FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of master's thesis

Title:	Recurrent Memory Models with Optimal Polynomial Projections
Student:	Bc. Ondřej Naňka
Supervisor:	Ing. Daniel Vašata, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

A central problem in learning from sequential data is representing cumulative history in an incremental fashion as more data is processed. The ordinary recurrent neural network suffers from a limited memory horizon. Several heuristics were proposed to overcome this, such as gates in the successful LSTM and GRU, or higher-order frequencies in the recent Fourier Recurrent Unit and Legendre Memory Unit. The most recent result in online function approximation problem field is given by a high-order polynomial projection operators (HiPPO) framework for the online compression of discrete and continuous signals by projection onto polynomial bases.

The aim of this thesis is to research the practical usability of this framework. The student should implement this HiPPO-LegS update mechanism within this framework a test it on at least two publicly available sequential datasets. The results should be compared to other state-of-the-art approaches applied to those datasets and properly discussed.

Electronically approved by Ing. Karel Klouda, Ph.D. on 11 February 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Recurrent Memory Models with Optimal Polynomial Projections

Bc. Ondřej Naňka

Department of Applied Mathematics
Supervisor: Ing. Daniel Vašata, Ph.D.

May 6, 2021

Acknowledgements

I would like to thank my supervisor, friends, family, girlfriend and my dog for their endless patience and generous support.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 6, 2021 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Naňka, Ondřej. *Recurrent Memory Models with Optimal Polynomial Projections*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Cílem této práce je prozkoumat možnosti praktického využití komprese signálu projekcí do polynomiálních bází při implementaci rekurentních neuronových sítí. Praktická část práce se zabývá klasifikací zvukových signálů a zpracováním textu pomocí frameworku Tensorflow a implementací jako "Spiking Neural Network" pomocí simulátoru NengoDL.

Klíčová slova LSTM,LMU,HIPPO,Polynomy,Optimální Projekce,Rekurentní neuronová sít,Klasifikace zvuku, NLP

Abstract

The aim of this thesis is to research the practical usability of high-order polynomial projection operators for compression of signals by projection onto polynomial bases for implementation of recurrent neural networks. Experiments in the field of sound classification and natural language processing are performed using Tensorflow framework and also as a spiking neural network using a simulator NengoDL.

Keywords LSTM,LMU,HIPPO,Polynomials,Optimal Projections,RNN,Audio classification, NLP

Contents

List of Figures

Introduction

Motivation

Sequence Learning

Many things in life vary trough time and therefore, represent a sequence. To perform machine learning on sequential data (text, speech, etc.), one could use a regular neural [12] network and feed it the entire sequence, but the input size of our data would be fixed, which is quite limiting. Other problems with this approach occur if important events in a sequence lie just outside of the input window. What would be actually useful is to have a network to which we can feed sequences of arbitrary length, one element of the sequence per time step (for example, a video is just a sequence of images; we feed the network one image at a time); and a network which has a type of memory to remember important events which happened over many time steps in the past. These problems and requirements have led to a variety of different recurrent neural networks [13].

Learning long-range dependencies in timeseries

One of the architectures for tasks that require learning long-range temporal dependencies in sequential data or time series data like speech recognition, named entity recognition, or machine translation is called recurrent neural network (RNN). One of the currently most popular architectures that have proved to be successful in modelling complex temporal relationships is the Long-Short-Term-Memory(LSTM). This kind of architecture is commonly used and incorporated into popular applications such as Siri, voice search, and Google Translate. Like other types of neural networks, recurrent neural networks utilize training data to learn. The special thing about these architectures is the memory - they are able to take information from prior inputs and take it into account when evaluating the current input and output. Most of the traditional deep neural networks assume that inputs and outputs are independent

of each other, while the outputs of recurrent neural networks depend on the prior elements within the sequence. While future events would also be helpful in determining the output of a given sequence, unidirectional recurrent neural networks cannot account for these events in their predictions [14].

When using the LSTMs, one has to be aware of their limitations in terms of time steps, specifically LSTMs have a memory of about T = 500–5,000 time steps [15]. That may be enough for some applications, but in general signals in realistic natural environments are continuous in time. It is unclear how existing RNNs can deal with conditions as $T \to \infty$. This limitation is a problem for models that must leverage long-range dependencies within an ongoing stream of continuous-time data [4].

Actually, mammalian brains are able to do much better regarding leveraging long-range dependencies in information processing tasks even though the data brain process are of continuous nature. Research in the field of neuroscience uncovers that biological nervous systems possess mechanisms that allow them to solve problems beyond the capabilities of currently widely used architectures like LSTM or GRU related to the processing of continuous-time information. Neurons in the brain transmit information using spikes and filter those spikes continuously over time through synaptic connections. Based on this work in the field of neuroscience, a novel memory cell LMU based on orthogonalization of the continuous-time history of its input signal was proposed in 2019, and in late 2020 it was identified as a member of a broader family of architectures united by HiPPO framework. Both of those works are achieving state-of-the-art performance on PSMNIST dataset in the domain of RNNs, which is the standard benchmark for RNN networks [4].

The aim of this work is to explore the practical usability of these novel approaches in the domain of audio classification, named entity recognition, and sentiment classification as audio and text data are naturally sequential and can be treated as timeseries.

Chapter 1

Recurrent neural networks

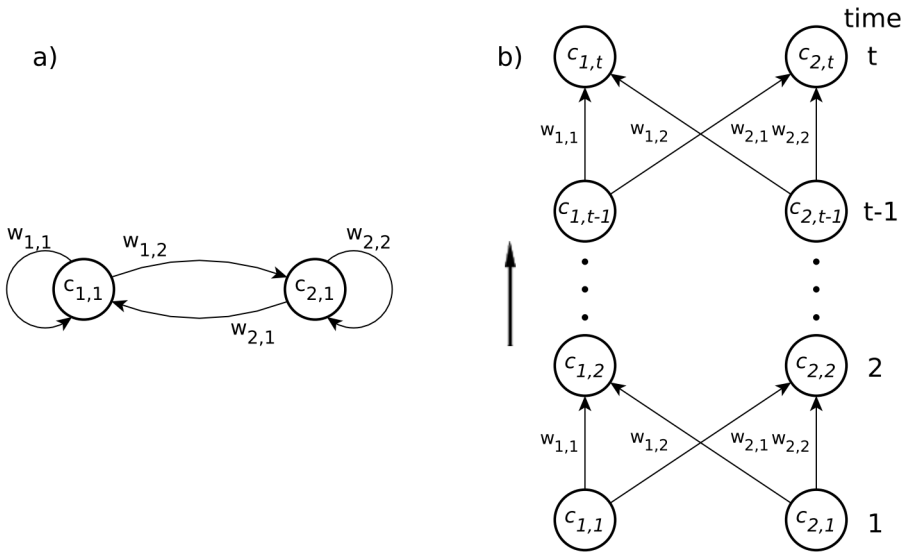
In this chapter, recurrent neural networks will be explained in greater detail, we will try to demonstrate what are the limitations of the baseline RNN architecture and the two essential improvements in the form of LSTM and GRU which as we will later discuss actually both can be interpreted as a special case of the novel approaches presented in the chapter 2.

1.1 Basic architecture

To extend the basic feed-forward neural networks to Recurrent Neural Networks, it is necessary to make the feeding of signals from previous timesteps back into the network possible. These networks with recurrent connections are called Recurrent Neural Networks (RNN) [16, 17]. The basic RNNs are limited to look back in time for approximately ten timesteps [18]. This is because the fed-back signal is either vanishing or exploding. This issue was addressed with Long Short-Term Memory Recurrent Neural Networks (LSTM-RNN) [19, 1, 20, 21]. LSTM networks are capable to learn more than 1,000 timesteps, depending on the specific problem and model. [15].

Recurrent neural networks (RNNs) [16, 17] are dynamic systems, which means that they have an internal state during each of the timesteps when performing inference. This is allowed by circular connections between higher- and lower-layer neurons and optional self-feedback connections. The presence of a feedback connection is what enables data propagation from the data in the processed sequence that occurred in the earlier processing timesteps. This can be perceived as the memory of previous timesteps which is preserved in its internal state [2].

Figure 1.1: In Figure a we can see a simple fully recurrent neural network with a two neuron layer. The same network unfolded over time with a separate layer for each time step is shown in Figure b. The representation in Figure b is a feed-forward neural network[1].



1.2 Training recurrent networks

To understand the limitations of the basic RNN models, it is necessary to understand how they are trained. The most common way to train recurrent neural networks is using methods called Backpropagation Through Time (BPTT) [16, 17, 22] and Real-Time Recurrent Learning (RTRL) [17, 23]. The main difference between BPTT and RTRL is the way the weight changes are calculated. The original formulation of LSTM-RNNs used a combination of BPTT and RTRL [17, 2].

1.2.1 Backpropagation Through Time

The backpropagation through time algorithm makes use of the fact that, for a finite period of time, there is a feedforward neural network with identical behaviour for every RNN. It is possible to unfold a recurrent network in time to obtain a new neural network that is feedforward.

Figure 1.1 shows a simple, fully recurrent neural network with a single two-neuron layer and the corresponding feed-forward neural network. It is clear that it requires a separate layer for each time step with the same weights for all layers. If the weights are identical to the RNN, both networks show the same behaviour. The unfolded network can be trained using the backpropagation

algorithm. At the end of a training sequence, the network is unfolded in time. The error is calculated for the output units with existing target values using some chosen error measure. The error is injected backwards into the network and the weights are updated for all time steps calculated. The weights in the recurrent version of the network are updated with the sum of its deltas over all time steps [2].

For a single unit, the error signal is calculated for all time steps in a single pass, using the following iterative backpropagation algorithm. Considering the discrete time steps indexed by the variable τ. Assuming that the network starts at a point in time t' and end at t. The time between t' and t is called an epoch. U is the set of non input units, and f_u is the differentiable, non-linear squashing function of the unit $u \in U$; the output $y_u(\tau)$ of u at time τ is given by

$$y_u(\tau) = \mathbf{f}_u\left(z_u(\tau)\right),$$

where the weighted input is given as

$$\begin{aligned} z_u(\tau+1) &= \sum_l W_{[u,l]} X_{[l,u]}(\tau+1), \qquad \text{with } l \in \mathrm{Pre}(u) \\ &= \sum_v W_{[u,v]} y_v(\tau) + \sum_i W_{[u,i]} y_i(\tau+1), \end{aligned}$$

where $v \in U \cap \mathrm{Pre}\ (u)$ and $i \in I$, the set of input units. It is important to note that there are two different types of inputs to u at time $\tau+1$. The input that arrives at time $\tau+1$ via the input units, and the recurrent output from all non-input units in the network produced at time τ. If the network is fully connected, then $U \cap \mathrm{Pre}\ (u)$ is equal to the set U of non-input units. Let $T(\tau)$ be the set of non-input units for which, at time τ, the output value $y_u(\tau)$ of the unit $u \in T(\tau)$ should match some target value $d_u(\tau)$. The cost function is the summed error $E_{\text{total}}\ (t', t)$ for the epoch $t', t'+1, \ldots, t$, which we want to minimise using a learning algorithm. The total error is defined by

$$E_{\text{total}}\ (t', t) = \sum_{\tau=t'}^{t} E(\tau),$$

with the error $E(\tau)$ at time τ defined using the squared error as an objective function by

$$E(\tau) = \frac{1}{2} \sum_{u \in U} \left(e_u(\tau)\right)^2,$$

and with the error $e_u(\tau)$ of the non-input unit u at time τ defined by

$$e_u(\tau) = \begin{cases} d_u(\tau) - y_u(\tau) & \text{if } u \in T(\tau) \\ 0 & \text{otherwise} \end{cases}.$$

To adjust the weights, we use the error signal $\vartheta_u(\tau)$ of a non-input unit u at a time τ, which is defined by

$$\vartheta_u(\tau) = \frac{\partial E(\tau)}{\partial z_u(\tau)}.$$

When we unroll ϑ_u over time, we obtain the equality

$$\vartheta_u(\tau) = \begin{cases} f'_u\left(z_u(\tau)\right) e_u(\tau) & \text{if } \tau = t \\ f'_u\left(z_u(\tau)\right) \left(\sum_{k \in U} W_{[k,u]} \vartheta_k(\tau + 1)\right) & \text{if } t' \leq \tau < t \end{cases}.$$

After the backpropagation computation is performed down to time t', we calculate the weight update $\Delta W_{[u,v]}$ in the recurrent version of the network. This is done by summing the corresponding weight updates for all time steps:

$$\Delta W_{[u,v]} = -\eta \frac{\partial E_{\text{total}}\ (t', t)}{\partial W_{[u,v]}},$$

with

$$\begin{aligned} \frac{\partial E_{\text{total}}\ (t', t)}{\partial W_{[u,v]}} &= \sum_{\tau=t'}^{t} \vartheta_u(\tau) \frac{\partial z_u(\tau)}{\partial W_{[u,v]}} \\ &= \sum_{\tau=t'}^{t} \vartheta_u(\tau) X_{[u,v]}(\tau). \end{aligned}$$

There is definitely more on the backpropagation through time and it is described in more detail in the following works [16, 23]. We just wanted to derive the weight update formula which will be useful when talking about the vanishing/exploding gradient problem later in this chapter.

1.2.2 Vanishing Error Problem

Standard RNN memory is not able to reasonably contain more than 5–10 time steps [19]. This is because back-propagated error signals will grow or shrink with every time step. Over many time steps, the error typically explodes towards large values or it vanishes, which means that from the computational perspective it is too close to zero [24, 25]. Error explosions lead to oscillating weights, and with a vanishing error the learning takes an enormous amount of time, or the model does not learn at all. The explanation of how the gradients are computed by the standard backpropagation algorithm and the basic vanishing error analysis is as follows: we update weights after the network has trained from time t' to time t using the formula

$$\Delta W_{[u,v]} = -\eta \frac{\partial E_{\text{total}}\ (t', t)}{\partial W_{[u,v]}},$$

with

$$\frac{\partial E_{\text{total}}\ (t', t)}{\partial W_{[u,v]}} = \sum_{\tau=t'}^{t} \vartheta_u(\tau) X_{[u,v]}(\tau),$$

where the backpropagated error signal at time τ (with $t' \leq \tau < t$) of the unit u is

$$\vartheta_u(\tau) = \mathbf{f}'_u\left(z_u(\tau)\right)\left(\sum_{v\in U} W_{vu}\vartheta_v(\tau+1)\right).$$

Given a fully recurrent neural network with a set of non-input units U, the error signal that occurs at any chosen output-layer neuron $o \in O$. at a time step, τ, is propagated back through time for $t-t'$ time-steps, with $t' < t$ to an arbitrary neuron v. This causes the error to be scaled by the following factor:

$$\frac{\partial\vartheta_v\left(t'\right)}{\partial\vartheta_o(t)} = \begin{cases} \mathrm{f}'_v\left(z_v\left(t'\right)\right)W_{[o,v]} & \text{if } t-t'=1 \\ \mathrm{f}'_v\left(z_v\left(t'\right)\right) & \left(\sum_{u\in U}\frac{\partial\vartheta_\alpha(t'+1)}{\partial\vartheta_d(t)}W_{[u,v]}\right) \\ \text{if } t-t'>1 & \end{cases}.$$

To solve the above equation, it is necessary to unroll it over time. For $t' \leq \tau \leq t$, let u_τ be a non-input-layer neuron in one of the replicas in the unrolled network at time τ. Now, by setting $u_t = v$ and $u_{t'} = o$, the following equation can be obtained

$$\frac{\partial\vartheta_v\left(t'\right)}{\partial\vartheta_o(t)} = \sum_{u_t,\in U}\cdots\sum_{u_{t-1}\in U}\left(\prod_{\tau=t'+1}^{t}\mathbf{f}'_{u_\tau}\left(z_{u_\tau}\left(t-\tau+t'\right)\right)W_{[u_\tau,u_{\tau-1}]}\right).$$

Observing the previous equation , it follows that if

$$|\,\mathbf{f}'_{u_\tau}\left(z_{u_\tau}\left(t-\tau+t'\right)\right)W_{\left[u_\tau,u_{\tau-1}\right]}\Big] > 1,$$

for all τ, then the product will grow exponentially, causing the error to explode. Conflicting error signals arriving at neuron v can lead to oscillating weights and unstable learning. If now

$$\left|\mathbf{f}'_{u_\tau}\left(z_{u_\tau}\left(t-\tau+t'\right)\right)W_{\left[u_\tau,u_{\tau-1}\right]}\right| < 1,$$

for all τ, then the product decreases exponentially, causing the error to vanish, preventing the network from effective learning within an acceptable time. Finally, the equation

$$\sum_{o\in O}\frac{\partial\vartheta_v\left(t'\right)}{\partial\vartheta_o(t)},$$

shows that if the local error vanishes, then the global error also vanishes [26].

