Aalborg University

Master Thesis Report

Recurrent Recognition Networks for Approximation of BN2O Inference Dedicated to the Medical IntMed System

Written by:
Jonas T. Henriksen

Supervised by:
Thomas D. Nielsen

Group mi105
Department of Computer Science

Spring semester 2021

June 11th 2021

AALBORG UNIVERSITY
STUDENT REPORT

Aalborg University
Department of Computer Science
http://www.cs.aau.dk

Title:
Recurrent Recognition Networks for Approximation of BN2O Inference Dedicated to the Medical IntMed System

Theme:
Inference, BN2O Networks,
CDSS, Neural Networks,
LSTM, Multi-Label Classification,
Real-Time Medical Diagnostics

Project period:
Spring semester 2021

Project group:
Group mi105

Participant:
Jonas Tvede Henriksen

Supervisor:
Thomas Dyhre Nielsen

Number of pages: 69

Date of completion:
June 11th 2021

Abstract

This report is an investigation of an alternate method for inference in a large graphical BN2O model used by the AI system IntMed. The system is used for clinical decision support in real-time medical consultations and is made by Ambolt ApS who have supplied the data necessary to conduct the research. The current inference calculation algorithm of IntMed is called 'Quickscore' and calculates exact probability estimations, but struggles to uphold the real-time constraint. We propose the usage of a special type of neural networks known as Recognition Networks for inference approximation. Specifically, we propose a Recurrent Recognition Network capable of analysing the temporal unveiling of symptoms through patient questioning that happens during consultations. We show how this recurrent network can be trained using forward sampling from a BN2O. To demonstrate the network's potential we compare it to Quickscore in various consultation scenarios, in terms of posterior estimation and importance order of diseases. These results show that a recurrent neural network is able to mimic the results of exact inference better than the sequential counterpart, but needs attention in terms of scaling and calibration.

Preface

This report is a master's thesis build upon work from two semesters of Aalborg University's international computer science master's programme. This report is thereby a continuation of a previous report [1] made by me and my erstwhile colleague. It should therefore be noted that part of the previous report's content has been reused in some of the sections of this report. This will be noted in the beginning of the chapters containing sections that reuses the contents.

I would like to thank Ambolt ApS for the opportunity to work on this project, and MSO Thomas Dyhre Nielsen for great supervision through the entire project and the Machine Intelligence related courses. I also apologize for having delayed his lunch with my questions on several occasions.

Summary

This report is a master's thesis that investigates the effect of using a recurrent neural network as a method for inference in a BN2O network consisting of medical data. The investigation takes its outset in the AI system IntMed. This system is developed by the company Ambolt ApS and is made to provide decisive support for healthcare professionals during patient consultations.

The system works by calculating the posterior of all possible diseases given the symptoms which the patients reveal through questioning. The calculation time for providing exact posteriors is however not fit for a consultation scenario where the system should provide its diagnostic suggestions in real time.

This problem stems from the probabilistic model which the system bases its calculations upon. The model is a large Bayesian network with a vast number of diseases and symptoms, where the diseases have connections pointing to a large number of symptoms, and the symptoms have connections pointing to them from a large number of diseases.

As an alternative to exact calculation of the posteriors, we propose the usage of a neural network to instead approximate them. The neural network type used for this purpose is called a Recognition Network which we propose to implement as a Recurrent Recognition Network. This enables it to use the question sequences given by patients for diagnosis prediction.

Before diving into the implementation of the recurrent version, we show how the basic idea of recognition networks have great potential for approximating inference of BN2O networks in general. We show how sampling of the large BN2O network can be used to produce simulations of patient cases for the recognition networks to train upon.

For the recurrent recognition network, we show how these samples are made to resemble a medical consultation, specifically in the context of modelling them as question sequences. We propose an architecture for the recurrent network based on ideas from literature and experimentation conducted throughout the project.

Finally, we conduct experiments where we compare the recurrent networks to IntMed's currently used exact inference algorithm known as Quickscore. We do this by analyzing the posteriors, the ordering, the KL-divergence, and the top-k similarity between the diseases produced by Quickscore and the recurrent networks, in various simulated consultation scenarios. The results of the experiments show that the recurrent network is able to produce outputs that closely resemble the estimations made by exact inference. It does however currently require more work in terms of scaling the concept to work for very large BN2O networks.

The data for the project as well as the overall project suggestion were provided by Ambolt ApS. The data are created through collaboration with healthcare professionals who created it based on disease prevalence and expert knowledge.

Table of Contents

1 Introduction

This entire chapter reuses content from [1].

In the field of machine intelligence one of the most common and fundamental tasks is prediction. This is the main aspect that makes an artificial intelligence (AI) seem intelligent compared to other kinds of software. The task of prediction comes in many flavors, such as regression, forecasting, or cause-effect explanation. The latter has been researched extensively to make sophisticated monitoring systems [2, 3] all with the common denominator of serving as diagnostic tools.

One of the most common cause-effect prediction tasks where such diagnostic tools are used is disease prediction. As a result of this, there are many different machine intelligence related techniques that have been applied in this area. This has led to the invention of an entire class of systems called Clinical Decision Support Systems (CDSS). These systems serve many purposes within the medical domain [4, 5], ranging from simple recommendations and/or notifications, to provision of complete diagnoses, all of which is usually based on patient data.

One such system is IntMed, a CDSS developed by the company Ambolt ApS [6]. This system is designed to provide real-time diagnostic assistance through the calculation of posterior disease distributions based on patient symptoms. It derives these calculations by using an extensive database of expert medical knowledge and patient data. This knowledge is organized in a probabilistic model that takes form as a large bi-partite Bayesian network, enabling it to include probabilistic knowledge about a very large variety of diseases.

In this regard the IntMed system is very different from most other CDSS. Most diagnostic systems presented in various papers [7–10] seem to focus on predicting either the severity or simply just the presence of very few diseases. Furthermore, these predictions are usually based upon very specific and limited attributes. The only other prominent example of a system with the same probabilistic model setup and data quantity as IntMed, is the QMR-DT system [11, 12]. This system contains prevalence knowledge of diseases and symptoms, where both of these entities amount in quadruple digits.

Even though the inclusion of many diseases and symptoms increases IntMed's versatility compared to systems designed for more simple tasks, it also creates a problem that has to be dealt with. Many of the symptoms are individually conditioned on a vast number of diseases. Inference of disease posteriors based these symptoms requires techniques that goes beyond both standard Bayesian network inference and network structure. However, even with these techniques the posterior calculation may not be carried out in time suitable for a real-time system.

An approach that does allow for real-time usage is to use a recognition network, a special type of neural network which can bypass the direct utilization of the Bayesian network for inference, and instead give an approximate prediction. This approach has great potential, but could use some tweaking to become a more robust artificially intelligent prediction tool for medical settings. A particularly interesting extension to the recognition network is the inclusion of the temporal symptom unveiling which is inherent in all consultations.

In this thesis we investigate appliance of state-of-the-art neural network techniques for enhancement of a recognition network. The goal is to make an implementation that allows for consideration of the temporal unveiling of symptoms within a medical consultation. In particular we investigate the representation of symptom evidence and usage of a recurrent neural network structure.

The rest of this thesis is structured as follows: In the remaining part of this chapter, we specify the disease and symptom data made available for the project by Ambolt ApS. In Section 2 we state the primary problem statement to be investigated throughout the report. In Section 3 we cover the necessary preliminaries, namely Bayesian networks, BN2O networks, the Quickscore algorithm, and neural networks. In Section 4 we examine the concept of the originally proposed recognition network idea which we subsequently make slightly altered implementations of to showcase the recognition network potential. In Section 5 we go through the entire design and implementation process of making the recognition network recurrent. In Section 6 we analyse and explain the results of experiments conducted with the recurrent recognition network, and we contrast different implementation varieties. Finally, in Section 7 we conclude upon the primary problem using the experimental results and observations made throughout the thesis.

1.1 Available IntMed Material

Ambolt has granted access to a database containing information of 514 diseases, 347 symptoms, and 2745 relations between them. The data contains the probabilities of the diseases, including information about their prevalence, statistical information based on disease incidents, and the rarity that for each disease specifies whether it is common or rare. Most of this information comes from various healthcare professionals. The disease data have some missing entries, so a priority system is in place to select which metric to base the probability on, if the desired metric is not available. The priority order is prevalence, incidents, and finally rarity.

The data set also includes demographic information about gender, age, ethnicity, and country, as well as specific characteristics for each symptom meant for guiding the formulation of questions. For the symptoms there is also a classification of the anatomical region a given symptom may appear in, specified as either 'general' or 'specific'. 'General' refers to a body part such as the torso, and 'specific' refers to the precise spot where the symptom manifests such as in one of the shoulders. Finally, there is also relations between diseases that describe which diseases that influences others.

The demographic, anatomical, and disease relational information will not be used for this project.

Aside from the data provided by Ambolt, MSO Thomas Dyhre Nielsen has provided an implementation of IntMed that uses these data to showcase how the system normally works.

2 Problem Formulation

The IntMed system is built upon expert knowledge of the medical domain which is gathered in a large Bayesian network. This network contains high quality data that may be used for developing an AI capable of disease prediction. However, due to the challenging structure of the data, the posterior calculation time of the AI is prone to exceed levels where it stops being useful for real time usage. Approximating the probabilities using a recognition network is a possible alternative. In addition, it would be interesting to examine the results of tailoring the network to the medical consultation scenario in which it should operate. A prominent part of this scenario is the temporal symptom unveiling which may become available through a recurrent neural network structure.

2.1 Problem Statement

Given the introductory considerations, the following is the problem statement to be investigated for the rest of this thesis:

"Is it possible to develop a Recurrent Recognition Network capable of using a sequence of unveiled symptoms for approximating the inference of a large Bayesian network such that the inference accuracy is on par with exact alternatives, if recurrent neural network techniques from literature and aspects of the domain where the system is designed to operate, are used as key parts of its development?"

The following questions have been made to help branch out the research:

- How are posteriors normally and currently being calculated for the type of Bayesian network IntMed uses, and why would these ways aspire the usage of a neural network to substitute them?

- What is the fundamental idea behind Recognition Networks, and how can it be shown that they can provide great approximations of the inference in the type of Bayesian network that IntMed uses?

- What state-of-the-art neural network structure is advisable for the creation of a Recurrent Recognition Network, and how can it be explicitly designed for medical patient questioning?

- How should the samples for the training of the recurrent neural network be semantically defined given the purpose of embodying aspects of a standard medical consultation?

- Given its usage intention, how should the developed recurrent neural network be tested, evaluated, and experimented with?

3 Preliminaries

All sections within this chapter reuses content from [1].

In this chapter all of the necessary preliminary information needed for the rest of the report will be covered. The main topics are Bayesian networks, exact and approximate inference, and neural networks.

3.1 Bayesian Networks

A Bayesian Network (BN) is a probabilistic model that takes form as a directed acyclic graph describing the conditional dependencies of variables within a domain. A BN can be described as a tuple [2] such that BN $= (\boldsymbol{V}, \boldsymbol{E})$ where $\boldsymbol{V}$ is the set of vertex nodes, and $\boldsymbol{E}$ is the set of edges. Each $v \in \boldsymbol{V}$ denotes a variable of the domain with a set of states it can take within the domain. Variables that have been observed to be in a specific state is said to have received *evidence*. Along with a state set, each variable has a Conditional Probability Table (CPT) where the conditional dependencies between a given variable and the remaining variables of the network are modeled by edges such that $\{\forall v \in \boldsymbol{V} | P(v|\pi_v)\}$ where π_v denote the set of vertex nodes with edges pointing to v. The node set π_v is said to be the *parents* of v, and it follows that v is their *child* node. A highly useful property of this probabilistic model type is that the joint probability distribution of the different variable state configurations can be calculated using the chain rule:

$$P(\boldsymbol{V}) = \prod_{v \in \boldsymbol{V}} P(v|\pi_v) \tag{3.1}$$

3.2 The Noisy-OR Concept

BN modelling of symptoms that are conditioned on a large number of diseases is a non-trivial task. In the medical domain the diseases are seen as causes and symptoms are seen as effects. This means that the disease parent set for each symptom child may have a very large cardinality which when modelled as a BN forms many converging connections. Let s_i be a symptom from the full set of symptoms $\boldsymbol{s} = \{s_1, \ldots, s_I\}$, and let $\boldsymbol{d} = \{d_1, \ldots, d_J\}$ be the full set of diseases, then d_{s_i} denotes the parent set of s_i where $d_{s_i} \subseteq \boldsymbol{d}$. With plain Bayesian inference the CPT to be specified for s_i will have $O(2^{|d_{s_i}|})$ entries and is thereby time- and space-wise highly intractable.

To enable modelling of this medical domain, the BN is enhanced with the Noisy-OR concept [13, 14] which assumes that each effect variable is conditioned on each of its cause variables individually.

With this tweak the number of probabilities to estimate becomes linear in the number of causes rather than exponential. For the disease-symptom domain the CPTs to estimate instead becomes $\{\forall s_i \in \boldsymbol{s}, \forall d_j \in d_{s_i} | P(s_i|d_j)\}$ such that each s_i will have a number of CPTs equal to $|d_{s_i}|$.

Both symptoms and diseases are binary variables, with both having the state set $\{+, -\}$ denoting positive and negative presence respectively. If a symptom is observed to be positive, denoted by s_i^+, it is assumed for that for each $d_j \in d_{s_i}$, d_j has caused s_i^+ with probability 1, unless a prevention factor has apprehended it. This prevention factor is the probability of the disease not having caused the symptom. With this setup, calculation of the posterior with respect to some s_i^+ is given by:

$$P(s_i^+|d_{si}) = 1 - \prod_{j=1}^{|d_{s_i}|} P(s_i^-|d_j) \tag{3.2}$$

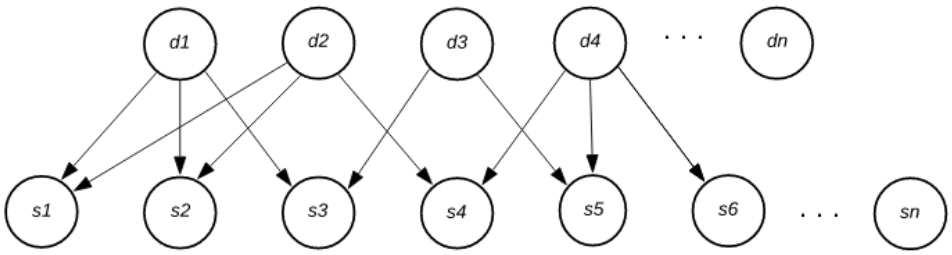
Conversely, the probability of s_i^- is then:

$$P(s_i^-|d_{si}) = \prod_{j=1}^{|d_{s_i}|} P(s_i^-|d_j) \tag{3.3}$$

3.3 BN2O Networks

The BN containing all the probabilistic information used by IntMed, is graphically organized in two horizontal layers where the top layer contains the diseases, and the bottom layer contains the symptoms. Along with this specific structure, the network also uses the Noisy-OR concept. This type of BN formed by 2 layers with Noisy-OR is known as a BN2O network. Figure (3.1) shows a depiction of this setup.

Figure 3.1: *Depiction of a BN2O network*



The joint probability distribution of a BN2O network where the conditionals are based on the Noisy-OR concept, is calculated by first using the chain rule for each state of the symptoms.

This specifies the conditional probabilities on the form:

$$P(s^+|\boldsymbol{d}) = \prod_{i=1}^{I} 1 - P(s_i^-|\boldsymbol{d}) \tag{3.4}$$

and

$$P(s^-|\boldsymbol{d}) = \prod_{i=1}^{I} P(s_i^-|\boldsymbol{d}) \tag{3.5}$$

where s^+ and s^- are the set of all positive and negative symptoms respectively such that $\boldsymbol{s} = s^+ \cup s^-$. By factoring in the disease marginals, the full joint probability table can be obtained:

$$P(\boldsymbol{s}, \boldsymbol{d}) = \left(\prod_{i=1}^{I} 1 - P(s_i^-|\boldsymbol{d}) \right) \prod_{i=1}^{I} P(s_i^-|\boldsymbol{d}) \prod_{j=1}^{J} P(d_j) \tag{3.6}$$

The disease states are unobservable in this domain and information of their states are essentially what users of the system are interested in, whereas the symptom states are observable through patient questioning. The system therefore works by receiving either positive or negative evidence on symptoms through questions to patients during consultations. The model is not the perfect representation of this domain, as multiple diseases in reality can cooperate to cause symptoms, and symptoms may cause other symptoms. The model should instead be seen as the best representation available.

3.4 Exact Inference in BN2O

Exact inference in BNs is the act of using Bayes' rule to calculate the posterior probability of the BN variables based on new evidence on the observable variables. This is the preferable way of updating the belief about the hidden state of the unobservable variables, especially in a medical domain where diagnostic accuracy is highly important. Using the joint probability calculation of Equation (3.6), exact inference in a system like IntMed that uses a BN2O structure is carried out in the following way:

$$P(\boldsymbol{d}|\boldsymbol{s}) = \frac{P(\boldsymbol{s}|\boldsymbol{d})p(\boldsymbol{d})}{\sum_d P(\boldsymbol{s}|\boldsymbol{d})p(\boldsymbol{d})} = \frac{\prod_{i=1}^{I} P(s_i|\boldsymbol{d}) \prod_{j=1}^{J} P(d_j)}{\sum_d \left[\left(\prod_{i=1}^{I} (1 - P(s_i^-|\boldsymbol{d})) \right) \prod_{i=1}^{I} P(s_i^-|\boldsymbol{d}) \prod_{j=1}^{J} P(d_j) \right]} \tag{3.7}$$

Isolation of the new evidence in the denominator to obtain the normalization constant needed for Bayes' rule requires summation over $2^{|\boldsymbol{d}|}$ terms to be calculated and this is unfortunately computationally intractable. There is however an algorithm developed specifically for BN2O structured networks that may be used instead.

3.4.1 The Quickscore Algorithm

The current inference algorithm in the IntMed system is known as Quickscore [15]. It effectively reduces the intractable complexity shown in Equation (3.7) through factorizations derived from consideration of the positive and negative evidence isolation.

The probability calculation of the negative evidence for one negative symptom takes form as the following expression:

$$P(s_i^-) = \sum_d \left[\prod_{j=1}^{|d_{s_i}^+|} P(s_i^- | only \;\; d_j) \prod_{j=1}^{|d_{s_i}^+|} P(d_j^+) \prod_{j=1}^{|d_{s_i}^-|} P(d_j^-) \right] \tag{3.8}$$

where d_j^+ represents d_j being present and $d_{s_i}^+$ are the present parents such that $d_{s_i}^+ \subseteq d_{s_i}$, all of which also applies for absent diseases. By moving the summation of d into the equation rather than taking products over the present and absent diseases of the parent set individually, the equation can be rewritten into a simpler expression:

$$P(s_i^-) = \prod_{j=1}^{|d_{s_i}|} \Big[P(s_i^- | only \;\; d_j) P(d_j^+) + P(d_j^-) \Big] \tag{3.9}$$

The rewriting reduces the complexity of Equation (3.8) from $O(2^n)$, to $O(j)$ thereby making it a linear product rather than an exponential summation. Given Equation (3.9), the joint probability of the entire set of negative symptoms can be found by:

$$P(s^-) = \prod_{j=1}^{J} \left[\prod_{i=1}^{|s^-|} P(s_i^- | only \;\; d_j) P(d_j^+) + P(d_j^-) \right] \tag{3.10}$$

Rewriting of the positive evidence isolation is significantly more complicated. For simplification, the case of having only two positive symptoms is considered. In this case the joint probability of interest is $P(s_1^+, s_2^+)$:

$$P(s_1^+, s_2^+) = \sum_d P(s_1^+ | \boldsymbol{d}) P(s_2^+ | \boldsymbol{d}) P(\boldsymbol{d}) \tag{3.11}$$

Because of Equation (3.2) $P(s_i^+ | \boldsymbol{d})$ can be written as $1 - P(s_i^- | \boldsymbol{d})$ which turns Equation (3.11) into the following:

$$\begin{aligned} P(s_1^+, s_2^+) = & \sum_d P(\boldsymbol{d}) - \\ & \sum_d P(s_1^- | \boldsymbol{d}) P(\boldsymbol{d}) - \\ & \sum_d P(s_2^- | \boldsymbol{d}) P(\boldsymbol{d}) + \\ & \sum_d P(s_1^- | \boldsymbol{d}) P(s_2^- | \boldsymbol{d}) P(\boldsymbol{d}) \end{aligned} \tag{3.12}$$

The sum of $P(\boldsymbol{d})$ in the expression above is equal to 1 and the three sums following it are $P(s_1^-)$, $P(s_2^-)$, and $P(s_1^-, s_2^-)$ respectively.

Using Equation (3.10), the expression can be rewritten as:

$$\begin{aligned} P(s_1^+, s_2^+) = \; & 1 - \\ & \prod_{j=1}^{|d_{s_1}|} P(s_1^- | only \;\; d_j) + P(d_j^-) - \\ & \prod_{j=1}^{|d_{s_2}|} P(s_2^- | only \;\; d_j) + P(d_j^-) + \\ & \prod_{j=1}^{|d_{s_1} \cup d_{s_2}|} P(s_1^- | only \;\; d_j) + P(s_2^- | only \;\; d_j) P(d_j^+) + P(d_j^-) \end{aligned} \tag{3.13}$$

By combining the rewriting of both the positive and negative evidence isolation, the following new expression for isolating all new evidence can be obtained:

$$P(s^+, s^-) = \sum_{s' \in 2^{s^+}} (-1)^{|s'|} \prod_{j=1}^{J} \left(\left[\prod_{s_i \in s' \cup s^-} P(s_i^- | only \;\; d_j) \right] P(d_j^+) + P(d_j^-) \right) \tag{3.14}$$

where 2^{s^+} is the power set of positive symptoms. Having obtained the normalization constant $P(s^+, s^-)$ it can used within Bayes' rule:

$$P(\boldsymbol{d} | s^+, s^-) = \frac{P(\boldsymbol{d}, s^+, s^-)}{P(s^+, s^-)} = \frac{P(s^+, s^- | \boldsymbol{d}) P(\boldsymbol{d})}{P(s^+, s^-)} \tag{3.15}$$

By using the fact that $P(\boldsymbol{s}) = P(s^+, s^-)$, the equation can be rewritten into something that highly resembles Equation (3.7), namely:

$$P(\boldsymbol{d} | \boldsymbol{s}) = \frac{P(\boldsymbol{d}, \boldsymbol{s})}{P(\boldsymbol{s})} = \frac{P(\boldsymbol{s} | \boldsymbol{d}) P(\boldsymbol{d})}{P(\boldsymbol{s})} \tag{3.16}$$

Quickscore makes exact inference possible, however, the algorithm has a flaw when considered in the context of being used for real time diagnostics. Notice that Equation (3.14) has to sum over the power set of positive symptoms. This means its calculation time is $O(2^{|s^+|})$. This is problematic, seen as a high number of positive symptoms is a common phenomenon in medical consultations. Because of this, exact inference is not possible in reasonable time given the real-time constraint. To make IntMed more useful as a real-time CDSS, alternate ways of calculating the disease posteriors based on symptom evidence must be investigated.

