`baller2vec`: A Multi-Entity Transformer For Multi-Agent Spatiotemporal Modeling

Michael A. Alcorn [1] **Anh Nguyen** [1]

Abstract

Multi-agent spatiotemporal modeling is a challenging task from both an algorithmic design and computational complexity perspective. Recent work has explored the efficacy of traditional deep sequential models in this domain, but these architectures are slow and cumbersome to train, particularly as model size increases. Further, prior attempts to model interactions between agents across time have limitations, such as imposing an order on the agents, or making assumptions about their relationships. In this paper, we introduce `baller2vec`, a multi-entity generalization of the standard Transformer that, with minimal assumptions, can *simultaneously and efficiently* integrate information across entities and time. We test the effectiveness of `baller2vec` for multi-agent spatiotemporal modeling by training it to perform two different basketball-related tasks: (1) simultaneously forecasting the trajectories of all players on the court and (2) forecasting the trajectory of the ball. Not only does `baller2vec` learn to perform these tasks well, it also appears to "understand" the game of basketball, encoding idiosyncratic qualities of players in its embeddings, and performing basketball-relevant functions with its attention heads.

1. Introduction

In a variety of settings, individuals attempt to predict phenomena that arise from multiple entities interacting through time, whether that is a defender anticipating where the point guard will make a pass in a basketball game, a marketing professional guessing the next trending topic on a social network, or a theme park manager forecasting the flow of visitor traffic. Researchers designing algorithms to perform such tasks face two main challenges:

1. Given that entities lack a natural ordering, how do you effectively model interactions between entities across time?
2. How do you *efficiently* learn from the large, high-dimensional inputs inherent to such sequential data?

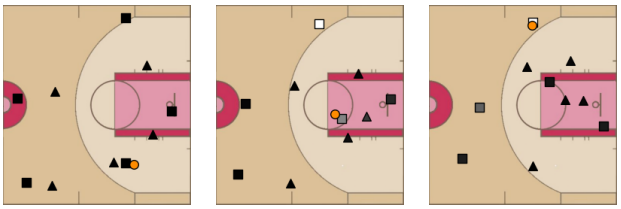


Figure 1. After solely being trained to predict the trajectory of the ball (●) given the locations of the players and the ball on the court through time, a self-attention (SA) head in `baller2vec` learned to anticipate passes. When the ball handler (■●) is driving towards the basket at $t = 3$, SA assigns near-zero weights (black) to all players, suggesting no passes will be made. Interestingly, the ball handler indeed did not pass and dribbled into the lane. At $t = 7$, SA assigns a high weight (white) to a teammate (□), which correctly identifies the recipient of the pass at $t = 12$.

Prior work in athlete trajectory forecasting, a widely studied application of multi-agent spatiotemporal modeling (MASM; where entities are agents moving through space), has attempted to model player interactions through "role-alignment" preprocessing steps (i.e., imposing an order on the players) (Felsen et al., 2018; Zhan et al., 2019) or graph neural networks (Yeh et al., 2019), but these approaches may destroy identity information in the former case (see Section 4.2) or limit personalization in the latter case (see Section 5.1). Recently, researchers have experimented with variational recurrent neural networks (VRNNs) (Chung et al., 2015) to model the temporal aspects of player trajectory data (Yeh et al., 2019; Zhan et al., 2019), but the inherently sequential design of this architecture limits the size of models that can feasibly be trained in experiments.

Transformers (Vaswani et al., 2017) were designed to circumvent the computational constraints imposed by other sequential models, and they have achieved state-of-the-art results in a wide variety of sequence learning tasks, both

[1] Department of Computer Science and Software Engineering, Auburn University, Auburn, Alabama, USA. Correspondence to: Michael A. Alcorn <alcorma@auburn.edu> and Anh Nguyen <anh.ng8@gmail.com>.

in natural language processing (NLP), e.g., GPT-3 (Brown et al., 2020), and computer vision, e.g., Vision Transformers (Dosovitskiy et al., 2021). While Transformers have successfully been applied to *static* multi-entity data, e.g., graphs (Veličković et al., 2018), the only published work we are aware of that attempts to model multi-entity *sequential* data with Transformers uses four different Transformers to *separately* process information temporally and spatially before merging the sub-Transformer outputs (Yu et al., 2020).

In this paper, we introduce a *multi-entity* Transformer that, with minimal assumptions, is capable of *simultaneously* integrating information across agents and time, which gives it powerful representational capabilities. We adapt the original Transformer architecture to suit multi-entity sequential data by converting the standard self-attention matrix used in NLP tasks into a novel self-attention *tensor*. To test the effectiveness of our multi-entity Transformer for MASM, we train it to perform two different basketball-related tasks (hence the name `baller2vec`): (1) simultaneously forecasting the trajectories of all players on the court (**Task P**) and (2) forecasting the trajectory of the ball (**Task B**). Further, we convert these tasks into classification problems by discretizing the trajectory space, which allows `baller2vec` to learn complex, multimodal trajectory distributions via strictly maximizing the likelihood of the data (in contrast to variational approaches, which maximize the evidence lower bound and thus require priors over the latent variables). We find that:

1. `baller2vec` is an effective algorithm for MASM, obtaining a perplexity of 1.68 on **Task P** (compared to 16.90 when simply using the label distribution from the training set) and 14.57 on **Task B** (vs. 322.40) (Section 4.1).
2. `baller2vec` demonstrably integrates information across *both* agents and time to achieve these results, as evidenced by ablation experiments (Section 4.2).
3. The identity embeddings learned by `baller2vec` capture idiosyncratic qualities of players, indicative of the model's deep personalization capabilities (Section 4.3).
4. `baller2vec`'s trajectory forecast distributions depend on both the historical and current context (Section 4.4), and several attention heads appear to perform different basketball-relevant functions (Figure 1; Section 4.5), which suggests the model learned to "understand" the sport.

2. Methods

2.1. Multi-entity sequences

Let $A = \{1, 2, \ldots, B\}$ be a set indexing B entities and $P = \{p_1, p_2, \ldots, p_K\} \subset A$ be the K entities involved in a particular sequence. Further, let $Z_t = \{z_{t,1}, z_{t,2}, \ldots, z_{t,K}\}$ be an *unordered* set of K feature vectors such that $z_{t,k}$ is the feature vector at time step t for entity p_k. $\mathcal{Z} = (Z_1, Z_2, \ldots, Z_T)$ is thus an *ordered* sequence of sets of feature vectors over T time steps. When $K = 1$, $\mathcal{Z}$ is a sequence of individual feature vectors, which is the underlying data structure for many NLP problems.

We now consider two different tasks: (1) sequential entity labeling, where each entity has its own label at each time step (which is conceptually similar to word-level language modeling), and (2) sequential labeling, where each time step has a single label (see Figure 3). For (1), let $\mathcal{V} = (V_1, V_2, \ldots, V_T)$ be a sequence of sets of labels corresponding to $\mathcal{Z}$ such that $V_t = \{v_{t,1}, v_{t,2}, \ldots, v_{t,K}\}$ and $v_{t,k}$ is the label at time step t for the entity indexed by k. For (2), let $W = (w_1, w_2, \ldots, w_T)$ be a sequence of labels corresponding to $\mathcal{Z}$ where w_t is the label at time step t. The goal is then to learn a function $f(\mathcal{Z})$ that maps a set of entities and their time-dependent feature vectors to a probability distribution over either (1) the entities' time-dependent labels or (2) the sequence of labels. Here, we use a multi-entity Transformer (described in Section 2.3) for our f.

