Understanding responses to environments for the Prisoner's Dilemma: A machine learning approach

Nikoleta E. Glynatsi

Month 2020

Submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy.

School of Mathematics
Ysgol Mathemateg

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Mae-

cenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus.

Acknowledgements

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Dissemination of Work

Publications (4 Published & 5 In preparation)

1. 2018: **Reinforcement learning produces dominant strategies for the Iterated Prisoner's Dilemma.** Marc Harper, Vincent Knight, Martin Jones, Georgios Koutsovoulos, Nikoleta E. Glynatsi, Owen Campbell - PLOS One - Preprint arXiv:1707.06307

2. 2018: **An evolutionary game theoretic model of rhino horn devaluation.** Nikoleta E. Glynatsi, Vincent Knight, Tamsin Lee. Ecological Modelling - Preprint arXiv:1712.07640

3. 2017: **Evolution reinforces cooperation with the emergence of self-recognition mechanisms: an empirical study of the Moran process for the Iterated Prisoner's dilemma.** Vincent Knight, Marc Harper, Nikoleta E. Glynatsi, Owen Campbell - PLOS ONE - Preprint arXiv:1707.06920

4. 2016: **An open framework for the reproducible study of the Iterated prisoner's dilemma.** Vincent Knight, Owen Campbell, Marc Harper et al - Journal of Open Research Software

In preparation

1. 2019: **A meta analysis of tournaments and an evaluation of performance in the Iterated Prisoner's Dilemma.** Nikoleta E. Glynatsi and Vincent A. Knight - In preparation to be submitted - Preprint arXiv:2001.05911

2. 2019: **A bibliometric study of research topics, collaboration and influence in the field of the Iterated Prisoner's Dilemma.** Nikoleta E. Glynatsi and Vincent A. Knight - In preparation to be submitted - Preprint arXiv:1911.06128

3. 2019: **Game Theory and Python: An educational tutorial to game theory and repeated games using Python.** Nikoleta E. Glynatsi and Vincent A. Knight - Submitted to the Journal of Open Source Education - Available on GitHub Nikoleta-v3/Game-Theory-and-Python

4. 2019: **A theory of mind: Best responses to memory-one strategies. The limitations of extortion and restricted memory.** Nikoleta E. Glynatsi and Vincent A. Knight - Submitted to Scientific Reports Nature - Preprint arXiv:1911.12112

5. 2019: **Recognising and evaluating the effectiveness of extortion in the Iterated Prisoner's Dilemma.** Vincent Knight, Marc Harper, Nikoleta E. Glynatsi, Jonathan Gillard - Submitted to Nature Communications - Preprint arXiv:1904.00973

Talks & Posters

Invited Talks (Keynotes)

- How does a smile make a difference?, PyCon UK, Cardiff, 2018.
- The Fallacy of Meritocracy, PyCon Balkan, Belgrade, 2019.

Others

- Accessing open research literature with Python - PyCon Namibia, Namibia 2017.
- Writing tests for research software - PyCon Namibia, Namibia 2017.
- Optimisation of short memory strategies in the Iterated Prisoners Dilemma - Wales Mathematics Colloquium, Gregynog Hall 2017.
- PIP INSTALL AXELROD (**poster**) - Euroscipy, Erlangen Germany 2017.
- Arcas: Using Python to access open research literature - Euroscipy, Erlangen Germany 2017.
- A trip to earth science with python as a companion - PyConUK, Cardiff 2017.
- The power of memory (**poster**) - SIAM UKIE Annual Meeting, Southampton 2018.
- Rhinos with a bit of Python - PyConNA, Namibia 2018.
- Memory size in the Prisoners Dilemma - Wales Mathematics Colloquium, Gregynog Hall 2018.
- Memory size in the Prisoners Dilemma - SIAM UKIE National Student Chapter, Bath University 2018.
- Stability of defection, optimisation of strategies and testing for extortion in the Prisoner's Dilemma (**poster**) - STEM for Britain, London 2019.
- Stability of defection, optimisation of strategies and testing for extortion in the Prisoner's Dilemma - 18th International Conference on Social Behaviour, Sedona, Arizona 2019.
- An introduction to Time Series - Joint workshop between CUBRIC & Mathematics Departments, Cardiff 2019.

Software Development

- Arcas, an open source package designed to help users collect academic articles' metadata from various prominent journals and pre print serves. **Contribution:** Main developer
- Axelrod-Python library, an open source framework decided to the study of the Iterated Prisoner's Dilemma. **Contributions:** Implementation of spatial tournaments functionality, implementation/addition of strategies (from the literature) to the library, reviewing of code contributed by other contributors
- SymPy, a Python library for symbolic mathematics. **Contributions:** Implementation of Dixon's and Macaulay's resultants which were developed for my 2019 publication "Sta-

bility of defection, optimisation of strategies and the limits of memory in the Prisoner's Dilemma"

- Pandas, an open source library providing high-performance, easy-to-use data structures and data analysis tools. **Contribution:** Fix bug which converted NaN values to strings

Contents

List of Figures

List of Tables

Chapter 1

Training long short-term memory networks to play the Iterated Prisoner's Dilemma

The research reported in this Chapter has been carried out with:

Axerod-Python library version: 4.2.0
Associated data set:

1.1 Introduction

In Chapter **??** it was mentioned that consepsualating and introducing new strategies has been an important aspect of research to the field. The aim of this Chapter is to introduce new IPD strategies based on an archetype that has not received much attention in the literature.

In Chapter **??** it was concluded that one of the properties successful strategies in a IPD competition need to have is cleverness/complexity. Complexity can confer to adaptability, and adaptability is important in performing well in diverse set of environments. This was established not only in Chapter **??** but also from the results of Chapter **??**. The set of complex strategies that ranked highly across distinct tournaments in Chapter **??** included strategies based on archetypes such as finite state automata, hidden Markov models and *artificial neural networks* (ANNs).

ANNs have successfully been trained to play games exterior to the IPD, such as checkers [4] and chess [8]. The usage of feed forward ANNs in the IPD was firstly introduced in [11], and have been used ever since [1, 2, 7, 9]. Possibly, the most successful ANNs strategies are the one introduced in [10], as they were ranked 7^{th}, 9^{th}, and 11^{th} in a tournament of 223 strategies.

A type of ANNs that have not received much attention in the literature are the *recurrent neural*

networks (RNNs). RNNs are a type of neural networks that include a feedback connection and are designed to work with inputs in the form of sequences. RNNs were firstly considered as an archetype in 1996. In [21] a RNN which considered a single previous step as an input was trained via a Q learning algorithm to play against a single strategy. The results of [21] were limited only in the network learning to play against a single strategy.

The limitations of [21] could potentially have been due to the limitations of the RNNs themselves. As it will be discussed later the RNNs quickly became unable to learn due to the vanishing gradient problem. To improve on the standard recurrent networks a new network called the long short-term memory networks (LSTMs) were introduced in [12]. LSTMs do not suffer from the vanishing gradient, can be successfully trained and are being used in a number of innovative applications such as time series analysis [15], speech recognition [20] and prediction in medical care pathways [22].

LSTMs have not received attention in the IPD literature. This Chapter trains and introduced a number of strategies based on LSTMs. The training has been possible due to the collection of best response sequences generated in Chapter **??**. The performance of the new strategies is evaluated and compared in a meta tournament analysis of standard tournaments. The results demonstrate

This Chapter is structured as follows:

- section 1.2, presents an introduction to artificial, recurrent and long short-term memory networks.
- section 1.3, cover the specifics of the LSTMs trained in ths Chapter. Their architectural details are presented as well as the different training sets they have been trained on.
- section 1.4, evaluates and compares the performance of 18 LSTMs based strategies in 300 standard IPD computer tournaments.

1.2 Artificial, recurrent neural and long short-term memory networks

ANNs are computing systems vaguely inspired by the biological neural networks that constitute animal brains. As stated in [13] the research on ANNs has experienced three periods of extensive activity. The first peak was in the 1940s. The work of [17] opened the subject by creating a computational model for neural networks. The second peak occurred in 1960s with the introduction of the perceptron [19]. The perceptron is a linear binary classifier and in the 1960s in was shown that it could be used to learn to classify simple shapes with 20×20 pixels input. However, it was impossible for the perceptron to be extended to classification tasks with many categories. The limitations of the model were presented in [18] which alted the enthusiasm of most researchers in ANNs, resulting in no activity in field for almost 20 years. The third peak occurred in the early 1980s with the introduction of the back propagation algorithm for multilayered networks [23]. Though the algorithm was initially proposed by [23] it has been reinvented several times and it was popularizes by [16]. Following the introduction of the algorithm and the ability to now train more complex models, ANNs have received considerable interest, and have been used in a number of innovative applications such as [6, 14].

ANNs based on the connection pattern/architecture can be grouped into two categories:

- Feed forward networks.
- Recurrent or feedback networks.

ANNs can be viewed as weighted directed graphs in which artificial neurons are nodes and directed edges, with weights, are connections between the neuron outputs and the neuron inputs [13]. In feed forward networks the graphs have no loops whereas in recurrent loops occur because of feedback connections. A graphical representation of a feed forward network with a single hidden layer is given by Figure 1.1.

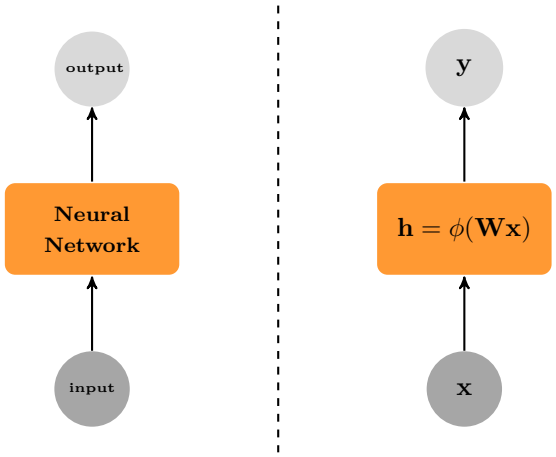


Figure 1.1: Graphical representation of an ANN.

A feed forward network is composed by an input layer, hidden layers, and an output layer. The networks' input is a vector x. The dimensionality, or number of nodes of the input layer is depended on the dimensionality of x. Each element of the input vector is connected to the hidden layer via a set of learned weights. The hidden layer also consists of nodes and the number of node/dimensionality of it is an architectural decision. The j^{th} hidden neuron outputs,

$$h_j = \phi(\sum_i w_{ij} x_i), \text{where } \phi \text{ is an activation function.}$$

Common activation functions for ϕ include:

- the sigmoid function $\sigma(x)$ which squashes numbers into the range $[0, 1]$
- the hyperbolic tangent, $tanh(x)$ which squashes numbers into the range $[-1, 1]$
- the rectified linear unit, $ReLU(x) = max(0, x)$

In turn, the hidden layer is fully connected to an output layer. The j^{th} output neuron outputs,

$$y_j = \sum_i v_{ij} h_i.$$

The output layer transforms the raw scores to probabilities via a softmax function.

Feed forward networks make predictions using forward propagation,

$$h = \phi(Wx) \tag{1.1}$$

$$y = \text{softmax}(Vh) \tag{1.2}$$

where W is a weight matrix connecting the input and hidden layers, V is a weight matrix connecting the hidden and output layers. W and V are the learning parameters of the network. Their dimensionality depends on the dimensionality of networks layers. If x and x are 2 dimensional and the hidden layer had 500 nodes then $W \in \mathbb{R}^{2\times200}$ and $V \in \mathbb{R}^{2\times200}$. Increasing the dimensionality of the hidden layer corresponds to more parameters.

