Pitch set up

Pitch Coordinates

The origin of the coordinate system is defined to be at the centre spot with the y axis directed along the width of the pitch and the x across along the length such that the sidelines are at $y = \pm 3400$ cm and the defending goal is at $x = -5250$ cm

[Focus on the pitch lines and direction of play patterns]

Standardisation

As we are working with a range of data recorded at different football grounds, the data needs to be standardised

As not every pitch is the same size

We define the standardised pitch dimensions to be 105m in the x and 68m in the y.

These were chosen as they are the only studying [redacted] they play half of their matches at home so we chose this to be the baseline as it is the most common

So every x coordinate is transformed by $\frac{\text{pitch x range (real)}}{105}$

and y coordinate by $\frac{\text{Pitch y range (real)}}{68}$

Normalisation

Each game also needs to be normalised → [redacted] always attack in the same direction

The opta data defaults to the home team attacking left to right (+ve x direction) and the away team right to left (-ve x)

However as [redacted] is a constant in all of our data we define that [redacted] should always be attacking right to left (normalising)

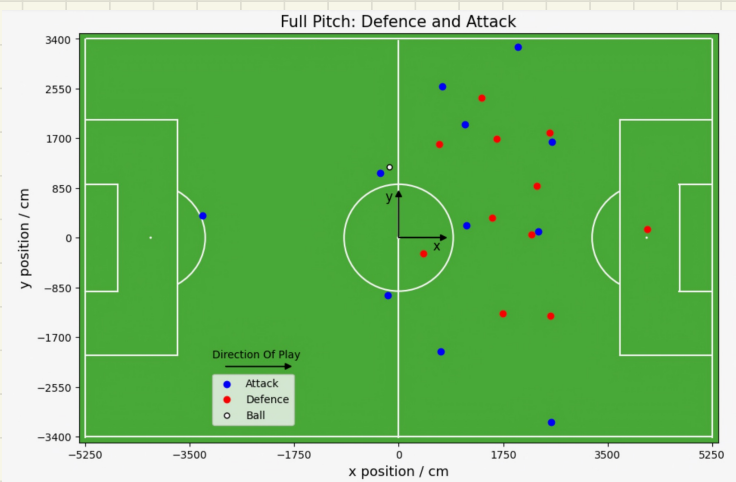
This is achieved by [redacted]

If they are they already are attacking RTL so no transformation is needed

If they are at home every player and ball coordinate is rotated around the origin (x and y multiplied by -1)

ie [redacted] (the home team) now plays right to left

A full representation of all 8 of these definitions



Data Structure

Inputs + outputs

Inputs : The model takes Coordinates of the 11 attacking players as an input + the coordinates of the ball

It still needs to be decided whether the 3 Coordinates of the ball adds any benefit to the model or if it just confuses it

Likewise it should be considered whether the model benefits from the position of the attacking GK

Outputs: The model returns the Coordinates of the 11 defending players

Data Shape

Each time step is defined as a frame, the opta data tracks at 5 fps so each time step is 0.2 seconds

The aim of the project is to accurately predict how [redacted] would structure defensively against a given attacking formation

It needs to be noted that the structure of the defense does not just depend on the current attacking frame but also preceding frames

Therefore a simple neural network cannot be used, one one is required to find a relationship between the current defensive formation and the current attacking formation but also preceding attacking formations

↳ rnn / can learn time series data.

Therefore the frames need to be built also sequences [defined as a series of frames]

the length of a sequence was determined by modelling a game as a Markov chain (See transition matrix notebook)

This proved that the position of the ball is 2 frames is completely independent if the frames are more than 10 seconds apart

So the sequence length was defined at 9 seconds to try to predict a sequence that is owing as possible but also only contains frames that are not statistically independent

↳ 45 frames

It is also worth noting that the data was split batched to train faster, ie train on multiple sequences simultaneously

So the input data had shape: (batch size, sequence length, number of coordinates)

Batch size was later treated as a hyperparameter, but Sequence length = 45

and number of coords = 25 for gk and 8 ball Coords

Output shape was the same but had 22 Coordinates instead

Sparse Matrix Data Shape

Another option is to structure the data as a matrix. The matrix has the same dimensions as the pitch -> ie quantised into a 1m² grid

The positions of players are then represented by a 1 in the corresponding grid cell, everywhere else is a zero

This (hopefully) could be more efficient than coordinates but we have Conv function issues (detailed in conv function section) and RNN issue (only take 1D input)

This is an area for next semester

Data filtering

The first filter on the data was [redacted] not having the ball. Note from here [redacted] will be referred to as the defending team

This ensures that the defending team are always facing an attack and we can investigate how they line up defensively

The next filter is cutting off when the defence is in a 'defensive' position

If the defending team have players pressing into the attacking team's half, this is not a defensive position, it's a pressing position

So we remove any frame that has defending players in the attacking half.

This selection cut is somewhat arbitrary as it is difficult to quantify a defensive position

But the cut was not made any deeper due to how much data would be lost

-> Think of a way to do this

note this doesn't filter for the actual formation just the lineup

Another cut is also made for 4 defenders, 4 midfielders and 2 attackers needing to be in the defending team

We defined a 4 4 2 as a 'box standard' defensive line up.

-> same for a 4-5-1 variant

Any 5 back formation would give too little data -> but would be 'more defensive'

and any 4-3 3 variation is too attacking

So any games / substitutions that don't meet this criteria are culled

And finally, simple cuts such as, there must be 22 players within the pitch boundaries as well as the ball, and the ball must be in play (not dead) for a frame to be 'alive'

So to summarise, the list of criteria that defines an 'alive' frame:

- 22 players + ball in pitch boundaries
- Ball is alive
- 4 4-2 lineup
- 11 defenders in their own half
- Attacking team having the ball

The continuous play is then split into episodes which are constant phases of play where each of these criteria are met.

As soon as one of these criteria isn't valid the episode will end.

Any episode shorter than 45 frames is deleted (maybe introduce padding later)

The episodes longer than 45 frames are then quantised into sequences.

Note that this is done from the end of the episode backwards.

This helps to remove any 'messy' starts such as a corner, free kick etc. as the disorder will be removed if the episode is not an integer multiple of 45.

RNNs do not accept variable sequence length
if we want to use shorter sequences, need to pad inputs, however, this caused problems.
initially padded with zeros then first frame
better to back-fill then forward fill

Ordering output / Expected

The opta data does not order the players in the same order for each frame.

Initially, the output and the predicted data was being compared element wise:

Pred : $[x_1, y_1, x_2, y_2, x_3, y_3, \ldots, x_{11}, y_{11}]$

Exp : $[x_1, y_1, x_2, y_2, x_3, y_3, \ldots, x_{11}, y_{11}]$

However, the expected array is not always in the same order, sometimes, x_1 may be the coordinate of the gk, sometimes the st.

So the model will struggle to learn what to put in the first index as the expected is always changing and this is what it is being compared to.

So we have 2 options:

- make sure the expected is always in the same order [Gk, Lb, Lcb, rcb, rb, ...]
- make the loss calculation non-index dependent

We first tried the first option by sorting: [Gk, Def, Def, Def, Def, mid, ...]

But there is no further player data to sort by position, and due to variations in the order of each group, we then sorted each group by shirt number.

However, there is no guarantee this order is constant between games → this caused a spike in loss throughout the season

So, making the cost independent of the order of the expected array was the next option.

ie, the model doesn't have to predict index 2 to match index 2 in the expected, it just needs to predict the coordinate at some index in the predicted array.

