Information Theory for Analyzing Neural Networks

Bård Sørngård

Master of Science in Computer Science
Submission date: June 2014
Supervisor: Keith Downing, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science
Abstract

The goal of this thesis was to investigate how information theory could be used to analyze artificial neural networks. For this purpose, two problems, a classification problem and a controller problem were considered. The classification problem was solved with a feedforward neural network trained with backpropagation, the controller problem was solved with a continuous-time recurrent neural network optimized with evolution.

Results from the classification problem shows that mutual information might indicate how much a particular neuron contributes to the classification. Tracking these neurons’ mutual information during training might serve as an indicator of their progression, including neurons in the hidden layers.

Results from the controller problem showed that time-delayed mutual information between a neuron and an environment variable might indicate what variable each neuron is estimating, and tracking this during evolution might tell us when this particular neuron started taking this role. Furthermore, unrolled transfer entropy appears to be a good measure for how neurons affect each other during simulation.
Sammendrag

Hensikten med denne masteroppgaven var å undersøke hvordan informasjonsteori kan brukes til å analysere kunstige nevrale nettverk. To problemer, et klassifiseringsproblem og et kontrollproblem ble undersøkt. Klassifiseringsproblemet ble løst med et feedforward nevralt nettverk trent med backpropagation, kontrollproblemet ble løst med et continuous-time rekurrent nevralt nettverk optimert med evolusjon.

Resultatene fra klassifiseringsproblemet viser at mutual information can indikere hvor mye et enkelt nevron bidrar til klassifiseringen. Å følge disse nevronenes mutual information under trening kan gi en indikasjon på hvordan de forbedres over tid, også for nevroner i skjulte lag.

Resultatene fra kontrollproblemet viser at time-delayed mutual information mellom et nevron og en variabel fra miljøet kan indikere hvilken variabel hvert nevron estimerer, og å spore denne under evolusjon kan fortelle oss ved hvilken generasjon dette nevronet påtok seg den rollen. Det viser seg også at unrolled transfer entropy kan være en god indikator på hvordan nevroner påvirker hverandre under simuleringen.
Contents

1 Introduction 9
  1.1 Motivation .............................................. 9
  1.2 Thesis Outline ....................................... 10

2 Information Theory 12
  2.1 Information Entropy .................................... 12
  2.2 Joint Entropy ......................................... 15
  2.3 Conditional Entropy .................................... 15
  2.4 Kullback Leibler Distance ............................. 17
  2.5 Mutual Information .................................... 18
  2.6 Conditional Mutual Information ....................... 20
  2.7 Time Series Analysis ................................... 21
  2.8 Differential Entropy .................................... 22
  2.9 Mixed Pair Mutual Information ....................... 23

3 Related Work 25
  3.1 Information Integration ............................... 25
  3.2 Information Dynamics ................................. 26
    3.2.1 Measure Unrolling ................................ 27
    3.2.2 Partial Information Decomposition .............. 28
    3.2.3 Information Flow .................................. 29
    3.2.4 Applications ..................................... 30
  3.3 Miscellaneous ......................................... 32

4 Methodology 33
  4.1 Continuous-Time Recurrent Neural Network ........... 33
  4.2 Numerical Methods for Ordinary Differential Equations 34
    4.2.1 The Euler Method ................................. 34
4.2.2 The Runge-Kutta Method 34
4.3 Transfer Functions 35
4.4 Probability Density Estimation 35

5 Problem 1: Classifier 39
5.1 Problem Description 39
5.2 Neural Network 39
5.3 Results 42
5.3.1 Mutual Information During Training 42
5.3.2 Mutual Information After Training 46
5.4 Discussion 46

6 Problem 2: Controller 50
6.1 Problem Description 50
6.2 Neural Network 52
6.3 Results 55
6.3.1 Trajectories 55
6.3.2 Time-Delayed Mutual Information 58
6.3.3 Transfer Entropy 62
6.4 Discussion 69

7 Summary 73
7.1 Conclusion 73
7.2 Future Work 75
List of Figures

2.1 Information entropy ........................................ 13
2.2 Venn diagram: Marginal entropy ............................ 14
2.3 Venn diagram: Joint entropy ................................ 15
2.4 Venn diagram: Conditional entropy ........................ 16
2.5 Venn diagram: Mutual information .......................... 19

3.1 Networks with high and low integration and complexity ... 25
3.2 Mutual information $I(Z; X, Y)$ decomposed ............... 28
3.3 Topology of ball-avoiding agent ............................ 31

4.1 Comparison of sigmoid functions ............................. 36
4.2 Kernel density estimation .................................... 37
4.3 Ordinary histogram vs. average shifted histogram ......... 38

5.1 Left, Right and Both classifiers as Boolean circuits ...... 40
5.2 Various retina classifications ................................. 40
5.3 Neural network topology: Classifier ........................ 41
5.4 Mutual information: During training, Left dataset ........ 43
5.5 Mutual information: During training, Right dataset ....... 44
5.6 Mutual information: During training, Both dataset ......... 45
5.7 Mutual information: After training, Left dataset .......... 46
5.8 Mutual information: After training, Right dataset ........ 47
5.9 Mutual information: After training, Both dataset .......... 47
5.10 Mutual information decomposed .............................. 48

6.1 Pole balancing problem ...................................... 51
6.2 Parameters: Pole-balancing problem ........................ 52
6.3 Neural network topology: Pole balancing .................... 54
6.4 Evolutionary algorithm parameters: Pole balancing ......... 54
6.5 Trajectories: Cart variables ........................................ 56
6.6 Trajectories: Pole variables ........................................ 56
6.7 Trajectories: Cart neurons .......................................... 57
6.8 Trajectories: Pole neurons .......................................... 57
6.9 Time-delayed mutual information: Cart neurons ............. 58
6.10 Time-delayed mutual information: Pole neurons .......... 59
6.11 $TDMI(N_{0,t}, x_{t})$ and fitness during evolution .......... 60
6.12 $TDMI(N_{1,t}, x_{t})$ and fitness during evolution .......... 60
6.13 $TDMI(N_{2,t}, \dot{\theta}_{t-2})$ and fitness during evolution ... 61
6.14 $TDMI(N_{3,t}, \theta_{t-1})$ and fitness during evolution ....... 61
6.15 Transfer entropy: Pole sensor to pole interneurons ....... 63
6.16 Average transfer entropy: Pole sensor to pole interneurons ... 64
6.17 Transfer entropy: Cart sensor to cart interneurons ....... 66
6.18 Average transfer entropy: Cart sensor to cart interneurons ... 67
6.19 Transfer entropy: Interneuron to motorneuron .............. 68
6.20 Histogram of transfer entropy: Interneuron to motorneuron at acceleration shift ........................................ 71
6.21 Information flow: Pole neurons .................................. 72
Chapter 1

Introduction

1.1 Motivation

Artificial neural networks are a popular form of optimization. Initially inspired by neuroscience to imitate neural structures found in the brain, they have been used to solve problems where analytical solutions have failed.

One often voiced criticism of artificial neural networks, as well as many machine learning models in general, are their "black box" nature [26]. Gaining insight into how a particular neural network solves a given problem is troublesome and often the only way to improve them is by experimentation. For this reason, their success outside of the computer science society has been limited. Indeed, despite their powerfulness, artificial neural networks are rarely a part of introductory optimization classes.

One aspect that separates neural networks from many other forms of machine learning is that they are distributed. Neural networks can be represented by a graph with weighted edges and nodes with values, and different parts of these networks solve different subproblems. The solutions to these subproblems are then combined together to solve the actual problem. Exactly how this problem is divided into subproblems, and what these subnetworks solve can be a great wonder. If it’s hard to know how it works, then knowing how to improve it will be even harder.

What we can do, then, is try looking at the interaction between neurons’ values, and systematically measuring how they change according to external factors. The most well-known statistic to measure dependence between variables is correlation. However, one important drawback with correlation
is that it is only able to pick up linear relationships; in most other cases it is practically useless. The relationships between activation levels in a neural network are usually determined by a sigmoid function; they are rarely linear.

A better suited statistic for this problem then, is mutual information from information theory. Information theory, initially introduced by Shannon in 1948 to measure the number of bits needed to send a message over a noisy channel, has become an immensely popular toolbox to measure the influence between variables. In fact, it became so popular Shannon himself published an article [21] expressing his concern over the fact it was used for several purposes he did not agree with.

What makes mutual information so interesting is that, unlike correlation, it is able to pick up non-linear dependencies between variables. Given two random variables $X$ and $Y$, it looks at the divergence of their probability distribution $p(x, y)$ from $p(x)p(y)$ to determine how dependent - or independent - they are.

With the motivation to make artificial neural networks more intuitive, my first research question was: How can information theory be used to help a user understand the dynamics of an artificial neural network? If we can somehow succeed in making artificial neural networks more intuitive, I believe this would help spread the use of artificial neural networks to other fields outside of the computer science society.

Then, my second and more ambitious research question was: How can a user use this insight to improve them? How can a user with the knowledge he gains from using the tools introduced in this thesis build better neural networks? Although one might argue that any insight would be helpful with this regard, I will still try to come up with some suggestions.

\section{Thesis Outline}

In Chapter 2, I will cover all information theory needed to understand the results of this thesis. The reader is assumed to know elementary probability theory, although no knowledge of information theory is needed. Since a lot of the formulae introduced in this chapter by themselves can seem daunting and hard to understand, I have tried drawing parallels to set theory where applicable. Likewise, as motivation, I try mentioning examples of how each statistic has been used in practice.

In Chapter 3, I give an overview of how others have previously applied in-
formation theory to neural networks. My main focus will be on the work done by Williams [28], as I will borrow some of his ideas for my own experiments.

In Chapter 4, I will give an overview of methodology used in this thesis not necessarily related to information theory.

In Chapter 5, I will look at a concrete problem, a feedforward neural network acting as a classifier. What makes this particular classification problem special is that it can be broken up into intuitive subtasks, and different modules of the neural network solve these subtasks and their results are then combined at the output neuron. I then measure the mutual information between these neurons’ activation levels and the classification both during and after training, and comment on these results.

In Chapter 6, I will look at a controller problem which is solved by a continuous-time recurrent neural network. In this chapter, as in the previous, I will measure mutual information between these neurons’ activation levels and environment variables as well as track them during evolution. But since this is a dynamic neural network, I will also study how its neurons change and respond to external events during simulation with unrolled transfer entropy.