3.5 Approximate Inference

Approximate inference methods sacrifice posterior accuracy, but in return speeds up the calculation time to fit the needs of a real-time system. In general, approximate methods take advantage of the following part of the posterior calculation:

$$P(\boldsymbol{s}, \boldsymbol{d}) = P(\boldsymbol{s}|\boldsymbol{d})P(\boldsymbol{d}) \tag{3.17}$$

This is equivalent to the calculation of the numerator (otherwise known as the hypothesis term) in Bayes' rule in Equation (3.7) which shows that it is calculated as a linear product over the same variables as the denominator. Approximate methods use this to produce posterior estimates that are well-reasoned, since they are calculated from the same hypothesis term used for exact inference.

Popular approximate methods applied to BN2O networks include sampling [16], where sample configurations of the hypothesis term are used to statistically determine the normalization constant. There are also variational Bayesian methods [12] where variables of the hypothesis term are made dependent on parameters. These parameters are then iteratively updated until the hypothesis term calculation yields a good approximation of the normalization constant.

Another approximation method that combines the concepts of sampling and iterative parameter learning are neural networks. Since many of the concepts that goes into neural networks are key for understanding the approach of using recognition networks, this particular approximation method will be explored further in the next subsection.

3.5.1 Neural Networks

Neural networks (NN) in computer science [17–19][1] are artificial representations of their biological counterpart and are able to learn prediction of a given input's classification[2]. Despite the name resemblance, NNs are to be seen as function approximators rather than artificial brains.

Plain NNs are structured as a sequential stack of layers, where each layer contains a set of neurons. The first layer is known as the input layer and has a number of neurons equal to the number of input features. Subsequent layers after the input layer are called hidden layers, and each of these have an arbitrary number of neurons. The final layer is the output layer which has a number of neurons equal to the set of possible classification classes. Inputs to the network are always in the form of real numbers, meaning that inputs which are not on this form must be transformed before being inputted[3].

More formally, let $\boldsymbol{a}$ be a vector containing j total input features, and let $\boldsymbol{f}$ be a vector of output neurons, then each output neuron is a function approximator for the collective input such that $\{a \in \mathbb{R} \mid \forall f \in \boldsymbol{f},\ \ f(a_1, \ldots, a_j) \in \mathbb{R}\}$.

[1] These references are the foundation of this entire subsection.

[2] Neural networks are used for other types of prediction than just classification, but since classification is the main concern in this thesis they will be explained in this context.

[3] This transformation is known as one-hot encoding.

The neurons in the layer sequence are connected through tensor edges where each edge have an associated *weight*. Each neuron of a layer is connected to all the neurons in the next. To formalize this concept, consider a network with two layers, an input and an output layer. The weights between these layers forms a weight matrix:

$$\boldsymbol{W} = \begin{bmatrix} w_{1,1} & \dots & w_{1,j} \\ \vdots & \ddots & \vdots \\ w_{i,1} & \dots & w_{i,j} \end{bmatrix} \tag{3.18}$$

Where i iterates over the neurons in the second layer, and j iterates over the weights connected to the ith neuron in the second layer, meaning it essentially iterates over the number of neurons in the first layer.

When the network receives input, each input is individually multiplied with each weight connected to the neuron it was inputted to, all of which is summed together. More formally, the dot product is taken between the weight matrix and the input vector:

$$\boldsymbol{W}^\top \boldsymbol{a} = \begin{bmatrix} w_{1,1} & \dots & w_{i,1} \\ \vdots & \ddots & \vdots \\ w_{1,j} & \dots & w_{i,j} \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ \vdots \\ a_j \end{bmatrix} \tag{3.19}$$

where the transposed weight matrix $\boldsymbol{W}^\top$ allows for columnwise multiplication, thereby producing a *weighted sum* for each neuron in the next layer.

All of the neurons in the hidden and output layers have an *activation function*. These functions squish the weighted sum into a value ranging in a small interval. An example is the sigmoid function where any input is outputted as a value in the range $[0, 1]$. If the output is over a certain threshold, the neuron is said to become *activated*. To control this activation, each neuron of the hidden layers and the output layer has a *bias* term b_i which is added to the weighted sum before it is squished through the given neuron's activation function. The final vector notation of this entire process thereby becomes:

$$\boldsymbol{q} = \boldsymbol{f}\left(\begin{bmatrix} w_{1,1} & \dots & w_{i,1} \\ \vdots & \ddots & \vdots \\ w_{1,j} & \dots & w_{i,j} \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ \vdots \\ a_j \end{bmatrix} + \begin{bmatrix} b_1 \\ \vdots \\ b_j \end{bmatrix} \right) \tag{3.20}$$

Where f_j is the output neuron activation function that approximates the probability q_j of the collective input belonging to the j'th class. Written in a more compact notation this becomes:

$$\boldsymbol{q} = \boldsymbol{f}(\boldsymbol{W}^\top \boldsymbol{a} + \boldsymbol{B}) \tag{3.21}$$

where $\boldsymbol{B}$ is the bias vector. It follows from the way the network is structured that its outputs are dependant on the weight and bias parameters. It is through tweaking of these parameters that the network learns.

The learning process starts off with calculation of the difference between the actual output and the desired output, where the latter is known as the *target*. This calculation happens according to a *loss function*. Just like with the activation functions there exist various loss functions, but in essence they all do the same thing; takes targets and actual outputs and calculate their difference. The loss is calculated over all training inputs and targets.

Formally, let M be the number of training examples, let $\boldsymbol{y}$ be a vector of targets, let $\boldsymbol{q}$ be the actual outputs, let $\boldsymbol{\chi} = \boldsymbol{W} :: \boldsymbol{B}$ be the concatenation of the weights and biases, and let L be a loss function, then the loss of an NN is calculated by:

$$loss = \frac{1}{M} \sum_{i=1}^{M} L(\boldsymbol{y}_i, \boldsymbol{q}_i{}^{(\boldsymbol{\chi})}) \tag{3.22}$$

where $\boldsymbol{\chi}$ are the weights and biases that is used by the network to output $\boldsymbol{q}_i$, resulting in the loss being transitively affected by them.

The goal is then to find the configuration of weight and bias values that minimizes this loss calculation. Searching for this configuration is done by exploiting that the negative gradient of a function gives the steepest descent. Therefore, if the negative gradient vector with respect to all weights and biases are calculated and then added to the current weights and biases, this calculation will tweak them towards the minimal loss function. This process is known as *backpropagation* and is applied iteratively for each average of the loss function taken over M training examples. One iteration of this process is given by:

$$\boldsymbol{\chi}' = -\nabla loss(\boldsymbol{\chi}) + \boldsymbol{\chi} \tag{3.23}$$

Where $\boldsymbol{\chi}'$ are the updated weights and biases after one backpropagation iteration.
In practice the NN applies an iteration of backpropagation after having averaged the loss of a subset of n training examples out of the total M. The subset of n training examples is called a *batch*. A traversal of all the training examples using the batch size n such that $\frac{M}{n}$ backpropagation iterations have been conducted, is known as an *epoch*. NNs are usually trained with carefully chosen values for these two numbers.
In summary, an NN is a function approximator that learns parameters to be combined with an input, resulting in an approximated output in close proximity to the exact. It calculates this proximity to the target values and uses it to iteratively update the parameter set. After having learned the parameter set, it can be used for prediction of classes for a given input. Furthermore, the prediction process is timewise feasible for a medical consultation, making an NN a viable candidate for being IntMed's inference engine.

4 Recognition Networks

All sections within this chapter reuses content from [1], but the main reusage happens in the sections 4.1 and 4.2, and subsection 4.2.1.

In this chapter, the concept of Recognition Networks is explored. Firstly, the original version of Quaid Morris' Recognition Network [20] is examined in greater detail. Based on the original version we will then see two new implementations that uses the same basic idea, but with a few variations. Finally, these two new versions are experimented with, in order to examine how well the idea applies to the IntMed case.

4.1 The Original Recognition Network Idea

A Recognition Network (RN) [20] is an NN trained with the specific purpose of inverting a generative model [21], such as a BN2O network constructed from symptom and disease variables. Quaid Morris created an RN with the purpose of optimizing disease posterior calculation time for the QMR-DT system.

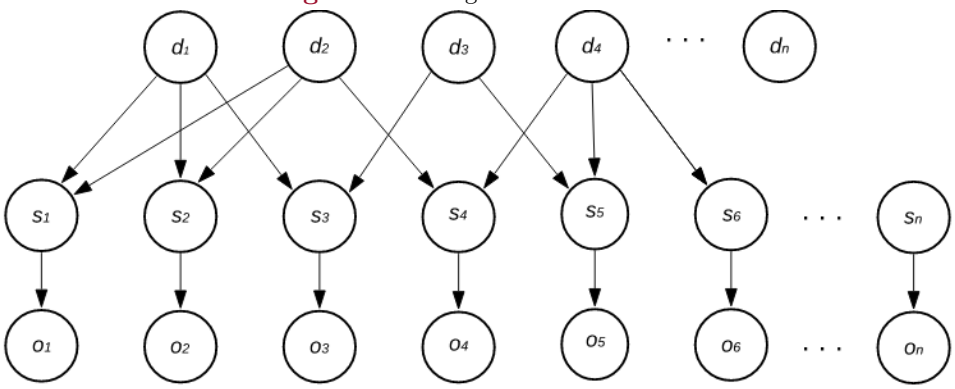
The structure, sampling, and training of this RN was centered around exploitation of two aspects related to the patient consultation process [22]. Firstly, any given consultation has an implicit symptom evidence observation process which must be taken advantage of. The healthcare practitioner and the patient have different outsets for the type of evidence they provide. Healthcare professionals will try to ask the most revealing questions and may notice symptoms the patients themselves might not notice. Patients on the other hand will mainly report positive symptom evidence, and provide it based on physical sensation.

Secondly, there is a tendency towards positive symptoms being revealed more often than negative symptoms. This is the case because healthcare professionals ask revealing questions. In addition, the patients mainly report symptoms they feel are present. This creates an apparent *observation bias ratio* which dictates that one type of evidence is more prominent than the other for any given consultation, with a positive bias ratio being the most common.

To take advantage of these aspects, the BN2O model must be extended to include information about the observation process and bias. This is done by extending it with an extra layer of *observation nodes*, each of them being conditioned on one of the symptoms.

This extension will graphically make the BN2O network look as depicted by Figure (4.1).

Figure 4.1: *Augmented BN2O network*



An observation node o_i is a ternary variable where its states are assigned according to:

$$o_i = \begin{cases} + & s_{o_i} \in s^+ \\ - & s_{o_i} \in s^- \\ ? & otherwise \end{cases} \tag{4.1}$$

where s_{o_i} is the parent symptom of o_i, and $+$, $-$, and ? denote positive, negative, and unobserved, respectively. This introduces a new type of evidence, namely 'unobserved', which is necessary if the observation process is to be modelled. The idea is to depict the consultation such that unobserved symptoms may turn out to be either positive or negative, and then use the bias ratio for guiding the assumption of what state they might be in. The proportion of unobserved nodes that will turn out to be positive and negative, are respectively given by:

$$\begin{aligned} P(o_i = ?|s_i^+) &= p^+ \\ P(o_i = ?|s_i^-) &= p^- \end{aligned} \tag{4.2}$$

thereby, p^+ and p^- are the probabilities that $o_i = +$ and $o_i = -$ respectively, once s_i receives evidence. This concept allows the RN to invert the BN2O model, as it can infer the diagnosis probability from symptom observations and knowledge of the p^+ and p^- ratios. More formally, the probability of an observation node being unobserved given the hidden diseases is calculated by:

$$P(o_i = ?|\, d^+) = p^+ P(s_i^+|\, d^+) + p^- P(s_i^-|\, d^+) \tag{4.3}$$

where d^+ is the set of underlying present diseases.

With a bit of rewriting, it can be shown that this conditional probability also describes the joint probability of the observation being unobserved along with having the hidden diagnosis:

$$\begin{aligned} P(o_i =?|\ d^+) &= P(o_i =?|s_i^+)P(s_i^+|\ d^+) + P(o_i =?|s_i^-)P(s_i^-|\ d^+) \\ &= P(o_i =?,\ s_i^+,\ d^+) + P(o_i =?,\ s_i^-,\ d^+) \\ &= P(o_i =?,\ d^+) \end{aligned} \tag{4.4}$$

Because of this relationship, it can be derived that the probability of hidden diseases conditioned on the unobserved observations, must be proportional to the joint probability $P(o_i =?,\ s_i,\ d^+)$. Through this knowledge, an equation for approximating the desired disease posterior can be written as:

$$P(d^+\ |o_i =?) \propto \left[p^+/p^- P(s_i^+|\ d^+) + P(s_i^-|\ d^+)\right] P(d^+) \tag{4.5}$$

To utilize this new model and its incorporated knowledge of the observation process, the augmented network is forward sampled, thereby producing *observation vectors*, defined by $\boldsymbol{o} = \{0,1\}^{|\boldsymbol{s}|}$, and *reference diagnoses*, defined by $\hat{\boldsymbol{d}} = \{0,1\}^{|\boldsymbol{d}|}$. These serve as inputs and targets respectively, meaning the RN has a number of input layer neurons equal to $|\boldsymbol{s}|$ and output layer neurons equal to $|\boldsymbol{d}|$. Morris experimented with different values for p^+ and p^- such that symptoms were made unobserved in the samples according to values that differed between experiments.

Before sampling from the network, the probabilities of each disease are initially normalized. Diseases with low probabilities are unlikely to be sampled often, resulting in the network training on extremely few cases where the low probability diseases have caused their symptoms. Though it is desirable that the network learns how low probability diseases rarely are the cause of symptoms, the probability values of these disease are often low by several orders of magnitude. Because of this, there is a mismatch between the sample size needed for learning prediction of common diseases, and the sample size needed for learning prediction of the rare diseases with low prevalence [23]. To prevent this mismatch, all diseases are normalized using the following formula:

$$P'(d_k = 1) = \begin{cases} P(d_k = 1) & if\ \ P(d_k = 1) > 0.04 \\ 0.04 & otherwise \end{cases} \tag{4.6}$$

where $P'(d_k = 1)$ is the normalized probability of d_k. Rather than using the diseases generated to be present directly as targets, Morris instead distributed all diseases into vectors with exactly five diseases per vector. For each vector $\vec{d}_m$, the joint probability of having contracted $\vec{d}_m$ along with the specific observation vector $\boldsymbol{o}_n$ is calculated by:

$$P(\vec{d}_m, \boldsymbol{o}_n) = P(\vec{d}_m|\boldsymbol{o}_n)P(\boldsymbol{o}_n) \tag{4.7}$$

These joint probabilities are added to a diagnosis list (*D-List*) in descending order, ranking the most likely diagnoses given the sampled $\boldsymbol{o}_n$ vector. Repetition of this process thereby created samples of $\boldsymbol{o}$ vectors, and their correct reference D-list, $\{\vec{d}_1, \vec{d}_2, ..., \vec{d}_m\}$.

Morris made two RN versions: One with only the in- and output layers, and one with a single hidden layer containing 1000 neurons. The hidden layer of the second version used the hyperbolic tangent (*tanh*) as activation function. Both versions used sigmoid as the output layer activation function.

By having a number of output layer neurons equal to $|\boldsymbol{d}|$ in combination with using sigmoid for these neurons, the output becomes a vector, denoted $\tilde{\boldsymbol{d}}$, of approximated posteriors for each disease. It is desired that the network predicts high probabilities for the diseases from the benchmark D-list $\vec{d}$ and low probabilities for all others. Subsequently it should also penalize incorrect predictions. To obtain this effect, the RN uses the *cross-entropy* loss function given by:

$$XEnt(\boldsymbol{\chi}) = -\sum_{j}\sum_{k} \hat{d}_k^{(\boldsymbol{o}_j)} \log \tilde{d}_k^{(\boldsymbol{o}_j)} + (1 - \hat{d}_k^{(\boldsymbol{o}_j)}) \log(1 - \tilde{d}_k^{(\boldsymbol{o}_j)}) \tag{4.8}$$

where $\tilde{d}_k^{(\boldsymbol{o}_j)}$ and $\hat{d}_k^{(\boldsymbol{o}_j)}$ are the actual and target outputs based on the given input observation vector $\boldsymbol{o}_j$, and $\boldsymbol{\chi}$ is the current set of weights and biases with the notation and interpretation being equal to that of Equation (3.22).

Despite having based the entire argumentation on the concept of having unobserved evidence, Morris strangely did not include a representation for it in his observation vectors, seen as all of these only had binary values for positive and negative. Furthermore, he argued that the D-Lists were a fairer benchmark compared to the output posteriors because the methods he measured against [12, 23] were unable to incorporate the unobserved state.

These remarks are by no means meant for undermining Morris' results, but are instead meant to spark ideas for possible extensions to his implementation, as the idea of using an RN for BN2O inference is quite interesting. In the next section, an RN version that resembles the original will be presented, along with a version that differentiates between the positive, negative, and unobserved evidence.

4.2 Reworked Recognition Network

In this section the implementation of two slightly extended and altered versions of the original RN will be presented. After this presentation, the results of a consultation comparison experiment between the two RN versions and Quickscore will be shown. The purpose of this section is to showcase the usefulness of the basic RN concepts in a medical setting, before diving into a more sophisticated RN structure in the next chapter.

4.2.1 Basic Recognition Network

The Basic Recognition Network (BRN) implementation is made to represent the original implementation, but with tweaks based on a nuanced view on neural network structure.

The BRN is implemented using TensorFlow with Keras as interfacing library. It is structured as a sequential stack of 5 fully connected dense layers; the in- and output layers and three hidden layers.

The input layer has a number of neurons equal to the number of symptoms in the BN2O which with the data from Ambolt amounts to 347. The output layer has 514 neurons based on the number of diseases. The three hidden layers have 600 neurons each.

The number of hidden layers and neurons in them were initially inspired by [24] which indicates that a number of neurons lying between in- and output size is adequate, but it turned out that a large number of trainable parameters is necessary to capture the BN2O relations[1].

The hidden layer neurons use the rectified linear unit (*ReLU*) activation function, and the output layer uses sigmoid. The change of activation function choice for the hidden layers compared to the original RN is made due to *ReLU* being more computationally efficient. In contrast to *tanh* it only has to determine a logical statement rather than calculating a formula for each neuron, making it about 6 times faster [25, 26]. The output layer neurons use sigmoid in order to get a probability output for each disease. The BRN uses the cross-entropy loss function just like the original RN.

The BRN is trained using forward samples of the BN2O network, though without decorating it with observation nodes. The samples are pairs of observation vectors and reference diagnoses. Each observation vector mimics a symptom configuration acquired from having contracted n diseases that are in turn stored in the corresponding reference diagnosis. Each pair is represented as two vectors, $\boldsymbol{o}^{(m)} = \{0,1\}^{|\boldsymbol{s}|}$ and $\hat{\boldsymbol{d}}^{(m)} = \{0,1\}^{|\boldsymbol{d}|}$, where each index in each vector corresponds to the position of the symptom/disease in the BN2O. '1' denotes positive presence, and '0' denotes negative for both symptoms and diseases. This resembles the original RN with no state differentiation, and because of this there is no need for decorating the network, nor incorporate the probabilities p^+ and p^- for determining if positive/negative symptoms are unobserved in the samples.

The sampling process is carried out by first sampling n diseases, where n is predetermined and $n > 0$, resulting in a set of diseases sampled as present, denoted $\hat{d}^+$. Each $d_j \in \hat{d}^+$ are added as '1' values on their respective indices in $\hat{\boldsymbol{d}}$ and the rest are set to '0', thereby finishing the sampling of the diseases. Each symptom conditioned on at least one of the sampled diseases are then sampled according to:

$$P(o_i = +) = 1 - \prod_{j=1}^{|\hat{d}^+ \cap d_{s_i}|} P(s_i^- | d_j) \tag{4.9}$$

Since the database contains no information of the noisy-OR probabilities for the symptoms, each value is given by:

$$P(s_i^- | d_j) = \begin{cases} 0.1 & if\ d_j\ is\ common \\ 0.9 & if\ d_j\ is\ rare \end{cases} \tag{4.10}$$

which models that symptoms are usually being caused by common diseases.

[1] This was discovered through various experiments with network structure. There is unfortunately no direct example showcasing it and backing up this claim. A possible example could be Figure (A.2) in Appendix A, though it depicts a different network type than what is used here.

The sampled symptoms are added to the observation vector $\boldsymbol{o}$ the same way as the diseases which finishes the sampling of the symptoms and thereby the sample-pair.

Two BRNs called BRN1 and BRN3 are trained with 10^7 of such samples, with the number suffix denoting the n of their sampling process. The BRNs were both trained over the course of 10 epochs for each 10^6 samples, with a batch size of 1000.

