RADBOUD UNIVERSITY

BACHELOR THESIS

# Automatic Text Summarization As A Text Extraction Strategy For Effective Automated Highlighting

*Author:*
Wesley VAN HOORN
s4018044

*Supervisors:*
Dr. Franc A. GROOTJEN &
Dr. George E. KACHERGIS

*A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science*

*in*

Artificial Intelligence
Department of Artificial Intelligence

February 25, 2018

Radboud University

RADBOUD UNIVERSITY

# *Abstract*

Social Sciences
Department of Artificial Intelligence

Bachelor of Science

**Automatic Text Summarization As A Text Extraction Strategy For Effective Automated Highlighting**

by Wesley VAN HOORN

Automatic text highlighting is capable of becoming a new tool in textual information processing. Preliminary research is done to examine the potential of a new application for text summarization algorithms. A framework is built for parsing and highlighting PDF files, to which extraction strategies can be applied. Also a small dataset of highlighted documents is gathered to test the highlighting capabilities of four text summarization algorithms on. Thereby, ROUGE-N scores are obtained, the performance per task and the performance per algorithm are evaluated. In the end, the algorithms perform surprisingly well and may encourage more investment in automatic text highlighting.

# *Acknowledgements*

I would like to thank Franc Grootjen and George Kachergis for their insights into this thesis. I am also grateful for the loving support of family and friends, in particular Loes van Druenen.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **IDF** | Inverse Document Frequency |
| **LRK** | LexRank |
| **LSA** | Latent Semantic Analysis |
| **NLP** | Natural Language Processing |
| **SVD** | Singular Vector Decomposition |
| **TRK** | TextRank |
| **TTS** | TextTeaser |

*Dedicated to my father, may he rest in peace*

# Chapter 1

# Introduction

People need to read increasingly more text in seemingly less time than ever before (SINTEF, 2013; Economist, 2010). For literature as well as electronic literature reading strategies exist to increase the number of words per minute read or to skim through an entire text (Sillburn, 2017). Although strategies can decrease the time a reader spent per document, it requires a lot of dedication from the readers. Computers, on the other hand excel at data processing. This is the age of Big Data, enormous datasets can be computationally analyzed to reveal patterns, threads and associations (Peters, 2012). A multidisciplinary field consisting of computer science, artificial intelligence and computational linguistics is dedicated to Natural Language Processing i.e. the interactions between human (natural) language and computers. Computers are capable of aiding people in reducing text to its essence. Text summarization and text highlighting are selective and purposeful techniques people use to comprehend text (Jones, 2012; Yue et al., 2015). While text summarization is highly adopted in NLP research (Das and Martins, 2007; Saggion and Poibeau, 2013), this research will focus on the latter to broaden the spectrum. Both techniques will discussed in detail in chapter 2.

## 1.1 Research question

Automated text highlighting may be a helpful application for people, such as academics, politicians or managers, who need to read and review many texts. While automatic text summarization is currently available, there is no proper implementation for text highlighting yet. Aspects of automatic text summarization can be shared and implemented in a text highlighting application. This research is an attempt to find an answer to how to implement automatic text summarization as a text extraction strategy for effective automated text highlighting.

## 1.2 Structure

The research consists of three stages. First of all, constructing a framework capable of extracting and highlighting texts in PDF. A robust framework, with the means to accept scanned documents is outside the scope of this thesis considering that the focus lies on the second and third stage. The framework is constructed in Java, utilizing the iText library[1]. It is designed in a way a PDF document as input will result in a copy of the PDF document as output, of which the text will be highlighted according to the chosen text extraction strategy.

[1] iText Library - https://itextpdf.com

Secondly, the implementation of several text summarization algorithms as different text extraction strategies for effective highlighting to fit the constructed framework. A pilot will give insight into human text extraction strategies. In this pilot a group of young adults is asked to highlight a specified set of documents conditioned that the subjects have to highlight the documents as if they had to study for it. The participants are from different disciplines, therefore slight variation in strategy is expected. The pilot is held to generate reference material to evaluate the strategies with. This stage is an attempt to find and closely match an averaged human text extraction strategy. The implementation of the text extraction strategy will be distinguished by the following four text summarization algorithms. Two algorithm implementations from the Sumy library [2] named Latent Semantic Analysis and LexRank. An implementation of the TextRank algorithm (Mihalcea and Tarau, 2004) from the Gensim library [3]. And a Python implementation of TextTeaser (Jagadeesh, Pingali, and Varma, 2005), PyTeaser [4] (Gunawan et al., 2017). The mentioned algorithms are described in more detail in chapter 2.

Finally, an experiment is conducted in an attempt to validate the implementation. Comparisons are made between human-highlighted documents and application-highlighted documents captured in a ROUGE-N score (Lin, 2004), facilitating every implemented text summarization algorithm. The performance of the algorithms on text highlighting will also be compared against the performance of the same algorithms on text summarization. The data on text summarization acquired from (Mathur, Gill, and Yadav, 2017). On the bold assumption that highlighted documents from untrained people are similar to summaries of the document and that the algorithms remain unchanged, it is hypothesized that the application and its strategies perform on text highlighting similarly to the algorithms tested in (Mathur, Gill, and Yadav, 2017). Also, it is hypothesized that LexRank performs better than the other algorithms because LexRank performed best in (Mathur, Gill, and Yadav, 2017). The hypotheses are described in more detail in chapter 3.

[2] Sumy Library - https://github.com/miso-belica/sumy
[3] Gensim Library - https://radimrehurek.com/gensim/
[4] PyTeaser - https://github.com/xiaoxu193/PyTeaser

# Chapter 2

# Background

This chapter contains a more in depth perspective on the similarities and differences of text highlighting and text summarization. The application relies on existing algorithms therefore a detailed description of the implemented algorithms is given here.

## 2.1 Text Highlighting

On the assumption made in chapter 1, this thesis discloses a definition of text highlighting that deviates from text synthesizing. Text highlighting is the typographic application of post-print visual distinction of characters to emphasize purposeful segments of a text. It is believed that emphasizing text will help to remember information better or make reviewing more effective. The actual benefit of text highlighting is debatable as numerous studies have mixed results about its effectiveness (Yue et al., 2015). E.g. students do often not know how to highlight effectively (Bell and Limber, 2009; Stordahl and Christensen, 1956) and forcing readers to use text highlighting may be even counter-productive (Howe and Singer, 1975). However, when students are trained in highlighting techniques, they perform better than students who are not (Leutner, Leopold, and Elzen-Rump, 2007). Moreover, when students are presented pre-highlighted texts, they recall the highlighted passages better than the non-highlighted passages compared to students who were presented with the unmarked text (Fowler and Barker, 1974; Silvers and Kreiner, 1997). This concludes that text highlighting is a skill that must be trained and when applied correctly, is able to be a helpful tool for text reduction.

To emphasize important segments authors can choose to visually distinct characters to alter the font color, size, style or weight of the selected segment. Post-print, readers are unable to modify the characters itself, only its field by e.g. adding a background color, an underlining or adding a rectangular border to parts of interest. A combination of visually distinct characters and its field is preferred according to (Strobelt et al., 2016). For the reason that this thesis conducts documents post-print, it will act from a perspective of readers and due to time constraints only focus on the application of background color.

## 2.2 Automatic Text Summarization

A summary is a "text that is produced from one or more texts, that conveys important information in the original text(s), and that is no longer than half of the original text(s) and usually significantly less than that" according to (Radev, Hovy, and McKeown, 2002). The focus lies on what is most important while it omits lesser details and examples. A summary is a decomposed and rebuilt version of its source only

consisting of the essence of the text (Das and Martins, 2007). Because of the decomposing nature, summaries of low quality suffer from information loss. Contrary to text summarization, text highlighting keeps the source intact and is build on top of it. Therefore, in case of imperfect highlighting, its reference remains and can still be reviewed.

The two main approaches for automatic text summarization are extractive and abstractive (Hahn and Mani, 2000). Extractive approaches generate summaries by the concatenation of extracts from the text. Although they are limited by passages or sentences of the source, this property allows extractive approaches to be widely applicable in contrast to abstractive approaches. Abstractive approaches uses linguistic methods to decompose and interpret the text so it is able to generate a summary based on the formal representation of the content. They are capable of paraphrasing. A drawback is that abstractive approaches, i.e. knowledge-based methods, need to be trained with text corpa. This makes abstractive approaches domain specific. While the concatenation of extracts can be considered a limitation for automatic text summarization, it is not for text highlighting since annotations are added to the source as is. That is why this thesis exclusively focuses on extractive approaches.

The concept of automated text summarization dates back to the fifties when Luhn proposed that a convenient measure of word significance is in fact its frequency in a document (Luhn, 1958). Through the seventies and eighties linguistic, statistical or combined approaches began to be explored (Goldstein et al., 1999). Most literature represents word for word extraction of sentences for single document summarization. This group can be categorized as heuristic approaches, where words or sentences are ranked conform features (Edmundson, 1969). In the more recent years attention has been shifted towards abstractive approaches and multi-document summarization (Das and Martins, 2007; Gupta and Lehal, 2010). I.e. machine learning methods like Naive-Bayes, Hidden Markov models, Neural Networks and on the other hand deep language analysis, where discourse structure is taken into account with e.g. lexical chains and rhetorical structure theory (Barzilay and Elhadad, 1999). While multi-document summarization contributes to the reduction of textual data, it lies outside the scope of this thesis. The application will be employed per document.

## 2.3 LexRank and TextRank

LexRank (Erkan and Radev, 2004) and TextRank (Mihalcea and Tarau, 2004) are extractive summarization algorithms, which apply unsupervised graph-based ranking to build a summary. Essentially, they decide the importance of a sentence within a text. It is derived from Google's PageRank (Page et al., 1999) and takes also edge weights into account. LexRank is optimised to handle multiple documents. This thesis utilizes LexRank implemted in the Sumy Library and an adaptation of TextRank (Barrios et al., 2016) implemented in the Gensim library.

The algorithms are executed in three stages. First, it pre-processes the text by removing stop words and stem the remaining. Stop words refer to the most common words e.g. the articles 'a' and 'the' in the English language. Secondly, it creates a complete graph of which the vertices are the sentences of the text and the edges are weighted by the similarity between the sentences. The similarity is determined by the percentage of word overlap between sentences; see definition 1. The inverse of that, *inverse document frequency* is used as a measure to assess the importance of the words (Sparck Jones, 1972); see definition 5. This is where LexRank and TextRank

divert. LexRank uses IDF-modified Cosine as its similarity measure and TextRank uses a simply modified version of IDF (Sparck Jones, 1972); see definitions 2 and 4 respectively. This is the sole difference between the two, would it not be that the Gensim implementation of TextRank (Barrios et al., 2016) uses Okapi BM25 as its ranking function as described in definition 3. Okapi BM25 will result a negative value if a word occurs in more than half the document. To prevent irregularities Barrios made a simple modification to the IDF (Barrios et al., 2016); see definition 4. Finally, the ranking function is executed on the graph and the highest scoring sentences are selected.

The equations in the following definitions are derived from (Erkan and Radev, 2004; Barrios et al., 2016; Sparck Jones, 1972) respectively. The equations have been modified to create consistency in variable names throughout the thesis.

**Definition 1** *Let $G = (V, E)$ be a directed graph. For a given vertex $V_i$, let $adj(V_i)$ be the set of predecessors and successors of $V_i$. Given three vertices (senteces) R, S and T, the LexRank score (Erkan and Radev, 2004) is defined as:*

$$L(R) = (1-d) + \frac{d}{N} \sum_{S \in adj(R)} \frac{IDF_{cos}(R,S)}{\sum_{T \in adj(S)} IDF_{cos}(T,S)} L(S) \tag{2.1}$$

*where d is a damping factor between 0 and 1 but usually set to 0.85 (Page et al., 1999). And N is the total number of documents.*

**Definition 2** *Given two sentences R and S, the IDF-modified Cosine (Erkan and Radev, 2004) is defined as:*

$$IDF_{cos}(R,S) = \frac{\sum_{w \in R,S} f(w,R) \cdot f(w,S) \cdot IDF(w)^2}{\sqrt{\sum_{w_r \in R} f(w_r,R) \cdot IDF(w_r)^2} \cdot \sqrt{\sum_{w_s \in S} f(w_s,S) \cdot IDF(w_s)^2}} \tag{2.2}$$

*where w is a word, $w_r$ and $w_s$ represent a word in R and S respectively. And $f(x,y)$ is the frequency of x in y.*

**Definition 3** *Given two sentences R and S, Okapi BM25 (Barrios et al., 2016) defined as:*

$$BM25(R,S) = \sum_{i=1}^{N} IDF_{Barrios}(w_s) \frac{f(w_s,R)(k+1)}{f(w_s,R) + k \cdot (1 - b + b\frac{|R|}{avgDL})} \tag{2.3}$$

*where N the total number of words in the document. k and b are parameters, $k = 1.2$, $b = 0.75$ (Barrios et al., 2016) and $avgDL$ is the average sentence length.*

**Definition 4** *Given word w, IDF (Barrios et al., 2016) is defined as:*

$$IDF_{Barrios}(w) = \begin{cases} log(N - f(w,n) + 0.5) - log(f(w,n) + 0.5) & if\ f(w,n) > N/2 \\ \varepsilon \cdot avgIDF & if\ f(w,n) \leq N/2 \end{cases} \tag{2.4}$$

*where n is the document, ε is a value between 0.3 and 0.5 and $avgIDF$ is the average IDF for all words.*

**Definition 5** *Given word w, IDF (Sparck Jones, 1972) is defined as:*

$$IDF(w) = log(\frac{N}{f(w,n)}) \tag{2.5}$$

## 2.4 Latent Semantic Analysis

LSA (Gong and Liu, 2001) is a technique for a vector-based representation of text. It applies *singular value decomposition* to a text to find similarities in pairs of text (Wiemer-Hastings, 2004; Steinberger and Jezek, 2004); see definition 6. These text pairs are represented as sentences of a text but could also represent whole documents.

For a total of $m$ words and $n$ sentences in a text an $m \times n$ matrix $M$ is created. Since not every word is in every sentence, matrix $M$ is sparse. $M_i$ represents a weighted word-frequency vector of sentence $i$.

**Definition 6** *Given an $m \times n$ matrix $M$, the SVD of $M$ is defined as:*

$$M = U\Sigma V^T \tag{2.6}$$

*where $U$ is an $m \times n$ column-orthonormal matrix, of which the column are the left singular vectors. $\Sigma$ is an $n \times n$ diagonal matrix, of which the diagonal elements are sorted in descending order. And $V$ is an $n \times n$ orthonormal matrix, of which the column are the right singular vectors.*

Matrix $V^T$ describes for each topic of each sentence its importance. So to build a summary the most important sentences are chosen. The approach of LSA from (Steinberger and Jezek, 2004), mentioned above, is implemented in the Sumy Library.

## 2.5 TextTeaser

TextTeaser is a feature-based extractive summarization algorithm. It links a score to every sentence of a text, the score is based on extracted features from each sentence. The highest scoring sentences are used to build the summary. The algorithm is capable to extract sentence level features as well as word level features (Jagadeesh, Pingali, and Varma, 2005). However, the implemented TextTeaser, PyTeaser (Gunawan et al., 2017) only utilizes the following sentence level features labeled: *titleFeature, sentenceLength, sentencePosition* and *keywordFrequency*.

The *titleFeature* is the number of words a sentence has in common with the title of the text. Gunawan implemented *sentenceLength* as the normalized distance from a set constant that represents the ideal length of a sentence (Gunawan et al., 2017), which in PyTeaser is set to 20 whereas, Jagadeesh describes *sentenceLength* as a way to normalize word level feature scores over the length of a sentence (Jagadeesh, Pingali, and Varma, 2005). The *sentencePosition* is a normalization score over the position of a sentence in the text. Sentences that occur early or late in a text are considered more important. The presence of infrequent keywords in a sentence determines the *keywordFrequency*. In PyTeaser, it is based on the occurrence of ten keywords with the lowest frequency.