Training a ANN corresponds to finding the learning parameters that minimise the error on the training data. The function that measures error is called the *loss function*. A common loss function with the choice of a softmax output is the *cross entropy* loss. Consider N training examples and K classes, the function for a prediction $\hat{y}$ with respect to the true output y us give by,

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{n\in N}\sum_{k\in K} y_{n,k}\log(\hat{y_{n,k}})$$

The minimum of the loss function is calculated using the gradients decent algorithm. The gradient decent algorithm relies on the gradients. The gradients are the vector of derivatives of the loss function with respect to the learning parameters. Thus, $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial V}$. These gradients are calculated with the back propagation algorithm [24], which is a way to efficiently calculate the gradients starting from the output.

An extension of feed forward networks are the recurrent networks. RNNs are capable of processing variable length sequences of inputs. More importantly, RNNs use their internal state/knowledge to a make a prediction on the input at time t based on the decisions made at the previous time steps. A graphical representation of a RNN is given by Figure 1.2. RNN are networks with loops. The loop represents that information can be passed down from one step of the network to the next. In fact they can be thought of as multiple copies of the same network, each passing a message to a successor, as demonstrated by Figure 1.2.

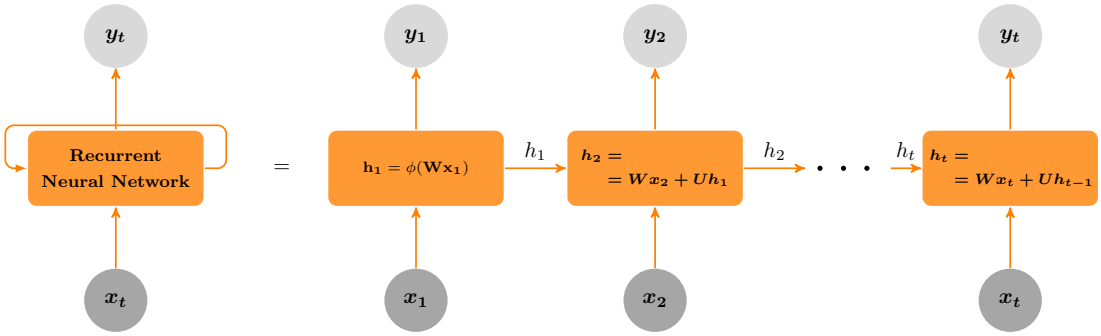


Figure 1.2: Graphical representation of a RNN.

The hidden layers pass down information by considering,

$$h_t = \phi(Wx_t + Uh_{t-1})$$

where h_{t-1} is the hidden state computed at time $t-1$ multiplied by some weight vector U. In matrix notation RNNs are described by,

$$h_t = \phi(Wx_t + Uh_{t-1}) \tag{1.3}$$

$$y_t = Vh \tag{1.4}$$

Unfortunately, in practice RNNs quickly become unable to learn to connect the information. The more time steps the higher the chance that the back propagation gradient either accumulate and explode or vanish down to zero. The fundamental difficulties of RNNs were explored in depth by [3].

A network specifically designed to avoid the long-term dependency problem. was introduced by [12] called the long short-term memory network (LSTM). LSTMs work tremendously well on a large variety of problems, and are now widely used. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn.

The core idea to LSTMs is the *cell state* also refereed to as the long term memory, denote as C_t. The cell state is designed to pass down information with only a few carefully regulated changes being applied to it by structures called *gates*. Gates are composed out of a sigmoid neural net layer and a point wise multiplication operator. In order to explain the cell gate and subsequently how LSTMs make predictions consider a LSTM's hidden layer at time step t give by Figure 1.3.

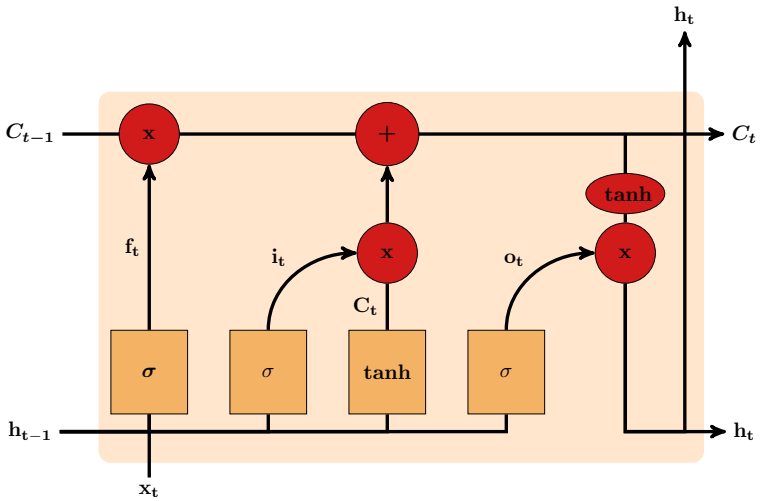


Figure 1.3: An LSTM hidden layer at time step t.

The cell state and hidden state from from the previous time step are feed back into the network. Initially, the network decides what information from the previous cell state is to be discarded. This decision is made by the *forget state*. The forget state considers the hidden state at time

$t-1$ and the input at time t,

$$f_t = \sigma(W_f x_t + U_f h_{t-1}) \tag{1.5}$$

Secondly, the network decides what information is going to be stored at the cell gate. There are two parts to this. Firstly, the input gate decides which values are going to be updated,

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i). \tag{1.6}$$

and secondly, a tanh layer creates a vector of candidate values denoted as $\tilde{C}_t$ that could be added to the cell state. $\tilde{C}_t$ is given by Equation (1.9).

$$\tilde{C}_t = tanh(W_c x_t + U_c h_{t-1} + b_c). \tag{1.7}$$

The cell state C_{t-1} is multiplied by f_t, forgetting the values which have been decided to discard. The new candidate values are scaled by how much information has been decided to keep from the input. Thus,

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t. \tag{1.8}$$

The LSTM outputs a hidden state at each time step. This is either used to through an activation function to output y_t, and it is passed to the next time step cell. The output is based on the current cell state. The cell state goes through a tanh function and it is multiplied by a sigmoid gate that decides which parts to output from the cell state.

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \tag{1.9}$$

$$h_t = o_t * tanh(C_t) \tag{1.10}$$

This process is being carried out though each time step of the input sequence. At each the cell state and hidden state are feed back into the network and the hidden state can also be used to make a prediction for each time step as demonstrated by Figure 1.4.

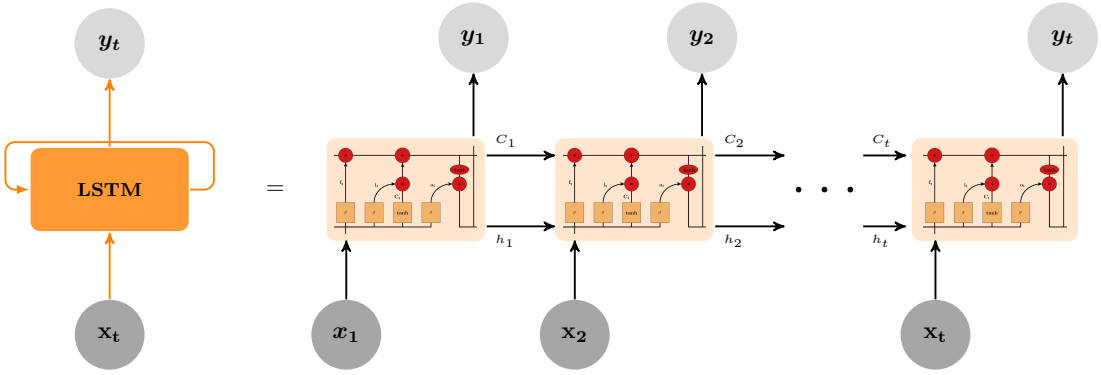


Figure 1.4: Graphical representation of an LSTM network.

The LSTM is a complex architecture and contains a large number of parameters that need to be trained. The LSTM networks have managed to achieve remarkable results. The training of an LSTM model in this Chapter is carried with the open source project Keras. The details for the training are presented in section 1.3.

1.3 Training LSTM networks to play the Iterated Prisoner's Dilemma

The LSTMs of this Chapter are trained based on the best response sequence data set that was generated in Chapter **??**. Consider the strategy Adaptive and a best response to it as presented in section **??**.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	...	**204**	**205**
Adaptive	C	C	C	C	C	C	D	D	D	D	D	C	...	C	C
S^*	C	C	D	D	D	D	D	D	D	D	D	D	...	D	D

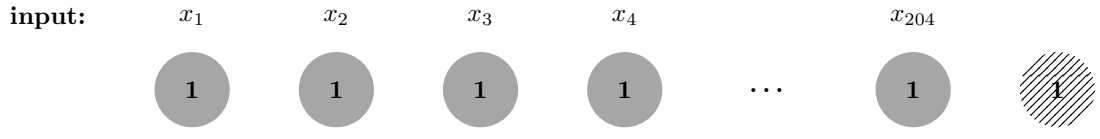
Table 1.1: The interactions between Adaptive and one of the best response sequences to the strategy.

For $0 \to D$ and $1 \to C$ Table 1.1 is given by,

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	...	**204**	**205**
Adaptive	1	1	1	1	1	1	1	0	0	0	0	1	...	1	1
S^*	1	1	0	0	0	0	0	0	0	0	0	0	...	0	0

Table 1.2: The interactions between Adaptive and one of the best response sequences to the strategy were $0 \to D$ and $1 \to C$.

In order to train the LSTM as a player the network should predict the best response to strategy's action. Thus, the input to the network is the actions of the opponents Q, discarding the last turn.

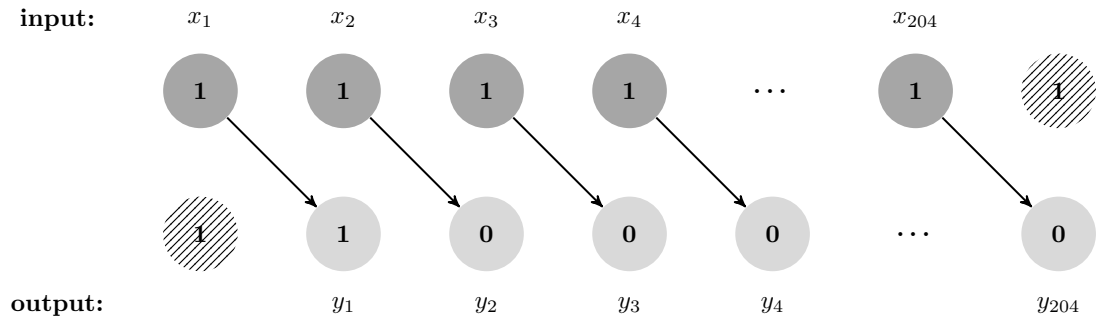


and the expected output it's the response of the S^* to those actions. Thus the genes from $t = 2$ instead of $t = 1$.

Two variants of LSTM network have been training in this Chapter. These are refereed to:

- sequence to sequence
- sequence to probability

The sequence to sequence model as an input of actions a each time step by a strategy an it outputs a response for each at each time step. The sequence to probability has an input the action of strategy up to time t, and outputs a response to the history at time t.



A graphical representation of the two networks are given by Figures 1.5 and 1.6.