2.2. Multi-agent spatiotemporal modeling

In the MASM setting, P is a set of K different agents and $C_t = \{(x_{t,1}, y_{t,1}), (x_{t,2}, y_{t,2}), \ldots, (x_{t,K}, y_{t,K})\}$ is an unordered set of K coordinate pairs such that $(x_{t,k}, y_{t,k})$ are the coordinates for agent p_k at time step t. The ordered sequence of sets of coordinates $\mathcal{C} = (C_1, C_2, \ldots, C_T)$, together with P, thus defines the trajectories for the K agents over T time steps. We then define $z_{t,k}$ as:

$$z_{t,k} = g([e_a(p_k), x_{t,k}, y_{t,k}, h_{t,k}]) \tag{1}$$

where g is a multilayer perceptron (MLP), e_a is an agent embedding layer, and $h_{t,k}$ is a vector of optional contextual features for agent p_k at time step t. The trajectory for agent p_k at time step t is defined as $(x_{t+1,k} - x_{t,k}, y_{t+1,k} - y_{t,k})$. Similar to Zheng et al. (2016), to fully capture the multimodal nature of the trajectory distributions, we discretize the 2D Euclidean space into an $n \times n$ grid (Figure 2) and treat the problem as a classification task. Therefore, $\mathcal{Z}$ has a corresponding sequence of sets of trajectory labels (i.e., $v_{t,k}$ is an integer from one to n^2), and the loss for each sample in **Task P** is:

$$\mathcal{L} = \sum_{t=1}^{T} \sum_{k=1}^{K} -\ln(f(P, \mathcal{C}_{1:t}, \mathcal{H}_{1:t})[v_{t,k}]) \tag{2}$$

where $\mathcal{H}$ is sequence of sets of contextual vectors corresponding to $\mathcal{C}$ and $f(P, \mathcal{C}_{1:t}, \mathcal{H}_{1:t})[v_{t,k}]$ is the probability assigned to agent p_k's trajectory label at time step t by f, i.e., (2) is the negative log-likelihood (NLL) of the data according to the model.

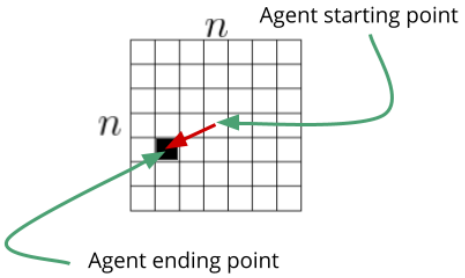


Figure 2. An example of a discretized trajectory. The agent's starting position is in the center cell, and the cell containing the agent's ending position is used as the label (of which there are n^2 possibilities).

For **Task B**, we optimize the following loss for each sample:

$$\mathcal{L} = \sum_{t=1}^{T} -\ln(f(P, \mathcal{C}_{1:t}, \mathcal{H}_{1:t})[w_t]) \tag{3}$$

where $f(P, \mathcal{C}_{1:t}, \mathcal{H}_{1:t})[w_t]$ is the probability assigned to the ball's trajectory label at time step t by f, and the labels correspond to a discretized 3D Euclidean space (i.e., w_t is an integer from one to n^3).

2.3. The multi-entity Transformer

We now describe our multi-entity Transformer, `baller2vec` (Figure 3). For NLP tasks, the Transformer self-attention mask M takes the form of a $T \times T$ matrix (Figure 4) where T is the length of the sequence. The element at M_{t_1,t_2} thus indicates whether or not the model can "look" at time step t_2 when processing time step t_1. We generalize the standard Transformer to the multi-entity setting by employing a $T \times K \times T \times K$ mask *tensor* where element M_{t_1,k_1,t_2,k_2} indicates whether or not the model can "look" at agent p_{k_2} at time step t_2 when processing agent p_{k_1} at time step t_1. In practice, we reshape M into a $TK \times TK$ matrix (Figure 4) to be compatible with typical Transformer implementations, and the input to the Transformer is a matrix with shape $TK \times F$ where F is the dimension of each $z_{t,k}$. Similar to Vaswani et al. (2017), to encode temporal position, we add *the same* position embedding e_t to the feature vector for *each* entity at each time step, i.e.:

$$\hat{z}_{t,k} = z_{t,k} + e_t \tag{4}$$

The remaining computations are identical to the standard Transformer (see implementation in Section S1).

3. Experiments

All data and code for the paper are available at `https://github.com/airalcorn2/baller2vec`.

3.1. Dataset

We trained `baller2vec` on a publicly available dataset of player and ball trajectories recorded during 631 National Basketball Association (NBA) games from the 2015-2016 season. All 30 NBA teams and 450 different players were represented. Because transition sequences are a strategically important part of basketball, unlike prior work, e.g., (Felsen et al., 2018; Yeh et al., 2019; Zhan et al., 2019), we did not terminate sequences on a change of possession, nor did we constrain ourselves to a fixed subset of sequences. Instead, each training sample was generated on the fly by first randomly sampling a game, and then randomly sampling a time point from that game. The following four seconds of data were downsampled to 5 Hz from the original 25 Hz and used as the input.

Because we did not terminate sequences on a change of possession, we could not "normalize" the direction of the court as was done in prior work (Felsen et al., 2018; Yeh et al., 2019; Zhan et al., 2019). Instead, for each sampled sequence, we randomly (with a probability of 0.5) rotated the court 180°(because the court's direction is arbitrary), effectively doubling the size of the dataset (we used a binary variable to indicate the side of the frontcourt for each player). As a result, we had access to ∼82 million different (albeit overlapping) training sequences (2 rotations × 569 games × 4 periods per game × 12 minutes per period × 60 seconds per minute × 25 Hz). We used a training/validation/test split of 569/30/32 games, respectively (i.e., 5% of the games were used for testing, and 5% of the remaining 95% of games were used for validation). For both the validation and test sets, ∼1,000 different, *non-overlapping* sequences were selected for evaluation by dividing each game into $\lfloor \frac{1{,}000}{N} \rfloor$ non-overlapping chunks (where N is the number of games), and using the starting four seconds from each chunk as the evaluation sequence.

3.2. Model

We trained separate models for **Task P** and **Task B**. For all experiments, we used a single Transformer architecture that was nearly identical to the original model described in Vaswani et al. (2017), with $d_{\text{model}} = 512$ (the dimension of the input and output of each Transformer layer), eight attention heads, $d_{\text{ff}} = 2048$ (the dimension of the inner feedforward layer), and six layers, although we did not use dropout, and we used learned embeddings to encode position instead of sine/cosine vector functions. For *both* **Task P** and **Task B**, the model was provided the players *and* the ball as input. Both the players and the ball were embedded to 20-dimensional vectors. The input features for each player consisted of his identity, (x, y) coordinates on the court at each time step in the sequence, and a binary variable indicating the side of his frontcourt. The input