This was done by defining a cost matrix for each frame, this describes the loss of each pairing in the pred and exp arrays.

$$\text{Cost\_matrix} = \begin{pmatrix} L_{11} & L_{12} & L_{13} & \cdots \\ L_{21} & L_{22} & & \\ L_{23} & \ddots & & \\ \vdots & & & \end{pmatrix}$$

where $L_{ij} = (r_i - r_j)^2$

and r_i is an expected coordinate and r_j is a predicted coordinate

The Hungarian algorithm is then used to find the minimum of each row without repeating combinations.

Along the same vein, the model should be invariant to changes in the order of the inputs.

The idea is to sort the attaching coords (x only) in increasing order, then have the ball x and y at the end.

Transition matrix

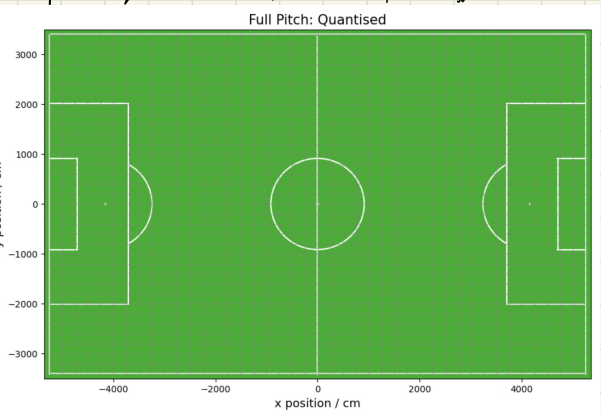
weeks 4-6

Here we focus on how long each sequence of play should be.

The aim is to find how long it takes for 2 events to be independent in a game of football

For this we modelled a game of football as a discrete - homogeneous Markov process.

The pitch was quantised into $0.25m^2$ boxes → note each pitch has different dimensions so this was normalised to be (136×210) → most common size



the cell of the ball at each time was recorded along with which cell the ball is in at the next time.

Defining transition matrix

This data was then used to define the transition matrix for the Markov process.

The transition matrix is defined as:

$$\underline{P}_{i+1} = \underline{\underline{Q}}\,\underline{P}_i$$

where $\underline{P}_i$ and $\underline{P}_{i+1}$ are the probabilities of the location of the ball at i and $i+1$ respectively

$$\underline{\underline{Q}} = \begin{pmatrix} q_{11} & q_{12} & q_{13} & q_{14} & \cdot & \cdot \\ q_{21} & q_{22} & \cdot & \cdot & \cdot & \cdot \\ q_{31} & \cdot & & & & \\ q_{41} & \cdot & & & & \\ \cdot & \cdot & & & & \\ \cdot & \cdot & & & & \end{pmatrix}$$

In this case $\underline{\underline{Q}}$ is $([68 \times 105] \times [68 \times 105])$ dimensional.

where q_{ij} represents the probability of the ball moving from cell $j \to i$ in one time step

So after all positions of the ball were found, each index of the $\underline{\underline{Q}}$ matrix was filled in

ie if the ball was in cell (1, 1) at $t=0$ then (1, 2) at $t=1$ then $\underline{\underline{Q}}_{(1,2),(1,1)}$ was given a count.

To finalise defining $\underline{\underline{Q}}$ it needed to be turned into probabilities as opposed to counts

As each column should sum to 1 (all possible movements from $i \to$ all $j = 1$)

So each column was normalised by its sum

Stationary state

The stationary state of a markov process is defined as:

$$\underline{\underline{Q}}\,\underline{P}^{st} = \underline{P}^{st}$$

ie, any further application of $\underline{\underline{Q}}$ will have no effect on $\underline{P}$, the system will evolve no more

This corresponds to the eigenvector of $\underline{\underline{Q}}$ that has an eigenvalue of 1.

Due to the large size of $\underline{\underline{Q}}$ here, the eigenvectors could not be solved for analytically

So power iteration was used to find $\underline{P}^{st}$

The eigenvector with the largest eigenvalue can always be found by iteratively calculating

$$\underline{V}_{n+1} = \frac{\underline{\underline{Q}}\,\underline{V}_n}{|\underline{\underline{Q}}\cdot\underline{V}_n|}$$

Until $\underline{V}_n$ converges. Note V_0 is a random vector.

↳ below a chosen tolerance (we chose the modulus to be $< 10^{-6}$)

Note that for a given t, $\underline{P}(t) = \sum_i C_i (\lambda_i)^t \underline{V}_i$

so as $t \to \infty$, $\underline{P} \to C_n \underline{V}_n$ where n has eigenvalue 1.

Also note that $|\underline{P}| = 1$ (probability) so C_n and thus P^{st} can be found by normalising $\underline{V}_n$

Once $\underline{P}^{st}$ was found, the time taken for an initial condition to reach the steady state was found.

This was done by iteratively calculating $\underline{P}_{t+1} = \underline{\underline{Q}}\,\underline{P}_t$ until $\underline{P}_{t+1} = \underline{P}^{st}$ with a 5% tolerance on the magnitude.

This proves how long it takes two events to be independent as the stationary state is independent of the initial state.

So a sequence longer than this should not be used as it will contain frames that are statistically independent.

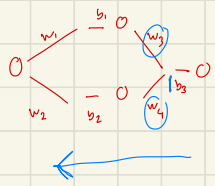
The time taken to reach the stationary state was found to be $\sim 10s$

$$P_{t+1} = Q\,P_t$$

$$P_{t+2} = Q^2 P_t$$

$$P_{st} = \boxed{Q^{\wedge}} P_t$$

$$P_{st} = \boxed{Q^{n+1}} P_t$$



Cost Function

weeks 2-3 & 7

initial definition

For the network to learn there needs to be a quality of output measure -> between true data and the prediction

Initially, the idea was to simply calculate the mean squared error of each frame: $Cost = \frac{1}{N}\sum_{i=1}^{N}\left(r_i^{exp} - r_i^{pred}\right)^2$ => N = 11 (reduce to 10 and ignore GK?)

This would then be averaged over the 45 frames of the sequence -> the sequence has one value for the cost (not one for each frame)

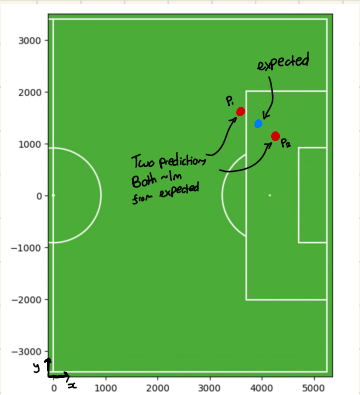
And if the data is batched, the average of each sequence would be calculated for the cost of the batch.

Angular Cost

It quickly became apparent that the model would never converge to a zero cost -> it would always be finite (especially for the testing data)

But the same modulus between the predicted and expected position can correspond to very different positions.

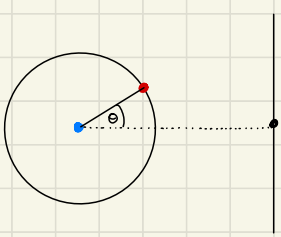
ie consider a modulus of ~1m between the predicted and expected:



Despite both predictions having the same cost, P2 is obviously a better prediction from a footballing opinion.

So we needed to define a cost function that would return a sensible defending position if the exact position is not predicted

If the expected position has the same y coordinate as the centre of the goal this is straight forward



A circle is drawn to show the loci of points would have the same distance based cost.

Our new cost function seeks to minimise the angle between the line connecting the centre of the goal and the expected and the line between the predicted and the expected.

ie θ=0 corresponds to the expected, predicted and goal being in line.