# Chapter 3

# Methods

The construction of the framework is the first stage. It represents a proof of concept and provides a starting point for the text extraction strategies. The second stage is to implement the text summarization algorithms and to find a measure of validation. A pilot was held with human subjects to identify a point of reference. The validation of the application is the final stage. An experiment is held to determine the performance of the strategies. Texts highlighted by the strategies are compared against the same texts highlighted by human subjects.

## 3.1 Framework

The process of constructing the framework gave insight in the possibilities of modifying post-print PDF documents. One of the goals was to have a practical application therefore it was desired to work with raw unedited PDF files to simulate a real-life application. Due to the availability of multiple supported PDF libraries, it became evident to develop this application in either Java or Python. Hereafter, through personal preference of Java and familiarity with the iText library, the framework of the application is constructed in Java. The framework is capable of parsing an input PDF file and output an altered copy of the PDF file, of which the text is highlighted according to the text extraction strategy.

In the Public Document Format, there is no concept of lines of text, every character is appended at absolute positions. With the use of iText, it is possible to parse characters at specified positions i.e. two dimensional coordinates, every character is queried on the x-axis for each step on the y-axis. The step size is determined by a threshold, which is usually the font size. The default text extraction strategy parses chunks of text. These chunks are random in size and therefore do not take the concept of words into account. For text highlighting it is important that, rather than chunks, words can be highlighted individually.

The framework of the application uses an adaptation on the default text extraction strategy, which is capable of parsing words instead of chunks. Characters and their characteristics are pulled one by one from the document. When a distance is measured of more than a quarter the width of a space character between characters, a word boundary is met. Only then a word is build from the characters and stored for further use. For each word an enclosing rectangle of coordinates is kept. This rectangle represents the space in the PDF that can potentially be highlighted. For highlighting multiple adjacent words or sentences, these rectangles will be linked into one larger rectangle.

Applicable input documents are restricted as a result of the inability of parsing poorly scanned documents i.e. digitalized books by using an image scanner with

poor alignment. Also PDF documents of which parts of the file structure are corrupted or missing, although they can sometimes still be opened, they can not be parsed.

Afterwards, a simple straight forward GUI is built for user friendly testing. It consist of a small window with an option to supply a PDF document to input, a list to choose an extraction strategy from, a field to set the ratio of the strategy and a button which, once pressed, applies the strategy and saves the result.

## 3.2 Implementation

After an extensive search it became clear that the supply of different implemented text summarization algorithms is greater in Python than in other languages. So it is decided the, in Java constructed, framework has to work with Python implementations. The framework has been adjusted, it writes the parsed PDF file to a txt file so a Python script is able to work with that file. And after the script is executed it has generated another txt file containing the extraction strategy for the framework to use. Once the libraries were installed, the implementation was straight forward. Only the PyTeaser algorithm needed manual adjusting. Originally, the results of the algorithm did not respect sentence's order of occurrence but returned a list of of sentences ordered by importance. After adjusting, it keeps track of the order of occurrence and sorts the resulted list accordingly. This does not alter the actions PyTeaser takes, the order is tracked beforehand and restored afterwards.

For each text summarization algorithm a small script is written. The script reads the file written by the framework, applies the algorithm and writes the results to a file so that the framework is able to highlight the PDF according to that extraction strategy. The algorithm is applied with a compression ratio set by the framework as an argument.

## 3.3 Measure of validation

Engelert evaluated students' performance, in their ability to highlight text, by reflecting on four distinguished traits (Englert et al., 2009). Their ability to represent the hierarchical arrangement of the major and minor ideas. The extent of content coverage, typified by breadth and depth. The selectivity of the students and finally the usefulness of the result. Engelert shows that quality is a subjective evaluation measure for text highlighting and that there exists no one correct way of highlighting. These traits could also be measures to evaluate automated text highlighting. However, without the presence of a linguistic expert, it would be difficult to value this method of validation.

To argue over the performance of the application, more statistical alternatives exist. To evaluate summarization algorithms there are datasets consisting of news articles and *golden summaries*. These are summaries written by experts or averaged over a large set of summaries. *Golden summaries* are utilized as reference to measure the performance of summarization algorithms in emulating these references. An agreed upon statistical analysis to measure similarity between samples is to determine the Sørensen-Dice coefficient also known as F-Score (Cha, 2007). Nevertheless, in the field of NLP a variation on the F-Score is more favored, namely the ROUGE-N score (Lin, 2004; Manning, Schütze, and others, 1999). More details on ROUGE-N will be in section 3.5.

TABLE 3.1: Content of dataset consisting of four documents

| DocumentID | Title | Ref |
|---|---|---|
| d0 | Predicting imdb movie ratings using social media | (Oghina et al., 2012) |
| d1 | Training a 3-node neural network is NP-complete | (Blum and Rivest, 1989) |
| d2 | Pigeons'discrimination of paintings by Monet and Picasso | (Watanabe, Sakamoto, and Wakita, 1995) |
| d3 | Measurements of Young's modulus, Poisson's ratio, and tensile strength of polysilicon | (Sharpe et al., 1997) |

A method to validate the utilization of automatic text summarization on automated text highlighting is to compare the performance of the algorithm on text summarization with the performance on text highlighting. Mathur compared the ROUGE-1 score of a collection of algorithms ,including the four implemented algorithms, on the Opinosis dataset (Mathur, Gill, and Yadav, 2017; Ganesan, Zhai, and Han, 2010). The dataset consist of 100-sentence-long user reviews over 51 topics with corresponding *golden summaries*. User reviews sounded as an odd document type for text highlighting, therefore it was decided to choose a different set of documents, preferably of texts of which comprehension of the text is more important than mere text reduction.

## 3.4 Pilot

A pilot is held to identify text extraction strategies humans employ. The identification of strategies will delimit the perspective of text extraction and these strategies will serve as a reference point for the implementation. Subjects are asked to highlight a specified set of documents conditioned that they have to highlight the documents as it is examination material and they have to study for it.

A small group of twelve people participated in the pilot. Their educational attainment varies from high school to post-master, with interest and background in different fields, e.g. Artificial Intelligence, Economics, Engineering, Design, IT, Language, Law and Media, aging between 22 and 27 years old. No constraints were enforced, participants were free to apply their own strategy and compression ratio. Therefore, slight variation in applied strategies is expected. It was difficult to find participants who would voluntarily read a set of documents within a time frame so the documents were chosen accordingly. Academic papers are chosen as document type due to the educational attainment of the participants and the availability of a variety of academic papers. The topics divert from accessible and concrete to unfamiliar and abstract and most importantly, the documents were short. They were given a week to highlight the set of documents presented in table 3.1. Lastly, the participants were asked to rate their challenge in processing the documents on a five-point Likert-scale. *1* representing *not challenging* and *5* representing *very challenging*.

## 3.5 Experiment

This thesis will conduct an experiment corresponding to the experiment from (Mathur, Gill, and Yadav, 2017). The application, with the four extraction strategies LexRank, LSA, TextRank and TextTeaser, is executed on the same four documents mentioned in table 3.1. To evaluate the extraction strategies the ROUGE-N score will be computed. To test the performance on text highlighting with a ROUGE-N score, the input files need to mimic summarization output. Therefore, the raw extraction strategy files, created in between the execution, are compared with reformatted reference texts from the pilot. The raw extraction strategy files will be referred to as strategy texts.

ROUGE-N is the n-gram recall between a target summary and a set of reference summaries. A n-gram is a $n$ sized consecutive sequence of items of a given sample (Lin, 2004). Mathur computed the ROUGE-1 score, i.e. unigram score, on relative small summaries (Mathur, Gill, and Yadav, 2017). For larger extractive summaries and strategies, one might consider a ROUGE-N score with larger n-grams. In this thesis a ROUGE-1 score is computed to compare against the data from (Mathur, Gill, and Yadav, 2017) and also a ROUGE-N score of n-grams equal to the smallest highlighted sequence from the pilot. Rouge 2.0 [1] is used to compute the ROUGE-N score. It is a Java implementation of ROUGE-N, which also computes the F-Score and the corresponding precision.

A recall score would not be reliable if the strategy text is significantly smaller or larger than its reference text. The reference texts are therefore categorized into three groups per document based on their compression ratio: $r < 0.15$, $0.15 \leq r \leq 0.25$ and $r > 0.25$. Subsequently, for each extraction strategy a strategy text is created per group, with a ratio equal to the average ratio in the group. This ratio is consistently determined based on the number of lines in the text extracted from the framework. That is why the reference texts are manually extracted from the extracted text to match the same line structure and character encoding as the strategy texts to prevent complications during comparisons.

### 3.5.1 Hypotheses

The following hypotheses are based on the assumptions made in chapter 1 and tested on the computed ROUGE-N scores. The implemented strategies score with ROUGE-1 equally on text highlighting and text summarization. This hypothesis can be decomposed into four smaller hypotheses, one for every comparison. The ROUGE-1 scores on text summarization is computed and acquired from (Mathur, Gill, and Yadav, 2017).

In addition to the assumptions, Mathur concludes that LexRank scores a higher ROUGE-N score than LSA, TextRank or TextTeaser on text summarization (Mathur, Gill, and Yadav, 2017). It is reasonable to hypothesize that LexRank will also score a higher ROUGE-N score than the implemented strategies on text highlighting. This hypothesis can also be decomposed into three smaller hypotheses, one for every comparison.

[1] Rouge 2.0 - https://github.com/RxNLP/ROUGE-2.0

# Chapter 4

# Results

The results of the pilot and the experiment are presented in this chapter. The highlighted documents collected with the pilot are utilized as reference documents in the experiment. To increase readability, the compression ratio is multiplied by a factor of one hundred.

## 4.1 Pilot

Twelve participants returned highlighted documents, of whom eleven returned all four documents mentioned in table 3.1. The other participant stopped after the first document because of difficulties with the language. This lead to 45 highlighted documents. No participant has had explicit training in text highlighting. Little consistency can be found between participants in their compression ratio and challenge rating; see table 4.1. On the other hand, it seems that a lower challenge rating results in a smaller standard deviation but this has not been tested.

On a side note, the smallest highlighted sequence measured was 2. These sequences were mostly two word concepts or references e.g. "Table 1". Therefore the second set of hypotheses are tested on data obtained through ROUGE-2.

For every document, its reference text is manually build and allocated into one of the three ratio groups; see table 4.2. Due to the variability of participants' compression ratio, participants happened to be allocated into different groups per document. For example, participant 4 has a ratio of 18, 26, 11 and 12, what leads to the allocation of the reference text to $d0g1$, $d1g2$, $d2g1$ and $d3g1$ respectively. The goal was to decrease the distance between participant ratio and strategy ratio without redundant dispersion. The strategy ratio is determined by the average participant ratio per group.

### 4.1.1 Strategy identification

The identification of applied strategies was a difficult process. Each participant inconsistently used multiple techniques that flow through each other. The process of identification would be subjective and prone to error because it would be executed with the naked eye. In addition, it added little value to the experiment, as it was no part of the selection criteria of the algorithms to implement.

At a first glance, a couple of techniques could be identified. These techniques did not hold for all participants but are mentioned as they occurred. Passages between parentheses were often skipped, these usually contained a reference, explanation of an abbreviation or an example. Some participants started highlighting after the occurrence of document type specific *trigger words*, e.g. *hypothesize* or *concluding* and after conjunctions. Keywords and text with a deviating font or size were also more

in favor of getting highlighted. Moreover, among the participants, one or two participants attempted to change the structure of sentences by only highlighting parts of sentences but in a way that highlighted parts could be read independently as grammatically correct sentences. And one participant explained afterwards that when she considered a whole section important, she highlighted only the head of the section as an implicit reference to the complete section.

## 4.2 Experiment

The applicable data from (Mathur, Gill, and Yadav, 2017) is presented in table 4.3. Other algorithms and the analogous BLEU scores (Papineni et al., 2002) are omitted. This data is obtained from 51 user reviews with each 5 summaries, which leads to a total sample size of 255 per algorithm.

For every document, for every group, the application is executed with the four strategies and with the corresponding strategy ratio. This resulted in 48 unique strategy texts to be tested. ROUGE-1 is computed to test the first set of hypotheses and Rouge 2.0 typically computes for ROUGE-N the Mean recall, precision and F-Score. However, to keep more control over the data Rouge 2.0 was executed per participant what resulted in individual recall, precision and F-Scores. In this way, 192 data entries were obtained of which 12 entries were regarded as missing data.

After evaluation of the results on the second hypothesis, it became interesting to test another hypothesis. The third hypothesis states that TextRank scores a higher ROUGE-2 score than LexRank, LSA or TextTeaser on text highlighting. This hypothesis can be decomposed into three smaller hypotheses, one for every comparison.

### 4.2.1 Hypothesis 1

For ROUGE-1 the Mean recall, precision and F-Scores, joined with the Median, standard deviation and max value are presented in table 4.4. Based on the data from table 4.3 and 4.4 the first hypothesis is tested:

(a) Mean recall of LRK scored significantly higher on text highlighting (0.6520, $n = 45$) than on text summarization (0.260, $n = 255$) (twosample t-test, $p < 0.001$);

(b) Mean recall of LSA scored significantly higher on text highlighting (0.6123, $n = 45$) than on text summarization (0.211, $n = 255$) (twosample t-test, $p < 0.001$);

(c) Mean recall of TRK scored significantly higher on text highlighting (0.6667, $n = 45$) than on text summarization (0.230, $n = 255$) (twosample t-test, $p < 0.001$) and

(d) Mean recall of TTS scored significantly higher on text highlighting (0.5309, $n = 45$) than on text summarization (0.221, $n = 255$) (twosample t-test, $p < 0.001$).

### 4.2.2 Hypothesis 2

For ROUGE-2 the Mean recall, precision and F-Scores, joined with the Median, standard deviation and max value are presented in table 4.5. Based on the data from table 4.5 the second hypothesis is tested:

(a) Mean recall of LRK (0.3992, $n = 45$) scored not significantly higher than the Mean recall of LSA (0.3511, $n = 45$) (twosample t-test, $p > 0.05$);

(b) Mean recall of LRK (0.3992, $n = 45$) scored not significantly higher than the Mean recall of TRK (0.4322, $n = 45$) (twosample t-test, $p > 0.05$) and

(c) Mean recall of LRK (0.3992, $n = 45$) scored significantly higher than the Mean recall of TTS (0.3178, $n = 45$) (twosample t-test, $p < 0.005$).

### 4.2.3 Hypothesis 3

For ROUGE-2 the Mean recall, precision and F-Scores, joined with the Median, standard deviation and max value are presented in table 4.5. Based on the data from table 4.5 the third hypothesis is tested:

(a) Mean recall of TRK (0.4322, $n = 45$) scored significantly higher than the Mean recall of LSA (0.3511, $n = 45$) (twosample t-test, $p < 0.025$);

(b) Mean recall of TRK (0.4322, $n = 45$) scored not significantly higher than the Mean recall of LRK (0.3992, $n = 45$) (twosample t-test, $p > 0.05$) and

(c) Mean recall of TRK (0.4322, $n = 45$) scored significantly higher than the Mean recall of TTS (0.3178, $n = 45$) (twosample t-test, $p < 0.001$).