4.2.2 State Differentiation Recognition Network

The State Differentiation Recognition Network (SDRN) is an implementation of the RN made to support the unobserved state of symptoms. The internal structure of the SDRN is essentially the same as the BRN, except that the number of neurons for the input layer is different. Support of a third symptom state requires an additional bit for each symptom, meaning the new number of input layer neurons becomes $2|\boldsymbol{s}|$. The observation vector $\boldsymbol{o}$ is for SDRN redefined as:

$$\boldsymbol{o} = \{o_i\}^{|\boldsymbol{s}|},\ o_i = \begin{cases} (1,0) & s_{o_i} \in s^+ \\ (0,1) & s_{o_i} \in s^- \\ (0,0) & otherwise \end{cases} \tag{4.11}$$

such that each symptom is represented as a tuple where each tuple denotes one of the states given by Equation (4.1). The network cannot receive tuples as input so to accommodate for this, the dimensionality of the observation vectors is reduced by one which removes the tuple status and therefore requires the number of input layer neurons to be $2|\boldsymbol{s}|$.

The sampling process for SDRN follows the same pattern as that of BRN, but now an additional step is added after Equation (4.9) has been used to sample the symptom's positive/negative status. Based on the probabilities p^+ and p^- used for positive and negative symptoms respectively, it is determined if a newly sampled symptom should in fact be labeled as unobserved. The value of p^+ was set to 0.9 and the value of p^- was set to 0.5, according to the idea that positive symptoms are more likely to be observed.

Just like the BRN counterparts, an SDRN1 and SDRN3 were trained with 10^7 samples, 10 epochs per 10^6 samples, and a batch size of 1000.

4.3 Reworked Recognition Network Experimentation

To test the RN idea, the two reworked RN versions are matched against the current exact Quickscore algorithm used in IntMed. This is done by having the BRNs, the SDRNs, and Quickscore calculate posterior distributions in a simulated consultation scenario. In this simulated scenario, the patient repeatedly reports positive symptoms for a disease that has connections to many symptoms.

Quickscore struggles calculation timewise with large numbers of positive symptoms, meaning the RNs will prove very useful if they can provide posteriors comparable to those calculated by Quickscore, given the same large quantity of positive evidence.

In order to make the experiment timewise manageable, a disease with 15 symptoms in total is chosen to be the underlying disease which makes Quickscore able to finish in feasible time. For the multi-disease trained RNs, the experiment changes slightly; the simulated patient now have 3 underlying diseases that are connected to 5 symptoms each.

The networks and Quickscore are analyzed according to several metrics. The first major metric is the predicted posteriors based on the current evidence set for the underlying disease. This is the most fundamental metric that shows how well the RN approximates the posteriors compared to Quickscore. For this metric higher is better, though close proximity to Quickscore is also desirable.

The second major metric is the ranking of diseases in descending order of posterior probability, with the highest rank being 1. In the plots lower is better for the metric, but ranks described as 'high' will refer to ranks close to 1.

Ranking is important to consider for two main reasons. The first reason is that the system is supposed to be used for diagnostic guidance. Placement of the correct diseases high enough for a doctor to notice them is crucial. Even if the probabilities of all diseases are predicted to be low, the system should still predict the correct diseases to be within the top view of all available diseases.

Secondly, if ranking of the correct diseases is very different from the probabilities predicted for them, it may be a sign that the network over- or underestimates the probabilities. For this experiment $k = 10$ when referring to top-k. Ranks predicted to be outside top-10 are normalized to be 11 in the plots because all ranks outside the top-10 view should be considered equally inferior.

The third major metric is a small ensemble of smaller metrics defined as *top-k similarity metrics*. These metrics measure how similar the top-k predicted diseases are between Quickscore and the given RN. Let $QS_{topk} = \{\hat{d}_j^{(1)}, \ldots, \hat{d}_j^{(k)}\}$ and $RN_{topk} = \{\tilde{d}_j^{(1)}, \ldots, \tilde{d}_j^{(k)}\}$ be the sets of top-k diseases predicted by Quickscore and the RN respectively, then the first top-k metric (TK1) is given by the following set:

$$TK1 = QS_{topk} \cap RN_{topk} \tag{4.12}$$

such that the number for the metric is defined as $|TK1|$. The second top-k metric (TK2) is the same as TK1, but where each element in TK1 is within the same probability interval:

$$TK2 = \left\{ d_j \in TK1, 0 \leq i < 10 \;\middle|\; \frac{i}{10} < P(d_j) \leq \frac{i+1}{10} \bigwedge \frac{i}{10} < Q(d_j) \leq \frac{i+1}{10} \right\} \tag{4.13}$$

where $P(d_j)$ is the posterior calculated by Quickscore and $Q(d_j)$ is the posterior approximated by the given RN for the disease d_j. The number for the metric is given by $|TK2|$.

TK3 is defined as the number of diseases that are found in both QS_{topk} and RN_{topk}, placed on the exact same indices:

$$TK3 = \sum_{i=1}^{k} \mathbf{1}\left(\hat{d}_j^{(i)} = \tilde{d}_j^{(i)}\right) \tag{4.14}$$

where $\hat{d}_j^{(i)} \in QS_{topk}$ and $\tilde{d}_j^{(i)} \in RN_{topk}$.

Finally, TK4 counts all the diseases that satisfies the collective conditions of TK1, TK2, and TK3 simultaneously:

$$TK4 = \sum_{d_j \in TK2} \sum_{i=1}^{k} \mathbf{1}\left(d_j = \hat{d}_j^{(i)} \bigwedge d_j = \tilde{d}_j^{(i)}\right) \tag{4.15}$$

This metric ensemble is included in the experimentation to compare Quickscore with the RNs in a way that does not solely focus on posterior estimation of the underlying diseases. Even though high probabilities given the evidence on the underlying diseases is important, it us arguably just as important that the RN provides a sensible probability and ranking of diseases that are symptom-wise related to the underlying causes.

If the system is unable to reason properly for the related diseases, it would not provide assistance that classifies as advisory, but rather provide a series of educated guesses, and in that case only its highest probability and/or ranked predictions would be of use. It is important to remember that the underlying diseases in a real consultation scenario in fact can be the ones estimated by the system to have low posterior distributions. The top-k content should therefore in those cases give the medical practitioner a valid overview of the possible symptom causes, who may then manually decide upon an appropriate diagnosis.

Because of these reasons it is important to measure how proportional the content of the RN's approximated top-k predictions is to that of exact inference. Aside from enabling top-k similarity measurement, the similarity metrics also provide it on different levels of strictness, where TK2 and TK3 are stricter than TK1, and TK4 are more strict than all the others. A perfect top-k similar RN will predict k diseases that has the same ranking indices and the approximately same estimated posteriors as Quickscore.

The final major metric is the arithmetic mean of the Kullback-Leibler divergence (KL) calculated in bits for all diseases of the BN2O network. Where the similarity metrics are used to give a broad overview of the top-k predictions, the KL-divergence for all diseases is intended to give the broad overview of all predicted probabilities in total. This is calculated and plotted for each prediction. The lower the better for this metric.

The calculation follows the standard KL-divergence equation $KL(P||Q)$ where Quickscore's posteriors are used as the true distribution P and the RN's predictions as the approximated distribution Q. For each disease, the sum of the probabilities of its present and absent states are taken and summed together to get the total divergence between all of Quickscore's and the RN's estimated disease probabilities.

Finally, the average is brought into the equation to obtain the final result:

$$KL(QS||RN)_{avg} = \frac{1}{|\boldsymbol{d}|}\sum_{j=1}^{|\boldsymbol{d}|} P(d_j^+)\log_2\frac{P(d_j^+)}{Q(d_j^+)} + P(d_j^-)\log_2\frac{P(d_j^-)}{Q(d_j^-)} \tag{4.16}$$

For both of the comparison experiments depicted in Figure (4.2) and in Figure (4.3), the left column of diagrams shows the results of the simulated consultation where the patient has 1 underlying disease, and the right column show the results of the simulated consultation where the patient has 3 diseases. The RNs with matching suffixes to these numbers are used accordingly. The results of each column were produced during the same experiment.

4.3.1 BRN1 Experimentation

We first consider Figure (4.2) which shows the experimental results of the BRN, and we start of by examining the results of BRN1.

For the probability prediction of the underlying disease BRN1 predicts probabilities close to 0 for the first 7 positive symptoms compared to Quickscore which estimates the probability to be in the range $[0.4, 0.45]$ after just 3 positive symptoms. By the 8th positive symptom both Quickscore and BRN1 estimates the correct disease to have a probability within $[0.9, 1]$ for the rest of the consultation, with the only exception being BRN1's prediction at the 8th positive symptom.

Even though BRN1 is behind Quickscore probability-wise in the beginning, it is vastly ahead in ranking. The correct disease is placed within top-10 after just 2 positive symptoms and after 4 it is ranked as the first consistently, only deviating from the spot at the 6th prediction where Quickscore also deviates.

The similarity metrics show quite poor results with only about 1 disease predicted by the RN being found within Quickscore's top-10 highest posterior calculations. The presence of all similarity metrics from question 8 and onwards undoubtably stems from the correct disease, seen as both Quickscore and the RN place it as top-1 at this point.

The KL-divergence show a vast deviation of all diseases from 3 to 7 positive symptoms, whereafter it slowly moves downwards. The steepest ascend in divergence happens after 3 questions, which is also depicted by the other metrics where ranking is vastly different from probability prediction, and similarity is non-existent at that point in the consultation.

BRN1 is best when it comes to predicted probability and ranking. Its predicted posteriors are however very steep in their ascendance rather than increasingly progressive. The KL-divergence is overall quite low considering the number for each plot is an average of the divergence of all diseases. The similarity metrics are BRN1s weakest metric, showing that it is unable to infer adequate posteriors for the diseases related to the underlying.

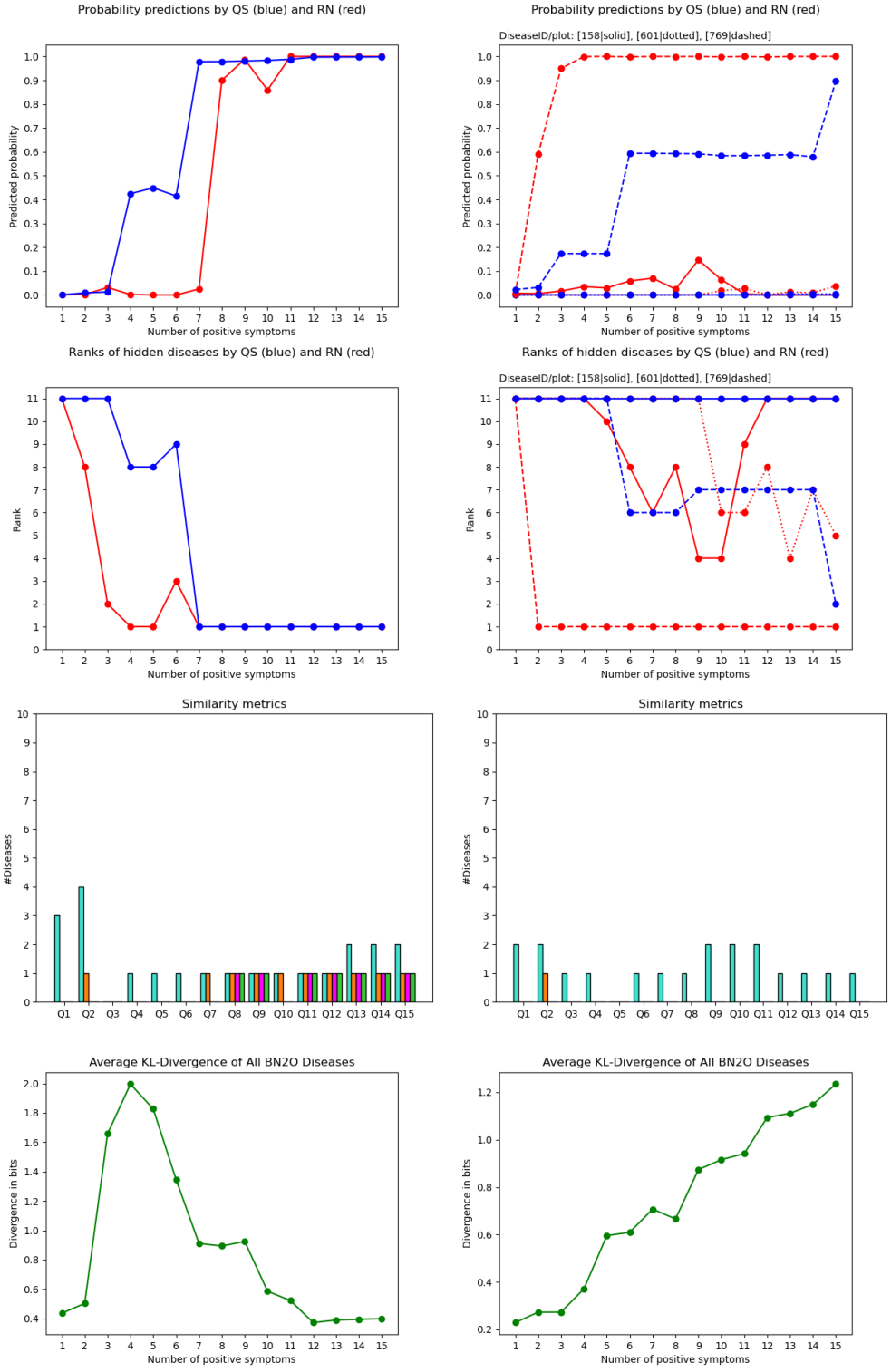


Figure 4.2: *BRN1 (left column) and BRN3 (right column) compared to Quickscore. Rows from top to bottom: Predicted probabilities for the underlying diseases, ranking of underlying diseases, the similarity metrics where TK1 is cyan, TK2 is orange, TK3 is purple, and TK4 is green, and the average KL-divergence of all diseases.*

4.3.2 BRN3 Experimentation

We now turn the attention to BRN3 in Figure (4.2).

Probability-wise, BRN3 outperforms Quickscore quite a bit on one of the diseases with its probability being constantly predicted to be within $[0.9, 1]$ after just 3 positive symptoms. Though the probabilities for the remaining diseases are estimated to be quite low by BRN3, it still gives them quite high ranking by placing them in between top-8 and top-4.

Its similarity metrics are worse than those of BRN1, seen as no diseases at all are found on the same indices in Quickscore's top-10 set. The KL-divergence is overall lower, but steadily increases.

The overall result of the BRN shows that it is quite extreme in its predictions and tends to either predict vastly low or high probabilities. Despite of this it produces some decent estimations with some of them outperforming Quickscore. Its ranking results are very promising, with the correct diseases being found within top-10 after very few positive symptoms. BRN lacks most in terms of the similarity metrics, indicating that it has been fit to predict high probabilities for the correct diseases, and disregard all others.

4.3.3 SDRN1 Experimentation

We now consider the results of the experimentation with SDRN shown by Figure (4.3), and start off with SDRN1. These experiments use the same diseases as the ones used for the BRN experiment, but with the order of incoming positive evidence being different. The posteriors of Quickscore are thus different from those calculated in Figure (4.2).

Surprisingly, despite the new order, SDRN1 predicts roughly according to the same pattern as BRN1 with the only difference being that SDRN1 predicts the probability to be within $[0.9, 1]$ one additional positive symptom later. Overall, for this metric, SDRN1 outperforms Quickscore quite a bit as it takes 5 additional positive symptoms for Quickscore to reach the same probability output for the correct disease.

The ranking shows an even greater result for SDRN1 where the prediction based on 4 positive symptoms of the correct disease places it at top-2 with the subsequent positive symptom bringing it to top-1 which it keeps, except at 7 symptoms.

The similarity metrics are however very poor for SDRN; $\frac{7}{15}$ of the predictions have completely different top-10 content from Quickscore, and the similarity shown for question 13 to 15 are the correct disease only.

The KL-divergence shows about the same result as that of BRN1, where the average divergence is highest at the 4th prediction after which it rapidly falls. The effect of this is visible via the ranking metric which shows completely different rankings at this point.

The conclusion of SDRN1 is the same as that of BRN1, except that SDRN1 is even more extreme in its predictions for the underlying disease. It predicts either 0% or 100% posterior probability, and estimates ranks that are either completely outside of top-10, or within top-2. This subsequently increases the KL-divergence and dissimilarity from Quickscore's posteriors.

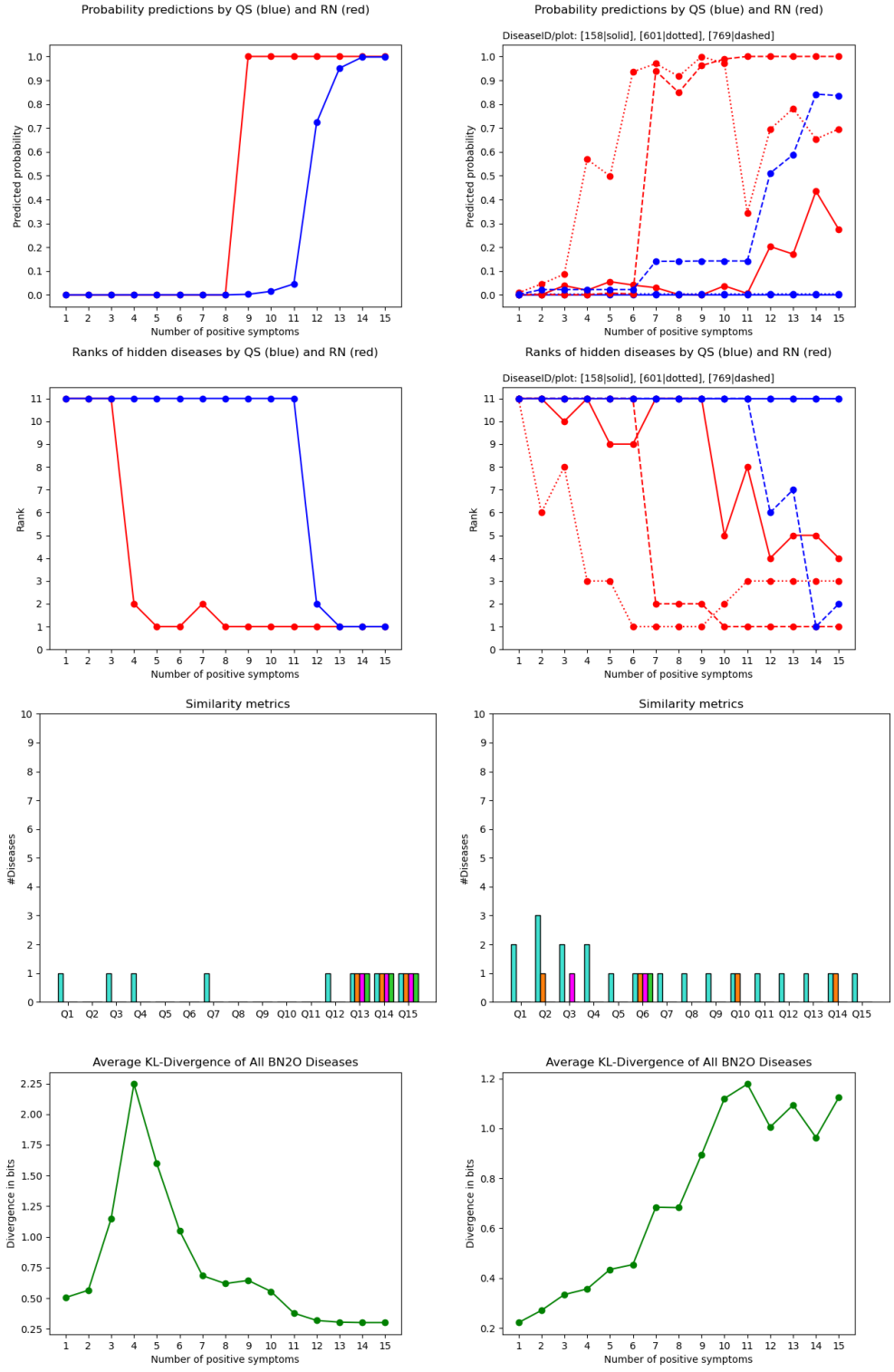


Figure 4.3: *SDRN1 (left column) and SDRN3 (right column) compared to Quickscore. Rows from top to bottom: Predicted probabilities for the underlying diseases, ranking of underlying diseases, the similarity metrics where TK1 is cyan, TK2 is orange, TK3 is purple, and TK4 is green, and the average KL-divergence of all diseases.*

4.3.4 SDRN3 Experimentation

We now turn the attention towards SDRN3 in Figure (4.3).

SDRN3 outperforms both BRN3 and Quickscore in the first major metric. The probability of the disease depicted by the dotted line increases slightly from 1 to 3 positive symptoms, before going above 0.5 where it (approximately) stays above for the rest of the consultation. The dashed line disease behaves roughly like the disease predicted by BRN1, with a high surge and a slight fall in posterior probability after 7 positive symptoms. The disease represented by the solid line struggles quite a bit in terms of probability, even towards the end of the consultation where all of its symptoms have received evidence.

In comparison, Quickscore only provides a high probability for the dashed disease towards the end of the consultation.

The ranking mirrors the underlying diseases output probabilities, showing that SDRN3 aside from outputting better posteriors, also includes all diseases in the top-10 view. SDRN3 have two of the three diseases ranked outside top-10 in $\frac{6}{15}$ of the predictions, whereas that number for Quickscore is $\frac{14}{15}$.

The similarity metrics look better for SDRN3 compared to those of SDRN1. The initial 4 predictions have more than 2 diseases in common with Quickscore, where some of them are even located at the same index or has about the same posterior. The rest of the predictions are quite dissimilar, with the only slight outliers to this being the predictions made based on 6, 10, and 14 positive symptoms.

The average KL-divergence throughout the consultation shows the same result as that of BRN3, but rather than being increasingly ascending, it approximately evens out towards the end with slight increases and decreases in bits.

4.3.5 Recognition Network Concept Evaluation

In conclusion of the RN's medical consultation applicability, it definitely has great potential, but could use tweaking towards producing more similar results to those of the current IntMed system. Both RNs are capable of approximating high posterior probabilities when predicting based on symptom configurations that has a high number of positive symptoms for the correct diseases. In addition, the RN's ranking of the correct diseases turns out to be a metric where it outperforms Quickscore significantly.

The RN is however majorly behind behind the current IntMed system when it comes to classifying as an assistance tool. It does not produce a valid top-view to be used by medical practitioners for guidance because its top-10 diseases are vastly different from those produced by exact inference.

