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 4 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 4 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 4 but also from the results of Chapter 5. The set of complex strategies that ranked highly across distinct tournaments in Chapter 4 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 [5] and chess [10]. The usage of feed forward ANNs in the IPD was firstly introduced in [14], and have been used ever since [1, 2, 8, 11]. Possibly, the most successful ANNs strategies are the one introduced in [13], 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 [31] 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 [31] were limited only in the network learning to play against a single strategy.

The limitations of [31] 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 [15]. 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 [23], speech recognition [30] and prediction in medical care pathways [34].

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 6. 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 7.2, presents an introduction to artificial, recurrent and long short-term memory networks.
- section 7.3, 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 7.4, 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 [17] the research on ANNs has experienced three periods of extensive activity. The first peak was in the 1940s. The work of [25] opened the subject by creating a computational model for neural networks. The second peak occurred in 1960s with the introduction of the perceptron [29]. 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 [26] 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 [35]. Though the algorithm was initially proposed by [35] it has been reinvented several times and it was popularizes by [24]. 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 [7, 19].

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 [17]. 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 7.1.

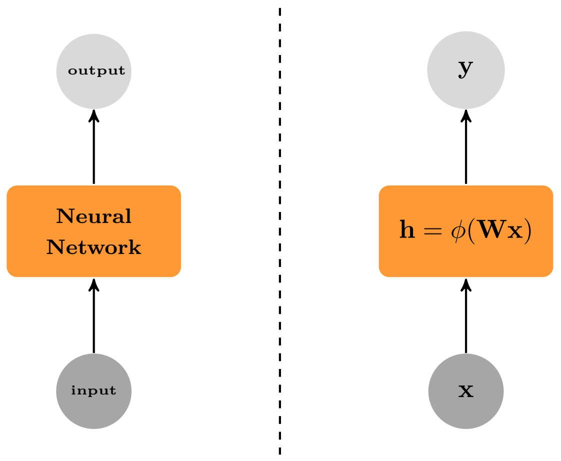


Figure 1.1: Graphical representation of an ANN.

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 $W \in \mathbb{R}^{2\times 200}$ and $V \in \mathbb{R}^{2\times 200}$. 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 [37], 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 7.2. 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 7.2.

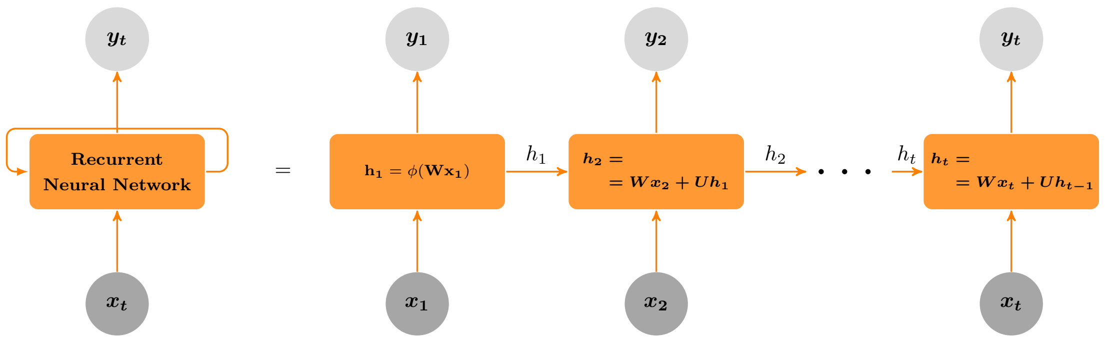


Figure 1.2: Graphical representation of a RNN.

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 [4].

A network specifically designed to avoid the long-term dependency problem. was introduced by [15] 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 7.3.

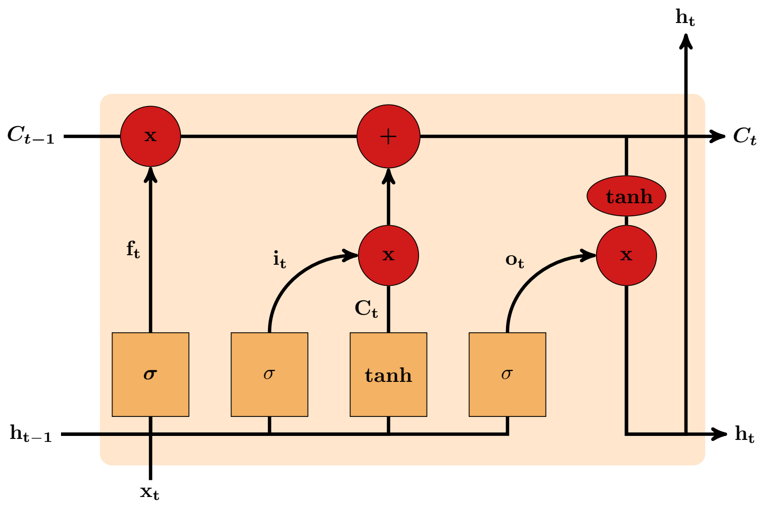


Figure 1.3: An LSTM hidden layer at time step t.

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 (7.9).

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

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 7.4.

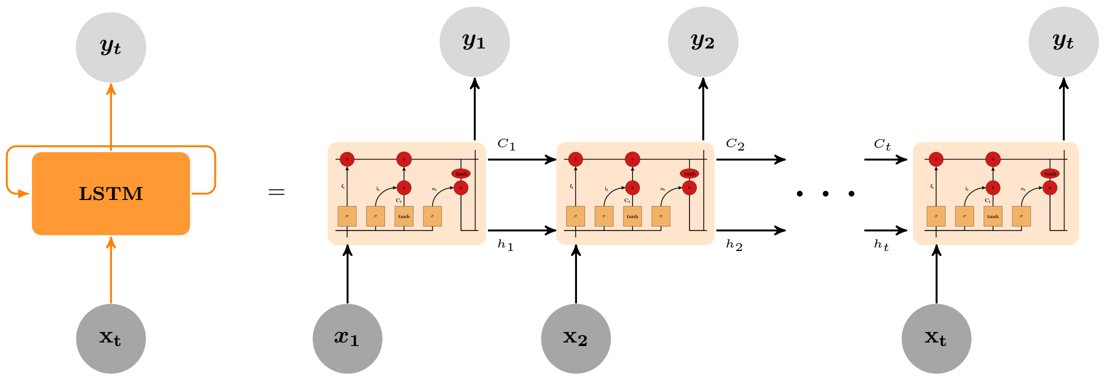


Figure 1.4: Graphical representation of an LSTM network.

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 7.3.

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

LSTM are trained in a supervised fashion on a set of training sequences. The purpose of training a network in this Chapter is so it can learn to play the IPD. For that reason the networks are going to be trained on the collection of best response sequences generated in Chapter **??**. The collection was purposely generated so that the networks could be trained to play optimal against a list of known IPD strategies.

The inputs of a network are the actions of a given strategy and the expected output is the response of the best response sequence to that action. More specifically, each pair of opponent's - best response sequence's actions correspond to 204 training inputs with a maximum dimensionality of 204.

Consider the actions of the strategy Adaptive and a best response to it as presented in section 6.4.2 given by Table 7.1.

	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 1.1: The actions of the strategy Adaptive against one of the best response sequences to the strategy. Note that $0 \to D$ and $1 \to C$.

The highest dimensionality the input vector x to the network can have is 204. This is because the expected output of to Adaptive's action in turn 1 is the action of the best response sequence at turn 2. Moreover, the expected output to the Adaptive's 204$^{\text{th}}$ action is the best response's 205$^{\text{th}}$ action. This is demonstrated by Figure 7.5.

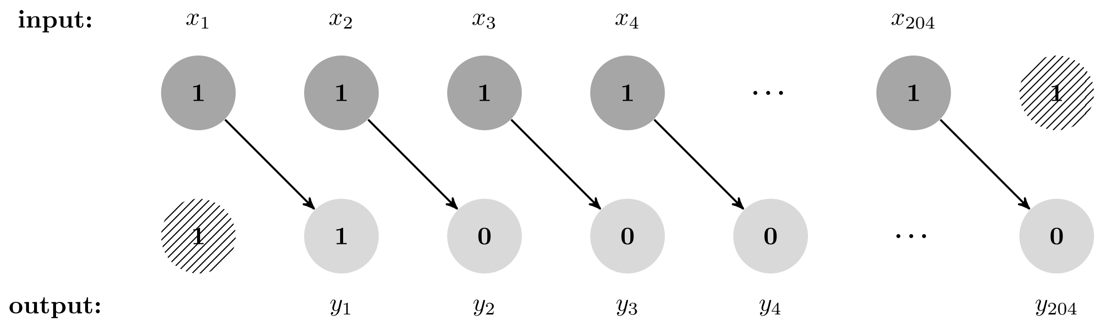


Figure 1.5: An example of a networks input and output of $t = 204$. The last action of Adaptive as well as the first action of the best response sequence are discarded.

The networks of this Chapter have been trained for a non fixed sequence length. In order to train the networks on different input lengths each pair opponent's - best response sequence's actions is broken down to 204 distinct inputs. This is done so that each input captures the play of 204 IPD matches between the pairs where the number of turns $N \in [1, 204]$. For example the training points from Table 7.1 are given by,

$$\text{inputs} = \begin{bmatrix} 1 & & & & & & \\ 1 & 1 & & & & & \\ 1 & 1 & 1 & & & & \\ 1 & 1 & 1 & 1 & & & \\ 1 & 1 & 1 & 1 & 1 & & \\ 1 & 1 & 1 & 1 & 1 & 1 & \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \text{expected outputs} = \begin{bmatrix} 1 & & & & & & \\ 1 & 1 & & & & & \\ 1 & 1 & 0 & & & & \\ 1 & 1 & 0 & 0 & & & \\ 1 & 1 & 0 & 0 & 0 & & \\ 1 & 1 & 0 & 0 & 0 & 0 & \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{1.11}$$

Thus the training data set created by the best response collection correspond to $5258 \times 204 = 1122612$ training inputs.

Two variants of LSTMs have been trained 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. Both networks output a probability which is translated at the probability of cooperating after the opponent's last t actions. A graphical representation of the two networks are given by Figures 7.6 and 7.7.