TABLE 4.1: Consistency between and within participants

| | | Participant Ratio | | | | | |
|---|---|---|---|---|---|---|---|
| | | All | | Challenge <3 | | Challenge >= 3 | |
| ParticipantID | N Documents | Mean | s | Mean | s | Mean | s |
| 0 | 4 | 17.50 | 05.26 | 15.00 | 02.83 | 20.00 | 07.07 |
| 1 | 4 | 10.50 | 05.74 | 13.00 | 08.49 | 08.00 | 01.41 |
| 2 | 1 | 42.00 | 00.00 | 00.00 | 00.00 | 42.00 | 00.00 |
| 3 | 4 | 33.25 | 14.55 | 24.50 | 06.36 | 42.00 | 16.97 |
| 4 | 4 | 16.75 | 06.90 | 14.50 | 04.95 | 19.00 | 09.90 |
| 5 | 4 | 14.25 | 04.57 | 15.50 | 04.95 | 13.00 | 05.66 |
| 6 | 4 | 26.00 | 08.49 | 20.00 | 00.00 | 28.00 | 04.24 |
| 7 | 4 | 28.00 | 08.76 | 27.50 | 09.19 | 28.50 | 12.02 |
| 8 | 4 | 20.50 | 08.89 | 22.50 | 14.85 | 18.50 | 00.71 |
| 9 | 4 | 06.75 | 02.87 | 08.50 | 02.12 | 05.00 | 02.83 |
| 10 | 4 | 21.25 | 11.67 | 19.50 | 02.12 | 23.00 | 19.80 |
| 11 | 4 | 11.25 | 05.32 | 14.00 | 07.07 | 08.50 | 02.12 |

TABLE 4.2: The subdivision of references with corresponding strategy ratio

| DocumentID | GroupID | N References | Strategy Ratio |
|---|---|---|---|
| d0 | 0 | 1 | 10 |
| | 1 | 7 | 19 |
| | 2 | 4 | 37 |
| d1 | 0 | 3 | 09 |
| | 1 | 5 | 23 |
| | 2 | 3 | 43 |
| d2 | 0 | 7 | 10 |
| | 1 | 3 | 20 |
| | 2 | 1 | 29 |
| d3 | 0 | 6 | 08 |
| | 1 | 4 | 18 |
| | 2 | 1 | 30 |

TABLE 4.3: ROUGE-1 scores on text summarization (Mathur, Gill, and Yadav, 2017)

| ROUGE-1 | Recall | |
|---|---|---|
| Strategy | Mean | s |
| LRK | 0.260 | 0.060 |
| LSA | 0.211 | 0.059 |
| TRK | 0.230 | 0.058 |
| TTS | 0.221 | 0.053 |

TABLE 4.4: ROUGE-1 scores for every implemented strategy

| ROUGE-1 | | Recall | | | | Precision | | | | F-Score | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Strategy | N Texts | Mean | Median | s | Max | Mean | Median | s | Max | Mean | Median | s | Max |
| LRK | 45 | 0.6520 | 0.6524 | 0.1142 | 0.8510 | 0.4910 | 0.4928 | 0.1529 | 0.8155 | 0.5447 | 0.5530 | 0.1207 | 0.8198 |
| LSA | 45 | 0.6123 | 0.5792 | 0.1215 | 0.8479 | 0.4737 | 0.4391 | 0.1546 | 0.7748 | 0.5169 | 0.5226 | 0.1180 | 0.7502 |
| TRK | 45 | 0.6759 | 0.6667 | 0.1180 | 0.9274 | 0.4940 | 0.5064 | 0.1409 | 0.7860 | 0.5596 | 0.5585 | 0.1227 | 0.8073 |
| TTS | 45 | 0.5309 | 0.5207 | 0.1208 | 0.7496 | 0.5866 | 0.6204 | 0.1325 | 0.7983 | 0.5469 | 0.5430 | 0.1056 | 0.7145 |

TABLE 4.5: ROUGE-2 scores for every implemented strategy

| ROUGE-2 | | Recall | | | | Precision | | | | F-Score | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Strategy | N Texts | Mean | Median | s | Max | Mean | Median | s | Max | Mean | Median | s | Max |
| LRK | 45 | 0.3992 | 0.3926 | 0.1537 | 0.6874 | 0.3037 | 0.2979 | 0.1514 | 0.6690 | 0.3357 | 0.3272 | 0.1458 | 0.6781 |
| LSA | 45 | 0.3511 | 0.3251 | 0.1543 | 0.6637 | 0.2726 | 0.2558 | 0.1423 | 0.6010 | 0.2971 | 0.2962 | 0.1373 | 0.5862 |
| TRK | 45 | 0.4322 | 0.4350 | 0.1775 | 0.8399 | 0.3149 | 0.2983 | 0.1545 | 0.6674 | 0.3577 | 0.3372 | 0.1596 | 0.6912 |
| TTS | 45 | 0.3178 | 0.3050 | 0.1349 | 0.5441 | 0.3512 | 0.3492 | 0.1493 | 0.6688 | 0.3274 | 0.3400 | 0.1342 | 0.5821 |

# Chapter 5

# Discussion

The results are unable to justify the earlier made assumption that highlighted documents from untrained people are similar to summaries of the document. Therefore the pilot needed to be more extensive. Even then it would be difficult to substantiate because participants should not summarize and highlight the same document to prevent interaction.

## 5.1 Hypothesis 1

On the basis of the results on the first hypothesis, it seems that the implemented algorithms score better on text highlighting than text summarization. On all four tests, the algorithms score unbelievably high in contrast to the data from (Mathur, Gill, and Yadav, 2017). A combination of factors could be the underlying cause.

For one, the experiment tested the implemented algorithms in a range of compression ratios instead of a fixed compression ratio as Mathur did in (Mathur, Gill, and Yadav, 2017). This would give the algorithms a bit more leeway in their strict sentence selection process, resulting in a higher score. Also, there is no absolute text highlighting format. The slight variation, in the gathered highlighted documents, that was expected turned out to be much larger. A highlighting heat-map of a document would light up on the complete document with only a few sentences to be *extra hot*. Since the variation is large and the ROUGE-N scores are computed for every document individually, one would expect a low precision score and therefore a drop in the F-Score. However, as can be seen in table 4.4, as well as in table 4.5, the F-Score falls in line of expectation. Lastly, a ROUGE-1 score tends to give a better representation of performance on a smaller text with few to no duplicate words. As duplicate words score multiple times as an unigrams even if the reference text only contains the word once.

Apart from the average ROUGE-1 score and the standard deviation no other data was presented by (Mathur, Gill, and Yadav, 2017). To reliably execute a two sample t-test, the data needs to have an approximation of a normal distribution, i.e. a bell curve. This could not be verified for the data from (Mathur, Gill, and Yadav, 2017). Besides, the distribution for the data in table 4.4 is verified and therefore the distribution is assumed for the data from (Mathur, Gill, and Yadav, 2017); see Appendix A.

In retrospect, the ROUGE-N score was not the best fit to compare datasets, containing different sizes in text, with. For every chosen size of n-grams, ROUGE-N would always over- or underestimate one of the datasets. This leads to a skewed perspective.

## 5.2 Hypotheses 2 & 3

On the basis of the results on the second hypothesis, it seems that LexRank scores not significantly better on text highlighting than LSA or TextRank. And LexRank scores significantly better on text highlighting than TextTeaser. In the case of LSA and TextRank, the two sample t-test rejects the hypotheses. However, TextRank is the only algorithm of which the ROUGE-N score is located on the right tail of the distribution. For that reason a third hypothesis was tested.

From the results on the final hypothesis it can be concluded that TextRank scores significantly better on text highlighting than the other algorithms, with the exception of LexRank. Although, none of the implemented algorithms undoubtedly stand out, it seems that the the derivatives of PageRank, i.e. LexRank and TextRank, have a slight edge over the others.

The presented ROUGE-2 scores are notably lower than the ROUGE-1 scores. This can be explained with bigrams that are used in the computations of ROUGE-2. The use of bigrams reduces the misleading *true positive* hits on duplicate words as the order within n-grams becomes more important for a larger $n$.

The distribution for the data in table 4.5 corresponds with the distribution criteria to execute a two sample t-test; see Appendix A. In contrast to hypothesis 1, the samples for hypotheses 2 & 3 originate from the same dataset containing similar sized texts, so the representation of the ROUGE-N score should be more consistent. In opposition, subjective evaluation might suggest a different algorithm to perform better. In Appendix B, a set of examples from the data is included to be judged.

## 5.3 Framework

The constructed framework, in its current state, is still in its infancy. The parsing process of a PDF file contains some troublesome drawbacks. Although, PDF is a robust standard which makes PDF files easy to read by people, the generation of PDF files is not standard. This causes a lot of loose variables to be freely interpreted by parsing agents that actually need to be controlled. For example, some documents register both hard and soft returns in their text, some only register only hard returns, while other do not register returns at all. Even so, that text columns are ignored and most of the parsed text end up on the same line. To be consistent between reference texts and strategy texts regarding odd parsing artifacts, the documents from the pilot had to be manually extracted into a text file. There was not enough time for this thesis to take every variation of PDF into account in the framework. To overcome this issue a lot of fine-tuning is needed or one should consider a library other than the iText Library.

To add to that, the framework contains a couple of non-fundamental bugs regarding the user experience. The user experience was not an essential goal for the thesis, no time has yet been invested in ironing out these bugs. For instance, the framework and the implemented strategies work with the same two files. However, no checks are in place to make sure that both parts await turns. This causes the user to execute the application multiple times to make sure the correct two files are used for the given strategy.

## 5.4 Implementation

No research could be found on automatic text highlighting besides one paper, and a patent from the same author, describing a document specific system based on a spreading activation model (Chi et al., 2005). The implemented algorithms are chosen by reason of their presence in related work and their availability. They are facilitated, without changing their functioning, to explore the results of one to one mapping.

In this state, the algorithms were limited in ranking texts per sentence as punctuation is their only splitting condition. Before the pilot was held, this was considered a major limitation for the capability of automated text highlighting. In view of, an early perspective that believed the technique of highlighting parts of sentences instead of full sentences was more common. As it turned out, this was not the case and the algorithms held up surprisingly well.

## 5.5 Pilot

Multiple datasets exist containing a large set of, often news, articles with associated *golden summaries*. Unfortunately, no applicable dataset could be found with associated highlighting or other annotations. With limited resources a dataset had to be realized.

The pilot was dependent on voluntary participation. This causes the applied techniques to documents in the dataset, that is build from the pilot, to be demographically limited. Therefore applied techniques to create the highlighted documents might be a skewed representation of the population. Although, participants were given a week to work with the four documents, some might have procrastinated until the last day. This could have interfered with the potential quality of the highlighting due to cognitive exhaustion. Unfortunately, after the event, this is difficult to test. In hindsight, the participants should not have been given this amount of freedom but should have multiple bite-sized assignments instead.

In this thesis, the focus remained towards the experiment and a functioning application would be ancillary. As the implemented algorithms were chosen by availability there was little reason anymore to prioritize the identification of strategies employed by participants. After a short consideration, it became evident that empirically charting highlighting strategies is a study on its own. No explanation has been given for the inconsistency of highlighting strategies within participants but it is hinted at that this is document dependent.

# Chapter 6

# Conclusion

In this thesis, it is shown that it is not unthinkable to utilize text summarization algorithms as text extraction strategies for automated text highlighting. A method and an application have been realized as an example of how to implement and test a text highlighting application. Still a lot has to be researched before automatic generated summaries are indistinguishable from human generated summaries. This, unquestionably, holds also true for text highlighting. In ways, text highlighting is more dependent of personal taste than text summarization is. However, this should not stand in the way of creating smart tools for information processing in a time of an ever-increasing demand of information.

## 6.1 Future research

A couple of questions can be raised as a result of this thesis. This leaves an opening for for future research. The algorithms implemented in the application are only heuristic of type. It would be interesting to research other types of text summarization algorithms. How do corpus-based text summarization algorithms or machine learning algorithms perform on text highlighting?

There is no single applicable strategy to fit every variation on text highlighting. More layers of these strategies need to be unraveled to be able to build a high performing application. What concrete variables determine a highlighting strategy? In the end, does automatic text highlighting offer the same level of text comprehension as the analogous counterpart? If so, who could actually benefit from an automatic text highlighting application?

# Appendix A

# Distribution of data

This appendix contains the additional justification towards a close to normal distribution of the data and is presented as two box plots in figure A.1 and A.2.

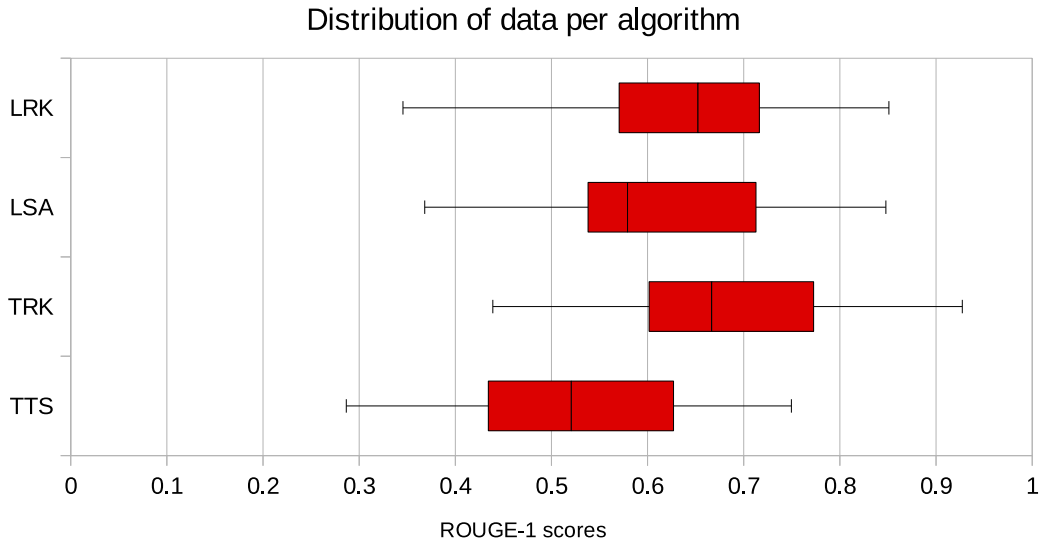



FIGURE A.1: The distribution of ROUGE-1 scores per algorithm depicted in quartiles.

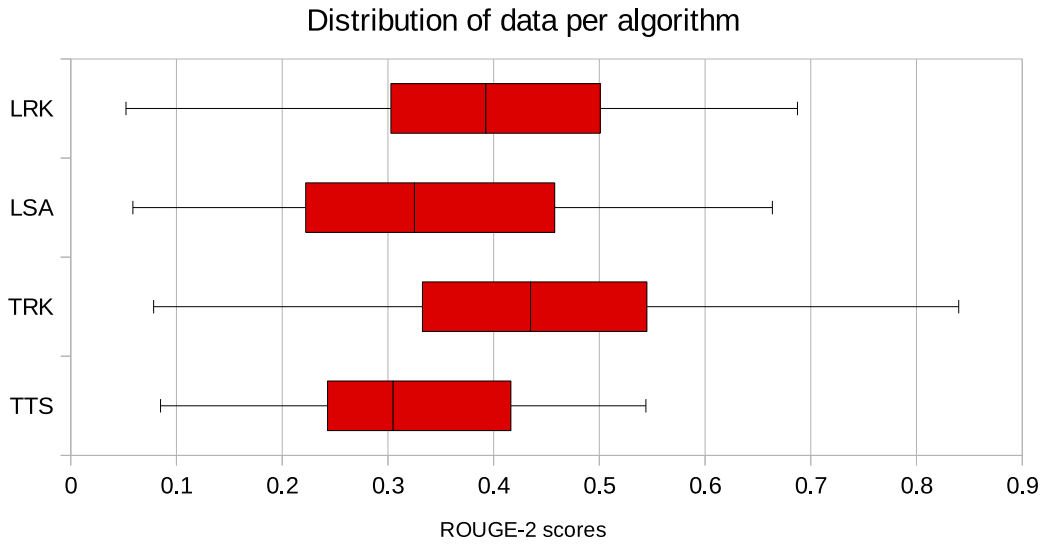
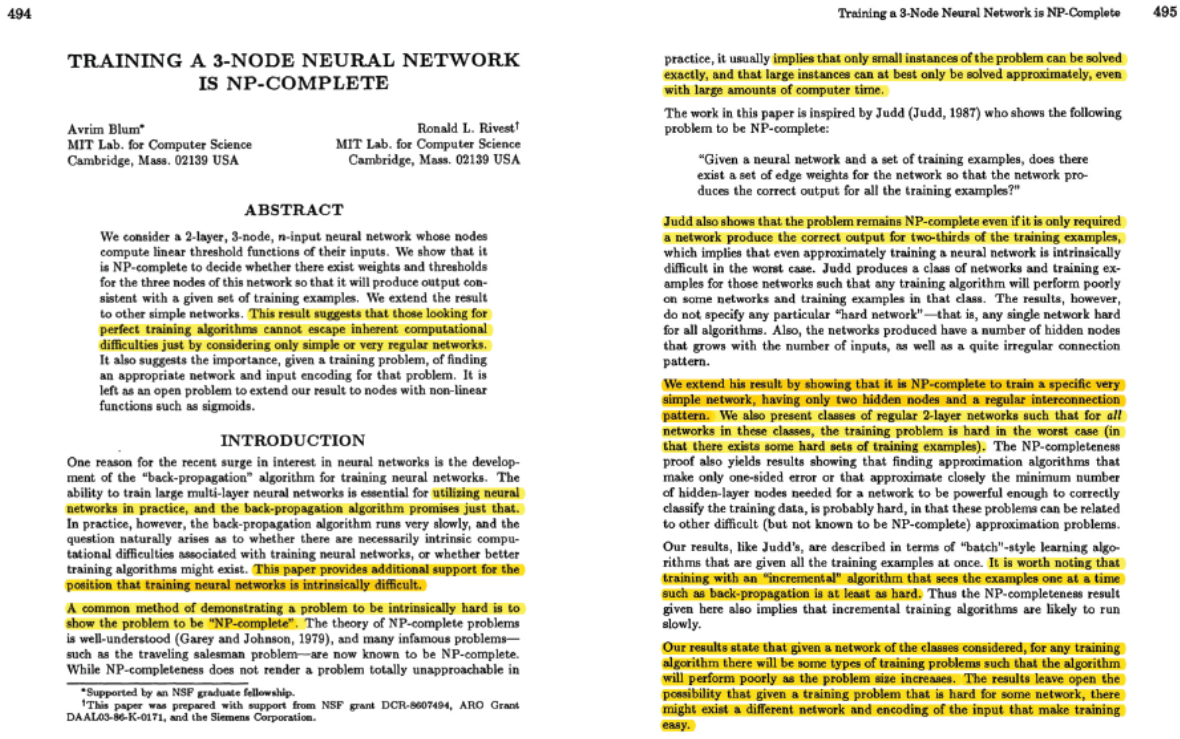



FIGURE A.2: The distribution of ROUGE-2 scores per algorithm depicted in quartiles.

