FREQUENCY DOMAIN METHODS IN RECURRENT NEURAL NETWORKS FOR SEQUENTIAL DATA PROCESSING

DISSERTATION

zur Erlangung des Doktorgrades (Dr. rer. nat.)

der Mathematisch-Naturwissenschaftlichen Fakultät

der Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

MORITZ WOLTER

aus Siegburg

Bonn, 2021

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

Moritz Wolter: *Frequency Domain Methods in Recurrent Neural Networks for Sequential Data Processing,* a cumulative doctoral thesis, © 30.06.2021

GUTACHTER:

1. Prof. Dr. Angela Yao
2. Prof. Dr. Reinhard Klein

ERSCHEINUNGSJAHR:
2021

Wir hoffen, mit Hilfe eines neu zu errichtenden wissenschaftlichen Systems neue Vorgänge zu entdecken; an dem falsifizierenden Experiment haben wir höchstes Interesse, wir buchen es als Erfolg, denn es eröffnet uns Aussichten in eine neue Welt von Erfahrungen; und wir begrüßen es, wenn diese uns neue Argumente gegen die neuen Theorien liefert.

— Karl Popper, Logik der Forschung, [Pop35]

ABSTRACT

Machine learning algorithms now make it possible for computers to solve problems, which were thought to be impossible to automize. Neural Speech processing [Cha+16], convolutional neural networks [Vas+15], and other recent advances are powered by frequency-domain methods like the fast Fourier transform (FFT).

This cumulative thesis presents applications of frequency-domain methods in recurrent machine learning. It starts by exploring the combination of the short time Fourier transform (STFT) and recurrent neural networks. This combination allows faster training through windowing, end-to-end window function optimization, while low-pass filtering the Fourier coefficients can reduce the model size. Fourier coefficients are complex numbers, and therefore best processed in $\mathbb{C}$. The development of a complex recurrent memory cell is an additional contribution of this text. To move a modern recurrent neural network (RNN)-cell into the complex domain, we must make various design choices regarding the gating mechanism, state transition matrix, and activation functions. The design process introduces a new complex gate activation function the *modSigmoid*. Afterwards, we explore the interplay of state transition matrices and cell activation functions. It is confirmed that unbounded non-linearities require unitary or orthogonal state transition matrices to be stable.

General-purpose machine learning models often produce blurry video predictions. By using the phase of frames in their frequency domain representation, it is possible to do better. Image registration methods allow the extraction of transformation parameters. For single pre-segmented objects on input video frames, phase modification can help to predict future images.

The FFT represents all inputs in the fixed Fourier representation. The fast wavelet transform (FWT) works with infinitely many wavelets, all of which can serve as potential bases. This text proposes a loss function, which allows wavelet optimization and integrates the FWT into convolutional and recurrent neural networks. Replacing dense linear weight matrices with sparse diagonal matrices and fast wavelet transforms allows spectacular parameter reductions without performance loss in some cases. Finally, the last chapter finds that wavelet quantization can reduce the memory space required to store and transmit a convolutional neural network.

ZUSAMMENFASSUNG

Maschinelle Lernalgorithmen erlauben es, Programme zu entwickeln, die Probleme lösen, die noch vor Kurzem für Computer als unlösbar galten. Fortschritte in der neuronalen Sprachverarbeitung [Cha+16], schnelle Faltungsnetze [Vas+15], und andere Neuentwicklungen jüngerer Zeit nutzen die schnelle Fourier Transformation (FFT).

Diese kumulative Arbeit widmet sich der Kombination von maschinellen Lernalgorithmen und Datenverarbeitung im Frequenzbereich. Die Kurzzeit-Fourier-Transformation wird mit rückgekoppelten neuronalen Netzen kombiniert. Diese Kombination erlaubt es, die Fenster-Funktion gemeinsam mit allen Gewichten zu optimieren. Sie beschleunigt den Lernprozess und ermöglicht durch Tiefpass-Filtern die Netzgröße zu reduzieren.

Fourier-Koeffizienten sind komplexe Zahlen, um sie im komplexen Zahlenraum verarbeiten zu können, wird der Entwurf komplexer rückgekoppelter Speicherzellen diskutiert. Hierbei kommt den Zell-Toren, der Aktivierungs-Funktion sowie der Zustands-Matrix besondere Bedeutung zu. Für komplexwertige Tor-Gleichungen wird die *ModSigmoid*-Aktivierung vorgeschlagen. Darüber hinaus wird bestätigt, dass unbeschränkte Zell-Aktivierungs-Funktionen orthogonale oder unitäre Zustandsmatrizen benötigen, um eine stabile Zelle zu bilden.

Klassische maschinelle Lernmodelle produzieren oft verschmierte Vorhersagen auf Video-Daten. Diese Arbeit enthält einen Lösungsvorschlag für Video-Bilder mit nur einem präsegmentierten Objekt. In diesem Fall lassen sich, mit Hilfe von Bildregistrierungsmethoden, Transformationsparameter aus der Phase vorheriger Frames ableiten. Mit Hilfe dieser Parameter lässt sich dann eine Vorhersage errechnen, indem die Phase des aktuellen Bildes modifiziert wird. Eine rückgekoppelte Zelle zu diesem Zweck wird vorgestellt.

Im Vergleich zur schnellen Fourier Transformation, die immer die gleiche Basis Nutzt stehen für die schnelle Wavelet Transformation unendlich viele Basis-Funktionen zur Verfügung. Aus allen möglichen Wavelets das Richtige auszuwählen ist nicht immer leicht. In dieser Arbeit wird daher eine Kostenfunktion zur automatischen Optimierung von Wavelets vorgeschlagen und die schnelle Wavelet Transformationen zur Kompression neuronaler Netze genutzt. Anstelle dicht besetzter Gewichtsmatrizen lassen sich Diagonalmatrizen in Verbindung mit den Vorwärts- und Rückwärts-Transformationen verwenden. In einigen Fälle hat dieser Ansatz keinen Genauigkeitsverlust zur Folge. Im letzten Kapitel wird abschließend ein Faltungsnetz mit Hilfe von Wavelet Quantisierung und Huffman Kodierung komprimiert.

PUBLICATIONS

The chapters in the research part of this cumulative thesis have previously appeared in scientific papers. I am grateful to have had the opportunity to publish the following first author papers:

[WGY20] Moritz Wolter, Juergen Gall, and Angela Yao. "Sequence Prediction using Spectral RNNs." In: *29th International Conference on Artificial Neural Networks*. 2020. DOI: 10.1007/978-3-030-61609-0_65.

[WLY20] Moritz Wolter, Shaohui Lin, and Angela Yao. "Neural network compression via learnable wavelet transforms." In: *29th International Conference on Artificial Neural Networks*. 2020. DOI: 10.1007/978-3-030-61616-8_4.

[WY18] Moritz Wolter and Angela Yao. "Complex Gated Recurrent Neural Networks." In: *Advances in Neural Information Processing Systems 31*. 2018. URL: papers.nips.cc/paper/8253-complex-gated-recurrent-neural-networks.

[WYB20] Moritz Wolter, Angela Yao, and Sven Behnke. "Object-centered Fourier Motion Estimation and Segment-Transformation Prediction." In: *28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. 2020. URL: esann.org/sites/default/files/proceedings/2020/ES2020-100.pdf.

The papers listed above reappear unchanged as chapters in the research part of this text. Additionally I contributed to the following publications, which are not part of this thesis:

[Bru+20] Lilli Bruckschen, Kira Bungert, Moritz Wolter, Stefan Krumpen, Michael Weinmann, Reinhard Klein, and Maren Bennewitz. "Where Can I Help? Human-Aware Placement of Service Robots." In: *International Symposium on Robot and Human Interactive Communication*. 2020. DOI: 10.1109/RO-MAN47096.2020.9223331.

[Ene+20] Kristina Enes, Hassan Errami, Moritz Wolter, Tim Krake, Bernhard Eberhardt, Andreas Weber, and Jörg Zimmermann. "Unsupervised and Generic Short-Term Anticipation of Human Body Motions." In: *Sensors* 20.4 (2020), p. 976. DOI: 10.3390/s20040976.

[Fra+20] Lukas Franken, Bogdan Georgiev, Sascha Muecke, Moritz Wolter, Nico Piatkowski, and Christian Bauckhage. "Gradient-free quantum optimization on NISQ devices." In: *arXiv preprint arXiv:2012.13453* (2020).

ACKNOWLEDGMENTS

I would like to thank my supervisor, Angela Yao, for giving me the opportunity and freedom to pursue my ideas and write this thesis. Without her encouragement, enthusiasm, support, and input this text would not exist.

My co-supervisor Reinhard Klein was also extremely helpful, even though we met less often, he was always ready to help and guided me through even the hardest master and bachelor thesis supervision challenges.

I am also indebted to my colleagues Michael, Linlin, Soumajit, and Fadime, in coronavirus times, we don't meet as much at the office anymore, but when we did, I very much enjoyed the discussions and fun we had.

Last but not least, I would like to thank my family. My parents Udo and Beatrice, my siblings Nicola and Felix, for supporting, and encouraging me during all these years of study.

CONTENTS

1 INTRODUCTION 1
- 1.1 Motivation 1
- 1.2 Understanding recurrent memory 2
  - 1.2.1 Contributions to complex network architectures [WY18] 2
- 1.3 Time series prediction 3
  - 1.3.1 Contributions to STFT and (complex) RNN integration [WGY20] 3
- 1.4 Video segment prediction challenges 4
  - 1.4.1 Contribution: Segment prediction via phase modification [WYB20] 4
- 1.5 The challenge of compressing neural networks 5
  - 1.5.1 Contribution: Wavelet optimization and network compression via the fast wavelet transform (FWT) [WLY20] 5
- 1.6 Outline of the thesis structure 5
- 1.7 Open Research Approach 6

I BACKGROUND

2 NETWORK STRUCTURES 9
- 2.1 Cost functions 9
- 2.2 Fully Connected Layers and Networks 10
- 2.3 Convolutional neural networks 11
  - 2.3.1 Convolution as Matrix Mutliplication 13
  - 2.3.2 Backpropagation 13
- 2.4 Recurrent neural networks 14
  - 2.4.1 Simple-RNN 14
  - 2.4.2 Long Short Term Memory 16
  - 2.4.3 Gated Recurrent Units 21

3 OPTIMIZATION 27
- 3.1 Momentum 27
- 3.2 RMSProp 28
- 3.3 Adam 28

4 SIGNAL PROCESSING 31
- 4.1 Complex numbers 31
- 4.2 The Discrete Fourier Transform 32
- 4.3 The Fast Fourier Transform 33
- 4.4 The Short Time Fourier Transform 34
  - 4.4.1 Forwards STFT 35
  - 4.4.2 Inverse STFT 35
- 4.5 Wavelets 36
  - 4.5.1 The fast wavelet transform 37

4.5.2 Wavelet properties 39

II RESEARCH

5 RELATED WORK 43
5.1 Recurrent Neural Networks 43
5.2 Complex Networks 44
5.3 Fourier Networks 46
5.4 Wavelet Networks 47
5.5 Network compression 47
6 SPECTRAL RECURRENT NEURAL NETWORKS [WGY20] 49
6.1 Introduction 49
6.2 Related Works 50
6.3 Complex Spectral Recurrent Neural Networks 51
6.3.1 Network Structure 51
6.3.2 Loss Functions 52
6.4 Mackey-Glass Chaotic Sequence Prediction 52
6.4.1 Implementation Details 53
6.4.2 Experimental Results and Ablation Studies 53
6.5 Power Load Forecasting 55
6.5.1 Data 55
6.5.2 Day-Ahead Prediction 55
6.5.3 Long-Term Forecast 56
6.6 Human Motion Forecasting 57
6.6.1 Dataset & Evaluation Measure 57
6.6.2 Implementation Details 57
6.6.3 Motion Forecasting Results 58
6.7 Summary and outlook 58
7 COMPLEX GATED RECURRENT NEURAL NETWORKS [WY18] 61
7.1 Introduction 61
7.2 Related work 62
7.3 Preliminaries 63
7.3.1 Complex Gradients 63
7.3.2 A Split Complex Approach 64
7.4 Complex Gated RNNs 65
7.4.1 Basic Complex RNN Formulation 65
7.4.2 Complex Non-linear Activation Functions 65
7.4.3 Real to Complex input and Complex to Real output mappings 66
7.4.4 Optimization on the Stiefel Manifold for Norm Preservation 66
7.4.5 Complex-Valued Gating Units 67
7.5 Experimentation 67
7.5.1 Tasks & Evaluation Metrics 67
7.5.2 RNN Implementation Details 69
7.5.3 Impact of Gating and Choice of Gating Functions 70

7.5.4 Non-Linearity Choice and Norm Preservation 71
7.5.5 Real World Tasks: Human Motion Prediction & Music Transcription 73
7.6 Conclusion 74
8 PHASE BASED FRAME PREDICTION [WYB20] 77
8.1 Related Work 77
8.2 Methods for Motion Estimation 78
8.3 Neural Network Parameter Correction 78
8.4 Fourier Domain Image Transformation 79
8.5 Video Frame Prediction 80
8.5.1 Translation 80
8.5.2 Rotation and Translation 81
8.6 Conclusion 82
9 WAVELET-LEARNING [WLY20] 83
9.1 Related Work 85
9.1.1 Structured Efficient Linear Transforms 85
9.1.2 Compressing Recurrent Neural Networks 86
9.2 Method 86
9.2.1 Wavelet Basis Learning 86
9.2.2 Efficient Wavelet-based Linear Layers 87
9.3 Retraining a compressed wavelet layer 88
9.3.1 MNIST-Digit Recognition 88
9.3.2 RNN-Compression 90
9.4 Summary and Outlook 94
9.5 Compression trough wavelet quantization 94
9.5.1 Quantization 95
9.5.2 Wavelet compression of a quantized convolutional neural network 95
9.6 Summary and Outlook 96
10 CONCLUSION 97
10.1 Lessons learned 97
10.1.1 STFT learning 97
10.1.2 Complex Networks 98
10.1.3 Video Prediction 98
10.1.4 FWT integration in neural nets 98
10.1.5 Choosing a Transform 98
10.2 Reproducibility and open source 99
10.3 Future Work 99

BIBLIOGRAPHY 101

LIST OF FIGURES

Figure 1.1 Example memory and adding problem benchmark inputs for T=100. These problems were first proposed in [HS97a] and later adapted by [ASB16]. 2

Figure 1.2 Surface plots showing the magnitude of the bounded Hirose and unbounded modReLU activations [WY18]. 3

Figure 1.3 Mono- and multi-variate sequence data. Belgian power load in January 2016 (left) as well as a sequence snippet from the human 3.6m data set. The right image previously appeard in [WGY20]. 4

Figure 2.1 Visualization of multi-channel convolution. Convolution computation using a 3x2x3x3 kernel on a 2x5x5 image is shown. The image is read from left to right, starting with the two-channel five by five image on the very left. In a first step, the two input kernels are convolved with the input channels three times, as indicated by the convolution blocks $*$. Without padding, this operation leads to the 3x3 results. Moving towards the right, we add $+$ the two channels for each of the three resulting tensors. Finally, everything is stacked, which leads to the final result. [DV16] inspired this illustration, it contains an excellent introduction to the topic. 12

Figure 2.2 Visualization of a simple recurrent cell. Output $\mathbf{y}_t$ and cell state $\mathbf{c}_t$ at time t depend on the previous state $\mathbf{c}_{t-1}$ and the current input $\mathbf{x}_t$. Figure similar to[Wol17]. 14

Figure 2.3 The rolled (left) cell can be unrolled (right) by considering all inputs it saw during the current gradient computation iteration. Figure shown as found in [Wol17]. 15

Figure 2.4 Visualization of a long short term memory cell. Green boxes contain concatenation operations. The state line on the very left records all changes to the cell's memory. All gate equations appear in yellow. The gates use sigmoid activations. The sigmoid activation produces values within zero and one. A zero output means the corresponding value is blocked. One means it is allowed to pass through. State candidate and output values are run through the tanh function shown in orange. This ensures the cell's memory contents and output values are within $[-1, 1]$. This Figure is similar to [Wol17]. 18

Figure 2.5 Visualization of a gated recurrent unit (GRU). This long short term memory (LSTM) variant uses only two gates and a single state activation function. The two yellow gates use sigmoidal activations and produce outputs within zero and one. The gate outputs govern cell memory management. The update gate replaces input and forget gates, its output decides with memory contents can be modified. The reset gate allows state value deletion from the block input. The tanh activation appears only once. No output gate is used. 23

Figure 4.1 Visualization of a Fourier Series approximating a rectangular pulse (left). As more and more terms are used the approximation's accuracy improves. On the right the magnitued of all fourier coefficients is shown. 32

Figure 4.2 Repeated multiplication by i, causes rotation around the origin. Its important to note that 1 can also be represented by i^4, a full rotation. Full rotations can also be present in representations for any other point on the unit circle. Minus one for example could also be i^6. 33

Figure 4.3 Wavelet approximation of a rectangular pulse function with an increasing number of scales (left). Wavelet coefficients for all scales (right). 36

Figure 4.4 Haar analysis fast wavelet transformation matrix on the left followed by individual scale processing matrices. 37

Figure 4.5 Haar synthesis backward fast wavelet transformation matrices for three scales as well as the complete inverse matrix. 38

Figure 4.6 Efficient wavelet signal analysis and synthesis following a tree structure [SN96]. **H** denotes analysis filters and **F** stands for synthesis filters. Up (↑) and down (↓)- sampling by a factor of two is written as the arrow followed by the factor. Filtering and sampling can be accomplished jointly in deep learning frameworks by using strided convolutions for analysis and strided transposed convolutions for synthesis. In place of the dotted arrow, more scale-levels can be included. 38
Figure 5.1 Popular Feedforward activation functions in $\mathbb{C}$. The zReLU [Gub16] allows complex numbers with positive real and imaginary parts to pass. While the cReLU [Tra+18], applies standard-ReLUs seperately on the real and imaginary parts. 45
Figure 5.2 Popular activation functions in $\mathbb{C}$. The Hirose activation emplys the hyperbolic tangent function to bound the radius at 1. The modReLU activation creates a learnable dead zone around the origin. Like its real counterpart, the modReLU is unbounded. Note that both activations preserve phase information. 45
Figure 6.1 Mackey-Glass series predictions in for different RNN methods. As gradients flow through the STFT we can optimize the width of the gaussian σ. The learned window width for increasing degrees of low-pass filtering is shown here. Figure best viewed in colour. 53
Figure 6.2 Day ahead prediction results convergence (left) and prediction examples (right). We observe that all deep learning approaches beat the entsoe.eu baseline shown as the red line, which suggests that their approach could benefit from deep learning. 56
Figure 6.3 A test set sample for showing the 60 day prediction results for all architectures under consideration. Close up for the last week of the 60 day prediction. 57
Figure 6.4 Visualization of input and prediction results using a STFT-RNN combination and low pass filtering. Input is shown in red and blue, predictions in green and yellow. 58
Figure 7.1 Surface plots of the magnitude of the Hirose ($m^2 = 1$) and modReLU ($b = -0.5$) activations. 65

Figure 7.2 Comparison of our cgRNN (blue, $n_h = 80$) with the uRNN [ASB16] (orange, $n_h = 140$) and standard GRU [Cho+14] (green, $n_h = 112$) on the memory (a) and adding (b) problem for $T = 250$. The hidden state size n_h for each network are chosen so as to approximately match the number of parameters (approximately 44k parameters total). On the memory problem, having norm-preserving state transition matrices is critical for stable learning, while on the adding problem, having gates is important. 70
Figure 7.3 Comparison of non-linearities and norm preserving state transition matrices on the cgRNNs for the memory (a) and adding (b) problems for T=250. The unbounded modReLU (see Equation 7.7) performs best for both problems, but only if the state transition matrices are kept unitary. Without unitary state-transition matrices, the bounded Hirose non-linearity (see Equation 7.6) performs better. We use $n_h = 80$ for all experiments. 73
Figure 8.1 Overview of our estimation, correction and transformation framework. The estimator (est) finds transformation parameters between the last and current frame based on phase correlation and computes the object centroid. The parameters are corrected by the network (net) based on its encoding of the history s, by computing a residual which is added to the current estimate. Finally, the transformer (trans) transforms the last image using the phase-shift property of the Fourier transform to create the prediction. 79
Figure 8.2 Moving MNIST translation prediction. Ground truth (top), our estimation correction cell prediction (middle) and a standard GRU of size 512 (bottom) are shown. Predictions are made using 4 context frames. We observe that our approach produces predictions which remain sharp, while the much larger GRU cell's predictions are blurry. 80
Figure 8.3 Rotating moving MNIST prediction, ground truth (top), our estimation correction transformation cell output (middle) and standard GRU (bottom). 81

Figure 9.1 Wavelet loss sum of a randomly and haar initialized wavelet array. In both cases, filter values converge to a product filter as indicated by trend of the wavelet loss towards zero. 89

Figure 9.2 Learned wavelet filter coefficients. Coefficients have been initialized at random. After training, the effects of the alias cancellation constraint are prominently visible. We must have $F_0(z) = H_1(-z)$ and $F_1(z) = -H_0(-z)$ for alias to cancel itself. Inserting $(-z)$ into the coefficient polynomial leads to a minus sign at odd powers. Additional multiplication with (-1) shifts it to even powers. Alias cancellation therefore imposes an alternating sign pattern. When F_0 and H_1 share the same sign F_1 and H_0 do not and vice versa. 89

Figure 9.3 Accuracy and parameters of gated recurrent units with and without a compressed reset gate on the memory problem with $T = 60$. Both models start with 1K parameters. As we increase the state size from 12 to 108, the parameter count rises to 39K for the standard GRU and to 29K for the compressed version. We observe that reset gate compression increases parameter efficiency in this case. 90

Figure 9.4 Accuracy versus scale and file size for a MNIST 4 layer CNN example. We observe that the wavelet coefficient are more resistant to quantization in this case. The shifted peak in the right indicates that the file size can roughly be cut in half here. 95

LIST OF TABLES

Table 6.1 Short time Fourier, Windowed and Time Domain results obtained using GRU cells of size 64. Windowed experiments process multiple samples of data without computing the STFT. Additionally we compare low-pass filtering the spectrum and downsampling the time domain windows. All models where trained for 30k iterations. We downsample and lowpass-filter with a factor of 1/32. 54
Table 6.2 Real and complex valued architecture comparison on the mackey-glass data, with increasing complex cell size. The complex architectures take longer to run but are more parameter efficient. The last row shows a complex RNN cell in STFT space without iSTFT backpropagation. 54
Table 6.3 60 day ahead power load prediction using GRUs of size 64. We downsample and lowpass-filter with a factor of 1/4. We observe that windowing leads to large training and inference speed-ups. Our STFT approach performs better in the full spectrum case and with a reduced input-dimensionality. 56
Table 6.4 3d-Human motion forecast of 64 frames or approximately one second. Mean absolute error (mae) is measured in mm. Mean squared errors are reported in mm^2. We downsample and lowpass-filter with a factor of 1/4. Windowing runs much faster than the naive time domain approach. Among windowed approaches the STFT allows more aggressive input size reductions. 59
Table 7.1 Comparison of gating functions on the adding and memory problems. 72
Table 7.2 Comparison of our cgRNN with the GRU [MBR17] on human motion prediction. 74

Table 8.1 Evaluation using 550 prediction and ground truth sequences. The mean and standard deviation of our predictions are very close to the ground truth. Our approach performs slightly worse in terms of mean squared error with significantly fewer parameters and remains sharp. 81
Table 9.1 Experimental results on the MNIST digit-recognition. We work with a LeNet architecture as proposed in previous work. In comparison to the fast-food approach [Yan+15] we obtain comparable performance with slightly fewer parameters. The size of our learnable-wavelet compression layer is set to 800. 89
Table 9.2 RNN compression results on the adding and memory problems, exploring the impact of our efficient wavelet-based linear layer at various locations in the GRU. On the adding problem all tested variants are functional. Compressing the state and reset equations has virtually no effect on performance. Compressing the update gate leads to a working cell, but cells with a compressed update gate perform significantly worse. Note that on the adding problem, predicting a sum of 1 regardless of the input leads to an mse of 0.167. On the copy-memory benchmark, replacing the the state and reset weight matrices with our efficient wavelet version is possible without significant performance losses. A state size of 512 was used for all models. The expected cross entropy for a random guess is 0.12 with n=8. 91
Table 9.3 RNN compression results on the sequential MNIST benchmark. The pattern here reflects what we saw on the adding and copy-memory benchmarks. Touching the update gate has a negative impact. All other equations can be compressed. our method (WaveGRU-64) achieves a comparable performance, compared to [TSN17]. 93
Table 9.4 Results for the best performing architectures on the Penn-Treebank data set, we compare to a TCN as proposed in [BKK18]. We can compress the GRU cells state and reset equations without a significant drop in performance. 94

ACRONYMS

FFT	fast Fourier transform
iFFT	inverse fast Fourier transform
STFT	short time Fourier transform
iSTFT	inverse short time Fourier transform
FWT	fast wavelet transform
iFWT	inverse fast wavelet transform
RNN	recurrent neural network
CNN	convolutional neural network
LSTM	long short term memory
GRU	gated recurrent unit
MLP	multilayer perceptron
cgRNN	complex gated recurrent neural network
uRNN	unitary recurrent neural network
WHT	Walsh-Hadamard transform

1 INTRODUCTION

This cumulative thesis explores links between frequency-domain methods and recurrent machine learning for sequential data processing. Each chapter in the research part corresponds to a previously published paper. The background part appears in this text for the first time. It explains the methods and algorithms the research part builds upon.

1.1 MOTIVATION

Recurrent neural networks (RNN) are the go-to choice for sequence processing [GBC16], however, the best design of their internal machinery is not immediately obvious. Three goals lie at the heart of the RNN design process. The first is stability, ideally a good cell-structure should be provably stable. The second and equally important is to make the memory capacity as large as possible. RNN should be able to take as many past inputs into account as possible. Last but not least, any design must be efficient. This includes the number of parameters and the overall computational cost.

Gated recurrent memory cells outperform simple RNN in terms of memory capacity because gradients rarely vanish. The added gates come at the expense of increased network size, and gated architectures still suffer from the exploding gradient problem. Exploding gradients limit network stability over long time horizons. The three design goals are coupled. Reducing network sizes and the study of network stability and efficiency are the main motivations for the research questions and challenges studied in this text.

Frequency domain methods often introduce structure into our network weights or input data. Depending on the situation, such structure allows sparse diagonal weights instead of dense matrices or lowpass filtered inputs instead of the entire spectrum. Using structure not only allows us to reduce the number of network parameters but also unlocks input features, which improve training convergence. Advancing the integration of frequency-domain methods into neural networks further is an additional goal of this thesis.

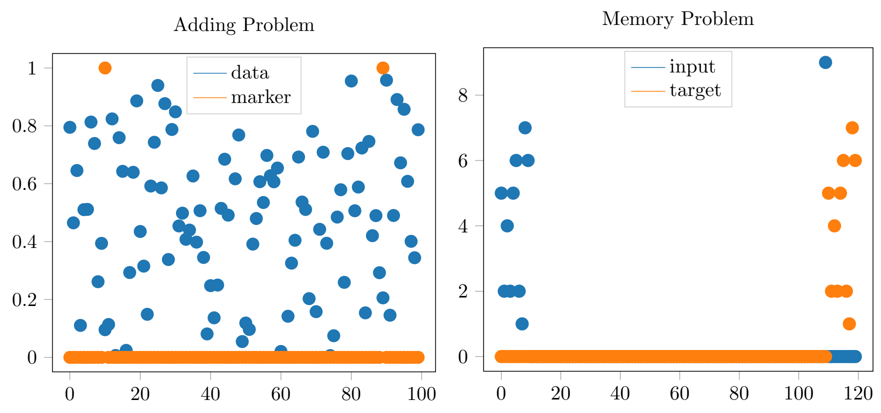


Figure 1.1: Example memory and adding problem benchmark inputs for T=100. These problems were first proposed in [HS97a] and later adapted by [ASB16].

1.2 UNDERSTANDING RECURRENT MEMORY

The adding and memory problems [Hoc91] [ASB16] are benchmark challenges for recurrent neural networks. New architectures are often first evaluated on these two problems. The two are visualized in figure 1.1.

The problems have length T. Example adding problem network input sequences are shown on the left. This problem uses two-channel inputs. The first channel consists of T samples drawn from a uniform distribution $\mathcal{U}[0, 1]$. The second channel marks a sample in the first and another sample in the second half. All other samples are marked with zeros. After observing all sequence pairs, the benchmarked RNN architecture must produce the sum of both marked samples.

The right side of figure 1.1 shows a copy-memory problem-input as well as the corresponding desired output. The challenge for the network architecture under evaluation is to observe a sequence of ten integers and to later reproduce this sequence after T additional zero inputs.

1.2.1 *Contributions to complex network architectures [WY18]*

In chapter 7 both problems are used to design a complex gated recurrent neural network (cgRNN), while keeping stability, memory capacity, and efficiency in mind. To do so requires a closer look at complex activation functions and their effects on network stability. For recurrent models multiple achitecture choices are possible. Solving both adding- and memory-problem with few weights is difficult. Gates work very well on the adding problem, while unitary approaches perform better on the memory problem. To combine both, complex state and gate activation functions and their interplay with unitary state transition matrices are studied. Bounded and unbounded state activation func-

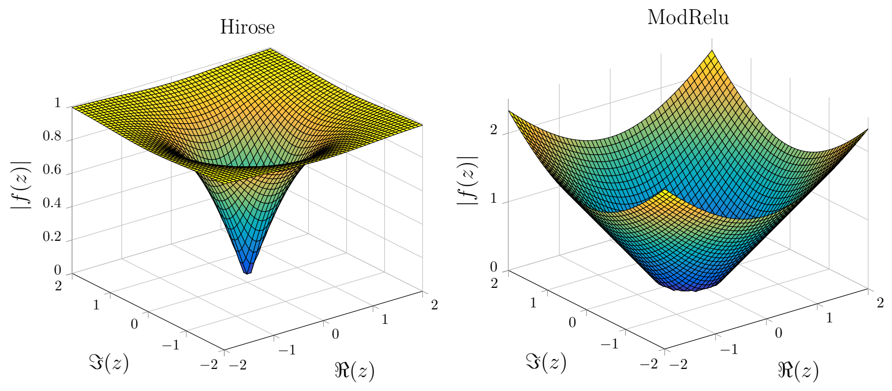


Figure 1.2: Surface plots showing the magnitude of the bounded Hirose and unbounded modReLU activations [WY18].

tions such as the Hirose and modRelu activations shown in figure 1.2 exist. The notion that unbounded RNN-state activation functions require unitary state transition functions is confirmed. A new gating function for complex RNN the *modSigmoid* is proposed.

In terms of memory capacity and convergence, cgRNNs combine the best of the two worlds. This architecture displays unitary recurrent neural network (uRNN) like stability on the memory problem and the noise resistance commonly observed for gated RNN on the adding problem.

1.3 TIME SERIES PREDICTION

Adjusting T changes the difficulty of adding and memory problems. This attribute makes the study and debugging of new recurrent cells easier, but measured real-world data does not share this property. Proper evaluation of our RNN architectures requires additional sequential data with different time-relations and structures. Therefore time series problems, including real-world measurements, are considered in all chapters of this text. The mono-variate chaotic Mackey-Glass series and electric power load data are studied in chapter 6. A challenging multivariate joint position [Ion+14] forecasting problem, as shown in figure 1.3 on the right, is considered in chapters 6 and 7. Accurate forecasts of human joint positions may enable more accurate human-robot collaboration in the future, as robots require an idea of where humans may move to in the future, to avoid collisions [Bru+20]. Figure 1.3 shows the power load data on the left, and human pose data on the right.

1.3.1 *Contributions to STFT and (complex) RNN integration [WGY20]*

RNN cells do not have to be evaluated at every time step. The frequency of evaluation is called the clock rate [Kou+14]. Instead of processing

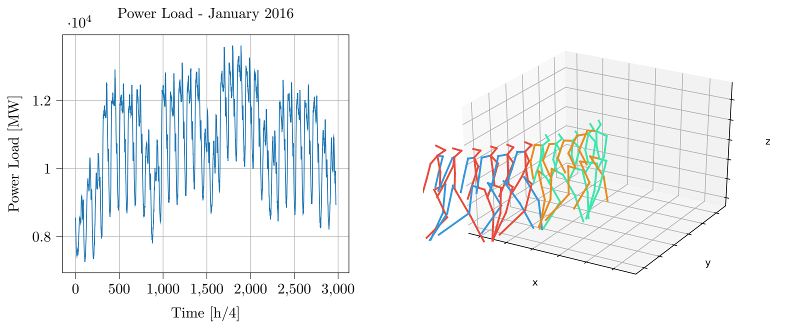


Figure 1.3: Mono- and multi-variate sequence data. Belgian power load in January 2016 (left) as well as a sequence snippet from the human 3.6m data set. The right image previously appeard in [WGY20].

individual data points, the STFT moves data windows consisting of multiple samples into the frequency domain. By processing the resulting complex coefficients per window, the clock rate is reduced significantly. Clock rate reductions make the network computationally more efficient by reducing the overall number of cell executions. Network stability is improved as well because unstable cells have fewer opportunities to blow up. Propagating gradients trough the STFT enables window function optimization. Additionally, in some cases, low pass filtering makes it possible to cut network parameters because discarded coefficients do not appear at input layers. Fourier coefficients are complex numbers. Ideally, complex-valued machine learning models should process these without taking them apart. Chapter 6 presents efficient processing of these complex Fourier coefficients using cgRNN.

1.4 VIDEO SEGMENT PREDICTION CHALLENGES

Videos or image sequences are perhaps the hardest to predict because standard methods tend to produce blurry predictions. As a precursor to real video, [SMS15] proposed the moving-MNIST data set to study this problem. The benchmark consists of MNIST-digits moving on a white background. Digit wall collisions are elastic and lead to a change of direction. The data set appears in chapter 8.

1.4.1 *Contribution: Segment prediction via phase modification [WYB20]*

Translation in space or a shift in time causes a phase-shift in the frequency-domain and vice-versa. Similarly, by comparing complex image representations, the transformation parameters can be estimated. A small RNN can then handle collisions in a predictor-corrector setup. Chapter 8 proposes a specialized RNN. The predictor-corrector approach is small, efficient, and does not produce blurry predictions.

