Final Project Report

Edward Banner*

December 9, 2015

1 Introduction

The keeper of a robot soccer team is an important component to the overall team. A keeper's job, put quite bluntly, is to keep the game ball out of its goal. One skill necessary for achieving this is being able to detect when the game ball is on course to roll across the keeper's goal plane.

Having a good shot prediction model could be the difference between a winning and losing team of robot soccer players. For instance, failure to detect a shot on goal could mean an uncontested goal for the other team, while falsely determining a shot on goal could result in an opportunity for the attacking player to dribble around the keeper and score. It is developing a keeper behavior capable of *reliably* determining when a shot is underway which is the focus of this project.

To address this problem, I took a data-driven machine learning approach. Specifically, I trained discriminative models on kick data generated in simulation.

The outline of the this report will be as follows. In section 2 I will formalize the problem of shot detectoin and outline the main components of my approach. In section 3, I will discuss how my approach differs from keeper behaviors in past robocup tournaments. In section 4 I will discuss the details of generating a training set of shots via simulation, training discriminative models, and evaluation both in simulation and on an Aldebran Nao robot. In section 5.3 I will conclude with limitations of my approach, future directions of work and highlight my contributions.

Throughout this report, I will refer to my Aldebaran nao robot as *Cheesy*.

*ebanner@cs.utexas.edu

2 Approach

The problem is to develop a keeper behavior capable of determining whether a shot is underway on the keeper's goal. Formally, I treat the problem of shot detection as a supervised learning problem, where the goal is to find a function $f : X \rightarrow Y$ where X contains information about the game ball and $Y = \{$no-shot, left-shot, middle-shot, right-shot$\}$. In particular, I investigate the following features for shot detection:

- (x, y): the position of ball in global coordinates
- (dx, dy): the velocity of the ball in global coordinates
- (distance, bearing): the distance and bearing of the ball in Cheesy's local coordinate space

Position and velocity estimates are derived via a Kalman filter and the distance and bearing are calculated from the ball detection code in Cheesy's vision module. Position and velocity are in global "top-down" coordinates (see section 4.1.1) and distance and bearing are in Cheesy's local "egocentric" coordinate frame.

In this project, I explore different feature spaces during training. Specifically I ran experiments with [x, y, dx, dy], [x, y], and [distance, bearing].

3 Related Work

Perhaps the most important prerequisite of reliable shot detection is having an accurate model of the

game ball. Previous approaches to modelling the ball by past robocup teams typically include estimating the position of the ball, as well as its velocity. In [8], the team models velocity by populating a fixed-sized window of the most recent global ball positions. The queue is then split in half. The two halves are averaged, resulting in two average positions. These two positions are then used to calculate a velocity vector. My approach is similar in that I also explicitly model the velocity of the ball, but instead with a Kalman filter.

Another approach to modeling the ball involves the use of a kalman Filter [9]. A kalman filter here is maintained for each distinct hypothesis regarding where the ball currently is. The team in [9] encodes their prediction model with 2D kinematics, which relates future ball positions with its estimated velocity and position at the previous time step. Friction between the ball and floor is also incorporated. [5]. In my approach, I do not incorporate friction into the estimates for position, it is implicitly encoded in the kick trajectories generated in simulation. But I do use a Kalman filter for modelling the ball.

One drawback with kalman filters is that there are several parameters that need to be tuned by hand for them to work well. In [1], the authors circumvent this issue by training discriminative models to learn these parameters, circumventing this issue. They induce supervision by placing a highly accurate GPS sensor on the robot whose position they are estimating. I did not pursue this approach because I did not have a similar way to induce supervision in a similar way. The overlap here is in the use of discriminative models.

Once a model of the ball has been obtained, it can be used as input to a shot detection algorithm. In [8], the team uses velocity of the ball to drive their decision whether to step the keeper left, right, or execute a save (although they do not specify how this is done). In [7], the team simply executes a save motion when the ball gets within a certain distance. The team in [3] forgoes discrete "save" motions altogether, rather opting for more of an active approach by pursuing the ball in hopes of intercepting it. My approach is most similar to [8] in that the models I train will likely predict a shot if the ball is close, but a key difference is my models have the ability of predicting no-shot even when the ball is very close (but not moving).

