# Week 2 - 05/02/24

## Dataset size

Needed more data, from [redacted] alone we were getting ~10 sequences per game -> ~400 sequences a season -> 2,000 sequences from all seasons. (Based on the 2016 season)

Therefore, we decided to include more teams so chose teams with < 45% average possession over the season. This gave:

2015/16 - leicester, sunderland, west brom,
2016/17 - west brom, sunderland, burnley, leicester
2017/18 - west brom, stoke, newcastle, burnley, brighton
2018/19 - cardiff, newcastle, burnley, brighton, southampon
2019/20 - newcastle, burnley, watford, sheffield united, west ham, bournemouth, villa, palace
2020/21 - west brom, newcastle, palace, sheffield united, burnley, west ham
2021/22 - everton, burnley, newcastle, watford, norwich, brentford

Which is 37 sets of data of a team in a season.
If each team gives the same number of sequences as [redacted] did in 2016, this is ~ 15,000 sequences.

This still doesn't seem like a large dataset, we will try training the model with this and if it still isn't large enough then we will look at altering selection cuts to get more data.

We could also consider using all teams; however, to do this we would have to fine tune our 'park the bus' selection cuts. -> could potentially switch to all players being behind the ball.

Could also resort to some level of padding to increase size of dataset -> ie if sequence is say >40 frames then pad up to 45. So there is minimal padding but still increasing dataset

## Initial 4-4-2 prediction

Tried to give the model an 'initial guess' as to what the defensive structure should look like - ie a basic 4-4-2 structure in the defensive half. However, from research it seems like this isn't possible. We can only set initial values for the parameters (weights and bias') and the hidden layers. In theory it would be possible to tweak these values such that the initial prediction is a well defined 4-4-2 however due to the complexity of the model this is not realistic.

However, we did change the initialisation of the hidden layers (long and short term memory) from zeros to random floats to try to help the learning process. As results are so inconclusive at the moment it is hard to tell if this has helped.
Another method we have found is to use the hidden layer from the end of the previous iteration of the training process as the initialisation of the hidden layer in the next iteration. But there are

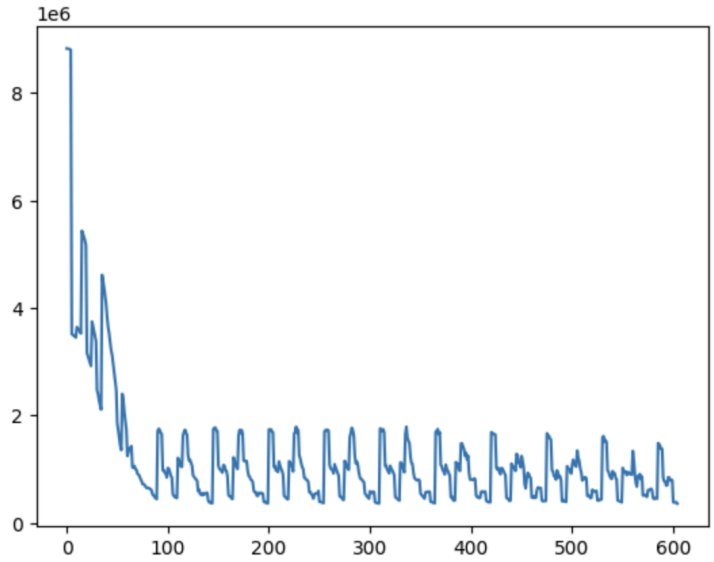
bigger problems to fix at the moment, we will potentially try this later in the semester to fine tune results.

Also note that we are using Xavier initialisation for the initial values of the weights - this is recommended when using an LSTM.

Bias' are currently set to zero to avoid bias towards a certain output, this could be a route to pushing the model to give us a well defined defensive structure. However, it is good practice to set all bias' to a constant value (i.e. all bias' are the same) so we don't think this will be able to recreate the initial 4-4-2 we are looking for but it might help.

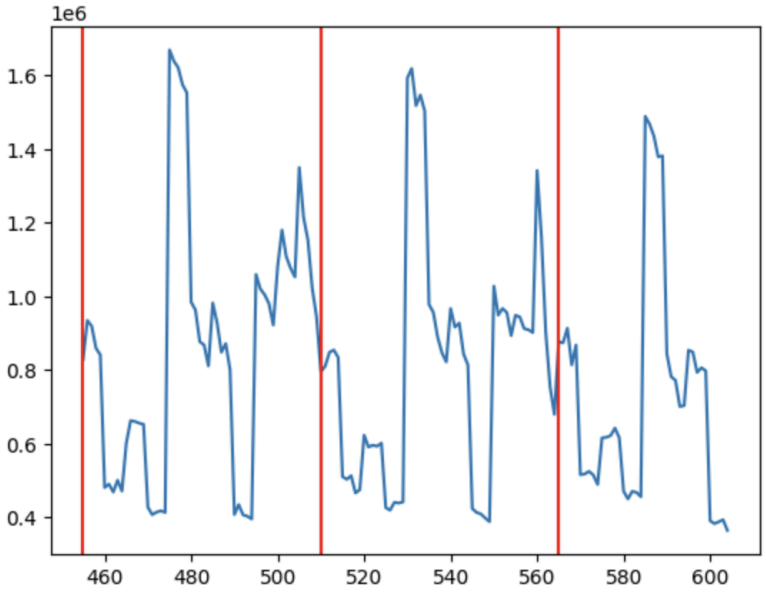
## Fine Tuning Hyperparameters

At the time of our interview, we were using 10,000 iterations over the training set. Since then we have experimented with reducing this.




The above graph is the loss after iterating over the training data 5 times then repeating this 100 times. The first observation is that the model is not learning past a loss of ~ 1e6 - 2e6, however this could be fixed with more cycles over the dataset. The other observation is that despite reducing the iterations from 10,000 to 5, the loss still spikes between sequences. This is seen

better from singling out a few cycles:




It is easily seen how the loss is still spiking after each sequence. This is a problem we would like to talk about in the meeting. We tried a range of different values for number of iterations and number of cycles over the whole dataset, however, the spikes in the loss were always present. We think this might be due to the size of the data set (we are still using a small extract of [redacted] games) . We are working on extracting more data but the run time is long! Other methods of increasing the size of the data set mentioned above could also help here.

The model is taking longer to run than we remembered so the tuning of the hyperparameters is taking longer than expected. However, we have now managed to turn on the GPU so the run time should decrease from now.

## Sorting the order of the input

The Opta data does not output the players in the same order in every frame, if we can find a way to sort these into a constant order between sequences it will help the model learn.

The method we spoke about last week was to average out the positions of the players over a game and assign a position based on their average coordinate.
However, we think that over a season the attacking teams will play too many different formations that sorting will become irrelevant due to the different positions. Ie if in two games the attack plays a 3-5-2 and a 4-3-3 there is no point sorting the attack into a defined order as the players are never going to line up to make it easier for the model. The same argument can be had for sorting to just a [def, mid, att] structure. If we use the same formations as before, this would lead to the model comparing a right back to a central midfielder.

Any sorting will help the model compared to the order being random but we don't think it is beneficial to calculate the average positions, we will sort to the [def, mid, att] structure. This will be enough until we move to the matrix format.

# Week 3 - 12/02/24

## Data Extraction

Started extracting data from all of the teams discussed last week.
Taking ~2 hours per team so it took most of the working week to extract all data.
Concatenated all data into 1 dataset which is ~20,000 sequences.
Also have all team data in separate datasets so we can make new datasets / train on individual teams if we want.

## Sparse Matrices

As we could not do much whilst we were extracting data and training (both clusters were in use) we started trying to fix the sparse matrix data extraction code. Therefore, if training on a larger dataset doesn't work, we can potentially move towards the CRNN.

## Training

Started training on all [redacted] data (this is all we had extracted at the start of the week.
However, throughout Tuesday we were having memory issues with the GPU.
We were training for ~2 hours until we had used up the memory on the GPU and got the following error message:

```
› OutOfMemoryError: CUDA out of memory. Tried to allocate 2.00 MiB (GPU 0; 15.77 GiB total capacity; 644.31 MiB alread
allocated; 1.12 MiB free; 674.00 MiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_
split_size_mb to avoid fragmentation.  See documentation for Memory Management and PYTORCH_CUDA_ALLOC_CONF
```

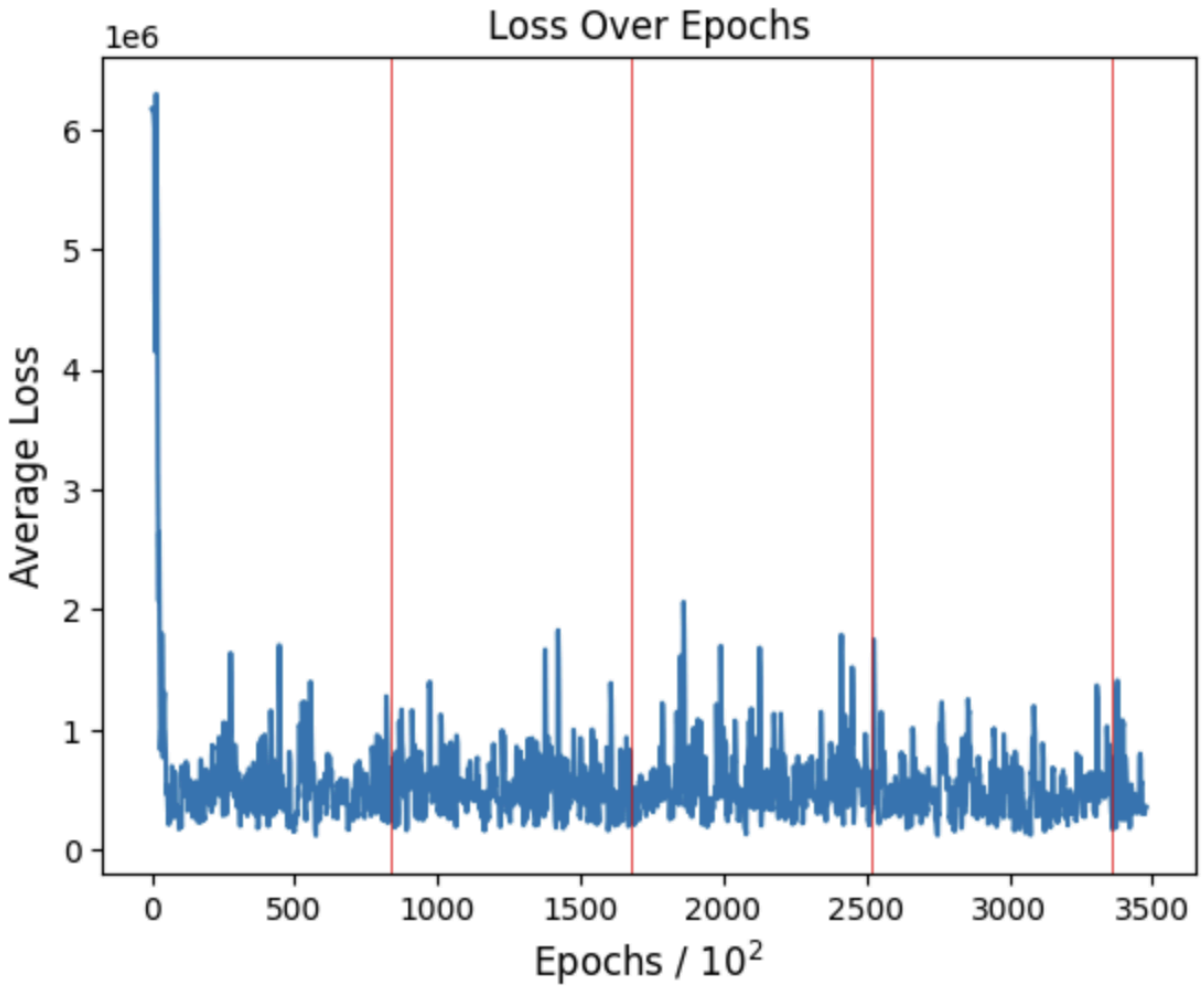
This stunted progress but we managed to get some loss graphs:

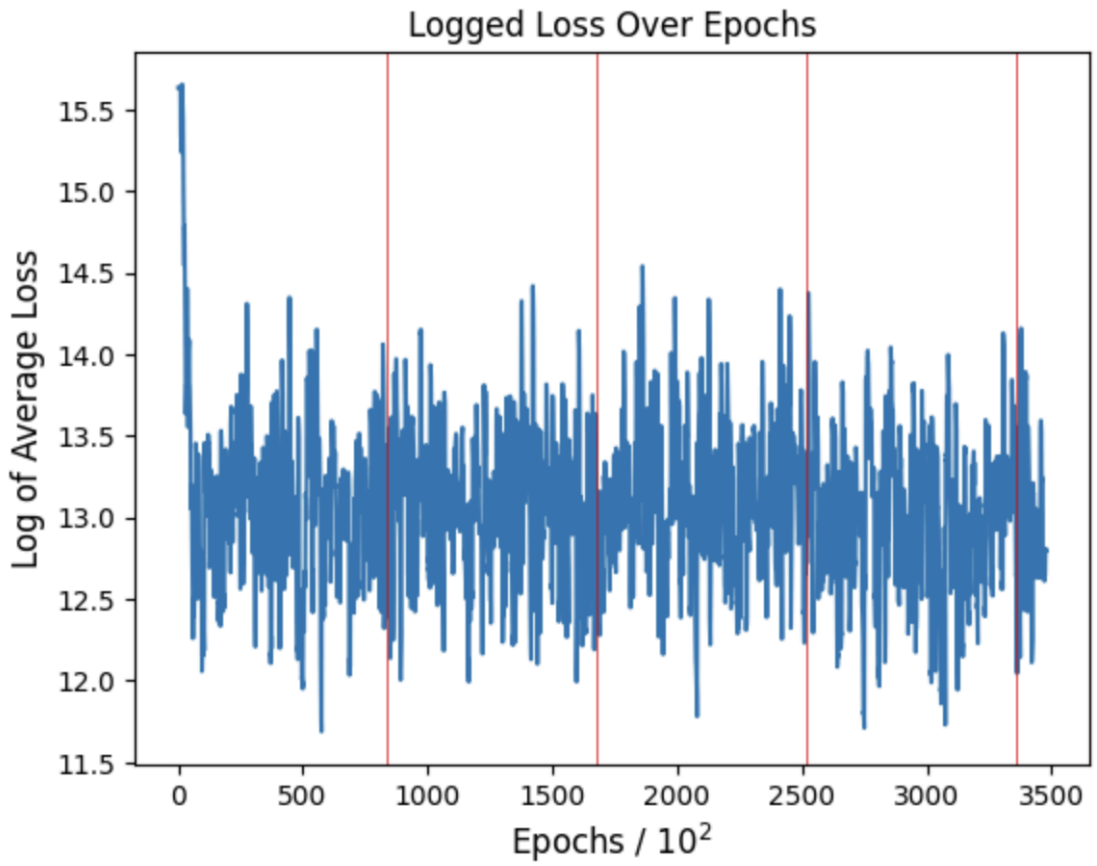
However, as this was crashing before finishing a single cycle (only saw all of the data 5 times once) we could not interpret much from this.

We were initially training with a batch size of 32, so we reduced this down to 2 to try and avoid the memory issues, however, this didn't work.

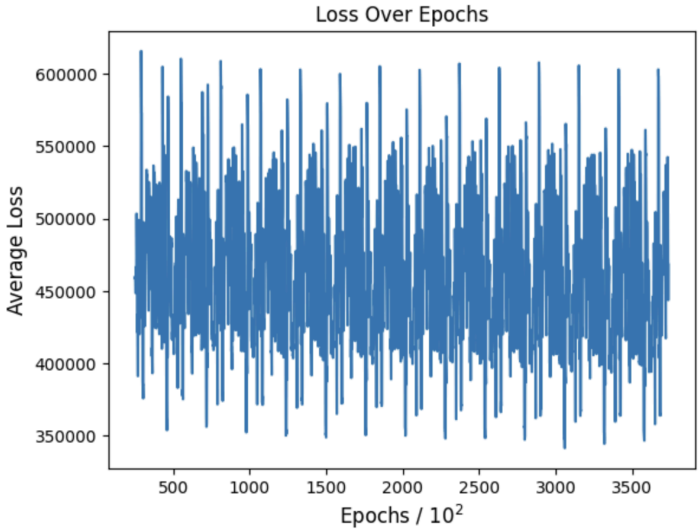
To get round this problem we started clearing unwanted variables, such as the output that the model returned after each iteration. These take up a large amount of memory, but as soon as it is used in the loss function it is no longer needed. As we are doing thousands of iterations, this was taking up a lot of memory.

We then started training again using this new method of clearing the memory. We wanted to guarantee we would get a result so we stuck with a low batch size in the hope to avoid any memory issues. However, this came at the expense of taking a long time to train.. We left it to run for 4 days and got the graphs on the following page. As we had to terminate early, we only managed to train 4 cycles. There is no obvious sign of the model learning so we increased the batch size and ran again with more cycles. This is still training but the training time is much better, this might be done by the time of the meeting so we might be able to show you.

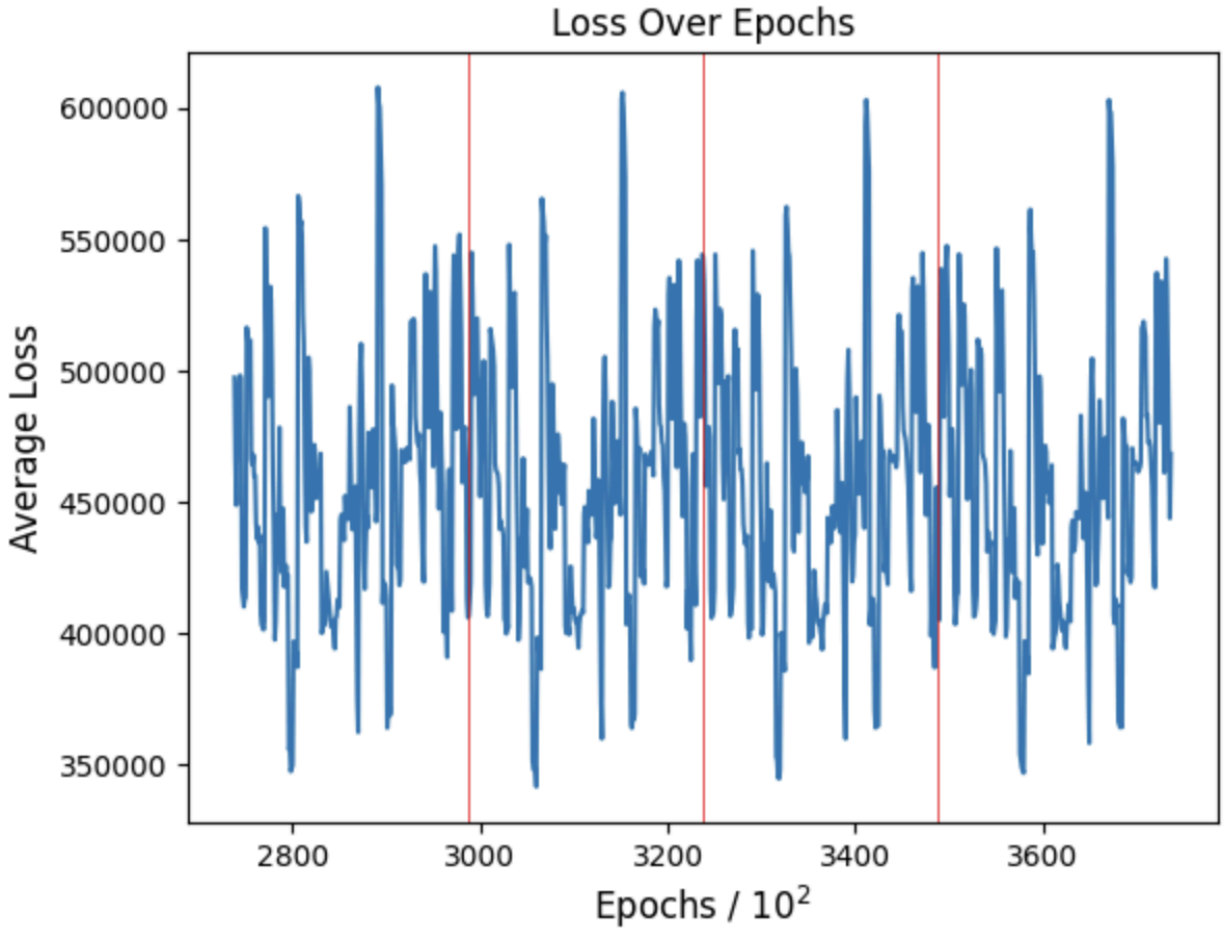




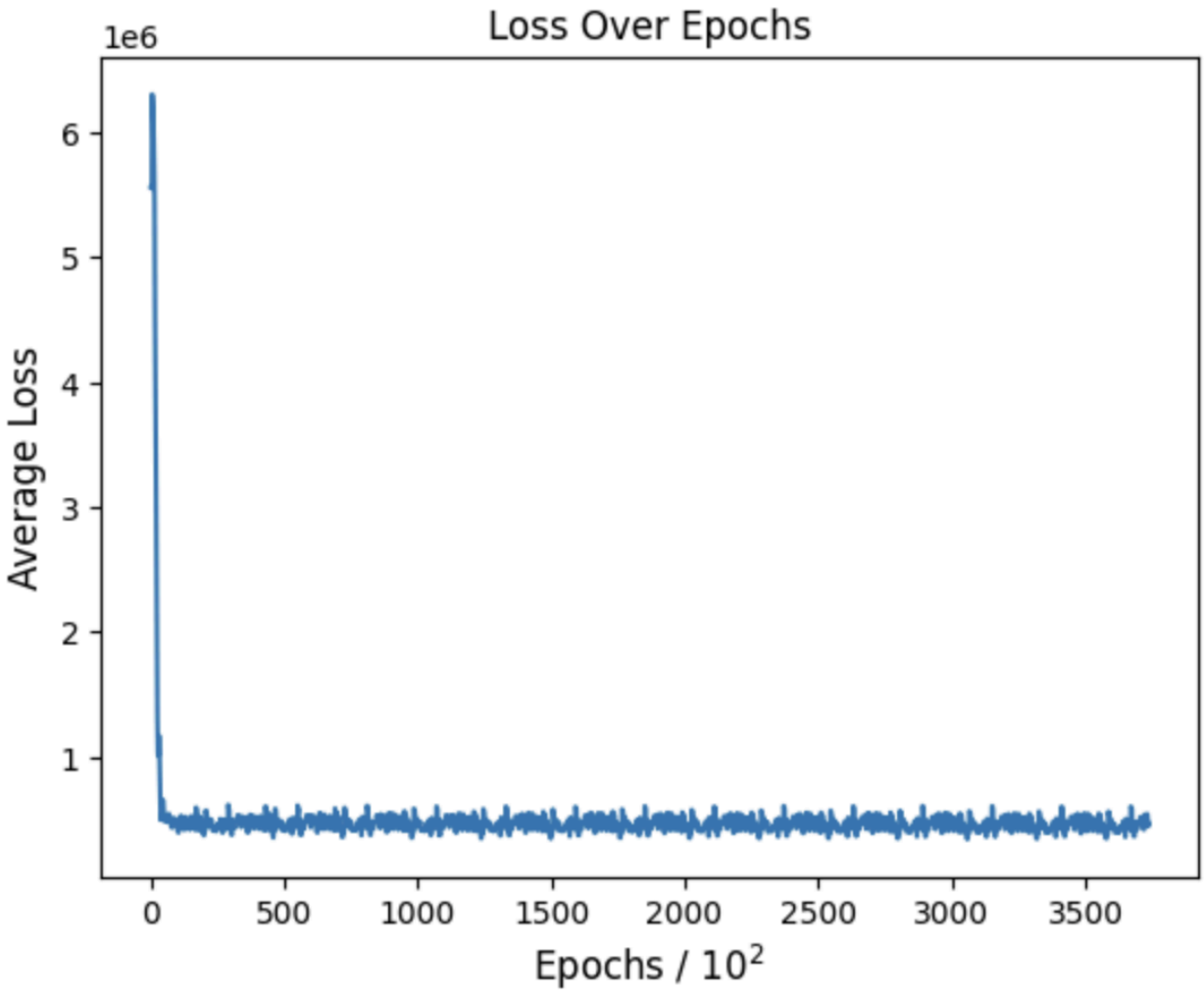

**Batch size 32:**
**First 250 removed (50 batches, 5 losses per batch)**



## Last 4 cycles




Again, no obvious sign of learning.