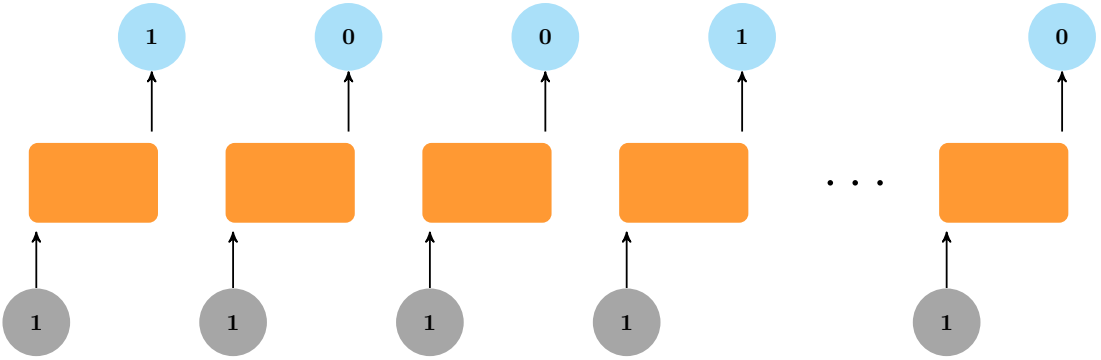


Figure 1.5: A graphical representation of a sequence to sequence LSTM model

.

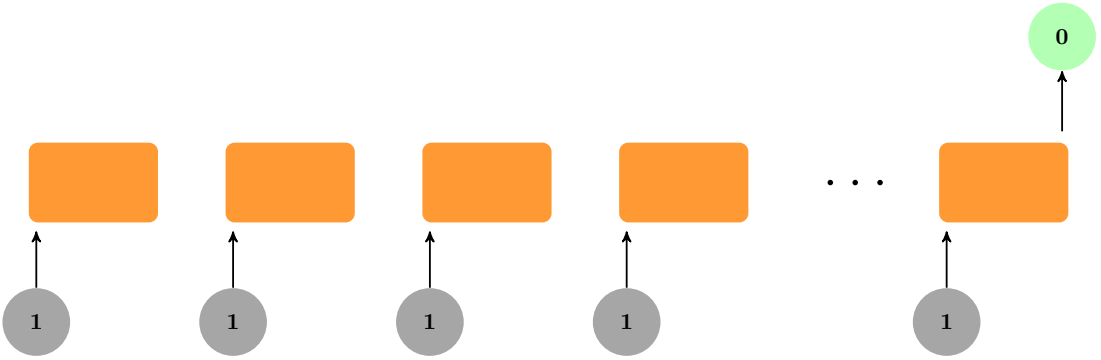


Figure 1.6: A graphical representation of a sequence to probability LSTM model

.

The networks are constructed in such way as the length of sequences they receive it is not fixed. The are not trained for the 5258 but each best response sequence is broken down to 204 sets

of input.

$$\begin{bmatrix} 1 & & & & & & \\ 1 & 1 & & & & & \\ 1 & 1 & 1 & & & & \\ 1 & 1 & 1 & 1 & & & \\ 1 & 1 & 1 & 1 & 1 & & \\ 1 & 1 & 1 & 1 & 1 & 1 & \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \rightarrow \begin{bmatrix} 1 & & & & & & \\ 1 & 1 & & & & & \\ 1 & 1 & 0 & & & & \\ 1 & 1 & 0 & 0 & & & \\ 1 & 1 & 0 & 0 & 0 & & \\ 1 & 1 & 0 & 0 & 0 & 0 & \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{1.11}$$

$$\begin{bmatrix} 1 & & & & & & & \\ 1 & 1 & & & & & & \\ 1 & 1 & 1 & & & & & \\ 1 & 1 & 1 & 1 & & & & \\ 1 & 1 & 1 & 1 & 1 & & & \\ 1 & 1 & 1 & 1 & 1 & 1 & & \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \tag{1.12}$$

1.3.1 Building the networks with Keras

To construct and train the network model the open-source neural-network package Keras [5] is used. Its library written in Python, designed to enable fast experimentation with neural networks. Neural layers, cost functions, optimizers, initialization schemes, activation functions, and regularization schemes are all standalone modules that you can combine to create new models.

The code for implementing the sequence to sequence model is given by Figure 1.7. There are two main types of models available in Keras: the Sequential model, and the Model class. The Sequential model is a linear stack of layers. There is a single layer of an LSTM model that returns the hidden state at each time step. The input shape is (None, 1) because there is not a fixed length and there is a single time step between the of the input. There are 100 the outputs turn into a probability with a sigmoid . The dropout layer There are a total of 41301 parameters that need to be trained for the sequence to sequence model.

Similarly, for the sequence to probability network the implementation of the network using Keras is given by Figure 1.8. There are now two LSTM layers. The first one outputs a hidden state at each time step and the second layer outputs only at time t the output then goes

```
>>> from keras.models import Sequential

>>> from keras.layers import (
...     Dense,
...     Dropout,
...     CuDNNLSTM,
... )

>>> num_hidden_cells = 100
>>> drop_out_rate = 0.2

>>> model = Sequential()

>>> model.add(
...    CuDNNLSTM(
...        num_hidden_cells, return_sequences=True, input_shape=(None, 1))
...)

>>> model.add(Dropout(rate=drop_out_rate))

>>> model.add(Dense(1, activation="sigmoid"))
>>> model.summary()
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
cu_dnnlstm_1 (CuDNNLSTM)     (None, None, 100)         41200
_________________________________________________________________
dropout_1 (Dropout)          (None, None, 100)         0
_________________________________________________________________
dense_1 (Dense)              (None, None, 1)           101
=================================================================
Total params: 41,301
Trainable params: 41,301
Non-trainable params: 0
_________________________________________________________________
```

Figure 1.7: Keras sequence to sequence.

through Dropout then turned into probability with a sigmoid function. There are total of 122101 parameters.

```
    >>> from keras.models import Sequential
>>> from keras.layers import (
...     Dense,
...     Dropout,
...     CuDNNLSTM,
... )

>>> num_hidden_cells = 100
>>> drop_out_rate = 0.2

>>> model = Sequential()

>>> model.add(
...     CuDNNLSTM(num_hidden_cells, return_sequences=True, input_shape=(None, 1))
... )

>>> model.add(CuDNNLSTM(num_hidden_cells))
>>> model.add(Dropout(rate=drop_out_rate))

>>> model.add((Dense(1, activation="sigmoid")))
>>> model.summary()
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
cu_dnnlstm_2 (CuDNNLSTM)     (None, None, 100)         41200
_________________________________________________________________
cu_dnnlstm_3 (CuDNNLSTM)     (None, 100)               80800
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 122,101
Trainable params: 122,101
Non-trainable params: 0
_________________________________________________________________
```

Figure 1.8: Keras sequence to probability.

Keras contains two implementations on an LSTM model. That is the class LSTM and the CCNLSTM which was used here. CCNLSTM is a fast LSTM implementation with NVIDIA CUDA Deep Neural Network library (cuDNN) which run on a graphics processing unit (GPU).

1.3.2 Training on GPU

Conventionally most training and running computer code happens on the central processing unit (CPU). A CPU also called a central processor or main processor is essentially the brain of any computing device, carrying out the instructions of a program by performing control, logical, and input/output (I/O) operations. CPUs, can have a number of cores, and its basic design and purpose—to process complex computations—has not really changed. A CPU core is designed to support an extremely broad variety of tasks.

A graphical processing unit (GPU), on the other hand, has smaller-sized but many more logical cores (arithmetic logic units or ALUs, control units and memory cache) whose basic design is to process a set of simpler and more identical computations in parallel.

Initially designed mainly as dedicated graphical rendering workhorses of computer games, GPUs were later enhanced to accelerate other geometric calculations (for instance, transforming polygons or rotating verticals into different coordinate systems like 3D). Nvidia created a parallel computing architecture and platform for its GPUs called CUDA, which gave developers access and the ability to express simple processing operations in parallel through code.

While a CPU core is more powerful than a GPU core, the vast majority of this power goes unused by ML applications. whereas a GPU core is optimized exclusively for data computations. Because of this singular focus, a GPU core is simpler and has a smaller die area than a CPU, allowing many more GPU cores to be crammed onto a single chip. Consequently, ML applications, which perform large numbers of computations on a vast amount of data, can see huge (i.e., 5 to 10 times) performance improvements when running on a GPU versus a CPU.. As many have said GPUs are so fast because they are so efficient for matrix multiplication.

Keras CCNLSTM class makes running on GPU easy. Cardiff's super computer was used for the training of the models.

1.3.3 Training data sets

It was explained in the the best response data set from Chapter 6 is used as training data. Moreover, each best responses has broken down to 204 different inputs. In order to understand the difference the effect of the training data the models have been trained on different subsets of the data set. More specifically in total of four different sub sets. These with a brief explanation and number are given by Table 1.3.

Data set	number of Q	Explanation	number of S^*
all strategies	192	The data set as presented in Chapter ??	5258
top performing strategies	18	Top 18 performing strategies in a standard tournaments performed with APL	714
strategies across ranks	15	A set 15 that there rank is across	212
basic strategies	11	A set 11 which are classified as the basic in APL	84

Table 1.3: Subset of data set that the networks are trained on. The APL project performs a round robin tournament with all the strategies implemented on every release. The top strategies and the strategies across are based on the results of the tournament with APL version 4.5.0.

Due to the different size of the data set the networks have trained for a different number of epochs. The number of epochs is the number of complete passes through the training data set. The number of epochs each network has trained for each data set is given by Table 1.4.

	all strategies	top strategies	strategies across ranks	basic strategies
sequence to sequence	19999	83999	39999	129999
sequence to probability	7999	74999	39999	84999

Table 1.4: Number of epochs each both networks, sequence to sequence and sequence to probability, were trained for each subset.

The data are split into training and test training set. The test set is 20% of the data. The validation and loss is calculated at each epoch. The validation is the measure that. The loss is based on the.