In Chapter 7, I will summarize the contributions of this thesis and relate it to the work of others. Then, I round off by writing down my own thoughts for future work.

The reader is assumed to already be familiar with artificial neural networks and how to optimize them, both with learning and evolution. Solid knowledge of elementary probability theory will probably be needed to understand the results of this thesis, although no knowledge of information theory should be needed; I have covered everything I think will be necessary in the following Chapter.
Chapter 2

Information Theory

2.1 Information Entropy

Given a discrete random variable \( X \in \{x_1, x_2, ..., x_n\} \), with probabilities \( P(X = x_i) = p(x_i) \) we would like to define a function \( H(X) \) to measure its uncertainty. A function doing this is the information entropy, as defined by Shannon [18].

\[
H(X) = \sum_x p(x) \log_2 \frac{1}{p(x)} \tag{2.1}
\]

\[
= - \sum_x p(x) \log_2 p(x) \tag{2.2}
\]

Both of these are commonly seen in the literature. The unit for information entropy is called a bit.

Looking at Figure 2.1, it is easy to see why \( H(X) \) is supposed to measure the uncertainty of \( X \). When it is uniform, it is at its maximum. When it is deterministic, it is 0. As \( H(X) \) is concave, the more \( X \) strays away from being completely unpredictable, the more it is punished by \( H(X) \).

\(-\log_2 x\) is often called the "surprisal function", and \( H(X) \) could be seen in the light of the expected surprise of \( X \) as

\[
H(X) = E[-\log_2 X] = - \sum_x p(x) \log_2 p(x) \tag{2.3}
\]

The attentive reader might notice that as \( p(x) \to 0 \), the surprisal function
Figure 2.1: Information entropy $H(X)$ plotted for a Bernoulli variable $X$ with probabilities $p$ and $1 - p$.

$- \log_2 p(x) \to \infty$. That means we might have to calculate $0 \cdot \infty$, which is undefined. Fortunately, using l’Hopital’s rule, we can easily find its limit.

$$
\lim_{p \to 0} p \log_2 p = \lim_{p \to 0} \frac{\ln p}{\ln 2} \frac{1}{p}
$$

$$
= \lim_{p \to 0} \frac{p \ln 2}{p}
$$

$$
= \lim_{p \to 0} -p \ln 2
$$

$$
= 0 \quad (2.4)
$$

In other words, a probability of 0 does not contribute to the entropy. If we imagine implementing this in a computer program where we iterate over all possible outcomes $x \in X$ and summing up their individual contribution in a loop, we can just skip all $x$ where $p(x) = 0$.

The reader should also note that since discrete probabilities $p \in [0, 1]$, $H(X)$ will always be positive. Also, the probability distribution with the
highest entropy is the uniform distribution, which means that for any $X$ with $n$ possible outcomes

$$H(X) \leq n \frac{1}{n} \log_2 n$$

$$\leq \log_2 n$$ (2.5)

That gives us the constraint that $H(X) \in [0, \log_2 n]$ for any $X$ with $n$ possible outcomes.

One way information entropy is used is to verify random number generators [11]. Consider an alleged random number generator that is supposed to draw a completely random number $X \in \{1, 2, ..., n\}$. Using information theory, we can measure its entropy $H(X)$ and expect it to be $\log_2 n$ if it is completely random; if it isn’t, that means our random number generator is skewed.
2.2 Joint Entropy

Given two discrete random variables $X$ and $Y$, their joint entropy $H(X, Y)$ [5] is

$$H(X, Y) = - \sum_x \sum_y p(x, y) \log_2 p(x, y)$$  \hspace{1cm} (2.6)

If we generalize joint entropy to an arbitrary number of random variables, we get the following

$$H(X_1, X_2, ..., X_n) = - \sum_{x_1} \sum_{x_2} \cdots \sum_{x_n} p(x_1, x_2, ..., x_n) \log_2 p(x_1, x_2, ..., x_n)$$  \hspace{1cm} (2.7)

2.3 Conditional Entropy

Given two discrete random variables $X$ and $Y$, their conditional entropy $H(X|Y)$ [5] is defined as

$$H(X|Y) = - \sum_x \sum_y p(x, y) \log_2 \frac{p(x, y)}{p(y)}$$  \hspace{1cm} (2.8)

From this equation we can also derive the following interesting identity
CHAPTER 2. INFORMATION THEORY

16

\[ H(X|Y) = - \sum_x \sum_y p(x,y)(\log_2 p(x,y) - \log_2 p(y)) \]

\[ = - \sum_x \sum_y p(x,y) \log_2 p(x,y) - \left( - \sum_x \sum_y p(x,y) \log_2 p(y) \right) \]

\[ = - \sum_x \sum_y p(x,y) \log_2 p(x,y) - \left( - \sum_y p(y) \log_2 p(y) \right) \]

\[ = H(X,Y) - H(Y) \]  

(2.9)

Figure 2.4: Conditional entropy

\( H(X|Y) \) tells us how unpredictable \( X \) will be, given that we already know \( Y \). If \( X \) and \( Y \) are statistically independent, that means \( H(X|Y) = H(X) \).

Again, as motivation, we look at an example from cryptography. Consider we would like to encrypt a message \( M \) in the alphabet \( L_0 \) to an encrypted message, a ciphertext, \( C \) in the alphabet \( L_1 \) with a key \( K : L_0 \to L_1 \).
Considering the entire space $M$ of plaintext messages, some of these will be more likely than the others and with a poor cipher, attacks such as frequency analysis can be used. For instance, in the English language, the most common character is ' ' (whitespace) with a frequency of 0.182, followed by 'e' with a frequency of 0.107. With a simple monoalphabetic cipher, every character in $M$ is mapped with the key $K$, and by comparing the distribution of the characters of $C$ with the distribution of the frequency of characters used in $L_0$, it will be possible to recover the key $K$ and ultimately the plaintext message $M$ [20]. A desirable property of a cipher then would be something Shannon termed ideal secrecy, $H(K|C) = H(K)$ [19], that is, information about the ciphertext $C$ does not give an attacker any information about the key $K$.

### 2.4 Kullback Leibler Distance

A common measure used to compare two probability distributions $p$ and $q$ is their Kullback Leibler distance, defined as

$$D(p||q) = \sum_x p(x) \log_2 \frac{p(x)}{q(x)}$$

$$= \sum_x p(x) D(p(x)||q(x))$$  \hspace{1cm} (2.10)

If $p(x) = q(x)$ for a particular $x$, then it will not contribute to $D(p||q)$. In other words, if for all $x$, $p(x) = q(x)$, then $D(p||q) = 0$. A low Kullback Leibler distance means $p$ and $q$ are similar, a high Kullback Leibler distance means they are dissimilar. Also, note that in general $D(p||q)$ isn’t symmetric. I would also like to stress the difference between $D(p||q)$ and $D(p(x)||q(x))$ where the first is the Kullback Leibler distance over all $x$, whereas the latter is the Kullback Leibler distance for a specific $x$.

Kullback Leibler is often used to analyze other information-theoretic statistics. Likewise, I will use Kullback Leibler distance to show interesting properties about other statistics in Section 2.5, Section 2.6 and Section 2.7, but not apply it on any data myself.
2.5 Mutual Information

Given two discrete random variables X and Y, their mutual information \( I(X; Y) \) [5] is defined as

\[
I(X; Y) = \sum_x \sum_y p(x, y) \log_2 \left( \frac{p(x, y)}{p(x)p(y)} \right)
\]

\[
= H(X) + H(Y) - H(X, Y)
\]

If \( x \in X \) and \( y \in Y \) are statistically independent, i.e. \( p(x, y) = p(x)p(y) \), that means that particular combination of \( x \) and \( y \) doesn’t contribute anything to the mutual information. From Equation 2.11 we can derive the following interesting identity

\[
I(X; Y) = \sum_x \sum_y p(x, y) \log_2 \frac{p(x|y)}{p(x)}
\]

\[
= \sum_x \sum_y p(x, y) (\log_2 p(x|y) - \log_2 p(x))
\]

\[
= -\sum_x \sum_y p(x, y) \log_2 p(x) + \sum_x \sum_y p(x, y) \log_2 p(x|y)
\]

\[
= -\sum_x p(x) \log_2 p(x) - (-\sum_x \sum_y p(x, y) \log_2 p(x|y))
\]

\[
= H(X) - H(X|Y)
\]

One way to look at this equation is: “If I know the value of \( Y \), how much information do I have about \( X \)?”. Note from Equation 2.11 we might as well swap \( x \) and \( y \) and get

\[
I(X; Y) = H(Y) - H(Y|X)
\]

\[
= H(X) - H(X|Y)
\]

\[
= I(Y; X)
\]

In other words, mutual information is symmetric. Since we know that any conditional entropy \( H(Y|X) \geq 0 \), we can also infer that
\[ I(X; Y) \leq H(Y) \]
\[ I(X; Y) \leq H(X) \]
\[ I(X; Y) \leq \min(H(X), H(Y)) \] (2.14)

If you think about it, this should make perfect sense since a random variable cannot share more information with another variable than it actually contains itself!

Again, looking at Equation 2.11, we can also express it using the Kullback-Leibler distance

\[ I(X; Y) = \sum_x \sum_y p(x, y) \log_2 \frac{p(x, y)}{p(x)p(y)} \]
\[ = \sum_x \sum_y p(x, y) D(p(x, y) || p(x)p(y)) \] (2.15)

From Bayes’ formula we know that if two random variables \( X \) and \( Y \) are independent, then \( p(x, y) = p(x)p(y) \). This makes it clearer to us that \( I(X; Y) \) is a measure of how dependent \( X \) and \( Y \) are.

![Figure 2.5: Mutual information](image)

While originally meant by Shannon to be used to quantify the number of bits needed to transmit a message over a noisy channel, mutual information has due to the fact that it is able to pick up non-linear dependencies proven to be widely successful in a number of fields. For instance, in machine learning
it has been used to pre-select features with high mutual information against
the classification for classification algorithms [25] and as a similarity function
in clustering [4].