Furthermore, there is a mismatch between its ranking and probability output of the correct diseases. The ranking for these diseases were shown to be good from the beginning of the consultation, even though their posterior probabilities were set to be low, indicating that the network underestimates the posteriors.

Despite the underestimation and the dissimilarity, the overall comparison expressed via the average KL-divergence between all disease posteriors shows quite decent results. The number of diseases in the BN2O is after all within triple digits meaning that a divergence of 1 bit on average for each disease is quite good for these straightforward RN implementations. It could however be useful to bring down the spikes in difference, which is essentially a conclusion that applies to all plots and metrics in both Figure (4.2) and Figure (4.3).

In summary, the RN idea has proven to be useful in a consultation scenario and has potential to become the inference engine of IntMed, but it needs correction towards being more similar to the currently used Quickscore algorithm. In the next chapter, a new RN structure that exploits certain consultation aspects will be presented, in order to carry out this correction.

5 Recurrent Recognition Network

In the previous chapter, it was shown how the RN concept has great potential to act as a new inference engine in IntMed, where very straightforward RN implementations were shown to give promising results. In this chapter, the network will be redone using a recurrent NN structure that is inherently able to represent unobserved inputs and allows the system to exploit the temporal unveiling of evidence inputs.

5.1 The Motivation for Inclusion of Recurrence

Before going into the design of a recurrent RN version, some proper motivation for doing so should be given.

The greatest tendency in literature regarding machine learning solutions for medical diagnostic purposes where NNs are used as the base model, has mostly been concerned with analyzing images to determine a diagnosis. [27–30]. The networks used in this regard have also in some cases been extended to analyze the temporal development through different images [31, 32].

Literature concerning usage of NNs for medical patient consultation is however scarce, especially in terms including temporal analysis. Most examples described in papers specifically designed for patient consultation, concerns with analyzing Electronic Health Records (EHR) [33–35]. These networks are usually deployed in scenarios where the network must be able to infer a diagnosis from text descriptions of symptoms made by patients. Aside from text analysis they have also been used to analyze temporal changes of patients' real-number statistics, such as blood-pressure or cholesterol-levels.

The motivation of using networks that support temporal analysis is clear for systems where the main goal is to analyse EHR. Changes in medical statistics over time is important to monitor and may provide valuable information with regards to a given patient's disease course.

Even though the RN approach is more focused on real time diagnosis using positive/negative presence of symptoms, it can still benefit from considering temporal symptom revealing. Rather than learning the relation between changes in real numbers and disease courses, the RN would instead learn the relation between symptom evidence time sequences, and disease diagnoses. Through this idea it will be possible to more closely fit the way the network predicts to the consultation scenario. This is however a challenge to implement for the IntMed case because the BN2O does not contain any information regarding temporal symptom revealing. To realize this concept, some thought must therefore be given to the design of the sampling.

Modelling of the temporal aspect introduces an opportunity to conveniently address a problem related to the current representation of unobserved evidence. The main idea of including this concept is to introduce the idea that some proportion of symptoms in a consultation will remain missing. This was the reason why the values of p^+ and p^- were used in the previous chapter to simulate how some symptoms would remain hidden in the samples. Modelling this using temporal input comes more naturally; if a symptom remains unobserved, it will simply not be given as input.

The question is whether the representation of unobserved should be kept for the input vectors. Given that the effect of unobserved evidence can be achieved through absent input by using a recurrent network, it might be excessive to model all three states for each symptom. Contrarily, it might make the network unable to properly differentiate between negative and unobserved evidence. The results of Section (4.3) indicate that there is not much difference, but since these are not based on recurrent structures, they cannot be decisive in this regard.

Deciding upon the best representation comes down to experimentation. For now, inclusion of recurrence gives the network more options in terms of how evidence should be represented, which is an advantage over the sequential networks shown in the previous chapter.

5.2 Modelling the Diagnostic Process

In this section the diagnostic process is analyzed with the purpose of determining a way to integrate it into a recurrent RN implementation. This will enable the RN to utilize the most important aspects of a standard consultation to its advantage.

To establish some requirements for the new recurrent RN implementation, it is first necessary to consider the typical aspects of a medical consultation between a patient and a medical practitioner. In this scenario, the patient is likely to have initial symptoms which have sparked the need for a medical consultation in the first place. Aside from the initial symptoms, the patient might have additional symptoms that only the medical practitioner is able to reveal through questioning, due to the patient being unaware of the significance of their physiological signs.

Even though the questioning process may reveal the majority of relevant symptom information, it is likely that some information stays unobserved due to various reasons. This can happen if the patient for instance unknowingly gives false negative evidence that would otherwise have led the medical practitioner to deduce additional information [36, 37]. Because this effect is common in medical consultations, a proper RN implementation needs to be able to differ between positive, negative, and unobserved evidence on symptoms.

Another important consultation aspect that is the temporal information unveiling. In Section 4.1 the observation bias of a consultation was discussed, where it was noted that positive symptoms are likely to be in abundance compared to negative symptoms. Patients will mainly report positive symptoms because of the physiological effect they feel from these. In turn, patients will not report symptoms they do not feel the presence of, and so negative evidence is mostly reported based on the medical practitioner's questions.

Given these factors it is valid to assume that patients will initially report the symptoms they feel the most which may indicate a greater probability for the diseases connected to these pressing symptoms. This concept is interesting to consider because it indicates that the order of incoming evidence matters and may be used in the inference process.

It should however be noted that the exact order is not what is interesting here, but rather the concept modelling the symptoms as an unveiled sequence, rather than an accumulated symptom set.

With these considerations in mind, the question is now how to make an RN model that can support these aspects. As a starting point we consider the SDRN presented in Section 4.2.2 which is able to represent symptoms as either positive, negative, or unobserved.

The SDRN is trained using forward samples of IntMed's BN2O. Informally, each sample represents a simulation of a patient who consults a medical practitioner with a medical case. Formally, each of such medical cases has an observation vector $\boldsymbol{o}$ and a corresponding target diagnosis $\hat{\boldsymbol{d}}$ that caused it. Before any evidence has been added to $\boldsymbol{o}$, its content is defined by:

$$\boldsymbol{o}_0 = \{(0,0)\}^{|\boldsymbol{s}|} \tag{5.1}$$

where the tuple representation is from Equation (4.11). With this setup it is assumed that after the vector has been sampled, it contains all the information of the consultation it represents. Each observation o_i is at this point in a state based on the evidence its corresponding symptom s_{o_i} received during the consultation. The possible states they may have taken (or kept if they were not observed) are defined by Equation (4.11).

When the network has been trained and then carries out predictions, the input sequence given to SDRN as the consultation progresses happens according to the following description: Let ζ be a consultation where $T = |\zeta|$ and let $1 \leq t \leq T$, then we have that:

$$\zeta = (\boldsymbol{o}_1, \ldots, \boldsymbol{o}_T) \tag{5.2}$$

where $\boldsymbol{o}_t = [o_1, \ldots, o_{i-1}, o_i, o_{i+1}, \ldots, o_I]$ such that $o_i = o_t^{(i)}$, with $o_t^{(i)}$ being the representation of the symptom that were observed with question t.

SDRN predicts by using the symptom configuration made from the symptom states accumulated in $\boldsymbol{o}_t$. It only considers past observed symptoms in the sense that they are part of the configuration to base the prediction upon at t. Thereby, the input vector $\boldsymbol{o}$ given to the SDRN after t symptoms have been revealed, has a positive/negative value representation for each previously revealed symptom on their respective indices. The trail of evidence from prior predictions given by $(\boldsymbol{o}_1, \ldots, \boldsymbol{o}_{t-1})$ is thus completely neglected, and the prediction is solely based upon the accumulated evidence in $\boldsymbol{o}_t$.

A recurrent RN will be able to include the observation vector inputs given by $(\boldsymbol{o}_1, \ldots, \boldsymbol{o}_{t-1})$ into its prediction. The vectors given as input to the recurrent network, $\boldsymbol{o}_t$, should be redefined such that the entire vector only contains information about $o_t^{(i)}$ and nothing else.

Without this redefinition, the RN will learn a temporal accumulation which will model a scenario where the patient reports all already reported symptoms for each question.

This defines the requirements needed for making a temporal supportive RN structure. In the next section, the NN type which can be used to implement them with will be examined.

5.3 Recurrent Neural Networks

Recurrent Neural Networks (RNN)[38–41][1] are the fundamental model that is deployed in machine learning when the problem to solve requires analysis of input sequences rather than a single input. Common use cases for this are audio, text, or frame processing, where these entities might come in different quantities. The predictions of interest in these scenarios relate to how a given quantity in a certain sequence should be interpreted. In this section the general concept an RNN will be explained using the RN case as a reference for notation.

An RNN fundamentally works by sharing and passing information to itself. Specifically, an RNN *shares all weights and biases across time*, while also passing a *hidden state* to its future self which contains information about the time steps computed in the past.

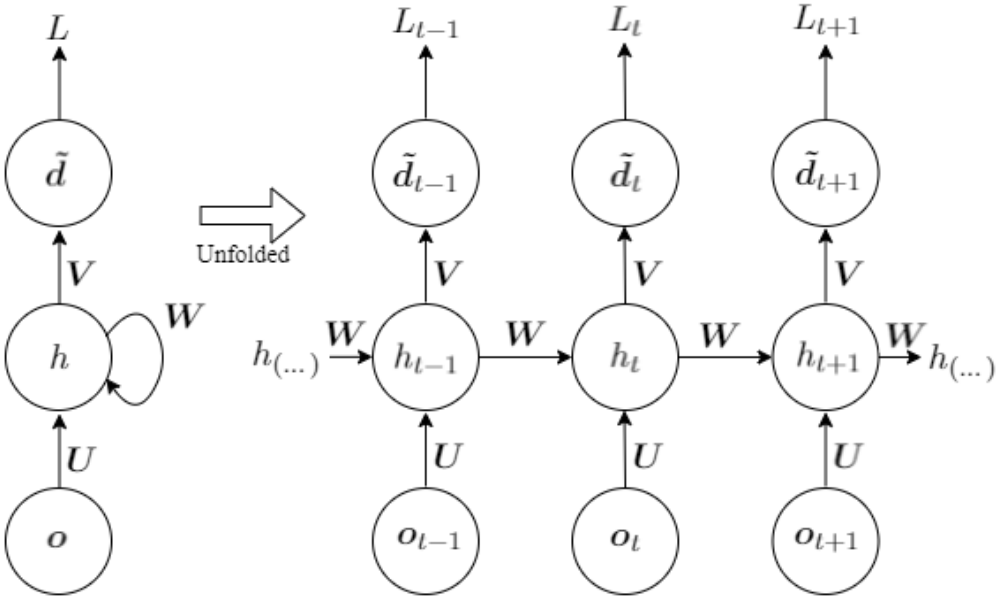


Figure 5.1: *The computational graph of an RNN network where the inputs are depicted as observation vectors and the output is depicted as predicted diagnoses. Note that the bias vectors* $\boldsymbol{B}$ *and* $\boldsymbol{C}$*, used in the recurrent and output connections respectively, have been omitted for notational simplicity.*
Source of inspiration: [40].

[1] These references are the foundation of this section, and its subsection.

Figure (5.1) shows this process, using the RN notation to denote the input and output. The leftmost model depicts the computational graph of an RNN in a compact way. Upon receiving an input, it recursively computes the hidden state h using the input weight matrix $\boldsymbol{U}$, and the weight matrix $\boldsymbol{W}$. These weight matrices have been passed forward in time from its past self to its current self. Using the hidden state and the output weights $\boldsymbol{V}$, it computes a weighted sum which is passed through an output activation function to produce the output. Finally, this output is passed to the loss function L where its proportionality to the target is calculated.

L for the RN case takes form as the cross-entropy loss defined by Equation (4.8). Though it is not depicted, this function receives the target corresponding to the given time step such that $L_t = L(\hat{\boldsymbol{d}}_t, \tilde{\boldsymbol{d}}_t)$.

The forward propagation process of the left side model in Figure (5.1) is unfolded in the right side of the Figure. Notice that the weight matrices are not subscripted by t, as these are shared across time steps. The formal definition of the forward propagation, and incidentally also the entities of the graph is is given by:

$$h_t = f(\boldsymbol{B} + \boldsymbol{W} h_{t-1} + \boldsymbol{U} \boldsymbol{o}_t) \tag{5.3}$$

$$\tilde{\boldsymbol{d}}_t = \sigma(\boldsymbol{C} + \boldsymbol{V} h_t) \tag{5.4}$$

where $\boldsymbol{B}$ and $\boldsymbol{C}$ are the bias vectors of the recurrent and the output connections respectively, f is an activation function, and σ is the sigmoid activation function.

The activation function applied for computation of the hidden state can be any activation function desired by the network engineer, but usually it is defined as *tanh*.

Finally, the way an RNN learns is by using backpropagation just like standard NNs, with the only addition of calculating it across time-steps. The intuition for this calculation is that the RNNs time steps are unfolded as depicted in the right side of Figure (5.1), then the loss is calculated for each unfolded time step, and finally it is rolled backed, summed, and averaged. This is known as *backpropagation through time* and is calculated by:

$$loss_T = \frac{1}{T} \sum_{t=1}^{T} loss_t \tag{5.5}$$

where $loss_t$ is the *loss* defined by Equation (3.22), calculated at time step t.

An RNN defines a very powerful computational model, capable of taking time sequences into account when predicting. It does however have an undeniable flaw that basically make it unfitting for most tasks.

Just like standard NNs, RNNs can suffer from the *vanishing gradient* problem. This problem occurs when the gradient calculated during backpropagation becomes small or large by several orders of magnitude. This happens because the backpropagation calculation shrinks or expands the gradient exponentially as the process moves through the network layers.

For RNNs this is especially prone to happen because the exponential scaling is amplified further when the gradient is calculated across multiple time steps. This is called *vanishing gradient through time* and, and happens due to a property of the weight matrix $\boldsymbol{W}$, namely that it allows for an Eigen decomposition on the form:

$$\boldsymbol{W} = \boldsymbol{\Theta}\boldsymbol{\Lambda}\boldsymbol{\Theta}^{\top} \tag{5.6}$$

where $\boldsymbol{\Theta}$ are the weights represented as eigenvectors and $\boldsymbol{\Lambda}$ is a matrix where the diagonal consists of eigenvalues. This decomposition describes an exponential relation between the initial hidden state h_0 and h_t. If the decomposition of $\boldsymbol{W}$ in Equation (5.6) is substituted into Equation (5.3), then the effect of calculating the weighted sum of $\boldsymbol{W}$ and the initial hidden state h_0 across t time steps can be seen:

$$h_t = f(\boldsymbol{B} + \boldsymbol{\Theta}\boldsymbol{\Lambda}^t\boldsymbol{\Theta}^{\top}h_0 + \boldsymbol{U}\boldsymbol{o}_t) \tag{5.7}$$

where t acts as an exponent for the diagonal eigenvalues. The effect of this is that the weights of $\boldsymbol{W}$ along the diagonal will either exponentially shrink towards 0 if they are less than 1 or expand exponentially if they are greater than 1 with t as the exponent. The intuition of this problem is that the networks will either forget or overemphasize past information. This effect is inevitable, so the only way to counteract it is by slowing down the exponential scaling. For this purpose, there has been developed an alternate version of the basic RNN, designed to lessen the forward propagated information by remembering long-term dependencies across time steps.

5.3.1 Long Short-Term Memory Networks

The Long Short-Term Memory (LSTM) [42] is an extension of the plain RNN which filters information as it recurs through time. This regulates h in a way that decreases the exponential weight scaling it would otherwise suffer from due to Equation (5.7).

LSTM works by vastly modifying the computation of h_t. Firstly, it introduces the concept of *cells* that differs from the sequential layer structure by having a fixed number of layers per cell. Secondly, in addition to the forward propagation of h, it also passes forward a *cell state*, denoted c. This cell state is used to regulate the result of the computation between the hidden state and the input by adjusting how much information that should be kept or discarded.

Figure (5.2) shows a depiction of the internal structure and calculation flow of an LSTM cell. Once again, the RN notation is used where appropriate to exemplify where it would be part of the calculation.

The Figure is read from left to right, starting in the bottom-left corner where the previous hidden state h_{t-1} and the current input observation vector $\boldsymbol{o}_t$ gets concatenated, before being passed on to four different *gates*. These gates are essentially NN layers with weights, biases and activation functions applied elementwise to the input.

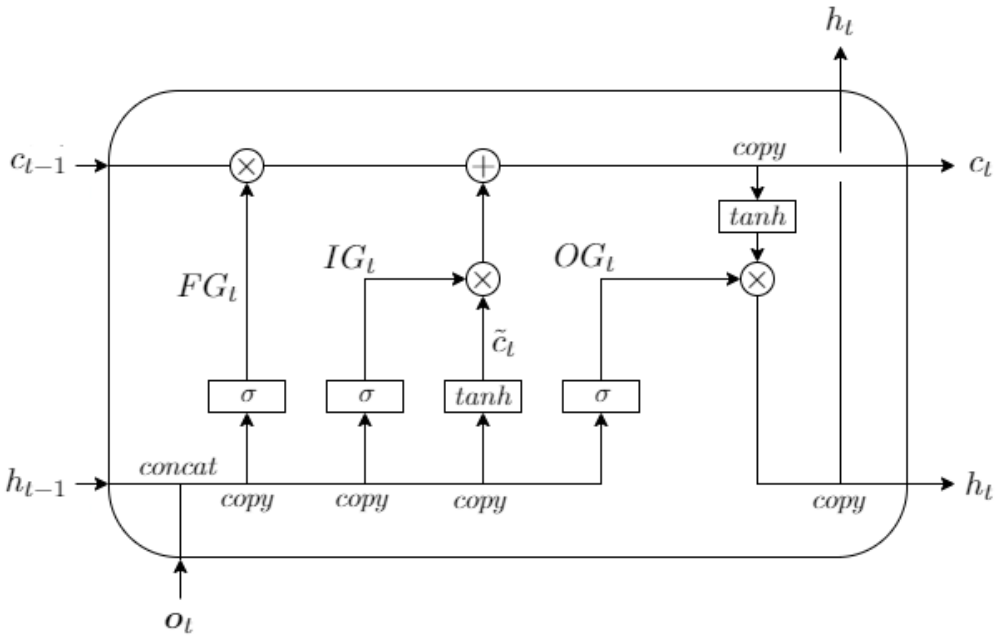


Figure 5.2: *The internal structure of an LSTM cell. The notation of concat and copy denotes the action taken when the left-to-right flow either merges or splits. Source of inspiration: [41].*

The first gate is the *forget gate* which uses sigmoid to scale unnecessary information towards 0 and necessary information towards 1. The output of this computation, FG_t, is given by:

$$FG_t = \sigma(\boldsymbol{W}_{FG} \cdot [h_{t-1}, \boldsymbol{o}_t]^\top + \boldsymbol{B}_{FG}) \tag{5.8}$$

The next gate is the *input gate* which computes its outputs exactly like the forget gate, with the only exception being that it uses its own weights and biases:

$$IG_t = \sigma(\boldsymbol{W}_{IG} \cdot [h_{t-1}, \boldsymbol{o}_t]^\top + \boldsymbol{B}_{IG}) \tag{5.9}$$

The next gate computes the *candidate cell state* $\tilde{c}_t$ meant for potentially updating the information of c_{t-1}:

$$\tilde{c}_t = tanh(\boldsymbol{W}_c \cdot [h_{t-1}, \boldsymbol{o}_t]^\top + \boldsymbol{B}_c) \tag{5.10}$$

The first three gates collectively provide the components needed for computing the new cell state c_t. After having produced the components, the new cell state can be computed by elementwise appliance of the arithmetic operators shown in Figure (5.2). The definition of the new cell state is thus given by:

$$c_t = FG_t \cdot c_{t-1} + IG_t \cdot \tilde{c}_t \tag{5.11}$$

These arithmetic operations have certain interpretations. FG_t is multiplied with the previous cell state c_{t-1} such that the information carried by this cell state is filtered by importance where important information will be kept (multiplied by 1) and unnecessary information will be discarded (multiplied by 0). The multiplication of IG_t and $\tilde{c}_t$ followed by the addition of c_{t-1} which has been multiplied with FG_t, creates the effect of filtering both old and new information, and then adding it together. In extension to this, since $\tilde{c}_t$ is the output of *tanh*, its sign will simulate the effect of having new information dictate if something should be emphasized (positive sign) or forgotten (negative sign).

The final gate is the *output gate* which is meant for computing h_t by firstly filtering information just like the forget gate:

$$OG_t = \sigma(\boldsymbol{W}_{OG} \cdot [h_{t-1}, \boldsymbol{o}_t]^\top + \boldsymbol{B}_{OG}) \tag{5.12}$$

OG_t is then multiplied with the new cell state that has been squished through *tanh*, thereby producing the final output h_t:

$$h_t = OG_t \cdot tanh(c_t) \tag{5.13}$$

this computation is interpreted as the new cell state regulating h_t, using the sign of *tanh* to scale its information.

Notice in Figure (5.2) that h_t is outputted twice. This is because h_t should both be recurred to the next time step (the horizontal h_t output), but also be available as the network's actual output (the vertical h_t output). The actual output can be used directly as a prediction or be further refined by additional NN layers or even have its loss calculated such that loss is calculated for each time step.

With this structure, an LSTM can regulate the forward propagated information enough for the RNN principle to be useful, though it should be noted that LSTM does not remove the problem with vanishing/exploding gradient through time.

LSTM's architecture is however quite involved to the point where an alternate version of it has been suggested. This version is known as the *Gated Recurrent Unit* (GRU) [43]. It uses the same cell principle as the LSTM with gates and arithmetic operations, but differs mainly by not using a separate cell state and instead incorporates the cell state information into the hidden state. In addition, it only uses two gates: The *update gate* which is a combination of LSTM's input and forget gates, and the *reset gate* which serves to emphasize or reduce information.

Since the GRU has fewer gates and in turn less weights, biases, and activation function computations, it is a more light-weight alternative to LSTM. LSTM is still sometimes preferred over GRU as it with more complexity is able to capture dependencies across longer time distances. Thereby, none of the two models are better than the other, so the best practice is to test both models for a given case to see what works best.