# Appendix B

# Figures of Example Data

For every group, two figures are listed that represent the first two pages of document $d1$ twice. For the figure of the human-highlighted document, the first two frames present a heat map of all participants of the group, followed by two frames that depict one participant. For the figure of the application-highlighted documents, the first two frames represent the worst performing algorithm, followed by the best performing algorithm for the particular participant. Details will be present in the captions.



494

## TRAINING A 3-NODE NEURAL NETWORK IS NP-COMPLETE

Avrim Blum*
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

Ronald L. Rivest†
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

### ABSTRACT

We consider a 2-layer, 3-node, $n$-input neural network whose nodes compute linear threshold functions of their inputs. We show that it is NP-complete to decide whether there exist weights and thresholds for the three nodes of this network so that it will produce output consistent with a given set of training examples. We extend the result to other simple networks. This result suggests that those looking for perfect training algorithms cannot escape inherent computational difficulties just by considering only simple or very regular networks. It also suggests the importance, given a training problem, of finding an appropriate network and input encoding for that problem. It is left as an open problem to extend our result to nodes with non-linear functions such as sigmoids.

### INTRODUCTION

One reason for the recent surge in interest in neural networks is the development of the "back-propagation" algorithm for training neural networks. The ability to train large multi-layer neural networks is essential for utilizing neural networks in practice, and the back-propagation algorithm promises just that. In practice, however, the back-propagation algorithm runs very slowly, and the question naturally arises as to whether there are necessarily intrinsic computational difficulties associated with training neural networks, or whether better training algorithms might exist. This paper provides additional support for the position that training neural networks is intrinsically difficult.

A common method of demonstrating a problem to be intrinsically hard is to show the problem to be "NP-complete". The theory of NP-complete problems is well-understood (Garey and Johnson, 1979), and many infamous problems—such as the traveling salesman problem—are now known to be NP-complete. While NP-completeness does not render a problem totally unapproachable in

*Supported by an NSF graduate fellowship.
†This paper was prepared with support from NSF grant DCR-8607494, ARO Grant DAAL03-86-K-0171, and the Siemens Corporation.



practice, it usually implies that only small instances of the problem can be solved exactly, and that large instances can at best only be solved approximately, even with large amounts of computer time.

The work in this paper is inspired by Judd (Judd, 1987) who shows the following problem to be NP-complete:

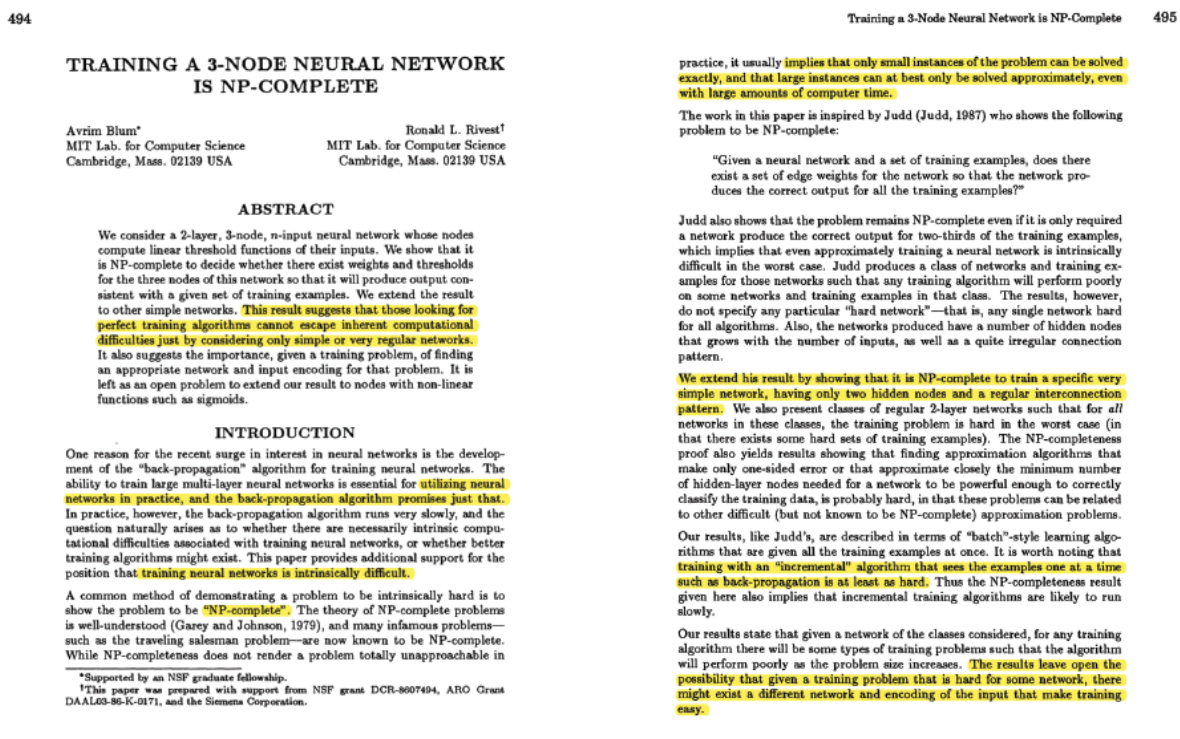
> "Given a neural network and a set of training examples, does there exist a set of edge weights for the network so that the network produces the correct output for all the training examples?"

Judd also shows that the problem remains NP-complete even if it is only required a network produce the correct output for two-thirds of the training examples, which implies that even approximately training a neural network is intrinsically difficult in the worst case. Judd produces a class of networks and training examples for those networks such that any training algorithm will perform poorly on some networks and training examples in that class. The results, however, do not specify any particular "hard network"—that is, any single network hard for all algorithms. Also, the networks produced have a number of hidden nodes that grows with the number of inputs, as well as a quite irregular connection pattern.

We extend his result by showing that it is NP-complete to train a specific very simple network, having only two hidden nodes and a regular interconnection pattern. We also present classes of regular 2-layer networks such that for *all* networks in these classes, the training problem is hard in the worst case (in that there exists some hard sets of training examples). The NP-completeness proof also yields results showing that finding approximation algorithms that make only one-sided error or that approximate closely the minimum number of hidden-layer nodes needed for a network to be powerful enough to correctly classify the training data, is probably hard, in that these problems can be related to other difficult (but not known to be NP-complete) approximation problems.

Our results, like Judd's, are described in terms of "batch"-style learning algorithms that are given all the training examples at once. It is worth noting that training with an "incremental" algorithm that sees the examples one at a time such as back-propagation is at least as hard. Thus the NP-completeness result given here also implies that incremental training algorithms are likely to run slowly.

Our results state that given a network of the classes considered, for any training algorithm there will be some types of training problems such that the algorithm will perform poorly as the problem size increases. The results leave open the possibility that given a training problem that is hard for some network, there might exist a different network and encoding of the input that make training easy.



494

TRAINING A 3-NODE NEURAL NETWORK IS NP-COMPLETE

Avrim Blum*
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

Ronald L. Rivest†
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

ABSTRACT

We consider a 2-layer, 3-node, $n$-input neural network whose nodes compute linear threshold functions of their inputs. We show that it is NP-complete to decide whether there exist weights and thresholds for the three nodes of this network so that it will produce output consistent with a given set of training examples. We extend the result to other simple networks. This result suggests that those looking for perfect training algorithms cannot escape inherent computational difficulties just by considering only simple or very regular networks. It also suggests the importance, given a training problem, of finding an appropriate network and input encoding for that problem. It is left as an open problem to extend our result to nodes with non-linear functions such as sigmoids.

INTRODUCTION

One reason for the recent surge in interest in neural networks is the development of the "back-propagation" algorithm for training neural networks. The ability to train large multi-layer neural networks is essential for utilizing neural networks in practice, and the back-propagation algorithm promises just that. In practice, however, the back-propagation algorithm runs very slowly, and the question naturally arises as to whether there are necessarily intrinsic computational difficulties associated with training neural networks, or whether better training algorithms might exist. This paper provides additional support for the position that training neural networks is intrinsically difficult.

A common method of demonstrating a problem to be intrinsically hard is to show the problem to be "NP-complete". The theory of NP-complete problems is well-understood (Garey and Johnson, 1979), and many infamous problems—such as the traveling salesman problem—are now known to be NP-complete. While NP-completeness does not render a problem totally unapproachable in

*Supported by an NSF graduate fellowship.
†This paper was prepared with support from NSF grant DCR-8607494, ARO Grant DAAL03-86-K-0171, and the Siemens Corporation.

Training a 3-Node Neural Network is NP-Complete 495

practice, it usually implies that only small instances of the problem can be solved exactly, and that large instances can at best only be solved approximately, even with large amounts of computer time.

The work in this paper is inspired by Judd (Judd, 1987) who shows the following problem to be NP-complete:

> "Given a neural network and a set of training examples, does there exist a set of edge weights for the network so that the network produces the correct output for all the training examples?"

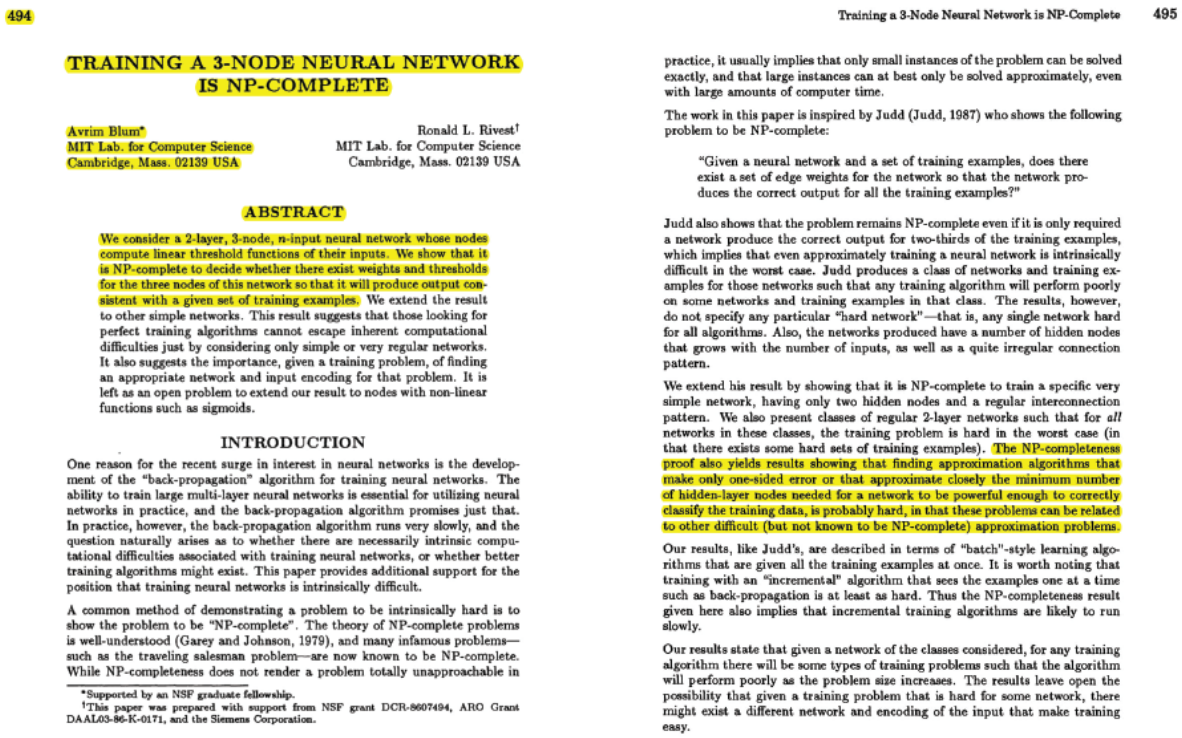
Judd also shows that the problem remains NP-complete even if it is only required a network produce the correct output for two-thirds of the training examples, which implies that even approximately training a neural network is intrinsically difficult in the worst case. Judd produces a class of networks and training examples for those networks such that any training algorithm will perform poorly on some networks and training examples in that class. The results, however, do not specify any particular "hard network"—that is, any single network hard for all algorithms. Also, the networks produced have a number of hidden nodes that grows with the number of inputs, as well as a quite irregular connection pattern.

We extend his result by showing that it is NP-complete to train a specific very simple network, having only two hidden nodes and a regular interconnection pattern. We also present classes of regular 2-layer networks such that for *all* networks in these classes, the training problem is hard in the worst case (in that there exists some hard sets of training examples). The NP-completeness proof also yields results showing that finding approximation algorithms that make only one-sided error or that approximate closely the minimum number of hidden-layer nodes needed for a network to be powerful enough to correctly classify the training data, is probably hard, in that these problems can be related to other difficult (but not known to be NP-complete) approximation problems.

Our results, like Judd's, are described in terms of "batch"-style learning algorithms that are given all the training examples at once. It is worth noting that training with an "incremental" algorithm that sees the examples one at a time such as back-propagation is at least as hard. Thus the NP-completeness result given here also implies that incremental training algorithms are likely to run slowly.

Our results state that given a network of the classes considered, for any training algorithm there will be some types of training problems such that the algorithm will perform poorly as the problem size increases. The results leave open the possibility that given a training problem that is hard for some network, there might exist a different network and encoding of the input that make training easy.

FIGURE B.1: Top: heat map of group 0; Bottom: participant no. 9, ratio = 0.07.



494

## TRAINING A 3-NODE NEURAL NETWORK IS NP-COMPLETE

Avrim Blum*
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

Ronald L. Rivest†
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

### ABSTRACT

We consider a 2-layer, 3-node, $n$-input neural network whose nodes compute linear threshold functions of their inputs. We show that it is NP-complete to decide whether there exist weights and thresholds for the three nodes of this network so that it will produce output consistent with a given set of training examples. We extend the result to other simple networks. This result suggests that those looking for perfect training algorithms cannot escape inherent computational difficulties just by considering only simple or very regular networks. It also suggests the importance, given a training problem, of finding an appropriate network and input encoding for that problem. It is left as an open problem to extend our result to nodes with non-linear functions such as sigmoids.