1.3 LSTM

One way to deal with the vanishing error problem is a gradient-based method called long short-term memory (LSTM) [1, 25, 19, 21]. LSTM is able to learn how to bridge minimal time lags of more than 1,000 discrete time steps. It is

able to do that by using constant error carousels, which make sure that there is a constant error flow within special cells. Granting access to the cells is done by multiplicative gate units, which learn when to grant access. Picture is worth a thousand words in this case see figure 1.3 For any additional details see the original publication [1].

1.4 GRU

Gated Recurrent Unit (GRU) architecture for RNN as an alternative to LSTM proposed in [27]. GRU has empirically been found to outperform LSTM on nearly all tasks [28]. GRU units, unlike LSTM memory blocks, do not have a memory cell; although they do have gating units: a reset gate and an update gate. GRU reset and input gates behave like normal units in a recurrent network. The main characteristic of GRU is the way the activation of the GRU units is defined [1] . See the original publication [27] for details.

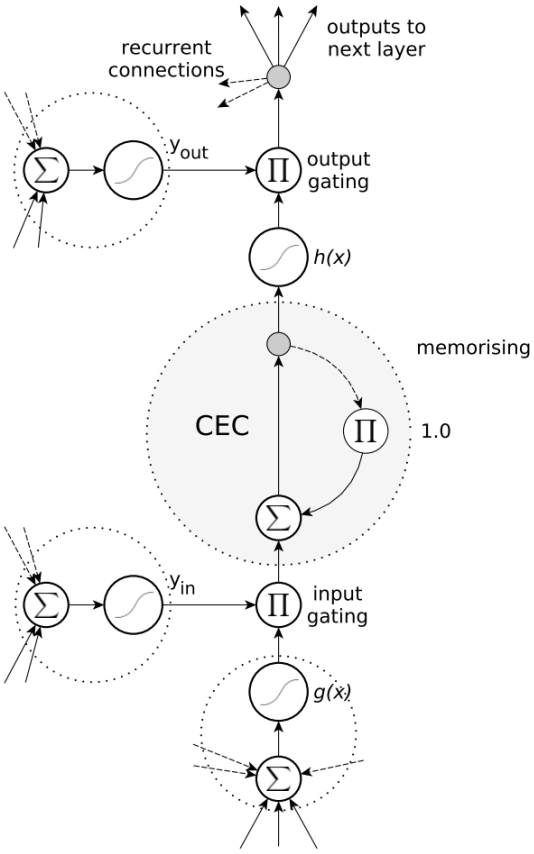


Figure 1.2: A standard LSTM memory block. The block contains (at least) one cell with a recurrent self-connection (Constant Error Carousel) and weight of '1'. The state of the cell is denoted as s_c. Read and write access is regulated by the input gate, y_{in} , and the output gate, y_{out} . The internal cell state is calculated by multiplying the result of the squashed input, g, by the result of the input gate, y_{in}, and then adding the state of the last time step, $s_c(t-1)$. Finally, the cell output is calculated by multiplying the cell state, s_c, by the activation of the output gate, y_{out} [2].

Chapter 2

Novel approaches

Many areas of machine learning require processing long sequential data. For example, a time series may be observed in real time where the future needs to be continuously predicted, or an agent in a partially observed environment must learn how to encode its cumulative experience into a state to navigate and make decisions. The fundamental problem in modeling long-term and complex temporal dependencies is memory, that means storing and incorporating information from previous time steps, which, as shown in the previous chapters, gets increasingly difficult as the length of the processed sequence increases not only because of the vanishing gradient problem. In this chapter, we will try to explain two novel approaches, LMU and HiPPO-LegS.

2.1 Mathematical prerequisites

In the following section, we will introduce some advanced concepts from mathematics in a gentle way. For example, the concept of measure is needed to further explore and generalize the concepts in [4] using the idea of function approximation with respect to a measure introduced in [5]. It is strongly recommended to consult [29, 30, 31, 32, 33] for more correct interpretation of the topics in this section. Properly introducing all the concepts used in both [4, 5] would be very technical and definitely out of the scope of this thesis. An attempt will be made to introduce the concepts in a somewhat informal way, even though for the sake of preservation of our own sanity we will still use a formal notation where it is suitable. Basic knowledge of linear algebra and analysis is assumed.

2.1.1 Measurable space

We can think about a measurable space as if it was a collection of events $\mathcal{M}$, and the set of all outcomes X, which is sometimes called the sample space. Given a collection of possible events $\mathcal{M}$ why do we need to state X? Having

it makes it possible to define complements of sets. If the event $F \in \mathcal{M}$, then the event F^C is the set of outcomes in X that are disjoint from F [29, 30, 31].

Definition 1 *By a measurable space we mean a couple $(X, \mathcal{M})$ consisting of a set X and a σ -algebra $\mathcal{M}$ of subsets of X. A subset E of X is called measurable (or measurable with respect to $\mathcal{M}$) provided E belongs to $\mathcal{M}$*

2.1.2 Algebras and Fields

Often, we will see that the collection of events $\mathcal{M}$ in a measurable space is a σ -algebra. A σ -algebra is a special kind of collection of subsets of the sample space X : a σ -algebra is complete in that if some set A is in our σ -algebra, then we have to have A^C (the complement of A) in our set too. In addition, it must be that if we have two sets A and B in our collection of sets, then the union $A \cup B$ must also be in our collection of sets (in fact, σ -algebras are closed under countable unions, not just finite unions) [31].

Definition 2 *A collection $\mathcal{M}$ of subsets of a set X is said to be a σ -algebra in X if $\mathcal{M}$ has the following properties:*

- $X \in \mathcal{M}$.
- *If $A \in \mathcal{M}$, then $A^c \in \mathcal{M}$, where A^c is the complement of A relative to X.*
- *If $A = \bigcup_{n=1}^{\infty} A_n$ and if $A_n \in \mathcal{M}$ for $n = 1, 2, 3, \ldots$, then $A \in \mathcal{M}$.*

- *If $\mathcal{M}$ is a σ -algebra in X, then X is called a measurable space, and the members of $\mathcal{M}$ are called the measurable sets in X.*
- *If X is a measurable space, Y is a topological space, and f is a mapping of X into Y, then f is said to be measurable provided that $f^{-1}(V)$ is a measurable set in X for every open set V in Y.*

Another term sometimes used to mean the same thing as σ -algebra is σ -field. The smallest possible σ -field is a collection of just two sets, $\{X, \emptyset\}$. The largest possible σ -field is the collection of all the possible subsets of Ω, this is called the powerset [29, 30, 31].

2.1.3 Measure

Definition 3 *By a measure μ on a measurable space $(X, \mathcal{M})$ we mean an extended real-valued nonnegative set function $\mu : \mathcal{M} \to [0, \infty]$ for which $\mu(\emptyset) = 0$ and which is countably additive in the sense that for any countable disjoint collection $\{E_k\}_{k=1}^{\infty}$ of measurable sets,*

$$\mu\left(\bigcup_{k=1}^{\infty} E_k\right) = \sum_{k=1}^{\infty} \mu\left(E_k\right).$$

By a measure space $(X, \mathcal{M}, \mu)$ we mean a measurable space $(X, \mathcal{M})$ together with a measure μ defined on $\mathcal{M}$

A measure μ takes a set A (from a measurable collection of sets $\mathcal{M}$), and returns "the measure of A," which is some positive real number. So ones writes $\mu : \mathcal{M} \to [0, \infty)$. An example measure is volume, which goes by the name Lebesgue measure. In general, measures are generalized notions of volume. The triple $(X, \mathcal{M}, \mu)$ combines a measurable space and a measure, and thus the triple is called a measure space. A measure has these two properties:

1. Nonnegativity: $\mu(A) \geq 0$ for all $A \in \mathcal{M}$
2. Countable Additivity: If $A_i \in \mathcal{M}$ are disjoint sets for $i = 1, 2, \ldots$, then the measure of the union of the A_i is equal to the sum of the measures of the A_i.

We can see how our ordinary notion of volume satisfies these two properties. There are a couple variations on the term measure that we will run into. One is a signed measure, which can be negative. A special case of measure is the probability measure in the probability space. The probability measure P has the two above properties of a measure but it's also normalized, such that $P(X) = 1$.

A probability measure P over a discrete set of events is what we know as a probability mass function. For example, given a probability measure P and two sets $A, B \in \mathcal{M}$, it is possible to write this pretty well-known formula for conditional probability [29, 30, 31].

$$P(B \mid A) = \frac{P(A \cap B)}{P(A)}.$$

Having a measure allows the definition of an inner product of two functions, which we will later use to define the orthogonality of polynomials.

2.1.4 Function spaces

Function space is a class X of functions (with fixed domain and range). Simply put, a function space is a space made of functions. Each function in the space can be thought of as a point. An example of that would be $C[a, b]$, the set of all real-valued continuous functions in the interval $[a, b]$ [34].

There are many ways in which knowledge of the structure of function spaces can assist in the study of functions. For instance, if one has a good basis for the function space, so that every function in the space is a (possibly infinite) linear combination of basis elements, and one has some quantitative estimates on how this linear combination converges to the original function, then this allows one to represent that function efficiently in terms of a number of co-efficients, and also allows one to approximate that function by smoother functions [34].

2.1.5 Metric Spaces

Let X be a nonempty set. Function $d : X \times X \to [0, \infty)$ is called a metric on X if it satisfies the following conditions:

1. $d(x, y) = 0 \iff x = y$.
2. $d(x, y) = d(y, x)$.
3. $d(x, z) \leq d(x, y) + d(y, z)$ for all $x, y, z \in X$.

A set X equipped with a metric d is called a metric space and is denoted by (X, d). One of the examples of metric spaces is well-known distance on the real line. More formally, the set of all real numbers equipped with distance

$$d(x, y) = |x - y|,$$

is the metric space $\mathbb{R}^1$.

2.1.6 Inner Product Space

A complex vector space H is called an inner product space [30] (or unitary space) if to each ordered pair of vectors x and $y \in H$ there is associated a complex number (x, y), the so-called "inner product" (or "scalar product") of x and y, such that the following rules hold:

- $(y, x) = \overline{(x, y)}$. (The bar denotes complex conjugation.)
- $(x + y, z) = (x, z) + (y, z)$ if x, y, and $z \in H$.
- $(\alpha x, y) = \alpha(x, y)$ if x and $y \in H$ and α is a scalar.
- $(x, x) \geq 0$ for all $x \in H$.
- $(x, x) = 0$ only if $x = 0$.

2.1.7 Cauchy sequence

Cauchy sequence is a sequence whose elements become arbitrarily close to each other as the sequence progresses. More precisely, given any small positive distance, all but a finite number of elements of the sequence are less than that given distance from each other. We say that a sequence of real numbers $\{a_n\}$ is a Cauchy sequence provided that for every $\epsilon > 0$, there is a natural number N so that when $n, m \geq N$, we have that $|a_n - a_m| \leq \epsilon$ [32].

2.1.8 Completeness of metric space

In the context of a metric space $[M = (X, d)]$, Cauchy sequence is defined as the sequence whose distance becomes smaller as the series proceeds. Such that, given $m, n > N$ (a positive integer), the following holds true

$$d(x_m, x_n) < \epsilon.$$

A sequence of functions f_n is fundamental if $\forall \epsilon > 0 \quad \exists N_\epsilon$ such that

$$\forall n \text{ and } m > N_\epsilon, \quad d(f_n, f_m) < \epsilon.$$

A metric space is complete if all fundamental sequences converge to a point in the space. A sequence of functions f_n is fundamental if $\forall \epsilon > 0 \exists N_\epsilon$ such that

$$\forall n \text{ and } m > N_\epsilon, \quad d(f_n, f_m) < \epsilon.$$

A metric space is complete if all fundamental sequences converge to a point in the space [30, 32]. In other words completeness grants us that there are not any gaps.

2.1.9 Hilbert space

Having a metric space H. If this metric space is complete, that means if every cauchy sequence converges in H, then H is called a Hilbert space. [30]

2.1.10 L^2 space and $L^2(\mu)$ norm of a function

If μ is any positive measure, $L^2(\mu)$ is a Hilbert space, with inner product

$$(f, g) = \int_X f \bar{g} d\mu.$$

The L^2 space is a special case of an L^p space, which is also known as the Lebesgue space. Let X be a measure space. Given a complex function f, we say $f \in L^2$ on X if f is (Lebesgue) measurable and if

$$\int_X |f|^2 \, \mathrm{d}\mu < +\infty.$$

Then the function f is also said to be square-integrable. In other words, L^2 is the set of square-integrable functions. For $f \in L^2(\mu)$ define

$$\|f\| = \left(\int_X |f|^2 \, \mathrm{d}\mu \right)^{1/2}.$$

We call $\|f\|$ the $L^2(\mu)$ norm of f To give a notion of distance in $L^2(\mu)$, we define the distance between between two functions f and g in $L^2(\mu)$ as

$$d(f, g) = \|f - g\|.$$

The space L^2 is unique among L^p spaces as a Hilbert space. Hilbert spaces have many useful properties. In particular, its similarity to Euclidean space enables the use of geometric notions such as distance and orthogonality. The Pythagorean identity also holds true in $\mathcal{L}^2$. In addition, Hilbert spaces, and in particular the $\mathcal{L}^2$ Hilbert space, are very important in many different parts of physics and mathematics [35, 32, 30].

2.1.11 Lipchitz functions

Lipchitz functions are a class of functions which appear not only in many branches of mathematics including computer science. Lipschitz function $f : \Omega \to \mathbb{R}^m$, where Ω is an open subset of $\mathbb{R}^n$, is differentiable outside of a Lebesgue null subset of Ω, the condition of being Lipschitz could be viewed as a weakened version of differentiability, and therefore, these functions are a good substitute for smooth functions in the framework of metric spaces. Lipschitz functions are smooth functions of metric spaces. A real-valued function f on a metric space X is said to be L -Lipschitz if there is a constant $L \geq 1$ such that

$$|f(x) - f(y)| \leq L|x - y|,$$

for all x and y in X [36, 37].