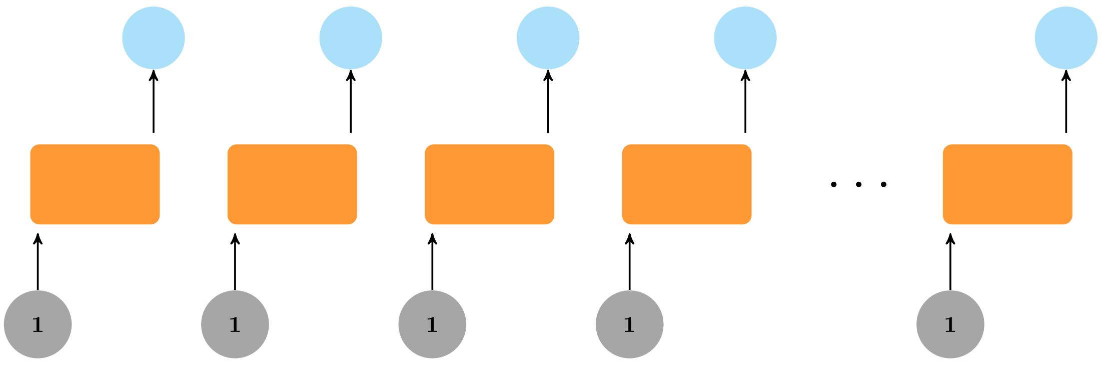


Figure 1.6: A graphical representation of a sequence to sequence LSTM model
.

The training data of the sequence to sequence model is the form of Equation (7.11) whereas the

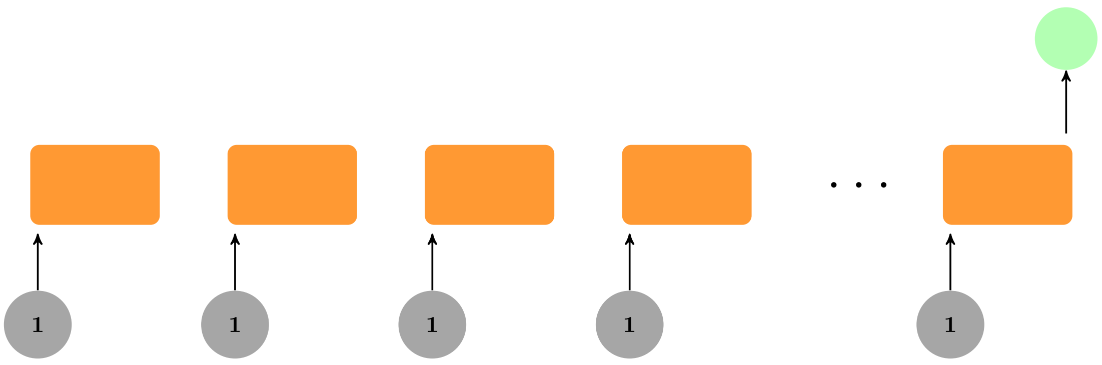


Figure 1.7: A graphical representation of a sequence to probability LSTM model
.

training data for the sequence to probability model are in the form of Equation (7.12).

$$\text{inputs} = \begin{bmatrix} 1 & & & & & & \\ 1 & 1 & & & & & \\ 1 & 1 & 1 & & & & \\ 1 & 1 & 1 & 1 & & & \\ 1 & 1 & 1 & 1 & 1 & & \\ 1 & 1 & 1 & 1 & 1 & 1 & \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \text{expected output} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \tag{1.12}$$

1.3.1 Building the networks with Keras

The open source neural-network package Keras [6] is used to construct and train the networks. Keras is a library written in Python designed to enable fast experimentation with neural networks. Neural layers, cost functions, optimizers and activation functions are all standalone modules that can be combined to create new network models.

The Python code for implementing the sequence to sequence model is given by Figure 7.8. In line 12 the model is defined to be of the `Sequential` class. This mean that the model will constructed layer by layer. The sequence to sequence network has a single LSTM layer with 100 nodes. The input to the LSTM network is not a fixed length and there a single time step between the elements of each input sequence. Those are defined by the argument `input_shape=(None, 1)`. The LSTM layer outputs the hidden states at each time step and they go through a dropout layer then are transformed into an a probability output via a sigmoid layer. There are a total of 41301 learning parameters to the sequence to sequence model.

Regarding the dimensionality of the hidden layer there is no direct answer as to what is the optimal number of nodes. Several methods for determining the dimensionality include experimentation, intuition and examples of other trained networks. Trained networks and tutorial have suggested that using a number smaller than the maximum sequence length and around a

```
>>> 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 1.8: Python code for implementing the sequence to sequence LSTM with Keras.

100 is a good value. As with all machine learning over fitting is an issue that can occur during training. A simple and yet efficient method to reduce over fitting is randomly dropping out nodes during training. That is the usage of the dropout layer.

The Python code for implementing the sequence to probability network with Keras is given by Figure 7.9. The implementations of the networks are similar, however, the sequence to probability model contains 2 LSTM layers. The first LSTM layer outputs the hidden states at each time steps and these are feed to the second layer which only output the hidden state at final time step. The output of the network is a probability. The sequence to probability network has a higher number of learning parameters due to the two LSTM layers. More specifically, there are 122101 learning parameters to the network.

```
    >>> 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 1.9: Python code for implementing the sequence to probability LSTM with Keras.

Keras includes two implementations for an LSTM layer. The class `LSTM` and the class `CCNLSTM` which is the class used here. `CCNLSTM` provides a faster implementation of `LSTM` the NVIDIA CUDA Deep Neural Network library. The `CCNLSTM` class allows the networks to be trained on a graphics processing unit (GPU).

1.3.2 Training on GPU

Conventionally executing computer code happens on the central processing unit (CPU). The CPU, also called main processor, is essentially the brain of any computing device. Architecturally the CPU is composed of just a few cores designed to support an extremely broad variety of tasks.

A graphical processing unit (GPU), on the other hand, is composed of hundred of cores designed to process a set of simpler and more identical computations in parallel. GPUs were initially designed as dedicated graphical rendering workhorses of computer games but ere later enhanced to accelerate other geometric calculations. 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 machine learning applications. A GPU core is optimised exclusively for data computations and 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, machine learning applications, which perform large numbers of computations on a vast amount of data, can see huge performance improvements when running on a GPU versus a CPU.

The training process of the LSTM networks here is carried out on a GPU. More specifically, the training process was carried out on the High-Performance Computing (HPC) cluster of Cardiff University.

1.3.3 Training data sets

Section 7.3 covered the training data used to train the LSTM networks. There are a total 1122612 training inputs and expected outputs to the training data set. In order to understand the effect of the training data on the LSTM strategy the networks are not trained only of the entire training set but also on three subsets.

The subset as well as their details and explanation of the best response sequences included are given by Table 7.2.

Data set	# of opponents	Explanation	# of best response sequences
all strategies	192	The data set as presented in Chapter 6	5258
top performing strategies	18	A data set constructed in the same way as the training data set but only with the best response sequences to 18 strategies. These are the top performing strategies in a standard tournaments performed with APL with 218 opponents.	714
strategies across ranks	15	A data set generated only with the nest response sequences to 15 strategies whose ranks are across the 218 ranks of the standard tournament.	212
basic strategies	11	A data set generated with the best response sequences to 11 which are classified as a set of basic strategies in the APL	84

Table 1.2: Training data sets used to training the LSTM networks. The IPD standard tournament with the 218 opponent has been carried out using APL version 4.5.0. The results are available online: `https://github.com/Axelrod-Python/tournament`.

A list of the strategies' names using in each subset is given in the Appendix.

1.3.4 Training and validation

Two different LSTM networks were trained on four different training data sets. Thus, a total of $4 \times 2 = 8$ networks have been trained. Due to the different size of the training data the networks have been trained for a different number of epochs. The number of epochs is the number of times the training algorithm has work through the entire training data set. The number of epochs each network has been trained for is given by Table 7.3.

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

Table 1.3: Number of epochs each both networks, sequence to sequence and sequence to probability, were trained for each subset.

The networks based on the sequence to probability architecture have a larger number of learning parameters. Subsequently, their training corresponds to more computer time. It can been seen from Table 7.3 that those networks have been trained for less epochs compared to the sequence to sequence networks trained on the same training data sets.

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. Discussion on validation. In section 7.4 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

In section 7.3 the trained networks were evaluated based on the training data. In this section the networks are evaluated as IPD strategies in a number of standard tournaments.

A strategy class was implemented called the `LSTMPlayer` so that the predictions of the networks can be translated into actions in an IPD match simulated with APL, Figure 7.12. The `LSTMPlayer` takes as arguments a Keras model, a reshape history function and the probability that it's opening move it's a cooperation. The opening move is not given by the LSTM networks as they were designed to only give reactions to history of play. Thus, the probability of opening with a cooperation is an argument of the player and is denoted as p_o.

The reshape history function is used to transform the history of the match to an input used my the model to make a prediction. The player makes a deterministic decision based on the prediction. It cooperates if the prediction is greater than 0.5 and defects either wise.

Though there are only 8 different trained networks as it was described the opening move is an input of the player. Three different values are used for p_o. These are 0, 1 and 0.78. Opening deterministically is a common feature of IPD strategies and the probability 0.78 has been calculated as the probability the best response sequences of Chapter 6 open with a cooperation. Thus, the 2 different LSTM architectures have been trained separately on 4 different training set generating 8 different networks that each corresponds to 3 distic IPD strategies were the