5.4 Recurrent Recognition Network Architecture

In this section the architectural implementation of the Recurrent Recognition Network (RRN) will be presented and reasoned for using various papers.

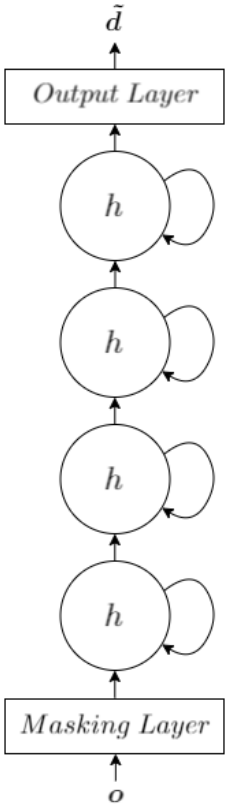


Figure 5.3: *The RRN architecture*

The RRN is implemented as a multilayered LSTM network. It consists of an input layer with a number of neurons equal to $2|\boldsymbol{s}|$ to account for the extra bit needed for representing the unobserved state. In succession of the input layer is a Keras masking layer[2], followed by 4 hidden LSTM cells, and finally a standard NN layer with sigmoid activation as output. An image description of this is given by Figure (5.3). This architecture is inspired by [44, 45], where cases similar to that of IntMed is considered.

The first fundamental choice made for the architecture is the choice of LSTM over GRU. This choice is a bit ambiguous given the inspirational sources. GRU could arguably be able to adequately capture the time step dependencies of the question process, even with its simpler architecture. Since [46] seem to suggest that LSTM is more often applied to diagnostic time dependency cases, and because the RRN should be able to handle really long sequences with many questions, LSTM is chosen as the initial RNN type for the RRN.

[2]This will be explained in the next section

Usage of GRU in the RRN case has not been tested throughout the project because other fundamental parts of the implementation required more attention. It is therefore left as future experimentation to see what works for this part of the architecture.

The next choice for the architecture is the number of hidden LSTM cells. Initially the RRN was constructed with 2 cells which is described in literature as being the optimal number of cells for diagnostic LSTMs [44, 45, 47]. Through experimentation on early versions of the RRN it was however discovered that more trainable parameters are necessary[3]. Because of this the number of cells was increased to 4, allowing for a large number of parameters.

The 4 LSTM cells are implemented with *dropout* which is an NN regularization technique that drops some of the weights randomly during training by multiplying them with 0. This forces the remaining weights to be shaped to solve the problem, hence fitting them to be more independent of other connections. This reduces the chance of overfitting.

Dropout can for an LSTM network be applied to each LSTM cell's input weights, its recurrent weights, or both. The RRN applies dropout to the input of each LSTM cell, following the example of [44, 45] once again. It can be argued that recurrent dropout might be necessary given that 4 cells results in a quite large number of total parameters.

The next architectural choice is the number of *units* for each cell. This number is the dimension of the hidden state h_t and the cell state c_t described by Figure (5.2). The equal size between them is necessary because their elements on corresponding indices will be pairwise added and multiplied according to Equation (5.11). Upon defining an LSTM cell, the dimensionality of the network input $\boldsymbol{o}$ is merged with *units*, which is then used to define the LSTM cell's number of trainable parameters:

$$|\boldsymbol{W}_G| = 4(input \cdot units + units^2 + \boldsymbol{B}) \tag{5.14}$$

Where *input* is the dimension of the input given to the cell, and $\boldsymbol{W}_G$ is the collective weight set for all gates of a cell. It follows that the number defined for *units* has high impact on the number of model parameters, especially when 4 cells is used.

To choose an appropriate number of *units*, [45] is used as a reference. In that paper the network is tuned to predict 128 out of 429 possible diseases, based on 13 real valued input features. For this purpose, they obtain the best results with $units = 128$ and a dropout of 0.5.

The recurrent network of the reference paper only has to learn prediction of a subset of the diseases, whereas the RRN must be able to make predictions for all diseases. Through experimentation with different RRN sizes, it was discovered that setting the number of units to be quite high is necessary. Therefore, the number of units is set as 300 to give the network enough power to capture the BN2O relations.

It seems there is a tendency for LSTMs developed for diagnostic purposes to choose *units* to be a number which can be factorized into a power of 2 [45, 47], possibly indicating that choosing 256 would be the better option. The optimal number is however decisively found through further experimentation.

[3] One such experiment is depicted in Figure (A.2) in Appendix A

5.5 Recurrent Recognition Network Sampling

In this section the sampling and prediction procedures of the RRN will be explained. Firstly, we consider the number of inputs and outputs to be supported by the RRN, which will also allow for the definition of the wanted posterior to be produced. Using this information, the samples to train the network with are defined such that they may be used to obtain the wanted posterior. Included in this are a definition of the time step semantics related to the RRN's inputs and outputs.

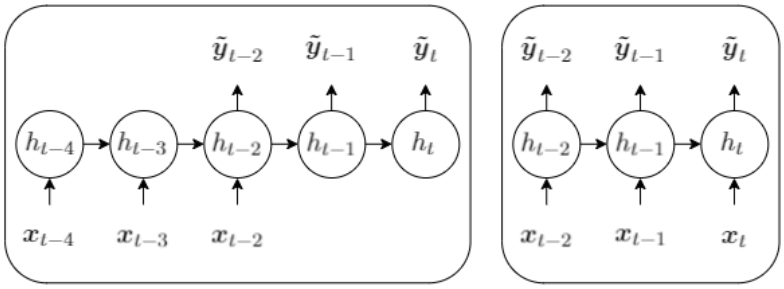


Figure 5.4: *Many-to-many type RNN (left) and equally many-to-many RNN (right). Source of inspiration: [38].*

One of the main advantages of recurrent networks are their ability to manage input and output sequences of varying sizes. Figure (5.4) depicts two input-output sequence size relations that are useful to consider for the RRN. Both of the general RRNs depicted get many inputs and produces many outputs, with the only difference being that the 'many' of the right model is a fixed size.

The RRN must be able to infer disease probabilities based on each t where $1 \leq t \leq T$, and T is defined as the finite consultation length of a given consultation. The RRNs input is thus given as a sequence of observed symptoms and its output is a diagnosis related to that sequence. The wanted posterior distribution hence take form as:

$$P(\boldsymbol{d}|\boldsymbol{s}) \approx P(\tilde{\boldsymbol{d}}_t \mid \boldsymbol{o}_1, \ldots, \boldsymbol{o}_t) \tag{5.15}$$

Even though the output should be one single diagnosis, it is important that the RRN makes a prediction for every reported symptom observation, otherwise the system's assistance aspect is lost. Because this is the case it should be implemented as an equally many-to-many RNN as per depicted in the right side of Figure (5.4). The loss must be calculated per time step such that the RRN is optimized to predict the most probable diagnosis for any t in $1, \ldots, T$.

To learn prediction of the wanted disease posterior, the RRN requires samples of medical consultations where the questions act as the time steps. Each time step will display all symptoms as unobserved, except for the symptom which receives evidence at that time step. This symptom will instead be labeled as positive or negative depending on what state it was sampled as.

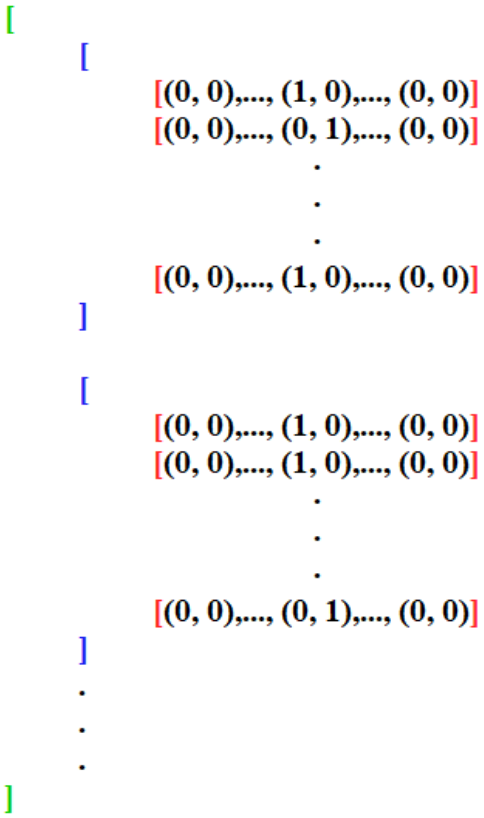


Figure 5.5: *Data structure of the vector of consultation observation vector sequences used during training by the RRN. The green brackets denote the vector that holds all the sampled consultations. The blue brackets denote a sampled consultation containing T time steps. The red brackets denote the vector of a time step which has a tuple representation for each symptom. In each time step vector, only one symptom is sampled as either positive or negative.*

Data structure-wise this resembles Figure (5.5) where the green brackets are the consultation sample dimension, the blue brackets are the time step dimension, and the red brackets are the symptom dimension with each symptom being 2-dimensional due to their two-bit tuple representation. The two consultations depicted may contain a different number of time steps, and as depicted by the observed symptom tuple shown in the middle of each time step[4] the evidence given may differ. The first consultation sample starts of with a positive symptom, then a negative, and in the end a positive. The second starts off with two positive symptoms revealed in succession and ends with a negative. The depiction may give the impression that the same symptom can be revealed multiple times throughout the time steps, but this is not the case. With each time step, a new symptom is observed. Each consultation sample depicted by the blue brackets are defined as ζ in Equation (5.2).

[4]'The middle' refers to the Figure depiction, and not (necessarily) the actual middle index position.

The content of ζ is the sequence of $\boldsymbol{o}_t$ time steps depicted with the red brackets. These now only contain evidence on one symptom, rather than all evidence accumulated from time steps $1, \ldots, t-1$. The vector of consultation observation vector sequences, depicted by the green brackets, can now be defined as:

$$\boldsymbol{\zeta} = [\zeta^{(1)}, \ldots, \zeta^{(M)}] \tag{5.16}$$

where M is the desired number of consultation samples.

The training process also requires disease vectors to serve as target values for each of the consultation samples. The time step observation vectors should have corresponding disease target vectors that caused them such that each time step has an associated target.

The associated target for each $\boldsymbol{o}_t$ in a sequence given by ζ should be the same because a consultation has the same underlying diagnosis. Data structure-wise the vector of consultation targets to be defined is thus almost identical to what is depicted by Figure (5.5). The only difference is that each red time step bracket is now an instance of $\hat{\boldsymbol{d}} = \{0,1\}^{|\boldsymbol{d}|}$, where each index represents a disease of the BN2O being present (when set as 1) or absent (when set as 0).

Formally, let $\hat{\boldsymbol{d}}^{(m)}$ be the reference diagnosis that caused the sequence of observations given by $\zeta^{(m)}$, then the consultation reference diagnosis sequence of $\zeta^{(m)}$, denoted $\beta^{(m)}$, is defined as:

$$\beta^{(m)} = [\hat{\boldsymbol{d}}_1^{(m)}, \ldots, \hat{\boldsymbol{d}}_T^{(m)}] \tag{5.17}$$

such that $\hat{\boldsymbol{d}}^{(m)}$ remains the same throughout the sequence.

With this definition in place, the vector of consultation reference diagnosis sequences to be created is defined as:

$$\boldsymbol{\beta} = [\beta^{(1)}, \ldots, \beta^{(M)}] \tag{5.18}$$

where M is identical to that of Equation (5.16) such that for $1 \leq m \leq M$ we have that the underlying explanation of $\zeta^{(m)}$ is $\beta^{(m)}$.

The generation process of the sample sets $\boldsymbol{\zeta}$ and $\boldsymbol{\beta}$ happens as follows: Firstly, the reference diagnosis $\hat{\boldsymbol{d}}^{(m)}$ is sampled the exact same way as the non-recurrent RNs described in Section 4.2.1.

Secondly, the consultation question sequence $\zeta^{(m)}$ is to be sampled. This happens by using Equation (4.9), but rather than inserting each sampled symptom into its respective index in an observation vector, it is instead concatenated to a sequence of sampled symptoms $\phi^{(m)} = (o_1^{(i)}, \ldots, o_T^{(i)})$.

When all sampled symptom observations have been concatenated to $\phi^{(m)}$, its order is then randomized which ensures that evidence is given in different orders for each sample. This is important for two main reasons. Firstly, it mimics how patients may give evidence in different orders even though they have the same diagnosis.

Secondly, if the sequence is not randomized, its order will programming-wise depend on the order of the symptoms given by the BN2O network[5]. This would result in all samples consistently having some symptoms appearing before others which is undesirable.

The sequence of $\phi^{(m)}$ is then processed where each sampled symptom is handled as follows: Firstly, an observation vector of only unobserved values defined by Equation (5.1) is created. The sampled symptom observation is then inserted at i such that $\boldsymbol{o}_0^{(i)} = o_t^{(i)}$. The vector is at this point on the form depicted by the red bracket vectors in Figure (5.5). This vector is then concatenated to the consultation question sequence $\zeta^{(m)}$.

Upon this concatenation, an instance of $\hat{\boldsymbol{d}}^{(m)}$ is also concatenated to the corresponding consultation reference diagnosis sequence $\beta^{(m)}$. This entire process repeats until $\phi^{(m)}$ has been processed, whereafter both $\zeta^{(m)}$ and $\beta^{(m)}$ are concatenated to their respective consultation sample vectors given by Equations (5.16) and (5.18).

When the samples have just been made, they are incompatible as network input. This is because the number of time steps of the consultations are different from one another, making the size of the dimension containing them undefinable.

To counteract this, the samples are *padded* with values that will be ignored during training. These values are vectors containing purely (-1, -1) tuple values for the observation vectors, and plain -1 values for the reference diagnoses. The consultation with most time steps will dictate how many vectors of ignorable values that should be concatenated to each sample. Each consultation sample of lesser length than the longest will be padded with a number of ignorable vectors equal to the difference between its own length and the longest. The padding is concatenated to the end of the consultation sequence.

After the padding, all samples look as per depicted by Figure (5.6) such that the time step dimension now has a fixed size. To finalize their creation, the consultation observation vector time steps have their dimensionality reduced by one which removes the tuple status of the symptom representation.

When the RRN then receives the input during training, it has a special Keras masking layer that transform '-1' values into 'False' logical values which will be ignored during training. The value '-1' is chosen rather than '0' to prevent any confusion between the actual evidence representation given by Equation (4.11), and the ignorable values.

Finally, the way the RRN should receive input during prediction has the following semantics: Let ζ_t^{past} denote the vector of past asked questions prior to t, and let ζ_t^{pred} denote the vector of present network input to be used for prediction t, then ζ_t^{past} is defined as:

$$\zeta_t^{past} = [\boldsymbol{o}_1, \ldots, \boldsymbol{o}_{t-1}] \tag{5.19}$$

and $\boldsymbol{\zeta}_t^{pred}$ is then defined as:

$$\zeta_t^{pred} = \zeta_t^{past} :: \boldsymbol{o}_t \tag{5.20}$$

where both of these vectors resemble the blue brackets of Figure (5.5).

[5]For a visual intuition of this, consider the number sequence of the symptom nodes in Figure (3.1)

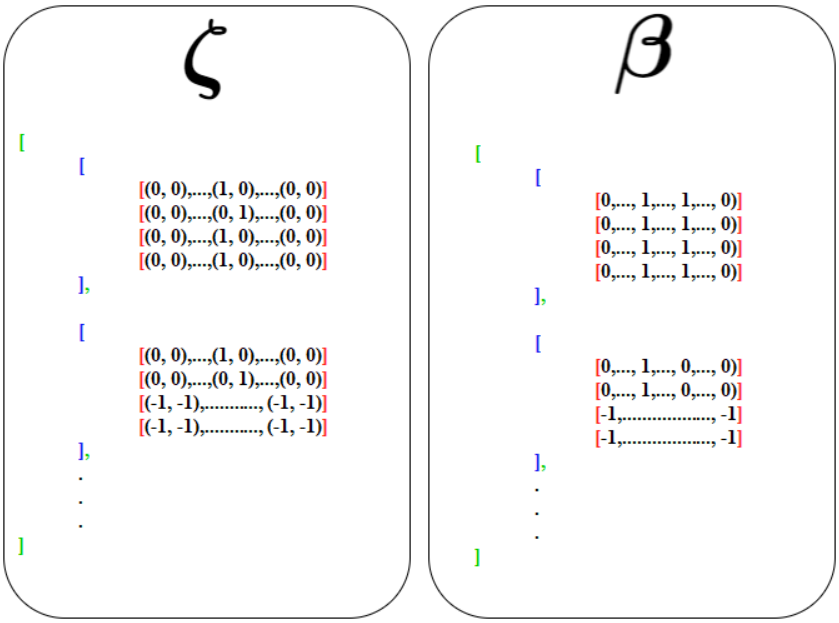


Figure 5.6: *Data structure of the vector containing consultation observation vector sequences (left) and the corresponding vector of consultation reference diagnosis sequences (right) where both have been padded. These are the code representations of $\boldsymbol{\zeta}$ and $\boldsymbol{\beta}$, as they appear upon completion of the sample creation. In this example, the first consultation is the longest with 4 time steps, meaning that the other consultation with only 2 time steps must have 2 vectors of ignorable values concatenated.*
Notice that the two consultation reference diagnoses depicted in $\boldsymbol{\beta}$ (shown with blue brackets) are different, with the first having an additional disease compared to the other. This has caused the first consultation to be longest, as two present diseases will have a larger collective symptom set to be questioned. This has been depicted to show that there is a correlation between the sampled consultation length, and the number of underlying diseases present within it.

It follows that each prediction is based on all previous questions with the addition of the new information from $\boldsymbol{o}_t$. The RRN prediction can thus be interpreted as an analysis of evidence trails. It outputs a number of disease predictions equal to the size of the input evidence trail. Formally, the disease prediction output β_t^{pred} is defined as:

$$\beta_t^{pred} = [\tilde{\boldsymbol{d}}_1, \ldots, \tilde{\boldsymbol{d}}_t] \tag{5.21}$$

and so the desired approximated probability of the disease diagnosis after t questions is finally obtained:

$$P(\boldsymbol{d}|\boldsymbol{s}) \approx P(\tilde{\boldsymbol{d}}_t \,|\, \zeta_t^{pred}) \tag{5.22}$$

which matches the wanted disease probability posterior shown in Equation (5.15).

6 Experiments

In this chapter the results of the RRN training and experimentation will be presented.

Firstly, the experimental setup will be explained. This includes the introduction of a new metric for NN calibration measurement, a special training technique deployed for the RRN, and metrics used for monitoring the training phase.

Then two RRNs which will be the main entities of the chapter are presented, and their training results are analyzed.

Lastly, the experimental results of the two RRNs will be analyzed, where we will subsequently see an alternate implementation based on the reflections of the analysis.

6.1 Experimental Setup

6.1.1 Expected Calibration Error

The experimentation with the BRN and SDRN showed how these networks tended to underestimate their output. This was concluded from observing that their output posteriors for the underlying diseases were low, while the ranking of the same diseases were high. This is a sign of poor network calibration, where the networks in this case are underconfident. They assign very low probability posteriors to all diseases, even the ones predicted to be most correct given their high ranking.

To investigate this further, a new metric known as *Expected Calibration Error* (ECE) is introduced [48, 49]. This metric shows the average difference between the proportion of an NN's correctly predicted elements (the accuracy), and the probabilities estimated for those elements (the confidence).

The metric is reported as a single number and is calculated with the same type of samples the network uses during its training, where each sample have an input and an associated target value. The way ECE is calculated in the original sources differs slightly from the way it is calculated here because it must be fitted to a multi-label classification problem.

The calculation of ECE for the RRN is carried out as follows: Firstly, M *bins* are defined. These are subsets whose collective union form the set of all predictions. Each bin, denoted B_m, holds the predictions whose predicted probabilities are within the range $(\frac{m-1}{M}, \frac{m}{M}]$. The accuracy of B_m is then calculated as the average number of correct predictions by the members of the bin:

$$acc(B_m) = \frac{1}{|B_m|} \sum_{j \in B_m} \mathbf{1}(\hat{d}_j = \tilde{d}_j) \tag{6.1}$$

where $\hat{d}_j$ is the target and $\tilde{d}_j$ is the predicted label. The confidence of B_m is the average of the posterior sum of all predicted labels found within B_m:

$$conf(B_m) = \frac{1}{|B_m|} \sum_{j \in B_m} Q(\tilde{d}_j) \tag{6.2}$$

having defined acc and $conf$, the ECE score of an RRN is obtained as the sum of average difference between these two values for each bin:

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n \cdot T_{sum} \cdot |\tilde{\boldsymbol{d}}|} \Big| acc(B_m) - conf(B_m) \Big| \tag{6.3}$$

where n is the sample batch size used for the calculation, and T_{sum} is the total number of time steps for the sample batch. The way the calculation is adapted to the multi-label scenario is by considering each predicted disease of the BN2O to be counted as one individual prediction. This is the reason why the normalization constant of the bins is set as $n \cdot T_{sum} \cdot |\tilde{\boldsymbol{d}}|$. The sample batch of size n has in total T_{sum} time steps, with $|\tilde{\boldsymbol{d}}|$ predicted diseases, which makes the total number of disease predictions equal to the product of these variables. It follows that the bins collectively contain this number of predicted diseases within them.

In the calculation of ECE reported in the upcoming section, the number of bins M are set as 10. The number of samples n are set to be 100 and are chosen to be fairly low in order to accommodate for the large number of predictions resulting from the multiplication with T_{sum} and $|\tilde{\boldsymbol{d}}|$. Finally, the way equality is decided between the predicted disease and the target disease in Equation (6.1) is not determined by strict equality of $\hat{d}_j$ and $\tilde{d}_j$, but are instead decided according to:

$$(\hat{d}_j = \tilde{d}_j) = \begin{cases} True & Q(\tilde{d}_j) \geq 0.5 \wedge \hat{d}_j = 1 \\ True & Q(\tilde{d}_j) < 0.5 \wedge \hat{d}_j = 0 \\ False & otherwise \end{cases} \tag{6.4}$$

which is a more meaningful definition for accuracy given that the RRN should output a relevant posterior probability rather than plain '1' and '0' values.