2.6 Conditional Mutual Information

We can extend mutual information from Equation 2.11 to give us the condi-
tional mutual information between $X$ and $Y$, given knowledge about $Z$

\[
I(X; Y|Z) = H(X|Z) - H(X|Y, Z)
\]

\[
= \sum_x \sum_y \sum_z p(x, y, z) \log_2 \frac{p(x, y|z)}{p(x|z)p(y|z)}
\]

(2.16)

Looking at its Kullback Leibler distance we get

\[
I(X; Y|Z) = H(X|Z) - H(X|Y, Z)
\]

\[
= \sum_x \sum_y \sum_z p(x, y, z)D(p(x, y|z)||p(x|z)p(y|z))
\]

(2.17)

As its Kullback Leibler distance shows us, $I(X; Y|Z)$ tells us how much
$p(x, y|z)$ deviates from $p(x|z)p(y|z)$; that is, for datapoints $(X, Y, Z)$ with
similar $Z$ values, how dependent are $X$ and $Y$ on each other?

As an application of conditional mutual information, notice that the feature
pre-selection for classification algorithms in Section 2.5 doesn’t take into
account that two features might provide the same information about the clas-
sification. Fleuret [6] came up with a pre-selection algorithm that conditions
the mutual information against previously selected features. For the first
feature $X$ selected, only the mutual information $I(X; Y)$ against the clas-
sification $Y$ is considered; but then the next features are selected according to
the conditional mutual information $I(X; Y|X_i)$ where $X_i$ are the previously
selected features. This has proven to give a lower error rate than just using
the mutual information for some problems.
2.7 Time Series Analysis

Considering two discrete time series $X_t$ and $Y_t$, we would like to measure how much information is shared between them, i.e. how statistically dependent one is on the other. One straightforward way of doing this is with what is called the time-delayed mutual information (TDMI), defined as

$$TDMI(X_t; Y_t) = I(X_t; Y_{t-1})$$ (2.18)

given a time delay $1$. Likewise, we also introduce a statistic called transfer entropy [16], which lets us quantify how much $X_t$ is affected by $Y_t$, defined as

$$T_{Y \rightarrow X} = I(X_t; Y_{t-1}|X_{t-1})$$

$$= \sum_x \sum_y \sum_y p(x_{t-1}, y_{t-1}, y_t) \log_2 \frac{p(x_t|y_{t-1}, x_{t-1})}{p(x_t|x_{t-1})}$$ (2.19)

What separates transfer entropy from TDMI is that transfer entropy is conditioned on the last state. In other words, what we are looking at with transfer entropy is: given $X_t$, how much information is shared between $Y_t$ and $X_{t+1}$? We can also have a look at them with the Kullback Leibler distance

$$TDMI(X_t; Y_t) = \sum_x \sum_y p(x_t, y_{t-1}) D(p(x_t, y_{t-1})||p(x_t)p(y_{t-1}))$$ (2.20)

$$T_{Y \rightarrow X} = \sum_x \sum_y \sum_y p(x_{t-1}, y_{t-1}, y_t) D(p(x_t|y_{t-1}, x_{t-1})||p(x_t|x_{t-1}))$$ (2.21)

As these Kullback Leibler distances demonstrate, TDMI tells us how much $p(x_t, y_{t-1})$ deviates from $p(x_t)p(y_{t-1})$, i.e. the statistical dependence between $X_t$ and $Y_{t-1}$. On the other hand, transfer entropy tells us how much $p(x_t|y_{t-1}, x_{t-1})$ deviates from $p(x_t|x_{t-1})$; in other words how much $Y_t$ affects the transitional probabilities $p(x_t|x_{t-1})$ of $X_t$.

In his original paper on transfer entropy [16], Schreiber considers the breath rate and heart rate over time of a person with a sleep disorder called sleep apnea, which causes long irregular delays in the breathing pattern of
In these time series data, the heart rate is the leading indicator of the breathing rate, but their time-delayed mutual information is nearly symmetrical. On the other hand, the transfer entropy from the heart rate to the breathing rate was consistently higher than vice versa.

### 2.8 Differential Entropy

Information entropy can also be generalized to continuous random variables. Let $X$ be a continuous random variable with a density $f(x)$ and support set $S$. Its differential entropy \[5\] then is

$$h(X) = -\int_S f(x) \log_2 f(x) dx$$ (2.22)

Recall from Chapter 2.1 that since $p \in [0, 1]$, $H(X) \geq 0$. However, for a continuous random variable $X$, the only constraint we have is $f(x) \geq 0$. It is fully possible, and does indeed often happen, that $f(x) > 1$, which can make $h(X)$ negative. That means we can have negative information, a counterintuitive concept.

One way of estimating $h(X)$ is by bucketing $X$ into equal intervals of $\Delta$. For this quantized version of $X$, we will use the notation $X^\Delta$. We then get the following result

$$H(X^\Delta) = -\sum_x \Delta f(x) \log_2(\Delta f(x))$$

$$= -\sum_x \Delta f(x) \log_2 f(x) - \sum_x \Delta f(x) \log_2 \Delta$$

$$= -\int_S f(x) \log_2 f(x) dx - \int_S f(x) dx \log_2 \Delta$$

$$= -\int_S f(x) \log_2 f(x) dx - \log_2 \Delta$$

$$= h(X) - \log_2 \Delta$$ (2.23)

As this equation shows us, $H(X^\Delta) \neq h(X)$. In fact, as we decrease the interval $\Delta$, $h(X)$ will stay constant while $\log_2 \Delta$ will blow up and our estimate $H(X^\Delta)$ becomes even more off. Fortunately, since we already know the interval size $\Delta$, getting $h(X)$ from $H(X^\Delta)$ is as simple as just adding $\log_2 \Delta$. 
Do note, however, that a prerequisite for this proof is that $\Delta \to 0$, that is $X$ has to be binned into infinitely small intervals.

\[ h(X) = H(X^\Delta) + \log_2 \Delta \]  

(2.24)

Consider two continuous random variables $X$ and $Y$ that both have been quantized with the same interval $\Delta$. Their mutual information [5] is then defined as

\[ I(X;Y) = -\int \int f(x,y) \log_2 \frac{f(x,y)}{f(x)f(y)} dxdy = h(Y) - h(Y|X) = (H(X^\Delta) + \log_2 \Delta) - (H(Y^\Delta|X^\Delta) + \log_2 \Delta) = I(X^\Delta;Y^\Delta) \]  

(2.25)

It is interesting to note that $I(X;Y) = I(X^\Delta;Y^\Delta)$. This means it inherits several desirable attributes from its discrete counterpart, such as being strictly positive and symmetric. On the other hand, its maximum will be $\log_2 \Delta$ - dependant upon its bucket size - which means its value by itself isn’t as intuitive as in the discrete case.

### 2.9 Mixed Pair Mutual Information

In Section 2.5 and Section 2.8 we defined the mutual information $I(X;Y)$ for pairs of discrete and continuous variables respectively. However, in the case where we have one discrete variable $X$ and continuous variable $Y$, a mixed pair, we can use the mutual information formula defined by Pardy [14] as

\[ I(X;Y) = \sum_i p(x_i) \int_S f_i(y) \log_2 \frac{f_i(y)}{f(y)} dy \]  

(2.26)

Here $f(y)$ represents the probability density function of $Y$, while $f_i(y)$ represents the probability density function of $Y$ where also $X = x_i$. This is
a pretty recent statistic, less than one year old at the time this thesis was written. It seems like the mutual information between a discrete and continuous random variable hasn’t garnered that much attention. Nevertheless, Pardy uses the mixed pair mutual information statistic in his PhD thesis to explore relations between discrete DNA sequencing data and continuous gene expression data found in rats.
Chapter 3

Related Work

3.1 Information Integration

The best known work within this field is probably the work by Tononi, Sporns and Edelman (TSE), although as neuroscientists they’ve looked at biological neural networks instead of artificial neural networks.

Looking back at Equation 2.11, we can generalize it to more than just two variables. TSE define information integration [24] for a set of random variables \( X = \{X_1, X_2, ..., X_n\} \) as

\[
I(X) = \sum_i H(X_i) - H(X) \tag{3.1}
\]

Like mutual information, information integration \( H(X) \) tells us how statistically dependent a set of variables \( X = \{X_1, X_2, ..., X_n\} \) are upon each other.

Furthermore, they introduce another statistic called neural complexity

(a) Low integration, low complexity
(b) High integration, high complexity
(c) High integration, low complexity

Figure 3.1: Networks with high and low integration and complexity
Let $X$ be a set of variables, its neural complexity $C_N(X)$ then is defined as

$$C_N(X) = \sum_{k=1}^{n}[(k/n)I(X) - \langle I(X^k_j) \rangle]$$

(3.2)

where $\langle I(X^k_j) \rangle$ is the average integration of all subsets of $X$ with size $k$.

Neural networks with low integration overall will also have low complexity. Likewise, neural networks with high intergration overall but where there are no subsets that stand out by having higher integration than the others will have low complexity as well. The networks that will have high complexity are networks with high integration where there also are subsets standing out by having higher integration than the others: in other words, one can say the neural network is "specialized" and consists of clusters of neurons working together and separated from the others. In Figure 3.1 I have shown some figures demonstrating this concept.

TSE hypothesize that what we think of as conscious experience is related to the amount of information a system is able to integrate. In their work they have reconstructed neural networks from the brain in simulations and measured their ability to integrate information [22]. Among their results, they have found that the thalamocortical system, which is believed to be very important for a human to experience what we consider to be conscious experience, is able to integrate a high amount of information. Likewise, parts of the brain believed not to be related to our conscious experience, such as the cerebellum, had low integration.

TSE also believe biological neural networks as they appear in nature usually are both highly integrated and specialized, which means they will have high complexity as well. Their simulations of biological neural networks such as those found in the brain have been found to have high complexity [23]. Yaeger [29] has evolved small-world neural networks and found that using fitness functions that maximize complexity can evolve faster than fitness function without.

### 3.2 Information Dynamics

Traditional use of information theory has usually focused on the study of static random variables. However, in the case of recurrent neural networks
with time-varying inputs, it is also interesting to study how information is shared between neurons at each specific timestep. The way this is done is by unrolling the statistics introduced in Chapter 2. Instead of considering the entropy contribution of all possible values, the probability density functions over the entire simulation is generated. Then, at every timestep, only the information in that particular state is considered. The most thorough work in this field has been done by Williams [28], who calls this information dynamics.