1.5 THE CHALLENGE OF COMPRESSING NEURAL NETWORKS

While neural networks are growing in size and accuracy, new resource-constrained mobile and embedded applications are emerging. Many of the most recent models will not run on low resource devices. The sparsity inducing properties of the Fourier-, Walsh-Hadamard- and wavelet-transforms are known to be useful for data compression.

1.5.1 *Contribution: Wavelet optimization and network compression via the fast wavelet transform (FWT) [WLY20]*

Network compression reduces storage and computational footprints. Wavelets can compress neural networks by replacing fully connected layers. Instead of dense matrix multiplication, a combination of diagonal matrices, FWT-, inverse fast wavelet transform (iFWT)-, and permutation matrices can be substituted. Proper wavelets satisfy the anti-aliasing and perfect reconstruction conditions. Both have historically been part of the product-filter approach to wavelet design by hand [SN96]. Since both conditions are differentiable, two new cost functions can be constructed and added to the overall objective. The addition effectively turns the conditions into soft constraints. As discussed in sections 9.3.1 and 9.3.2 this works for convolutional neural network (CNN) and RNN architectures. In the RNN case, gate parameters can be reduced selectively. With this approach, chapter 9 studies the relative importance of the various RNN cell gates.

Replacing dense layers can significantly reduce network parameters, but no obvious initialization is available for the replacement. Therefore re-training of the entire network or fine-tuning of the replaced layer is necessary. Compression and quantization in wavelet space avoids the re-training problem and drastically reduces Huffman-coded file size. As discussed in section 9.5, this works well for shallow CNN, here wavelet quantization outperforms simple quantization.

1.6 OUTLINE OF THE THESIS STRUCTURE

This text is divided into a background and research part. The background part discusses the foundations of the signal processing and machine learning algorithms required in the research part. The background part includes descriptions of the fast Fourier and wavelet transforms, additionally fully connected, convolutional and recurrent network architectures such as Long Short Term Memory (LSTM) and Gated Recurrent Units (GRU) are explained. The illustrated explanation includes forward and backward passes for all architectures. Backward passes are carefully derived. Furthermore, cost functions, as well as standard optimization methods, are discussed.

In the research part, chapter 6 combines the short-time Fourier transform and complex-valued recurrent neural network cells [WGY20]. Chapter 7 is concerned with the design of a complex gated recurrent network cell [WY18].
In chapter 8 phase-based motion registration and image transformation are integrated into a recurrent cell for object-centered frame prediction [WYB20].
Chapter 9 presents new adaptive wavelets for network compression [WLY20].

1.7 OPEN RESEARCH APPROACH

Scientific discourse requires empirical evidence, for example, in the form of measurements and observations. Checking evidence makes it possible to distinguish fact from fiction. Good scientific practice, therefore, must center around enabling others to reproduce and check our evidence. As previously cited right after the title in the words of Karl Popper:

We, and those who share our attitude, will hope to make new discoveries; and we shall hope to be helped in this by a newly erected scientific system. Thus we shall take the greatest interest in the falsifying experiment. We shall hail it as a success, for it has opened up new vistas into a world of new experiences. And we shall hail it even if these new experiences should furnish us with new arguments against our own most recent theories.

— Karl Popper, Logik der Forschung [Pop35] translation [Pop59]

In machine learning research, new ideas are often evaluated on standard data-sets. On standardized data, the new algorithms and their evaluation take the form of source code. Unlike other sciences where manual labour in a lab is required, we can often repeat experiments automatically by running code on a computer. Most papers report evaluation results along with a short description of new algorithms or network structures, yet currently, most authors choose not to reveal their source code [Hut18]. Machine learning projects typically involve numerous training and network structure hyper-parameters as well as various preprocessing steps. Often it is simply impossible to describe everything in detail without violating the mandatory page limits imposed by virtually all major conferences. Consequently, the information present in the paper is not always sufficient to exactly reproduce the measurements described. Making reproduction and thereby confirmation or falsification as easy as possible not only means taking Popper seriously, it stands to reason that open code implementations enable accountability and will help future scientists drive progress in machine learning. This thesis comes with free and open-source code for every chapter. The code will make repetition and extension of the described algorithms and experiments easier.

Part I

BACKGROUND

2 NETWORK STRUCTURES

This chapter explores the machine learning architectures used in this thesis. Forward-, backward passes and cost will be discussed in detail. To fully understand the most important related work, a NumPy implementation of the methods and tools described in this chapter is available at `https://github.com/v0lta/NumPy-backprop`. Feedforward networks are evaluated on MNIST and recurrent structures on the adding and memory problems. Numpy does not come with automatic differentiation and automatic gradient computation tools. Everything discussed in this chapter has been reimplemented, using only Numpy functions from the ground up.

2.1 COST FUNCTIONS

Neural network optimization is gradient-based, and gradients are computed with respect to a performance measure or cost-function. A widespread cost function, in particular for prediction problems, is the mean squared error function. Given a network output **o** and a ground truth value **t**, the distance between the actual and the desired value can be measured as [Nie15]

$$C_{\text{mse}}(\mathbf{t}, \mathbf{o}) = \frac{1}{2} \sum_{k}^{n_o} \left(\mathbf{o}_k - \mathbf{t}_k\right)^2 = \frac{1}{2}\left(\mathbf{o}_k - \mathbf{t}_k\right)^T \cdot \left(\mathbf{o}_k - \mathbf{t}_k\right). \tag{2.1}$$

With $\cdot$ denoting the matrix product. Taking the derivative with respect to the output **o** leads to

$$\frac{\partial C_{\text{mse}}(\mathbf{y}, \mathbf{o})}{\partial \mathbf{o}} = \mathbf{o} - \mathbf{y} = \triangle_{\text{mse}}. \tag{2.2}$$

With n_o the number of output channels required for the problem under consideration. The difference above can be used as error input. Computing the error then is the first step of the backward pass.

For classification problems, the cross-entropy cost function is more common. Here the last activation function is chosen to be a sigmoid, to squash the output values into $[0, 1]$, which allows interpretation as probabilities. The cross entropy loss function is defined as [Nie15; Bis06]

$$C_{ce}(\mathbf{t}, \mathbf{o}) = -\sum_{k}^{n_o} (\mathbf{t}_k \ln \mathbf{o}_k) + (\mathbf{1} - \mathbf{t}_k) \ln(\mathbf{1} - \mathbf{o}_k). \tag{2.3}$$

If a sigmoidal activation function produced **o** the gradients can be computed using [Nie15; Bis06]

$$\frac{\partial C_{ce}}{\partial \mathbf{h}} = \sigma(\mathbf{h}) - \mathbf{y} = \triangle_{ce} \tag{2.4}$$

For the network output **h**. The following section will use $\triangle$ for cost function gradients and gradients from previous layers.

2.2 FULLY CONNECTED LAYERS AND NETWORKS

Feed-forward networks can be considered as useful pattern extractors in their own right. At the same time these layers form important building blocks in many more complex network architectures. Typically a simple fully connected layer is defined as

$$\overline{\mathbf{h}} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{2.5}$$

$$\mathbf{h} = f(\overline{\mathbf{h}}), \tag{2.6}$$

with $\mathbf{h} \in \mathbb{R}^{n_h,1}$ the output if the network ends after the layer or hidden representation if other layers follow. The layer weight matrix $\mathbf{W} \in \mathbb{R}^{n_h,n_x}$ and bias vector $\mathbf{b} \in \mathbb{R}^{n_h,1}$ contain the layer weights.

During the backward pass the gradients for the weight matrix and bias vector are computed using [Nie15]

$$\delta\mathbf{W} = [f'(\overline{\mathbf{h}}) \odot \triangle]\mathbf{x}^T, \qquad \delta\mathbf{b} = f'(\overline{\mathbf{h}}) \odot \triangle, \tag{2.7}$$

$$\delta\mathbf{x} = \mathbf{W}^T[f'(\overline{\mathbf{h}}) \odot \triangle]. \tag{2.8}$$

The expressions above follow from the application of the chain rule. $\triangle$ represents the input from the previous layer or the cost function backward pass if the fully connected layer was the last one. The δ is used to indicate the gradient with respect to the value following it [Gre+16]. $\delta\mathbf{W}$ and $\delta\mathbf{b}$ can be used to update parameters, while $\delta\mathbf{x}$ flows into subsequent layers. Element-wise product is denoted using $\odot$.

By combining cross-entropy cost and feedforward networks, it is possible to solve the MNIST digit recognition problem with accuracies above 90%. However, the dense matrix multiplies, use n_h^2 parameters per layer. It's possible to do better with fewer parameters using convolutional layers.

2.3 CONVOLUTIONAL NEURAL NETWORKS

Digit recognition using CNN architectures [LeC+98] was an early success, which helped to establish convolutional structures for image processing tasks [Bis06]. Neighbouring pixels are more likely to have the same colour than distant pixels or, in other words, are more likely to be correlated. Convolutional networks make use of this fact by extracting features locally [Bis06] using filters much smaller than its input. The network can learn about complex-features at higher layers through repeated convolution, which integrates local features [NYC16].

Using local receptive fields additionally enables weight sharing. A corner detector will be equally useful in the top left and bottom right of an input image. Avoiding to re-learn it at different locations means sharing the weights.

Coupling the parameters not only leads to a sparse transformation matrix compared to the fully connected case [GBC16], it also means that shifting the input shifts the features. The feature shift property is desirable because moving objects would disappear from view if it was impossible to compute identical activations elsewhere in an input.

Finally, subsampling operations are part of most convolutional neural networks [Bis06]. Subsampling eases the computational burden on higher levels considerably, making it possible to add additional filters, and thereby more flexibility.

Formally for an image I and a kernel $K \in \mathbf{R}^{k_w \times k_h}$, with k_w rows and k_h columns, two-dimensional convolution is defined as [GBC16],

$$S(i,j) = (\mathbf{I} * \mathbf{K})(i,j) = \sum_{m}^{k_w} \sum_{n}^{k_h} (\mathbf{I})(i+m, j+n)\mathbf{K}(m,n). \tag{2.9}$$

This means that for each position in the resulting feature image all $n \cdot m$ kernel elements which overlap with the pixel at i, j and those surrounding it must be summed up. In practice, networks consist of multiple stacked convolutions, which process batches of inputs. Figure 2.1 shows the computations which make up a CNN-layer. For efficiency n_b images are typically processed at once. n_c is used to denote the channel count. Convolution layers require an input tensor $\mathbf{I} \in \mathbb{R}^{n_b \times n_c \times n_w \times n_h}$, which is shown in the left, and a kernel tensor $\mathbf{K} \in \mathbb{R}^{k_o \times k_i \times k_w \times k_h}$, which is not shown. Additionally to the kernel-width k_w and height k_h, input k_i and output channel numbers k_o appear here. The number of image input channels n_c must be equal to the input kernel dimension k_i. The kernel tensor stacks k_i filters for each output dimension k_o. Figure 2.1 depicts the computational steps in a convolution layer from left to right. The convolution operation $*$ as defined in equation 2.9 must be evaluated $k_i \cdot k_o$ times. Once for each input dimension and output dimension in the convolution kernel tensor. Summations $+$ along the input channel dimension follow the

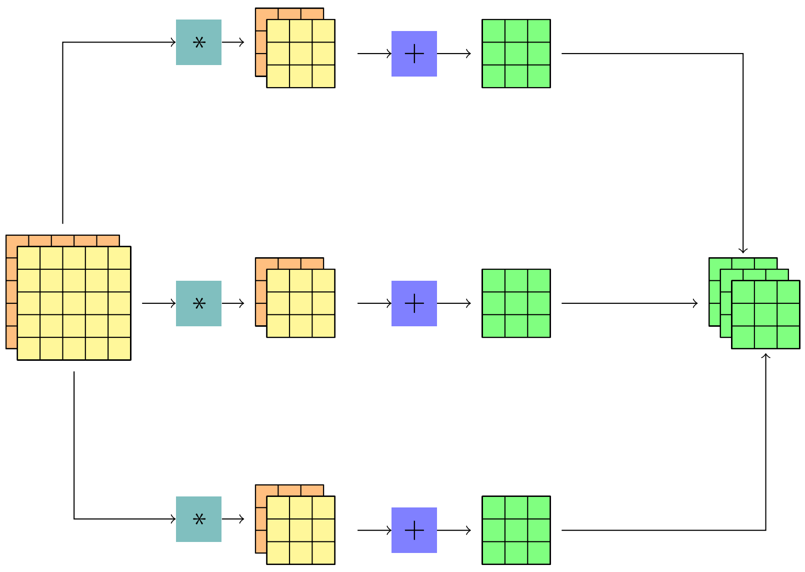


Figure 2.1: Visualization of multi-channel convolution. Convolution computation using a 3x2x3x3 kernel on a 2x5x5 image is shown. The image is read from left to right, starting with the two-channel five by five image on the very left. In a first step, the two input kernels are convolved with the input channels three times, as indicated by the convolution blocks $*$. Without padding, this operation leads to the 3x3 results. Moving towards the right, we add $+$ the two channels for each of the three resulting tensors. Finally, everything is stacked, which leads to the final result. [DV16] inspired this illustration, it contains an excellent introduction to the topic.

convolutions, leaving k_o features. Finally, these k_o features are stacked, leading to the convolution layer output.

2.3.1 *Convolution as Matrix Mutliplication*

The convolution operation lies not only at the heart of CNNs is also is the key operation powering the FWT and iFWT. Writing it as a matrix operator allows the definition of backpropagation equations in section 2.3.2.

Since convolution is ultimately a linear operation it can be written as matrix multiplication $\mathbf{Ax} = \mathbf{b}$. Using doubly block circulants $\mathbf{C}_b$ [GBC16] puts convolution structures into matrices. The numerical values depend on the kernel, but the structure is a consequence of the convolution operation. Unfortunately, the resulting block circulants are sparse and require dedicated matrix multiplication algorithms for efficiency.

A more straightforward solution is to introduce the convolution structure into the vector instead of the matrix. This approach uses image-to-column and column to image functions. These seek to write convolution as $\mathbf{K}_f\mathbf{C}_I$, with $\mathbf{K} \in \mathbb{R}^{k_o,k_k}$ the flattened kernel matrix and $\mathbf{C}_I \in \mathbb{R}^{n_k,n_v}$ the image matrix, which is structured such that multiplication with the flat kernel matrix $\mathbf{K}_f$ results in convolution.

The structure of the kernel matrix is straightforward with k_o the number of output channels and k_k the product of the remaining kernel tensor dimensions $k_i \cdot k_h \cdot k_w$.

The image matrix now lists the image pixels from each patch along channels, such that matrix multiplication leads to the convolution layer operation shown in figure 2.1. The image matrices second dimension therefore must be the product of the output width and height and the batch size $n_v = n_b \cdot o_h \cdot o_w$ (See [DV16] for an excellent description on how to compute o_h and o_w).

Using the restructured input $\mathbf{v}_I$ and the flattened kernel $\mathbf{K}_f$ the forward pass for a convolutional layer turns into

$$\bar{\mathbf{h}} = \mathbf{K}_f\mathbf{v}_I + \mathbf{b} \tag{2.10}$$

$$\mathbf{h}_f = f(\bar{\mathbf{h}}). \tag{2.11}$$

After matrix multiplication the proper shape of the output must be restored. A good way to do this is a reshape.

2.3.2 *Backpropagation*

Using the matrix multiplication form makes it possible to draw on the backward pass for linear layers as described in equations 2.7 and 2.8.

$$\delta\mathbf{K}_f = [f'(\bar{\mathbf{h}}) \odot \triangle]_f \mathbf{v}_I^T, \qquad \delta\mathbf{b} = f'(\bar{\mathbf{h}}) \odot \triangle, \tag{2.12}$$

$$\delta\mathbf{x} = \left(\mathbf{K}_f^T [f'(\bar{\mathbf{h}}) \odot \triangle]_f\right)_{I^{-1}}. \tag{2.13}$$

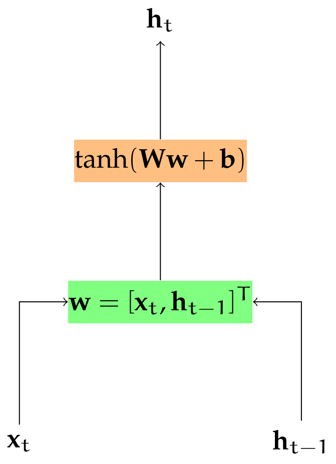

$\mathbf{h}_t$
$\tanh(\mathbf{W}\mathbf{w} + \mathbf{b})$
$\mathbf{w} = [\mathbf{x}_t, \mathbf{h}_{t-1}]^T$
$\mathbf{x}_t$
$\mathbf{h}_{t-1}$

Figure 2.2: Visualization of a simple recurrent cell. Output $\mathbf{y}_t$ and cell state $\mathbf{c}_t$ at time t depend on the previous state $\mathbf{c}_{t-1}$ and the current input $\mathbf{x}_t$. Figure similar to[Wol17].

In the equations above the subscript, f denotes flattening of the channel dimensions, while I and I^{-1} indicate the image to column and column to image operations.

2.4 RECURRENT NEURAL NETWORKS

Typically RNN structures are chosen to solve sequence modelling tasks. Words are sequences of characters. Language models, for example, are used to predict letters or words based on the previously observed context. Alternatively, in the case of power load prediction, the network has to estimate the load for tomorrow at noon today, given the consumption over the last couple of days.

2.4.1 *Simple-RNN*

A simple solution is to add a state to the network and feed this state recurrently back into the network [Elm90]. Such an approach would suggest a definition for the forward pass like,

$$\overline{\mathbf{h}_t} = \mathbf{W}_h \mathbf{h}_t + \mathbf{W}_x \mathbf{x}_t + \mathbf{b}, \tag{2.14}$$

$$\mathbf{h}_{t+1} = f(\overline{\mathbf{h}_t}). \tag{2.15}$$

The network consists out of the recurrent state weights $\mathbf{W}_c \in \mathbb{R}^{n_c \times n_c}$, the input weights $\mathbf{W}_x \in \mathbb{R}^{n_c \times n_x}$, and the bias term $\mathbf{b} \in \mathbb{R}^{n_c \times 1}$. The state size n_c determines the capacity of the network, while n_x denotes the input dimension. The hyperbolic tangent function is often chosen to be the activation function f.

This approach is visualized in figure 2.2. Output values depend on the current input, the state representation from the previous time step, the weight matrix, and bias. The dependence on the previous state

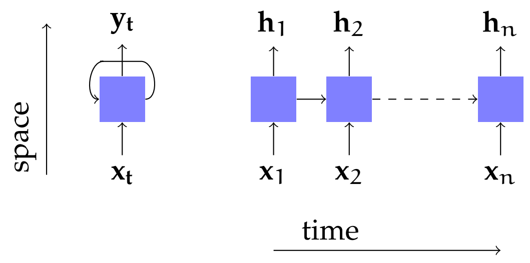


Figure 2.3: The rolled (left) cell can be unrolled (right) by considering all inputs it saw during the current gradient computation iteration. Figure shown as found in [Wol17].

creates a cycle in the graph. Since backpropagation cannot handle cycles, it is common practice to work with unrolled representation of recurrent networks as shown in figure 2.3. By considering all time steps separately cycles are eliminated. Analogous to standard backward sweeps, the backward pass through a recurrent network rests on the chain rule. Weights are shared over time, therefore all time steps have an impact on the gradients. In order to take all time steps into account and compute the gradients for the unrolled recurrent network shown in figure 2.3, a recurrent term must be added to the gradient computation formulae. During the backward pass using the chain rule we now obtain,

$$\delta\overline{\mathbf{h}_t} = f'(\overline{\mathbf{h}_t}) \odot (\triangle_t + \delta\mathbf{h}_{t+1}), \tag{2.16}$$

$$\delta\mathbf{h}_t = \mathbf{W}_h^T\delta\overline{\mathbf{h}_t}, \qquad \delta\mathbf{x}_t = \mathbf{W}_x^T\delta\overline{\mathbf{h}_t}, \tag{2.17}$$

$$\delta\mathbf{W}_x = \sum_{t=0}^{L}\delta\overline{\mathbf{h}_t}\mathbf{x}^T, \qquad \delta\mathbf{W}_h = \sum_{t=0}^{L}\delta\overline{\mathbf{h}_t}\,\mathbf{h}_t^T, \tag{2.18}$$

$$\delta\mathbf{b} = \sum_{t=0}^{L}\delta\overline{\mathbf{h}_t}. \tag{2.19}$$

per time step t. At $t = L + 1$ which is one step more than the total length L, the recurrent delta $\delta\mathbf{h}_{t+1}$ does not exist and is set to zero. This approach is also referred to as back-propagation through time.

While conceptually simple, this approach is unstable in practice. It suffers from exploding and vanishing gradients [Hoc91; Hoc+01; BSF94; GBC16]. In a simplified linear case, the network dynamics would depend on the largest eigenvalue [GBC16]. Imagine, for example, the recurrent weight matrix **W** had an eigenvalue larger than one. In this case, since the state is multiplied with the recurrent weight matrix once per time step, the norm or length of the state vector will continue to grow over time. Similarly, if the largest eigenvalue was smaller than one, the state's norm must continue to shrink. The linear case suggests that an orthogonal or unitary state matrix with eigenvalues equal to one are an interesting choice for recurrent weight matrices. Assuming no error input for all but the last time step $\triangle_t = 0$

for all $t \neq T$, looking at the backward pass for the the network state in more detail [ASB16; WY18] results in,

$$\frac{\partial C}{\partial \mathbf{h}_t} = \frac{\partial C}{\partial \mathbf{h}_T} \frac{\mathbf{h}_T}{\mathbf{h}_t} \tag{2.20}$$

$$= \frac{\partial C}{\partial \mathbf{h}_T} \prod_{k=T}^{T-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} \tag{2.21}$$

$$= \frac{\partial C}{\partial \mathbf{h}_T} \prod_{k=T}^{T-1} \mathbf{D}_{k+1} \mathbf{W}_h^T \tag{2.22}$$

$$= \triangle_T \prod_{k=T}^{T-1} \mathbf{D}_{k+1} \mathbf{W}_h^T. \tag{2.23}$$

With $\mathbf{D}_{k+1} = \text{diag}(f'(\overline{\mathbf{h}_k}))$, here the diagonal matrix is just another way to express the element wise Hadamard product. Chossing the two-norm we have for any combinations of matrices $\mathbf{A}, \mathbf{B}$ and vectors $\mathbf{v}$, $\|\mathbf{A}\mathbf{v}\| \leqslant \|\mathbf{A}\| \; \|\mathbf{v}\|$, as well as, $\|\mathbf{A}\mathbf{B}\| \leqslant \|\mathbf{A}\| \; \|\mathbf{B}\|$. Application to the RNN-state gradient leads to [ASB16; WY18],

$$\|\frac{\partial C}{\partial \mathbf{h}_t}\| = \|\frac{\partial C}{\partial \mathbf{h}_T} \prod_{k=T}^{T-1} \mathbf{D}_{k+1} \mathbf{W}_h^T\| \tag{2.24}$$

$$\leqslant \|\frac{\partial C}{\partial \mathbf{h}_T}\| \prod_{k=T}^{T-1} \|\mathbf{D}_{k+1} \mathbf{W}_h^T\|. \tag{2.25}$$

If the state transition matrix is orthogonal or unitary $\mathbf{W}_h$, it will be norm preserving, consequently we will have $\|\mathbf{W}_h = 1\|$ and are left with a product of diagnoal activation matrix norms$\|\mathbf{D}_k\|$. The ReLU's derivative is 1 if the forward pass was active. Since diagonal matrices such as $\mathbf{D}$ carry their eigenvalues on the diagonal, we can show [ASB16]

$$\|\frac{\partial C}{\partial \mathbf{h}_t}\| \leqslant \|\frac{\partial C}{\partial \mathbf{h}_T}\| \prod_{k=T}^{T-1} \|\mathbf{D}_{k+1} \mathbf{W}_h^T\| \tag{2.26}$$

$$= \|\frac{\partial C}{\partial \mathbf{h}_T}\| \prod_{k=T}^{T-1} \|\mathbf{D}_{k+1}\| \tag{2.27}$$

$$= \|\frac{\partial C}{\partial \mathbf{h}_T}\| = \|\triangle_T\|. \tag{2.28}$$

Which guarantess stability unless all activations are zero. In the non-linear case the eigenvalues and activation function are therefore connected [WY18]. Chapter 7 explores this connection further.

2.4.2 *Long Short Term Memory*

The most popular remedy for the vanishing gradient problem is the memory management that comes with the gates introduced in the

LSTM cell [HS97b]. This approach is widely considered an algorithm that has stood the test of time [Hut20] and has since found applications in handwriting and speech recognition, machine translation, image captioning, and more [GBC16]. It utilizes three gates to manage the cell state. Not unlike a differentiable memory chip [Gra12] the LSTM memory cell has a memory size n_h and three gates which govern the changes to the cell state. All gate output match the cell's size and are evaluated using a sigmoidal activation function σ therefore all gates are real vectors in $\mathbb{R}^{n_h} \in [0, 1]$. The gate and state computation equations for an LSTM cell are defined as [HS97b; Gre+16]

$$\begin{aligned}
\overline{\mathbf{z}_t} &= \mathbf{W}_z\mathbf{x}_t + \mathbf{R}_z\mathbf{h}_{t-1} + \mathbf{b}_z, && (2.29)\\
\mathbf{z}_t &= \tanh(\overline{\mathbf{z}_t}), && (2.30)\\
\overline{\mathbf{i}_t} &= \mathbf{W}_i\mathbf{x}_t + \mathbf{R}_i\mathbf{h}_{t-1} + \mathbf{p}_i \odot \mathbf{c}_{t-1} + \mathbf{b}_i, && (2.31)\\
\mathbf{i}_t &= \sigma(\overline{\mathbf{i}_t}), && (2.32)\\
\overline{\mathbf{f}_t} &= \mathbf{W}_f\mathbf{x}_t + \mathbf{R}_f\mathbf{h}_{t-1} + \mathbf{p}_f \odot \mathbf{c}_{t-1} + \mathbf{b}_f, && (2.33)\\
\mathbf{f}_t &= \sigma(\overline{\mathbf{f}_t}), && (2.34)\\
\mathbf{c}_t &= \mathbf{z}_t \odot \mathbf{i}_t + \mathbf{c}_{t-1} \odot \mathbf{f}_t, && (2.35)\\
\overline{\mathbf{o}_t} &= \mathbf{W}_o\mathbf{x}_t + \mathbf{R}_o\mathbf{h}_{t-1} + \mathbf{p}_o \odot \mathbf{c}_t + \mathbf{b}_o, && (2.36)\\
\mathbf{o}_t &= \sigma(\overline{\mathbf{o}_t}), && (2.37)\\
\mathbf{h}_t &= \tanh(\mathbf{c}_t) \odot \mathbf{o}_t. && (2.38)
\end{aligned}$$

The potential new state values $\mathbf{z}_t$ are called block input. The vector $\mathbf{i}$ is called the input gate. The forget gate is labelled $\mathbf{f}$ and finally $\mathbf{o}$ denotes the output gate. Peephole weights are denoted using $\mathbf{p} \in \mathcal{R}^{n_h}$, $\mathbf{W} \in \mathcal{R}^{n_i \times n_h}$ denotes input, while $\mathbf{R} \in \mathcal{R}^{n_o \times n_h}$ are the recurrent output matrices. The element-wise or Hadamard product indicated by the $\odot$ symbol. Figure 2.4 shows a schematic of the LSTM cell equations. The line on the left running from bottom to top is the state line. All cell parts which modify the cell state eventually connect to it. Moving along the line, we first encounter the forget gate. The forget gate decides which state entries to store and which to delete. Recall that the sigmoid function produces output values within $[0, 1]$ and that $\mathbf{f}$ has just as many entries as $\mathbf{c}$. A value of zero, therefore, means that the corresponding state entry is forgotten. Similarly, a value of one means that it will be kept. Following the state line, just after the forget gate, the input gate governs the addition of new entries to the memory. These new values are chosen through the input gate through $\mathbf{i} \odot \mathbf{z}_t$. Since the input and forget gate share the same activation function, they behave similarly. The input gate picks candidate values from the block input by producing ones. Potential candidates are blocked if the corresponding entry is zero. Finally, the output gate chooses which values from the state $\mathbf{c}$ will turn into output values $\mathbf{h}$. Mechanically its function is identical to the two other gates. Block input and output values are run through a hyperbolic tangent activation function. State and output values are therefore within $[-1, 1]$.

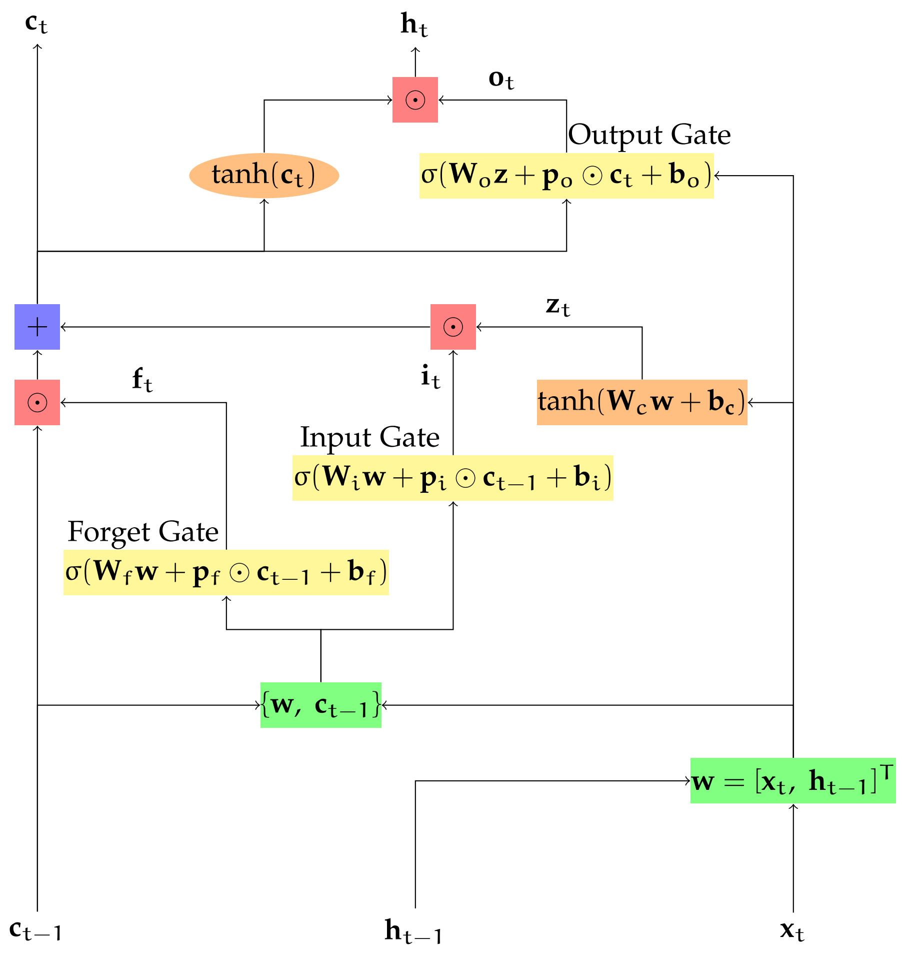

$\mathbf{c}_t$
$\mathbf{h}_t$
$\mathbf{o}_t$
Output Gate
$\tanh(\mathbf{c}_t)$
$\sigma(\mathbf{W}_o\mathbf{z} + \mathbf{p}_o \odot \mathbf{c}_t + \mathbf{b}_o)$
$\mathbf{z}_t$
$\mathbf{f}_t$
$\mathbf{i}_t$
$\tanh(\mathbf{W}_c\mathbf{w} + \mathbf{b}_c)$
Input Gate
$\sigma(\mathbf{W}_i\mathbf{w} + \mathbf{p}_i \odot \mathbf{c}_{t-1} + \mathbf{b}_i)$
Forget Gate
$\sigma(\mathbf{W}_f\mathbf{w} + \mathbf{p}_f \odot \mathbf{c}_{t-1} + \mathbf{b}_f)$
$\{\mathbf{w},\ \mathbf{c}_{t-1}\}$
$\mathbf{w} = [\mathbf{x}_t,\ \mathbf{h}_{t-1}]^T$
$\mathbf{c}_{t-1}$
$\mathbf{h}_{t-1}$
$\mathbf{x}_t$

Figure 2.4: Visualization of a long short term memory cell. Green boxes contain concatenation operations. The state line on the very left records all changes to the cell's memory. All gate equations appear in yellow. The gates use sigmoid activations. The sigmoid activation produces values within zero and one. A zero output means the corresponding value is blocked. One means it is allowed to pass through. State candidate and output values are run through the tanh function shown in orange. This ensures the cell's memory contents and output values are within $[-1, 1]$. This Figure is similar to [Wol17].