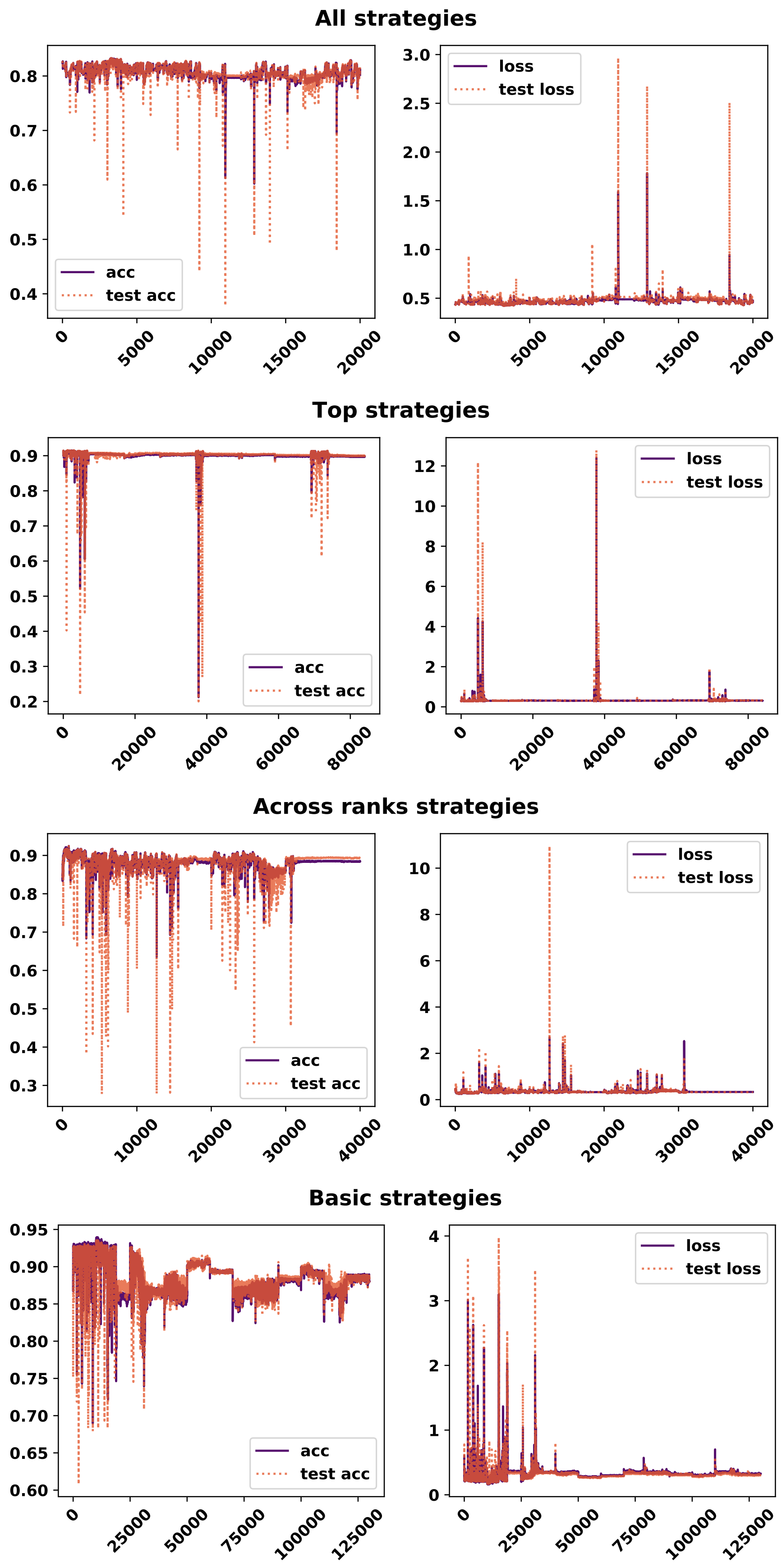


Figure 1.10: Loss function and accuracy of the networks based on the sequence to sequence model over the number of epochs.

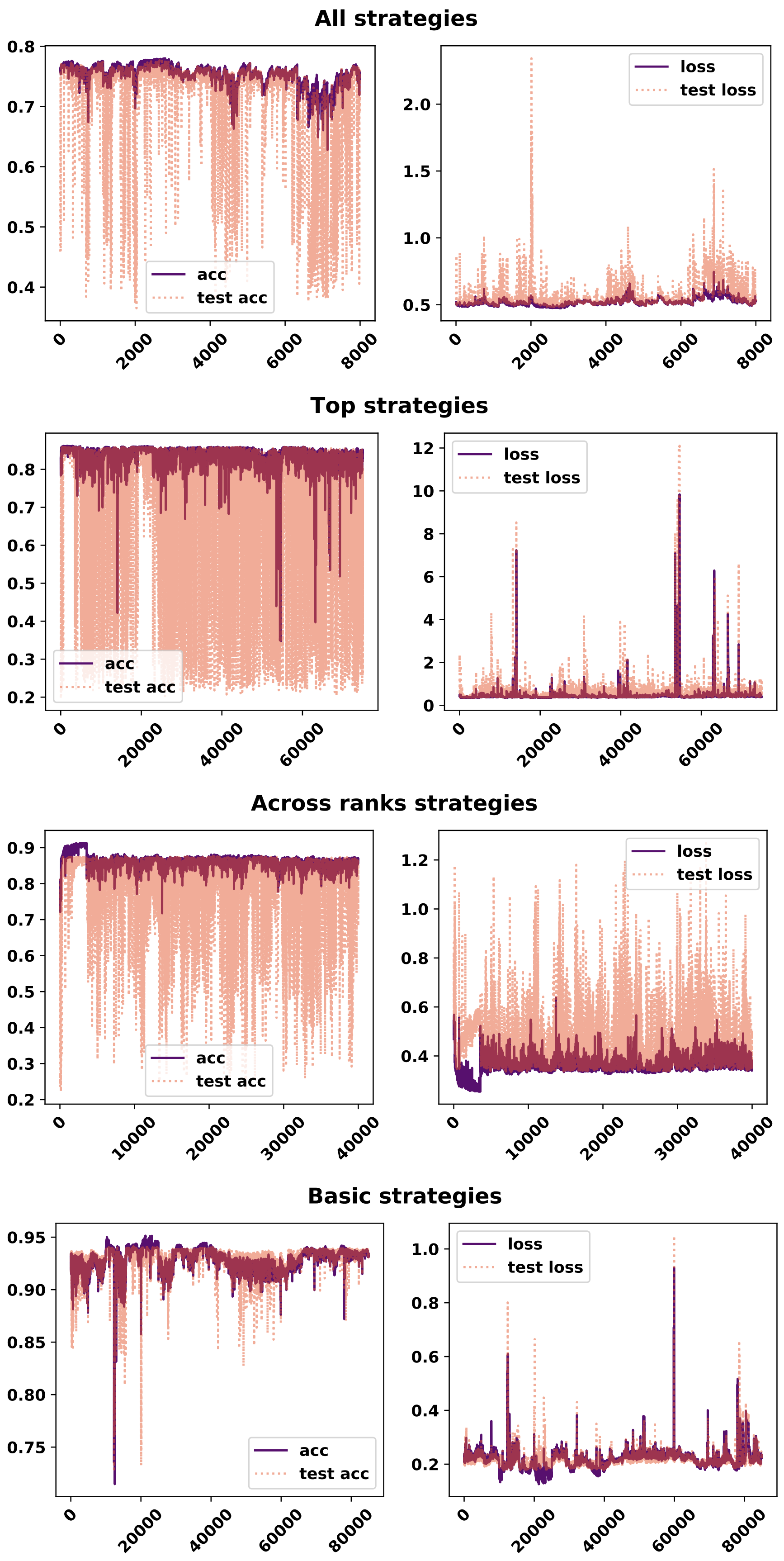


Figure 1.11: Loss function and accuracy of the networks based on the sequence to probability model over the number of epochs.

```python
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
```

Figure 1.12: Implementation of the `LSTMPlayer` class.

opening probability is different. Thus, a total of $2 \times 4 \times 3 = 24$ IPD strategies have been trained and are evaluated in this section.

The performance of the 24 LSTM strategies are evaluated and compared in 300 standard tournaments. The process of collecting the tournament results for each strategy is given by Algorithm **??**.