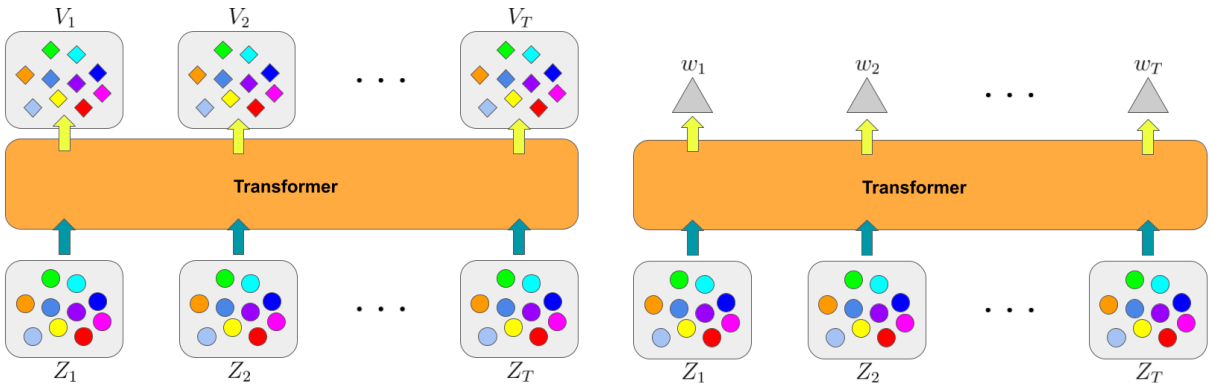


Figure 3. An overview of the multi-entity Transformer. Each time step t consists of an *unordered* set of feature vectors Z_t (colored circles) with either (left) a corresponding set of entity labels V_t (colored diamonds) or (right) a single label w_t. Here, each circle represents the input features for a different entity (in our experiments, a basketball player) at a specific time step, and matching colored circles/diamonds across time steps correspond to the same entity.

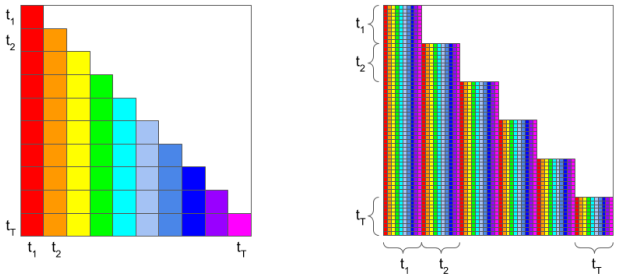


Figure 4. Left: the standard self-attention mask matrix M. The element at M_{t_1,t_2} indicates whether or not the model can "look" at time step t_2 when processing time step t_1. Right: the matrix form of our multi-entity self-attention mask tensor. In tensor form, element M_{t_1,k_1,t_2,k_2} indicates whether or not the model can "look" at agent p_{k_2} at time step t_2 when processing agent p_{k_1} at time step t_1. In matrix form, this corresponds to element $M_{t_1K+k_1,t_2K+k_2}$ when using zero-based indexing. The M shown here is for a static, fully connected graph, but other, potentially evolving network structures can be encoded in the attention tensor.

features for the ball were its (x, y, z) coordinates at each time step.

The input features for the players and the ball were processed by separate, three layer MLPs before being fed into the Transformer. Each MLP had 128, 256, and 512 nodes in its three layers, respectively, and a ReLU nonlinearity following each of the first two layers. Similarly, separate positional embeddings were used to encode the temporal order of the player and ball Transformer inputs. A single linear layer on top of the Transformer output followed by a softmax was used for classification. For players, we discretized an 11 ft $\times$ 11 ft 2D Euclidean trajectory space into an 11 $\times$ 11 grid of 1 ft $\times$ 1 ft squares for a total of 121 player trajectory labels (Figure 5 shows the distribution of the labels for the entire dataset). Similarly, for the ball, we discretized a 19 ft $\times$ 19 ft $\times$ 19 ft 3D Euclidean trajectory space into an 19 $\times$ 19 $\times$ 19 grid of 1 ft $\times$ 1 ft $\times$ 1 ft cubes for a total of 6,859 ball trajectory labels.

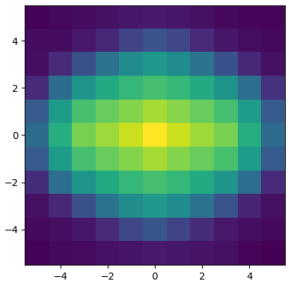


Figure 5. A log-normalized heat map of the discretized player trajectory labels in the full NBA dataset. The center cell corresponds to the "stationary" trajectory (i.e., the player did not move further than 0.5 ft in either the x or y direction). The elongated shape of the distribution reflects the rectangular shape of the court.

We used the Adam optimizer (Kingma & Ba, 2015) with a learning rate of 10^{-6}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-9}$ to update the model's parameters, of which there were $\sim$23 million. Models were implemented in PyTorch and trained on a single NVIDIA GTX 1080 Ti GPU for $\sim$250 epochs where each epoch consisted of 20,000 training samples, and the validation set was used for early stopping.

3.3. Ablation studies

To assess the impact of the multi-entity design of `baller2vec` and the inclusion of player identity on model performance, we trained three variations of our player trajectory forecasting model: (1) one player as input without player identity, (2) all 10 players as input without player identities, and (3) all 10 players as input with player identities. In experiments where player identity was not used, a single generic player embedding was used in place of the player identity embeddings. We also trained two variations of our ball trajectory forecasting model: one with player

identity and one without. Lastly, to determine the extent to which baller2vec uses historical information in its forecasts, we compared the model's average NLL for the full sequence test set on **Task P** to the model's average NLL for the test set when only predicting the trajectories for the first frame.

4. Results

4.1. baller2vec is an effective algorithm for multi-agent spatiotemporal modeling.

The average NLL on the test set for our best **Task P** model was 0.519, while the average NLL for our best **Task B** model was 2.717. In NLP, model performance is often expressed in terms of the "perplexity" per word, which, intuitively, is the number of faces on a fair die that has the same amount of uncertainty as the model per word (i.e., a uniform distribution over M labels has a perplexity of M, so a model with a per word perplexity of six has the same amount of uncertainty as rolling a fair six-sided die). In our case, we consider the perplexity per trajectory, defined as:

$$PP = e^{\frac{1}{NTK}\sum_{n=1}^{N}\sum_{t=1}^{T}\sum_{k=1}^{K} - \ln(p(v_{n,t,k}))} \quad (5)$$

where N is the number of sequences. Our best **Task P** model achieved a perplexity per trajectory of 1.68, i.e., baller2vec was, on average, as uncertain as rolling a 1.68-sided fair die (better than a coin flip) when predicting player trajectories. For comparison, when using the distribution of the player trajectory labels in the training set as the predicted probabilities, the perplexity on the test set was 16.90. Our best **Task B** model achieved a perplexity per trajectory of 14.57 (compared to 322.40 when using the training set distribution).

4.2. baller2vec uses information about all players on the court through time, in addition to player identity, to model spatiotemporal dynamics.

Results for our ablation experiments can be seen in Table 1. Including all 10 players as input dramatically improved the performance of our **Task P** model by 12.1% over only using a single player. Including player identity improved the model's performance a further 3.8%. This stands in contrast to Felsen et al. (2018) where the inclusion of player identities led to slightly *worse* model performance, a counterintuitive result given the range of skills among NBA players, but possibly a side effect of their role-alignment procedure. Interestingly, including player identity as input for **Task B** only improved the model's performance by 1.1%. Lastly, the model's average NLL on **Task P** for the full sequence test set (0.519) was 71% lower than its average NLL for the single frame test set (1.78), i.e., baller2vec is clearly using historical information to model the spatiotemporal dynamics of basketball.

Table 1. The average NLL on the test set for each of the models in our ablation experiments (lower is better). For **Task P**, using all 10 players improved model performance by 12.1%, while using player identity improved model performance by an additional 3.8%. For **Task B**, using player identity improved model performance by 1.1%. 1/10 indicates whether one or 10 players were used as input, respectively, while I/NI indicates whether or not player identities were used, respectively.