2.4.2.1 *Backpropagation through time*

In the following equations an overline is used to denote the preactivation input to an equation $\bar{\cdot}$. For example in the case of the input gate $\bar{\mathbf{i}} = \mathbf{W}_i \mathbf{x}_t + \mathbf{R}_i \mathbf{h}_{t-1} + \mathbf{p}_i \odot \mathbf{c}_{t-1} + \mathbf{b}_i$, or for the block input $\bar{\mathbf{z}} = \mathbf{W}_z \mathbf{x}_t + \mathbf{R}_z \mathbf{h}_{t-1} + \mathbf{b}_z$. ^{T} denotes the transpose. The deltas for the LSTM block are given by [Gre+16];

$$
\begin{aligned}
\delta \mathbf{h}_t &= \triangle_t + \mathbf{R}_z^T \delta \overline{\mathbf{z}_{t+1}} + \mathbf{R}_i^T \delta \overline{\mathbf{i}_{t+1}} + \mathbf{R}_f^T \delta \overline{\mathbf{f}_{t+1}} + \mathbf{R}_o^T \delta \overline{\mathbf{o}_{t+1}}, && (2.39)\\
\delta \overline{\mathbf{o}_t} &= \delta \mathbf{h}_t \odot \tanh(\mathbf{c}_t) \odot \sigma'(\overline{\mathbf{o}_t}), && (2.40)\\
\delta \mathbf{c}_t &= \delta \mathbf{h}_t \odot \mathbf{o}_t \odot \tanh'(\mathbf{c}_t) + \mathbf{p}_o \odot \delta \overline{\mathbf{o}_t} + \mathbf{p}_i \odot \delta \overline{\mathbf{i}_{t+1}} \\
&\quad + \mathbf{p}_f \odot \delta \overline{\mathbf{f}_{t+1}} + \delta \mathbf{c}_{t+1} \odot \mathbf{f}_{t+1}, && (2.41)\\
\delta \overline{\mathbf{f}_t} &= \delta \mathbf{c}_t \odot \mathbf{c}_{t-1} \odot \sigma'(\overline{\mathbf{f}_t}), && (2.42)\\
\delta \overline{\mathbf{i}_t} &= \delta \mathbf{c}_t \odot \mathbf{z}_t \odot \sigma'(\overline{\mathbf{i}_t}), && (2.43)\\
\delta \overline{\mathbf{z}_t} &= \delta \mathbf{c}_t \odot \mathbf{i}_t \odot \tanh'(\overline{\mathbf{z}_t}). && (2.44)
\end{aligned}
$$

The error flowing into the cell from a previous step is denoted by $\triangle_t$. If the cell has been placed directly below the cost function $\triangle_t = \frac{\partial C}{\partial \mathbf{h}_t}$. Equation 2.41 describes the error flow backward in time. In comparison to the simple RNN cell design it is not multiplied by a leading matrix and therefore not rescaled by it's largest eigenvalue. The first and the last summands in equation 2.41 are of particular importance. Here the output $\mathbf{o}_t$ and forget $\mathbf{f}_t$ gates regulate gradient flow back in time. If the output gate is closed, the error flow within the cell is shielded from the outside. The last term contains the forget gate and the state gradients from the previous step in time, when the forget gates removes a state entry, it also ends the flow of error values over time. Cell weight gradients can be computed using the cell-deltas [Gre+16];

$$
\begin{aligned}
\delta \mathbf{W}_\star &= \sum_{t=0}^{L} \delta \star_t \mathbf{x}_t^T, & \delta \mathbf{p}_i &= \sum_{t=0}^{L} \mathbf{c}_t \odot \delta \overline{\mathbf{i}_{t+1}}, && (2.45)\\
\delta \mathbf{R}_\star &= \sum_{t=0}^{L} \delta \star_{t+1} \mathbf{h}_t^T, & \delta \mathbf{p}_f &= \sum_{t=0}^{L} \mathbf{c}_t \odot \delta \overline{\mathbf{f}_{t+1}}, && (2.46)\\
\delta \mathbf{b}_\star &= \sum_{t=0}^{L} \delta \star_t, & \delta \mathbf{p}_o &= \sum_{t=0}^{L} \mathbf{c}_t \odot \delta \overline{\mathbf{o}_t}. && (2.47)
\end{aligned}
$$

With $\star \in \{\bar{\mathbf{z}}, \bar{\mathbf{i}}, \bar{\mathbf{f}}, \bar{\mathbf{o}}\}$. Backpropagation through time sums up the individual gradient contributions found at each time step. Until the total sequence length L. In multi-layer architectures the error to a lower layer is given by [Gre+16]

$$
\delta \mathbf{x}_t = \mathbf{W}_z^T \delta \overline{\mathbf{z}_t} + \mathbf{W}_i^T \delta \overline{\mathbf{i}_t} + \mathbf{W}_f^T \delta \overline{\mathbf{f}_t} + \mathbf{W}_o^T \delta \overline{\mathbf{o}_t}. \qquad (2.48)
$$

The delta $\delta \mathbf{x}_t$ above can be propagated into lower layers. Multiple LSTM variants exist. A popular version couples the input and forget gates. Since it is often required to delete a value which is going to be replaced later instead of learning the input gate, it is set to $\mathbf{f}_t = \mathbf{1} - \mathbf{i}_t$ [Gre+16]. Other LSTM variants do not include the peephole connections, which feed the state into the gates using the $\mathbf{p}$ weights. During the forward pass, this forces the gates to work without knowledge of the cell state, however, during the backward pass, this variant greatly simplifies equation 2.41, giving control of the error flow entirely to the output and forget gate.

Early versions of LSTM did not include the peephole connections and forget gate [Gra12], which were added later. The forget gate was added to give the cell the means to reset its state. The peephole connections allow the cell to look at the memory content while deciding should be stored, erased, or shared.

2.4.2.2 *Derivation of the LSTM backward pass*

To derive the LSTM backward pass equations from section 2.4.2.1 the chain rule must be applied to all equations in the forward pass. Here the rules as described in [LKJ15] are followed, starting at the output $\mathbf{h}$ and working backward. Some equations contribute gradient summands to a variable that appeared elsewhere previously. $+=$ signs mark these cases. We start at the cell output $\mathbf{h}_t = \tanh(\mathbf{c}_t) \odot \mathbf{o}_t$, and obtain

$$\delta\mathbf{o}_t = \tanh(\mathbf{c}_t) \odot \delta\mathbf{h}_t, \tag{2.49}$$

$$\delta\mathbf{c}_t = \tanh'(\mathbf{c}_t) \odot \mathbf{o}_t \odot \delta\mathbf{h}_t. \tag{2.50}$$

The output $\mathbf{h}_t$ connects to the output gate and the cell state. Moving on to the output gate $\mathbf{o}_t = \sigma(\overline{\mathbf{o}_t})$ yields

$$\delta\overline{\mathbf{o}_t} = \sigma'(\overline{\mathbf{o}_t}) \odot \delta\mathbf{o}_t. \tag{2.51}$$

Next we move through the activation function into the gate $\overline{\mathbf{o}_t} = \mathbf{W}_o\mathbf{x}_t + \mathbf{R}_o\mathbf{y}_{t-1} + \mathbf{p}_o \odot \mathbf{c}_t + \mathbf{b}_o$ and obtain

$$\delta\mathbf{x}_t = \mathbf{W}_o^T \delta\overline{\mathbf{o}_t}, \tag{2.52}$$

$$\delta\mathbf{h}_{t-1} = \mathbf{R}_o^T \delta\overline{\mathbf{o}_t}, \tag{2.53}$$

$$\delta\mathbf{c}_t = \mathbf{c}_t \odot \delta\overline{\mathbf{o}_t}, \tag{2.54}$$

$$\delta\mathbf{b}_o = \delta\overline{\mathbf{o}_t}. \tag{2.55}$$

From y we have now exhausted the backward path in the o direction and continue to move into the cell state $\mathbf{c}_t = \mathbf{z}_t \odot \mathbf{i}_t + \mathbf{c}_{t-1} \odot \mathbf{f}_t$ which leads to

$$\delta\mathbf{z}_t = \mathbf{i}_t \odot \delta\mathbf{c}_t, \tag{2.56}$$

$$\delta\mathbf{i}_t = \mathbf{z}_t \odot \delta\mathbf{c}_t, \tag{2.57}$$

$$\delta\mathbf{c}_{t-1} = \mathbf{f}_t \odot \delta\mathbf{c}_t, \tag{2.58}$$

$$\delta\mathbf{f}_t = \mathbf{c}_{t-1} \odot \delta\mathbf{c}_t. \tag{2.59}$$

The cell state equation depends on input 2.31 and forget 2.33 gate as well as the state-candidate value 2.29 equation. Starting with the forget gate $\mathbf{f}_t = \sigma(\overline{\mathbf{f}_t})$ results in

$$\delta\overline{\mathbf{f}_t} = \sigma'(\overline{\mathbf{f}_t})\delta\mathbf{f}_t. \tag{2.60}$$

The continuation of the backprop process into $\overline{\mathbf{f}_t} = \mathbf{W}_f\mathbf{x}_t + \mathbf{R}_f\mathbf{h}_{t-1} + \mathbf{p}_f \odot \mathbf{c}_{t-1} + \mathbf{b}_f$ contributes,

$$\delta\mathbf{x}_t + = \mathbf{W}_f^T\delta\bar{\mathbf{f}}_t, \tag{2.61}$$
$$\delta\mathbf{h}_{t-1} + = \mathbf{R}_f^T\delta\bar{\mathbf{f}}_t, \tag{2.62}$$
$$\delta\mathbf{c}_{t-1} + = \mathbf{p}_f \odot \delta\overline{\mathbf{f}_t}, \tag{2.63}$$
$$\delta\mathbf{b}_f = \delta\overline{\mathbf{f}_t}. \tag{2.64}$$

At this point the input gate and state candidates remain. Starting with the input gate $\mathbf{i}_t = \sigma(\overline{\mathbf{i}_t})$ leads to

$$\delta\overline{\mathbf{i}_t} = \sigma'(\overline{\mathbf{i}_t}) \odot \delta\mathbf{i}_t, \tag{2.65}$$

and $\overline{\mathbf{i}_t} = \mathbf{W}_i\mathbf{x}_t + \mathbf{R}_i\mathbf{h}_{t-1} + \mathbf{p}_i \odot \mathbf{c}_{t-1} + \mathbf{b}_i$ leads to,

$$\delta\mathbf{x}_t + = \mathbf{W}_i^T\delta\overline{\mathbf{i}_t}, \tag{2.66}$$
$$\delta\mathbf{h}_{t-1} + = \mathbf{R}_i^T\delta\overline{\mathbf{i}_t}, \tag{2.67}$$
$$\delta\mathbf{c}_{t-1} + = \mathbf{p}_i \odot \delta\overline{\mathbf{i}_t}, \tag{2.68}$$
$$\delta\mathbf{b}_i = \delta\overline{\mathbf{i}_t}. \tag{2.69}$$

Finally only the state candidate computation $\overline{\mathbf{z}_t} = \mathbf{W}_z\mathbf{x}_t + \mathbf{R}_z\mathbf{h}_{t-1} + \mathbf{b}_z$ remains it contributes

$$\delta\mathbf{x}_t + = \delta\mathbf{W}_z^T\overline{\mathbf{z}_t}, \tag{2.70}$$
$$\delta\mathbf{h}_{t-1} + = \mathbf{R}_z^T\delta\overline{\mathbf{z}_t}, \tag{2.71}$$
$$\delta\mathbf{b}_z = \delta\overline{\mathbf{z}_t}. \tag{2.72}$$

The equations discussed in section 2.4.2.1 are the result of combining the individual contributions derived above. The most popular LSTM [HS97b] variant is the GRU[Cho+14], the main subject of the upcoming section.

2.4.3 *Gated Recurrent Units*

Introduced in [Cho+14] in the context of machine translation, GRU use coupled input/forget gates, no output gate, and an additional reset gate. For some problems removing the forget gate does not have a large impact on performance [Gre+16], which perhaps motivates its absence in GRU. Input and forget gates coupling seems reasonable from a conceptional point of view, since typically whenever **i** saturates at one the corresponding value for $\mathbf{f}$ should be zero to make room for a new value. The GRU equations are defined as [Cho+14],

$$\overline{\mathbf{r}_t} = \mathbf{W}_r \mathbf{h}_{t-1} + \mathbf{V}_r \mathbf{x}_t + \mathbf{b}_r, \tag{2.73}$$
$$\mathbf{r}_t = \sigma(\overline{\mathbf{r}_t}), \tag{2.74}$$
$$\overline{\mathbf{u}_t} = \mathbf{W}_u \mathbf{h}_{t-1} + \mathbf{V}_u \mathbf{x}_t + \mathbf{b}_u, \tag{2.75}$$
$$\mathbf{u}_t = \sigma(\overline{\mathbf{u}_t}), \tag{2.76}$$
$$\overline{\mathbf{h}_{t-1}} = \mathbf{r}_t \odot \mathbf{h}_{t-1}, \tag{2.77}$$
$$\overline{\mathbf{z}_t} = \mathbf{W}\overline{\mathbf{h}_{t-1}} + \mathbf{V}\mathbf{x}_t + \mathbf{b}, \tag{2.78}$$
$$\mathbf{z}_t = \tanh(\overline{\mathbf{z}_t}), \tag{2.79}$$
$$\mathbf{h}_t = \mathbf{u}_t \odot \mathbf{z}_t + (1 - \mathbf{u}_t) \odot \mathbf{h}_{t-1}. \tag{2.80}$$

Again the Hadamard or element-wise product is written as $\odot$. Throughout the GRU-equations $\mathbf{h}_t \in \mathbb{R}^{n_h}$ denotes the cell state and output at time t. The block input is called $\mathbf{z}_t \in \mathbb{R}^{n_h}$ the same convention appeared in the LSTM equations. The reset $\mathbf{r} \in \mathbb{R}^{n_h}$ and update gates $\mathbf{u} \in \mathbb{R}^{n_h}$ take care of memory management. As in the LSTM case, $\mathbf{W} \in \mathbb{R}^{n_i \times n_h}$ denotes input matrices, $\mathbf{V} \in \mathbb{R}^{n_h \times n_h}$ is used for recurrent weight matrices. Figure 2.5 visualizes the GRU architecture. This visualization again shows a state on the left. The first connection comes from the update gate, which replaces the forget and input gate. Since the update gate uses a sigmoid σ activation function, the first Hadamard-product on the state line will erase products if $\mathbf{u}_t$ contains a zero. Thanks to the coupling $\mathbf{1} - \mathbf{u}_t$, erased state entries will be replaced by the corresponding value from the block input $\mathbf{z}_t$. In addition to the update gate, GRUs come with the reset gate, which enables the cell to delete values from the state just before the block input is computed. Similar to a forget gate, this gate re-introduces an independent deletion mechanism.

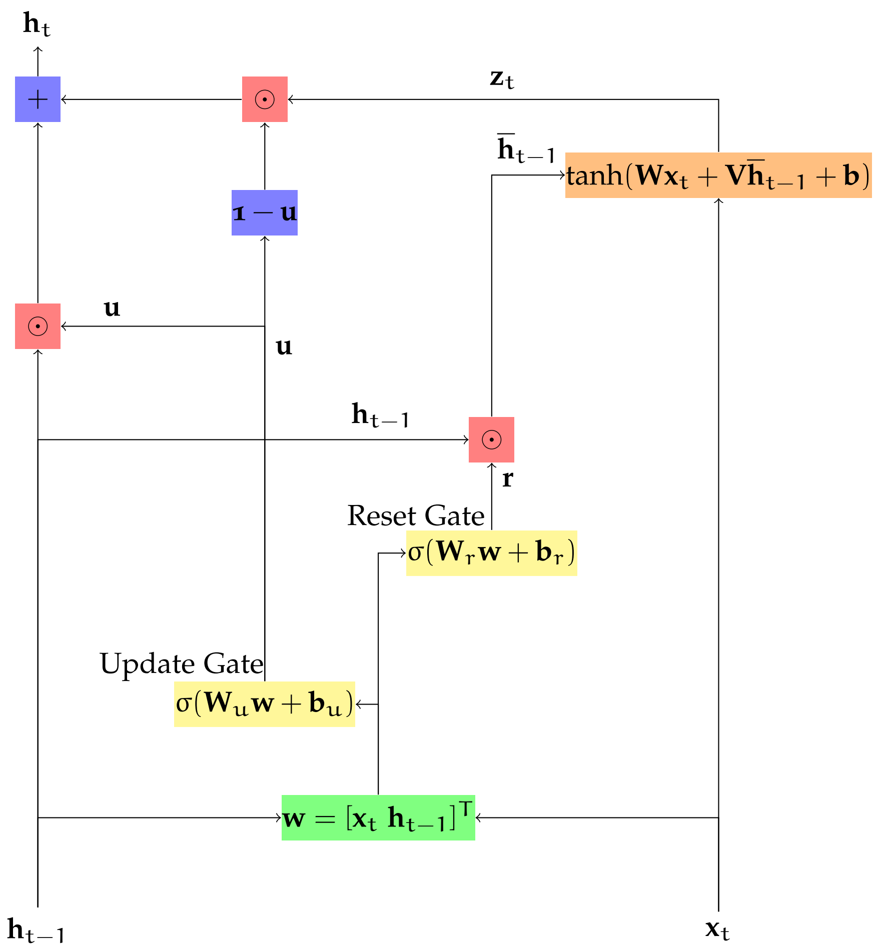
2.4.3.1 *GRU Backward pass*

Again following the notation in [Gre+16], deltas for the GRU block are given by

$$\delta\mathbf{h}_t = \triangle_t + (1 - \mathbf{u}_{t+1}) \odot \delta\mathbf{h}_{t+1} + \mathbf{r}_{t+1} \odot \mathbf{W}^T \delta\overline{\mathbf{z}_{t+1}} + \mathbf{W}_u^T \delta\overline{\mathbf{u}}_{t+1} + \mathbf{W}_r^T \delta\overline{\mathbf{r}}_{t+1}, \tag{2.81}$$
$$\delta\overline{\mathbf{z}_t} = \mathbf{u}_t \odot \delta\mathbf{h}_t \odot \tanh'(\overline{\mathbf{z}_t}), \tag{2.82}$$
$$\delta\overline{\mathbf{u}_t} = (\mathbf{z}_t - \mathbf{h}_{t-1}) \odot \delta\mathbf{h}_t \odot \sigma'(\overline{\mathbf{u}_t}), \tag{2.83}$$
$$\delta\overline{\mathbf{r}_t} = \mathbf{h}_t \odot \delta\overline{\mathbf{h}_{t-1}} \odot \sigma'(\overline{\mathbf{r}_t}). \tag{2.84}$$

Equation 2.81 governs the state error flow. The update gate term $(1 - \mathbf{u}_{t+1})$ is perhaps the most important summand, it can allow, dampen or stop the error flow through time.

With gated RNNs gradients do not vanish as much. This insight is a consequence of equations 2.41 and 2.81. Which unlike 2.17 are not



$\mathbf{h}_t$
$+$
$\odot$
$\mathbf{z}_t$
$\bar{\mathbf{h}}_{t-1}$
$\tanh(\mathbf{W}\mathbf{x}_t + \mathbf{V}\bar{\mathbf{h}}_{t-1} + \mathbf{b})$
$\mathbf{1} - \mathbf{u}$
$\odot$
$\mathbf{u}$
$\mathbf{u}$
$\mathbf{h}_{t-1}$
$\odot$
$\mathbf{r}$
Reset Gate
$\sigma(\mathbf{W}_r\mathbf{w} + \mathbf{b}_r)$
Update Gate
$\sigma(\mathbf{W}_u\mathbf{w} + \mathbf{b}_u)$
$\mathbf{w} = [\mathbf{x}_t \ \mathbf{h}_{t-1}]^T$
$\mathbf{h}_{t-1}$
$\mathbf{x}_t$

Figure 2.5: Visualization of a GRU. This LSTM variant uses only two gates and a single state activation function. The two yellow gates use sigmoidal activations and produce outputs within zero and one. The gate outputs govern cell memory management. The update gate replaces input and forget gates, its output decides with memory contents can be modified. The reset gate allows state value deletion from the block input. The tanh activation appears only once. No output gate is used.

repeatedly remapped by a recurrent weight matrix. Gradients can still explode. As a remedy, it is often necessary to clip gradients.

In our experiments, this simplified LSTM variant performed equally well on sequence modelling tasks. It will frequently appear in the research part of this thesis. Chapter 7 moves a GRU like structure into the complex domain and presents a complex gated recurrent network. Chapter 9 explores the relative importance of the GRU equations by selectively compressing some.

2.4.3.2 *Derivation of the GRU Backward pass*

Equations 2.81 to 2.84 have been derived by applying the chain rule to equations 2.73 to 2.80, by following the backpropagation procedure [LKJ15]. Backpropagation accumulates contributions from multiple equations. If a variable has already appeared, the $+=$ notation is used to indicate summation with the existing value. Let's start with $\mathbf{h}_t = \mathbf{u}_t \odot \mathbf{z}_t + (1 - \mathbf{u}_t) \odot \mathbf{h}_{t-1}$, backpropagation yields

$$\delta \mathbf{u}_t = (\mathbf{z}_t - \mathbf{h}_{t-1}) \odot \delta \mathbf{h}_t, \tag{2.85}$$
$$\delta \mathbf{z}_t = \mathbf{u}_t \odot \delta \mathbf{h}_t, \tag{2.86}$$
$$\delta \mathbf{h}_{t-1} = (1 - \mathbf{u}_t) \odot \delta \mathbf{h}_t. \tag{2.87}$$

moving thought the hyperbolic tangent $\mathbf{z}_t = \tanh(\overline{\mathbf{z}_t})$ leads to

$$\delta \overline{\mathbf{z}_t} = \tanh'(\overline{\mathbf{z}_t}) \odot \delta \mathbf{z}_t. \tag{2.88}$$

The Backpropagation process has now arrived at the state candidate value computation, $\overline{\mathbf{z}_t} = \mathbf{W}\overline{\mathbf{h}_{t-1}} + \mathbf{V}\mathbf{x}_t + \mathbf{b}$, its contributions to the gradients are,

$$\delta \overline{\mathbf{h}_{t-1}} = \mathbf{W}^T \delta \overline{\mathbf{z}_t}, \tag{2.89}$$
$$\delta \mathbf{x}_t = \mathbf{V}^T \delta \overline{\mathbf{z}_t}, \tag{2.90}$$
$$\delta \mathbf{b} = \delta \overline{\mathbf{z}_t}. \tag{2.91}$$
$$\tag{2.92}$$

Moving further into the cell the process arrives at the multiplication with the reset gate output $\overline{\mathbf{h}_{t-1}} = \mathbf{r}_t \odot \mathbf{h}_{t-1}$, here the gradients

$$\delta \mathbf{h}_{t-1} \mathrel{+}= \mathbf{r}_t \odot \delta \overline{\mathbf{h}_{t-1}}, \tag{2.93}$$
$$\delta \mathbf{r}_t = \mathbf{h}_{t-1} \odot \delta \overline{\mathbf{h}_{t-1}} \tag{2.94}$$

are obtained. To get to the update gate we must pass through it's sigmoidal activation $\mathbf{u}_t = \sigma(\overline{\mathbf{u}_t})$ and obtain the following deltas,

$$\delta \overline{\mathbf{u}_t} = \sigma'(\overline{\mathbf{u}_t}) \odot \delta \mathbf{u}_t. \tag{2.95}$$

Using the definition of the update gate $\overline{\mathbf{u}_t} = \mathbf{W}_u \mathbf{h}_{t-1} + \mathbf{V}_u \mathbf{x}_t + \mathbf{b}_u$ it's deltas are,

$$\delta \mathbf{h}_{t-1} \mathrel{+}= \mathbf{W}_u^T \delta \overline{\mathbf{u}_t}, \tag{2.96}$$
$$\delta \mathbf{x}_t \mathrel{+}= \mathbf{V}_u^T \delta \overline{\mathbf{u}_t}, \tag{2.97}$$
$$\delta \mathbf{b} = \delta \overline{\mathbf{u}_t}. \tag{2.98}$$

The process for the reset gate architecture is identical to $\mathbf{r}_t = \sigma(\overline{\mathbf{r}_t})$ the update gate,

$$\delta\overline{\mathbf{r}_t} = \sigma'(\overline{\mathbf{r}_t}) \odot \delta\mathbf{r}_t. \tag{2.99}$$

Similarly for $\overline{\mathbf{r}_t} = \mathbf{W}_r\mathbf{h}_{t-1} + \mathbf{V}_r\mathbf{x}_t + \mathbf{b}_r$ the deltas are,

$$\delta\mathbf{h}_{t-1} \mathrel{+}= \mathbf{W}_r^T\delta\overline{\mathbf{r}_t}, \tag{2.100}$$

$$\delta\mathbf{x}_t \mathrel{+}= \mathbf{V}_r^T\delta\overline{\mathbf{r}_t}, \tag{2.101}$$

$$\delta\mathbf{b}_r = \delta\overline{\mathbf{r}_t}. \tag{2.102}$$

Going through the equations above and writing all summation expressions as individal equations leads to the gru block backward equations described earlier.

3

OPTIMIZATION

Having discussed the cost functions as well as multiple neural network architectures their gradients and how to obtain them in the previous chapter, the next step is to use these gradients to iteratively optimize neural networks.

In its most basic form a gradient update for a weight matrix **W** can be [Bis06; GBC16; Nie15],

$$\mathbf{W}_{\tau+1} = \mathbf{W}_{\tau} - \frac{\eta}{n_b} \sum_{b=0}^{n_b} \delta \mathbf{W}_{\tau,b}, \tag{3.1}$$

with the update step τ, batch size n_b and the gradient tensor $\delta\mathbf{W} \in \mathbb{R}^{n_b, n_o, n_i}$. A stochastic gradient descent step computes the mean gradients for the current batch. The learning rate η can be thought of as the step size the optimization process is going to take along the gradient.

3.1 MOMENTUM

A common modification to standard stochastic gradient descent is the addition of momentum. Using only small subsets or batches instead of the entire training set, leads to stochastic approximations of the accurate gradient. In order the prevent the stochastic approach from producing too noisy and therefore rapidly changing gradients momentum is added to the gradient update. Averaging operations are well known to reduce high-frequency noise. The momentum term gathers gradient information over multiple batches by running the gradients trough a moving average. In essence the update rule 3.1 is replaced by [Nie15; GBC16]

$$\mathbf{V}_{\tau} = \mu \mathbf{V}_{\tau-1} - \frac{\eta}{n_b} \sum_{b=0}^{n_b}, \delta \mathbf{W}_{\tau,b} \tag{3.2}$$

$$\mathbf{W}_{\tau+1} = \mathbf{W}_{\tau} + \mathbf{V}_{\tau}. \tag{3.3}$$

The new parameter μ, controls the momentum or resistance to direction change that is introduced into the optimization procedure. **V** is used to store the momentum gradients over multiple iterations.

Numerous other modifications of gradient descent exist. Most variants improve the scaling and integration of information over multiple iterations.

3.2 RMSPROP

Using a global learning rate for all parameters may not be ideal for all problems. Especially given the non-convex and stochastic nature of many neural network optimization problems, more sophisticated gradient scaling is required. When the learning rate is too big, the optimization process tends to oscillate. RMSProp attempts to make the optimization process more robust by moving quickly in the direction of small but consistent gradients and slowly in directions of larger but unreliable gradients [HSS12]. One way to scale gradients individually for each parameter is to divide gradients by running averages of their recent squared magnitude [HSS12].

Using the learning rate τ as well as the new parameter ρ to control the learning rate, RMSProp updates the weights during each step [GBC16],

$$\mathbf{G}_\tau = \frac{1}{n_b} \sum_{b=0}^{n_b} \delta \mathbf{W}_{\tau,b}, \tag{3.4}$$

$$\mathbf{R}_\tau = \rho \mathbf{R}_{\tau-1} + (1-\rho)\mathbf{G}_\tau \odot \mathbf{G}_\tau, \tag{3.5}$$

$$\mathbf{W}_{\tau+1} = \mathbf{W}_\tau - \frac{\eta}{\sqrt{\delta + \mathbf{R}_{\tau+1}}} \odot \mathbf{G}_\tau. \tag{3.6}$$

After computing the gradients $\mathbf{G}_\tau$, squared gradients are accumulated in $\mathbf{R}_\tau$, the learning rate is then adapted element-wise based on the values in $\mathbf{R}_\tau$ [GBC16]. The small constant δ is meant to increase numerical stability, by guarding against division by tiny numbers.

3.3 ADAM

The adaptive moments or Adam [GBC16; KB15] approach keeps track of first and second moments and adds correction terms for both. The moments are averages of the gradient and the squared gradient. Based on the corrected moments, individual learning rates for each parameter are computed [KB15]. Two accumulation hyper-parameters

ρ_1 and ρ_2 are present, and must chosen by hand. For a single weight update Adam computes

$$\mathbf{G}_\tau = \frac{1}{n_b}\sum_{b=0}^{n_b} \delta\mathbf{W}_{\tau,b}, \tag{3.7}$$

$$\mathbf{S}_\tau = \rho_1\mathbf{S}_{\tau-1} + (1-\rho_1)\mathbf{G}_\tau, \tag{3.8}$$

$$\mathbf{R}_\tau = \rho_2\mathbf{R}_{\tau-1} + (1-\rho_2)\mathbf{G}_\tau \odot \mathbf{G}_\tau, \tag{3.9}$$

$$\hat{\mathbf{S}}_\tau = \frac{1}{1-\rho_1^t}\mathbf{S}_\tau, \tag{3.10}$$

$$\hat{\mathbf{R}}_\tau = \frac{1}{1-\rho_2^t}\mathbf{R}_\tau, \tag{3.11}$$

$$\mathbf{W}_{\tau+1} = \mathbf{W}_\tau - \eta\frac{\hat{\mathbf{S}}_\tau}{\delta + \sqrt{\hat{\mathbf{R}}_{\tau+1}}}. \tag{3.12}$$

Overall Adam combines RMSProp and Momentum and adds bias correction. It is regarded as very robust with respect to its hyperparameters but overall no dominant optimzation approach to neural networks has emerged [GBC16].

4

SIGNAL PROCESSING

This thesis explores machine learning using frequency-domain methods, most notably, the fast Fourier and wavelet transform. Where complex numbers emerge after the transform, complex-valued machine learning approaches are developed and deployed.

4.1 COMPLEX NUMBERS

Complex numbers frequently appear in the signal processing and complex machine learning parts of this text. This section aims to provide a short introduction to the topic. Squaring a real number is always positive. $x^2 = -1$ has no real solutions. Complex numbers start from $i^2 = -1$[Str06]. An agreed solution to an old problem. Without it, not all quadratic or cubic equations have a solution. Proposed in 1572 by Rafael Bombelli [Bor13], complex numbers were not immediately accepted and were referred to as absurd, useless, or imaginary. The last term stuck and is still used today, albeit without negative connotation [Bor13].

Typically written as $z = x + iy$, complex numbers consist of a real $\Re(z) = x$ and imaginary $\Im(z) = y$ part. Both parts combined can be used as coordinates in a two-dimensional plane. Like we do for two dimensional real vectors, complex numbers are added by adding real and imaginary parts separately. Complex conjugation has no real equivalent, because it flips the sign of the imaginary part $\bar{z} = x - iy$. Conjugation mirrors z along the real axis. Instead of using two coordinates points, angle and radius equivalently define a point in two dimensions. Conventionally the angle is measured counter-clockwise starting at the x-axis at the origin. This idea leads to the polar form of complex numbers $z = re^{i\phi}$, with radius r and angle ϕ. The polar form simplifies multiplication. Complex numbers are multiplied by multiplication of both radii and addition of the phase-angles. Orthogonal complex matrices are called unitary matrices. The word unitary signals orthogonality in addtion to the complex nature of the matrix it refers to.

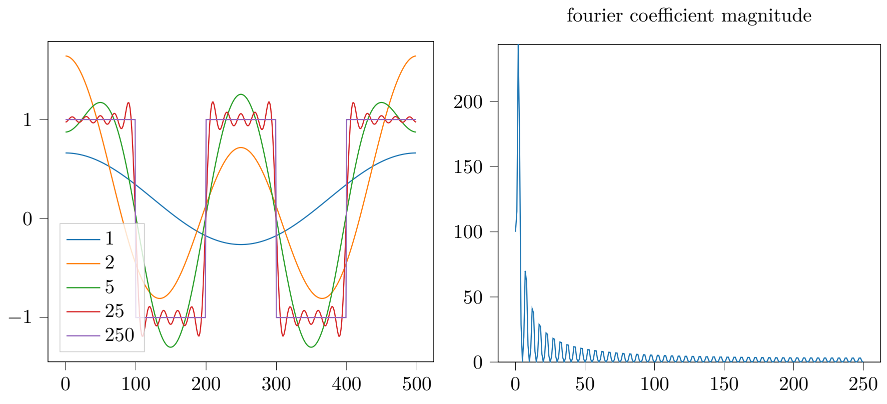


Figure 4.1: Visualization of a Fourier Series approximating a rectangular pulse (left). As more and more terms are used the approximation's accuracy improves. On the right the magnitued of all fourier coefficients is shown.

4.2 THE DISCRETE FOURIER TRANSFORM

The Fourier transform moves a function f from physical space into frequency space. Once transformed it is represented as a sum of harmonics $c_k e^{i\omega t}$ [Str06; Zei12],

$$f(t) = \sum_{\omega} c_k e^{i\omega t}. \tag{4.1}$$

With time t, frequency ω and the imaginary unit i. Once moved over into the frequency domain representation F, the coefficients can be analyzed and filtered. During the conversion the Fourier coefficients c_k are computed using:

$$F(\omega) = \sum_{t} x_t e^{-i\omega t}, \tag{4.2}$$

In figure 4.1 a signal consisting of rectangular pulses is Fourier transformed. To explore the effect of the sum as described in equation 4.2 more and more coefficients are added to the summed representation of the signal. As terms are added the approximation's accuracy increases. The sum consists of complex exponentials. According to Euler's Formula $e^{ix} = \cos(x) + i\sin(x)$ we require the complex part to cancel for real valued signals, which leads to the relation $c_{-k} = \bar{c}_k$, in this case [Zei12]. Convolutions turn into multiplications in the Fourier domain [Zei12]:

$$\mathcal{F}\{f * g\} = \mathcal{F}(f)\mathcal{F}(g). \tag{4.3}$$

Together with the fast fourier transform, this finding is extremely important for the efficent implementation of convolutional neural networks [Vas+15].

Looking at the Fourier coefficients c_k means looking at the function in frequency space. The process of computing the c_ks from the function

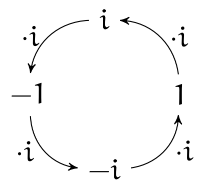


Figure 4.2: Repeated multiplication by i, causes rotation around the origin. Its important to note that 1 can also be represented by i^4, a full rotation. Full rotations can also be present in representations for any other point on the unit circle. Minus one for example could also be i^6.

is referred to as forward transform $\mathcal{F}$. The reconstruction of the original signal from the coefficients is called the inverse transform $\mathcal{F}^{-1}$.

The synthesis operation $\mathcal{F}^{-1}(\mathbf{c}) = \mathbf{Fc}$, can be seen as multiplication with a Fourier matrix [Stro6]. The matrix follows from formula 4.1. Time t and frequency ω increase along the rows and columns,

$$\mathbf{F} = \begin{pmatrix} 1 & 1 & 1 & 1 & . & 1 \\ 1 & w & w^2 & w^3 & . & w^{(n-1)} \\ 1 & w^2 & w^4 & w^6 & . & w^{2(n-1)} \\ 1 & . & . & . & . & . \\ 1 & w^{n-1} & w^{2(n-1)} & w^{3(n-1)} & . & w^{(n-1)^2} \end{pmatrix}. \tag{4.4}$$

The ws in the Fourier matrix equal complex exponentials $w = e^{2\pi j/n}$. For n=4 the discrete Fourier matrix is given by,

$$\mathbf{F}_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & i^2 & i^3 \\ 1 & i^2 & i^4 & i^6 \\ 1 & i^3 & i^6 & i^9 \end{pmatrix}. \tag{4.5}$$

With this definition, the synthesis case which takes us from the time to the frequency space is the complex conjugate $\bar{\mathbf{F}}$.

