DISSERTATIONES
INFORMATICAE
UNIVERSITATIS
TARTUENSIS
21

ARDI TAMPUU

Neural Networks for Analyzing Biological Data

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS

21

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS

21

ARDI TAMPUU

Neural Networks
for Analyzing Biological Data

UNIVERSITY OF TARTU
Press

Institute of Computer Science, Faculty of Science and Technology, University of Tartu, Estonia.

Dissertation has been accepted for the commencement of the degree of Doctor of Philosophy (PhD) in informatics on 26th of August, 2020 by the Council of the Institute of Computer Science, University of Tartu.

Supervisor

Prof. Dr.	Raul Vicente Zafra Computational Neuroscience Lab Institute of Computer Science University of Tartu, Tartu, Estonia

Opponents

Dr.	Oliver Stegle European Molecular Biology Laboratory Genome Biology Unit, Heidelberg, Germany
	Divison of Computational Genomics and Systems Genetics, German Cancer Research Center (DKFZ), Heidelberg, Germany
Prof. Dr.	Aušra Saudargienė Laboratory of Biophysics and Bioinformatics Neuroscience Institute Lithuanian University of Health Sciences, Kaunas, Lithuania

The public defense will take place on 13th of October, 2020 at 14:15 in University of Tartu Delta building, Narva maantee 18, room 2049.

The publication of this dissertation was financed by the Institute of Computer Science, University of Tartu.

Copyright © 2020 by Ardi Tampuu

ISSN 2613-5906
ISBN 978-9949-03-439-0 (print)
ISBN 978-9949-03-440-6 (pdf)

University of Tartu Press
`http://www.tyk.ee/`

Kallitele

ABSTRACT

Deep learning, i.e. the application of artificial neural networks, has become the prevalent machine learning approach in some fields of data science. Significant improvements in computer vision and natural language processing can be attributed to the recent re-discovery of neural networks. In this thesis we investigate if deep learning methods can help solve problems also in the fields of bioinformatics and neuroinformatics.

In particular, we first apply fully-connected neural networks and convolutional neural networks to data derived from metagenomic experiments. We show that convolutional networks can reliably separate viral DNA from non-viral DNA, without the need to query a genome database. This facilitates the identification of new, yet unknown viral species from the samples.

Secondly, we show that recurrent neural networks (RNNs) can effectively decode information from single-neuron recordings. In particular, our RNN-based decoder outperforms baseline Bayesian models on the task of decoding an animal's location from its hippocampal neural activity. Recurrent neural networks possess the ability to accumulate information over a series of inputs, i.e. build a context over past inputs. This allows them to deal more efficiently with noisy and scarce data.

Compared to the baseline methods used, neural networks required less input pre-processing, made fewer explicit assumptions about the data and allowed to use more of the data. Effectively, our approaches were able to better access the information contained in the data, which in turn led to better performance. We believe that such ability is likely to prove useful in many other applications in bioinformatics, neuroinformatics and elsewhere.

CONTENTS

List of original publications **9**

Preface **10**

1. Introduction **13**

2. Background **15**

2.1. Background on metagenomics 15
2.2. Background on place cell recordings 16
2.3. Biological and historical origins of artificial neural networks . . . 19
2.4. General machine learning terms 22
    2.4.1. Machine learning 22
    2.4.2. Supervised learning - classification and regression 22
    2.4.3. Train-val-test splitting, generalization and overfitting 23
    2.4.4. Losses and metrics 24
    2.4.5. Imbalanced classes problem 26
2.5. Neural networks methods 26
    2.5.1. Fully-connected neural networks 27
    2.5.2. Convolutional neural networks 31
    2.5.3. Recurrent neural networks 34

3. Analyzing codon-based metagenomic data with feedforward neural networks on metagenomic data (Publication I) **40**

3.1. Biological question 40
3.2. Data and models 41
    3.2.1. Data aquisition and pre-processing 41
    3.2.2. Data partitioning 42
    3.2.3. Machine Learning algorithms used 42
3.3. Viral nature can be predicted 43
3.4. Discussion 44

4. Analyzing raw DNA sequences from metagenomic experiments with convolutional neural networks (Publication II) **46**

4.1. Biological question 46
4.2. Data preprocessing 46
4.3. CNN model and baseline ML algorithms 47
4.4. Improved performance in viral classification task 50
    4.4.1. Additional measures of generalization ability 51
4.5. Discussion 52
    4.5.1. Discussion of results 52
    4.5.2. Future work 53
    4.5.3. Limitations 55

5. Applying recurrent neural networks on single-neuron recordings (Publication III) 57
5.1. Neuroscience question 57
5.1.1. Baseline Bayesian models 58
5.2. Data and Methods 60
5.2.1. Single-cell recordings 60
5.2.2. Input data to the RNNs and baseline models 60
5.2.3. RNN model 61
5.3. Results 62
5.3.1. High position decoding accuracy with RNNs 62
5.3.2. Detailed analysis of the RNN model 62
5.4. Discussion 66
5.4.1. Discussion of results 66
5.4.2. Future work 67

6. Conclusion 70

Bibliography 73

Appendix A. Neural network architectures in detail 81
A.1. Publication I: feedforward neural network architecture 81
A.2. Publication II: ViraMiner convolutional neural network architecture 81
A.3. Publication III: Recurrent neural network architecture for decoding self-location 82

Acknowledgement 84

Summary in Estonian 85

Publications 87

Curriculum Vitae 147

Elulookirjeldus 148

LIST OF ORIGINAL PUBLICATIONS

Publications included in the thesis

1. Bzhalava, Z., Tampuu, A., Bała, P., Vicente, R. and Dillner, J., 2018. Machine Learning for detection of viral sequences in human metagenomic datasets. BMC bioinformatics, 19(1), p.336.
2. Tampuu, A., Bzhalava, Z., Dillner, J. and Vicente, R., 2019. ViraMiner: deep learning on raw DNA sequences for identifying viral genomes in human samples. PLOS ONE 14(9), p. e0222271.
3. Tampuu, A., Matiisen, T., Ólafsdóttir, H.F., Barry, C. and Vicente, R., 2019. Efficient neural decoding of self-location with a deep recurrent network. PLoS computational biology, 15(2), p.e1006822.

My contributions to these articles were as follows:

1. Contributed to decisions made in sequence preprocessing, performed machine learning analysis based on extracted RSCU values, decided the metrics and data partitioning to use, designed and drew the figures, wrote most of the Methods and Results sections, contributed importantly to other sections.
2. Designed the machine learning approaches and implemented them, decided the metrics and data partitioning to use, designed and drew the figures, wrote most of the Methods and Results sections, contributed importantly to other sections.
3. Designed the recurrent neural network approach and implemented it, decided the metrics, data partitioning and sensitivity analysis to use, designed and drew the figures, wrote the Methods sections relating to neural networks, wrote the Results section, contributed importantly to other sections.

Publications not included in the thesis

1. Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J. and Vicente, R. (2017). Multiagent cooperation and competition with deep reinforcement learning. PloS one, 12(4), e0172395.
2. Labash, A., Tampuu, A., Matiisen, T., Aru, J. and Vicente, R., 2018. APES: a Python toolbox for simulating reinforcement learning environments. arXiv preprint arXiv:1808.10692.
3. Labash, A., Aru, J., Matiisen, T., Tampuu, A. and Vicente, R., 2019. Perspective Taking in Deep Reinforcement Learning Agents. arXiv preprint arXiv:1907.01851.

PREFACE

This section contains a personal discussion about how and why I ended up doing the research that is summarized in this thesis. I will begin with a short description of my path to data science. Later, in a deeper, philosophical discussion, I aim to illustrate the importance of interdisciplinary research.

When I started my PhD studies, my thesis topic was ought to be much more related to neuroscience and much less associated with data science or machine learning. I was working with detailed simulations of populations of thalamo-cortical neurons. Diverse and intricately interconnected these neurons formed the most complex neural network I have ever worked with. With thousands of parameters to tune, little knowledge of how to optimize them or even what the desired outcome is (how should this brain area behave?), the project stalled. At the same time, early 2014, a new challenge emerged - our group of young researchers decided to replicate the work of DeepMind from the article “Playing Atari with Deep Reinforcement Learning”. In essence, the article describes how to teach computers, artificial intelligence if you wish, to play computer games by trial and error. Accompanied by the authors’ explanations how this is the way to reach truly intelligent machines (artificial general intelligence), it seemed (and still does) like an important breakthrough. We wanted to understand this work, replicate it and maybe do something cool with it. The only problem was that the article was packed with terms and methods that we did not understand. What is reinforcement learning? What is deep learning? Back-propagation? RMSProp optimizer?

In a year, to understand this one paper, I obtained from scratch a decent understanding of machine learning, artificial neural networks and reinforcement learning. Despite not reaching our goal of replicating DeepMind’s results before they released their own codebase, this work eventually led to my most successful scientific article "Multiagent cooperation and competition with deep reinforcement learning". This article is not included in this thesis, as I want to put emphasis on what followed - using the machine learning and neural networks expertise (gained only thanks to this "DeepMind replication" project) for analyzing biological datasets. Since my previous diplomas are on “Bioinformatics and modelling” and “Mathematics and informatics of life”, working on biological data, rather than AI playing computer games, is a more natural environment for me.

In the second part of this Preface, I want to discuss how exporting existing methods from one scientific field to another (i.e. interdisciplinary research) can be more worthwhile than tinkering and tweaking to come up with new, better methods. To do that, however, I need to talk about human knowledge in its entirety.

At least for me, it is impossible to grasp the extent of human knowledge, let alone describe or quantify it. To talk about it, one needs to use extreme level of abstraction hoping that the point does not get lost due to simplifications. In here,

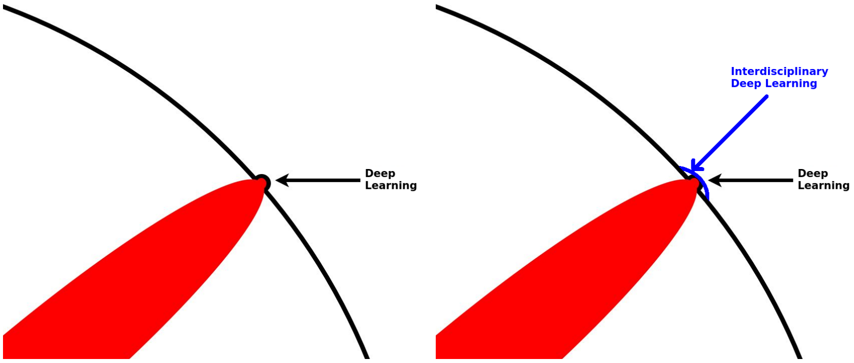


Figure 1. A zoomed in section on the sphere of human knowledge. Consider that the red area corresponds to data science in general. (a) I propose to think of the impact made by Deep Learning to human knowledge as a small bump on the surface of the very huge "sphere of human knowledge". Notice that the surface area of the bump can be increased by making it wider. This corresponds to interdisciplinary research where knowledge is spread out by transferring it to other fields. Figures adapted from http://matt.might.net/articles/phd-school-in-pictures/

I gratefully use the illustrations made by Matt Might in his blog post on how to think about PhD. The point of his “The illustrated guide to PhD”[1] is to soothe the young researchers for not making huge impact during their PhD studies. It is normal that after years of specialization one adds just a tiny bump to the surface of human knowledge. In here, I want to re-purpose these figures to illustrate the importance of interdisciplinary research.

Consider the impact that deep learning has had on the sphere of human knowledge. In my imagination, it forms a bump on the surface of "the sphere of knowledge of mankind" as a PhD work did in Matt Might’s illustrations (obviously a bump of a lot larger scale). Interdisciplinary work, including this thesis, that applies the methods of deep learning to all kinds of different fields - may it be neuroscience, astrophysics or art - widens this bump. Such research helps to spread the knowledge. Notice that if the bump is very deep, but very narrow - a lot of advanced methods without a wide range of applications - the gained surface is smaller than in the case of a less pronounced but wider bump. This is what I believe - spreading the knowledge of these powerful and very useful algorithms to other fields is oftentimes more impactful, more important than minor improvements to the algorithms.

It is impossible for anyone to single-handedly introduce a method to a field. The main goal must be to inspire and encourage people already in the field to learn and start using the method. To do this, one does not only need to show good results, but also make the method look simple, understandable and easy to use. For this reason it is not always crucial to use the most advanced, most

---

[1] http://matt.might.net/articles/phd-school-in-pictures/

complicated methods that will remain obscure to the uninitiated reader. In short, I imagine that the more deep-learning related articles a researcher encounters in his/her field and the more understandable they are, the more likely he/she is to invest time in learning the methods. Or to include a student with this knowledge in their research group. This is where I see my thesis' contribution to science - through showing the methods work and they are accessible, promote the usage of them in more fields.

1. INTRODUCTION

Interdisciplinary research combines the knowledge, methods and data from two or more scientific fields. The goal of this research is to tackle problems that would not be solvable with the tools of only one of the involved fields. Often, just the overall way of doing things varies from field to field, for example in some fields quantitative measures are more heavily relied on than in others. Frequently the methods applied and the metrics to compare results differ.

Deep learning, i.e. the application of multi-layer neural networks, has revolutionized many sub-fields of computer science, including machine learning, computer vision and natural language processing [36, 46, 55]. The goal of this thesis is to export this very powerful machine learning algorithm to new scientific fields. It is likely that neural networks would perform equally well also on many biological datasets [3]. In this thesis, we apply deep learning methods to data from neuroscience and from metagenomics.

In the Background section we will first describe the biological fields that the data originates from. We will then introduce the machine learning terminology and the methods that we apply on the datasets. Clearly, a particular emphasis is put on artificial neural networks. Hopefully the included formulas will allow the reader to understand the internal functioning of these networks in detail.

Chapter 3 summarizes the work done in the article "Machine Learning for detection of viral sequences in human metagenomic datasets" [16]. We apply the rather easy-to-understand fully-connected neural networks to tackle the problem of detecting viral DNA sequences. Random forest (using scikit-learn Python package [74]) is used as a baseline method. Given extracted features from DNA sequences of unknown origin, the machine learning tools are required to estimate the likelihood of each sequence being of viral origin. The results are satisfactory, with performance well above chance level. Using the proposed model as a recommendation system to determine which sequences are more likely viral and should be studied further, the user noticeably improves the chances of finding viruses.

Chapter 4 we further improve on the work done for viral identification. We apply convolutional neural networks (CNNs) directly on raw DNA sequences, without any prior feature-extraction steps (which were used in Chapter 3). The proposed CNN models achieve a significantly improved classification ability compared to baseline models and compared to the results in the previous chapter. The difference in performance is likely to originate from CNNs being able to learn by themselves the features that are extracted from the data, instead of using a predefined feature extraction. This allows to find, through optimization, the most efficient representation of the inputs for the classification task.

Chapter 5 we change topic and apply artificial neural networks to decode information from the activity of real neurons. Neuroscientists have discovered that certain type of cells in CA1 region of rat hippocampus contain information about the location of the animal [70, 71]. We apply recurrent neural networks (RNNs) to decode the location of the animal based on single-neuron recordings from a tiny subset of cells in the CA1 region. Given a timeseries of neural activity our RNN model is able to learn how to use past activity as contextual information. This ability - to flexibly learn from data how to build and use context - is rare among machine learning tools. We believe that using context plays a vital role in the brain and is useful for decoding variables from neural data. Indeed, the proposed RNN decoder outperforms the baseline methods, including an advanced Bayesian model that also has access to past activity.

We finalize by drawing a few overall conclusions. The work done covers three major types of neural networks - fully-connected, convolutional and recurrent neural networks. We discuss their performance on their respective tasks, their strengths and limitations. With these publications we have taken a small step in introducing these powerful methods to the fields of metagenomics and neuroscience (more particularly, single-cell recordings).

2. BACKGROUND

In this chapter we first introduce the types of biological data that were used in this thesis. We will then give some background to the methods that were applied. We aim to introduce and explain the core terminology, so that we can use the terms without confusion in later chapters. Hopefully this chapter helps the reader to put the methods and the work done into a larger context. As applying neural networks to biological data (including brain data) is at the center of this thesis, we also discuss how biologically plausible and brain-like different network types are.

2.1. Background on metagenomics

Sequencing the genomes of animal and microbial species has hugely improved our understanding of how life works. Understanding the human genome - what it consists of, how it is regulated and how it might malfunction - is clearly an important scientific study that has helped identify and cure diseases. For understanding the processes happening inside our body the sequencing, classifying and understanding the genomes of bacteria, fungi and viruses is as important as understanding the human genome [103]. In fact, there are more bacterial cells in our body than there are our own cells [87]. Most of these bacteria are friendly and not pathogenic - they help and protect our body. Imbalance or malfunction in such "good" bacterial populations can cause health issues ranging from mild upset in digestion up to misregulation of the entire immune system [62, 95, 99]. Similarly, many viruses are present in our body, roughly 380 trillion [65]. Some are harmless, some infect our good bacteria, some infect our own cells. To understand if some of these bacteria or viruses cause diseases we need to first identify and characterize them. It is clearly not sufficient to just know that there are lots of viruses, without knowing what they are like and what they can do.

The problem when trying to learn about the microbes living inside and on the surface of our body is that 99% of them cannot survive outside that specific environment [51, 65]. That means one cannot grow them on a petri dish. The classical microbial method of cultivating clean clonal cultures cannot be used. Unable to cultivate the microbes in the lab one is restricted to just taking samples from the environment of interest and sequencing all the DNA material in that sample. This sequencing of all DNA in an environment is called metagenomics [41, 102].

The pipeline of extracting DNA from an environmental sample is the following - we randomly cut all genomes found in the sample into smaller pieces and then sequence these pieces. The shearing of the genomes is done because the high-throughput sequencing machines can only sequence up to a few hundred base pairs (bp) in a row. The next step is to reconstruct longer sequences by methods known as sequence assembly [102]. Essentially we align the short sequences, find sequences that have sufficiently long overlapping parts and merge them into a longer sequence. The overlapping regions might disagree in some nucleotides and conflicts must be resolved in some way. The simplest option is to replace the conflicting nucleotides with N (meaning "unknown nucleotide"). Because of containing letters other than ATCG the result of assembly is not really like a real DNA sequence and we refer to it as "contig" in our work (short for "contiguous sequence", a term introduced in 1980 by Staden [94]).

The contigs can then be aligned with known genomes with BLAST [2] or other tools. In short, the *basic local alignment search tool* (BLAST) compares a nucleotide sequence (called the *query sequence*) with a library (i.e. database) of known sequences, and identifies the library sequences that resemble the query sequence above a certain threshold. This allows to taxonomically classify part of the contigs (the ones that align well with some known sequence). However, many of the contigs are left unclassified as they come from new species that have not been studied and sequenced before and are therefore not found in the genome database [65]. Finding and characterizing these new species as well as estimating the total diversity in the environment is one one the major goals of metagenomics. With cultivation-based methods we would not know that there are millions of bacterial and viral species living inside our body. With metagenomics we can estimate the amount of unknown species, try to identify and characterize them, understand their function and investigate if they might cause diseases.

Human papilloma virus causing cervical cancer is the most famous, but not the only proven case of carcinogenic viruses. For example, Epstein-Barr virus, Kaposi's sarcoma herpes virus and human T-cell lymphotropic are also associated with cancer development [12]. There might be many other cancer-related viruses undiscovered. It has been observed that immunodeficient patients develop some cancers (e.g. non-mealanoma skin cancers, lip, bladder, eye, lung, colon, etc.) with noticeably higher rate, while certain other cancer types (e.g. brain, breast, prostate) do not show any increase in incidence. It gives rise to the hypothesis, that many carcinogenic viruses are yet undiscovered.

In the fist and second contributions of this thesis, we build and improve a recommendation system that helps to more easily discover the yet unknown viruses in human metagenomic samples. These viruses can then be studied further by virologists and their possible relations to diseases investigated.

2.2. Background on place cell recordings

Brain is one of the most complex structures in the known universe and solely this fact makes it a fascinating study. Moreover, understanding the brain can help fight brain-related illnesses. These illnesses do not only include degenerative diseases such as Parkinsons, but also mental disorders that influence our daily life such as depression and addictions. OECD estimates the cost of mental illnesses to the economies of EU member states to be above 500 billion euros per year [69]. Hence, both intellectually and economically there are not many more worthwhile scientific endeavours than studying the brain.

To understand what is happening inside the brain when we think and act, various recording methods have been designed. As there are around 80 billion neurons in the human brain [6], it is unrealistic to record the state changes of all of them at the same time. Instead, many methods record activation level of populations of neurons. For example, electro-encelography, magento-encelography, near-infrared spectroscopy and functional magnetic resonance imaging all provide population based measures of brain activity at different spatial and temporal precisions. These measuring techniques have helped us understand the structure and the large-scale connectivity of the brain. They tell us where and when computations happen as we see populations getting activated in response to stimuli. From other sources, we also know the neuron-level anatomy of each region (from brain slices), and how the particular neuron types roughly function (from growing and testing them on a plate). By combining these pieces of knowledge we can make hypothesis on how the specific area performs the operation we have attributed to it by population-level imaging techniques. However, without recording what individual cells do in a living (!) animal, it is tricky to confirm any such hypothesis.

As stated before, it is unfeasible to record all neurons in the brain. Furthermore, the brain is very tightly packed and inserting electrodes to measure each and every cell is equally unrealistic, even in some relatively small population. There would be no space for that many electrodes and we have no guiding mechanism to aim an electrode to specific cells. In fact, the best we can do is insert electrodes with multiple measuring points into a brain area and hope that the cells we can measure from there happen to be informative. Even though at each measuring point we can detect and separate (using “spike sorting” [59]) the activity of multiple surrounding cells, we end up with at most a few hundred cells recorded (maximum a few thousand with special hardware [67]). This is a tiny randomly selected fraction of the cells in an area. These sparse and random measurements are nevertheless an interesting source of information about the brain.