**Ideas:**

-track hungarian algorithm

-adjust park the bus definition

-reduce no. of players its training on (just back 4 for example- can give it rest of defence as well in input)

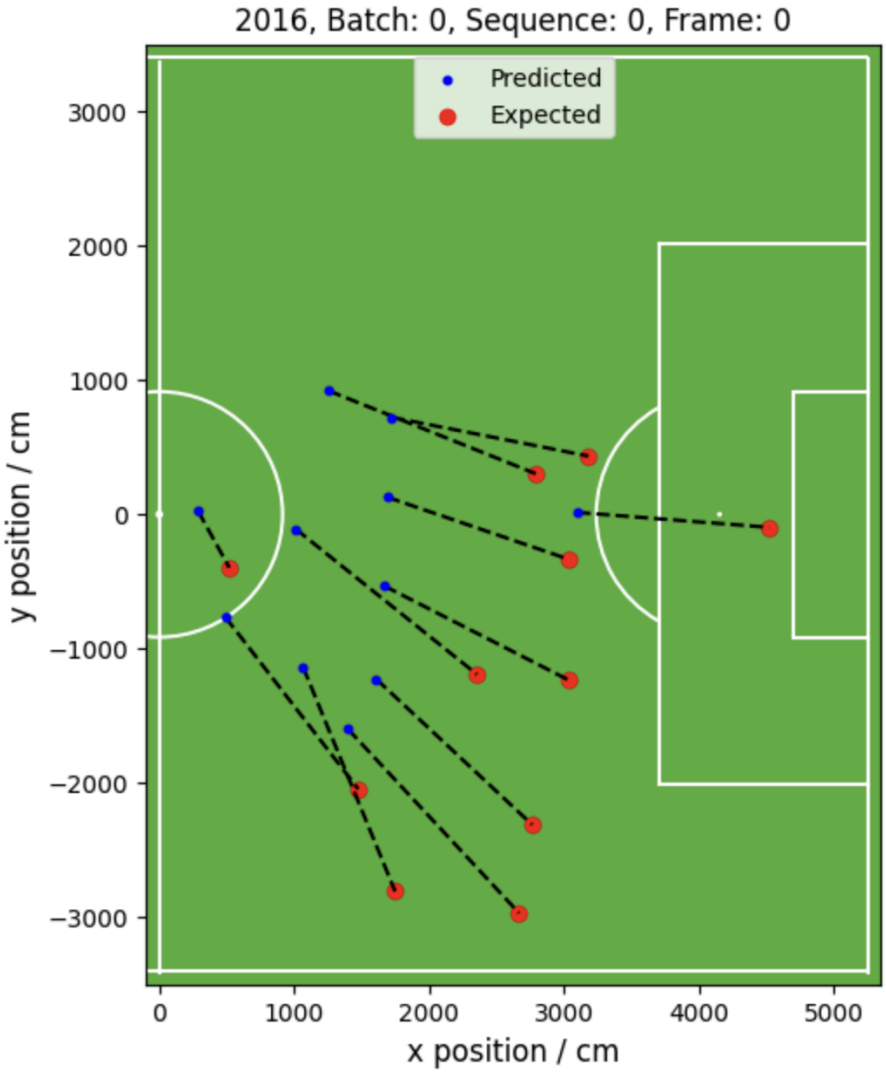
**Week 4**

-After the meeting it was clear that we may need to refine the conditions for the data we are extracting. As we have a data set that is now

# Week 5 - 26/02/24

## Hungarian Algorithm

Fixed the Hungarian algorithm. It now works for any batch size, after inspecting plots the results seem reasonable. Plot of one frame is shown below. From looking at full sequences, it doesn't seem like the connections are changing between frames -> they stay constant for the whole sequence. I'm not sure if this is coincidental, a mistake in the algorithm or an expected result. An example plot is shown below.

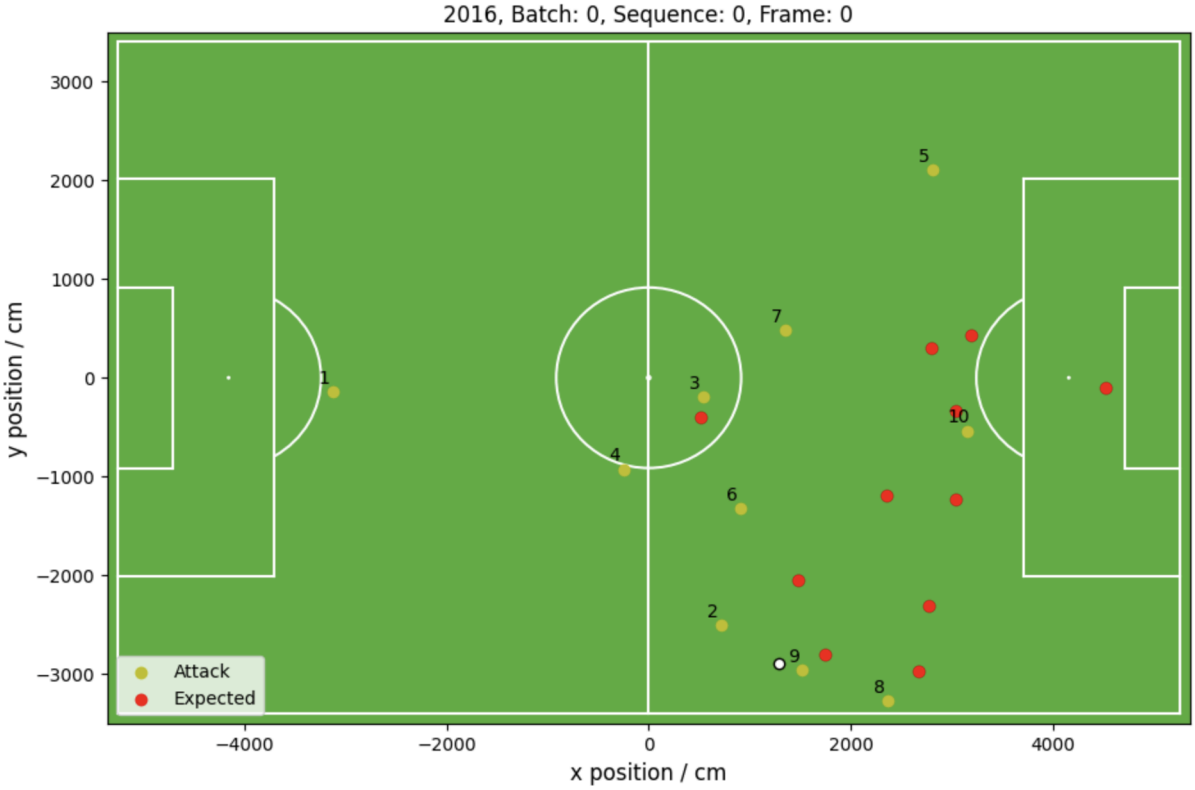


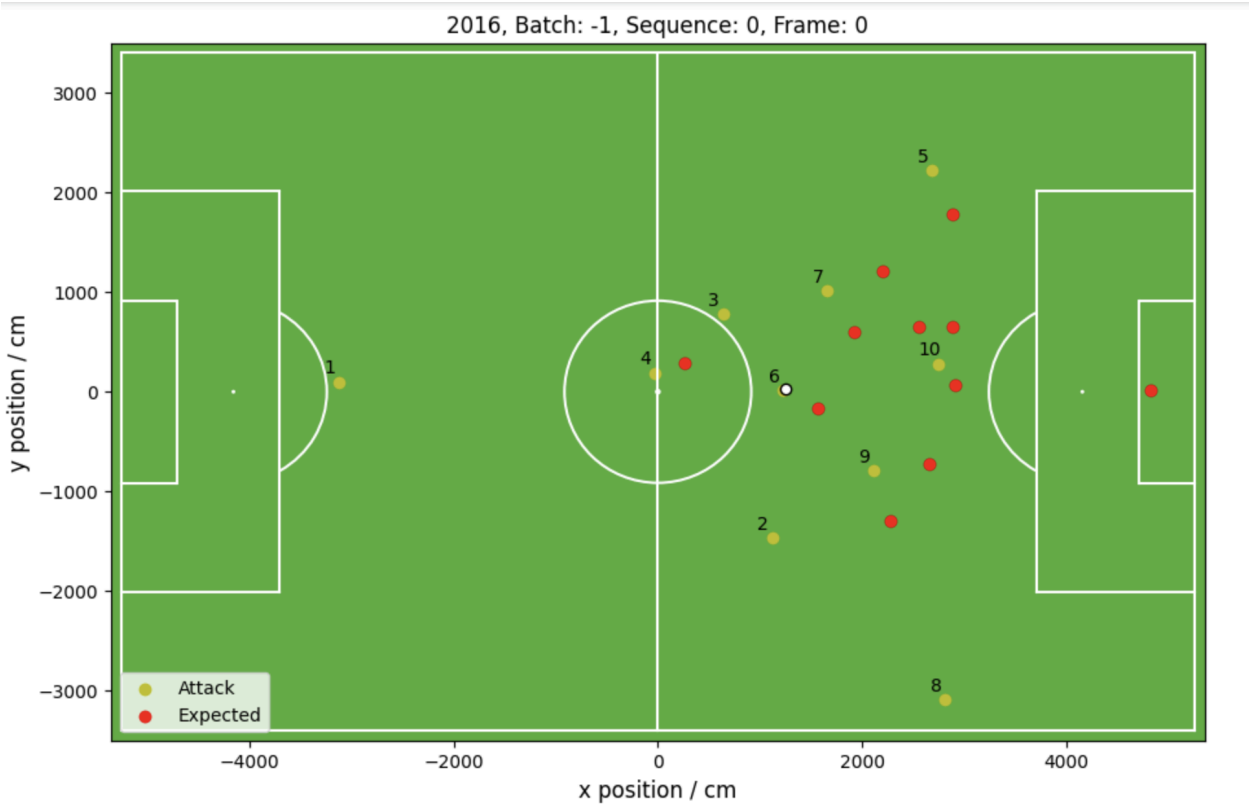

To inspect whether connections change throughout a sequence, we made an animation with the connections. I can't attach videos to this file but we will show them in the meeting.

## Attacking Order

Last week we spoke about whether the order of the attack is constant during a sequence. We checked the code and are fairly certain it should be constant. We are sorting by position (ie [gk, def, mid, att]) and then by shirt number within each category (ie  def: [5,6,7,8]). This is done to every sequence independently (before sorting into sequences ect.) Therefore, as a sequence is a constant phase of play (cannot be any substitutions) the order of the attacking team should be the same within a sequence. As we are dealing with different line ups between sequences, the players will not be in the same order in different sequences, but complications due to this are minimised by sorting by position and shirt number.

To check whether the order of the attacking players is constant we built the plot below, the number next to each attacking player represents their corresponding index in the array.  I've attached the first and last frame in the sequence, although this isn't conclusive, it looks like the players have the same index. We have inspected the sequence frame by frame and the indices are the same throughout. We also made an animation to make this easier to see.






**Current Data**

-Code has been finalised to define park the bus but is subject to change if further decisions are made regarding what we define as park the bus. Any changes will take no time at all in this respect. Currently each sequence is defined to begin with the ball in front of all 11 defenders and have the defenders within the middle 80% of the pitch. This gets rid of quite a lot of data though so more inspection needs to be done to confirm that this is a satisfactory parameter.

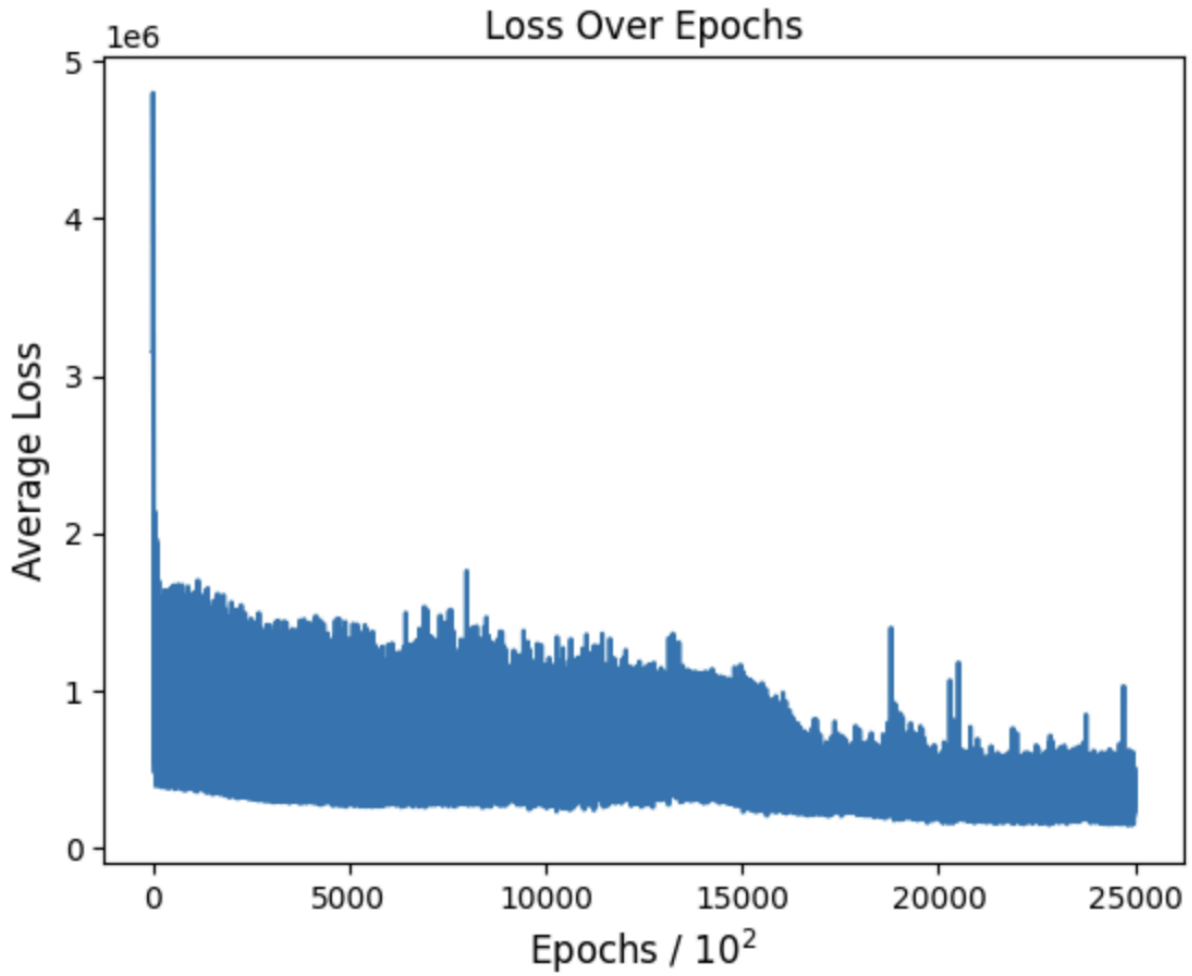
-We have code to convert the current dataset into sparse matrices meaning that we can now move onto finalising the CRNN model alongside the Enhanced LSTM.

-After our error in the last extraction process we now have data for all burnley and newcastle seasons which equate  to around ⅓ of all data. Training is ready to be done on these sequences while extraction continues in the background for the other teams. Just a matter of time to get this done.

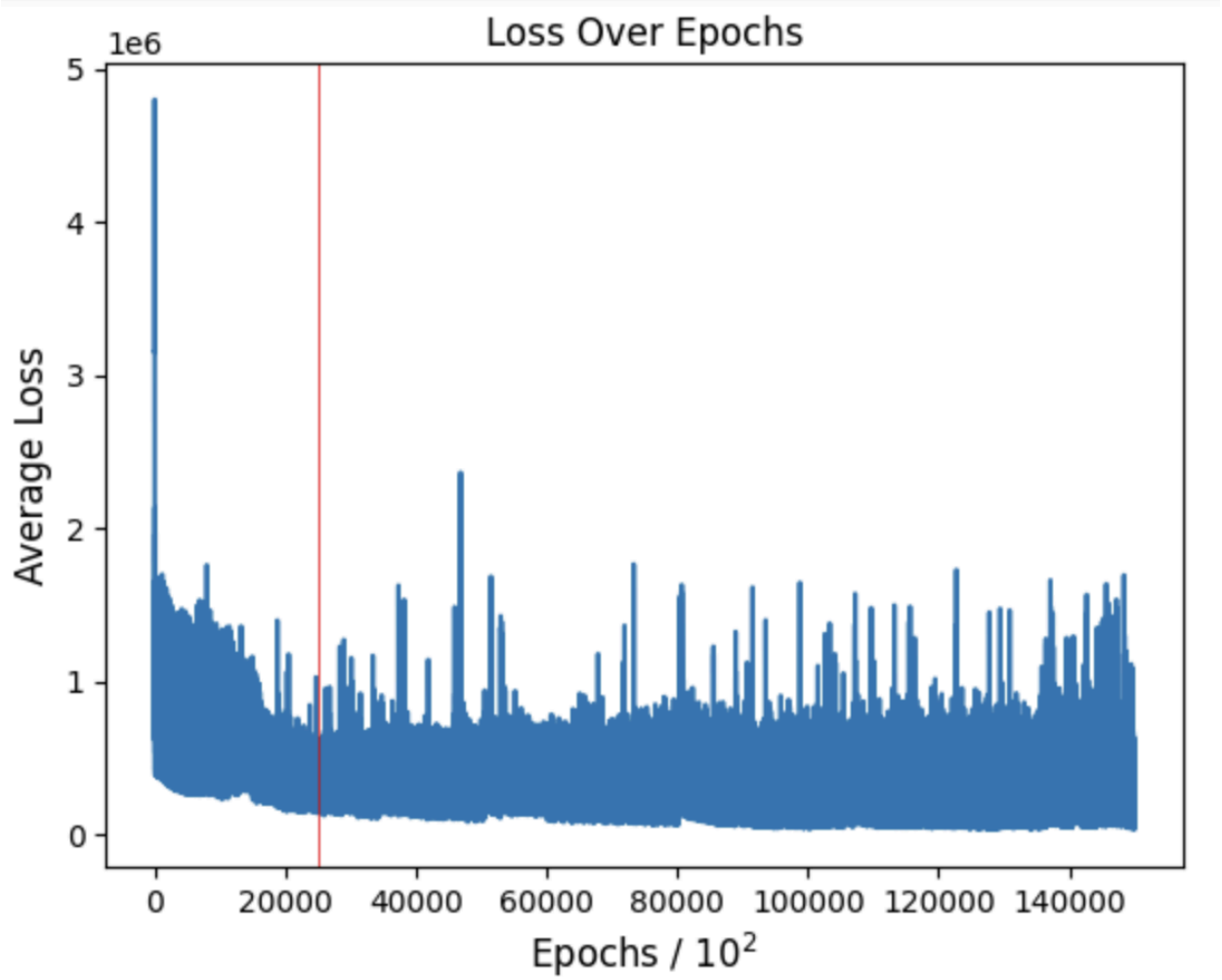
Ensemble Learning???

# Week 6 - 26/02/24

We began the week by extracting 10 sequences from [redacted] 2017 season that looked similar by eye just from the defensive point of view. We then trained these sequences on 5 epochs 1000 cycles and tracked the loss:

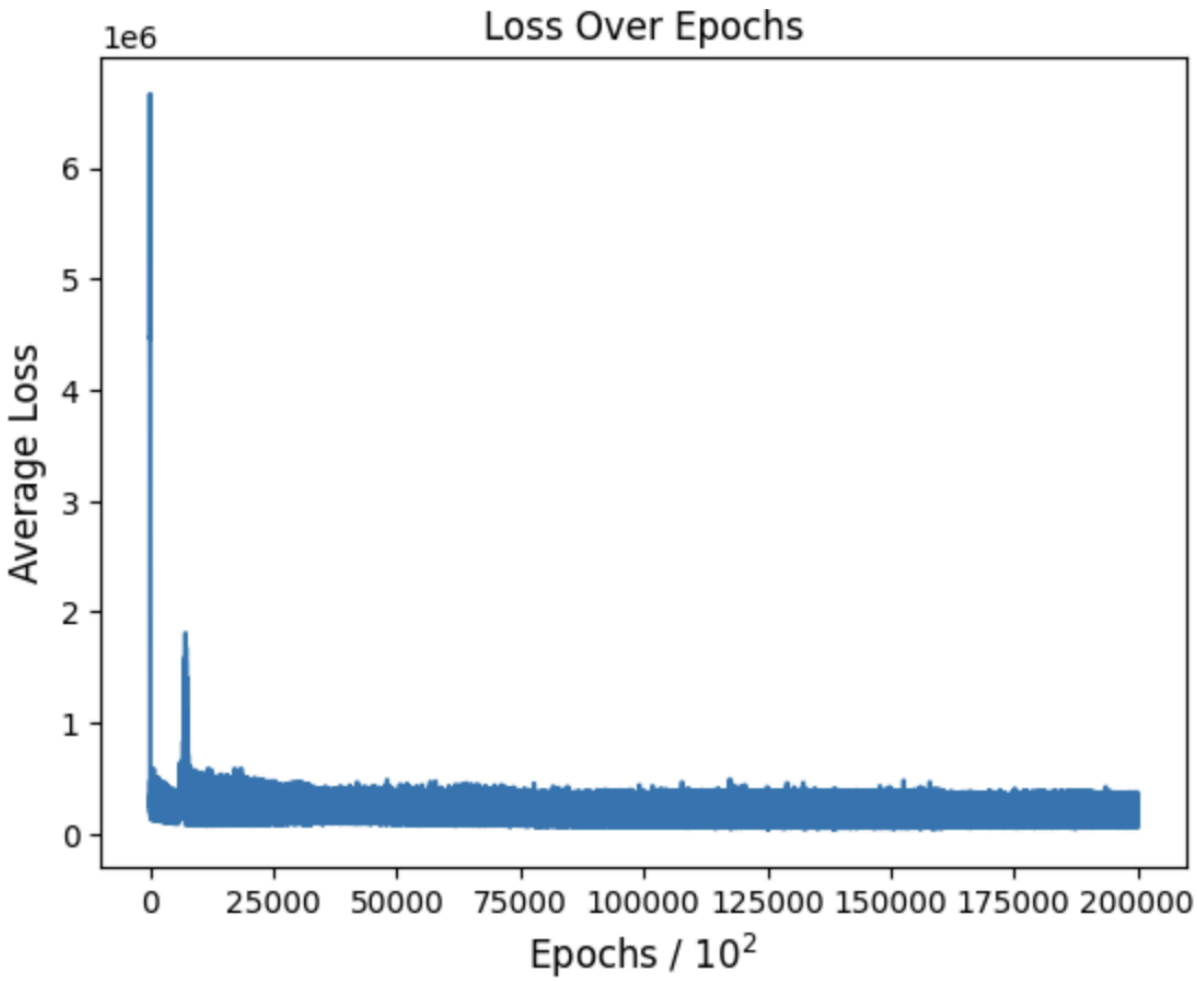



There seemed to be a downward trend so we trained for another 2000 cycles:




This then became noisy with no downward trend (if anything an upward trend)

So we then tried the same method and took 10 sequences where the back 5 defenders (4 defenders + gk) and the front 3 attackers were similar by eye.
After training for ~ 10,000 cycles (5 hours):




No obvious downward trend here. [file path: 10000 cycles reduced data].