4.3 THE FAST FOURIER TRANSFORM

Typically the computation of a matrix vector product $\mathbf{Fx}$ uses n^2 operations, an operation for each of the n^2 entries in the matrix F. For sparse matrices with many zeros its possible to do significantly better and reduce the number of operations considerably. But the Fourier matrices does not have any zeros in its original form. The key idea behind the fast Fourier transform is to introduce zeros by factoring the Fourier matrix. This is possible if the number of inputs is a power of two [Zei12]. $\mathbf{F}_4$ from equation 4.5 can be factored into a combination of $\mathbf{F}_2$ matrices [Str06]:

$$\mathbf{F}_4 = \begin{pmatrix} 1 & & 1 & \\ & 1 & & i \\ 1 & & -1 & \\ & 1 & & -i \end{pmatrix} \begin{pmatrix} 1 & 1 & & \\ 1 & i^2 & & \\ & & 1 & 1 \\ & & 1 & i^2 \end{pmatrix} \begin{pmatrix} 1 & & & \\ & & 1 & \\ & 1 & & \\ & & & 1 \end{pmatrix}. \qquad (4.6)$$

$\mathbf{F}_2 = \begin{pmatrix} 1 & 1 \\ 1 & i^2 \end{pmatrix}$

Above $\mathbf{F_2}$ appears twice in the center block matrix. Its flanked by two additional matrices which turn the product into the discrete fourier transform matrix, all three matrices are now sparse. Taking into account the rotation effect of multiplication by i as shown in figure 4.2, multiplication of the matrices in equation 4.6 produces $\mathbf{F}_4$. Larger transforms can be brocken down into smaller transforms with increasing levels of sparsity using the same principle. In this case much more points follow from $w = e^{2\pi i/n}$, and therefore the circle in figure 4.2 will involve more than just $1, i, -1$ and $-i$. In the case of $\mathbf{F}_{1024}$ this leads to [Str06]:

$$\mathbf{F}_{1024} = \begin{pmatrix} \mathbf{I}_{512} & \mathbf{D}_{512} \\ \mathbf{I}_{512} & -\mathbf{D}_{512} \end{pmatrix} \begin{pmatrix} \mathbf{F}_{512} & \\ & \mathbf{F}_{512} \end{pmatrix} \begin{pmatrix} \text{even-odd} \\ \text{permutation} \end{pmatrix}. \qquad (4.7)$$

Above the $\mathbf{F}_{512}$ blocks can be broken down further [Str06]:

$$\begin{pmatrix} \mathbf{I}_{256} & \mathbf{D}_{256} & & \\ \mathbf{I}_{256} & -\mathbf{D}_{256} & & \\ & & \mathbf{I}_{256} & \mathbf{D}_{256} \\ & & \mathbf{I}_{256} & -\mathbf{D}_{256} \end{pmatrix} \begin{pmatrix} \mathbf{F}_{256} & & & \\ & \mathbf{F}_{256} & & \\ & & \mathbf{F}_{256} & \\ & & & \mathbf{F}_{256} \end{pmatrix} \qquad (4.8)$$

$$\begin{pmatrix} \text{pick}\ , 0, 4, 8, \ldots \\ \text{pick}\ , 2, 6, 10, \ldots \\ \text{pick}\ , 1, 5, 9, \ldots \\ \text{pick}\ , 3, 7, 11, \ldots \end{pmatrix} = \begin{pmatrix} \mathbf{F}_{512} & \\ & \mathbf{F}_{512} \end{pmatrix}. \qquad (4.9)$$

The pick operation refers to the rows of the matrix product on it's left. The above result can be inserted into equation 4.7, repeating this process leads to the full FFT. Using a FFT instead of the full matrix multiplication requires $0.5n\log_2 n$ operations instead of the n^2 for the full matrix multiplication [Str06; Zei12].

4.4 THE SHORT TIME FOURIER TRANSFORM

If applied directly the Fourier transform decomposes a signal in its entirety into frequency domain components. In order to learn something about the evolution of the signal's frequency composition over time, windowing is used, which leads to the STFT.

4.4.1 *Forwards STFT*

The Fourier transform maps a signal into the spectral or frequency domain by decomposing it into a linear combination of sinusoids. In its regular, the transform requires infinite support of the time signal to estimate the frequency response. For many real-world applications, however, including those which we wish to model with Recurrent Neural Networks (RNNs), this requirement is not only infeasible, but it may also be necessary to obtain insights into changes in the frequency spectrum as a function of time or signal index. To do so, one can partition the signal into overlapping segments and approximating the Fourier transform of each segment separately. This is the core idea behind the short-time Fourier transform (STFT), which is used to determine a signal's frequency domain representations as it changes over time.

More formally, given a signal $\mathbf{x}$, we can partition it into segments of length T, extracted every S time steps. The STFT $\mathcal{F}_s$ of $\mathbf{x}$ is defined by [GL84] as the discrete Fourier transform of $\mathbf{x}$, i. e.

$$\begin{aligned} \mathbf{X}[\omega, Sm] &= \mathcal{F}_s\left(\mathbf{x}\right) \\ = \mathcal{F}\left(\mathbf{w}[Sm-l]\mathbf{x}[l]\right) &= \sum_{l=-\infty}^{\infty} \mathbf{w}[Sm-l]\mathbf{x}[l]e^{-i\omega l} \end{aligned} \tag{4.10}$$

where $\mathcal{F}$ denotes the classic discrete fast Fourier transform. Segments of $\mathbf{x}$ are multiplied with the windowing function $\mathbf{w}$ and transformed afterwards.

Historically, the shape and width of the window function has been selected by hand. To hand the task of window selection to the optimizer, we work with a truncated Gaussian window [Har78],

$$w[n] = \exp\left(-\frac{1}{2}\left(\frac{n-N/2}{\sigma N/2}\right)^2\right) \tag{4.11}$$

of size N and learn the standard deviation σ. The larger that σ is, the more the window function approaches a rectangular window; the smaller the sigma, the more extreme the edge tapering and as a result, the narrower the window width.

4.4.2 *Inverse STFT*

Supposing that we are given some frequency signal $\mathbf{X}$; the time signal $\hat{\mathbf{x}}$ represented by $\mathbf{X}$ can be recovered with the inverse short time Fourier transform (iSTFT) $\mathcal{F}_s^{-1}$ and is defined by [GL84] as:

$$\begin{aligned} \hat{\mathbf{x}}[n] &= \mathcal{F}_s^{-1}(\mathbf{X}[n, Sm]) \\ &= \frac{\sum_{m=-\infty}^{\infty} \mathbf{w}[Sm-n]\hat{\mathbf{x}}_w[n, Sm]}{\sum_{m=-\infty}^{\infty} \mathbf{w}^2[Sm-n]} \end{aligned} \tag{4.12}$$

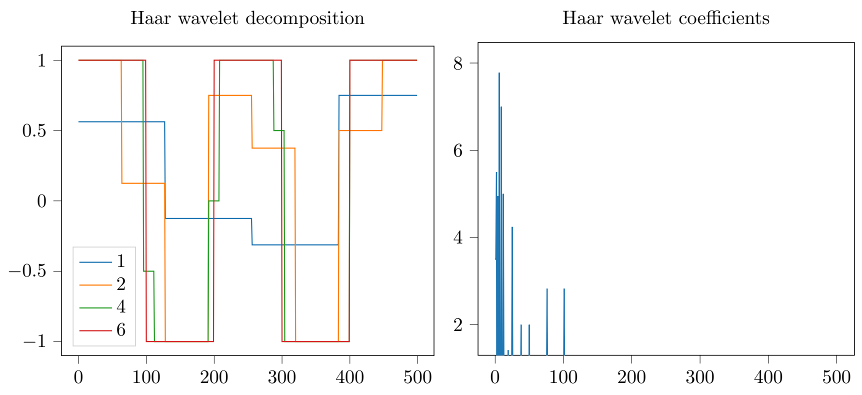


Figure 4.3: Wavelet approximation of a rectangular pulse function with an increasing number of scales (left). Wavelet coefficients for all scales (right).

where the signal $\hat{x}_w$ is the inverse Fourier transform of $\mathbf{X}$:

$$\hat{x}_w = \frac{1}{T}\sum_{l=-\infty}^{\infty} \mathbf{X}[l, Sm]e^{j\omega l} \tag{4.13}$$

and l indexing the frequency dimension of $\mathbf{X}_m$.

Eq. 4.12 reverses the effect of the windowing function, but implementations require careful treatment of the denominator to prevent division by near-zero values[1]. In Eq. 4.12, Sm generally evaluates to an integer smaller than the window size T and subsequent elements in the sum overlap, hence the alternative naming of it being an *"overlap and add"* method [Grö13].

4.5 WAVELETS

What if we could work with finite basis functions, instead of having to window the infinite sine and cosine waves, which make up the Fourier basis? This leads to the FWT and iFWT. [2] Derived from the French *ondelette* the word wavelet was coined to express the notion of a small wave, which exists only for a limited amount of time. A rectangular pulse was Fourier transformed in figure 4.1, to explore the fast wavelet transform the same pulse is again transformed using wavelets in figure 4.3. As the FWT is a multiscale approach, scales are considered together. For an input signal of length 512, an FWT produces scale coefficients of length $256, 128, 64, 32, 18, 8, 4$ and 2. In figure 4.3 these scales are concatenated from small to large. In comparison to figure 4.1 the resulting coefficients are much sparser. The Haar wavelets used here are rectangular functions, which are much better suited to represent the rectangular pulse than the Fourier basis, which requires many terms to

1 We adopt the common strategy of adding a small tolerance $\epsilon = 0.001$
2 A PyTorch implementation is available at `https://github.com/v0lta/PyTorch-Wavelet-Toolbox` and Jax code at `https://github.com/v0lta/jaxlets`

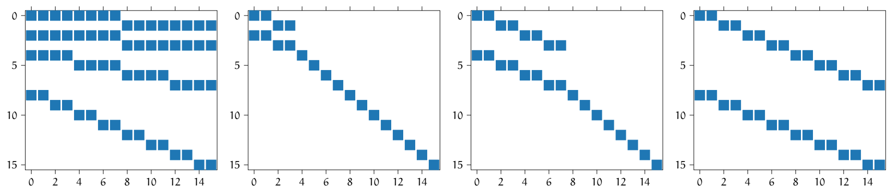


Figure 4.4: Haar analysis fast wavelet transformation matrix on the left followed by individual scale processing matrices.

approximate the sharp corners. In machine learning, wavelets form the core of scatter-nets. Initially introduced as translation-invariant signal representation [Mal12], optimizable variants have since been proposed [Cot19; CK19]. Infinitely many wavelets exist. Typically hand-crafted wavelets are used, recently wavelet optimization has been explored in the machine learning literature [RG13; WLY20]. Wavelet optimization will play an important role in the experimental part of this thesis.

4.5.1 *The fast wavelet transform*

The fast wavelet transform constitutes a change of representation, it expresses the input data points in terms of a wavelet filter pair by computing [SN96]:

$$\mathbf{b} = \mathbf{A}\mathbf{x} \tag{4.14}$$

The analysis matrix $\mathbf{A}$ is the product of multiple matrices, each decomposes the input signal $\mathbf{x}$ at an individual scale. Finally multiplication of the total matrix $\mathbf{A}$ with the input-data yields the wavelet coefficients $\mathbf{b}$. We show the structure of the individual matrices in figure 4.4, on the left. The full resulting operator is shown on the right. The three plots on the right of figure 4.4 show a growing identity block matrix as the fwt moves trough the different scales. After every step, the data we must process is cut in half. The reoccurring diagonals denote convolution operations with the filter pair H_0 and H_1. Overall we observe the pattern [SN96],

$$\mathbf{A} = \dots \left(\begin{array}{c|c} \mathbf{H}_0 & \\ \mathbf{H}_1 & \\ \hline & \mathbf{I} \end{array}\right) \begin{pmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{pmatrix}. \tag{4.15}$$

We have seen how to construct the analysis matrix $\mathbf{A}$ and will now consider inverting this process, overall the inverse fast wavelt transform (iFWT) can again be thought of as a linear operation,

$$\mathbf{S}\mathbf{b} = \mathbf{x}. \tag{4.16}$$

The synthesis matrix $\mathbf{S}$ is constructed using the synthesis filter pair F_0, F_1. We show the multi-scale reconstruction matrices in figure 4.5

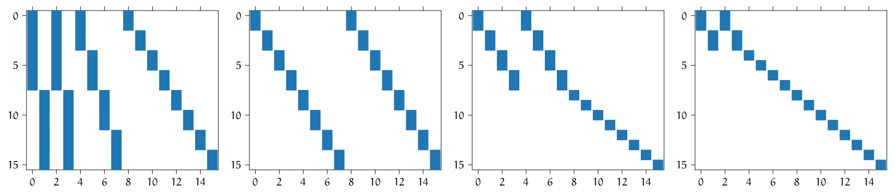
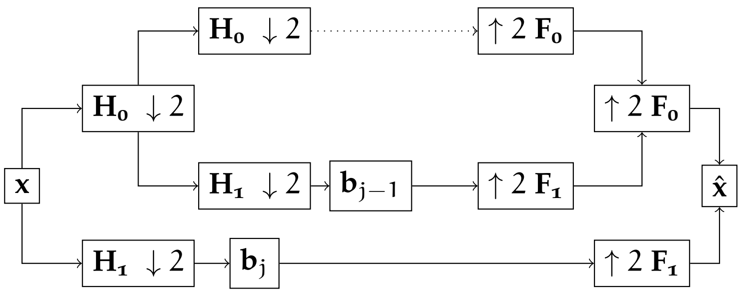


Figure 4.5: Haar synthesis backward fast wavelet transformation matrices for three scales as well as the complete inverse matrix.



$\mathbf{H_0} \downarrow 2$
$\uparrow 2\ \mathbf{F_0}$
$\mathbf{H_0} \downarrow 2$
$\uparrow 2\ \mathbf{F_0}$
$\mathbf{x}$
$\mathbf{H_1} \downarrow 2$
$\mathbf{b}_{j-1}$
$\uparrow 2\ \mathbf{F_1}$
$\hat{\mathbf{x}}$
$\mathbf{H_1} \downarrow 2$
$\mathbf{b}_j$
$\uparrow 2\ \mathbf{F_1}$

Figure 4.6: Efficient wavelet signal analysis and synthesis following a tree structure [SN96]. **H** denotes analysis filters and **F** stands for synthesis filters. Up (↑) and down (↓)- sampling by a factor of two is written as the arrow followed by the factor. Filtering and sampling can be accomplished jointly in deep learning frameworks by using strided convolutions for analysis and strided transposed convolutions for synthesis. In place of the dotted arrow, more scale-levels can be included.

on the left and the complete synthesis matrix structure on the right. Structurally the synthesis matrices are transposed in comparison to their analysis counterparts. In general we observe:

$$\mathbf{S} = \begin{pmatrix} \mathbf{F}_0 & \mathbf{F}_1 \end{pmatrix} \left(\begin{array}{cc|c} \mathbf{F}_0 & \mathbf{F}_1 & \\ \hline & & \mathbf{I} \end{array} \right) \cdots \tag{4.17}$$

We have $\mathbf{SA} = \mathbf{I}$, which guarantees invertability, which in turn leads to conditions on the filter pairs H_0, H_1 as well as F_0, F_1.

The fast wavelet transform can also be computed using strided convolutions instead of sparse matrix multiplications. This approach is shown in figure 4.6. Each block represents a convolution with stride two. The double indices jk of b_{jk} denote the scale j and time k positions of each coefficient.

Coefficients b_{jk} can be found by recursively convolving $\mathbf{x}$ with the analysis filters $\mathbf{h}_0$ and $\mathbf{h}_1$ with a stride of 2. This process is depicted in Figure 4.6. The depth of the multi-resolution representation depends on the number of scales [JC01] and is chosen depending on the problem. As a result of the strided convolution, the number of time steps is halved after each scale step. By working backwards through the scales, one can reconstruct $\hat{\mathbf{x}}$ from b_{jk} through transposed convolutions with the synthesis filters $\mathbf{f}_0$ and $\mathbf{f}_1$.

4.5.2 *Wavelet properties*

Wavelets are typically selected from a library of established wavelets. The rectangular Haar wavelet which previously appeared in Figure 4.3, is a common choice. Alternatively wavelets can be arbitrarily designed by hand by the practitioner. Based on the product filter approach, designed wavelets must fulfill conditions of perfect reconstruction and alias cancellation [SN96]. Given filters **h** and **f** as well as their z-transformed counterparts $H(z) = \sum_n \mathbf{h}(n)z^{-n}$ and $F(z)$ respectively, the reconstruction condition can be expressed as

$$H_0(z)F_0(z) + H_1(z)F_1(z) = 2z^l, \tag{4.18}$$

and the anti-aliasing condition as

$$H_0(-z)F_0(z) + H_1(-z)F_1(z) = 0. \tag{4.19}$$

For the perfect reconstruction condition in Eq. 4.18, the center term z^l of the resulting z-transformed expression must be a two; all other coefficients should be zero. l denotes the power of the center. Custom wavelet designs are often build around these two conditions.

Part II

RESEARCH

5 RELATED WORK

Frequency domain methods play a key role in modern machine learning. Fast Fourier transforms speed up convolution computations in CNNs [MHL13]. Furthermore, frequency domain representations can provide a rich space for feature learning, where complex weights are optimized [RSA15] or allow linear layer compression trough sparse representation [LSS13; WLY20]. The Fourier transform's ability to separate the contributions at various frequencies makes transformed signals useful input features. Notable areas of application are for example speech [Cha+16; Wol17], music [Thi+18] or motion capture processing [Ene+20].

5.1 RECURRENT NEURAL NETWORKS

Recurrent networks are a very common approach to sequence modeling [GBC16]. Historically the exploding and vanishing gradient problems long inhibited the use of recurrent neural networks [Hoc91; Hoc+01; BSF94; GBC16]. The long short term memory cell [HS97b] mitigates the vanishing gradient problem by introducing the gating mechanism presented in chapter 2.4.2. Instead of repeated multiplication with a recurrent weight matrix in the backward pass, LSTM distributes the gradients over a sum [HS97b; SMS15]. Matrix multiplication changes the norm of the resulting vector if the matrix is not orthogonal or unitary. Gradient distribution over sums instead of (matrix)-products avoids this problem. It is most likely no coincidence that very deep residual convolutional neural networks rely on a sum based mechanism for stability as well [SGS15b; SGS15a; He+16; Kus+18]. Feedforward-gating, as proposed in [SGS15b], has not become mainstream in architectures such as resnets [He+16]. Perhaps an indication that summation is more important than gating or that gating is only required, in cases where matrices are reused over time.

More than 10 years after LSTM was first proposed [HS97b] its most important variant the gated recurrent unit GRU [Cho+14] appeard. An in depth discussion can be found in chapter 2.4.3. In order to process complex Fourier coeffcients in $\mathbb{C}$ a complex-valued version is discussed in [WY18] and chapter 7 of this text. Gates do not solve exploding gradients, which is why gradient clipping is still necessary [GBC16]. Ungated unitary or orthogonal RNN-cells can be stable [ASB16; Wis+16] and do not require gradient clipping[ASB16]. Without gates, however, the cell state is unprotected. As a result, it is susceptible to corruption by input noise [WY18]. Recently convolutional structures have been explored as an alternative to recurrent connections and gating [BKK18]. A later follow-up paper from the same authors found that convolutional structures too benefit from recurrent connections and gating [BKK19].

5.2 COMPLEX NETWORKS

When working with complex network weights, complex activation functions [Hir12; MG09], as well as update rules [MG09], are required.

It is not obvious how to best optimize complex networks. Their optimization has long been discussed in the literature [KM94; LH91; BP92; Kre09]. Complex gradients exist, when activation and cost functions are complex-differentiable or holomorph. A function $f(z) = u(x, y) + iv(x, z)$ of variable $z = x + iy$ is holomorph when [Bor13]

$$\partial_x u = \partial_y v \text{ and } \partial_y u = -\partial_x v. \tag{5.1}$$

Holomorphy is desirable, but holomorph functions are also unbounded [MG09; Lio79]. Holomorphy can alternatively be described in terms of Wirtinger-Operators [Bor13][Kre09]

$$\partial = \frac{1}{2}(\partial_x - i\partial_y) \text{ and } \overline{\partial} = \frac{1}{2}(\partial_x + i\partial_y). \tag{5.2}$$

In the holomorph case we have $\overline{\partial} f(z) = 0$ and the derivateve $\partial f(z) = f'(z)$. In our experiments, singularities turned out to be a particularly important problem. It is not always possible to work with unbounded activations, and therefore very convenient to sidestep holomorphy and optimize complex networks with respect to their real and complex parts [Kre09; MG09]. In this case we are working with a complex pseudoderivative $\partial f(z) + \overline{\partial} f(z) = f'(z)$. The chain rule holds still holds in this case if the real and complex parts are considered seperately [Kre09]. All activations considered next fall into this category.

For complex CNNs the cRelu allows complex numbers with positive real and imagniary parts to pass. This early non-holomorph actovation function appeared in [Gub16],

$$\text{zReLU}(z) = \begin{cases} re^{i\phi_z} & \text{if } \phi_z \in [0, \pi/2] \\ 0 & \text{otherwise.} \end{cases} \tag{5.3}$$

A plot of its magnitude r in the complex plane is shown in figure 5.1 on the right. The cRelu which considers the real and imaginary parts separately [Tra+18],

$$\text{cReLU}(z) = \text{ReLU}(x) + i\text{ReLU}(y). \tag{5.4}$$

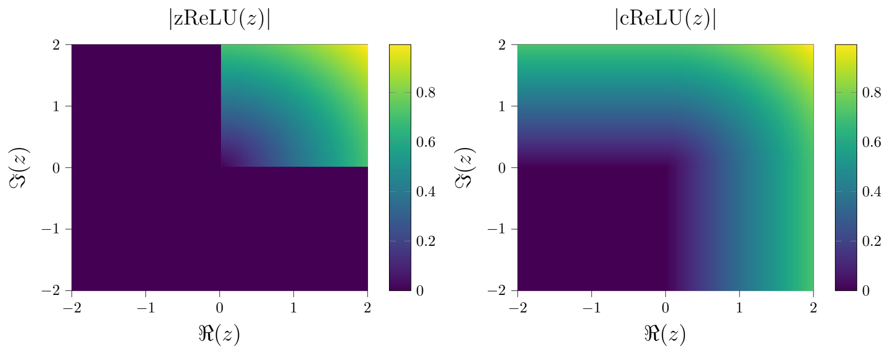


Figure 5.1: Popular Feedforward activation functions in ℂ. The zReLU [Gub16] allows complex numbers with positive real and imaginary parts to pass. While the cReLU [Tra+18], applies standard-ReLUs seperately on the real and imaginary parts.

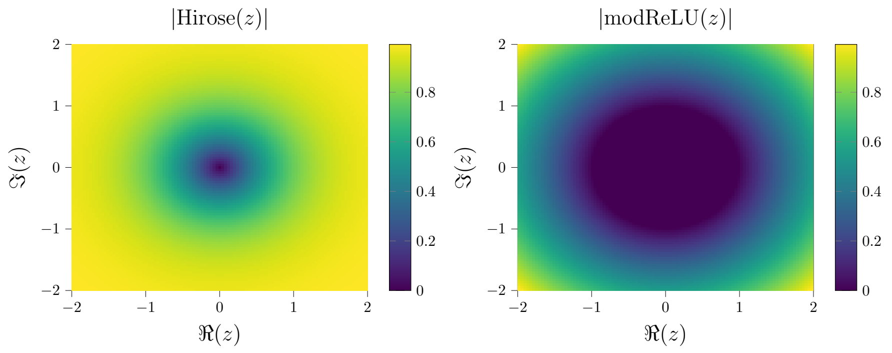


Figure 5.2: Popular activation functions in ℂ. The Hirose activation emplys the hyperbolic tangent function to bound the radius at 1. The modReLU activation creates a learnable dead zone around the origin. Like its real counterpart, the modReLU is unbounded. Note that both activations preserve phase information.

An evaluation of both architectures is part of [Tra+18]. The cReLU was found to perform better. A notable phase-sensitive Relu extension is the cardioid [VSL17]. Recurrent work usually relies mostly on two activations. The Hirose activation [Hir13] is defined as

$$\text{Hirose}(z) = \tanh(r)e^{i\phi_z}. \tag{5.5}$$

It is the older choice is an adaptation of the very popular hyperbolic tangent for complex RNN. The modRelu activation [ASB16],

$$\text{modReLU}(z) = \text{ReLU}(r + b)e^{i\phi_z}, \tag{5.6}$$

introduces a dead zone around the origin as can be seen in figure 5.2 on the left. It was proposed for use in RNNs, with unitary state transition matrices. Notably the Hirose and modReLU activations preserve phase information, it is passed trough the activation unchanged. This observation is discussed further in capter 7.

Complex recurrent networks are actively investigated. Early work focused on ways to update complex network weights [KM94; HO99]. Norm preservation stabilizes the training process [ASB16]. Specialized training schemes work with additive [HR17] and multiplicative [Wis+16] updates, to ensure that recurrent weights remain unitary. An orthogonal line of work considers gated complex cells using complex cell states [Dan+16b], or fully complex cells [WY18].

Appliactions of complex neural networks include the processing of fourier coeffcients for motion prediction[WGY20], music recognition [Tra+18], speech processing [Wis+16], video prediction [FB19; Est+18; WYB20], medical imaging [Col+19; VSL17], and quantum machine learning [Bau20; Fra+20].

5.3 FOURIER NETWORKS

The Fourier transform, in particular, is a prominent source of inspiration in machine learning. Early pioneering work combined the Fourier transform and Elman-cells [Zha01], with a complex weight optimization algorithm [KM94]. Research in this direction continued and deep networks with sinusoidal activations for forecasting tasks appeared [GA14]. Recently [Sit+20] explored the usage of sinusoidal activation functions for image and audio processing problems using a multilayer perceptron (MLP). Using a similar method [Tan+20] finds that Fourier features enable MLPs to represent and process high-frequency image components more efficiently.

In the recurrent case [Zha+18] studies cell dynamics and proposes to model the evolution of RNN cell states using Fourier basis functions. Fourier features can also serve as input and prediction space [PPL20; WGY20], for forecasting tasks on periodic data such as power-load, resource utilization or human motion prediction.

CNN benefit from efficient implementations based on the FFT. In the frequency-domain, convolution turns into multiplication. Individual convolutions can be re-expressed through [Vas+15],

$$\mathbf{x} * \mathbf{w} = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{x}) \odot \mathcal{F}(\mathbf{w})). \tag{5.7}$$

Above $\mathbf{x}$ is used to denote the input and $\mathbf{w}$ a kernel slice. This formulation suggests moving the weights as well as pooling into the frequency domain as well [RSA15]. However according to [RSA15] weights are moved back into the time domain, before convolving, perhaps to sort out the padding. More recently [Pra+17] proposed training CNN completely in the frequency domain.

Attention or transformer structures appear in mordern nerual networks [Wan+18b; Vas+17]. Layers which similarly create non-local connections can be created using the fast Fourier transform. The main idea is to compute one-sided FFTs, and utilize cReLUs [Tra+18] to zero out some coefficients. A onesided inverse fast Fourier transform (iFFT) will then produce real output values [CJM20].

GANs have been found to create network-specific artifacts is the images they create [Mar+19]. To detect GAN-generated image content [DSW20], proposes to use Fourier features to automatically detect artificially generated pictures.

5.4 WAVELET NETWORKS

Wavelet features have long been used for in signal pre-processing and denoiseing purposes [Abi09]. The FWT forms the core of the scattering transform [Mal12; CCM14], which is an alternative to convolutional layers for early stage feature processing. Integrating the FWT transform into CNNs [BM13] makes it possible to place bounds on the effects of image deformations and noise [CK19]. The fixed nature of the scattering transform limits its utility in machine learning. More flexible variants [CK19; Cot19], address this shortcoming, in the hope to improve scatter net classification accuracies.

By treating the wavelet transform and it's inverse as an autoencoder [RM18], proposes a soft-constraint for the optimization of quadrature mirror filters. Previously the autoencoder view has also been used to optimize graph-wavelets [RG13], the optimization-process is constrained by enforcing vanishing moments within a lifting scheme.

Wavelet features are used in time series forecasting [DAC16; CYD06]. Wavelet neural networks produce forecasts based on multiple resolutions and can be pruned based on predictability measures [DAC16].

5.5 NETWORK COMPRESSION

For audio and image compression, frequency domain approaches are established best practices [SN96]. Recently an adaptive wavelet transform based linear layer has been proposed which reduces the parameters in convolutional and recurrent neural networks [WLY20]. Other network compression techniques include, pruning [LDS90], quantization[Kri18] and Huffman coding [HMD16a].

Pruning algorithms require a measure of parameter importance. Based on the measure the least important parameters are removed. Hessian matrix approximations [LDS90] have been proposed an importance measure, alternatively it is possible to remove neurons with small connection weights and fine-tune the resulting network [Han+15].

Quantization can substantially reduce model sizes by representing model weights as integers, which require fewer bits to store. Conversion typically utilizes scaling, addition and rounding [Kri18]. Dequantization undoes the scaling, and addition, but the rounding makes it a lossy compression approach.

After quantization Huffman coding further reduces the network size on disk [HMD16a]. The reduction comes at the cost of a coding and decoding step, which adds overhead, whenever the network is stored or loaded from disk. Since Huffman codes require a fixed limited set of weights it can only be used on quantized networks. Size reductions of up to 22% have been reported for an Alex-Net architecture [HMD16a].

For shallow CNNs, wavelet coding can further reduce the huffman coded network size. When the CNN filters are sufficiently correlated, the wavelet transform squishes the weight histogram towards zero. See chapter 9.5.2 for a more detailed report.

Given the multidimensional nature of the tensors used in neural networks, generalizations of principal axis methods are useful compression tools. These can compress CNN [JVZ14a] and lead to a speedup when a convolution formulation for the compressed parameter-tensor is defined. Definitions exist for the the CP-decomposition [Leb+14]. Similarly, CNN and RNN see significant size reductions when a tensor train representation is used [Nov+15; TSN17]. No having to compute dense matrix approximations of the compressed tensor train factors makes evaluating the compressed networks much more efficient. [Nov+15] provides formulae, which allow direct evaluation of linear layers represented in the compressed tensor train format.

6 SPECTRAL RECURRENT NEURAL NETWORKS [WGY20]

6.1 INTRODUCTION

This chapter corresponds to the paper: [WGY20] Moritz Wolter, Juergen Gall, and Angela Yao. SequencePrediction using Spectral RNNs. In: ICANN. 2020.

Deployment of machine learning methods in embedded and real-time scenarios leads to challenging memory and computational constraints. We propose to use the short time Fourier transform (STFT) to make sequential data forecasting using recurrent neural networks (RNNs) more efficient. The STFT windows the data and moves each window into the frequency domain. A subsequent recurrent network processes multi sample windows instead of single data points and therefore runs at a lower clock rate, which reduces the total number of RNN cell evaluations per time step. Additionally working in the frequency domain allows parameter reductions trough low-pass filtering. Combining both ideas reduces the computational footprint significantly. We hope this reduction will help deploying RNNs on VR-devices for human pose prediction or on embedded systems used for load forecasting and management.

We show in our work that it is possible to propagate gradients through the STFT *and* its inverse. This means that we can use time domain-based losses that match the (temporal) evaluation measures, but still apply the RNN in the frequency domain, which we find imparts several computation and learning benefits. Firstly, representations can often be more compact and informative in the frequency domain. The Fourier basis clusters the most important information in the low-frequency coefficients. It conveniently allows us to integrate a low-pass filter to not only reduce the representation size but also remove undesirable noise effects that may otherwise corrupt the learning of the RNN.

One of the challenges of working with the Fourier transform is that it requires handling of complex numbers. While not yet mainstream, complex-valued representations have been integrated into deep convolutional architectures [Tra+18], RNNs [ASB16; Wis+16; WY18] and Hopfield like networks [GO19]. Complex-valued networks require some additional overhead for book-keeping between the real and imaginary components of the weight matrices and vector states, but tend to be more parameter efficient. We compare fully complex and real valued-concatenation formulations.

Naively applying the RNN on a frame-by-frame basis does not always lead to smooth and coherent predictions [Mao+19]. When working with STFT-windows of data, smoothness can be built-in by design through low-pass filtering before the iSTFT. Furthermore, by reducing the RNN's effective clockrate, we extend the memory capacity (which usually depends on the number of recurrent steps taken) and achieve computational gains.

In areas such as speech recognition [Cha+16] and audio processing [DS14], using the STFT is a common pre-processing step, e. g. as a part of deriving cepstral coefficients. In these areas, however, the interest is primarily for classification based on the spectrum coefficients and the complex phase is discarded, so there is no need for the inverse transform and the recovery of a temporal signal as output. We advocate the use of the forwards and inverse STFT directly before and after recurrent memory cells and propagate gradients through both the forwards and inverse transform. In summary,

- we propose a novel RNN architecture for analyzing temporal sequences using the STFT and its inverse.
- We investigate the effect of real and complex valued recurrent cells.
- We demonstrate how our windowed formulation of sequence prediction in the spectral domain, based on the reduction in data dimensionality and rate in RNN recursion can significantly improve efficiency and reduce overall training and inference time.

Source code for this project is available at `https://github.com/v0lta/Spectral-RNN`.

6.2 RELATED WORKS

In machine learning, Fourier analysis is mostly associated with signal-processing heavy domains such as speech recognition [Cha+16], biological signal processing [MNT99], medical image [VYL17] and audio-processing [DS14]. The Fourier transform has already been noted in the past for improving the computational efficiency of convolutional neural networks (CNNs) [BL+07; Pra+17]. In CNNs, the efficiency comes from the duality of convolution and multiplication in space and frequency. Fast GPU implementations of convolution, e. g.in the NVIDIA cuDNN library [Nvi20] use Fourier transforms to speed up their processing. Such gains are especially relevant for convolutional neural networks in 2D [BL+07; Pra+17] and even more so in 3D [Wan+17]. However, the improvement in computational efficiency for the STFT-RNN combination comes from reductions in data dimensionality and RNN clock rate recursion. The Fourier transform has been used in neural networks in various contexts [Hec92; ZZC08]. Particularly notable are networks, which have Fourier-like units [GG18] that decompose time series into constituent bases. Fourier-based pooling has been explored for CNNs as an alternative to standard spatial max-pooling [RYL18].

