StratFormer : How language deep-learning models can help us to better understand NFL games ?

Samuel Chaineau

April 3, 2023

Contents

1 Abstract

This report presents "StratFormer", an attention-based model applied on NFL data. The model extracts information from trajectories performed by players on the field durng a play. To train the model, we ask it to guess the team and position of the player. The model also has to complete the trajectory, and decide whether two trajectories were drawn from the same play.

The model is trained on the NFL Big Data Bowl data set of 2021. It yields a 80% accuracy for the team classification task, 40% for the position classification task (with the correct position being in the top 3 predictions 67% of time). The correct path is completed correctly 52% of time (with the correct completion being in the top 3 predictions 84% of time). On the task of detecting if two trajectories were made during the same play, the model achieves an accuracy of 54%.

We later assess those embeddings' quality on a concrete application with the objective of classifying passing plays results. While using only the trajectories and no other information, we achieve a 57% accuracy on a three-classes classification problem. Benchmarks found on Kaggle are around 40 to 55% but made on a different data set. This performance should be back-tested.

We highlight some limitations of the findings as the work has been done on a relatively small, sparse and unbalanced data set. Further work may affect first findings. However we clearly prove the relevancy of attention-based models for sports analytic, especially in the NFL.

2 Introduction

Deep-learning has been highly impacted by the publication of the famous paper "Attention is all you need". Firstly dedicated to NLP applications, the underlying transformer's architecture quickly found applications in many fields such as Computer Vision, Video tracking and even Time-Series. An advantage of these models is their ability to learn in a self-supervised manner from complex and unstructured data. The core exercise is how to design a task enabling the model to learn in a self-supervised fashion, requiring deep understanding of your data and what you expect from it. For instance, the famous BERT model learns to predict a word masked in a sentence and whether if two sentences follow each others or not. This simple exercise enables the model to produce high quality representations, called "embeddings", for the sequence's tokens. Those representations become input for other tasks such as sentiment classification or topic modeling in BERT's case.

Sport always stood out as a great playground for data science and ML skills according to me. The application is very concrete and manipulating the data is quite intuitive as long as you have a basic knowledge of the sport and its rules. One particular sport that is very well documented and monitored is Football (for my US fellows) or American Football (for everyone else like me). The sport presents two major advantages making it a perfect match for ML, and my use case:

- It is completely **sequential**: meaning that each play are divided in time. You can process and predict play-by-play. For instance rugby is less sequential as you can have an uninterrupted sequence of plays with the ball changing sides etc.
- It is highly **strategic**: each team in each play follow a very well tailored strategy with indications for each player. Players are expected to behave according to the strategy (at least at the beginning) meaning it's less chaotic than other sports.

However, football is a very complex sport with playbooks counting hundreds of strategies, several dozens of roles as a player and a large number of players per team. Data is key to synthesize all of the information and help to optimize staff/coach decisions.

This report aims at demonstrating that transformers' architectures can be derived on football data with a significant contribution. The derived architecture is called "*StratFormer*" in reference to the movie "MoneyBall" translated in "The Strategist" in French. Its pre-training tasks are NFL specific. However the core concept is very similar to BERT with an objective of producing high quality embeddings for team, players, plays and more, which can be manipulated later on for other tasks. The model finds its information in real moves made by players, their paths on the field and how they relate. The report shows one use case: pass prediction. No real benchmarks are easily accessible on Kaggle but from some research "StratFormer" comepte with other models while only using trajectories as predictors.

3 Data

3.1 Description

The data used for this report comes from the NFL Big Data Bowl 2021 challenge on Kaggle. The NFL provided several data sets with a goal of helping to evaluate defensive performance on passing plays. The data provided covers 253 games. Those games are broken down into 192,329 plays. Each play follows up to 22 players at the same time (up to 11 per team). Some plays might have less or more players as the offensive and defensive lines are not monitored in the data set. Each player is tracked on a 0.1 second basis with his position on the field being provided as y and x coordinates. Each player is affected to a specific position (16 positions in the data set, more in practice). A position is the specific role attributed to a player (Wide-receiver, Quarter-Back, Line-Backer...) with an expected behavior according to the play.

The first step performed on the data is reducing the number of frames kept. We keep the first 9 seconds of the play. We assume that after 9 seconds trajectories become more and more chaotic, hence noisy. Also, we keep the frames at 0.4 seconds of interval as minor changes occur at a 0.1 scale.

Each play can be plotted on a scatter plot where each point represents the position of a player at a moment, its size changing based on the moment as shown in figure 1.

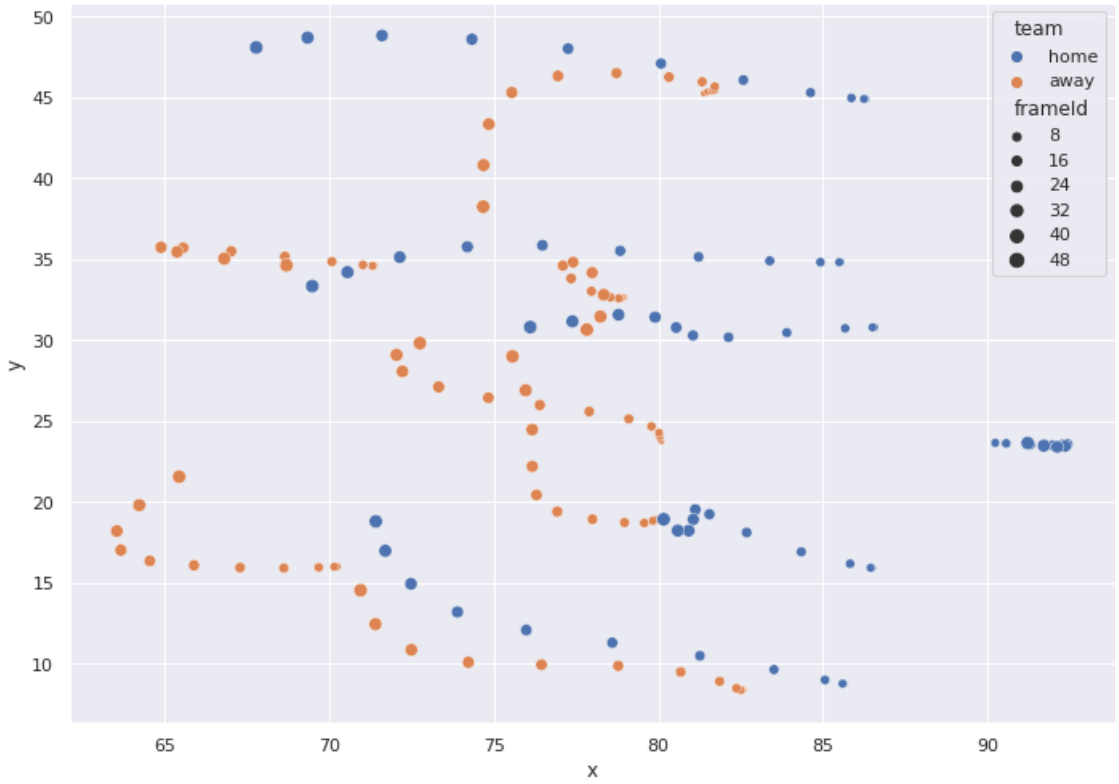


Figure 1: Example of a play plotted as scatter plot. Each point is separated by 0.4 seconds and size of dots grow over time.

Hence, for each play we have an complete tracking of each players' coordinates on the field. Those coordinates, now called trajectories, are expected to be a relevant signal to predict the associated team, position and whether two trajectories are drawn from a similar play.

3.2 Standardization

In the NFL data set, each player is tracked identically by the NFL system on the field. However, they behave differently if you put yourself into their shoes. An offensive player is expected to go in the direction in front of him in order to gain yards, while the defensive player is expected to go backward to cover passes, tackle players and block runs. We want to build new trajectories that reflects how each player moved at their own scale, all starting at the same point: the coordinates [0,0], then moving according to his trajectory.