In the third contribution of this thesis we analyze the activity of neurons in CA1 area of hippocampus of 5 rats. The rats have electrodes fixed into their brains and they can move around freely in environments that we place them in. We simultaneously record their position and their brain activity. The electrodes can detect between 26 and 72 neurons depending on the animal. From the activity of these few neurons we aim to predict (decode) where the animal is located in the environment. With 70 random neurons from a random part of the brain this task would be unfeasible, but the CA1 region is special. In 1971 O’Keefe and Dostrovsky discovered that there are cells in the hippocampus that get activated every time the animal is located in a specific part of space [70]. Elsewhere the neuron is inactivated. These cells are called "place cells" and the region where they get activated is their "place field"(see example on Figure 2). A combination of many place cells makes up a map of the environment [28,71] - by observing the

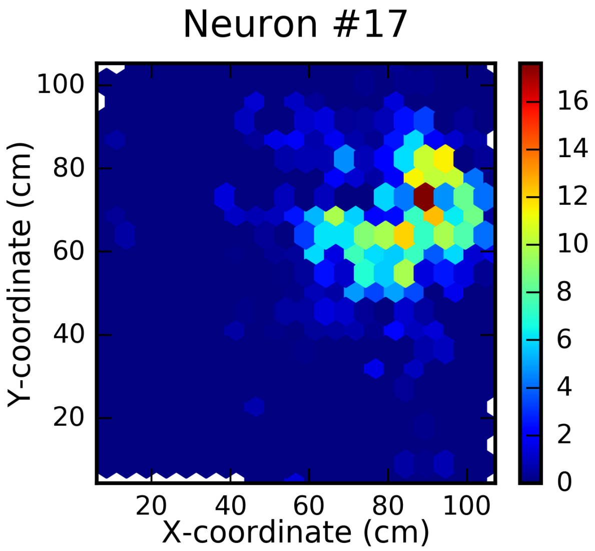


Figure 2. Example place field from the data used in Publication III. The x and y axis represent locations in 1x1 m area. The color code reflects the activity of the example neuron in spikes per second averaged over the periods of time the animal was in the corresponding location. A cell with a clearly visible localized area of higher activity is called a "place cell", the zone with more activity is called the "place field" of this cell. Only around 1/10-th of cells in CA1 (and in our recordings) are place cells.

firing patterns of these neurons the rest of the brain can know where the animal is. Our task is similar- to decode the animal's location, but based on only the subset of neurons we managed to record (as opposed to all CA1 neurons).

I would like to stress further the absurd difficulty of this position decoding task. We place some electrodes inside an area we know contains the necessary information. Each electrode measures the change in electric potential. Using existing tools we cut out the time periods where this electric signal looks to correspond to a spike in a nearby neuron. This is not simple, because the distance of that neuron from the electrode and its orientation matter. Some spikes might get thrown away, if the outside noise made them unrecognizable. For example, if two nearby neurons spike in quick succession the measured electric signal is a combination of their effects and might no longer be recognizable as a spike. We then cluster the detected spikes to group together spikes likely to have come from the same neuron. Clusters with too few spikes are discarded. After this the experimenter can verify and clean the results of clustering -for example separate a cluster that actually corresponds to two neurons or merge two clusters that are actually the same neuron. We will then declare that each of these clusters is one neuron and the spikes in the cluster are the times when this neuron fired. This is the input data to all further processing - visualizations of the firing patterns, extracting firing statistics or decoding the animal's position via machine learning. With so few neurons recorded and so many noisy processing steps, it is amazing that we can decode anything at all.

2.3. Biological and historical origins of artificial neural networks

Before explaining the inner workings of modern neural networks used in this thesis, we wish to give credit to the long history of methods aiming to mimic the brain's computations. Neural networks might seem a recent technology, but they have been around since computers were invented. In here we wish to inform the reader of the seminal works that underlie the modern success. Also, we wish to discuss in which ways neural networks are inspired by the brain and how this has been beneficial.

The human brain is the most powerful computer that we know of [60, 66]. It can detect objects, plan actions and give the commands to execute these actions in a fraction of a second. It is capable of highly abstract thought, absurdly complex motor control and amazing levels of creativity. It can solve tasks it has never seen before via generalizations and knowledge transfer from one task to another. It is constantly learning and so plastic (adaptable) that it can function without noticeable behavioural deficits even if half of it has been lost [7, 27, 105]. It is no surprise that for as long as computers and computing have existed brain has been the baseline computers want to beat - from the first computers that could add and multiply faster than humans, to today's algorithms that trade on stock markets or detect objects.

Whereas there are more glial cells than neurons in the brain [6], and glial cells have been shown to participate in certain computations [4, 68], the main computing power of the brain can be accredited to the 80 billion neurons and hundreds of trillions of synapses formed between them. These neurons present a huge diversity – every cubic millimeter of the brain contains neurons with extremely different morphology and function. The same can be said about the synapses - they vary in strength, duration and type (inhibitory or excitatory) of the stimulation. As a neuron is activated only by a combination of multiple co-occurring excitatory signals, the timing of incoming signals and their location (for example, how far from cell body) also matters. This extreme complexity is the reason why we still do not understand the brain. We do not even know what is the computation performed by cortical columns, the repeated structure in the cerebral cortex [76]. Cortex, especially the relative increase of its frontal regions, is suspected to be a major source of our species' intelligence [25, 86, 93].

When making algorithms mimicking how the brain computes, people have usually found it necessary to get rid of this confusing amount of diversity. As we will see below, in artificial neural networks (both historical and modern) we do not consider the diversity of neurons nor the importance of timing and spatial location of connections. There has been some increase in the complexity considered over time, but artificial neural networks remain a caricature of the biological brain.

The first noteworthy computational (mathematical) model aiming to imitate the brain is not much younger than the Turing machine, having been proposed by McCulloch and Pitts already in 1943 [61]. The activation of McCulloch-Pitts (MC) neurons is represented as a binary variable (1 or 0, for active vs inactive), while synapse strengths between neurons are 1 or -1 (positive or negative meaning if the synapse is excitatory or inhibitory). In a MC neuron, the weighted sum of incoming synapses (sum on ones and minus ones) is compared to a threshold (usually set at zero), which determines the activation state of the neuron. With this extremely simple model, all logical operations can be implemented (XOR needs more than one neuron) [61]. Sounds good, however, the authors did not propose how to come up with an optimal network of neurons and connections to perform a task - i.e. there is no learning algorithm and one needs to build the networks by hand.

In 1958 by Frank Rosenblatt proposed a learning algorithm and introduced non-integer connection weights, calling the resulting algorithm the Perceptron [79]. Using real numbered weights made the system more flexible and was a step (albeit tiny) closer to the biological complexity in the brain. The Perceptron ideas were fine-tuned by Minsky and Papert [63]. However, as the perceptron learning algorithm can only "teach" networks with one layer of weights, Minsky and Papert proved that not-linearly-separable functions (such as XOR) could not be learned by it [63]. The discovery of this major limitation caused the scientific world to lose interest in Perceptrons and in neural networks as a whole for a decade [82].

The interest in artificial neurons and networks of these neurons was revived in the 80ies. Kunihiko Fukushima's 1980 neocognitron [34] was ahead of its time. Directly inspired by Hubel and Wiesel's work on visual cortex [50], Fukushima essentially invented **convolutional neural networks**. However, the learning algorithms were still not good enough to make this -in hindsight revolutionary-invention catch on immediately. A major breakthrough by Rummelhart, Hinton and Williams [80] was the proposal of an efficient way to backpropagate error gradients through multiple layers of neurons. This allowed to train "multilayer perceptrons", which had more than one weight layer. Multiple layers of thresholded (or otherwise non-linear) neurons can represent non-linear functions and overcome the limitations discovered by Minsky and Papert [49, 80]. Today we still use layers exactly like the ones in multilayer perceptrons, but we call them **fully-connected layers** as the neurons of consecutive layers are all-to-all connected [36]. These layers are present in vast majority of neural networks. The backpropagation algorithm marked the beginning of modern *deep learning*- deep refers to the depth, i.e number of layers in the networks. Deep, multi-step processing is also preformed in the brain - for example visual information goes through a series of brain regions in the ventral stream before reaching deeper, hierarchically higher regions where objects, faces, words etc. are detected. Indeed, recent research shows that the hierarchical processing of visual information in brain and in deep convolutional networks is highly similar [21,40,56,104].

The currently familiar form of convolutional neural networks (as opposed to neocognitron) was proposed in 1989 by Yann LeCun [58]. As mentioned above, convolutional networks mimic known properties of visual cortex where simple and complex cells combine to achieve space invariance [34, 50, 58]. CNNs are traditionally used to process images - find objects, letters or faces from them. The original 1989 CNN was designed to recognize hand-written ZIP codes [58]. A decade later an improved version of this network, LeNet-5 [57], was used to recognize digits on bank cheques.

With the algorithms reaching industry-level reliability by late 90ies (LeNet-5 [57]), it is surprising to discover that the next remarkable application of CNNs was in 2012 [55], more than a decade later. Whereas the loss of interest in neural networks after 1969 Minsky and Papert's critique is clearly understandable, the loss of interest in the 90ies and 00's is more obscure. It is claimed that the methods were over-hyped and did not live up to the expectations, hence becoming a synonym for empty promises [20]. Also, other machine learning methods emerged that were simpler and as powerful. The negative attitude towards neural networks was ended by Alex Krizhevsky's AlexNet winning the ImageNet 2012 object recognition competition [24, 55, 81]. With a combination of convolutional and fully-connected layers AlexNet learned to classify objects into 1000 different categories with precision way above (15% errors compared to 25%) all other methods. Such huge improvement over existing methods immediately sparked new interest in CNNs and by extension to other types of neural networks. Today the best CNNs can achieve 2% TOP-5 error in the ImageNet task [97], which is claimed to be better than human performance.

A more complex type of neural networks, **recurrent neural networks**, was also proposed in Rummelhart's 1986 groundbreaking "Learning representations by back-propagating errors" article [80]. This type of network is designed to deal with a series of inputs, such as time series. In particular, these networks are able to use contextual information from past inputs to process the current input [36, 80]. Hochreiter and Schmidhuber proposed "long-short term memory"(LSTM), a more powerful version of RNNs, in 1997 [48]. However, once again it took more than a decade from the original invention of LSTMs to see the real impact. Since the end of 00's [37], more pronouncedly since 2012 [46], LSTMs have become essential in natural language processing and speech recognition [18,46,47,83]. In many image processing tasks, just a glimpse, a snapshot is often enough to understand the scene, so the vision models often get away with not considering time, i.e. not considering the sequence of consecutive frames. In language and speech, however, a single word or sound carries little meaning and how the input changes over time is crucial, hence the need for RNNs. Our brains also accumulate and use context to make decisions in response to stimuli that change over time. In this sense, recurrent neural networks are perhaps the closest model of the brain among the artificial neural networks introduced here.

In all, neural networks have always been brain-inspired, though hugely simplified compared to the real complexity of the brain. Nevertheless, important ideas, such as space-invariant cells and recurrent computation are similar in both artificial and neural networks. Researchers are still looking for aspects of the brain that could be instilled into the networks (e.g. attention, explicit memory or prioritized memory replaying).

2.4. General machine learning terms

2.4.1. Machine learning

Machine learning refers to studying and using the set of algorithms that allow computers to learn to solve specific tasks solely based on empirical data. The algorithms are not provided with specific instructions (are not pre-programmed) how to solve the task. Instead, by observing a set of *"training data points"* the algorithms discover useful regularities that help solve the task [9]. Iterating over training samples to discover and fine-tune the set of patterns and rules that allow to solve the task with maximal performance is called *"training"* of the model. Once the model is trained (training has converged to a good set of rules) it can be applied to new data points that were not part of the original training data (validation and testing, covered in a later subsection).

2.4.2. Supervised learning - classification and regression

Machine learning (ML) can solve a variety of tasks. In this thesis we work solely with *supervised learning*. In all three articles considered here, we have datasets containing not only input data, but also the desired outputs. The goal in supervised learning tasks is to learn a function that maps the training inputs to the corresponding outputs as precisely as possible. The precision is measured according to some metric (often called *"loss function"*). Notice that it might not be possible to map all inputs to their correct outputs - either due to the limitations of the model or due to the errors or stochasticity in the desired outputs. The ML algorithm's aim is simply to minimize the sum error (sum loss) across all training input-output pairs.

Depending on the type of desired outputs the machine learning tasks are further divided into classification and regression tasks [1]. In classification tasks the outputs can take a limited set of values (categorical values). For example, classification task might correspond to answering the question: to which class does the input belong to? In this thesis we only see a binary classification problem - does a given DNA sequence (the input) originate from a virus or not.

In regression tasks, the desired outputs are real numbered values. These tasks can answer questions when? where? how much? and so on. In the third article of this thesis we decode the location of an animal from its neural activity. It means we answer the question where the animal is. Or, what are the X and Y coordinate values of the animal's position.

Classification and regression tasks demand the use of different learning algorithms and distinct sets of loss measures. Not all ML algorithms are equally useful in classification and regression tasks [1, 9], but artificial neural networks can be successfully applied to both [36]. The loss functions and metrics are discussed in a later subsection.

2.4.3. Train-val-test splitting, generalization and overfitting

As mentioned above, supervised machine learning algorithms learn a set of rules based on observing the training examples and the corresponding desired outputs. One way of achieving the lowest possible error on training examples is to simply memorize them - learn to identify the training input and memorize its corresponding output. While such approach does exactly what is asked of it - minimizes the training loss, we see empirically that it tends to perform badly on new samples that the model was not trained on. In such case we say that the model lacks the *ability to generalize* and that it has *overfitted* to the training set [1, 44]. Generalization is important because usually the goal is not to just classify already known data points correctly, but rather build a model that can classify unknown points. To measure how well a model performs on data it has not seen during the training process we set aside part of the data points and call them *the validation set*. Performance on validation set is a much more accurate estimation of the model's true capabilities than performance on training set [9].

In most cases we do not know beforehand the model hyper-parameters that are best suitable for solving the task. For example how deep and wide the neural network should be, what optimizer and regularization to use. Hyper-parameter tuning consists in training many models with different configurations on the same training data set and then picking the best model according to the validation performance [1, 54]. Notice that because our validation set is just a subset of all possible data points, two equally good (but not identical) models might show slightly different performance. It just happens that the data points in our limited validation set were favourable for a given model. Hence the "best" model is not only best because it is good, but because it got lucky. It is unlikely to get as lucky on future data points. So the validation accuracy of the best model is likely to overestimate the true generalization ability. We need a further set of data points, called the test set [1]. Test set samples are not used neither in training nor in picking the best model. The performance on test samples is the most fair estimate of a model's ability to generalize that we can get [54].

Notice, however, that given a limited amount of data, we need to choose how much of the data to use in each set. Using more training data is likely to yield better models. At the same time, leaving less data for validation makes the generalization estimation noisy and we might accidentally pick a sub-optimal model [1].

Similarly, small test set might - due to randomness - either over or underestimate the actual ability of the model. It is common to randomly split the data in a way that training set contains 80% and other sets both 10% of the data points. However, in different tasks, the optimal trade-off might be different.

2.4.4. Losses and metrics

For training machine learning models we need to select the measure we want to optimize (minimize or maximize). For example one could minimize the risk or maximize the gains. Also, we need to decide what are the metrics we report as results. The measures that we optimize and those that we actually care about and report are not always the same. [36]

For example, accuracy is an easily understandable measure, however, it is not differentiable. When optimizing a model (e.g. neural network) with gradient descent, the loss function must be differentiable. We cannot use accuracy as the loss function, but we can still use it as the metric to evaluate a trained model's performance.

Binary classification tasks, Chapters 3 and 4. In the binary classification tasks, as in Chapters 3 and 4, our neural networks' loss function is binary cross-entropy (BCE) loss. Given N samples, the predictions by the model p_i and the true values t_i, the BCE is given by:

$$BCE = \sum_{i}^{N}[-t_i \cdot log(p_i) - (1-t_i) \cdot log(1-p_i)] \tag{2.1}$$

where p_i is the probability of the i-th sample belonging to the positive class according to the model. If the correct answer for i-th datapoint was positive, then $t_i = 1$, and if correct answer was negative, then $t_i = 0$. Training of the model consists in minimizing, across all training points, the BCE value between the model's predicted probabilities p_i and the ground truth t_i. However, reporting this loss value (on validation or test set) as the final result is not very informative. We know that smaller value is better, but saying that the average BCE loss was 0.001 is not intuitively informative. Therefore we need to turn to other metrics to describe the performance.

Precision and recall are metrics that are immediately understandable to the reader. One can report the overall precision and recall, or provide values for each class separately. Due to imbalanced class distribution (discussed in the next subsection), in our work we report the precision and recall of the positive class, not the overall measures. Precision of the positive class corresponds to "what proportion of the samples labelled as positive by the model were actually positive". Recall corresponds to "what proportion of positive samples did the model label as positive" (i.e. recovery rate). Both of these measures are immediately understandable and are exactly what we want to know. While useful and understandable, we are still not satisfied with these measures. First of all, the precision of positive class depends on the **prevalence** (proportion of positive classes in the dataset) in the dataset. Hence this precision is data-set specific, not a universal measure of model capability. Secondly, when calculating precision and recall a hidden parameter intervenes - the classification threshold, usually set at 0.5. If $p_i > threshold$, the model labels the sample as positive. With a stricter threshold we get higher accuracy and lower recall, with lower thresholds the other way round [22, 33]. As we have no a priori assumption (or cost function) of how this precision-recall trade-off should be solved, in our work (Chapters 3 and 4) we plot the precision and recall values at all possible thresholds (0 to 1).

Importantly, we also provide the Receiver Operator Characteristic (ROC) curve [31]. The curve is obtained by plotting recall (i.e. true positive rate) against the *probability of false alarm* (i.e. false positive rate) at all possible (relevant) threshold values.

$$\begin{aligned} precision &= \frac{true\ positives}{true\ positives + false\ positives} \\ recall &= \frac{true\ positives}{true\ positives + false\ negatives} \\ false\ alarm\ rate &= \frac{false\ positives}{false\ positives + true\ negatives} \end{aligned} \tag{2.2}$$

Neither recall nor false alarm rate depend on prevalence (proportion of positive classes in the dataset) and the plot summarizes behaviour over all possible thresholds. Hence the area under the ROC curve (AUROC) is a metric that does not depend on prevalence nor a threshold. If we downsampled or upsampled one of the classes (changing prevalence), AUROC would stay the same except for some noise due to sampling. In our particular case, AUROC allows us to directly comparable model performance on datasets of different prevalence. AUROC is the main metric in Chapters 3 and 4.

Regression task, Chapter 5. In the regression task of predicting rat's location based on its neuronal activity the training minimizes mean squared error (MSE) of both coordinates simultaneously. With N samples, p_i^x and p_i^y the predictions for X and Y coordinates at i-th data point, and t_i^x and t_i^y the corresponding true coordinate values, the loss is given by:

$$MSE = \frac{1}{N}\sum_i^N [(t_i^x - p_i^x)^2 + (t_i^y - p_i^y)^2] \tag{2.3}$$

In Chapter 5, the main reported results are, however, measured in mean euclidean distance (MED) between true and predicted locations.

$$MED = \frac{1}{N}\sum_i^N \sqrt{(t_i^x - p_i^x)^2 + (t_i^y - p_i^y)^2} \tag{2.4}$$

Again, the optimized and reported measures are slightly different. Minimizing MSE of the coordinates monotonically minimizes MED, but when optimizing for the entire training set the trade-offs (between data points) made for minimal MSE might not always minimize MED. One could actually directly optimize MED loss when training, but this loss is not available in the Keras neural networks toolbox [19] and the expected gain is small. Hence, we used the more common MSE loss for simplicity.

2.4.5. Imbalanced classes problem

In our metagenomic datasets there are a lot more DNA sequences originating from non-viruses than from viruses. The proportion is roughly 98 to 2. This class imbalance makes the classification task harder [30, 52], because there is a strong incentive to predict the more populated class whenever in doubt. In fact, always predicting the non-viral class in our metagenomic dataset would lead to 98% overall accuracy, which might sound really good. However, a model always giving the same output is useless for distinguishing viruses from non-viruses. The metrics we employ in Chapters 3 and 4 (positive class precision-recall values and AUROC, discussed above) would reveal the weakness of such deceptively "accurate" model.

Notice that preferring to predict the more populated class is not wrong in essence. Given a noisy data point containing no information about its true class, labelling it as the majority class is the best solution. However, it might unfortunately happen that the model learns to rely only on this class-bias and disregards the weak signal in the data. In response, many techniques have been invented to force the model to use the information in the samples, not only the bias in labels [30, 52]. Oversampling consists in reducing class imbalance by adding duplicates of the minority class samples. However, using the same samples many times is likely to lead to overfitting and bad generalization ability. Undersampling just discards part of the data points from the more prominent class, but this means throwing away potentially crucial information. As a third alternative, in case of loss-based methods we can artificially increase the cost of misclassifying the minority class items, effectively forcing the model to pay more attention to getting them right. This is the most common method for dealing with imbalanced classes for neural networks [107]. However, as per our experiments, none of the three methods mentioned improved the results in terms of area under the receiver-operating characteristic curve, neither for baselines nor for neural networks.

2.5. Neural networks methods

In the following subsections we will introduce the basic concepts of modern neural networks and the types of networks used in the articles that are part of this thesis. The articles are ordered by increasing complexity of the networks, so we can start

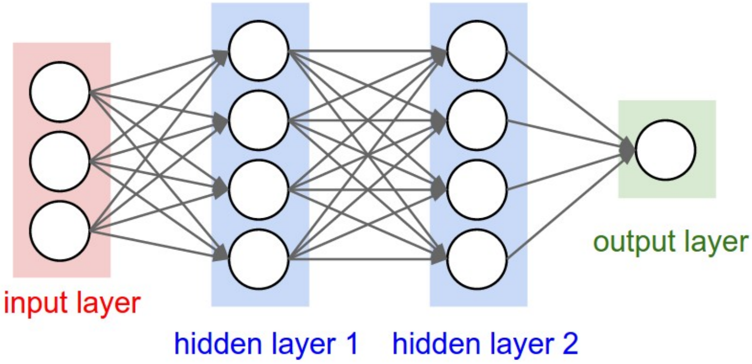


Figure 3. Fully connected neural network with multiple layers. The inputs are inserted as the activations of the input layer neurons. Each neuron in the second layer (first hidden layer) is connected with all the input neurons. Each neuron in the third layer is connected with all neurons in the second layer and each neuron in the output layer is connected with all neurons in the third layer. Connections have real-numbered weights. Output is the activation of last layer neurons or is calculated based on it. Image originates from http://cs231n.github.io/neural-networks-1/

with simpler architectures to introduce basic concepts and then build on them to describe more complex networks.

2.5.1. Fully-connected neural networks

The term "artificial neural networks" (ANN) refers to a versatile machine learning algorithm or rather a family of algorithms. There are indeed very different neural networks out there, with very different structure and learning mechanisms [36]. For example the learning algorithm for restricted Boltzmann machines is contrastive divergence [84], while for most other ANNs it is backpropagation. Neural Turing machines [38] have an extremely complicated internal structure and connectivity patterns, while the most common type of artificial neural networks, fully-connected neural networks (FCNNs), are relatively simple. In the remainder of this subsection we describe what a FCNN looks like and how it learns from examples.