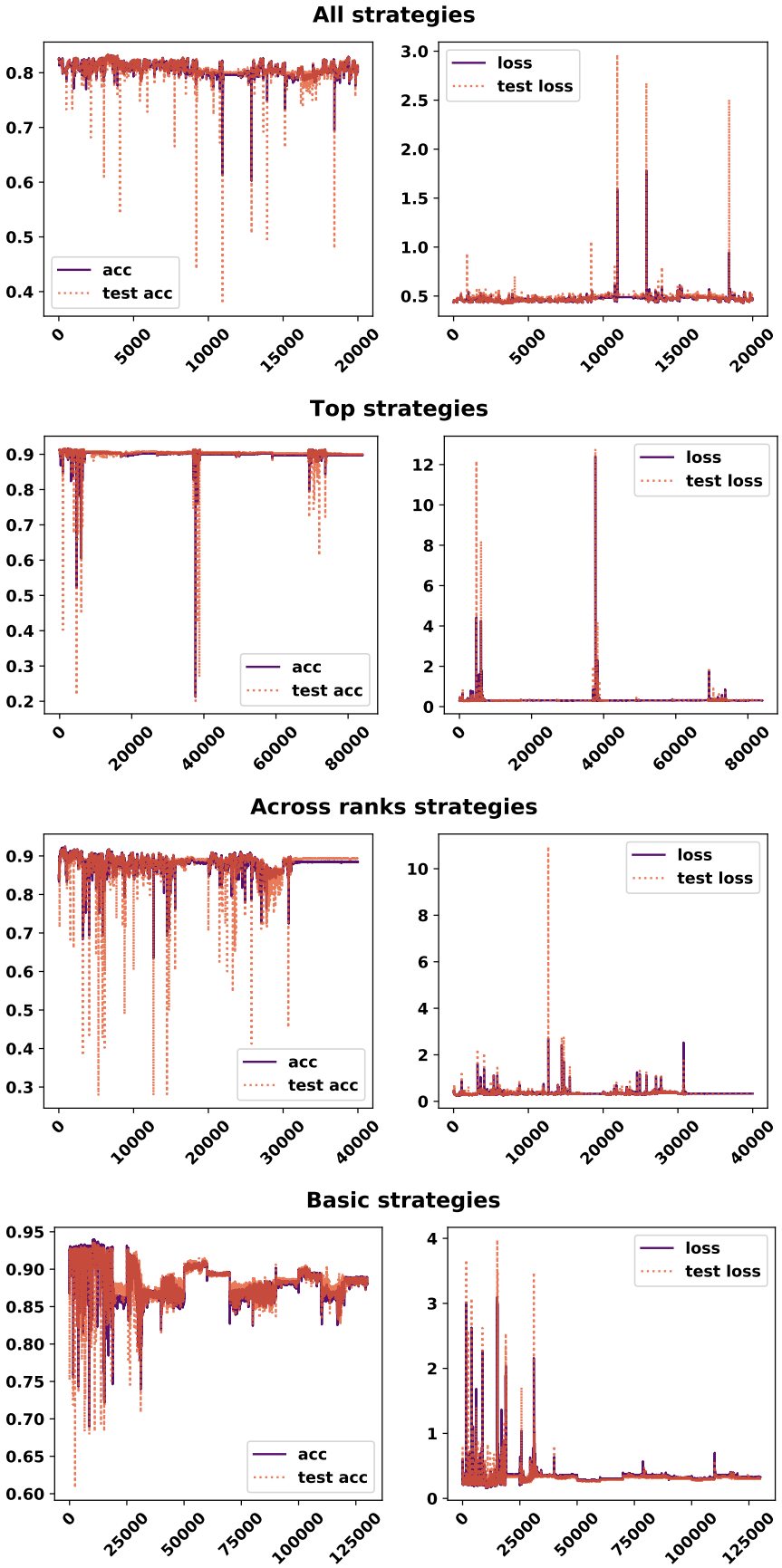


Figure 1.9: Validation of sequence to sequence

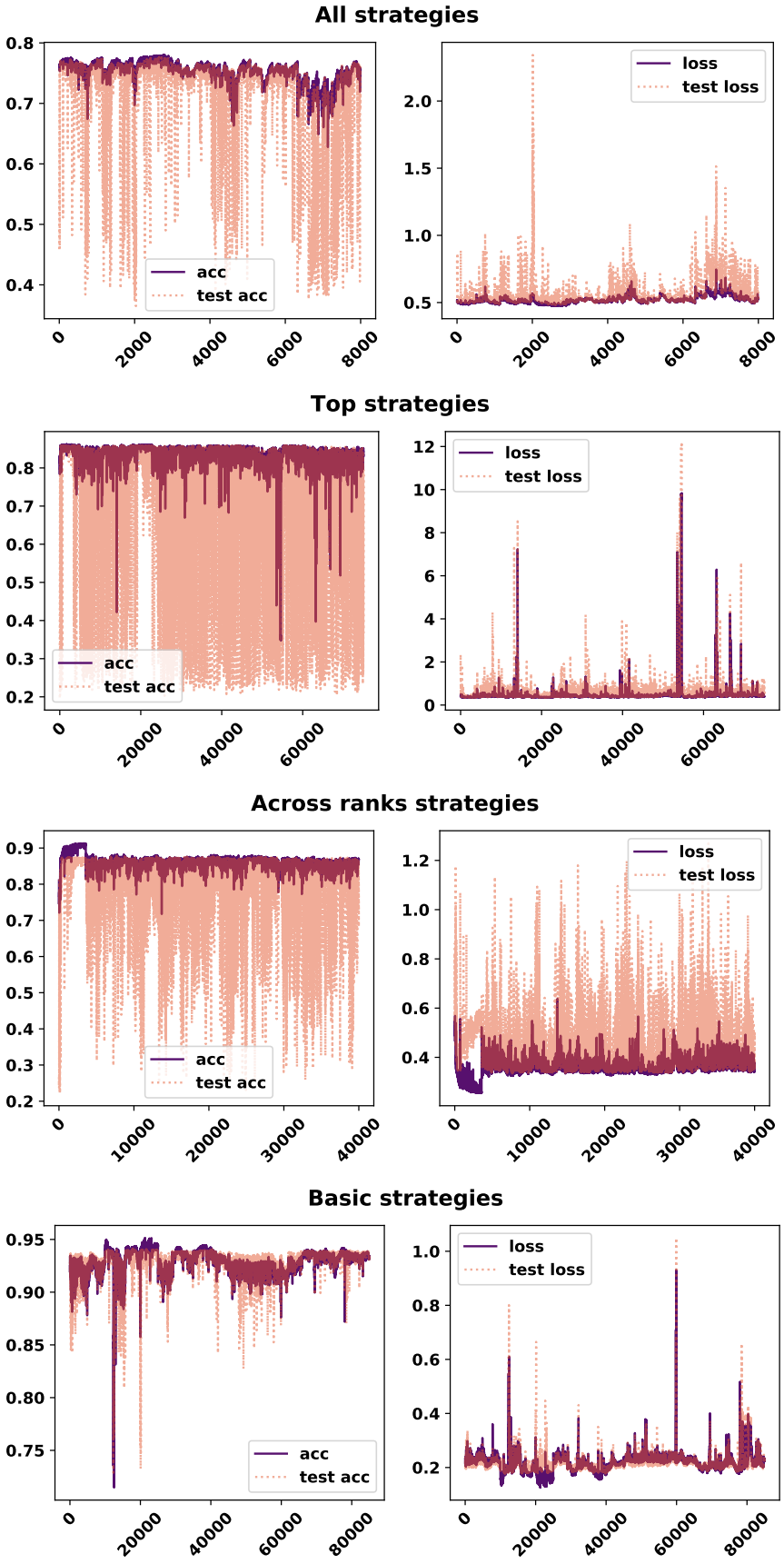


Figure 1.10: Validation of sequence to probability

Discussion on validation. In section 1.4 the LSTM networks are validated by as an IPD strategy using a meta tournament analysis.

1.4 Validation of LSTM base strategies using a meta tournament analysis

To evaluate the LSTM players as strategies, a strategy class was implemented that takes as an argument the a Keras model. The player then uses the history of the match translate it to an input in the form that the networks can then translate into an action. Note that the network was not trained to consider it's opening move. The opening move is also an input to the strategy class. In the meta tournament analysis different strategies that have a different opening play are considered.

```
import numpy as np

import axelrod as axl
from axelrod.random_ import random_choice
from keras.layers import LSTM, Dense, Dropout
from keras.models import Sequential

C, D = axl.Action.C, axl.Action.D


class LSTMPlayer(axl.Player):
    name = "The LSTM player"
    classifier = {
        "memory_depth": float("inf"),
        "stochastic": True,
        "inspects_source": False,
        "manipulates_source": False,
        "manipulates_state": False,
    }

    def __init__(self, model, reshape_history_funct, opening_probability=0.78):
        self.model = model
        self.opening_probability = opening_probability
        self.reshape_history_function = reshape_history_funct
        super().__init__()
        if opening_probability in [0, 1]:
            self.classifier["stochastic"] = False

    def strategy(self, opponent):
        if len(self.history) == 0:
            return random_choice(self.opening_probability)

        history = [action.value for action in opponent.history]
        prediction = float(
            self.model.predict(self.reshape_history_function(history))[0][-1]
        )

        return axl.Action(round(prediction))

    def __repr__(self):
        return self.name
```

The implemented class can interact with any opponent from APL. An example of usage of the LSTMPlayer is given by Figure.

To evaluate the performance of the LSTM strategy a meta tournament analysis, similar to Chapter **??**, is carried out. The tournament type considered here are standard tournaments.

The data collections is given by Algorithm **??**. At each trial a number of opponents, a list of opponent From 5 to 10. The number of turns and repetitions are fixed to 200 and 50.

The number 205 was hard coded. The evaluation does not want to take into account the last turns. 205 was selected so the last turns could have beed discarded and the match length would be 200 which is the number of turns commonly used in the literature.

Algorithm 1: Data collection Algorithm

foreach *seed* $\in [0, 300]$ **do**

$N \leftarrow$ randomly select integer $\in [N_{min}, N_{max}]$;
players $\leftarrow$ randomly select N players;
players $\leftarrow$ players + LSTM strategy;
$N \leftarrow N + 1$;
$k \leftarrow 50$;
$n \leftarrow 200$;

result standard $\leftarrow$ Axelrod.tournament(players, n, k);

return *result standard*;

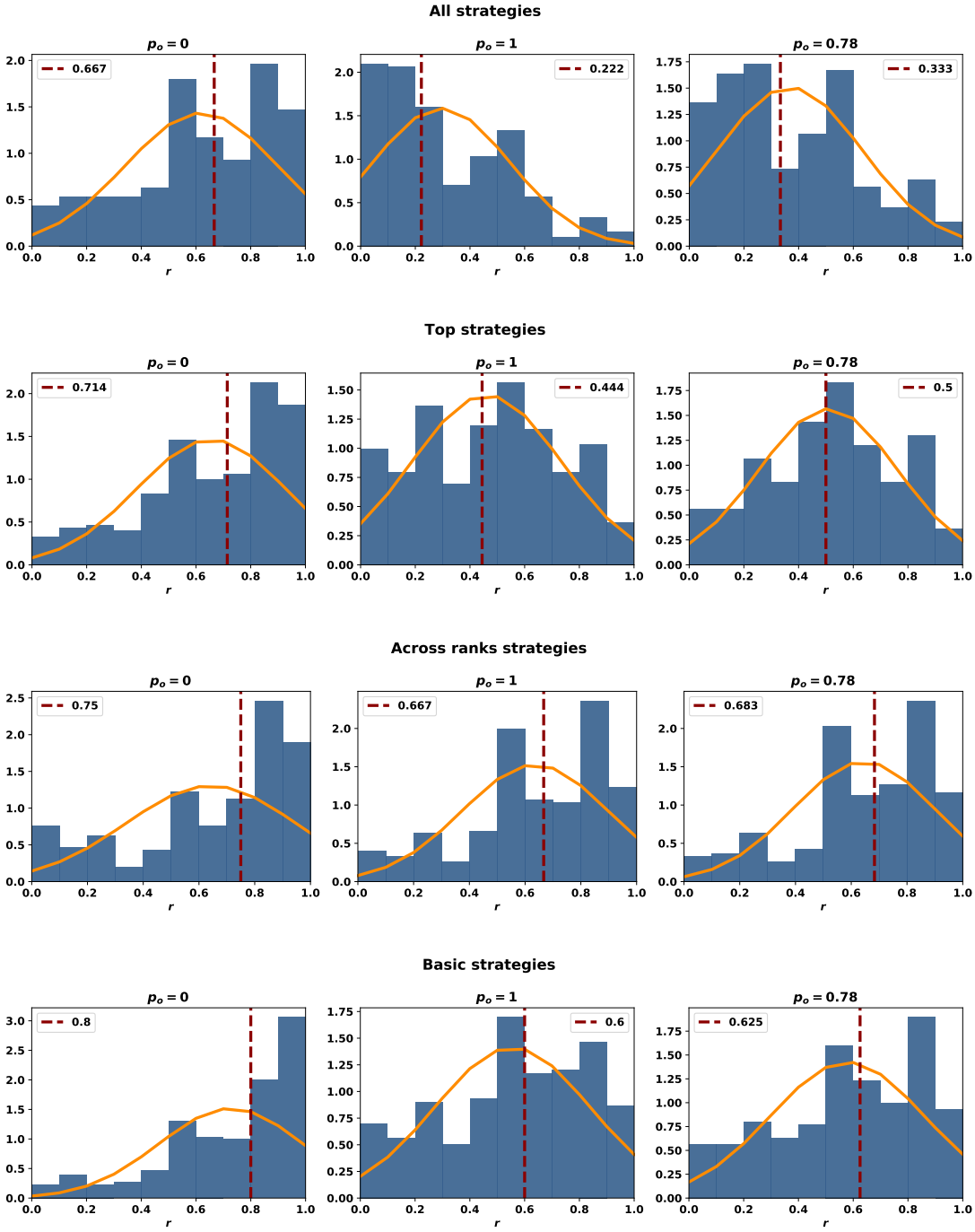
A total of 300 trials of Algorithm 1 have been performed. Similarly to Chapter **??** each trial outputs a result summary of the tournament. The performance of the strategies are evaluated on the normalised rank r, and more specifically on the median normalised rank $\bar{r}$. A total of different strategies competed in the tournaments, and the strategy with the maximum participation, excluding the LSTM strategy, was.