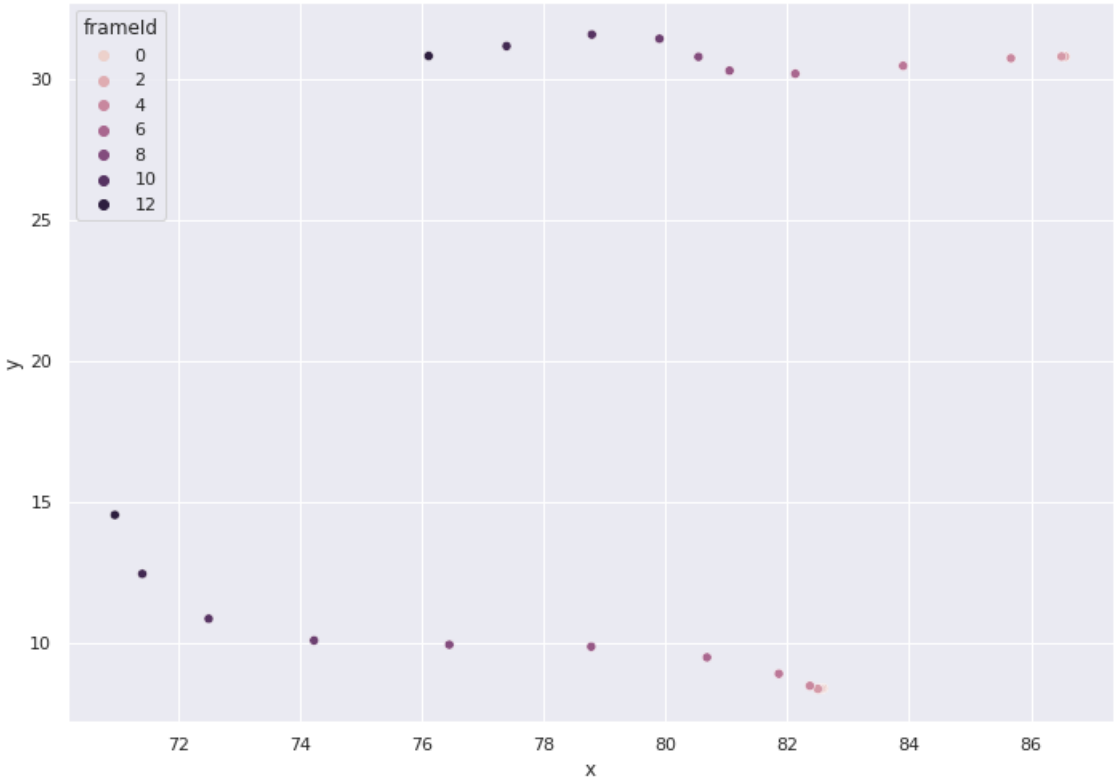


Figure 2: Two trajectories not being standardized. Offense and defense go in the same direction.

Trajectories are standardized using a two step process. Firstly, I subtracted the starting coordinates (the first one in the array) to all the coordinates in the trajectory. Thus, my array always start by the [0, 0] coordinates. Then, I compute the first difference between each coordinates in the trajectories, these differences are the vectors representing the distance ran between time t and t+1. If my player is an offensive one, I keep it unchanged. If he is defensive, I multiply by -1 to reverse the movement and align it with the offensive team. The practical idea behind this transformation is to clearly explain that a player went backward while the other one went frontward. Finally, I used a cumulative sum to rebuild my full trajectory as expected.

The new trajectories are comparable and aware of who's going frontward and backward as shown by the difference between figure 2 and figure 3.

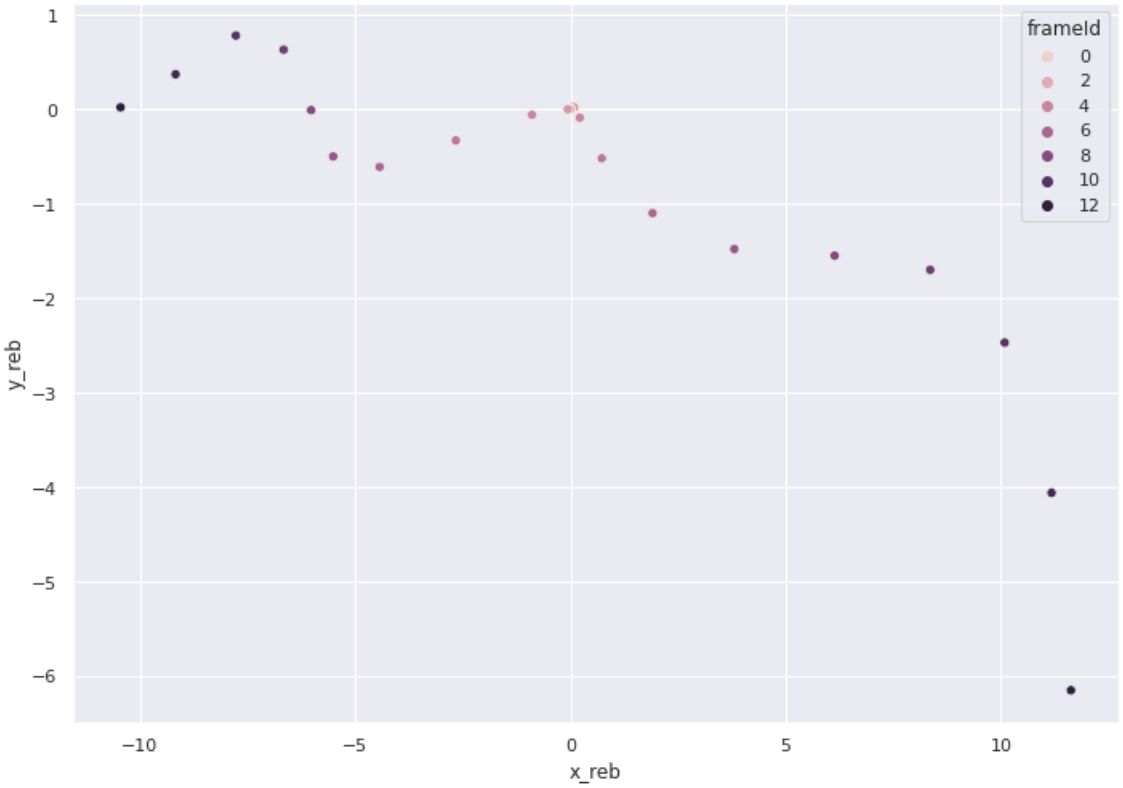


Figure 3: Two trajectories standardized. Both of them start at the same point and move in opposite directions.

On the +260,000 standardized trajectories, we can plot the distributions of each single coordinates on a 2-d space to see how space has been explored by player. Coordinates are strongly represented in a -40:40 range both for X and Y.

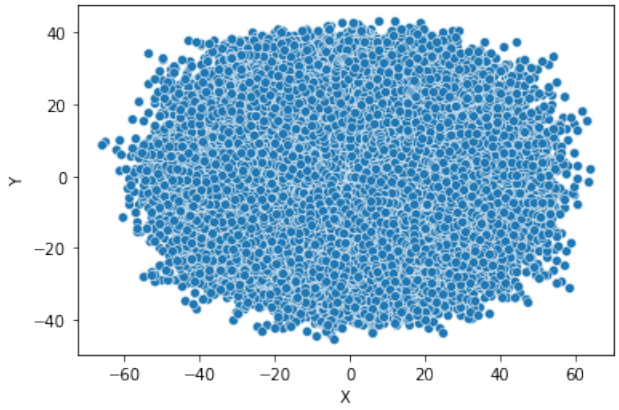


Figure 4: Coordinates distributions after standardization.

3.3 Mapping the trajectories

At this stage, our trajectories are expressed in X and Y coordinates. If we project ourselves in a realistic application, we are more interested by a relative zone on the field where the player can be, rather than the exact coordinates. Also, projecting ourselves in a modeling exercise, it's tough to see and adapted loss as neither RMSE or Cosine Similarity answer well to the problem of trajectory completion, while Cross Entropy might be a good fit. This motivated me to shift from a continuous variable to a discrete one. Each 1 yard squared on the field would be a zone. The ones with an X and Y lower/higher than -40/40 are considered as a single zone as very few player pass by it.