FCNNs are composed of layers of artificial neurons, with consecutive layers all-to-all connected (i.e. fully-connected) with each other (Figure 3). The connections are weighted with real numbers.

Each layer performs the same basic (linear) operation: given a row-vector of inputs i the layer multiplies this vector with a weight matrix W and adds a bias vector b to return an activation vector a:

$$a = i * W + b \tag{2.5}$$

The values in the matrix W and biases in vector b are the learnable parameters of the layer. Notice also that the desired length of the resulting activation vector also determines the second dimension of W and the length of b, hence influencing the number of parameters introduced. This output size is a hyperparameter (also called *layer size* or *layer width*).

As the next processing step, an activation function (usually non-linear function) can be applied to each element of the activation vector a. The most common activation functions include ReLU (rectified linear unit, $g(z) : max(0,z)$), sigmoid and tanh [36]. The non-linearity of activation functions is important for learning complicated, non-linear functions. Hence, the output of the layer becomes:

$$h = elementwise\_activation\_function(a) \tag{2.6}$$

As h is a row-vector, a subsequent FC-layer can be applied to it, again multiplying h with a weight matrix W_2, adding bias b_2 applying activation function and returning a vector of outputs h_2. Many layers can be stacked this way, making the network "deep". This depth is where the name "deep learning" comes from.

$$\begin{aligned}
a_1 &= i * W_1 + b_1 \\
h_1 &= activation(a_1) \\
a_2 &= h_1 * W_2 + b_2 \\
&\dots \\
h_{n-1} &= activation(a_{n-1}) \\
a_n &= h_{n-1} * W_n + b_n
\end{aligned} \tag{2.7}$$

If there are no further layers, the output of the last FC layer is the output of the FCNN model.

Notice there are no cyclic connections in the network and the information moves layer by layer from the inputs towards output nodes. Such networks with no cycles are referred to as *feedforward neural networks*.

FCNNs in regression tasks. Consider the well-known regression task of predicting the housing prices in Boston [43]. In this case we expect one real-numbered output - the price. Hence the last layer's weight matrix is of dimensions $(D\_incoming, 1)$ and there is just one output. In the case of such unbounded real-numbered output usually no activation function is applied (notice no h_n added to the equations above) and the result of the last matrix multiplication (+bias) is the output. In the supervised learning settings, we can then compare this output with the desired output (the true price) and calculate the size of the error (loss) our network made, for example in terms of mean squared error (MSE).

FCNNs in classification tasks. In contrast to regression tasks, in classification the desired output is a vector of probabilities, p_n, with one value per possible class. Each element represents how likely the input is to belong to one of the classes. If the classes are mutually exclusive the sum of these probabilities should be 1. To achieve this, the final layer activation a_n (of length num_classes) is passed through *softmax activation function* [36]. This gives $p_n = softmax(a_n)$. However, in the case of binary classification, as in articles 1 and 2 of this thesis, just one node in the last layer suffices (even though we have 2 classes). The real-numbered a_n is transformed into a probability (to range [0,1]) by *sigmoid activation function*. This number is interpreted as the probability of the positive class *P_pos* (*P_virus*). The probability of the input belonging to negative class is simply $1 - P_{pos}$. In the cases of multi-class classification and binary classification the size of error (loss) is usually measured by cross-entropy loss and binary cross entropy (Eq 2.1 above) loss respectively.

Learning in FCNNs. To learn from examples we find the gradients of the loss (MSE, BCE or any other differentiable function) with respect to each parameter in each of the weight matrices and bias vectors used [36, 80]. By changing each parameter value by a small step in the direction opposite to the respective gradient we are likely to decrease the error a little bit. This is the idea behind gradient descent [17, 80]. Iterating over all the training samples we optimize the weights to minimize the errors made in all training points. Minimizing sum error across training data points is the goal of the learning process.

However, different training data points might pull the parameter values in different directions and such noisiness might make the learning progress slower. In *full batch training* the gradients in all training points are calculated and averaged before a learning step is taken, assuring that we change the parameter values to a direction that, on average, improves performance. This is however computationally costly - lots of gradient calculations for one learning step. In *mini-batch training* we calculate the gradients for all parameters for each data sample in a mini-batch (relatively small subset of points) and average over the samples. These averages are a decent estimate of the gradient values we would get by using full-batch. We use these averaged gradient values to update the parameters. By assigning the data points into batches randomly, we reach the stochastic gradient descent (SGD) optimization algorithm [11], where the size of each update is simply the averaged gradient times a learning rate (LR):

$$\Delta_{parameter} = -LR \cdot avg(\frac{\delta L}{\delta \; parameter}) \tag{2.8}$$

More efficient optimization algorithms can reach better results and faster by making the parameter update depend on the history of gradient values over many past mini-batches. Such methods include SGD with momentum, RMSProp [96], Adam [53] and others. These methods help to further reduce noisiness, but also allow to adapt learning speed for each parameter separately depending on the consistency and magnitude of the gradients. RMSprop and Adam optimizers, that are used in this work, use past gradient magnitudes to amplify the updates in parameters where the updates are otherwise small (meaning the learning is slow) and to reduce updates where the updates are otherwise large (risk leading to instability). The Adam optimizer makes use of a concept similar to momentum, making it in theory, but not always in practice, the more efficient. However, depending on model architecture and dataset even the more simple SGD with momentum can in practice still sometimes outperform the other methods [77]. Hence, the choice of optimizer is either quite arbitrary or subject to hyperparameter search. In the contributions presented in this thesis, we have chosen to not hyper-parameter search the most optimal optimizer, due to such search being time-consuming. To save computational effort, we accept the possibility that a slightly more optimal model might be achieved with another optimizer.

Until now we have not mentioned why the learning method of ANNs is often called “backpropagation”. First of all notice that in some of the modern networks, such as the famous object-detection networks AlexNet [55], VGG-Net [90] and ResNet [45], the number of trainable parameters reaches tens or hundreds of millions. To train the network we usually iterate multiple times over the entire training set (a million images in case of these networks). This means the number of gradient calculations needed to train a network might reach billions. Finding these gradients is made computationally more affordable by noticing that to find gradients of the loss w.r.t layer N-1, one only needs to know the gradients in layer N and the relation that ties these two layers.
In particular, using the chain rule:

$$\begin{gathered}\frac{\delta L}{\delta W_n} = \frac{\delta L}{\delta a_n} * \frac{\delta a_n}{\delta W_n} \\ \frac{\delta L}{\delta b_n} = \frac{\delta L}{\delta a_n} * \frac{\delta a_n}{\delta b_n} \\ \frac{\delta L}{\delta a_{n-1}} = \frac{\delta L}{\delta a_n} * \frac{\delta a_n}{\delta a_{n-1}}\end{gathered} \tag{2.9}$$

so knowing $\frac{\delta L}{\delta a_n}$, we just need to find the derivatives of the local relationships $\frac{\delta a_n}{\delta a_{n-1}}$, $\frac{\delta a_n}{\delta W_n}$ and $\frac{\delta a_n}{\delta b_n}$ and multiply them. If the function tying the layers is a simple matrix multiplication plus bias (and no activation function), getting these derivatives is very simple:

Given that $a_n = a_{n-1} * W_n + b_n$, we have

$$\begin{gathered}\frac{\delta L}{\delta a_{n-1}} = \frac{\delta L}{\delta a_n} * \frac{\delta a_n}{\delta a_{n-1}} = \frac{\delta L}{\delta a_n} * W_n^T \\ \frac{\delta L}{\delta W_n} = \frac{\delta L}{\delta a_n} * \frac{\delta a_n}{\delta W_n} = (a_{n-1})^T * \frac{\delta L}{\delta a_n} \\ \frac{\delta L}{\delta b_n} = \frac{\delta L}{\delta a_n}\end{gathered} \tag{2.10}$$

Using activation functions slightly complicates things, but the idea stays the same - we just have another term in the chain rule:

$$\frac{\delta L}{\delta a_{n-1}} = \frac{\delta L}{\delta a_n} * \frac{\delta a_n}{\delta h_{n-1}} * \frac{\delta h_{n-1}}{\delta a_{n-1}} \tag{2.11}$$

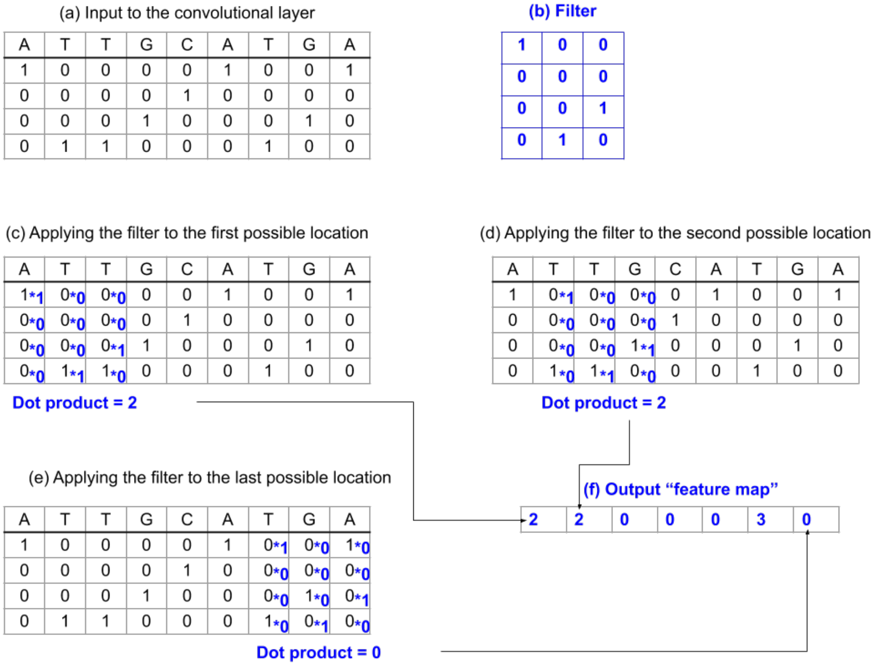
In summary, to find the gradients we can start by calculating the derivatives in the last layer and then move layer by layer back towards the inputs, finding all gradients (w.r.t. all W_i and b_i) needed for learning on the way. This **backward propagation** gives the name to the algorithm.

2.5.2. Convolutional neural networks

Convolutional neural networks (CNNs), used in Chapter 4 of this thesis, are a type of feedforward neural networks similarly to FCNNs. This means there are no cyclic connections in the network and the information moves layer by layer from the inputs towards output nodes. The first notable difference with FCNNs is that CNNs are not fully-connected. While a CNN model might contain FC layers, by definition it also contains *convolutional layers*. In a convolutional layer each node is influenced by only a subset of nodes in the previous layer [36,58]. Furthermore, each parameter of the convolutional layer is shared - it is used repeatedly for a series of calculations (explained in detail below) [36,58]. This allows to reduce the number of learnable parameters and reduce the risk of overfitting to the training data [36].

The particular way of reusing weights consist in convolving a set of weights over the inputs. This set of weights is always used together and is called a filter. Convolving a filter over inputs means calculating dot products between the parameter values in the filter and different possible regions/areas of the input. Each dot product, i.e. each application of the filter, results in one output value (one output node/neuron).

While CNNs are most often used in image processing, here we use them to analyze DNA sequences. Hence to avoid explaining it repeatedly, let us explain the convolution operation by showing it on DNA data. Let us consider a short



(a) Input to the convolutional layer

A	T	T	G	C	A	T	G	A
1	0	0	0	0	1	0	0	1
0	0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	1	0
0	1	1	0	0	0	1	0	0

(b) Filter

1	0	0
0	0	0
0	0	1
0	1	0

(c) Applying the filter to the first possible location

A	T	T	G	C	A	T	G	A
1*1	0*0	0*0	0	0	1	0	0	1
0*0	0*0	0*0	0	1	0	0	0	0
0*0	0*0	0*1	1	0	0	0	1	0
0*0	1*1	1*0	0	0	0	1	0	0

Dot product = 2

(d) Applying the filter to the second possible location

A	T	T	G	C	A	T	G	A
1	0*1	0*0	0*0	0	1	0	0	1
0	0*0	0*0	0*0	1	0	0	0	0
0	0*0	0*0	1*1	0	0	0	1	0
0	1*0	1*1	0*0	0	0	1	0	0

Dot product = 2

(e) Applying the filter to the last possible location

A	T	T	G	C	A	T	G	A
1	0	0	0	0	1	0*1	0*0	1*0
0	0	0	0	1	0	0*0	0*0	0*0
0	0	0	1	0	0	0*0	1*0	0*1
0	1	1	0	0	0	1*0	0*1	0*0

Dot product = 0

(f) Output "feature map"

2	2	0	0	0	3	0

Figure 4. Example of 1D convolution. We are given (a) a 4x9 dimensional input array derived from a DNA sequence, and (b) a 4x3 dimensional convolutional filter. Applying this filter to all possible locations of the input yields a 1x7 feature vector given in (f). (c-e) The elements of the feature vector are obtained by dot products between the filter and different subsets of the input.

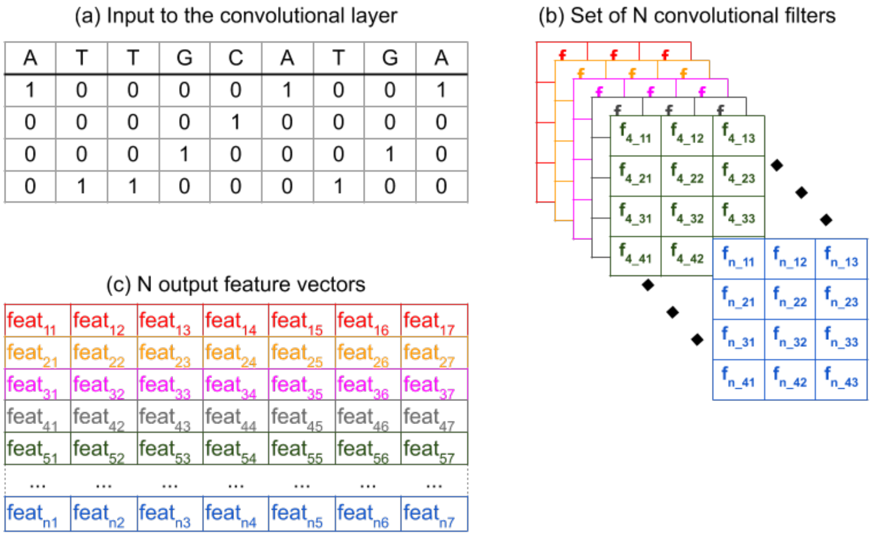
random sequence ATTGCATGA as input. The network needs numbers as input, not letters, so we use one-hot encoding of the letters. This means each letter is made to correspond to a vector containing one 1 and bunch of 0s. The position of this 1 is different for each of the possible letters. For example, we can map:

A->(1,0,0,0)   C->(0,1,0,0)   G->(0,0,1,0)   T->(0,0,0,1)

With this mapping the original short DNA sequence is seen as a matrix of size 4x9, given in Figure 4a.

Now let us consider a set of weights, called a filter, organized also in a matrix form (4x3), as given in Figure 4b. Applying the 4x3 filter to the 4x9 input consists in elementwise multiplying the filter weights with different possible 4x3 areas of the input and summing the results (i.e. dot product, examples shown in Figure 4c-e). In here we apply the filter to all possible locations along the input, but it is possible to skip some if needed [36]. This results in a row-vector given in Figure 4f. In particular, we see that input triplets map to the output as

ATT -> 2, TTG -> 2, TGC -> 0, GCA -> 0,
CAT -> 0, ATG -> 3, TGA -> 0 ,


(a) Input to the convolutional layer
(b) Set of N convolutional filters
f
f_{4_11} f_{4_12} f_{4_13}
f_{4_21} f_{4_22} f_{4_23}
f_{4_31} f_{4_32} f_{4_33}
f_{4_41} f_{4_42}
f_{n_11} f_{n_12} f_{n_13}
f_{n_21} f_{n_22} f_{n_23}
f_{n_31} f_{n_32} f_{n_33}
f_{n_41} f_{n_42} f_{n_43}
(c) N output feature vectors

A	T	T	G	C	A	T	G	A
1	0	0	0	0	1	0	0	1
0	0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	1	0
0	1	1	0	0	0	1	0	0

feat_{11}	feat_{12}	feat_{13}	feat_{14}	feat_{15}	feat_{16}	feat_{17}
feat_{21}	feat_{22}	feat_{23}	feat_{24}	feat_{25}	feat_{26}	feat_{27}
feat_{31}	feat_{32}	feat_{33}	feat_{34}	feat_{35}	feat_{36}	feat_{37}
feat_{41}	feat_{42}	feat_{43}	feat_{44}	feat_{45}	feat_{46}	feat_{47}
feat_{51}	feat_{52}	feat_{53}	feat_{54}	feat_{55}	feat_{56}	feat_{57}
...	...	...	...	...	...	...
feat_{n1}	feat_{n2}	feat_{n3}	feat_{n4}	feat_{n5}	feat_{n6}	feat_{n7}

Figure 5. Multiple convolutional filters form a convolutional layer. (a) The input to the layer is the same as in the figure above - a 4x9 matrix. (b) A set of N filters, 4x3 dimensional, are applied. (c) The output in the presented case is Nx7 dimensional, where each row is calculated using one of the N filters. Notice that a next convolutional layer can be applied directly to this output, with filters of size NxM (where M is the width).

so this example filter gives the largest output in case of ATG, but also triplets with some similarity to ATG (ATT, TTG) result in positive values. Essentially this filter is a sort of ATG detector. We could hand-craft similar detectors for all possible triplets, or for all 4-mers (4x4 filters), 5-mers (4x5) and so on. In practice, a convolutional layer indeed applies many filters (all of the same size though) to the input, not just one. So the output of a convolutional layer is not one vector of filter activations, but a matrix of activations, with one row per filter. This is illustrated on Figure 5 below.

In this example we used a pre-defined filter that gave maximal output in case of detecting ATG as input. The point of CNNs, however, as all ANNs, is to learn the parameters from examples. Hence, in practice we randomly initialize these filters and learn the optimal values via gradient descent, similarly to what was described for FCNNs. The only added difficulty in learning is the fact that each weight influences multiple output values (all values in the feature vector, entire row of output activations) and the gradient calculation needs to sum the gradients. Therefore gradient backpropagation algorithm can still be applied to convolutional layers, despite partial connectivity and re-use of parameters.

Convolutional neural networks often also contain pooling layers. Pooling layers directly follow convolutional layers and decrease the size of feature vectors, but do not decrease their number [36]. The decrease is achieved by summarizing multiple neighbouring values in the feature vector by some fixed (not learnable) function, most often the max function or average function. Using max pooling with pool size 2 and step size 2 on feature vectors, as is often done in literature, means we just take the maximal value from consecutive non-overlapping pairs of activation values. The other value is discarded. This way we would be left with half the amount of nodes after pooling. However, instead of pooling over local pairs it is also possible to pool over all the elements in the feature vector. This leaves just one value per filter (per feature vector) and is called global pooling. We use both global average pooling and global max pooling in different parts of the network in Chapter 4.

In conclusion, a convolutional neural network is a feedforward neural network that contains convolutional layers and might also contain fully-connected layers, pooling layers and other types of layers not mentioned here. The network parameters, including the weight values in convolutional filters, can be optimized via gradient descent.

2.5.3. Recurrent neural networks

Recurrent Neural Networks (RNNs) [80] are the most complicated type of neural networks used in this thesis. Similarly to CNNs they reuse their weights, but in a different manner, i.e. across time (on sequential inputs) rather than across space (as in CNN). As a second difference, the input to an RNN is a type of series - e.g. a series of words in a sentence or, for example, a time series of neuron activations as in the third article of this thesis. The final and the most defining difference with CNNs is that RNNs have cyclic connections. Due to these recurrent connections the state of the network after processing one input influences the way the next input is treated [36, 80].

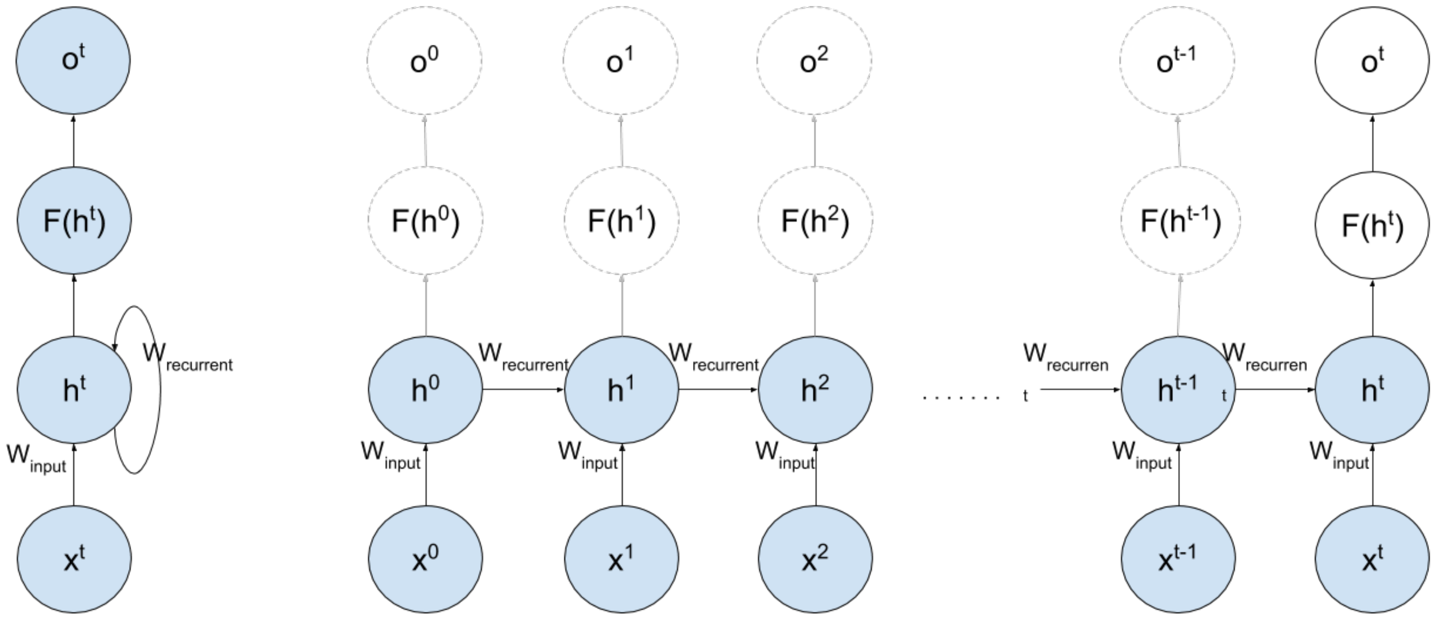


Figure 6. Illustration of recurrent neural network architecture. (a) Recurrent network in condensed form - input at timestep t (the x^t) influences the hidden state h^t via a weight matrix W_{input}. h^t also has a recurrent connection with itself. The hidden state h^t is further processed by some function $F(h^t)$, which yields the output. (b) Recurrent network in expanded view, where each timestep is shown explicitly (often called "unrolled in time" view). In here we see that the recurrent connections correspond to h^{t-1} influencing h^t via a weight matrix $W_{recurrent}$. Applications of $F(.)$ and outputs are dotted for all timesteps but the last, because they might or might not be computed, depending on the way the particular RNN is used.