Task	1-NI	10-NI	10-I
Task P	0.613	0.539	0.519
Task B	N/A	2.709	2.679

4.3. Player embeddings encode individual attributes.

Neural language models are widely known for their ability to encode semantic relationships between words and phrases as geometric relationships between embeddings—see, e.g., (Mikolov et al., 2013b;a; Le & Mikolov, 2014; Sutskever et al., 2014). Alcorn (2018) observed a similar phenomenon in a baseball setting, where batters and pitchers with similar skills were found next to each other in the embedding space learned by a neural network trained to predict the outcome of an at-bat. Figure 6 displays a 2D UMAP (McInnes et al., 2018) generated from the player embeddings learned by baller2vec for **Task B**. Like (batter|pitcher)2vec (Alcorn, 2018), baller2vec seems to encode skills and physical attributes in its player embeddings.

Querying the nearest neighbors for individual players reveals further insights about the baller2vec embeddings. For example, the nearest neighbor for Russell Westbrook, an extremely athletic 6'3" point guard, is Derrick Rose, a 6'2" point guard also known for his athleticism (Figure 7). Amusingly, the nearest neighbor for Pau Gasol, a 7'1" center with a respectable shooting range, is his younger brother Marc Gasol, a 6'11" center, also with a respectable shooting range.

4.4. baller2vec's trajectory forecast distributions depend on both the historical and current context.

Because baller2vec *explicitly* models the distribution of the player trajectories (unlike variational methods), we can easily visualize how its trajectory forecast distributions shift in different situations. As can be seen in Figure 8, baller2vec's trajectory forecast distributions depend on both the historical and current context. When provided with limited historical information, baller2vec tends to be less certain about where the players might go. baller2vec also tends to be more certain when forecasting "easy" trajectories (e.g., a player moving into open

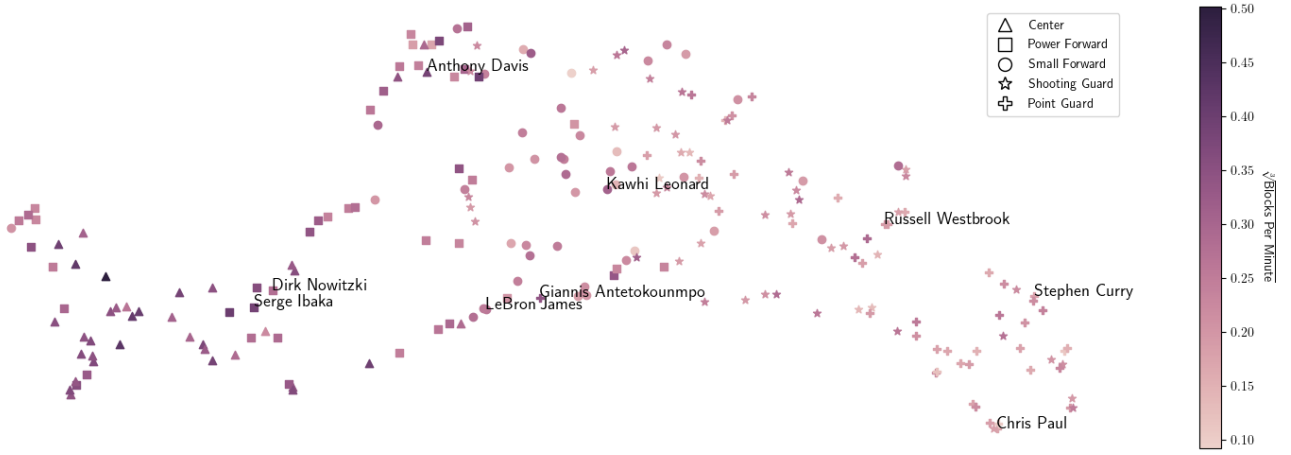


Figure 6. By exclusively learning to predict the trajectory of the ball, `baller2vec` was able to infer idiosyncratic player attributes (as can be seen in this 2D UMAP of the player embeddings). The left-hand side of the plot contains tall post players (△, □), e.g., Serge Ibaka, while the right-hand side of the plot contains shorter shooting guards (☆) and point guards (+), e.g., Stephen Curry. The connecting transition region contains forwards (□, ○) and other "hybrid" players, i.e., individuals possessing both guard and post skills, e.g., LeBron James. Further, players with similar defensive abilities, measured here by the cube root of the players' blocks per minute in the 2015-2016 season (Basketball-Reference.com, 2021), cluster together.

Figure 7. Nearest neighbors in `baller2vec`'s embedding space are plausible doppelgängers, such as the explosive point guards Russell Westbrook and Derrick Rose, and seven-foot tall brothers Pau and Marc Gasol. Images credits can be found in Table S1.

space) vs. "hard" trajectories (e.g., an offensive player choosing which direction to move around a defender). Similarly, as can be seen in Figure 9, player trajectories *generated* by `baller2vec` are noisier when conditioned on less historical context.

4.5. Different attention heads in `baller2vec` appear to perform different basketball-relevant functions.

One intriguing property of the attention mechanism (Graves, 2013; Graves et al., 2014; Weston et al., 2015; Bahdanau et al., 2015) is how, when visualized, the attention scores often seem to reveal how a model is "thinking". For example, Vaswani et al. (2017) discovered examples of attention heads in their Transformer that appear to be performing various language understanding subtasks, such as anaphora resolution. As can be seen in Figure 10, some of the attention heads in `baller2vec` seem to be performing basketball understanding subtasks, such as keeping track of the ball handler's teammates, and anticipating who the ball handler will pass to, which, intuitively, helps with our task of predicting the ball's trajectory.

5. Related Work

5.1. Trajectory modeling in sports

There is a rich literature on MASM, particularly in the context of sports, e.g., (Kim et al., 2010; Zheng et al., 2016; Le et al., 2017b;a; Qi et al., 2020; Zhan et al., 2020). Most relevant to our work is Yeh et al. (2019), which used a variational recurrent neural network combined with a graph neural network to forecast trajectories in a multi-agent setting. Like their approach, our model is permutation invariant with regard to the ordering of the agents; however, we use a multi-head attention mechanism to achieve this permutation invariance while the permutation invariance in Yeh et al. (2019) is provided by the graph neural network. Specifically, Yeh et al. (2019) define:

$$v \to e : \mathbf{e}_{i,j} = f_e([\mathbf{v}_i, \mathbf{v}_j, \mathbf{t}_{i,j}]) \tag{6}$$

$$e \to v : \mathbf{o}_i = f_v(\sum_{j \in \mathcal{N}_i} [\mathbf{e}_{i,j}, \mathbf{t}_i]) \tag{7}$$

where $\mathbf{v}_i$ is the initial state of agent i, $\mathbf{t}_{i,j}$ is an embedding for the edge between agents i and j, $\mathbf{e}_{i,j}$ is the representation