2.1.12 Padé Approximants

If we define f as a power series.

$$f = \sum_{n=0}^{\infty} a_n (x - c)^n = a_0 + a_1 (x - c)^1 + a_2 (x - c)^2 + \cdots$$

A Padé approximant of f is the "best" approximation of f by a rational function of a given order. It is a rational function whose numerator and denominator are chosen so that its power series expansion (obtained by dividing the numerator by the denominator in ascending powers of the variable) agrees with f as far as possible that is, at least, up to the term whose degree equals the sum of the degrees of the numerator and the denominator of the rational function. Such approximants have a long history and they play an important role in the solution of many problems such as the transcendence of the numbers e and π. Sixty years ago, Padé approximants proved to be very efficient not only to improve existing methods but also for extracting important information from power series [38].

2.1.13 Orthogonal Polynomials

Orthogonal polynomials are a standard tool for working with function spaces [5]. Every measure μ induces a unique (up to a scalar) sequence of orthogonal

polynomials (OPs) $P_0(x), P_1(x), \ldots$ satisfying

$$\deg(P_i) = i,$$

and

$$\langle P_i, P_j \rangle_\mu := \int P_i(x) P_j(x) \mathrm{d}\mu(x) = 0,$$

for all $i \neq j$. This is the sequence found by orthogonalizing the monomial basis $\{x^i\}$ with Gram-Schmidt with respect to $\langle \cdot, - \rangle_\mu$. The fact that OPs form an orthogonal basis is useful because the optimal polynomial g of degree $\deg(g) < N$ that approximates a function f is then given by

$$\sum_{i=0}^{N-1} c_i P_i(x) / \|P_i\|_\mu^2 \quad \text{where } c_i = \langle f, P_i \rangle_\mu = \int f(x) P_i(x) \mathrm{d}\mu(x),$$

It may not look like that on a first sight, but computing the approximation of f like that is actually an easy job, at least for a computer. The computation is easy because when using an orthogonal basis, many of the computations that would be necessary if we chose a nonorthogonal basis can be omitted. This observation also kind of explains why the mononomial basis $\{1, x, x^2, ...\}$ is not a good basis because the polynomials are not orthogonal under any measure. Even though it may look like a good basis, it is way less computationally efficient to compute the same approximation of f using a nonorthogonal basis. Simply put, it is because we cannot take advantage of the simple way to find the representation of f. It stems from a concept introduced in the very first linear algebra courses. Basically if $v_1, \cdots, v_k$ is an orthogonal basis for a subspace V of $\mathbb{R}^n$, and v is a vector in V then:

$$v = \left(\frac{v \cdot v_1}{|v_1|^2} \right) v_1 + \cdots + \left(\frac{v \cdot v_k}{|v_k|^2} \right) v_k,$$

and $v_i \cdot v_j = 0$ if $i \neq j$ and $v_i \cdot v_i = \|v_i\|^2$. Which is exactly what we see in the formula for the approximation.

There are different families of orthogonal polynomials. The family of Jacobi polynomials includes Legendre polynomials which also includes Chebyshev polynomials. Other polynomials used in the work [5] include Laguerre, and Hermite polynomials. They also show that the Fourier basis can be interpreted as orthogonal polynomials on the unit circle in the complex plane [29, 30, 31].

2.1.14 Legendre Polynomials properties

Several properties of Legendre polynomials are used in both [4, 5].

2.1.14.1 Legendre polynomials

Under the usual definition [4, 5] of the canonical Legendre polynomial P_n, they are orthogonal with respect to the measure $\omega^{\text{leg}} = 1_{[-1,1]}$:

$$\frac{2n+1}{2}\int_{-1}^{1} P_n(x)P_m(x)\mathrm{d}x = \delta_{nm}. \tag{2.1}$$

Additionally, they satisfy

$$P_n(1) = 1$$
$$P_n(-1) = (-1)^n.$$

2.1.14.2 Shifted and Scaled Legendre polynomials

It is also possible to consider the scaling Legendre polynomials [4, 5] to be orthogonal on the interval $[0, t]$. A change of variables on the 2.1 yields

$$\begin{aligned}
&(2n+1)\int_0^t P_n\left(\frac{2x}{t}-1\right)P_m\left(\frac{2x}{t}-1\right)\frac{1}{t}\,\mathrm{d}x \\
&= (2n+1)\int P_n\left(\frac{2x}{t}-1\right)P_m\left(\frac{2x}{t}-1\right)\omega^{\text{leg}}\left(\frac{2x}{t}-1\right)\frac{1}{t}\,\mathrm{d}x \\
&= \frac{2n+1}{2}\int P_n(x)P_m(x)\omega^{\text{leg}}(x)\mathrm{d}x \\
&= \delta_{nm}.
\end{aligned}$$

Therefore, with respect to the measure $\omega_t = 1_{[0,t]}/t$ (which is a probability measure for all t), the normalized orthogonal polynomials are

$$(2n+1)^{1/2}P_n\left(\frac{2x}{t}-1\right).$$

Similarly, the basis

$$(2n+1)^{1/2}P_n\left(2\frac{x-t}{\theta}+1\right),$$

is orthonormal for the uniform measure $\frac{1}{\theta}\mathbb{I}_{[t-\theta,t]}$. In general, the orthonormal basis for any uniform measure consists of $(2n+1)^{\frac{1}{2}}$ times the corresponding linearly shifted version of P_n.

2.1.14.3 Derivatives of Legendre polynomials

Legendre polynomials can be described using recurrence relations on Legendre polynomials [5], this is useful as it can be used to for an elegant definition of their derivative:

$$(2n+1)P_n = P'_{n+1} - P'_{n-1}$$
$$P'_{n+1} = (n+1)P_n + xP'_n.$$

The first equation yields

$$P'_{n+1} = (2n+1)P_n + (2n-3)P_{n-2} + \dots$$

where the sum stops at P_0 or P_1. These equations directly imply

$$P'_n = (2n-1)P_{n-1} + (2n-5)P_{n-3} + \dots$$

and

$$\begin{aligned}(x+1)P'_n(x) &= P'_{n+1} + P'_n - (n+1)P_n \\ &= nP_n + (2n-1)P_{n-1} + (2n-3)P_{n-2} + \dots\end{aligned}$$

2.1.15 Control theory

Control theory is often regarded as a branch of the general and more abstract subject of systems theory. Control theory can be approached from a number of directions. The first systematic method of dealing with what is now called control theory began to emerge in the 1930 s. Transfer function and frequency domain techniques were predominant in these "classical" approaches to control theory. Starting in the late 1950s and early 1960s, a time-domain approach using state variable descriptions came into prominence. For a number of years, the state variable approach was synonymous with "modern control theory." Nowadays, the state variable approach and various transfer function-based methods are considered on an equal level and nicely complement each other. Distinctions exist, and the major one appears to be in the kinds of mathematical tools used. The state variable approach uses linear algebra based on the real or complex number field. The newer approach involves multivariable transfer functions and the algebra of polynomial matrices and related concepts. By defining the number field properly, the major mathematical tool is once again linear algebra, but on a level most of the mortals are less familiar with [33].

Many control problems are solved in literature in the following way using the analytical approach and state variable modelling. Performing the analysis and control of dynamical systems consists of three major steps:

1. Developing an idealized mathematical representation of the real physical system
2. Applying mathematical analysis and design techniques to the model
3. Interpreting the mathematical results. If the resulting implications are not acceptable or do not seem to match reality or experimental observations start over again

In this case, the real physical problem is computing the continuous delay of the input signal, which was proposed as a way to represent memory since computing delay without memory is not possible [10].

2.1.16 Dynamic systems and the concept of state

The concept of state is essential in modern control theory. However, the concept of state appears in many other technical and nontechnical contexts as well. In everyday life, monthly financial statements are commonplace. The annual message delivered by the president near the beginning of each calendar year on the current condition of the nation is another familiar example. In all these examples, the concept of state is essentially the same. It is a complete summary of the status of the system at a particular point in time. Knowledge of the state at some initial time t_0, plus knowledge of the system inputs after t_0, allows the determination of the state at a later time t_1. As far as the state at t_1 is concerned, it makes no difference how the initial state was computed. The state at t_0 represents a complete history of the system behavior prior to t_0, so that the history affects future behavior[33].

One of the main challenges in computational neuroscience is understanding how dynamic stimuli can be processed by neural mechanisms to drive behavior. Recurrent connections, cellular responses, and synaptic responses are sources of dynamics throughout the mammalian brain that must cooperate to support dynamic information processing. How these low-level mechanisms interact to encode information about the history of a stimulus across time is a subject of many works in the field of neuroscience. The neural engineering framework proposes a method to model such dynamical systems in networks of spiking neurons and will be discussed in greater detail in the following chapters [39].

2.1.17 State-space model

Even though it is not the main focus of this chapter, it is important to note that the Principle 3 of the neural engineering framework [39] allows the neural implementation (using spiking neurons) of continuous linear time-invariant (LTI) systems which are defined as:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= A\mathbf{x}(t) + B\mathbf{u}(t)\\ \mathbf{y}(t) &= C\mathbf{x}(t) + D\mathbf{u}(t),\end{aligned}$$

where the time-varying signal $\mathbf{x}(t)$ represents the system state, $\dot{\mathbf{x}}(t)$ its time derivative, $\mathbf{y}(t)$ the output, $\mathbf{u}(t)$ the input, and the time-invariant matrices (A, B, C, D) fully describe the system [33]. This form of an LTI system is commonly referred to as the state-space model. By dynamical primitive one means the source of the dynamics for the system. For LTI systems, the dynamical primitive is the integrator, see figure 2.1.17. [3]

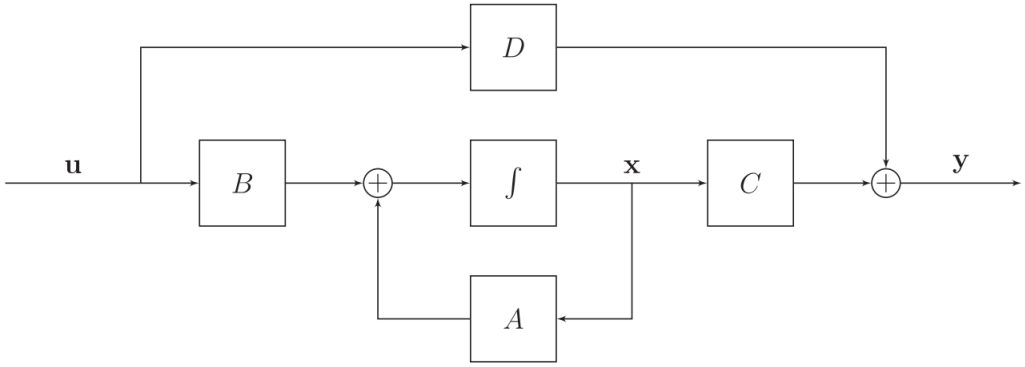


Figure 2.1: Block diagram for and LTI system. The integrator is driven by the signal $\dot{\mathbf{x}}(t)$[3]

2.1.18 Laplace transform

By the Laplace transform of function $f : [0, \infty) \to \mathbb{C}$ we mean $L[f] = F$ defined by

$$F(s) = \int_0^\infty f(t)e^{-ts}\,\mathrm{d}t,$$

if for at least one s the integral converges. For example, if $f = 1$ the laplace transform L of the function f is the following.

$$L[f = 1](s) = \frac{1}{s} \text{ for each } s \in \mathbb{C} \text{ which has its real part } Re(s) > 0.$$

2.1.19 Transfer function of a dynamic system

The transfer function of a dynamic system is defined as the ratio of the Laplace transform of the output variable to Laplace transform of the input variable assuming all initial conditions to be zero. If the input is represented by $y(s)$ and output by $u(s)$

$$F(s) = \frac{y(s)}{u(s)}.$$

Transfer function represents the relationship between the output signal of a control system and the input signal, for all possible input values [33].

The transfer function is related to the LTI system defined in the previous section is given by the following:

$$F(s) = \frac{y(s)}{u(s)} = C(sI - A)^{-1}B + D.$$

We can see that the time invariant matrices (A, B, C, D) are present in the equation. The transfer function $F(s)$ can be converted into the state-space model (A, B, C, D) if and only if it can be written as a proper ratio of finite polynomials in s. Ratio is proper when the degree of the numerator does not exceed that of the denominator. In this case, we can notice that the output does not depend on the future input. The order of the denominator corresponds to the dimensionality of $\mathbf{x}$, and therefore must be finite. Both of these conditions can be interpreted as physically realistic constraints where time may only progress forward, and neural resources are finite [40].

2.2 LMU

Legendre Memory Unit (LMU) is a new recurrent architecture and method of weight initialization that provides interesting theoretical guarantees for learning long-range dependencies, even as the discrete time-step, Δt, approaches zero. This enables the gradient to flow across the continuous history of internal feature representations [4].

2.2.1 Delay network and Ideal Delay

Delay network, which was first described in [10] is an imporant dynamical system that is useful for improving the memory of recurrent networks, it is a system realizing a delay. It is necessary to define the delay problem and show how a delay is optimally realized by the DN, which is in a LTI system. LMU employs a single-input DN coupled to a nonlinear dynamical system to process sequential data. A system is said to be an ideal delay system if it takes in an input, $u(t)$, and outputs a function, $y(t)$, which is the delayed version of the input. Mathematically, this can be described in the following manner:

$$y(t) = \mathcal{D}[u(t)] = \begin{cases} 0 & t < \theta \\ u(t-\theta) & t \geq \theta, \end{cases}$$

where D is the ideal delay operator and $\theta \in \mathbb{R}$ is the length of the delay. The ideal delay system is linear, which means that for any two functions, $f(t)$ and $g(t)$, and any $a, b \in \mathbb{R}$, it respects the following equation:

$$\mathcal{D}[af(t) + bg(t)] = a\mathcal{D}[f(t)] + b\mathcal{D}[g(t)],$$

it takes a system with infinite memory to take in and store a continuous input for θ seconds and then reproduce the entire input without error.

The optimal system that implements delay must be linear and even the most optimal physical implementation can be approximate as the resources will always be finite. Given the fact that the delay can be at best approximated, we can move on to search for the best approximation [41, 4, 10].

2.2.2 Approximating Delay

When constructing a dynamical system that implements delay, it is possible to narrow the search space from the general system of ordinary differential equations to the following form thanks to the linearity constraint:

$$\begin{aligned} \dot{m} &= \mathbf{f}(\boldsymbol{m}, u) \\ y &= \mathbf{g}(\boldsymbol{m}, u), \end{aligned}$$

to just finding the four matrices $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$ that define an LTI system:

$$\begin{aligned} \dot{m} &= Am + Bu \\ y &= Cm + Du. \end{aligned}$$

Considering the transfer function of the delay system, which for a single input single output system is defined as:

$$G(s) = \frac{y(s)}{u(s)} = e^{-\theta s},$$

where $y(s)$ and $u(s)$ are found by taking the Laplace transform of the input and output functions in time. This defines an infinite dimensional transfer function, capturing the intuitive difficulty of constructing a continuous delay [4, 41, 10].

The transfer function can be converted to a finite state space realization if and only if it can be written as a proper ratio of finite dimensional polynomials in s. A ratio $\frac{a(s)}{b(s)}$ is said to be proper if the order of the numerator does not exceed the order of the denominator. $G(s)$, however, is irrational, that means that it cannot be written as a proper finite dimensional ratio. This observation suggests that making an approximation is necessary [33].