The process of learning has been exploited in robot soccer. For example, the team in [5] learns to model the friction between the floor and the ball by rolling the ball in front of the robot several times and recording the decelleration of the ball. In my approach, I do not learn the friction of the surface, but rather set it to a fixed value. Another example is the team in [4], where they make use of reinforcement learning and policy gradient methods, in addition to genetic algorithms to optimize dive times. Unfortunately they do not go into details. Further, it is not clear whether they use these learning approaches for optimizing how *fast* to dive or when to *decide* to dive.

Learning from simulation in particular has also been explored in the robot soccer domain. [2] learns parameters for a robotic walk in simulation. However, the main difference in my work is that I apply what is learned in simulation directly to a physical robot.

4 Methodology

In this section, I detail the end-to-end task of creating reliable shot detection for a keeper behavior which makes use of disciminative models trained on data generated via simulation. In section 4.1, I discuss the data generation process. In section 4.2, I detail the models and training process. Finally in section 4.3, I will discuss evaluation on synthetic data, as well as on Cheesy himself.

4.1 Data Generation

4.1.1 The World

The first step in a supervised learning problem is to secure training data. I forgoed the idea of collecting data on Cheesy and chose instead to generate shots in simulation due to the degree of control it allows.

In order to generate shots which closely matched what Cheesy would experience, I started off studying the Goalie Simulation tool Jake developed for the

class for the Kalman Filter assignment. In this simulator, a kick corresponds to setting the initial velocity shot on the game ball, stepping the ball forward by an amount proportional to the velocity, and decaying the velocity by a constant factor after each time step. Keeping true with the parameters which were used in the Goalie Simulator, I used a decay of 0.966 and 150 time steps for a kick. This corresponds to 3 seconds since Cheesy's vision module runs at 50 Hz. Figure 1 provides a visual of the keeper simulation tool.

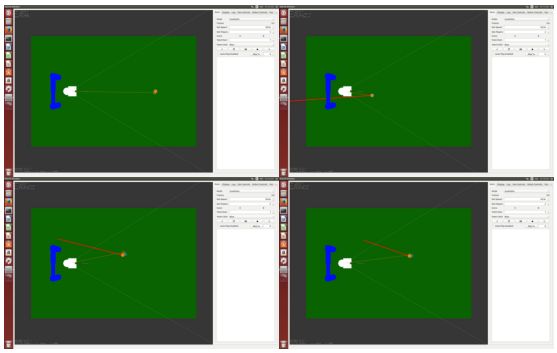

Figure 1: Visual of the Goalie Simulation tool we used in the Kalman filter assignment. Top left: the ball starts out in front of the keeper. Top right: a shot is fired and is approching the keeper. Bottom left: the kick is wide-right. Bottom right: the kick is short.

The green rectangle is 2,000 units tall and 3,000 units wide. 1 unit corresponds to 1mm in physical space. This field corresponds exactly to the miniature field in the robot soccer lab. These dimensions correspond to the dimensions of the field in the soccer lab.

4.1.2 My Simulator

Because I wanted to be able to generate many shots in a systematic way and because success in Jake's simulator translated to success on Cheesy during the kalman filter assignment, I decided to write my own version of the keeper simulator. Figure 2 shows my approach to generating an arbitrary number of shots via simulation.

In a departure from the Goalie Simulator, my goal is half the height of the field, which is ostensibly significantly larger than in the Goalie Simulator. This was the width assigned to the goal by Cheesy's keeper behavior for the kalman filter assignment. I'll refer to this behavior of KK (Kalman Keeper) from this point forward. Since I wanted to use this behavior as a baseline, it made sense to encode the same width for both behaviors.

To generate kicks, two grids are drawn on the field; one in front of the goal and one around the goal. A kick is generated by choosing a point in each of the two grids and solving for the initial velocity required to have the ball land on the latter point after 150 time steps, taking into account the decay factor. A kick is deemed a shot on goal if it crosses the goal plane before 150 time steps. Further, the regions within the goal are divided up into left, middle and right. The exact values for these boundaries once again came from the KK behavior in order to maintain consistency with the baseline. A small amount of gaussian random noise is applied to the position component of each kick data point. In general, I found that increasing the noise to each kick point yields greater coverage of the field, at the expense of decreased overall accuracy.