There are two distinct networks that have been trained on four different data sets. Moreover, each networks translates into a player that the opening probability can have three different values. Thus a total of $2 \times 4 \times 3$ LSTM players performance is evaluated in this section.

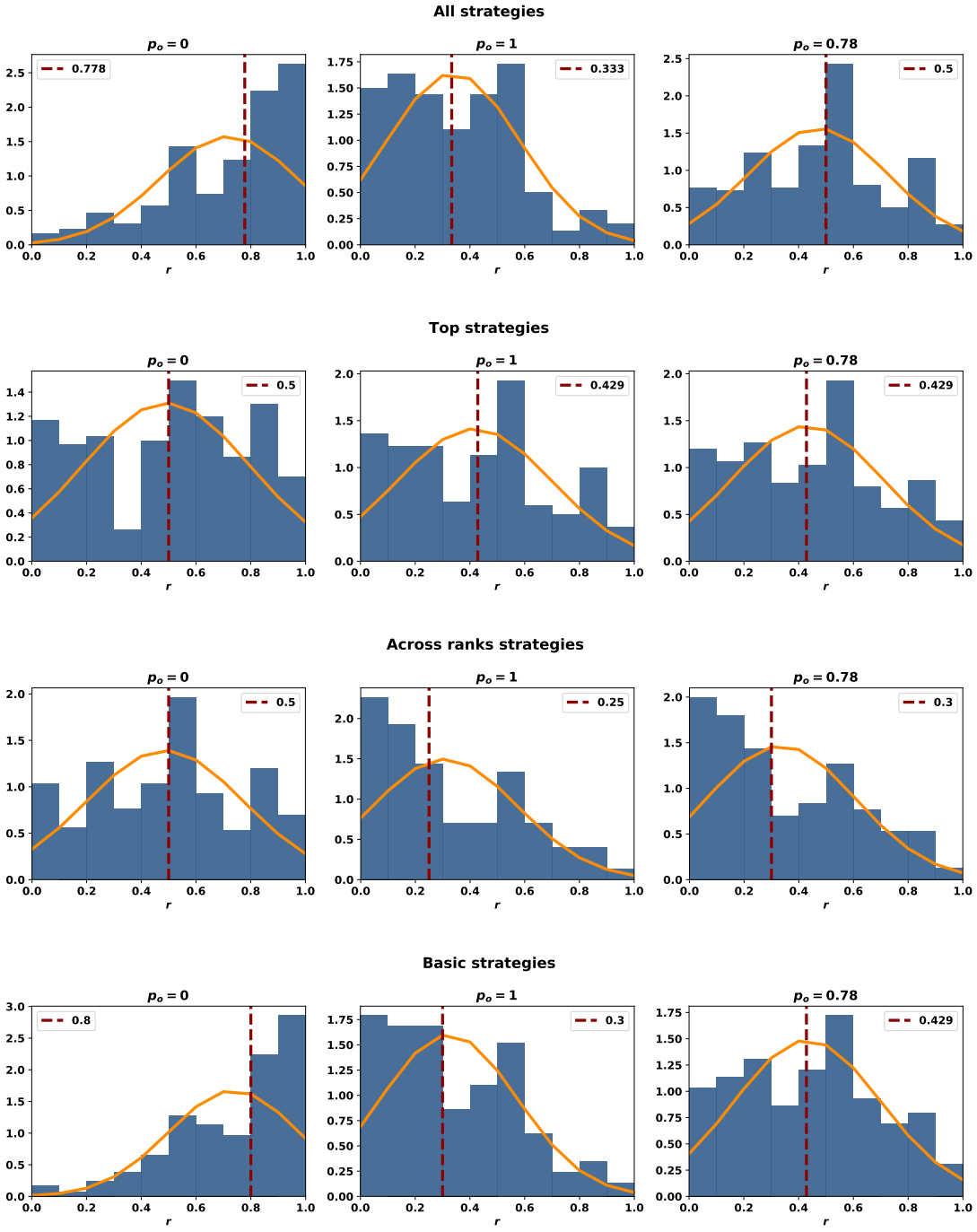
	sequence to sequence			**sequence to probability**		
	$p_o = 0$	$p_o = 1$	$p_o = 0.78$	$p_o = 0$	$p_o = 1$	$p_o = 0.78$
All strategies	0.667	0.222	0.333	0.778	0.333	0.500
Top strategies	0.714	0.444	0.500	0.500	0.429	0.429
Across ranks strategies	0.750	0.667	0.683	0.500	0.250	0.300
Basic strategies	0.800	0.600	0.625	0.800	0.300	0.429

Table 1.5: Median normalised ranks of th 24 LSTM strategies over standard tournaments.

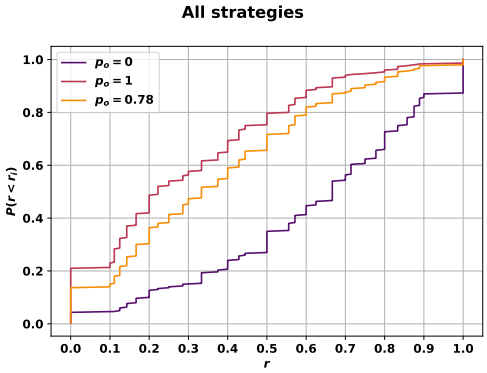
1.4.1 Fingerprinting the LSTM based strategies



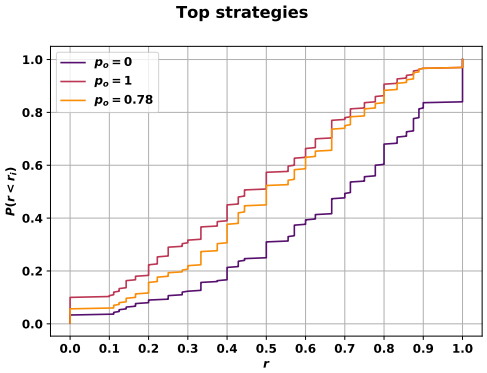
		count	mean	std	min	10%	25%	50%	75%	95%	max	skew	kurt
All strategies	$p_o = 0$	300.0	0.620	0.278	0.0	0.200	0.429	0.667	0.839	1.000	1.0	-0.523	-0.587
	$p_o = 1$	300.0	0.295	0.252	0.0	0.000	0.111	0.222	0.458	0.779	1.0	0.702	-0.251
	$p_o = 0.78$	300.0	0.368	0.265	0.0	0.000	0.143	0.333	0.560	0.833	1.0	0.433	-0.647
Top strategies	$p_o = 0$	300.0	0.655	0.273	0.0	0.250	0.500	0.714	0.875	1.000	1.0	-0.629	-0.436
	$p_o = 1$	300.0	0.461	0.274	0.0	0.090	0.222	0.444	0.667	0.875	1.0	-0.029	-0.940
	$p_o = 0.78$	300.0	0.509	0.255	0.0	0.167	0.333	0.500	0.704	0.876	1.0	-0.158	-0.710
Across ranks strategies	$p_o = 0$	300.0	0.643	0.306	0.0	0.141	0.486	0.750	0.875	1.000	1.0	-0.768	-0.523
	$p_o = 1$	300.0	0.636	0.262	0.0	0.250	0.500	0.667	0.833	1.000	1.0	-0.680	-0.198
	$p_o = 0.78$	300.0	0.645	0.255	0.0	0.250	0.500	0.683	0.833	1.000	1.0	-0.754	-0.034
Basic strategies	$p_o = 0$	300.0	0.728	0.263	0.0	0.333	0.600	0.800	1.000	1.000	1.0	-0.949	0.229
	$p_o = 1$	300.0	0.556	0.283	0.0	0.143	0.375	0.600	0.778	1.000	1.0	-0.365	-0.803
	$p_o = 0.78$	300.0	0.579	0.280	0.0	0.167	0.375	0.625	0.800	1.000	1.0	-0.435	-0.772



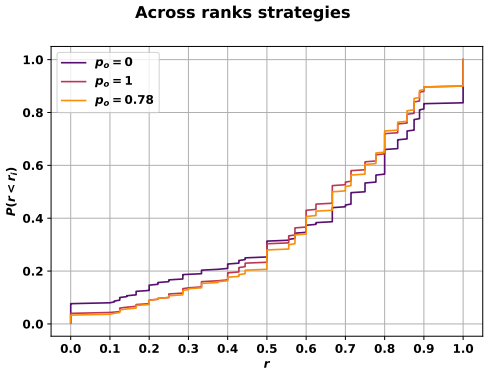
		count	mean	std	min	10%	25%	50%	75%	95%	max	skew	kurt
All strategies	$p_o = 0$	300.0	0.720	0.254	0.0	0.333	0.571	0.778	0.900	1.000	1.0	-0.860	0.056
	$p_o = 1$	300.0	0.339	0.244	0.0	0.000	0.125	0.333	0.500	0.800	1.0	0.458	-0.280
	$p_o = 0.78$	300.0	0.471	0.255	0.0	0.125	0.286	0.500	0.625	0.875	1.0	-0.064	-0.696
Top strategies	$p_o = 0$	300.0	0.491	0.305	0.0	0.000	0.200	0.500	0.714	1.000	1.0	-0.131	-1.140
	$p_o = 1$	300.0	0.417	0.283	0.0	0.000	0.167	0.429	0.600	0.875	1.0	0.157	-0.974
	$p_o = 0.78$	300.0	0.432	0.277	0.0	0.000	0.200	0.429	0.625	0.876	1.0	0.109	-0.891
Across ranks strategies	$p_o = 0$	300.0	0.487	0.287	0.0	0.000	0.286	0.500	0.700	1.000	1.0	-0.037	-0.899
	$p_o = 1$	300.0	0.308	0.267	0.0	0.000	0.111	0.250	0.500	0.800	1.0	0.586	-0.695
	$p_o = 0.78$	300.0	0.335	0.272	0.0	0.000	0.125	0.300	0.556	0.800	1.0	0.465	-0.867
Basic strategies	$p_o = 0$	290.0	0.738	0.239	0.0	0.400	0.600	0.800	0.975	1.000	1.0	-0.881	0.315
	$p_o = 1$	290.0	0.323	0.249	0.0	0.000	0.125	0.300	0.500	0.778	1.0	0.491	-0.535
	$p_o = 0.78$	290.0	0.432	0.269	0.0	0.000	0.200	0.429	0.625	0.875	1.0	0.096	-0.916