To achieve an optimal convergence rate in the least square error point of view, Padé approximants are used [42]. Choosing the order of the numerator to be one less than the order of the denominator and accounting for numerical issues in the state-space realization [10, 4], gives the following realization:

$$\begin{aligned} A_{i,j} &= \frac{(2i+1)}{\theta} \begin{cases} -1 & i < j \\ (-1)^{i-j+1} & i \geq j \end{cases} \\ B_i &= \frac{(2i+1)(-1)^i}{\theta} \\ C_i &= (-1)^i \sum_{l=0}^{i} \binom{i}{l} \binom{i+l}{j} (-1)^l \\ D &= 0, \quad i, j \in [0, d-1], \end{aligned}$$

the variable d is the order of the system. The LTI system $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$ is a Delay Network (DN). The order of the system, d, and the delay length, θ are the main hyperparameters to choose when using a DN. Higher order

systems require more resources, but provide a more accurate emulation of the ideal delay. Because of the usage of the Padé approximants, each order is optimal for that dimension of the state vector m [41, 10]. The realization of the matrices defining the LTI system corresponds to the following `numpy` code.

```
import numpy as np
# 'order' is the number of
# Legendre polynomials used to
# orthogonally represent the sliding window.
# 'theta' is the length of the in
Q=np.arange(order,dtype=np.float64)
R=(2*Q+1)[:,None]/theta
j,i=np.meshgrid(Q,Q)
A=np.where(i<j,-1,(-1.0)**(i-j + 1))*R
B=(-1.0)**Q[:,None]*R
C=np.ones((1,order))
D=np.zeros((1,))
```

2.2.3 Legendre polynomials

Legendre Polynomials are a special class of polynomials with interesting properties as discussed earlier, they are one of the standard tools for working with function spaces. Constructing a system defined in the previous section is done by using the $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$ matrices defined above, and providing it with an input signal, $u(t)$. When given the state m_t, it is possible to use $\boldsymbol{C}$ to decode $u(t-\theta)$ to a degree of accuracy determined by the order of the system.

$$u(t-\theta) \approx \boldsymbol{C}^T \boldsymbol{m}_t. \tag{2.2}$$

Intuitively, given m_t, it seems possible to decode not only $u(t-\theta)$ but also $u\left(t-\theta'\right) \forall 0 \leq \theta' \leq \theta$. This can be done using a slightly modified $\boldsymbol{C}$ for a given θ' :

$$u\left(t-\theta'\right) \approx \boldsymbol{C}\left(\theta'\right)^T \boldsymbol{m}_t,$$

where

$$C_i\left(\theta'\right) = (-1)^i \sum_{l=0}^{i} \binom{i}{l} \binom{i+l}{j} \left(-\frac{\theta'}{\theta}\right)^l, 0 \leq \theta' \leq \theta, \tag{2.3}$$

and $C\left(\theta'=\theta\right)$ corresponds to the C defined in the equation

$$C_i = (-1)^i \sum_{l=0}^{i} \binom{i}{l} \binom{i+l}{j} (-1)^l.$$

The functions in 2.3 turn out to be the shifted Legendre polynomials.

2.2.4 Memory Cell Dynamics

In this section, we will try to shed some light on the Memory Cell Dynamics used to realize the storage and retrieval of the cell state which can be perceived as memory. The main component of the Legendre Memory Unit (LMU), which is a memory cell that orthogonalizes the continuous-time history of its input signal, $u(t) \in \mathbb{R}$, across a sliding window of length $\theta \in \mathbb{R}_{>0}$. The cell is derived from the linear transfer function for a continuous-time delay, $F(s) = e^{-\theta s}$, which is best-approximated by d coupled ordinary differential equations (ODEs):

$$\theta \dot{\mathbf{m}}(t) = \mathbf{A}\mathbf{m}(t) + \mathbf{B}u(t), \tag{2.4}$$

where $\mathbf{m}(t) \in \mathbb{R}^d$ is a state-vector with d dimensions. The ideal state-space matrices, $(\mathbf{A}, \mathbf{B})$, are derived through the use of Padé [43] approximants [10]:

$$\begin{aligned} \mathbf{A} &= [a]_{ij} \in \mathbb{R}^{d\times d}, \quad a_{ij} = (2i+1)\begin{cases} -1 & i < j \\ (-1)^{i-j+1} & i \geq j \end{cases} \\ \mathbf{B} &= [b]_i \in \mathbb{R}^{d\times 1}, \quad b_i = (2i+1)(-1)^i, \quad i,j \in [0, d-1]. \end{aligned} \tag{2.5}$$

The key property of this dynamical system is that m represents sliding windows of u via the Legendre [44] polynomials up to degree $d-1$:

$$\begin{aligned} u\left(t-\theta'\right) &\approx \sum_{i=0}^{d-1} \mathcal{P}_i\left(\frac{\theta'}{\theta}\right) m_i(t), \quad 0 \leq \theta' \leq \theta, \\ \mathcal{P}_i(r) &= (-1)^i \sum_{j=0}^{i} \binom{i}{j} \binom{i+j}{j} (-r)^j, \end{aligned} \tag{2.6}$$

where $\mathcal{P}_i(r)$ is the i^{th} shifted Legendre polynomial [45]. This gives a unique and optimal decomposition. The functions of m correspond to computations across windows of length θ, projected onto d orthogonal basis functions [4].

2.2.5 Discretization

To actually perform any computation, we need to perform a discretization of the matrices describing the ideal state-space model. Discretization process in this case means mapping the equations onto the memory of a recurrent neural network, $\mathbf{m}_t \in \mathbb{R}^d$, given some input $u_t \in \mathbb{R}$, indexed at discrete moments in time, $t \in \mathbb{N}$:

$$\mathbf{m}_t = \overline{\mathbf{A}}\mathbf{m}_{t-1} + \overline{\mathbf{B}}u_t,$$

where $(\overline{\mathbf{A}}, \overline{\mathbf{B}})$ are the discretized matrices provided by the ODE solver for some time-step Δt relative to the window length θ. For Euler's method if Δt is sufficiently small:

$$\overline{\mathbf{A}} = (\Delta t/\theta)\mathbf{A} + \mathbf{I}, \quad \overline{\mathbf{B}} = (\Delta t/\theta)\mathbf{B}.$$

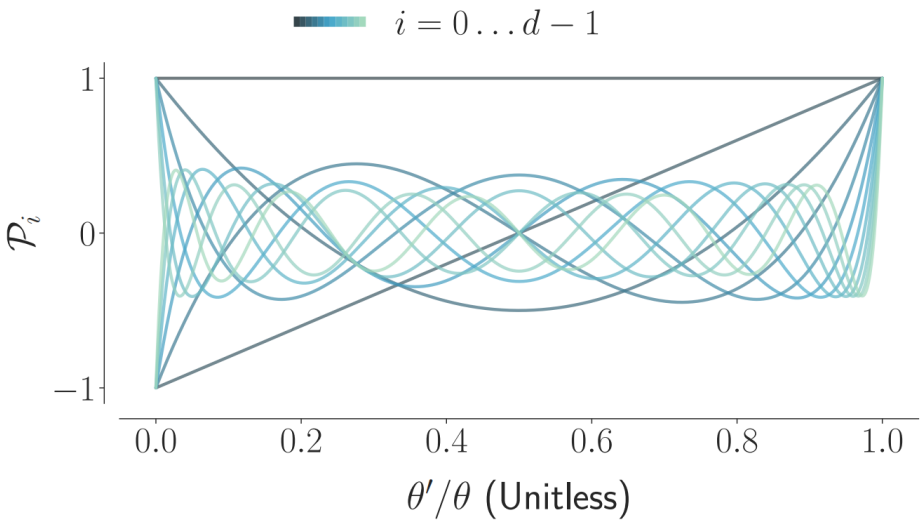


Figure 2.2: Shifted Legendre polynomials (d = 12). The memory of the LMU represents the entire sliding window of input history as a linear combination of these scale-invariant polynomials. Increasing the number of dimensions supports the storage of higher-frequency inputs relative to the time-scale [4].

Other discretization methods can be considered as well, for example zero-order hold (ZOH) for more information on the actual implementation of the discretization see documentation of `scipy.signal.cont2discrete` [46, 47].

2.2.5.1 Bilinear transform

The bilinear transform is a transformation from continuous-time systems to discrete-time systems. It uses the trapezoidal rule for numerical integration. It is used for simulation of dynamical systems, and even thought its not the best method available. It is fairly popular because of theoretically simple compared to other available options and it is easy to implement. [47]

2.2.6 Approximation Error

When $d = 1$, the memory is analogous to a single-unit LSTM without any gating mechanisms (a leaky integrator with time-constant θ). As d increases, so does its memory capacity relative to frequency content. In particular, the approximation error in equation 2.6 scales as $\mathcal{O}(\theta\omega/d)$, where ω is the frequency of the input u that is to be committed to memory [3].

2.2.7 Layer Design

The LMU takes an input vector, $\mathbf{x}_t$, and generates a hidden state, $\mathbf{h}_t \in \mathbb{R}^n$. Each layer maintains its own hidden state and memory vector. The state mutually interacts with the memory, $\mathbf{m}_t \in \mathbb{R}^d$, in order to compute nonlinear functions across time, while dynamically writing to memory. The state is a function of the input, previous state, and current memory:

$$\mathbf{h}_t = f\left(\mathbf{W}_\mathbf{x}\mathbf{x}_t + \mathbf{W}_\mathbf{h}\mathbf{h}_{t-1} + \mathbf{W}_\mathbf{m}\mathbf{m}_t\right),$$

where f is some chosen nonlinearity (e.g., tanh) and $\mathbf{W}_\mathrm{x}, \mathrm{W}_\mathrm{h}, \mathrm{W}_\mathrm{m}$ are learned kernels. Note that this decouples the size of the layer's hidden state (n) from the size of the layer's memory (d), and requires holding $n + d$ variables in memory between time steps. The input signal that writes to the memory via the equation $\mathbf{m}_t = \overline{\mathbf{A}}\mathbf{m}_{t-1} + \overline{\mathbf{B}}u_t$ is:

$$u_t = \mathbf{e}_\mathbf{x}^\top \mathbf{x}_t + \mathbf{e}_\mathbf{h}^\top \mathbf{h}_{t-1} + \mathbf{e}_\mathbf{m}^\top \mathbf{m}_{t-1},$$

where $\mathbf{e}_\mathbf{x}, \mathbf{e}_\mathbf{h}, \mathbf{e}_\mathbf{m}$ are learned encoding vectors. Intuitively, the kernels ($\mathbf{W}$) learn to compute nonlinear functions across the memory, while the encoders (e) learn to project the relevant information into the memory.

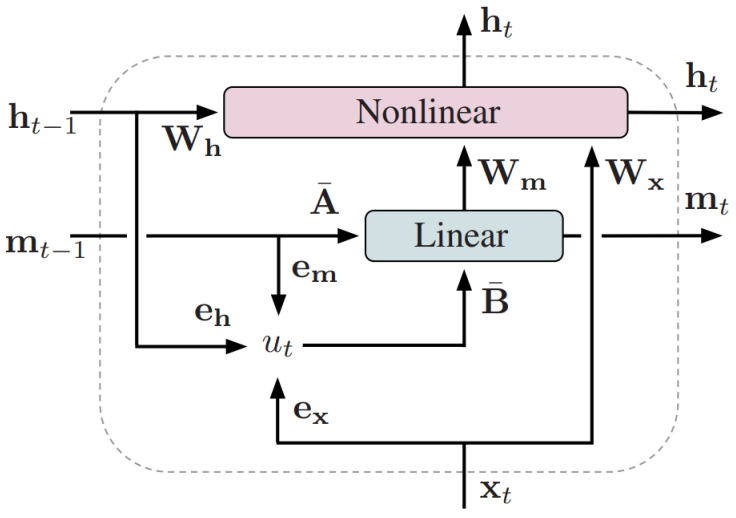


Figure 2.3: Time-unrolled LMU layer. An n dimensional state-vector ($\mathbf{h}_t$) is dynamically coupled with a d -dimensional memory vector ($\mathbf{m}_t$). The memory represents a sliding window of u_t, projected onto the first d Legendre polynomials [4].

2.3 HiPPO Framework

HiPPO framework was first introduced in [5] , it describes a method for addressing the fundamental problem of incrementally maintaining a memory representation of sequences. They carefully formulate this problem and analyze it. The main result is the derivation of a closed-form solution using the HiPPO framework for multiple memory mechanisms. The particular model HiPPO-LegS, which is computationally efficient, deals with vanishing gradients, and is the first known method be robust to timescaling.

They follow similar logic as in [4] but move from discrete-time to the continuous-time setting, which is often easier to analyze theoretically. They are trying to analyze the following problem: given a continuous function (in one dimension) $f(t)$, is it possible to maintain a fixed-size representation $c(t) \in \mathbb{R}^N$ at all times t such that $c(t)$ optimally captures the history of f from times 0 to t. Two things need to be specified in order to analyze such problem. What is the "optimal approximation" of the function's history and what basis is gonna be used for $c(t)$. Assuming a polynomial space we can think of the memory representation $c(t) \in \mathbb{R}^N$ as being the coefficient vector of the optimal polynomial approximation to the history of $f(t)$. That enables a thorough theoretical analysis, we will only discuss 4 basic propositions in this thesis regarding timescale invariance, inference speed, gradient norms and error bounds. Complete proofs of all propositions in this section are presented in the original paper as they are rather technical, we will only touch the way gradient norms are analysed and derivation of the two memory mechanisms used in the practical part of this thesis in this section [5].

2.3.1 Problem

As stated in the earlier sections of this work, given an input function $f(t) \in \mathbb{R}$ on $t \geq 0$, many problems require operating on the cumulative history $f_{\leq t} := f(x)|_{x \leq t}$ at every time $t \geq 0$, to understand the inputs seen so far and make future predictions. The space of functions is enormous, which means that the history cannot be perfectly memorized and must be compressed, the work [5] proposes the general approach of projecting it onto a subspace of bounded dimension. The goal is to maintain this compressed representation of the history. To further specify this problem, we need a way to quantify the approximation and a suitable subspace [5].

2.3.2 Function Approximation with respect to a measure.

Simply put, a measure induces a Hilbert space structure on the space of functions, so that there is a unique optimal approximation - the projection onto the desired subspace [6]. If we want to quantify the quality of an approximation, it requires defining a distance in the function space. Any probability

measure μ on $[0, \infty)$ equips the space of square integrable functions with inner product

$$\langle f, g \rangle_\mu = \int_0^\infty f(x)g(x)\mathrm{d}\mu(x), \tag{2.7}$$

inducing a Hilbert space structure (a vector space equipped with an inner product) $\mathcal{H}_\mu$ and corresponding norm

$$\|f\|_{L_2(\mu)} = \langle f, f \rangle_\mu^{1/2},$$

which may be percieved in a similar way we percieve norms in the basics of linear algebra even thought it works with Hilbert Spaces instead [5].

2.3.3 Polynomial Basis Expansion

Any N-dimensional subspace $\mathcal{G}$ of this function space is a suitable candidate for the approximation. The parameter N corresponds to the order of the approximation, or the size of the compression. The projected history can be represented by the N coefficients of its expansion in any basis of $\mathcal{G}$. The polynomials are used as a natural basis for simplicity in [5], so that $\mathcal{G}$ is the set of polynomials of degree less than N [5].