After the kicks are generated, the kick points points behind the goal are filtered away because they cannot be sensed by Cheesy. Finally, shots into the left and right portion of the goal are undersampled until they match the number of middle shots. Likewise, misses are undersampled until the number of kick points for misses matches the total number of shot points. This is a simple way to correct for class imbalance, a phenomenon which hurts model performance if not corrected for. Figure 3 gives a visual of the total number of kick points for each type of kick.

4.2 Models

4.2.1 Feature Sets

The following feature sets were used as input to the discriminative models:

- Global position and velocity

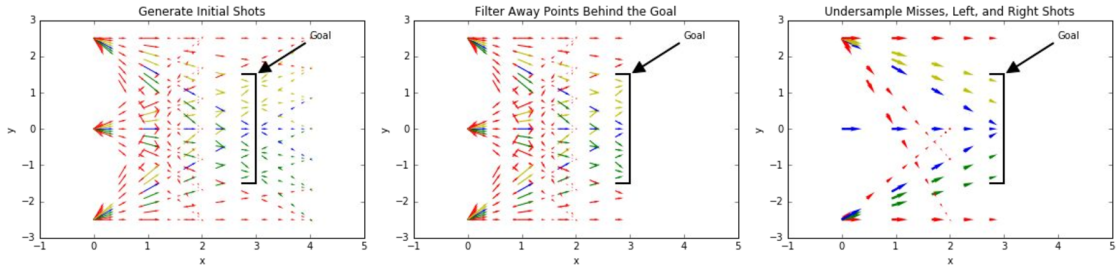

Figure 2: Illustration of the entire kick-generation pipeline. First, kicks are generated. Next, kick points behind the goal (i.e. out of Cheesy's sight) are filtered away. Finally, shots into the left and right portions of the goal, as well as misses are undersampled to achieve so that the resulting number of shots and misses are approximately the same. A time step value of 10 and a decay of 0.8 was used to generate these plots. Red indicates misses. Yellow, blue and green indicate shots into the left, middle, and right regions of the goal, respectively.

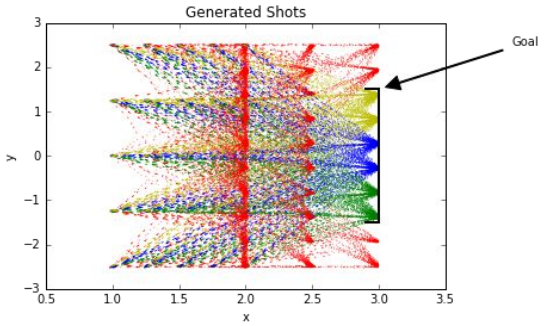

Figure 3: Full set of generated kicks. The class breakdown is as follows: 27,230 points for missed shots and 9,039, 9112, and 9017 points for shots into the left, middle, and right portions of the goal, respectively.

- Global position
- Distance and Bearing

The motivation for each of these sets is as follows: training on position and velocity is what the KK behavior uses. Training on just position is interesting because the velocity estimates tend to be quite noisy. Training on just ball distance and bearing is interesting because it throws away the kalman filter altogether.

Since it is impossible to predict shots when using the position and distance and bearing feature sets without history, models trained with these feature sets are given windows of the past 10 observations. If the ball is not seen for five consecutive frames, then the window is cleared.

4.2.2 Models

The discriminative models consist of softmax (SM), fully-connected neural network (FCNN), and recurrent neural network (RNN) classifiers. I implemented each of these models from scratch using numpy.

Each model uses a cross-entropy function to measure loss. The SM and FCNN are trained with backpropagation [6] on full batches of the training input. The RNN is trained with backpropagation through time [10] with a rollout of 10. The FCNN has a one hidden layer of size 5 with sigmoid activation units and the RNN has one hidden vector with size 30 and uses tanh nonlinearities. Each model was trained with a regularization term of 0.001. The softmax and FCNN used learning rates of 0.1 and the RNN used a learning rate of 0.005.

4.3 Evaluation

4.3.1 Simulation Evaluation

Each model was tested on the entirety of the generated kick data. Table 1 shows the resulting accuracies.