To understand the basic concepts, let us look at the simplest type of RNNs (often called “simple RNN”), illustrated in Figure 6. In the figure, each circle corresponds to a layer of neurons - i.e. a vector of activations (length >=1).

In the left side of the Figure 6 the RNN is depicted in a condensed form. At each timestep the network receives an input vector x^t and uses weight matrices W_{input} and $W_{recurrent}$ to calculate hidden state h^t. The looping connection on h^t means that h^t also depends on the previous value of h, the h^{t-1}. Similarly, h^t will influence the value of h^{t+1}. However, h^t can at the same time also be used as input for further computations, marked with $F(h^t)$ yielding an output o^t.

In the right panel of Figure 6 the same RNN is “unrolled in time” - instead of summarizing the values at each timestep by x^t and h^t we illustrate each timestep individually and show the connections between them. It can now be clearly seen that h^1 does not only depend on x^1, but also h^0. In can also be clearly seen that the same weight matrices $W_{recurrent}$ and W_{input} are used at each timestep (weight-sharing). Also the function $F(.)$ is the same in all timesteps. Notice that this function can be, and usually is, a neural network. All but the last application of $F(.)$ and all but the last output o are dotted and grayed because they can be removed. With them we have t inputs and t outputs. Without them we have t inputs and 1 output. These two variations of RNN are called *many-to-many* and *many-to-one architecture*. In this thesis we use only many-to-one type of RNNs.

In "simple RNNs" the exact formula for calculating the hidden state h^t at timestep t, given the weight matrices $W_{recurrent}$ and W_{input} and the state at previous time step h^{t-1} is the following:

$$h(t) = tanh(W_{input} \cdot x^t \; + \; W_{recurrent} \cdot \; h^{t-1} \; + \; bias) \tag{2.12}$$

where $tanh$ is the most commonly used activation function, but one could also choose to use other functions [36].

As for FCNNs and CNNs, the training of this network can be done with gradient descent, using backpropagation algorithm to find the gradients (assuming the loss function and the function F are differentiable) [80]. When looking at the network structure in its unrolled form, it looks simply like a very deep fully-connected network with not one input, but a series of inputs added to consecutive layers. Also, the structure is very regular - layer widths are the same and the weights are shared. Especially in the case of many-to-one type of network it is easy to see the equivalence between unrolled RNN and a very deep FCNN. There is only one output from the last timestep, corresponding to the top layer when looking at it as FCNN. The only complicating factor for calculating the gradients is, like it was for convolutional layers, that we need to add up the contributions of the shared weights over all the places they were used in. The fact that there are multiple inputs at different layers does not hinder backpropagation algorithm. Even in many-to-many variation, the comparison with FCNN is valid and backpropagation can be applied. Simply there are multiple outputs, multiple losses and hence multiple sources of gradients that need to be added.

Long short-term memory (LSTM) networks. The simple RNN described above was introduced already in the 80ies by Rummelhart and Hinton [80]. However, researchers soon discovered that these networks are very difficult to optimize [47]. In particular, the simple RNN learns short-term interactions much better than long-term dependencies. So, for example in many-to-one type of network, the model struggles to learn dependencies between the output and the first inputs in the series (far away in terms of timesteps, long-term), but is able to discover dependencies between outputs and the latest inputs (fewer timesteps away, short-term). This effect is mainly caused by vanishing-gradient problem [47] - i.e. gradients getting weaker and weaker the further we go (in terms of layers/timesteps) from the output. To clarify, the model might still be able to learn the long-term dependencies, but the learning is very very slow, so much so that that no-one can actually afford to wait for this learning to happen.

Being unable to learn longer-term interactions means the model is much less useful. Imagine you train a network on a time-series where each input corresponds to one day - simple RNN might manage to discover weekly patterns (7 timesteps), but will fail to detect monthly or yearly patterns. This is a major limitation that was discovered and investigated already in the beginning of 1990ies [8, 26, 73]. In 1997 Hochreiter et al. suggested a more complicated architecture for RNNs, called Long short-term memory (LSTM) [48]. This architecture is famed for making learning of long-term dependencies easier. In simple RNN, essentially, important information contained in the first timesteps is likely overridden by noise in later timesteps [36]. LSTM architecture introduces specific weight matrices that control at each timestep to what extent the network should keep previous information and to what extent overwrite it with new. This way, the network can learn to almost completely ignore some "not-interesting" inputs and instead maintain the information already accumulated.

Introducing these specific weight matrices makes the calculations within a recurrent layer more complicated:

$$\begin{aligned} f^t &= sigmoid(W_f\, x^t + U_f\, h^{t-1} + b_f) \\ i^t &= sigmoid(W_i\, x^t + U_i\, h^{t-1} + b_i) \\ o^t &= sigmoid(W_o\, x^t + U_o\, h^{t-1} + b_o) \\ \tilde{c}^t &= sigmoid(W_c\, x^t + U_c\, h^{t-1} + b_c) \\ c^t &= f^t \cdot c^{t-1} + i^t \cdot \tilde{c}^t \\ h^t &= o^t \cdot tanh(c^t) \end{aligned} \tag{2.13}$$

There are now 8 weight matrices, marked with W and U, this is 4 times more than for simpleRNN ($W_{input}; W_{recurrent}$). The intermediate values have the following interpretation:

- f^t is the forget gate, it controls how much of each element in previous c value (c^{t-1}) is included in c^t ;
- i^t is the input gate, controlling how much of a new candidate value $\tilde{c}^t$ is included in c^t;
- o^t is the output gate, controlling how the internal cell state c is mapped to the hidden state h_t;
- $\tilde{c}^t$ is the candidate value for c_t based on current input and current hidden state h_{t-1};
- c_t is the internal cell state;
- h_t is the hidden state - this is the output value that subsequent layers and subsequent timesteps (gates f,i and o) receive.

As can be seen, the latest hidden state and the current input are combined to decide the states of input and forget gates. The network can learn weights such that given a particular input or particular state of the network, all past information can be forgotten and the new input considered maximally; or all past information can be preserved and input disregarded; or any intermediate combination. This gives the network a lot of freedom to learn what information and how to accumulate to form the internal representation of context.

During learning of an LSTM network, the presence of multiple sources contributing to the cell state c_t (cf. Eq 13) means that the vanishing of the gradients is less likely to happen and is more controllable by the network itself. It has been shown that for the gradients to neither vanish nor explode when backpropagating through many timesteps, the magnitude of the derivative $\frac{\delta c^t}{\delta c_{t-1}}$ (in the case of simple RNN $\frac{\delta h^t}{\delta h_{t-1}}$) needs to equal to 1 (or be very close to 1) on average [73]. It has been shown that this is very hard to achieve in the case of simple RNNs [47,73]. In LSTMs this derivative has proven to take more suitable values more often, leading to better learning of long-term effects, while usually also avoiding gradient explosion. Nevertheless, neither vanishing nor exploding is not explicitly guaranteed to never happen in LSTMs [35, 39, 73], it just happens less often or slowly enough (over more timesteps).

In LSTMs, this partial derivative takes an additive form:

$$\frac{\delta c^t}{\delta c_{t-1}} = f_t + c_{t-1}\frac{\delta f_t}{\delta c_{t-1}} + i_t\frac{\delta \tilde{c}_t}{\delta c_{t-1}} + \tilde{c}_t\frac{\delta i_t}{\delta c_{t-1}} \tag{2.14}$$

The fact that the sum contains the term f^t means that if the gradients are too weak or too strong (vanish or explode), the model can control the strength of $\frac{\delta c^t}{\delta c_{t-1}}$ easily by just changing the scale of the forget gate activations, f_t, which are defined positive in range (0,1). Secondly, the additive nature of the derivative makes it easier to achieve not-too-low values when backpropagating through many timesteps [35, 39].

3. ANALYZING CODON-BASED METAGENOMIC DATA WITH FEEDFORWARD NEURAL NETWORKS ON METAGENOMIC DATA (PUBLICATION I)

This chapter describes the application of feedforward neural networks on a dataset derived from metagenomic sequencing experiments. This work was done in collaboration with Zurab Bzhalava and Joakim Dillner from Karolinska Institute in Stockholm, Sweden. They provided the datasets and the expertise in bioinformatics, needed for the processing steps preceding the application of machine learning methods. As experts in the field, they provided the background knowledge and put the work into larger context in the field of human metagenomics.

The supervised learning goal in this task is to learn a classifier that can separate viral DNA sequences from non-viral. We compare the network's performance with a baseline model - random forest (RF) [13]. We see that the proposed feedforward network does not outperform RF. The eventual error analysis and sensitivity analysis is performed on the RF model, as it is the more well-known and understandable for the readers.

3.1. Biological question

In the "Background on Metagenomics" section we briefly mentioned that sequencing and assembling the DNA found in an environment yields many sequences (called contigs). Some of these contigs we can map to existing genomes by using BLAST (or other methods) - if a contig aligns well with a known sequence, we can say from which organism it originates from. However, many of the contigs do not align well with the genome of any known species. These contigs are marked "unknown" and we literally do not know what they are - even if they belonged to a virus, a bacteria or an eukaryote.

In and on the human body there are many viral species that we do not know yet [65] and in human metagenomic experiments the DNA fragments from these viruses get marked as "unknown". As these yet unknown viruses might be the the hidden causes of some illnesses, virologists are interested in identifying and characterizing them [14, 15, 29]. To learn more about the viruses, the first step is to figure out which of the "unknown" contigs might belong to viruses. Having identified a potentially viral contig, methods exist to go and retrieve the entire genome from the environment. However, these virological methods are expensive and time-consuming (lab-work). One cannot afford to apply them to all "unknown" contigs. It is necessary to narrow down the search and first identify some candidate sequences that we have reason to believe are viral.

In this work we build a recommendation system to find such candidate sequences for further investigation. We create a machine learning model that could tell for any contig, including those marked "unknown" by BLAST, how likely it has belonged to a virus.

3.2. Data and models

3.2.1. Data aquisition and pre-processing

In the genetic code some amino acids are encoded by several, synonymous codons. Usage of these codons is not random and differs among species. This phenomenon is called Codon Usage Bias [5, 89]. How often each possible synonym is used in a given open reading frame (ORF) is measured by relative synonymous codon usage [88]:

$$RSCU = \frac{C_i}{\frac{1}{N_i}\sum_{j=1}^{N_i} C_j}, \tag{3.1}$$

where i is a codon, N_i is the number of synonyms it has and C_i and C_j are counts. Hence, in case all synonyms are equally used, the RSCU value will tend towards 1. RSCU values cannot me calculated for codons with no synonyms, so while there are 64 codons, there are 59 RSCU values.

In this work we used RSCU values extracted from contigs as the input to our machine learning algorithms. In particular, given a set of contigs:

1. we label the contigs with BLAST and HHMER3 [92] models, as "virus" or "non-virus". There are many contigs originating from human genome, bacteria and phages. These are removed. Also, sequences too similar to each other (duplicates) are removed.
2. we generate the reverse strand for each contig and use it as an independent sample
3. all sequences with an open reading frame (ORF) of length at least 120 bp (base pairs) are kept, other contigs are discarded
4. we calculate the RSCU values in the ORFs of each contig. This means we summarize each DNA sequence in 59 RSCU values
5. the RSCU value vector and the label given by BLAST/HMMER3 make up a data point.

We apply this procedure to contigs originating from 19 metagenomic sequencing experiments. In these experiments human metagenome in different parts of human body (serum, prostate, skin, cervix, etc) was sequenced. The sequencing was done with MiSeq, NextSeq and HiSeq (Illumina) sequencing platforms. The amount of obtained data points and the prevalence of viral sequences varies a lot from experiment to experiment. In total we have roughly 200 000 data points, 5000 of which are labelled as virus.

3.2.2. Data partitioning

Despite removing very similar sequences (duplicates) in the pre-processing phase, we noticed that sequences originating from the same metagenomic experiment still share more similarity with each other than with sequences from other experiments (measured by Hamming distance between sequences, cosine distance between RSCU vectors). Consider a pair of highly similar data points (originating from the same experiment) one in training set and one in validation set. The validation performance on this validation point is likely high, as a very similar sample was used during training. In the context of building a recommendation system that can be applied to new data from any new metagenomic sequencing run (that an user might have), however, this performance would be an overestimation. As said, another metagenomic experiment would not contain such easy-to-classify similar data points. Hence, to give an honest estimation of our model's generalization ability, we partition the data in a way that the validation set never contains data points from the same metagenomic experiment as train set.

In particular we use so called "leave-one-experiment-out" (LOEO) cross validation procedure. We have 19 metagenomic experiments. We train 19 models, each time leaving one of the experiments out of the training set and using it as the validation set. The training is done on the 18 remaining sets. The validation sets are of different size and prevalence and their results need to be combined to give a unique measure of generalization ability. For completeness, we use both micro and macro averaged measures.

3.2.3. Machine Learning algorithms used

To learn the function that maps RSCU values to the probability of viralness of the sequence, we first trained a random forest (RF) classifiers [13]. We tested various sizes of forests and also experimented with other hyperparametes, including class weight balancing and up/down sampling. However, the only improvement in validation set AUROCs was achieved by increasing forest size to 1000 trees. Further increase in size improved performance only marginally, with increased computational cost. Other changes in hyperparameters either lowered AUROC or just changed the precision-recall trade-off with AUROC staying the same.

We then investigated if fully-connected neural networks (FCNN) can outperform random forests in this prediction task. We varied network depth, layer sizes, dropout rate, learning rate, activation function, batch size and optimization method. Also, as for random forest, we tried using class weight balancing and up/down sampling. In all, a network with two hidden layers (relatively shallow) proved to perform the best. The model was not very sensitive to hyperparameters and many other model configurations performed almost equally well. The exact model used is summarized in the Appendix.

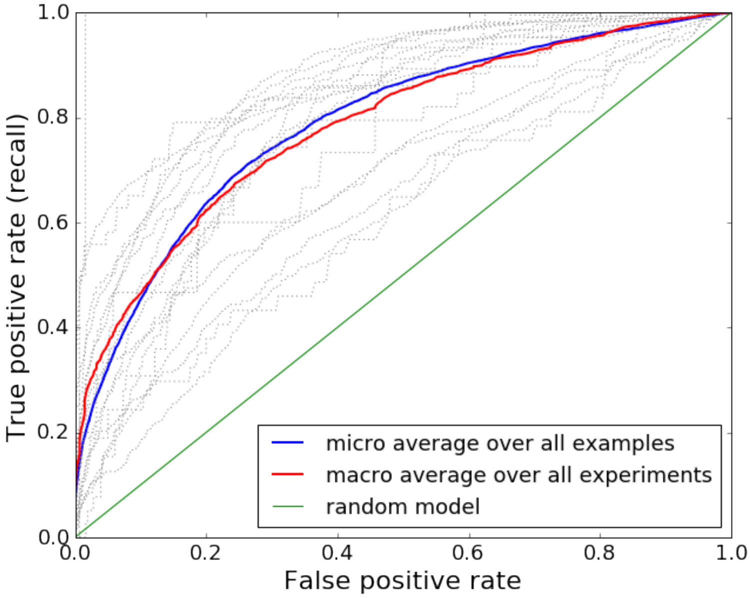


Figure 7. ROC curves of the 19 models of the leave-one-experiment-out procedure on their respective validation sets (gray dotted lines). The micro and macro averaged ROC curves are given in blue and red, respectively.

3.3. Viral nature can be predicted

For each of the 19 datasets, with both random forest and FCNNs, the model achieved predictive performance clearly above random level. This means that RSCU values contain at least some information about if the sequence is viral or not. However, the results varied a lot from validation set to validation set (remember, each dataset was left as validation set once). With random forest the highest validation AUROC among the 19 models (validation sets) was 0.98 and the lowest 0.59. Averaging the results by merging all test sequences and predictions into one big dataset (micro-averaging), we achieved AUROC 0.789 with RF and 0.790 with FCNN. Macro-averaged (averaged over the 19 mean results, one per test set) are also similar (around AUROC 0.785). We conclude that the results are essentially equal with the two ML methods. We decided to present random forest results as the main results, because RF algorithm is more established and easier to understand and interpret for a wider audience. The ROC curves for all 19 validation sets, the micro and macro averages are drawn on Figure 7.

As mentioned in the "Losses and metrics" subsection in the Background, the precision and recall values depend on prevalence and the choice of classification threshold. Also, we argued that due to high class imbalance we care about the precision and recall of the viral class, not the overall measures. Our RF models can, with different thresholds, achieve for the positive class (viruses) [precision, recall] pairs [0.95, 0.055], [0.9, 0.086] and [0.75, 0.105]. We expect the intended

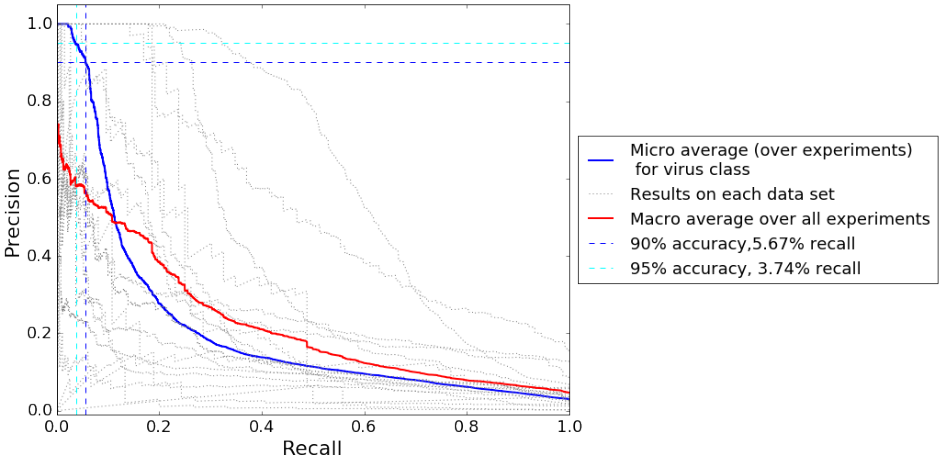


Figure 8. Precision-recall curves for the viral class. Gray dotted lines illustrate the performance of the 19 models on their respective validation sets. Blue and red lines are the micro and macro averages.

users to be more interested in fewer (low recall) but more accurate results, but it might depend on use-case. Hence the full precision-recall curve is provided (Figure 8).

Sensitivity analysis of RF model revealed that codons TCG, CGC, CGA, GCG, GTA and CCG stand out as more important than others. All of these codons were relatively more prevalent among viruses (higher RSCU) than among non-viral samples. Also in absolute terms, these codons were among the least commonly found codons (as per RSCU values) in non-viral sequences. Why such differences in codon usage exist and how to further exploit this is an interesting future research topic for virologists.

3.4. Discussion

In this work, we used RSCU values extracted from metagenomic contigs as input to binary(viral/non-viral) classifiers based on fully-connected neural networks and random forests. When extracting features from some input, we make decisions what information to keep and what to discard. For example, in RSCU values most of positional information is lost, because we count the codons and do not pay attention to their order. Also, as we only count the relative frequency of codons (3mers), we do not consider longer patterns. In short, we have a priori decided that RSCU values is a good representation of the input sequences and the models do not have freedom to change that decision.

However, notice that the most quoted ability that sets neural networks apart from other algorithms is the ability to learn useful hierarchical representations of the data [10, 36, 80]. Indeed, the seminal Rummelhart and Hinton paper in

1986 was named "Learning representations by back-propagating errors" [80]. In consecutive layers, the networks can transform the original input in non-linear fashion and map it to some feature space, where the final classification layer can more easily separate the classes [36]. In here, the input sequences are already preprocessed and mapped to a certain feature space (RSCU-space), hence the ability to find good representations is less useful. With this observation in mind, it is not surprising that FCNN did not significantly outperform random forests. In fact, having optimized both methods extensively and having reached the same result, it is possible that both methods have used the information contained in RSCU values near optimally. It is possible that based on this high-level input, no other model could do much better.

The RSCU values can be extracted from all ORF-containing contigs found in the metagenomic experiments, including the ones marked "unknown" by conventional methods. The proposed RF and FCNN models can be applied to these sequences (these RSCU value-vectors). The predictions they return, even if not perfectly reliable, can be used to order the unknown sequences by likelihood of being viral. This way, virologists can prioritize the study of sequences that are more likely to be viruses, saving time and money compared to randomly picking the sequences to be studied further.

4. ANALYZING RAW DNA SEQUENCES FROM METAGENOMIC EXPERIMENTS WITH CONVOLUTIONAL NEURAL NETWORKS (PUBLICATION II)

In this section we investigate further the problem of identifying viral sequences among metagenomic contigs. In particular, having assessed the weaknesses of the methods used in the previous chapter (Publication I), we propose to use convolutional neural networks to tackle this prediction task.

As for the previous chapter, the work was done in collaboration with Zurab Bzhalava and Joakim Dillner from Karolinska Institute in Stockholm, Sweden.

4.1. Biological question

In the first publication we demonstrated that machine learning models can learn to detect the viral nature of a DNA sequence based on only the information contained within that sequence (without using an external database, see also [78]). In here, we ask if the prediction accuracy can be further improved by changing the way the inputs are pre-processed.

In particular, we hypothesize that by reducing a DNA sequence (contig) to 59 RSCU values (or k-mer counts, as in [78]), as was done above, one discards a lot of information that could be useful for the classification task. Instead, in this second publication we propose an approach where the machine learning model can access all information within the sequence and can decide by itself what information to extract and use. We use convolutional neural networks that apply learnable filters directly one the "raw" DNA sequence. These filters are optimized through gradient descent and can learn to extract more diverse information than k-mer counts or RSCU values.

4.2. Data preprocessing

Our CNN models will work on "raw" sequences, so the pre-processing pipeline does not contain any feature extraction:

1. we label the contigs with BLAST and HHMER3 models, as "virus" or "non-virus". There are many contigs originating from human genome, bacteria and phages. These are removed. Also, sequences too similar to each other (duplicates) are removed
2. all sequences shorter than 300 bp are discarded. All longer sequences are cut into pieces of length 300, with the remainders discarded. E.g. a sequence of length 700 yields two non-overlapping sequences of length 300 and a remainder of 100 bp that is discarded.