By doing this we have simplified the problem from predicting 22 coordinates from 24 coordinates to predicting 10 coordinates from 6 coordinates. However, this still doesn't work. Perhaps we need more sequences as we only used 10.

The rest of the week was spent refining the code to build sparse matrices and the CRNN to see if a different style of model will learn any better. The idea is that this will offer some comparison between models in the report. We will first try exactly what we tried above (5 defenders and 3 attackers) to see if a CRNN can learn any better than the models we have tried already.

Notes:
Talk about rounding data in the meeting.

## Week 6 - 26/02/24

Main focus of the week was to introduce velocities into the model prediction. Note we are not giving the model the velocities of the attack, we are just asking the model to predict the velocities of the defenders as well as the positions.

The expected distribution is of the form [positions, velocities] ie
[x_1, y_1, x_2, y_2, …. , v_x1, v_y1, v_x2, v_y2]
The predictions for the velocities are compared to the expected values in the same way as the positions -> MSE loss. The total loss is then the sum of the positional loss and the velocity loss: MSE_pos + MSE_vel. The hungarian algorithm is still being used for the velocities so now there are 4 numbers tied into each element of the cost matrix.

## Week 7- 21/03/24

The goal for this week was to begin training the model on different inputs over the same training cycles and compare learning. The different inputs we want to try are:

- ALL PLAYERS X/Y POSITIONS
- ALL ATTACKERS POSITIONS IN INPUT - LOSS FUNCTION/OUTPUT ONLY CONTAINING THE BACK 4 AND GOALKEEPER
- ALL PLAYERS X/Y POSITIONS & VELOCITIES
- ALL ATTACKERS POSITIONS AND VELOCITIES IN INPUT - LOSS FUNCTION/OUTPUT ONLY CONTAINING THE BACK 4 AND GOALKEEPER VELOCITIES INCLUDED
- ALL ATTACKERS POSITIONS IN INPUT - LOSS FUNCTION/OUTPUT ONLY CONTAINING THE BACK 4 AND GOALKEEPER VELOCITIES INCLUDED

- SPARSE MATRIX OF PLAYER LOCI WITH ALL PLAYERS INCLUDED
- SPARSE MATRIX WITH ALL ATTACKERS POSITIONS IN INPUT - LOSS FUNCTION/OUTPUT ONLY CONTAINING THE BACK 4 AND GOALKEEPER
- POTENTIALLY SPARSE MATRIX INPUT WITH MORE CHANNELS THAT CONTAIN VELOCITIES

Tweak x threshold for ‘corners’ and number of players. DONE
Try all average positions in a frame fall within a std of the average. Can change this to 2 standard deviations.. DONE

Do distance between defence and midfield, and defence and goal line. DONE

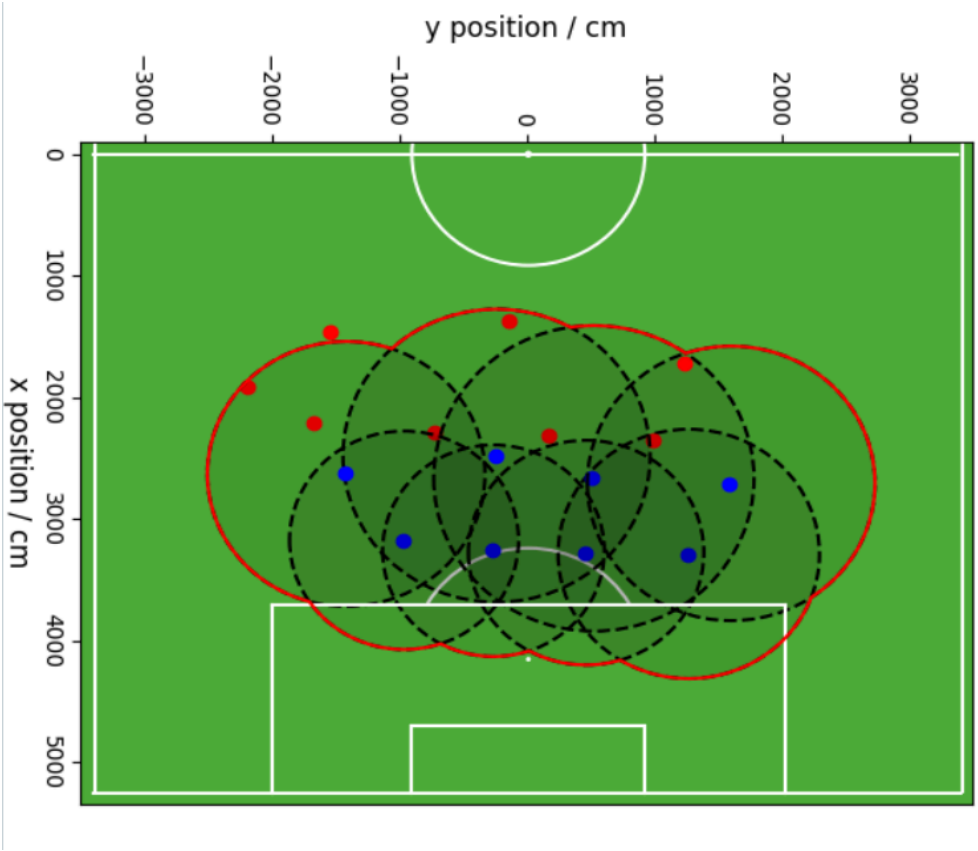
As of writing this we need to choose adequate sequences that are similar enough to increase likelihood of the model learning well. Issues in data extraction mean that we now have many more sequences at our disposal so need to find new sequences so we can identify the sequences consistently between input type datasets.

## Easter

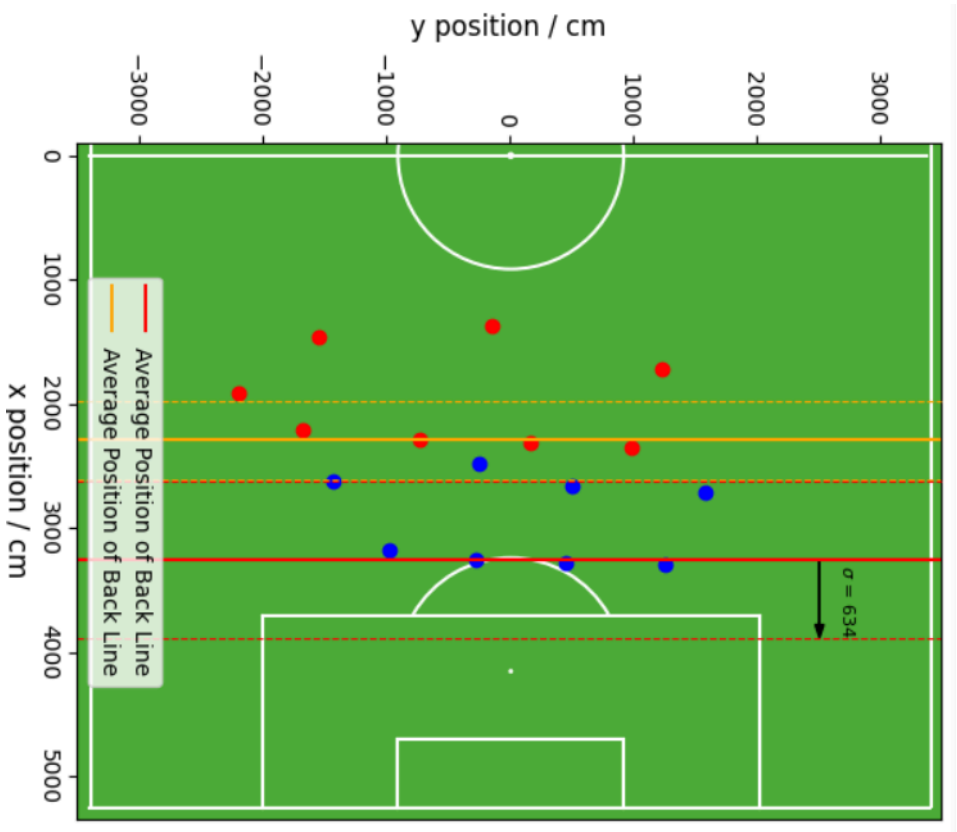
-Before training we felt that it was necessary to ensure that our datasets were final and accurately reflected the type of football that we wanted the model to be able to replicate. In a sense we wanted to define 'park-the-bus' more accurately and deliberately. We had discovered in our research for our january interview that in 2016 [redacted] had beaten [redacted] (according to premierleague.com). We therefore decided to analyse the defensive tactics of this fixture.

All frames of the game were analysed and x,y coordinates of the frames in the game were extracted in which the entire [redacted] team was sat in their own half while [redacted] had the ball. Corners and deep set pieces were handled by eliminating sequences in which [redacted] had more than 6 [redacted] players in their own box. This is not perfect but upon inspection through our animation little to no set pieces were seen.

Once we had these frames we calculated the average position of each player and their standard deviation. We then plotted this on a graph with the standard deviations in x and y being standardised by adding the x and y in quadrature and plotting a circle with this radius around the avg position for each player. This created a shape in which we will define that the average position for each player in a desired sequence will fall in for the sequence to be kept.




We also calculated the distance between the average position of the back line and the midfield line:

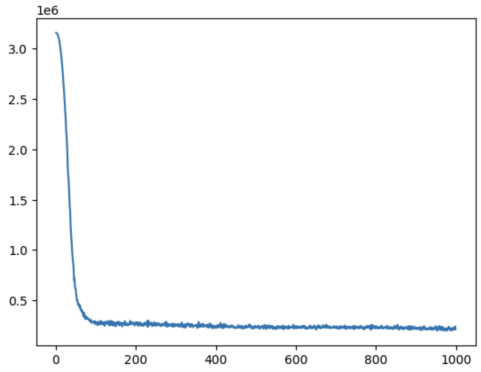



We want to filter for sequences in which the back line and midfield line are not further than this distance plus its standard deviation (while also all conditions stated previously are satisfied). With all this together we believe this is an adequate definition of ‘park the bus’.

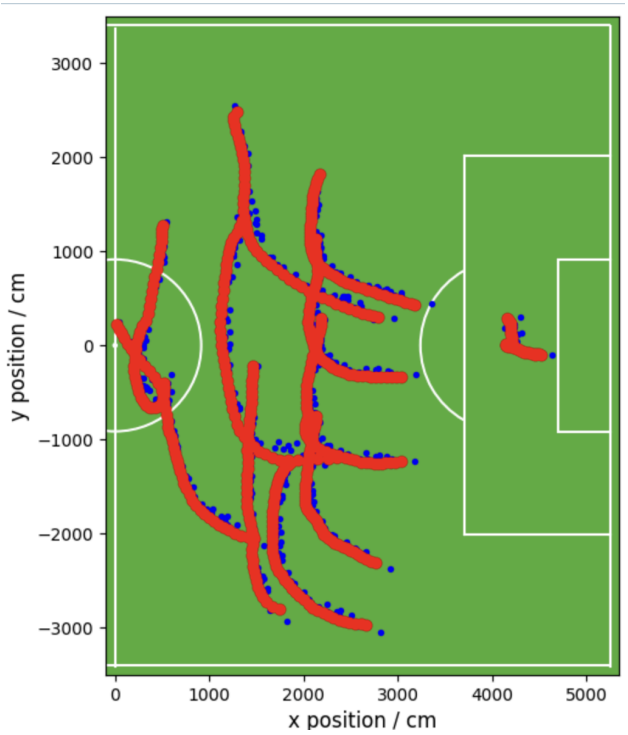
## **Week 8**

Going back and testing the model:

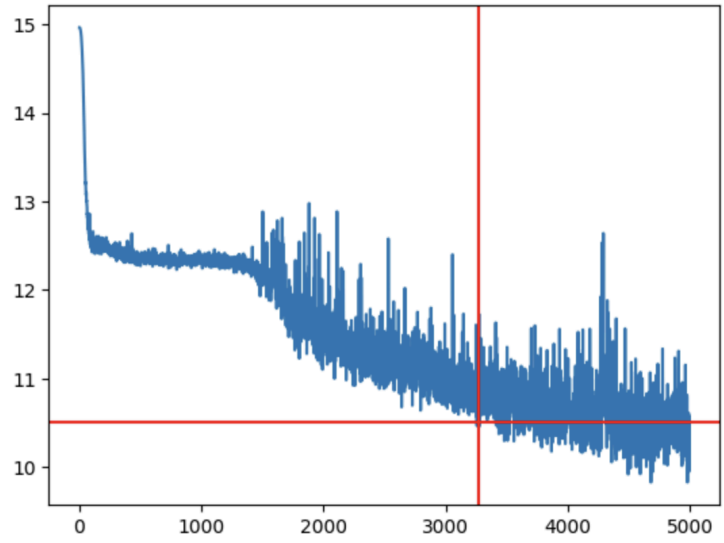
Tried training on one sequence, reached a loss of ~20,000 -> Model can learn!



Showing the predicted output compared to the expected output for the sequence:




And plotting the log loss for a clearer analysis:

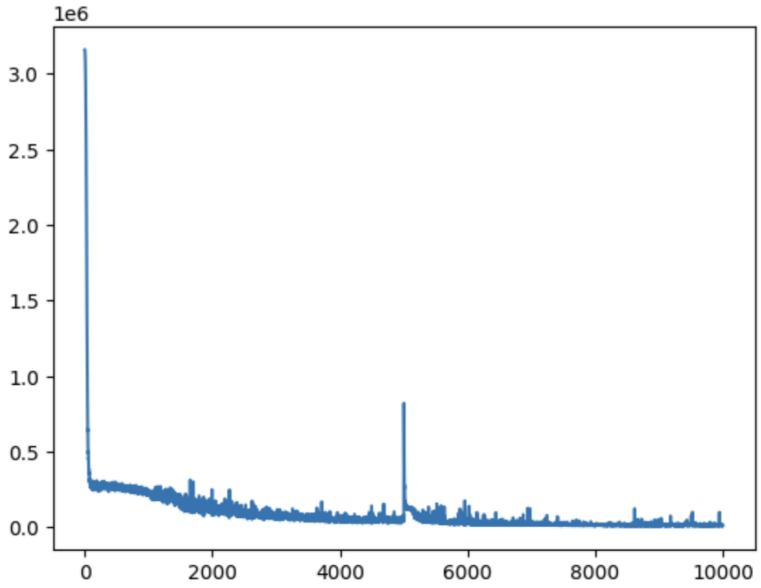



But took ~3250 epochs to reach this point, more like 3500 to reach this consistently

Show video of model training

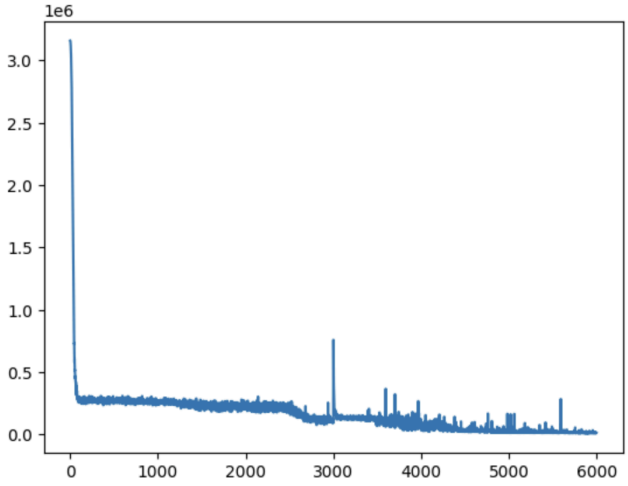
Tried training on 2 sequences, 5000 epochs each, spikes up after first sequence. After training on both, it has no memory of the first.

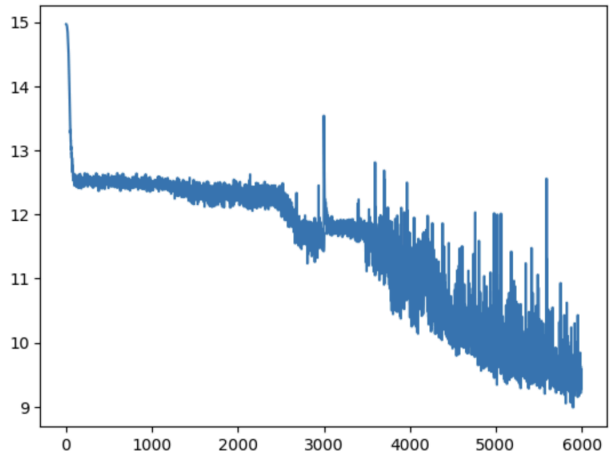
This was tested by simply training on 5000 epochs on each sequence sequentially. The model was then retested on sequence 1, the loss reached ~10,000 during the training process but spiked up to 800,000 when retested on model 1.




Perhaps there is a sweet spot of epochs that gives a good loss (ie what is shown from 1 seq) but the model still can retain memory.

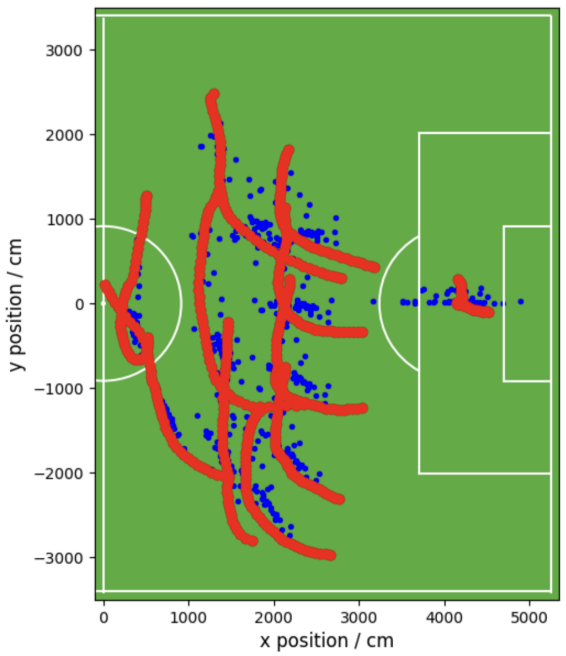
So trying exactly the same as above but with 3000 epochs (this is what one sequence needed to get a ‘good loss)

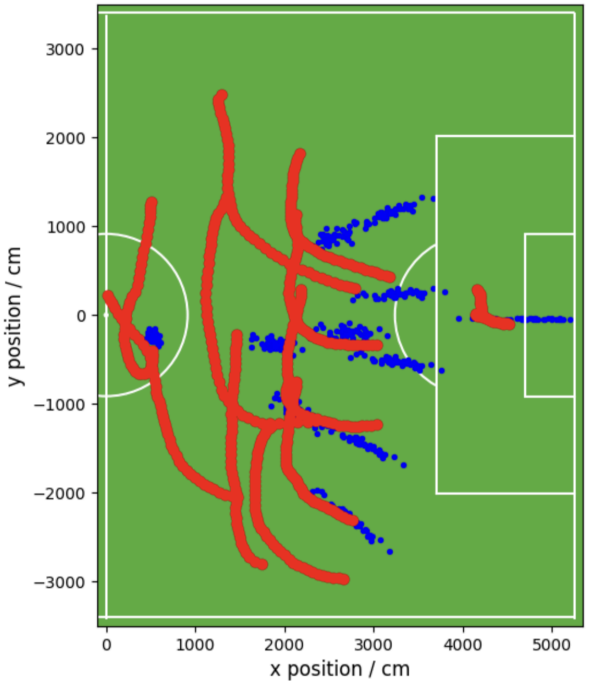




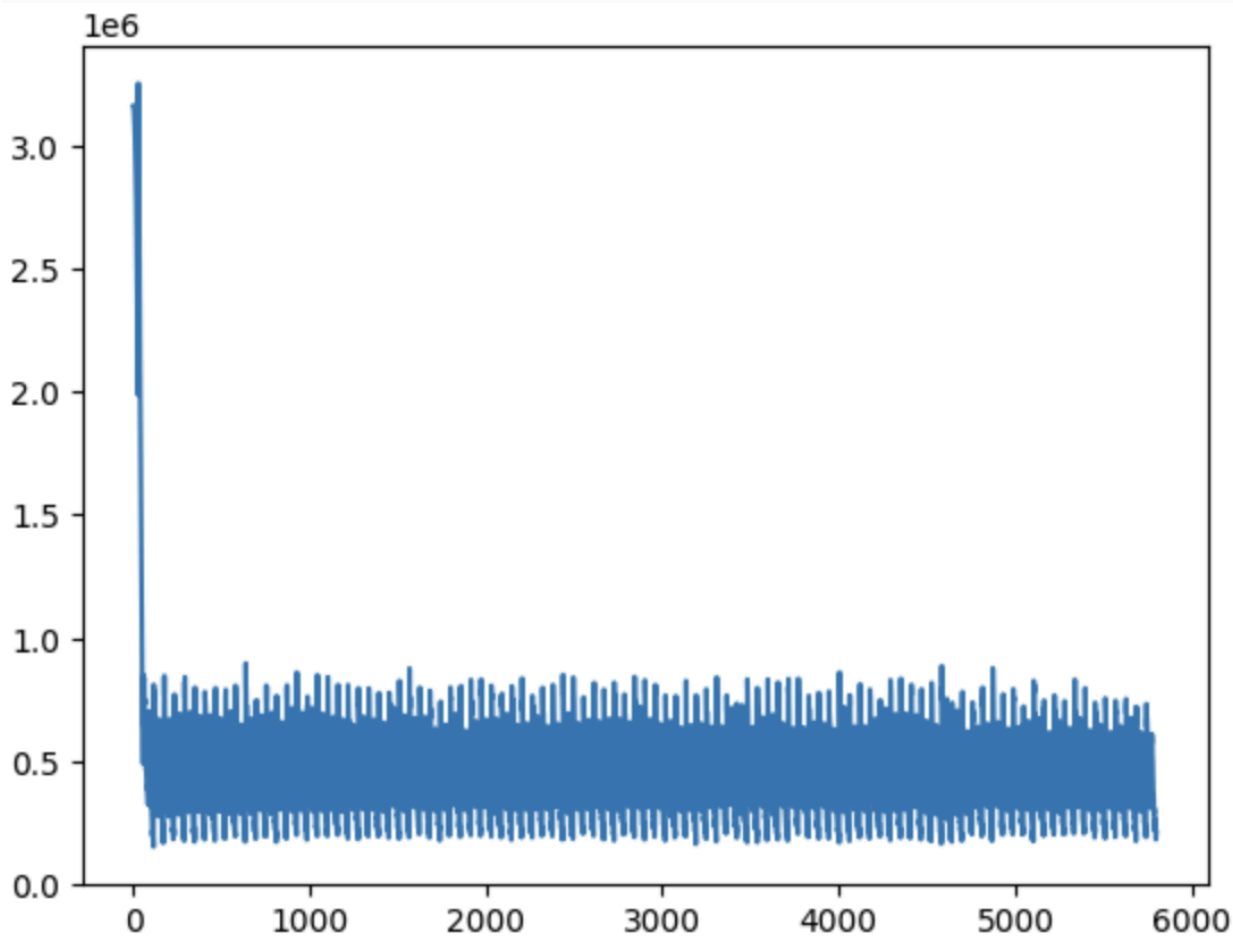

Model still reached a loss of ~10,000 which is way better than from 1 sequence training. Note that the loss decreases after both sequences (ie almost analogous to training on 6000 epochs), small spike after first sequence.