Table 1: Training accuracies of models on synthetic kick data. Raw accuracy is how many times the model predicted the correct class divided by the total number of kicks. Shot accuracy counts the model's prediction as correct if it predicts shot on a shot (e.g. predicting shot-middle when the correct label was shot-left).

	PV	PV	P	P	DB
Raw Accuracy	Raw	Shot	Raw	Shot	Raw
SM	0.547	0.547	0.616	0.648	0.774
FCNN	0.703	0.759	0.846	0.884	0.884
RNN*	0.685	0.736	0.521	0.632	0.864

Note the choice was made to not hold out a separate cross-validation/test set because of the closed-world assumption. The idea was that high performance on the kick data would translate to high performance on Cheesy. This assumption is revisited in section 4.3.2.

The FCNN consistently outperforms the softmax classifier. As seen in section 4.3.2, the SM classifier is only capable of making coarse predictions (e.g. a ball that starts left will most likely end up left), whereas the FCNN is able to make more fine-grained decisions. The increase in performance for the position and distance and bearing feature spaces is most likely attributed to the fact that they are being presented in windows.

The RNN has an asterisk because it was trained on a dataset which was an order of magnitude smaller than the dataset the SM and FCNN classifiers were trained on. This was due to the fact that RNNs take much longer to train, given its training process cannot be parallelized in any obvious way. The smaller training set consisted of roughly 1,000 kick points and each kick only lasted 10 time steps. This drastically reduced the complexity of the domain and sped up training time substantially. The RNN data set is detailed in section 4.3.5.

4.3.2 Cheesy Evaluation

Each of the models trained on each of the feature spaces were put on Cheesy and an evaluation was conducted. The evaluation consisted of *far* kicks (i.e. kicks starting off near center field) and *near* kicks (i.e. kicks starting off very near Cheesy). Figure 4 shows pictorially the trajectory of each of the shots.

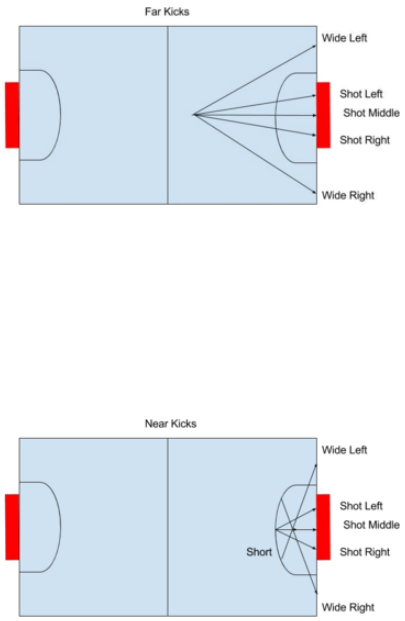

Figure 4: Trajectories of shots used during Cheesy evaluation. The kicks were grouped into *near* kicks and *far* kicks. Each kick was repeated five times. Note there is no short kick among the far kicks.

Each kick was repeated five times in an attempt to control for the variance in my kicks. I chose not to perform a short kick among the far kicks because

it was too difficult to kick the ball sufficiently close to Cheesy with good accuracy. Figure 5 shows the confusion matrices for the KK behavior on both near and far shots.

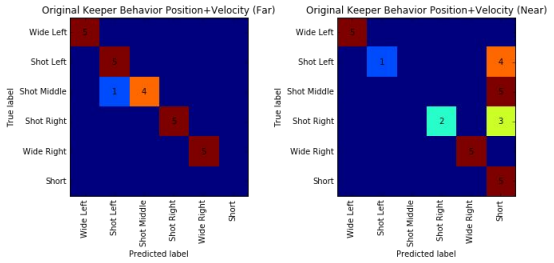

Figure 5: Confusion matrices for the KK behavior. Note there is no short kick among the far kicks; it is only included to maintain symmetry with the near kicks. Number on the diagonal indicate correct predictions by the behavior. Number on the off-diagonal indicate misclassifications.

The KK keeper does very well on long shots. However, it struggles significantly on near shots. This is due primarily to the fact the KK keeper only predicts shot if its thresholds are met for a number of consecutive frames. Additionally, since it uses velocities as estimated by a kalman filter, the ball has likely passed by by the time velocity estimates are able to "catch up".

Figure 6 shows the predictions on the synthetic kick data as well as the confusion matrices generated during the Cheesy evaluation for the SM models trained on each of the feature spaces.