### 3.2.1 Measure Unrolling

A downside about all of the information-theoretic measures introduced so far is that they are all static. In several cases it can for instance be interesting to analyze the shared information between two stochastic processes $X_t$ and $Y_t$ at every timestep. Realizing that mutual information is in fact averaged over all possible combinations of $x$ and $y$, we can instead consider the contribution of that particular combination to the mutual information. We define the point-wise mutual information (PMI) as

$$pmi(x, y) = -p(x, y) \log_2 \frac{p(x, y)}{p(x)p(y)} \quad (3.3)$$

As you might have noticed, the PMI is the term used in the definition of mutual information in Equation 2.11, except that it is not summed up over all possible combinations of $X$ and $Y$. Instead, considering $X_t$ and $Y_t$ are two time series, we can find an estimate $p(x, y)$ from these time series and then, using point-wise mutual information, measure how much information $X_t$ and $Y_t$ share at each particular timestep.

For the point-wise mutual information, we unrolled mutual information for both variables. Alternatively, we can instead unroll it for only one of the variables. We call this specific information, which is implicitly defined as

$$I(X; Y) = \sum_x p(x) I_{spec}(X = x; Y) \quad (3.4)$$

Since the only constraint is that the expected specific information has to sum up to the mutual information as specified by Equation 3.4, there are several alternatives for $I_{spec}(X = x; Y)$ that can be used. However, the one used by Williams is
\[ I(X = x; Y) = \sum_y p(x|y) \log_2 \frac{p(x|y)}{p(x)} \] (3.5)

### 3.2.2 Partial Information Decomposition

One of the new discoveries from Williams' PhD thesis is his way of dividing information into unique information, redundant information and synergistic information. The unique information between \( X \) and \( Z \) is the amount of information shared only by \( X \) and \( Z \), denoted as \( I(X; Z) \). Likewise, he defines the redundant information about \( Z \) between \( X \) and \( Y \) to be the information about \( Z \) that is shared by both \( X \) and \( Y \); by knowing either of them we will also know their redundant information \( I(Z; \{X\}\{Y\}) \). At last, he also defines synergistic information to be the information about \( Z \) available only if we know both \( X \) and \( Y \), denoted \( I(Z; \{X,Y\}) \).

In his PhD thesis, Williams comes up with a list of axioms the partial information function \( I(Z; X_1, X_2, ..., X_n) \) will have to satisfy, and proves that the following candidate satisfies all of them

\[
I_{\text{min}}(Z; X_1, X_2, ..., X_n) = \sum_z \min_{X_i} I(Z = z, X_i) \tag{3.6}
\]

As Figure 3.2 shows, the inclusion-exclusion principle applies to these new
measures, which makes calculating redundant and synergistic information trivial provided we know their unique and mutual information.

### 3.2.3 Information Flow

Williams also introduces what he calls information flow, which he divides into intrinsic and extrinsic information flow. In intrinsic information flow, we only study how information flows through a system, without considering what this information is. Transfer entropy from Equation 2.19 is an example of this, as it only tells us how much $Y_t$ affects $X_t$.

In extrinsic information flow, the goal is to measure how much information is passed around inside the system about some value existing outside of the system. One measure he considers is the specific information between a process $X_t$ and some external stimulus $F$. He then expands the specific information into information gain and information loss, which are ways to measure how information about an external stimulus is either gained or lost at each timestep.

Let $X_t$ be a time-variant random variable and $F$ be some constant, external stimulus. The information gain $I_G(F; X_{t+1})$ [28] of the mutual information $I(F; X)$ is then defined as

$$I_G(F; X_{t+1}) = I(F; X_{t+1}) - I_{\min}(F; X_{t+1}, X_t)$$  \hspace{1cm} (3.7)

Similarly, the information loss $I_L(F; X_{t+1})$ is defined as

$$I_L(F; X_{t+1}) = I(F; X_t) - I_{\min}(F; X_{t+1}, X_t)$$  \hspace{1cm} (3.8)

Here $I_{\min}(F; X_{t+1}, X_t)$ represents the redundant information between the pairs $(X_t, F)$ and $(X_{t+1}, F)$. In other words, the information gain tells us how much information about $F X$ gains at timestep $t+1$ that it did not contain at timestep $t$. Likewise, information loss tells us how much information about $F$ in $X$ is lost at timestep $t+1$ from timestep $t$.

Building on the notion of redundant information from Section 3.2.2, Williams also expands Schneider’s transfer entropy from Equation 2.19 to what he calls information transfer from $Y$ to $X$ about $F$, which he defines as

$$T(F; Y \rightarrow X) = I_{\min}(F; X_{t+1}, \{X_t, Y_t\}) - I_{\min}(F; X_{t+1}, X_t)$$  \hspace{1cm} (3.9)
With Equation 3.9, instead of just measuring how much information is transferred between processes, it is possible to measure how much information about a specific stimulus is transferred between processes. Interestingly, it is also easy to prove that normal transfer entropy can be considered a special case of extrinsic transfer entropy if we set the stimulus $F = X_{t+1}$.

$$T(X_{t+1}; Y \rightarrow X) = I_{\min}(X_{t+1}; X_{t+1}, \{X_t, Y_t\}) - I_{\min}(X_{t+1}; X_{t+1}, X_t)$$

$$= I(X_{t+1}; X_t, Y_t) - I(X_{t+1}, X_t)$$

$$= I(X_{t+1}; Y_t | X_t) = T_{Y \rightarrow X} \quad (3.10)$$

### 3.2.4 Applications

In his PhD thesis, Williams applies his information dynamics framework to a continuous-time recurrent neural network, which is a model I will introduce in Section 4.1. In this model, an agent is able to move either left or right with motion sensors with a specified range pointing out of the agent. These motion sensors feed a signal into a layer of interneurons if an object is detected within this sensor’s range. Figure 3.3 shows an illustration of this agent.

Then, two balls falling consecutively against the agent is simulated. For the first ball, the agent does not do anything, but if the second ball is bigger than the first ball, it has been trained to move away from it. Since these two falling balls occur during two different phases of the simulation, this means the CTRNN’s interneurons have to remember the radius of the first ball and compare it to the radius of the second ball.

The most striking result from Williams’ experiment was the gradual buildup of specific information $I(F = f; N)$ about the radius of the first ball $F$ in one of the interneurons. Specifically, $I(F = f; N)$ would increase every time the ball intersected a new sensor. Likewise, measuring $T(F; S \rightarrow N)$ to this neuron from each of the sensors, showed spikes of transfer entropy occurring from each of these sensors when the ball intersected them.

At the second phase, he introduces a binary variable $R$ telling whether the size of the first ball $F$ is bigger than the size of the second ball $S$ or not, i.e. the relational information about the two radii. Then, by looking at the extrinsic transfer entropy between the agent’s neurons, we see a short spike occurring in $T(R; N_1 \rightarrow M)$ before $T(R; M \rightarrow X)$ which is a demonstration of how information "flows" from the environment, through the agent and back to the environment along the path $S \rightarrow N_1 \rightarrow M \rightarrow X$. 
Figure 3.3: Topology of the CTRNN used in Williams’ ball-avoiding agent. 
\{S_1, \ldots, S_7\} represent sensors returning 1 if an object intersects its line, 
\{N_0, N_1, N_2\} represents recurrent interneurons used to integrate information 
from previous timesteps, \{M_0, M_1\} represents motor neurons deciding 
acceleration in the left and right directions respectively.
CHAPTER 3. RELATED WORK

3.3 Miscellaneous

Other ways information theory has been used in neural networks include the class of training algorithms for unsupervised learning with feedforward neural networks known as the Infomax algorithms. In Infomax, the mutual information $I(X; Y)$ between the inputs $X$ and the output $Y$ is maximized [12].

Bullinaria [3] has looked at a problem where the traditional sum-of-squares error function used during training seems to stall when the output of a sigmoid function approaches either 0 or 1, where it starts flattening out and its derivative becomes 0. Let the cross entropy $H(p, q)$ between probability distributions $p$ and $q$ be

$$H(p, q) = H(p) + D(p||q)$$  \hspace{1cm} (3.11)

where $H(p)$ is the information entropy of $p$ from Equation 2.1 and $D(p||q)$ is the Kullback Leibler distance from Equation 2.10. Let $t$ be the distribution of true labels while $o$ is the distribution of output values. Bullinaria uses $H(t, o)$ as a cost function, and since $H(t)$ is a constant, this is the same as minimizing the Kullback Leibler distance from Section 2.4.
Chapter 4

Methodology

4.1 Continuous-Time Recurrent Neural Network

Let $y_i(t)$ and $I_i(t)$ represent a neuron’s time-variant activation level and input respectively. $\tau_i$, $w_{ij}$ and $\theta_j$ represent fixed parameters optimized on beforehand, usually with a genetic or evolutionary algorithm. Then, in a continuous-time recurrent neural network \cite{2} $y_i(t)$ is determined by the following ordinary differential equation

$$\tau_i \frac{dy_i}{dt} = -y_i + \sum_{j=1}^{n} w_{ij} \sigma(y_j + \theta_j) + I_i(t)$$  \hspace{1cm} (4.1)

At every timestep $t$, $y_i(t)$ is evaluated using a numerical ODE method such as the Euler method or Runge-Kutta method, as described in Section 4.2.

What makes CTRNNs so interesting is that each neuron has some activation level that will eventually go to 0 unless new stimuli to the neuron is applied. In this sense, we can think of them as ”leaking”. How fast a neuron is leaking is determined by its time constant $\tau_i$; the fact that neurons can have different $\tau_i$ also means that they can have different leaking rates, and this flexibility makes CTRNNs feasible for both short-term and long-term memory tasks.
CHAPTER 4. METHODOLOGY

4.2 Numerical Methods for Ordinary Differential Equations

As the calculation of the activation level of a CTRNN neuron involves solving an ordinary differential equation, I will spend this Section explaining the two methods used in this thesis.

4.2.1 The Euler Method

The Euler method [10] is arguably the simplest and most well-known method for numerically estimating an ODE. Given an initial value problem \( \dot{y} = f(x, y) \) and \( y(t_0) = y_0 \), we can estimate it with the following equation

\[
y_{n+1} = y_n + hf(t_n, y_n) \tag{4.2}
\]
\[
t_{n+1} = t_n + h \tag{4.3}
\]

where \( h \) is our step size.