3. a 300 bp sequence and the label given by BLAST/HMMER3 forms a data point

The baseline models use k-mer frequencies as inputs, in that case an additional step of k-mer counting is included (i.e. feature extraction).

In contrast to the previous publication, in this work we did not generate the complementary sequence for each contig for using it as an independent sample. First of all, we had sufficient data and did not need this data augmentation technique. Secondly notice that there is no new information in the complementary sequence compared to the original one. In terms of information content, the complementary sequence is a duplicate of the original data point. With random data partitioning, used in this work, such duplicates would be split between training validation and test. This could lead to overestimating the generalization ability. As for the main results of this chapter we use 80/10/10 split of the data, we judged safest not to include the complementary sequences (to avoid reporting overestimated results).

4.3. CNN model and baseline ML algorithms

When deciding the CNN architecture to use, we analyzed the weaknesses of count-based (k-mer, RSCU) models. First of all, notice that the number of possible k-mers grows exponentially with increasing k. In this work we allow 5 possible values for each position (including "N" for "unknown"). The highest k value used was 7, which transforms a 300 bp input sequence to a vector of length 78125. Such representation is very sparse and introduces many parameters to any machine learning model built on it, which in turn can lead to overfitting. Also, having so many inputs makes any model computationally more costly (which is why we stopped at k=7). Furthermore, notice that despite using 78125 input features, the presence of patterns longer than 7 bp is not directly readable from this representation. While the presence of a certain 8-mer can almost always be deduced from observing the two 7-mers it contains, needing to learn such "AND" rule makes the learning task more complicated for the machine learning algorithm. We hypothesize that a more efficient representation of the input sequence exists and propose that a CNN might be able to learn it.

We also hypothesize that for some particular patterns detecting just the presence, not the exact count, is enough. Whereas presence can always be deducted from count (count > 0), an algorithm might or might not be able to learn to consider just the presence and not the magnitude of the count. Hence, we hypothesize that bestowing a CNN model with the ability to detect just the presence of a pattern, not its usage frequency, might be beneficial.

In this work we designed a two-branched CNN, called ViraMiner, that is capable of extracting both pattern frequency information and pattern presence information. This network receives as input 300 base-pair long DNA sequences (contigs) encoded in one-hot format (example given in Figure 9).

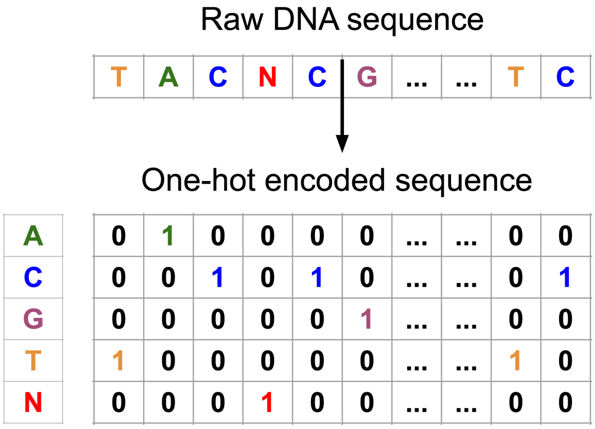

Raw DNA sequence
T A C N C G ... ... T C
One-hot encoded sequence
A 0 1 0 0 0 0 ... ... 0 0
C 0 0 1 0 1 0 ... ... 0 1
G 0 0 0 0 0 1 ... ... 0 0
T 1 0 0 0 0 0 ... ... 1 0
N 0 0 0 1 0 0 ... ... 0 0

Figure 9. Example of one-hot encoding transforming a DNA sequence of length L base-pairs into 5xL matrix.

The overall structure of this proposed ViraMiner model is illustrated in Figure 10. In here, we do not specify the layer hyperparameter values (layer sizes) as they will be subject to hyperparameter tuning. The model receives the one-hot encoded DNA sequence and returns one real number - the probability of this input sequence being of viral origin. During optimization, the binary cross-entropy (Eq 2.1 in Methods) loss is minimized between the output probabilities and the true labels.

The "frequency branch" contains a 1D convolutional layer followed by global-average-pooling layer. While the convolutional layer outputs multiple values per convolutional filter (each filter is applied to multiple places along the input), the global averaging reduces these to just the average value per filter. As a result, this pair of layers measures the average presence of a pattern similarly to k-mer frequency extraction, only that the patterns are more complex and the matching in each position is a continuous measure, not binary. In fact, the frequency branch is also capable of learning to just count k-mers if it deems it necessary. In such case the filters would have one-hot vectors as columns.

The "pattern branch" contains a 1D convolutional layer followed by global-max-pooling operation. Due to using maximum operation instead of averaging, this combination of layers detects how well (maximally) a learned pattern can be matched with the input sequence.

To conclude, the proposed approach is able to learn by itself the complex patterns the layers should look for in the data and is able to detect both presence and usage frequency. As an additional benefit, the number of used parameters in the convolutional layers grows linearly with the filter-size used, not exponentially. This means we can experiment with very high filter sizes (up to 40 bp, as opposed to k=7 with k-mer counts) and decide empirically if considering longer patterns matters for the classification task.

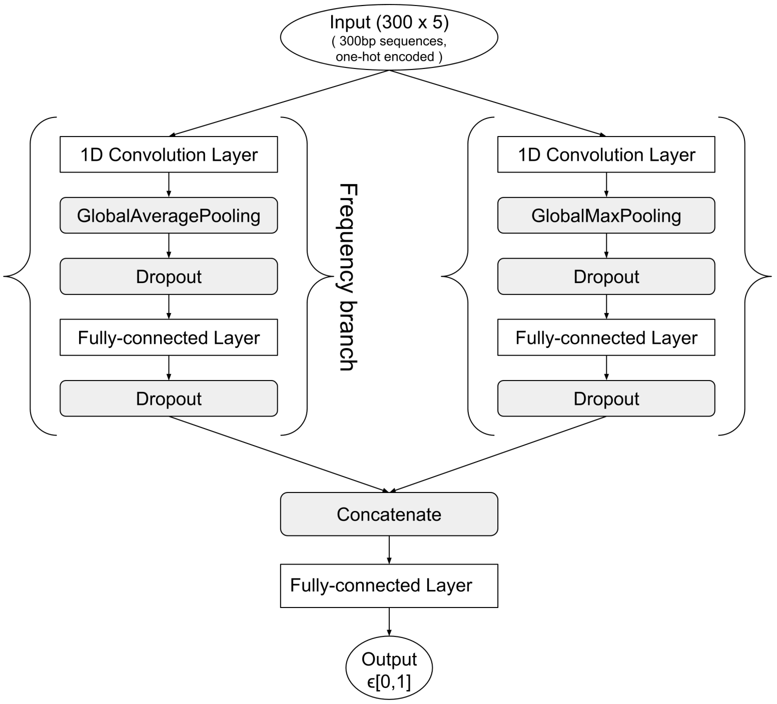


Figure 10. ViraMiner architecture without exact hyperparameter values. The hyperparameters are tuned for each branch separately.

In hyperparameter search we find the optimal values for the number and size of the convolutional filters, as well as other tunable parameters (dropout rate, learning rate, FC-layer sizes, etc). However, to reduce the number of possible hyperparameter combinations, we tune the sizes of each branch separately. Notice that by just removing one of the branches (and half of the concatenated layer), we still have a fully functioning classifier. We call these one-branched models "frequency model" and "pattern model". We fit many instances of these partial models with a wide range of filter sizes and counts. We then pick for both partial architectures the models that yielded the highest validation AUROC. We can use these partial models to produce the full two-branched ViraMiner model in different ways:

1. we use the best hyperparameters for each partial model to set the sizes of the branches in ViraMiner model. We then randomly initialize the weights and fit the entire two-branched model from scratch

2. we use the best hyperparameters to set the sizes of branches in ViraMiner model. In addition, we initialize the weights in the branches to be the weights from the best partial models. The last fully-connected layer (after concatenation) is randomly initialized. We can then train the ViraMiner

model in three different ways:

(a) optimize all the weights together, both the randomly initialized last layer and the branches
(b) optimize only the last layer of the model (leaving the parameter values in branches fixed)
(c) optimize the weights in two steps. First only optimize the last layer. In a second step fine-tune all weights of the model together

All tunable hyperparameters are defined by the partial models, the only choice made at the full model level is evaluating which of the three above-mentioned ways of using partial model is the most effective.

The ViraMiner model's exact shape (layer, filter sizes etc.), as defined by best partial model shapes, is given in the Appendix A.

As the main, most competitive baseline algorithm we used Random Forest. To demonstrate that RFs are unable to learn from the raw sequence, we first trained them with one-hot encoded sequences as input. We attribute RF's inability to work with raw sequences to the fact that RF has no built-in mechanism to achieve space-invariance (such as global pooling in our CNN model). This means that shifting a sequence by one value (removing one value from the beginning and placing it to the end) makes it unrecognizable for random forest, as all input values will change. To have a more competitive baseline, we also trained RF models using k-mer counts as input.

4.4. Improved performance in viral classification task

Trained on a 80/10/10 split of the data, CNN-based models clearly outperform random forest models (Figure 11). Random forest trained with raw sequences as input only achieved AUROC 0.573. The best random forest model trained on extracted k-mers reached AUROC 0.875. The one-branched pattern and frequency CNN models surpassed this performance, reaching 0.905 and 0.917 AUROC respectively. The best result, AUROC 0.923, was reached with the two-branched ViraMiner architecture. The best ViraMiner model's branches were initialized with weights from the best partial models and only the last layer was optimized (option 2b from previous subsection). When used as recommendation system, this performance translates into getting 166 out of 200 top predictions correct, whereas the top 20 are all correct.

Using CNN-based classifiers clearly improves the performance over models with k-mer counts as inputs. This supports our initial hypothesis that more optimal representations of DNA sequences exist and that CNNs can learn them.

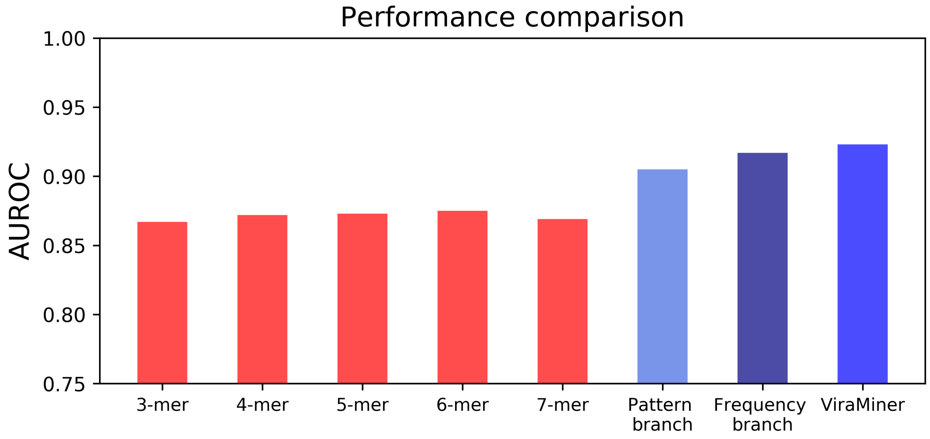


Figure 11. Area under ROC curve for baseline random forest models, partial CNN models and the two-branched ViraMiner model. CNN-based models (blue) outperform RF baselines (red).

4.4.1. Additional measures of generalization ability

The additional experiments described in this section were performed to demonstrate i) that ViraMiner's high performance is not due to hidden biases in our dataset and ii) that the model can generalize to outside its training domain, i.e. to extrapolate.

We ran the following experiments:

- Training the model on simulated data, using simulated validation set for early stopping and simulated test data for performance estimation. The test results are equivalent (AUROC 0.93) with the main results on real data. This shows the methods work on a dataset that we know exactly how it was generated and that should be bias-free.
- Training a model on data originating from 18 metagenomic experiments and testing it on the 19th. We repeated it 5 times, each time leaving out one of the metagenomic sets from human serum. The results consistently show that models trained on 18 experiments generalize very well on the 19th (AUROCs range from 0.84 to 0.98, micro-average equals 0.94).
- Training a model on data from all 19 experiments, but with a certain viral class, anelloviruses, removed from the training and validation sets. We test this model on a test set containing non-viral samples and only anellovirus samples (all other viral samples were moved to train and validation). A significant performance, AUROC 0.78, shows that the model can identify even viral samples that are very distant from the training viral contigs. This performance translates to getting 11 of the top 20 "most viral" predictions correct.
- Training a ViraMiner model on the 19 real metagenomic datasets and test-

ing it on simulated data. The resulting AUROC=0.78 is similar to the results on unseen viral class (the previous item in this list). This similarity is likely due to the simulated data containing viruses from randomly selected viral families, including those not present in the training data.

In all, the model generalized to all tested cases. It is not uncommon for a model to not generalize at all outside its training domain (only interpolate, not extrapolate), hence it is particularly pleasing that ViraMiner obtains a useful performance on sequences from completely unseen viral class.

4.5. Discussion

4.5.1. Discussion of results

The appeal of using k-mer counts as input features is the fact they are so universal - they are used in very different DNA-decoding tasks and are generally seen as a reasonable set of features to extract. In contrast, the features that our model's convolutional layers extract are task-specific. They have been optimized to maximize performance in this specific task of virus identification on this specific data type. If applied to another task (e.g. for predicting if the contig contains a coding gene) or different data (e.g. to discover bacterial or plant viruses), the learned-features of our model are probably sub-optimal. Task-specific means less general, but also gives the possibility of being more efficient, if the architecture and optimization allow it. As an additional benefit, using convolution allows to consider longer patterns - the number of possible k-mers grows exponentially, while the number of parameters in our filters increases linearly with the pattern length (i.e. value of k). Our model's strength lies in allowing to detect longer and more task-specific patterns than k-mer counting.

The improved classification performance allows us to indeed claim that CNNs were able to extract more optimal features from the raw input than k-mer counts or RSCU values. The AUROCs achieved here were also superior to the results in Chapter 3, where we used relative synonymous codon usage as the model input.

We also demonstrated that the ViraMiner model generalizes well to unseen data of different types. First of all, it achieves 0.923 AUROC on test set samples. Secondly, we see that even on data originating from a new metagenomic experiment, the performance remains very high (0.94 average AUROC across serum experiments). Thirdly, even when applied to viruses from an unseen viral family, the performance is well beyond chance level with AUROC 0.755. This ability to generalize to viral types not present in the training and validation data is surprising, because machine learning models are often not good in extrapolation. Finally, we also confirmed that the ViraMiner methodology performs well on simulated data generated via a controllably unbiased mechanism.

Coming up with a CNN architecture that has such good performance and generalization ability on our task was not straight-forward. In an initial stage of this project we also experimented with a convolutional architecture originally designed for text-classification based on character-level inputs. While proven good at understanding natural language, this architecture performed equally with k-mer based random forest models in our DNA classification task, i.e. clearly worse than our two-branched ViraMiner model. In contrast to just exporting an architecture from another domain, ViraMiner model was designed based on biological expertise. First, we consider convolution followed by average pooling (frequency branch) as a direct generalization of k-mer counting - instead of counting exact letter-by-letter matches we measure average match with complex, learned patterns (the filters). Second, we consider convolution followed by max pooling (pattern branch) important to remedy the limitations of averaging in the frequency branch. Specific patterns such as regulatory sequences might appear only once in any given sequence, but carry a lot of information as they can be different across organisms. The averaging would dilute the information about a very good match in one position and make it indistinguishable from a simply slightly above average match in many positions. In contrast, maximum pooling preserves the information about extremes and allows the network to look for some patterns very specifically. Ablation studies show that indeed both of these branches matter for obtaining the best results. We consider the ViraMiner architecture itself a main contribution that can be re-used in other DNA-analysis tasks.

The other main output of our work is the trained ViraMiner model. It is intended to be used as a recommendation system applied to the "unknown" sequences. Methods that compare metagenomics contigs with a reference database (BLAST, HMMER3) will remain the default first step when trying to separate viral sequences from non-viral. These methods can identify known sequences and their not very distant homologues, but still leave many contigs unlabeled. These contigs contain DNA from the truly unknown species. ViraMiner model can be applied to these contigs - the viralness probability can be computed for all sufficiently long (at least 300b) sequences. We envision that ViraMiner will be used as a recommendation system to sort the "unknown" sequences by their likelihood of being viral. The most probably viral sequences can then be further studied by computational methods or in the lab.

4.5.2. Future work

The capability of the recommendation system based on our trained ViraMiner model can be further improved. First of all, the complementary sequence can be generated for each of our test sequences and the model applied to that sequence. The two probability estimates for the same 300bp long input contig can be averaged or combined in some other way. In the complementary strand there is no new information that is not contained in the original sequence and an optimal model should give equal probabilities for the two strands. However, in practice all models are noisy as they have been trained on incomplete, noisy data and by using optimization that is not guaranteed to converge to the global minima. We assume also our model is noisy at least to some extent and ensembling over the two estimates - on the two strands - could average out the noise and improve performance. Measuring the difference of the estimates would also give us a better understanding of the uncertainty in our model's predictions and if there is room for improvement (via training on more data for example, or ensembling over many instances of the model).

The second potential performance improvement also relies on ensembling. In our pipeline, a longer sequence is cut into 300bp pieces and can yield mutliple data points to our data set. At the moment, the fact that certain data points actually refer to the same original contig is ignored. However, when ensembling the results over all pieces originating from the same original sequence, a performance gain can be expected. This step was not implemented in the current work, because the number of such longer sequences is low and the overall performance gain is expected to be small. However, such analysis helps us understand how information is distributed along a longer sequence. If the information about viralness is not equally distributed, the output probabilities for individual 300 bp regions would differ significantly. It might help reveal what are the characteristics that make a part of the sequence more informative than others. In this sense, it can result in a type of sensitivity analysis or error analysis.

Error and sensitivity analysis were not performed in the present work, because we decided the main results were convincing enough alone. Adding such analysis would have shifted attention from the main results. However, to understand better what sets viral and non-viral sequences apart, such analysis can be performed in a future publication.

In the field of natural language processing, different types of sensitivity analysis have been developed to interpret the inner workings of structurally similar models with similar type of inputs (one-hot). For example gradient-based and counterfactual-based approaches might help reveal which patterns matter most for accurate classification in our model. Changing the model to include attention layers opens another avenue for revealing the most influential sub-sequences.

A type of sensitivity analysis is also possible by studying the convolutional filters our model learned. By looking at the maximal and minimal values of each column in a filter, the filter weights directly reveal what input pattern would maximally and minimally activate it. However, there are 2200 filters in the two branches of ViraMiner in total and it is unfeasible to visualize them all. Even if these extreme patterns could all be studied, they are tricky to interpret. The effect of one filter's output on the final prediction depends on the context of all other filters' outputs - there are two more hidden layers after convolution. As the interpretation of such study of filter weights is not straight-forward, we judged it to be more confusing than useful and decided not to include it in our work.

Applying our trained model or the model architecture to new data types is another avenue for future work. Recently, the research group where our data originates from in Karolinska Institute turned their attention on RNA-metagenomics, which reveals not all DNA in the environment, but the active genes that get transcribed. Our DNA-trained model might or might not generalize to the RNA sequences. An evaluation if DNA model is transferable to RNA data must be performed. If needed, a new model on RNA data must be trained. We see no reason why the model architecture is not suitable for RNA data and should not work well.

Similarly, the model could be evaluated or a new model trained on unprocessed reads of the sequencing machines, that in some cases are not much shorter compared to the 300 bp contigs used here. A model capable of predicting the origin of raw reads might help speed up assembly and reduce assembly errors. However, if only a small fraction of reads are viral (prevalence was 2% here, but is probably even lower on read level), these speed benefits might be minimal. Nevertheless, showing the virus identification ability on raw reads level would be an interesting proof of concept that could inspire multi-class discriminators specifically aimed for being helpful for assembly. In this work we used assembled data, not raw reads, as it was available to us and we assumed that longer sequences are more informative and result in a more precise model.

4.5.3. Limitations

We present ViraMiner architecture itself as a contribution in this work. No model is universally useful and we wish to define more clearly the area of applicability of our proposed architecture.

We did not endow our model with the ability to consider the long-scale order of patterns in the sequence. In particular, using global pooling on convolutional feature vectors, we lose information about how well the filter matched in different areas of the input. The order and distance between functional regions might actually matter for our task or other classification tasks, but our approach is unable to consider it. The decision to not consider this information in our model was conscious. When using local pooling in order to keep more than one value per convolutional filter, the number of output nodes and number of parameters grows linearly. Considering that we work with random pieces from genomes (not necessarily coding or regulatory regions) we judged this source of information is not crucial for the current task and does not justify the growth in model complexity (number of parameters).

This limitation demonstrates that the ViraMiner architecture is not the universal approach for all tasks of decoding information from DNA. In other tasks the positioning of patterns along the sequence might matter more. Consider the case where input data corresponds to a cut-out area just before the start codon. The area contains specific regulatory regions and the exact position of them might matter. For extracting such location information, the ViraMiner architecture could be modified by replacing global pooling with local pooling or no pooling at all. It would however be more reasonable to start from scratch and build a specific architecture for the task at hand.

Recurrent neural networks are useful for another distinct set of decoding tasks, where ViraMiner architecture would probably not be optimal. RNNs receive inputs in a sequence and by construction rely on the most recent information more heavily. Hence, recurrent networks should be applied if we have reason to believe the latest inputs are somehow more relevant. Such problems are, for example, predicting methylation levels or predicting single-nucleotide polymorphisms - obviously the base we are predicting for and its immediate neighbours are more relevant than the more distant values that just help form a context. In contrast to RNNs, ViraMiner by construction considers the entire input sequence equally relevant, due to global pooling layers.

There is a diversity of prediction problems in bioinformatics and the choice of neural network to apply must rely on previous knowledge about what type of models have been successful in the past and what type of information matters. While not generalizable to all tasks, we believe that ViraMiner architecture is a direct improvement and replacement for k-mer counting. It is reasonable to believe ViraMiner would improve performance in tasks where currently k-mer counts are used as the default inputs. In future work, a more generalized claim could be made about the information extraction ability of ViraMiner, k-mers and other feature extraction methods by comparing their capabilities on simulated data. Something similar was done in the early stages of this work (unpublished) to test our architecture.

5. APPLYING RECURRENT NEURAL NETWORKS ON SINGLE-NEURON RECORDINGS (PUBLICATION III)

In this section we describe the application of recurrent neural networks (RNNs) to decoding information from neural recordings. This is the technically most complicated contribution, both because the data type is challenging and because the applied model is hard to understand without background knowledge. Nevertheless, the aim of this work was to demonstrate that despite its apparent complexity, applying recurrent neural networks on neuronal activity timeseries is worthwhile.