Various methods for adjusting the rate of RNN recurrency have been explored in the past. One example [Lin+96] introduces additional recurrent connections with time lags. Others apply different clock rates to different parts of the hidden layer [AHW16; Kou+14], or apply recursion at multiple (time) scales and or hierarchies [CAB17; Ser+17]. All of these approaches require changes to the basic RNN architecture. Our proposal, however, is a simple alternative which does not require adding any new structures or connections.

Among others [MBR17] approached mocap prediction using RNNs, while [Mao+19] applied a cosine transform CNN combination. In this paper we explore the combination of the complex-valued windowed STFT and RNNs on Mackey-Glass, power-load and mocap time series data.

6.3 COMPLEX SPECTRAL RECURRENT NEURAL NETWORKS

6.3.1 *Network Structure*

We can now move RNN processing into the frequency domain by applying the STFT to the input signal x. If the output or projection of the final hidden vector is also a signal of the temporal domain, we can also apply the iSTFT to recover the output y. This can be summarized by the following set of RNN equations:

$$\mathbf{X}_\tau = \mathcal{F}(\{\mathbf{x}_{S\tau-T/2}, \dots, \mathbf{x}_{S\tau+T/2}\}) \tag{6.1}$$
$$\mathbf{z}_\tau = \mathbf{W}_c\mathbf{h}_{\tau-1} + \mathbf{V}_c\mathbf{X}_\tau + \mathbf{b}_c \tag{6.2}$$
$$\mathbf{h}_\tau = f_a(\mathbf{z}_\tau) \tag{6.3}$$
$$\mathbf{y}_\tau = \mathcal{F}^{-1}(\{\mathbf{W}_{pc}\mathbf{h}_0, \dots, \mathbf{W}_{pc}\mathbf{h}_\tau\}) \tag{6.4}$$

where $\tau = [0, n_s]$, i.e. from zero to the total number of segments n_s. The output y_τ may be computed based on the available outputs $\{\mathbf{W}_p\mathbf{h}_0, \dots, \mathbf{W}_p\mathbf{h}_\tau\}$ at step τ. Adding the STFT-iSTFT pair has as two key implications. First of all, because $\mathbf{X}_\tau \in \mathbb{C}^{n_f \times 1}$ is a complex signal, the hidden state as well as subsequent weight matrices all become complex, i.e. $\mathbf{h}_\tau \in \mathbb{C}^{n_h \times 1}$, $\mathbf{W}_c \in \mathbb{C}^{n_h \times n_h}$, $\mathbf{V}_c \in \mathbb{C}^{n_h \times n_f}$, $\mathbf{b}_c \in \mathbb{C}^{n_h \times 1}$ and $\mathbf{W}_{pc} \in \mathbb{C}^{n_h \times n_f}$, where n_h is the hidden size of the networks as before and n_f is the number of frequencies in the STFT.

The second implication to note is that the step index changes from t to τ, which means that the spectral RNN effectively covers S time steps of the standard RNN per step. This has significant bearing on the overall memory consumption as well as the computational cost, both of which influence the overall network training time. Considering only the multiplication of the state matrix $\mathbf{W}_c$ and the state vector $\mathbf{h}_\tau$, which is the most expensive operation, the basic RNN requires $N \cdot O(n_h^3)$ operations for N total time steps. When using the Fourier RNN with an FFT implementation of the STFT, one requires only

$$N/S \cdot (O(T \log T) + O(n_h^3)), \tag{6.5}$$

where the $T \log T$ term comes from the FFT operation. The architectural changes lead to larger input layers and fewer RNN iterations. $\mathbf{X}$ is higher dimensional than $\mathbf{x}$, but we save on overall computation is the step size is large enough which will make N/S much smaller than N. We can generalise the approach described above into:

$$\mathbf{X}_\tau = \mathcal{F}(\{\mathbf{x}_{S\tau-T/2}, \dots, \mathbf{x}_{S\tau+T/2}\}) \tag{6.6}$$
$$\mathbf{h}_t = \mathrm{RNN}_\mathbb{C}(\mathbf{X}_\tau, \mathbf{h}_{t-1}) \tag{6.7}$$
$$\mathbf{y}_t = \mathcal{F}^{-1}(\{\mathbf{W}_{pc}\mathbf{h}_0, \dots, \mathbf{W}_{pc}\mathbf{h}_\tau\}), \tag{6.8}$$

where instead of the basic formulation outlined above, more sophisticated complex-valued RNN-architectures [**Arjovsky**; Wis+16; WY18] represented by $\mathrm{RNN}_\mathbb{C}$ may be substituted. We experiment with a complex-valued GRU, as proposed in [WY18]. An alternative to a complex approach is to concatenate the real and imaginary components into one (real-valued) hidden vector. The experimental section compares both methods in Table 6.2.

6.3.2 *Loss Functions*

In standard sequence prediction, the loss function applied is an L2 mean squared error, applied in the time domain:

$$\mathcal{L}_{mse}(\mathbf{y}_t, \mathbf{y}_{gt}) = \frac{1}{n_y} \sum_{l=0}^{n_y} (\mathbf{y}_t - \mathbf{y}_{gt})^2. \tag{6.9}$$

We experiment with a similar variation applied to the STFT coefficients applied in the frequency domain (see Table 6.2) but find that it performs on par but usually a little bit worse than the time-domain MSE. This is not surprising, as the evaluation measure applied is still in the time domain for sequence prediction so it works best to use the same function as the loss.

6.4 MACKEY-GLASS CHAOTIC SEQUENCE PREDICTION

Initially we study our method by applying it to make predictions on the Mackey-Glass series [GES02]. The Mackey-Glass equation is a

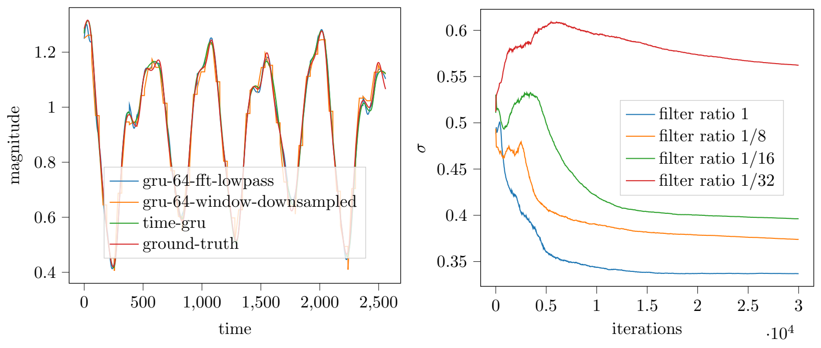


Figure 6.1: Mackey-Glass series predictions in for different RNN methods. As gradients flow through the STFT we can optimize the width of the gaussian σ. The learned window width for increasing degrees of low-pass filtering is shown here. Figure best viewed in colour.

non-linear time-delay differential and defines a chaotic system that is semi-periodic: We first evaluate different aspects of our model on the chaotic Mackey-Glass time series [GES02]:

$$\frac{dx}{dt} = \frac{\beta x_\tau}{1 + x_\tau^n} - \gamma x, \tag{6.10}$$

with $\gamma = 0.1$, $\beta = 0.2$ and the power parameter to $n = 10$. x_τ denotes the value from τ time steps ago; we use a delay of $\tau = 17$ and simulate the equation in the interval $t \in [0, 512]$, using a forward Euler scheme with a time step of size 0.1. During the warm-up, when true values are not yet known, using a uniform distribution we randomly draw values from $1 + U[-0.1, 0.1]$. An example of the time series can be found in Figure 6.1; we split the signal in half, conditioning on the first half as input and predicting the second half.

6.4.1 *Implementation Details*

In all experiments, we use RNN architectures based on real [Cho+14] and complex [WY18] GRU-cells with a state size of 64 and a Gaussian window of width 128 initialized at $\sigma = 0.5$ unless otherwise stated. The learning rate was set intially to 0.001 and then reduced with a stair-wise exponential decay with a decay rate of 0.9 every 1000 steps. Training was stopped after 30k iterations. Our run-time measurements include ode simulation, RNN execution as well as back-propagation time.

6.4.2 *Experimental Results and Ablation Studies*

Fourier transform ablation; We first compare against two time-based networks: a standard GRU (time-GRU) and a windowed version (time-GRU-windowed) in which we reshape the input and process windows of data together instead of single scalars. This effectively sets the clock

Table 6.1: Short time Fourier, Windowed and Time Domain results obtained using GRU cells of size 64. Windowed experiments process multiple samples of data without computing the STFT. Additionally we compare low-pass filtering the spectrum and downsampling the time domain windows. All models where trained for 30k iterations. We downsample and lowpass-filter with a factor of 1/32.

net	weights	mse	training-time [min]
time-GRU	13k	$3.8 \cdot 10^{-4}$	355
time-GRU-window	29k	$6.9 \cdot 10^{-4}$	53
time-GRU-window-down	13k	$12 \cdot 10^{-4}$	48
STFT-GRU	46k	$3.5 \cdot 10^{-4}$	57
STFT-GRU-lowpass	14k	$2.7 \cdot 10^{-4}$	56

Table 6.2: Real and complex valued architecture comparison on the mackey-glass data, with increasing complex cell size. The complex architectures take longer to run but are more parameter efficient. The last row shows a complex RNN cell in STFT space without iSTFT backpropagation.

net	weights	mse	training-time [min]
STFT-GRU-64	46k	$3.5 \cdot 10^{-4}$	57
STFT-cgRNN-32	23k	$2.1 \cdot 10^{-4}$	63
STFT-cgRNN-54	46k	$1.6 \cdot 10^{-4}$	63
STFT-cgRNN-64	58k	$1.1 \cdot 10^{-4}$	64
STFT-cgRNN-64-$\mathcal{L}_{\mathcal{C}}$	58k	$210 \cdot 10^{-4}$	64

rate of the RNN to be per window rather than per time step. As comparison, we look at the STFT-GRU combination as described in Section 6.3.1 with a GRU-cell and a low-pass filtered version keeping only the first four coefficients (STFT-GRU-lowpass). Additionally we compare lopass filtering to time windowed time domain downsampling (time-GRU-window-down). For all five networks, we use a fixed hidden-state size of 64. From Figure 6.1, we observe that all five architecture variants are able to predict the overall time series trajectory. Results in Table 6.1 indicate that reducing the RNN clock rate through windowing and parameters trough low pass filtering also improves prediction quality. In comparison to time domain downsampling, frequency-domain lowpass filtering allows us to reduce parameters more aggressively.

Runtime; As discussed in Equation 6.5, windowing reduces the computational load significantly. In Table 6.1, we see that the windowed experiments run much faster than the naive approach. The STFT networks are slightly slower, since the Fourier Transform adds a small amount of computational overhead.

The window function Figure 6.1 shows the Gaussian width for multiple filtering scenarios. We observe a gradient signal on the window function. For more aggressive filtering the optimizer chooses a wider kernel.

Complex vs. real cell; We explore the effect of a complex valued cell in Table 6.2. We apply the complex GRU proposed in [WY18], and compare to a real valued GRU. We speculate that complex cells are better suited to process Fourier representations due to their complex nature. Our observations show that both cells solve the task, while the complex cell does so with fewer parameters at a higher computational cost. The increased parameter efficiency of the complex cell could indicate that complex arithmetic is better suited than real arithmetic four Frequency domain machine learning. However due to the increased run-time we proceed using the concatenation approach.

Time vs. Frequency mse; One may argue that propagating gradients through the iSTFT is unnecessary. We tested this setting and show a result in the bottom row of Table 6.2. In comparison to the accuracy of the otherwise equivalent complex network shown above the second horizontal line, computing the error on the fourier coefficients performs significantly worse. We conclude that if our metric is based in the time domain the loss needs to be there as well and therefore require gradient propagation through the STFT.

6.5 POWER LOAD FORECASTING

6.5.1 *Data*

We apply our Fourier RNN to the problem of forecasting power consumption. We use the power load data of 36 EU countries from 2011 to 2019 as provided by the European Network of Transmission System Operators for Electricity[1]. The data is partitioned into two groups; countries reporting with a 15 minute frequency are used for day-ahead predictions, while those with hourly reports are used for longer-term predictions. For testing we hold back the German Tennet load recordings from 2015, all of Belgium's recordings of 2016, Austria's load of 2017 and finally the consumption of the German Ampiron grid in 2018.

6.5.2 *Day-Ahead Prediction*

We start with the task of day-ahead prediction; using 14 days of context, at 12:00, we are asked to predict 24 hours of load from midnight onwards (00:00 until 24:00 o' clock at 12:00 o'clock on the previous day). We therefore forecast the load from noon until midnight on the prediction day plus the next day and ignore the values from

1 `https://transparency.entsoe.eu/`

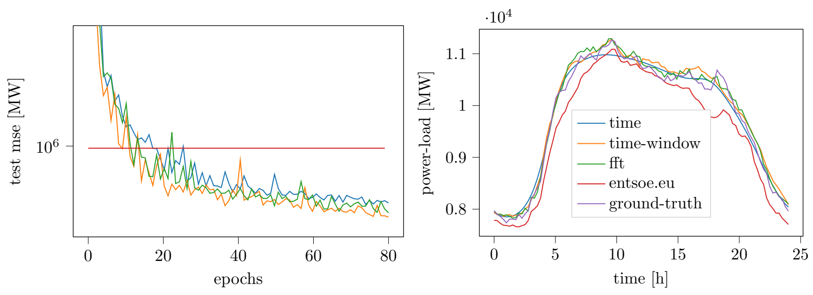


Figure 6.2: Day ahead prediction results convergence (left) and prediction examples (right). We observe that all deep learning approaches beat the entsoe.eu baseline shown as the red line, which suggests that their approach could benefit from deep learning.

Network	mse $[MW]^2$	weights	inference [ms]	train [min]
time	$261 \cdot 10^5$	13k	1360	1472
time-win	$8.12 \cdot 10^5$	74k	9.25	15
time-win-down	$8.05 \cdot 10^5$	28k	8.2	15
STFT	$7.62 \cdot 10^5$	136k	9.67	19
STFT-lowpass	$7.25 \cdot 10^5$	43k	9.69	18

Table 6.3: 60 day ahead power load prediction using GRUs of size 64. We downsample and lowpass-filter with a factor of 1/4. We observe that windowing leads to large training and inference speed-ups. Our STFT approach performs better in the full spectrum case and with a reduced input-dimensionality.

the prediction day. We use the same network architecture as in the previous section. During training, the initial learning rate was set to 0.004 and exponentially decayed using a decay of 0.95 every epoch. We train for 80 epochs overall. We compare time domain, windowed time as well as windowed Fourier approaches. The window size was set to 96 which corresponds to 24 hours of data at a sampling rate of 15 minutes per sample.

In Figure 6.2 we observe that all approaches we tried produce reasonable solutions and outperform the prediction produced by the European Network of Transmission System Operators for Electricity which suggests that their approach could benefit from deep learning.

6.5.3 *Long-Term Forecast*

Next we consider the more challenging task of 60 day load prediction using 120 days of context. We use 12 load samples per day from all

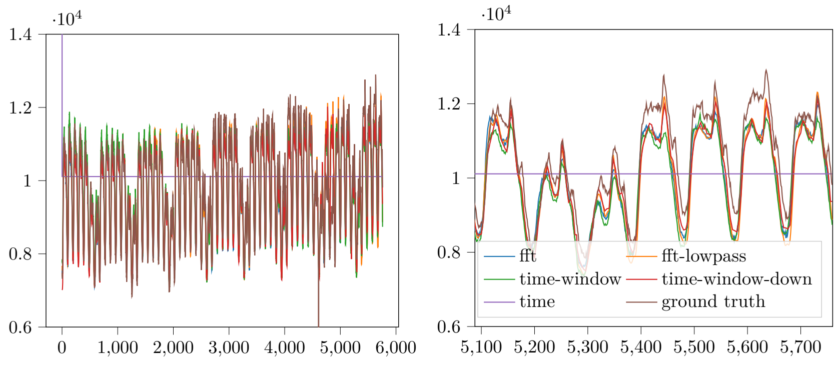


Figure 6.3: A test set sample for showing the 60 day prediction results for all architectures under consideration. Close up for the last week of the 60 day prediction.

entenso-e member states from 2015 until 2019. We choose a window size of 120 samples or five days; all other parameters are left the same as the day-ahead prediction setting. Table 6.3 shows that the windowed methods are able to extract patterns, but the scalar-time domain approach failed. Additionally we observe that we do not require the full set of Fourier coefficients to construct useful predictions on the power-data set. The results are tabulated in Table 6.3. It turns out that the lower quarter of Fourier coefficients is enough, which allowed us to reduce the number of parameters considerably. Again we observe that Fourier low-pass filtering outperforms down-sampling and windowing.

6.6 human motion forecasting

6.6.1 *Dataset & Evaluation Measure*

The Human3.6M data set [Ion+14] contains 3.6 million human poses of seven human actors each performing 15 different tasks sampled at 50 Herz. As in previous work [MBR17] we test on the data from actor 5 and train on all others. We work in Cartesian coordinates and predict the absolute 3D-position of 17 joints per frame, which means that we model the skeleton joint movements as well as global translation.

6.6.2 *Implementation Details*

We use standard GRU-cells with a state size of 3072. We move the data into the frequency domain by computing a short time Fourier transform over each joint dimension separately using a window size of 24. The learning rate was set initially to 0.001 which we reduced using an exponential stair wise decay every thousand iterations by a factor of 0.97. Training was stopped after 19k iterations.

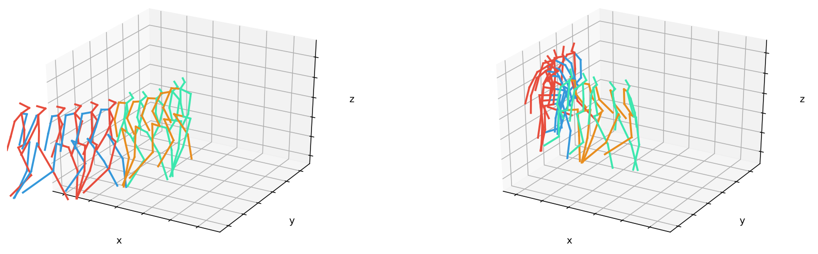


Figure 6.4: Visualization of input and prediction results using a STFT-RNN combination and low pass filtering. Input is shown in red and blue, predictions in green and yellow.

6.6.3 *Motion Forecasting Results*

Prediction results using our STFT approach are shown in Figure 6.4, poses drawn in green and yello are predictions, conditioned on the pose sequences set in blue and red. We observe predictions look realistic and smooth. In our experiments low-pass filtering helped us to enforce smoothness in the predictions. Quantitative motion capture prediction results are shown in Table 6.4. We observe that all windowed approaches run approximately five times faster than the time domain approach during inference, a significant improvement. In terms of accuracy windowing does comparably well, while the STFT approach does better when the input sampling rate is reduced.

6.7 SUMMARY AND OUTLOOK

In this paper we explored frequency domain machine learning using a recurrent neural network. We have proposed to integrate the Short Time Fourier transform and the inverse transfrom into RNN architectures and evaluated the performance of real and complex valued cells in this setting, we found that complex cells are more parameter efficient, but run slighly longer. Frequency domain RNNs allow us to learn window function parameters and make high-frequency time sequence predictions for both synthetic and real-world data while using less computation time and memory. Low-pass filtering reduced network parameters and outperformed time-domain down-sampling in our experiments.

Network	mae [mm]	mse $[\text{mm}]^2$	weights	inference [ms]	train [min]
time [MBR17]	75.44	$1.45 \cdot 10^4$	29M	115	129
time-win	68.25	$1.47 \cdot 10^4$	33M	18	27
time-win-down	70.22	$1.41 \cdot 10^4$	30M	18	27
STFT	67.88	$1.25 \cdot 10^4$	45M	25	38
STFT-lowpass	66.71	$1.30 \cdot 10^4$	32M	20	31

Table 6.4: 3d-Human motion forecast of 64 frames or approximately one second. Mean absolute error (mae) is measured in mm. Mean squared errors are reported in mm^2. We downsample and lowpass-filter with a factor of 1/4. Windowing runs much faster than the naive time domain approach. Among windowed approaches the STFT allows more aggressive input size reductions.

7 COMPLEX GATED RECURRENT NEURAL NETWORKS [WY18]

7.1 INTRODUCTION

This chapter corresponds to: [WY18] Moritz Wolter and Angela Yao. Complex Gated Recurrent Neural Networks. In:NeurIPS. 2018.

Recurrent neural networks (RNNs) are widely used for processing time series and sequential information. The difficulties of training RNNs, especially when trying to learn long-term dependencies, are well-established, as RNNs are prone to vanishing and exploding gradients [BSF94; HS97a; PMB13]. Heuristics developed to alleviate some of the optimization instabilities and learning difficulties include gradient clipping [GBC16; Mik12], gating [Cho+14; HS97a], and using norm-preserving state transition matrices [ASB16; HR17; Jin+18; Wis+16].

Gating, as used in gated recurrent units (GRUs) [Cho+14] and long short-term memory (LSTM) networks [HS97a], has become commonplace in recurrent architectures. Gates facilitate the learning of longer-term temporal relationships [HS97a]. Furthermore, in the presence of noise in the input signal, gates can protect the cell state from undesired updates, thereby improving overall stability and convergence.

A matrix $\mathbf{W}$ is norm-preserving if its repeated multiplication with a vector leaves the vector norm unchanged, i. e.$\|\mathbf{W}h\|_2 = \|h\|_2$. Norm-preserving state transition matrices are particularly interesting for RNNs because they preserve gradients over time [ASB16], thereby preventing both the vanishing and exploding gradient problem. To be norm-preserving, state transition matrices need to be either orthogonal or unitary[1]. Complex numbers have long been favored for signal processing [Hir13; LA08; MG09]. A complex signal does not simply double the dimensionality of the signal. Instead, the representational richness of complex signals is rooted in its physical relevance and the mathematical theory of complex analysis. Complex arithmetic, and in particular multiplication, is different from its real counterpart and allows us to construct novel network architectures with several desirable properties. Despite networks being complex-valued, however, it is often necessary to work with real-valued cost functions and/or existing real-valued network components. Map-

1 Unitary matrices are the complex analogue of orthogonal matrices, i. e. a complex matrix $\mathbf{W}$ is unitary if $\mathbf{W}\overline{\mathbf{W}}^T = \overline{\mathbf{W}}^T\mathbf{W} = I$, where $\overline{\mathbf{W}}^T$ is its conjugate transpose and $\mathbf{I}$ is the identity matrix.

pings from $\mathbb{C} \to \mathbb{R}$ are therefore indispensable. Unfortunately such functions violate the Cauchy-Riemann equations and are not complex-differentiable in the traditional sense. We advocate the use of Wirtinger calculus [Wir27] (also known as CR-calculus [Kre09]), which makes it possible to define complex (partial) derivatives, even when working with non-holomorph or non-analytic functions.

Complex-valued representations have begun receiving some attention in the the deep learning community but they have been applied only to the most basic of architectures [ASB16; Gub16; Tra+18]. For recurrent networks, complex representations could gain more acceptance if they were shown to be compatible with more commonly used gated architectures and also competitive for real-world data. This is exactly the aim of this work, where we propose a complex-valued gated recurrent network and show how it can easily be implemented with a standard deep learning library such as TensorFlow. Our contributions can be summarized as follows:

- We introduce a novel complex-gated recurrent unit; to the best of our knowledge, we are the first to explore such a structure using complex number representations.
- We compare experimentally the effects of a bounded versus unbounded non-linearity in recurrent networks, finding additional evidence countering the commonly held heuristic that only bounded non-linearities should be applied in RNNs. In our case unbounded non-linearities perform better, but must be coupled with the stabilizing measure of using norm-preserving state transition matrices.
- Our complex gated network is stable and fast to train; it outperforms the state of the art with equal parameters on synthetic tasks and delivers state-of-the-art performance one the real-world application of predicting poses in human motion capture using fewer weights.

For reproduction purposes source code is available at `https://github.com/v0lta/Complex-gated-recurrent-neural-networks`.

7.2 RELATED WORK

The current body of literature in deep learning focuses predominantly on real-valued neural networks. Theory for learning with complex-valued data, however, was established long before the breakthroughs of deep learning. This includes the development of complex non-linearities and activation functions [GK92; KA01], the computation of complex gradients and Hessians [Van94], and complex backpropagation [BP92; LH91].

Complex-valued representations were first used in deep networks to model phase dependencies for more biologically plausible neurons [RS14] and to augment the memory of LSTMs [Dan+16a], i. e. whereby half of the cell state is interpreted as the imaginary component. In contrast, true complex-valued networks (including this work) have not only complex valued states but also kernels. Recently, complex CNNs have been proposed as an alternative for classifying natural images [Gub16; Tra+18] and inverse mapping of MRI signals [VYL17]. Complex CNNs were found to be competitive or better than state-of-the-art [Tra+18] and significantly less prone to over-fitting [Gub16].

For temporal sequences, complex-valued RNNs have also been explored [ASB16; HR17; Jin+17; Wis+16], though interest in complex representations stems from improved learning stability. In [ASB16], norm-preserving state transition matrices are used to prevent vanishing and exploding gradients. Since it is difficult to parameterize real-valued orthogonal weights, [ASB16] recommends shifting to the complex domain, resulting in a unitary RNN (uRNN). The weights of the uRNN in [ASB16], for computational efficiency, are constructed as a product of component unitary matrices. As such, they span only a reduced subset of unitary matrices and do not have the expressiveness of the full set. Alternative methods of parameterizing the unitary matrices have been explored [HR17; Jin+17; Wis+16]. Our proposed cgRNN builds on these works in that we also use unitary state transition matrices. In particular, we adopt the parameterization of [Wis+16] in which weights are parameterized by full-dimensional unitary matrices, though any of the other parameterizations [ASB16; HR17; Jin+17] can also be substituted.

7.3 PRELIMINARIES

We represent a complex number $z \in \mathbb{C}$ as $z = x + ib$, where $x = \Re(z)$ and $y = \Im(z)$ are the real and imaginary parts respectively. The complex conjugate of z is $\bar{z} = x - iy$. In polar coordinates, z can be expressed as $z = |z|e^{i\theta_z}$, where $|z|$ and θ are the magnitude and phase respectively and $\theta_z = \text{atan2}(x, y)$. Note that $z_1 \cdot z_2 = |z_1||z_2|e^{i(\theta_1+\theta_2)}$, $z_1 + z_2 = x_1 + x_2 + i(y_1 + y_2)$ and $s \cdot z = s \cdot re^{i\theta}, s \in \mathbb{R}$. The expression $s \cdot z$ scales z's magnitude, while leaving the phase intact.

7.3.1 *Complex Gradients*

A complex-valued function $f : \mathbb{C} \to \mathbb{C}$ can be expressed as $f(z) = u(x, y) + iv(x, y)$ where $u(\cdot, \cdot)$ and $v(\cdot, \cdot)$ are two real-valued functions. The complex derivative of $f(z)$, or the $\mathbb{C}$-derivative, is defined if and only if f is *holomorph*. In such a case, the partial derivatives of u and v must not only exist but also satisfy the Cauchy-Riemann equations, where $\partial u/\partial x = \partial v/\partial y$ and $\partial v/\partial x = -\partial u/\partial y$.

Strict holomorphy can be overly stringent for deep learning purposes. In fact, Liouville's theorem [Lio79] states that the only complex function which is both holomorph and bounded is a constant function. This implies that for complex (activation) functions, one must trade off either boundedness or differentiability. One can forgo holomorphy and still leverage the theoretical framework of Wirtinger or CR-calculus [Kre09; MG09] to work separately with the $\mathbb{R}$- and $\overline{\mathbb{R}}$-derivatives[2]:

$$\mathbb{R}\text{-derivative} \triangleq \frac{\partial f}{\partial z}|_{\bar{z}=\text{const}} = \frac{1}{2}(\frac{\partial f}{\partial x} - i\frac{\partial f}{\partial y}), \tag{7.1}$$

$$\overline{\mathbb{R}}\text{-derivative} \triangleq \frac{\partial f}{\partial \bar{z}}|_{z=\text{const}} = \frac{1}{2}(\frac{\partial f}{\partial x} + i\frac{\partial f}{\partial y}). \tag{7.2}$$

Based on these derivatives, one can define the chain rule for a function $g(f(z))$ as follows:

$$\frac{\partial g(f(z))}{\partial z} = \frac{\partial g}{\partial f}\frac{\partial f}{\partial z} + \frac{\partial g}{\partial \bar{f}}\frac{\partial \bar{f}}{\partial z} \qquad \text{where} \qquad \bar{f} = u(x,y) - iv(x,y). \tag{7.3}$$

Since mappings from $\mathbb{C} \to \mathbb{R}$ can generally be expressed in terms of the complex variable z and its conjugate $\bar{z}$, the Wirtinger-Calculus allows us to formulate and theoretically understand the gradient of real-valued loss functions in an easy yet principled way.

7.3.2 *A Split Complex Approach*

We work with a split-complex approach, where real-valued non-linear activations are applied separately to the real and imaginary parts of the complex number. This makes it convenient for implementation, since standard deep learning libraries are not designed to work natively with complex representations. Instead, we store complex numbers as two real-valued components. Split-complex activation functions process either the magnitude and phase, or the real and imaginary components with two real-valued nonlinear functions and then recombine the two into a new complex quantity. While some may argue this reduces the utility of having complex representations, we prefer this to fully complex activations. Fully complex non-linearities do exist and may seem favorable [Tra+18], since one needs to keep track of only the $\mathbb{R}$ derivatives, but due to Liouville's theorem, we must forgo boundedness and then deal with forward pass instabilities.

2 For holomorph functions the $\overline{\mathbb{R}}$-derivative is zero and the $\mathbb{C}$- derivative is equal to the $\mathbb{R}$-derivative.

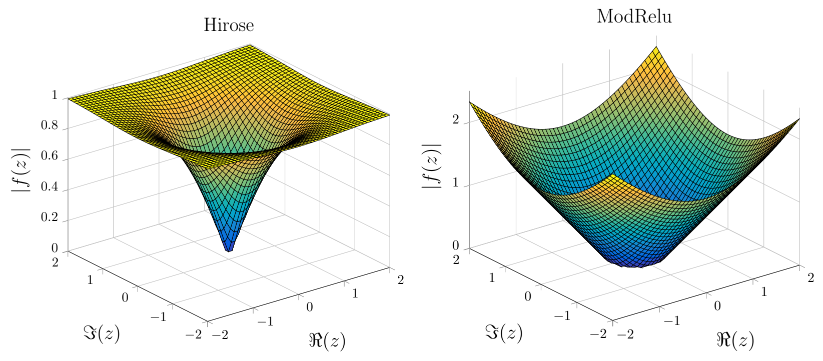


Figure 7.1: Surface plots of the magnitude of the Hirose ($m^2 = 1$) and modReLU ($b = -0.5$) activations.

7.4 COMPLEX GATED RNNS

7.4.1 *Basic Complex RNN Formulation*

Without any assumptions on real versus complex representations, we define a basic RNN as follows:

$$\mathbf{z}_t = \mathbf{W}\mathbf{h}_{t-1} + \mathbf{V}\mathbf{x}_t + \mathbf{b} \tag{7.4}$$

$$\mathbf{h}_t = f_a(\mathbf{z}_t) \tag{7.5}$$

where $\mathbf{x}_t$ and $\mathbf{h}_t$ represent the input and hidden unit vectors at time t. f_a is a point-wise non-linear activation function, and W and V are the hidden and input state transition matrices respectively. In working with complex networks, $\mathbf{x}_t \in \mathbb{C}^{n_x \times 1}$, $\mathbf{h}_t \in \mathbb{C}^{n_h \times 1}$, $\mathbf{W} \in \mathbb{C}^{n_h \times n_h}$, $\mathbf{V} \in \mathbb{C}^{n_h \times n_x}$ and $\mathbf{b} \in \mathbb{C}^{n_h \times 1}$, where n_x and n_h are the dimensionalities of the input and hidden states respectively.

7.4.2 *Complex Non-linear Activation Functions*

Choosing a non-linear activation function f_a for complex networks can be non-trivial. Though holomorph non-linearities using transcendental functions have also been explored in the literature [MG09], the presence of singularities makes them difficult to learn in a stable manner. Instead, bounded non-holomorph non-linearities tend to be favoured [Hir13; MG09], where bounded real-valued non-linearities are applied on the real and imaginary part separately. This also parallels the convention of using (bounded) tanh non-linearities in real RNNs.

A common split is with respect to the magnitude and phase. This non-linearity was popularized by Hirose [Hir13] and scales the magnitude by a factor m^2 before passing it through a tanh:

$$f_{\text{Hirose}}(z) = \tanh\left(\frac{|z|}{m^2}\right) e^{-i\cdot\theta_z} = \tanh\left(\frac{|z|}{m^2}\right) \frac{z}{|z|}. \tag{7.6}$$

In other areas of deep learning, the rectified linear unit (ReLU) is now the go-to non-linearity. In comparison to sigmoid or tanh activations, they are computationally cheap, expedite convergence [KSH12a] and also perform better [NH10; MHN13; Zei+13]. However, there is no direct extension into the complex domain, and as such, modified versions have been proposed [Gub16; VYL17]. The most popular is the modReLU [ASB16] – a variation of the Hirose non-linearity, where the tanh is replaced with a ReLU and b is an offset:

$$f_{\text{modReLU}}(z) = \text{ReLU}(|z| + b)e^{-i \cdot \theta_z} = \text{ReLU}(|z| + b)\frac{z}{|z|}. \tag{7.7}$$

7.4.3 *Real to Complex input and Complex to Real output mappings*

While several time series problems are inherently complex, especially when considering their Fourier representations, the majority of benchmark problems in machine learning are still only defined in the real number domain. However, one can still solve these problems with complex representations, since a real z has simply a zero imaginary component, i. e.$\mathfrak{I}(z) = 0$ and $z = x + i \cdot 0$.