Each coordinates are mapped into a single ID representing a defined zone on the field. Originally we have 6404 possibilities ((80*80)+4 outlier zones). When keeping only the ones with actual occurrences we end up with 6062 valid IDs for each zone. From an NLP perspective, let's assume that these IDs are the equivalent to words or tokens for people familiar with BERT and other language models.

If we plot the count of occurrences per zone, it confirms a circular shape with more variance on the X axis than the Y, as the field goal are on the extremes of this axis. Of course the original point is the more represented as every trajectories start by it.

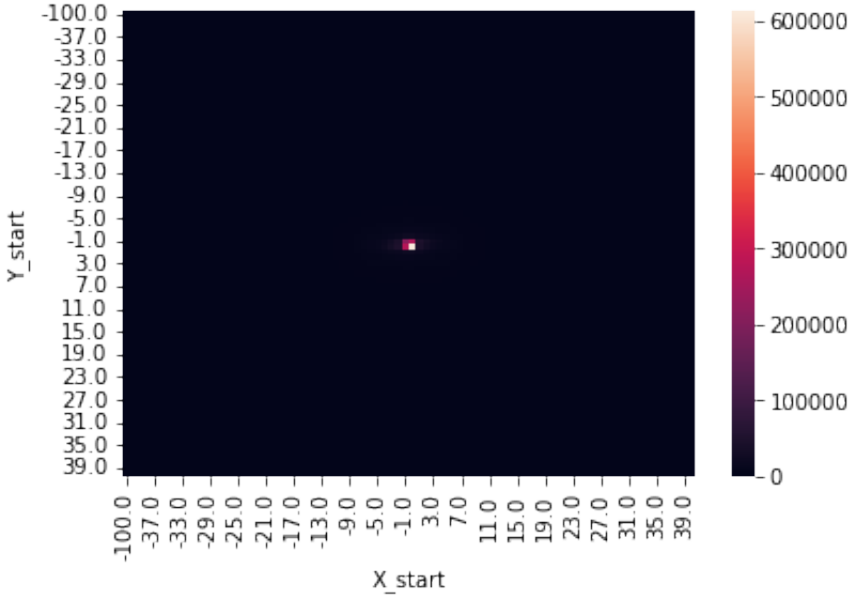


Figure 5: Zones with the most occurrences. No limit to the count.

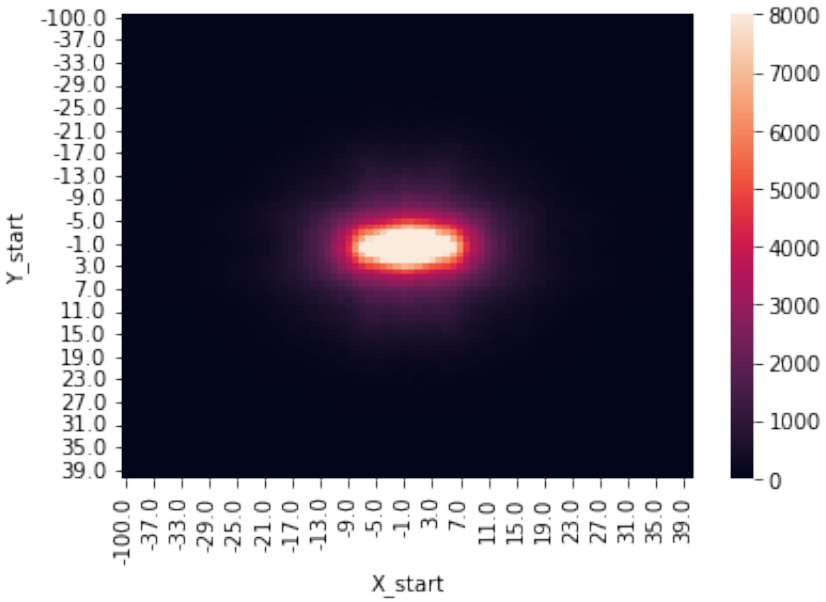


Figure 6: Zones with the most occurrences. Limit the count to 8000.

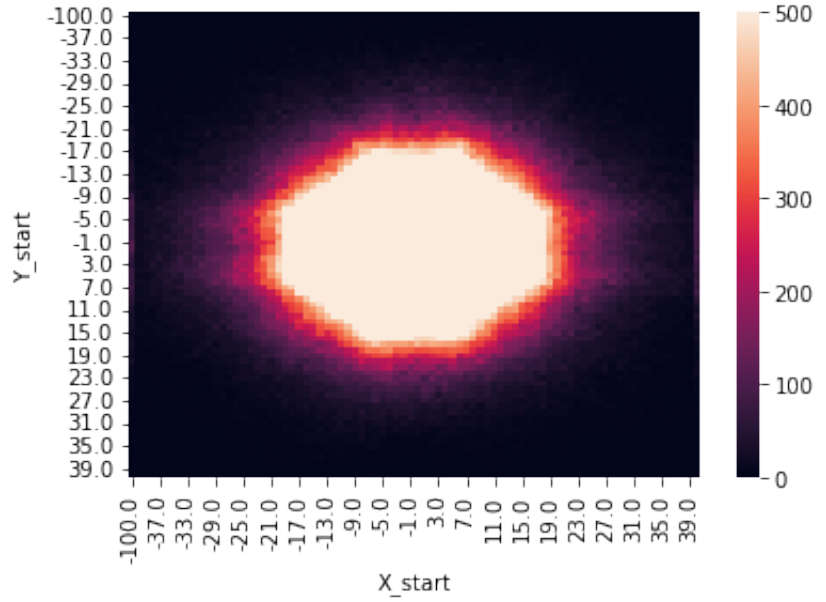


Figure 7: Zones with the most occurrences. Limit the count to 500.

4 StratFormer

4.1 Architecture

4.1.1 Input

"*StratFormer*" is a Transformers-based model taking a two identically shape and processed sequences as input. A single input sequence is defined by two components:

- The tokens: a list of special tokens "[PLAY]", "[TEAM]" and "[POSITION]" followed by a list of zone tokens representing the zone where the player is at time t during the action
- The temporal ids: when appended for special tokens, we use a special temporal id common for every special tokens. Otherwise, we use temporal tokens to represent the time (starting at 1) for each time step. Those time indicators help the model to better grasp trajectories' dynamics and speed.

From this sequence, we randomly mask one zone token (always between the fourth and last one, as earlier zones tend to be easier to predict).

The input sequence is processed using an Embedding Layer which maps tokens and temporal ids to embeddings of shape 256. The final embedding is the sum of the token embedding and the temporal one, producing temporally aware token embeddings.

