This is related to graph recurrent neural networks (GRNNs, e.g. Sanchez-Gonzalez et al., 2018). We consider this a design choice as we can arrive at context representations accounting for interactions in different ways. Regarding the general problem of estimating conditional distributions with aligned observations, we however consider the proposed encoder approach a somewhat universal solution.

7 CONCLUSION

In this paper, we proposed an autoregressive approach to model conditional distributions over joint movements in multiagent trajectories, or more general conditional distributions with aligned random and observed variables. We further proposed a method (CTSA) to train such conditional distributions efficiently while adhering to a random autoregressive factorization. We validated the proposed autoregressive approach, training scheme and model choices on trajectory data from professional soccer matches and find that our method models low frequency data better than other approaches. We conjecture, that this is due to low frequency data being prone to more conditional dependencies in joint distribution over movements within each timestep.

Acknowledgements

We would like to thank Marius Lehne at SAP SE for his essential support and help while writing this paper as well as all reviewers for their valuable feedback.

References

Michael A. Alcorn and Anh Nguyen. baller2vec++: A Look-Ahead Multi-Entity Transformer For Modeling Coordinated Agents. *arXiv preprint arXiv:2104.11980*, 2021a.

Michael A. Alcorn and Anh Nguyen. The DEformer: An Order-Agnostic Distribution Estimating Transformer. *(Workshop) International Conference on Machine Learning*, 2021b.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations*, 2015.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational Inductive Biases, Deep Learning, and Graph Networks. *arXiv preprint arXiv:1806.01261*, 2018.

Christopher M. Bishop. Mixture Density Networks. Neural Computing Research Group Report, 1994.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance Weighted Autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-End Object Detection with Transformers. In *European Conference on Computer Vision*, 2020.

Sergio Casas, Cole Gulino, Simon Suo, Katie Luo, Renjie Liao, and Raquel Urtasun. Implicit Latent Variable Model for Scene-Consistent Motion Forecasting. In *European Conference on Computer Vision*, 2020.

Xi Chen, Diederik P. Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational Lossy Autoencoder. In *International Conference on Learning Representations*, 2017.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv preprint arXiv:1406.1078*, 2014.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C. Courville, and Yoshua Bengio. A Recurrent Latent Variable Model for Sequential Data. In *Advances in Neural Information Processing Systems*, pages 2980–2988, 2015.

Uwe Dick, Maryam Tavakol, and Ulf Brefeld. Rating Player Actions in Soccer. *Frontiers in Sports and Active Living*, 3, 2021.

Samuel G. Fadel, Sebastian Mair, Ricardo da Silva Torres, and Ulf Brefeld. Contextual movement models based on normalizing flows. *Advances in Statistical Analysis*, 2021.

Panna Felsen, Patrick Lucey, and Sujoy Ganguly. Where Will They Go? Predicting Fine-Grained Adversarial Multi-Agent Motion using Conditional Variational Autoencoders. In *European Conference on Computer Vision*, pages 732–747, 2018.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-Predict: Parallel Decoding of Conditional Masked Language Models. In *Conference on Empirical Methods in Natural Language Processing*, April 2019.

Laurent Girin, Simon Leglaive, Xiaoyu Bie, Julien Diard, Thomas Hueber, and Xavier Alameda-Pineda. Dynamical Variational Autoencoders: A Comprehensive Review. *arXiv preprint arXiv:2008.12595*, 2020.

Yedid Hoshen. Vain: Attentional Multi-Agent Predictive Modeling. In *Advances in Neural Information Processing Systems*, pages 2701–2711, 2017.

Oleg Ivanov, Michael Figurnov, and Dmitry Vetrov. Variational Autoencoder with Arbitrary Conditioning. In *International Conference on Learning Representations*, 2019.

Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*, 2014.

Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural Relational Inference for Interacting Systems. In *International Conference on Machine Learning*, pages 2693–2702, 2018.

Hugo Larochelle and Iain Murray. The Neural Autoregressive Distribution Estimator. In *International Conference on Artificial Intelligence and Statistics*, pages 29–37, 2011.