2.3.4 Online Approximation

When approximating $f_{\leq t}$ for every time t, it is useful to also let the measure vary through time. For every t, let $\mu^{(t)}$ be a measure supported on $(-\infty, t]$ (since $f_{\leq t}$ is only defined up to time t) [5]. In essence we are after some $g^{(t)} \in \mathcal{G}$ that minimizes

$$\left\| f_{\leq t} - g^{(t)} \right\|_{L_2\left(\mu^{(t)}\right)}.$$

Intuitively, the measure μ controls the importance of various parts of the input domain, and the basis defines the possible approximations. The challenge is how to solve the optimization problem in closed form given $\mu^{(t)}$, and how these coefficients can be maintained online as $t \to \infty$ [5, 4].

Definition 4 *Given a time-varying measure family $\mu^{(t)}$ supported on $(-\infty, t]$, an N -dimensional subspace $\mathcal{G}$ of polynomials, and a continuous function $f : \mathbb{R}_{\geq 0} \to \mathbb{R}$, HiPPO defines a projection operator proj_t and a coefficient extraction operator coef_t at every time t, with the following properties:*

1. *proj_t takes the function f restricted up to time t, $f_{\geq t} := f(x)|_{x<t}$, and maps it to a polynomial $g^{(t)} \in \mathcal{G}$, that minimizes the approximation error $\left\| f_{\leq t} - g^{(t)} \right\|_{L_2\left(\mu^{(t)}\right)}$*
2. *$\mathrm{coef}_t : \mathcal{G} \to \mathbb{R}^N$ maps the polymomial $g^{(t)}$ to the coefficients $c(t) \in \mathbb{R}^N$ of the basis of orthogonal polynomials defined with respect to the measure $\mu^{(t)}$.*

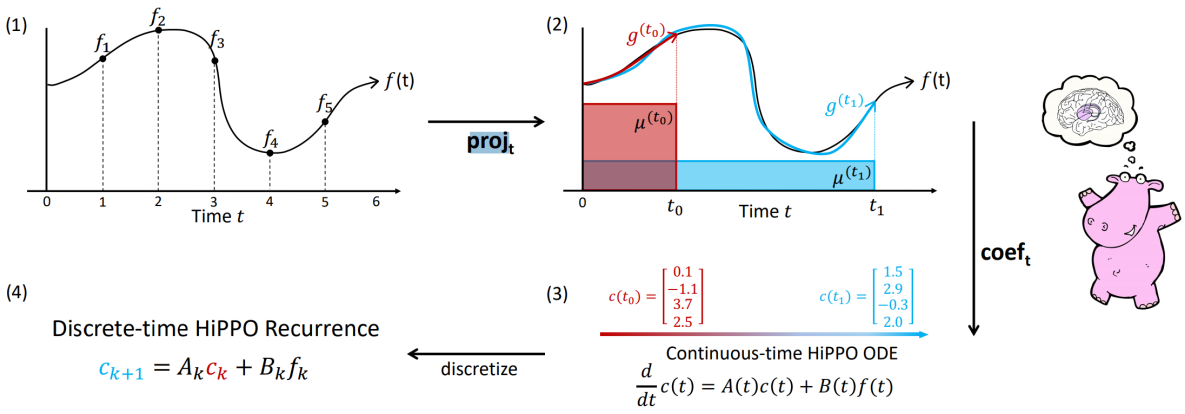


Figure 2.4: Illustration of the HiPPO framework. (1) For any function f, (2) at every time t there is an optimal projection $g^{(t)}$ of f onto the space of polynomials, with respect to a measure $\mu^{(t)}$ weighing the past. (3) For an appropriately chosen basis, the corresponding coefficients $c(t) \in \mathbb{R}^N$ representing a compression of the history of f satisfy linear dynamics. (4) Discretizing the dynamics yields an efficient closed-form recurrence for online compression of time series $(f_k)_{k\in\mathbb{N}}$. This illustrates the overall framework when we use uniform measures [5].

The composition coef $\circ$ proj is called hippo, which is an operator mapping a function $f : \mathbb{R}_{\geq 0} \to \mathbb{R}$ to the optimal projection coefficients $c : \mathbb{R}_{\geq 0} \to \mathbb{R}^N$, for example

$$(\text{hippo}(f))(t) = \text{coef}_t\left(\text{proj}_t(f)\right).$$

For each t, the problem of optimal projection $\text{proj}_t(f)$ is well-defined by the above inner products, but this is not computable using a naive approach. The derivation in the original paper [5] shows that the coefficient function

$$c(t) = \text{coef}_t\left(\text{proj}_t(f)\right),$$

has the form of an ODE satisfying

$$\frac{d}{dt}c(t) = A(t)c(t) + B(t)f(t) \text{ for some } A(t) \in \mathbb{R}^{N\times N} B(t) \in \mathbb{R}^{N\times 1}.$$

It is demonstrated how to obtain $c^{(t)}$ online by solving an ODE and running a discrete recurrence. When discretized, HiPPO takes in a sequence of real values and produces a sequence of N -dimensional vectors [5]. Next, several concrete applications of the framework will be shown.

2.3.5 High Order Projections

Given any input function $f(t)$, the desired coefficient vectors $c(t)$, which are the desired memory representation, are completely defined. However, we still need a way to calculate them. The HiPPO framework formalizes this problem and provides a closed form solution. The desired coefficients $c(t)$ are defined as the implicit solution to an approximation problem, there is a closed-form solution which is easy to compute. Leveraging concepts from approximation theory such as orthogonal polynomials [33, 5]. The solution takes on the form of a linear differential equation, which is called the HiPPO operator:

$$\dot{c}(t) = A(t)c(t) + B(t)f(t).$$

The HiPPO framework takes a family of measures, and gives an ODE with closed form transition matrices $A(t), B(t)$. These matrices depend on the measure, and following these dynamics finds the coefficients $c(t)$ that optimally approximate the history of $f(t)$ according to the measure [6].

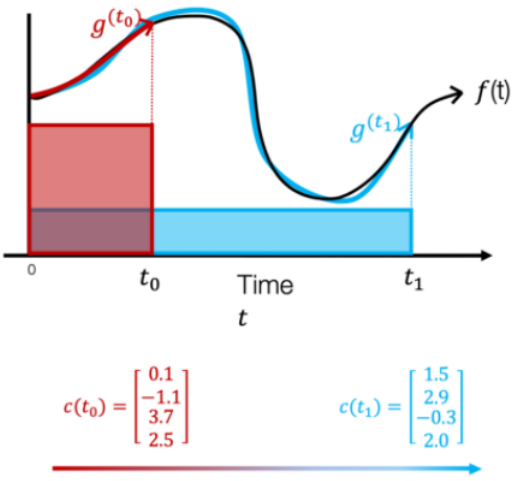


Figure 2.5: Tne input function $f(t)$ (black line) is continually approximated by storing the coefficients of its optimal polynomial projections (colored lines) according to specified measures (colored boxes). These coefficients evolve through time (red, blue) according to a linear dynamical system [5, 6].

Choosing a specific measure family $\mu^{(t)}$ results in a specific way past weighted in when projecting into a memory of a lower dimension.The unified perspective on memory mechanisms allows one to derive these closed-form solutions using the strategy from [5]. The first LegT explains the core Legendre Memory Unit (LMU) [4] update. In the figure 2.3.6 we can see the tradeoffs of these measures. The original paper derives additional HiPPO instantia-

tions using other bases such as Fourier which uncovers the previously used architecture called Fourier Recurrent Unit [48] and also Chebyshev base.

The translated Legendre (LegT) measures assign a uniform weight to the most recent history $[t-\theta, t]$. There is a hyperparameter θ representing the length of the sliding window, or the length of history that is being summarized. The translated Laguerre (LagT) measures instead use the exponentially decaying measure, assigning more importance to recent history.

$$\text{LegT} : \mu^{(t)}(x) = \frac{1}{\theta}\mathbb{I}_{[t-\theta,t]}(x)$$

$$\text{LagT} \ : \mu^{(t)}(x) = e^{-(t-x)}\mathbb{I}_{(-\infty,t]}(x) = \begin{cases} e^{x-t} & \text{if } x \leq t \\ 0 & \text{if } x > t. \end{cases}$$

Theorem 1 (Proof [5] Appendix E) *For LegT and LagT, the hippo operators satisfying 4 are given by the linear time-invariant ordinary differential equation*

$$\frac{d}{dt}c(t) = -Ac(t) + Bf(t),$$

where

$$A \in \mathbb{R}^{N\times N}, B \in \mathbb{R}^{N\times 1} \ :$$

LegT:

$$A_{nk} = \frac{1}{\theta}\begin{cases} (-1)^{n-k}(2n+1) & \text{if } n \geq k \\ 2n+1 & \text{if } n \leq k \end{cases}, \quad B_n = \frac{1}{\theta}(2n+1)(-1)^n \tag{2.8}$$

LagT:

$$A_{nk} = \begin{cases} 1 & \text{if } n \geq k \\ 0 & \text{if } n < k \end{cases}, \quad B_n = 1 \ . \tag{2.9}$$

Equation 2.8 proves the LMU update [4]. Additionally the [5] shows that outside of the projections, there is another source of approximation. This sliding window update rule requires access to $f(t-\theta)$, which is no longer available. It works with the current coefficients $c(t)$, while hoping that the results are an accurate enough model of the function $f(x)_{x\leq t}$ that $f(t-\theta)$ can be recovered.

2.3.6 Discretization

Since the data one usually works with are discrete, it is useful to know how the HiPPO projection operators can be discretized using standard techniques for approximating the evolution of dynamical systems, so that the continuous-time HiPPO ODEs become discrete-time linear recurrences. Simply put, to construct a memory representation c_t of an input sequence f_t, HiPPO is implemented as a linear recurrence

$$c_{t+1} = A_t c_t + B_t f_t,$$

where the transition matrices A_t, B_t have closed-form formulas [5, 6]. In the continuous case, these operators consume an input function $f(t)$ and produce an output function $c(t)$. The discrete time case consumes an input sequence $(f_k)_{k\in\mathbb{N}}$, implicitly defines a function $f(t)$ where

$$f(k \cdot \Delta t) = f_k,$$

for some step size Δt, produces a function $c(t)$ through the ODE dynamics, and discretizes back to an output sequence:

$$c_k := c(k \cdot \Delta t).$$

The basic method of discretizating an ODE:

$$\frac{d}{dt}c(t) = u(t, c(t), f(t)),$$

chooses a step size Δt and performs the discrete updates

$$c(t + \Delta t) = c(t) + \Delta t \cdot u(t, c(t), f(t)).$$

In [5] they note that this process is sensitive to the discretization step size hyperparameter Δt. They also show that this provides a way to seamlessly handle timestamped data, even with missing values: the difference between timestamps indicates the (adaptive) Δt to use in discretization [46].

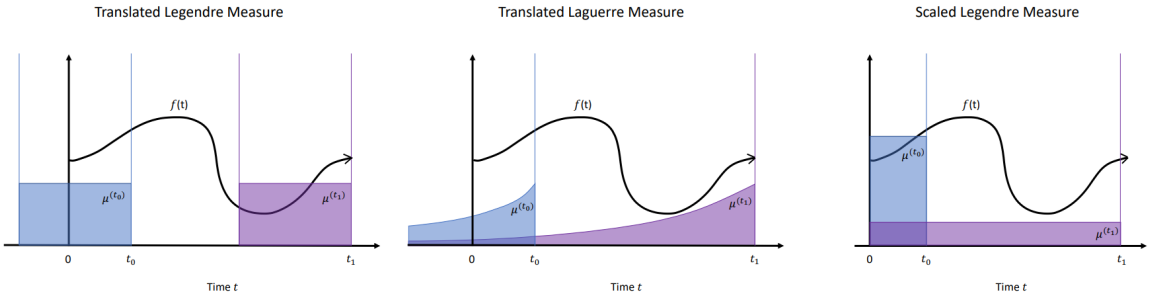
2.3.7 Memory Mechanisms of gated architectures

As a special case, one may decide not to include high order polynomials. Specifically, if $N = 1$, then the discretized version of HiPPO-LagT 2.9 becomes

$$c(t + \Delta t) = c(t) + \Delta t(-Ac(t) + Bf(t)) = (1 - \Delta t)c(t) + \Delta t f(t),$$

since $A = B = 1$. If the inputs $f(t)$ can depend on the hidden state $c(t)$ and the discretization step size Δt is chosen adaptively (as a function of input $f(t)$ and state $c(t)$), as in RNNs, then this becomes exactly a gated RNN. For instance, by stacking multiple units in parallel and choosing a specific update

Figure 2.6: Illustration of HiPPO measures. At time t_0, the history of a function $f(x)_{x \leq t_0}$ is summarized by polynomial approximation with respect to the measure $\mu^{(t_0)}$ (blue line), and similarly for time t_1 (purple line). The Translated Legendre measure (LegT) assigns weight in the window $[t - \theta, t]$. For small $t, \mu^{(t)}$ is supported on a region $x < 0$ where f is not defined. When t is large, the measure is not supported near 0, causing the projection of f to forget the beginning of the function. The Translated Laguerre (LagT) measure decays the past exponentially. It does not forget, but also assigns weight on $x < 0$. The Scaled Legendre measure (LegS) weights the entire history $[0, t]$ uniformly [5].



function, it is possible to recover the GRU update cell as a special case. The LSTM cell update is similar, with a parameterization known as tied gates. In contrast to HiPPO which uses one hidden feature and projects it onto high order polynomials, these models use many hidden features but only project them with degree 1. This view sheds light on these classic techniques by showing how they could be derived in a different way [5].

2.3.8 Scaled Legendre Measure LegS

Exposing the connection between function approximation and memory enables memory mechanisms with better theoretical properties. We only need to choose the measure carefully. Sliding windows are very popular in signal processing. However more rational approach for modeling memory is processing a the whole signal rather than a sliding window, memory should scale the window over the whole signal perceived so far, so nothing is completely forgotten [5].

2.3.8.1 Time Dynamics

The novel scaled Legendre measure (LegS) assigns a uniform weight to all previously seen data

$$[0,t] : \mu^{(t)} = \frac{1}{t}\mathbb{I}_{[0,t]}.$$

A picture is worth a thousand words in this case, great explanation of how the importance of history differs for those measures is illustrated in 2.6, which compares LegS, LegT, and LagT visually, showing the advantages of the scaled measure. To create such a memory mechanism using HiPPO framework, we only need to choose the right measure [5].

Theorem 2 (Proof [5] Appendix E) *The continuous- 2.10 and discrete- 2.11 time dynamics for HiPPO-LegS are:*

$$\frac{d}{dt}c(t) = -\frac{1}{t}Ac(t) + \frac{1}{t}Bf(t) \tag{2.10}$$

$$c_{k+1} = \left(1 - \frac{A}{k}\right)c_k + \frac{1}{k}Bf_k \tag{2.11}$$

$$A_{nk} = \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k, \\ 0 & \text{if } n < k \end{cases} \quad B_n = (2n+1)^{\frac{1}{2}} \quad .$$

The corresponding Python code is the following.

```
import numpy as np
#'N' is the order of the memory
q=np.arange(N, dtype=np.float64)
col,row=np.meshgrid(q,q)
r=2*q+1
M=-(np.where(row>=col,r,0)-np.diag(q))
T=np.sqrt(np.diag(2*q+1))
A=T @ M @ np.linalg.inv(T)
B=np.diag(T)[:, None]
C=np.ones((1, N))
D=np.zeros((1,))
```

It is possible to show that HiPPO-LegS possesses good theoretical properties as it is invariant to the input timescale, fast to compute during inference, and has bounded gradients and approximation error. Attentive readers may have noticed the absence of the window length parameter because, as mentioned before, the window size of LegS is adaptive. The projection of this measure is thus robust even when the input signal is scaled in time. Formally, it can be said that the HiPPO-LegS operator is timescale-equivariant: dilating the input f does not change the approximation coefficients.