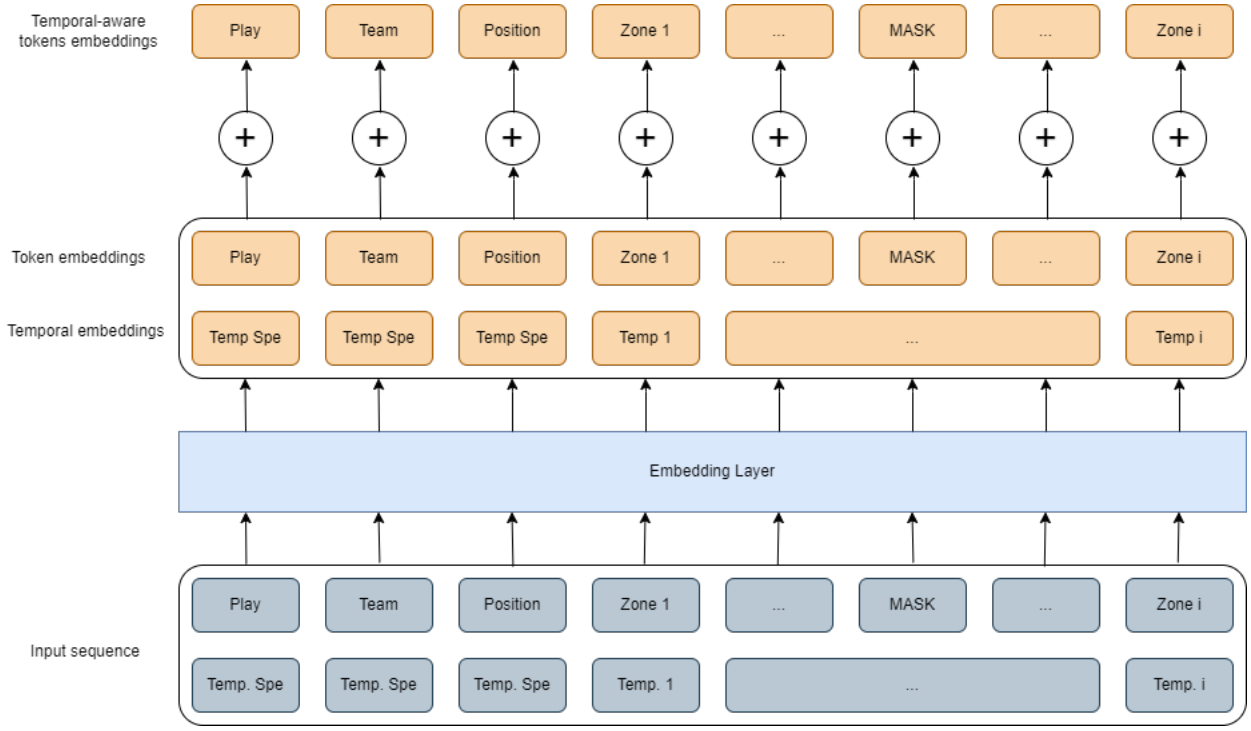


Figure 8: Input example of StratFormer

4.1.2 Model

The core part of the model follows a structure very similar to a BERT, with 3 attention blocks stacked and skip connections enabling gradients to flow down to first layers.

The output of this core part is a sequence of embeddings of shape 256 fed into a linear dense layer (without bias) mapping the output to a 128-dimensional space. Hence, at the end of this encoding process, we have for each token an embedding of dimension 128.

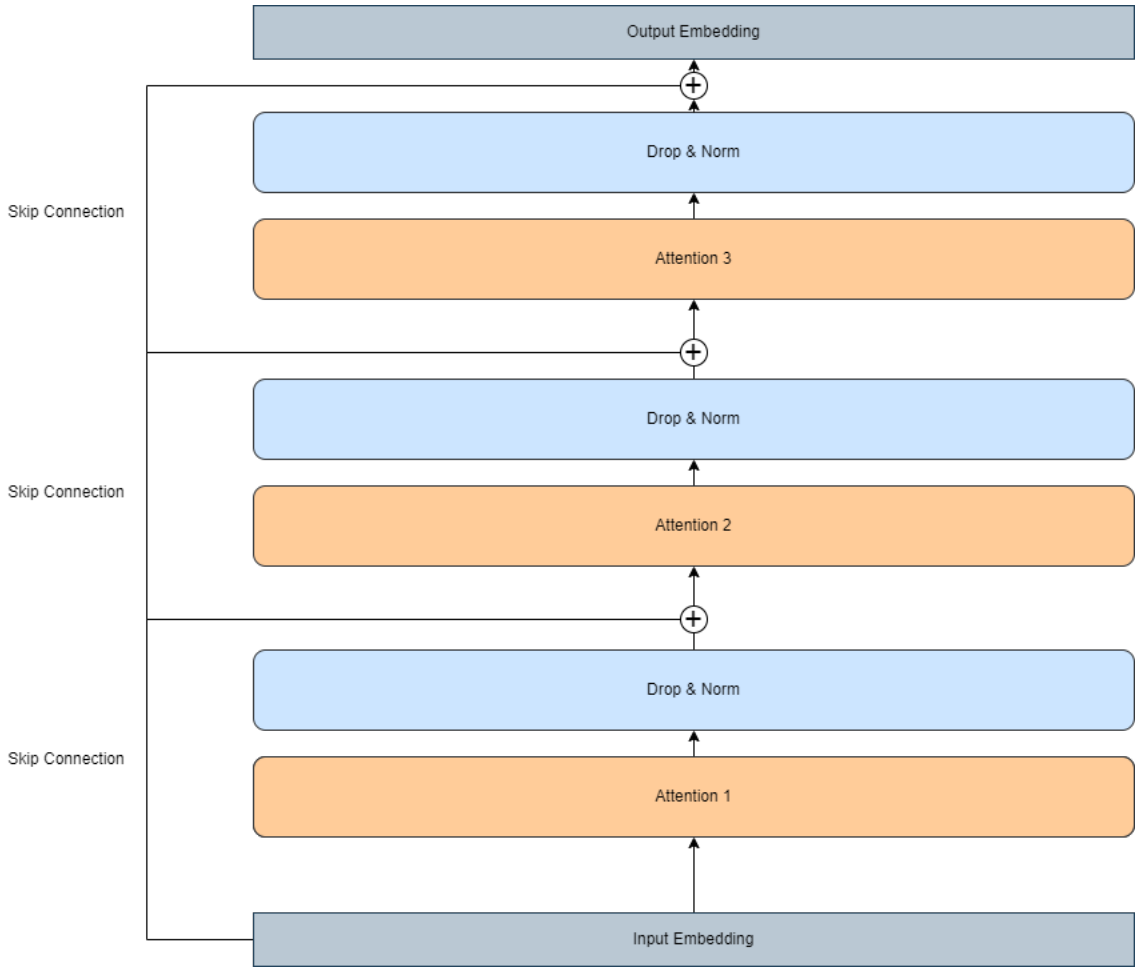


Figure 9: Encoder structure

4.1.3 Output

The output sequence is used for different tasks. Four heads are jointly trained on top of the output embeddings. During pre-training, we use the two sequences as input, processed exactly the same way as explained earlier.

The output for each sequence is made of 4 elements:

- "[PLAY]" embedding: Embedding capturing the notion of "play similarity". Its training is defined in the following section.
- "[TEAM]" embedding: Embedding capturing the notion of defense and offense.
- "[POSITION]" embedding: Embedding capturing the notion of role as a player in a team
- "[MASK]" embedding: Embedding capturing the notion of trajectory as it represents the masked zone

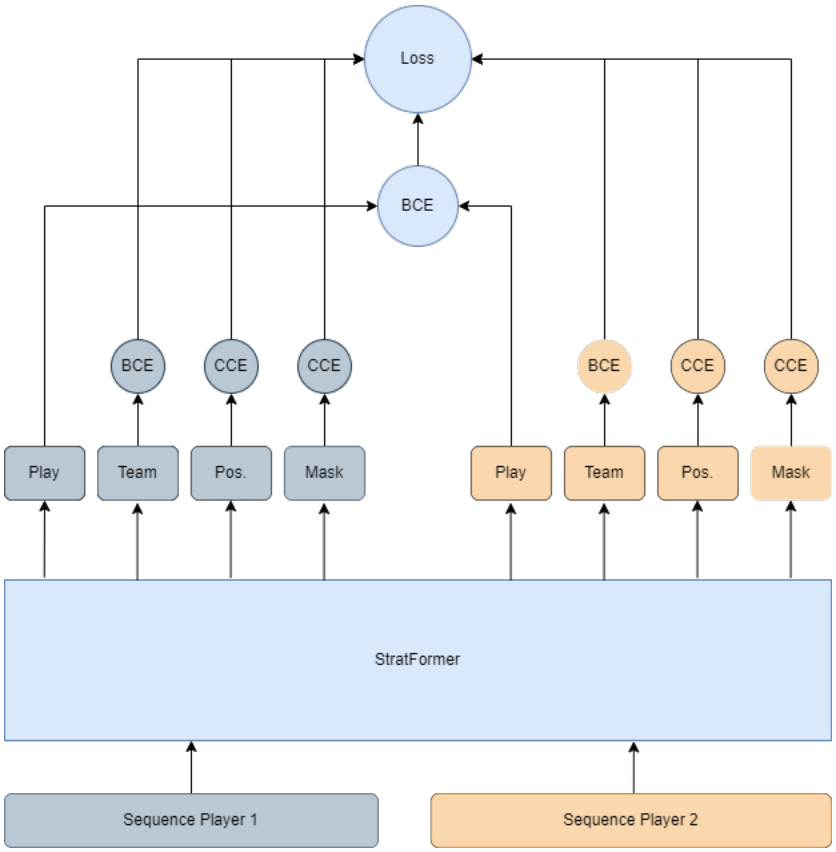


Figure 10: Output view of "*StratFormer*"

It is important to keep in mind that those embeddings are produced using only information found in standardized trajectories sliced in 0.4 seconds frames. Their representations are not bounded by specific roles, team or plays but pay attention to the paths of each player on the field.