Hoang M. Le, Yisong Yue, and Peter Carr. Coordinated Multi-Agent Imitation Learning. In *International Conference on Machine Learning*, pages 1995–2003, 2017.

You Lu and Bert Huang. Structured Output Learning with Conditional Generative Flows. In *AAAI Conference on Artificial Intelligence*, 2020.

Svein Arne Pettersen, Pål Halvorsen, Dag Johansen, et al. Soccer Video and Player Position Dataset. In *ACM Multimedia Systems Conference*, pages 18–23, 2014.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *International Conference on Machine Learning*, pages 1278–1286, 2014.

Yannick Rudolph, Ulf Brefeld, and Uwe Dick. Graph Conditional Variational Models: Too Complex for Multiagent Trajectories? In *(Workshop) Advances in Neural Information Processing Systems*, 2020.

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia

Hadsell, and Peter Battaglia. Graph Networks as Learnable Physics Engines for Inference and Control. In *International Conference on Machine Learning*, pages 4470–4479, 2018.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning Structured Output Representation using Deep Conditional Generative Models. In *Advances in Neural Information Processing Systems*, pages 3483–3491, 2015.

Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder Variational Autoencoders. In *Advances in Neural Information Processing Systems*, pages 3738–3746, 2016.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Karl Tuyls, Shayegan Omidshafiei, Paul Muller, Zhe Wang, Jerome Connor, Daniel Hennes, Ian Graham, William Spearman, Tim Waskett, Dafydd Steel, et al. Game Plan: What AI can do for Football, and What Football can do for AI. *Journal of Artificial Intelligence Research*, 71:41–88, 2021.

Benigno Uria, Iain Murray, and Hugo Larochelle. A Deep and Tractable Density Estimator. *International Conference on Machine Learning*, October 2014.

Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural Autoregressive Distribution Estimation. *The Journal of Machine Learning Research*, 17(1): 7184–7220, 2016.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A Generative Model for Raw Audio. *arXiv preprint arXiv:1609.03499*, 2016a.

Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel Recurrent Neural Networks. In *International Conference on Machine Learning*, 2016b.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057, 2015.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *Advances in Neural Information Processing Systems*, 2019.

Raymond A. Yeh, Alexander G. Schwing, Jonathan Huang, and Kevin Murphy. Diverse Generation for Multi-Agent Sports Games. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Poczós, Ruslan R. Salakhutdinov, and Alexander J. Smola. Deep Sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401, 2017.

Eric Zhan, Stephan Zheng, Yisong Yue, Long Sha, and Patrick Lucey. Generating Multi-Agent Trajectories using Programmatic Weak Supervision. In *International Conference on Learning Representations*, 2019.

Shengjia Zhao, Jiaming Song, and Stefano Ermon. InfoVAE: Balancing Learning and Inference in Variational Autoencoders. In *AAAI Conference on Artificial Intelligence*, volume 33, pages 5885–5892, 2019.

Supplementary Material: Modeling Conditional Dependencies in Multiagent Trajectories

A ADDITIONAL EXPERIMENTS

In this section, we present the results of additional experiments, which refute the presumption that the success of CTSA on the 1 Hz data is connected to shorter sequence lengths. The results strengthen our conjecture, that the performance of CTSA on the 1 Hz data is rather due to the low data frequency, which probably comes with more conditional dependencies in the joint distribution over movements.

Since we fixed the lengths of the sequences we experimented with in Section 5 to five seconds each, but experimented wrt. different amounts of frames per second, the sequences at 5 Hz, 2.5 Hz and 1 Hz differ in terms of absolute timesteps. As we point out in Section 5, the resulting sequences have 25, 13 and 5 frames respectively. While the experiments in this section are still wrt. different Hz, we fix the amount of timesteps for sequences at all frequencies to five timesteps each. When generating trajectories from the models, we condition on two *frames* of data and generate the final three *frames* (instead of *seconds*, as in Section 5).

We present our findings in Table 3. The results for the experiment on the 1 Hz data carry over one to one from Section 5, as the experiment was run with five frames in the first place. For the 2.5 Hz and 5 Hz experiments, we cut the sequences to the first five frames each. This way, the models train on the same amount of timesteps per sequence and on the same overall amount of data in experiments at all three frequencies.