When retraining on sequence 1, loss as 1,000,000. Ie it has become a lot worse than when training for 5000 epochs. This is because it still has no memory AND it didnt learn sequence 1 well enough in the first place.



So decreasing the epochs more will not help. Now we try using cycles, so we still aim for 3000 epochs on both sequences but we break it up into smaller chunks. Ie 100 cycles of seeing each sequence 30 times.

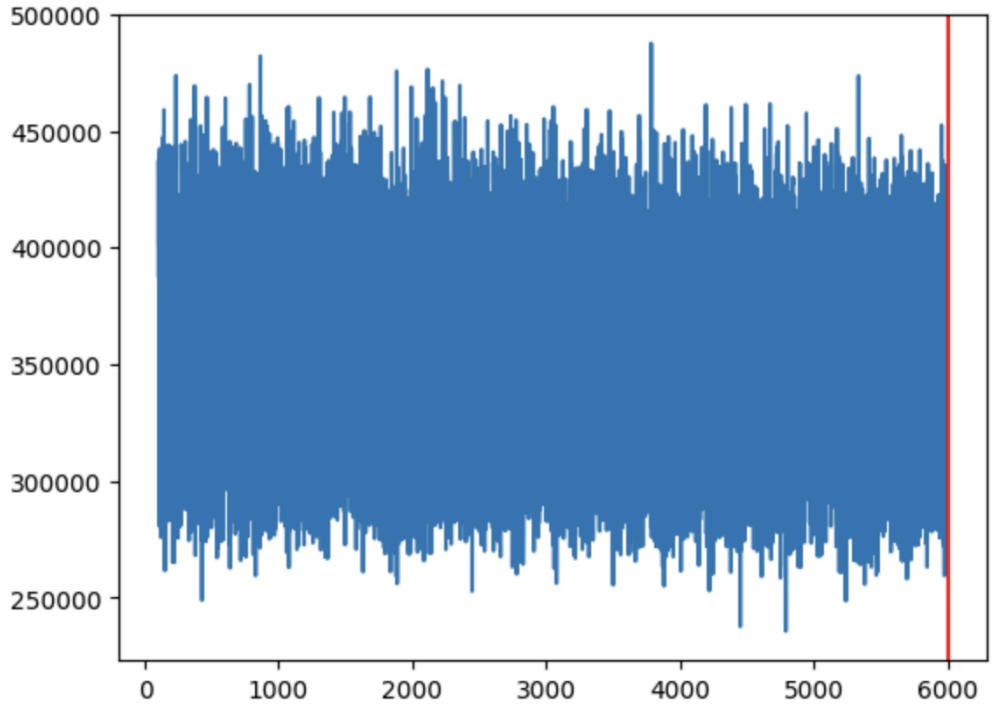



Model still retained no memory of sequence 1 after seeing sequence 2. Training hit a minimum loss of ~150,000 and was at ~200,000 at the last training epoch on sequence 2.

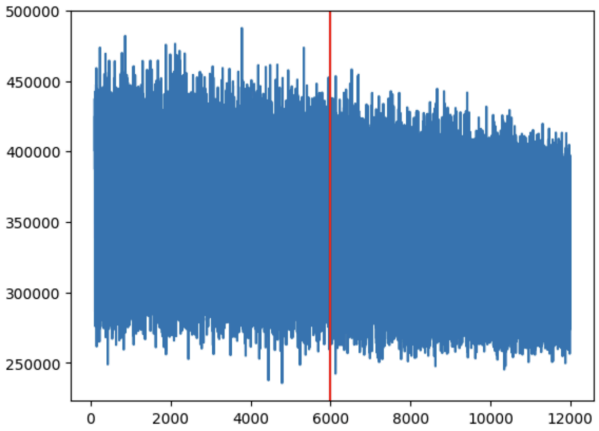
Then when retesting on sequence 1, gave a loss of ~700,000, ie no memory after 30 epochs of seeing another sequence.

Also, note when training without cycles, the model seemed to keep a downward trend between sequences, this no longer seems to be the case.

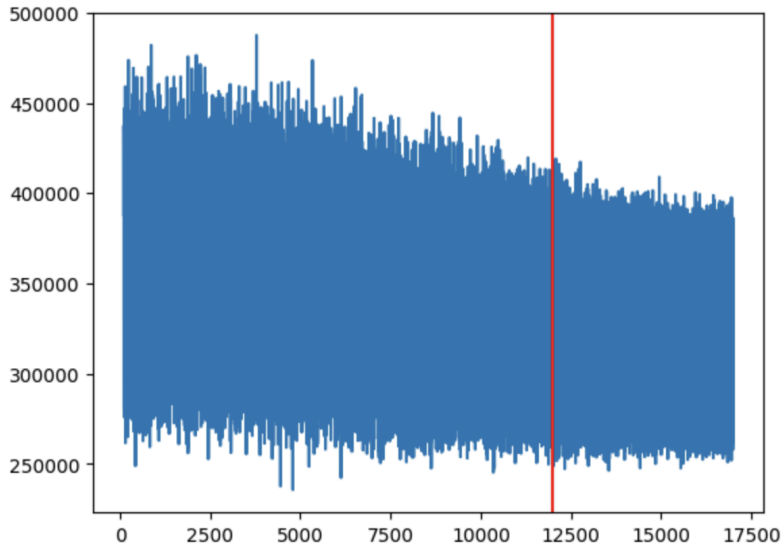
Repeating with a (600:5) (epoch: cycle) cycle to epoch ratio to see if the model retains any memory between sequences:



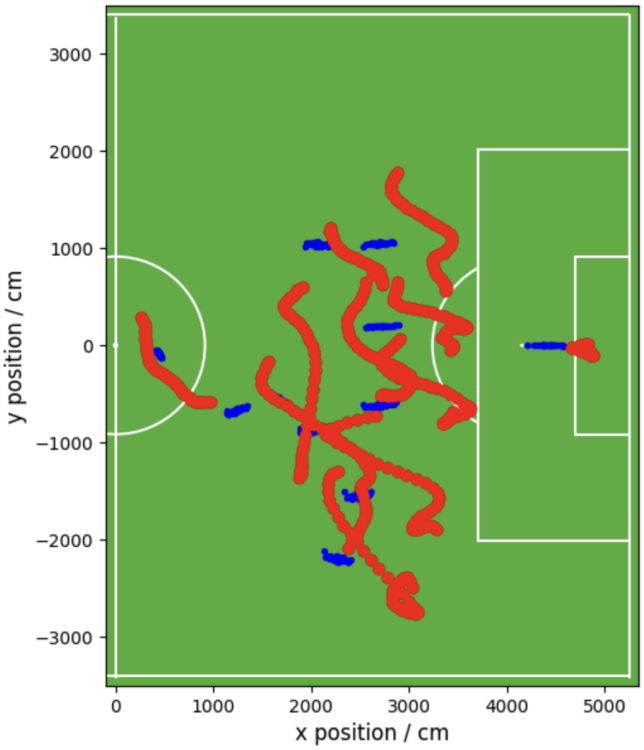

Training (600:5) ommittting first  100 loss entries (10 cycles)




Trained (1200, 5) still indication of a downward trend so going to keep training to see if we can reach a better loss

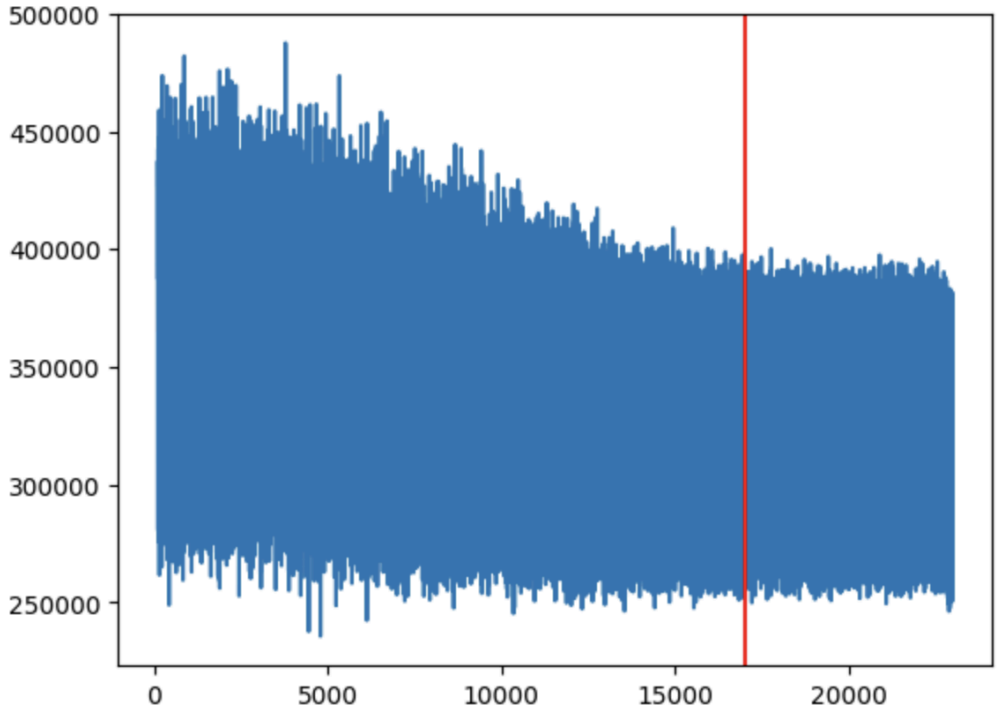


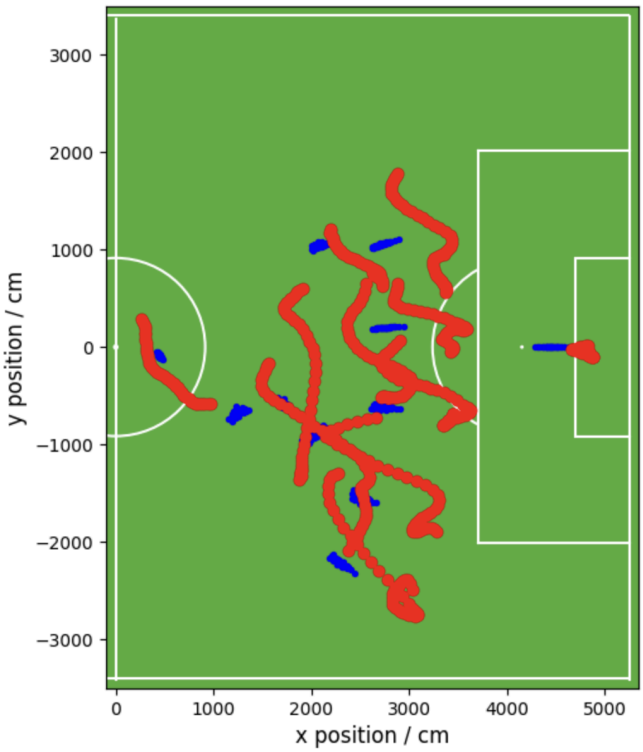

After (17000, 5) Still getting a downward trend but definitely slowing down. Will train another 600 if no downward trend will stop.
Seems to now be plateauing, but to be safe will train to 30,000 epochs to see if there is suddenly an unexpected trend.
If there is not, will try increasing a 10 epoch cycle for a similar amount of cycles.



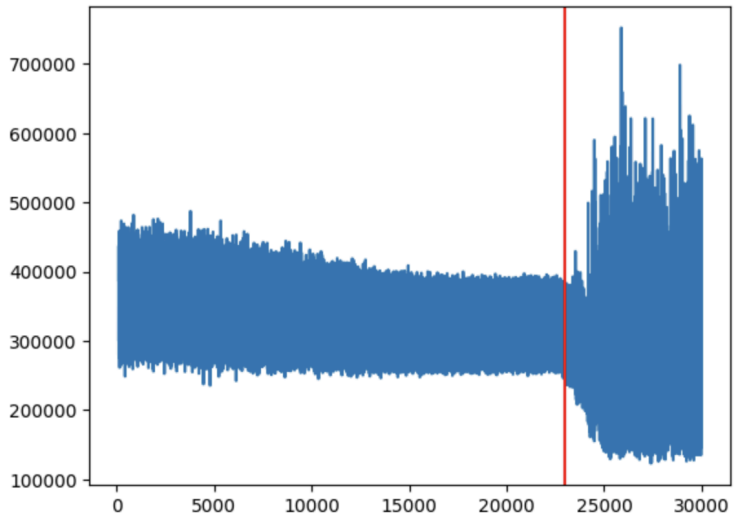

Note that this is the last plot for sequence 2. Still producing the straight lines -> hasnt started spreading in the y yet.

Note for self, loss will be saved to `train_loss_path = f"{loss_output_directory}/train_loss_data_iteration_{0}.pt"`

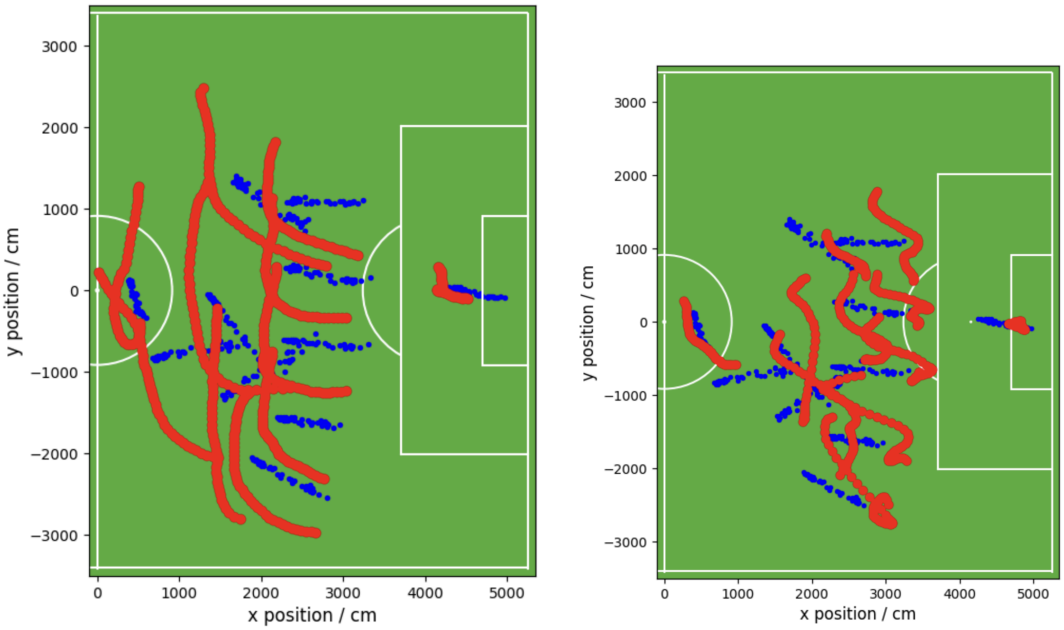



Note that predictions have become shorter in the x, this could be random fluctuations or could indicate a small amount of learning



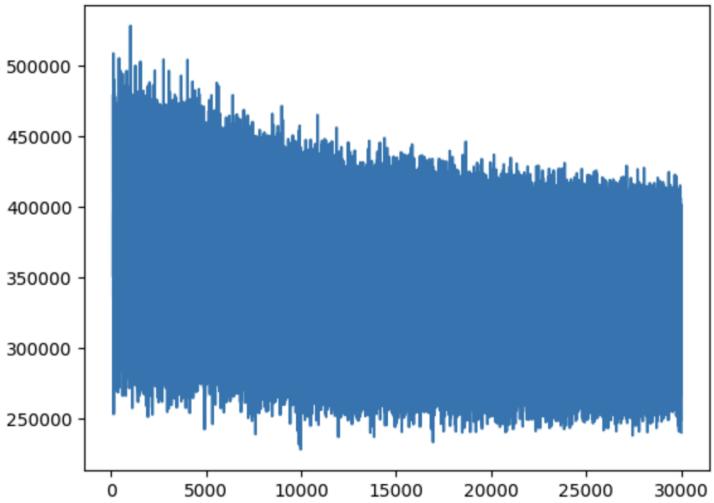

After ~22000 epochs, loss started spiking up a lot. Not sure why this happened but the model wasn't learning enough so this was abandoned.



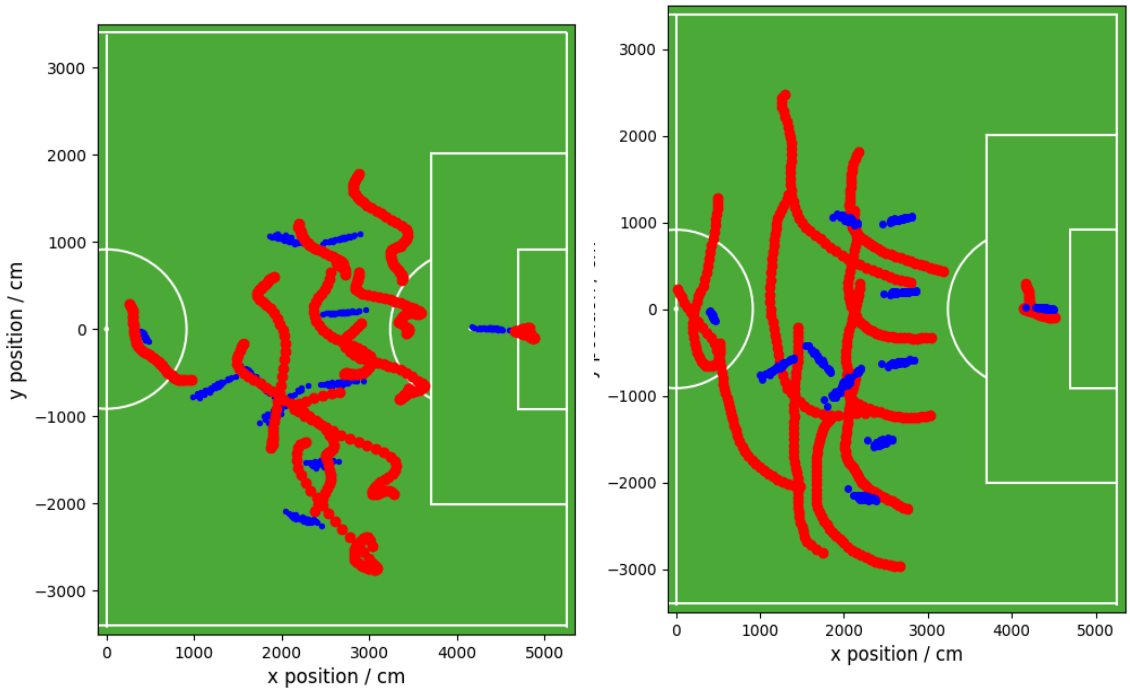

Seems to be learning an average between the two sequences. This would explain why the model is not taking any notice of the input and returning the

same output regardless. Would also explain the plateau in the loss. It learns The average and gets stuck there.

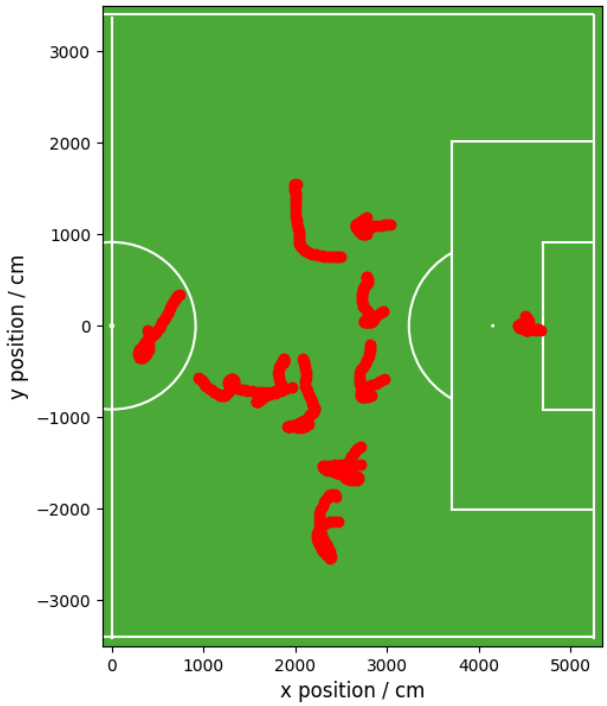
Show learning video 2




Results for training on 30,000 epochs (20,1500), seems to be a downward trend, will continue to train, however, model seems to be learning the average between the 2 sequences we are giving it. The hope is more training will overcome this.




This is the average position of the two sequences, it can be seen that this matches the prediction almost perfectly.  The model is learning, but not the right thing!




## Week 9

Analysing the convergence time of the model on one sequence with different hyperparameters may show an optimal architecture that will allow it to train better when expanding to multiple sequences.