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;
```

For each trial a random size N is selected and from the 192 strategies a random list of N strategies is chosen. The LSTM player is then added to the list of player increasing the size to $N + 1$. For the given list a standard tournament of 200 turns is performed and repeated 50 times. The only parameter that varies in the result collection is N which can have values between 5 and 10.

The number of turns is fixed at 200. In Chapter 6 the sequences were fixed to 205 turns. That had an effect that most best response sequences defected on the last turns, as the match was coming to an end. The LSTM networks have been trained on those sequences and are biased towards defecting on the last turns. To avoid unconditional defections by the LSTM strategies their performance is evaluated 200 turns which are a common number of turns used in the IPD literature.

For each turn a result summary in the same format of that presented in Chapter **??** is exported. The performance of the strategies are evaluated on the normalised rank r, and more specifically on the median normalised rank $\bar{r}$. As a reminder r is calculated as a strategy's rank divided by $N - 1$. The $\bar{r}$ of each of the 24 strategies over the 300 standard tournaments are given by Table 7.4.

Based on $\bar{r}$ the LSTM strategy with the more successful performance is the strategy based on a sequence to sequence network trained over the entire data set and opens with a cooperation. Followed by the strategy trained over the across ranks data set that opens with a cooperation based on the sequence to probability network. There are a few strategies that have achieved a $\bar{r}$ close to 0.3.

Overall the strategies that open with cooperation outperform the strategies based on the same networks and training data set that open with a probability of 0.78 and they in order outperform the strategies that open with defection. The strategies that have been trained on the sub sets

	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 1.4: The median normalised ranks of the 24 LSTM strategies over the standard tournaments. A $\bar{r}$ closer to 0 indicates a more successful performance.

and specifically on the basic strategies and across the ranks strategies perform better when they are based on the sequence to probability network.

The distributions of the normalised ranks for each of the players are given by Figures 7.13 and 7.14.

Figure 7.13 gives the normalised rank distributions for the strategies based on the sequence to sequence network. The best performing LSTM strategy is the one trained on the entire data set with $p_o = 1$. It's normalised rank distribution is skewed towards 0. It can be seen that strategy performed well overall with a few exceptions. The rest of distributions appear to be either skewed in the middle or towards 1. These can be confirmed by the skewness and kurtosis measures which are reported in Table 7.5.

		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

Table 1.5: Statistics summary of the r distributions for the strategies based on the sequence to sequence network.

Figure 7.14 gives the normalised rank distributions for the strategies based on the sequence to probability networks. Overall the distributions are more skewed towards 0 compared to the distributions of Figure 7.13. The strategies that have been trained on the subsets are performing better. The statistics summary of the distributions is given by Table 7.6

An interesting question that arises is what is the probability the players rank in the top half of a tournament? The cumulative distribution function (CFD) for the r distributions are given by Figures 7.15 and 7.16.

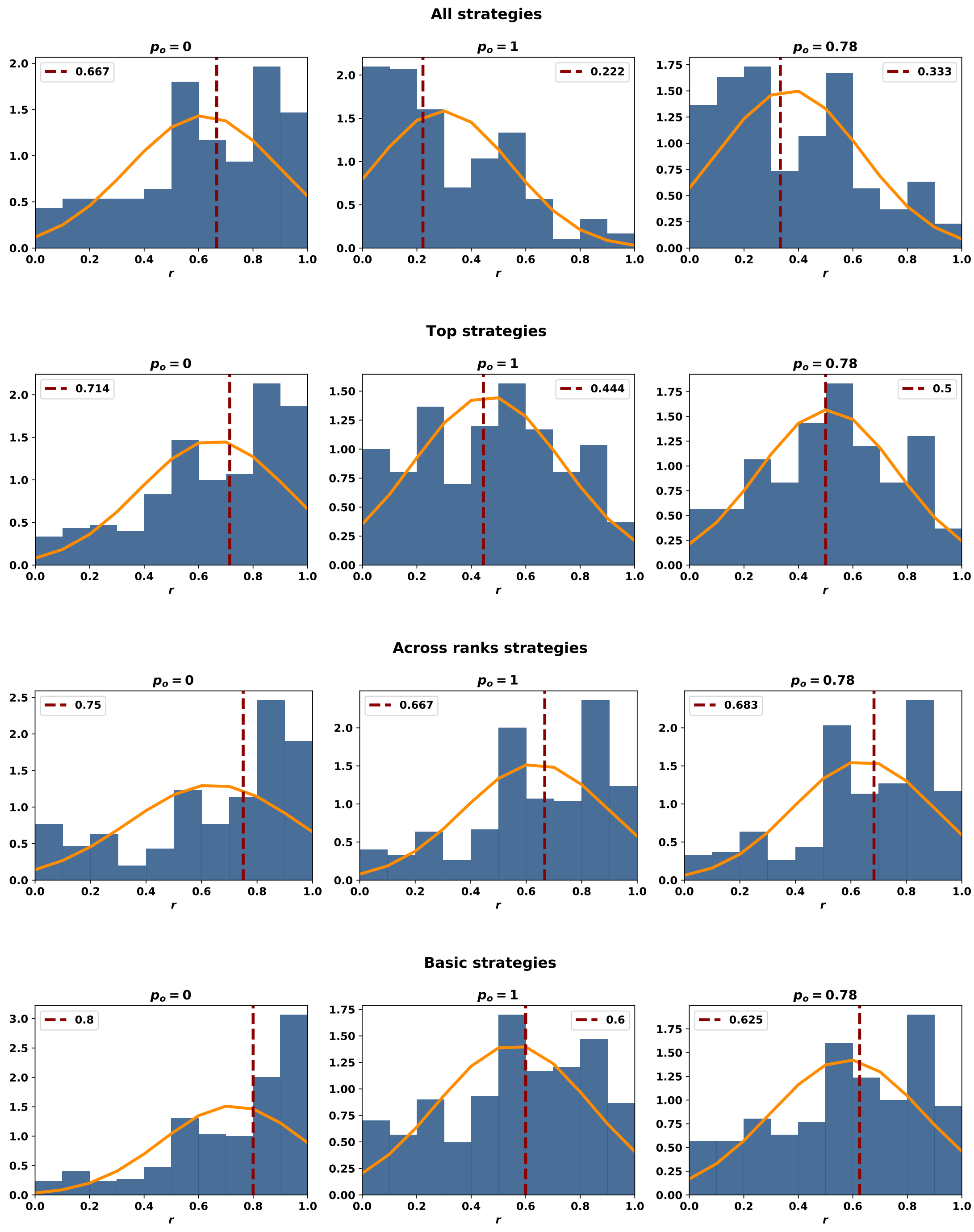


Figure 1.13: Normalised rank distributions for the strategies which are based on the sequence to sequence LSTM.

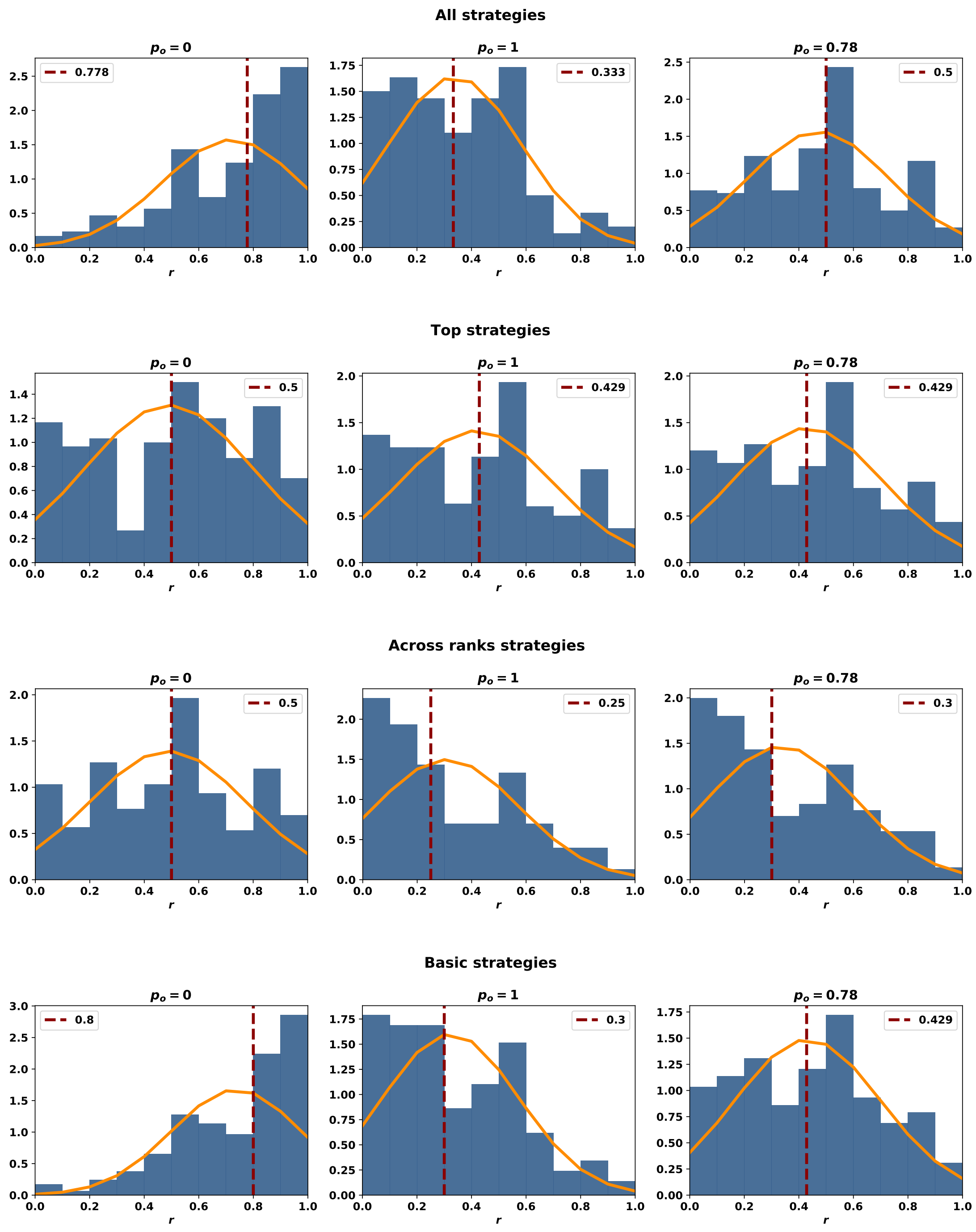


Figure 1.14: Normalised rank distributions for the strategies which are based on the sequence to probability LSTM.