Overall, we find that the additional experiments support the results presented in Section 5. While CTSA is best at 1 Hz, and thus probably models conditional dependencies better than the other models, it does not provide benefits at higher frequencies. Instead, we find that all models operating on five frames sequences at 5 Hz and 2.5 Hz perform roughly the same. That CTSA results in the best log-likelihood for the 2.5 Hz data is in accordance with the experiments presented in Section 5, although it seems that the result is a bit more pronounced for the experiments that are presented in this section.

Unrelated to our findings, we want to note something wrt. L_2 deviations at 5 Hz, where the samples generated from the models have larger mean and final deviations than the velocity baseline: We find that this result is due to the regularity in the data and the noise that is introduced via sampling. If we generate the trajectories wrt. *means* of intra-timestep distributions rather than wrt. *samples*, we find that all three models are consistently better than the baseline.

B MODEL ARCHITECTURES

In this section, we describe implementation details and especially the model architectures which we used in our experiments on the soccer data. Figures 6 through 9 sketch our feature extraction process, the GNN, GVNN and CTSA models, as well as the mixture density network part (MDN, Bishop, 1994). All operations other than the transformer and CTSA layers are applied to individual nodes.

B.1 Feature Extraction

As noted in Section 5, for every timestep t we encode past trajectories into representations $\mathbf{h}^t$ by means of an RNN with GRU cells (Cho et al., 2014). Specifically, we chose to use two layers with 256 dimensional hidden state. We further provide the models with current positions $\mathbf{x}^{t-1}$, as well as last time differences $\Delta\mathbf{x}^{t-1}$, on both of which we apply simple feed-forward networks for feature extraction. The input to the GNN, GVNN and CTSA models is a concatenation of the representation extracted by the RNN with the representations of the current positions as well as the last time differences. Positions are normalized such that the pitch ranges from -1 to 1 in both, x and y dimension. Time differences are standardized wrt. the training data; this also holds for targets, as used in the GVNN and CTSA models. See Figure 6 for a visualization, including details on the feed-forward networks. The output of the feature extraction is fed into the step models. This is also where role embeddings are added (see Figures 7 and 8 for details).

B.2 GNN, GVNN and CTSA Step Models

The GNN, GVNN and CTSA components make up the core of our models. Architectures for the GNN and CTSA component are depicted in Figure 7, the architecture for the GVNN is depicted in Figure 8. All operations other than transformer and CTSA layers are applied to individual nodes.

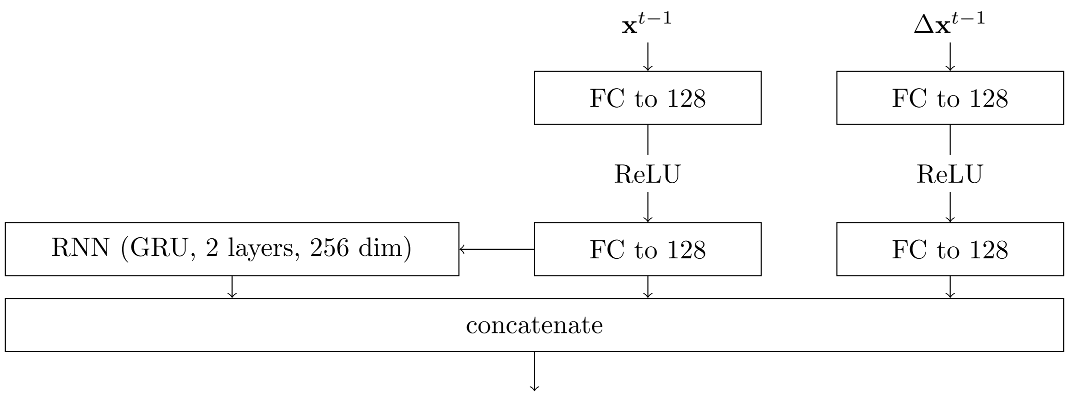


Figure 6: Feature extraction (FC = fully-connected layers, ReLU = rectified linear unit functions)