Proposition 1 (Proof [5] Appendix E) *For any scalar* $\alpha > 0$*, if* $h(t) = f(\alpha t)$*, then* $\text{hippo}(h)(t) = \text{hippo}(f)(\alpha t)$ *In other words, if* $\gamma : t \mapsto \alpha t$ *is any dilation function, then hippo* $(f \circ \gamma) = \text{hippo}(f) \circ \gamma$

Informally, this can be noticed when inspecting the HiPPO-LegS model which has no timescale hyperparameters. The discrete recurrence 2.11 is invariant to the discretization step size. When using the LegT measure which is equivalent to LMU, we can notice it has a hyperparameter θ for the window size, and both LegT and LagT have a step size hyperparameter Δt in the discrete time case. The θ and Δt hyperparameters are important and getting their values right may require hyperparameter optimization in practice(in our case grad student descent [49] and random search was used). [28, 50, 51]. In [5] the timescale robustness is empirically demonstrated by using upsampled/-downsampled timeseries.

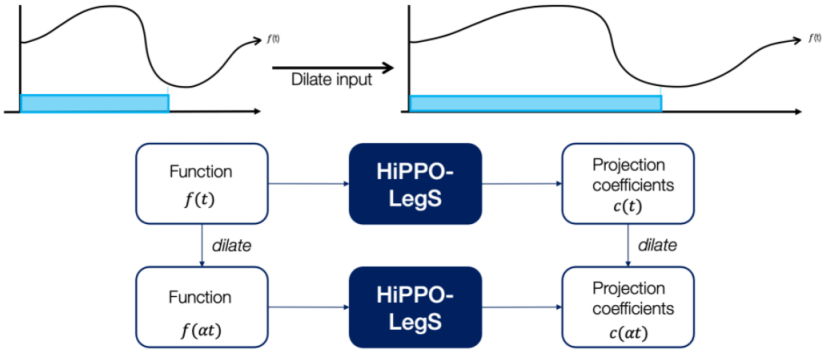


Figure 2.7: Illustration of how HiPPO-LegS is intuitively dilation equivariant [5, 6].

2.3.8.2 Computational efficiency

During the inference in HiPPO framework, computing a single step of the discrete HiPPO update requires multiplication by the (discretized) square matrix A it is the first operation that one usually focuses on when determining the efficiency of such architectures. The transition matrices A_t actually have a special structure, and the recurrence can be computed in linear instead of quadratic time. Discretization requires fast multiplication for any matrix of the form

$$I + \Delta t \cdot A \text{ and } (I - \Delta t \cdot A)^{-1},$$

for arbitrary step sizes Δt. In general that would be a $O\left(N^2\right)$ operation, LegS operators however use a fixed A matrix with a special structure that has fast multiplication algorithms for any discretization. The [5] provides an efficient implementation for fast inference using PyTorch C++ binding, the possibilities of speeding up training are discussed in the section about future work of this thesis. All experiments in this thesis were however implemented using Keras.

Proposition 2 (Proof [5] Appendix E) *Under any generalized bilinear transform discretization, each step of the HiPPO-LegS recurrence in equation 2.11 can be computed in $O(N)$ operations.*

2.3.8.3 Gradient flow

To deal with the vanishing gradient problem in RNNs [52] there is no need for an effort similar to introducing gating mechanisms in LSTM and GRU architectures as LegS is designed for memory, it avoids the vanishing gradient issue entirely. It can be shown that the gradient norms of the model decay polynomialy in time, instead of exponentially [5].

Proposition 3 (Proof [5] Appendix E) *For any time $t_0 < t_1$, the gradient norm of HiPPO-LegS operator for the output at time t_1 with respect to input at time t_0 is $\left\| \frac{\partial c(t_1)}{\partial f(t_0)} \right\| = \Theta\left(1/t_1\right)$.*

2.3.8.4 Approximation error bounds

The error rate of LegS decreases with the smoothness of the input.

Proposition 4 (Proof [5] Appendix E) *Let $f : \mathbb{R}_+ \to \mathbb{R}$ be a differentiable function, and let $g^{(t)} = \text{proj}_t(f)$ be its projection at time t by HiPPO-LegS with maximum polynomial degree $N-1$. If f is L -Lipschitz then $\left\| f_{\leq t} - g^{(t)} \right\| = O(tL/\sqrt{N})$. If f has order-k bounded derivatives then $\left\| f_{\leq t} - g^{(t)} \right\| = O\left(t^k N^{-k+1/2}\right)$*

2.3.9 Architecture of RNN with HiPPO framework

HiPPO is a simple linear recurrence that can be integrated into deep learning models in multiple ways. The attempt to construct a recurrent neural network (RNN) using HiPPO is the first that comes to mind because of their connection to dynamic systems involving a state evolving over time, just as in HiPPO. The HiPPO-RNN is the simplest way to construct this kind of architecture:

1. Begin with the standard RNN recurrence $h_t = \tau\left(h_{t-1}, x_t\right)$ which evolves the hidden state h_t by any nonlinear function τ given the input x_t
2. Project the state to a lower dimension feature f_t

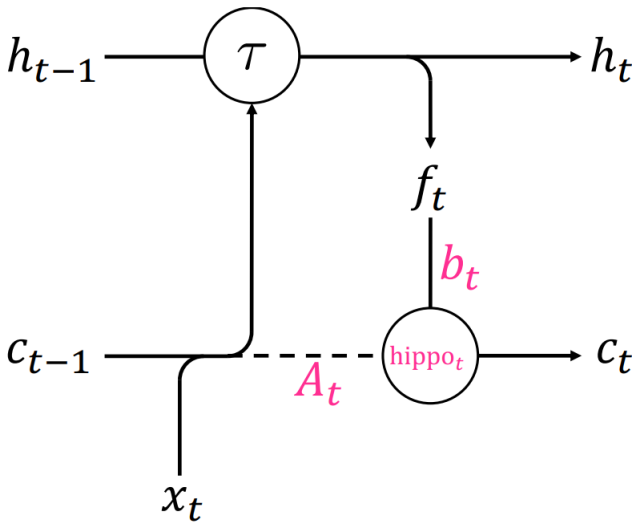


Figure 2.8: This figure shows HiPPO incorporated into a simple RNN model. hippo is the HiPPO memory operator which projects the history of the f_t features depending on the chosen measure [5].

3. Use the HiPPO recurrence to create a representation c_t of the history of f_t, which is also fed back into τ

This looks very similar to cell diagrams for other architectures such as LSTMs and GRU. [1, 27]. These classical architectures which rely on gating are closely related. The cell state of an LSTM performs the recurrence

$$c_{t+1} = \alpha_t c_t + \beta_t f_t,$$

where α_t, β are known as and "input" gates. Reccurences representing these gates are similar to the HiPPO recurrence:

$$c_{t+1} = A_t c_t + B_t f_t.$$

These gated RNNs may be percieved as a special case of HiPPO with low-order (N = 1) approximations and input dependent discretization.

In this way, HiPPO sheds light on these popular architectures already widely deployed in the wild and shows how the gating mechanism, which was originally introduced as a heuristic, could have been derived. The HiPPO-LegT model, which is the instantiation of HiPPO using the translated Legendre measure, is exactly equivalent to the previously discussed Legendre Memory Unit [4]. This means that the LMU, which is the main result of [4] is part of the HiPPO framework as it is equivalent to using the LegT measure.

The original LMU was motivated by neurobiological research and approaches the problem from a specific direction: it considers approximating

spiking neurons in the frequency domain, while directly solving an interpretable optimization problem in the time domain. More specifically, they consider time-lagged linear time invariant (LTI) dynamical systems and approximate the dynamics with Padé approximants. [4] observes that the result also has an interpretation in terms of Legendre polynomials, but not that it is the optimal solution to a natural projection problem. There is no complete proof of the update mechanism in [4, 41].

HiPPO on the other hand, suggests and online signal approximation problem, which is related to orthogonal polynomial families and makes the derivation of several related memory mechanisms simpler. The interpretation in time rather than frequency space, and the associated derivation of the LegT measure, reveals a different set of approximations stemming from the sliding window.[5].

The motivations of the two works [4, 5] are seemingly different, but they still manage to find the same memory mechanism. This hints a potential connection between sequence models and biological nervous systems. HiPPO is integrated into an RNN architecture which is shown in figure 2.3.9 with slight improvements to the LMU architecture [5].

2.3.10 HiPPO operators

Given a specific choice of measure and basis functions, we are interested in how the coefficients $c(t)$ can be computed. As an input for this process, we are given a function $f : [0, \infty) \to \mathbb{R}$ which is seen online, for which we wish to maintain a compressed representation of its history $f(x)_{\leq t} = f(x)_{x \leq t}$ at every time t. Not considering tilted measures we have $\chi = 1$ (no tilting), we also have $\zeta = 1$ and $g_n = \lambda_n p_n$.

Given a function f we want to find its approximation coeficients as it can be approximated by storing its coefficients with respect to the basis $\{g_n\}_{n<N}$. For example, in the case of no tilting $\chi = 1$, this encodes the optimal polynomial approximation of f by polynomials of degree less than N. Specifically, at time t we are trying to represent $f_{\leq t}$ as a linear combination of polynomials $g_n^{(t)}$. Since they are orthogonal, more specifically $g_n^{(t)}$ are orthogonal with respect to the scalar product of the Hilbert space defined by $\langle \cdot, \cdot \rangle_{\nu^{(t)}}$ as defined in equation 2.7, it is enough to calculate the coefficients

$$\begin{aligned} c_n(t) &= \left\langle f_{\leq t}, g_n^{(t)} \right\rangle_{\nu^{(t)}} \\ &= \int f g_n^{(t)} \frac{\omega^{(t)}}{\zeta(t) \left(\chi^{(t)}\right)^2} \\ &= \zeta(t)^{-\frac{1}{2}} \lambda_n \int f p_n^{(t)} \frac{\omega^{(t)}}{\chi^{(t)}}. \end{aligned} \tag{2.12}$$

Reconstruction at any time t, $f_{\leq t}$ is then given by

$$\begin{aligned} f_{\leq t} \approx g^{(t)} &:= \sum_{n=0}^{N-1} \left\langle f_{\leq t}, g_n^{(t)} \right\rangle_{\nu^{(t)}} \frac{g_n^{(t)}}{\left\| g_n^{(t)} \right\|_{\nu^{(t)}}^2} \\ &= \sum_{n=0}^{N-1} \lambda_n^{-2} c_n(t) g_n^{(t)} \\ &= \sum_{n=0}^{N-1} \lambda_n^{-1} \zeta^{\frac{2}{2}} c_n(t) p_n^{(t)} \chi^{(t)}. \end{aligned} \tag{2.13}$$

The equation 2.13 is the proj $_t$ operator, given the measure and basis parameters. That means that it defines the optimal approximation of $f_{\leq t}$. On the other hand, the coef $_t$ operator extracts the vector of coefficients $c(t) = (c_n(t))_{n \in [N]}$.

Consuming an input function $f(t)$, the coefficients $c(t)$ are enough to encode information about the history of f and allow online predictions needed when trying to construct a model. Defining $c(t)$ to be the vector of $c_n(t)$ from equation 2.12, the focus will be on how to calculate the coefficient function

$$c : \mathbb{R}_{\geq 0} \to \mathbb{R}^N,$$

from the input function

$$f : \mathbb{R}_{\geq 0} \to \mathbb{R} \ .$$

In the HiPPO framework, computing these coefficients over time is done by viewing the problem as a dynamical system. Obtaining the following by differentiating the formula 2.12.

$$\begin{aligned} \frac{d}{dt} c_n(t) =& \zeta(t)^{-\frac{1}{2}} \lambda_n \int f(x) \left(\frac{\partial}{\partial t} p_n(t,x) \right) \frac{\omega}{\chi}(t,x) \mathrm{d}x \\ &+ \int f(x) \left(\zeta^{-\frac{1}{2}} \lambda_n p_n(t,x) \right) \left(\frac{\partial}{\partial t} \frac{\omega}{\chi}(t,x) \right) \mathrm{d}x. \end{aligned} \tag{2.14}$$

Here they are using the assumption that ζ is constant for all t. Formally the function $c(t) \in \mathbb{R}^{N-1}$ denotes the vector of all coefficients $(c_n(t))_{0 \leq n < N}$. The key idea is that if $\frac{\partial}{\partial t} P_n$ and $\frac{\partial}{\partial t} \frac{\omega}{\chi}$ have closed forms that can be related back to the polynomials P_k, then an ordinary differential equation can be written for $c(t)$. This allows these coefficients $c(t)$ to be computed online. Since $\frac{d}{dt} P_n^{(t)}$ is a polynomial (in x) of degree $n-1$, it can be written as linear combinations of $P_0, \ldots, P_{n-1}$, so the first term in 2.14 is a linear combination of $c_0, \ldots, c_{n-1}$. For many weight functions ω, it is possible to find scaling function χ such that $\frac{\partial}{\partial t} \frac{\omega}{x}$ can also be written in terms of $\frac{\omega}{x}$ itself, if that is the case 2.14 is also a linear combination of $c_0, \ldots, c_{N-1}$ and the input f. Thus this often yields a closed-form linear ODE for $c(t)$. Suppose that equation 2.14 is reduced to

dynamics of the form

$$\frac{d}{dt}c(t) = -A(t)c(t) + B(t)f(t).$$

Then, letting $\Lambda = \text{diag}_{n\in[N]}\{\lambda_n\}$

$$\frac{d}{dt}\Lambda^{-1}c(t) = -\Lambda^{-1}A(t)\Lambda\Lambda^{-1}c(t) + \Lambda^{-1}B(t)f(t),$$

allows them to reparameterize the coefficients $(\Lambda^{-1}c(t) \to c(t))$ then the normalized coefficients projected onto the orthonormal basis satisfy dynamics and associated reconstruction

$$\frac{d}{dt}c(t) = -\left(\Lambda^{-1}A(t)\Lambda\right)c(t) + \left(\Lambda^{-1}B(t)\right)f(t)$$
$$f_{\leq t} \approx g^{(t)} = \sum_{n=0}^{N-1} \zeta^{\frac{1}{2}} c_n(t) p_n^{(t)} \chi^{(t)}.$$

These are the hippo and proj_t operators [5].

2.3.11 Derivation of LMU in HiPPO framework

Given the previously established foundations, it is possible to show the derivation of Translated Legendre (HiPPO-LegT). The way to derive a HiPPO operator in general consists of four basic steps.

- First, we need to have some measure and basis, so we define the measure $\mu^{(t)}$ or weight $\omega(t,x)$ and basis functions $p_n(t,x)$.
- After that, we need to have the derivatives at hand, so we compute the derivatives of the measure and basis functions,
- Then Coefficient Dynamics need to be figured out. Therefore, we plug them into the coefficient dynamics from the equation 2.14 to derive the ODE that describes how to compute the coefficients $c(t)$,
- The last step is the reconstruction. It is necessary to provide a complete formula to reconstruct an approximation to the function $f_{\leq t}$, which is the optimal projection under this measure and basis.