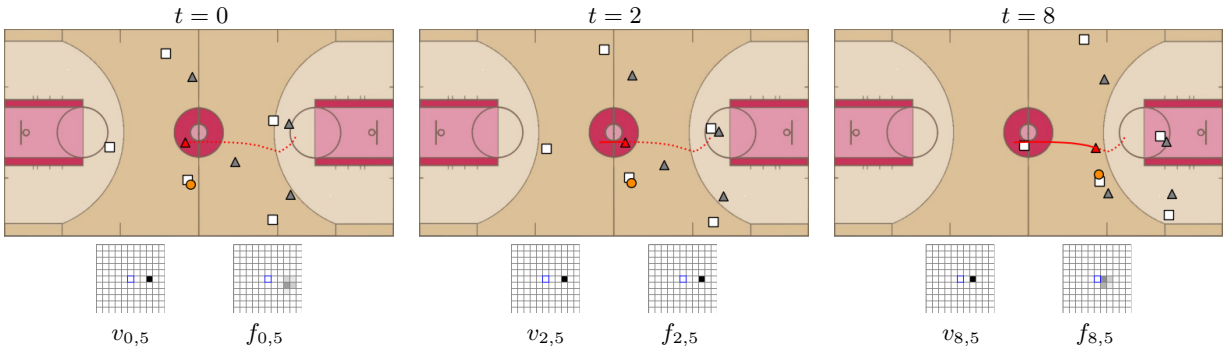
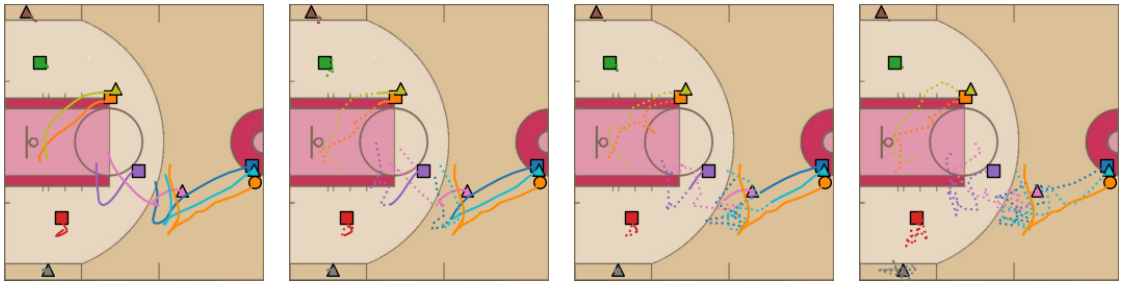
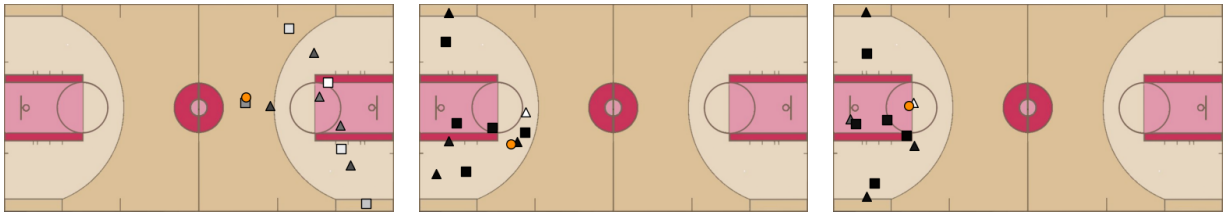


Figure 8. `baller2vec`'s trajectory forecast distributions are affected by both the historical and current context. At $t = 0$, `baller2vec` is fairly uncertain about the target player's (▲; $k = 5$) future trajectory (left grid and dotted red line; the blue bordered center cell is the "stationary" trajectory), with most of the probability mass (right grid; black = 1.0; white = 0.0) divided between trajectories moving towards the opponent's basket and where the ball handler appears to be headed. After observing a portion of the sequence ($t = 2$), `baller2vec` becomes very certain about the target player's trajectory ($f_{2,5}$), but when the ball handler makes a move past his defender ($t = 8$), `baller2vec` becomes split between trajectories (committing to the ball handler or staying in position for the trailing man). Additional examples can be found in Figure S1. ● = ball, □ = offense, ▲ = defense, and $f_{t,k} = f(P, \mathcal{C}_{1:t}, \mathcal{H}_{1:t})_{t,k}$.



(a) Ground truth. (b) 2 s observed. (c) 1 s observed. (d) 0 s observed.

Figure 9. With less historical information (solid lines), `baller2vec`'s generated trajectories (dotted lines) tend to be noisier. Each player is a different color, and the colored lines are their corresponding trajectories. The shapes are in the *starting* positions for the trajectories, and the ground truth location of the ball is used as input at each time step (i.e., the ball trajectories are not generated). △ = offense, □ = defense, and ● = ball.



(a) Attention head 2-7 (layer-head) appears to focus on teammates of the ball handler (■●).

(b) Attention head 6-2 seems to predict who the ball handler will be in future frames (△); see (c).

(c) In this frame, the highlighted player in (b) receives a pass from the previous ball handler.

Figure 10. The attention outputs from `baller2vec` suggest it learned basketball-relevant functions. Players are shaded according to the *sum* of the attention scores assigned to the players *through time* with reference to the ball in the current frame (recall that each player occurs multiple times in the input). Higher attention scores are lighter. For both of these attention heads, the sum of the attention scores assigned to the ball through time was small (0.01 for both the left and middle frames where the maximum is 1.00). Additional examples can be found in Figures S2 and S3.

for edge (i, j), $\mathcal{N}_i$ is the neighborhood for agent i, $\mathbf{t}_i$ is a node embedding for agent i, $\mathbf{o}_i$ is the output state for agent i, and f_e and f_v are deep neural networks.

Assuming *each individual player* is a different "type" in (6) (i.e., attempting to maximize the level of personalization) would require $450^2 = 202{,}500$ (i.e., B^2) different $t_{i,j}$ edge embeddings, many of which would never be used dur-

ing training and thus inevitably lead to poor out-of-sample performance. Reducing the number of type embeddings requires making assumptions about the nature of the relationships between nodes. By using a multi-head attention mechanism, our model learns to integrate information about different agents in a highly flexible manner that is both agent and time-dependent, and can generalize to unseen agent combinations.

Additionally, unlike recent works that use variational methods to train their generative models (Yeh et al., 2019; Felsen et al., 2018; Zhan et al., 2019), we translate the multi-agent trajectory forecasting problem into a classification task, which allows us to train our model by strictly maximizing the likelihood of the data. As a result, we do not make any assumptions about the distributions of the trajectories nor do we need to set any priors over latent variables. Zheng et al. (2016) also predicted discretized trajectories, but they included "macro-goals" (i.e., the approximate destination of the player) as input to their model.

5.2. Transformers for multi-agent spatiotemporal modeling

Giuliari et al. (2020) used a Transformer to forecast the trajectories of *individual* pedestrians, i.e., the model does not consider interactions between individuals. Yu et al. (2020) used *separate* temporal and spatial Transformers to forecast the trajectories of multiple, interacting pedestrians. Specifically, the temporal Transformer processes the coordinates of each pedestrian *independently* (i.e., it does not model interactions), while the spatial Transformer, which is inspired by Graph Attention Networks (Veličković et al., 2018), processes the pedestrians *independently at each time step*. Sanford et al. (2020) used a Transformer to classify on-the-ball events from sequences in soccer games; however, only the coordinates of the K-nearest players to the ball were included in the input (along with the ball's coordinates). Further, the *order* of the included players was based on their average distance from the ball for a given temporal window, which can lead to specific players changing position in the input between temporal windows. As far as we are aware, `baller2vec` is the **first** Transformer capable of processing all agents *simultaneously across time* without imposing an arbitrary order on the agents.