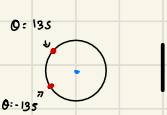
Using atan2 numpy function

It is also worth noting that we defined the maximum cost to be 180

Any thing larger than this would be negative. This speeds up the minimisation of the cost =)

as it can be reduced by moving anticlockwise in the negative segment.

ie if $0<\theta<360$, and θ: 359, to reduce it the position would have to be moved through 359°.

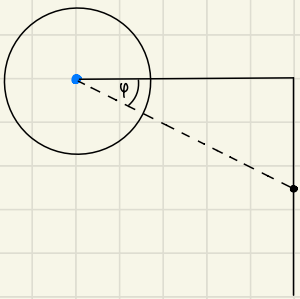


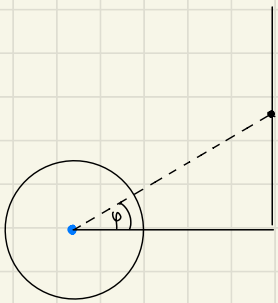
However, with $0 < |\theta| < 180$, if $\theta = -1$, it only has to be moved $1°$ to be zero. => much more efficient

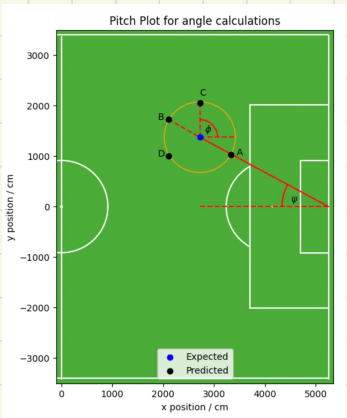
↳ back up with learning time graph?

Or simple enough to just claim?

When the expected isn't inline with the centre of the goal, it is slightly more complicated.







The angle between the expected position and the centre of the goal is defined as φ. This is $0 \to 90$ for above the $y = 0$ axis

and $0 \to -90$ below $y = 0$

The angle from the horizontal is then calculated as before but the difference between θ and φ is minimised => zero angular cost leaves the prediction inline with the centre of the goal.

Exponential loss

We have played with the idea of introducing an exponential decay to the importance of the distance -> ie, when the prediction is far from the expected, the angular cost is negligable, only changing the distance will reduce the overall cost.

But when the distance $\sim 1m$, the angle becomes equally important as the distance to reduce the cost.

mathematically, without an exponential, the total cost is:

Cost = distance cost + angular cost

As the distance cost is measured in cm^2, the two are comparable when the avg distance$^2 \simeq 200\ cm^2$

So the average distance needs to be $\sim 15cm$ -> this is way too small

Initially we added a simple multiplying factor to the angular cost

Cost = distance cost + (50 x angular cost)

This means the angle cost becomes important at an average distance of $\sim 1m$

This is much better, however, we would like the distance cost to not be linear in hope to train the model faster

So the new cost will have the form:

Note that the loss for a sequence is calculated as the average of each frame

And a batch is the average of each sequence

Sparse matrix loss (issues)

week 2

Down the line we will implement a sparse matrix format.

Initially the idea was to simply extract the coordinate of each position from the matrix, then do all of the previous steps

However, extracting the coordinates is a non differentiable operation, there is no associated gradient function so back propagation can't be done --> model doesn't learn

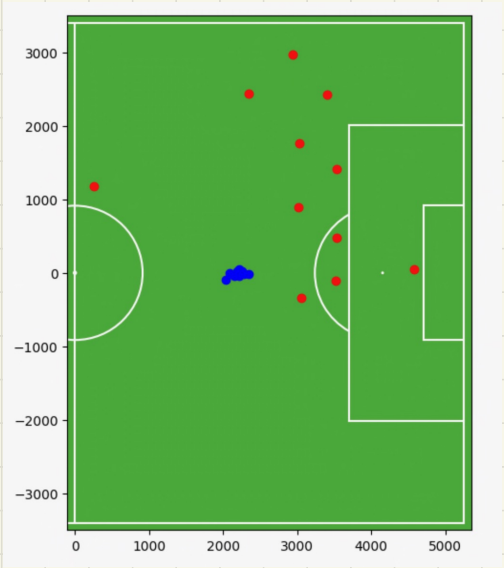
One method to fix this involves simply finding the difference between the matrices and summing the components.

However, this will be very step like, the model will only be rewarded when predicting the exact position, it will not be guided in the right direction

more thought is needed on this next semester.

For a long time (weeks 5-6) had problems with the model learning.

Kept getting results like this:

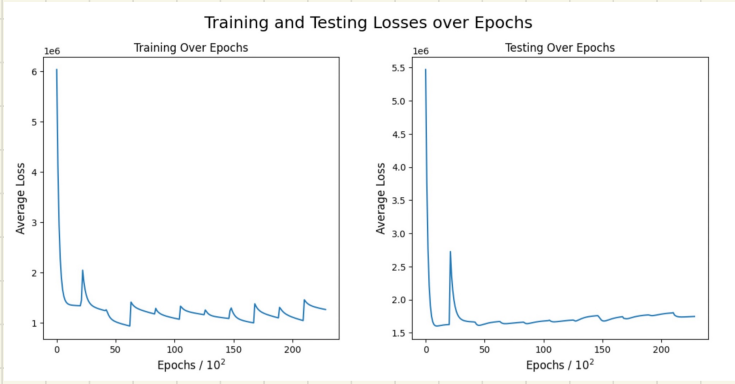


model wanted to predict all players together.

I think this was a issue with padding sequences with the RNN -> RNN tended to predict average positions.

If padding, -> predicting own position

So we removed padding (for now) just took sequences longer than 45 frames



loss never got below $\sim 10^6$

Network architecture

Week 8-9

Note for this section, the hyperparameters include:

- Batch size
- Epochs
- layers
- hidden dimensions
- Learning rate

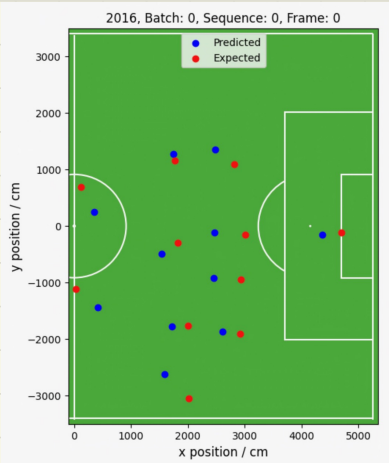
most of these are tuned later for fixing overfitting

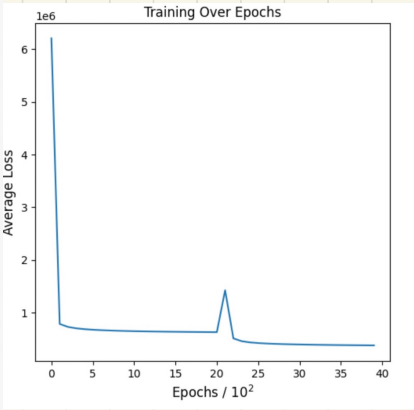
Apart from LR and layers.

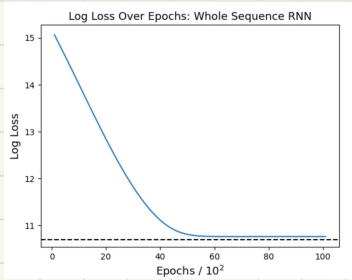
RNN

The first aim is to get a model that can accurately learn the training data -> not worried about testing

First we used a basic 1 layer RNN, however this struggled to capture the time series data, it tended to predict the average position of each player across the sequence.