4.2.2 The Runge-Kutta Method

While Euler’s method is appreciated for its simplicity and intuitivity, it is not the most numerically stable method. A more stable method would be the Runge-Kutta method [10]. Given an initial value problem \( \dot{y} = f(t, y) \) and \( y(t_0) = y_0 \) we can estimate it with the following equation

\[
y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{4.4}
\]
\[
t_{n+1} = t_n + h \tag{4.5}
\]

where

\[
k_1 = f(t_n, y_n) \tag{4.6}
\]
\[
k_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \tag{4.7}
\]
\[
k_3 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) \tag{4.8}
\]
\[
k_4 = f(t_n + h, y_n + hk_3) \tag{4.9}
\]
where \( h \) again is our step size.

Intuitively, we can think of the Runge-Kutta method as making four estimations with the Euler method, where each estimation starts from the previous, and then taking their average. The Runge-Kutta method gives us better estimates than the Euler method for bigger step sizes \( h \).

### 4.3 Transfer Functions

Both CTRNNs and feedforward neural networks use transfer functions to map inputs to a specific range, so I will spend this section discussing the ones used in this thesis.

We would like a function \( \sigma(x) \) that maps an arbitrary real number \( x \) into the range \([0, 1] \). A common alternative used for \( \sigma(x) \) is the logistic function

\[
\sigma(x) = \frac{1}{1 + e^{x}}
\]

However, since this function will be repeated over and over again, it might be a good idea to use a less computationally expensive one. Calculating an exponent is a computationally expensive task, so one way to speed up the evolution or training phase is to use another \( \sigma(x) \) such as

\[
\sigma(x) = \frac{0.5x}{1 + |x|} + 0.5
\]

Figure shows a comparison between these two. As we can see from the plot, Equation 4.10 can be considered a better threshold function as it’s curve is steeper and don’t need really high or really low input values to map to values close to 0 or 1.

### 4.4 Probability Density Estimation

In order to estimate differential entropies, we will also need a way to estimate continuous probability distributions from discrete data. Even though it is possible to avoid this step altogether and calculate their entropies directly [9], this was not attempted for this project.

The most straightforward way would be with a histogram: Consider that from \( n \) samples of the random variable \( X \), we would like to estimate its
probability density function. With histograms we divide the domain of $X$ into $b$ equally spaced bins, bin all of our $n$ samples into these bins, count the numbers in each bin and divide it by $n$. The time complexity for both adding and evaluating a point for this approach is $O(1)$, which makes it ideal for dealing with large datasets. On the other hand, as it needs to keep a bin for each entry of the histogram its memory complexity is $O(b^d)$ which means that for PDFs of high dimensions $d$ we will run out of memory. Another problem with this approach is that it is not continuous: similar yet still different $X$ may be bucketed into the same bin and considered to be the same. Another problem with these is that they are not smooth, especially in the case where our data set is undersampled.

Another popular method is one called Gaussian kernel density estimation (KDE) [17]. In Gaussian KDE each data point is given its own Gaussian kernel function with a fixed bandwidth $h$ and to determine the PDF at a point $x$ its deviation from the Gaussian kernel of each point in the distribution is calculated and averaged as shown in Figure 4.2. Unlike histograms this does not require us to store a big data structure in memory, which means memory complexity of this approach can be very small. Likewise, it will also give us a
Figure 4.2: Kernel density estimation generated from 2 points. Dashed lines show the contribution of each point, solid line shows their average better estimate than histograms if our dataset is undersampled. On the other hand, the naïve implementation of this gives us the time complexity $O(nd)$ to estimate one point, which makes it unfeasible for large datasets. There does indeed exist heuristics to speed up KDE extensively, by for instance storing data points in a tree structure [13]; however, this was not used for this project.

\[
\hat{f}(x) = \frac{1}{hn} \sum_{i=1}^{n} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2} \left( \frac{x-x_i}{h} \right)^2} 
\]  

Equation 4.12 shows the definition of Gaussian kernel density estimation, and you might have noticed the parameter $h$. $h$ is called the bandwidth and determines how smoothed out the distribution is supposed to be. The optimal bandwidth $h$ depends on the underlying distribution, and implementations often choose one for us automatically, although that does not mean it is ideal.

A good middle-way would be average shifted histograms [17]. In average shifted histograms with $b$ bins and $m$ shifts, a histogram with $b$ bins is first estimated and then each point is replaced with the average of its $m$ clos-
est bins. Readers already familiar with image and signal processing might recognize this as a smoothing or averaging filter. With a sufficiently high number $b$ of bins this gets rid of the undesirable unsmoothness property of histograms. However, it still is a histogram which means we still have to deal with its memory complexity and discrete nature.

Figure 4.3: Ordinary histogram vs. average shifted histogram
Chapter 5

Problem 1: Classifier

5.1 Problem Description

For this problem we would like to classify patterns in a binary 4x2 grid, inspired by Kashtan and Alon [8]. We introduce a function that tells us if there is a pattern in the 2x2 leftmost pixels, as well as a function that tells us if there is a pattern in the 2x2 rightmost pixels. We then introduce a classifier that classifies all combinations of the 4x2 grid into either of $C \in \{None, Left, Right, Both\}$. If there is no pattern in the leftmost or rightmost pixels, then the classification is $None$. If there is a pattern in the leftmost pixels but not in the rightmost pixels, then the classification is $Left$; likewise, if there is a pattern in the rightmost pixels but not in the leftmost pixels, then the classification is $Right$. At last, if there is a pattern in both the leftmost and the rightmost pixels, then the classification is $Both$. Figure 5.1 shows these classification rules as Boolean circuits, while Figure 5.2 shows us some example grids and their respective classification. Since AND and OR are linearly separable functions, it should be possible to train a feedforward neural network with 1 hidden layer with 2 neurons to work as a classifier.

5.2 Neural Network

The neural network used for this classifier was a feed-forward neural network. The topology used is a modular one as shown in Figure 5.3, where we have a $Left$ module that classifies the left-most pixels, a $Right$ module classifying the right-most pixels, and finally a neuron combining these two results for
Figure 5.1: *Left*, *Right* and *Both* classifiers as Boolean circuits

Figure 5.2: Various retina classifications
the actual classification. The learning algorithm used was backpropagation with gradient descent and a step size $\lambda = 0.1$. For every iteration, a random sample was drawn and used for training. Since our training set spans the entire space of inputs possible for this problem, this means overfitting will not be a problem and therefore no cross-validation or gradient regularization was used.

Neurons 1-8 represent our input neurons, and these neurons’ values are the values of pixels 1-8 as shown in Figure 5.2 respectively. Since all 8 inputs are binary, this means we have $2^8 = 256$ different combinations of inputs. We then introduce three different datasets we call Left, Right and Both, all of whom have 1 binary label. For the Left dataset, the label will be 1 if and only if the classification $C$ for that sample is Left; for the Right dataset, the label will be 1 if and only if the classification $C$ is Right; for the Both dataset, the label will be 1 if and only if the classification $C$ is Both.

To calculate the mutual information for a node and the classification $C$, all samples were run through the network and the probability density function was estimated from these samples’ classification $C$ and the neuron’s activation level $N$. Due to the low number of samples $n = 256$ used, kernel
density estimation could be used. Then, with the probability density functions estimated, the mutual information could be calculated using the mixed pair mutual information from Equation 2.26.

Let \( L \) be the activation level of neuron 19, \( R \) be the activation level of neuron 20 and \( O \) be the activation level of neuron 21. During training, at every 100th iteration, the mutual informations \( I(L; C) \), \( I(R; C) \) and \( I(O; C) \) were calculated and plotted together with the root-mean-square error. Then, when the error of the neural network had fallen below a threshold and assumed to be properly trained, the mutual information between every neuron in the network’s activation level \( N \) and the classification \( C \), \( I(N; C) \), was calculated to quantify how much that particular \( N \) contributed to the classification of the network.

The code for this problem was implemented in Python. To estimate the probability density functions the Gaussian KDE implementation gaussian_kde in SciPy 0.12 was used, which uses a fixed bandwidth \( h \). Afterwards, these probability density functions were integrated over with a numerical integration algorithm called QUADPACK.

### 5.3 Results

#### 5.3.1 Mutual Information During Training

As we can see from Figure 5.4, \( I(O; C) \) and \( I(L; C) \) start rising from the start while \( I(R; C) \) lays flat. Also notice the short decrease of mutual information happening at the same time as the error rate drops.

For Figure 5.5, we see the opposite of what we saw in Figure 5.4. \( I(O; C) \) and \( I(R; C) \) start rising from the start while \( I(L; C) \) lays flat. Here we also see the same decrease of mutual information happening at the same time as the error rate drops.

In Figure 5.6, we see that \( I(O; C) \) and \( I(R; C) \) start rising at the same time, then \( I(R; C) \) lies flat while both \( I(O; C) \) and \( I(L; C) \) continue rising. Yet again, the mutual information decreases for a short time while the error rate drops.
Figure 5.4: Mutual information: During training, *Left* dataset
Figure 5.5: Mutual information: During training, *Right* dataset
Figure 5.6: Mutual information: During training, *Both* dataset
5.3.2 Mutual Information After Training

For the Left dataset in Figure 5.7, mutual information is high in the neurons in the Left module and low in the neurons in the Right module.

For the Right dataset in Figure 5.8 we see the opposite. Mutual information is high in the Right module and low in the Left module.

As expected, in Figure 5.9, mutual information is high in both parts of the neural network.

For all of these datasets, we also see that in the first hidden layer there is 1 neuron with low mutual information in each module. This should not come as a surprise, as we can see from Figure 5.1 that no more than 2 neurons should be required for the first hidden layer, and therefore these neurons are superfluous.

5.4 Discussion

For this problem, two experiments were attempted: using mutual information to quantify how much each neuron contributed to the classification, and using
Figure 5.8: Mutual information: After training, *Right* dataset

Figure 5.9: Mutual information: After training, *Both* dataset
mutual information to quantify how the classification is advancing during training.

Looking at mutual information during training, we see in the Right dataset, both $I(O;C)$ and $I(R;C)$ start increasing from the start, while $I(L;C)$ lays flat. Likewise, for the Left dataset, both $I(O;C)$ and $I(L;C)$ start increasing from the start, while $I(R;C)$ lays flat. For the Both dataset, $I(R;C)$ and $I(O;C)$ start increasing before $I(L;C)$ does; $I(L;C)$ starts increasing at the same time as the error rate drops. This leads me to believe that mutual information during training can show how a neuron is progressing. However, we do not know which value they are supposed to converge to; for the Both dataset in Figure 5.6 we would expect $I(L;C)$ and $I(R;C)$ to converge to around 4 and 5 as they did in Figure 5.4 and Figure 5.5, but here they have been switched.