6. Limitations

At least two different factors may explain why including player identity as an input to `baller2vec` only leads to relatively small performance improvements. First, both player and ball trajectories are fairly generic—players tend to move into open space, defenders tend to move towards their man or the ball, point guards tend to pass to their teammates, and so on. Further, the location of a player on the court is often indicative of their position, and players playing the same position tend to have similar skills and physical attributes. As a result, we might expect `baller2vec` to be able to make reasonable guesses about a player's/ball's trajectory just given the location of the players and the ball on the court.

Second, `baller2vec` may be able to *infer* the identity of the players directly from the spatiotemporal data. Unlike `(batter|pitcher)2vec` (Alcorn, 2018), which was trained on several seasons of Major League Baseball data, `baller2vec` only had access to one half of one season's worth of NBA data for training. As a result, player identity may be entangled with season-specific factors (e.g., certain rosters or coaches) that are actually exogenous to the player's intrinsic qualities, i.e., `baller2vec` may be overfitting to the season. To provide an example, the Golden State Warriors ran a very specific kind of offense in the 2015-2016 season (breaking the previous record for most three-pointers made in the regular season by 15.4%), and many basketball fans could probably recognize them from a bird's eye view (i.e., without access to any identifying information). Given additional seasons of data, `baller2vec` would no longer be able to exploit the implicit identifying information contained in static lineups and coaching strategies, so including player identity in the input would likely be more beneficial in that case.

7. Conclusion

In this paper, we introduced `baller2vec`, a generalization of the standard Transformer that can model sequential data consisting of multiple, unordered entities at each time step. As an architecture that both is computationally efficient and has powerful representational capabilities, we believe `baller2vec` represents an exciting new direction for MASM. As discussed in Section 6, training `baller2vec` on more training data may allow the model to more accurately factor players away from season-specific patterns. With additional data, more contextual information about agents (e.g., a player's age, injury history, or minutes played in the game) and the game (e.g., the time left in the period or the score difference) could be included as input, which might allow `baller2vec` to learn an even more complete model of the game of basketball. Although we only experimented with static, fully connected graphs here, `baller2vec` can be easily applied to more complex inputs—for example, a sequence of graphs with changing nodes and edges—by simply adapting the attention tensor as appropriate. Lastly, as a generative model, `baller2vec` could be used for counterfactual simulations (e.g., assessing the impact of different rosters), or combined with a controller to discover optimal play calling strategies through reinforcement learning.

Acknowledgements

We would like to thank Sudha Lakshmi, Katherine Silliman, Jan Van Haaren, Hans-Werner Van Wyk, and Eric Winsberg for their helpful suggestions on how to improve the manuscript.

References

Alcorn, M. A. (batter|pitcher)2vec: Statistic-free talent modeling with neural player embeddings. In *MIT Sloan Sports Analytics Conference*, 2018.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

Basketball-Reference.com. 2015-16 nba player stats: Totals, February 2021. URL https://www.basketball-reference.com/leagues/NBA_2016_totals.html.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A., and Bengio, Y. A recurrent latent variable model for sequential data. In *Advances in Neural Information Processing Systems*, 2015.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Felsen, P., Lucey, P., and Ganguly, S. Where will they go? predicting fine-grained adversarial multi-agent motion using conditional variational autoencoders. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 732–747, 2018.

Giuliari, F., Hasan, I., Cristani, M., and Galasso, F. Transformer networks for trajectory forecasting. In *International Conference on Pattern Recognition*, 2020.

Graves, A. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Graves, A., Wayne, G., and Danihelka, I. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Kim, K., Grundmann, M., Shamir, A., Matthews, I., Hodgins, J., and Essa, I. Motion fields to predict play evolution in dynamic sport scenes. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 840–847. IEEE, 2010.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Le, H. M., Carr, P., Yue, Y., and Lucey, P. Data-driven ghosting using deep imitation learning. In *MIT Sloan Sports Analytics Conference*, 2017a.

Le, H. M., Yue, Y., Carr, P., and Lucey, P. Coordinated multi-agent imitation learning. In *International Conference on Machine Learning*, volume 70, pp. 1995–2003, 2017b.

Le, Q. and Mikolov, T. Distributed representations of sentences and documents. In *International conference on machine learning*, pp. 1188–1196. PMLR, 2014.

McInnes, L., Healy, J., and Melville, J. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, 2013b.

Qi, M., Qin, J., Wu, Y., and Yang, Y. Imitative non-autoregressive modeling for trajectory forecasting and imputation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12736–12745, 2020.

Sanford, R., Gorji, S., Hafemann, L. G., Pourbabaee, B., and Javan, M. Group activity detection from trajectory and video data in soccer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 898–899, 2020.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, 2014.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. In

International Conference on Learning Representations, 2018.

Weston, J., Chopra, S., and Bordes, A. Memory networks. In *International Conference on Learning Representations*, 2015.

Yeh, R. A., Schwing, A. G., Huang, J., and Murphy, K. Diverse generation for multi-agent sports games. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4610–4619, 2019.

Yu, C., Ma, X., Ren, J., Zhao, H., and Yi, S. Spatio-temporal graph transformer networks for pedestrian trajectory prediction. In *Proceedings of the European Conference on Computer Vision (ECCV)*, August 2020.

Zhan, E., Zheng, S., Yue, Y., Sha, L., and Lucey, P. Generating multi-agent trajectories using programmatic weak supervision. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rkxw-hAcFQ.

Zhan, E., Tseng, A., Yue, Y., Swaminathan, A., and Hausknecht, M. Learning calibratable policies using programmatic style-consistency. In *International Conference on Machine Learning*, pp. 11001–11011. PMLR, 2020.

Zheng, S., Yue, Y., and Hobbs, J. Generating long-term trajectories using deep hierarchical networks. *Advances in Neural Information Processing Systems*, 29:1543–1551, 2016.