For our implementation of CTSA, we intentionally choose to stay as close as possible to a standard transformer encoder-decoder architecture. To that end, we use standard transformer layers (or *blocks*, including all residual-, normalization- and positional feed-forward-layers, as introduced in Vaswani et al., 2017) in the encoder. For the CTSA decoder, we also share most details with the transformer architecture. In principle we only add the *query-stream* in the decoder and ensure appropriate masking. We share parameters of transformer blocks for queries and targets. Due to the above choices, we decided to implement the GNN and GVNN with transformer encoder layers as well. As stated in Section 5: all three models have been designed to have roughly 4.5 million parameters and were trained with Adam (Kingma and Ba, 2014) at a learning rate of 0.001 and standard parameters otherwise. For all models we fairly tweaked architectures and experimented with different levels of regularization via dropout (cf. Srivastava et al., 2014) in the transformer layers, as well as with different learning rates in preliminary experiments.

In all transformer layers, we ended up using 256 dimensional node representations and eight heads. We set the dimensionality for positional feed-forward networks to be 512. While attention within the encoders is always unmasked, we found that it helped with training when we restrict the attention layers to only the nodes which result in outputs (i.e. the nodes corresponding to the ten defending field players) for the later layers. For the GVNN, we employ a 32 dimensional latent variable that is modeled with a Gaussian with diagonal covariance matrix. It turns out, that a dropout rate of 0.1 in the transformer and CTSA layers provided good regularization for all models. Note however, that we further regularize individual runs by early stopping wrt. log-likelihood or ELBO values on the validation data.

B.3 Mixture Density Network

As stated in Section 5, output distributions for individual agents, i.e. $p(\Delta\mathbf{x}_k^t|\dots)$, are modeled with a mixture density network approach with ten mixture components. Given the trajectory data, components are two-dimensional. We further chose to restrict the models to use diagonal covariance matrices. The MDN part of our models is depicted in Figure 9.

C PSEUDO CODE

With Algorithm 1, we provide some pseudo code for the implementation of CTSA. Specifically, we sketch neural network modules to be defined, the initialization and the forward pass through the CTSA architecture. For the backward pass we can rely on back-propagation; hence pseudo code for the backward pass is not provided. Note, that the code only describes the CTSA part of our architecture and that the whole architecture is trained end-to-end. Further note, that within the pseudo code we use the notation introduced in Section 4 but depart from it in minor details: For succinctness, we refrain from using *tilde* symbols and *out* subscripts for updated representations. What we refer to as *context mask* in the pseudo code is depicted as ∼*target mask* in Figure 3.