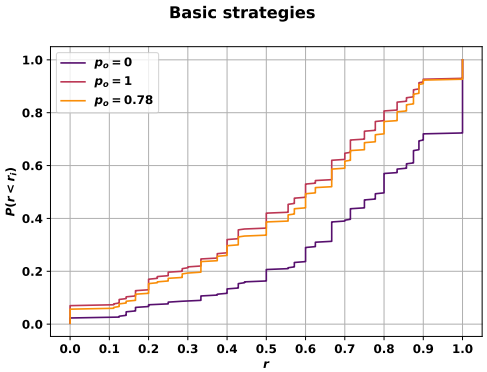
(a) $P(r < 0.5) : 0.350 P(r < 0.5) : 0.797 P(r < 0.5) : 0.717$



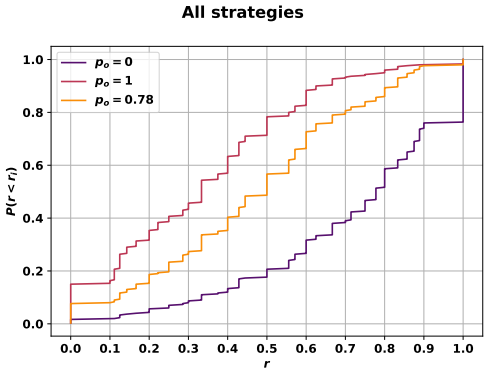
(b) $P(r < 0.5) : 0.310 P(r < 0.5) : 0.573 P(r < 0.5) : 0.523$



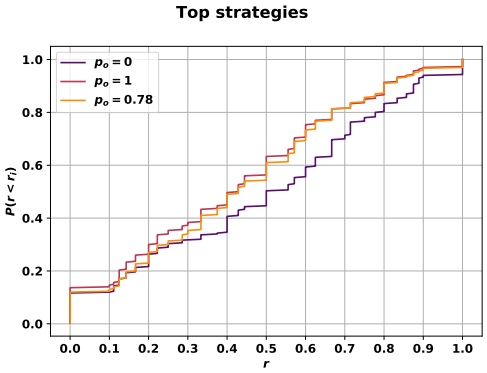
(c) ($P(r < 0.5) : 0.313 P(r < 0.5) : 0.303 P(r < 0.5) : 0.280$)



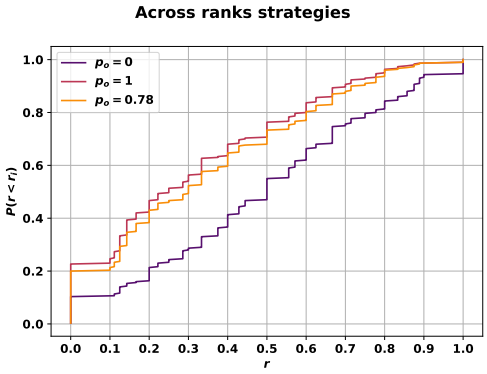
(d) $P(r < 0.5) : 0.207 P(r < 0.5) : 0.420 P(r < 0.5) : 0.387$



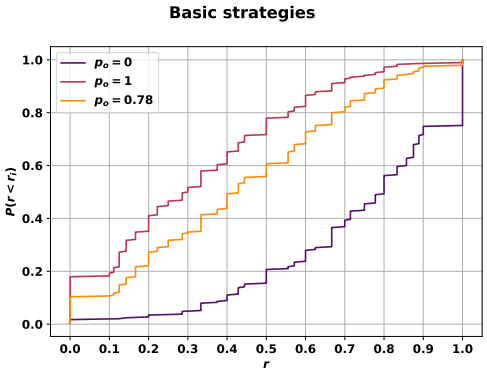
(a) $P(r < 0.5) : 0.207 P(r < 0.5) : 0.783 P(r < 0.5) : 0.567$



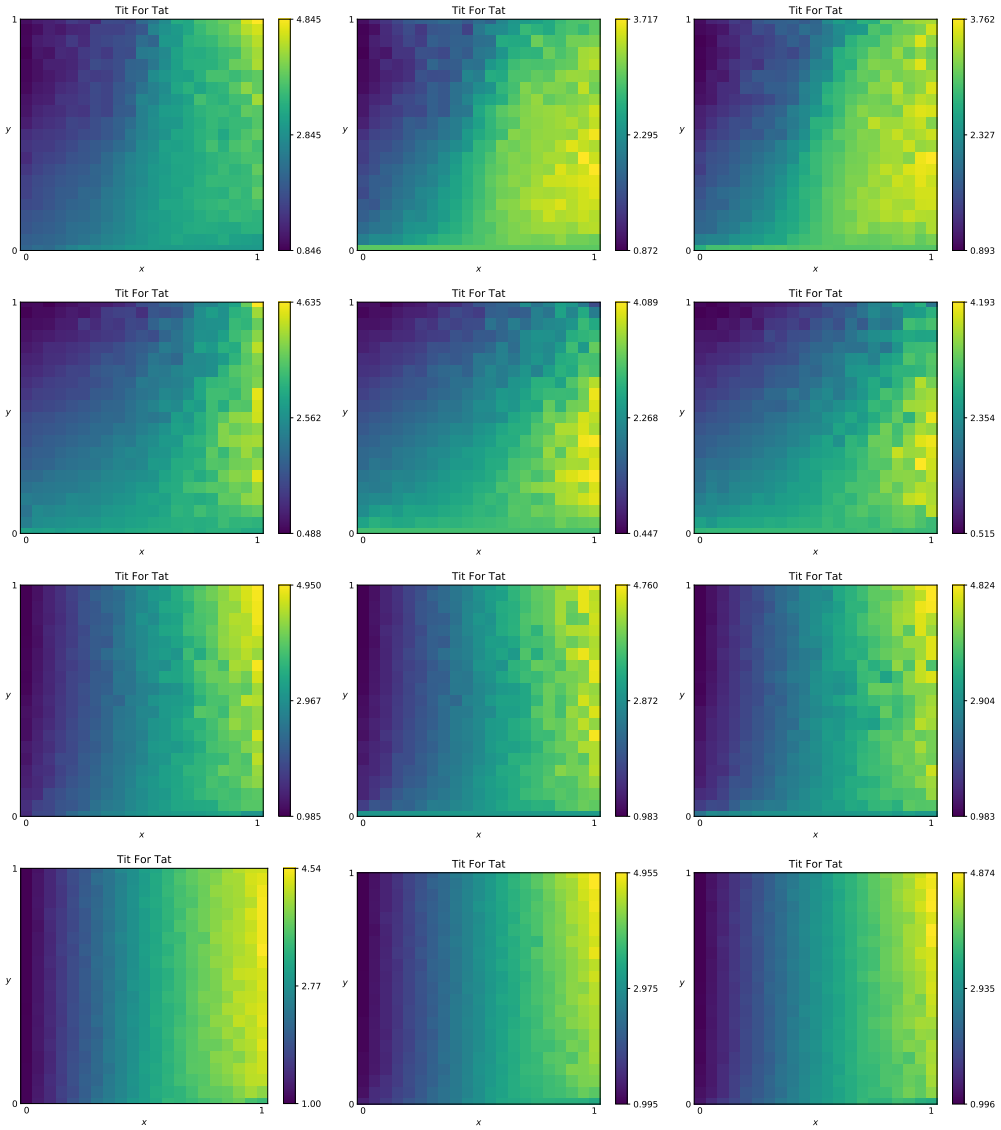
(b) $P(r < 0.5) : 0.503 P(r < 0.5) : 0.633 P(r < 0.5) : 0.610$