Supplementary Materials

baller2vec: A Multi-Entity Transformer For Multi-Agent Spatiotemporal Modeling

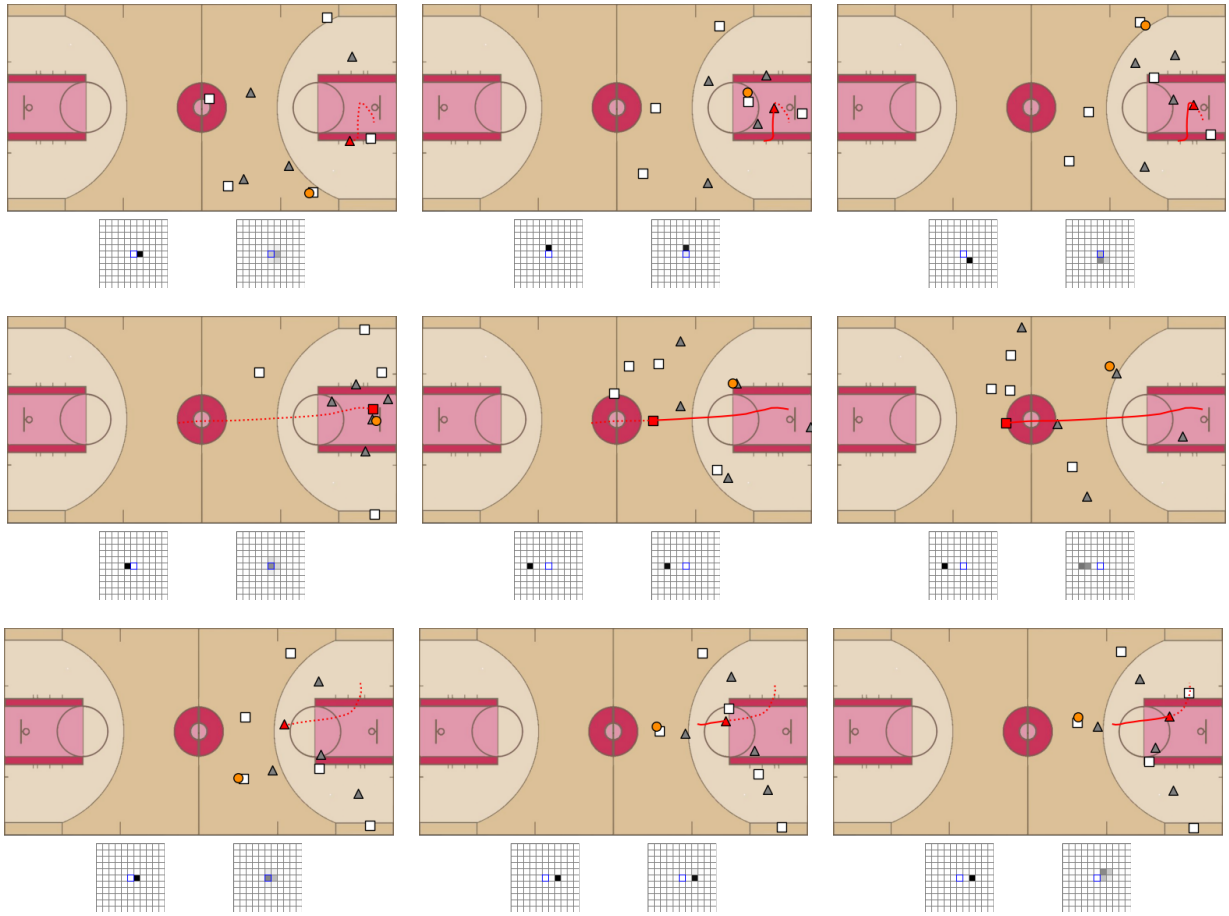


Figure S1. Additional examples of player trajectory forecasts in different contexts. Each row contains a different sequence, and the first column always contains the first frame from the sequence.

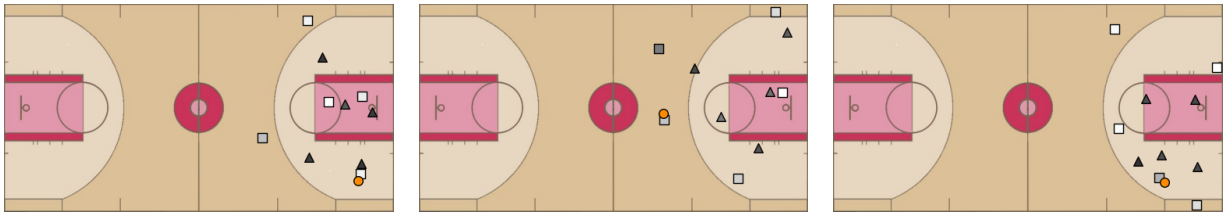


Figure S2. Additional examples of attention outputs for the head that focuses on the ball handler's teammates.

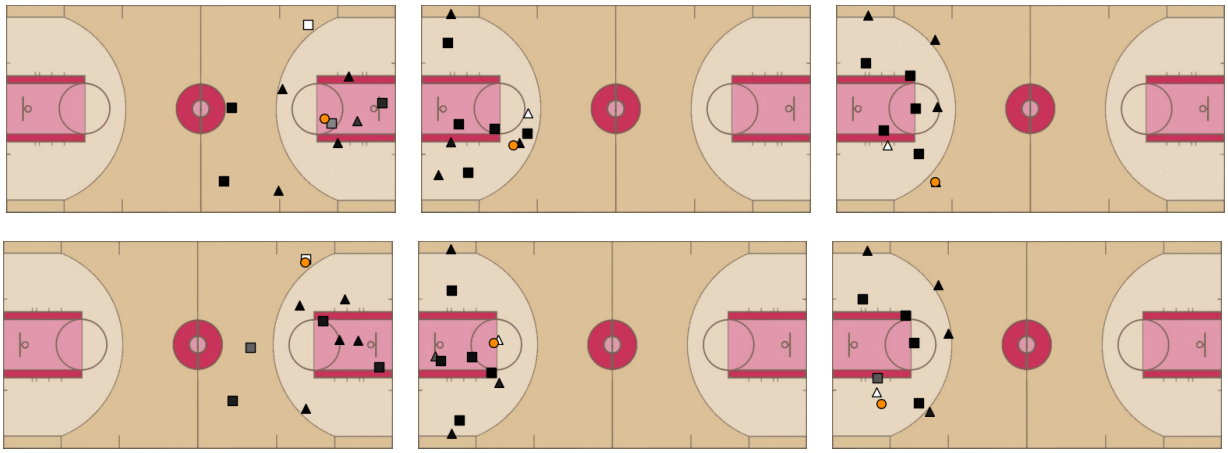


Figure S3. Additional examples of attention outputs for the head that anticipates passes. Each column contains a different sequence, and the top frame precedes the bottom frame in time.

Table S1. Image credits for Figure 7.

Image	Source	URL
Russell Westbrook	Erik Drost	https://en.wikipedia.org/wiki/Russell_Westbrook#/media/File:Rus
Pau Gasol	Keith Allison	https://en.wikipedia.org/wiki/Pau_Gasol#/media/File:Pau_Gasol_b
Kawhi Leonard	Jose Garcia	https://en.wikipedia.org/wiki/Kawhi_Leonard#/media/File:Kawhi_L
Derrick Rose	Keith Allison	https://en.wikipedia.org/wiki/Derrick_Rose#/media/File:Derrick_
Marc Gasol	Verse Photography	https://en.wikipedia.org/wiki/Marc_Gasol#/media/File:Marc_Gasol
Jimmy Butler	Joe Gorioso	https://en.wikipedia.org/wiki/Jimmy_Butler#/media/File:Jimmy_Bu