To map the complex state $\mathbf{h}$ into a real output $\mathbf{o}_r$, we use a linear combination of the real and imaginary components, similar to [ASB16], with $\mathbf{W}_o$ and $\mathbf{b}_o$ as weights and offset:

$$\mathbf{o}_r = \mathbf{W}_o[\mathfrak{R}(\mathbf{h})\ \mathfrak{I}(\mathbf{h})] + \mathbf{b}_o. \tag{7.8}$$

7.4.4 *Optimization on the Stiefel Manifold for Norm Preservation*

In [ASB16], it was proven that a unitary [3] $\mathbf{W}$ would prevent vanishing and exploding gradients of the cost function C with respect to h_t, since the gradient magnitude is bounded. However, this proof hinges on the assumption that the derivative of f_a is also unity. This assumption is valid if the pre-activations are real and one chooses the ReLU as the non-linearity. For complex pre-activations, however, this is no longer a valid assumption. Neither the Hirose non-linearity (Equation 7.6) nor the modReLU (Equation 7.7) can guarantee stability (despite the suggestion otherwise in the original proof [ASB16]).

Even though it is not possible to guarantee stability, we strongly advocate using norm-preserving state transition matrices, since they do still have excellent stabilizing effects. This was proven experimentally in [ASB16; HR17; Wis+16] and we find similar evidence in our own experiments (see Figure 7.2). Ensuring that $\mathbf{W}$ remains unitary during the optimization can be challenging, especially since the group of unitary matrices is not closed under addition. As such, it is not possible to learn $\mathbf{W}$ with standard update-based gradient descent.

3 Since $\mathbb{R} \subseteq \mathbb{C}$, we use the term unitary to refer to both real orthogonal and complex unitary matrices and make a distinction for clarity purposes only where necessary.

Alternatively, one can learn $\mathbf{W}$ on the Stiefel manifold [Wis+16], with the $k+1$ update $\mathbf{W}_{k+1}$ given as follows by [Tag11], where λ is the learning rate, $\mathbf{I}$ the identity matrix, and F the cost function:

$$\mathbf{W}_{k+1} = (\mathbf{I} + \frac{\lambda}{2}\mathbf{A}_k)^{-1}(\mathbf{I} - \frac{\lambda}{2}\mathbf{A}_k)\mathbf{W}_k \tag{7.9}$$

$$\text{where} \qquad \mathbf{A} = \mathbf{W}\overline{\nabla_{\mathbf{w}}F}^T - \overline{\mathbf{W}}^T\nabla_{\mathbf{w}}F. \tag{7.10}$$

7.4.5 *Complex-Valued Gating Units*

In keeping with the spirit that gates determine the amount of a signal to pass, we construct a complex gate as a $\mathbb{C}^{n_h \times n_h} \to \mathbb{R}^{n_h \times 1}$ mapping. Like in real gated RNNs, the gate is applied as an element-wise product, i. e. $\mathbf{g} \odot \mathbf{h} = \mathbf{g} \odot |\mathbf{h}|e^{i\theta_h}$. In our complex case, this type of operation results in an element-wise scaling of the hidden state's magnitude. When the gate is 0, it completely resets a signal, whereas if it is 1, then it ensures that the signal is passed entirely. We introduce our gates into the RNN in a similar fashion as the classic GRU [Cho+14]:

$$\widetilde{\mathbf{z}}_t = \mathbf{W}(\mathbf{g}_r \odot \mathbf{h}_{t-1}) + \mathbf{V}\mathbf{x}_t + \mathbf{b}, \tag{7.11}$$

$$\mathbf{h}_t = \mathbf{g}_z \odot f_a(\widetilde{\mathbf{z}}_t) + (1 - \mathbf{g}_z) \odot \mathbf{h}_{t-1}, \tag{7.12}$$

where $\mathbf{g}_r$ and $\mathbf{g}_z$ represent reset and update gates respectively and are defined with corresponding subscripts r and z as

$$\mathbf{g}_r = f_g(\mathbf{z}_r), \qquad \text{where} \qquad \mathbf{z}_r = \mathbf{W}_r\mathbf{h} + \mathbf{V}_r\mathbf{x}_t + \mathbf{b}_r, \tag{7.13}$$

$$\mathbf{g}_z = f_g(\mathbf{z}_z), \qquad \text{where} \qquad \mathbf{z}_z = \mathbf{W}_z\mathbf{h} + \mathbf{V}_z\mathbf{x}_t + \mathbf{b}_z. \tag{7.14}$$

Above, f_g denotes the gate activation, $\mathbf{W}_r \in \mathbb{C}^{n_h \times n_h}$ and $\mathbf{W}_z \in \mathbb{C}^{n_h \times n_h}$ denote state to state transition matrices, $\mathbf{V}_r \in \mathbb{C}^{n_h \times n_i}$ and $\mathbf{V}_z \in \mathbb{C}^{n_h \times n_i}$ the input to state transition matrices, and $\mathbf{b}_r \in \mathbb{C}^{n_h}$ and $\mathbf{b}_z \in \mathbb{C}^{n_h}$ the biases. f_g is a non-linear gate activation function defined as:

$$f_{\text{mod sigmoid}}(\mathbf{z}) = \sigma(\alpha\mathfrak{R}(\mathbf{z}) + \beta\mathfrak{I}(\mathbf{z})), \qquad \alpha, \beta \in [0, 1]. \tag{7.15}$$

We call this the modSigmoid and justify the choice experimentally in section 7.5.3.

As mentioned previously, even with unitary state transition matrices, this type of gating is not mathematically guaranteed to be stable. However, the effects of vanishing gradients are mitigated by the fact that the derivatives are distributed over a sum [HS97a; Cho+14]. Exploding gradients are clipped.

7.5 EXPERIMENTATION

7.5.1 *Tasks & Evaluation Metrics*

We test our cgRNN on two benchmark synthetic tasks: the memory problem and the adding problem [HS97a]. These problems are designed especially to challenge RNNs, and require the networks to store information over time scales on the order of hundreds of time steps. The first is the **memory problem**, where the RNN should remember n input symbols over a time period of length $T + 2n$ based on a dictionary set $\{s^1, s^2, ..., s^n, s^b, s^d\}$, where s^1 to s^n are symbols to memorize and s^b and s^i are blank and delimiter symbols respectively. The input sequence, of length $T + 2n$, is composed of n symbols drawn randomly with replacement from $\{s^1, ..., s^n\}$, followed by $T - 1$ repetitions of s^b, s^d, and another n repetitions of s^b. The objective of the RNN, after being presented with the initial n symbols, is to generate an output sequence of length $T + S$, with T repetitions of s^b, and upon seeing s^d, recall the original n input symbols. A network without memory would output s^b and once presented with s^d, randomly predict any of the original n symbols; this results in a categorical cross entropy of ${}^{(n+1\log(8))}/_{(T+2(n+2))}$. For our experiments, we choose $n = 8$ and $T = 250$.

In the **adding problem**, two sequences of length T are given as input, where the first sequence consists of numbers randomly sampled from $\mathcal{U}[0, 1]$ [4], while the second is an indicator sequence of all $0'$s and exactly two $1'$s, with the first 1 placed randomly in the first half of the sequence and the second 1 randomly in the second half. The objective of the RNN is to predict the sum of the two entries of the first input sequence once the second 1 is presented in the indicator input sequence. A naive baseline would predict 1 at every time step, regardless of the input indicator sequence's value; this produces an mean squared error (MSE) of 0.167, i. e. the variance of the sum of two independent uniform distributions. For our experiments, we choose $T = 250$.

We apply the cgRNN to the real-world task of **human motion prediction**, i. e. predicting future 3D poses of a person given the past motion sequence. This task is of interest to diverse areas of research, including 3D tracking in computer vision [YGV12], motion synthesis for graphics [KGP02] as well as pose and action predictions for assistive robotics [KS16]. We follow the same experimental setting as [MBR17], working with the full Human 3.6M dataset [Ion+14]. For training, we use six of the seven actors and test on actor five. We use the pre-processed data of [Jai+16], which converts the motion capture into exponential map representations of each joint. Based on an input sequence of body poses from 50 frames, the future 10 frames are predicted. This is equivalent of predicting 400ms. The error is measured by the euclidean distance in Euler angles with respect to the ground truth poses.

4 Note that this is a variant of [HS97a]'s original adding problem, which draws numbers from $\mathcal{U}[-1, 1]$ and used three indicators $\{-1, 0, 1\}$. Our variant is consistent with state-of-the-art [ASB16; HR17; Wis+16]

We also test the cgRNN on native complex data drawn from the frequency domain by testing it on the real world task of **music transcription**. Given a music wave form file, the network should determine the notes of each instrument. We use the Music-Net dataset [THK17], which consists of 330 classical music recordings, of which 327 are used for training and 3 are held out for testing. Each recording, sampled at 11kHz, is divided into segments of 2048 samples with a step size of 512 samples. The transcription problem is defined as a multi-label classification problem, where for each segment, a label vector $y \in 0, 1^{128}$ describing the active keys in the corresponding midi file has to be found. We use the windowed Fourier-transform of each segment as network input, the real and imaginary parts of the Fourier transform, i. e.the odd and even components respectively, are used directly as inputs into the cgRNN.

7.5.2 *RNN Implementation Details*

We work in Tensorflow, using RMS-prop to update standard weights and the multiplicative Stiefel-manifold update as described Equation 7.10 for all unitary state transition matrices. The unitary state transition matrices are initialized the same as [ASB16] as the product of component unitary matrices. All other weights are initialized using the uniform initialisation method recommended in [GB10], i. e. $U[-l, l]$ with $l = \sqrt{6/(n_{in} + n_{out})}$, where n_{in} and n_{out} are the input and output dimensions of the tensor to be initialised. All biases are intialized as zero, with the exception of the gate biases $\mathbf{b}_r$ and $\mathbf{b}_z$, which are initialized at 4 to ensure fully open gates and linear behaviour at the start of training. All synthetic tasks are run for $2 \cdot 10^4$ iterations with a batch-size of 50 and a constant learning rate of 0.001 for both the RMS-Prop and the Stiefel-Manifold updates.

For the human motion prediction task, we adopt the state-of-the-art implementation of [MBR17], which introduces residual velocity connections into the standard GRU. Our setup shares these modifications; we simply replace their core GRU cell with our cgRNN cell. The learning rate and batch size are kept the same (0.005, 16) though we reduce our state size to 512 to be compatible with [MBR17]'s 1024[5]. For music transcription, we work with a bidirectional cgRNN encoder followed by a simple cgRNN decoder. All cells are set with $n_h = 1024$; the learning rate is set to 0.0001 and batch size to 5.

5 This reduction is larger than necessary – parameter-wise, the equivalent state size is $\sqrt{\frac{1024^2}{2}} \approx 724$

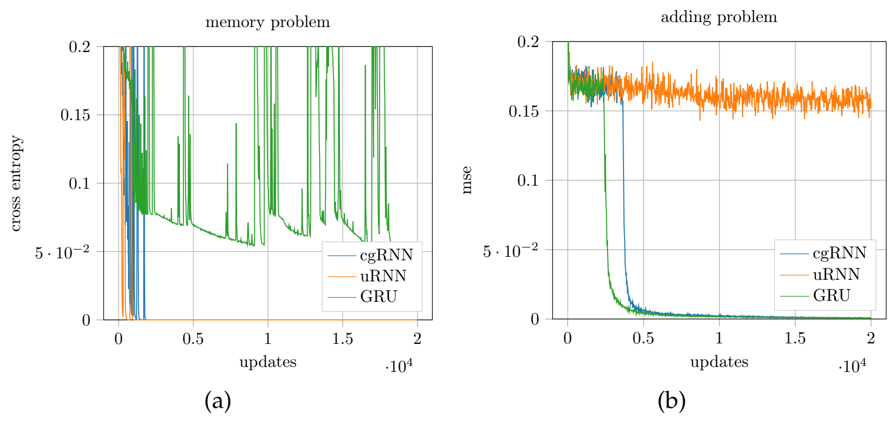


Figure 7.2: Comparison of our cgRNN (blue, $n_h = 80$) with the uRNN [ASB16] (orange, $n_h = 140$) and standard GRU [Cho+14] (green, $n_h = 112$) on the memory (a) and adding (b) problem for $T = 250$. The hidden state size n_h for each network are chosen so as to approximately match the number of parameters (approximately 44k parameters total). On the memory problem, having norm-preserving state transition matrices is critical for stable learning, while on the adding problem, having gates is important.

7.5.3 *Impact of Gating and Choice of Gating Functions*

We first analyse the impact that gating has on the synthetic tasks by comparing our cgRNN with the gateless uRNN from [ASB16]. Both networks use complex representations and also unitary state transition matrices. As additional baselines, we also compare with TensorFlow's out-of-the-box GRU. We choose the hidden state size n_h of each network to ensure that the resulting number of parameters is approximately equivalent (around 44k). We find that our cgRNN successfully solves both the memory problem as well as the adding problem. On the memory problem (see Figure 7.2(a), Table 7.1), gating does not play a role. Instead, having norm-preserving weight matrices is key to ensure stability during the learning. The GRU, which does not have norm-preserving state matrices, is highly unstable and fails to solve the problem. Our cgRNN achieves very similar performance as the uRNN. This has to do with the fact that we initialize our gate bias term to be fully open, i. e. $\mathbf{g}_r = \mathbf{1}$, $\mathbf{g}_z = \mathbf{1}$. Under this setting, the formulation is the same as the uRNN, and the unitary $\mathbf{W}$ dominates the cell's dynamics.

For the adding problem, previous works [ASB16; HR17; Wis+16] have suggested that gates are beneficial and we confirm this result in Figure 7.2(b) and Table 7.1. We speculate that the advantage comes from the gates shielding the network from the irrelevant inputs of the adding problem, hence the success of our cgRNN as well as the GRU, but not the uRNN. Surprisingly, the standard GRU baseline, without any norm-preserving state transition matrices works very well on the adding problem; in fact, it marginally outperforms our cgRNN. However, we believe this result does not speak to the inferiority of complex representations; instead, it is likely that the adding problem, as a synthetic task, is not able to leverage the advantages offered by the representation.

The gating function (Equation 7.15) was selected experimentally based on a systematic comparison of various functions. The performance of different gate functions are compared statistically in Table 7.1, where we look at the fraction of converged experiments over 20 runs as well as the mean number of iterations required until convergence. The product as well as the tied and free weighted sum variations of the gating function are designed to resemble the bilinear gating mechanism used in [Geh+17]. From our experiments, we find that it is important to scale the real and imaginary components before passing through the sigmoid to leverage the saturation constraint, and that the real and imaginary components should be combined linearly. The exact weighting seems not to be important and the best performing variants are the *tied 2* and the *free*; to preserve generality, we advocate the use of the *free* variant. We note that over 20 runs, our cgRNN converged only on 15-16 runs; adding the gates introduces instabilities, however, we find the ability to solve the adding problem a reasonable trade-off.

Finally, we compare the cgRNN to a *free real* variant (see last row of Table 7.1), which is the most similar architecture in $\mathbb{R}$, i. e.normalized hidden transition matrices, same gate formulation, and two independently real-valued versions of Equations 7.13 and 7.14. This real variant has similar performance on the adding problem (for which having gates is critical), but cannot solve the memory problem. This is likely due to the set of real orthogonal matrices being too restrictive, making the problem more difficult in the real domain than the complex.

7.5.4 *Non-Linearity Choice and Norm Preservation*

We compare the bounded Hirose tanh non-linearity versus the unbounded modReLU (see Section 7.4.2) in our cgRNN in Figure 7.3 and discover a strong interaction effect from the norm-preservation. First, we find that optimizing on the Stiefel manifold to preserve norms for the state transition matrices significantly improves learning, regardless of the non-linearity. In both the memory and the adding problem, keeping the state transition matrices unitary ensures faster and smoother convergence of the learning curve.

Without unitary state transition matrices, the bounded tanh non-linearity, i. e.the conventional choice is better than the unbounded modReLU. However, with unitary state transition matrices, the modReLU pulls ahead. We speculate that the modReLU, like the ReLU in the real setting, is a better choice of non-linearity. The advantages

Table 7.1: Comparison of gating functions on the adding and memory problems.

			memory problem		adding problem	
		gating function	frac.conv.	avg.iters.	frac.conv.	avg.iters.
uRNN [Wis+16]		no gate	1.0	2235	0.0	-
cgRNN	product	$\sigma(\Re(\mathbf{z})) \cdot \sigma(\Im(\mathbf{z}))$	0.10	4625	1.0	4245
	tied 1	$\alpha\sigma(\Re(\mathbf{z})) + (1-\alpha) \cdot \sigma(\Im(\mathbf{z}))$	0.55	4186	1.0	5458
	tied 2	$\sigma(\alpha\Re(\mathbf{z}) + (1-\alpha) \cdot \Im(\mathbf{z}))$	0.80	3800	1.0	5070
	free	$\sigma(\alpha\Re(\mathbf{z}) + \beta\Im(\mathbf{z}))$	0.75	2850	1.0	5235
	free real	$\sigma(\alpha\mathbf{z}_1 + \beta\mathbf{z}_2),\ (\mathbf{z}_1, \mathbf{z}_2) \in \mathbb{R}$	0.0	-	1.0	5313

The different gates are evaluated over 20 runs by looking at the fraction of convergence (frac.conv.) and average number of iterations required for convergence (avg.iters.) if convergent. A run is considered convergent if the loss falls below $5 \cdot 10^{-7}$ for the memory problem and 0.01 for the adding problem. We find that gating has no impact for the memory problem, i. e. the gateless uRNN [Wis+16] always converges, but is necessary for the adding problem. All experiments use weight normalized recurrent weights, a cell size of $n_h = 80$, and have networks with approximately 44k parameters; to keep approximately the same number of parameters, we set $n_h = 140$ for the uRNN and two independent gates each with $n_h = 90$ for the real free real case.

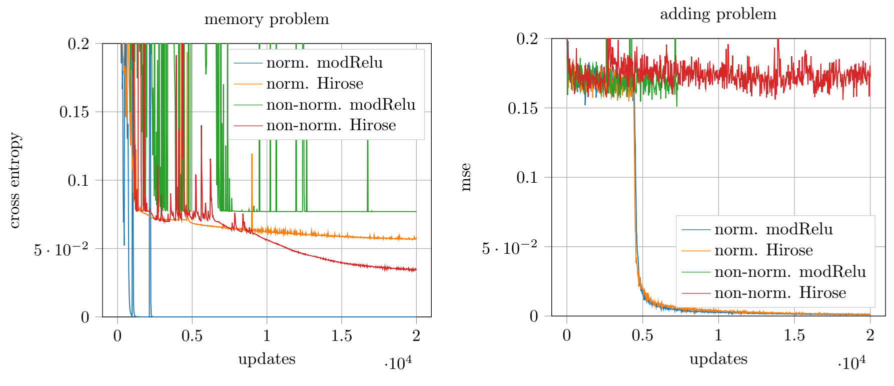


Figure 7.3: Comparison of non-linearities and norm preserving state transition matrices on the cgRNNs for the memory (a) and adding (b) problems for T=250. The unbounded modReLU (see Equation 7.7) performs best for both problems, but only if the state transition matrices are kept unitary. Without unitary state-transition matrices, the bounded Hirose non-linearity (see Equation 7.6) performs better. We use $n_h = 80$ for all experiments.

afforded upon it by being unbounded, however, also makes it more sensitive to stability, which is why these advantages are present only when the state-transition matrices are kept unitary. Similar effects were observed in real RNNs in [Rav+17], in which batch normalization was required in order to learn a standard RNN with the ReLU non-linearity.

7.5.5 *Real World Tasks: Human Motion Prediction & Music Transcription*

We compare our cgRNN to the state of the art GRU proposed by [MBR17] on the task of human motion prediction, showing the results in Table 7.2. Our cgRNN delivers state-of-the-art performance, while reducing the number of network parameters by almost 50%. However this reduction comes at the cost of having to compute the matrix inverse in Equation 7.10.

On the music transcription task, we are able to accurately transcribe the input signals with an accuracy of 53%. While this falls short of the complex convolutional state-of-the-art 72.9% of [Tra+18], their complex convolution-based network is fundamentally different from our approach. We conclude that our cgRNN is able to extract meaningful information from complex valued input data and will look into integrating complex convolutions into our RNN as future work.

Table 7.2: Comparison of our cgRNN with the GRU [MBR17] on human motion prediction.

	cgRNN				GRU[MBR17]			
Action	80 [ms]	160 [ms]	320 [ms]	400 [ms]	80 [ms]	160 [ms]	320 [ms]	400 [ms]
walking	0.29	0.48	0.74	0.84	**0.27**	**0.47**	**0.67**	**0.73**
eating	0.23	**0.38**	0.66	0.82	0.23	0.39	**0.62**	**0.77**
smo - king	**0.31**	**0.58**	**1.01**	**1.1**	0.32	0.6	1.02	1.13
dis - cus-sion	0.33	0.72	**1.02**	**1.08**	**0.31**	**0.7**	1.05	1.12
direct - ions	0.41	**0.65**	**0.83**	**0.93**	**0.41**	0.65	0.83	0.96
greeting	0.53	0.87	**1.26**	**1.43**	**0.52**	**0.86**	1.30	1.47
phoning	**0.58**	1.09	1.57	1.72	0.59	**1.07**	**1.50**	**1.67**
posing	**0.37**	**0.72**	**1.38**	**1.65**	0.64	1.16	1.82	2.1
pur - chases	0.61	0.86	**1.21**	1.31	**0.6**	**0.82**	1.13	**1.21**
sitting	0.46	0.75	1.22	**1.44**	**0.44**	**0.73**	**1.21**	1.45
sitting down	0.55	1.02	1.54	1.73	**0.48**	**0.89**	**1.36**	**1.57**
taking photo	**0.29**	**0.59**	**0.92**	**1.07**	0.29	0.59	0.95	1.1
waiting	0.35	0.68	1.16	**1.36**	**0.33**	**0.65**	**1.14**	1.37
walking dog	0.57	1.09	1.45	1.55	**0.54**	**0.94**	**1.32**	**1.49**
walking to-gether	**0.27**	**0.53**	**0.77**	**0.86**	0.28	0.56	0.8	0.88
average	**0.41**	**0.73**	**1.12**	**1.26**	0.42	0.74	**1.12**	1.27

Our cgRNN ($n_h = 512$, 1.8M params) predicts human motions which are either comparable or slightly better than the real-valued GRU [MBR17] ($n_h = 1024$, 3.4M params) despite having only approximately half the parameters.

7.6 CONCLUSION

In this paper, we have proposed a novel complex gated recurrent unit which we use together with unitary state transition matrices to form a stable and fast to train recurrent neural network. To enforce unitarity, we optimize the state transition matrices on the Stiefel manifold, which we show to work well with the modReLU. Our complex gated RNN achieves state-of-the-art performance on the adding problem while remaining competitive on the memory problem. We further demonstrate the applicability of our network on real-world tasks. In particular, for human motion prediction we achieve state-of-the-art performance while significantly reducing the number of weights. The experimental success of the cgRNN leads us to believe that complex representations have significant potential and advocate for their use not only in recurrent networks but in deep learning as a whole.

8

PHASE BASED FRAME PREDICTION [WYB20]

In this chapter, we investigate the problem of video frame prediction. Previous approaches have used recurrent neural networks [SMS15] to directly synthesize the predicted image [Xin+15; Shi+17]. These were trained using a mean squared error loss function in tandem with learned image synthesis. This approach introduces blur into the prediction as the network hedges its bets to minimize the error and smears predictions, to cover as many eventualities as possible.

This chapter corresponds to the paper: [WYB20] Moritz Wolter, Angela Yao, and Sven Behnke. Object-centered Fourier Motion Estimation and Segment Transformation Prediction. In: ESANN. 2020.

To address this issue, we focus on single objects that we assume have been fully pre-segmented. We model their movement separately, decoupling transformation learning from image synthesis. Our main contributions are:

- Given a moving pre-segmented object we estimate its centroid, as well as translational and rotational velocity based on phase correlation.
- We model the estimated velocities and their changes, e.g. at image boundaries, using neural networks and
- demonstrate that the frequency-based three pass image transformation method is able to produce sharp predictions for multiple time steps by transforming the input according to the estimated parameters.

The source code of this project is available at `https://github.com/v0lta/Fourier-Motion-Estimation-and-Segment-Transformation`.

8.1 RELATED WORK

Early works on future frame prediction [SMS15] rely on general-purpose recurrent architectures such as gated recurrent units (GRU) to encode an input video sequence and use a decoder with an identical structure to predict future frames. The moving MNIST test-problem consisting of handwritten digits moving in a box was introduced in [SMS15]. Later convolutional structures and flow models were integrated into recurrent cells [Xin+15; Shi+17]. All of the aforementioned methods employ a mean squared error loss function and use learned weights to synthesize the prediction. Generative adversarial network

(GAN) based formulations include [Wan+18a], but these are computationally expensive and hard to train. As a step towards a more efficient approach, we leverage image registration methods to estimate motion and rotation parameters [RC96; Xie+03; MZE09]. Work similar to ours couples CNNs and the log polar transform [Est+18] or estimates image translation by phase correlation [FB19]. Compared to [FB19], we additionally estimate and predict *image rotation* using a log-polar transformation.

8.2 METHODS FOR MOTION ESTIMATION

Due to its robustness, we estimate displacement and rotation of images by computing the normalized cross-correlation in the frequency domain [RC96]. Based on the two-dimensional discrete Fourier transformations of the current image $F_1 = \mathcal{F}(I_1)$ and that of its predecessor $F_2 = \mathcal{F}(I_2)$, we compute

$$C = \mathcal{F}^{-1}\Big(\frac{F_1 \odot \overline{F_2}}{\|F_1 \odot \overline{F_2}\|}\Big), \tag{8.1}$$

using the Hadamard product $\odot$. Afterwards, we find the displacement $\triangle\hat{x}$ and $\triangle\hat{y}$ by locating the correlation peaks in C. We employ the same strategy on log-polar transformed images to estimate the rotation velocity $\triangle\hat{\theta}$ [MZE09]. We further apply high-pass filtering of the log-polar transformation [RC96] to increase the accuracy. As we assume to be working with a pre-segmented object, we compute the centroid c_x, c_y by multiplying a normalized image $I_n = I/\sum I$ with the coordinate grids X,Y:

$$c_x = \sum I_n \odot X \text{ and } c_y = \sum I_n \odot Y. \tag{8.2}$$

8.3 NEURAL NETWORK PARAMETER CORRECTION

We choose a machine learning approach to correct the motion estimates, with a residual formulation modelling velocity. This leads to a predictor-corrector setting, where the learned model produces a correction based on current and previous estimates. More formally, we evaluate

$$(\triangle x, \triangle y, \triangle\theta)^T = (\triangle\hat{x}, \triangle\hat{y}, \triangle\hat{\theta})^T + \mathrm{net}(\hat{c}_x, \hat{c}_y, \triangle\hat{x}, \triangle\hat{y}, \triangle\hat{\theta}, \mathbf{s})^T, \tag{8.3}$$

where the network computes the correction applied to the parameter estimation based on the object centroid, velocity estimations and its own internal state **s**. Hats indicate estimates. The sum of network corrections and estimates are the transformation parameters which we use to transform the current image into the prediction. The object's centroid serves as prior knowledge to guide the motion estimation.

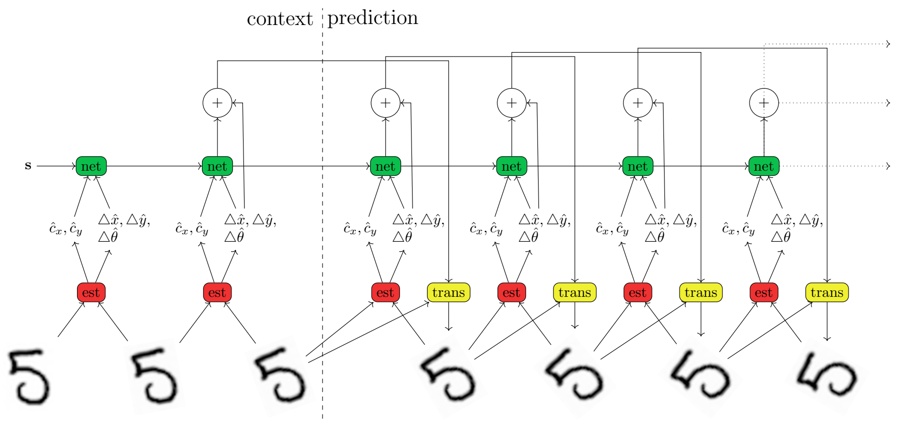


Figure 8.1: Overview of our estimation, correction and transformation framework. The estimator (est) finds transformation parameters between the last and current frame based on phase correlation and computes the object centroid. The parameters are corrected by the network (net) based on its encoding of the history s, by computing a residual which is added to the current estimate. Finally, the transformer (trans) transforms the last image using the phase-shift property of the Fourier transform to create the prediction.

Figure 8.1 illustrates our proposed approach. We pick a GRU structure in Equation 8.3 and update **s** accordingly.

8.4 FOURIER DOMAIN IMAGE TRANSFORMATION

We employ the three pass frequency domain method [Pan99][LOK97] to transform the current image based on the estimated parameters. It relies on the Fourier shift theorem for both translation and rotation. Given the desired translation $\triangle x$, we compute one-dimensional Fourier transforms and shift the phase using

$$I_t = \mathcal{F}^{-1}(\mathcal{F}(I)\exp(-i2\pi\triangle xf)) \tag{8.4}$$

to translate the image in x-direction. f denotes the frequencies and i the imaginary unit. For a translation of $\triangle y$ in y, we apply the same formulation but use the transposed image I^t instead of I. For a rotation by angle $\theta \in [-0.25\pi, 0.25\pi]$, we compute the shear parameters $a = \tan(\theta/2)$ and $b = -\sin(\theta)$ [LOK97]. We modify the phase in order to obtain a shear effect in x-direction using,

$$I_{sa} = \mathcal{F}^{-1}(\mathcal{F}(I)\exp(-i2\pi afy)), \tag{8.5}$$

with the same notation as in Equation 8.4 with y being the y-coordinate. In the second pass we shear in y-direction by transposing I and using b instead of a,

$$I_{sb} = \mathcal{F}^{-1}(\mathcal{F}(I^T)\exp(-i2\pi bfy))^T, \tag{8.6}$$

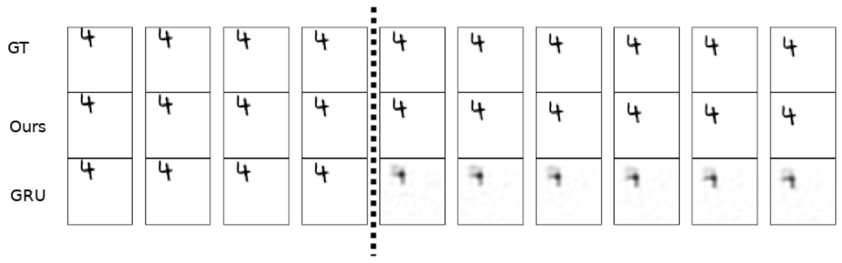


Figure 8.2: Moving MNIST translation prediction. Ground truth (top), our estimation correction cell prediction (middle) and a standard GRU of size 512 (bottom) are shown. Predictions are made using 4 context frames. We observe that our approach produces predictions which remain sharp, while the much larger GRU cell's predictions are blurry.

where we replace the shear parameters a and b. The transpose shifts x and y coordinates, therefore y appears. Finally, the third pass is a repetition of the first so that we obtain a full rotation.

8.5 VIDEO FRAME PREDICTION

We evaluate our approach using the popular moving MNIST data set [SMS15] and its rotating cousin [Shi+17]. We normalize inputs to be within $[0, 1]$ and choose a GRU for the correction-net in Equation 8.3. The state size is set to 50, the learning rate to 0.0005 and the batch size to 550. We stop training after 5000 iterations. In addition, we use the same parameters to train an off-the-shelf gated recurrent unit with a state size of 512 as a baseline.

8.5.1 *Translation*

In the classic MNIST translation setting [SMS15], digits move with a random velocity on a 64 by 64 pixel canvas and bounce off the walls. An important limitation of the velocity estimation described in Equation 8.1 is it's restriction to pixel-level accuracy. When used in recurrent operation, this restriction can lead to instabilities: Systematically underestimating movement may cause it to halt eventually. Systematic overestimation can lead to acceleration over time. The high level GRU corrects these errors and handles wall bounces. The state size of this cell can be comparatively small, because it integrates velocity and centroid information over time instead of entire images. We compare our approach to a conventional cell without estimation and image transformation capabilities. This recurrent cell directly synthesizes its prediction of the upcoming frame based on its internal state. While this low-level approach is more flexible, it suffers from blurred prediction as well as miss-classification. Results are shown in

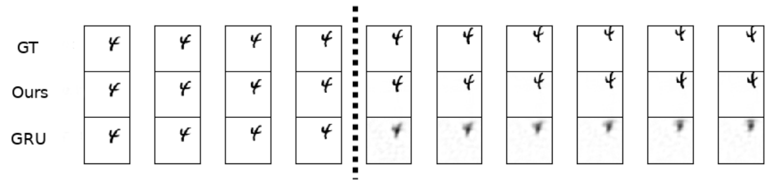


Figure 8.3: Rotating moving MNIST prediction, ground truth (top), our estimation correction transformation cell output (middle) and standard GRU (bottom).

Table 8.1: Evaluation using 550 prediction and ground truth sequences. The mean and standard deviation of our predictions are very close to the ground truth. Our approach performs slightly worse in terms of mean squared error with significantly fewer parameters and remains sharp.

	mean	std	gt-mse	#weights
Ground truth	0.025	0.137	-	-
Ours	**0.026**	**0.135**	0.015	**32k**
GRU	0.036	0.103	**0.009**	9182k

Figure 8.2. We observe that our results are very hard to distinguish from the ground truth. Even though the vanilla GRU state is ten times as large and its architecture is more flexible, the result is blurry and can suffer from miss-classification. Our higher-level approach prevents both.

8.5.2 *Rotation and Translation*

In Equations 8.5 and 8.6, we introduced our Fourier shift approach to rotation. We have already shown in Figure 8.2 that gradients can be back-propagated through our frequency-domain image translation operation. Figure 8.3 demonstrates that this also works for our multi-step rotation by shearing procedure. We can stack multiple phase modification transforms within recurrent cells. The Fourier transform is a unitary operation and our phase modification matrices do not modify the magnitude and therefore do not change the scaling. This enables us to run a stable process with multiple transformations per time step in a recurrent manner. In Table 8.1, we compare mean standard deviation and mean squared error of the ground truth as well as our and the GRU-baseline predictions. We observe that our approach does not significantly alter the mean and standard deviation. Using the standard GRU leads to slightly better performance in terms of mean squared error, but its predictions are not naturally distributed, which significantly lifts the mean and reduces the standard deviation.