Each embedding is used for a classification pre-training task detailed in the following section.

4.2 Pretraining tasks

4.2.1 Play similarity

Inspired by the paper *"Sentence Embeddings using Siamese BERT-networks"*, I used a similar approach to define the notion of "play similarity". The pairs of sequences used to build "StratFormer" are half of the time "positive" and half of the time "negative".

If they are positive, it means that both of the fed trajectories come from the same play, regardless their respective teams (it can be offensive and defensive).

If they are negative, it means that both trajectories have the same length but occurred during different plays in a different game.

The main idea behind it is to force the model to understand what trajectories are correlated. This similarity can be later use to specifically understand offensive or defensive strategies' effectiveness.

To predict such similarity, each embedding "[PLAY]" from both sequences are used. Following the formula used in sentence-BERT, we concatenate both embeddings and their absolute differences into a single vector. A dense layer with sigmoid activation predicts, based on this vector, whether both plays are similar or not. The "Play similarity loss" for a single example is the binary cross entropy loss between the prediction and the label.

$$Prediction = sigmoid([PLAY]_1, [PLAY]_2, |[PLAY]_1 - [PLAY]_2|)$$

$$PlaySimilarityLoss = BCE(Prediction, Label)$$

4.2.2 Team classification

Each sequence has a "[TEAM]" special token. This token is fed to a dense layer of dimension 128 with a "relu" activation then to a dense layer with a sigmoid activation. If the team is offense the label is 1, otherwise it's 0. Based on the predictions, we compute a binary cross entropy for each prediction.

The "Team loss" for a single example is the average of both losses from sequences 1 and 2.

$$Prediction_i = sigmoid([TEAM]_i)$$
$$Loss_i = BCE(Prediction, Label)$$
$$TeamLoss = (Loss_1 + Loss_2)/2$$

4.2.3 Position classification

Each sequence has a "[POSITION]" special token. This token is fed to a dense layer of dimension 128 with a "relu" activation then to a dense layer with a softmax activation. There are 16 possible positions in the data set (greater number in practice).

Some positions tend to be easy to predict (because we have many examples or because their trajectories are very discriminant), while other are particularly tough (few examples or trajectories very similar to other positions).

Hence, I used a categorical focal loss to mitigate the impact of easy examples on my loss. If you're not familiar with focal losses, I strongly recommend this paper and repo to get yourself used to it: FocalLoss.

The "Position loss" for a single example is the average of each sequence loss.

$$Prediction_i = softmax([POSITION]_i)$$

$$Loss_i = FocalCCE(Prediction, Label, gamma = 2)$$

$$PositionLoss = (Loss_1 + Loss_2)/2$$

4.2.4 Trajectory completion

Each sequence has a "[MASK]" special token, masking one time step in the trajectory. This token is fed to a dense layer of dimension 128 with a "relu" activation then to a dense layer with a softmax activation. There are 6062 possible zones.

Similarly to what I did for positions, I used a categorical focal loss to compute the predicted zone and the true one. This choice is motivated by the fact that some zones are less likely to be explored than others, leading to easy and hard examples.

The "Trajectory loss" for a single example is the average of each sequence loss.

$$Prediction_i = softmax([MASK]_i)$$

$$Loss_i = FocalCCE(Prediction, Label, gamma = 2)$$

$$MaskLoss = (Loss_1 + Loss_2)/2$$

4.2.5 Total loss

The model's loss is the weighted sum of each task loss. I kept as values the following weights:

- Play similarity : 3
- Team : 1.2
- Positions : 0.8
- Mask : 0.4

$$TotalLoss = (3 \times PlaySimilarityLoss) + (1.2 \times TeamLoss) + (0.8 \times PositionLoss) + (0.4 \times MaskLoss)$$

5 Experiments results

5.1 Pretraining

Overall, the data set is divided in a train and test set with 177 games in the train and 76 in the test.

The model has been trained with an Adam optimizer over 10 epochs with batches of size 32. The learning rate is scheduled to be 3e-3 for the three first epochs, then 1e-3 for the three following, then 5e-4 for the two others following and finally 3e-4 for the last ones. Those levels are purely chosen from empirical tests as I don't have too much money to put in Colab Pro.

Originally, the data set is strongly unbalanced with positions being present more than 1/5 of the time. I capped the positions occurrences to reduce the bias.

The train and test set specifications are displayed here under:

Team	Positions	Train count	Train percentage	Test count	Test percentage
Offense	QB	23,393	9.17	10029	9.1
Defense	SS	17,452	6.84	7627	6.92
Offense	WR	39,374	15.44	16972	15.4
Defense	FS	19,743	7.74	8182	7.42
Offense	RB	23,376	9.17	9854	8.94
Defense	MLB	9,990	3.92	3611	3.27
Defense	CB	35,473	13.91	15338	13.92
Offense	TE	25,009	9.8	10736	9.75
Defense	LB	11,709	4.59	4880	4.42
Offense	FB	1,200	0.47	533	0.48
Defense	OLB	20,657	8.1	9231	8.38
Defense	ILB	15,749	6.18	7354	6.68
Defense	DB	7,873	3.08	3987	3.62
Offense	HB	1,262	0.49	567	0.51
Defense	S	2,508	0.98	1187	1.08
Defense	DE	224	0.09	80	0.07
Offense	*Total*	141,378	55.44	61,477	55.8
Defense	*Total*	113,614	44.56	48,691	44.2
Total	**Total**	254,992	100	110,168	100

5.1.1 Overall

The loss is quickly reduced on the first 4-5 epochs. After, we can see that the model over fits. When checking some metrics, we will see that even during those ”late” epochs the model keeps learning from the data.

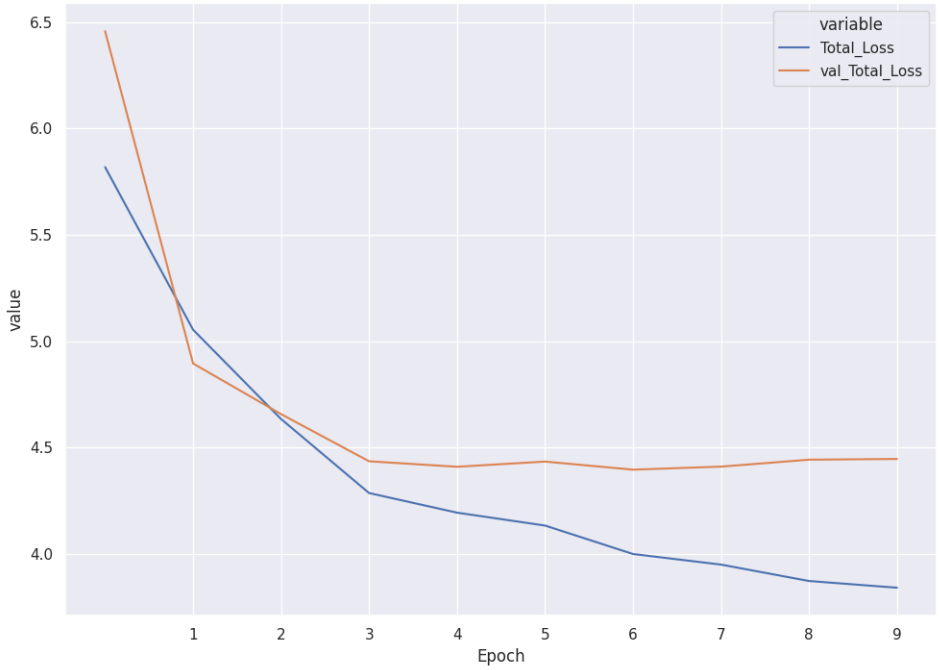


Figure 11: Total loss during training

5.1.2 Play similarity

While having a relatively small data set, my experiments show that it's possible to learn a representation of what's a common play. However, the task is particularly hard as having just two trajectories is tough to say whether the two occurred during the same play or not.