S1. `baller2vec` implementation.

```
# Adapted from: https://pytorch.org/tutorials/beginner/transformer_tutorial.html

import math
import torch

from torch import nn


class TimeEncoder(nn.Module):
    def __init__(self, seq_len, d_model, dropout):
        super().__init__()
        self.dropout = nn.Dropout(p=dropout)
        self.time_embeddings = nn.Parameter(torch.Tensor(seq_len, d_model))
        nn.init.normal_(self.time_embeddings)

    def forward(self, x, repeat):
        repeated = self.time_embeddings.repeat(repeat, 1)
        x = x + repeated
        return self.dropout(x)


class Baller2Vec(nn.Module):
    def __init__(
        self,
        n_player_ids,
        embedding_dim,
        sigmoid,
        seq_len,
        mlp_layers,
        n_players,
        n_player_labels,
        n_ball_labels,
        n_seq_labels,
        nhead,
        dim_feedforward,
        num_layers,
        dropout,
        use_cls,
        embed_before_mlp,
    ):
        super().__init__()
        self.sigmoid = sigmoid
        self.seq_len = seq_len
        self.use_cls = use_cls
        self.n_players = n_players
        self.embed_before_mlp = embed_before_mlp

        # Initialize players, ball, and CLS (if used) embeddings.
        initrange = 0.1
        self.player_embedding = nn.Embedding(n_player_ids, embedding_dim)
        self.player_embedding.weight.data.uniform_(-initrange, initrange)

        self.ball_embedding = nn.Parameter(torch.Tensor(embedding_dim))
        nn.init.uniform_(self.ball_embedding, -initrange, initrange)
        if use_cls:
            self.cls_embedding = nn.Parameter(torch.Tensor(mlp_layers[-1]))
            nn.init.uniform_(self.cls_embedding, -initrange, initrange)

        # Initialize preprocessing MLPs.
        player_mlp = nn.Sequential()
        ball_mlp = nn.Sequential()
        # Extra dimensions for (x, y) coordinates and hoop side (for players) or z
        # coordinate (for ball).
```

```
        in_feats = embedding_dim + 3 if embed_before_mlp else 3
        for (layer_idx, out_feats) in enumerate(mlp_layers):
            if (not embed_before_mlp) and (layer_idx == len(mlp_layers) - 1):
                out_feats = out_feats - embedding_dim

            player_mlp.add_module(f"layer{layer_idx}", nn.Linear(in_feats, out_feats))
            ball_mlp.add_module(f"layer{layer_idx}", nn.Linear(in_feats, out_feats))

            if layer_idx < len(mlp_layers) - 1:
                player_mlp.add_module(f"relu{layer_idx}", nn.ReLU())
                ball_mlp.add_module(f"relu{layer_idx}", nn.ReLU())

            in_feats = out_feats

        self.player_mlp = player_mlp
        self.ball_mlp = ball_mlp

        # Initialize time encoders.
        d_model = mlp_layers[-1]
        self.d_model = d_model
        self.player_time_encoder = TimeEncoder(seq_len, d_model, dropout)
        self.ball_time_encoder = TimeEncoder(seq_len, d_model, dropout)
        if use_cls:
            self.cls_time_encoder = TimeEncoder(seq_len, d_model, dropout)

        # Initialize Transformer.
        encoder_layer = nn.TransformerEncoderLayer(
            d_model, nhead, dim_feedforward, dropout
        )
        self.transformer = nn.TransformerEncoder(encoder_layer, num_layers)

        # Initialize classification layers.
        self.player_classifier = nn.Linear(d_model, n_player_labels)
        self.player_classifier.weight.data.uniform_(-initrange, initrange)
        self.player_classifier.bias.data.zero_()

        self.ball_classifier = nn.Linear(d_model, n_ball_labels)
        self.ball_classifier.weight.data.uniform_(-initrange, initrange)
        self.ball_classifier.bias.data.zero_()

        if use_cls:
            self.event_classifier = nn.Linear(d_model, n_seq_labels)
            self.event_classifier.weight.data.uniform_(-initrange, initrange)
            self.event_classifier.bias.data.zero_()

        # Initialize mask.
        self.register_buffer("mask", self.generate_square_subsequent_mask())

    def generate_square_subsequent_mask(self):
        # n players plus the ball and the CLS entity (if used).
        if self.use_cls:
            sz = (self.n_players + 2) * self.seq_len
        else:
            sz = (self.n_players + 1) * self.seq_len

        mask = torch.zeros(sz, sz)
        ball_start = self.n_players * self.seq_len
        if self.use_cls:
            cls_start = ball_start + self.seq_len

        for step in range(self.seq_len):
            start = self.n_players * step
            stop = start + self.n_players
            ball_stop = ball_start + step + 1
```

```
            # The players can look at the players.
            mask[start:stop, :stop] = 1
            # The players can look at the ball.
            mask[start:stop, ball_start:ball_stop] = 1
            # The ball can look at the players.
            mask[ball_start + step, :stop] = 1
            # The ball can look at the ball.
            mask[ball_start + step, ball_start:ball_stop] = 1
            if self.use_cls:
                cls_stop = cls_start + step + 1
                # The players can look at the CLS.
                mask[start:stop, cls_start:cls_stop] = 1
                # The ball can look at the CLS.
                mask[ball_start + step, cls_start:cls_stop] = 1
                # The CLS can look at the players.
                mask[cls_start + step, :stop] = 1
                # The CLS can look at the ball.
                mask[cls_start + step, ball_start:ball_stop] = 1
                # The CLS can look at the CLS.
                mask[cls_start + step, cls_start:cls_stop] = 1

        mask = mask.masked_fill(mask == 0, float("-inf"))
        mask = mask.masked_fill(mask == 1, float(0.0))
        return mask

    def forward(self, tensors):
        device = list(self.player_mlp.parameters())[0].device

        # Get player position/time features.
        player_embeddings = self.player_embedding(
            tensors["player_idxs"].flatten().to(device)
        )
        if self.sigmoid == "logistic":
            player_embeddings = torch.sigmoid(player_embeddings)
        elif self.sigmoid == "tanh":
            player_embeddings = torch.tanh(player_embeddings)

        player_xs = tensors["player_xs"].flatten().unsqueeze(1).to(device)
        player_ys = tensors["player_ys"].flatten().unsqueeze(1).to(device)
        player_hoop_sides = (
            tensors["player_hoop_sides"].flatten().unsqueeze(1).to(device)
        )
        if self.embed_before_mlp:
            player_pos = torch.cat(
                [
                    player_embeddings,
                    player_xs,
                    player_ys,
                    player_hoop_sides,
                ],
                dim=1,
            )
            player_pos_feats = self.player_mlp(player_pos) * math.sqrt(self.d_model)
        else:
            player_pos = torch.cat(
                [
                    player_xs,
                    player_ys,
                    player_hoop_sides,
                ],
                dim=1,
            )
            player_pos_feats = self.player_mlp(player_pos) * math.sqrt(self.d_model)
            player_pos_feats = torch.cat([player_embeddings, player_pos_feats], dim=1)
```

```
player_pos_time_feats = self.player_time_encoder(
    player_pos_feats, self.n_players
)

# Get ball position/time features.
ball_embeddings = self.ball_embedding.repeat(self.seq_len, 1)
ball_xs = tensors["ball_xs"].unsqueeze(1).to(device)
ball_ys = tensors["ball_ys"].unsqueeze(1).to(device)
ball_zs = tensors["ball_zs"].unsqueeze(1).to(device)
if self.embed_before_mlp:
    ball_pos = torch.cat(
        [
            ball_embeddings,
            ball_xs,
            ball_ys,
            ball_zs,
        ],
        dim=1,
    )
    ball_pos_feats = self.ball_mlp(ball_pos) * math.sqrt(self.d_model)
else:
    ball_pos = torch.cat(
        [
            ball_xs,
            ball_ys,
            ball_zs,
        ],
        dim=1,
    )
    ball_pos_feats = self.player_mlp(ball_pos) * math.sqrt(self.d_model)
    ball_pos_feats = torch.cat([ball_embeddings, ball_pos_feats], dim=1)

ball_pos_time_feats = self.ball_time_encoder(ball_pos_feats, 1)

# Combine players and ball features.
combined = torch.cat([player_pos_time_feats, ball_pos_time_feats], dim=0)

if self.use_cls:
    # Get CLS time features.
    cls_feats = self.cls_embedding.repeat(self.seq_len, 1)
    cls_time_feats = self.cls_time_encoder(cls_feats, 1)

    # Combine with CLS features.
    combined = torch.cat([combined, cls_time_feats], dim=0)

output = self.transformer(combined.unsqueeze(1), self.mask)
preds = {
    "player": self.player_classifier(output).squeeze(1),
    "ball": self.ball_classifier(output).squeeze(1),
}
if self.use_cls:
    preds["seq_label"] = self.event_classifier(output).squeeze(1)

return preds
```