8.6 CONCLUSION

In this paper, we studied a new approach to avoid the smearing effect which arises when learned image synthesis and mean squared error functions are combined. By modelling object movement at a higher level, we prevent our system from spreading out its predictions. Our approach could be an effective potential replacement for expensive GAN-based approaches for transformation scenarios, which are used to achieve the same goal. Object segmentation methods could be combined with our approach in the future.

9 WAVELET-LEARNING [WLY20]

This chapter corresponds to: [WLY20] Moritz Wolter, Shaohui Lin, and Angela Yao. Neural network compression via learnable wavelet transforms. In: ICANN. 2020.

Deep neural networks are routinely used in many artificial intelligence applications of computer vision, natural language processing, speech recognition, etc. [KSH12b; He+16; Ren+15], natural language processing [Dev+18] and speech recognition [Han+14; Amo+16]. However, the success of deep networks has often been accompanied by significant increases in network size and depth. Network compression aims to reduce the computational footprint of deep networks so that they can be applied on embedded, mobile, and low-range hardware. Compression methods range from quantization [CBD15; Ras+16], pruning [Han+15; Lin+18], to (low-rank) decomposition of network weights [Den+14; Lin+16].

Early methods [Den+14; Han+15] separated compression from the learning; compression is performed after network training and then followed by fine-tuning. Such a multi-stage procedure is both complicated and may also degrade performance. Ideally, one should integrate compression into the network structure itself; this has the dual benefit of learning less parameters and also ensures that the compression can be accounted for during learning in an end-to-end manner. The more direct form of integrated compression and learning has been adopted in recent approaches [Nov+15; Ioa+17; Yan+15], typically by enforcing a fixed structure on the weight matrices. Specifically, the structure must lend itself to some form of efficient projection or decomposition in order to have a compression effect, e. g. via circulant projections or tensor train decompositions. Maintaining such a structure throughout learning, however, can be challenging, especially using only the first-order gradient descent algorithms favoured in deep learning. Typically, constrained optimization requires managing active and inactive constraints, and or evaluating the Karush-Kuhn-Tucker conditions, all of which can be very expensive. We therefore require alternative ways to enforce weight structure.

One way to simplify the learning is to fix the projection bases *a priori* e. g. to sinusoids, as per the Fourier transform, or rectangular functions, as per the Walsh-Hadamard transform (WHT) and its derivative the Fastfood transform [AC09]. The latter has been used to compress the linear layers of CNNs [LSS13; Yan+15]. As it relies on non-local basis functions, the Fastfood transform has a complexity of $O(n \log n)$ for projecting a signal of length n. More importantly, however, using fixed bases limits the network flexibility and generalization power. Since the choice of basis functions determine the level of sparsity when representing data, the flexibility to choose a (more compact) basis could bring significant compression gains.

We advocate the use of wavelets as an alternative for representing the weight matrices of linear layers. Using wavelets offers us two key advantages. Firstly, we can apply the fast wavelet transform (FWT), which has only a complexity of $O(n)$ for projection and comes with a large selection of possible basis functions. Secondly, we can build upon the product filter approach for wavelet design [SN96] to directly integrate the learning of wavelet bases as a part of training CNNs or RNNs. Learning the bases gives us added flexibility in representing their weight matrices.

Motivated by these advantages, we propose a new linear layer which directly integrates the FWT into its formulation so layer weights can be represented as sparse wavelet coefficients. The compact nature of the wavelet representation can be considered a built-in form of network compression. Furthermore, rather than limit ourselves to predefined wavelets as basis functions, we learn the bases directly as a part of network training. Specifically, we adopt the product filter approach to wavelet design. We translate the two hard constraints posed by this approach into soft objectives, which serve as novel wavelet loss terms. By combining these terms with standard learning objectives, we can successfully learn linear layers which using only the wavelet transform, its inverse, diagonal matrices and a permutation matrix. As the reparametrisation is differentiable, it can trained end-to-end using standard gradient descent.

Our linear layer is general; as we later show in the experiments, it can be applied in both CNNs and RNNs. We focus primarily on gated recurrent units (GRUs), as they typically contain large and dense weight matrices for computing the cell's state and gate values. Our wavelet-based RNNs are compressed by design and have significantly fewer parameters than standard RNNs, yet still remain competitive in performance. Through exploring the use of compressed representations on individual cell components, we find a good trade-off between maintaining a large hidden vector state size while reducing the linear layer parameter count. Our main contributions can be summarized as follows:

- We propose a novel method of *learning FWT basis functions* using modern deep learning frameworks by incorporating the constraints of the product filter approach as soft objectives. By learning local basis functions, we are able to reduce the computational cost of the transform to $O(n)$ compared to existing $O(n \log n)$ methods that use fixed non-local basis functions.

- Based on this method, we propose an efficient linear layer which is compressed by design through its wavelet-based representation. This linear layer can be used flexibly in both CNNs and RNNs and allow for large feature or state sizes without the accompanying parameter blowup of dense linear layers.
- Extensive experiments explore the effect of efficient wavelet-based linear layers on the various parts of the GRU cell machinery. Our approach demonstrates comparable or better compression performance compared to state-of-the-art model compression methods.

Source code for this project is available at `https://github.com/v0lta/Wavelet-network-compression`.

9.1 RELATED WORK

9.1.1 *Structured Efficient Linear Transforms*

Our proposed approach can be considered a structured efficient linear transform, which replace unstructured dense matrices with efficiently structured ones. There are several types of structures, derived from fast random projections [AC09], circulant projections [Che+15; Ara+18], tensor train (TT) decompositions [Nov+15; TSN17; YKT17], low-rank decompositions [Den+13; Den+14; JVZ14b]

One of the main difficulties in using structured representation is maintaining the structure throughout learning. One line of work avoids this by simply doing away with the constraints during the learning phase. For example, low-rank decompositions [Den+13; Den+14; JVZ14b] split the learned dense weights into two low-rank orthogonal factors. The low-rank constraint then gradually disappears during the fine-tuning phase. The resulting representation is uncontrolled, and must trade off between the efficiency of the low rank and effectively satisfying the fine-tuning objective. In contrast, our proposed soft constraints can be applied jointly with the learning objective, and as such, not only does not require fine-tuning, but can ensure structure throughout the entire learning process.

Within the group of structured efficient linear transforms, the one most similar to the FWT that we are using is the Fastfood transform [LSS13]. The fastfood transform is applied to reparameterize linear layers as a combination of 5 types of matrices: three random diagonal matrices, a random permutation matrix and a Walsh-Hadamard matrix.

However, these three diagonal matrices are fixed after random initialization, resulting in a non-adaptive transform. Its fixed nature limits the representation power. As a remedy, [Yan+15] proposed an adaptive version in which the three diagonal matrices are optimized through standard backpropagation. Nevertheless, the approach still uses a fixed Walsh-Hadamard basis which may limit the generalization and flexibility of the linear layer. In contrast to the adaptive Fastfood transform, our method is more general and reduces computational complexity.

9.1.2 *Compressing Recurrent Neural Networks*

Compressing recurrent cells can be highly challenging; the recurrent connection forces the unit to be shared across all time steps in a sequence so minor changes in the unit can have a dramatic change in the output. To compress RNNs, previous approaches have explored pruning [Wen+17; Nar+17], quantization [WLW17] and structured linear transforms [TSN17; YKT17; Pan+19; Ye+18].

The use of structured efficient linear transforms for compressing RNNs has primarily focused on using tensor decompositions, either via tensor train decomposition [TSN17; YKT17] or block-term tensor decomposition [Ye+18]. The tensor decomposition replaces linear layers with tensors with a lower number of weights and operations than the original matrix. Tensor train decomposition can compress the input-to-hidden matrix in RNNs, but requires the restricted setting on the hyperparameters (e. g.ranks and the restrained order of core tensors) making the compressed models sensitive to parameter selection [TSN17].

Pan *et al.* [Pan+19] employ low-rank tensor ring decomposition to alleviate the strict constraints in tensor train decomposition. However, these methods need to approximate the matrix-vector operation by tensor-by-tensor multiplication, where the dense weights and input vectors are reshaped into higher-order tensors. This requires additional reshaping time and generates feature-agnostic representations during training. In contrast, our method shows more flexibility and efficiency performing on the fast wavelet transform, whose bases satisfy with wavelet design using soft objectives.

9.2 METHOD

9.2.1 *Wavelet Basis Learning*

To enforce the constraints discussed in section 4.5.2 in a learnable setting, we can design corresponding differentiable loss terms. We will call the sum of these terms wavelet loss. Instead of working in the $z-$space, we leverage the equivalence of polynomial multiplication and coefficient convolution ($*$) and reformulate Eq. 4.18 as:

$$\mathcal{L}_{\mathrm{pr}}(\theta_w) = \sum_{k=0}^{N} \Big[(\mathbf{h}_0 * \mathbf{f}_0)_k + (\mathbf{h}_1 * \mathbf{f}_1)_k - \mathbf{o}_{2,k}\Big]^2, \tag{9.1}$$

where $\mathbf{o}_2$ is a zero vector with a two at z^l in its center. This formulation amounts to a measure of the coefficient-wise squared deviation from the perfect reconstruction condition. For alias cancellation, we observe that Eq. 4.19 is satisfied if $F_0(z) = H_1(-z)$ and $F_1(z) = -H_0(-z)$ and formulate our anti-aliasing loss as:

$$\mathcal{L}_{ac}(\theta_w) = \sum_{k=0}^{N} \left(f_{0,k} - (-1)^k h_{1,k}\right)^2 + \left(f_{1,k} + (-1)^k h_{0,k}\right)^2. \tag{9.2}$$

This formulation leads to the common alternating sign pattern, which we will observe later. We refer to the sum of the two terms in Eqs. 9.1 and 9.2 as a wavelet loss. It can be added to standard loss functions in the learning of neural networks.

9.2.2 *Efficient Wavelet-based Linear Layers*

To use the efficient wavelet-based linear layer, we begin by decomposing the weight matrices $\mathbf{W}$ as follows:

$$\mathbf{W} = \mathbf{D}\mathcal{W}^{-1}\mathbf{G}\Pi\mathcal{W}\mathbf{B}, \tag{9.3}$$

where $\mathbf{D}, \mathbf{G}, \mathbf{B}$ are diagonal learnable matrices of size $n \times n$, and $\Pi \in \{0,1\}^{n \times n}$ is a random permutation matrix, which stays fixed during training. We use identity matrices as initialization for $\mathbf{D}, \mathbf{G}, \mathbf{B}$. $\mathcal{W}$ and $\mathcal{W}^{-1}$ denote the wavelet transform and it's inverse, which can also be optimized during training. This approach is similar to [Yan+15], which relies on the fast Welsh-Hadamard transform. $\mathbf{D}, \mathbf{G}, \mathbf{B}$ and ″ can be evaluated in $O(n)$ [ASB16]. The fast wavelet transform requires only $O(n)$ steps instead of the $O(n \ln n)$ used by the non-local fast Fourier and fast Welsh-Hadamard transforms [SN96, page 29]. This is asymptotically faster than the transforms used in [Yan+15] and [ASB16], who work with fixed Welsh-Hadamard and Fourier transforms. Non-square cases where the number of inputs is larger than n can be handled by concatenating square representations with tied wavelet weights, see [Yan+15].

We can replace standard weights in linear layers with the decomposition described above, and learn the matrices in Eq. 9.3 using wavelet loss as defined in Eq. 9.1 and 9.2 jointly with the standard network objective. Given the network parameters θ and all filter coefficients θ_w we minimize:

$$\min(\mathcal{L}(\theta)) = \min[(\mathcal{L}_o(\theta)) + \mathcal{L}_{pr}(\theta_w) + \mathcal{L}_{ac}(\theta_w)], \tag{9.4}$$

where $\mathcal{L}_o(\theta)$ the original loss function (e. g.cross-entropy loss) of the uncompressed network, and $\mathcal{L}_{pr}(\theta_w) + \mathcal{L}_{ac}(\theta_w)$ the extra terms for the learnable wavelet basis.

Our wavelet-based linear layer can be applied to fully-connected layers in CNNs and RNNs. For example, suppose we are given a gated recurrent unit (GRU) as follows:

$$\begin{aligned} \mathbf{g}_r &= \sigma(\mathbf{W}_r \mathbf{h}_{t-1} + \mathbf{V}_r \mathbf{x}_t + \mathbf{b}_r), && (9.5) \\ \mathbf{g}_z &= \sigma(\mathbf{W}_z \mathbf{h}_{t-1} + \mathbf{V}_z \mathbf{x}_t + \mathbf{b}_z), && (9.6) \\ \mathbf{z}_t &= \mathbf{W}_t(\mathbf{g}_r \odot \mathbf{h}_{t-1}) + \mathbf{V}\mathbf{x}_t + \mathbf{b}, && (9.7) \\ \mathbf{h}_t &= \mathbf{g}_z \odot \tanh(\mathbf{z}_t) + (1 - \mathbf{g}_z) \odot \mathbf{h}_{t-1}, && (9.8) \end{aligned}$$

where $\sigma(\cdot)$ and $\tanh(\cdot)$ are sigmoid and tanh activation functions, $\mathbf{z}_t$ is the candidate hidden layer values, $\mathbf{h}_t$ is the hidden layer state at the t-th time, and $\mathbf{g}_r, \mathbf{g}_z$ are the reset and update gates, respectively. We can learn efficient gating units by applying the representation from Eq. 9.3 to the recurrent weight matrices $\mathbf{W}_t$, $\mathbf{W}_r$ and $\mathbf{W}_z$. The recurrent weight matrices are of size $n_h \times n_h$ and are typically the largest matrices in a GRU and learning efficient versions of them can reduce the number of network parameters up to 90%.

9.3 RETRAINING A COMPRESSED WAVELET LAYER

We evaluate the effectiveness of our linear layer in both CNNs and RNNs. For CNNs, we select LeNet-5 on MNIST digit recognition classification benchmark [LeC+98] as a baseline. For RNNs, we test several GRU models on sequence learning tasks including the copy-memory and adding problem [HS97b], sequential MNIST and Penn-Treebank (PTB) character modelling [MSM93].

9.3.1 *MNIST-Digit Recognition*

We first apply our efficient wavelet layers to the MNIST digit recognition problem [LeC+98]. MNIST has 60K training and 10K test images with a size of 28×28 from 10 classes. We train using an Adadelta optimizer for 14 epochs using a learning rate of 1. The adaptive Fastfood transform is replaced with our proposed learnable-wavelet compression layer. In this feed-forward experiment, we apply dropout to the learned diagonal matrices $\mathbf{D}, \mathbf{G}, \mathbf{B}$ in Eq. 9.3.

We start by randomly initializing the wavelet array, consisting of 6 parameters per filter, with values drawn from the uniform distribution $\mathcal{U}_{-1}^{1}$. In this case, the initial condition is not a valid product filter so the wavelet loss is initially very large, as shown in Figure 9.1. However, as this loss term decreases as the random values in the wavelet array start to approximately satisfy the conditions of Eq. 4.18 and Eq. 4.19. Correspondingly, the accuracy also rises. With the random initialization we achieve a recognition accuracy of 98.33% with only 36k parameters. We visualize the learned filters in Figure 9.2. The alias cancellation condition causes an alternating sign pattern. Whenever

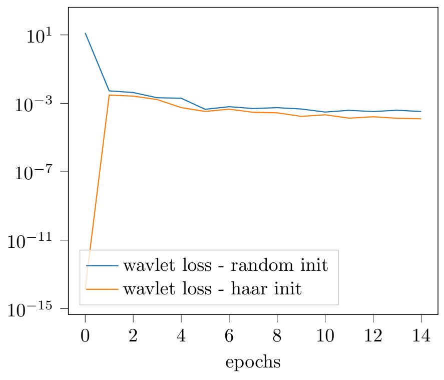


Figure 9.1: Wavelet loss sum of a randomly and haar initialized wavelet array. In both cases, filter values converge to a product filter as indicated by trend of the wavelet loss towards zero.

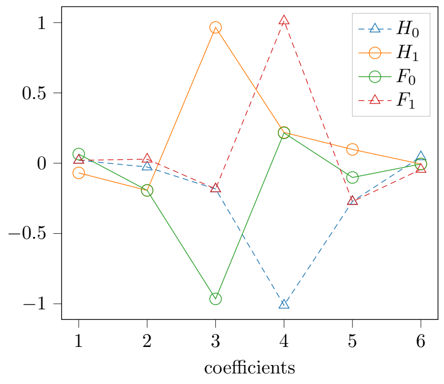


Figure 9.2: Learned wavelet filter coefficients. Coefficients have been initialized at random. After training, the effects of the alias cancellation constraint are prominently visible. We must have $F_0(z) = H_1(-z)$ and $F_1(z) = -H_0(-z)$ for alias to cancel itself. Inserting $(-z)$ into the coefficient polynomial leads to a minus sign at odd powers. Additional multiplication with (-1) shifts it to even powers. Alias cancellation therefore imposes an alternating sign pattern. When F_0 and H_1 share the same sign F_1 and H_0 do not and vice versa.

Net	Error	Parameters	Reduction
LeNet-5	0.87%	431K	-
LeNet-Fastfood	0.71%	39K	91%
LeNet-random	1.67%	36K	92%
LeNet-Wavelet	0.74%	36K	92%

Table 9.1: Experimental results on the MNIST digit-recognition. We work with a LeNet architecture as proposed in previous work. In comparison to the fastfood approach [Yan+15] we obtain comparable performance with slightly fewer parameters. The size of our learnable-wavelet compression layer is set to 800.

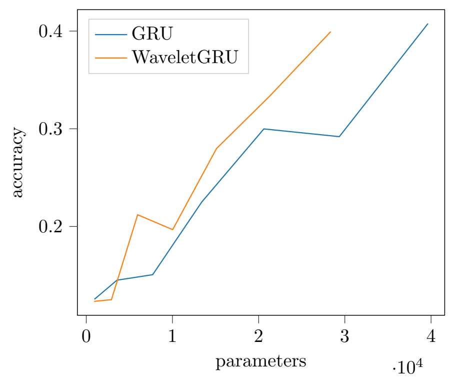


Figure 9.3: Accuracy and parameters of gated recurrent units with and without a compressed reset gate on the memory problem with $T = 60$. Both models start with 1K parameters. As we increase the state size from 12 to 108, the parameter count rises to 39K for the standard GRU and to 29K for the compressed version. We observe that reset gate compression increases parameter efficiency in this case.

F_0 and H_1 have the the same sign F_1 and H_0 do not and vice versa. As we observe high recognition rates and small wavelet loss values, we are confident that we have learned valid wavelet basis functions.

Inspired by the Welsh-Hadamard matrix, we also test with a zero-padded Haar wavelet as a initialization; this should ensure a valid FWT at all ties. In this case, the initial wavelet loss is very small as shown in Figure 9.1, as we already start with a valid wavelet that perfectly satisfies conditions 4.18 and 4.19. However, as learning progresses, the condition is no longer satisfied, hence the jump in the loss, before the gradual decrease once again. In Table 9.1, we compare our result to the Fastfood transform [Yan+15]. Our method achieves a comparable result with a higher parameter reduction rate of 92% (*vs.* 91%), which is quite impressive.

9.3.2 *RNN-Compression*

9.3.2.1 *The Copy-Memory and Adding Benchmarks*

To explore the effect of our method on recurrent neural networks, we consider the challenging copy-memory and adding tasks as benchmarks [HS97b].

The copy memory benchmark consists out of a sequence of 10 numbers, T zeros, a marker and 10 more zeros. The tested cell observes the input sequence. It must learn to reproduce the original input sequence after the marker. The numbers in the input sequence are drawn from $\mathcal{U}_0^n$. Element $n+1$ marks the point where to reproduce the original sequence. We choose to work with n=8 in our experiments and use a cross entropy loss. Accuracy is defined as the percentage of correctly memorized integers.

		Adding problem			Copy-memory problem		
	reduced	mse-loss	accuracy	weights	ce-loss	accuracy	weights
GRU	-	4.9e-4	99.23%	792K	4.8e-5	99.99%	808K
WGRU	$\mathbf{z_t}$	3.0e-4	99.45%	531K	2.4e-3	99.1%	548K
WGRU	$\mathbf{g_r}$	1.1e-4	99.96%	531K	3.7e-5	99.98%	548K
WGRU	$\mathbf{g_z}$	4.4e-4	97.78%	531K	1.1e-1	21.63%	548K
WGRU	$\mathbf{g_r}, \mathbf{g_z}$	0.9e-4	99.85%	270K	3.7e-2	73.63%	288K
WGRU	$\mathbf{z_t}, \mathbf{g_r}$	3.0e-4	98.56%	270K	2.4e-3	99.05%	288K
WGRU	$\mathbf{z_t}, \mathbf{g_z}$	1.1e-3	92.64%	270K	1.2e-1	12.67%	288K
WGRU	$\mathbf{z_t}, \mathbf{g_r}, \mathbf{g_z}$	1.0e-3	91.64%	10K	1.2e-1	16.84%	27K
Ff-GRU	$\mathbf{z_t}, \mathbf{g_r}, \mathbf{g_z}$	1.3e-3	85.99%	10K	1.2e-1	16.44%	27K

Table 9.2: RNN compression results on the adding and memory problems, exploring the impact of our efficient wavelet-based linear layer at various locations in the GRU. On the adding problem all tested variants are functional. Compressing the state and reset equations has virtually no effect on performance. Compressing the update gate leads to a working cell, but cells with a compressed update gate perform significantly worse. Note that on the adding problem, predicting a sum of 1 regardless of the input leads to an mse of 0.167. On the copy-memory benchmark, replacing the the state and reset weight matrices with our efficient wavelet version is possible without significant performance losses. A state size of 512 was used for all models. The expected cross entropy for a random guess is 0.12 with n=8.

For the adding problem, T random numbers are drawn from $\mathcal{U}_0^1$ out of which two are marked. The two marks are randomly placed in the first and second half of the sequence. After observing the entire sequence, the network should produce the sum of the two marked elements. A mean squared error loss function is used to measure the difference between the true and the expected sum. We count a sum as correct if $|\hat{y} - y| < 0.05$. We test on GRU cells with a state size of 512. T is set to 150 for the adding problem. Optimization uses a learning rate of 0.001 and RMSprop.

We first explore the effect of our efficient wavelet layer on the reset g_r and update g_z equations (Eq. 9.5, 9.6) as well as the state z_t equation (Eq. 9.7). As shown in Table 9.2, We observed that efficient representations of the state and reset equations has little impact on performance, while significantly reducing the weights. In the combined case, our method has 2.8× less parameters than dense weight matrices with only 0.91% accuracy drop in copy memory problem. For the adding problem, our method achieves a factor of 2.9× reduction with only 0.67% accuracy drop. When incorporating this into multiple weight matrices, we find that using the efficient representation is problematic for the update matrix W_z next state computation. We found that compressing the update gate has a large impact on the performance, especially the combination with state compression. The update mechanism plays an important role for stability and should not be compressed.

Compared to the Fastfood transform [Yan+15], our method is better at compressing entire cells. It achieves higher accuracy with the same number of weights both in adding problem and copy memory problem. For example, in adding problem, our method achieves a higher accuracy of 91.64% (*vs.* 85.99%) with the same number of weights, compared to the Fastfood transform [Yan+15]. Figure 9.3 compares the relationship of weight growth and accuracy for reset compression and a standard GRU on the memory problem. We observe that the compressed case is more efficient.

9.3.2.2 *Sequential-MNIST*

The sequential MNIST benchmark dataset has previously been described in Section 9.3.1. A gray-scale image with a size of 28×28 is interpreted as a sequence of 784 pixels. The entire sequence is an input to the GRU, which will generate a classification score. We select a GRU with a hidden size of 512 as our baseline and an RMSProp optimizer with a learning rate of 0.001.

Results of our method are shown in Table 9.3. Similar to the results of Table 9.2, we also observe that having efficient representations for the state and reset gate matrices work well, but compressing the update gate adversely impacts the results. Compared to only state compression, the combination of state and reset compression achieves

Sequential MNIST				
	reduced	loss	accuracy	weights
GRU	-	6.49e-2	100%	795K
Wave-GRU	$\mathbf{z_t}$	8.98e-2	98%	534K
Wave-GRU	$\mathbf{g_r}$	6.06e-2	100%	534K
Wave-GRU	$\mathbf{g_z}$	1.82	26%	534K
Wave-GRU	$\mathbf{g_r}, \mathbf{g_z}$	1.33	46%	274K
Wave-GRU	$\mathbf{z_t}, \mathbf{g_r}$	9.48e-2	98%	274K
Wave-GRU	$\mathbf{z_t}, \mathbf{g_z}$	1.60	34%	274K
Wave-GRU	$\mathbf{z_t}, \mathbf{g_z}, \mathbf{g_r}$	1.52	36%	13K
WaveGRU-64	$\mathbf{z_t}, \mathbf{g_r}$	0.127	96.4%	4.9K
TT-GRU	-	-	98.3%	5.1K

Table 9.3: RNN compression results on the sequential MNIST benchmark. The pattern here reflects what we saw on the adding and copy-memory benchmarks. Touching the update gate has a negative impact. All other equations can be compressed. our method (WaveGRU-64) achieves a comparable performance, compared to [TSN17].

a higher compression rate of 2.9× (*vs.* 1.5×), without the accuracy drop. We also compare to the tensor train approach used in [TSN17]. We apply the efficient wavelet layer only on the reset and state weight matrices and reduce the cell size to 64. Our approach does reasonably well with fewer parameters.

9.3.2.3 *Penn Treebank Character Modelling*

We verify our approach on the Penn-Treebank (PTB) character modelling benchmark [MSM93]. We split the dataset into training, validation and test sequences, which contains 5,059K training characters, 396K validation characters and 446K testing characters. Given an input sequence of 320 characters, the model should predict the next character. We work with a GRU of size 512 trained using an Adam optimizer with an initial learning rate of 0.005. Training is done using a cross entropy loss in addition to the wavelet loss, and results are reported using bits per character (bpc), where lower pbc is better. In Table 9.4, we show results for a temporal convolutional network (TCN) [BKK18], a vanilla GRU cell as well as state, reset and state reset compression, which we found to be successfully earlier. We confirm that our wavelet-based compression method can be used to compress reset gate and cell state without significant performance loss.

Penn Treebank Character Prediction

	reduced	loss	bpc	weights
TCN	-	0.916	1.322	2,221K
GRU	-	0.925	1.335	972K
Wave-GRU	$\mathbf{z}_t$	0.97	1.399	711K
Wave-GRU	$\mathbf{g}_r$	0.925	1.335	711K
Wave-GRU	$\mathbf{z}_t, \mathbf{g}_r$	0.969	1.398	450K

Table 9.4: Results for the best performing architectures on the Penn-Treebank data set, we compare to a TCN as proposed in [BKK18]. We can compress the GRU cells state and reset equations without a significant drop in performance.

9.4 SUMMARY AND OUTLOOK

We presented a novel wavelet-based efficient linear layer which demonstrates competitive performance within convolutional and recurrent network structures. On the MNIST digit recognition benchmark, we show state of the art compression results as well as convergence from randomly initialized filters.

We explore RNN compression and observe comparable performance on the sequential MNIST task. In a gated recurrent unit we can compress the reset and state equations without a significant impact on performance. The update gate equation was hard to compress, in particular in combination with the state equation. Joint update gate and reset gate equation compression generally worked better than update and state compression. We conclude that the update mechanism plays the most important role within a GRU-cell, followed by the state equation, and finally, the reset gate. Results indicate that selective compression can significantly reduce cell parameters while retaining good performance.

Product filters are only one way of wavelet design alternative methods include lifting or spectral factorization approaches. We look forward to exploring some of these in the future. Efficient implementation of the FWT on GPUs is no simple matter. We will open our framework upon acceptance of this paper in the hope that it will spark future work on highly optimized implementations.

9.5 COMPRESSION TROUGH WAVELET QUANTIZATION

Quantization of wavelet coefficients plays an important role in image compression for example in the JPEG 2000 standard [SN96]. This section explores the combination of weight quantization, Huffman coding, and wavelet transformations [HMD16b]. If the input data

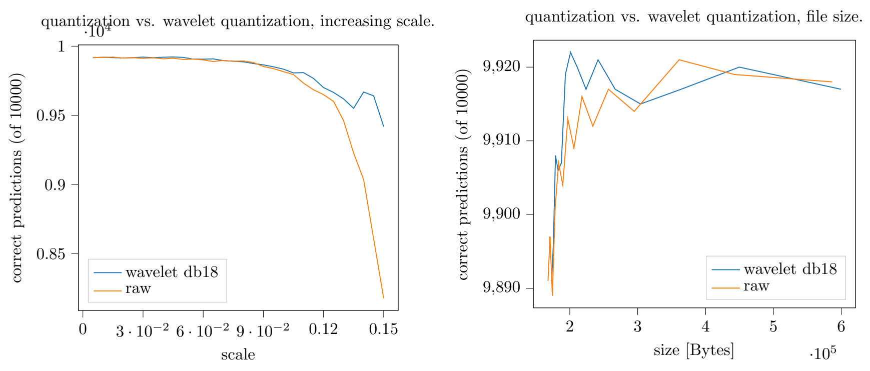


Figure 9.4: Accuracy versus scale and file size for a MNIST 4 layer CNN example. We observe that the wavelet coefficient are more resistant to quantization in this case. The shifted peak in the right indicates that the file size can roughly be cut in half here.

is sufficiently correlated, wavelet transforms can introduce sparsity into the outputs. Increasing sparsity reduces the network size on disk, when combined with Huffman-coding, because the histogram of network weights is shifted towards the center, where Huffman codes are short.

9.5.1 *Quantization*

Weights are quantized by rounding to integer values after scaling and shifting [Pas+19]:

$$Q_w = \text{round}(x/s + p_0), \tag{9.9}$$

above s denotes the scale and p_0 the shift. After rounding, the network weights can be stored as 8 bit integers or Huffman coded for additional compression. The range of integers which the network weights are mapped to is controlled by s. Choosing a small s increases the size of the resulting integers and the distance between these. By decreasing s two similar floats are less likely to be mapped to the same integer. A small s typically preserves more information but produces a more diverse set of integers which will increase the network size after Huffman coding.

9.5.2 *Wavelet compression of a quantized convolutional neural network*

In the following experiments, the wavelet transformed networks are compressed using quantization and Huffman coding. The wavelet transform is applied along the input and output channels for convolution filters and along the rows of the fully connected matrices.

After decompression, the performance is recorded. figure 9.4 depicts results for a network trained on MNIST consisting out of two convolutional and two fully connected layers. On the left of figure 9.4, we show the evolution of the MNIST digit classification accuracy for an increasing compression rate s. The blue line shows the wavelet case, the orange line simple Huffman quantization. The measurements indicate that the wavelet transformed network weights are more robust to quantization than their un-transformed counterparts. On the right of figure 9.4 we show file size versus classification accuracy. We find that wavelet quantization creates a peak on the right, which can be exploited for compression purposes. In the wavelet case we find a file size of reduction of roughly 40%, for the four layer CNN.

9.6 summary and outlook

The MNIST wavelet compression experiment discussed above showed file size reductions for a wavelet compressed network. Follow up experiments on deeper res-net architectures did not yield the same results. Probably because in higher layers convolution filters are less correlated along the input and output filter dimensions, which reduces the sparsity and thereby the compression efficiency of the wavelet quantization approach. As shown by the mnist experiments, wavelet compression can be beneficial for shallower network architectures.

10 CONCLUSION

This thesis combined spectral methods and machine learning in different ways. Time and the frequency domain are linked through the Fourier and wavelet transforms. The complex nature of Fourier coefficients and the compression power of the wavelet transform motivated the design of the new network architectures we saw in the research part.

10.1 LESSONS LEARNED

Frequency-based methods have long a long track record as preprocessing tools. Due to their enormous flexibility, neural networks often did not benefit from additional preprocessing using static linear transforms. Consequently, flexible transforms should be preferred, but more importantly, additional insights must be built into the learning system.

10.1.1 *STFT learning*

The Fourier transform has the useful property of clustering the most essential information into the low-frequency bands. Low pass filtering, therefore, allows parameter reductions in some cases. Fewer parameters lead to faster training and evaluation. An important side effect of the STFT is that it groups input signals into windows. In chapter 6 we learned how much of a difference this makes for long time series. RNNs are not infinitely stable and the granularity of time information matters [Kou+14]. Running at lower clock rates reduces the computational load and state matrix evaluations. Windowing, therefore, improves both runtime and gradient stability in our experiments. Finally, previous work on end to end learning has shown that it is important to establish gradient signals whenever possible [Cha+16]. We enable gradient flow trough the STFT, which makes it possible to optimize window shapes.

10.1.2 *Complex Networks*

The Fourier transform produces complex coefficients. A complex recurrent cell to process these was designed in Chapter 7. It described automatic-gradient descent with complex weights and functions. The choice of a complex activation and the design of the gating functions took holomorphy and stability considerations into account. The interplay of unitary state updates and the activation function was explored. The presented experiments confirmed that unitary state matrix updates allow unbounded recurrent activation functions. If the state matrix norm is allowed to grow freely, the activation function cannot do the same. As new complex gate activation function chapter 7 introduced the modSigmoid, a mapping from $\mathbb{C}$ to $\mathbb{R}$. This new function leads to functional gates, which maintain cell stability by combining information from both real and complex parts. It allows complex memory cells to manage their memory, while keeping the complex-phase intact, a prerequisite for stability. Overall complex networks worked well. The complex formulation allowed parameter reductions. For CNNs [Tra+18] made similar observations.

10.1.3 *Video Prediction*

Video prediction using image registration and transformation was the main topic of Chapter 8. Phase based image registration and transformation methods were integrated into a recurrent cell. The resulting network transforms input images with a single object into predictions based on estimated and corrected parameters. The proposed cell does not suffer from the common smearing problems that standard cells have.

10.1.4 *FWT integration in neural nets*

Integration of wavelets into neural networks was the focus of chapter 9. The fast wavelet transform relies on convolutions, this fact enables us to implement the FWT using the highly optimized convolution functions of modern machine-learning frameworks. Experiments showed that the FWT works well within neural networks, especially for compression purposes. Additional loss functions allow wavelet optimization, which adds extra flexibility in comparison to previous approaches.