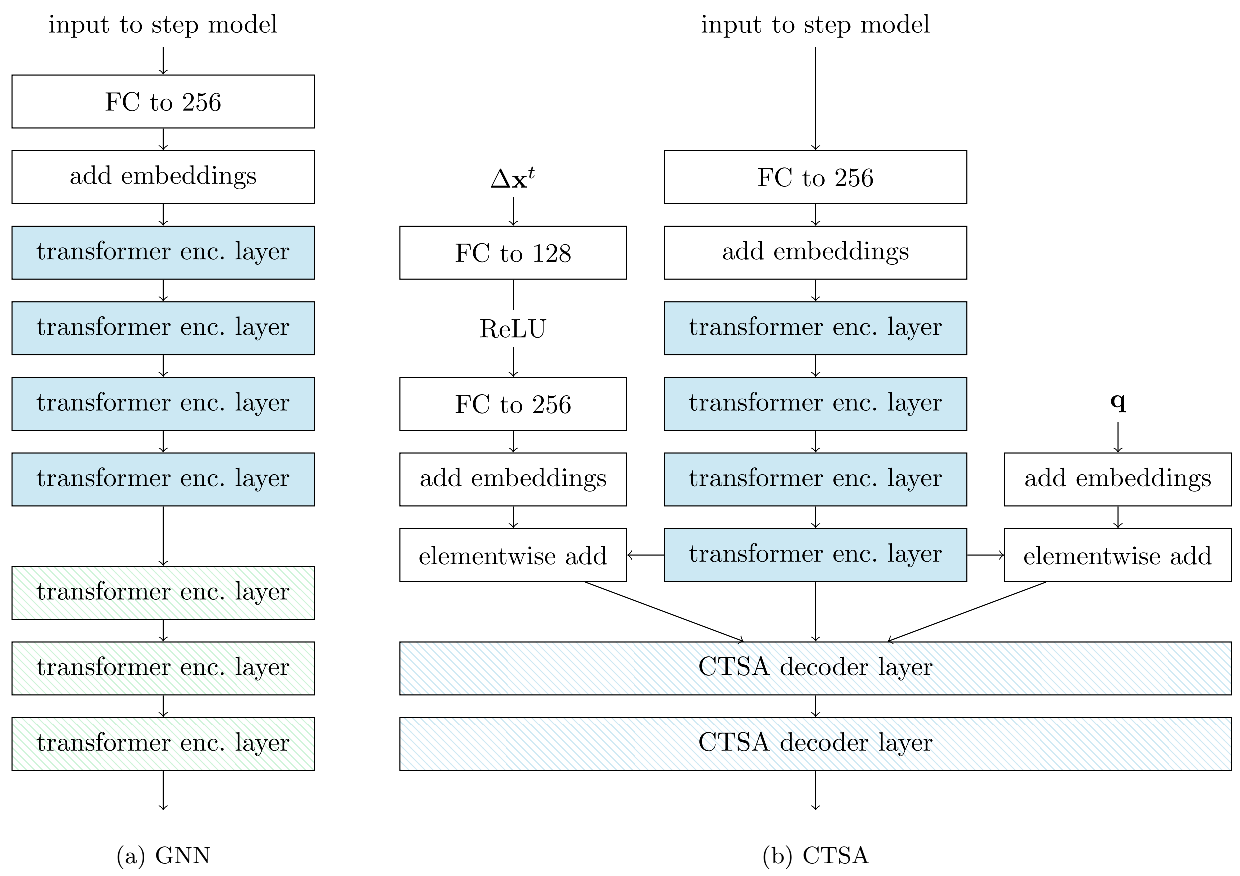
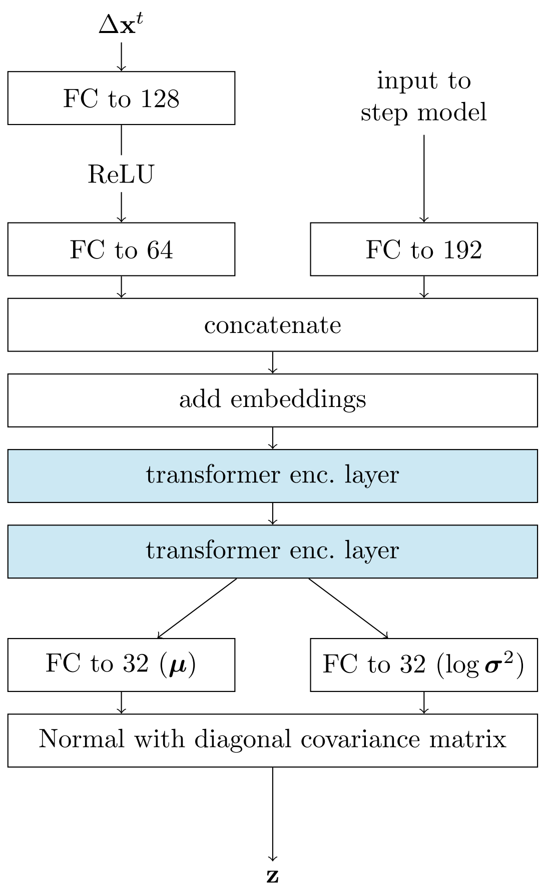