---

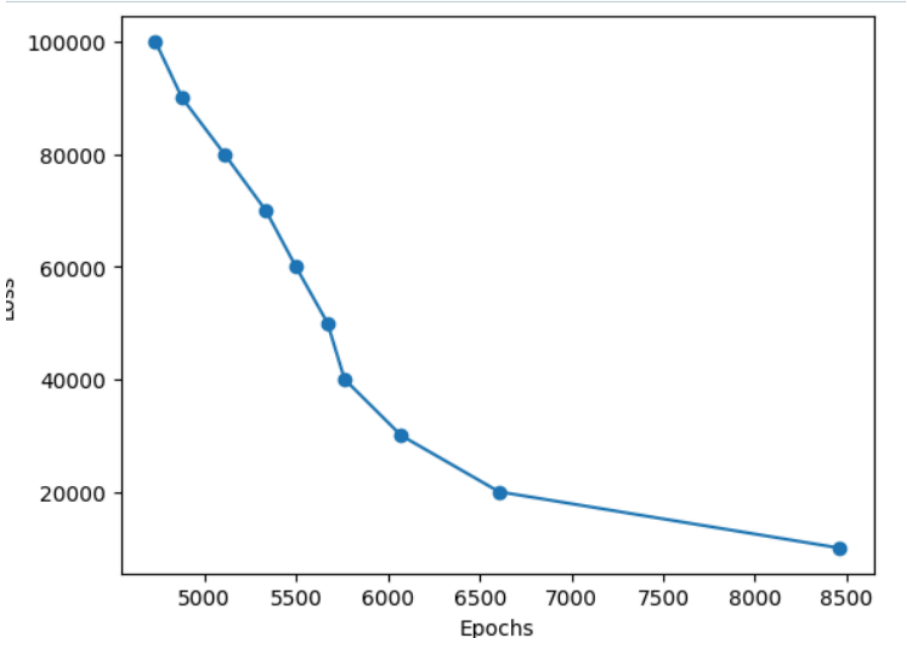
initial hyperparameters that were tested

```
model = EnhancedLSTMModel(input_size=24, hidden_dim=64, output_size=22, n_layers=3,
dropout=0.4).to(device)

n_epochs = 10000
lr=0.01

loss_function = CustomLoss()
optimizer = torch.optim.Adam(params = model.parameters(), lr=lr)
```

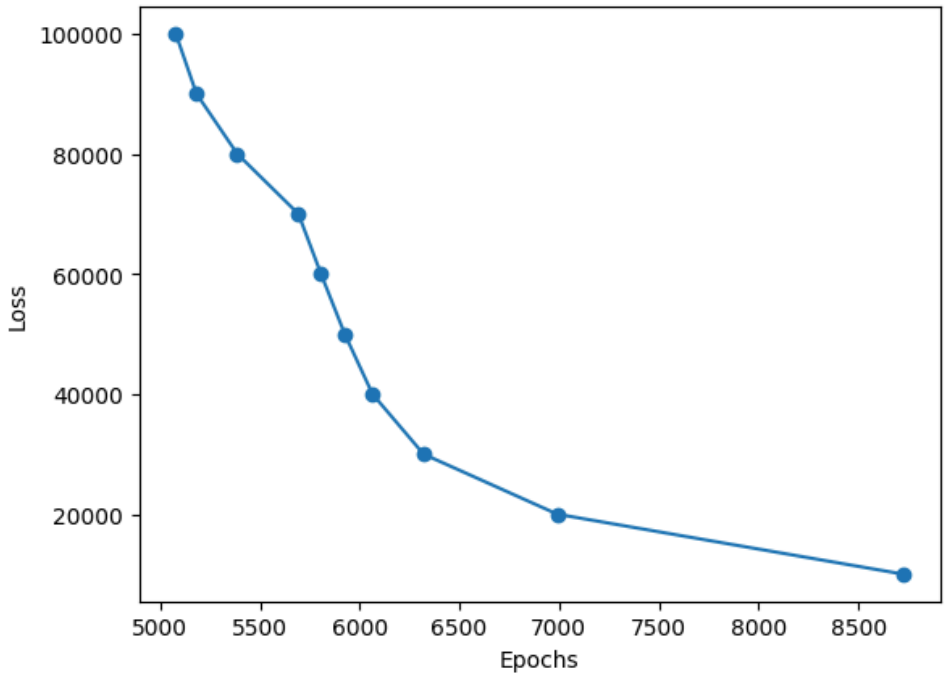
```
[4732.7, 4877.7, 5109.1, 5334.1, 5498.5, 5670.6, 5763.5, 6072.3, 6605.6,
8462.9]
```




---

## Adjust above model to have n_layers = 2:

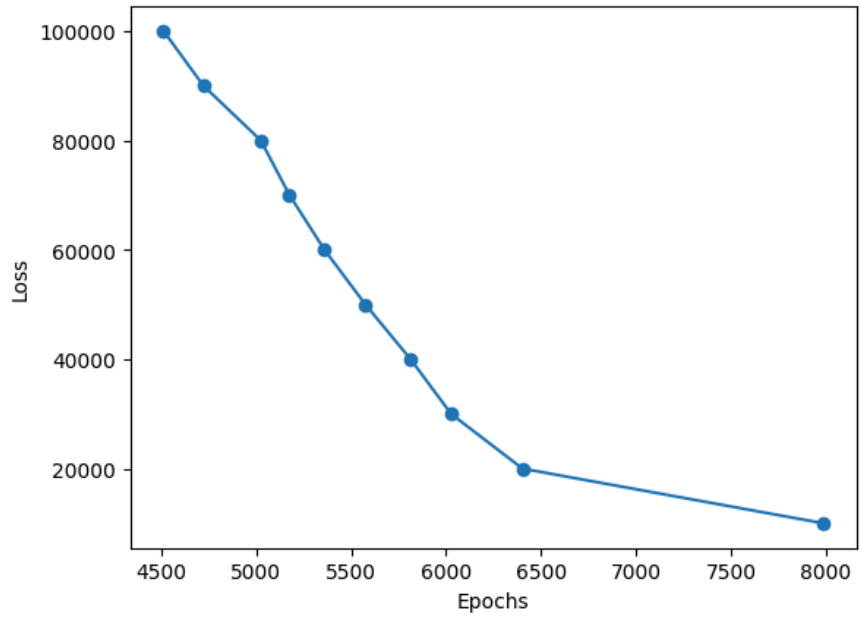
`[5078.6, 5182.4, 5386.6, 5693.3, 5806.6, 5925.5, 6064.7, 6320.6, 6991.3, 8728.4]`




---

## n_layers = 4:

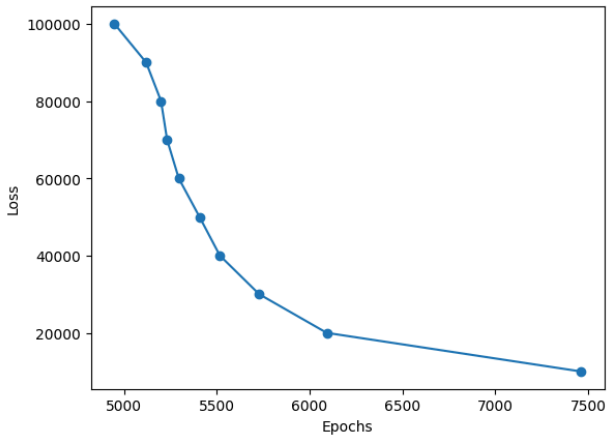
[4512.2, 4722.1, 5024.1, 5175.4, 5359.6, 5575.2, 5813.0, 6026.5, 6402.4, 7988.3]




---

N_layers = 8

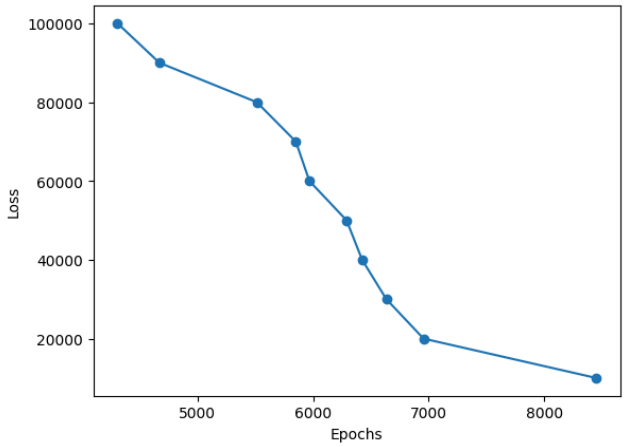
[4949.4, 5120.1, 5200.3, 5232.9, 5295.7, 5407.7, 5517.2, 5729.2, 6093.0, 7463.2]




---

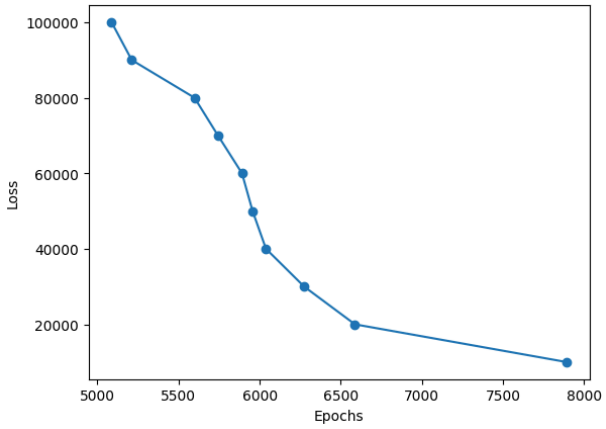
N_layers = 8 but with a second lstm layer added to the model:

[4306.0, 4668.3, 5511.7, 5849.8, 5967.1, 6289.4, 6423.6, 6633.5, 6955.5, 8451.8]



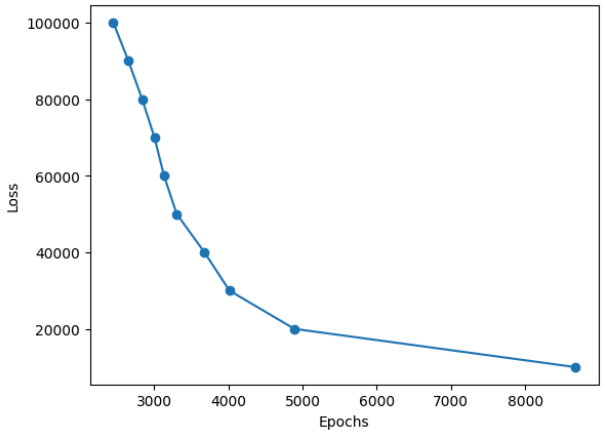

---

N_layers = 10:
[5086.9, 5211.1, 5598.1, 5742.9, 5890.8, 5956.1, 6038.6, 6273.9, 6587.3, 7893.6]




---

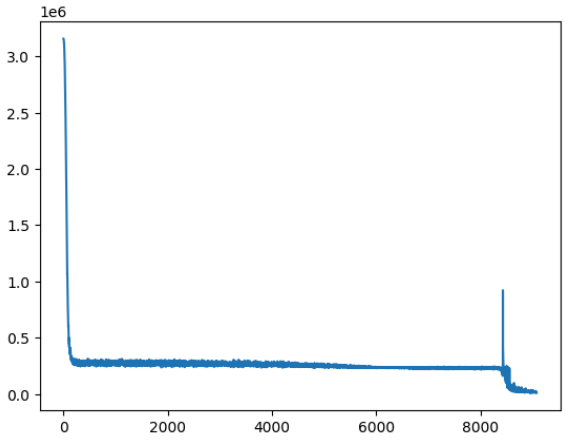
N_layers = 8, lr=0.02:
[2450.2, 2653.5, 2838.8, 3004.6, 3133.3, 3304.9, 3678.5, 4014.7, 4890.1, 8680.7]



*higher learning rates (to a point) tend to result in quicker early threshold numbers but struggle to push below 20,000 and 10,000 in comparison - will keep investigating

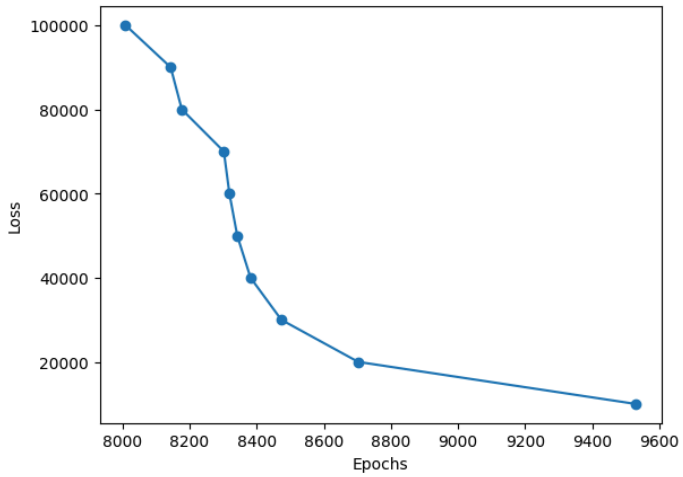
---

n_layers=8, lr=0.005:




interesting behaviour

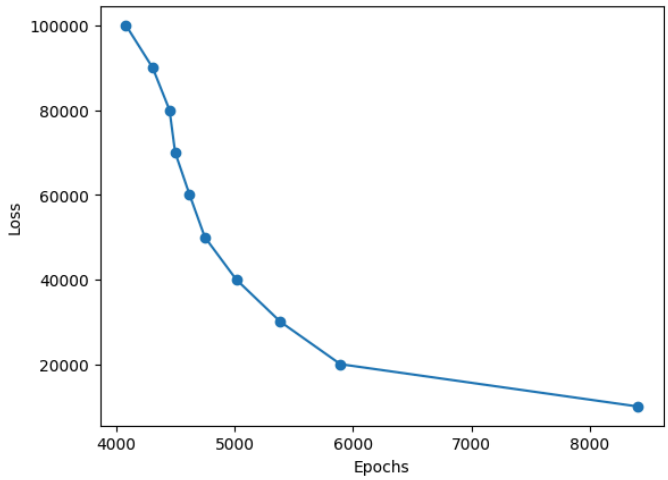
[8009.3, 8143.7, 8176.9, 8302.2, 8318.7, 8342.1, 8381.7, 8473.5, 8702.6, 9530.3]




---

n_layers=8, lr=0.015:

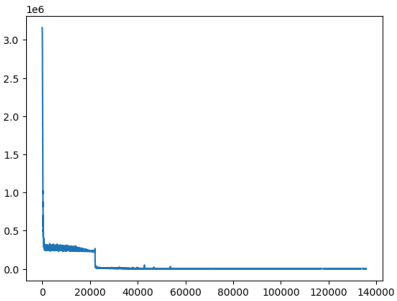
[4084.3, 4309.5, 4451.9, 4499.2, 4621.2, 4749.6, 5014.5, 5388.1, 5893.7, 8405.6]

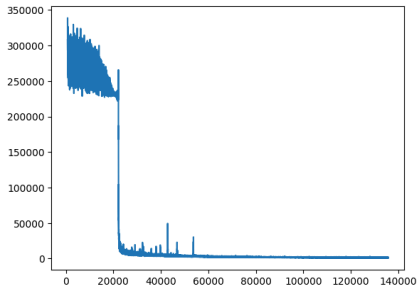


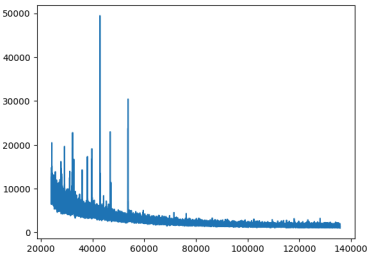

---

Not significant enough to implement in training so the original model hyperparameters were continued with.

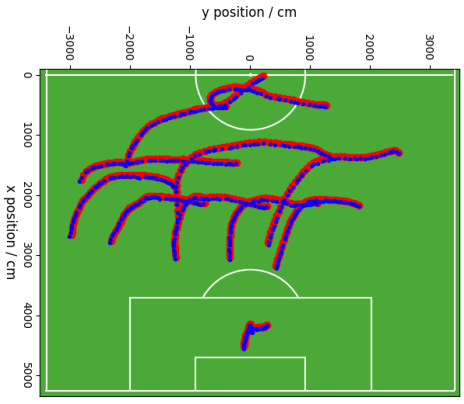
SIDE EXPERIMENT:
-when lr was made to 0.001 and allowed to run to see if this allows the model to reach an extremely good score (loss < 1000):
1000 reached after: 135792 epochs



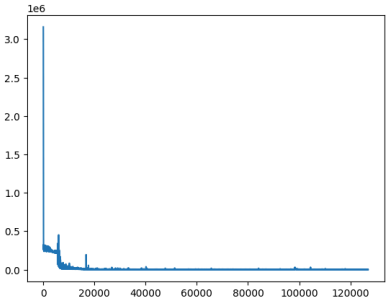



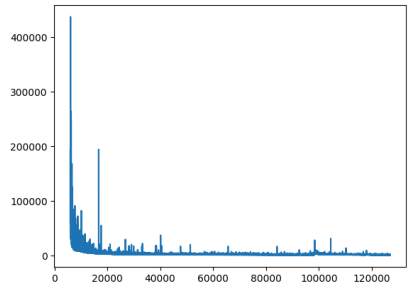


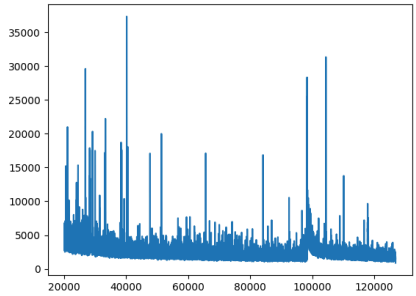
Pitch plot of best loss below:

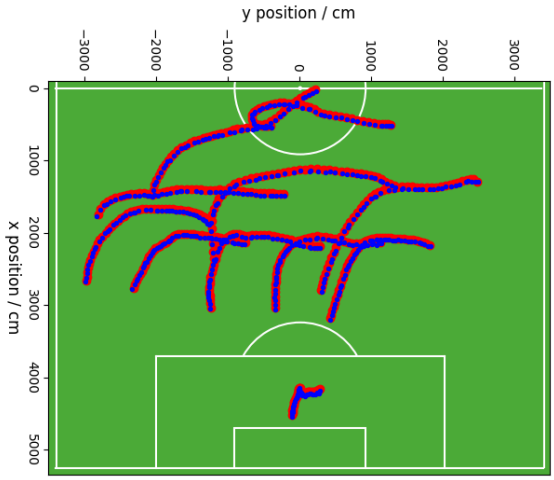


Learning rate upped to 0.01:
1000 reached at: 126918



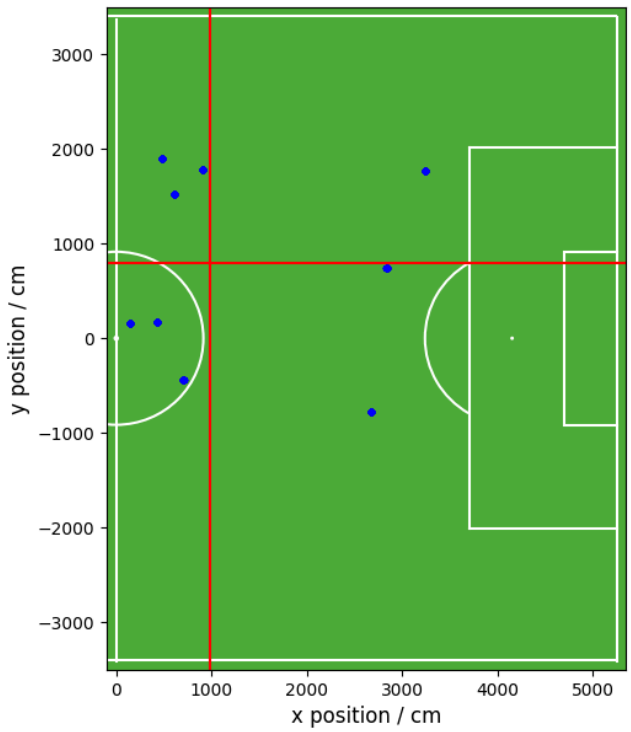



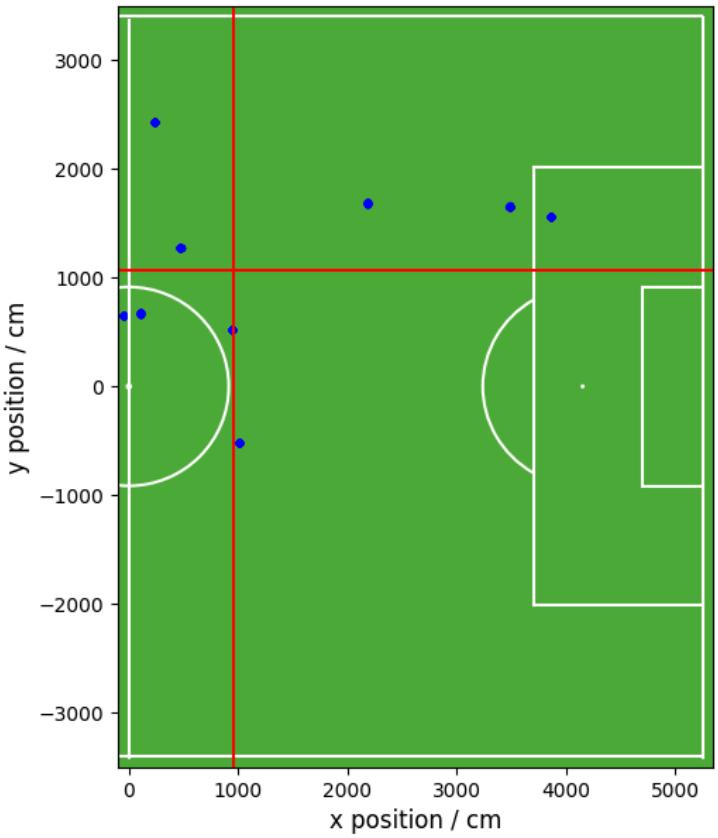




-not used but shows the model can get to extremely good results on one sequence if it has the time (suggests that local minima are not a problem)
-Doesn’t seem to make a difference between 0.01 and 0.001 lr, the percentage difference is too small.

## Week 10

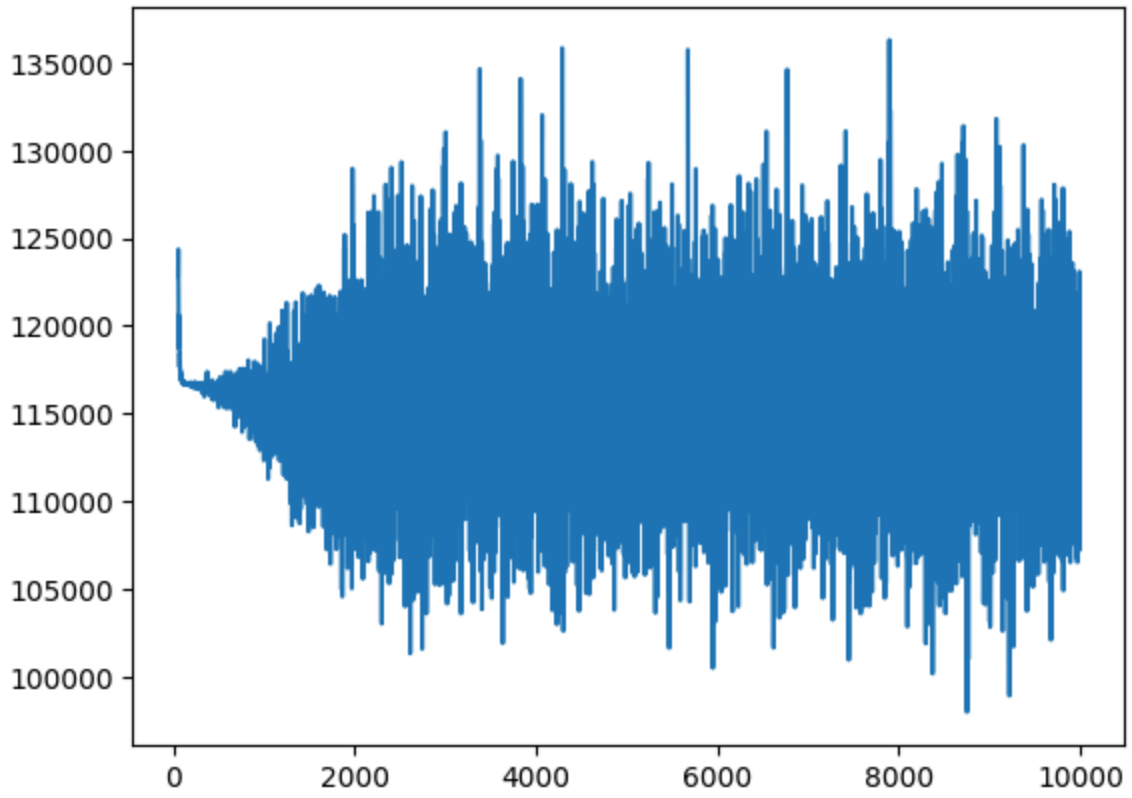



Changed bias from 0 -> 1000, player positions seem to start randomly around the pitch. Upon further inspection the average in the x and y seems to be around the (1000,1000) point. Will retrain and see if it gives another random distribution around this point.