### INTRODUCTION

One reason for the recent surge in interest in neural networks is the development of the "back-propagation" algorithm for training neural networks. The ability to train large multi-layer neural networks is essential for utilizing neural networks in practice, and the back-propagation algorithm promises just that. In practice, however, the back-propagation algorithm runs very slowly, and the question naturally arises as to whether there are necessarily intrinsic computational difficulties associated with training neural networks, or whether better training algorithms might exist. This paper provides additional support for the position that training neural networks is intrinsically difficult.

A common method of demonstrating a problem to be intrinsically hard is to show the problem to be "NP-complete". The theory of NP-complete problems is well-understood (Garey and Johnson, 1979), and many infamous problems—such as the traveling salesman problem—are now known to be NP-complete. While NP-completeness does not render a problem totally unapproachable in

*Supported by an NSF graduate fellowship.
†This paper was prepared with support from NSF grant DCR-8607494, ARO Grant DAAL03-86-K-0171, and the Siemens Corporation.



practice, it usually implies that only small instances of the problem can be solved exactly, and that large instances can at best only be solved approximately, even with large amounts of computer time.

The work in this paper is inspired by Judd (Judd, 1987) who shows the following problem to be NP-complete:

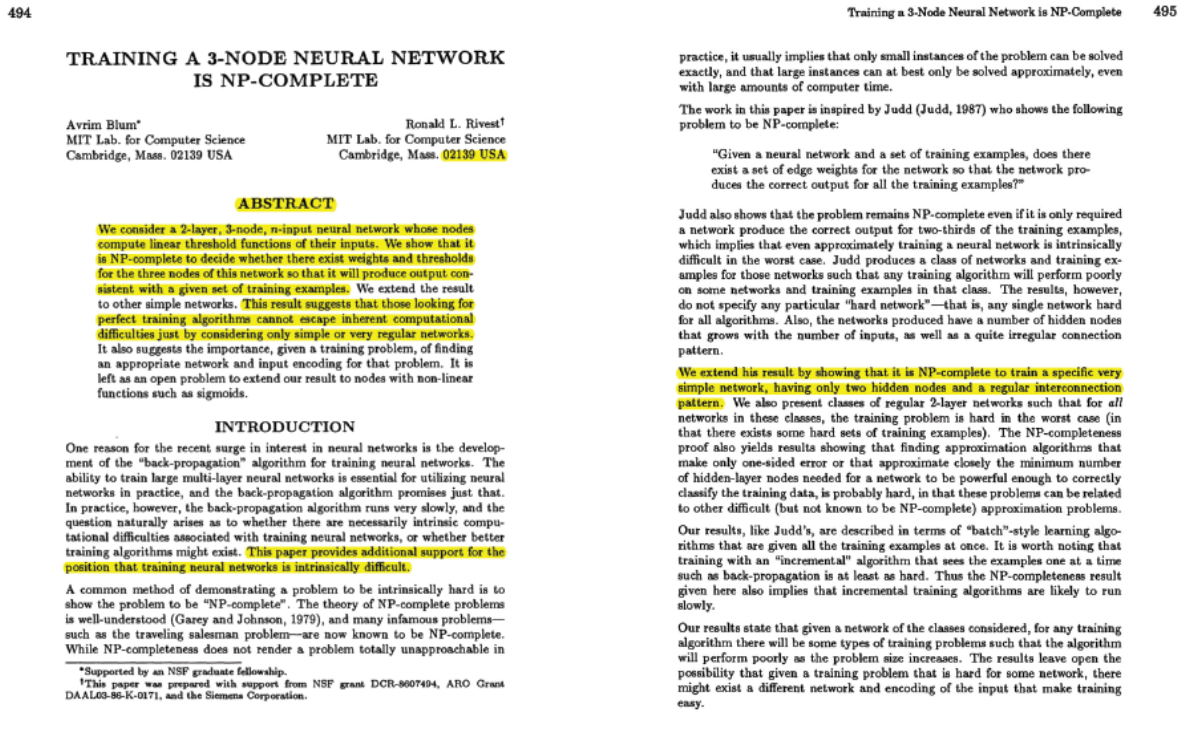
> "Given a neural network and a set of training examples, does there exist a set of edge weights for the network so that the network produces the correct output for all the training examples?"

Judd also shows that the problem remains NP-complete even if it is only required a network produce the correct output for two-thirds of the training examples, which implies that even approximately training a neural network is intrinsically difficult in the worst case. Judd produces a class of networks and training examples for those networks such that any training algorithm will perform poorly on some networks and training examples in that class. The results, however, do not specify any particular "hard network"—that is, any single network hard for all algorithms. Also, the networks produced have a number of hidden nodes that grows with the number of inputs, as well as a quite irregular connection pattern.

We extend his result by showing that it is NP-complete to train a specific very simple network, having only two hidden nodes and a regular interconnection pattern. We also present classes of regular 2-layer networks such that for *all* networks in these classes, the training problem is hard in the worst case (in that there exists some hard sets of training examples). The NP-completeness proof also yields results showing that finding approximation algorithms that make only one-sided error or that approximate closely the minimum number of hidden-layer nodes needed for a network to be powerful enough to correctly classify the training data, is probably hard, in that these problems can be related to other difficult (but not known to be NP-complete) approximation problems.

Our results, like Judd's, are described in terms of "batch"-style learning algorithms that are given all the training examples at once. It is worth noting that training with an "incremental" algorithm that sees the examples one at a time such as back-propagation is at least as hard. Thus the NP-completeness result given here also implies that incremental training algorithms are likely to run slowly.

Our results state that given a network of the classes considered, for any training algorithm there will be some types of training problems such that the algorithm will perform poorly as the problem size increases. The results leave open the possibility that given a training problem that is hard for some network, there might exist a different network and encoding of the input that make training easy.



494

TRAINING A 3-NODE NEURAL NETWORK IS NP-COMPLETE

Avrim Blum*
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

Ronald L. Rivest†
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

ABSTRACT

We consider a 2-layer, 3-node, $n$-input neural network whose nodes compute linear threshold functions of their inputs. We show that it is NP-complete to decide whether there exist weights and thresholds for the three nodes of this network so that it will produce output consistent with a given set of training examples. We extend the result to other simple networks. This result suggests that those looking for perfect training algorithms cannot escape inherent computational difficulties just by considering only simple or very regular networks. It also suggests the importance, given a training problem, of finding an appropriate network and input encoding for that problem. It is left as an open problem to extend our result to nodes with non-linear functions such as sigmoids.

INTRODUCTION

One reason for the recent surge in interest in neural networks is the development of the "back-propagation" algorithm for training neural networks. The ability to train large multi-layer neural networks is essential for utilizing neural networks in practice, and the back-propagation algorithm promises just that. In practice, however, the back-propagation algorithm runs very slowly, and the question naturally arises as to whether there are necessarily intrinsic computational difficulties associated with training neural networks, or whether better training algorithms might exist. This paper provides additional support for the position that training neural networks is intrinsically difficult.

A common method of demonstrating a problem to be intrinsically hard is to show the problem to be "NP-complete". The theory of NP-complete problems is well-understood (Garey and Johnson, 1979), and many infamous problems—such as the traveling salesman problem—are now known to be NP-complete. While NP-completeness does not render a problem totally unapproachable in

*Supported by an NSF graduate fellowship.
†This paper was prepared with support from NSF grant DCR-8607494, ARO Grant DAAL03-86-K-0171, and the Siemens Corporation.

Training a 3-Node Neural Network is NP-Complete 495

practice, it usually implies that only small instances of the problem can be solved exactly, and that large instances can at best only be solved approximately, even with large amounts of computer time.

The work in this paper is inspired by Judd (Judd, 1987) who shows the following problem to be NP-complete:

> "Given a neural network and a set of training examples, does there exist a set of edge weights for the network so that the network produces the correct output for all the training examples?"

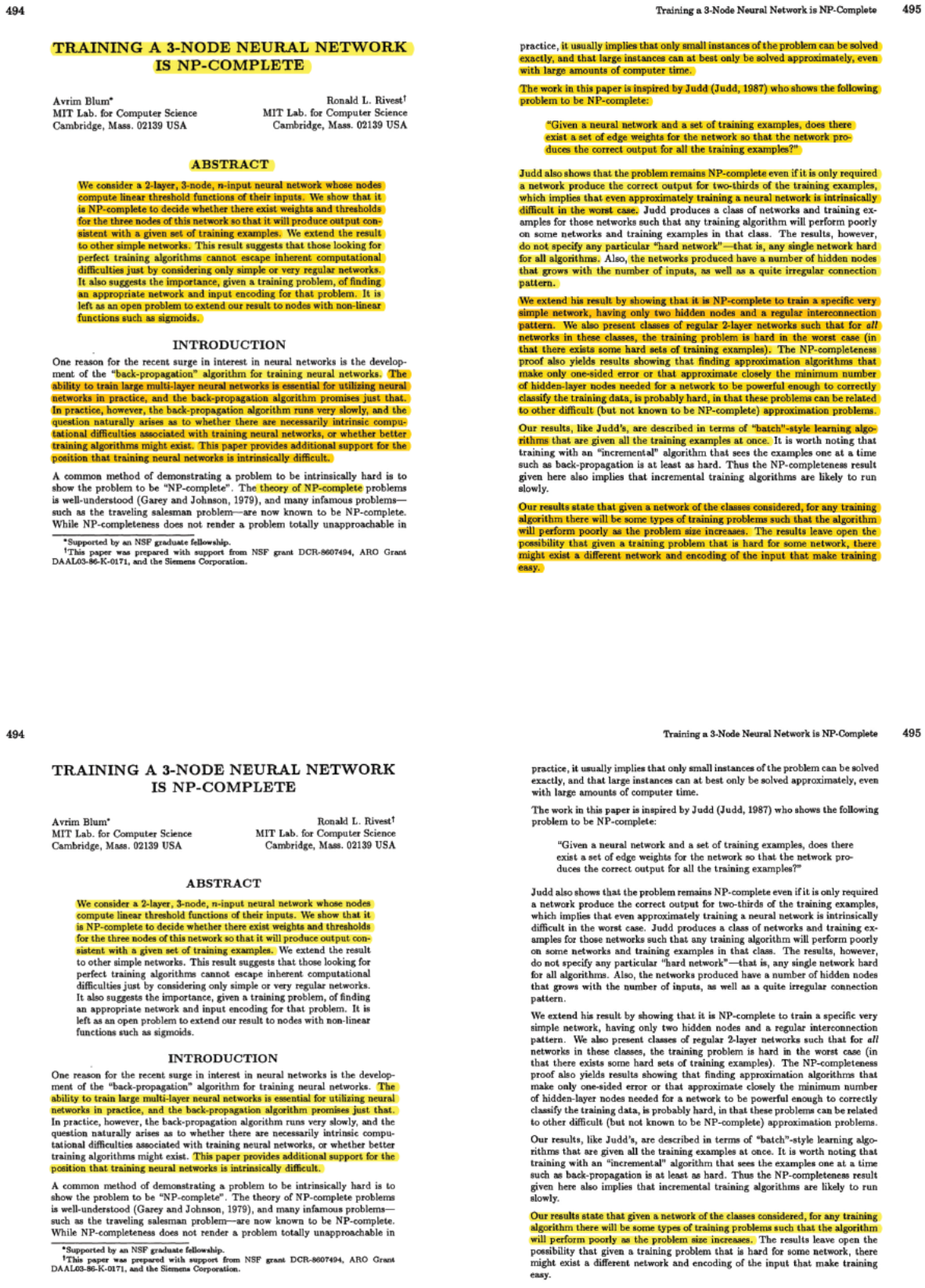
Judd also shows that the problem remains NP-complete even if it is only required a network produce the correct output for two-thirds of the training examples, which implies that even approximately training a neural network is intrinsically difficult in the worst case. Judd produces a class of networks and training examples for those networks such that any training algorithm will perform poorly on some networks and training examples in that class. The results, however, do not specify any particular "hard network"—that is, any single network hard for all algorithms. Also, the networks produced have a number of hidden nodes that grows with the number of inputs, as well as a quite irregular connection pattern.

We extend his result by showing that it is NP-complete to train a specific very simple network, having only two hidden nodes and a regular interconnection pattern. We also present classes of regular 2-layer networks such that for *all* networks in these classes, the training problem is hard in the worst case (in that there exists some hard sets of training examples). The NP-completeness proof also yields results showing that finding approximation algorithms that make only one-sided error or that approximate closely the minimum number of hidden-layer nodes needed for a network to be powerful enough to correctly classify the training data, is probably hard, in that these problems can be related to other difficult (but not known to be NP-complete) approximation problems.

Our results, like Judd's, are described in terms of "batch"-style learning algorithms that are given all the training examples at once. It is worth noting that training with an "incremental" algorithm that sees the examples one at a time such as back-propagation is at least as hard. Thus the NP-completeness result given here also implies that incremental training algorithms are likely to run slowly.

Our results state that given a network of the classes considered, for any training algorithm there will be some types of training problems such that the algorithm will perform poorly as the problem size increases. The results leave open the possibility that given a training problem that is hard for some network, there might exist a different network and encoding of the input that make training easy.

FIGURE B.2: Top: TRK with recall = 0.1150 and F-Score = 0.0989; Bottom: LSA with recall = 0.3274 and F-Score = 0.2808. Ratio = 0.09.



494

## TRAINING A 3-NODE NEURAL NETWORK IS NP-COMPLETE

Avrim Blum*
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

Ronald L. Rivest†
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

### ABSTRACT

We consider a 2-layer, 3-node, $n$-input neural network whose nodes compute linear threshold functions of their inputs. We show that it is NP-complete to decide whether there exist weights and thresholds for the three nodes of this network so that it will produce output consistent with a given set of training examples. We extend the result to other simple networks. This result suggests that those looking for perfect training algorithms cannot escape inherent computational difficulties just by considering only simple or very regular networks. It also suggests the importance, given a training problem, of finding an appropriate network and input encoding for that problem. It is left as an open problem to extend our result to nodes with non-linear functions such as sigmoids.

### INTRODUCTION

One reason for the recent surge in interest in neural networks is the development of the "back-propagation" algorithm for training neural networks. The ability to train large multi-layer neural networks is essential for utilizing neural networks in practice, and the back-propagation algorithm promises just that. In practice, however, the back-propagation algorithm runs very slowly, and the question naturally arises as to whether there are necessarily intrinsic computational difficulties associated with training neural networks, or whether better training algorithms might exist. This paper provides additional support for the position that training neural networks is intrinsically difficult.

A common method of demonstrating a problem to be intrinsically hard is to show the problem to be "NP-complete". The theory of NP-complete problems is well-understood (Garey and Johnson, 1979), and many infamous problems—such as the traveling salesman problem—are now known to be NP-complete. While NP-completeness does not render a problem totally unapproachable in

*Supported by an NSF graduate fellowship.
†This paper was prepared with support from NSF grant DCR-8607494, ARO Grant DAAL03-86-K-0171, and the Siemens Corporation.



practice, it usually implies that only small instances of the problem can be solved exactly, and that large instances can at best only be solved approximately, even with large amounts of computer time.

The work in this paper is inspired by Judd (Judd, 1987) who shows the following problem to be NP-complete:

> "Given a neural network and a set of training examples, does there exist a set of edge weights for the network so that the network produces the correct output for all the training examples?"

Judd also shows that the problem remains NP-complete even if it is only required a network produce the correct output for two-thirds of the training examples, which implies that even approximately training a neural network is intrinsically difficult in the worst case. Judd produces a class of networks and training examples for those networks such that any training algorithm will perform poorly on some networks and training examples in that class. The results, however, do not specify any particular "hard network"—that is, any single network hard for all algorithms. Also, the networks produced have a number of hidden nodes that grows with the number of inputs, as well as a quite irregular connection pattern.