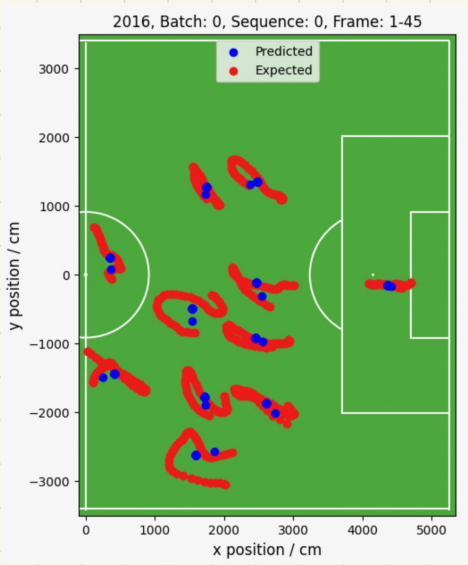






Viewing the whole sequence shows that the prediction doesn't really move across the sequence, but is still being predicted in the right area.

↳ each prediction is on the right expected path

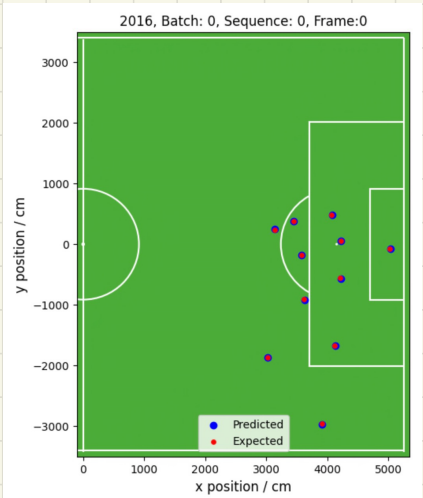


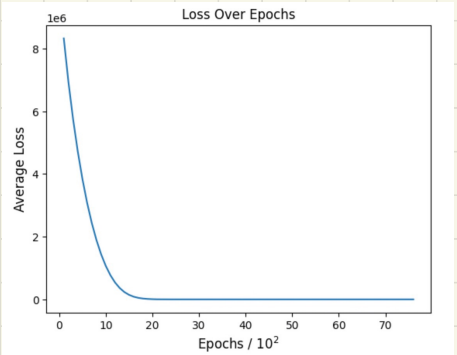
its clear to see from the loss graph that the model has the ability to learn, however, it asymptotes at around 50,000. This is an average distance per player of ~ 2 m

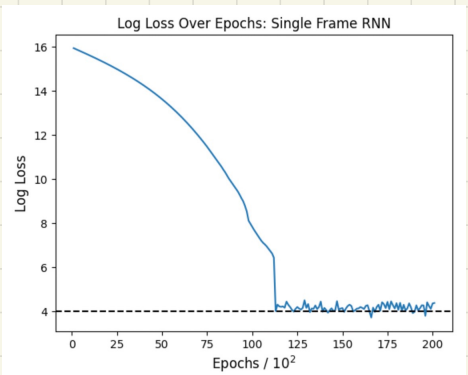
This isn't bad but it's clear that the time essence of the data isn't being captured.

Just to Solidify this argument, we trained the model on a stationary sequence (every frame is the same)

If our previous hypothesis is right, this should be almost perfect.







It can be seen from the pitch plot that this gives a perfect solution, this corresponds to a log loss of ~ 4, ie

loss ~ 50 cm per player.

We define loss of order ~ 100 as optimum.

We will use this as a basis to build a model that can predict sequences, games and seasons.

LSTM

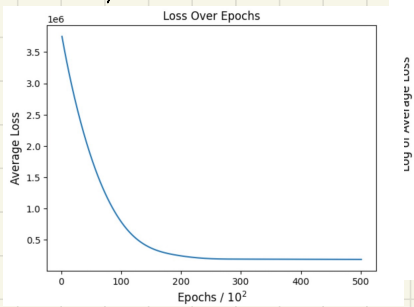
Week 10 -12

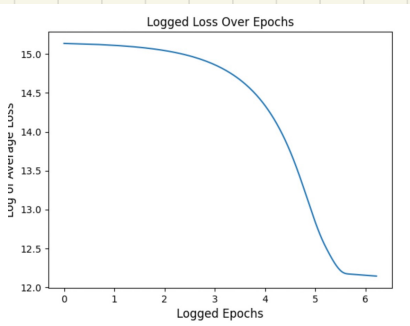
We now move towards an LSTM, these are more capable at capturing long term time dependencies.

We initially start with just a sequence, we've accurately predicted a frame, the sequence is the next step.

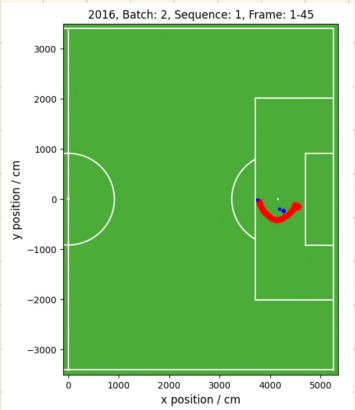
We start with a 1 layer LSTM with 64 hidden dimensions

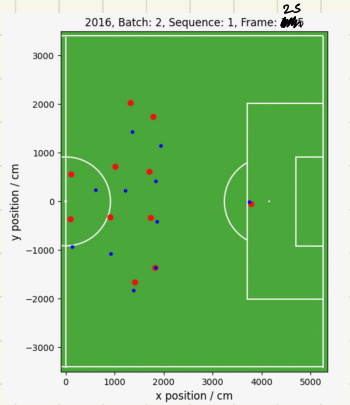
For 1 Sequence:





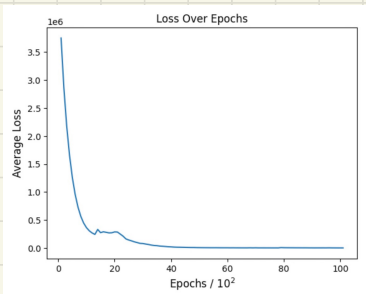
This loss asymptotes at $\sim 150\_000$ (ie quite bad) but looking at the pitch:

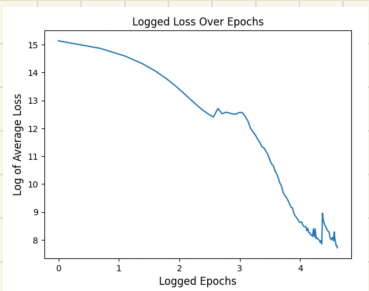


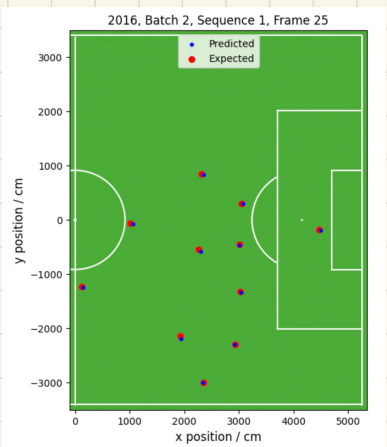


it is starting to be able to predict some time evolution.