This measure fixes a window length θ and slides it across time. As a Measure and Basis uniform weight function supported on the interval $[t-\theta, t]$ is used and Legendre polynomials $P_n(x)$, translated from [-1,1] to $[t-\theta, t]$, are used as basis functions:

$$\omega(t,x) = \frac{1}{\theta}\mathbb{I}_{[t-\theta,t]}$$
$$p_n(t,x) = (2n+1)^{1/2}P_n\left(\frac{2(x-t)}{\theta} + 1\right)$$
$$g_n(t,x) = \lambda_n p_n(t,x).$$

The derivations included in this thesis use use no tilting so $\chi = 1$ and $\zeta = 1$ see [5] for further explanation of what the tilted measure means. To further simplify this problem, we can leave λ_n unspecified for now, they specify constants which we are able to leave unspecified - the reason we are to leave them unspecified is that the orthogonal basis is not unique. At the endpoints, these basis functions satisfy

$$g_n(t,t) = \lambda_n(2n+1)^{\frac{1}{2}}$$
$$g_n(t,t-\theta) = \lambda_n(-1)^n(2n+1)^{\frac{1}{2}}.$$

The derivative of the measure is

$$\tfrac{\partial}{\partial t}\omega(t,x) = \tfrac{1}{\theta}\delta_t - \tfrac{1}{\theta}\delta_{t-\theta}.$$

The derivative of Legendre polynomials can be expressed as linear combinations of other Legendre polynomials as mentioned earlier, we make use of that fact to get

$$\frac{\partial}{\partial t}g_n(t,x) = \lambda_n(2n+1)^{\frac{1}{2}} \cdot \frac{-2}{\theta}P_n'\left(\frac{2(x-t)}{\theta}+1\right)$$

$$= \lambda_n(2n+1)^{\frac{1}{2}}\frac{-2}{\theta}\left[(2n-1)P_{n-1}\left(\frac{2(x-t)}{\theta}+1\right) + (2n-5)P_{n-3}\left(\frac{2(x-t)}{\theta}+1\right) + \ldots\right]$$

$$= -\lambda_n(2n+1)^{\frac{1}{2}}\frac{2}{\theta}\left[\lambda_{n-1}^{-1}(2n-1)^{\frac{1}{2}}g_{n-1}(t,x) + \lambda_{n-3}^{-1}(2n-3)^{\frac{1}{2}}g_{n-3}(t,x) + \ldots\right], \tag{2.15}$$

using the equation $P_n' = (2n-1)P_{n-1} + (2n-5)P_{n-3} + \ldots$ here. As a special case of the LegT measure, it is necessary to consider an approximation due to the nature of the sliding window measure. When analyzing $\frac{d}{dt}c(t)$, the value $f(t-\theta)$ will be used. However, at time t this input is no longer available. Instead, it relies on compressed representation of the function, by the reconstruction equation that says that at any time t, $f_{\leq t}$ can be explicitly reconstructed as

$$\begin{aligned} f_{\leq t} \approx g^{(t)} &:= \sum_{n=0}^{N-1} \left\langle f_{\leq t}, g_n^{(t)} \right\rangle_{\nu^{(t)}} \frac{g_n^{(t)}}{\left\|g_n^{(t)}\right\|_{\nu^{(t)}}^2} \\ &= \sum_{n=0}^{N-1} \lambda_n^{-2} c_n(t) g_n^{(t)} \\ &= \sum_{n=0}^{N-1} \lambda_n^{-1} \zeta^{\frac{1}{2}} c_n(t) p_n^{(t)} \chi^{(t)}. \end{aligned}$$

In the case where the approximation is succeeding so far, this results in

$$f_{\leq t}(x) \approx \sum_{k=0}^{N-1} \lambda_k^{-1} c_k(t)(2k+1)^{\frac{1}{2}} P_k\left(\tfrac{2(x-t)}{\theta}+1\right)$$
$$f(t-\theta) \approx \sum_{k=0}^{N-1} \lambda_k^{-1} c_k(t)(2k+1)^{\frac{1}{2}}(-1)^k,$$

and the coefficient dynamics can now be derived. Plugging the derivatives of this measure and basis into the equation 2.14 gives

$$\begin{aligned}\frac{d}{dt}c_n(t) &= \int f(x)\left(\frac{\partial}{\partial t}g_n(t,x)\right)\omega(t,x)\mathrm{d}x \\ &\quad + \int f(x) g_n(t,x)\left(\frac{\partial}{\partial t}\omega(t,x)\right)\mathrm{d}x \\ &= -\frac{\lambda_n(t)}{\theta_k} + (2n+1)^{\frac{1}{2}}\frac{\lambda_n}{\theta}f(t),\end{aligned}$$

where

$$M_{nk} = \begin{cases} 1 & \text{if } k \leq n \\ (-1)^{n-k} & \text{if } k \geq n. \end{cases}$$

Considering different possible values of λ_n. The first one is the more natural $\lambda_n = 1$, which turns g_n into an orthonormal basis. This yields

$$\begin{aligned}\frac{d}{dt}c(t) &= -\frac{1}{\theta}Ac(t) + \frac{1}{\theta}Bf(t) \\ A_{nk} &= (2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}}\begin{cases} 1 & \text{if } k \leq n \\ (-1)^{n-k} & \text{if } k \geq n \end{cases} \\ B_n &= (2n+1)^{\frac{1}{2}}.\end{aligned}$$

The second case takes $\lambda_n = (2n+1)^{\frac{1}{2}}(-1)^n$. This yields

$$\begin{aligned}\frac{d}{dt}c(t) &= -\frac{1}{\theta}Ac(t) + \frac{1}{\theta}Bf(t) \\ A_{nk} &= (2n+1)\begin{cases} (-1)^{n-k} & \text{if } k \leq n \\ 1 & \text{if } k \geq n \end{cases} \\ B_n &= (2n+1)(-1)^n.\end{aligned}$$

This is exactly the LMU update equation [4, 5]. Using the reconstruction by equation

$$\begin{aligned}f_{\leq t} \approx g^{(t)} &:= \sum_{n=0}^{N-1}\left\langle f_{\leq t}, g_n^{(t)}\right\rangle_{\nu^{(t)}} \frac{g_n^{(t)}}{\left\|g_n^{(t)}\right\|_{\nu^{(t)}}^2} \\ &= \sum_{n=0}^{N-1}\lambda_n^{-2}c_n(t)g_n^{(t)} \\ &= \sum_{n=0}^{N-1}\lambda_n^{-1}\zeta^{\frac{1}{2}}c_n(t)p_n^{(t)}\chi^{(t)},\end{aligned}$$

at every time t

$$f(x) \approx g^{(t)}(x) = \sum_n \lambda_n^{-1} c_n(t)(2n+1)^{\frac{1}{2}} P_n\left(\frac{2(x-t)}{\theta} + 1\right).$$

2.3.12 Derivation of LegS

Following the same general steps as in the previous section, the derivation of Scaled Legendre (HiPPO-LegS) can be done in the following way. The scaled Legendre is the only method that uses a measure with varying width from the measures presented in [5]. That is an interesting property as it allows it to get rid of the timescale parameter. It is necessary to choose a measure and basis to create an instantiation of the framework in the case

$$\begin{aligned}\omega(t,x) &= \frac{1}{t}\mathbb{I}_{[0,t]} \\ g_n(t,x) = p_n(t,x) &= (2n+1)^{\frac{1}{2}} P_n\left(\frac{2x}{t} - 1\right).\end{aligned}$$

Here, P_n are the basic Legendre polynomials. They use no tilting, that means $\chi(t,x) = 1, \zeta(t) = 1$, and $\lambda_n = 1$ so that the functions $g_n(t,x)$ are an orthonormal basis. Derivatives are derived in a way where we first differentiate the measure and basis:

$$\begin{aligned}\frac{\partial}{\partial t}\omega(t,\cdot) &= -t^{-2}\mathbb{I}_{[0,t]} + t^{-1}\delta_t = t^{-1}\left(-\omega(t) + \delta_t\right) \\ \frac{\partial}{\partial t} g_n(t,x) &= -(2n+1)^{\frac{1}{2}} 2xt^{-2} P_n'\left(\frac{2x}{t} - 1\right) \\ &= -(2n+1)^{\frac{1}{2}} t^{-1}\left(\frac{2x}{t} - 1 + 1\right) P_n'\left(\frac{2x}{t} - 1\right).\end{aligned}$$

Now defining $z = \frac{2x}{t} - 1$ and applying the properties of derivatives of Legendre polynomials

$$\begin{aligned}(x+1)P_n'(x) &= P_{n+1}' + P_n' - (n+1)P_n \\ &= nP_n + (2n-1)P_{n-1} + (2n-3)P_{n-2} + \dots\end{aligned}$$

this yields

$$\begin{aligned}\frac{\partial}{\partial t} g_n(t,x) &= -(2n+1)^{\frac{1}{2}} t^{-1}(z+1)P_n'(z) \\ &= -(2n+1)^{\frac{1}{2}} t^{-1}\left[nP_n(z) + (2n-1)P_{n-1}(z) + (2n-3)P_{n-2}(z) + \dots\right] \\ &= -t^{-1}(2n+1)^{\frac{1}{2}}\left[n(2n+1)^{-\frac{1}{2}} g_n(t,x) + (2n-1)^{\frac{1}{2}} g_{n-1}(t,x) + (2n-3)^{\frac{1}{2}} g_{n-2}(t,x) + \dots\right],\end{aligned}$$

the Coefficient Dynamics are obtained by plugging these into

$$\begin{aligned}\frac{d}{dt}c_n(t) =&\zeta(t)^{-\frac{1}{2}}\lambda_n \int f(x)\left(\frac{\partial}{\partial t}p_n(t,x)\right)\frac{\omega}{\chi}(t,x)\mathrm{d}x \\ &+\int f(x)\left(\zeta^{-\frac{1}{2}}\lambda_n p_n(t,x)\right)\left(\frac{\partial}{\partial t}\frac{\omega}{\chi}(t,x)\right)\mathrm{d}x,\end{aligned}$$

from which the following is obtained

$$\begin{aligned}\frac{d}{dt}c_n(t) =& \int f(x)\left(\frac{\partial}{\partial t}g_n(t,x)\right)\omega(t,x)\mathrm{d}x + \int f(x)g_n(t,x)\left(\frac{\partial}{\partial t}\omega(t,x)\right)\mathrm{d}x \\ =& -t^{-1}(2n+1)^{\frac{1}{2}}\left[n(2n+1)^{-\frac{1}{2}}c_n(t) + (2n-1)^{\frac{1}{2}}c_{n-1}(t) + (2n-3)^{\frac{1}{2}}c_{n-2}(t) + \ldots\right] \\ & - t^{-1}c_n(t) + t^{-1}f(t)g_n(t,t) \\ =& t^{-1}(2n+1)^{\frac{1}{2}}\left[(n+1)(2n+1)^{-\frac{1}{2}}c_n(t) + (2n-1)^{\frac{1}{2}}c_{n-1}(t) + (2n-3)^{\frac{1}{2}}c_{n-2}(t) + \ldots\right] \\ & + t^{-1}(2n+1)^{\frac{1}{2}}f(t),\end{aligned}$$

where $g_n(t,t) = (2n+1)^{\frac{1}{2}}P_n(1) = (2n+1)^{\frac{1}{2}}$ has been used, next vectorizing this yields equation:

$$\frac{d}{dt}c(t) = -\frac{1}{t}Ac(t) + \frac{1}{t}Bf(t) \tag{2.16}$$

$$\begin{aligned}A_{nk} &= \begin{cases}(2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k.\end{cases} \\ B_n &= (2n+1)^{\frac{1}{2}}\end{aligned} \tag{2.17}$$

This can also be written as

$$\frac{d}{dt}c(t) = -t^{-1}D\left[MD^{-1}c(t) + \mathbf{1}f(t)\right],$$

where $D := \mathrm{diag}\left[(2n+1)^{\frac{1}{2}}\right]_{n=0}^{N-1}$, $\mathbf{1}$ is the all ones vector, and the state matrix M is

$$M = \begin{bmatrix}1 & 0 & 0 & 0 & \ldots & 0 \\ 1 & 2 & 0 & 0 & \ldots & 0 \\ 1 & 3 & 3 & 0 & \ldots & 0 \\ 1 & 3 & 5 & 4 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 3 & 5 & 7 & \ldots & N\end{bmatrix}, \quad \text{that is,} \quad M_{nk} = \begin{cases}2k+1 & \text{if } k < n \\ k+1 & \text{if } k = n. \\ 0 & \text{if } k > n\end{cases}$$

Previously shown equations 2.16 and 2.17 form a linear dynamical system, they are however dilated by a time-varying factor t^{-1}, which due to the scaled measure [5, 6].

2.3.13 Comparing LegS and LegT

After the detailed description of both LMU(LegT) and LegS, it would probably be beneficial to compare them. LegS does not need any hyperparameters governing the timescale. However, if we choose the θ in a LegT just right. Meaning, we choose it to match the length of the sequence. $\theta = T$ where T is the final time range. We can see that at the end of consuming the input function (time $t = T$), the measures $\mu^{(t)}$ for LegS and LegT are both equal to $\frac{1}{T}\mathbb{I}_{[0,T]}$. This means that the aproximation $\text{proj}_T(f)$ is specifying the same function for LegS and LegT at time $t = T$. There is however still one difference, LegT has an additional approximation term for $f(t-\theta)$ while calculating the update at every time t because it is using a sliding window [5].