We extend his result by showing that it is NP-complete to train a specific very simple network, having only two hidden nodes and a regular interconnection pattern. We also present classes of regular 2-layer networks such that for *all* networks in these classes, the training problem is hard in the worst case (in that there exists some hard sets of training examples). The NP-completeness proof also yields results showing that finding approximation algorithms that make only one-sided error or that approximate closely the minimum number of hidden-layer nodes needed for a network to be powerful enough to correctly classify the training data, is probably hard, in that these problems can be related to other difficult (but not known to be NP-complete) approximation problems.

Our results, like Judd's, are described in terms of "batch"-style learning algorithms that are given all the training examples at once. It is worth noting that training with an "incremental" algorithm that sees the examples one at a time such as back-propagation is at least as hard. Thus the NP-completeness result given here also implies that incremental training algorithms are likely to run slowly.

Our results state that given a network of the classes considered, for any training algorithm there will be some types of training problems such that the algorithm will perform poorly as the problem size increases. The results leave open the possibility that given a training problem that is hard for some network, there might exist a different network and encoding of the input that make training easy.

494

TRAINING A 3-NODE NEURAL NETWORK IS NP-COMPLETE

Avrim Blum*
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

Ronald L. Rivest†
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

ABSTRACT

We consider a 2-layer, 3-node, $n$-input neural network whose nodes compute linear threshold functions of their inputs. We show that it is NP-complete to decide whether there exist weights and thresholds for the three nodes of this network so that it will produce output consistent with a given set of training examples. We extend the result to other simple networks. This result suggests that those looking for perfect training algorithms cannot escape inherent computational difficulties just by considering only simple or very regular networks. It also suggests the importance, given a training problem, of finding an appropriate network and input encoding for that problem. It is left as an open problem to extend our result to nodes with non-linear functions such as sigmoids.

INTRODUCTION

One reason for the recent surge in interest in neural networks is the development of the "back-propagation" algorithm for training neural networks. The ability to train large multi-layer neural networks is essential for utilizing neural networks in practice, and the back-propagation algorithm promises just that. In practice, however, the back-propagation algorithm runs very slowly, and the question naturally arises as to whether there are necessarily intrinsic computational difficulties associated with training neural networks, or whether better training algorithms might exist. This paper provides additional support for the position that training neural networks is intrinsically difficult.

A common method of demonstrating a problem to be intrinsically hard is to show the problem to be "NP-complete". The theory of NP-complete problems is well-understood (Garey and Johnson, 1979), and many infamous problems—such as the traveling salesman problem—are now known to be NP-complete. While NP-completeness does not render a problem totally unapproachable in

*Supported by an NSF graduate fellowship.
†This paper was prepared with support from NSF grant DCR-8607494, ARO Grant DAAL03-86-K-0171, and the Siemens Corporation.

Training a 3-Node Neural Network is NP-Complete 495

practice, it usually implies that only small instances of the problem can be solved exactly, and that large instances can at best only be solved approximately, even with large amounts of computer time.

The work in this paper is inspired by Judd (Judd, 1987) who shows the following problem to be NP-complete:

> "Given a neural network and a set of training examples, does there exist a set of edge weights for the network so that the network produces the correct output for all the training examples?"

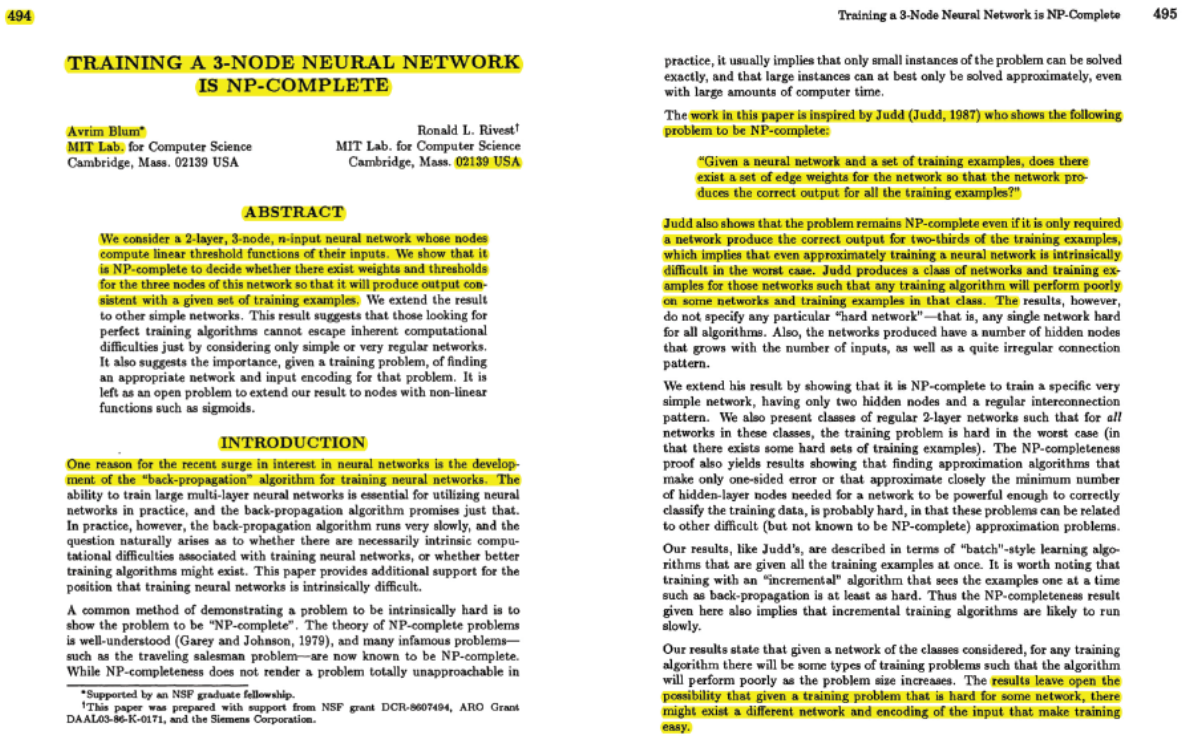
Judd also shows that the problem remains NP-complete even if it is only required a network produce the correct output for two-thirds of the training examples, which implies that even approximately training a neural network is intrinsically difficult in the worst case. Judd produces a class of networks and training examples for those networks such that any training algorithm will perform poorly on some networks and training examples in that class. The results, however, do not specify any particular "hard network"—that is, any single network hard for all algorithms. Also, the networks produced have a number of hidden nodes that grows with the number of inputs, as well as a quite irregular connection pattern.

We extend his result by showing that it is NP-complete to train a specific very simple network, having only two hidden nodes and a regular interconnection pattern. We also present classes of regular 2-layer networks such that for *all* networks in these classes, the training problem is hard in the worst case (in that there exists some hard sets of training examples). The NP-completeness proof also yields results showing that finding approximation algorithms that make only one-sided error or that approximate closely the minimum number of hidden-layer nodes needed for a network to be powerful enough to correctly classify the training data, is probably hard, in that these problems can be related to other difficult (but not known to be NP-complete) approximation problems.

Our results, like Judd's, are described in terms of "batch"-style learning algorithms that are given all the training examples at once. It is worth noting that training with an "incremental" algorithm that sees the examples one at a time such as back-propagation is at least as hard. Thus the NP-completeness result given here also implies that incremental training algorithms are likely to run slowly.

Our results state that given a network of the classes considered, for any training algorithm there will be some types of training problems such that the algorithm will perform poorly as the problem size increases. The results leave open the possibility that given a training problem that is hard for some network, there might exist a different network and encoding of the input that make training easy.

FIGURE B.3: Top: heat map of group 1; Bottom: participant no. 5, ratio = 0.17.



494

## TRAINING A 3-NODE NEURAL NETWORK IS NP-COMPLETE

Avrim Blum*
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

Ronald L. Rivest†
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

### ABSTRACT

We consider a 2-layer, 3-node, $n$-input neural network whose nodes compute linear threshold functions of their inputs. We show that it is NP-complete to decide whether there exist weights and thresholds for the three nodes of this network so that it will produce output consistent with a given set of training examples. We extend the result to other simple networks. This result suggests that those looking for perfect training algorithms cannot escape inherent computational difficulties just by considering only simple or very regular networks. It also suggests the importance, given a training problem, of finding an appropriate network and input encoding for that problem. It is left as an open problem to extend our result to nodes with non-linear functions such as sigmoids.

### INTRODUCTION

One reason for the recent surge in interest in neural networks is the development of the "back-propagation" algorithm for training neural networks. The ability to train large multi-layer neural networks is essential for utilizing neural networks in practice, and the back-propagation algorithm promises just that. In practice, however, the back-propagation algorithm runs very slowly, and the question naturally arises as to whether there are necessarily intrinsic computational difficulties associated with training neural networks, or whether better training algorithms might exist. This paper provides additional support for the position that training neural networks is intrinsically difficult.

A common method of demonstrating a problem to be intrinsically hard is to show the problem to be "NP-complete". The theory of NP-complete problems is well-understood (Garey and Johnson, 1979), and many infamous problems—such as the traveling salesman problem—are now known to be NP-complete. While NP-completeness does not render a problem totally unapproachable in

*Supported by an NSF graduate fellowship.
†This paper was prepared with support from NSF grant DCR-8607494, ARO Grant DAAL03-86-K-0171, and the Siemens Corporation.



practice, it usually implies that only small instances of the problem can be solved exactly, and that large instances can at best only be solved approximately, even with large amounts of computer time.

The work in this paper is inspired by Judd (Judd, 1987) who shows the following problem to be NP-complete:

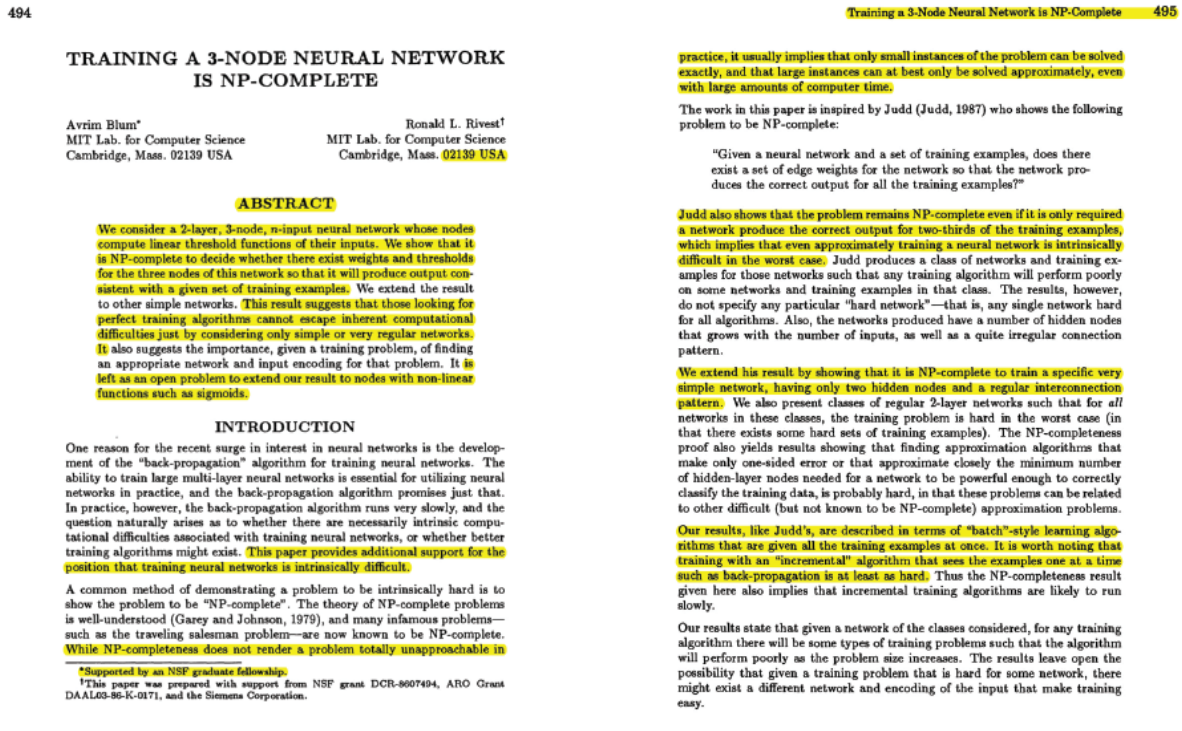
> "Given a neural network and a set of training examples, does there exist a set of edge weights for the network so that the network produces the correct output for all the training examples?"

Judd also shows that the problem remains NP-complete even if it is only required a network produce the correct output for two-thirds of the training examples, which implies that even approximately training a neural network is intrinsically difficult in the worst case. Judd produces a class of networks and training examples for those networks such that any training algorithm will perform poorly on some networks and training examples in that class. The results, however, do not specify any particular "hard network"—that is, any single network hard for all algorithms. Also, the networks produced have a number of hidden nodes that grows with the number of inputs, as well as a quite irregular connection pattern.

We extend his result by showing that it is NP-complete to train a specific very simple network, having only two hidden nodes and a regular interconnection pattern. We also present classes of regular 2-layer networks such that for *all* networks in these classes, the training problem is hard in the worst case (in that there exists some hard sets of training examples). The NP-completeness proof also yields results showing that finding approximation algorithms that make only one-sided error or that approximate closely the minimum number of hidden-layer nodes needed for a network to be powerful enough to correctly classify the training data, is probably hard, in that these problems can be related to other difficult (but not known to be NP-complete) approximation problems.

Our results, like Judd's, are described in terms of "batch"-style learning algorithms that are given all the training examples at once. It is worth noting that training with an "incremental" algorithm that sees the examples one at a time such as back-propagation is at least as hard. Thus the NP-completeness result given here also implies that incremental training algorithms are likely to run slowly.

Our results state that given a network of the classes considered, for any training algorithm there will be some types of training problems such that the algorithm will perform poorly as the problem size increases. The results leave open the possibility that given a training problem that is hard for some network, there might exist a different network and encoding of the input that make training easy.



494

## TRAINING A 3-NODE NEURAL NETWORK IS NP-COMPLETE

Avrim Blum*
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

Ronald L. Rivest†
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

### ABSTRACT

We consider a 2-layer, 3-node, $n$-input neural network whose nodes compute linear threshold functions of their inputs. We show that it is NP-complete to decide whether there exist weights and thresholds for the three nodes of this network so that it will produce output consistent with a given set of training examples. We extend the result to other simple networks. This result suggests that those looking for perfect training algorithms cannot escape inherent computational difficulties just by considering only simple or very regular networks. It also suggests the importance, given a training problem, of finding an appropriate network and input encoding for that problem. It is left as an open problem to extend our result to nodes with non-linear functions such as sigmoids.

### INTRODUCTION

One reason for the recent surge in interest in neural networks is the development of the "back-propagation" algorithm for training neural networks. The ability to train large multi-layer neural networks is essential for utilizing neural networks in practice, and the back-propagation algorithm promises just that. In practice, however, the back-propagation algorithm runs very slowly, and the question naturally arises as to whether there are necessarily intrinsic computational difficulties associated with training neural networks, or whether better training algorithms might exist. This paper provides additional support for the position that training neural networks is intrinsically difficult.

A common method of demonstrating a problem to be intrinsically hard is to show the problem to be "NP-complete". The theory of NP-complete problems is well-understood (Garey and Johnson, 1979), and many infamous problems—such as the traveling salesman problem—are now known to be NP-complete. While NP-completeness does not render a problem totally unapproachable in

*Supported by an NSF graduate fellowship.
†This paper was prepared with support from NSF grant DCR-8607494, ARO Grant DAAL03-86-K-0171, and the Siemens Corporation.



practice, it usually implies that only small instances of the problem can be solved exactly, and that large instances can at best only be solved approximately, even with large amounts of computer time.