This task can be improved. One way is by feeding the 11 players each time and trying to guess if effectively they played together or not. I will note this as a quick improvement. The limitation that I faced is the computing resource required to perform such task. In addition, I don't have all the 11 players for each play, making my input sparse and flawed.

The test accuracy is 54% meaning that the model has slightly over perform randomness as I have 50% of similar and not similar plays.

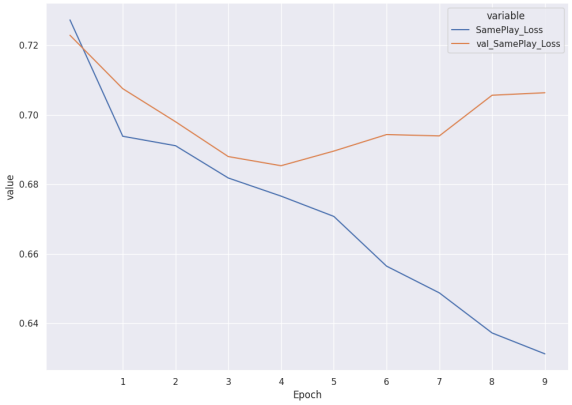


Figure 12: SamePlay Loss

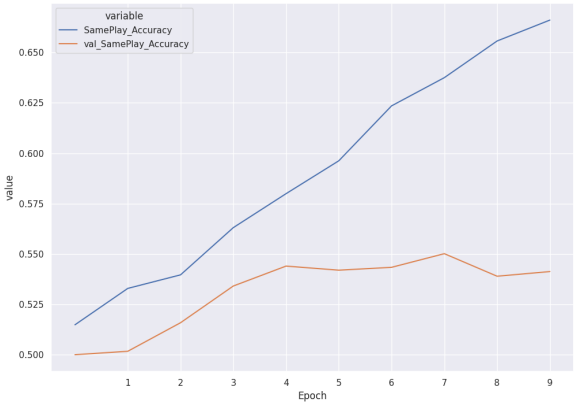
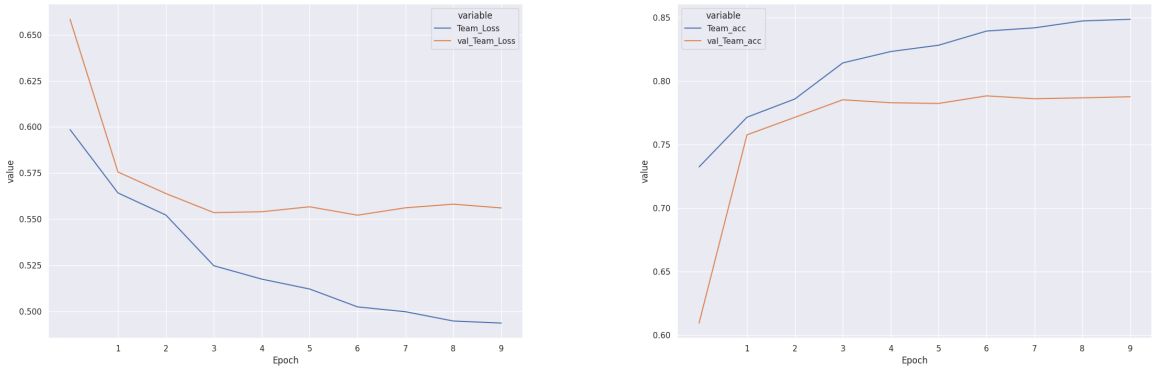


Figure 13: SamePlay Acc

5.1.3 Team classification

The model tends to grasp pretty quickly which trajectories are offensive or defensive. The test accuracy is 80% with a good ROC curve.

Attention matrices showed at the end tend to indicate that to assess whether a player attacks or defend you have to pay attention at his starting and ending zone, which is fair as you have to understand if he goes frontward or backward. The 20% that are misclassified can come player having strange trajectories or very short plays.



(a) Team Loss

(b) Team Accuracy

Figure 14: Loss and Accuracy of Team task

The ROC curve for the Team prediction exercise is very good.

Fine-tuning use cases with this embedding could be to assess the degree of offensiveness, risk-taking or play style for each team.

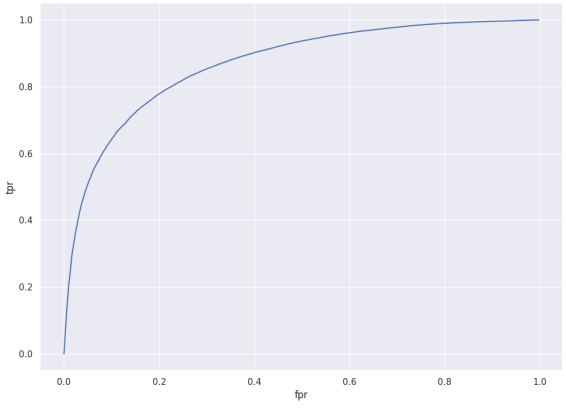
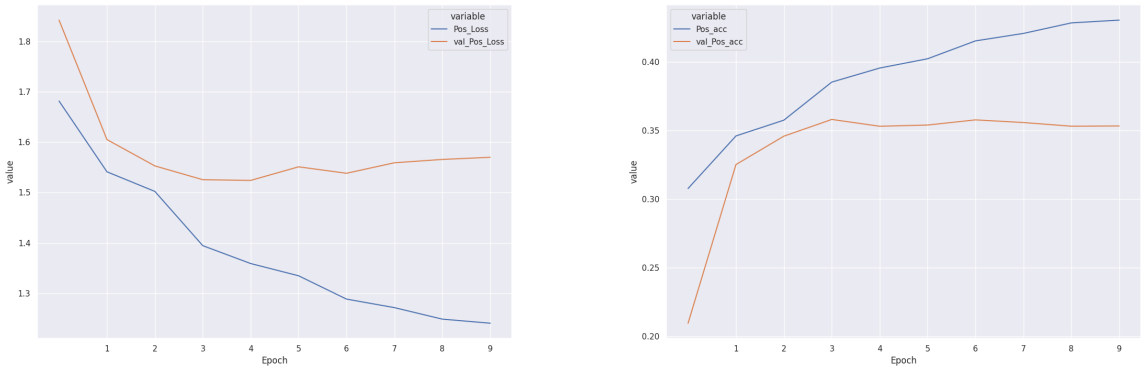


Figure 15: Team ROC

5.1.4 Position classification

16 positions are possible. Overall, the model captures well the notion of position with some pitfalls coming from the unbalanced nature of the data set. The loss tends to overfit on the training set. The test accuracy is 40% which is quite good. Moreover, if we look at the Top 3 accuracy, the model continues to learn what are the most possible roles even after the 5th epoch. The top k accuracy metrics are: Top 2 : 56%, Top 3: 67% andTop 5: 80%.