The SM PV model predicts shots well as long as they are on goal, but struggles with wide kicks. The SM DB model struggles to discriminate between left, middle, and right shots at close distances. The SM P model is the best combination of the two, performing well at long and short distances.

Figure 7 shows the predictions on the synthetic kick data as well as confusion matrices for the FCNN models trained on each of the input feature spaces as well as confusion matrices generated during Cheesy evaluation.

The FCNN PV model learns a similar decision boundary to the softmax classifier, but is capable of detecting wide shots. The Like the SM-DB model, the FCNN-DB model struggles with near shots, but excels at long shots. The FCNN-P model performs the best all around, similar to the SM-P model.

Table 2 lists the accuracies for each keeper behavior. Overall, the best behaviors are the KK keeper and the FCNN-DB models. The SM-P and FCNN-P models are most consistent between far and near shots.

4.3.3 Takeaways

The SM-DB and FCNN-DB performance on near shots was the biggest disappointment out of any of the models. During my class demo, it looked as though the SM-DB and FCNN-DB models were performing the best. But that was before Cheesy had to differentiate between left, middle, and right shots. In general, it seems difficult to rely solely on distance and bearing at close distances because the window is so much smaller to make a prediction with far shots.

Surprisingly, even though the SM-P model was far less accurate than the FCNN-P model during simulation (see table 1), it does better in the Cheesy evaluation. This is perhaps due to overfitting on the FCNN-P on the training data. Using a separate held-out cross-validation/test would have determined this for sure (see section 4.3.1).

The SM-DB and FCNN-DB do well at long distances perform very well at long distances. This is partly surprising because distance and bearing are unfiltered and are presumably noisier when the ball is further away.

Because the models trained on position do well with near shots and models trained on ball distance and bearing do well with far shots, one could *ensemble* them together to achieve superior performance. A simple weighting scheme where the distance and bearing prediction is weighted proportionally with the ball distance while the position prediction is weight inversely proportional to the ball distance seems like a promising way to ensemble these two models.

In order to increase performance with these models, it is tempting to try and generate more data. One must take care with this approach as more data

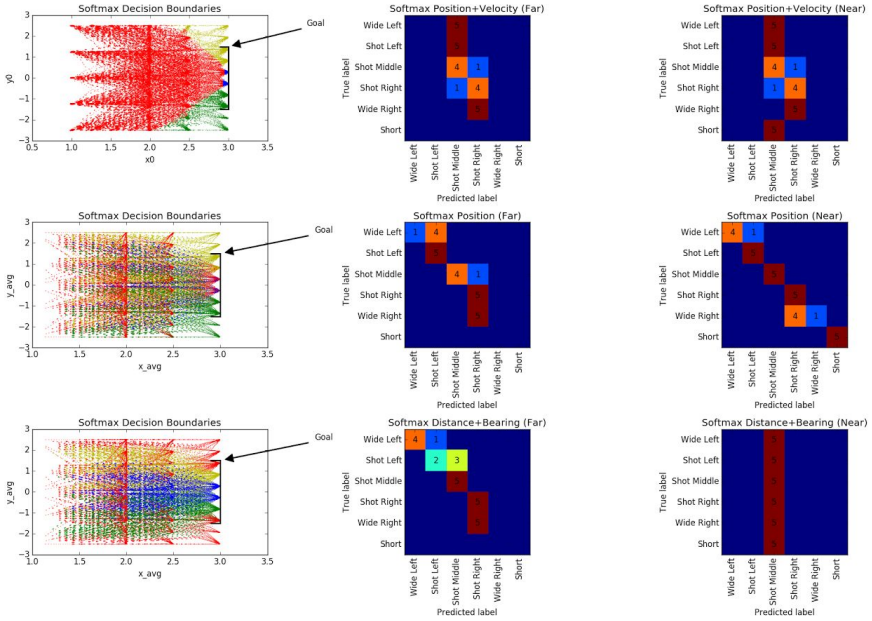

Figure 6: Predictions made by SM models on synthetic data generated in simulation (left) and confusion matrices during Cheesy evaluation (right). Red corresponds to misses and yellow, blue, and green correspond to left, middle and right shots into the goal, respectively. Numbers along the diagonal correspond to correct predictions (confusion matrices). Numbers on the off-diagonal correspond to misclassification (confusion matrices).