		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

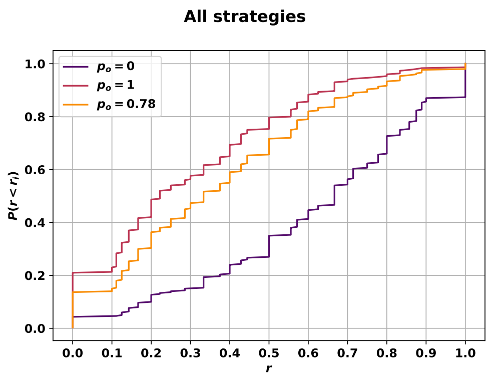
Table 1.6: Statistics summary of the r distributions for the strategies based on the sequence to probability network.

Figure 7.15 gives the CDF for the strategies trained with the sequence to sequence model. Opening with a cooperation or with a probability 0.78 of a cooperation increases a strategy's probability of scoring in the top half of a tournament. The highest probability is that of best performing strategy which a probability of 0.8 of placing in the top ranks for $p_o = 1$ and a 0.7 for $p_o = 0.78$.

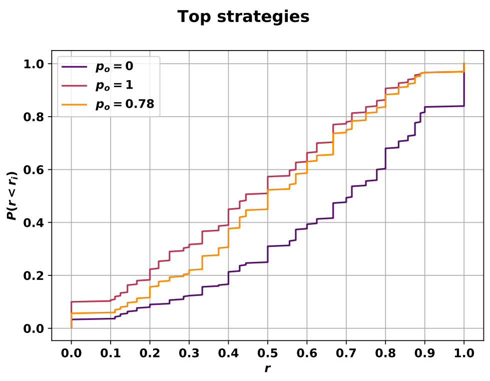
Figure 7.15 gives the CDF for the strategies trained with the sequence to probability model. The probabilities of placing in the top ranks are higher for the strategies based on the subsets with this network with many strategies placing in the top half with a probability higher than 0.7.

This section has evaluated the performance of 24 distinct IPD strategies based on LSTM networks in 300 . The differences between these players included the architecture of their networks, the training data and the their opening move. On the whole, the analysis of this section has shown that:

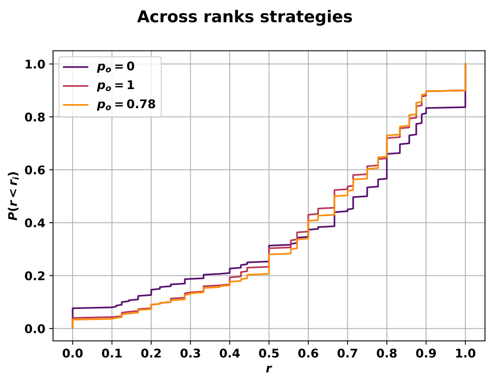
- The strategies trained on the entire training data set have a more successfully performance compared to the strategies trained on the subsets.
- The strategies trained on the subsets are performing better when they are based on the sequence to probability networks. The sequence to probability networks for the subsets have been trained for a larger number of epochs compared to the network on the entire data set. An interesting question is whether the strategy based on this network would perform even better than the sequence to probability if it was trained for longer.
- The strategies that have been trained only on the top ranked strategies are overall performing poorly.
- Opening with a probability makes a difference of the LSTM strategies. Overall the most successful strategies are the ones that open with a cooperation. This enforces the discussion that opening with cooperation is a property of a successful strategy started by Axerod.



(a) For $p_o = 0, P(r < 0.5) = 0.350$. For $p_o = 1, P(r < 0.5) = 0.797$. For $p_o = 0.78, P(r < 0.5) = 0.717$.

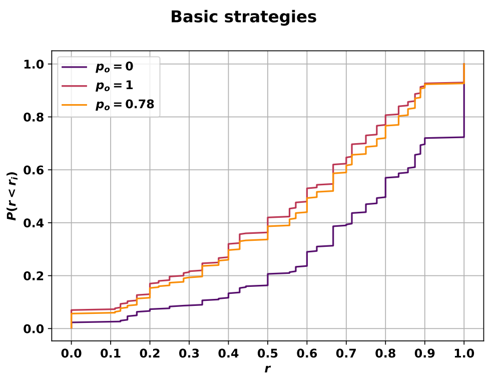


(b) For $p_o = 0, P(r < 0.5) = 0.310$. For $p_o = 1, P(r < 0.5) = 0.573$. For$p_o = 0.78, P(r < 0.5) = 0.523$.



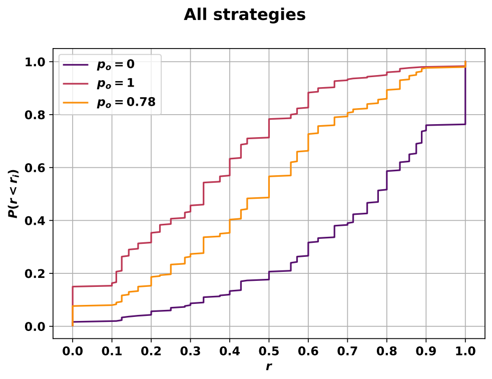
(c) (

For $p_o = 0, P(r < 0.5) = 0.313$. For $p_o = 1, P(r < 0.5) = 0.303$. For $p_o = 0.78, P(r < 0.5) = 0.280$.)

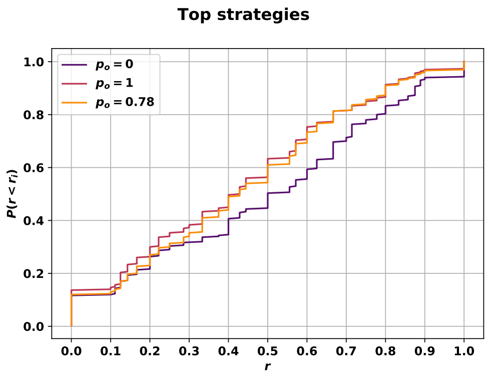


(d) For $p_o = 0, P(r < 0.5) = 0.207$. For $p_o = 0, P(r < 0.5) = 0.420$. For $p_o = 0.78, P(r < 0.5) = 0.387$.

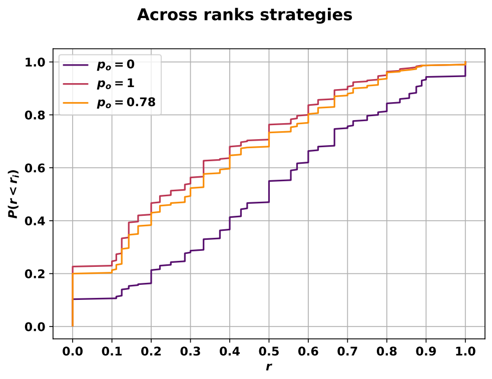
Figure 1.15: The cumulative distribution function (CFD) for the r distributions for the LSTM strategies based on the sequence to sequence network.



(a) For $p_o = 0, P(r < 0.5) = 0.207$ For $p_o = 0, P(r < 0.5) = 0.783$. For $p_o = 0, P(r < 0.5) = 0.567$.

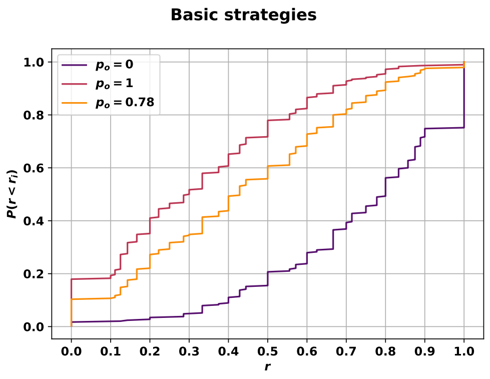


(b) For $p_o = 0, P(r < 0.5) = 0.503$. For $p_o = 0, P(r < 0.5) = 0.633$. For $p_o = 0, P(r < 0.5) = 0.610$.



(c) (

For $p_o = 0, P(r < 0.5) = 0.550$. For $p_o = 0, P(r < 0.5) = 0.763$. For $p_o = 0, P(r < 0.5) = 0.733$.)



(d) For $p_o = 0, P(r < 0.5) = 0.207$. For $p_o = 0, P(r < 0.5) = 0.779$. For $p_o = 0, P(r < 0.5) = 0.607$.

Figure 1.16: The cumulative distribution function (CFD) for the r distributions for the LSTM strategies based on the sequence to probability network.

1.4.1 Fingerprinting the LSTM based strategies