(a) Position Loss

(b) Position Accuracy

Figure 16: Loss and Accuracy of Position task

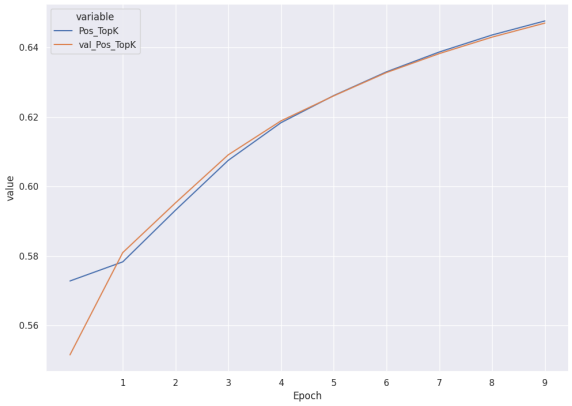


Figure 17: Position Top 3 Accuracy

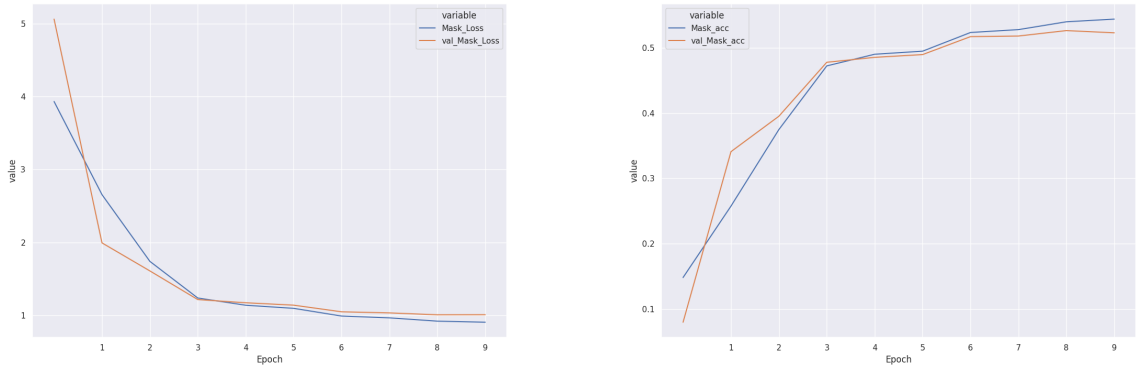
A good insight from this approach is that using the [POSITION] embedding helps to have a broader understanding of the role as we are in a continuous space. We can look at how player are different within the same role and what roles are close to each other from a trajectory perspective.

5.1.5 Trajectory completion

6062 zones are possible for the model. It is the hardest task and it looks very promising as it is by far the best loss between the three. It does not over fit that much and we can see that the loss continues to diminish on late epochs.

The test accuracy is 52%. To get a benchmark, which is quite naive, predicting the zone just before the [MASK] yields an accuracy of 16% and just after 12%. We improve the accuracy by more than 35%.

As for the positions, we can check the top k accuracy. The curve for the top 3 accuracy continues to grow even during late epochs meaning that the understanding of trajectories continue to be useful. The top k accuracy metrics are: Top 2 : 75%, Top 3: 84% and Top 5: 92%.



(a) Mask Loss

(b) Mask Accuracy

Figure 18: Loss and Accuracy of Mask task

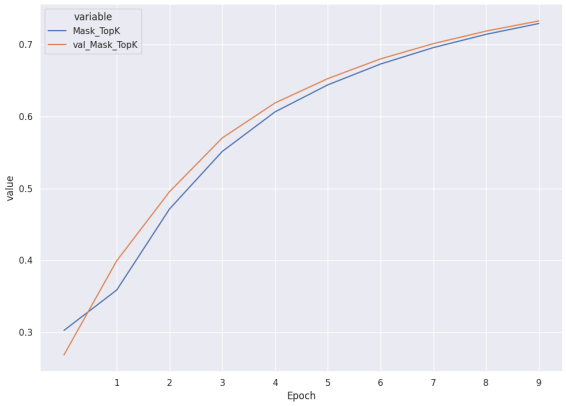


Figure 19: Mask Top 3 Accuracy

I don't see use cases for this embedding. It can be helpful when designing new strategies maybe, as we could only specify the team, position and starting zones of the sequence then let the model guess what is the most probable path.

6 Fine tuning on pass prediction

I tried to evaluate the quality of the produced embeddings by StratFormer. One way to assess it is to see whether those representations bring signals to predict a pass result. The key idea is to say that if the offensive and defensive teams behave as expected based on their respective playbooks. Let's see whether it brings some signal.

Many notebooks relying on other factors such as scores, trends, auto-regressive factors or team composition are used. No leaderboards exist on Kaggle but based on some research the accuracy is expected to be between 40 to 55%. Most of the time the model used is a boosted tree.

I have a very sparse and unbalanced data set because for each play I don't have every player on the field, meaning I have partial information about what actually happened. My architecture is to encode each player individually for each team of the play. If I have 8 players out of 11, I append 3 null embeddings.

I use the following process. For each player, I encode the trajectory and average every embeddings from the sequence. I call this embedding "*TRAJ*". I concatenate each TRAJ embeddings per team to get a representation. I apply a dense layer to the defense and offense. Then I subtract the defense embedding to the offense one. Eventually, I apply a softmax layer. The statement is to say that the result is equal to the offense capacity subtracted by the defense one.

$$TRAJ = average([PLAY], [TEAM], [POSITION], [ZONE_1], ..., [ZONE_i])$$

$$OffenseRepresentation = concatenate([PLAY]_1, ..., [PLAY]_1 1)$$

$$DefenseRepresentation = concatenate([PLAY]_1, ..., [PLAY]_1 1)$$

$$Offense = softmax(OffenseRepresentation)$$

$$Defense = softmax(OffenseRepresentation)$$

$$Prediction = softmax(OffensePred - DefensePred)$$

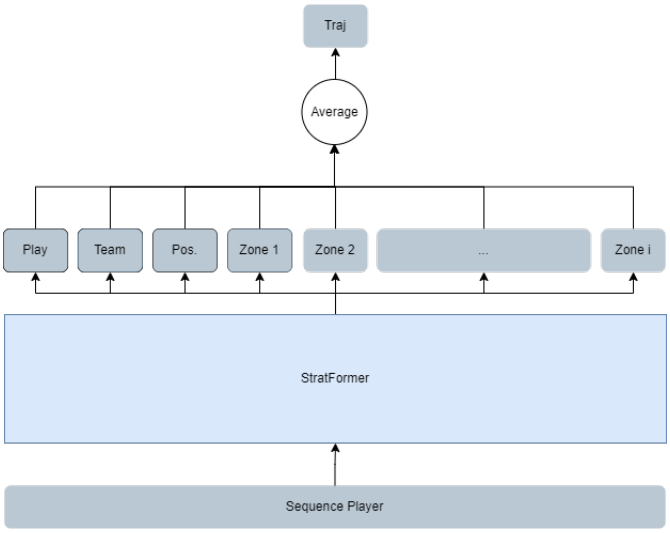


Figure 20: TRAJ embedding process

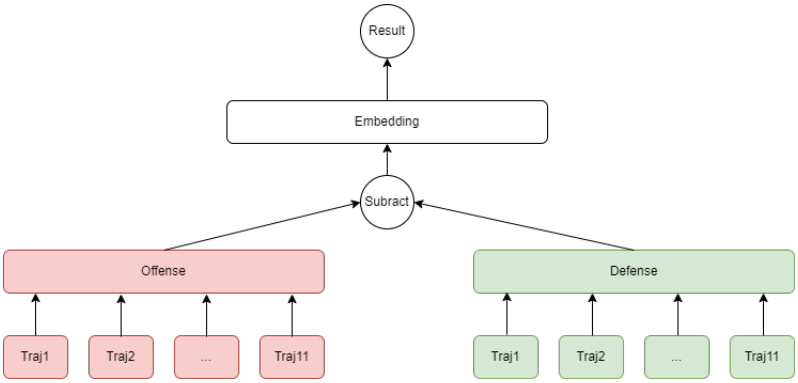


Figure 21: Model's architecture to predict pass result

According to the NFL data set, there are 3 possibilities for a passing play:

- Complete pass
- Intercepted pass
- Sack

My train and test set summaries are displayed here under:

Pass Result	Train count	Test count
Complete	3,000	2000
Intercepted	3,000	1,949
Sack	913	398
Total	6,913	4,347

With my frozen embeddings, I achieve a 57% accuracy on the test games without using any external data that might bring signals (scores, down, remaining yards...) which is extremely encouraging as I work with incomplete data and pre-trained my StratFormer on relatively few observations.