6.1.2 Recognition Network Sharpening

In Section 4.3 it was shown how the initial RN implementations mostly estimated posteriors within the ranges $[0, 0.1]$ and $[0.9, 1]$. The section also showed how the disease ranking was non-proportional with the posteriors that created it which is a sign of poor network calibration.

In an attempt to counteract this, a calibration method known as *temperature scaling* was deployed [48], where a parameter was trained and used to normalize the RNs' outputs, before they were squished through the output sigmoid activation function. After deploying this method, the networks did unfortunately still show results similar to those of Figures (4.2) and (4.3). This is showcased by Figure (A.1) in Appendix A.

Through experimentation with RNs trained on a subset of the BN2O with 3 diseases, it was however discovered that RNs in general learn how diseases often are absent rather than present. Essentially, an RN trained on the full set of 514 diseases, will have 1 to 5 disease targets for every sample, and thus there are 509 targets set as '0' for every sample. Because of this the networks will learn that a disease usually should be given an output close to 0. The experiments with the subset trained RNs can be found in Figure (A.4) in Appendix A.

To counteract this, a technique which we denote as *RN sharpening* is applied. This training technique is derived from the subset network experimentation. It is thus not based on literature, but is purely based on experimentation with the RN concept. An example of its effect is shown in Figure (A.3) in Appendix A.

RN sharpening is carried out by initially training the given RN with samples containing a very large number of present diseases. The network will have its weights fit towards a high possibility of each disease being present. Instead of training the network with a fixed number of diseases set as present, RN sharpening iteratively decrements the total number of diseases to be set as present. This is done to prevent the network from being fitted too much to the idea of all diseases being present.

RN sharpening are done through a sequence of training rounds. The number of training rounds is equal to $|\boldsymbol{d}|$. Every training round uses 500 samples which the network is trained upon for two epochs. The number of present diseases is determined according to the following description: Let P_{rounds} be the number of already conducted rounds such that $P_{rounds} \leq |\boldsymbol{d}|$, then the number of diseases set as present in a given training round, $d^{+}_{sharpen}$, is calculated by:

$$d^{+}_{sharpen} = |\boldsymbol{d}| - P_{rounds}$$

When the RN has undergone the sharpening process, it can be trained with larger sample sizes with few diseases present. This will fit it towards estimating lower probabilities for most diseases as intended, but it will now output larger values on average.

6.1.3 Training, Validation, and Testing Metrics

The network training is monitored through loss, precision, and recall. The loss is reported as the average $loss_T$ as it was defined in Equation (5.5), with the only addition of being averaged over all consultation samples.

The precision and recall are calculated for the top-k predictions, with a threshold of 0.5. Formally, the metrics with these constraints are defined as follows: Let $\hat{d}^+$ be the set of diseases that were sampled as present in a given consultation such that they are the diseases set as present in the target vector for each time step, and let $RN_{topk} = \{\tilde{d}_j^{(1)}, \ldots, \tilde{d}_j^{(k)}\}$ be the top-k disease predictions by the RRN for that time step, then precision in top-k with a threshold of 0.5 is calculated by:

$$P@k_{0.5} = \frac{\sum_{i=1}^{k} \mathbf{1}\left(\tilde{d}_j^{(i)} \in \hat{d}^+ \bigwedge Q(\tilde{d}_j^{(i)}) \geq 0.5\right)}{\left[\sum_{i=1}^{k} \mathbf{1}\left(\tilde{d}_j^{(i)} \in \hat{d}^+ \bigwedge Q(\tilde{d}_j^{(i)}) \geq 0.5\right)\right] + \left[\sum_{i=1}^{k} \mathbf{1}\left(\tilde{d}_j^{(i)} \notin \hat{d}^+ \bigwedge Q(\tilde{d}_j^{(i)}) \geq 0.5\right)\right]} \tag{6.5}$$

and recall with the same constraints are calculated by:

$$R@k_{0.5} = \frac{\sum_{i=1}^{k} \mathbf{1}\left(\tilde{d}_j^{(i)} \in \hat{d}^+ \bigwedge Q(\tilde{d}_j^{(i)}) \geq 0.5\right)}{|\hat{d}^+|} \tag{6.6}$$

6.2 Recurrent Recognition Network Training, Validation, and Testing

In this section the training phase of the RRN will be covered. This includes an introduction of an additional RRN version to be included in the experimentation.

The RRN was trained with two sessions of RN sharpening, resulting in approximately $5 \cdot 10^5$ consultation samples and approximately 1000 epochs. To fit it to an appropriate number of diseases, it was subsequently trained with 10^5 sampled consultations with a uniform distribution of targets with 1 to 5 diseases. The network was trained over the course of 10 training rounds with 10^4 samples per round. Each round ran 10 epochs with a batch size of 1.

In a addition to the main RRN, henceforth denoted RRN_{main}, a smaller version of it known as RRN_{sub} was also trained. This network was trained using a subset version of the BN2O. This BN2O contains 50 diseases, 125 symptoms, and 294 relations between them. RRN_{sub} have the exact same internal structure as RRN_{main} with the only exception being the in- and output sizes. It was sharpened through two sharpening sessions the same way as RRN_{main} and trained using the same number of samples and epochs. Given that its number of diseases is smaller, its total number of sharpening samples amounts to $5 \cdot 10^4$.

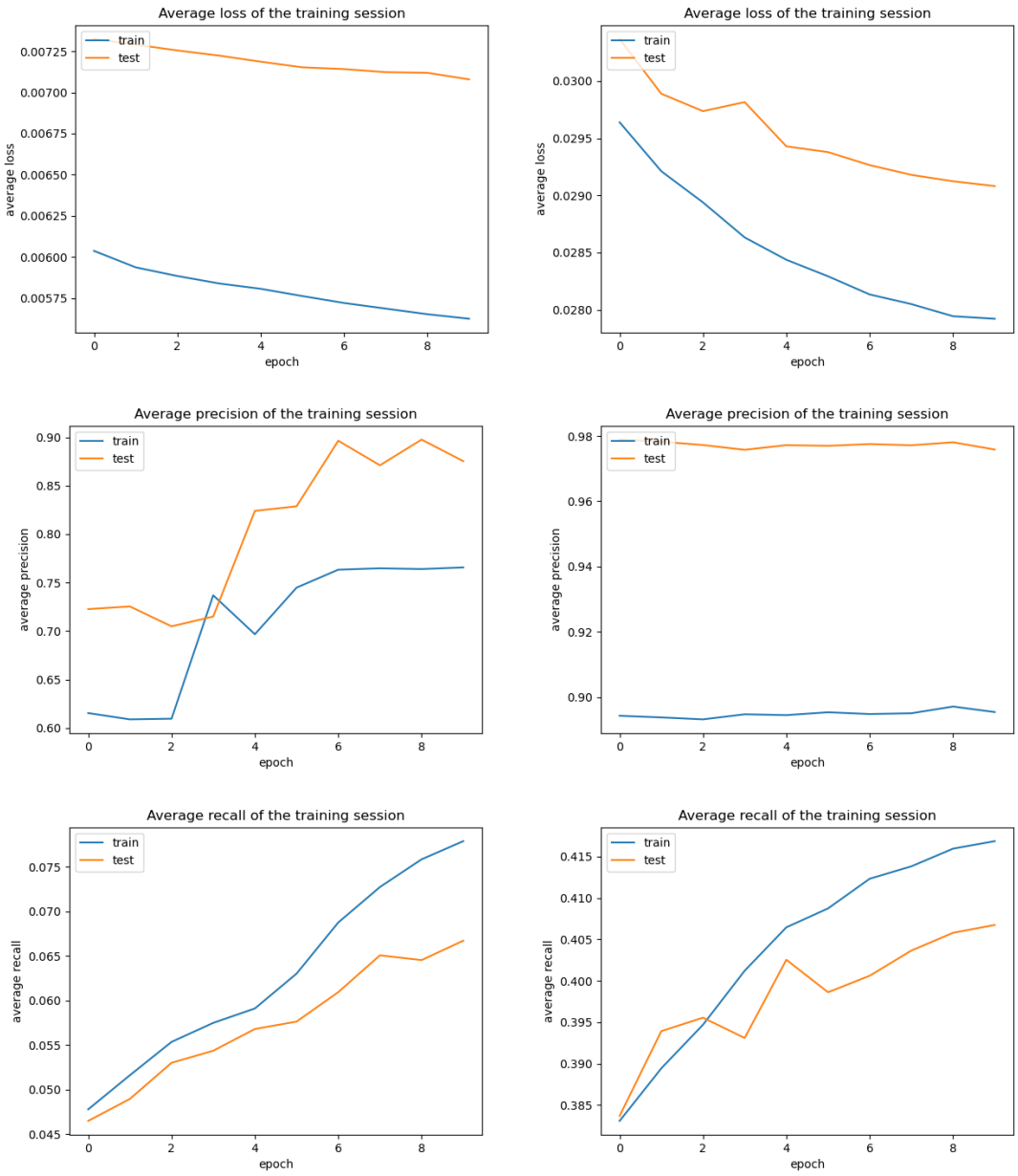


Figure 6.1: *The average training results of RRN_{main} (left column) and RRN_{sub} (right column). The average is taken over 10 training rounds where each round has trained the given RRN with 10^4 samples for 10 epochs.*
'Test' in this case refers to a validation sample set of samples used to validate the given RRN for each epoch. This set has a size of 1000 samples and has a uniform distribution of targets with 1 to 5 diseases.
Prior to this training, two sessions of RN sharpening have been conducted, hence the low decrease in loss.

Model	*loss*	$P@k_{0.5}$	$R@k_{0.5}$
RRN_{main}	**0.58%**	90.05%	8.07%
RRN_{sub}	3.15%	**96.54%**	32.5%
DLIRRN	4%	86.83%	**56.78%**

Table 6.1: *Test sample set prediction results of all RRNs. Each test set used for each model contained 1000 sampled consultations with a random number of diseases from 1 to 5 in the targets.*

Figure (6.1) shows the average results of all the training monitoring metrics calculated for both the training and validation samples during training of RRN_{main} and RRN_{sub}. We start by considering the loss.

The loss of RRN_{main} is due to the sharpening sessions already low compared to RRN_{sub}, undoubtedly due to the difference in the number diseases which determines the number of samples and epochs of the sharpening. Both RRNs have a downward going loss, but the reduction per epoch is low by several orders of magnitude. The downward going loss means that the RRNs can be trained even further and that the models consistently converge towards 0 which is the most important point of the training.

We now consider $P@k_{0.5}$ where $k = 10$.
For $P@k_{0.5}$ there is a major difference between the development of the curves for the two RRNs. RRN_{main} gains a considerable increase from the 3rd to the 4th epoch whereafter it moves into the $[0.85, 0.9]$ range, in comparison to RRN_{sub} which has no change in this metric for all training rounds. This is especially interesting given that the loss of RRN_{main} decreases less than that of RRN_{sub}. Both RRNs have better $P@k_{0.5}$ on the validation data than the training data with about 10% difference on average, with the only exception being RRN_{main} at the 3rd epoch.

Finally, we consider $R@k_{0.5}$ where $k = 10$.
$R@k_{0.5}$ is shown for both RRNs to be increasing, though only in both cases with about 0.3%. For RRN_{main} the numbers for this metric are low. RRN_{sub} achieves a much better result, even though it is still considered low from being below 50%.

In addition to those results, the two RRNs were also tested with separate testing sample sets of 1000 consultations[1]. Each testing consultation had a random number of diseases from 1 to 5, to represent how real data would not have a uniform distribution of diseases. The results of these tests are shown in Table (6.1).

[1]Not to be confused with the validation data with the same quantity.

To summarize the results of the training and testing, the two RRNs differ the most when it comes to loss and $R@k_{0.5}$. Both RRNs also have a significant different value for precision and recall, despite both metrics being calculated based on the number of true positives which is the numerator in both Equations (6.5) and (6.6).

This is likely the case because the metrics are calculated per time step. Early time steps are likely to output very low posteriors which is understandable given that little evidence have been given at this point. Because of this, the numerator of $R@k_{0.5}$ in Equation (6.6) is close to 0 on average, which will in turn yield 0 on average for the metric.

$P@k_{0.5}$ is instead conditioned on false positives calculated on the right-hand side of the denominator in Equation (6.5). This number is basically 0 if the average predictions yield low posteriors. This means that the calculation of $P@k_{0.5}$ simply need just one disease on average to be predicted in top-10 with a posterior ≥ 0.5 to yield a high result.

It is thus probably more desirable to calculate these two metrics at the end of a training consultation sample, rather than considering all time steps in the average.

Despite of this it can still be seen that RRN_{sub} achieve a significantly better $R@k_{0.5}$. This indicates that the RRN concept is not necessarily flawed given the poor $R@k_{0.5}$ of RRN_{main}, but may instead simply be a question of scaling.

The last and probably most important point is that the loss progressively falls for both models, despite it happening slowly. This turns the RRN concept's convergence into a matter of scaling samples and/or epochs.

6.3 Experimental Results

In this section, the results of the experimentation with RRN_{main} and RRN_{sub} will be presented and analyzed. Based on the reflections of this analysis, a final extra version of the RRN will be introduced, and its results will be shown.

The suite of experiments to be conducted for the RRNs are the ones already presented in Section 4.3. These experiments are useful to test the RRN concept with, and allows for comparison with the results presented in that section. The experiments will use the same diseases as in the prior section, but with new question sequences.

The idea is to compare the results of the sequential RNs to the recurrent ones, and to compare the results of RRN_{main} and RRN_{sub} to get a grasp of the RRN concept's scalability.

6.3.1 RRN_{main} Single Disease Prediction

We start of by considering the results of RRN_{main} as depicted by Figure (6.2). Firstly, the simulated consultation with one underlying disease is analyzed which is depicted by the left column of diagrams in the Figure.

The predicted posteriors by RRN_{main} in this experiment are lower than what is calculated by Quickscore, but the slopes of the two curves seem to be decently proportional.

Even though the posteriors are lower, it is remarkable that RRN_{main} compared to the RNs of Section (4.3) now progressively increases the posterior output as more positive evidence is given.

This progression can also be seen by the ranking which is shown to progress downward towards a high rank very nicely, in opposition to Quickscore which requires 6 positive symptoms before placing the underlying disease within top-10. Despite the vast posterior difference from Quickscore, RRN_{main} still places the underlying disease as top-1 from the 6th prediction and onwards.

The top-k similarity metrics show a far greater result throughout the consultation than what was achieved by BRN1 and SDRN1 in Figures (4.2) and (4.3). The highest value for the metric achieved by the prior RNs is the lowest value achieved for the metric by RRN_{main} in this 1-disease scenario. The rest of the similarity metrics are however not as impressive, with TK3 being the only other metric that is present.

The average KL-divergence is another significant improvement compared to the RNs of Section (4.3). Where both BRN1 and SDRN1 had their average KL-divergence in the interval $[0.25, 2.25]$, RRN_{main} achieves scores approximately within $[0.075, 0.12]$. It also stays fairly consistent in the range $[0.095, 0.11]$ from 4 to 12 positive symptoms.

6.3.2 RRN_{main} Multi Disease Prediction

Now RRN_{main} is considered in the consultation with 3 underlying diseases, shown in the right column of diagrams in Figure (6.2).

The posterior predictions of this consultation show poor results for RRN_{main}, compared to those produced by BRN3 and SDRN3. It can however be seen that Quickscore does not estimate high probabilities for any of the diseases either, except for one of them after 9 positive symptoms. The result of the posterior estimation dictates the ranking very closely for both Quickscore and RRN_{main}

The top-k similarity is worsened compared to the 1-disease scenario, but not to a large extent. It is also remarkable how TK1 shows that $\frac{9}{10}$ of the diseases are identical after just one positive symptom.

Just like the top-k similarity, the average KL-divergence is not affected much by the poor posterior and ranking results. The range it lies within is however expanded from $[0.075, 0.12]$ to $[0.075, 0.16]$. It also increases rapidly from the 6th positive symptom and onwards, but seen as the increase is low by two orders of magnitude this is insignificant.

To summarize the performance of RRN_{main}, it is evident that inclusion of the temporal order has a high effect on similarity with Quickscore. Both the similarity metrics and the average KL-divergence shows considerably better results compared to BRN and SDRN. In terms of predicted posteriors and ranking of the underlying diseases RRN_{main} shows quite underwhelming results for the 3-disease prediction scenario, and mediocre results for the 1-disease scenario.

For the 1-disease scenario there is once again a mismatch between ranking and posterior volume. This is now further backed up by the achieved ECE score of RRN_{main} which is shown in Table (6.2). The network produces highly miscalibrated probabilities which is made evident by the posterior-ranking relation.

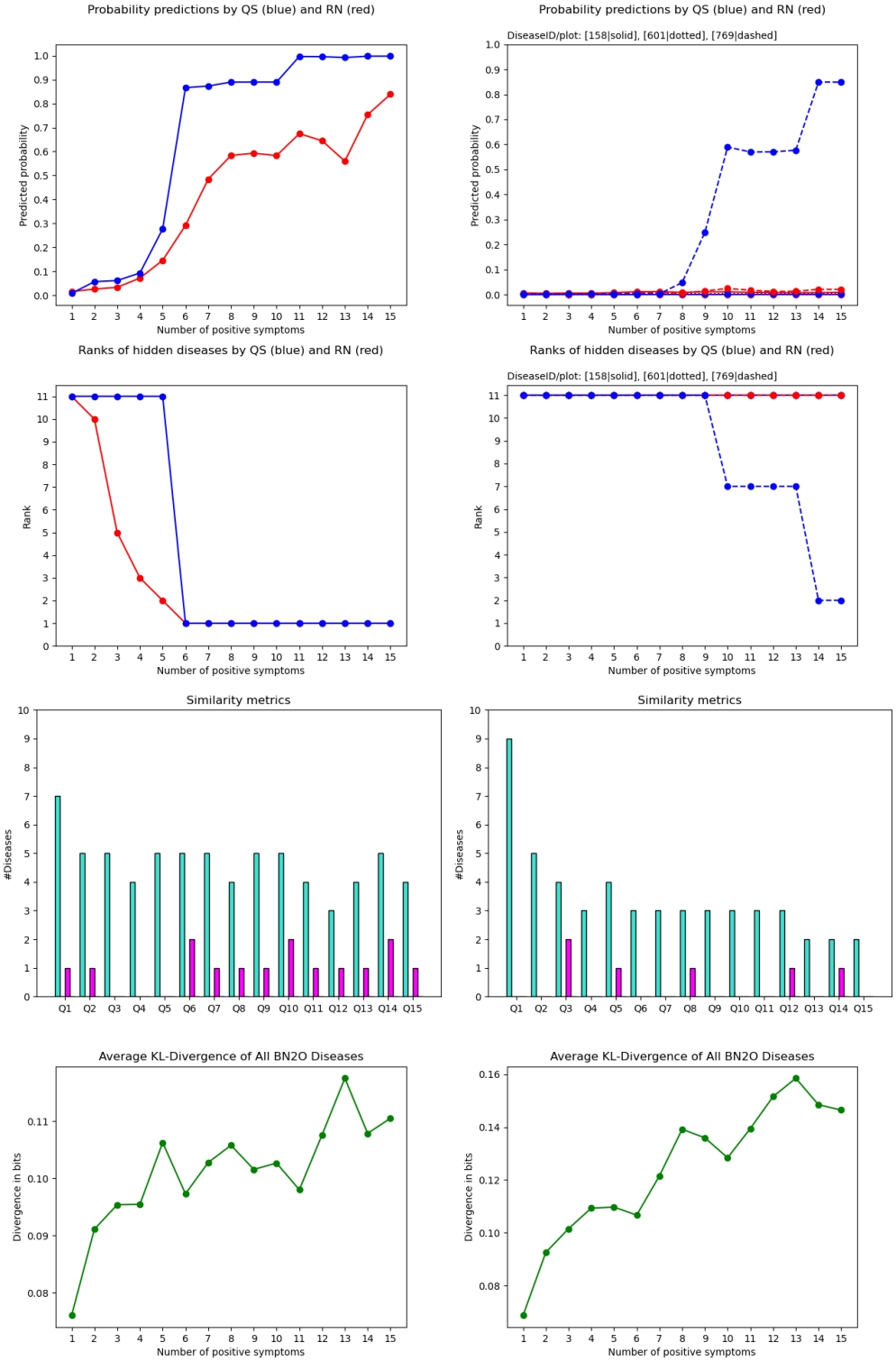


Figure 6.2: *Experimental results of RRN_{main} with 1 (left column) and 3 (right column) underlying diseases. Rows from top to bottom: Predicted posteriors, ranking, the similarity metrics where TK1 is cyan, TK2 is orange, TK3 is purple, and TK4 is green, and the average KL-divergence of all diseases.*

From 1 to 6 positive symptoms, it ranks the underlying disease to be within top-10 out of 514 diseases, but estimates posteriors for it to be within $[0, 0.3]$.

6.3.3 RRN$_{sub}$ Single Disease Prediction

Now the attention is turned towards the results of RRN$_{sub}$ which is shown in Figure (6.3). It should be noted that the BN2O subset which RRN$_{sub}$ is built to predict for is used in these experiments. Firstly the 1-disease scenario is considered, shown by the left column of diagrams the Figure.

The posterior estimation by RRN$_{sub}$ compared to Quickscore in this scenario is remarkably better. The posteriors are increasing steadily as more positive symptoms are revealed, compared to Quickscore where the probability decreases from 2 to 6 positive symptoms given. This is the best result produced in this experiment by any RN so far. The ranking of the disease by RRN$_{sub}$ is only outside top-1 in the initial prediction, whereas Quickscore ranks the disease from top-7 to top-2 in the first half of the consultation.

The similarity metrics achieved are also remarkable. All metrics have almost consistent presence and TK2 can be seen to amount to about 2 on average. TK1 shows the highest average seen yet, with most predictions having 6 diseases in the intersection between the two top-10 sets.

The average KL-divergence is shown to be low. There is a spike from 3 to 6 positive symptoms, which is undoubtedly because of the higher posterior predicted by RRN$_{sub}$. This spike amounts to about 0.13 which is significant in the plotting, but insignificant overall.

6.3.4 RRN$_{sub}$ Multi Disease Prediction

Now the the predictions made by RRN$_{sub}$ is considered in the 3-disease scenario, shown in the right column diagrams in Figure (6.3).