The 24 strategies that have been introduced in this Chapter are based on an LSTM archetype. The strategies are based on a complex structure and interpreting them is not trivial. The difference between the strategies is not straightforward and even thought there differences between them it's not sure whether they behave in a completely different manner. In Chapter **??** the method of fingerprinting was introduced. Fingerprinting is a method that produces a function signature of a strategy. Two types of fingerprinting were discussed in Chapter **??**. The Ashlock's fingerprints and transitive fingerprints.

Ashlock's fingerprints compute the score of a strategy against a spectrum of opponents. The basic method is to play the strategy against a probe strategy with varying noise parameters. The fingerprints for the 24 strategies based on Ashlock's approach have been generated for probe strategy Tit For Tat and Pavlov (Win-Shift Lose-Stay). These are given by Figures 7.17, 7.18, 7.19 and 7.20.

For Figures 7.17 and 7.18 Tit For Tat is used as a probe. It can been seen that between the strategies there are some similarities between the players based on same network, trained on the same training data set when they open with 1 or 0.78 change of cooperation. However, across the models there appears to be no similarity between the strategies.

Figures 7.19 and 7.20 give the Ashlock fingerprints whilst Pavlov is used as a probe. The previous results still stand however now there is more similarities across the strategies. This includes the players when the only difference is the opening move but all the strategies that have been trained on the subsets.

To further explore the similarities of the strategies a a set of more interpretable fingerprints the transitive fingerprints implemented in APL have also been estimated. The transitive fingerprints represent the cooperation rate of a strategy against a set of opponents over a number of turns. There are two set of opponents used here to generate the transitive fingerprints, these are the collection of strategies from [33] and from [3].

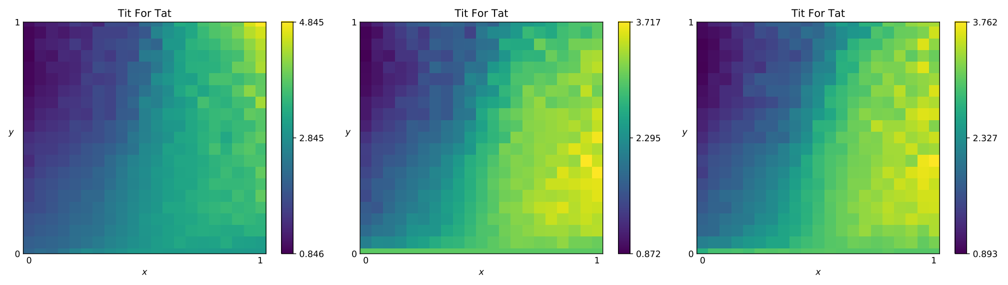
The differences now are clearer. The opening move can lead to slightly different behaviour but also to completely different one. Between the set of strategies that only their networks are different there can sometimes be some similarities but others there are completely different. This lead to the conclusion that the training process has managed to train a set of strategies that ar different. Though there are playing the IPD the training set or even the opening move can have a important effect.

1.4.2 Stochastic LSTM strategies

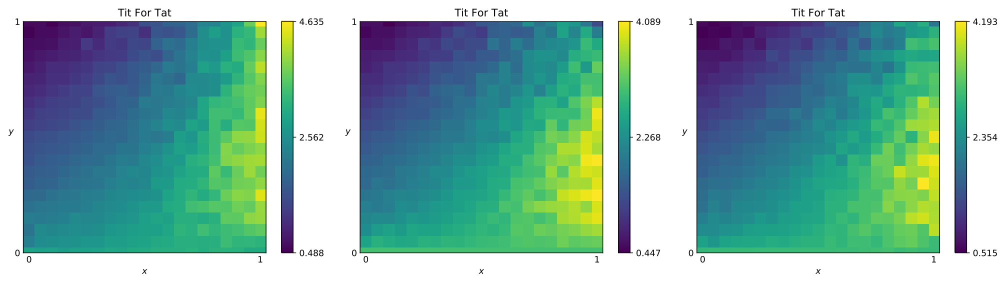
Another question is how is the performance affected if the player did not make deterministic actions but probabilistic.

1.5 Chapter Summary

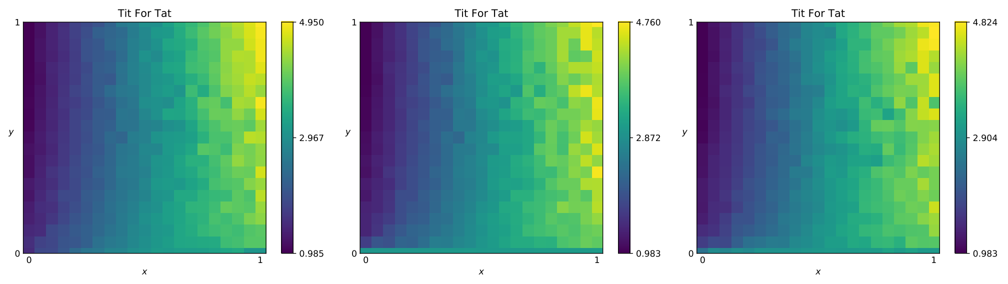
This Chapter introduced a total of 24 new IPD strategies based on LSTM networks. The LSTMs have received a lot of attention in machine learning due to their incredible results



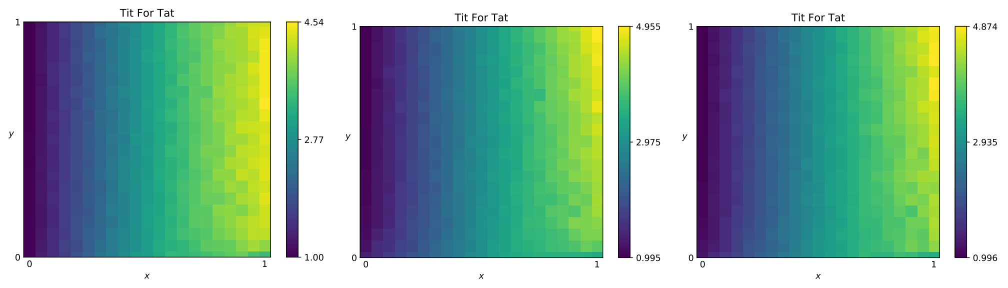
(a) Strategies trained on the entire training data set.



(b) Strategies trained against the top performing strategies.

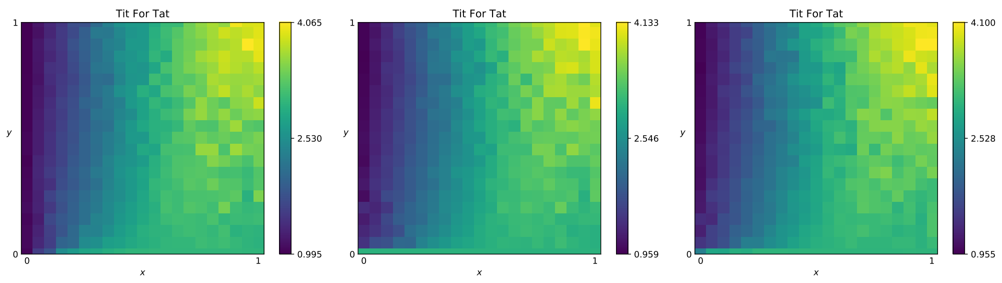


(c) Strategies trained against the across the ranks strategies.

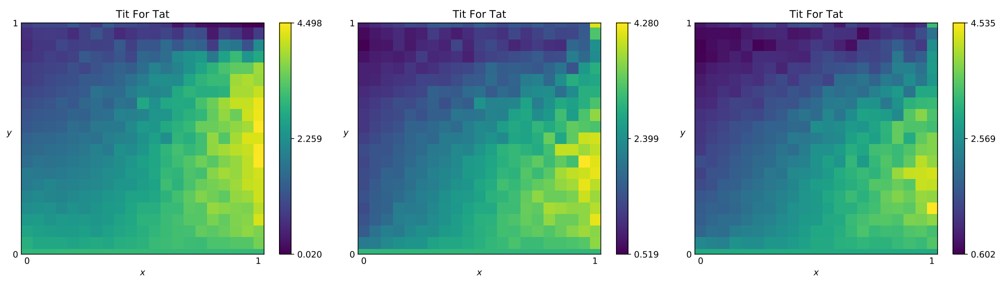


(d) Strategies trained against basic strategies.

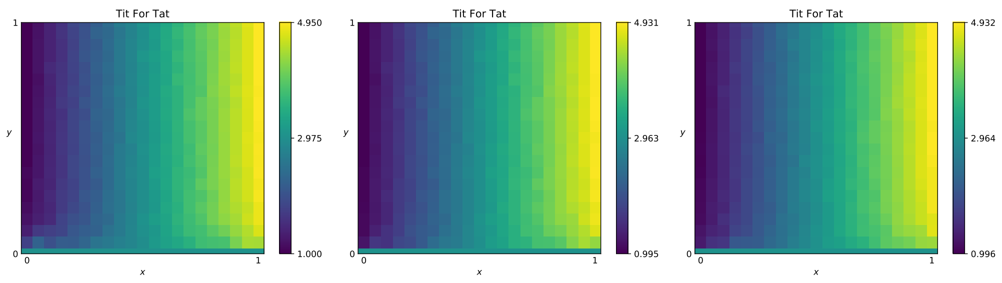
Figure 1.17: Ashlock's fingerprints for the LSTM strategies based on the sequence to sequence network when Tit For Tat is the probe strategy.



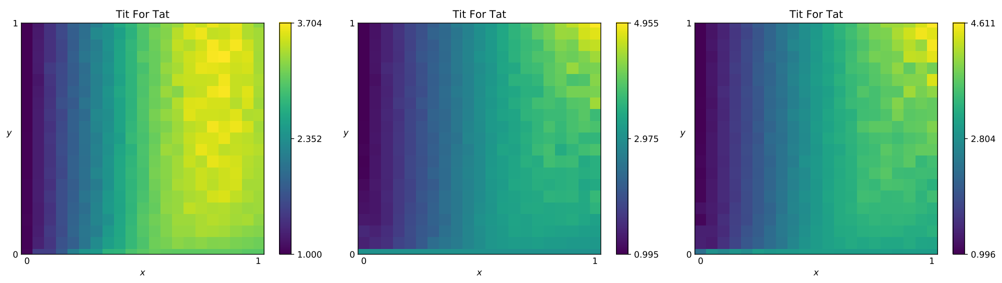
(a) Strategies trained on the entire training data set.



(b) Strategies trained against the top performing strategies.

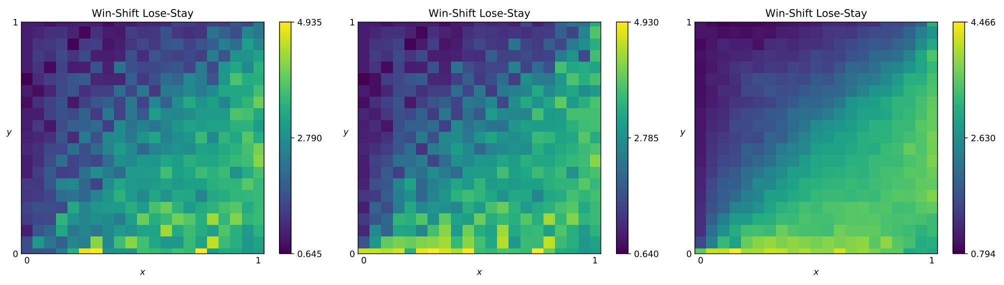


(c) Strategies trained against the across the ranks strategies.

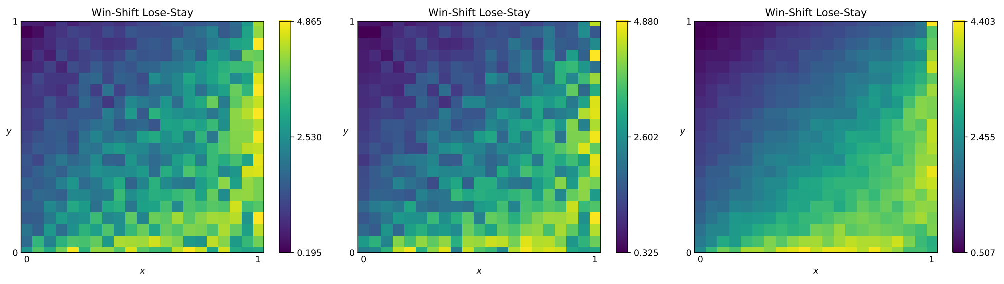


(d) Strategies trained against basic strategies.

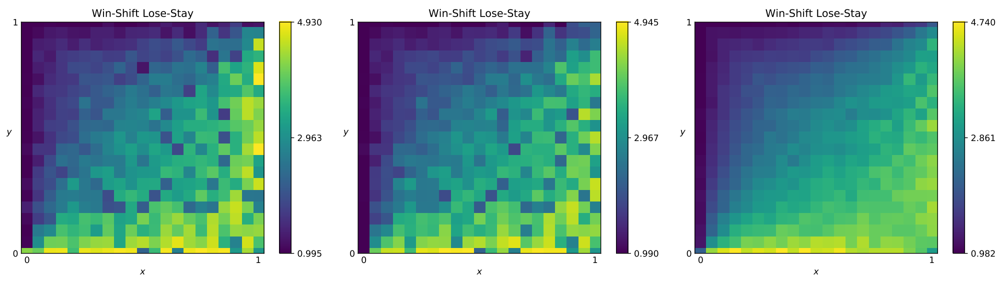
Figure 1.18: Ashlock's fingerprints for the LSTM strategies based on the sequence to probability network when Tit For Tat is the probe strategy.



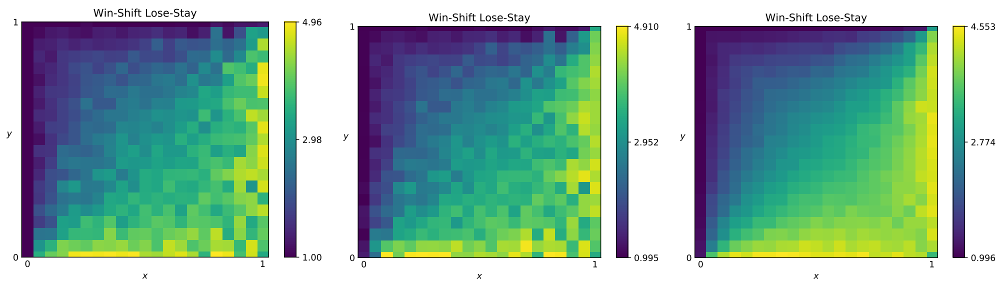
(a) Strategies trained on the entire training data set.



(b) Strategies trained against the top performing strategies.