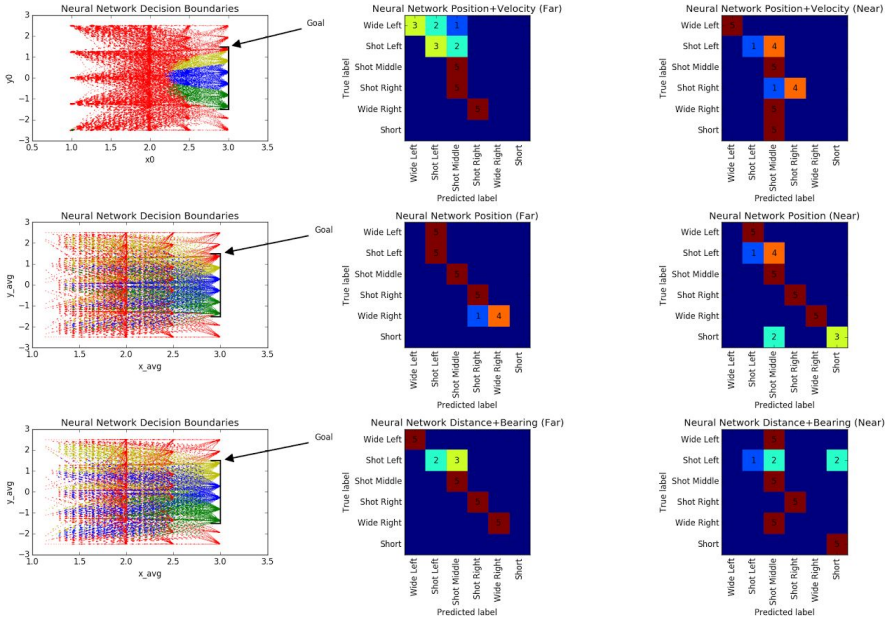

Figure 7: Predictions made by FCNN models on synthetic data generated in simulation (left) and confusion matrices during Cheesy evaluation (right). Red corresponds to misses and yellow, blue, and green correspond to left, middle and right shots into the goal, respectively. Numbers along the diagonal correspond to correct predictions (confusion matrices). Numbers on the off-diagonal correspond to misclassification (confusion matrices).

Table 2: Training accuracies of keeper behaviors during Cheesy evaluation. Raw accuracy is how many times the behavior predicted the correct class. Shot accuracy counts the behavior's prediction as correct if it predicts shot when shot (e.g. predicting shot-middle when the correct label was shot-left). SM-PV, SM-P, SM-DB corresponds to the SM keeper trained on the position and velocity, position, and distance and bearing feature sets, respectively. FCNN-PV, FCNN-P, FCNN-DB corresponds to the FCNN keeper trained on the position and velocity, position, and distance and bearing feature sets, respectively. The KK behavior predicts a shot if the average of the last three seen velocities is greater than 65cm/sec and the ball is within a distance of 75cm.

	Far Raw	Far Shot	Close Raw	Close Shot	Total Raw	Total Shot
KK	**0.960**	**1.000**	0.600	0.600	**0.764**	**0.782**
SM-PV	0.320	0.600	0.267	0.500	0.291	0.545
FCNN-PV	0.440	0.720	0.500	0.667	0.473	0.691
SM-P	0.600	0.640	**0.833**	**0.833**	0.727	0.745
FCNN-P	0.760	0.760	0.633	0.767	0.691	0.764
SM-DB	0.840	0.960	0.167	0.500	0.473	0.709
FCNN-DB	0.880	**1.000**	0.533	0.600	0.691	**0.782**

points cluster around in front of Cheesy with different labels, the more difficult it becomes to discriminate between them. A curious effect of adding misses closer to Cheesy was his reaction time slowed down substantially. This is presumably because he can not be completely sure a shot is underway as misses were seen in that same location. Another temptation is to increase the window size, but this comes at the cost of it taking more time to fill up the window, which could result in a slower reaction time.