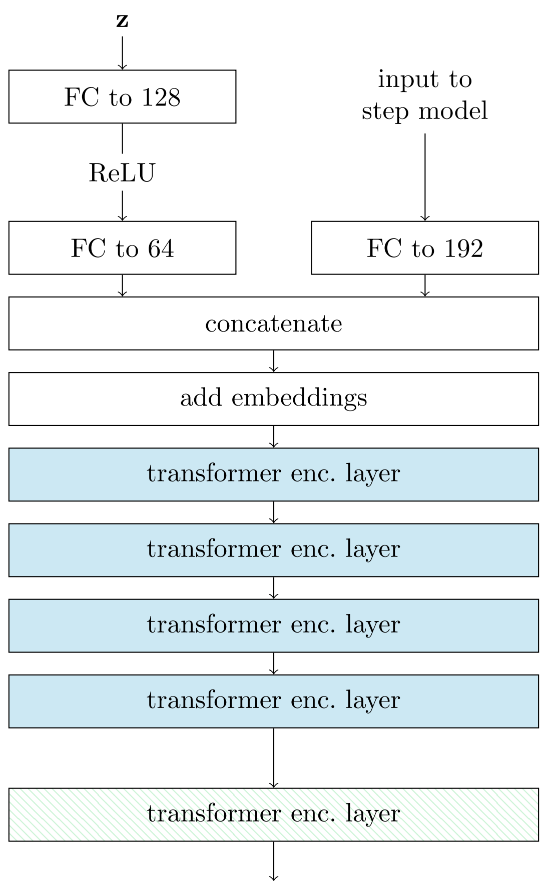
Figure 7: GNN and CTSA models (in blue transformer encoder layers attention is wrt. all nodes; striped green: attention only wrt. output nodes; striped blue highlights CTSA layers with autoregressive attention)

Table 3: Results for experiments at different frequencies, but with sequence lengths fixed to five timesteps each. For details about methods, baselines and evaluation please refer to Section 5. We again want to point out, that metrics are *not* comparable over experiments at different Hz. Further: LL = log-likelihood, SE = standard error, and L_2 deviations are in meter.

		LL		Avg. L_2		Avg. L_2 (best)		Fin. L_2		Fin. L_2 (best)	
Hz	Step model	Mean	SE	Mean	SE	Mean	SE	Mean	SE	Mean	SE
1	Velocity			3.35	±0.01			5.58	±0.01		
1	CTSA	**−1.49**	±0.01	**2.24**	±0.01	**1.81**	±0.01	**3.31**	±0.02	**2.61**	±0.01
1	GVNN	−1.65	±0.00	2.29	±0.01	1.90	±0.01	3.39	±0.02	2.74	±0.01
1	GNN	−1.65	±0.00	2.29	±0.02	1.90	±0.02	3.39	±0.03	2.74	±0.03
2.5	Velocity			0.76	±0.00			1.30	±0.00		
2.5	CTSA	**1.21**	±0.01	**0.70**	±0.00	**0.57**	±0.00	**1.13**	±0.00	**0.91**	±0.00
2.5	GVNN	1.17	±0.01	**0.71**	±0.00	**0.58**	±0.00	**1.13**	±0.01	**0.92**	±0.01
2.5	GNN	1.17	±0.01	**0.70**	±0.00	**0.58**	±0.00	**1.12**	±0.01	**0.91**	±0.01
5	Velocity			0.23	±0.00			0.39	±0.00		
5	CTSA	**3.30**	±0.04	**0.25**	±0.00	**0.20**	±0.00	**0.41**	±0.00	**0.34**	±0.00
5	GVNN	**3.32**	±0.02	**0.25**	±0.00	**0.21**	±0.00	**0.42**	±0.00	**0.34**	±0.00
5	GNN	3.26	±0.00	**0.25**	±0.00	**0.21**	±0.00	**0.41**	±0.00	**0.34**	±0.00



(a) Encoder



(b) Decoder

Figure 8: GVNN model (32 dimensional latent variable; in blue layers attention is wrt. all nodes; striped green: attention only wrt. output nodes; $\boldsymbol{\mu}$ denotes the mean and $\boldsymbol{\sigma}^2$ denotes the variance of the variational distribution)