The work in this paper is inspired by Judd (Judd, 1987) who shows the following problem to be NP-complete:

> "Given a neural network and a set of training examples, does there exist a set of edge weights for the network so that the network produces the correct output for all the training examples?"

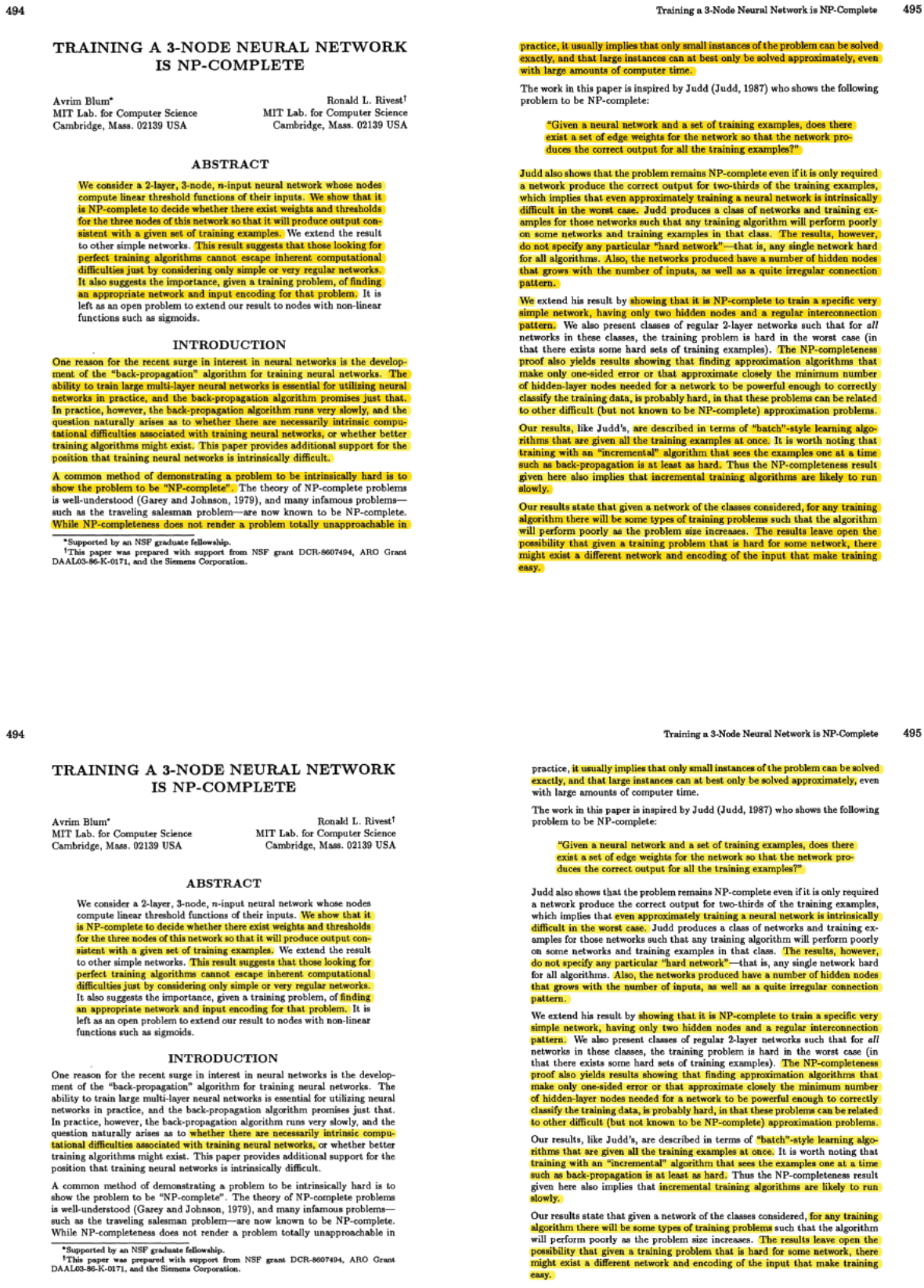
Judd also shows that the problem remains NP-complete even if it is only required a network produce the correct output for two-thirds of the training examples, which implies that even approximately training a neural network is intrinsically difficult in the worst case. Judd produces a class of networks and training examples for those networks such that any training algorithm will perform poorly on some networks and training examples in that class. The results, however, do not specify any particular "hard network"—that is, any single network hard for all algorithms. Also, the networks produced have a number of hidden nodes that grows with the number of inputs, as well as a quite irregular connection pattern.

We extend his result by showing that it is NP-complete to train a specific very simple network, having only two hidden nodes and a regular interconnection pattern. We also present classes of regular 2-layer networks such that for *all* networks in these classes, the training problem is hard in the worst case (in that there exists some hard sets of training examples). The NP-completeness proof also yields results showing that finding approximation algorithms that make only one-sided error or that approximate closely the minimum number of hidden-layer nodes needed for a network to be powerful enough to correctly classify the training data, is probably hard, in that these problems can be related to other difficult (but not known to be NP-complete) approximation problems.

Our results, like Judd's, are described in terms of "batch"-style learning algorithms that are given all the training examples at once. It is worth noting that training with an "incremental" algorithm that sees the examples one at a time such as back-propagation is at least as hard. Thus the NP-completeness result given here also implies that incremental training algorithms are likely to run slowly.

Our results state that given a network of the classes considered, for any training algorithm there will be some types of training problems such that the algorithm will perform poorly as the problem size increases. The results leave open the possibility that given a training problem that is hard for some network, there might exist a different network and encoding of the input that make training easy.

FIGURE B.4: Top: LRK with recall = 0.4757 and F-Score = 0.3064; Bottom: LSA with recall = 0.5320 and F-Score = 0.4298. Ratio = 0.23.



494

## TRAINING A 3-NODE NEURAL NETWORK IS NP-COMPLETE

Avrim Blum*
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

Ronald L. Rivest†
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

### ABSTRACT

We consider a 2-layer, 3-node, $n$-input neural network whose nodes compute linear threshold functions of their inputs. We show that it is NP-complete to decide whether there exist weights and thresholds for the three nodes of this network so that it will produce output consistent with a given set of training examples. We extend the result to other simple networks. This result suggests that those looking for perfect training algorithms cannot escape inherent computational difficulties just by considering only simple or very regular networks. It also suggests the importance, given a training problem, of finding an appropriate network and input encoding for that problem. It is left as an open problem to extend our result to nodes with non-linear functions such as sigmoids.

### INTRODUCTION

One reason for the recent surge in interest in neural networks is the development of the "back-propagation" algorithm for training neural networks. The ability to train large multi-layer neural networks is essential for utilizing neural networks in practice, and the back-propagation algorithm promises just that. In practice, however, the back-propagation algorithm runs very slowly, and the question naturally arises as to whether there are necessarily intrinsic computational difficulties associated with training neural networks, or whether better training algorithms might exist. This paper provides additional support for the position that training neural networks is intrinsically difficult.

A common method of demonstrating a problem to be intrinsically hard is to show the problem to be "NP-complete". The theory of NP-complete problems is well-understood (Garey and Johnson, 1979), and many infamous problems—such as the traveling salesman problem—are now known to be NP-complete. While NP-completeness does not render a problem totally unapproachable in

*Supported by an NSF graduate fellowship.
†This paper was prepared with support from NSF grant DCR-8607494, ARO Grant DAAL03-86-K-0171, and the Siemens Corporation.



practice, it usually implies that only small instances of the problem can be solved exactly, and that large instances can at best only be solved approximately, even with large amounts of computer time.

The work in this paper is inspired by Judd (Judd, 1987) who shows the following problem to be NP-complete:

> "Given a neural network and a set of training examples, does there exist a set of edge weights for the network so that the network produces the correct output for all the training examples?"

Judd also shows that the problem remains NP-complete even if it is only required a network produce the correct output for two-thirds of the training examples, which implies that even approximately training a neural network is intrinsically difficult in the worst case. Judd produces a class of networks and training examples for those networks such that any training algorithm will perform poorly on some networks and training examples in that class. The results, however, do not specify any particular "hard network"—that is, any single network hard for all algorithms. Also, the networks produced have a number of hidden nodes that grows with the number of inputs, as well as a quite irregular connection pattern.

We extend his result by showing that it is NP-complete to train a specific very simple network, having only two hidden nodes and a regular interconnection pattern. We also present classes of regular 2-layer networks such that for *all* networks in these classes, the training problem is hard in the worst case (in that there exists some hard sets of training examples). The NP-completeness proof also yields results showing that finding approximation algorithms that make only one-sided error or that approximate closely the minimum number of hidden-layer nodes needed for a network to be powerful enough to correctly classify the training data, is probably hard, in that these problems can be related to other difficult (but not known to be NP-complete) approximation problems.

Our results, like Judd's, are described in terms of "batch"-style learning algorithms that are given all the training examples at once. It is worth noting that training with an "incremental" algorithm that sees the examples one at a time such as back-propagation is at least as hard. Thus the NP-completeness result given here also implies that incremental training algorithms are likely to run slowly.

Our results state that given a network of the classes considered, for any training algorithm there will be some types of training problems such that the algorithm will perform poorly as the problem size increases. The results leave open the possibility that given a training problem that is hard for some network, there might exist a different network and encoding of the input that make training easy.

494

TRAINING A 3-NODE NEURAL NETWORK IS NP-COMPLETE

Avrim Blum*
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

Ronald L. Rivest†
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

ABSTRACT

We consider a 2-layer, 3-node, $n$-input neural network whose nodes compute linear threshold functions of their inputs. We show that it is NP-complete to decide whether there exist weights and thresholds for the three nodes of this network so that it will produce output consistent with a given set of training examples. We extend the result to other simple networks. This result suggests that those looking for perfect training algorithms cannot escape inherent computational difficulties just by considering only simple or very regular networks. It also suggests the importance, given a training problem, of finding an appropriate network and input encoding for that problem. It is left as an open problem to extend our result to nodes with non-linear functions such as sigmoids.

INTRODUCTION

One reason for the recent surge in interest in neural networks is the development of the "back-propagation" algorithm for training neural networks. The ability to train large multi-layer neural networks is essential for utilizing neural networks in practice, and the back-propagation algorithm promises just that. In practice, however, the back-propagation algorithm runs very slowly, and the question naturally arises as to whether there are necessarily intrinsic computational difficulties associated with training neural networks, or whether better training algorithms might exist. This paper provides additional support for the position that training neural networks is intrinsically difficult.

A common method of demonstrating a problem to be intrinsically hard is to show the problem to be "NP-complete". The theory of NP-complete problems is well-understood (Garey and Johnson, 1979), and many infamous problems—such as the traveling salesman problem—are now known to be NP-complete. While NP-completeness does not render a problem totally unapproachable in

*Supported by an NSF graduate fellowship.
†This paper was prepared with support from NSF grant DCR-8607494, ARO Grant DAAL03-86-K-0171, and the Siemens Corporation.

Training a 3-Node Neural Network is NP-Complete 495

practice, it usually implies that only small instances of the problem can be solved exactly, and that large instances can at best only be solved approximately, even with large amounts of computer time.

The work in this paper is inspired by Judd (Judd, 1987) who shows the following problem to be NP-complete:

> "Given a neural network and a set of training examples, does there exist a set of edge weights for the network so that the network produces the correct output for all the training examples?"

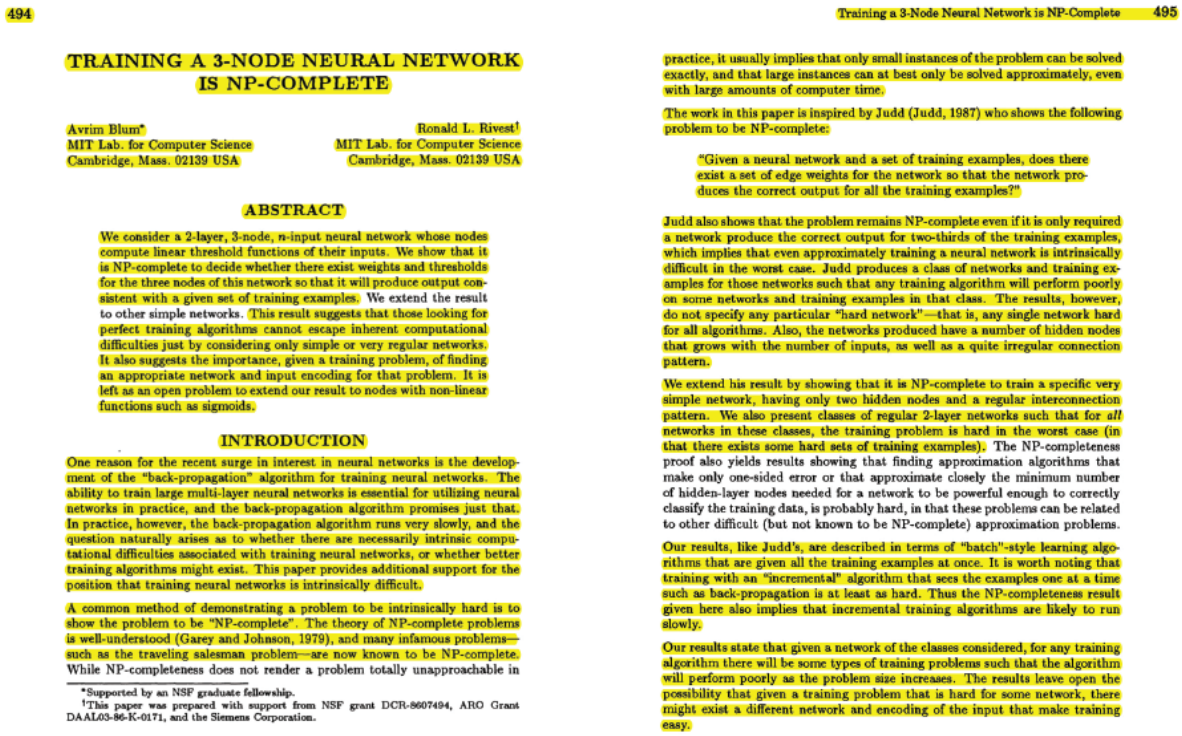
Judd also shows that the problem remains NP-complete even if it is only required a network produce the correct output for two-thirds of the training examples, which implies that even approximately training a neural network is intrinsically difficult in the worst case. Judd produces a class of networks and training examples for those networks such that any training algorithm will perform poorly on some networks and training examples in that class. The results, however, do not specify any particular "hard network"—that is, any single network hard for all algorithms. Also, the networks produced have a number of hidden nodes that grows with the number of inputs, as well as a quite irregular connection pattern.

We extend his result by showing that it is NP-complete to train a specific very simple network, having only two hidden nodes and a regular interconnection pattern. We also present classes of regular 2-layer networks such that for *all* networks in these classes, the training problem is hard in the worst case (in that there exists some hard sets of training examples). The NP-completeness proof also yields results showing that finding approximation algorithms that make only one-sided error or that approximate closely the minimum number of hidden-layer nodes needed for a network to be powerful enough to correctly classify the training data, is probably hard, in that these problems can be related to other difficult (but not known to be NP-complete) approximation problems.

Our results, like Judd's, are described in terms of "batch"-style learning algorithms that are given all the training examples at once. It is worth noting that training with an "incremental" algorithm that sees the examples one at a time such as back-propagation is at least as hard. Thus the NP-completeness result given here also implies that incremental training algorithms are likely to run slowly.

Our results state that given a network of the classes considered, for any training algorithm there will be some types of training problems such that the algorithm will perform poorly as the problem size increases. The results leave open the possibility that given a training problem that is hard for some network, there might exist a different network and encoding of the input that make training easy.

FIGURE B.5: Top: heat map of group 2; Bottom: participant no. 7, ratio = 0.37.