With respect to the short decline in mutual information happening at the same time as the error rate drops, we can look at how much each classification $C$ contributes to the mixed pair mutual information in Equation 2.26. Let $T(O,c)$ be the term $p(c)H(O|C = c)$ from Equation 2.26. By looking at their change during training, we see that $T(O,Both)$ rises and falls in the
timesteps 300-400, which is at the same time as the error rate is improving. The reason for this is, I believe, is that before this phase, all output values will be around some initial value, and after this phase, they will all be around 1. However, during this phase, we will have points at both places. In other words, we might have what is called a bimodal distribution, that is a distribution with two maxima, and Gaussian KDE with fixed bandwidth is known to oversmooth in these cases [15].

Looking at the mutual information after training, on the other hand, gave very clear results. As expected, when classifying the *Left* dataset, the neurons in the right module are just noise and are most likely suppressed at the output neuron anyway, and vice versa for the *Right* dataset. Also, notice how some neurons have low mutual information despite being in a module where most neurons have high mutual information. As previously mentioned, there are superfluous neurons in our hidden layers, and I would believe that the neurons with low mutual information are indeed irrelevant for our network.
Chapter 6

Problem 2: Controller

6.1 Problem Description

For this chapter we will look at another problem: an artificial neural network acting as a controller. A well-known and commonly used benchmark for this class of problems is the pole-balancing task. Imagine a cart that is able to move along an axis. At every timestep, the cart has to move either left or right with a constant force, it cannot stand still. Now imagine a pole balancing on this cart, with an initial position so that if left by itself gravity will make it fall to the ground. By moving the cart in the opposite direction of the way the pole is falling, we can pull it back up again, but then again the pole will fall in the other direction and the cart will have to move in the opposite direction again and so on ad infinitum. The goal of this problem is to balance the pole for as long as possible, preferably infinitely. And to make the problem a bit more interesting, the cart has to keep itself within a specified range. Let $x_t$ and $\theta_t$ be the cart’s position and pole’s angle as shown in Figure 6.1. Then, what we would like is a control policy such that $\theta_t$ and $x_t$, updated every timestep according to some physical laws, never exceed the ranges $[\theta_{\text{min}}, \theta_{\text{max}}]$ and $[x_{\text{min}}, x_{\text{max}}]$ respectively.

There are several variants of the pole balancing problem. The simplest one is the single-pole balancing problem with known velocities where we in addition to $\theta_t$ and $x_t$ are given their rates of change wrt. time, $\dot{\theta}_t = \frac{d\theta_t}{dt}$ and $\dot{x}_t = \frac{dx_t}{dt}$. Another variant is the double-pole balancing problem, where we introduce an additional pole that has to be balanced as well. Other variants of this kind are the hinged pole problem, where the second pole is hinged to
the first, as well as the single-input pole balancing problem, where for each
timestep we are only given the current angle $\theta_t$.

It can be proven [27] that for a single-pole balancing problem with known
velocities, the optimal control policy would be

$$F = F_{\text{max}} \text{sign}(k_1 x + k_2 \dot{x} + k_3 \theta + k_4 \dot{\theta})$$ (6.1)

where $k_1$, $k_2$, $k_3$ and $k_4$ are coefficients that have to be optimized. In other
words, it is linearly separable which means it can be solved by a single per-
ceptron. However, if we take away the velocities, the problem gets more
interesting. Just using the current position $x_t$ and angle $\theta_t$ themselves is not
enough, we will also need information about previous states as well. For that
we will need a neural network that is able to remember information from pre-
vious timesteps, which is why we will use a continuous-time recurrent neural
network.

The equations used for this problem were taken from Wieland [27]. For
a pole-balancing problem with an arbitrary number of poles, the cart accel-
eration $\ddot{x}$ and pole acceleration $\ddot{\theta}$ are calculated from the following equations

$$\ddot{x} = \frac{F - \mu_c \text{sign}(\dot{x}) + \sum_{i=1}^{N} \tilde{F}_i}{M + \sum_{i=1}^{N} \tilde{m}_i}$$ (6.2)

$$\ddot{\theta} = -\frac{3}{4l_i} (\ddot{x} \cos \theta_i + g \sin \theta_i + \frac{\mu_p \dot{\theta}_i}{m_i l_i})$$ (6.3)
Parameter | Value
---|---
Force ($F$) | 10
Gravitational constant ($g$) | -9.8
Pole length ($l_1$) | 0.5
Pole mass ($m_1$) | 0.1
Pole friction ($\mu_p$) | 0.0
Cart mass ($M$) | 1.0
Cart friction ($\mu_c$) | 0.0
Step size ($\tau$) | 0.01

Figure 6.2: Parameters: Pole-balancing problem

$\tilde{F}_i$ is the effective force from the $i$th pole of the cart:

$$\tilde{F}_i = m_i l_i \dot{\theta}_i^2 \sin \theta_i + \frac{3}{4} m_i \cos \theta_i (\frac{\mu_p \dot{\theta}_i}{m_i l_i} + g \sin \theta_i) \quad (6.4)$$

and $\tilde{m}_i$ is the effective mass of the $i$th pole:

$$\tilde{m}_i = m_i (1 - \frac{3}{4} \cos^2 \theta_i) \quad (6.5)$$

At every timestep $t$, $x_{t+1}$ and $\dot{x}_{t+1}$, $\dot{\theta}_{t+1}$ and $\ddot{\theta}_{t+1}$ are updated with the Euler method using the step size $\tau$ from Figure 6.2. It is important to note that this $\tau$ does not have to be the same as the $\tau$ used to update the activation levels $y_i$ of the neurons in Equation 4.1.

As initial conditions, the parameters were drawn from a uniform distribution over the ranges $x \in [-2.4, 2.4]$, $\dot{x} \in [-1.0, 1.0]$, $\theta \in [-0.2, 0.2]$ and $\dot{\theta} \in [-1.5, 1.5]$. The boundary conditions used were $x_{\min} = -2.5$, $x_{\max} = 2.5$, $\theta_{\min} = -\frac{\pi}{15}$ and $\theta_{\max} = \frac{\pi}{15}$, where $\frac{\pi}{15} \approx 0.2094384$. The controller will be assumed to be successful if it has been able to keep $x_t$ and $\theta_t$ within their respective boundaries for $100000 = 10^5$ timesteps.

### 6.2 Neural Network

As mentioned, for this problem we will use a continuous-time recurrent neural network with the topology in Figure 6.3. $x_t$ and $\theta_t$ are fed as inputs to
neurons $S_0$ and $S_1$ respectively. The transfer function $\sigma(x)$ used was the one in Equation 4.11. The acceleration of the cart was determined by feeding the activation level for $M$ into $\sigma(x)$; if this value was $< 0.5$, the cart accelerates with the force $F$ to the left, otherwise it accelerates with the force $F$ to the right. For optimization, the weights $w_{ij}$, bias $\theta_i$ and time constants $\tau_j$ from Equation 4.1 were evolved with the evolutionary algorithm explained in the next paragraph.

Every chromosome represented an individual neural network, and its genes represented the parameters $w_{ij}$, $\theta_i$ and $\tau_j$ from Equation 4.1 as real numbers. At startup, the values for the genes were drawn from a uniform distribution over the ranges specified in Figure 6.4. After each generation, surviving chromosomes were drawn at random where their probability of being drawn was proportional to their fitness, also known as proportional selection [1]. Their fitness function was the average number of timesteps the controller had been able to keep the cart and pole within their boundaries for 10 attempted runs. For the mutation, all genes in the chromosome were offset by a random number drawn from the distribution $N(0, \sigma^2)$ [7], with different $\sigma$ for $w_{ij}$, $\theta_i$ and $\tau_j$ used as in Figure 6.4. No crossover function was used.

The controller using the Euler method had a success rate of $28.5\% \pm 0.8\%$, while the controller using the Runge-Kutta method had a success rate of $56.0\% \pm 1.2\%$. For my results I have chosen to use the Euler method, as in it we only take 1 step every iteration and therefore information only propagates 1 edge per timestep. In the Runge Kutta method, on the other hand, we take 4 steps in one iteration, and information might end up propagating 4 edges every iteration. While this might be more realistic if we were to look at these differential equations in a continuous system, this would make studying causal relationships between neurons difficult.

The CTRNN simulation, evolutionary algorithm and the entropy estimations were all implemented from scratch in C++. For mutation in the evolutionary algorithm described the Gaussian random number generator from the Boost library version 1.48 was used. Results from the CTRNN simulations were written to comma-separated value files, and from these files entropies were estimated. Since I had to deal with datasets too big to fit into memory, a special-purpose reader class for C++ that handled these results in manageable chunks had to be implemented. For the entropy estimations, due to the vast amount of data points, average shifted histograms with 100 bins and 3 shifts were used. The range of each dimension was determined by
Figure 6.3: Neural network topology: Pole balancing

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Min</th>
<th>Max</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_{ij}$</td>
<td>-1.0</td>
<td>1.0</td>
<td>0.015</td>
</tr>
<tr>
<td>$\theta_i$</td>
<td>-1.0</td>
<td>1.0</td>
<td>0.015</td>
</tr>
<tr>
<td>$\tau_j$</td>
<td>1.0</td>
<td>2.0</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Figure 6.4: Evolutionary algorithm parameters: Pole balancing
probing the entire dataset in advance, finding the minimum and maximum for each specific dimension and dividing it into bins of this range.

In the results section, in Section 6.3.2 we will first look at the time-delayed mutual information between each of the four interneurons - $N_0$, $N_1$, $N_2$ and $N_3$ - and the environment variables $x_{t-k}$, $\dot{x}_{t-k}$, $\ddot{x}_{t-k}$, $\theta_{t-k}$, $\dot{\theta}_{t-k}$ and $\ddot{\theta}_{t-k}$ over the offsets $k \in \{0, 1, 2, 3, 4\}$ to figure out which of these variables they are estimating. Then, we will look at the history of each interneuron and the variable with the highest TDMI for that particular chromosome and look at how it changes together with the fitness during evolution.