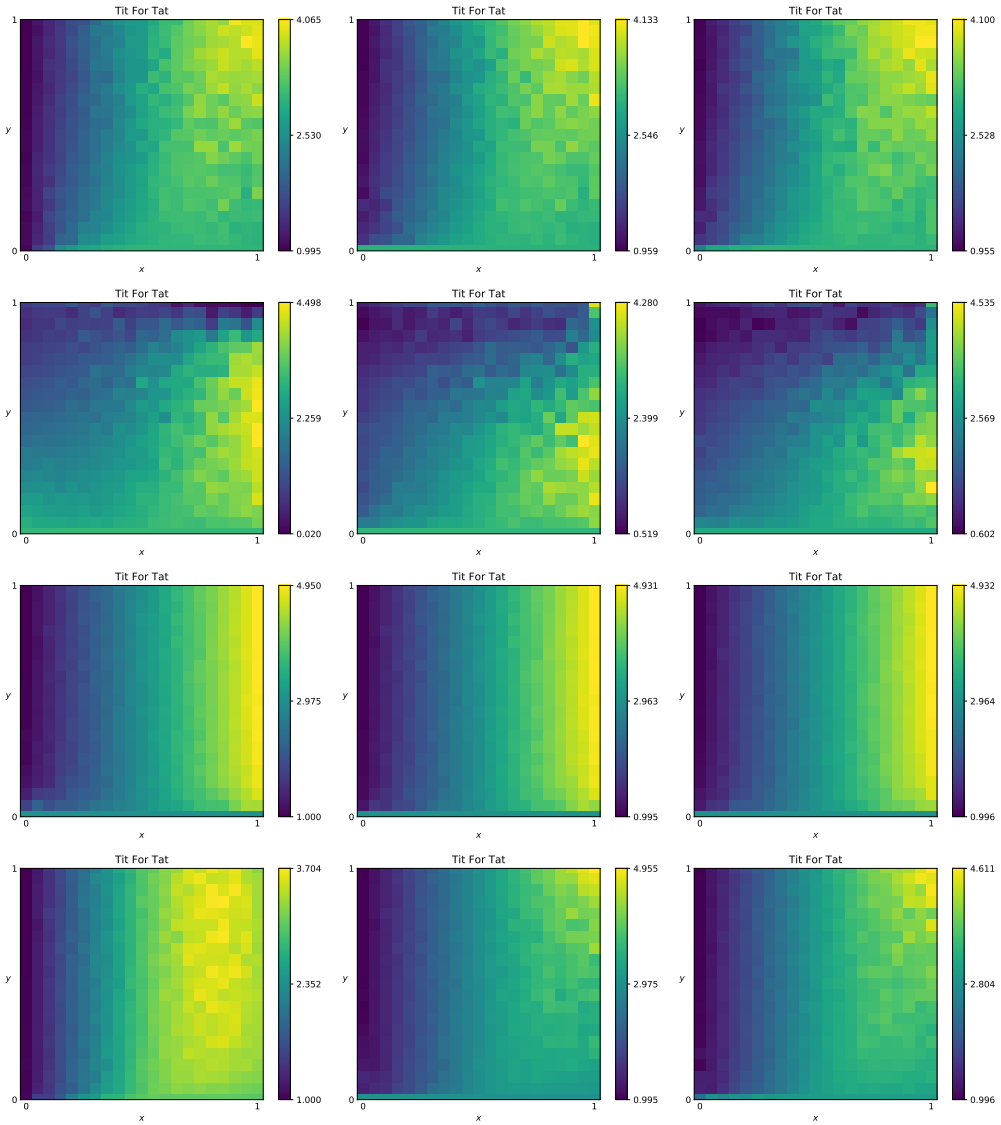


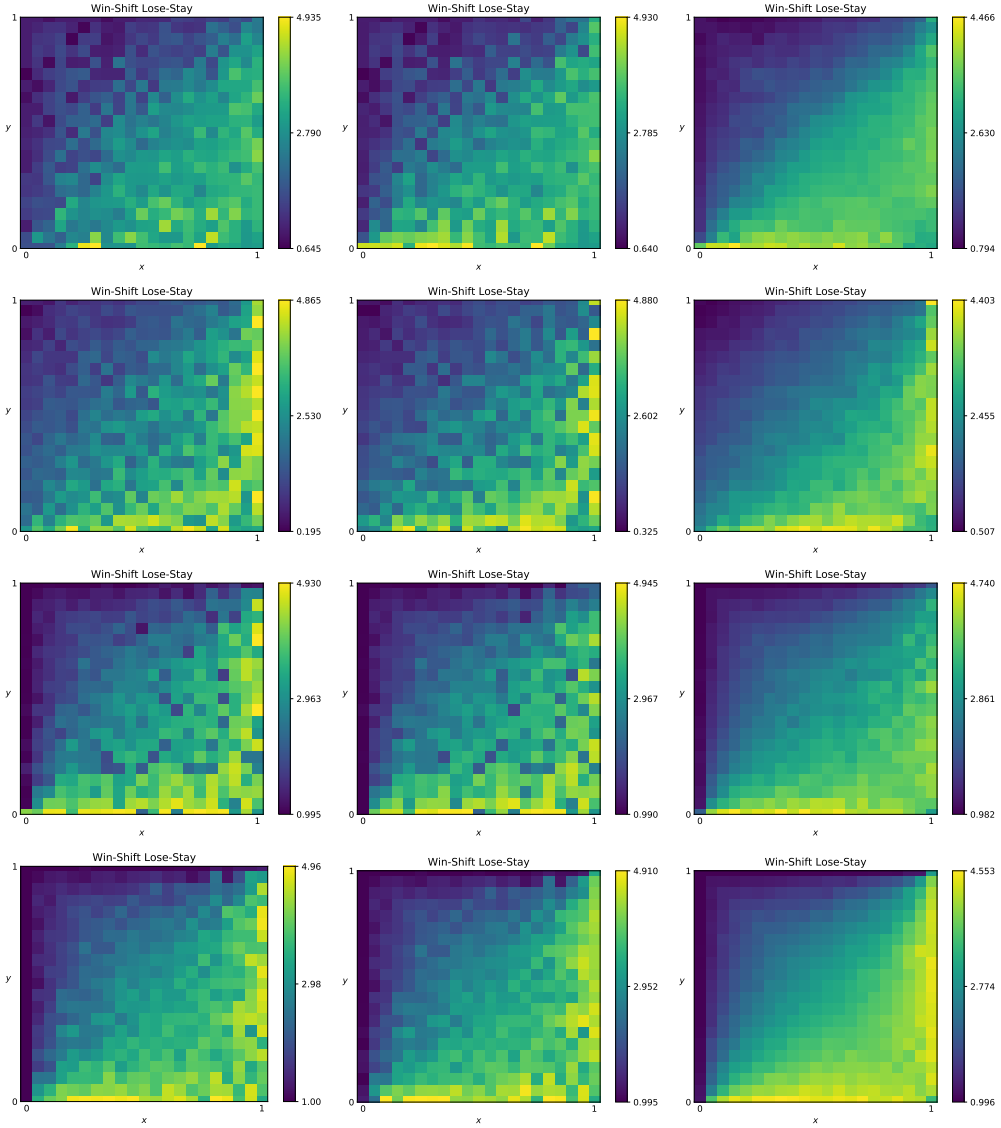
(c) ($P(r < 0.5) : 0.550 P(r < 0.5) : 0.763 P(r < 0.5) : 0.733$)



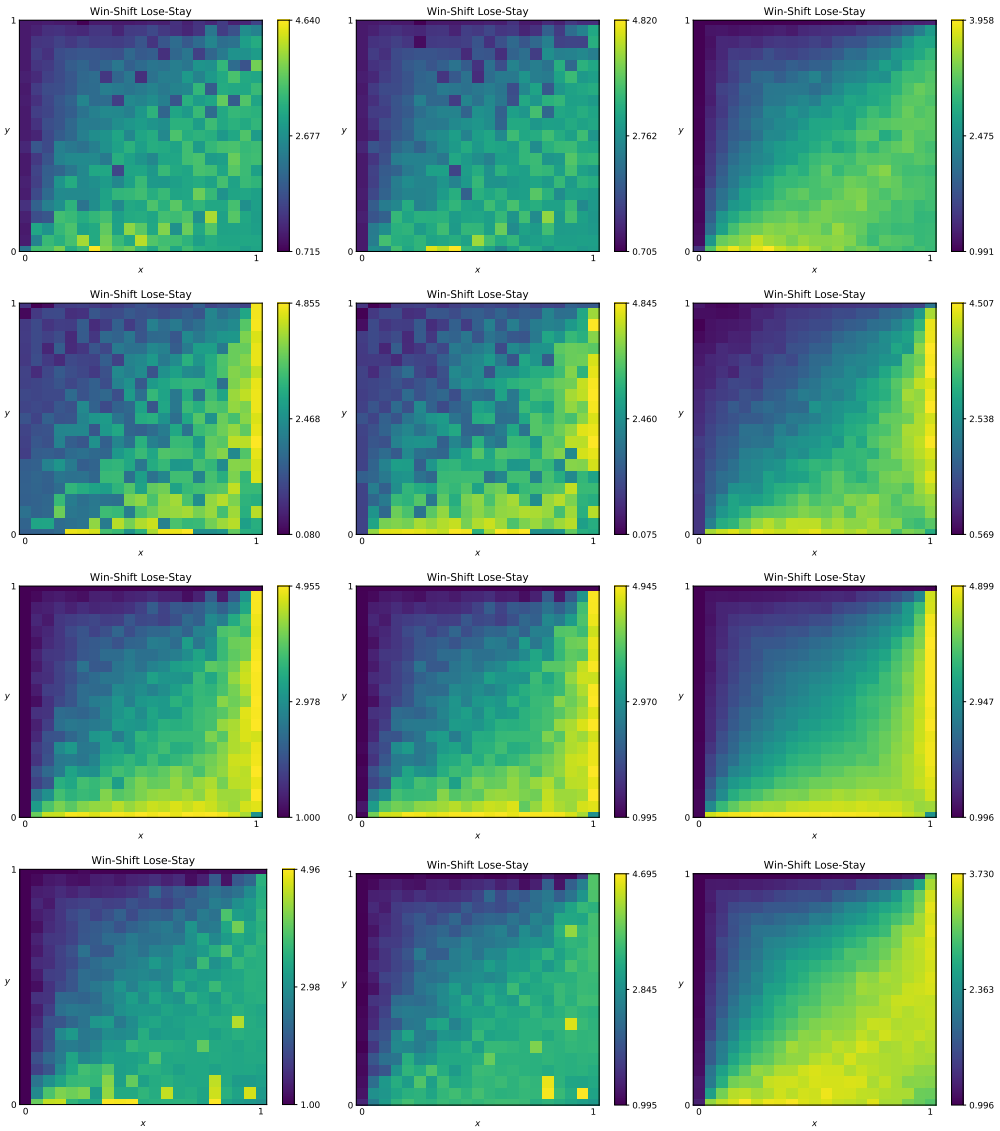
(d) $P(r < 0.5) : 0.207, P(r < 0.5) : 0.779, P(r < 0.5) : 0.607$

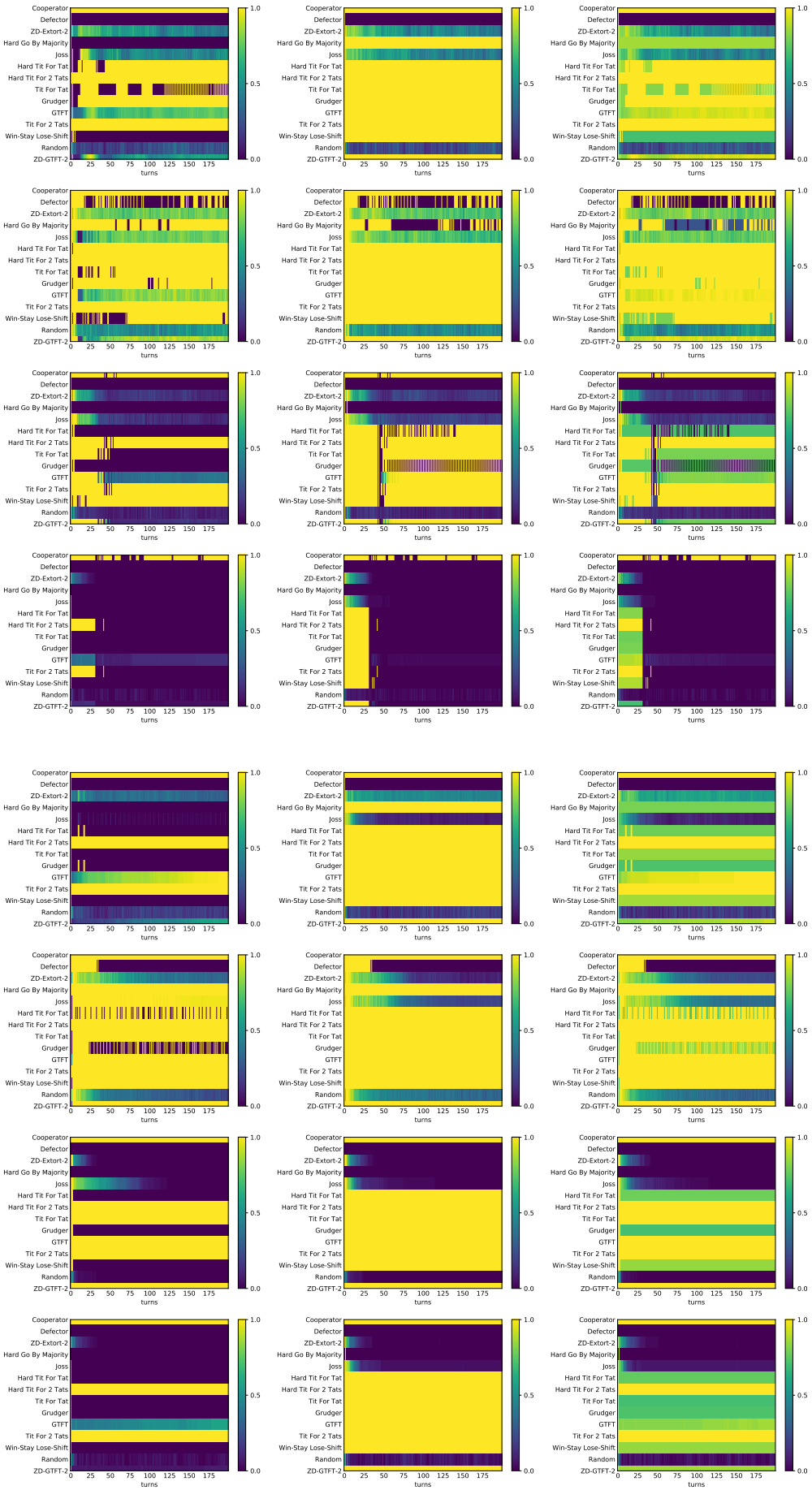

Tit For Tat
4.845
2.845
0.846
Tit For Tat
3.717
2.295
0.872
Tit For Tat
3.762
2.327
0.893
Tit For Tat
4.635
2.562
0.488
Tit For Tat
4.089
2.268
0.447
Tit For Tat
4.193
2.354
0.515
Tit For Tat
4.950
2.967
0.985
Tit For Tat
4.760
2.872
0.983
Tit For Tat
4.824
2.904
0.983
Tit For Tat
4.54
2.77
1.00
Tit For Tat
4.955
2.975
0.995
Tit For Tat
4.874
2.935
0.996
x
y