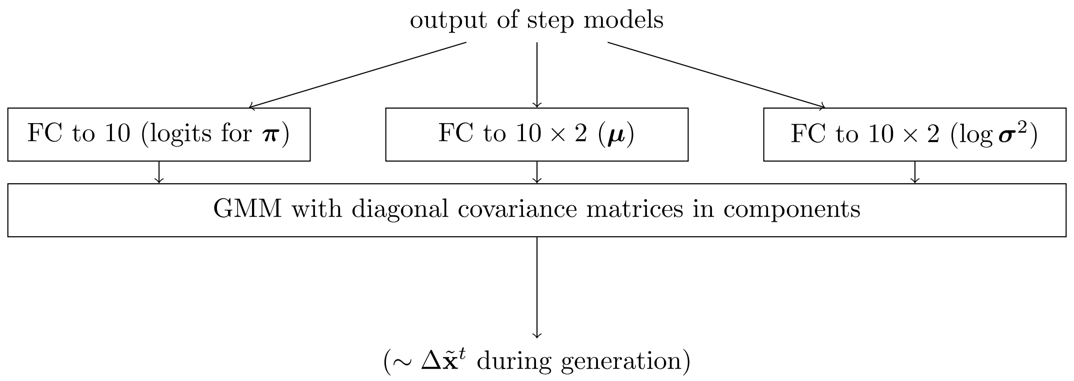

output of step models
FC to 10 (logits for $\boldsymbol{\pi}$)
FC to 10×2 ($\boldsymbol{\mu}$)
FC to 10×2 ($\log \boldsymbol{\sigma}^2$)
GMM with diagonal covariance matrices in components
($\sim \Delta\tilde{\mathbf{x}}^t$ during generation)

Figure 9: Mixture density network (with 10 mixture components; $\boldsymbol{\pi}$ are mixture weights, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ denote means and variances respectively)

Algorithm 1 CTSA pseudo code

```
Definitions:                                   ▷ define stateful procedures / neural network modules
  procedure TransformerEncoder(context)
      ...
  end procedure                                                  ▷ standard transformer encoder

  procedure MaskedMultiHeadAttention(query, key, value, mask)
      ...
  end procedure      ▷ standard transformer multi-head attention, attending to values according to mask

  procedure QueryStreamLayer(query, target, context, query mask, context mask)
      query ← MaskedMultiHeadAttention(query, target, target, query mask)
      ...
      query ← MaskedMultiHeadAttention(query, context, context, context mask)
      ...
  end procedure  ▷ standard transformer decoder layer (including all residual-, normalization-, dropout- and
  positional feed-forward-layers) with custom calls to MaskedMultiHeadAttention

  procedure TargetStreamLayer(target, context, target mask, context mask)
      target ← MaskedMultiHeadAttention(target, target, target, target mask)
      ...
      target ← MaskedMultiHeadAttention(target, context, context, context mask)
      ...
  end procedure  ▷ standard transformer decoder layer (including all residual-, normalization-, dropout- and
  positional feed-forward-layers) with custom calls to MaskedMultiHeadAttention

Initialization:
  initialize shared query embedding $\mathbf{q}$ (i.e. $\mathbf{q}_1 = \cdots = \mathbf{q}_K$) and necessary transformer modules

Forward pass:
  for instances $\mathbf{h}^t$ and $\phi(\Delta\mathbf{x}^t)$ do
      set $o$ randomly                                            ▷ shuffled integers 1 through $K$
      generate query mask, such that attention query with index $o_k$ attends only to values with indices $o_{<k}$
      generate target mask, such that attention query with index $o_k$ attends only to values with indices $o_{\leq k}$
      generate context mask, such that attention query with index $o_k$ attends only to values with indices $o_{>k}$
      $\mathbf{h}^t \leftarrow$ TransformerEncoder($\mathbf{h}^t$)
      $\mathbf{q}^t \leftarrow \mathbf{q} \oplus \mathbf{h}^t$
      $\phi(\Delta\mathbf{x}^t) \leftarrow \phi(\Delta\mathbf{x}^t) \oplus \mathbf{h}^t$
      for $i$ in number of decoder layers do
          $\mathbf{q}^t \leftarrow$ QueryStreamLayer$_i$($\mathbf{q}^t$, $\phi(\Delta\mathbf{x}^t)$, $\mathbf{h}^t$, query mask, context mask)
          $\phi(\Delta\mathbf{x}^t) \leftarrow$ TargetStreamLayer$_i$($\phi(\Delta\mathbf{x}^t)$, $\mathbf{h}^t$, target mask, context mask)
      end for
      $\psi^t \leftarrow \mathbf{q}^t$
  end for
```