Afterwards, in Section 6.3.3 we will look at the unrolled transfer entropy between neurons. At last, I will plot the transfer entropies in the Pole module together to see how information flows between them, inspired by what we saw about William’s ball-avoiding agent in Section 3.2.4.

For the results in both Section 6.3.2 and Section 6.3.3, probability density functions were estimated from 10 successful runs of 100000 timesteps.

## 6.3 Results

In this section I will show various plots from this problem. Please be aware, however, that some of these time series have been scaled to amplify trends and do not necessarily convey their actual values.

### 6.3.1 Trajectories

We start the results section by looking at how the environment variables change over time during a successful simulation. Figure 6.5 shows $x$, $\dot{x}$ and $\ddot{x}$, while Figure 6.6 shows $\theta$, $\dot{\theta}$ and $\ddot{\theta}$.

Next, we would like to see how the activations of the various neurons change as well. As can be seen from Figure 6.7 it seems like the these neurons contain information about $x$ from previous timesteps. For Figure 6.8 it is harder to see what these neurons contain information about. However, looking at the maxima and minima for $N_2$ they seem to correspond to the ones of $\theta$, while the ones in $N_3$ seems to correspond to the ones in $\dot{\theta}$ in Figure 6.6.
Figure 6.5: Trajectories: Cart variables

Figure 6.6: Trajectories: Pole variables
CHAPTER 6. PROBLEM 2: CONTROLLER

Figure 6.7: Trajectories: Cart neurons

Figure 6.8: Trajectories: Pole neurons
6.3.2 Time-Delayed Mutual Information

As mentioned in Section 6.2, I used time-delayed mutual information to pinpoint which environment variable each interneuron was estimating. For each of the interneurons $N_0$, $N_1$, $N_2$ and $N_3$, their time-delayed mutual information against each of the environment variables $x_{t-k}$, $\dot{x}_{t-k}$, $\ddot{x}_{t-k}$, $\theta_{t-k}$, $\dot{\theta}_{t-k}$ and $\ddot{\theta}_{t-k}$ with time delays from the range $k \in \{0, 1, 2, 3, 4\}$ were calculated and ranked from lowest to highest. Figure 6.9 and Figure 6.10 show a table of the 5 environment variables giving the highest TDMI for each interneuron, along with their TDMI normalized against the other environment variables.

Looking at Figure 6.9, we can see that the 5 variables with highest TDMI for both $N_0$ and $N_1$ all are the position $x$ at some previous timestep. Both of them seem to estimate $x_t$, although for $N_1$ the TDMI is considerably higher than for $N_0$. This is actually interesting, since it means that the cart does not need information about previous positions in order to evade its boundaries. It may be that $x$ moves that slowly, that when it’s at its boundaries this by itself should be considered a signal to go back to the origin without having to take its velocity into account. It is also interesting to note that it should take the information at least 2 timesteps to flow from $x$ to $N_0$ or $N_1$, so this might mean that these neurons are able to successfully estimate the current
Fig. 6.10: Time-delayed mutual information: Pole neurons

Table: Time-Delayed Mutual Information

<table>
<thead>
<tr>
<th>X</th>
<th>TDMI($N_{2,t}$; $X$)</th>
<th>Normalized TDMI($N_{2,t}$; $X$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\dot{\theta}_{t-2}$</td>
<td>1.50014</td>
<td>1</td>
</tr>
<tr>
<td>$\dot{\theta}_{t-3}$</td>
<td>1.29093</td>
<td>0.80334</td>
</tr>
<tr>
<td>$\theta_{t-1}$</td>
<td>1.25466</td>
<td>0.76924</td>
</tr>
<tr>
<td>$\ddot{x}_{t-4}$</td>
<td>1.16557</td>
<td>0.685503</td>
</tr>
<tr>
<td>$\dot{\theta}_{t-4}$</td>
<td>1.16435</td>
<td>0.684355</td>
</tr>
</tbody>
</table>

(a) Time-delayed mutual information for $N_2$

<table>
<thead>
<tr>
<th>X</th>
<th>TDMI($N_{3,t}$; $X$)</th>
<th>Normalized TDMI($N_{3,t}$; $X$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ddot{x}_{t-1}$</td>
<td>1.18957</td>
<td>1</td>
</tr>
<tr>
<td>$\theta_{t-1}$</td>
<td>1.09539</td>
<td>0.88918</td>
</tr>
<tr>
<td>$\ddot{x}_{t-2}$</td>
<td>1.06055</td>
<td>0.847818</td>
</tr>
<tr>
<td>$\ddot{x}_{t}$</td>
<td>1.04125</td>
<td>0.825054</td>
</tr>
<tr>
<td>$\theta_{t-2}$</td>
<td>1.03289</td>
<td>0.815188</td>
</tr>
</tbody>
</table>

(b) Time-delayed mutual information for $N_3$

Figure 6.10 shows us some interesting results as well. $N_2$ seems to contain information about $\dot{\theta}_{t-2}$ whereas the top scorer for $N_3$ is $\ddot{x}_{t-1}$. On the other hand, if we compare its normalized TDMI to the other variables, it does not really strike us as being much higher than the others. Looking at Figure 6.5 we see that $\ddot{x}_{t-1}$ has a trajectory that might be mistaken as a harmonic oscillation, so this does make sense. For the rest of this section, however, I will assume it contains information about $\theta_{t-1}$.

Likewise, we can also look at how the TDMIs change over time. In Figure 6.11 - 6.14 I have backtracked the ancestor chromosomes for the winner to see how the TDMI for the top scorers in Figure 6.9 and Figure 6.10 have changed over time during evolution together with the fitness. The fitness plotted is the logarithm of the fitness, used to amplify the fitness at the earliest stages during evolution. Both the fitness and the TDMIs have been normalized into the range $[0, 1]$. These plots are rather noisy, which is why I have decided to plot each of the TDMI series separately.

Among the most interesting results, notice how the rise of $TDMI(N_0; x_t)$ at around generation 200-400 and $TDMI(N_1; x_t)$ at around generation 1100 seem to coincide with a rise of the fitness. Also notice how sudden drops in $x_t$ based on previous information.
Figure 6.11: $TDMI(N_{0,t}, x_t)$ and fitness during evolution

Figure 6.12: $TDMI(N_{1,t}, x_t)$ and fitness during evolution
Figure 6.13: $TDMI(N_{2,t}, \dot{\theta}_{t-2})$ and fitness during evolution

Figure 6.14: $TDMI(N_{3,t}, \theta_{t-1})$ and fitness during evolution
fitness fall together with drops of TDMI, especially in Figure 6.13 and Figure 6.14. It seems like we with this plot are able to pinpoint which neuron failed at that particular generation.

6.3.3 Transfer Entropy

We proceed by studying how neurons affect each other at every timestep during a successful simulation. As mentioned in Section 2.7, the transfer entropy \( T_{Y \to X} \) tells us how much \( Y \) affects \( X \). Furthermore, by unrolling this measure as mentioned in Section 3.2.1, we can look at how much \( Y_t \) affects \( X_t \) at each particular timestep. In this section I will use this to look at how the various neurons’ activation levels affect each other during simulation.

In order to calculate \( T_{Y \to X} \), it was decomposed into entropies as given by Equation 6.6

\[
T_{Y \to X} = I(X_t; Y_{t-1}|X_{t-1}) \\
= H(X_t|X_{t-1}) - H(X_t|Y_{t-1}, X_{t-1}) \\
= H(X_t, X_{t-1}) - H(X_{t-1}) - (H(X_t, Y_{t-1}, X_{t-1}) - H(Y_{t-1}, X_{t-1})) \\
= -p(x_t, x_{t-1}) \log_2 p(x_t, x_{t-1}) - p(y_{t-1}, x_{t-1}) \log_2 p(y_{t-1}, x_{t-1}) \\
+ p(x_{t-1}) \log_2 p(x_{t-1}) + p(x_t, y_{t-1}, x_{t-1}) \log_2 p(x_t, y_{t-1}, x_{t-1})
\]

(6.6)

Over all the simulations, the probability distributions in Equation 6.6 were calculated. Then, using these probability distributions, at every timestep \( t \), the transfer entropy in Equation 6.6 was calculated.

I would also like to make the reader aware that I in this section will simplify \( T_{Y \to X} \) as \( Y \to X \). The reader might also notice that the entropies in this section are negative; an explanation for this can be found in the discussion in Section 6.4.

Angle Sensor to Angle Module

We start out with the simplest case, measuring transfer entropy \( S_1 \to N_2 \) and \( S_1 \to N_3 \), the part of the neural network processing information about the angle of the pole. As we can see from Figure 6.15, we get small spikes of transfer entropy from \( S_1 \) to \( N_3 \) occurring after each maximum and minimum in \( \theta_t \), that is when the pole changes direction. We also get spikes of transfer
entropy from $S_1$ to $N_2$ when the change of $\theta_t$ is at its steepest, that is when $\dot{\theta}_t$ is at its maximum or minimum.

Also notice how transfer entropy $S_1 \rightarrow N_3$ is higher the higher the amplitude is. In Figure 6.16, the angle $\theta$ has been binned into 100 bins and the average transfer entropy $S_1 \rightarrow N_2$ and $S_1 \rightarrow N_3$ for these $\theta$ over all 10 simulations has been plotted. As we can see, $S_1 \rightarrow N_3$ is higher the closer $\theta$ is to its boundaries.

Likewise, we can also confirm that transfer entropy $S_1 \rightarrow N_2$ is higher when $\dot{\theta}$ is at its maximum or minimum. This also supports our hypothesis from Section 6.3.2 that $N_2$ estimates $\dot{\theta}$.
(a) Transfer entropy vs. angle

(b) Transfer entropy vs. angular velocity

Figure 6.16: Average transfer entropy: Pole sensor to pole interneurons
Position Sensor to Position Module

We can also look at the transfer entropy \( S_0 \rightarrow N_0 \) and \( S_0 \rightarrow N_1 \) from the module processing information about the position. Looking at Figure 6.17 we see high transfer entropy \( S_0 \rightarrow N_0 \) and \( S_0 \rightarrow N_1 \) in the cases where the position \( x \) is at its local maximum or minimum or near the origin.