Both RRN$_{sub}$ and Quickscore show better performance in the posterior estimation experiment which is most likely due to the BN2O size reduction. RRN$_{sub}$ is able to predict somewhat useful posteriors for at least 1 of the 3 diseases compared to RRN$_{main}$. It is still quite low compared to Quickscore which for the same disease consistently estimates 100% probability and even breaks the scale at the 10th positive symptom. This probability however plummets towards 0% from the 10th to the 13th positive symptom, in opposition to RRN$_{sub}$ which keeps the posterior above 50%.

The ranking by RRN$_{sub}$ is on par with Quickscore in this scenario. The ranking and the posterior outputs are once again a bit off, seen as the diseases shown by the dashed and dotted lines have quite low probabilities, but still achieves great ranking. Especially the disease of the dotted line should not be ranked as 6 or 7 based on a posterior within the range $[0.1, 0.2]$. After 8 positive symptoms it has even lower probability, but is still ranked as the disease with the 8th highest probability.

The similarity metrics are decreased compared to the 1-disease scenario. Not all metrics are represented anymore, with TK4 only being found at the prediction based on question 3. TK2 is also vastly lower on average compared to the 1-disease scenario.

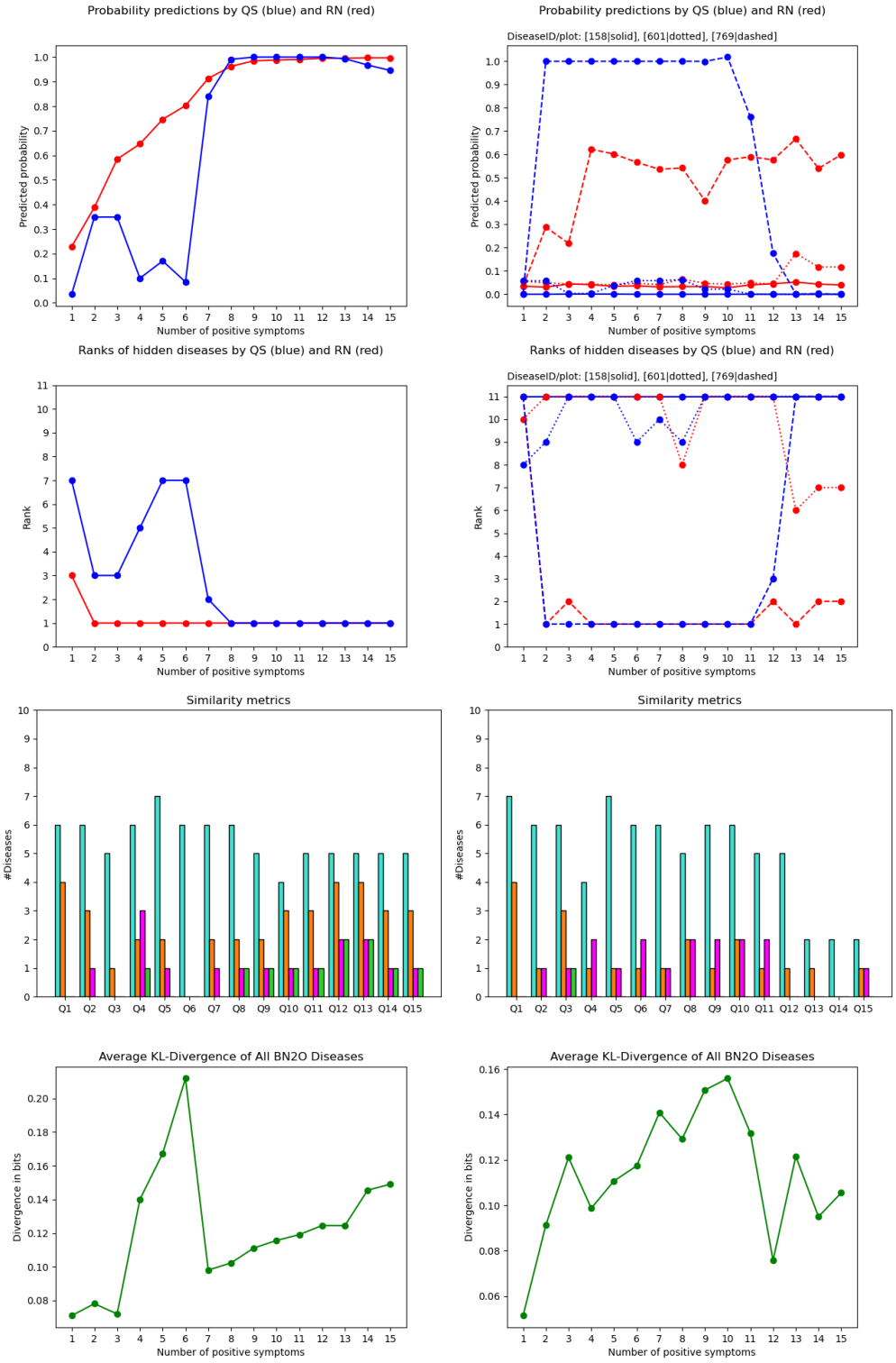


Figure 6.3: *Experimental results of RRN_{sub}. Figure description is the same as Figure (6.2).*

Model	ECE
RRN_{main}	99.11%
RRN_{sub}	**92.04%**
DLIRRN	93.97%

Table 6.2: *Expected Calibration Error of the different RRN types. Lower percentage is better.*

The similarity metrics are lower between the 1- and 3-disease scenarios for RRN_{sub}, just like it was the case with RRN_{main}.

The average KL-divergence per disease is initially on the lowest number seen yet, which is approximately 0.05. The graph is very volatile towards the end of the consultation. This is most likely due to all diseases being estimated equally low at this point such that the only real difference is given by the dashed line disease whose divergence changes rapidly.

Finally, the ECE score of RRN_{sub} can be seen in Table (6.2) to be a bit lower than that of RRN_{main}, but still quite high. The effect of this is especially shown in the 3-disease scenario, given the mismatch between the high rankings and the low probabilities. It is essentially also shown in the 1-disease scenario, where the underlying disease is top-1 in $\frac{14}{15}$ of the predictions, but has a probability within $[0.9, 1]$ in only $\frac{9}{15}$ predictions.

6.3.5 Recurrent Recognition Network Concept Evaluation

Now that the experimental results of both RRN implementations have been evaluated it is possible to assess the RRN concept as a whole. RRN_{main} which is built to predict for a triple digit disease number can from its results be seen to produce decent posteriors that resemble those of exact inference, but its calibration and outputs are still quite lacking. The experimentation with RRN_{sub} which has the same network size and is trained the exact same way as RRN_{main}, yielded results that were better.

Both recurrent models showed a much greater resemblance with exact inference than the initial sequential RN implementations. All in all, this clearly indicates that the idea of using a recurrent structure for the task of predicting diseases of a BN2O is useful. The question is how the concept should be implemented to be able to confidently predict any arbitrary number of diseases.

Throughout the thesis it has been subtly hinted that literature ideas did not seem to work for the RRN, especially in terms of network depth and size. Given all the considerations made for the RRN implementation, it is strange that the results are still quite lacking. The explanation for this may be deduced from the results of the experimentation with RRN_{sub}. Since better results were achieved by an RRN built from a subset of the BN2O, the answer of how to make the model confidently predict any diagnosis, is simply a matter of scaling.

This raises the question of how to scale the network. The final structure used for RRN_{main} was researched and experimented with extensively to produce decent results.

Because the results are still lacking, all this points at the baseline of using a large symptom vector to predict a large disease vector. It may be that the recurrent network is simply unable to capture the BN2O relations using the current implementation. It is after all only given is a series of large vectors that hold but one index of useful information. From this, the model is supposed to learn a vast number of relations which may in fact be the problem. The key to a more scalable implementation might be to utilize the BN2O relations themselves for the RRN. Instead of having to learn the relations between all entities of the BN2O network simultaneously, it might be possible for the model to learn them individually.

From these reflections, one last model is now presented and evaluated using the same experimental setup as in this section.

6.3.6 Disease Layer Input Recurrent Recognition Network

The Disease Layer Input Recurrent Recognition Network (DLIRRN) is an RRN implementation which has a different way of receiving input compared to RRN_{main} and RRN_{sub}. Instead of receiving one vector with the information of all symptoms, it instead receives a vector for each disease.

The architecture of DLIRRN is shown in Figure (6.4). The DLIRRN works by using a sequence of small parallel LSTM cells such that it has one cell per disease of the BN2O it is built to carry out inference for. These cells are defined as follows: Let $h^{(d_j)}$ denote a disease layer input LSTM cell where $J = |\boldsymbol{d}|$ such that $1 \leq j \leq J$, then the input observation vector of $h^{(d_j)}$, denoted $\boldsymbol{o}^{(d_j)}$, is based on the symptom children of d_j, namely s_{d_j}. It follows that the size of $\boldsymbol{o}^{(d_j)}$ is $|s_{d_j}|$. DLIRRN takes J of these vectors as input for each time step t. The output produced of $h^{(d_1)}, \ldots, h^{(d_J)}$ is concatenated and given to the main LSTM cell, simply denoted h, which produces the output $\tilde{\boldsymbol{d}}$.

DLIRRN is trained on the same BN2O subset as RRN_{sub} because the training of a DLIRRN based on the full BN2O turned out to require excessive time. Instead of testing on the full BN2O right away, the idea is to see if DLIRRN performs as well as RRN_{sub}. Better performance than RRN_{sub} might indicate that the model can then be used to scale the RRN concept.

For each $h^{(d_j)}$ the number of *units* is set as 1 such that the number of parameters for any given $h^{(d_j)}$ is directly proportional to the number of symptom children of the d_j it is based on. The number of *units* for h is set as 300.

DLIRRN is trained the exact same way as RRN_{main} and RRN_{sub} with 2 rounds of RN sharpening, and 10^5 samples with a uniform distribution of 1 to 5 diseases in the reference diagnosis targets. The results of DLIRRN on its testing sample set are shown in Table (6.1).

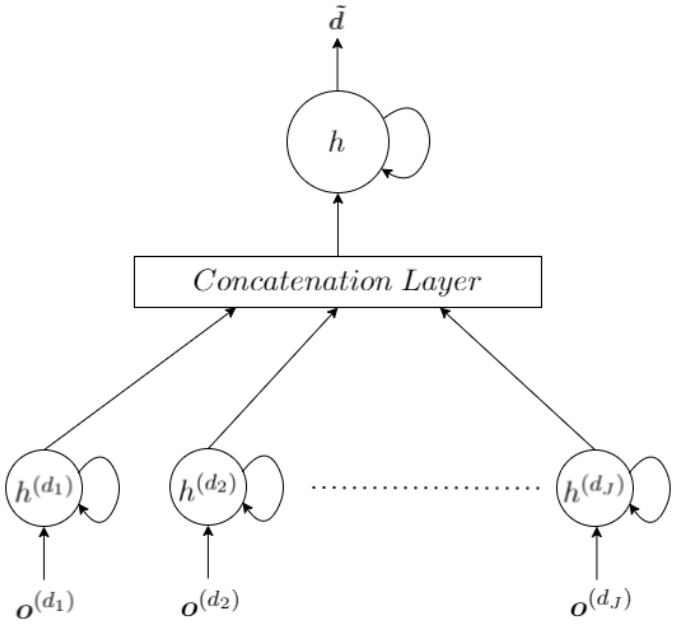


Figure 6.4: *The architecture of the Disease Layer Input Recurrent Recognition Network*

Figure (6.5) shows the experimental results of the DLIRRN model. Throughout the analysis of these experimental results, it is useful to take notice of Figure (6.3), as this will be consistently referenced. Firstly, the 1-disease scenario is considered.

DLIRRN produces a very nice curve of posterior outputs for the underlying disease in this experiment. It starts of by predicting posteriors close to 0 like Quickscore, but then proceeds to make a high spike followed by a nicely rounded curve that slowly progresses. The ranking outperforms Quickscore as well, with the correct diseases being top-1 after the 3rd positive symptom. Its probability outputs are thus indicated to be better than those of RRN_{sub}, but its ranking is a bit worse.

The similarity metrics shows about equal results to those of RRN_{sub}. Where RRN_{sub} has a higher number on average for TK1, DLIRRN seem to have higher presence for the rest of the metrics.

The average KL-divergence of DLIRRN highly resembles that of RRN_{sub} with the spike around 6 positive symptoms. DLIRRN has higher average divergence throughout the consultation, but the difference is minor.

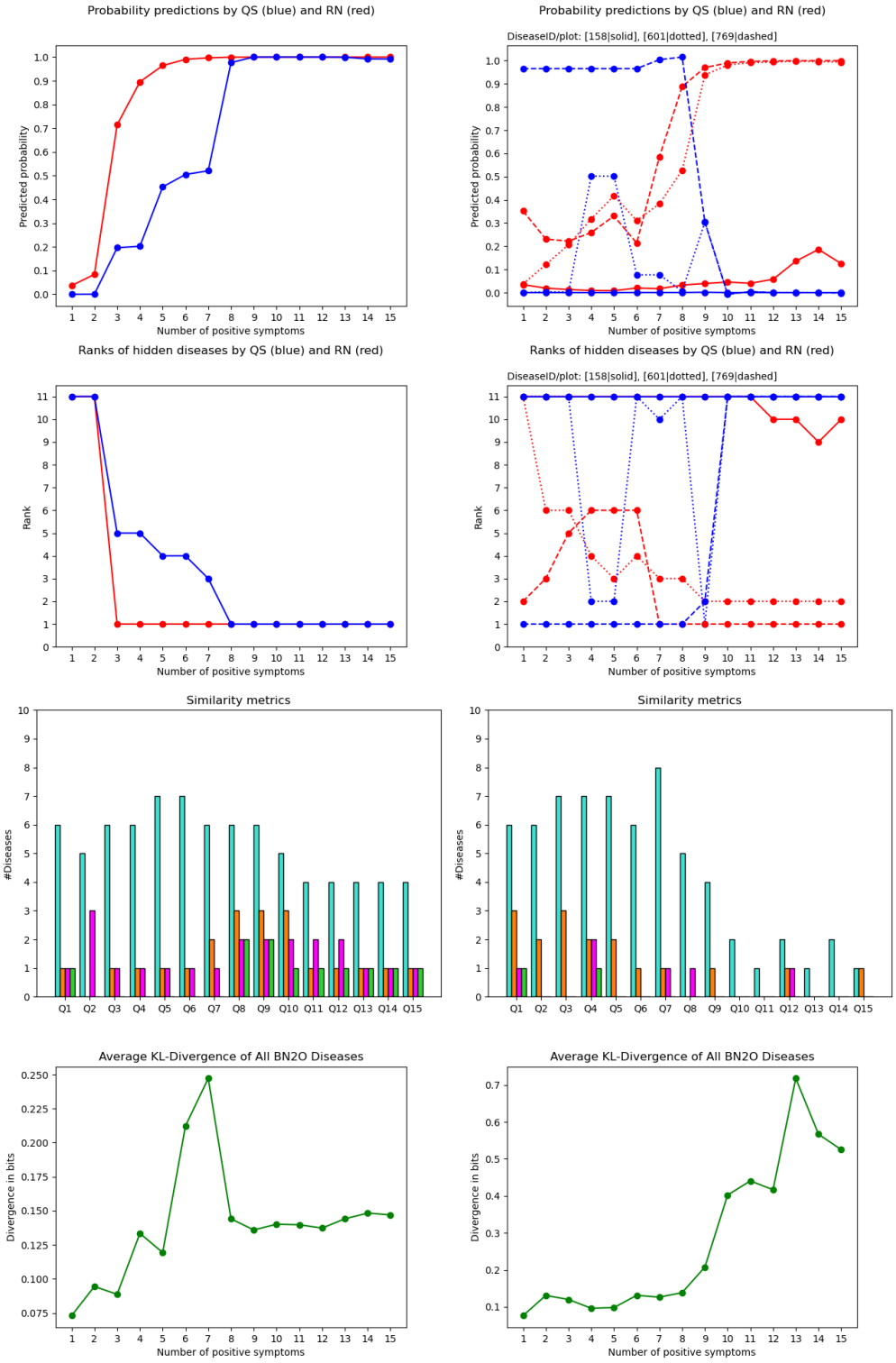


Figure 6.5: *Experimental results of DLIRRN. Figure description is the same as Figure (6.2).*

Now the 3-disease scenario is considered. For the probability prediction, DLIRRN performs about as good as Quickscore, but outperforms RRN_{sub} significantly. The probabilities of $\frac{2}{3}$ diseases steadily rise throughout the consultation, which is the first time this is seen for an RRN model.

The ranking produced by DLIRRN in this scenario shows the first example seen so far where the ranking is somewhat proportional to the predicted posteriors. The dotted and dashed lines of both the probability prediction and ranking plots move according to each other. They are however still a bit mismatched, given that the disease of the dashed line becomes top-1 after 7 positive symptoms, with a posterior of about 0.6. Overall, both plots show a much greater result than shown by RRN_{sub}.

The similarity metrics show a worse result compared to RRN_{sub}, even though the first 7 predictions show a better result. From question 7 and onwards, the similarity decreases to the lowest point seen for any of the recurrent network experimented with so far.

Finally, the average KL-divergence achieved for the 3-disease scenario by DLIRRN shows the highest divergence achieved by an RRN model so far. Like it was concluded for RRN_{sub}, this is likely only due to the difference in posteriors for the underlying diseases, and not because of divergence from all other diseases.

To conclude upon the results of the DLIRRN experimentation, it can be seen that the model achieves the highest multi disease prediction result in comparison with RRN_{main} and RRN_{sub}. This comes at the cost of top-k dissimilarity and higher KL-divergence. DLIRRN seem to be better calibrated than RRN_{sub} by comparison of its predicted ranks and posteriors, but its ECE score shown in Table (6.2) tells a different story.

It should be noted that this implementation is not backed up by literature or any other sources, meaning it has great potential for improvement. This straightforward implementation shows great potential given that higher posterior probabilities is highly desirable. Basing the RRN architecture on the BN2O it is supposed to predict for, is likely the best approach to scaling the RRN concept.

7 Conclusion

We have in this report investigated the appliance of a Recurrent Recognition Network for inference time optimization in a large BN2O network, used by the AI system IntMed.

This investigation was carried out by firstly examining the standard way of handling inference in a BN2O network. In extension, the Quickscore algorithm currently used in the IntMed system which optimizes the BN2O inference process was also examined. Quickscore was shown to have a computation timewise downside which then led the investigation into the topic of Recognition Networks. These were shown to have great potential in terms of providing inference approximation for BN2O networks, but required tweaking which led to an investigation of how to implement them as recurrent neural networks. This sparked the introduction of the Long-Short Term Memory neural network which became the basis of a new recognition network type known as the Recurrent Recognition Network. Two networks of this type were created and trained using consultation samples which included time steps that modelled different kinds of disease scenarios, evidence types, and consultation lengths. After the training, experimentation was conducted with the networks in the form of measuring output posteriors, disease ranking, top-k similarity with Quickscore, and average KL-divergence. This showed that the Recurrent Recognition Network produced great results, but currently requires work on its scalability for larger BN2O networks. Finally, based on the reflection about the scalability requirement, one last recurrent model was created and experimented with, to investigate a possible way of scaling the recurrent recognition network concept to fit larger BN2O networks.

The main objective in this thesis was to test the applicability of a recurrent neural network for real time medical consultations. The most remarkable feature of using a recurrent model compared to a sequential, was the high increase in the overall similarity with exact inference. In this regard the recurrent model was really on par with exact inference and gave an incontestable reason to use that type of model rather than the original sequential version.

As it turned out, the ideas from literature were not directly applicable to this case, seen as most suggestions pointed towards fairly simple and small network structures which turned out to be ineffective. Based on the experiments conducted, especially with the Disease Layer Input Recurrent Recognition Network, it may very well be that the network simply needs redesign before the ideas from literature would show greater effect.

The aspects of the consultation domain in which the recognition network is to be used, comes in the form of evidence being given by patients based on urgency, and the fact that much of the information in a consultation remains unobserved.

Furthermore, the evidence of a consultation is unveiled in a sequence. The modelling of unobserved evidence and the temporal unveiling was included in the implementation, but the urgency aspect was left out.

In summary, it is possible to make a Recurrent Recognition Network model for approximate inference dedicated to a BN2O network like the one used by IntMed, but for it to be on par with the current exact alternative it requires work on its scalability, design, and calibration.

7.1 Future Work

The Recurrent Recognition Network concept has many great possibilities of meaningful future work.

Firstly, there is the further development of the DLIRRN. It is most likely that the structure it dictates is the way to scale the RRN concept. This assumption is based on the experimentation, and the intuition that closer architectural resemblance with the BN2O network should yield better results. Currently, the DLIRRN implementation needs optimization for being able to train on a very large BN2O networks. The first step in this regard is to enable inclusion of a large number of disease layers, as this otherwise greatly slows down its training. This could possibly be done by using simple dense layers rather than LSTM cells for the disease inputs instead. In that case it would have to be determined how to give an arbitrary number of time steps to these. Suffice to say, this model needs more attention.

Secondly, the calibration of the RRN should be addressed given the high ECE scores for all models shown by Table (6.2). In that regard it would also be useful to find a more meaningful calculation adaption for the ECE score. The adaption of considering every disease prediction to be a sample was somewhat arbitrarily chosen. Literature with examples where ECE is calculated for multi-label classifiers is unfortunately scarce. It would perhaps be more meaningful to only consider the posteriors predicted for the target diseases, rather than all diseases. This change would vastly reduce the number in the bins and the size of the normalization constant in Equation (6.3).

Another option would be to apply a new calibration method. Figure (A.1) in Appendix A details how the temperature scaling gave little to no results. Mentioned in [48] are a wide variety of alternate calibration methods which might be better options for the RN case. It is however a prominent possibility that the ECE will decrease vastly when a more scalable implementation is obtained through other means.

Another possible entry for future work is the symptom representation. As it was mentioned Section (5.1), the recurrent structure allows for a convenient representation of unobserved evidence through absent input. This was evidently not used throughout this project, due to the idea of keeping a more expressive symptom representation.