This work was done in collaboration with Caswell Barry and Freyja Ólafsdóttir from University College London. Their side recorded the data and preprocessed it into a list of spiking times of each neuron (spike detection and sorting). They performed the extensive baseline analysis using Bayesian models. As expert in the field, Caswell wrote most of the neuroscience-related sections.

5.1. Neuroscience question

In this work we decode the locations of 5 rats from the activation patterns of their hippocampal region CA1 neurons. The rats are trained to cover a lot of ground during the recording sessions, so that they would visit all possible locations, an example trajectory in 1x1 m arena is given in Figure 12.

From existing publications it is known that such decoding can be done, and that it can be done with good accuracy [100, 106]. It is not known how precisely the animal itself knows where it is located - so the accuracy with which self-location

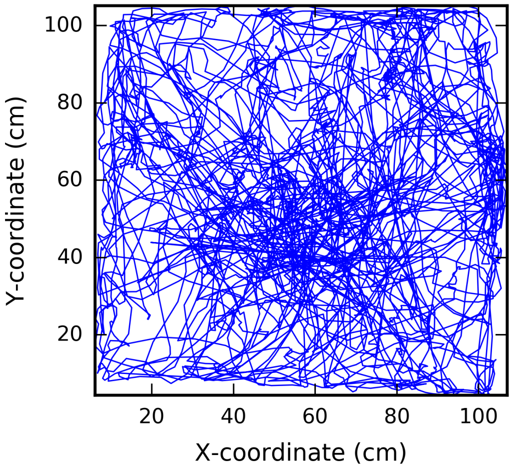


Figure 12. Locations visited (the trajectory) by the rat R2192 in 1x1 m arena during the 20 minutes recorded.

can be decoded from the brain places a useful lower limit on the amount of information present [100, 106]. Assuming the animal's brain is as efficient as our model, the animal knows its position at least as precisely as our model. However, in this work our main goal goes beyond simply improving on the prediction accuracy compared to previous publications - with different input data, the results are not directly comparable anyway.

Instead of minimizing error at all costs, the primary goal of this project was to show that a relatively standard RNN model is capable of efficiently using the very noisy single neuron recording data. In particular, we hypothesized that the past neuronal activity can be accumulated in the network and acts as a context to help decode the current location. This is in sharp contrast with the most commonly used decoding method in the field on single neuron recordings, Bayesian decoding [23, 98, 106]. In Bayesian approaches the context is either not considered or considered via building a complicated prior [106]. The former (no context) means using a flat prior, resulting in the maximum likelihood estimation (MLE) baseline model. The latter (with complicated prior) requires a good understanding of the data we work with and the mathematics used. We call the resulting second baseline model "Bayesian with memory".

By demonstrating that RNNs can outperform both of these baseline models, we aimed to showcase that the ability to use contextual information free of any prior choices and restrictions (that RNN does) can be more practical than optimally using the current information (that Bayesian does, supposing we got the assumptions right [64]). The brain functions in time and undeniably uses context in some form, so the models we use to study the brain should do that too. Otherwise we leave an important source of information unused.

As more directly neuroscience-related goals we also aimed to investigate which neurons contribute most to the position-decoding and if there are common descriptors for the most informative neurons (neuron type, firing rate, receptive field). We analyze the errors our RNN models make to find where (e.g. near the walls?) and when (e.g. when animal moves fast?) the errors increase.

5.1.1. Baseline Bayesian models

When decoding some variable from the activity of neurons, one usually has no clear idea what prior distribution to use. Hence a flat prior is applied and the Bayesian decoding is equivalent to to maximum likelihood estimation with the activity within some time window as input. This model does not consider any past neuronal activity and hence cannot use contextual information.

The likelihood of observing a spike count vector $K = (k_1, \ldots, k_i, \ldots, k_N)$ from the N neurons in a timewindow of length T given position (x) is:

$$P(\mathbf{K}|x) = \prod_{i=1}^{N} Poisson(k_i, T\alpha_i(x)) = \prod_{i=1}^{N} \frac{(T \times \alpha_i(x))^{k_i}}{k_i!} \times e^{-T\alpha_i(x)} \tag{5.1}$$

Considering a flat prior and using the Bayes rule, we can derive the probability of the animal being in location x, given the observed firing pattern K:

$$P(x|\mathbf{K}) = R \prod_{i=1}^{N} Poisson(k_i, T\alpha_i(x)) \tag{5.2}$$

where R is a normalizing constant depending on T and firing rate. See for further details in the publication.

Notice that in here we have decided to model neurons' firing as a Poisson process, which pre-defines how noise affects the result (and which is probably not perfectly correct [106]). In contrast, a RNN is free to choose how to interpret its inputs - e.g. ignore some inputs if they are unreliable. This MLE approach (also called "simple Bayesian" in our work) is our first and most important baseline model, as it is the standard approach in the field.

Additionally, in the second baseline model called "Bayesian with memory" we endow the Bayesian model with ability to use context. We compute an informed prior distribution using:

- knowledge about the locations the animal most often visits (during the entire trial, using also future locations) summarized in $p(x)$,
- a continuity constraint based on the position decoded in previous timestep $\hat{x}_{t-1}$ and the recent average movement speed.

This constraint ensures the previous and current position predictions are not further apart than what could be expected considering the animal's movement speed. As it considers past (predicted) position, it acts as a temporal context for the current prediction.

The formula for the "Bayesian with memory" baseline is:

$$P(x|\mathbf{K}) = C \times {\color{blue}p(x)} \times {\color{red}normDist(\hat{x}_{t-1},\ \sigma)} \times \prod_{i=1}^{N} Poisson(k_i, T\alpha_i(x)) \tag{5.3}$$

where C is a normalizing constant and the two added terms are colored. The continuity constraint is a 2D Gaussian distribution defined as:

$$P(x|\hat{x}_{t-1}) = normDist(\hat{x}_{t-1}, \sigma) = L \times exp(\frac{-||\hat{x}_{t-1} - x||^2}{2\sigma^2}) \tag{5.4}$$

where L is the normalizing constant and $\sigma = M \times velocity$. M is a scaling constant and velocity is computed as the average over last 5 timesteps.

Notice that while a RNN learns by itself how exactly to use the past activity, in here we need to decide beforehand how the previous activity is translated into an additional constraint. Also, in here we explicitly provide the model with information about most often visited locations in the environment ($p(x)$), whereas RNN might or might not learn this distribution from training data.

5.2. Data and Methods

5.2.1. Single-cell recordings

The extensive pre-processing to extract the activities of individual neurons from the raw electric signal recorded in electrodes is described in the Background section on place cell recordings . The machine learning part of this project assumes we have for each animal a data matrix where each row corresponds to one neuron and the columns to non-overlapping 20 ms timewindows. Each value in this matrix corresponds to the spike count of a neuron in a given timewindow (the count of times the neuron was activated). The number of neurons (rows) and timewindows (columns) depends on the recording session (animal; environment), as the recording quality and duration varies. The locations of the animals are also provided with 20 ms precision.

We have recordings for the same 5 rats in two different environments, a 2D arena of size 1m x 1m and a Z-shaped linear track of total length 6 m. These original data matrices are further processed to get the exact inputs we want the models to receive.

5.2.2. Input data to the RNNs and baseline models

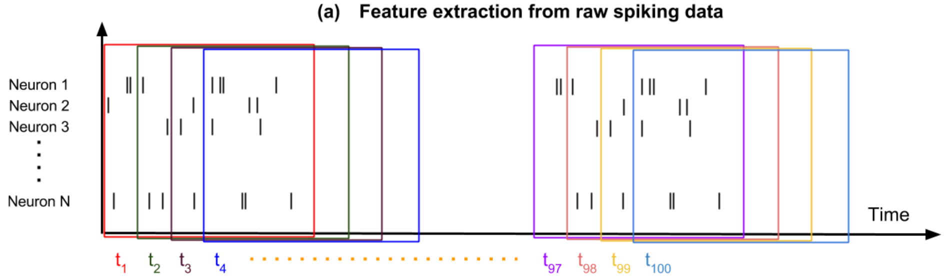
Starting from the simplest baseline model, in here we describe the type of inputs each of the applied models receives.

The "simple Bayesian" model (MLE) receives as input the spike counts in a single timewindow. The length of these timewindows is varied from 200 ms to 4000 ms in steps of 200 ms to find the most efficient timewindow size. The consecutive data points have a 50% overlap, meaning for example that in case of 1000 ms windows the first and second data point correspond to 0-1000 ms and 500-1500 ms intervals of the recording. A fixed temporal step of 200 ms between consecutive data points was also tested (leading to the second data point being from interval 200 ms to 1200 ms), but the results were weaker. The desired output is the location at the center of the time-window.

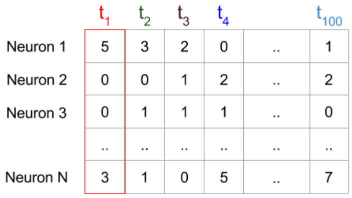
"Bayesian with Memory" gets as input spike counts from two timewindows. The first of these input vectors is used for computing the continuity constraint. The second is used for the likelihood estimation as in the simple Bayesian model. The model is also informed by the distribution of visited locations over the entire trial and the current movement speed. All this information is combined to get the posterior distribution, which is the output of the model. Time-window sizes, speed calculation and other hyperparameters are tuned. The desired output is the location at the center of the second time-window.

RNN gets as input a series of spike counts from 100 consecutive time windows (Figure 13). The time-windows are overlapping and consecutive widows are shifted by 200 ms in time. This means that, for example, when using 400 ms time windows the overlap is 50% (200 ms), whereas with 1000 ms the overlap is

(a) Feature extraction from raw spiking data



(b) Sequence of spike count vectors



	t_1	t_2	t_3	t_4		t_{100}
Neuron 1	5	3	2	0	..	1
Neuron 2	0	0	1	2	..	2
Neuron 3	0	1	1	1	..	0
	..	..	..	..	..	..
Neuron N	3	1	0	5	..	7

Figure 13. Input for RNN models. (a) First we count the spikes for each neuron in overlapping periods of time, consecutive windows are shifted 200 ms. Each window yields a spike count vector of length equal to number of recorded neurons. (b) we take 100 consecutive spike count vectors as the sequence of inputs that the RNN model receives.

80% (800 ms). In total, across the 100 time windows roughly 20 seconds of neuronal activity is used as input. After observing the inputs, the model is expected to return coordinate values (1 value in Z-track, 2 in 2D arena). Hence, a data point for RNN consists in 100 spike-count vectors and the true coordinate values at the center of last time window. Consecutive data points overlap by 99 time windows.

For all models 10-fold cross validation was used. The input data was cut into 10 pieces along the time-axis, meaning each fold contained data from consecutive time points. Notice that random assigning of data points to folds would have led to temporally overlapping (highly similar) data points ending up in train and test folds. This would have led to an overestimation of our models' predictive ability. Additionally, also data points in the beginning of a fold that have overlapping timewindows (in case of RNN) with previous fold are removed.

5.2.3. RNN model

We experimented with LSTM (long short-term memory [48]) and GRU (gated recurrent unit [18]) networks of different depth and width, various number of fully connected layers, dropout rate, optimizer, learning rate and learning rate schedule, activation functions, etc. Despite achieving the lowest prediction error with a bi-directional LSTM ([85]), for the simplicity of the methods, we eventually decided to use a simpler LSTM model. This model contained just two LSTM layers and the fully-connected output layer with two nodes (for X and Y coordinate predictions). The model architecture is given in Appendix A.

5.3. Results

In the following we describe the results obtained with our 2-layer LSTM decoder and with baseline models.

5.3.1. High position decoding accuracy with RNNs

The RNN decoder achieved equal or better results than *simple Bayesian (MLE)* and *Bayesian with memory* models for all 10 datasets (5 animals, 2 environments each). This difference in performance was especially pronounced when comparing mean euclidean distance between predicted and true locations, while in median errors Bayesian methods were more competitive. The difference between mean and median performance originates from Bayesian models making more very large mistakes. The large mistakes push the mean up, leaving median untouched. The results are summarized in the Figure 14. For each dataset and each model different time window sizes were tested and the optimal time window size was used for the figure.

When investigating the errors made by the RNN and Bayesian models at different time window sizes (for rat R2192 and 2D environment), we saw that RNN outperforms the baselines for all window sizes, with the difference being more pronounced with small window sizes. Also, the optimal window sizes for all datasets are smaller with RNN compared to baselines. We hypothesize that efficiently using context allows RNNs to combat the noise in spike counts. This allows RNN to successfully use smaller windows.

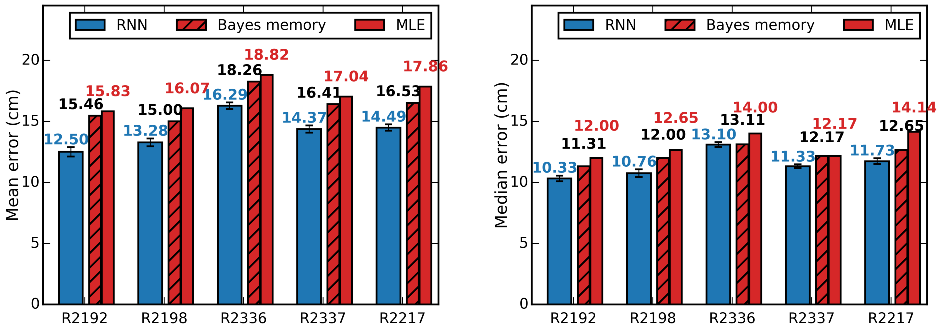
Analyzing the performance differences across the datasets, we noticed that the benefits of RNNs over Bayesian models are more pronounced when data is scarce - when there are fewer neurons recorded. To verify this claim we randomly downsampled the neurons in the 2D recording of rat R2192 and retrained the models. The results depicted in Figure 15 show that the difference between models is indeed accentuated by having fewer neurons available.

5.3.2. Detailed analysis of the RNN model

Having deomnstrated our 2-layer LSTM model's benefits over baseline models we next perform error and sensitivity analysis. We consider the results in 2D environments as the more interesting, because almost all prior work on decoding animal location has been done in 1D environments (T-shaped track, linear track). Hence, decoding position in 2D is somewhat novel even though the firing patterns of neurons with respect to 2D location - called place fields - have been investigated already for decades [70, 71]. As animals normally navigate in 2D surfaces (sometimes even 3D spaces) investigating the neuron types, neuron characteristics, model sensitivity, etc., on this task carries more behavioural meaning. We

(a) Mean errors in 2D decoding task

(b) Median errors in 2D decoding task



(c) Mean errors in 1D decoding task

(d) Median errors in 1D decoding task

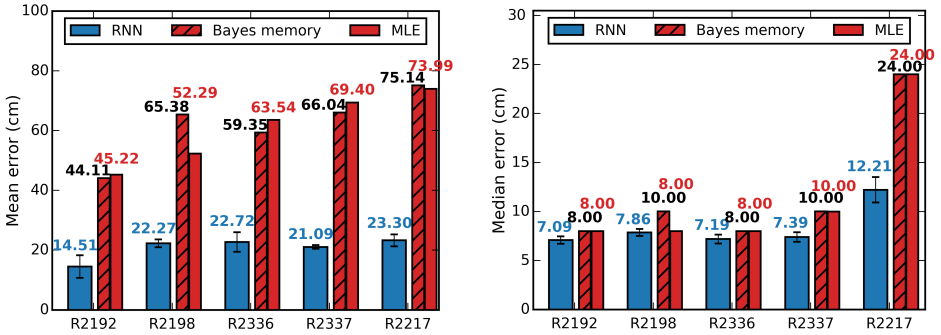
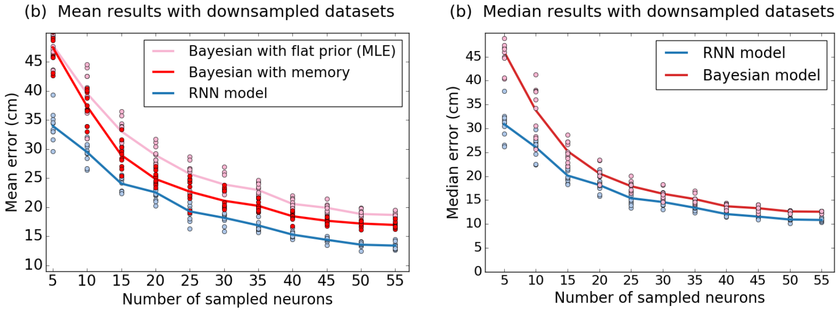


Figure 14. Spatial decoding across animals in 2D and 1D environments. (a-b) Decoding results in a 1m square environment. The RNN consistently outperforms the two Bayesian approaches in all 5 data sets. Mean and median errors across cross-validation folds, respectively. (c-d) Decoding errors from a 600 cm long Z-shaped track. RNN consistently yields lower decoding errors than the Bayesian approaches, the difference is more marked when mean (c) as opposed to median (d) errors are considered.


(b) Mean results with downsampled datasets

(b) Median results with downsampled datasets

Figure 15. Downsampling analysis demonstrates the RNN decoder is more robust to small dataset sizes. Data from R2192 was downsampled and all three decoders were trained with a random subset of the available neurons. For each sample size, 10 random sets of neurons were selected and independent models trained as before using 10-fold cross validation. Dots represents (b) mean and (c) median error for each downsampled dataset. Lines indicate the (b) mean of means and (c) mean of medians over sets of the same size.

use the 2D dataset with most recorded neurons, originating from rat R2192, for error and sensitivity analysis.

When analyzing where the model makes the largest errors we discovered the following correlations:

- the more training data for a certain region of space we have, the more precise the model is in that region. This is an expected effect in all machine learning approaches.
- the more neuronal activity (sum of spike counts) there is, the more precise the model is. In short, this correlation shows that with more recorded neurons (yielding more spikes) we'd get better prediction accuracy. To explain, the place cells are activated only if the animal is in a certain region of space - i.e. within the *place field* of that cell. With few neurons recorded, the place fields do not cover the space uniformly. Recording more neurons would increase the chance that all zones are sufficiently covered.
- when the animal is near a wall, the error in the direction perpendicular to the wall decreases (Figure 16). The error in the direction parallel to the wall does not improve by being near that wall. This is a previously known effect [42] that we wanted to further demonstrate - the animal's brain uses spatial cues to know where it is. Walls are one of such cues.

- the decoding errors are larger if the animal stands still, as opposed to moving around. This is supposedly due to the animal not being actively engaged in movement and hence not actively using the brain area we record from. Also, in resting state the brain is known to replay previous experiences - i.e. CA1 activity might reflect previous paths, not current location [23,72,101]. In addition we found that, interestingly, the decoding accuracy does not

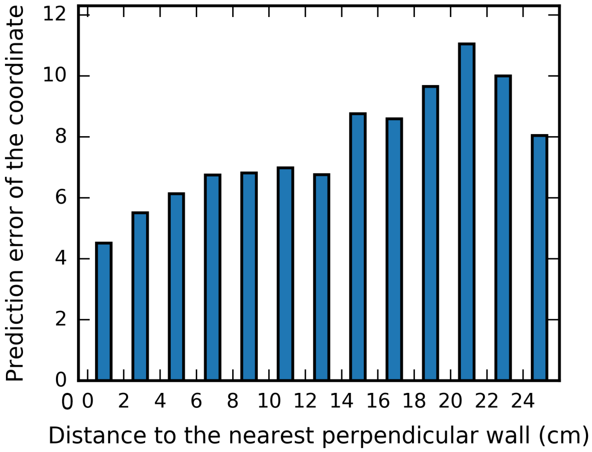


Figure 16. Decoding error decreases near the walls, but only in the direction perpendicular to the wall.

worsen at high movement speeds.

We also performed different types of sensitivity analysis to figure out what types of inputs and which aspects of the inputs are most informative for our model in this decoding task.

- According to knockout analysis, where all inputs from a neuron are set to zero, the firing rate of a neuron correlates strongly (r=0.50) with its knock-out importance. Indeed, the prediction error increased the most when knocking out the neuron with by far the highest firing rate. Interestingly, this neuron was an inhibitory neuron. Prior work has suggested that inhibitory neurons are not important for location-decoding. Other most influential neurons were excitatory cells with clear place fields and high firing rate, as expected.
- Gradient sensitivity analysis captures a different type of sensitivity to noise than knock-out analysis. Whereas in knock-out we remove an input completely, in gradient analysis we perturb it by an infinitely small amount. The neurons listed as most influential according to the two measures are similar (correlation r=0.57), but with clear differences. For example, according to gradient sensitivity inhibitory neurons do not show up among the most important cells. The knock-out and gradient sensitivity measures are complementary to each other and to the commonly used Skaggs information-theoretical measure [91].
- Gradient sensitivity analysis reveals that small perturbations in the firing rate of a given neuron influence the model less if the animal is at the center of that neuron's place field. This concurs with theoretical predictions stating that, in general, neural responses are most informative in the regions of their coding space where the firing rate changes most rapidly for a given change in the encoded variable [75] (in our case position). Also, the sensitivity is

lower outside the place field, as expected.

5.4. Discussion

5.4.1. Discussion of results

In this work we have shown that RNNs are a flexible and efficient machine learning tool to analyze neural recordings. Similarly to biological neuronal networks, RNNs are capable of efficiently using prior activity as context when processing the latest inputs. For all of the 10 datasets the location of the rat was decoded from its hippocampal activity with good accuracy.

The results were an improvement over the two Bayesian approaches used as baseline models. Bayesian decoders are known to be optimal decoders when using appropriate priors [64]. In here (and generally in neuroscience), it is difficult to determine what these appropriate priors are. In contrast, the suggested RNN approach allows to use contextual information without pre-defining any complicated priors.

As an additional advantage, the RNN model was more robust to noise than the baseline models. In particular, it achieved good prediction accuracy even with short time windows where the noise in spike counts is more pronounced. Furthermore, the RNN model also demonstrated higher robustness to scarce data than the baseline models. In particular, with fewer neurons recorded, the RNN model's performance deteriorated slower than the Bayesian models'. Consider that things happen fast in the brain and using 200ms and smaller time-windows is in many cases not a choice but a necessity. Decision-making, response to stimuli and many other tasks are performed in such small timescales. Similarly, it is often not possible to record more than a few informative neurons at the same time, especially if the brain region responsible for the task is small and lies deep in the brain.

Using neural networks as the machine learning model also opens new avenues for sensitivity analysis. Gradient sensitivity analysis was shown to be complementary to knock-out analysis and Skaggs information measure [91]. These "importance" or "informativeness" measures capture very different types of sensitivity. We believe that gradient-type of sensitivity, i.e. sensitivity to small perturbations, noise, is very interesting and relevant in neuroscience.