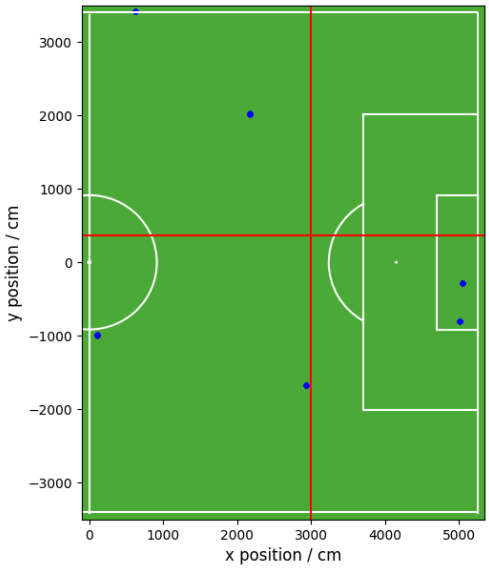



After restarting the training process and looking at the initial prediction, we see that it is infact a random distribution around (1000,1000). Note that we are changing all the bias' to 1000 here. Next step is to see if we can change bias' to different values to set the center of the distribution to the average position of the [redacted] Game.

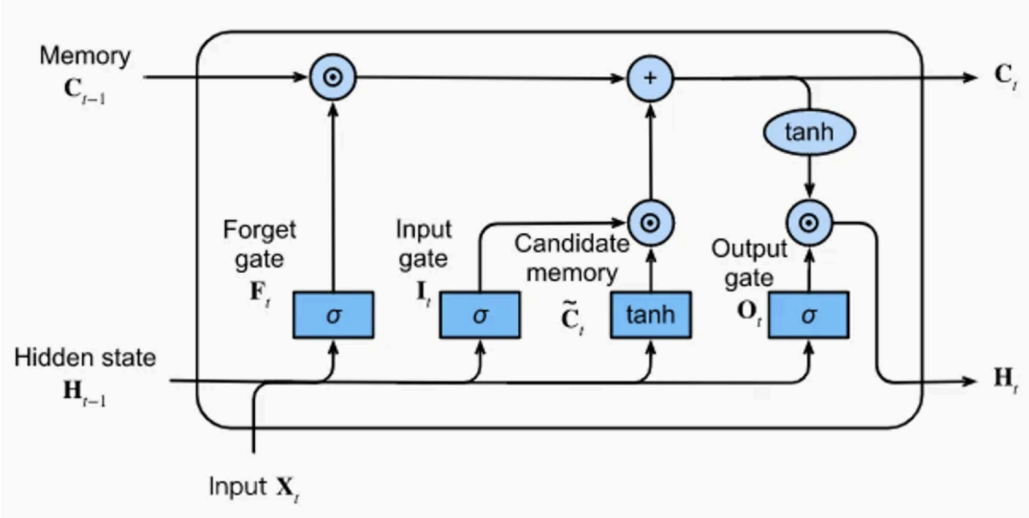
However, it seems that training is taking a much longer time despite giving it a better initial guess as seen below.




Note that i have omitted the first 50 epochs here to remove the initial fast decrease from 1e6. This is training over 10000 epochs compared to before we saw that it was converging to ~40,000 after ~3000 epochs. Learns quite well but then spikes up a lot.




Result from setting all initial bias' to alternating (3000,0), we get all initial points distributed roughly around (3000,0) this is rough because the weights and bias' throughout the network are changing where the initial guesses are.




[10.1. Long Short-Term Memory (LSTM) — Dive into Deep Learning 1.0.3 documention]

LSTM diagram for our own understanding of what the parameters are doing.

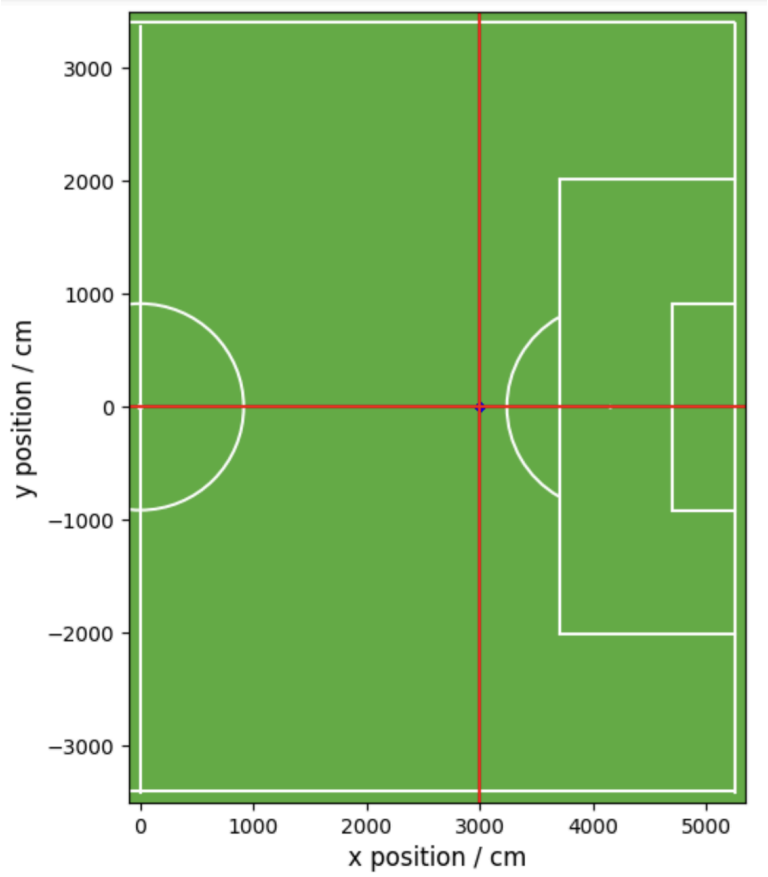
These are the initial parameters that we can set

- Parameters starting with LSTM are part of the hidden lstm layers
- Parameters starting with FC are part of the fully connected layers

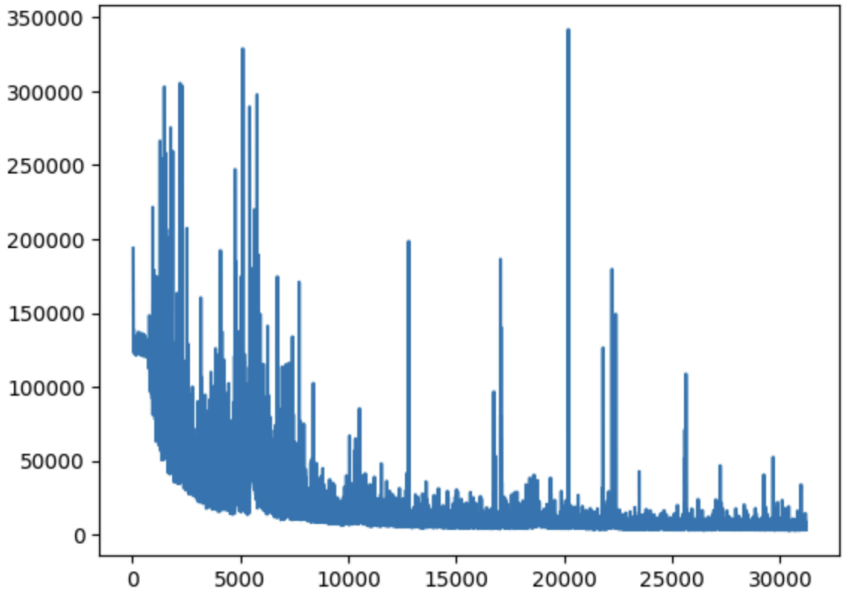
Within the LSTM parameters are IH ( Input Hidden) and HH ( Hidden Hidden)

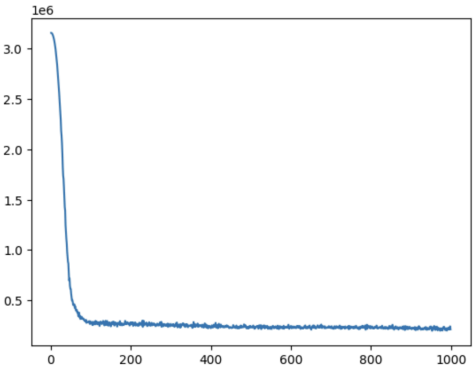
- IH weights and bias' act on the connections between the input data and the gates (how current input affects output)
- HH parameters act on the connections between the hidden layer and the current output (how previous time steps affect the current output)

However, if we just set the final fully connected layer bias' to [3000,0] and all other bias' to zero we can force all points to start at [3000,0]. As weights are initially very small they have little effect on the final output (final bias' dominate).



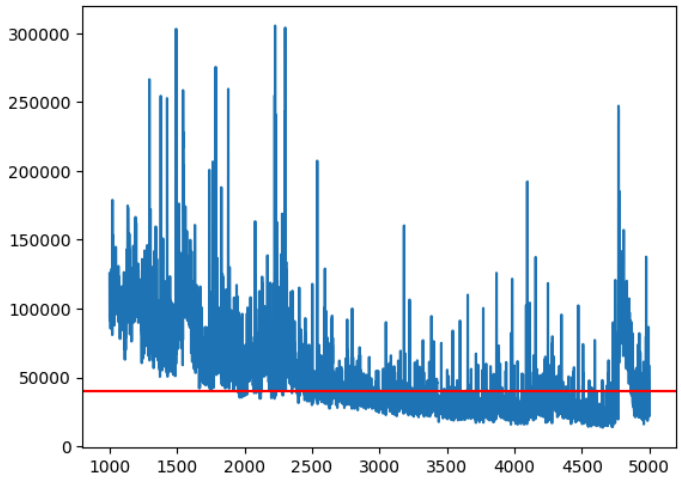
Training the model on one sequence but with the initial parameters forcing the model to start at (3000,0) but giving it no priori bias to any long/short term memory gives the following loss.




(reference for previous 1 sequence training)

This hits a loss of ~50,000 after ~1000 epochs (much better than before). However, compared to the previous 1 sequence loss graph this is very noisy. But does reach this better value quicker.



(comfortably under 2000 epochs)

Breaks from straight lines at 800 epochs

Will now investigate how quickly it reaches this good point from trying:

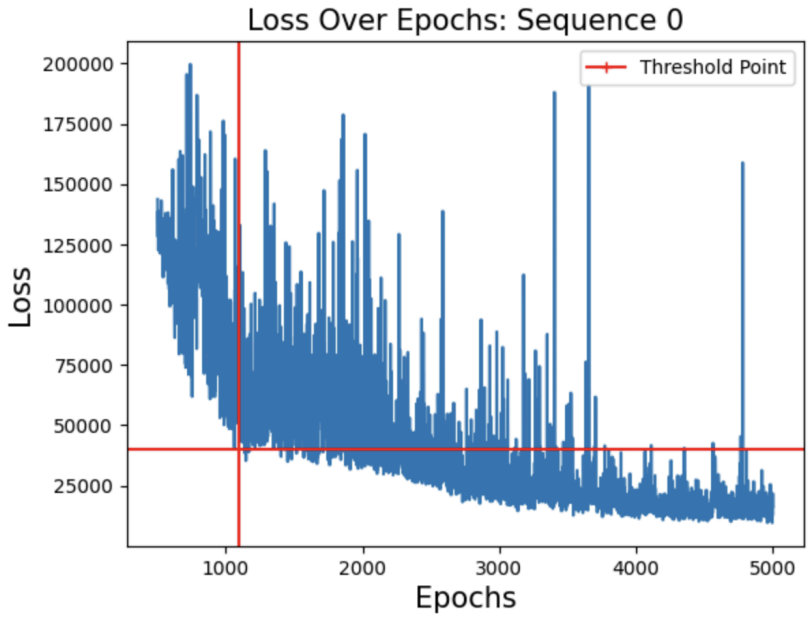
- Random bias allocation
- Training the model on a fixed sequence (one frame 45 times) and starting with these parameters
- Training the model on a normal sequence to perfection and starting with these parameters

**<u>Inspecting parameters</u>**

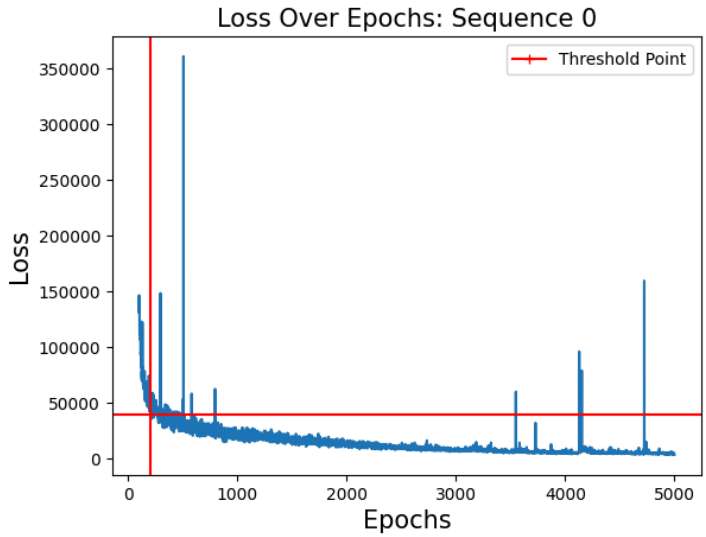
-Model trained on a static sequence and got down to a loss of 90. The parameters were saved. It is seen that the biases for fc2 have not deviated much at all from the set values [3000,0,3000,0…] but bias' in other layers have also not deviated much from their initial zero point.

Would be interesting to see if we set other initial values if it still reached the same loss but with a completely different set of initial values and still not deviating much.

Bias' and weights seem to all be very small (between 0->1)

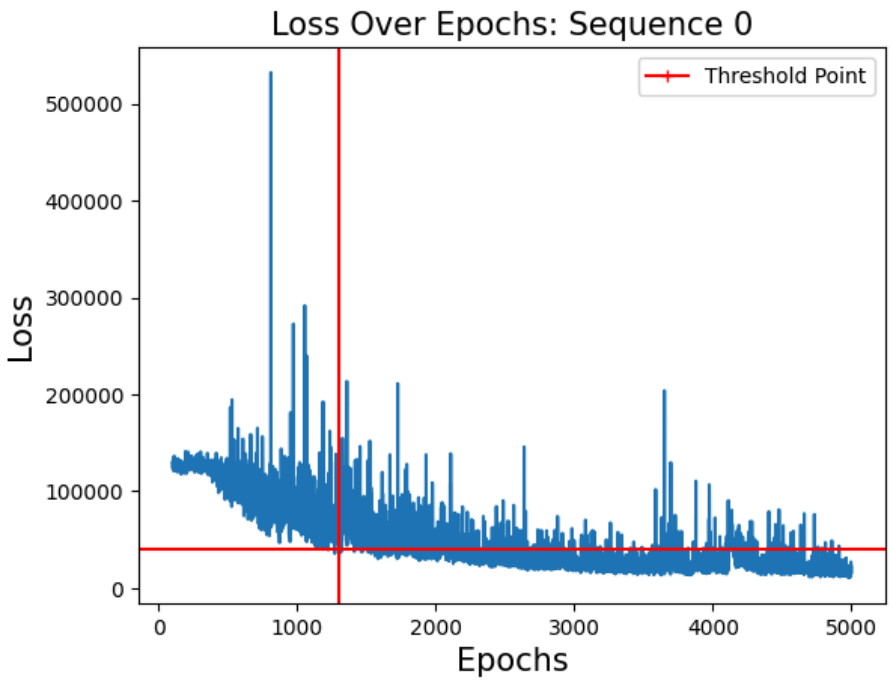



Result for randomising all other bias' (also video) definitely converges quicker. Also reaches minimum loss of ~2000
Breaks from straight lines at epoch 400




Loss from training the model after importing initial parameters from training on 1 good 442 sequence. [Sequence 660].
Minimum loss of ~1000 (after 30,000 epochs)

```
p_path = f"/dbfs/plots/stationary_still_pitch.pt"
l_path =  f"/dbfs/plots/stationary_still_loss.pt"
m_path = f"/dbfs/plots/stationary_still_model.pt"
```




Loss from training on stationary sequence (Sequence 660 still)
Takes ~1500 epochs to reach loss of 40,000
Still good but not as good as above. Would like to test different responses on more than one sequence.
Note minimum loss was ~2500

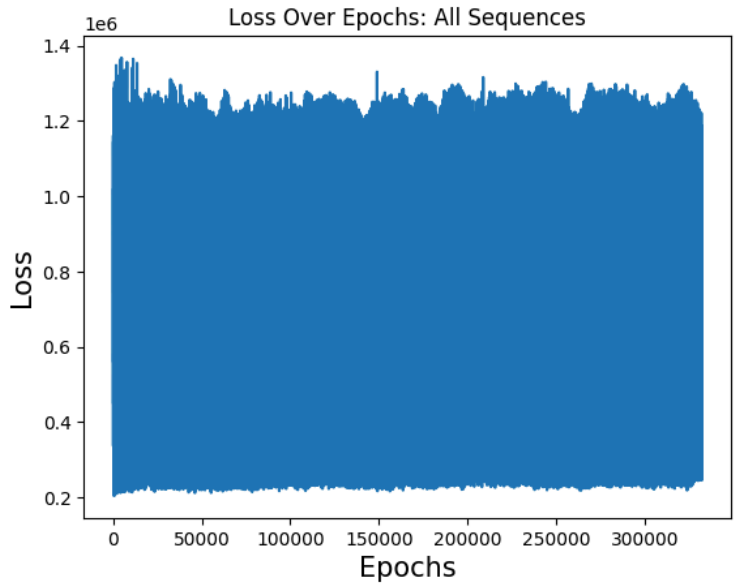
```
p_path = f"/dbfs/plots/stationary_seq_pitch.pt"
l_path =  f"/dbfs/plots/stationary_seq_loss.pt"
m_path = f"/dbfs/plots/stationary_seq_model.pt"
```

Looking at the training videos, training from a sequence that already has time dependence means that the training does not start in straight lines (converges much faster)

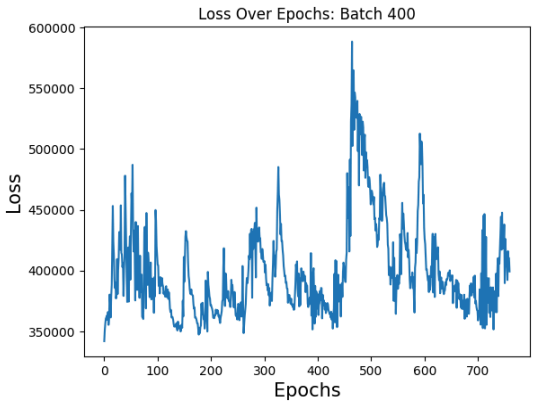
Even when initialising the bias' as random, the model still starts learning in straight lines.

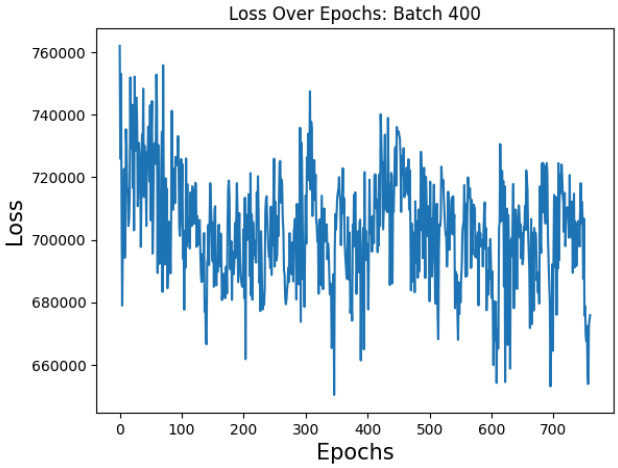
Started doing a long term training process on a large dataset. Used the 10 epoch split, (train on each batch 10 times) and currently done for 760 cycles (~6 hours of training time)

Full loss graph is as follows:

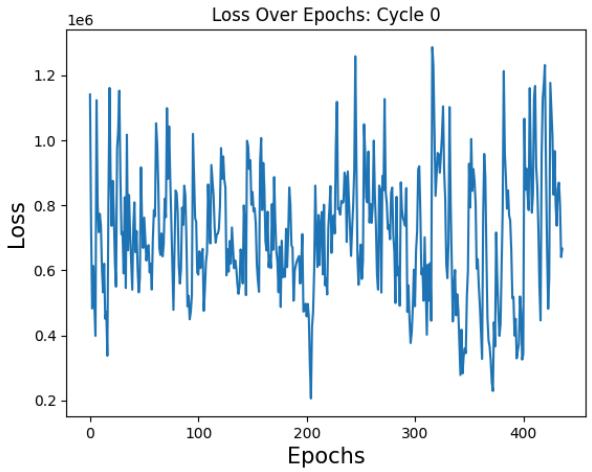


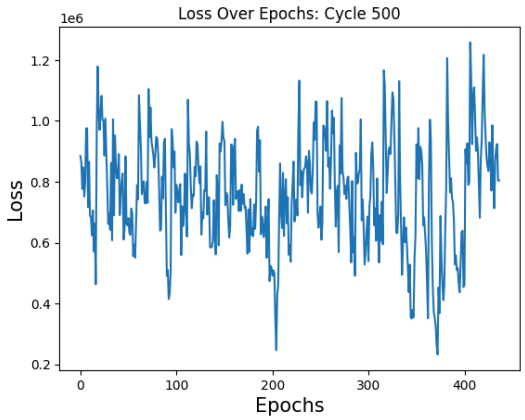

Or isolating a single sequence:



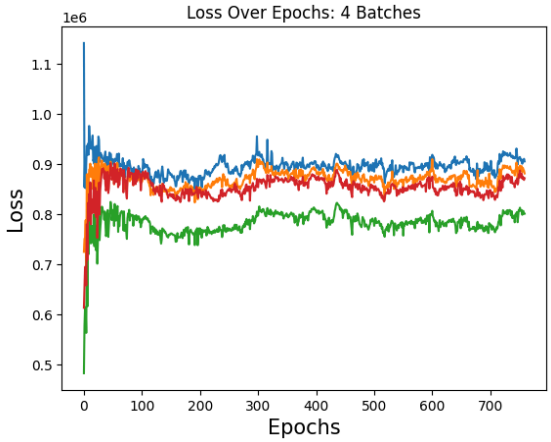



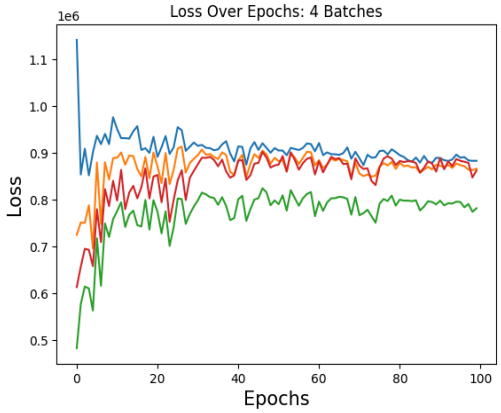

Or the loss for a single cycle:



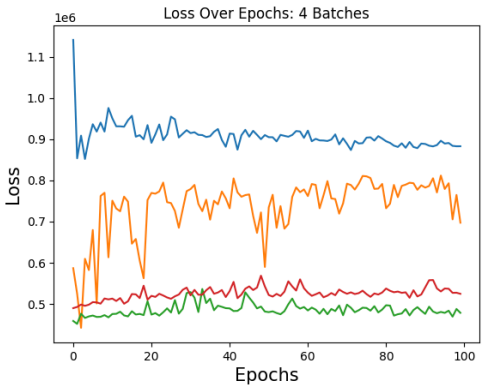



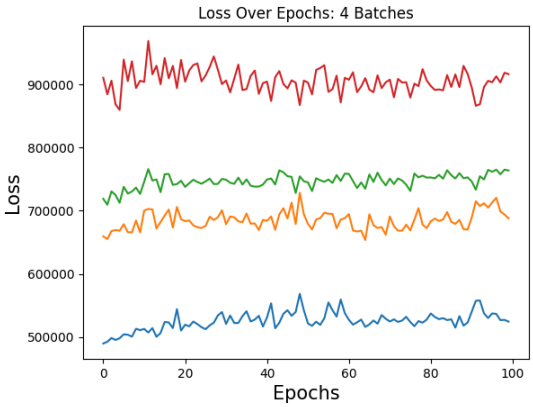

Plotting multiple Batches on one plot



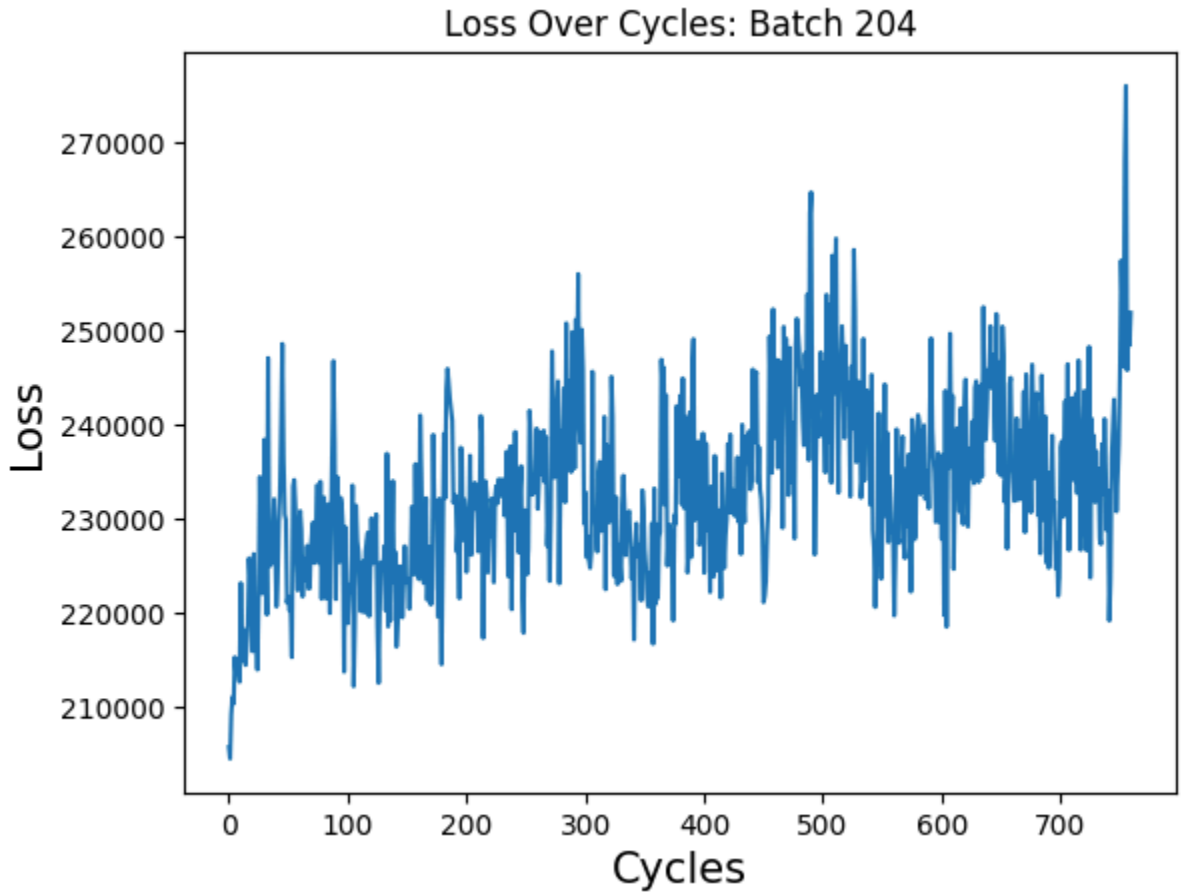

Plotting 4 batches that aren't next to each other just to be see if they are all converging to the same value (Batches 0,100,200,300 LEFT) (Batches 300,301,302,303 RIGHT)

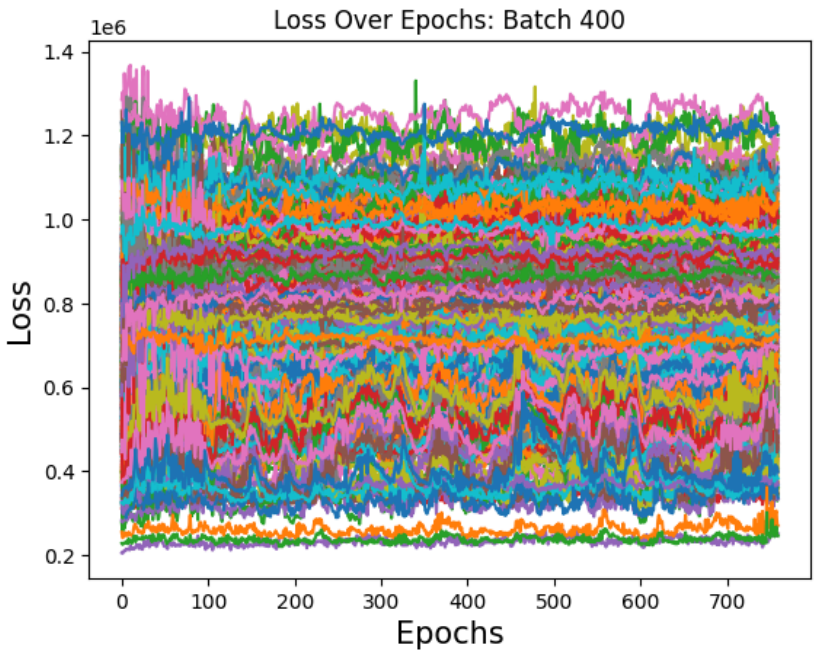


Note have only plotted first 100 cycles but checked all 760 and roughly constant.

And finally, the best batch (lowest loss)






Plotting all batches just to check all loses aren't converging on an average. Bring up Chebyshev inequality.

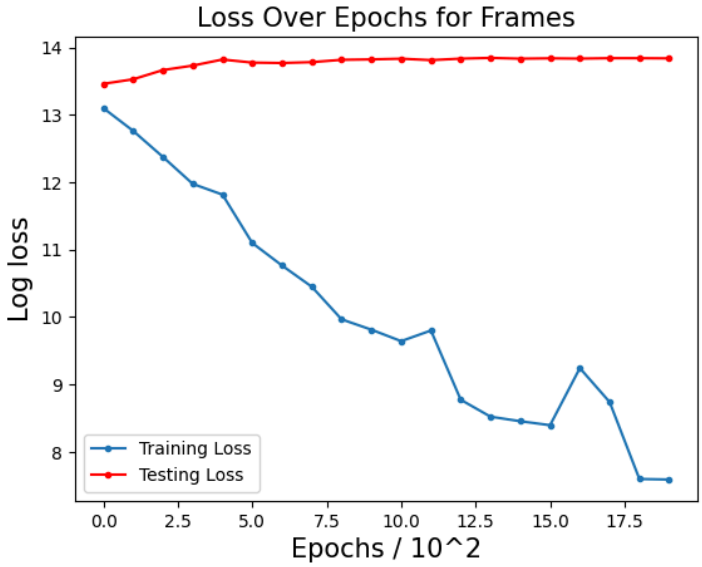
**Week 11**

It was decided that the idea was too complicated. Football is too complicated for the model to learn so two potential routes were considered:

- Decrease the length of the sequences to ~5 frames to make it easier for the model to learn - less complicated patterns to learn
- Reduce to a learning singular frames -> removes all time dependence from the network. Stop using an RNN/LSTM and use a normal Neural Network instead. This will be much easier for the model to learn.

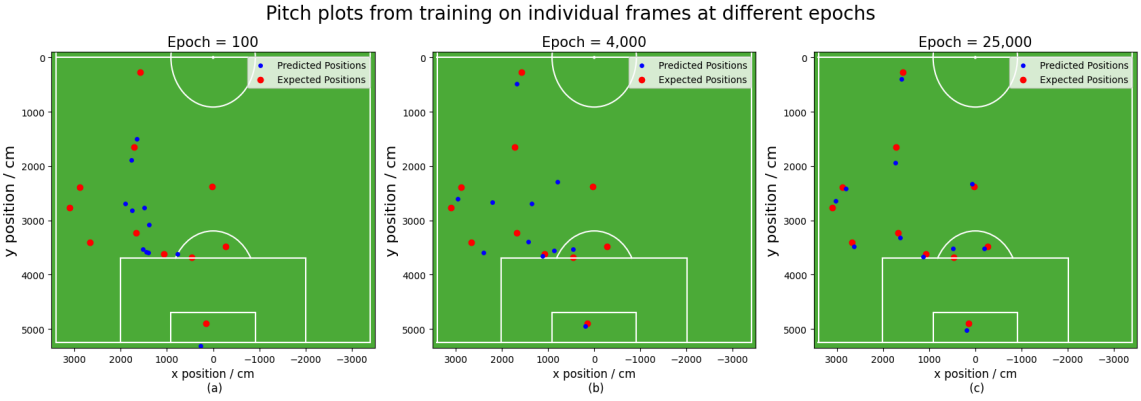
First started with individual frames. This meant redesigning a model but it would be much simpler. A singular frame was randomly chosen from each

sequence to remove any bias in the learning process. Therefore, the dataset was the same size. Could potentially pick 5 frames from a sequence to increase the size of the dataset, but this was safer to remove bias for now.




Also changed to training on all frames at once instead of training on a singular frame/minibatch. This was possible because the dataset was 45x smaller (no sequences) so not as memory intensive.

Model learnt all frames very well to a log loss around 8. This was an average across all frames so some frames were worse than others but this was a big improvement. Results are shown below.

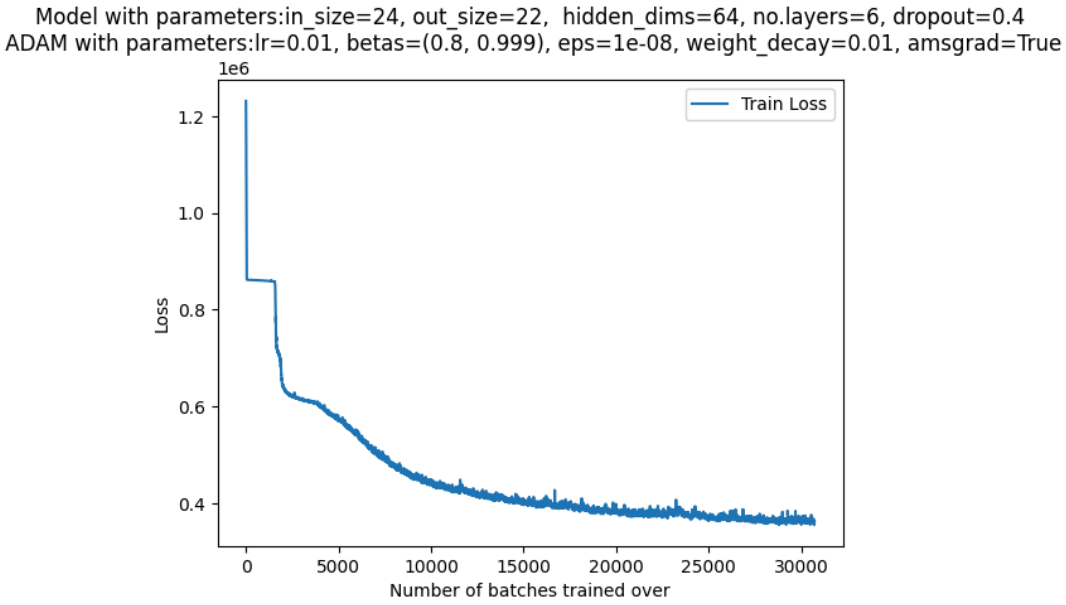



-All data was batched into one so the batch size is 'len(input_train)'

Almost perfect output, rather consistent over all frames. However, very heavily overfitting. Did not generalise to learning patterns but was finally

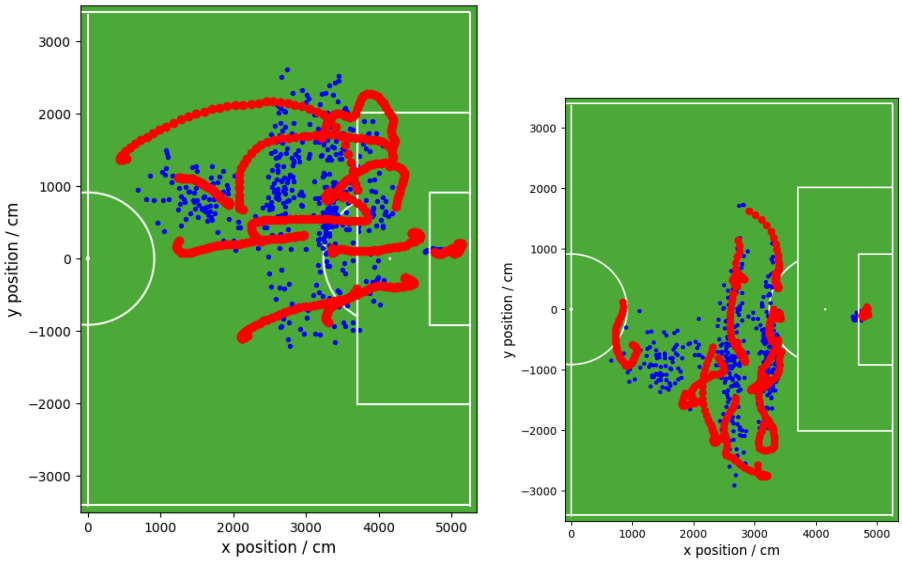
reproducing the training set well.  This was a big step.  An attempt to reduce the overfitting was made; however, as this was not the primary aim of the project, this was abandoned to try the new training technique on the sequences.  Any attempt to improve the overfitting (eg. decreasing model complexity / adding dropout layers) but this made the training process so much worse so this was left.

Returned to training on whole sequences but gave the model every sequence at once.



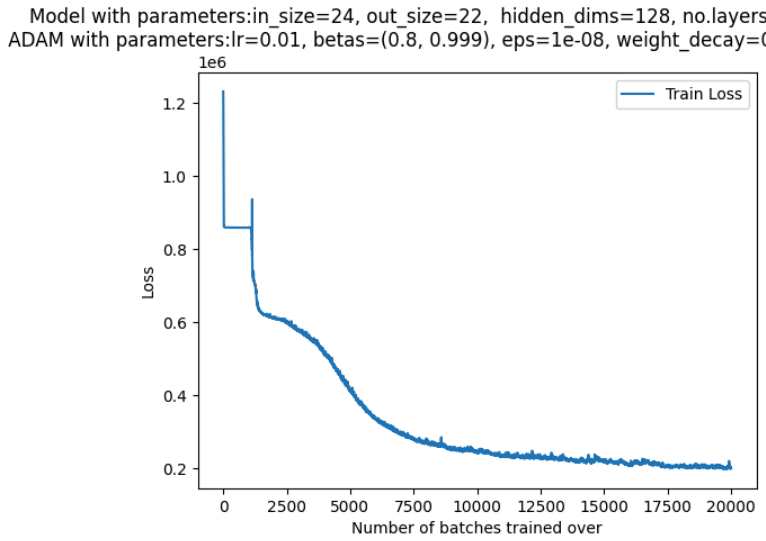

Training showing good promise in that it is asymptoting slowly at a decent value, improved on previous results considering the size of the dataset.

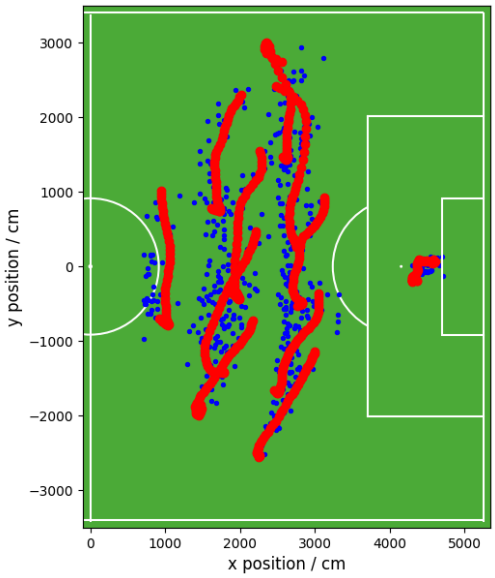
Below are 2 plots that show two random sequences from this training cycle:




```
p_path = f"/dbfs/plots/pitch0305.pt"
l_path =  f"/dbfs/plots/loss0305.pt"
m_path = f"/dbfs/plots/model0305.pt"
```

This was much better than before, the model was producing a different output for a given input.  However, outputs were still not perfect so the hidden layers were then increased to 128 to make the model more complex:

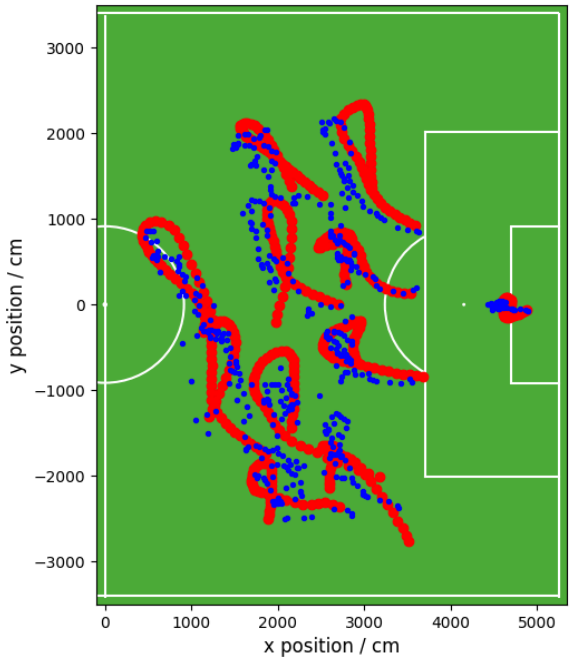


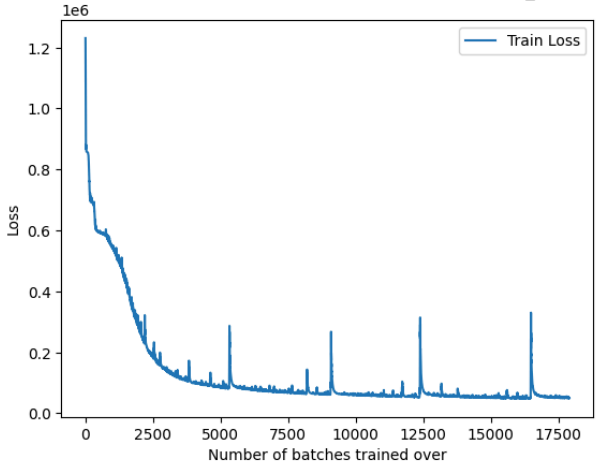

This reached a lower loss reached (∼200000) here. Which was a big improvement  but still not perfect

```
p_path = f"/dbfs/plots/pitch03052.pt"
l_path =  f"/dbfs/plots/loss03052.pt"
m_path = f"/dbfs/plots/model03052.pt"
```

Hidden layers increased once again to 256:

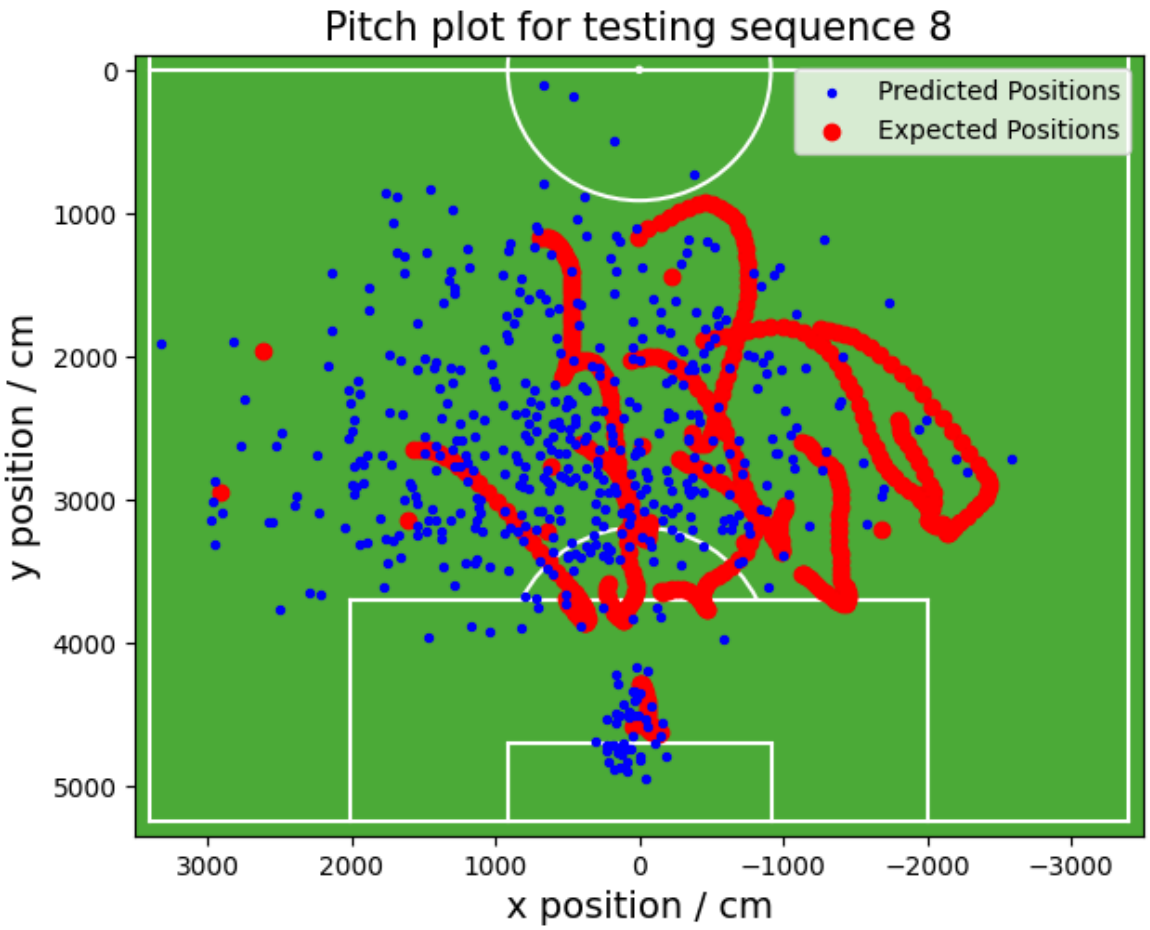



Model with parameters:in_size=24, out_size=22, hidden_dims=256, no.layers=3, dropout=0
ADAM with parameters:lr=0.01, betas=(0.9, 0.999), eps=1e-08, weight_decay=0.0, amsgrad=True