This might however have been the wrong approach, seen as the network input became fairly data heavy, due to the observation vectors being twice the size of the entire symptom set's cardinality. Whether or not simple bit representation for the evidence is better would have to be tested, but it is the most meaningful optimization to the basic RRN concept.

In extension of the future work on symptom representation, it might also be useful to redesign the padding of time steps. If the symptoms were instead represented with bits with '1' for positive and '0' for negative, and the ignorable value was set to '0', the network would learn to emphasize the positive evidence, as it would only read the '1' values. This was initially thought to be detrimental, but it might have the effect of making the network predict higher and more confident posteriors upon receiving positive evidence.

In conjunction with this, the padding could instead be concatenated to the beginning of the sampled consultation sequences. It is possible that the current post-padding strategy has a negative impact on the computation of the relevant input. The propagation of the ignorable vectors that happens after the relevant input has been processed might make the data of the relevant input noisy [50]. The initial wind up of pre-processing the ignorable vectors ensures that the information of the relevant input is not negatively impacted [51]. This also has the interpretation of teaching the network that no more time steps are given after the meaningful inputs have been processed. In turn, the network learns to output the most confident prediction with high posterior values in the end of the entire training sequence.

The TensorFlow documentation does not indicate any negative impact from using post padding which was the main reason for choosing it for the RRN development. The documentation instead states that every ignorable value at runtime is interpreted as a logical false value which is ignored completely [52]. Pre-padding also have the downside of being incompatible with TensorFlow cuDNN support which is currently used to speed up training significantly. Despite of this, the pre-padding approach still does provide the possibility of greater output which makes it worth to experiment with.

Then there is the implementation of a more meaningful way to calculate the precision and recall metrics used for monitoring the training. It was shown that the results of these metrics imply poor results compared to what the actual experimentation showed. It was furthermore explained that there is an inherent flaw in the way they are currently calculated during the training. Correcting this is most likely a simple matter of implementing a custom-made function for these metrics to be used instead of the currently used TensorFlow implementations.

Finally, there is a more speculative extension idea for the RRN, namely the addition of a feature for determination of the next symptom to query. This would essentially take form as a questioning process tool that can be used by the healthcare practitioner. The feature could be implemented as a new RRN running in parallel with the standard during a consultation, but where the standard RRN would provide disease posteriors, the new RRN would instead provide symptom posteriors.

To predict the symptom posterior, the new RRN would receive as input the current question sequence and current disease hypothesis from time step t in the consultation [53]. The disease hypothesis would be provided by the standard RRN's prediction, which in turn would be based on the current question sequence given by the new RRN. This would create a feedback loop system that yields inference for both diagnosis prediction and medical questioning, thereby making it a very convenient CDSS tool.

This idea is only speculative at this point, but it would be a very useful extension for further optimization of IntMed.

References

[1] J. T. Henriksen and M. Byrgesen. *Approximating Inference Using Recognition Networks in the Medical IntMed System*. (2021).

[2] Ole J. Mengshoel. "Designing Resource-Bounded Reasoners Using Bayesian Networks: System Health Monitoring and Diagnosis". In: Proc. 18th Int. Workshop DX, Nashville, TN (2007), pp. 330–337. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.872.5115&rep=rep1&type=pdf.

[3] L. C. van der Gaag et al. "Building Causal Interaction Models by Recursive Unfolding". In: Proceeding of the International Conference on Probabilistic Graphical Models (2020). URL: https://pgm2020.cs.aau.dk/wp-content/uploads/2020/09/vanderGaag20a.pdf.

[4] Greenes R.A. Musen M.A. Middleton B. *Clinical Decision-Support Systems*. Shortliffe E., Cimino J. (eds) Biomedica Informatics. Springer, London, 2014. ISBN: 978-1-4471-4474-8. DOI: https://doi.org/10.1007/978-1-4471-4474-8_22.

[5] Eta S. Berner. *Clinical Decision-Support Systems. Theory and Practice*. Springer, New York, NY, 1998. ISBN: 978-0-387-38319-4. DOI: https://doi.org/10.1007/978-0-387-38319-4.

[6] Ambolt ApS. *Artificial Intelligence Helps Doctors Diagnose Patients and Find the Best Treatment*. 2018. URL: https://ambolt.io/portfolio/intmed/ (visited on 03/29/2021).

[7] Rüdiger W. Brause. "Medical Analysis and Diagnosis by Neural Networks". In: Crespo J., Maojo V., Martin F. (eds) Medical Data Analysis. ISMDA 2001. Lecture Notes in Computer Science, vol 2199. Springer, Berlin, Heidelberg. (2001). URL: https://doi.org/10.1007/3-540-45497-7_1.

[8] William G. Baxt. "Use of an Artificial Neural Network for the Diagnosis of Myocardial Infarction". In: Annals of internal medicine (1991), pp. 843–848. URL: https://doi.org/10.1007/3-540-45497-7_1.

[9] S. Palaniappan and R. Awang. "Intelligent Heart Disease Prediction System Using Data Mining Techniques". In: 2008 IEEE/ACS International Conference on Computer Systems and Applications (2008), pp. 108–115. URL: https://doi.org/10.1007/978-1-4471-4474-8_22.

[10] Ramzi M. Sadek et al. "Parkinson's Disease Prediction Using Artificial Neural Network". In: (2008). URL: http://dstore.alazhar.edu.ps/xmlui/bitstream/handle/123456789/302/IJAHMR190101.pdf.

[11] M. Shwe et al. "Probabilistic Diagnosis Using a Reformulation of the INTERNIST-1/QMR Knowledge Base". In: *"Methods of Information in Medicine", 30* (1991), pp. 241–255. URL: https://pdfs.semanticscholar.org/a06b/0667e6b5fb30da8d3fb57f1c3d925023bfaf.pdf.

[12] T. S. Jaakkola and M. I. Jordan. "Variational Probabilistic Inference and the QMR-DT Network". In: *Journal of Articial Intelligence Research, 10* (1999), pp. 291–322. URL: https://doi.org/10.1613/jair.583.

[13] Finn V. Jensen and Thomas D. Nielsen. *Bayesian Networks and Decision Graphs*. Springer Science Business Media, 2007. Chap. 3, pp. 75–78. ISBN: 978-0-387-68281-5.

[14] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988. ISBN: 9781558604797.

[15] David Heckerman. "A Tractable Inference Algorithm for Diagnosing Multiple Diseases". In: *Machine Intelligence and Pattern Recognition. Vol. 10. North-Holland* (1990), pp. 163–171. URL: https://arxiv.org/abs/1304.1511.

[16] Michael Shwe and Gregory Cooper. "An Empirical Analysis of Likelihood-Weighting Simulation on a Large, Multiply Connected Medical Belief Network". In: *Computers and Biomedical Research* (1991), pp. 453–475. URL: https://arxiv.org/ftp/arxiv/papers/1304/1304.1141.pdf.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. Chap. 6. URL: https://www.deeplearningbook.org/contents/mlp.html.

[18] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. Chap. 1. URL: http://neuralnetworksanddeeplearning.com/chap1.html.

[19] Christopher Olah. *Neural Networks, Manifolds, and Topology*. 2014. URL: http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/ (visited on 03/31/2021).

[20] Quaid Morris. "Recognition Networks for Approximate Inference in BN20 Networks". In: Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (2001). URL: https://arxiv.org/abs/1301.2295.

[21] George Papamakarios and Iain Murray. "Fast ϵ-Free Inference of Simulation Models with Bayesian Conditional Density Estimation". In: (2016). URL: https://arxiv.org/pdf/1605.06376.pdf.

[22] Quaid Morris. "Practical Probabilistic Inference". In: (2003), pp. 99–106. URL: https://dspace.mit.edu/bitstream/handle/1721.1/29989/54792349-MIT.pdf.

[23] Jian Cheng and Marek J. Druzdzel. "AIS-BN: An Adaptive Importance Sampling Algorithm for Evidential Reasoning in Large Bayesian Networks". In: *Journal of Artificial Intelligence Research* (2000), pp. 55–188. URL: https://doi.org/10.1613/jair.764.

[24] Jeff Heaton. *The Number of Hidden Layers*. 2017. URL: https://www.heatonresearch.com/2017/06/01/hidden-layers.html.

[25] Leonid Datta. *A Survey on Activation Functions and their Relation with Xavier and He Normal Initialization*. 2020. URL: https://arxiv.org/pdf/2004.06632.pdf.

[26] Ilya Sutskever Alex Krizhevsky and Geoffrey E Hinton. "Imagenet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems* (2012), pp. 1097–1105. URL: https://kr.nvidia.com/content/tesla/pdf/machine-learning/imagenet-classification-with-deep-convolutional-nn.pdf.

[27] Samir S. Yadav, Shivajirao M. Jadhav, and Milos Hauskrecht. "Deep Convolutional Neural Network Based Medical Image Classification for Disease Diagnosis". In: *Journal of Big Data 6.1* (2019), pp. 1–18. URL: https://link.springer.com/article/10.1186/s40537-019-0276-2.

[28] Toshiaki Hirasawa et al. "Application of Artificial Intelligence using a Convolutional Neural Network for Detecting Gastric Cancer in Endoscopic Images". In: *Gastric Cancer 21.4* (2018), pp. 653–660. URL: https://doi.org/10.1007/s10120-018-0793-2.

[29] Holger A. Haenssle et al. "Man Against Machine: Diagnostic Performance of a Deep Learning Convolutional Neural Network for Dermoscopic Melanoma Recognition in Comparison to 58 Dermatologists". In: *Annals of Oncology 29.8* (2018), pp. 1836–1842. URL: https://doi.org/10.1093/annonc/mdy166.

[30] Mohammad Rahimzadeh and Abolfazl Attar. "A Modified Deep Convolutional Neural Network for Detecting COVID-19 and Pneumonia from Chest X-ray Images based on the Concatenation of Xception and ResNet50V2". In: *Informatics in Medicine Unlocked, vol. 19* (2020). URL: https://doi.org/10.1016/j.imu.2020.100360.

[31] Jen Hong Tan et al. "Application of Stacked Convolutional and Long Short-Term Memory Network for Accurate Identification of CAD ECG Signals". In: *Computers in biology and medicine 94* (2018), pp. 19–26. URL: https://doi.org/10.1016/j.compbiomed.2017.12.023.

[32] Gaobo Liang et al. "Combining Convolutional Neural Network with Recursive Neural Network for Blood Cell Image Classification". In: *IEEE Access 6* (2018). URL: https://ieeexplore.ieee.org/document/8402091.

[33] Edward Choi et al. "Multi-layer Representation Learning for Medical Concepts". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016). URL: https://dl.acm.org/doi/pdf/10.1145/2939672.2939823.

[34] Cristóbal Esteban et al. "Predicting Clinical Events by Combining Static and Dynamic Information Using Recurrent Neural Networks". In: *2016 IEEE International Conference on Healthcare Informatics (ICHI)* (2016). URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7776332.

[35] Donglin Guo et al. "Disease Inference with Symptom Extraction and Bidirectional Recurrent Neural Network". In: *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)* (2018). URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8621182.

[36] David G. Covell, Gwen C. Uman, and Phil R. Manning. "Information needs in office practice: are they being met?" In: *Annals of internal medicine 103.4* (1985), pp. 596–599. URL: https://doi.org/10.7326/0003-4819-103-4-596.

[37] Paul N. Gorman and Mark Helfand. "Information seeking in primary care: how physicians choose which clinical questions to pursue and which to leave unanswered". In: *Medical Decision Making 15.2* (1995), pp. 113–119. URL: https://journals.sagepub.com/doi/pdf/10.1177/0272989X9501500203.

[38] Andrej Karpathy. *The Unreasonable Effectiveness of Recurrent Neural Networks*. 2015. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/ (visited on 05/14/2021).

[39] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: (2014). URL: https://arxiv.org/pdf/1409.3215.pdf.

[40] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. Chap. 10. URL: https://www.deeplearningbook.org/contents/rnn.html.

[41] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 05/15/2021).

[42] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation 9.8* (1997), pp. 1735–1780. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.676.4320&rep=rep1&type=pdf.

[43] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: (2014). URL: https://arxiv.org/pdf/1406.1078.pdf.

[44] Edward Choi et al. "Doctor AI: Predicting Clinical Events via Recurrent Neural Networks". In: *Machine learning for healthcare conference* (2016). URL: http://proceedings.mlr.press/v56/Choi16.pdf.

[45] Zachary C. Lipton et al. "Learning to diagnose with LSTM recurrent neural networks". In: (2015). URL: https://arxiv.org/pdf/1511.03677.pdf.

[46] Benjamin Shickel et al. "Deep EHR: A Survey of Recent Advances in Deep Learning Techniques for Electronic Health Record (EHR) Analysis". In: *IEEE journal of biomedical and health informatics 22.5* (2017), pp. 1589–1604. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8086133.

[47] Sarah Taghavi Namin et al. "Deep phenotyping: Deep Learning for Temporal Phenotype/Genotype Classification". In: *Plant methods 14.1* (2018). URL: https://plantmethods.biomedcentral.com/track/pdf/10.1186/s13007-018-0333-4.pdf.

[48] Chuan Guo et al. "On Calibration of Modern Neural Networks". In: *International Conference on Machine Learning* (2017). URL: http://proceedings.mlr.press/v70/guo17a/guo17a.pdf.

[49] Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. "Obtaining Well Calibrated Probabilities using Bayesian Binning". In: *Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 29. No. 1.* (2015). URL: https://ojs.aaai.org/index.php/AAAI/article/view/9602.

[50] Peyman Sheikholharam Mashhadi, Sławomir Nowaczyk, and Sepideh Pashami. "Stacked Ensemble of Recurrent Neural Networks for Predicting Turbocharger Remaining Useful Life". In: *Applied Sciences 10.1* (2020). URL: https://doi.org/10.3390/app10010069.

[51] Mahidhar Dwarampudi and N. V. Subba Reddy. "Effects of Padding on LSTMs and CNNs". In: (2019). URL: https://arxiv.org/pdf/1903.07288.pdf.

[52] Google Brain Team. *Masking and padding with Keras*. 2021. URL: https://www.tensorflow.org/guide/keras/masking_and_padding.

[53] Chaochun Liu et al. "Augmented LSTM Framework to Construct Medical Self-Diagnosis Android". In: *2016 IEEE 16th International Conference on Data Mining (ICDM)* (2016). URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7837849.

A Development Documentation

This appendix contains documentation of development experiments that were used for investigating what worked best in terms of recognition network construction and architecture. These are but a subset of the entire set of investigative experiments conducted. The entire set of plots would be excessive to include. The main points to document with these plots are the choice of RN size, the effect of temperature scaling for the RNs, and the invention of the RN sharpening concept.

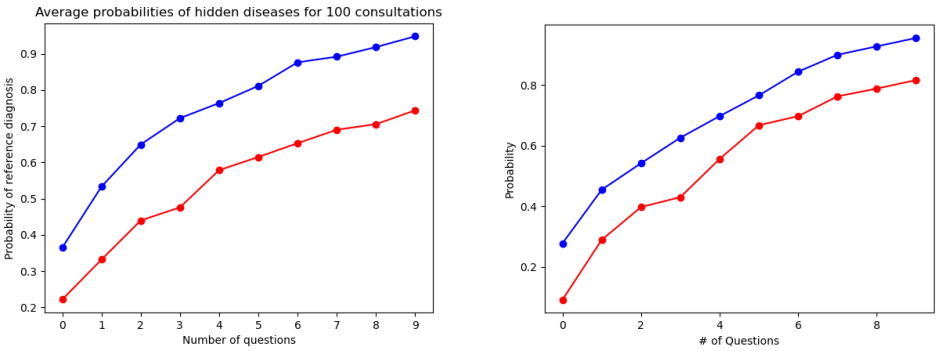


Figure A.1: *Early RN experiment showing averaged posteriors per question taken over many consultations. Left plot shows the experiment after temperature scaling was applied [48], and right shows the result before it was applied. The plots indicate that temperature scaling has little effect for the RN case.*

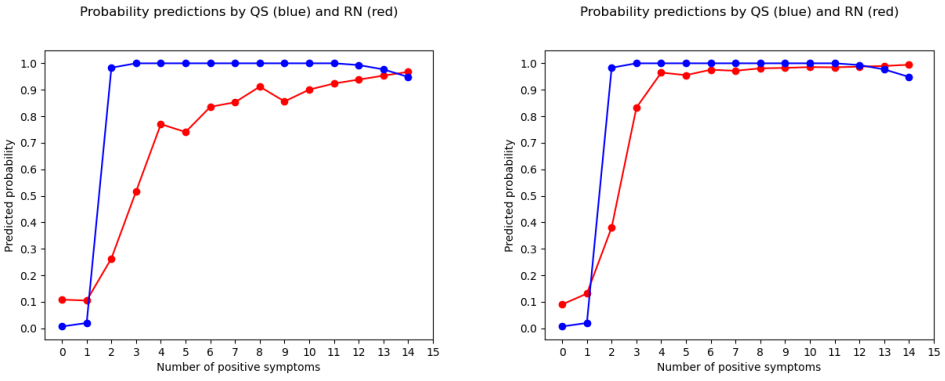


Figure A.2: *Posterior estimations by two different RRNs trained to predict for a BN2O with 50 diseases. The results to the left are made by an RRN with 2 cells and 150 units, and the results to the right are made by an RRN with 3 cells and 300 units. Both RRNs were trained with the exact same number of samples. The plots indicate that a bigger network size is better.*

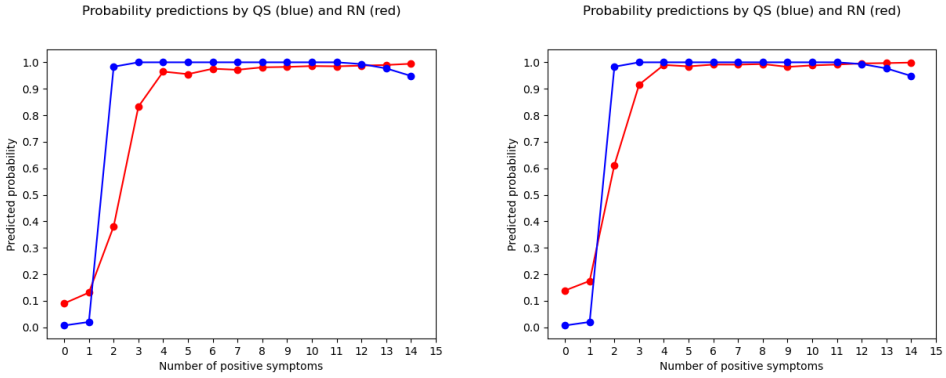
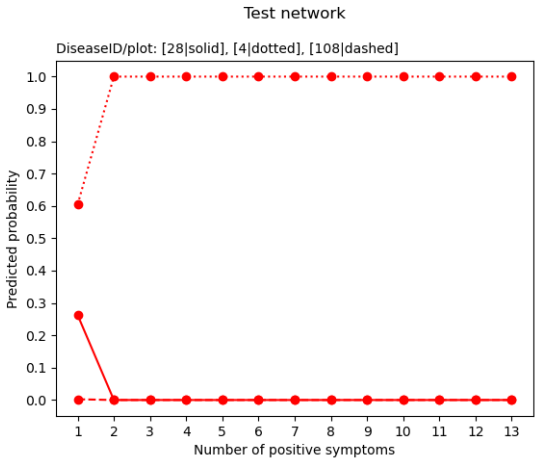
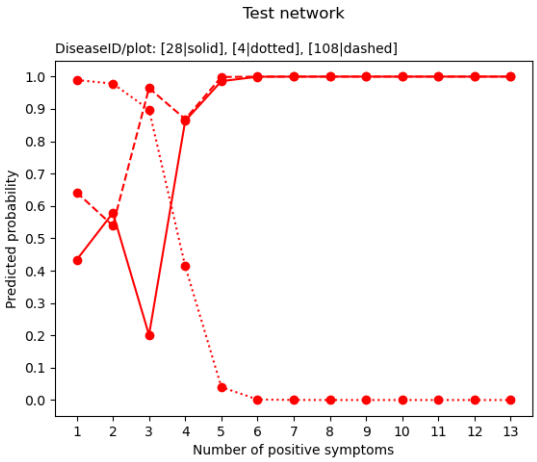


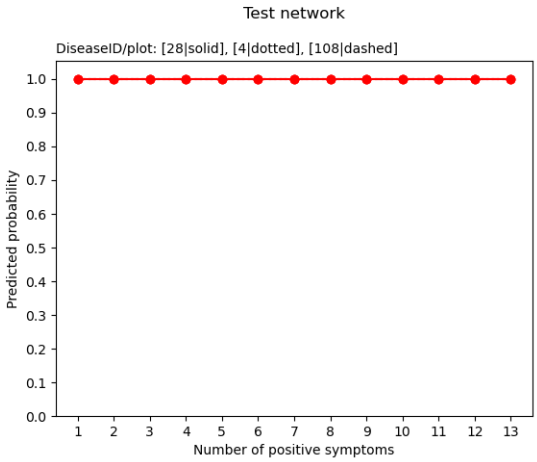
Figure A.3: *Ouput posteriors based on the same observation sequence by two different RRNs trained to predict for a BN2O with 50 diseases. The RRN which has produced the results of the left plot was trained until fairly low convergence, whereas the one that has produced the right first underwent sharpening, before being trained till fairly low convergence. The plots indicate a higher posterior output for the RRNs that have been sharpened.*



(a) *Trained with 1 disease in every target.*



(b) *Trained with 2 diseases in every target.*



(c) *Trained with all 3 diseases in every target.*

Figure A.4: *Experiments with RRNs trained for a very small BN2O with only 3 diseases. Each network has been trained with a number of diseases in each reference diagnosis as per described by their caption. In the experiment all symptoms of the network (13 symptoms total) are given positive evidence. It can clearly be seen that the number of target diseases each of these have been trained with dictates how many diseases that are given a high posterior consistently. Even though A.4a and A.4b have been trained with examples of every disease in the small BN2O and thereby also all symptoms, they still predict low posteriors for them, despite having been introduced to all disease-symptom relations. This indicates that the networks predict lower posteriors if they are trained with many examples where diseases are absent. This sparked the idea for the RN sharpening concept.*