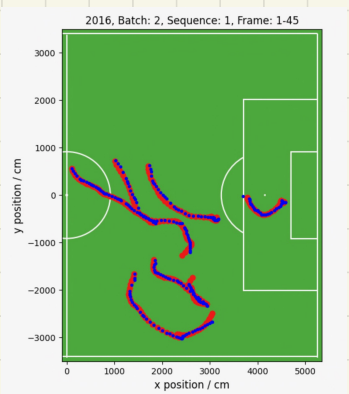
By simply increasing to 2 layers:







change



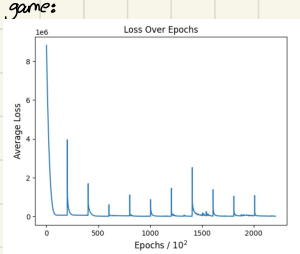
This is accurately replicating a frame but also a whole sequence

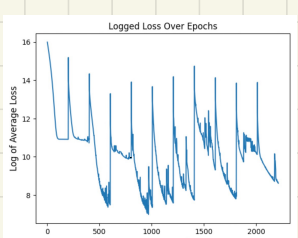
← Plotting 6 player paths to demonstrate Cartesian system

loss asymptotes at ~2000 ~ e^8

Slower this takes ~ 10^4 epochs

And then for one game:

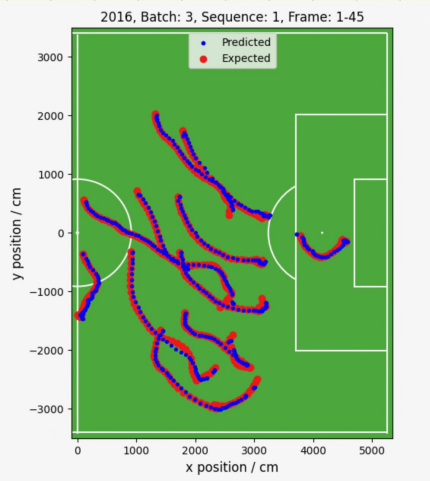


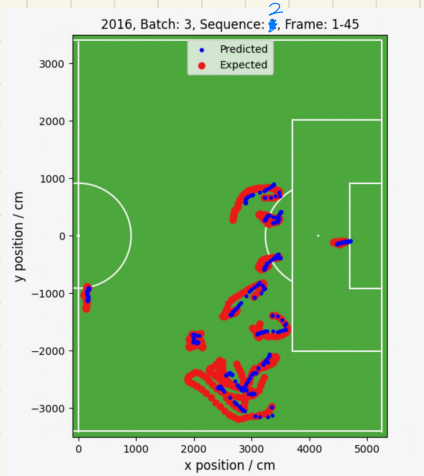


Its clear to see for some sequences, it learns very well -> reaching a loss of ~2000

However, there are a couple that it cannot learn well, asymptoting at a loss of ~ $e^{10} \sim 20,000$

investigating some of these batches.





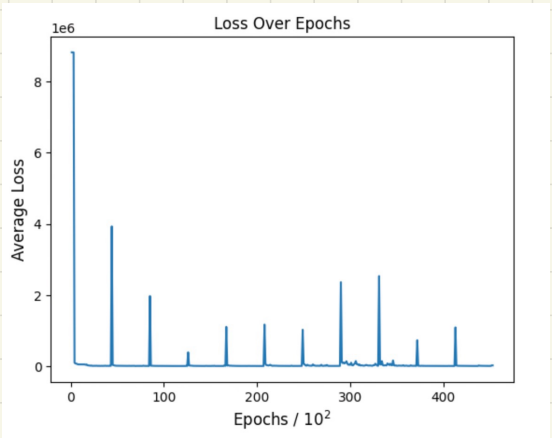
Batch 2 is roughly the same as before, but batch 3 which asymptotes at ~ 20,000 struggles to predict all of the player paths over the whole game.

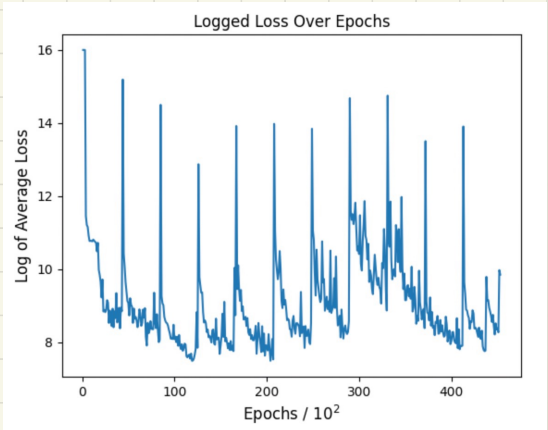
So we look into expanding the LSTM to improve the predictions over a game before looking into multiple games

After some trial and error with different activation functions and other layers

we settle on a 2 layer lstm with a dropout layer, 2 fully connected linear layers and a leaky ReLU activation ~~layer~~ function

For the same game:





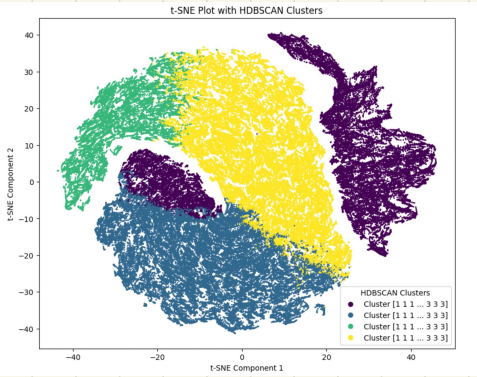
Its clear to see here, the batches that were previously asymptoting at ~20,000 are now ~2000 -> huge improvement.

The highest a batch asymptotes is $\sim e^9 \sim 8,000$.

this is not perfect (~ 8m between each player)

In an attempt to clean the data (remove groups that are difficult to predict)

we tried clustering:



This showed some promise at sorting corners ect...

But needs more work done.

Might be easier to just manually clean data

All Files are Saved on databricks

Rough notes From here

$Q = \begin{pmatrix} q_{11} & q_{12} & q_{13} \\ q_{21} & q_{22} & q_{23} \\ q_{31} & q_{32} & q_{33} \end{pmatrix}$ — Probability of 2→1 in 1 time step

$Q^2 =$ — Probability of 2→1 in 2 time steps ↳ $P(X_1 = 2 \cap$

$= \begin{pmatrix} q_{11} & q_{12} & q_{13} \\ q_{21} & q_{22} & q_{23} \\ q_{31} & q_{32} & q_{33} \end{pmatrix} \begin{pmatrix} q_{11} & q_{12} & q_{13} \\ q_{21} & q_{22} & q_{23} \\ q_{31} & q_{32} & q_{33} \end{pmatrix}$

$= \begin{pmatrix} q_{11}^2 + q_{12}q_{21} + q_{31}q_{13} & \cdots & \cdots \\ q_{21}q_{11} + q_{22}q_{21} + q_{23}q_{31} & \cdots & \cdots \\ q_{31}q_{11} + q_{32}q_{21} + q_{33}q_{31} & \cdots & \cdots \end{pmatrix} \begin{pmatrix} \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \end{pmatrix}$

$Q^2 = \begin{pmatrix} (q_{11}^2 + q_{12}q_{21} + q_{31}q_{13})q_{11} + (q_{21}q_{11} + q_{22}q_{21} + q_{23}q_{31})q_{12} + (q_{31}q_{11} + q_{32}q_{21} + q_{33}q_{31})q_{13} & \cdots & \cdots \\ (q_{11}^2 + q_{12}q_{21} + q_{31}q_{13})q_{21} + (q_{21}q_{11} + q_{22}q_{21} + q_{23}q_{31})q_{12} + (q_{31}q_{11} + q_{32}q_{21} + q_{33}q_{31})q_{23} & \cdots & \cdots \\ \cdots & \cdots & \cdots \end{pmatrix}$

$P(X_3 = 1 \mid X_0 = 1)$

$P_{i+1} = Q P_i$

$P_{i+2} = Q P_{i+1}$

$= Q^2 P_i$

$\vdots$

$P_n = Q^n P_0$

if $P^{11} = Q P^{11}$

$\begin{pmatrix} 4 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{pmatrix} \begin{pmatrix} 4 & 1 & 1 \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}$