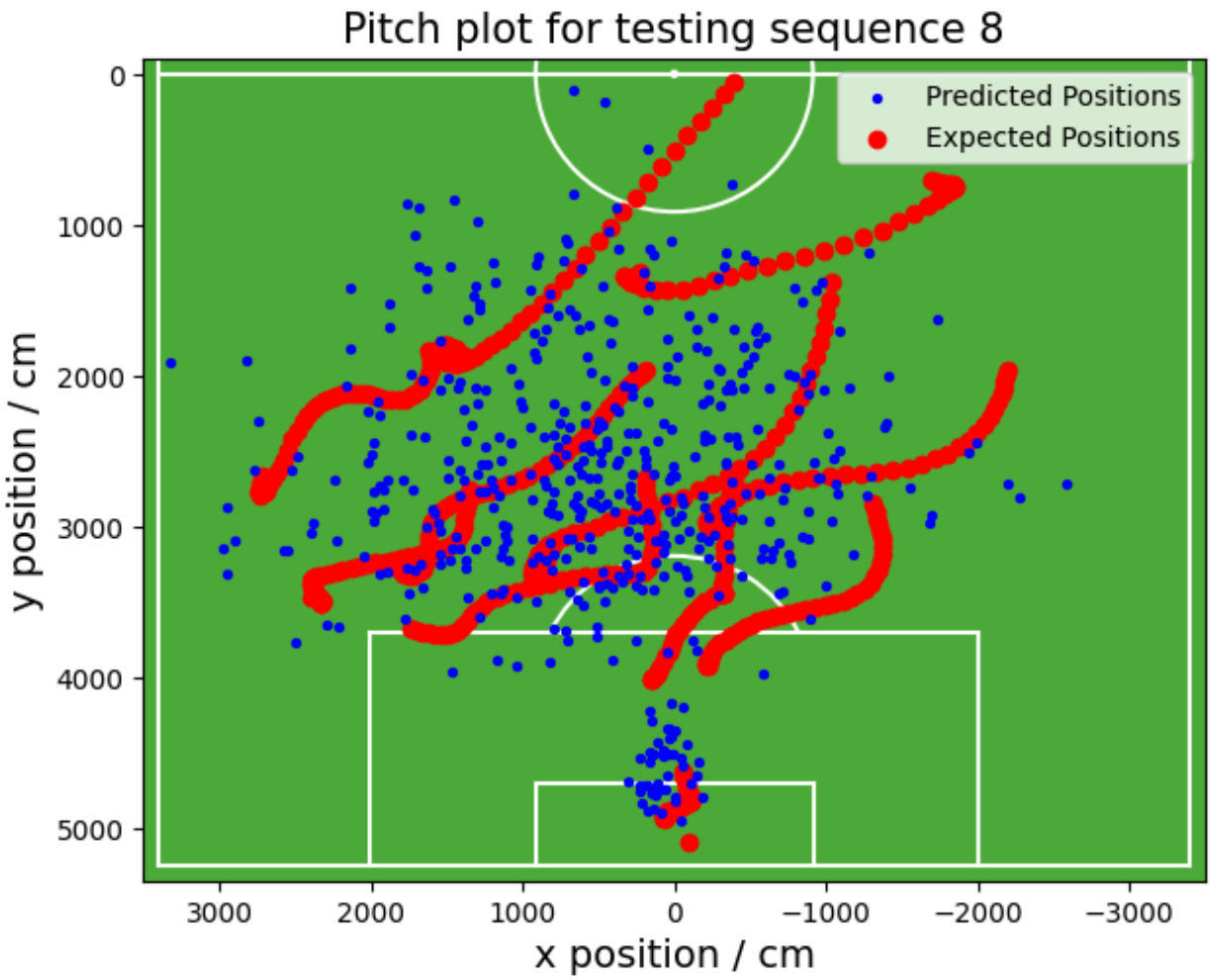



```
p_path = f"/dbfs/plots/pitch03053.pt"
l_path =  f"/dbfs/plots/loss03053.pt"
m_path = f"/dbfs/plots/model03053.pt"
```

Loss got even better (~40000) but testing showed strong overfitting. Spikes in the loss show the model trying to get out of local minima. Pitch plot below shows an output from testing:



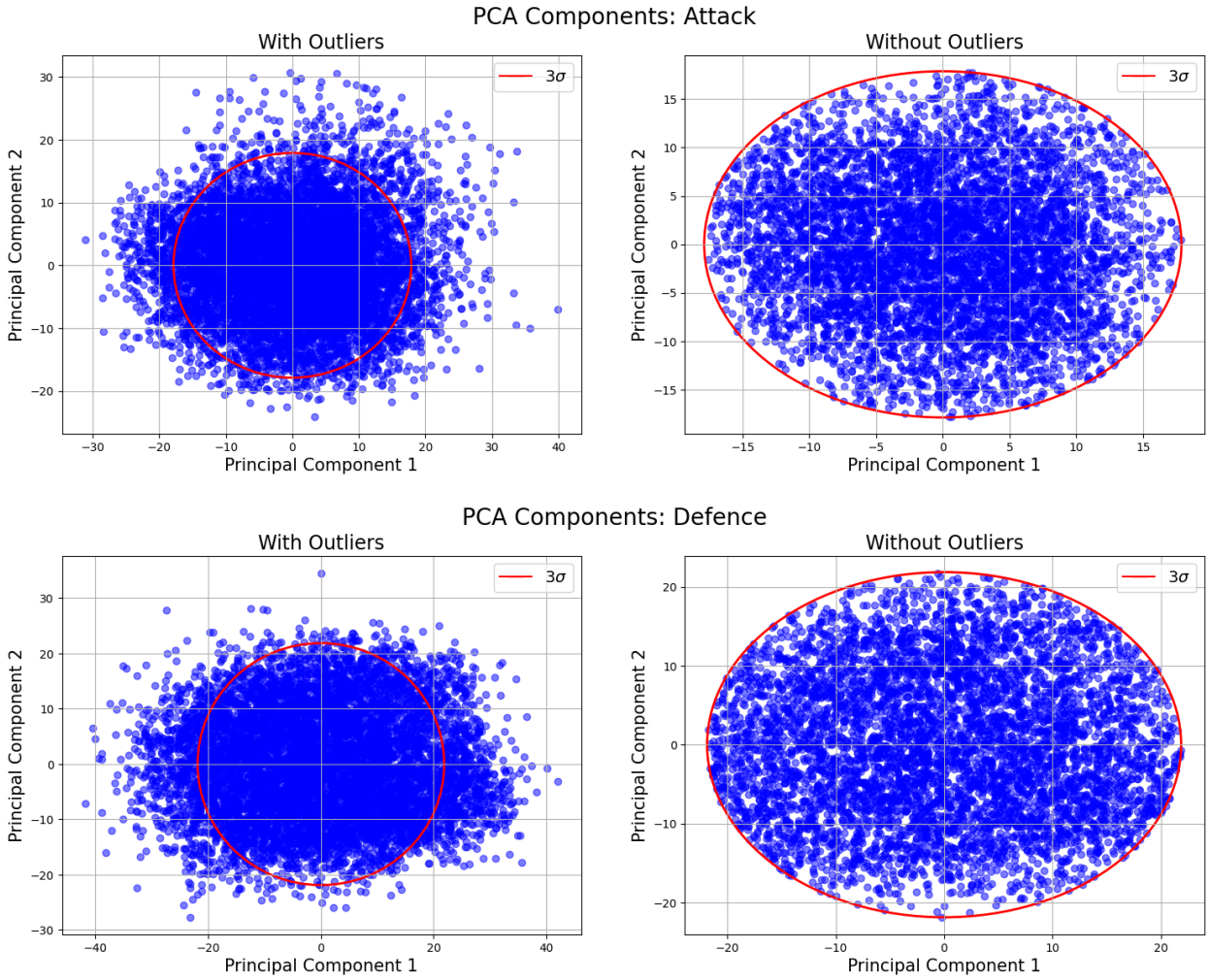

Model does not generalise to the testing set at all.  Producing an almost random output, however, it is different depending on the input so the model is now taking notice of the input (improvement).



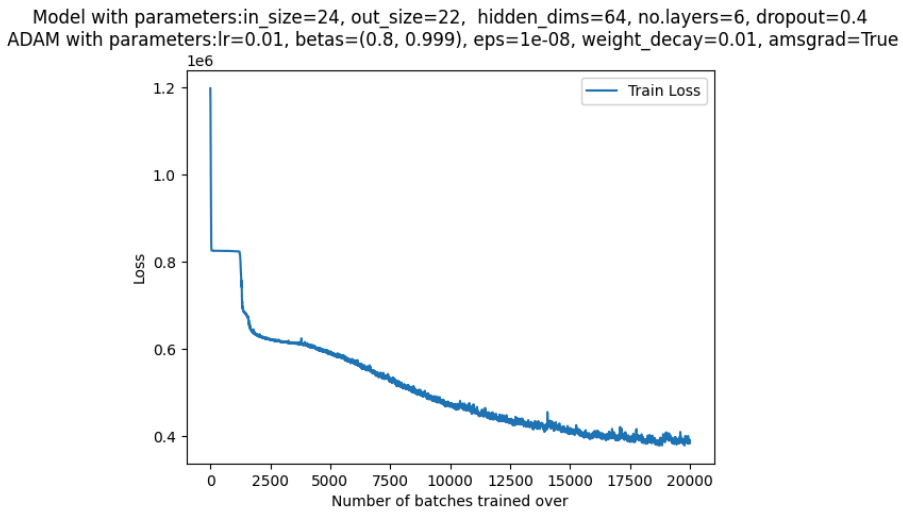

A balance between model complexity and overfitting could not be found. Anytime the model complexity was reduced slightly, the training process asymptoted at ~400,000 loss meanwhile the test loss barely dropped. So tuning hyperparameters would not stop the overfitting. There was still another problem.

<u>After PCA</u>

One problem could be the dataset was still too noisy.  So, PCA was done on attack and defence data to narrow down sequences to make it easier for the model.




Training loss plotted below:




Output from training over night produced near identical results to before. So this didnt work.

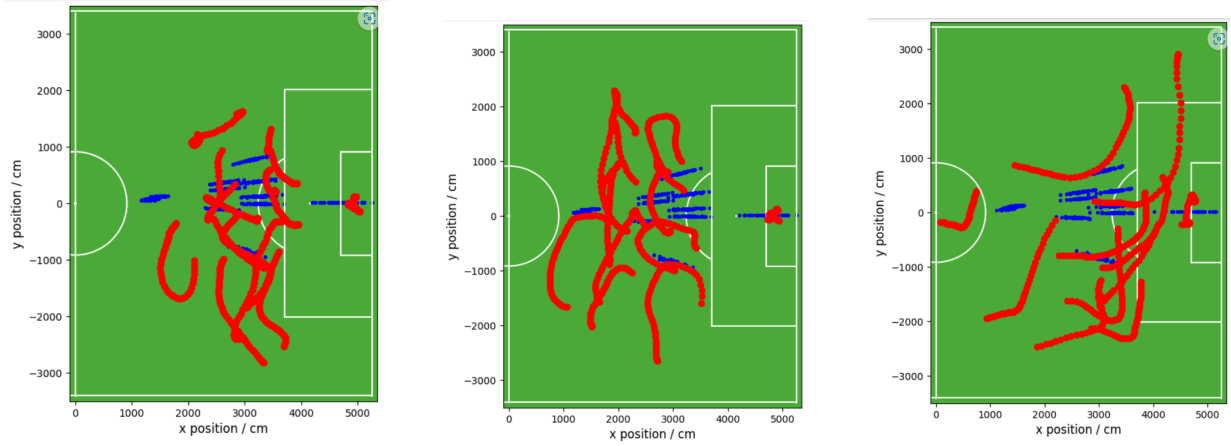
Potential solutions and recommendations for the future are included in the Lab Reports.

```
p_path = f"/dbfs/plots/pitch0405.pt"
l_path =  f"/dbfs/plots/loss0405.pt"
m_path = f"/dbfs/plots/model0405.pt"
```

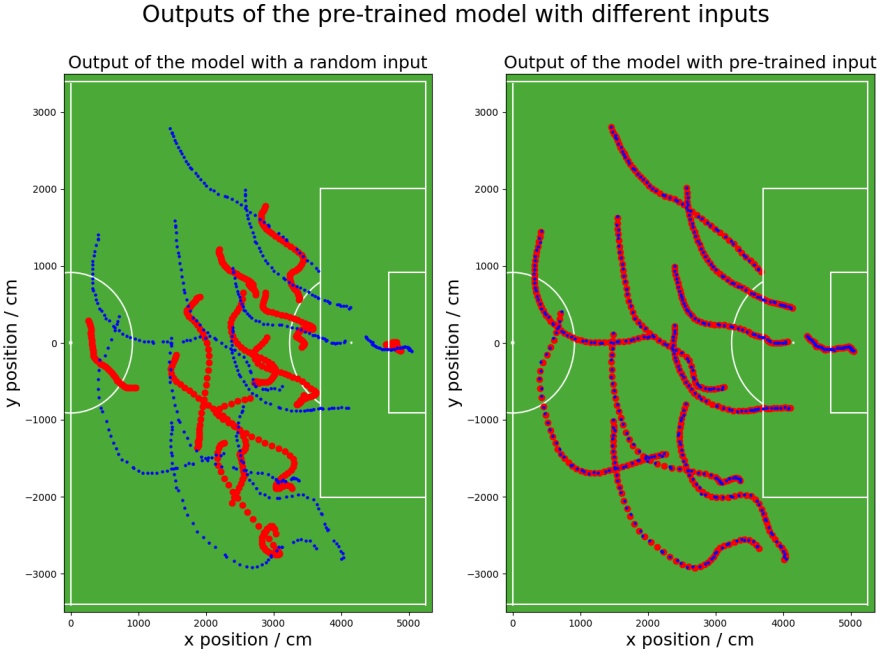
## Additional notes / really rough notes (ignore past here)

```
train_data_path = "/dbfs/[redacted] Data/train_loss_array_1_epoch_per_sequence.pt"
test_data_path = "/dbfs/[redacted] Data/test_loss_array_1_epoch_per_sequence.pt
training_seq_path = "/dbfs/[redacted] Data/training_1_epoch_seqs.pt"
testing_seq_path = "/dbfs/[redacted] Data/testing_1_epoch_seqs.pt"
```

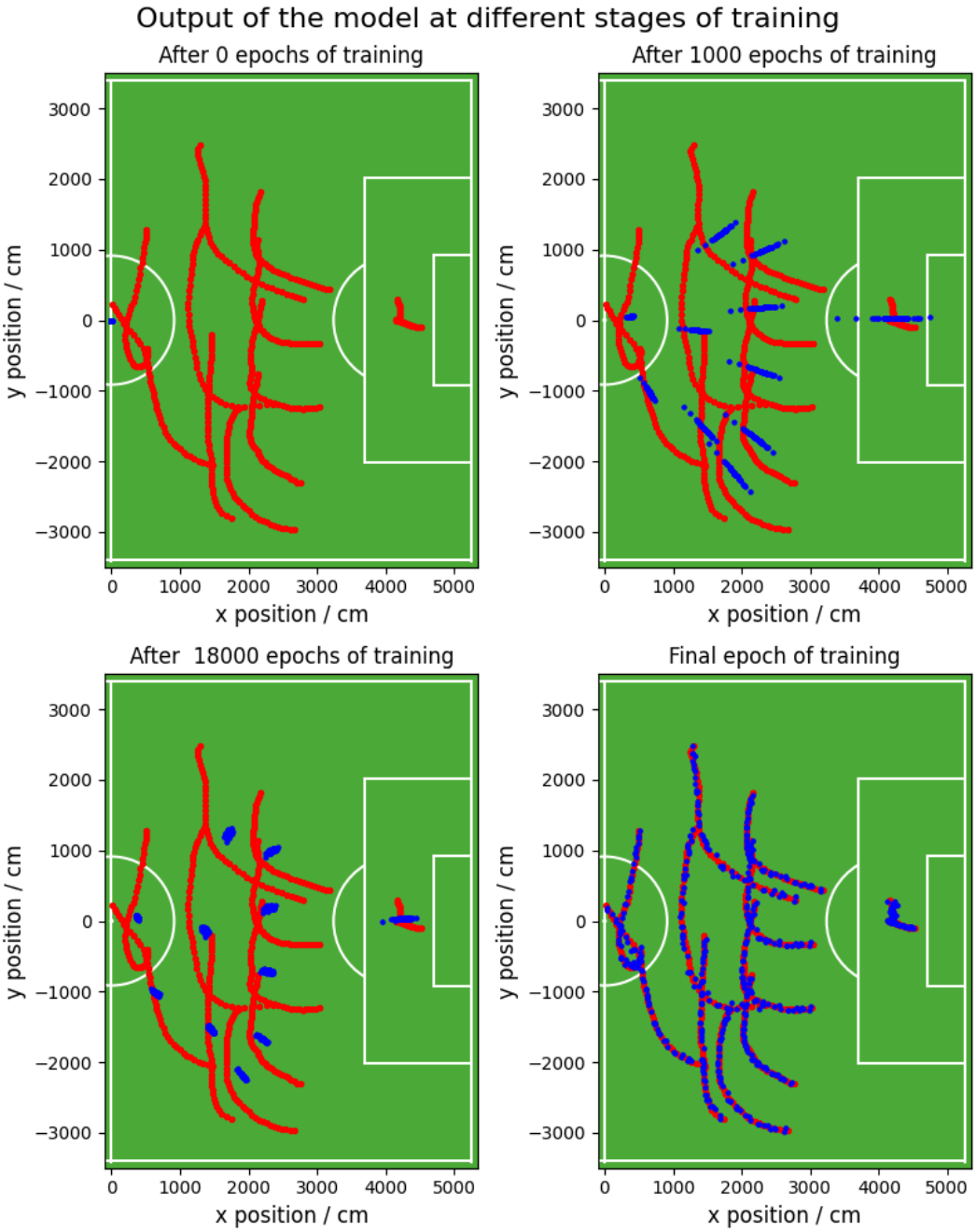
## Model producing same output for different inputs/plotting an ‘average’

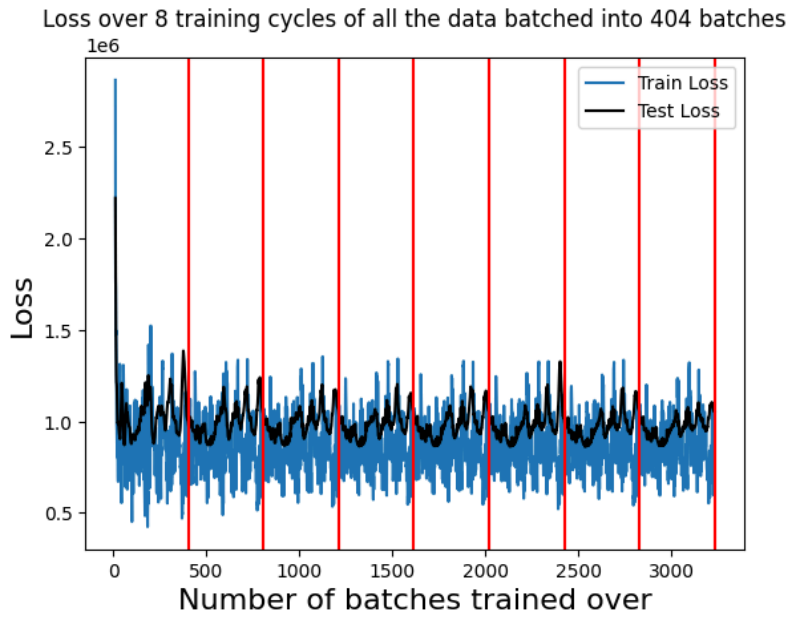



## Model taking no notice of input:



Training process:






## FILES

-”dbfs[redacted] Data/ptb_game_data.pt” - the ordered [x,y,x,y] data from [redacted]

-"/dbfs[redacted] Data/incvel_input_data_ALL [redacted].pt" [4140,45,48]

-”/dbfs/[redacted] Data/input_data_ALL [redacted].pt” [4140,45,24]

-"/dbfs/[redacted] Data/input_data [redacted].pt" [9293,45,24]

```
model_path = f"[redacted]Model/442_parameters.pth" - Model parameters for still sequence
model_path = f"[redacted] Model/442_parameters_seq.pth" - Model parameters for regular sequence
```

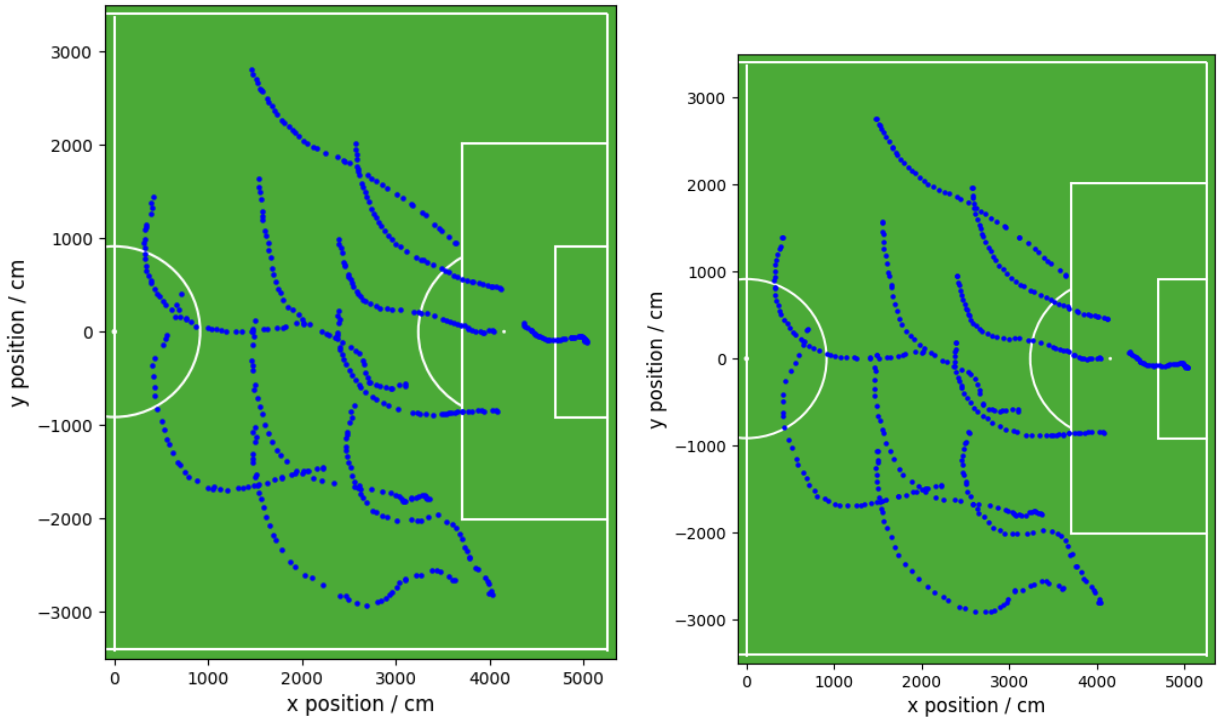
```
Shape of input for sparse matrices - (batch_size, num_time_steps, input_channels, num_rows, num_columns)
```

-Finalise DATASET
-adjust training epochs/batch size
-adjust model

VISUAL STUFF
https://stackoverflow.com/questions/52468956/how-do-i-visualize-a-net-in-pytorch

nc.dvi (psu.edu)

Similar outputs from meeting with Terry:




https://www.quora.com/Why-does-my-own-neural-network-give-me-the-same-output-for-different-input-sets