10.1.5 *Choosing a Transform*

Choosing a frequency domain representation can be difficult. For example, faced with the choice of a Fourier or Wavelet transform, it helps to think about these transforms as bases. The transform with basis functions most similar to the data will work best. An image with many sharp lines and borders will be more easily represented using wavelets with sharp edges. In this case, a Fourier representation, which only contains smooth functions, will not lead to a very sparse representation. The basis functions won't fit the data very well, and we will have to use more. This problem is also known as Gibbs phenomenon [SN96]. Within higher CNN layers or recurrent cells, representations are not very smooth. Choosing wavelets in Chapter 9 therefore seemed natural.

10.2 REPRODUCIBILITY AND OPEN SOURCE

Aiding future engineers and scientists with the reproduction and extension of all experiments described in this text was an important goal of this thesis project.

In addition to scientific reasons, as a researcher working in public institutions, I directly benefited from generous funding through tax money. To give back to the public that paid for it, the source code written for this thesis is available under permissive open-source licenses.

Despite my best efforts, the experimental design and network implementations are not perfect. The training process of randomly initialized neural networks does not always converge to the same solution. To make results perfectly reproducible all seeds to random number generators must be recorded, and stochastic code be removed where recording the seed is not possible. Given time constraints and the closed stochastic nature of parts of the GPU related training code making results exactly reproducible was not possible. In addition to stochasticity, software dependencies should be identical to allow perfect reproduction. Key library versions are documented in readme files, but the code is not dockerized, which would have been the preferred option.

Despite these shortcomings, all code releases have been carefully tested and evaluated. I hope that opening and documenting three and a half years of work will help future scientists working at the intersection of frequency domain and machine learning.

10.3 FUTURE WORK

Some progress has been made in this thesis, but of course the future still holds many potential discoveries.

One could, for example, place the convolution structure into a matrix instead of the image vector. This way, it should be possible to draw on the theory of doubly block circulants [Dav79]. Fourier matrices diagonalize block circulants efficiently [Dav79]. Having easy access to convolution operator eigenvalues might make novel orthogonality or unitary loss functions possible or enable learning the convolution weights directly in the eigenspace. The complex eigenvalue-weights will lead to real filter matrices if a Hermitian pattern is enforced. Having invertible machine learning layers may be particularly interesting in the context of autoencoders and for interpretable machine learning-purposes. Furthermore, unitary convolution operators based on doubly block circulants may enable us to develop CNN which run on quantum computers.

The phase-based approach presented in chapter 8, currently only works on videos with a single transformation. To make it work in more realistic settings, future work could couple it with an object segmentation algorithm. Multiple registration and transformation cells would then predict the movement of many objects separately.

Rescaling can be done very accurately in the frequency domain. Recently pooling papers using the fast Fourier and Wavelet transforms have appeared [WL18; RSA15]. Using the proposed wavelet learning approach from chapter 9, it is probably possible to propose an adaptive wavelet pooling layer, which would add extra flexibility to modern neural networks. In addition to the product filter loss, orthogonal wavelet loss functions could be proposed and explored. Orthogonal FWT layers may form useful additions to the convolutional case.

Quantum computers require complex numbers and unitary matrices by design, putting chapter 7's results to use in the design of quantum-RNN will make for exciting future work. Future complex-valued neural-networks may also feature holomorph activation functions such as the Möbius-transform. The Möbius transform is a learnable optimization function, which could add extra flexibility into the network. However, all holomorph functions have singularities, which cause instability. A solution or workaround for these singularities, would clear the path towards many new complex activation functions.

BIBLIOGRAPHY

[Abi09] Rahib H Abiyev. "Fuzzy wavelet neural network for prediction of electricity consumption." In: *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AI EDAM* 23.2 (2009), p. 109.

[AC09] Nir Ailon and Bernard Chazelle. "The fast Johnson-Lindenstrauss transform and approximate nearest neighbors." In: *SIAM Journal on computing* 39.1 (2009), pp. 302–322.

[AHW16] Tayfun Alpay, Stefan Heinrich, and Stefan Wermter. "Learning multiple timescales in recurrent neural networks." In: *International Conference on Artificial Neural Networks*. Springer. 2016, pp. 132–139.

[Amo+16] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. "Deep speech 2: End-to-end speech recognition in english and mandarin." In: *ICML*. 2016, pp. 173–182.

[Ara+18] Alexandre Araujo, Benjamin Negrevergne, Yann Chevaleyre, and Jamal Atif. "Training compact deep learning models for video classification using circulant matrices." In: *ECCV*. 2018.

[ASB16] Martin Arjovsky, Amar Shah, and Yoshua Bengio. "Unitary evolution recurrent neural networks." In: *ICML*. 2016, pp. 1120–1128.

[BKK18] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling." In: *arXiv:1803.01271* (2018).

[BKK19] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. "Trellis Networks for Sequence Modeling." In: *International Conference on Learning Representations*. 2019. URL: `https://openreview.net/forum?id=HyeVtoRqtQ`.

[Bau20] Johannes Bausch. "Recurrent quantum neural networks." In: *Advances in Neural Information Processing Systems* 33 (2020).

[BL+07] Yoshua Bengio, Yann LeCun, et al. "Scaling learning algorithms towards AI." In: *Large-scale kernel machines* 34.5 (2007), pp. 1–41.

[BSF94] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. “Learning long-term dependencies with gradient descent is difficult.” In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[BP92] N. Benvenuto and F. Piazza. “On the complex backpropagation algorithm.” In: *IEEE Trans. Signal Processing* 40.4 (1992), pp. 967–969.

[Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[Bor13] Bornemann. *Funktionentheorie*. Birkhäuser, 2013.

[Bru+20] Lilli Bruckschen, Kira Bungert, Moritz Wolter, Stefan Krumpen, Reinhard Weinmann Michael amd Klein, and Maren Bennewitz. “Where Can I Help? Human-Aware Placement of Service Robots.” In: *International Symposium on Robot and Human Interactive Communication*. 2020.

[BM13] Joan Bruna and Stéphane Mallat. “Invariant scattering convolution networks.” In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1872–1886.

[Cha+16] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. “Listen, attend and spell: A neural network for large vocabulary conversational speech recognition.” In: *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE. 2016, pp. 4960–4964.

[CCM14] Xu Chen, Xiuyuan Cheng, and Stéphane Mallat. “Unsupervised deep haar scattering on graphs.” In: *Advances in Neural Information Processing Systems*. 2014, pp. 1709–1717.

[CYD06] Yuehui Chen, Bo Yang, and Jiwen Dong. “Time-series prediction using a local linear wavelet neural network.” In: *Neurocomputing* 69.4-6 (2006), pp. 449–465.

[Che+15] Yu Cheng, Felix X Yu, Rogerio S Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. “An exploration of parameter redundancy in deep networks with circulant projections.” In: *ICCV*. 2015, pp. 2857–2865.

[CJM20] Lu Chi, Borui Jiang, and Yadong Mu. “Fast Fourier Convolution.” In: *Advances in Neural Information Processing Systems* 33 (2020).

[Cho+14] Kyunghyun Cho, Bart van Merrienboer, Caglar Gülcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.” In: *EMNLP*. 2014.

[CAB17] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. "Hierarchical multiscale recurrent neural networks." In: *ICLR*. 2017.

[Col+19] Elizabeth K Cole, Joseph Y Cheng, John M Pauly, and Shreyas S Vasanawala. "Complex-valued convolutional neural networks for MRI reconstruction." In: *Proceedings of the International Society of Magnetic Resonance in Medicine* (2019).

[CK19] F. Cotter and N. Kingsbury. "A Learnable Scatternet: Locally Invariant Convolutional Layers." In: *2019 IEEE International Conference on Image Processing (ICIP)*. 2019, pp. 350–354.

[Cot19] Fergal Cotter. "Uses of Complex Wavelets in Deep Convolutional Neural Networks." PhD thesis. Trumpington Street, Cambridge CB2 1PZ, United Kingdom: Department of Engineering, University of Cambridge, Aug. 2019.

[CBD15] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "Binaryconnect: Training deep neural networks with binary weights during propagations." In: *NIPS*. 2015.

[Dan+16a] I. Danihelka, G. Wayne, B. Uria, N. Kalchbrenner, and A. Graves. "Associative long short-term memory." In: *ICML*. 2016.

[Dan+16b] Ivo Danihelka, Greg Wayne, Benigno Uria, Nal Kalchbrenner, and Alex Graves. "Associative Long Short-Term Memory." In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 2016, pp. 1986–1994.

[Dav79] Davis. *Circulant Matrices*. John Wiley and Sons, 1979.

[Den+13] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas. "Predicting parameters in deep learning." In: *NIPS*. 2013, pp. 2148–2156.

[Den+14] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. "Exploiting linear structure within convolutional networks for efficient evaluation." In: *NIPS*. 2014, pp. 1269–1277.

[Dev+18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." In: *arXiv preprint arXiv:1810.04805* (2018).

[DS14] Sander Dieleman and Benjamin Schrauwen. "End-to-end learning for music audio." In: *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE. 2014.

[DAC16] Boubacar Doucoure, Kodjo Agbossou, and Alben Cardenas. "Time series prediction using artificial wavelet neural network and multi-resolution analysis: Application to wind speed data." In: *Renewable Energy* 92 (2016), pp. 202–211.

[DV16] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning." In: *ArXiv e-prints* (2016). eprint: `1603.07285`.

[DSW20] Tarik Dzanic, Karan Shah, and Freddie Witherden. "Fourier Spectrum Discrepancies in Deep Network Generated Images." In: *Advances in neural information processing systems* 33 (2020).

[Elm90] Jeffrey L Elman. "Finding structure in time." In: *Cognitive science* 14.2 (1990), pp. 179–211.

[Ene+20] Kristina Enes, Hassan Errami, Moritz Wolter, Tim Krake, Bernhard Eberhardt, Andreas Weber, and Jörg Zimmermann. "Unsupervised and Generic Short-Term Anticipation of Human Body Motions." In: *Sensors* 20.4 (2020), p. 976.

[Est+18] Carlos Esteves, Christine Allen-Blanchette, Xiaowei Zhou, and Kostas Daniilidis. "Polar Transformer Networks." In: *International Conference on Learning Representations*. 2018. URL: `https://openreview.net/forum?id=HktRlUlAZ`.

[FB19] Hafez Farazi and Sven Behnke. "Frequency Domain Transformer Networks for Video Prediction." In: *27th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*. 2019.

[Fra+20] L Franken, B Georgiev, S Muecke, M Wolter, N Piatkowski, and C Bauckhage. "Gradient-free quantum optimization on NISQ devices." In: *arXiv preprint arXiv:2012.13453* (2020).

[GA14] Michael S Gashler and Stephen C Ashmore. "Training deep fourier neural networks to fit time-series data." In: *International Conference on Intelligent Computing*. Springer. 2014, pp. 48–55.

[Geh+17] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y.N. Dauphin. "Convolutional Sequence to Sequence Learning." In: *ICML*. 2017.

[GK92] G. Georgiou and C. Koutsougeras. "Complex domain backpropagation." In: *IEEE Trans. Circuits and systems II: Analog and Digital Signal Processing* 39.5 (1992), pp. 330–334.

[GES02] Felix A Gers, Douglas Eck, and Jürgen Schmidhuber. "Applying LSTM to time series predictable through time-window approaches." In: *Neural Nets WIRN Vietri-01*. Springer, 2002, pp. 193–200.

[GB10] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *AISTATS*. 2010.

[GG18] L. B. Godfrey and M. S. Gashler. "Neural Decomposition of Time-Series Data for Effective Generalization." In: *IEEE Transactions on Neural Networks and Learning Systems* 29.7 (2018), pp. 2973–2985. ISSN: 2162-237X. DOI: `10.1109/TNNLS.2017.2709324`.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[GO19] Hiroaki Goto and Yuko Osana. "Chaotic Complex-Valued Associative Memory with Adaptive Scaling Factor Independent of Multi-values." In: *International Conference on Artificial Neural Networks*. Springer. 2019, pp. 76–86.

[Gra12] Alex Graves. "Supervised sequence labelling." In: *Supervised sequence labelling with recurrent neural networks*. Springer, 2012, pp. 5–13.

[Gre+16] Klaus Greff, Rupesh K Srivastava, Jan Koutnik, Bas R Steunebrink, and Jürgen Schmidhuber. "LSTM: A search space odyssey." In: *IEEE transactions on neural networks and learning systems* 28.10 (2016), pp. 2222–2232.

[GL84] D. Griffin and J. Lim. "Signal estimation from modified short-time Fourier transform." In: *IEEE Transactions on Acoustics, Speech, and Signal Processing*. 1984.

[Grö13] Karlheinz Gröchenig. *Foundations of time-frequency analysis*. Springer Science & Business Media, 2013.

[Gub16] N. Guberman. *On Complex Valued Convolutional Neural Networks*. Tech. rep. The Hebrew University of Jerusalem Israel, 2016.

[HMD16a] Song Han, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." In: *International Conference on Learning Representations*. 2016.

[HMD16b] Song Han, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." In: *ICLR*. 2016.

[Han+15] Song Han, Jeff Pool, John Tran, and William Dally. "Learning both weights and connections for efficient neural networks." In: *NIPS*. 2015, pp. 1135–1143.

[Han+14] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. "Deep speech: Scaling up end-to-end speech recognition." In: *arXiv preprint arXiv:1412.5567* (2014).

[Har78] Fredric J Harris. "On the use of windows for harmonic analysis with the discrete Fourier transform." In: *Proceedings of the IEEE* 66.1 (1978), pp. 51–83.

[He+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In: *CVPR*. 2016, pp. 770–778.

[Hec92] Robert Hecht-Nielsen. "Theory of the backpropagation neural network." In: *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[HSS12] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent." In: (2012).

[Hir13] A. Hirose. *Complex-valued neural networks: Advances and applications*. John Wiley & Sons, 2013.

[Hir12] Akira Hirose. *Complex-valued neural networks*. Vol. 400. Springer Science & Business Media, 2012.

[HO99] Akira Hirose and Hirofumi Onishi. "Proposal of relative-minimization learning for behavior stabilization of complex-valued recurrent neural networks." In: *Neurocomputing* 24.1-3 (1999), pp. 163–171.

[HS97a] S. Hochreiter and J. Schmidhuber. "Long Short Term Memory." In: *Neural Computation* (1997).

[Hoc91] Sepp Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen." In: *Diploma, Technische Universität München* 91.1 (1991).

[Hoc+01] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*. 2001.

[HS97b] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[Hut18] Matthew Hutson. "Artificial intelligence faces reproducibility crisis." In: *Science* 359.6377 (2018), pp. 725–726. ISSN: 0036-8075. DOI: 10.1126/science.359.6377.725. eprint: https://science.sciencemag.org/content/359/6377/725.full.pdf. URL: https://science.sciencemag.org/content/359/6377/725.

[Hut20] Matthew Hutson. "Core progress in AI has stalled in some fields." In: *Science* 368.6494 (2020), pp. 927–927. ISSN: 0036-8075. DOI: 10.1126/science.368.6494.927. eprint: https://science.sciencemag.org/content/368/6494/927.full.pdf. URL: https://science.sciencemag.org/content/368/6494/927.

[HR17] S. Hyland and G. Rätsch. "Learning Unitary Operators with Help From u(n)." In: *AAAI*. 2017.

[Ioa+17] Yani Ioannou, Duncan Robertson, Roberto Cipolla, and Antonio Criminisi. "Deep roots: Improving cnn efficiency with hierarchical filter groups." In: *CVPR*. 2017.

[Ion+14] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. "Human3.6M: Large Scale Datasets and Predictive Methods for 3D Human Sensing in Natural Environments." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.7 (2014), pp. 1325–1339.

[JVZ14a] Max Jaderberg, A. Vedaldi, and Andrew Zisserman. "Speeding up Convolutional Neural Networks with Low Rank Expansions." In: *ArXiv* abs/1405.3866 (2014).

[JVZ14b] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. "Speeding up convolutional neural networks with low rank expansions." In: *arXiv preprint arXiv:1405.3866* (2014).

[Jai+16] A. Jain, A.R. Zamir, S. Savarese, and A. Saxena. "Structural-RNN: Deep learning on spatio-temporal graphs." In: *CVPR*. 2016.

[JC01] Arne Jensen and Anders la Cour-Harbo. *Ripples in mathematics: the discrete wavelet transform*. Springer Science & Business Media, 2001.

[Jin+18] L. Jing, C. Gulcehre, J. Peurifoy, Y. Shen, M. Tegmark, M. Solja, and Y. Bengio. "Gated Orthogonal Recurrent Units: On Learning to Forget." In: *AAAI Workshops*. 2018.

[Jin+17] L. Jing, Y. Shen, T. Dubček, J. Peurifoy, S. Skirlo, Y. LeCun, M. Tegmark, and M. Soljačić. "Tunable efficient unitary neural networks (EUNN) and their application to RNNs." In: *ICML*. 2017.

[KM94] George Kechriotis and Elias S Manolakos. “Training fully recurrent neural networks with complex weights.” In: *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 41.3 (1994), pp. 235–238.

[KA01] T. Kim and T. Adali. “Complex backpropagation neural network using elementary transcendental activation functions.” In: *ICASSP*. 2001.

[KB15] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015.

[KS16] H. Koppula and A. Saxena. “Anticipating human activities using object affordances for reactive robotic response.” In: *IEEE Trans. Pattern Analysis and Machine Intelligence* 38.1 (2016), pp. 14–29.

[Kou+14] Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. “A Clockwork RNN.” In: *International Conference on Machine Learning*. 2014, pp. 1863–1871.

[KGP02] L. Kovar, M. Gleicher, and F. Pighin. “Motion graphs.” In: *ACM Tran. Graphics (TOG)* 21.3 (2002), pp. 473–482.

[Kre09] K. Kreutz-Delgado. “The complex gradient operator and the CR-calculus.” In: *arXiv preprint:0906.4835* (2009).

[Kri18] Raghuraman Krishnamoorthi. “Quantizing deep convolutional networks for efficient inference: A whitepaper.” In: *arXiv preprint arXiv:1806.08342* (2018).

[KSH12a] A. Krizhevsky, I. Sutskever, and G. Hinton. “Imagenet classification with deep convolutional neural networks.” In: *NIPS*. 2012.

[KSH12b] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks.” In: *NIPS*. 2012, pp. 1097–1105.

[Kus+18] Aditya Kusupati, Manish Singh, Kush Bhatia, Ashish Kumar, Prateek Jain, and Manik Varma. “Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network.” In: *Advances in Neural Information Processing Systems*. 2018, pp. 9017–9028.

[LOK97] Kieran G Larkin, Michael A Oldfield, and Hanno Klemm. “Fast Fourier method for the accurate rotation of sampled images.” In: *Optics Communications* 139.1-3 (1997), pp. 99–106.

[LSS13] Quoc Le, Tamás Sarlós, and Alex Smola. “Fastfood-approximating kernel expansions in loglinear time.” In: *ICML*. Vol. 85. 2013.

[Leb+14] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. "Speeding-up convolutional neural networks using fine-tuned cp-decomposition." In: *arXiv preprint arXiv:1412.6553* (2014).

[LeC+98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[LDS90] Yann LeCun, John S Denker, and Sara A Solla. "Optimal brain damage." In: *NIPS*. 1990, pp. 598–605.

[LH91] H. Leung and S. Haykin. "The complex backpropagation algorithm." In: *IEEE Trans. Signal Processing* 39.9 (1991), pp. 2101–2104.

[LKJ15] Fei-Fei Li, Andrej Karpathy, and Justin Johnson. "Cs231n: Convolutional neural networks for visual recognition." In: *University Lecture* (2015).

[LA08] H. Li and T. Adali. "Complex-Valued Adaptive Signal Processing Using Nonlinear Functions." In: *EURASIP Journal on Advances in Signal Processing* (2008).

[Lin+16] Shaohui Lin, Rongrong Ji, Xiaowei Guo, and Xuelong Li. "Towards Convolutional Neural Networks Compression via Global Error Reconstruction." In: *IJCAI*. 2016.

[Lin+18] Shaohui Lin, Rongrong Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and Baochang Zhang. "Accelerating Convolutional Networks via Global & Dynamic Filter Pruning." In: *IJCAI*. 2018, pp. 2425–2432.

[Lin+96] Tsungnan Lin, Bill G Horne, Peter Tino, and C Lee Giles. "Learning long-term dependencies in NARX recurrent neural networks." In: *IEEE Transactions on Neural Networks* 7.6 (1996), pp. 1329–1338.

[Li079] J. Liouville. "Leçons sur les fonctions doublement périodiques." In: *Journal für die reine und angewandte Mathematik / Zeitschriftenband (1879)* (1879).

[MHN13] A. Maas, A. Hannun, and A. Ng. "Rectifier nonlinearities improve neural network acoustic models." In: *ICML*. 2013.

[Mal12] Stéphane Mallat. "Group invariant scattering." In: *Communications on Pure and Applied Mathematics* 65.10 (2012), pp. 1331–1398.

[MG09] D.P. Mandic and V.S.L. Goh. *Complex valued nonlinear adaptive filters: noncircularity, widely linear and neural models*. John Wiley & Sons, 2009.

[Mao+19] Wei Mao, Miaomiao Liu, Mathieu Salzmann, and Hongdong Li. "Learning trajectory dependencies for human motion prediction." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 9489–9497.

[MSM93] Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. "Building a large annotated corpus of English: The Penn Treebank." In: (1993).

[Mar+19] F. Marra, D. Gragnaniello, L. Verdoliva, and G. Poggi. "Do GANs Leave Artificial Fingerprints?" In: *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*. 2019, pp. 506–511. DOI: 10.1109/MIPR.2019.00103.

[MBR17] J. Martinez, M.J. Black, and J. Romero. "On human motion prediction using recurrent neural networks." In: *CVPR*. 2017.

[MHL13] Michael Mathieu, Mikael Henaff, and Yann LeCun. "Fast training of convolutional networks through ffts." In: *arXiv preprint arXiv:1312.5851* (2013).

[MZE09] Rittavee Matungka, Yuan F Zheng, and Robert L Ewing. "Image registration using adaptive polar transform." In: *IEEE Transactions on Image Processing* 18.10 (2009), pp. 2340–2354.

[Mik12] T. Mikolov. *Statistical Language Models Based on Neural Networks*. Tech. rep. Brno University of Technology, 2012.

[MNT99] K. Minami, H. Nakajima, and T. Toyoshima. "Real-time discrimination of ventricular tachyarrhythmia with Fourier-transform neural network." In: *IEEE Transactions on Biomedical Engineering* 46.2 (1999), pp. 179–185. ISSN: 0018-9294. DOI: 10.1109/10.740880.

[NH10] V. Nair and G. E. Hinton. "Rectified linear units improve restricted Boltzmann machines." In: *ICML*. 2010.

[Nar+17] Sharan Narang, Erich Elsen, Gregory Diamos, and Shubho Sengupta. "Exploring sparsity in recurrent neural networks." In: *arXiv preprint arXiv:1704.05119* (2017).

[NYC16] Anh Nguyen, Jason Yosinski, and Jeff Clune. "Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks." In: *Visualization for Deep Learning workshop, International Conference in Machine Learning* (2016). arXiv preprint arXiv:1602.03616.

[Nie15] Michael A Nielsen. *Neural networks and deep learning*. Vol. 2018. Determination press, San Francisco, CA, 2015.

[Nov+15] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. "Tensorizing neural networks." In: *NIPS*. 2015, pp. 442–450.

[Nvi20] Nvidia. *NVIDIA Developer Fourier transforms in convolution*. `https://developer.nvidia.com/discover/convolution`. Accessed: 2020-03-25. 2020.

[Pan+19] Yu Pan, Jing Xu, Maolin Wang, Jinmian Ye, Fei Wang, Kun Bai, and Zenglin Xu. "Compressing recurrent neural networks with tensor ring for action recognition." In: *AAAI*. Vol. 33. 2019, pp. 4683–4690.

[Pan99] Erwin Pang. "Parameter estimation and efficient implementation of affine transforms for digital images." PhD thesis. National Library of Canada, 1999.

[PMB13] R. Pascanu, T. Mikolov, and Y. Bengio. "On the difficulty of training recurrent neural networks." In: *ICML*. 2013.

[Pas+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. "PyTorch: An imperative style, high-performance deep learning library." In: *Advances in Neural Information Processing Systems*. 2019, pp. 8024–8035.

[Pop35] Karl Popper. *Logik der Forschung*. Vol. 9. Springer-Verlag Wien GmbH, 1935.

[Pop59] Karl Popper. *The Logic of Scientific Discovery*. Hutchinson &Co., 1959.

[Pra+17] Harry Pratt, Bryan Williams, Frans Coenen, and Yalin Zheng. "FCNN: Fourier convolutional neural networks." In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2017, pp. 786–798.

[PPL20] Paul J. Pritz, D. Pérez, and K. K. Leung. "Fast-Fourier-Forecasting Resource Utilisation in Distributed Systems." In: *ArXiv* abs/2001.04281 (2020).

[Ras+16] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. "Xnor-net: Imagenet classification using binary convolutional neural networks." In: *ECCV*. Springer. 2016, pp. 525–542.

[Rav+17] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio. "Improving Speech Recognition by Revising Gated Recurrent Units." In: *INTERSPEECH*. 2017.

[RM18] Daniel Recoskie and Richard Mann. "Learning sparse wavelet representations." In: *arXiv preprint arXiv:1802.02961* (2018).

[RC96] B Srinivasa Reddy and Biswanath N Chatterji. "An FFT-based technique for translation, rotation, and scale-invariant image registration." In: *IEEE Transactions on Image Processing* 5.8 (1996), pp. 1266–1271.

[RS14] D.P. Reichert and T. Serre. "Neuronal synchrony in complex-valued deep networks." In: *ICLR*. 2014.

[Ren+15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks." In: *NIPS*. 2015, pp. 91–99.

[RSA15] Oren Rippel, Jasper Snoek, and Ryan P Adams. "Spectral representations for convolutional neural networks." In: *Advances in neural information processing systems*. 2015, pp. 2449–2457.

[RG13] Raif Rustamov and Leonidas J Guibas. "Wavelets on graphs via deep learning." In: *Advances in neural information processing systems*. 2013, pp. 998–1006.

[RYL18] Jongbin Ryu, Ming-Hsuan Yang, and Jongwoo Lim. "DFT-based Transformation Invariant Pooling Layer for Visual Classification." In: *ECCV*. 2018.

[Ser+17] Iulian Vlad Serban, Tim Klinger, Gerald Tesauro, Kartik Talamadupula, Bowen Zhou, Yoshua Bengio, and Aaron Courville. "Multiresolution recurrent neural networks: An application to dialogue response generation." In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[Shi+17] Xingjian Shi, Zhihan Gao, Leonard Lausen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. "Deep learning for precipitation nowcasting: A benchmark and a new model." In: *Advances in Neural Information Processing Systems (NIPS)*. 2017.

[Sit+20] Vincent Sitzmann, Julien NP Martel, Alexander W Bergman, David B Lindell, and Gordon Wetzstein. "Implicit neural representations with periodic activation functions." In: *arXiv preprint arXiv:2006.09661* (2020).

[SMS15] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. "Unsupervised learning of video representations using LSTMs." In: *International Conference on Machine Learning (ICML)*. 2015.

[SGS15a] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. "Training very deep networks." In: *Advances in neural information processing systems* 28 (2015), pp. 2377–2385.

[SGS15b] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. "Highway networks." In: *arXiv preprint arXiv:1505.00387* (2015).

[Str06] Strang. *Linear algebra*. MIT Press, 2006.

[SN96] Gilbert Strang and Truong Nguyen. *Wavelets and filter banks*. SIAM, 1996.

[Tag11] H. Tagare. *Notes on Optimization on Stiefel Manifolds*. Tech. rep. Yale University, 2011.

[Tan+20] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. "Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains." In: *arXiv preprint arXiv:2006.10739* (2020).

[THK17] J. Thickstun, Z. Harchaoui, and S.M. Kakade. "Learning Features of Music from Scratch." In: *ICLR*. 2017.

[Thi+18] John Thickstun, Zaid Harchaoui, Dean P Foster, and Sham M Kakade. "Invariances and data augmentation for supervised music transcription." In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 2241–2245.

[TSN17] Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. "Compressing recurrent neural network with tensor train." In: *IJCNN*. IEEE. 2017, pp. 4451–4458.

[Tra+18] C. Trabelsi, O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, J. F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C.J. Pal. "Deep Complex Networks." In: *ICLR*. 2018.

[Van94] A Van Den Bos. "Complex gradient and Hessian." In: *IEE Proceedings-Vision, Image and Signal Processing* 141.6 (1994), pp. 380–382.

[Vas+15] Nicolas Vasilache, Jeff Johnson, Michael Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. "Fast convolutional nets with fbfft: A GPU performance evaluation." In: *ICLR*. 2015.

[Vas+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In: *Advances in neural information processing systems* 30 (2017), pp. 5998–6008.

[VYL17] P. Virtue, S.X. Yu, and M. Lustig. "Better than real: Complex-valued neural nets for MRI fingerprinting." In: *ICIP*. 2017.

[VSL17] Patrick Virtue, X Yu Stella, and Michael Lustig. "Better than real: Complex-valued neural nets for MRI fingerprinting." In: *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2017, pp. 3953–3957.

[Wan+18a] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. "Video-to-Video Synthesis." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[Wan+18b] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. "Non-local neural networks." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7794–7803.

[Wan+17] Zelong Wang, Qiang Lan, Hongjun He, and Chunyuan Zhang. "Winograd Algorithm for 3D Convolution Neural Networks." In: *ICANN*. 2017.

[WLW17] Zhisheng Wang, Jun Lin, and Zhongfeng Wang. "Accelerating recurrent neural networks: A memory-efficient approach." In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.10 (2017), pp. 2763–2775.

[Wen+17] Wei Wen, Yuxiong He, Samyam Rajbhandari, Minjia Zhang, Wenhan Wang, Fang Liu, Bin Hu, Yiran Chen, and Hai Li. "Learning intrinsic sparse structures within long short-term memory." In: *arXiv preprint arXiv:1709.05027* (2017).

[WL18] Travis Williams and Robert Li. "Wavelet pooling for convolutional neural networks." In: *International Conference on Learning Representations*. 2018.

[Wir27] W. Wirtinger. "Zur formalen Theorie der Funktionen von mehr komplexen Veränderlichen." In: *Mathematische Annalen* (1927).

[Wis+16] Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. "Full-capacity unitary recurrent neural networks." In: *NIPS*. 2016.

[Wol17] Moritz Wolter. *Building an end-to-end speech recognizer*. KU Leuven, 2017.

[WGY20] Moritz Wolter, Juergen Gall, and Angela Yao. "Sequence Prediction using Spectral RNNs." In: *29th International Conference on Artificial Neural Networks*. 2020.

[WLY20] Moritz Wolter, Shaohui Lin, and Angela Yao. "Neural network compression via learnable wavelet transforms." In: *29th International Conference on Artificial Neural Networks (ICANN)*. 2020.

[WY18] Moritz Wolter and Angela Yao. "Complex Gated Recurrent Neural Networks." In: *NIPS*. 2018.

[WYB20] Moritz Wolter, Angela Yao, and Sven Behnke. "Object-centered Fourier Motion Estimation and Segment-Transformation Prediction." In: *28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. 2020.

[Xie+03] Hongjie Xie, Nigel Hicks, G Randy Keller, Haitao Huang, and Vladik Kreinovich. "An IDL/ENVI implementation of the FFT-based algorithm for automatic image registration." In: *Computers & Geosciences* 29.8 (2003), pp. 1045–1055.

[Xin+15] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. "Convolutional LSTM network: A machine learning approach for precipitation nowcasting." In: *Advances in Neural Information Processing Systems (NIPS)*. 2015.

[YKT17] Yinchong Yang, Denis Krompass, and Volker Tresp. "Tensor-train recurrent neural networks for video classification." In: *ICML*. JMLR. org. 2017, pp. 3891–3900.

[Yan+15] Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. "Deep fried convnets." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1476–1483.

[YGV12] A. Yao, J. Gall, and L. Van Gool. "Coupled action recognition and pose estimation from multiple views." In: *International Journal of Computer Vision* 100.1 (2012), pp. 16–37.

[Ye+18] Jinmian Ye, Linnan Wang, Guangxi Li, Di Chen, Shandian Zhe, Xinqi Chu, and Zenglin Xu. "Learning compact recurrent neural networks with block-term tensor decomposition." In: *CVPR*. 2018, pp. 9378–9387.

[Zei12] Eberhard Zeidler. *Springer-Handbuch der Mathematik I: Begründet von IN Bronstein und KA Semendjaew Weitergeführt von G. Grosche, V. Ziegler und D. Ziegler Herausgegeben von E. Zeidler*. Vol. 1. Springer-Verlag, 2012.

[Zei+13] M. Zeiler et al. "On rectified linear units for speech processing." In: *ICASSP*. 2013.

[Zha+18] Jiong Zhang, Yibo Lin, Zhao Song, and Inderjit S Dhillon. "Learning long term dependencies via fourier recurrent units." In: *arXiv preprint arXiv:1803.06585* (2018).

[Zha01] Ying-Quian Zhang. "ForeNet: Fourier Recurrent Neural Networks for Time Series Prediction." PhD thesis. The Chinese University of Hong Kong, 2001.

[ZZC08] Wei Zuo, Yang Zhu, and Lilong Cai. “Fourier-neural-network-based learning control for a class of nonlinear systems with flexible components.” In: *IEEE transactions on neural networks* 20.1 (2008), pp. 139–151.

ERKLÄRUNG

Ich versichere, dass ich die von mir vorgelegte Dissertation selbständig angefertigt, die benutzten Quellen und Hilfsmittel vollständig angegeben und die Stellen der Arbeit einschließlich Tabellen und Abbildungen, die anderen Werken im Wortlaut oder dem Sinn nach entnommen sind, in jedem Einzelfall als Entlehnung kenntlich gemacht habe; dass diese Dissertation noch keiner anderen Fakultät oder Universität zur Prüfung vorgelegen hat; dass sie noch nicht veröffentlicht worden ist sowie, dass ich eine solche Veröffentlichung vor Abschluss des Promotionsverfahrens nicht vornehmen werde. Die Bestimmungen dieser Promotionsordnung sind mir bekannt. Die von mir vorgelegte Dissertation ist von Prof. Dr. Angela Yao betreut worden.

Bonn, Januar 2021

Moritz Wolter