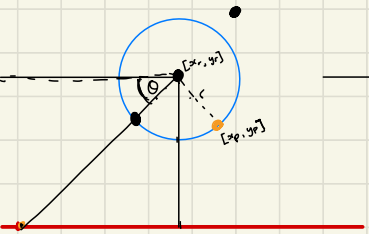
$1 = l_{21}(4) + 1(0) + 0(0)$

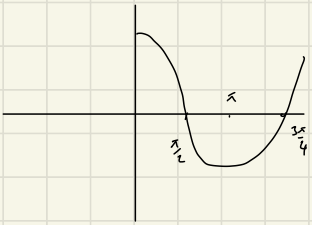
So independent is $P(X_1 = 2 \cap X_2 = 1) = P(X_1 = 2) \times P(X_2 = 1)$

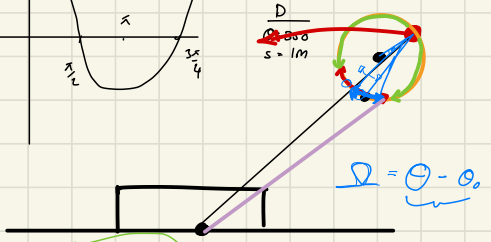
↑ Probability of being in State 2 at time 1

↑ Probability of being in State 1 at time 2

↑ Probability of being in State 2 at time 1 and State 1 at time 2



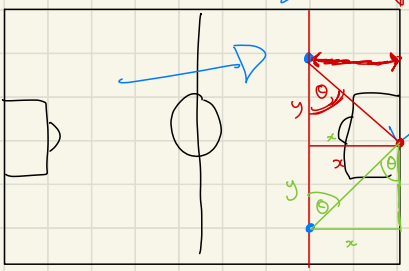




5250

10500

105/2



$\tan^{-1}\left(\frac{x}{y}\right)$

68

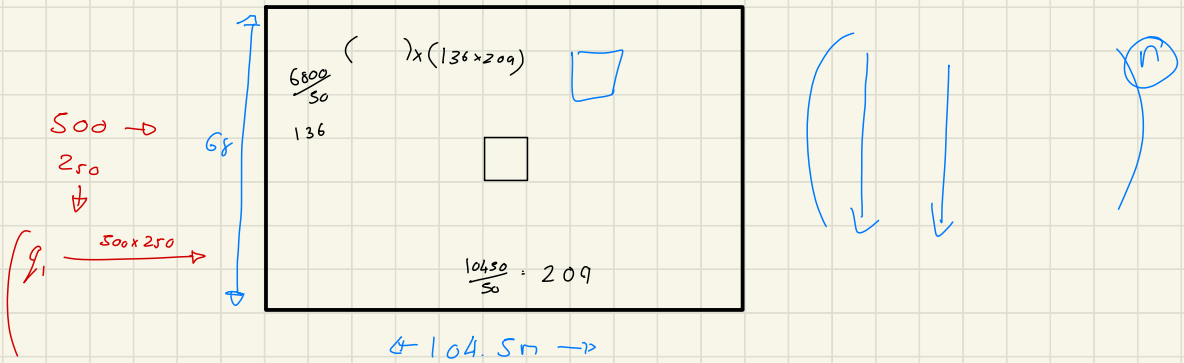
34

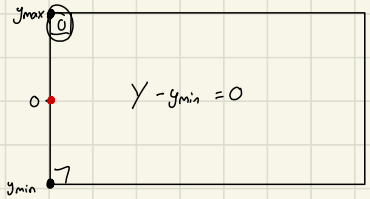
$a^2 = b^2 + c^2 - 2bc\cos\theta$

$\cos^{-1}\left(\frac{b^2+c^2-a^2}{2x}\right) = \theta$

68

105





$Y - (y_{min}) =$

$Y_{max} - (y_{min}) = \frac{6000}{50}$

$Q = \begin{pmatrix} q_{11} & q_{21} \\ q_{12} & q_{22} \end{pmatrix} \begin{pmatrix} q_{11} & q_{21} \\ q_{12} & q_{22} \end{pmatrix}$

$Q^2 = \begin{pmatrix} q_{11}q_{11} + q_{21}q_{12} & q_{11}q_{21} + q_{21}q_{22} \\ \vdots & \vdots \end{pmatrix}$

2169

3644

We want defence to always be playing R2L

So if TOI is away → no change

if TOI is home → reverse

now A/D doesn't matter

we always want City to be L2R

So if home → nothing,

if away → reverse

data = [x7, y, z, a]

$$\underline{\underline{Q}} = \begin{pmatrix} \frac{2}{3} & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} \end{pmatrix} \Rightarrow \lambda_1 = \frac{1}{6} \quad , \quad \underline{V}_1 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

$$\lambda_2 = 1 \quad , \quad \underline{V}_2 = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

↳ always has to have a $\lambda_i = 1$

↳ otherwise as $t \to \infty$ $\underline{P}(t) = 0$

Suppose $\underline{P}(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ (ie in state 1)

need to write this as a superposition of V_i:

$\underline{Q}$ could have 2 or more $\lambda_i = 1$ but this is a reducible Markov chain.

$$\underline{P}(0) = C_1 \begin{pmatrix} 1 \\ -1 \end{pmatrix} + C_2 \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

$$C_1 = \frac{2}{5} \quad , \quad C_2 = \frac{1}{5}$$

↳ steady state solution

So $\underline{P}(t) = \frac{2}{5}\begin{pmatrix} 1 \\ -1 \end{pmatrix}\left(\frac{1}{6}\right)^t + \frac{1}{5}\begin{pmatrix} 3 \\ 2 \end{pmatrix}(1)^t$

If $P(0) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ $\lambda_1 = \frac{1}{6}$, $\underline{V}_1 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$

as $t \to \infty$: $P(t) = \begin{pmatrix} \frac{2}{5} \\ \frac{3}{5} \end{pmatrix}$

$\lambda_2 = 1$, $\underline{V}_2 = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$

$C_1 = -3C_2$

$C_1 + 3C_2 = 0$

$-C_1 + 2C_2 = 1$

$5C_2 = 1$

$C_2 = \frac{1}{5}$; $C_1 = -\frac{3}{5}$

$$P(0) = -\frac{3}{5}\begin{pmatrix} 1 \\ -1 \end{pmatrix} + \frac{1}{5}\begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

So $P(t) = -\frac{3}{5}\begin{pmatrix} 1 \\ -1 \end{pmatrix}\left(\frac{1}{6}\right)^t + \boxed{\frac{1}{5}\begin{pmatrix} 3 \\ 2 \end{pmatrix}}$ ↳ Same steady state

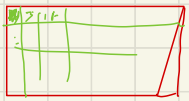
- Steady state solution is independant of initial conditions.
- Just need to:
  - find $\underline{\underline{Q}}$
  - Find $\underline{P}^{st}$
  - Calculate at what t $\underline{P}(t) = \underline{P}^{st}$ (Say with 5% certainty)
  - Remember that $\underline{\underline{Q}}$ has to be defined with probabilities over a constant time step
    - ↳ ie, probability of transition after 1 frame.
    - ↳ then step in t have to be the same as this time step.