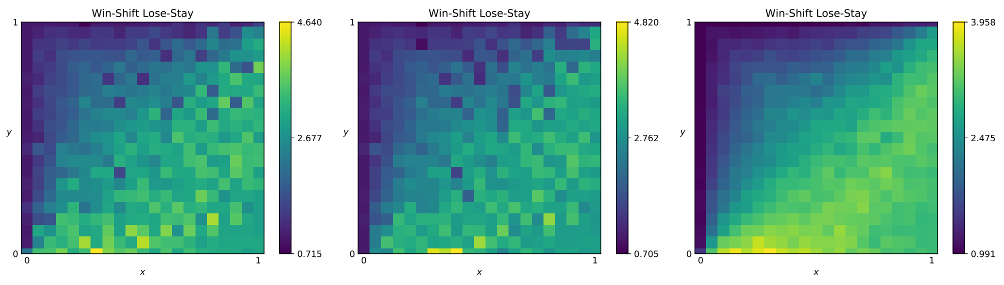


(c) Strategies trained against the across the ranks strategies.

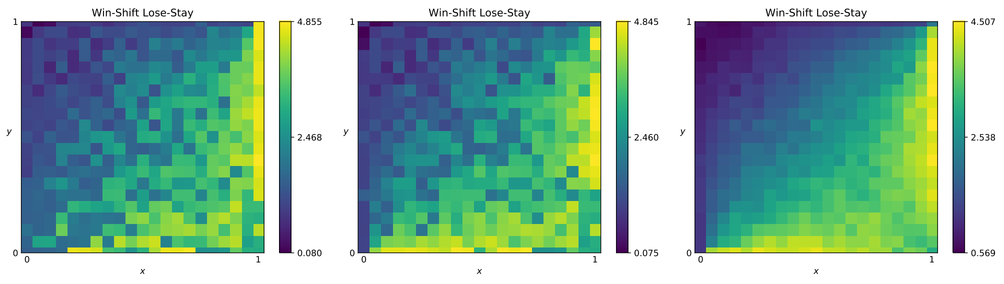


(d) Strategies trained against basic strategies.

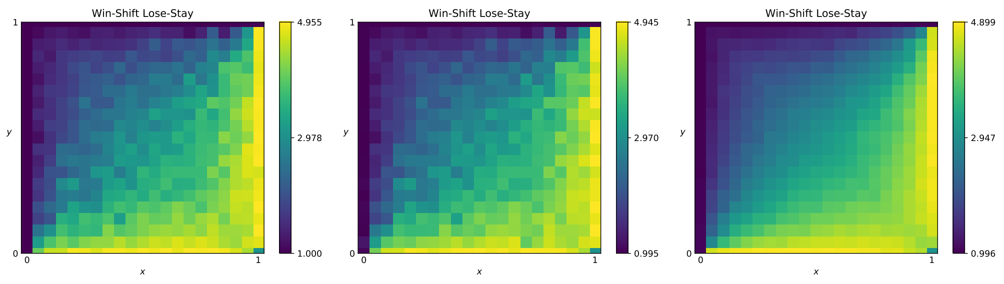
Figure 1.19: Ashlock's fingerprints for the LSTM strategies based on the sequence to sequence network when Pavlov is the probe strategy.



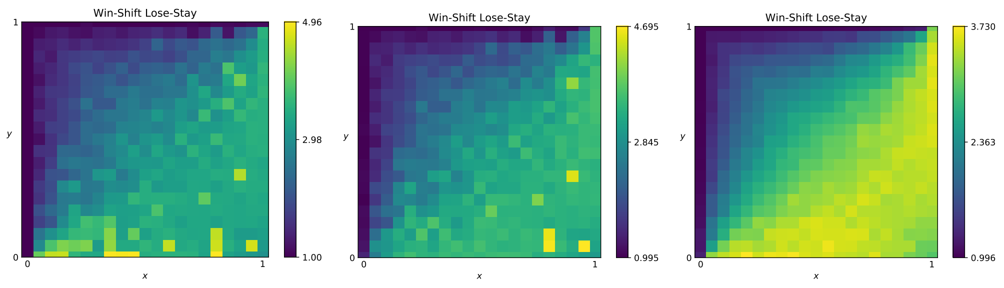
(a) Strategies trained on the entire training data set.



(b) Strategies trained against the top performing strategies.

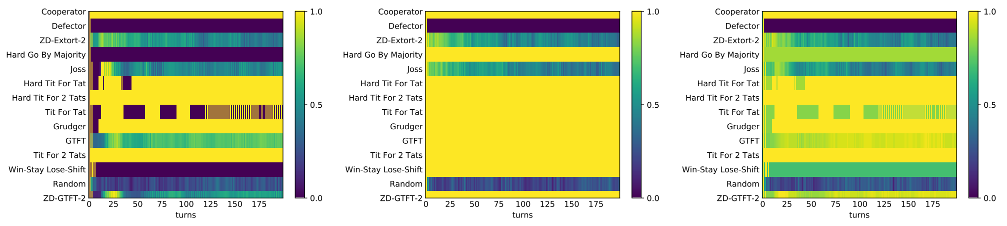


(c) Strategies trained against the across the ranks strategies.

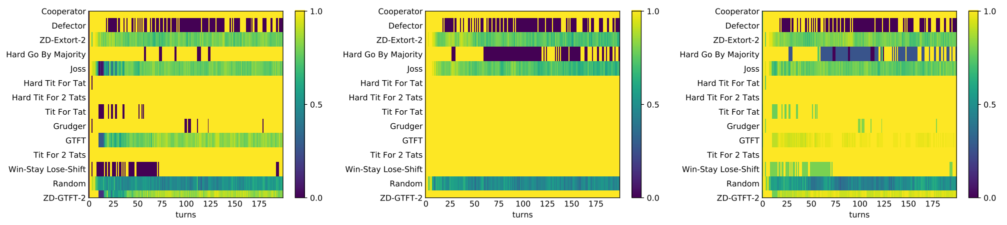


(d) Strategies trained against basic strategies.

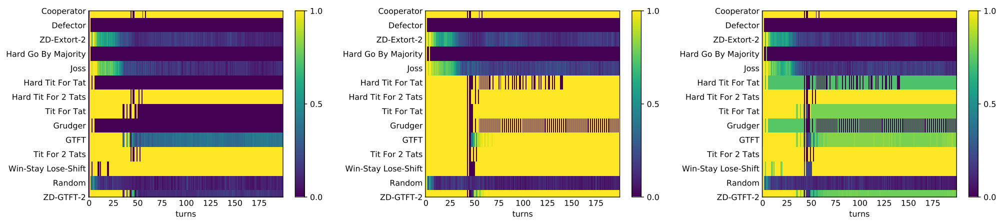
Figure 1.20: Ashlock's fingerprints for the LSTM strategies based on the sequence to probability network when Pavlov is the probe strategy.



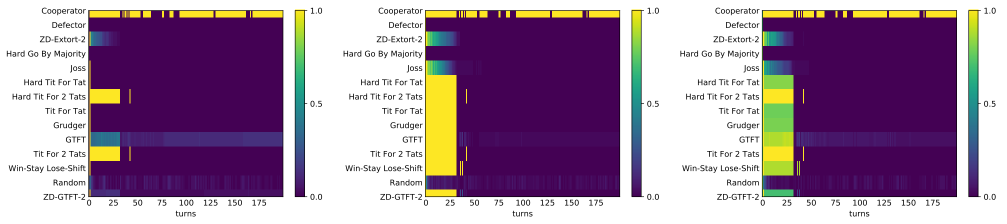
(a) Strategies trained on the entire training data set.



(b) Strategies trained against the top performing strategies.

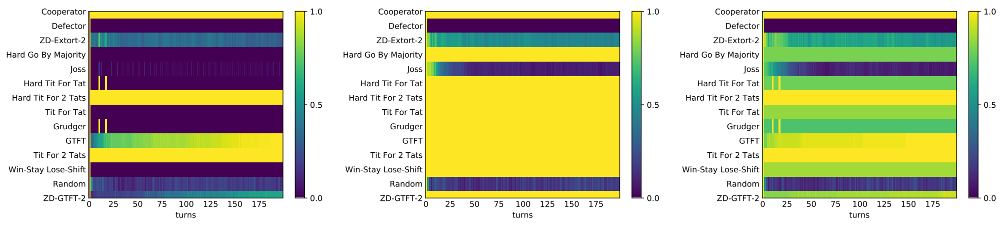


(c) Strategies trained against the across the ranks strategies.

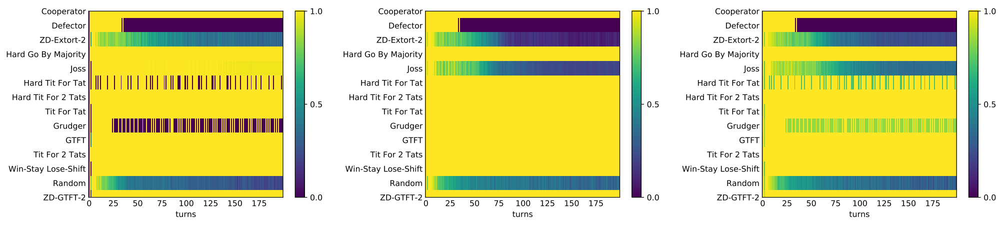


(d) Strategies trained against basic strategies.

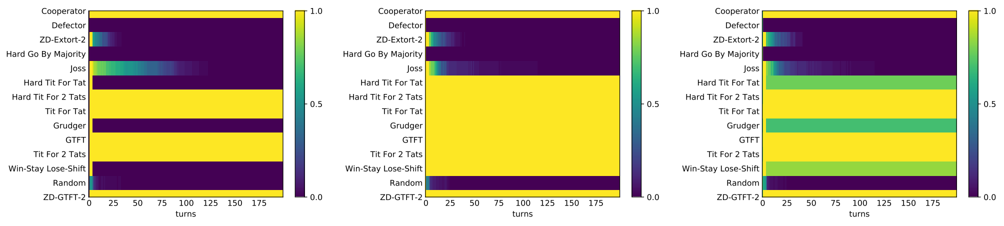
Figure 1.21: Transitive fingerprints for the LSTM strategies based on the sequence to sequence network against the list of opponents from [?].



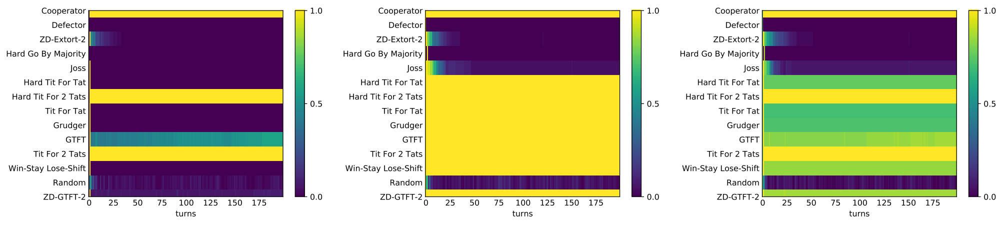
(a) Strategies trained on the entire training data set.



(b) Strategies trained against the top performing strategies.

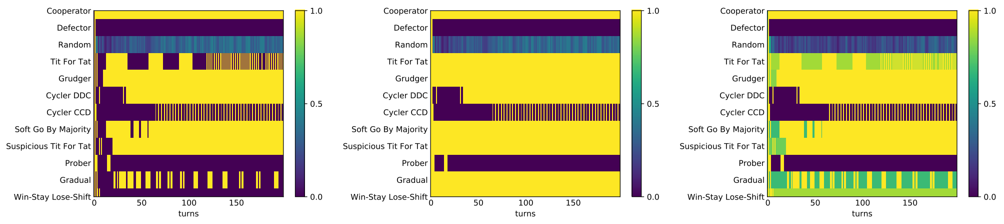


(c) Strategies trained against the across the ranks strategies.

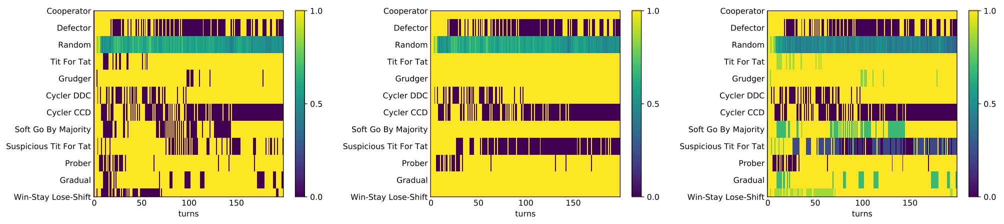


(d) Strategies trained against basic strategies.

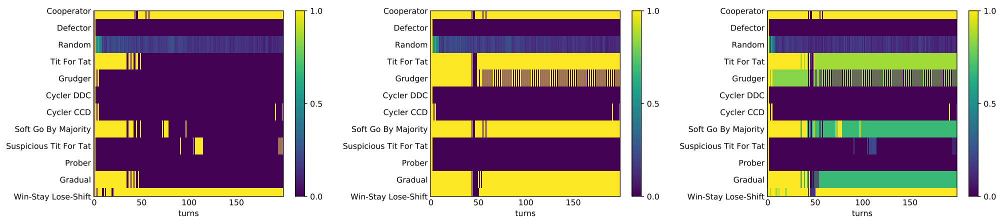
Figure 1.22: Transitive fingerprints for the LSTM strategies based on the sequence to probability network against the list of opponents from [?].



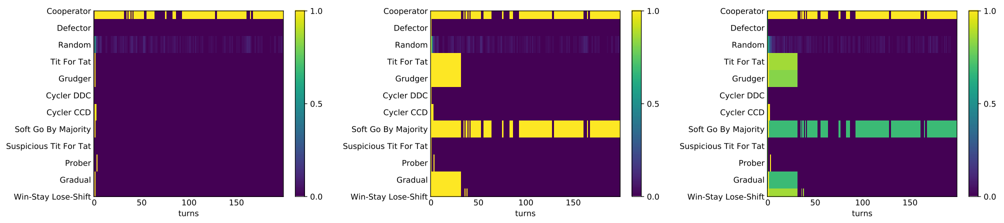
(a) Strategies trained on the entire training data set.



(b) Strategies trained against the top performing strategies.

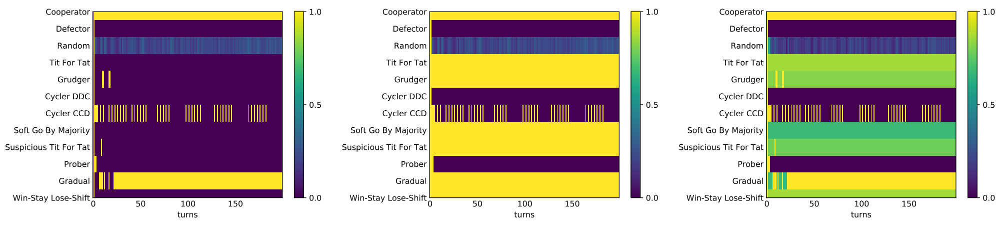


(c) Strategies trained against the across the ranks strategies.

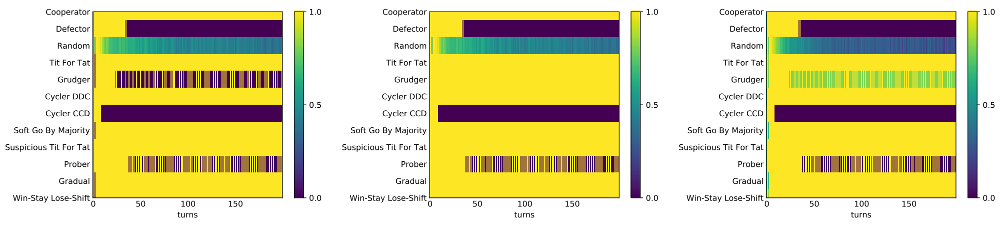


(d) Strategies trained against basic strategies.

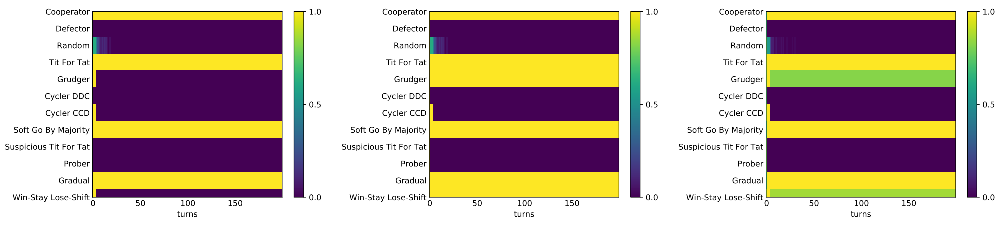
Figure 1.23: Transitive fingerprints for the LSTM strategies based on the sequence to sequence network against the list of opponents from [3].



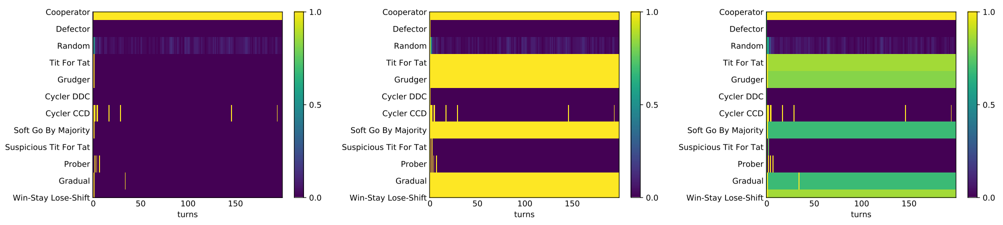
(a) Strategies trained on the entire training data set.



(b) Strategies trained against the top performing strategies.



(c) Strategies trained against the across the ranks strategies.



(d) Strategies trained against basic strategies.

Figure 1.24: Transitive fingerprints for the LSTM strategies based on the sequence to probability network against the list of opponents from [3].

on. Though ANNs have been considered in the IPD literature LSTM have not been given attention.

The advantage of using LSTM instead of feed forward networks is that the LSTM incorporate a mechanism of memory. The decision the make at time t is affected by their knowledge of time $t-1$. The networks were introduced in section. The networks were developed and trained using the open source package Keras. Two different LSTM networks were trained these are the sequence to sequence network and the sequence to probability network. The differences between the networks were discussed in section. The networks were trained not only on a single a training data set. The training data set has been made possible due to the best responses sequences generated in Chapter 6. That allowed the network to learn how to react to actions of well established IPD strategies. Sub sets of that training data set were consider to understand the effect of the training data on the network's performance in IPD match. The subsets included.