4.3.4 False Positives

In addition to accuracy, I recorded the number of false positives each model made during Cheesy evaluation. I defined a false positive as any prediction of shot in between kicks. Cheesy had a chance to predict shot as I was rolling the ball backwards to set up for the next kick and also when the ball was sitting still, waiting to be kicked. Figure 8 shows the number of false positive for each behavior.

The KK keeper has no false positives because it never detects shots at close range. The models trained on position and velocity have a high number of false positives due to the noisy velocity estimates encountered during Cheesy evaluation. The position models do not have as many false positives and the distance and bearing models are somewhere in between.

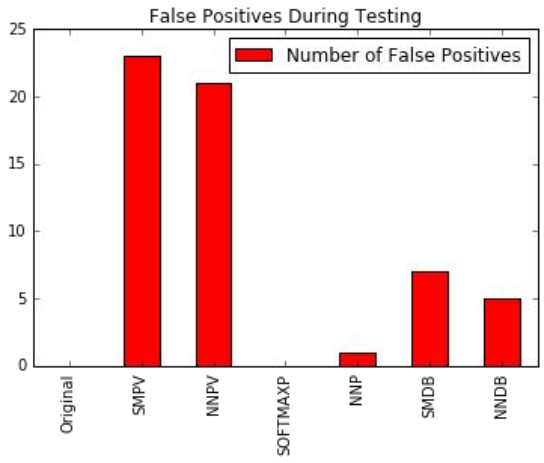

Figure 8: The number of false positives emitted by each behavior in between shots

4.3.5 Recurrent Neural Network

As stated in table 1, the RNN was trained on a much smaller training set than the SM and FCNN classifiers. Unfortunately, success high performance on

this smaller training set did not result in a usable keeper behavior on Cheesy. Figure 9 shows the kick data the RNN was trained on, as well as the predictions for each of the feature spaces.

The RNN is a very appealing model for a keeper behavior. It does not require windowed input (unlike the softmax and FCNN keepers) in the position and distance and bearing feature spaces. Given more time to train and adding an additional hidden layer in front of the softmax layer, I am confident that the RNN would achieve performance comparable to the softmax and FCNN behaviors.

5 Conclusion

5.1 Limitations

The SM, FCNN, and RNN behaviors make the assumption the keeper remains in a static position in the middle of the goal. In practice, it is often favorable for the keeper to take a more active approach and head off the attacker before the attacker can get into a good striking position. Hence adjustments would have to be made to these behaviors. As long as the keeper can localize reasonably well, these complications should not pose a large difficulty for this framework.

5.2 Future Work

The immediate future work would be to train these models for longer periods of time and do a formal hyperparameter search.

Along a more interesting line, the idea of using less and less hand-crafted features is an appealing direction for future work. Even the use of just ball distance and bearing introduces substantial noise that may prohibit the a keeper behavior from learning robust shot detection. An exciting next step forward is to remove ball distance and bearing features and rely solely on frame-space features such as the location of the ball in Cheesy's field of view, along with the joint angles of Cheesy's head. The final step would be foregoing even these features and going straight from vision to shot prediction.

5.3 Recap

I implemented keeper behaviors that learn functions for detecting shots on goal. I generated kicks in simulation, used this data to train discriminative models, implemented these models on Cheesy, and evaluated them against the KK keeper. I found models trained on position and ball distance and bearing features alone to be competitive with the kalman keeper behavior and their ensemble to likely outperform the KK keeper. Such work is one step towards moving away from hand-crafting behavior thresholds and taking a more data-driven approach to the robot soccer domain.

References

[1] Pieter Abbeel, Adam Coates, Michael Montemerlo, Andrew Y Ng, and Sebastian Thrun. Discriminative training of kalman filters. In *Robotics: Science and Systems*, pages 289–296, 2005.

[2] Alon Farchy, Samuel Barrett, Patrick MacAlpine, and Peter Stone. Humanoid robots learning to walk faster: From the real world to simulation and back. In *Proceedings of the 2013 international conference on autonomous agents and multi-agent systems*, pages 39–46. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[3] Adrian Ratter, Bernhard Hengst, Brad Hall, Brock White, Benjamin Vance, David Claridge, Hung Nguyen, Jayen Ashar, Stuart Robinson, and Yanjin Zhu. runswift team report 2010 robocup standard platform league. *University of New South Wales, Australia*, 2010.