Prior work has claimed that inhibitory neurons do not participate in location encoding. Our knock-out analysis showing an inhibitory neuron with high firing rate as the most important cell for decoding accuracy might be a bias introduced by our model. However, the prior studies might have used less powerful decoders and there is a possibility this neuron is actually informative. Despite not having a clear place field, there are ways for such cell to participate in location encoding. When not considering all neurons independent, the firing rate of this neuron can be combined with others and might carry information in a way not visible on firing-rate-in-space plots. Also, if this cell somehow carries contextual information, e.g.

information about past location, its importance for the decoding task might not be visible on place-field plots. In all, our result suggests that inhibitory neurons might participate in location decoding in some yet unspecified way.

Error analysis allowed to confirm some hypothesis about the internal processes inside a rat's brain during behaviour. It has been proposed that when the rat is not navigating and is attending to some other behaviour, the CA1 region of hippocampus replays previous experiences (for memory consolidation). The fact that our model's decoding accuracy goes down when the animal stands still helps to support such argument - amount of information about the current location decreases in those moments. Similarly, it has been shown that visual cues matter for accurate navigation. We support this claim by showing there is more information (or more precise information) about location in CA1 neurons when the animal is near a wall. Beyond further proving these two known effects (memory replay and effect of visual cues), the fact we discover these correlations in our model's error strengths adds value to our approach. Similar models could be used to discover completely unknown aspects or find evidence for less-proved claims about when and how information is present in some region of the brain.

In all, we believe that the field of neuroscience will benefit from the improved performance and additional sensitivity measures that recurrent neural networks allow.

5.4.2. Future work

During the investigation of the data recorded in rat R2192 in 2D task, we obtained many additional results that did not fit into the publication. We take our chance to mention them here. Many of these results open new questions and directions of research:

- We trained a model receiving not 100 timesteps of past neural activity, but instead spike counts from 100 time windows to the future. These future measures were fed into the RNN the last one first, so that the 100th input was the time window centered around location measurement. Such model trained on future activity had higher decoding errors than a model trained based on past activity. From this we can conclude that CA1 cells contain more information about future locations (e.g. they might participate in path planning) than about past location (e.g. when remembering where one has been). If there was no future or past information and the cells reflected just the momentary location, we'd expect the two models to yield very similar results.

  Designing a reliable way to identify neurons that contain information about future locations and that might participate in path planning is an interesting future direction.
- Including neurons recorded from entorhinal cortex (region containing grid cells that also participate in location encoding) significantly improves the

decoding accuracy. For rat R2192 there were 37 additional cells recorded from that area (there were 63 recorded from CA1). Adding them to the analysis improved mean accuracy by approximately 2 cm. However, we decided to concentrate only on neurons within CA1 in this work.

  Reporting the results when combining the recordings from the two areas might be of interest to set a new lower bound on how much the animal itself knows about its location. Predicting location based on only entorhinal cells is also a potential future direction.

- We can also improve decoding accuracy by using as "neurons" the results of spike clustering before manual verification of the clusters. During manual verification the clusters that do not look like one neuron are removed from further analysis. This removes spikes (i.e. information) from the analysis, but allows all inputs to be interpreted as individual neurons. In this work we decided to use only neurons and not clusters of unclear composition, despite 2 cm lower mean error if using the raw clusters.

  The spikes in these "other clusters" are an actual signal existing in the brain. There is actually no reason other than interpretability why not to include them. A future direction is to compare if the baselines can also benefit from such non-conventional inputs or they fail due to the built-in assumptions about the data. If they cannot benefit from such noisy data, it is another argument in favour of using RNNs.

- In contrast to the "raw clusters", including the "garbage cluster" (i.e. a cluster containing all spikes that did not match any cluster/neuron) as an additional input row (another "neuron") did not significantly influence performance. Changes in mean error were less than standard deviation in all cases, mostly lowering validation accuracy rather than improving it.

  While this observation does not open any avenues for future research, it helps to confirm that just adding noise is not beneficial and that the "raw clusters" are more meaningful that the "garbage cluster".

- We experimented with bidirectional LSTM model that can simultaneously consider past and future spike activity. This improved prediction accuracy by around 1cm. We excluded this model type because 1) it is a complicated architecture to understand for someone not familiar with neural networks (our aim was to keep it understandable) and 2) we decided to use only past neural activity, as this is what the brain can rely on in real life. Another architecture was also slightly superior to the final approach, but was too complicated to explain to an uninitiated reader. Complicated methods would give the false impression that RNNs work better than baselines only if one uses a highly advanced approach. Our published approach does not use future data and can be presented as a natural extension of approaches considering just one time-window or two time windows.

  Nevertheless, in the future applying the most effective model type with the most informative data type might be of interest in this decoding task or others.

We conclude that there are open research topics in the study on position encoding in the brain. Our colleagues in University College London are indeed continuing this line of research.

6. CONCLUSION

What part of the available information to present to the algorithm? In all machine learning tasks there is theoretically an infinite amount of information that we could include. For any measurement/image/data point we could include additional information such as the time of the measurements, the temperature, identify of the experimenter and so on. In specific cases these factors could indeed influence the results - the brain activity and behaviour of the rats (in Publication 3) might indeed change due to any of the aforementioned three factors. However, in most cases we exclude information that we deem (1) not likely to be useful, or (2) too difficult to work with. The former contains an unverified assumption, the latter strongly depends on the methods used. In this thesis we argue for using artificial neural networks and more generally, representation learning, because in the biological data applications here studied they allow to use (1) a larger amount of available information (2) without adding much extra difficulty.

In particular, Publications 1 and 2 taken together reveal that applying convolutional neural networks (CNNs) on raw DNA sequences is more effective than constructing models based on pre-defined features extracted from the same sequences. For the models in Publication 1 and the baseline models of Publication 2, the input DNA sequence was reduced to a set of extracted features (k-mers, RSCU values). In Publication 2 we show that a CNN with raw sequences as inputs outperforms these models by a large margin.

Despite rarely explicitly stating it, by extracting features one acknowledges that the extraction loses some information, but assumes that the relevant information is mostly maintained (assumption (1) from above). The researcher knows that some alternative more-information-preserving set of features might exist, but they might not suit the intended machine learning models ((2) too difficult to work with). CNNs have the capacity to work with raw DNA sequences (more information) and learn by themselves what information to extract (no added difficulty). This eliminates the need to pre-define what might be the maximally informative representation of the data. In the task of predicting whether a DNA contig originates from a virus or not, this benefit translates into a significant improvement of predictive power (AUROC 0.93 vs AUROC 0.79).

Similarly, in Publication 3 we demonstrate that by using a recurrent neural network (RNN) based approach, considering more information - in this case the past neural activity - is beneficial for the position decoding performance. Moreover, with RNNs, using more information is in fact rather natural (no added difficulty). In contrast, with the first baseline method, maximum likelihood estimation, past activity is not considered at all. The second baseline model, Bayesian with memory, is capable of using a small part of past information in a highly predefined manner. Using more of the past information with Bayesian approach would become increasingly complicated. Hence, again we claim that our neural network approach allows to use more information more easily.

Also we want to point out that when using maximum likelihood and Bayesian approaches, we need to make explicit assumptions about the brain. Most strikingly, we assume that given a certain stimuli, each neuron's firing rate is an independent Poisson process (not necessarily true in case of place cells [32]). While it is a common assumption, by doing this we essentially pre-define how the model reacts to noise. The proposed RNN-based approach might also make implicit assumptions due to its internal structure, activation functions used, the trade-offs that the chosen optimizer takes to minimize the chosen loss function and so on. However, it is much more flexible and can learn to deal with noise in more variable ways. For example, contrarily to baseline approaches, an RNN might learn to not consider certain neuron at all if it is too noisy; consider all activation values above 1 as equivalent; consider the mutual effects of signals; etc. Hence the RNN model uses more information, with less effort and fewer built-in assumptions.

In the future, the predicting self-location could be done on even more "raw" (unprocessed) inputs. Unpublished results we obtained during the analysis of the hippocampal recordings showed that classification on raw clustering results - before the step of an expert merging similar clusters and removing clusters that do not look right or have too few spikes - yield even better prediction accuracy. For the ease of interpreting sensitivity analysis results, we used the expert-revised clusters in our publication. In contrast, if one wishes to set a new lower bound on the amount of information contained about self-location in CA1 neurons, we suggest to use un-corrected clusters. Or, predict the location directly from the raw electrical signals recorded by the electrodes, skipping also the spike-sorting step. RNNs are well-suited for such tasks.

More globally for the field of neural decoding, it is useful to know that for recurrent neural networks the noisiness of the inputs is often not a problem - more of raw inputs is better than less of cleaner inputs. In many interesting problems we just want to know if some variable can be decoded from brain activity, no matter how. For example, if the person is lying. Or if the person has a neural disease. It is also crucial to stress that we only had roughly 5000-6000 datapoints in each dataset (per animal, per environment). We trained a model with more than 3 million parameters on each of these sets. The false belief that deep learning methods require millions of annotated training points to succeed is hindering the spreading of these powerful methods to this fascinating field.

Similarly, in publications 2 and 3, we worked with a dataset with roughly 5000 positive (viral) samples (there were 200 000+ negative samples, but downsampling it significantly yielded similar results). It is likely that similar amounts of data can be collected for other problems in (meta)genomics. Most immediately, as for viruses, there are unidentified bacterial species living in our bodies and in our environment. Work for identifying them is ongoing.

To conclude, there are thousands of scientists all over the world working on similar problems of extracting information from biological data similar to the datasets used in this work. We are confident that in a non-trivial part of these tasks, using appropriate neural network models would increase the performance significantly over the more traditional approaches that use pre-defined features instead of data-driven, learned features.

BIBLIOGRAPHY

[1] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2009.

[2] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[3] Christof Angermueller, Tanel Pärnamaa, Leopold Parts, and Oliver Stegle. Deep learning for computational biology. *Molecular systems biology*, 12(7):878, 2016.

[4] Alfonso Araque and Marta Navarrete. Glial cells in neuronal network function. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 365(1551):2375–2381, 2010.

[5] John Athey, Aikaterini Alexaki, Ekaterina Osipova, Alexandre Rostovtsev, Luis V Santana-Quintero, Upendra Katneni, Vahan Simonyan, and Chava Kimchi-Sarfaty. A new and updated resource for codon usage tables. *BMC bioinformatics*, 18(1):391, 2017.

[6] Frederico AC Azevedo, Ludmila RB Carvalho, Lea T Grinberg, José Marcelo Farfel, Renata EL Ferretti, Renata EP Leite, Wilson Jacob Filho, Roberto Lent, and Suzana Herculano-Houzel. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *Journal of Comparative Neurology*, 513(5):532–541, 2009.

[7] Antonio Battro. *Half a brain is enough: The story of Nico*. Odile Jacob, 2003.

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

[9] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[10] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[11] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[12] Véronique Bouvard, Robert Baan, Kurt Straif, Yann Grosse, Béatrice Secretan, Fatiha El Ghissassi, Lamia Benbrahim-Tallaa, Neela Guha, Crystal Freeman, Laurent Galichet, et al. A review of human carcinogens–part b: biological agents. *The Lancet. Oncology*, 10(4):321, 2009.

[13] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[14] Davit Bzhalava, Hanna Johansson, Johanna Ekström, Helena Faust, Birgitta Möller, Carina Eklund, Peter Nordin, Bo Stenquist, John Paoli, Bengt

Persson, et al. Unbiased approach for virus detection in skin lesions. *PLoS One*, 8(6):e65953, 2013.

[15] Davit Bzhalava, Laila Sara Arroyo Mühr, Camilla Lagheden, Johanna Ekström, Ola Forslund, Joakim Dillner, and Emilie Hultin. Deep sequencing extends the diversity of human papillomaviruses in human skin. *Scientific reports*, 4, 2014.

[16] Zurab Bzhalava, Ardi Tampuu, Piotr Bała, Raul Vicente, and Joakim Dillner. Machine learning for detection of viral sequences in human metagenomic datasets. *BMC bioinformatics*, 19(1):336, 2018.

[17] Augustin Cauchy. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

[18] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

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

[20] Kenneth Church. A pendulum swung too far. *Linguistic Issues in Language Technology*, 6(5):1–27, 2011.

[21] Radoslaw Martin Cichy, Aditya Khosla, Dimitrios Pantazis, Antonio Torralba, and Aude Oliva. Comparison of deep neural networks to spatio-temporal cortical dynamics of human visual object recognition reveals hierarchical correspondence. *Scientific reports*, 6:27755, 2016.

[22] Cyril W Cleverdon. On the inverse relationship of recall and precision. *Journal of documentation*, 28(3):195–201, 1972.

[23] Thomas J Davidson, Fabian Kloosterman, and Matthew A Wilson. Hippocampal replay of extended experience. *Neuron*, 63(4):497–507, 2009.

[24] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[25] Chad J Donahue, Matthew F Glasser, Todd M Preuss, James K Rilling, and David C Van Essen. Quantitative assessment of prefrontal cortex in humans relative to nonhuman primates. *Proceedings of the National Academy of Sciences*, 115(22):E5183–E5192, 2018.

[26] Kenji Doya. Bifurcations of recurrent neural networks in gradient descent learning. *IEEE Transactions on neural networks*, 1(75):164, 1993.

[27] David A Drachman. Do we have brain to spare?, 2005.

[28] Arne D Ekstrom, Michael J Kahana, Jeremy B Caplan, Tony A Fields, Eve A Isham, Ehren L Newman, and Itzhak Fried. Cellular networks underlying human spatial navigation. *Nature*, 425(6954):184, 2003.

[29] Johanna Ekström, Davit Bzhalava, Daniel Svenback, Ola Forslund, and Joakim Dillner. High throughput sequencing reveals diversity of human papillomaviruses in cutaneous lesions. *International journal of cancer*, 129(11):2643–2650, 2011.

[30] Shaza M Abd Elrahman and Ajith Abraham. A review of class imbalance problem. *Journal of Network and Innovative Computing*, 1(2013):332–340, 2013.

[31] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[32] André A Fenton and Robert U Muller. Place cell discharge is extremely variable during individual passes of the rat through the firing field. *Proceedings of the National Academy of Sciences*, 95(6):3182–3187, 1998.

[33] Peter Flach and Meelis Kull. Precision-recall-gain curves: Pr analysis done right. In *Advances in neural information processing systems*, pages 838–846, 2015.

[34] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[35] Felix Gers. *Long short-term memory in recurrent neural networks*. PhD thesis, Verlag nicht ermittelbar, 2001.

[36] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[37] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):855–868, 2008.

[38] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[39] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[40] Umut Güçlü and Marcel AJ van Gerven. Deep neural networks reveal a gradient in the complexity of neural representations across the ventral stream. *Journal of Neuroscience*, 35(27):10005–10014, 2015.

[41] Jo Handelsman, Michelle R Rondon, Sean F Brady, Jon Clardy, and Robert M Goodman. Molecular biological access to the chemistry of unknown soil microbes: a new frontier for natural products. *Chemistry & biology*, 5(10):R245–R249, 1998.

[42] Kiah Hardcastle, Surya Ganguli, and Lisa M Giocomo. Environmental boundaries as an error correction mechanism for grid cells. *Neuron*, 86(3):827–839, 2015.

[43] David Harrison Jr and Daniel L Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of environmental economics and management*, 5(1):81–102, 1978.

[44] Douglas M Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

[45] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[46] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.

[47] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

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

[49] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[50] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

[51] Philip Hugenholtz, Brett M Goebel, and Norman R Pace. Impact of culture-independent studies on the emerging phylogenetic view of bacterial diversity. *Journal of bacteriology*, 180(18):4765–4774, 1998.

[52] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.

[53] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[54] Kristjan Korjus, Martin N Hebart, and Raul Vicente. An efficient data partitioning to improve classification performance while keeping parameters interpretable. *PloS one*, 11(8):e0161788, 2016.

[55] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[56] Ilya Kuzovkin, Raul Vicente, Mathilde Petton, Jean-Philippe Lachaux, Monica Baciu, Philippe Kahane, Sylvain Rheims, Juan R Vidal, and Jaan Aru. Activations of deep convolutional neural networks are aligned with gamma band activity of human visual cortex. *Communications biology*, 1(1):107, 2018.

[57] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[58] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[59] Michael S Lewicki. A review of methods for spike sorting: the detection and classification of neural action potentials. *Network: Computation in Neural Systems*, 9(4):R53–R78, 1998.

[60] Abninder Litt, Chris Eliasmith, Frederick W Kroon, Steven Weinstein, and Paul Thagard. Is the brain a quantum computer? *Cognitive Science*, 30(3):593–603, 2006.

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

[62] A Mercalli, V Lampasona, K Klingel, L Albarello, C Lombardoni, Johanna Ekström, V Sordi, A Bolla, A Mariani, Davit Bzhalava, et al. No evidence of enteroviruses in the intestine of patients with type 1 diabetes. *Diabetologia*, 55(9):2479–2488, 2012.

[63] Marvin Minsky and Seymour Papert. Perceptron: an introduction to computational geometry. *The MIT Press, Cambridge, expanded edition*, 19(88):2, 1969.

[64] Tom M Mitchell et al. Machine learning. wcb, 1997.

[65] John L Mokili, Forest Rohwer, and Bas E Dutilh. Metagenomics and future perspectives in virus discovery. *Current opinion in virology*, 2(1):63–77, 2012.

[66] Hans Moravec. When will computer hardware match the human brain. *Journal of evolution and technology*, 1(1):10, 1998.

[67] Elon Musk et al. An integrated brain-machine interface platform with thousands of channels. *BioRxiv*, page 703801, 2019.

[68] Eric A Newman. New roles for astrocytes: regulation of synaptic transmission. *Trends in neurosciences*, 26(10):536–542, 2003.

[69] OECD. *HEALTH AT A GLANCE: Europe 2018*. ORGANIZATION FOR ECONOMIC, 2018.

[70] John O'Keefe and Jonathan Dostrovsky. The hippocampus as a spatial map. preliminary evidence from unit activity in the freely-moving rat. *Brain research*, 34(1):171–175, 1971.

[71] John O'keefe and Lynn Nadel. *The hippocampus as a cognitive map*. Oxford: Clarendon Press, 1978.

[72] H Freyja Ólafsdóttir, Francis Carpenter, and Caswell Barry. Task demands predict a dynamic switch in the content of awake hippocampal replay. *Neuron*, 96(4):925–935, 2017.

[73] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[74] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, and Vincent Dubourg. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[75] Alexandre Pouget, Sophie Deneve, Jean-Christophe Ducom, and Peter E Latham. Narrow versus wide tuning curves: What's best for a population code? *Neural computation*, 11(1):85–90, 1999.

[76] Pasko Rakic. Confusing cortical columns. *Proceedings of the National Academy of Sciences*, 105(34):12099–12100, 2008.

[77] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

[78] Jie Ren, Nathan A Ahlgren, Yang Young Lu, Jed A Fuhrman, and Fengzhu Sun. Virfinder: a novel k-mer based tool for identifying viral sequences from assembled metagenomic data. *Microbiome*, 5(1):69, 2017.

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

[80] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[81] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[82] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[83] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth annual conference of the international speech communication association*, 2014.

[84] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798. ACM, 2007.

[85] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[86] Katerina Semendeferi, Este Armstrong, Axel Schleicher, Karl Zilles, and Gary W Van Hoesen. Prefrontal cortex in humans and apes: a comparative study of area 10. *American Journal of Physical Anthropology: The Official Publication of the American Association of Physical Anthropologists*, 114(3):224–241, 2001.

[87] Ron Sender, Shai Fuchs, and Ron Milo. Revised estimates for the number of human and bacteria cells in the body. *PLoS biology*, 14(8):e1002533, 2016.

[88] Paul M Sharp, Therese MF Tuohy, and Krzysztof R Mosurski. Codon usage in yeast: cluster analysis clearly differentiates highly and lowly expressed genes. *Nucleic acids research*, 14(13):5125–5143, 1986.

[89] Young C Shin, Georg F Bischof, William A Lauer, and Ronald C Desrosiers. Importance of codon usage for the temporal regulation of viral gene expression. *Proceedings of the National Academy of Sciences*, 112(45):14030–14035, 2015.

[90] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[91] William E Skaggs, Bruce L McNaughton, and Katalin M Gothard. An information-theoretic approach to deciphering the hippocampal code. In *Advances in neural information processing systems*, pages 1030–1037, 1993.

[92] Peter Skewes-Cox, Thomas J Sharpton, Katherine S Pollard, and Joseph L DeRisi. Profile hidden markov models for the detection of viruses within metagenomic sequence data. *PloS one*, 9(8):e105067, 2014.

[93] Jeroen B Smaers, Aida Gómez-Robles, Ashley N Parks, and Chet C Sherwood. Exceptional evolutionary expansion of prefrontal cortex in great apes and humans. *Current Biology*, 27(5):714–720, 2017.

[94] R Staden. A mew computer method for the storage and manipulation of dna gel reading data. *Nucleic Acids Research*, 8(16):3673–3694, 1980.

[95] P Sundström, P Juto, G Wadell, Göran Hallmans, A Svenningsson, Lennarth Nyström, Joakim Dillner, and L Forsgren. An altered immune response to epstein-barr virus in multiple sclerosis: a prospective study. *Neurology*, 62(12):2277–2282, 2004.

[96] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[97] Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Hervé Jégou. Fixing the train-test resolution discrepancy. *arXiv preprint arXiv:1906.06423*, 2019.

[98] Benjamin W Towse, Caswell Barry, Daniel Bush, and Neil Burgess. Optimal configurations of spatial scale for grid cell firing under noise and uncertainty. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 369(1635):20130290, 2014.

[99] Dana Willner, Mike Furlan, Matthew Haynes, Robert Schmieder, Florent E Angly, Joas Silva, Sassan Tammadoni, Bahador Nosrat, Douglas Conrad, and Forest Rohwer. Metagenomic analysis of respiratory tract dna viral communities in cystic fibrosis and non-cystic fibrosis individuals. *PloS one*, 4(10):e7370, 2009.

[100] Matthew A Wilson and Bruce L McNaughton. Dynamics of the hippocampal ensemble code for space. *Science*, 261(5124):1055–1058, 1993.

[101] Matthew A Wilson and Bruce L McNaughton. Reactivation of hippocampal ensemble memories during sleep. *Science*, 265(5172):676–679, 1994.

[102] John C Wooley, Adam Godzik, and Iddo Friedberg. A primer on metagenomics. *PLoS computational biology*, 6(2):e1000667, 2010.

[103] Kristine M Wylie, George M Weinstock, and Gregory A Storch. Emerging view of the human virome. *Translational Research*, 160(4):283–290, 2012.