These were trained on the GPU.

A total of 8 networks were trained but these were 24 players. Their performance was explored in 300 tournaments. The results demonstrated that.

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] B. Beaufils, J. P. Delahaye, and P. Mathieu. Our meeting with gradual: A good strategy for the iterated prisoner's dilemma. 1997.

[4] 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.

[5] 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.

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

[7] 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.

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

[9] Zvi Drezner. Compounded genetic algorithms for the quadratic assignment problem. *Operations Research Letters*, 33(5):475–480, 2005.

[10] 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.

[11] 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.

[12] Nikoleta E. Glynatsi. Best response sequences in the prisoner's dilemma. `https://doi.org/10.5281/zenodo.3685251`, February 2020.

[13] 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.

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

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

[16] Edwin SH Hou, Nirwan Ansari, and Hong Ren. A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed systems*, 5(2):113–120, 1994.

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

[18] Gareth Jones, Peter Willett, Robert C Glen, Andrew R Leach, and Robin Taylor. Development and validation of a genetic algorithm for flexible docking. *Journal of molecular biology*, 267(3):727–748, 1997.

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

[20] Vincent Knight, Marc Harper, Nikoleta E. Glynatsi, and Owen Campbell. Evolution reinforces cooperation with the emergence of self-recognition mechanisms: An empirical study of strategies in the moran process for the iterated prisoner's dilemma. *PLOS ONE*, 13(10):1–33, 10 2018.

[21] Vincent Anthony Knight, Owen Campbell, Marc Harper, Karol M Langner, James Campbell, Thomas Campbell, Alex Carney, Martin Chorley, Cameron Davidson-Pilon, Kristian Glass, et al. An open framework for the reproducible study of the iterated prisoner's dilemma. *Journal of Open Research Software*, 4(1), 2016.

[22] J. Li, P. Hingston, S. Member, and G. Kendall. Engineering Design of Strategies for Winning Iterated Prisoner ' s Dilemma Competitions. 3(4):348–360, 2011.

[23] 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.

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

[25] 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.

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

[27] S. Mittal and K. Deb. Optimal strategies of the iterated prisoner's dilemma problem for multiple conflicting objectives. *IEEE Transactions on Evolutionary Computation*, 13(3):554–565, 2009.

[28] Eneko Osaba, Roberto Carballedo, Fernando Diaz, Enrique Onieva, P Lopez, and Asier Perallos. On the influence of using initialization functions on genetic algorithms solving combinatorial optimization problems: a first study on the tsp. pages 1–6, 2014.

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

[30] 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.

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

[32] Akhter Shameem and Roberts Jason. Multi-core programming-increasing performance through software multithreading. *Intel, Hillsboro, OR*, 2005.

[33] A. J. Stewart and J. B. Plotkin. Extortion and cooperation in the prisoner's dilemma. *Proceedings of the National Academy of Sciences*, 109(26):10134–10135, 2012.

[34] 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.

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

[36] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

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

[38] Jihoon Yang and Vasant Honavar. Feature subset selection using a genetic algorithm. In *Feature extraction, construction and selection*, pages 117–136. Springer, 1998.