494 Training a 3-Node Neural Network is NP-Complete 495

# TRAINING A 3-NODE NEURAL NETWORK IS NP-COMPLETE

Avrim Blum*
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

Ronald L. Rivest†
MIT Lab. for Computer Science
Cambridge, Mass. 02139 USA

## ABSTRACT

We consider a 2-layer, 3-node, $n$-input neural network whose nodes compute linear threshold functions of their inputs. We show that it is NP-complete to decide whether there exist weights and thresholds for the three nodes of this network so that it will produce output consistent with a given set of training examples. We extend the result to other simple networks. This result suggests that those looking for perfect training algorithms cannot escape inherent computational difficulties just by considering only simple or very regular networks. It also suggests the importance, given a training problem, of finding an appropriate network and input encoding for that problem. It is left as an open problem to extend our result to nodes with non-linear functions such as sigmoids.

## INTRODUCTION

One reason for the recent surge in interest in neural networks is the development of the "back-propagation" algorithm for training neural networks. The ability to train large multi-layer neural networks is essential for utilizing neural networks in practice, and the back-propagation algorithm promises just that. In practice, however, the back-propagation algorithm runs very slowly, and the question naturally arises as to whether there are necessarily intrinsic computational difficulties associated with training neural networks, or whether better training algorithms might exist. This paper provides additional support for the position that training neural networks is intrinsically difficult.

A common method of demonstrating a problem to be intrinsically hard is to show the problem to be "NP-complete". The theory of NP-complete problems is well-understood (Garey and Johnson, 1979), and many infamous problems—such as the traveling salesman problem—are now known to be NP-complete. While NP-completeness does not render a problem totally unapproachable in practice, it usually implies that only small instances of the problem can be solved exactly, and that large instances can at best only be solved approximately, even with large amounts of computer time.

The work in this paper is inspired by Judd (Judd, 1987) who shows the following problem to be NP-complete:

> "Given a neural network and a set of training examples, does there exist a set of edge weights for the network so that the network produces the correct output for all the training examples?"

Judd also shows that the problem remains NP-complete even if it is only required a network produce the correct output for two-thirds of the training examples, which implies that even approximately training a neural network is intrinsically difficult in the worst case. Judd produces a class of networks and training examples for those networks such that any training algorithm will perform poorly on some networks and training examples in that class. The results, however, do not specify any particular "hard network"—that is, any single network hard for all algorithms. Also, the networks produced have a number of hidden nodes that grows with the number of inputs, as well as a quite irregular connection pattern.

We extend his result by showing that it is NP-complete to train a specific very simple network, having only two hidden nodes and a regular interconnection pattern. We also present classes of regular 2-layer networks such that for *all* networks in these classes, the training problem is hard in the worst case (in that there exists some hard sets of training examples). The NP-completeness proof also yields results showing that finding approximation algorithms that make only one-sided error or that approximate closely the minimum number of hidden-layer nodes needed for a network to be powerful enough to correctly classify the training data, is probably hard, in that these problems can be related to other difficult (but not known to be NP-complete) approximation problems.

Our results, like Judd's, are described in terms of "batch"-style learning algorithms that are given all the training examples at once. It is worth noting that training with an "incremental" algorithm that sees the examples one at a time such as back-propagation is at least as hard. Thus the NP-completeness result given here also implies that incremental training algorithms are likely to run slowly.

Our results state that given a network of the classes considered, for any training algorithm there will be some types of training problems such that the algorithm will perform poorly as the problem size increases. The results leave open the possibility that given a training problem that is hard for some network, there might exist a different network and encoding of the input that make training easy.

*Supported by an NSF graduate fellowship.
†This paper was prepared with support from NSF grant DCR-8607494, ARO Grant DAAL03-86-K-0171, and the Siemens Corporation.

494 Training a 3-Node Neural Network is NP-Complete 495

TRAINING A 3-NODE NEURAL NETWORK IS NP-COMPLETE

Avrim Blum* MIT Lab. for Computer Science Cambridge, Mass. 02139 USA
Ronald L. Rivest† MIT Lab. for Computer Science Cambridge, Mass. 02139 USA

ABSTRACT

We consider a 2-layer, 3-node, n-input neural network whose nodes compute linear threshold functions of their inputs. We show that it is NP-complete to decide whether there exist weights and thresholds for the three nodes of this network so that it will produce output consistent with a given set of training examples. We extend the result to other simple networks. This result suggests that those looking for perfect training algorithms cannot escape inherent computational difficulties just by considering only simple or very regular networks. It also suggests the importance, given a training problem, of finding an appropriate network and input encoding for that problem. It is left as an open problem to extend our result to nodes with non-linear functions such as sigmoids.

INTRODUCTION

One reason for the recent surge in interest in neural networks is the development of the "back-propagation" algorithm for training neural networks. The ability to train large multi-layer neural networks is essential for utilizing neural networks in practice, and the back-propagation algorithm promises just that. In practice, however, the back-propagation algorithm runs very slowly, and the question naturally arises as to whether there are necessarily intrinsic computational difficulties associated with training neural networks, or whether better training algorithms might exist. This paper provides additional support for the position that training neural networks is intrinsically difficult.

A common method of demonstrating a problem to be intrinsically hard is to show the problem to be "NP-complete". The theory of NP-complete problems is well-understood (Garey and Johnson, 1979), and many infamous problems—such as the traveling salesman problem—are now known to be NP-complete. While NP-completeness does not render a problem totally unapproachable in practice, it usually implies that only small instances of the problem can be solved exactly, and that large instances can at best only be solved approximately, even with large amounts of computer time.

The work in this paper is inspired by Judd (Judd, 1987) who shows the following problem to be NP-complete:

> "Given a neural network and a set of training examples, does there exist a set of edge weights for the network so that the network produces the correct output for all the training examples?"

Judd also shows that the problem remains NP-complete even if it is only required a network produce the correct output for two-thirds of the training examples, which implies that even approximately training a neural network is intrinsically difficult in the worst case. Judd produces a class of networks and training examples for those networks such that any training algorithm will perform poorly on some networks and training examples in that class. The results, however, do not specify any particular "hard network"—that is, any single network hard for all algorithms. Also, the networks produced have a number of hidden nodes that grows with the number of inputs, as well as a quite irregular connection pattern.

We extend his result by showing that it is NP-complete to train a specific very simple network, having only two hidden nodes and a regular interconnection pattern. We also present classes of regular 2-layer networks such that for all networks in these classes, the training problem is hard in the worst case (in that there exists some hard sets of training examples). The NP-completeness proof also yields results showing that finding approximation algorithms that make only one-sided error or that approximate closely the minimum number of hidden-layer nodes needed for a network to be powerful enough to correctly classify the training data, is probably hard, in that these problems can be related to other difficult (but not known to be NP-complete) approximation problems.

Our results, like Judd's, are described in terms of "batch"-style learning algorithms that are given all the training examples at once. It is worth noting that training with an "incremental" algorithm that sees the examples one at a time such as back-propagation is at least as hard. Thus the NP-completeness result given here also implies that incremental training algorithms are likely to run slowly.

Our results state that given a network of the classes considered, for any training algorithm there will be some types of training problems such that the algorithm will perform poorly as the problem size increases. The results leave open the possibility that given a training problem that is hard for some network, there might exist a different network and encoding of the input that make training easy.

*Supported by an NSF graduate fellowship.
†This paper was prepared with support from NSF grant DCR-8607494, ARO Grant DAAL03-86-K-0171, and the Siemens Corporation.

FIGURE B.6: Top: TTS with recall = 0.5441 and F-Score = 0.5097; Bottom: TRK with recall = 0.8399 and F-Score = 0.6506. Ratio = 0.43.

# Bibliography


Barrios, Federico et al. (2016). "Variations of the Similarity Function of TextRank for Automated Summarization". In: *arXiv:1602.03606 [cs]*. arXiv: 1602.03606. URL: http://arxiv.org/abs/1602.03606.

Barzilay, Regina and Michael Elhadad (1999). "Using lexical chains for text summarization". In: *Advances in automatic text summarization*, pp. 111–121. URL: https://books.google.nl/books?hl=en&lr=&id=YtUZQaKDmzEC&oi=fnd&pg=PA111&dq=barzilay+lexical+chains&ots=ZpqAyoCVbB&sig=8OXOHd8DtIdfLgSF1Vx3TYw3gXQ.

Bell, Kenneth E. and John E. Limber (2009). "Reading skill, textbook marking, and course performance". In: *Literacy research and instruction* 49.1, pp. 56–67.

Blum, Avrim and Ronald L. Rivest (1989). "Training a 3-node neural network is NP-complete". In: *Advances in neural information processing systems*, pp. 494–501.

Cha, Sung-Hyuk (2007). "Comprehensive survey on distance/similarity measures between probability density functions". In: *City* 1.2, p. 1.

Chi, Ed H. et al. (2005). "ScentHighlights: highlighting conceptually-related sentences during reading". In: *Proceedings of the 10th international conference on Intelligent user interfaces*. ACM, pp. 272–274. URL: http://dl.acm.org/citation.cfm?id=1040895.

Das, Dipanjan and André FT Martins (2007). "A survey on automatic text summarization". In: *Literature Survey for the Language and Statistics II course at CMU* 4, pp. 192–195.

Economist, The (2010). "Data, data everywhere". In: *The Economist*. ISSN: 0013-0613. URL: http://www.economist.com/node/15557443.

Edmundson, Harold P. (1969). "New methods in automatic extracting". In: *Journal of the ACM (JACM)* 16.2, pp. 264–285. URL: http://dl.acm.org/citation.cfm?id=321519.

Englert, Carol Sue et al. (2009). "The learning-to-learn strategies of adolescent students with disabilities: Highlighting, note taking, planning, and writing expository texts". In: *Assessment for Effective Intervention* 34.3, pp. 147–161.

Erkan, Günes and Dragomir R. Radev (2004). "Lexrank: Graph-based lexical centrality as salience in text summarization". In: *Journal of Artificial Intelligence Research* 22, pp. 457–479.

Fowler, Robert L. and Anne S. Barker (1974). "Effectiveness of highlighting for retention of text material." In: *Journal of Applied Psychology* 59.3, p. 358.

Ganesan, Kavita, ChengXiang Zhai, and Jiawei Han (2010). "Opinosis: a graph-based approach to abstractive summarization of highly redundant opinions". In: *Proceedings of the 23rd International Conference on Computational Linguistics*. bibtex: ganesan2010opinosis bibtex[organization=Association for Computational Linguistics], pp. 340–348.

Goldstein, Jade et al. (1999). "Summarizing text documents: sentence selection and evaluation metrics". In: *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, pp. 121–128. URL: http://dl.acm.org/citation.cfm?id=312665.

Gong, Yihong and Xin Liu (2001). "Generic text summarization using relevance measure and latent semantic analysis". In: *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, pp. 19–25.

Gunawan, D et al. (2017). "Automatic Text Summarization for Indonesian Language Using TextTeaser". In: *IOP Conference Series: Materials Science and Engineering* 190, p. 012048. ISSN: 1757-8981, 1757-899X. DOI: 10.1088/1757-899X/190/1/012048. URL: http://stacks.iop.org/1757-899X/190/i=1/a=012048?key=crossref.f504fbc26525044cd72cdef88254e9dc.

Gupta, Vishal and Gurpreet Singh Lehal (2010). "A survey of text summarization extractive techniques". In: *Journal of emerging technologies in web intelligence* 2.3, pp. 258–268.

Hahn, Udo and Inderjeet Mani (2000). "The challenges of automatic summarization". In: *Computer* 33.11, pp. 29–36.

Howe, M. J. A. and Linda Singer (1975). "PRESENTATION VARIABLES AND STUDENTS'ACTIVITIES IN MEANINGFUL LEARNING". In: *British Journal of Educational Psychology* 45.1, pp. 52–61.

Jagadeesh, J., Prasad Pingali, and Vasudeva Varma (2005). "Sentence extraction based single document summarization". In: *International Institute of Information Technology, Hyderabad, India* 5.

Jones, Raymond (2012). *ReadingQuest Strategies | Selective Underlining*. URL: http://www.readingquest.org/strat/underline.html.

Leutner, Detlev, Claudia Leopold, and Viola den Elzen-Rump (2007). "Self-regulated learning with a text-highlighting strategy". In: *Zeitschrift für Psychologie/Journal of Psychology* 215.3, pp. 174–182.

Lin, Chin-Yew (2004). "Rouge: A package for automatic evaluation of summaries". In: *Text Summarization Branches Out*.

Luhn, Hans Peter (1958). "The automatic creation of literature abstracts". In: *IBM Journal of research and development* 2.2, pp. 159–165.

Manning, Christopher D., Hinrich Schütze, and others (1999). *Foundations of statistical natural language processing*. Vol. 999. MIT Press. URL: http://www.mitpressjournals.org/doi/pdf/10.1162/coli.2000.26.2.277.

Mathur, Paranay, Aman Gill, and Aayush Yadav (2017). *Text Summarization in Python: Extractive vs. Abstractive techniques revisited | RaRe Technologies*. URL: https://rare-technologies.com/text-summarization-in-python-extractive-vs-abstractive-techniques-revisited/#text_summarization_in_python.

Mihalcea, Rada and Paul Tarau (2004). "Textrank: Bringing order into text". In: *Proceedings of the 2004 conference on empirical methods in natural language processing*.

Oghina, Andrei et al. (2012). "Predicting imdb movie ratings using social media". In: *European Conference on Information Retrieval*. Springer, pp. 503–507.

Page, Lawrence et al. (1999). *The PageRank citation ranking: Bringing order to the web.* Tech. rep. Stanford InfoLab.

Papineni, Kishore et al. (2002). "BLEU: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, pp. 311–318.

Peters, Brad (2012). *The Age of Big Data*. URL: https://www.forbes.com/sites/bradpeters/2012/07/12/the-age-of-big-data/.

Radev, Dragomir R., Eduard Hovy, and Kathleen McKeown (2002). "Introduction to the special issue on summarization". In: *Computational linguistics* 28.4, pp. 399–408.

Saggion, Horacio and Thierry Poibeau (2013). "Automatic text summarization: Past, present and future". In: *Multi-source, multilingual information extraction and summarization*. Springer, pp. 3–21. URL: http://link.springer.com/chapter/10.1007/978-3-642-28569-1_1.

Sharpe, William N. et al. (1997). "Measurements of Young's modulus, Poisson's ratio, and tensile strength of polysilicon". In: *Micro Electro Mechanical Systems, 1997. MEMS'97, Proceedings, IEEE., Tenth Annual International Workshop on*. IEEE, pp. 424–429.

Sillburn, Jenny (2017). *Study Skills | Effective reading strategies*. URL: http://learnline.cdu.edu.au/studyskills/studyskills/reading.html.

Silvers, Vicki L. and David S. Kreiner (1997). "The effects of pre-existing inappropriate highlighting on reading comprehension". In: *Literacy Research and Instruction* 36.3, pp. 217–223.

SINTEF (2013). *Big Data, for better or worse: 90% of world's data generated over last two years*. URL: https://www.sciencedaily.com/releases/2013/05/130522085217.htm.

Sparck Jones, Karen (1972). "A statistical interpretation of term specificity and its application in retrieval". In: *Journal of documentation* 28.1, pp. 11–21.

Steinberger, Josef and Karel Jezek (2004). "Using latent semantic analysis in text summarization and summary evaluation". In: *Proc. ISIM'04*, pp. 93–100.

Stordahl, Kalmer E. and Clifford M. Christensen (1956). "The effect of study techniques on comprehension and retention". In: *The Journal of Educational Research* 49.8, pp. 561–570.

Strobelt, Hendrik et al. (2016). "Guidelines for effective usage of text highlighting techniques". In: *IEEE transactions on visualization and computer graphics* 22.1, pp. 489–498. URL: http://ieeexplore.ieee.org/abstract/document/7192718/.

Watanabe, Shigeru, Junko Sakamoto, and Masumi Wakita (1995). "PIGEONS'DISCRIMINATION OF PAINTINGS BY MONET AND PICASSO". In: *Journal of the experimental analysis of behavior* 63.2, pp. 165–174.

Wiemer-Hastings, Peter (2004). "Latent Semantic Analysis". In:

Yue, Carole L. et al. (2015). "Highlighting and its relation to distributed study and students' metacognitive beliefs". In: *Educational Psychology Review* 27.1, pp. 69–78.