[104] Daniel LK Yamins, Ha Hong, Charles F Cadieu, Ethan A Solomon, Darren Seibert, and James J DiCarlo. Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the National Academy of Sciences*, 111(23):8619–8624, 2014.

[105] Feng Yu, Qing-jun Jiang, Xi-yan Sun, and Rong-wei Zhang. A new case of complete primary cerebellar agenesis: clinical and imaging findings in a living patient. *Brain*, 138(6):e353–e353, 2014.

[106] Kechen Zhang, Iris Ginzburg, Bruce L McNaughton, and Terrence J Sejnowski. Interpreting neuronal population activity by reconstruction: unified framework with application to hippocampal place cells. *Journal of neurophysiology*, 79(2):1017–1044, 1998.

[107] Zhi-Hua Zhou and Xu-Ying Liu. Training cost-sensitive neural networks with methods addressing the class imbalance problem. *IEEE Transactions on knowledge and data engineering*, 18(1):63–77, 2005.

Appendix A. NEURAL NETWORK ARCHITECTURES IN DETAIL

A.1. Publication I: feedforward neural network architecture

The network receives as input 59-element vector (the RSCU values). The last layer is the sigmoid activation layer, that returns a probability. Other layers have ReLU non-linearity. Binary cross-entropy loss between the output and true label is optimized.

```
______________________________________________________________________
Layer (type)             Output Shape     Param #     Connected to
======================================================================
dense_1 (Dense)          (None, 1024)     61440      dense_input_1[0][0]
______________________________________________________________________
dropout_1 (Dropout)      (None, 1024)     0          dense_1[0][0]
______________________________________________________________________
dense_2 (Dense)          (None, 1024)     1049600    dropout_1[0][0]
______________________________________________________________________
dropout_2 (Dropout)      (None, 1024)     0          dense_2[0][0]
______________________________________________________________________
dense_3 (Dense)          (None, 1)        1025       dropout_2[0][0]
______________________________________________________________________
activation_1             (None, 1)        0          dense_3[0][0]
(Activation)
======================================================================
Total params: 1,112,065
Trainable params: 1,112,065
Non-trainable params: 0
```

A.2. Publication II: ViraMiner convolutional neural network architecture

Input is the one-hot encoded DNA sequence. Output is a probability (last Dense layer uses sigmoid activation, not shown as separate layer here). Binary cross-entropy loss between the output and true label is optimized. All fully-connected and convolutional layers, except the output layer, use ReLU activation function.

In the Keras summary below, it is important to notice which layers each layer is connected to (i.e. receives input from) - there are two branches and the concatenate layer receives input from both of them.

```
________________________________________________________________________
Layer (type)              Output Shape       Param #     Connected to
========================================================================
input_1 (InputLayer)      (None, 300, 5)     0
________________________________________________________________________
conv1d_1 (Conv1D)         (None, 293, 1000)  41000      input_1[0][0]
________________________________________________________________________
```

```
conv1d_2 (Conv1D)        (None, 290, 1200)  67200     input_1[0][0]
------------------------------------------------------------------------
gl_avg_pool_1d_1         (None, 1000)       0         conv1d_1[0][0]
(GlobalAveragePooling1D)
------------------------------------------------------------------------
gl_max_pool_1d_1         (None, 1200)       0         conv1d_2[0][0]
(GlobalMaxPooling1D)
------------------------------------------------------------------------
dropout_1 (Dropout)      (None, 1000)       0     gl_avg_pool_1d_1[0][0]
------------------------------------------------------------------------
dropout_2 (Dropout)      (None, 1200)       0     gl_max_pool_1d_1[0][0]
------------------------------------------------------------------------
fc_layer1 (Dense)        (None, 1000)       1001000   dropout_1[0][0]
------------------------------------------------------------------------
fc_layer2 (Dense)        (None, 1200)       1441200   dropout_2[0][0]
------------------------------------------------------------------------
concatenate_1            (None, 2200)       0         fc_layer1[0][0]
(Concatenate)                                         fc_layer2[0][0]
------------------------------------------------------------------------
drop_fc1 (Dropout)       (None, 2200)       0        concatenate_1[0][0]
------------------------------------------------------------------------
dense_1 (Dense)          (None, 1)          2201      drop_fc1[0][0]
========================================================================
Total params: 2,552,601
Trainable params: 2,552,601
Non-trainable params: 0
```

A.3. Publication III: Recurrent neural network architecture for decoding self-location

The model was formalized as Sequential type of model, using an earlier version of Keras. There is no "connected to" column as, by definition of "sequential model" in Keras, each layer is connected to the previous layer and previous layer only.

The first layer is the input sequence - the spike count vectors from 100 timewindows. The last dimension of the input (and hence number of parameters) depends on the animal, as it depends on the number of neurons recorded. The outputs are two real numbers. Mean squared error between the outputs and true location is minimized during optimization.

```
-----------------------------------------------------------------
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 100, 63)           0
-----------------------------------------------------------------
lstm_1 (LSTM)                (None, 100, 512)          1179648
-----------------------------------------------------------------
dropout_1 (Dropout)          (None, 100, 512)          0
-----------------------------------------------------------------
```

```
lstm_2 (LSTM)                (None, 512)               2099200

-----------------------------------------------------------------
dropout_2 (Dropout)          (None, 512)               0

-----------------------------------------------------------------
dense_1 (Dense)              (None, 2)                 1026
=================================================================
Total params: 3,279,874
Trainable params: 3,279,874
Non-trainable params: 0
```

We used dropout rate 0.5, batch size 64, learning rate 0.001 with RMSProp, trained for 50 epochs with no early stopping. The loss function was mean squared error.

ACKNOWLEDGEMENT

To successfully complete a PhD thesis one needs a long list of favourable conditions:

- a supporting social environment. It is invaluable to have around you people who believe without questions that what you do is meaningful and that you are indeed, no doubt, able to achieve your goals.
- a good supervisor who guides the student towards meaningful problems while leaving him the final say about what is interesting and what is not.
- colleagues willing to spend hours helping you figure out the problems you have with your project. Colleagues, who share their interests and passions and through doing so lead you to new challenges that you would not have found by yourself.
- some smarts

I had the luck to have all these favourable conditions fulfilled with a margin (the last one I leave for the reader to judge). Thanks!

SUMMARY IN ESTONIAN

Tehisnärvivõrgud bioloogiliste andmete analüüsimiseks

Tehisnärvivõrgud on üsna hiljuti (alates 2012. aasta AlexNeti artiklist [55]) populaarseks saanud masinõppe algoritm. Nagu ka teised masinõppe meetodid on tehisnärvivõrgud võimelised muuhulgas näidete põhjal õppima (juhendamisega õpe). Iga näide koosneb sisendist ja sellele sisendile vastavast oodatavast (õigest) vastusest. Tehisneuronite vahelisi ühendusi optimeeritakse nii, et mudeli õpetamiseks kasutatavete näidete (treeningandmepunktide) puhul annaks mudel võimalikult täpseid vastuseid.

Erinevad variatsioonid tehisnärvivõrkudest on oma kasulikkust tõestanud mitmetes arvutiteaduse harudes. Näiteks konvolutsioonilised närvivõrgud (CNN, convolutional neural networks) on täielikult asendanud senised meetodid masinnägemises. Konvolutsioonilised võrgud on väga efektiivsed näiteks objektituvastuses ja näotuvastuses. Rekurrentsed närvivõrgud (RNN, recurrent neural networks) on väga efektiivsed kõnetuvastuses ja keeletehnoloogias (lausete mõtte, emotsiooni jne mõistmiseks). Need näited aga ei ole ainsad võimalikud tehisnärvivõrkude rakendamise valdkonnad. Teadusmaailmas, ja ka ärimaailmas ja ühiskonnas üldiselt, on veel tohutu hulk andmestikke ja probleeme, mille puhul tehisnärvivõrkude rakendamine võiks anda väga häid tulemusi. Tulemusi, mis seniste meetoditega võimalikud pole.

Selles doktoritöös näitasime me tehisnärvivõrkude kasulikkust kahe bioloogilise probleemi lahendamisel. Esiteks rakendasime me konvolutsiooonilisi närvivõrke metagenoomikast tulenevatele DNA andmetele. Probleemipüstitus oli järgnev: kas ainult DNA järjestuses sisalduva info põhjal on võimalik ennustada, kas see järjestus pärineb viiruse genoomist ja mitte mõnda muud tüüpi organismi genoomist. Võrdluseks, enimlevinud DNA lõikude klassifitseerimiseks kasutatavad meetodid (näiteks BLAST) võrdlevad uuritavat DNA järjestust (või sellest transkribeeritud aminohapete järjestust) andmebaasiga, mis sisaldab kõiki juba teada olevaid genoome (või aminohapete järjestusi). Leides, et uuritav DNA lõik sobitub üsna hästi mõne teadaoleva genoomiga, võib järeldada, et tegu on sama või vähemalt samat tüüpi organismiga. DNA jupid, mis ühegi teadaoleva genoomiga hästi ei joondu, jäävad märgendamata - me lihtsalt ei tea, mis tüüpi organismist nad pärinevad. Käesolevas doktoritöös välja pakutud meetodid, mis ei kasuta andmebaasi ja teevad otsuse vaid DNA jupis endas sisalduva info põhjal, võimaldavad ennustada ka nende, muidu "märgendamata"jäävate DNA lõikude, päritolu. See omakorda võimaldab viroloogidel tuvastada seni täiesti tundmatuid viiruseliike, millel võib olla oluline mõju inimese tervisele.

Teine käesolevas doktoritöös käsitletud bioloogiline andmestik pärineb neuroteadusest. On teada, et imetajate hippokampuses esineb teatud tüüpi rakke, mis on "koha-tundlikud". Need nn. koharakud aktiveeruvad vaid juhul, kui loom (sh inimene) asub teatud kindlas ruumipunktis. Mujal viibides on rakk mitte-aktiivne.

Seega sisaldab nende rakkude aktiivsus infot looma asukoha kohta. Käesoleva töö kolmandas publikatsioonis näitasime me, et rekurrentsed närvivõrgud võimaldavad väga efektiivselt dekodeerida looma asukohta tema hippokampuse neuronite aktiivsuse põhjal. Mõõtes vaid mõnekümne raku aktiivsust roti hippokampuses, õnnestus rekurrentseid võrke kasutades ennustada roti asukohta 1x1m alal ligi 10cm täpsusega. Rekurrentsed võrgud osutusid täpsemaks dekodeerijaks kui neuroteaduses hetkel enim kasutatud Bayesi meetodid. Samuti näitas tulemuste analüüs, et rekurrentsed võrgud saavad müraste andmetega paremini hakkama kui Bayesi meetodid. Rekurrentsete tehisnärvivõrkude peamine eelis on võime rakkude eelnevat aktiivsust meeles pidada ja kasutada seda kontekstina, mis piirab ja täpsustab "käesoleva"hetke asukoha ennustust "käesolevaäjaperioodi aktiivuse põhjal. Samuti võimaldab väljapakutud meetod uurida, milliste neuronite aktiivsuse muutumine toob kaasa suurima muutuse ennustustes - ehk kui tundlik on mudel müra suhtes erinevates neuronites.

Ka teiste neuroteaduses uuritavate stiimulite puhul võib eelnev ajuaktiivsus peegeldada konteksti, mis omakorda võib sisaldada olulist infot hetkel toimuva kohta. Seega võivad rekurrentsed tehisnärvivõrgud osutuda ajusignaalide mõistmisel ülimalt kasulikuks. Samuti on bioinformaatikas kindlasti veel lugematul arvul andmestikke ja probleeme, kus on alust arvata, et konvolutsioonilised võrgud võiksid osutuda efektiivsemaks kui senised meetodid. Loodame, et käesolev doktoritöö julgustab ja innustab teadlasi uusi meetodeid proovima ka oma andmestikel.

PUBLICATIONS

CURRICULUM VITAE

Personal data

Name: Ardi Tampuu
Birth: 24 March 1988
Citizenship: Estonian
Languages: Estonian, English, French
Contact: arditampuu@gmail.com

Education

2014–2020	University of Tartu, Institute of Comupter Science, PhD studies
2008–2013	Institut National des Sciences Appliqués de Lyon, Engineering degree in "Bioinformatics and Modelling"
2012–2013	Université Claude Bernard Lyon 1, MSc thesis in "Mathematics and Informatics of Life"
2004-2007	Hugo Treffner Gymnasium

Employment

2013–2014	Scientific programmer, University of Tartu
2013–2019	Teaching assistant (variety of courses), University of Tartu
2014–2019	Junior researcher, University of Tartu
2019–2020	Research Software Engineer, University of Tartu

Scientific work

Main fields of interest:

- Machine Learning, with particular emphasis on deep learning
- Neuroscience, decoding neural code
- Computational biology

ELULOOKIRJELDUS

Isikuandmed

Nimi: Ardi Tampuu
Sünniaeg: 24 märts 1988
Kodakondsus: Eesti
Keeled: Eesti, Inglise, Prantsuse
Kontakt: arditampuu@gmail.com

Haridus

2014–2020	Tartu Ülikool, Arvutiteaduse Instituut, doktoriõpe
2008–2013	Lyoni Rahvuslik Rakendusteaduste Instituut, insenerikraad teemal "Bioinformatics and Modelling"
2012–2013	Claude Bernardi Ülikool Lyon 1, magistritöö teemal "Mathematics and Informatics of Life"
2004-2007	Hugo Treffneri Gümnaasium

Teenistuskäik

2013–2014	Teaduslik programmeerija, Tartu Ülikool
2013–2019	Praktikumide juhendaja erinevates õppeaintes, Tartu Ülikool
2014–2019	nooremteadur, Tartu Ülikool
2019–2020	teadustarkvara insener, Tartu Ülikool

Teadustegevus

Peamised uurimisvaldkonnad:

- masinõpe, põhirõhuga tehisnärvivõrkudel
- arvutuslik neuroteadus, neurokoodi dekodeerimine
- arvutuslik bioloogia

DISSERTATIONES INFORMATICAE PREVIOUSLY PUBLISHED IN DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS

19. **Helger Lipmaa.** Secure and efficient time-stamping systems. Tartu, 1999, 56 p.
22. **Kaili Müürisep.** Eesti keele arvutigrammatika: süntaks. Tartu, 2000, 107 lk.
23. **Varmo Vene.** Categorical programming with inductive and coinductive types. Tartu, 2000, 116 p.
24. **Olga Sokratova.** Ω-rings, their flat and projective acts with some applications. Tartu, 2000, 120 p.
27. **Tiina Puolakainen.** Eesti keele arvutigrammatika: morfoloogiline ühestamine. Tartu, 2001, 138 lk.
29. **Jan Villemson.** Size-efficient interval time stamps. Tartu, 2002, 82 p.
45. **Kristo Heero.** Path planning and learning strategies for mobile robots in dynamic partially unknown environments. Tartu 2006, 123 p.
49. **Härmel Nestra.** Iteratively defined transfinite trace semantics and program slicing with respect to them. Tartu 2006, 116 p.
53. **Marina Issakova.** Solving of linear equations, linear inequalities and systems of linear equations in interactive learning environment. Tartu 2007, 170 p.
55. **Kaarel Kaljurand.** Attempto controlled English as a Semantic Web language. Tartu 2007, 162 p.
56. **Mart Anton.** Mechanical modeling of IPMC actuators at large deformations. Tartu 2008, 123 p.
59. **Reimo Palm.** Numerical Comparison of Regularization Algorithms for Solving Ill-Posed Problems. Tartu 2010, 105 p.
61. **Jüri Reimand.** Functional analysis of gene lists, networks and regulatory systems. Tartu 2010, 153 p.
62. **Ahti Peder.** Superpositional Graphs and Finding the Description of Structure by Counting Method. Tartu 2010, 87 p.
64. **Vesal Vojdani.** Static Data Race Analysis of Heap-Manipulating C Programs. Tartu 2010, 137 p.
66. **Mark Fišel.** Optimizing Statistical Machine Translation via Input Modification. Tartu 2011, 104 p.
67. **Margus Niitsoo**. Black-box Oracle Separation Techniques with Applications in Time-stamping. Tartu 2011, 174 p.
71. **Siim Karus.** Maintainability of XML Transformations. Tartu 2011, 142 p.
72. **Margus Treumuth.** A Framework for Asynchronous Dialogue Systems: Concepts, Issues and Design Aspects. Tartu 2011, 95 p.
73. **Dmitri Lepp.** Solving simplification problems in the domain of exponents, monomials and polynomials in interactive learning environment T-algebra. Tartu 2011, 202 p.

74. **Meelis Kull.** Statistical enrichment analysis in algorithms for studying gene regulation. Tartu 2011, 151 p.
77. **Bingsheng Zhang.** Efficient cryptographic protocols for secure and private remote databases. Tartu 2011, 206 p.
78. **Reina Uba.** Merging business process models. Tartu 2011, 166 p.
79. **Uuno Puus.** Structural performance as a success factor in software development projects – Estonian experience. Tartu 2012, 106 p.
81. **Georg Singer.** Web search engines and complex information needs. Tartu 2012, 218 p.
83. **Dan Bogdanov.** Sharemind: programmable secure computations with practical applications. Tartu 2013, 191 p.
84. **Jevgeni Kabanov.** Towards a more productive Java EE ecosystem. Tartu 2013, 151 p.
87. **Margus Freudenthal.** Simpl: A toolkit for Domain-Specific Language development in enterprise information systems. Tartu, 2013, 151 p.
90. **Raivo Kolde.** Methods for re-using public gene expression data. Tartu, 2014, 121 p.
91. **Vladimir Šor.** Statistical Approach for Memory Leak Detection in Java Applications. Tartu, 2014, 155 p.
92. **Naved Ahmed.** Deriving Security Requirements from Business Process Models. Tartu, 2014, 171 p.
94. **Liina Kamm.** Privacy-preserving statistical analysis using secure multi-party computation. Tartu, 2015, 201 p.
100. **Abel Armas Cervantes.** Diagnosing Behavioral Differences between Business Process Models. Tartu, 2015, 193 p.
101. **Fredrik Milani.** On Sub-Processes, Process Variation and their Interplay: An Integrated Divide-and-Conquer Method for Modeling Business Processes with Variation. Tartu, 2015, 164 p.
102. **Huber Raul Flores Macario.** Service-Oriented and Evidence-aware Mobile Cloud Computing. Tartu, 2015, 163 p.
103. **Tauno Metsalu.** Statistical analysis of multivariate data in bioinformatics. Tartu, 2016, 197 p.
104. **Riivo Talviste.** Applying Secure Multi-party Computation in Practice. Tartu, 2016, 144 p.
108. **Siim Orasmaa.** Explorations of the Problem of Broad-coverage and General Domain Event Analysis: The Estonian Experience. Tartu, 2016, 186 p.
109. **Prastudy Mungkas Fauzi.** Efficient Non-interactive Zero-knowledge Protocols in the CRS Model. Tartu, 2017, 193 p.
110. **Pelle Jakovits.** Adapting Scientific Computing Algorithms to Distributed Computing Frameworks. Tartu, 2017, 168 p.
111. **Anna Leontjeva.** Using Generative Models to Combine Static and Sequential Features for Classification. Tartu, 2017, 167 p.
112. **Mozhgan Pourmoradnasseri.** Some Problems Related to Extensions of Polytopes. Tartu, 2017, 168 p.

113. **Jaak Randmets.** Programming Languages for Secure Multi-party Computation Application Development. Tartu, 2017, 172 p.
114. **Alisa Pankova.** Efficient Multiparty Computation Secure against Covert and Active Adversaries. Tartu, 2017, 316 p.
116. **Toomas Saarsen.** On the Structure and Use of Process Models and Their Interplay. Tartu, 2017, 123 p.
121. **Kristjan Korjus.** Analyzing EEG Data and Improving Data Partitioning for Machine Learning Algorithms. Tartu, 2017, 106 p.
122. **Eno Tõnisson.** Differences between Expected Answers and the Answers Offered by Computer Algebra Systems to School Mathematics Equations. Tartu, 2017, 195 p.

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS

1. **Abdullah Makkeh**. Applications of Optimization in Some Complex Systems. Tartu 2018, 179 p.
2. **Riivo Kikas**. Analysis of Issue and Dependency Management in Open-Source Software Projects. Tartu 2018, 115 p.
3. **Ehsan Ebrahimi**. Post-Quantum Security in the Presence of Superposition Queries. Tartu 2018, 200 p.
4. **Ilya Verenich**. Explainable Predictive Monitoring of Temporal Measures of Business Processes. Tartu 2019, 151 p.
5. **Yauhen Yakimenka**. Failure Structures of Message-Passing Algorithms in Erasure Decoding and Compressed Sensing. Tartu 2019, 134 p.
6. **Irene Teinemaa**. Predictive and Prescriptive Monitoring of Business Process Outcomes. Tartu 2019, 196 p.
7. **Mohan Liyanage.** A Framework for Mobile Web of Things. Tartu 2019, 131 p.
8. **Toomas Krips.** Improving performance of secure real-number operations. Tartu 2019, 146 p.
9. **Vijayachitra Modhukur.** Profiling of DNA methylation patterns as biomarkers of human disease. Tartu 2019, 134 p.
10. **Elena Sügis.** Integration Methods for Heterogeneous Biological Data. Tartu 2019, 250 p.
11. **Tõnis Tasa.** Bioinformatics Approaches in Personalised Pharmacotherapy. Tartu 2019, 150 p.
12. **Sulev Reisberg.** Developing Computational Solutions for Personalized Medicine. Tartu 2019, 126 p.
13. **Huishi Yin.** Using a Kano-like Model to Facilitate Open Innovation in Requirements Engineering. Tartu 2019, 129 p.
14. **Faiz Ali Shah.** Extracting Information from App Reviews to Facilitate Software Development Activities. Tartu 2020, 149 p.
15. **Adriano Augusto**. Accurate and Efficient Discovery of Process Models from Event Logs. Tartu 2020, 194 p.
16. **Karim Baghery.** Reducing Trust and Improving Security in zk-SNARKs and Commitments. Tartu 2020, 245 p.
17. **Behzad Abdolmaleki.** On Succinct Non-Interactive Zero-Knowledge Protocols Under Weaker Trust Assumptions. Tartu 2020, 209 p.
18. **Janno Siim.** Non-Interactive Shuffle Arguments. Tartu 2020, 154 p.
19. **Ilya Kuzovkin.** Understanding Information Processing in Human Brain by Interpreting Machine Learning Models. Tartu 2020, 149 p.
20. **Orlenys López Pintado**. Collaborative Business Process Execution on the Blockchain: The Caterpillar System. Tartu 2020, 170 p.