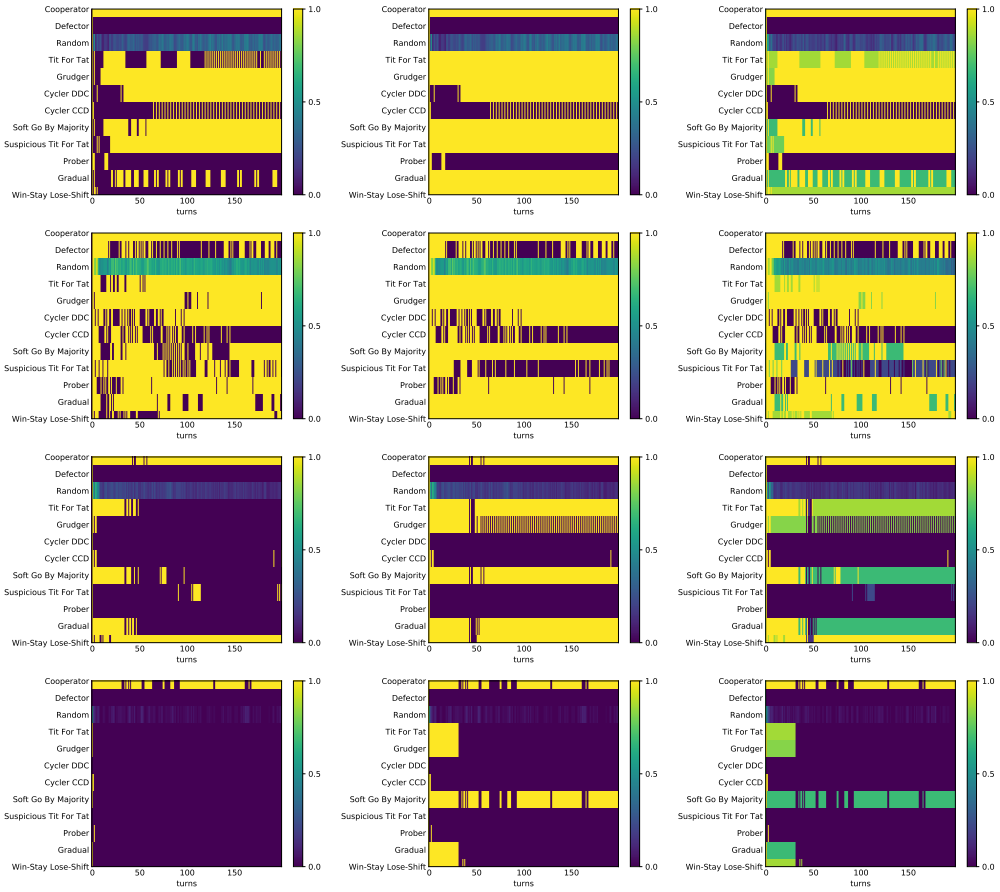

Tit For Tat
4.065
2.530
0.995
Tit For Tat
4.133
2.546
0.959
Tit For Tat
4.100
2.528
0.955
Tit For Tat
4.498
2.259
0.020
Tit For Tat
4.280
2.399
0.519
Tit For Tat
4.535
2.569
0.602
Tit For Tat
4.950
2.975
1.000
Tit For Tat
4.931
2.963
0.995
Tit For Tat
4.932
2.964
0.996
Tit For Tat
3.704
2.352
1.000
Tit For Tat
4.955
2.975
0.995
Tit For Tat
4.611
2.804
0.996
x
y

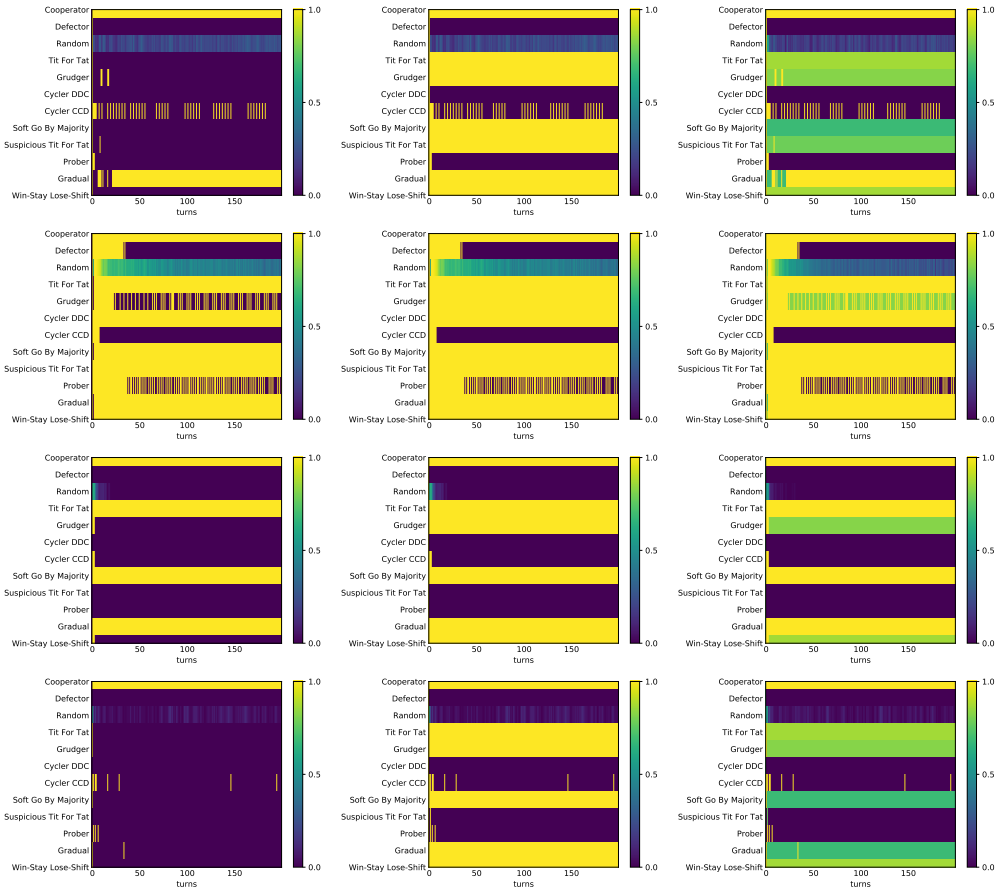


1.5 Chapter Summary


Win-Shift Lose-Stay
4.640
2.677
0.715
Win-Shift Lose-Stay
4.820
2.762
0.705
Win-Shift Lose-Stay
3.958
2.475
0.991
Win-Shift Lose-Stay
4.855
2.468
0.080
Win-Shift Lose-Stay
4.845
2.460
0.075
Win-Shift Lose-Stay
4.507
2.538
0.569
Win-Shift Lose-Stay
4.955
2.978
1.000
Win-Shift Lose-Stay
4.945
2.970
0.995
Win-Shift Lose-Stay
4.899
2.947
0.996
Win-Shift Lose-Stay
4.96
2.98
1.00
Win-Shift Lose-Stay
4.695
2.845
0.995
Win-Shift Lose-Stay
3.730
2.363
0.996
x
y


Cooperator
Defector
ZD-Extort-2
Hard Go By Majority
Joss
Hard Tit For Tat
Hard Tit For 2 Tats
Tit For Tat
Grudger
GTFT
Tit For 2 Tats
Win-Stay Lose-Shift
Random
ZD-GTFT-2
0 25 50 75 100 125 150 175
turns
1.0
0.5
0.0


Cooperator
Defector
Random
Tit For Tat
Grudger
Cycler DDC
Cycler CCD
Soft Go By Majority
Suspicious Tit For Tat
Prober
Gradual
Win-Stay Lose-Shift
0
50
100
150
turns
1.0
0.5
0.0


Cooperator
Defector
Random
Tit For Tat
Grudger
Cycler DDC
Cycler CCD
Soft Go By Majority
Suspicious Tit For Tat
Prober
Gradual
Win-Stay Lose-Shift
0
50
100
150
turns
1.0
0.5
0.0

Bibliography

[1] D. Ashlock and E. Y. Kim. Fingerprinting: Visualization and automatic analysis of prisoner's dilemma strategies. *IEEE Transactions on Evolutionary Computation*, 12(5):647–659, Oct 2008.

[2] D. Ashlock, E. Y. Kim, and N. Leahy. Understanding representational sensitivity in the iterated prisoner's dilemma with fingerprints. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 36(4):464–475, July 2006.

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[4] Kumar Chellapilla and David B Fogel. Evolving neural networks to play checkers without relying on expert knowledge. *IEEE transactions on neural networks*, 10(6):1382–1391, 1999.

[5] François Chollet et al. Keras. `https://keras.io`, 2015.

[6] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

[7] Paul J Darwen and Xin Yao. Why more choices cause less cooperation in iterated prisoner's dilemma. 2:987–994, 2001.

[8] David B Fogel, Timothy J Hays, Sarah L Hahn, and James Quon. A self-learning evolutionary chess program. *Proceedings of the IEEE*, 92(12):1947–1954, 2004.

[9] Nelis Franken and Andries Petrus Engelbrecht. Particle swarm optimization approaches to coevolve strategies for the iterated prisoner's dilemma. *IEEE Transactions on evolutionary computation*, 9(6):562–579, 2005.

[10] Marc Harper, Vincent Knight, Martin Jones, Georgios Koutsovoulos, Nikoleta E. Glynatsi, and Owen Campbell. Reinforcement learning produces dominant strategies for the iterated prisoner's dilemma. *PLOS ONE*, 12(12):1–33, 12 2017.

[11] P. G. Harrald and D. B. Fogel. Evolving continuous behaviors in the iterated prisoner's dilemma. *Biosystems*, 37(1):135 – 145, 1996.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[13] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, 1996.

[14] Soteris A Kalogirou. Applications of artificial neural-networks for energy systems. *Applied energy*, 67(1-2):17–35, 2000.

[15] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, volume 89. Presses universitaires de Louvain, 2015.

[16] James L McClelland, David E Rumelhart, PDP Research Group, et al. Parallel distributed processing. *Explorations in the Microstructure of Cognition*, 2:216–271, 1986.

[17] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[18] Marvin Minsky and Seymour Papert. An introduction to computational geometry. *Cambridge tiass., HIT*, 1969.

[19] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[20] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128*, 2014.

[21] Tuomas W Sandholm and Robert H Crites. Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems*, 37(1-2):147–166, 1996.

[22] Le Wang, Xuhuan Duan, Qilin Zhang, Zhenxing Niu, Gang Hua, and Nanning Zheng. Segment-tube: Spatio-temporal action localization in untrimmed videos with per-frame segmentation. *Sensors*, 18(5):1657, 2018.

[23] PJ Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. ph. d. thesis, harvard university, cambridge, ma, 1974. 1974.

[24] Barry J Wythoff. Backpropagation neural networks: a tutorial. *Chemometrics and Intelligent Laboratory Systems*, 18(2):115–155, 1993.