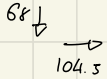
Transition matrix report notes

- Largest eigenvalue has to be 1 (Can't be larger than any entry)
- Can use power iteration methods to find corresponding eigenvector
- or Krylov subspace methods (Arnoldi method)

Finding Q:

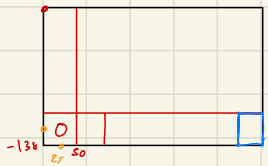
- Divide pitch into $0.25m^2$ areas:

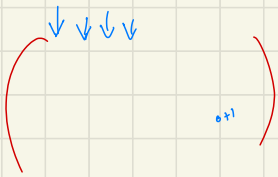




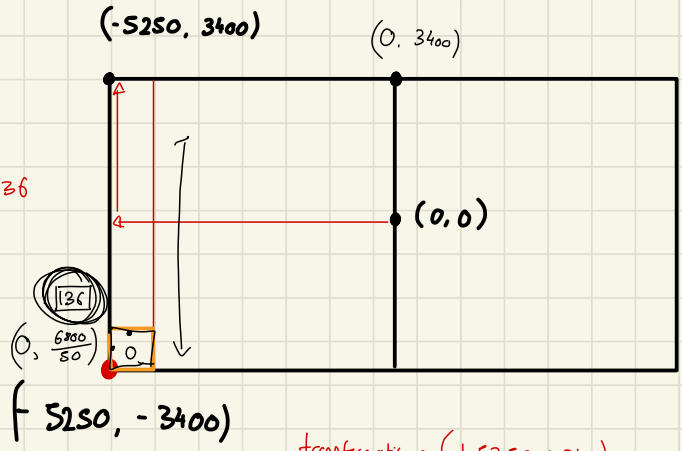
- Find all instances of being in Square 1
- Find position of the ball at the next time step for every instance.
- Calculate all probabilities
- Repeat for every Square (28424 Squares)

28424 x 28424 matrix





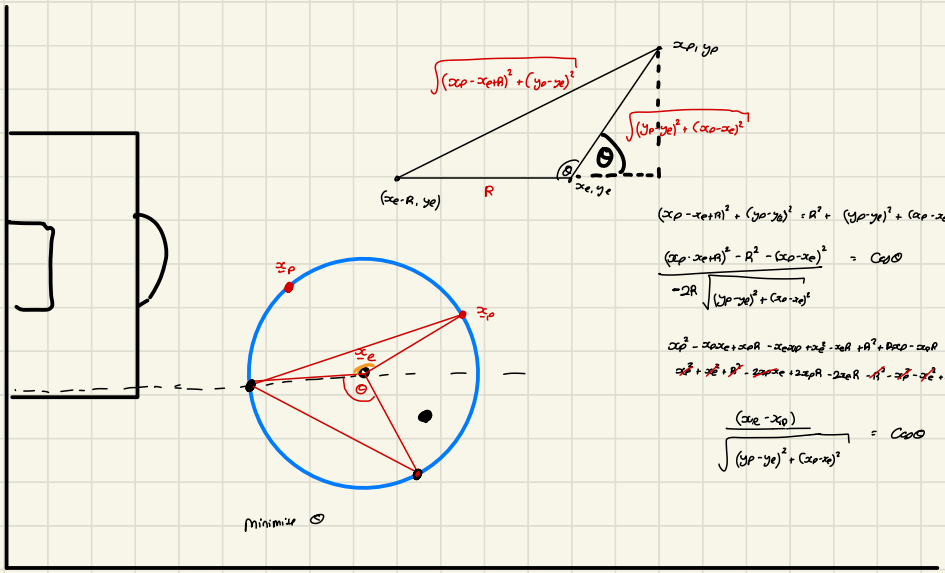
transformation: $(+5250, -3400)$



transformation: $(+5250, +3400)$

no. grid_y: 136

$\frac{\text{Coord}}{\text{grid\_size}}$



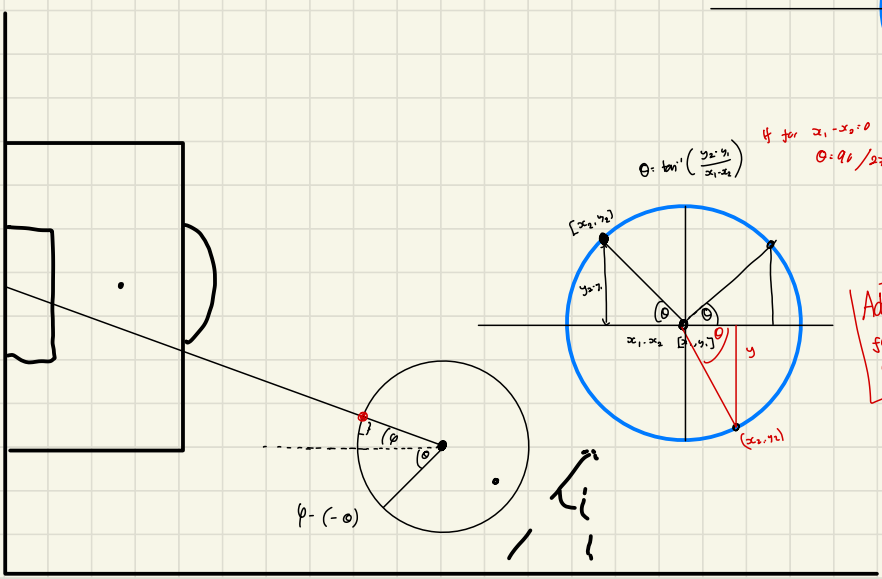
$$(x_p - x_e + R)^2 + (y_p - y_e)^2 = R^2 + (y_p - y_e)^2 + (x_p - x_e)^2 - 2R\sqrt{(y_p - y_e)^2 + (x_p - x_e)^2}\cos\theta$$

$$\frac{(x_p - x_e + R)^2 - R^2 - (x_p - x_e)^2}{-2R\sqrt{(y_p - y_e)^2 + (x_p - x_e)^2}} = \cos\theta$$

$$x_p^2 - x_p x_e + x_p R - x_e x_p + x_e^2 - x_e R + R^2 + R x_p - x_e R$$

$$\frac{(x_e - x_p)}{\sqrt{(y_p - y_e)^2 + (x_p - x_e)^2}} = \cos\theta$$

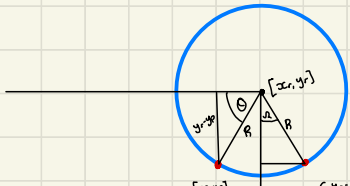
if off axis:



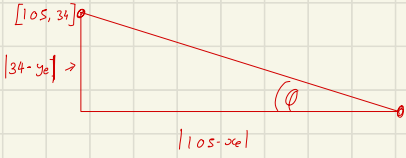
$\theta = \tan^{-1}\left(\frac{y_2 - y_1}{x_1 - x_2}\right)$

if for $x_1 - x_2 = 0$

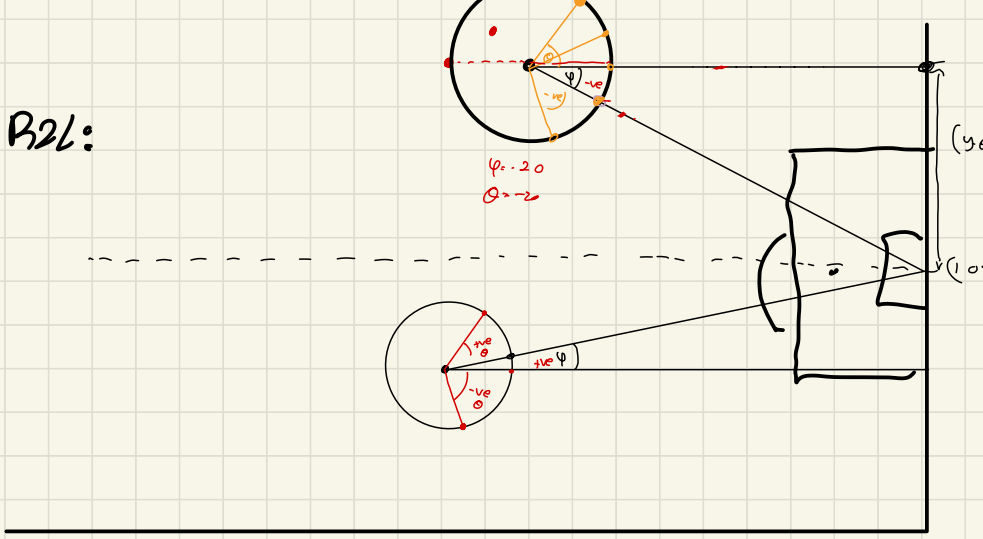
$\theta = 90 / 270$

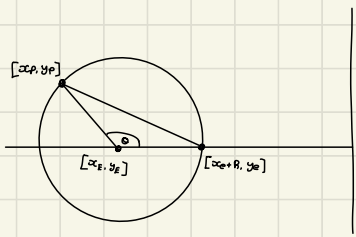


Add logic for which quadrant → no! use atan2



For R2L:





$$1 - \frac{(x_e+R-x_o)^2 + (y_e-y_o)^2}{2R^2} = \cos\theta$$

$$(x_e+R-x_o)^2 + (y_e-y_o)^2 = R^2+R^2 - 2R^2\cos\theta$$

$$(x_e+R-x_o)^2 + (y_e-y_o)^2 = 2R^2(1-\cos\theta)$$

$$= R^2 + (x_E-x_o)^2 + (y_E-y_o)^2 - 2R\sqrt{(x_E-x_o)^2+(y_E-y_o)^2}\cos\theta$$

$$x_e^2 + R^2 + x_o^2 + 2x_eR - 2x_oR - 2x_ex_o = R^2 + x_E^2 + x_o^2 - 2x_Ex_o - 2R\sqrt{(x_E-x_o)^2+(y_E-y_o)^2}\cos\theta$$

$$\frac{(x_p-x_E)}{\sqrt{(x_E-x_o)^2+(y_E-y_o)^2}} = \cos\theta$$

X atan2

Note sequences need to be same length → padding

↳ LSTM, GRU → Also consider activation functions.

- Consider how the initial hidden state is defined

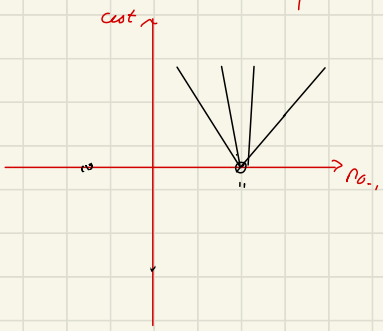
(45, 32, 105×68)

[45, 32, (105, 68)]

- 128 hidden dimensions
- 1 layer

Sparse matrix Cost

$$Cost = |1000\, n_{o.} - 11000|$$



- Cost for when # players != 11:

  C: 100,000 exp(0.0975|x-11|)

  So when x: 12, 10 the cost is ≈ 100,000

  (still leaves range for next cost function to behave exponentially)

  ↳ next cost function will always be below the minimum of this cost

  Cost maximum is less than python int so always have knowledge of increasing

- Cost for when # of players = 11

  $\exp(1376p) + 180 = 110{,}000$ $\exp(0.0083\,mod) + angle$

  P: 0.0084 maximum mod: when all players on opposite sides of pitch (mod: 1376.)

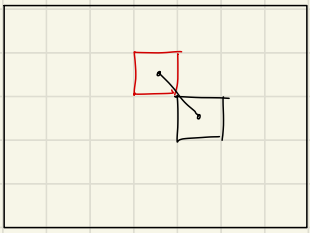
  want this to be just below 110,000

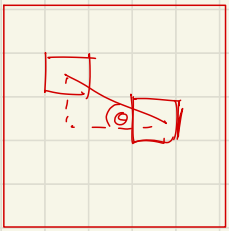
$$x + \frac{1}{1+e^{x}}$$

100,000

≈ 200

$$Cost = 10000(x-11) + e^{0.0975(x-11)} - 1$$





$$e^{0.0000083x} = 270$$

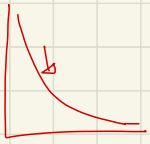
$$0.0000083x \approx \ln(270)$$

$$e^{\sqrt{68^2+105^2}}$$

$$11 \times \sqrt{68^2+105^2}$$

$$= e^{Px1376} = 100{,}000$$

0.0083



model → 12

Cost_A → 1,000

model → 11

Cost_B → 999 ←

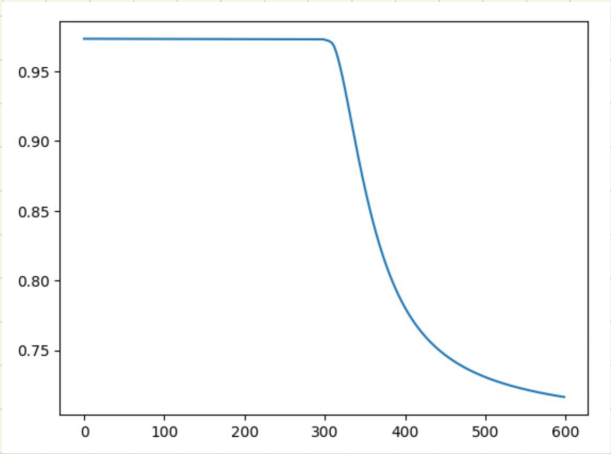
Sparse matrix Format

$$\begin{bmatrix}1\\0\\0\\0\end{bmatrix} - \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

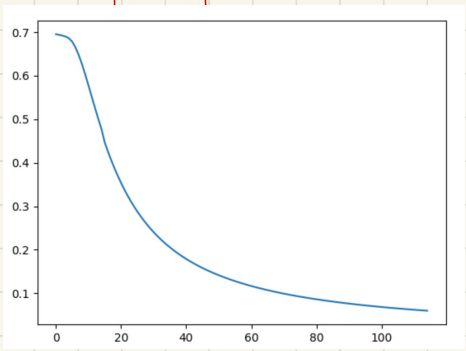
E — T

$$\begin{bmatrix}0\\0\\2\\0\\0\end{bmatrix} - \begin{bmatrix}0.1\\0.2\\0.8\\0.2\\0.3\end{bmatrix} \Rightarrow \begin{bmatrix}0\\0\\0.1\\0\\0\end{bmatrix} \rightsquigarrow 0$$

$$\begin{bmatrix}0\\0\\1\\0\\0\end{bmatrix}$$



BCE Cost (sigmoid before) to try and force model to give 11 one's per frame.



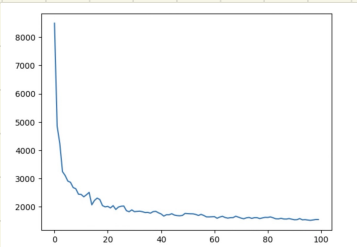
BCE, no sigmoid before.

Realised can't use BCE because of occasional 2's in sparse matrix!

Initially Cost not decreasing, tuned learning rate until Cost decreased as expected

- Tried tuning the output of the model, normalising between 0-1 using sigmoid then rounding to nearest int (turning network output to binary)
- Then put this into a cost function to force the model to predict 11 ones:

$$Cost = |Pred - Expected|$$



=> Cost function over epochs.

However, realised this will try and predict the correct position (not what we want at this point)

- Current goal is to force the model to predict only 11 players per frame. (ie, all zeros apart from 11 ones, or occasional 2, 3)

- So instead, sum each frame after the transform, and minimise Cost = Sum - 11, should reach zero when successfully predicting 11 Players.

However, at the moment, Cost is not reducing even after tuning learning rate. Possibly a problem with Sigmoid. Next, try a uniform transform.