As we did for the Pole neurons, we can bin the position \( x \) into 100 bins and measure the average transfer entropy \( S_0 \rightarrow N_0 \) and \( S_0 \rightarrow N_1 \) for each bin. Figure 6.18 shows us these results. The closer \( x \) is to its boundaries \( x_{\text{min}} \) and \( x_{\text{max}} \), the higher the transfer entropy is to both neurons. We can also see a small bulge of transfer entropy around \( x = 0 \), and from Figure 6.7 we can see that the cart indeed occasionally steers away from the origin.

We can also see some unexpected observations. For instance, \( S_0 \rightarrow N_0 \) is higher at \( x = -2 \) than at \( x = 2 \), while \( S_0 \rightarrow N_1 \) seems to be skewed to the right. My hypothesis here is that \( N_0 \) statistically might be more used to signal the cart is nearing \( x_{\text{min}} \) while \( N_1 \) is used to signal the cart is nearing \( x_{\text{max}} \), although no further attempts were made to investigate this issue.

Interneurons to Motorneurons

Next, let us look at the transfer entropy from the interneurons \( N_0, N_1, N_2 \) and \( N_3 \) to the motor neuron \( M \). In Figure 6.19 transfer entropy from \( N_3 \) to \( M \) is shown. Here we have the fact that we see higher transfer entropy occurring at the same time the cart starts accelerating in the opposite direction. As this may not be immediately noticeable from Figure 6.19, I have attached a normalized histogram in Figure 6.20 which shows overall transfer entropy from all interneurons \( N_0, N_1, N_2, N_3 \) to the motor neuron \( M \) and transfer entropy in the timestep at the same time the cart starts acceleration in the opposite direction; that is, for every \( t \) where \( F_t \neq F_{t+1} \). As we can see, transfer entropy in this case is clearly higher than the rest.

Notice how in general the transfer entropies from the pole interneurons are higher than the position neurons. I believe this comes from the fact that the pole has to be kept away from its boundaries more times than the cart does, which means the pole interneurons will take control at more occasions.

Information Flow

At last, we put the transfer entropy between several neurons together to see how information flows in between them. In Figure 6.21, we see that every
Figure 6.17: Transfer entropy: Cart sensor to cart interneurons
Figure 6.18: Average transfer entropy: Cart sensor to cart interneurons
Figure 6.19: Transfer entropy: Interneuron to motorneuron
ridge of information transfer from \( \theta \rightarrow S_1 \) is followed by one from \( S_1 \rightarrow N_3 \) and then from \( N_3 \rightarrow N_2 \), which indicates that we have a flow of information occurring along the path \( \theta \rightarrow S_1 \rightarrow N_3 \rightarrow N_2 \).

Notice, however, that we also see some ridges of information transfer from \( N_3 \rightarrow N_2 \) where there is no preceding ridge of information transfer from \( S_1 \rightarrow N_3 \). Examples of these are at timestep 18 and 40, and from Figure 6.15 we can see that we have high transfer \( S_1 \rightarrow N_2 \) occurring at the same time.

6.4 Discussion

My first experiment here was using time-delayed mutual information to pinpoint what variable each of the neurons contain information about. For \( N_0 \) and \( N_1 \), it is clear that they include information about \( x \) at some timestep, where \( x_t \) got the highest TDMI for both. On the other hand, for \( N_2 \) and \( N_3 \) several of our top scorers were \( \ddot{x} \) while we expected them to keep information about \( \theta \) and \( \dot{\theta} \). Looking at Figure 6.8, neither \( N_2 \) nor \( N_3 \) look like harmonic oscillations, which could be the reason why we did not see the same clear trend here. Another problem here is that \( x \) and \( \theta \) are part of the same system and therefore not completely independent.

Afterwards we looked at the change of TDMI during evolution for the winning chromosome. It is interesting to note here that our estimates during evolution are more noisy than our estimates during training in the classification problem in Section 5, but I believe this comes from an important difference between evolution and learning by gradient descent. In evolution, we do random mutations to our parameters, and this can give us sudden changes. On the other hand, in gradient descent, for every timestep we minimize the error, which gives us a smoother progression.

Another interesting difference from the classifier in Section 5 is how the TDMLs do not start out at 0. The reason for this, I believe, is that in evolution we start out with a set of random neural networks, pick the best ones and try improving them, whereas in learning by gradient descent we start out with one single random neural network that is improved. Therefore, it seems like our evolutionary algorithm picked a neural network where these neurons already were trained to a certain degree. Indeed, the neurons whose TDMLs do not change significantly during evolution are \( N_2 \) and \( N_3 \), and since balancing the pole is the first challenge the controller has to face, my guess
is that this is why this particular chromosome was picked to start with. On
the other hand, $N_0$’s TDMI sees a small drop from 0.4 to 0 at the beginning,
and then suddenly jumps to around 0.8 at around generation 200 where it
lays for the rest of the process. $N_1$ starts out somewhat trained with a TDMI
at around 0.5 and jumps to 0.8 at around generation 1100, which is also a
time when our fitness suddenly jumps.

For our second experiment we looked at transfer entropy. We could see
ridges of transfer entropy from the sensors to the interneurons happening
at the same time as the cart or pole were at their boundaries. Likewise, we
could see ridges of transfer entropy from the interneurons to the motorneuron
occurring just before the cart changes direction, which we could interpret as
the interneurons forcing the cart to turn. Plotting these ridges together, we
also looked at the information flow between neurons. For the information
flow plot, we also saw some ridges occurring because of another variable
than those we were considering. In general, this is a weakness with transfer
entropy; the change in the target variable might be caused by a factor we’re
not considering.

I would also like to comment on why the transfer entropies are nega-
tive, which I believe comes from the curse of dimensionality. Since $p(y) = \sum_x p(x, y)$, that means $p(x, y) \leq p(y)$. In other words, probabilities of in-
creasing dimensions are smaller and therefore give higher entropy. This
means the dominating term of Equation 6.6 will be $-H(X_t, Y_{t-1}, X_{t-1})$, which will be guaranteed to be negative since we are using histograms and
therefore discrete entropies. An attempt was made using the definition of
transfer entropy directly from Equation 2.19, but resulted in underflow er-
rors. Due to the unintuitive upper bound of quantized entropies mentioned
in Section 2.8, I did not deem these values interesting enough to investigate
this issue further.
Figure 6.20: Histogram of transfer entropy: Interneuron to motorneuron at acceleration shift. Blue represents overall transfer entropy, red represents transfer entropy at acceleration shift.
Figure 6.21: Information flow $\theta \rightarrow S_1 \rightarrow N_3 \rightarrow N_2$
Chapter 7

Summary

7.1 Conclusion

Looking back at my first research question posed in the Introduction, *how information theory can be used to help a user understand the dynamics of an artificial neural network*, I believe I have shown various ways during this thesis, although a user should be aware of the potential pitfalls.

For a feedforward neural network acting as a classifier, measuring the mutual information between the neurons’ activation levels and the classification let us quantify how important that particular neuron was to the classification. Likewise, tracking this mutual information during training gave us an indicator of progress for that neuron. I believe it might be worth comparing this to the work of Bullinaria [3], as his method of using cross entropy requires us to compare the neuron’s activation level to a true value. This might work for the output neuron, but not for the hidden neurons as they do not have a true value. Unfortunately, this is for the moment not a perfect metric; possible improvements have been suggested in Section 7.2.

For a dynamic neural network acting as a controller, using time-delayed mutual information let us see what environment variable each interneuron was estimating, as well as pinpointing which timestep it started taking this role by looking at the same TDMI for its ancestor chromosomes. Looking back at its history, we could also see why the controller failed at each generation by looking at which interneuron had a sudden drop of TDMI happening at the same time as the fitness. Unfortunately, this method requires us to know in advance what each interneuron is supposed to estimate, which means
it cannot be run online while the network is optimizing, as was the case for the classification problem.

With these two problems, even though they were solved with two very different neural network models, we also got to study the difference between learning by gradient descent and evolution. Specifically, in evolution our neurons were already somewhat trained at startup, unlike in gradient descent where they were not at all. Likewise, the progression of the neurons during evolution was inherently noisy, which comes from the nature of mutations in evolution.

In a dynamic neural network, using transfer entropy we can study how neurons react to external events and pass information around to each other. Even though we have only studied it on a controller problem in this thesis, I would believe it might be useful for other classes of problems as well, as it was in Williams’ agent [28] described in Section 3.2.4. A drawback with this method, however, is that our results might be ambiguous unless we know the causal relationships on beforehand. Changes in variables might as well happen due to changes in variables other than those we are considering. Using multivariate transfer entropy, or conditioning it on other neurons might be a remedy for this problem, but for the moment this is still being actively researched.

When it comes to my second research question, how information theory could be used to improve neural networks, I believe my most promising results were found with the static information theory measures. With mutual information we can tell what information is stored in each of the neurons, and tracking it during optimization can tell us how they are progressing. This, I believe, is very useful, especially for classifiers, as it can take several epochs before we see any significant drop in the error rate. With mutual information the user gets immediate feedback whether his neural network is improving or not.

For the classifier, the neurons with low mutual information I believe are superfluous neurons, and this might tell the user of a neural network if neurons can be removed from the network while still keeping its power. This could be useful for simplifying neural network topologies. Simpler topologies are easier to reason about and consume less computational power.

For a neural network optimized with evolution, looking at the mutual information between each neuron and the variable it is supposed to be estimating during evolution might help the user gain some insight into how his neural network is changing generation by generation. This could be help-
ful if he for instance would like to find the best parameters to use for his evolutionary algorithm.

As a result of this thesis, I hope to have contributed to the early steps required to make a powerful class of optimization methods accessible to a wider audience.

7.2 Future Work

For the classification problem I believe it could be interesting to further study how information theory can be used to measure progress during training. For the sudden drop in mutual information at the same time as the error dropped, I believe using another entropy estimation method than Gaussian KDE with a fixed bandwidth, for instance a method based on k-nearest neighbors [9] or Gaussian KDE with an adaptive bandwidth, could be worth looking at. For the unintuitive upper bound, the first measures I would look at would be one of the many forms of specific information and somehow normalizing them.

Another interesting topic I think would be worth looking at would be extrinsic information flow. Using Williams’ extrinsic transfer entropy to calculate the flow of information about a specific variable might give interesting results for the controller problem. With the modular topology I used in this thesis, it is obvious that each of the two modules only process information from their respective inputs. However, for a fully connected interlayer, looking at the transfer entropy about the angle $\theta$ or $x$ between the interneurons, we might be able to see how information about these specific variables flow around.
Bibliography