[4] F Riccio, F Patota, F Bella, E Borzi, D De Simone, V Suriani, L Iocchi, and D Nardi. Spqr robocup 2015 standard platform league team description paper. 2015.

[5] Thomas Röfer, Tim Laue, Judith Müller, Michel Bartsch, Malte Jonas Batram, Arne Böckmann,

Martin Böschen, Martin Kroker, Florian Maaß, Thomas Münder, et al. B-human team report and code release (2013), 2010.

[6] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5:3, 1988.

[7] Peter Stone, Kurt Dresner, Selim T Erdogan, Peggy Fidelman, Nicholas K Jong, Nate Kohl, Gregory Kuhlmann, Ellie Lin, Mohan Sridharan, Daniel Stronger, et al. The ut austin villa 2003 four-legged team. In *in RoboCup-2003: Robot Soccer World Cup VII*. Citeseer, 2004.

[8] Peter Stone, Kurt Dresner, Peggy Fidelman, Nicholas K Jong, Nate Kohl, Gregory Kuhlmann, Mohan Sridharan, and Daniel Stronger. *The UT Austin Villa 2004 RoboCup four-legged team: Coming of age*. Computer Science Department, University of Texas at Austin, 2004.

[9] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[10] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

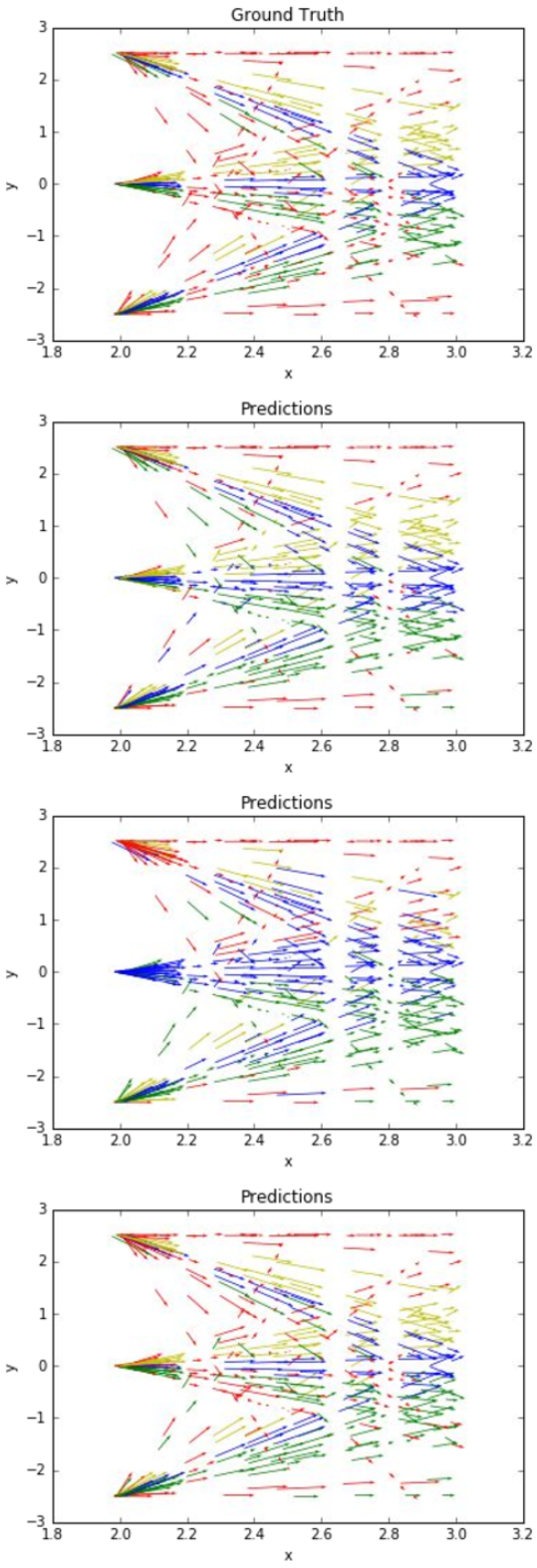

Figure 9: From top to bottom: the ground truth kick data used to train the RNN; predictions by the RNN when trained in the position and velocity, position, and distance and bearing feature spaces, respectively. Each shot consists of 10 time steps, and a decay of