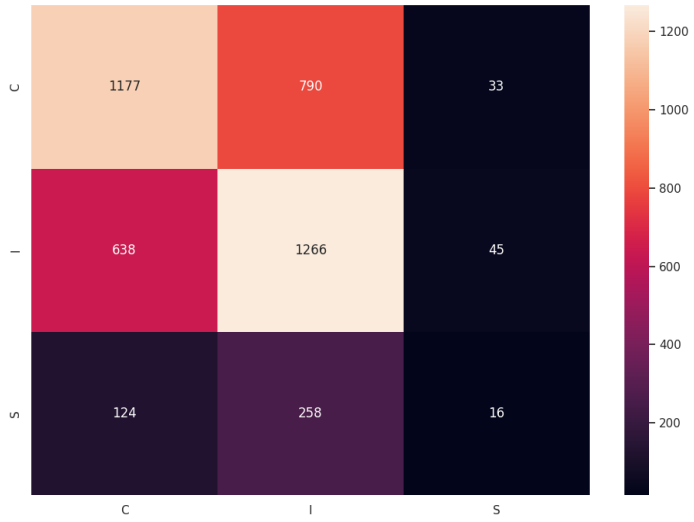


Figure 22: Confusion Matrix Play Prediction

7 Room for improvements

Many improvements can be made on existing proposed model. I made a list of my top 4 improvements if you want to assess the model in a proper manner.

- Use a more complete data set with running plays and every players' trajectories to build a complete model
- Build a play similarity representation by encoding the 11 trajectories at the same time and assessing if they all occurred during the same play or not.
- Evaluate the embeddings on regression/classification task that is more comprehensive than only passing plays' results as it is a strongly unbalanced task requiring more external data
- Try to check whether the model understands the strategies in the way the playbooks of teams are made

8 Conclusion

In this report, I present StratFormer, a transformers-based model that extracts information from NFL players' trajectories in order to guess their team, positions, paths and whether two trajectories were done during the same play. The representations extracted from this model have a wide variety of applications.

The model has a similar architecture to a BERT model and has been trained on relatively few data, meaning that my findings need a back test on a bigger data set to have a clear conclusion. However, we can assume that the model answers well its different pre-training tasks.

It proves that language models technologies can be used in different ways to extract information from unstructured data. While being applied to sport, many other fields could benefit from it. Coming from a background in economics, I would be fascinated to build a similar model to understand buyers' behaviors on websites, agents strategies on financial markets or price strategies in specific sectors.

9 Disclaimer

This project has been made on a NFL data set from a Kaggle competition. Views expressed in this report are my own. While being a report, this work does not stand as an academic paper, explaining while the tone can be more informal some times. I will shortly publish a clean repo on GitHub with the code being standardized and cleaned. Currently, all the code is on Colab Notebooks making it long to read and understand quickly.

10 Annex

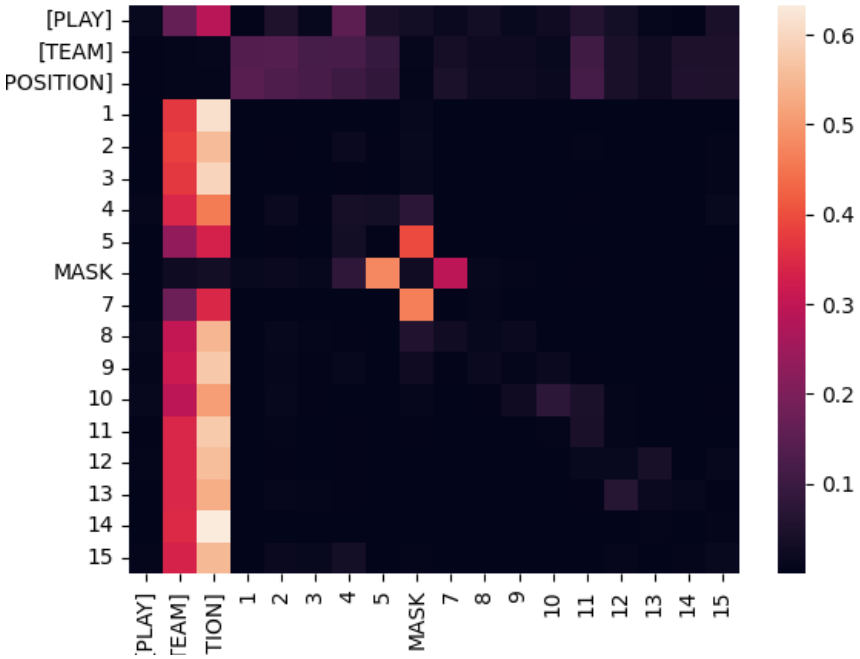


Figure 23: Example of Attention matrix from 1st layer

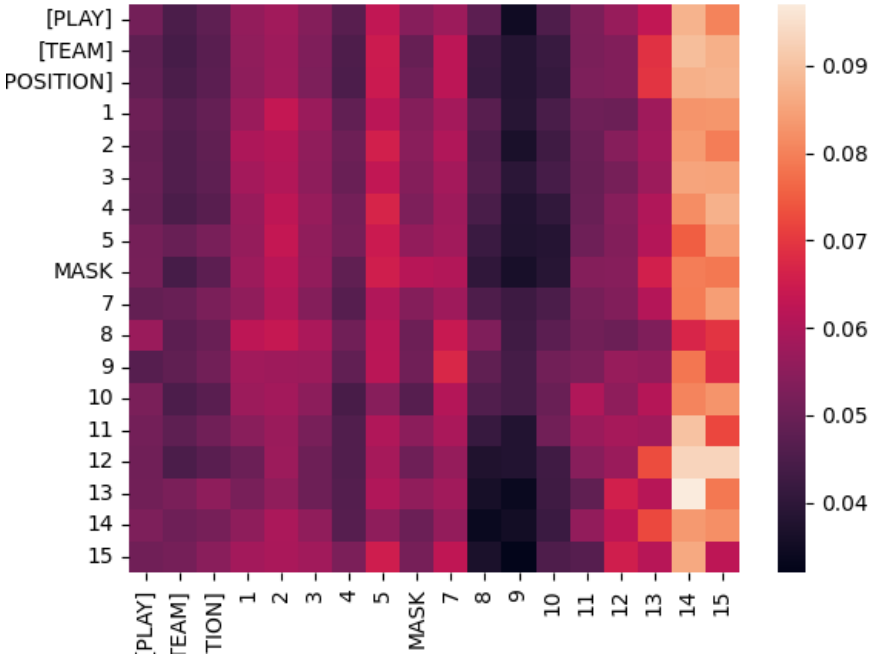


Figure 24: Example of Attention matrix from 2nd layer

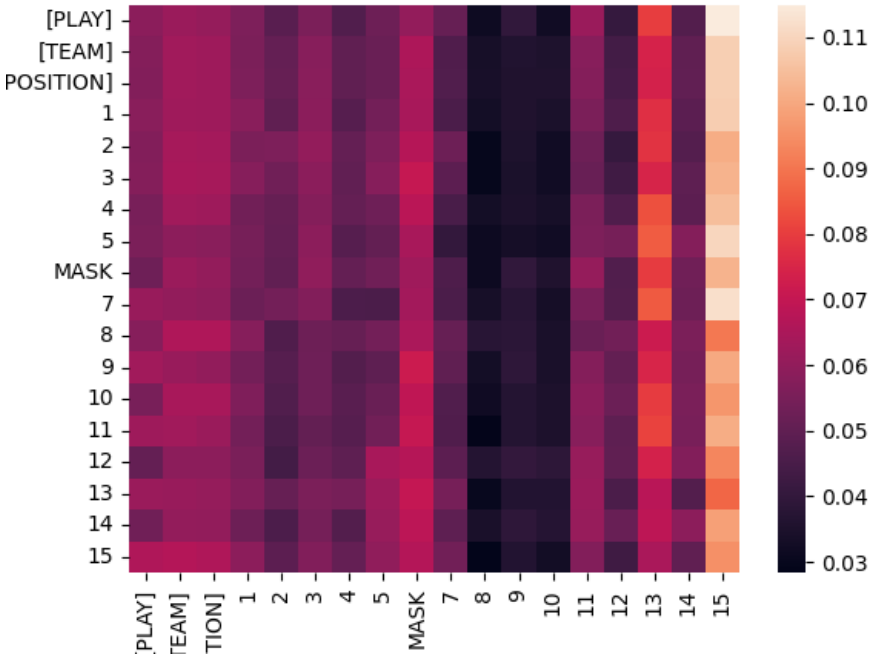


Figure 25: Example of Attention matrix from 3rd layer