2.3.14 Analyzing norm of the gradient

Analyzing the discrete time case under the Euler discretization , where the HiPPO-LegS recurrent update is equation 2.10, it is possible to restate for convenience in the following way:

$$c_{k+1} = \left(1 - \frac{A}{k}\right) c_k + \frac{1}{k} B f_k.$$

These gradient asymptotics hold under multiple other discretizations. It can be shown that for any time $k < \ell$, the gradient norm of the HiPPO-LegS operator for the output at time $\ell + 1$ with respect to input at time k is $\left\| \frac{\partial c_{t+1}}{\partial f_k} \right\| = \Theta(1/\ell)$. To actually prove that in [5] they start with setting N to be a constant. Without loss of generality, they assume $k > 2$, as the gradient change for a single initial step is bounded. By unrolling the recurrence (2.10), the dependence of $c_{\ell+1}$ on c_k and $f_k, \ldots, f_\ell$ can be made explicit:

$$\begin{aligned}
c_{\ell+1} = &\left(I - \frac{A}{\ell}\right) \ldots \left(I - \frac{A}{k}\right) c_k \\
&+ \left(I - \frac{A}{\ell}\right) \ldots \left(I - \frac{A}{k+1}\right) \frac{B}{k} f_k \\
&+ \left(I - \frac{A}{\ell}\right) \ldots \left(I - \frac{A}{k+2}\right) \frac{B}{k+1} f_{k+1} \\
&\vdots \\
&+ \left(I - \frac{A}{\ell}\right) \frac{B}{\ell - 1} f_{\ell-1} \\
&+ \frac{B}{\ell} f_\ell,
\end{aligned}$$

therefore

$$\frac{\partial c_{\ell+1}}{\partial f_k} = \left(I - \frac{A}{\ell}\right) \ldots \left(I - \frac{A}{k+1}\right) \frac{B}{k}.$$

Notice that A has distinct eigenvalues $1, 2, \ldots, N$, since those are the elements of its diagonal and A is triangular (Theorem 2). That means that the matrices $I - \frac{A}{\tau}, \ldots, I - \frac{A}{k+1}$ can be diagonalized using change of basis. The assumption of this method is that we suppose that the basis vectors are linearly independent eigenvectors, which is exactly the case here. Matrix multiplication does not commute in general, but diagonal matrices commute. The gradient can be be expressed as $PDP^{-1}B$ assuming an invertible matrix P and a diagonal matrix D. The actual norm is then bounded from below (up to constant) by the smallest singular value of P and $\|P^{-1}B\|$, both of which are nonzero constants, and the largest diagonal entry of D. It thus suffices to bound this largest diagonal entry of D, which is the largest eigenvalue of this product,

$$\rho = \left(1 - \frac{1}{\ell}\right) \ldots \left(1 - \frac{1}{k+1}\right) \frac{1}{k} \ .$$

The problem reduces to showing that $\rho = \Theta(1/l)$. They use the following facts about the function $\log\left(1 - \frac{1}{x}\right)$. It is an increasing function, so

$$\log\left(1 - \frac{1}{x}\right) \geq \int_{x-1}^{x} \log\left(1 - \frac{1}{\lambda}\right) \mathrm{d}.\lambda$$

Additionally, its antiderivative is

$$\begin{aligned}
\int \log\left(1 - \frac{1}{x}\right) \mathrm{d}x &= \int \log(x-1) - \log(x) \mathrm{d}x \\
&= (x-1)\log(x-1) - x\log(x) \\
&= x \log\left(1 - \frac{1}{x}\right) - \log(x-1).
\end{aligned}$$

Using those two facts we get

$$\begin{aligned}
\log\left(1 - \frac{1}{\ell}\right) \ldots \left(1 - \frac{1}{k+1}\right) &= \sum_{i=k+1}^{\ell} \log\left(1 - \frac{1}{i}\right) \\
&\geq \sum_{i=k+1}^{\ell} \int_{i-1}^{i} \log\left(1 - \frac{1}{x}\right) \mathrm{d}x \\
&= \int_{k}^{\ell} \log\left(1 - \frac{1}{x}\right) \mathrm{d}x \\
&= \left[(x-1)\log(x-1) - x\log(x)\right]|_{k}^{\ell} \\
&= \ell \log\left(1 - \frac{1}{\ell}\right) - \log(\ell - 1) \\
&\quad - \left(k \log\left(1 - \frac{1}{k}\right) - \log(k-1)\right).
\end{aligned}$$

Another thing that should be noticed is the fact that $x \log\left(1 - \frac{1}{x}\right)$ is an increasing function, and also a function bounded from above since it is negative,

from that we get that it is $\Theta(1)$. So we have

$$\log \rho \geq \Theta(1) - \log(\ell - 1) + \log(k - 1) - \log(k).$$

All inequalities are asymptotically tight, so that $\rho = \Theta(1/\ell)$ [5].

Chapter 3

Experiments

Here we describe the experiments conducted as part of this thesis. All of them were implemented using Keras framework, additionally one of the audio classification experiments was replicated using Nengo framework to demonstrate how such networks could be implemented on specialized hardware by using a simulator of such hardware on a classical CPU.

3.1 Audio classification

Audio classification in this thesis is concerned with classifying audio data points using a single label.

3.1.1 Preprocessing

First, we want to test the previously mentioned ability of the the novel architecture LMU and LegS capture longterm dependencies on downscaled/upscaled signal. We perform an unusual preprocessing on this dataset, converting the spectrogram of each audio sample to a sequence of fixed length 4096 and feed the network one element at a time. The aim of this experiment is to directly compare the ability of different architectures to handle long sequences of upsampled/downsampled signals. The upsampling/downsampling is achieved by taking audio recordings of different lengths and converting them to a spectrogram of a fixed length. This means that for some points in the dataset, the data point is shrinked to fit the fixed size and for other data points the data is stretched to fit the fixed size.

3.1.2 Mel-frequency cepstrum

The spectrogram mentioned in the previous section is actually Mel-frequency cepstrum (MFCC) [53]. MFCC is perhaps the best known and most popular method of audio preprocessing. MFCC's are based on the known variation

of the human ear's critical bandwidth with frequency. The MFCC technique makes use of two types of filters, namely, linearly spaced filters and logarithmically spaced filters. To capture the phonetically important characteristics of speech, the signal is expressed in the mel frequency scale. This scale has a linear frequency spacing below 1000 Hz and a logarithmic spacing above 1000 Hz. Normal speech waveforms may vary from time to time depending on the physical condition of the speaker and his/her vocal cord. Rather than the speech waveforms themselves, MFFCs are less susceptible to the said variations [53].

3.1.3 Sequential Spoken Digits

For this experiment a simple audio/speech dataset consisting of recordings of spoken digits is used [54]. To test the ability of the RNNs considered in this experiment to learn long-range dependencies, we perform preprocessing transforming the data into a spectrogram of fixed length, and then we flatten the spectrogram into a 1 dimensional sequence resulting in 4096 timesteps for each utterance. That should be enough timesteps to demonstrate the limitations of the currently popular architectures LSTM and GRU.

3.1.3.1 Results

Figure 3.1: Results in terms of validation accuracy on the spoken digit classification task, same architecture was used for all experiments we only switched the recurrent unit for each experiment.

Results			
Model	Accuracy	Trainable parameters	Training time
LSTM	0.1060	1057802	15 min
GRU	0.1160	791040	12 min
LMU	0.9440	101771	33 min
LMU	0.9520	101771	68 min
LMU(NengoDL)	0.9550	102017	38 min
LegS	0.8540	102027	31 min
LegS	**0.9540**	102027	133 min

We can see that LSTM and GRU failed to learn from these long sequences. Another thing we can notice is that LMU converges to a fair accuracy way faster. Even thought there are ways to make this dataset more convenient for architectures with limited ability to process long-time dependencies and upsampled/downsampled signals. That was not the aim of this experiment nor of this thesis in general. The part of this experiment using LMU(LegT) to perform this task was also replicated using Nengo [55]framework instead

of Keras to demonstrate how this kind of model could be implemented on specialized hardware like Intel Loihi which takes advantage of model of computation called spiking neural network. In order to perform the experiment on classic hardware we simulated the spiking network using TensorFlow backend for Nengo called NengoDL [56]. The topic of spiking neural networks will be briefly discussed in the next chapter.

3.1.4 Sequential Environmental Sound classification

After confirming that treating sound as a sequence is actually a feasible approach, we perform another experiment on a comparatively bigger dataset called Urban8k [57]. It is concerned with the classification of environmental sounds. Namely, there are ten different categories which we are trying to classify

```
0 = air_conditioner
1 = car_horn
2 = children_playing
3 = dog_bark
4 = drilling
5 = engine_idling
6 = gun_shot
7 = jackhammer
8 = siren
9 = street_music
```

We perform the same preprocessing steps like in the previous sound classification experiment, resulting in sequences of 4096 time-steps and use a similar architecture.

3.1.4.1 Results

After training the network 10 times, once for each fold, we report the mean validation accuracy. The LSTM and GRU were omitted.

Figure 3.2: Results in terms of validation accuracy on urban8k dataset under 10-fold crossvalidation.

Results			
Model	Accuracy	Trainable parameters	Training time
LMU	0.4877	402475	6 hours
LMU	0.5659	402475	50
LegS	0.4906	485811	7 hours
LegS	**0.5702**	485811	49 hours

As this is a popular dataset, we have an opportunity to compare the model with other approaches which are using different architectures and preprocessing methods [7].

Figure 3.3: Results gathered from litarature in terms of 10-fold crossvalidation mean accuracy on urban8k dataset under 10-fold crossvalidation. Data augumentation seems to be neccesary to achieve state of the art results [7]

Results	
Model	Accuracy
CNN	0.5118
LSTM + Attention	0.5950
Capsule Network + DA	0.7650
CNN + DA	**0.7900**

We can see from the figure 3.1.4.1 that this sequential approach is able to match with the approaches using convolutional neural networks and LSTMs with attention. The state-of-the-art results are mainly based on heavy data augumentation (DA), which we did not perform

3.2 Natural language processing

Recurrent neural networks are a popular architecture used for natural language processing tasks. We will compare the novel architecture-based models with the the models based on LSTM and GRU architectures. In essence, replacing the recurrent units in the currently used architectures by the novel ones LMU(LegT) and LegS. We want to determine if the novel architectures can be treated as drop-in replacement for the currently used LSTM and GRU architectures.

3.2.1 Named Entity Recognition

In this experiment, we are going to be dealing with English text. Some of the words in the text are rigid designators. A rigid designator designates the same object in all possible worlds in which that object exists and never designates anything else. Named entity recognition (NER) is the task to identify mentions of rigid designators from text belonging to predefined semantic types such as person, location, organization, etc. NER always serves as the foundation for many natural language applications such as question answering, text summarization, and machine translation [58].

The term “Named Entity” refers to the named entities in the task of identifying the names of organizations, people, and geographic locations in the text, as well as currency, time, and percentage expressions. There has been increas-

ing interest in NER, and various scientific events devote much effort to this topic. The problem was defined in [8] and restricted the definition of named entities to: "A NE is a proper noun, serving as a name for something or someone". This restriction is justified by the significant percentage of proper nouns present in the corpus. Another work [59] claimed that the word "Named" restricted the task to only those entities for which one or many rigid designators stands for the referent. Rigid designators, as defined in [60], include proper names and natural kind terms like biological species and substances. Despite the various definitions of NEs, researchers have reached a common consensus on the types of NEs to recognize. We generally divide NEs into two categories: generic NEs (e.g., person and location) and domain-specific NEs (e.g., proteins, enzymes, and genes). In this experiment, we focus on generic NEs in English language.

3.2.1.1 Dataset

The Groningen Meaning Bank (GMB) [61] is a dataset of multisentence texts, together with annotations for parts-of-speech, named entities, lexical categories, and other natural language structural phenomena.

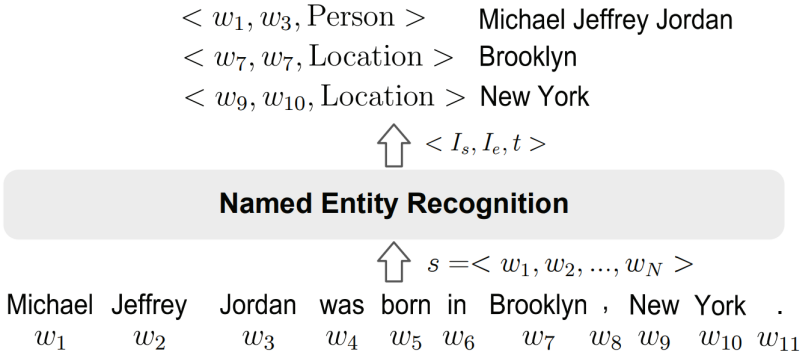


Figure 3.4: Illustration of named entity recognition task [8].

3.2.1.2 Preprocessing

There is 35178 unique words in the corpus and 17 unique named entity tags, we create a mapping of tags and words to natural numbers and use that for training, since we care about named entities no further processing of the corpus is done. We create sequences of maximum length 50 from the corpus with sequences of corresponding length representing the tags assigned to words in the sequence. We use that for supervised training. Specific example of a processed sentence about Rwandan genocide in 1994 is shown in figure 3.2.1.2.

Figure 3.5: An example sequence that is fed to the network and comparison of ground truth and predicted output. The specific model used for this example was LSTM based, but it does not differ for other models

Word	True	Pred
A	O	O
United	B–org	B–org
Nations	I–org	I–org
war	O	O
crimes	O	O
court	O	O
has	O	O
sentenced	O	O
a	O	O
former	O	O
Rwandan	B–gpe	B–gpe
mayor	O	O
to	O	O
15	O	B–tim
years	O	O
in	O	O
prison	O	O
for	O	O
his	O	O
role	O	O
in	O	O
the	O	O
1994	B–tim	B–tim
genocide	O	O

3.2.1.3 Architecture

Even though four different architectures were used in these experiments, they are all similar and differ only in the recurrent unit used. An embedding layer is used followed by spatial dropout and after that a recurrent layer is used in a bidirectional manner and it is followed by a timedistributed dense layer that outputs a sequence of tags for the input sequence of words.

The recurrent layer is used in a bidirectional manner as it is beneficial for the named entity recognition model where the context can come after the word the model was supposed to tag [59], this architecture is also easily appliable for the RNN models using LMU and LegS architectures.

3.2.1.4 Results

In this section, we compare the accuracy of the classic RNN approaches LSTM,GRU and the novel ones LMU and HiPPO LegS

Figure 3.6: Results in terms of validation accuracy on the named entity recognition task, same architecture was used for all experiments we only switched the recurrent unit for each experiment.

Results			
Model	Accuracy	Trainable parameters	Training time
Bidirectional-LSTM	0.9855	1883167	50 min
Bidirectional-GRU	**0.9860**	1853567	52 min
Bidirectional-LMU	0.9849	1986331	21 min
Bidirectional-LegS	0.9853	1986843	34 min

We can see that the novel approaches perform on par with GRU and LSTM while bringing new theoretical guarantees and the possibility of implementation on neuromorphic hardware to the table. In this particular case, the training using novel architectures was actually faster.

3.2.2 Sentiment Classification

The last experiment conducted uses the widely known IMDB movie reviews [62] dataset, which provides a set of 25,000 highly polar movie reviews for training and 25,000 for testing. It is one of the standard benchmarks in the field of NLP.

3.2.2.1 Results

We see that aside from longer training time, the novel architectures seem to perform almost on par with the LSTM and GRU architectures.

Figure 3.7: Results in terms of validation accuracy on the sentiment classification task, same architecture was used for GRU and LSTM that is two bidirectional recurrent layers, for LegT and LegS one recurrent layer proved to be enough instead of two.

Results			
Model	Accuracy	Trainable parameters	Training time
Bidirectional-LSTM	0.8590	2757761	3 min
Bidirectional-GRU	**0.8784**	2709121	2 min
Bidirectional-LMU	0.8648	2574721	40 min
Bidirectional-LegS	0.8630	2593409	73 min

3.2.3 Exploring hyperparameters of the novel approaches

Because the literature concerned with the topic is still rather sparse, the hyperparameters available in the HiPPO framework instantiations are worth exploring.

The first conducted experiment was concerned with performance of the models as we change the number of recurrent units, which results in fewer trainable parameters in the recurrent layer. In essence, we started with models from the previous experiment and decreased the size of the recurrent layers.

We can see that using less recurrent units does reduce the accuracy considerably when using LSTM and GRU, however with LMU and LegS it is not the case. The rationale behind this observation may be that both LMU and LegS are able to store more complex relationships because of the way their memory works even when using a single unit.

In the second experiment, we explore how the choice of order d of the space used for projection in both LMU and HiPPO-LegS affects accuracy of the model. Order is the number of degrees in the transfer function of the LTI system used to represent the history. As explained earlier, the novel approaches use projection into a lower dimension, this parameter sets the number of Legendre polynomials used to orthogonally represent the signal. In the case of LMU, a sliding window of the signal is represented instead. We stick to using 64 recurrent units in this experiment. The [4] uses rather high order of polynomials (256) to achieve state-of-art performance on the PSMNIST problem. We will explore if lower dimension memory is sufficient for the problem of sentiment classification.

We can see that reducing the memory order to as low as 1 or 2 dimensions seems to be sufficient for this problem. That is an interesting observation, which may suggest that the novel approach introduces unnecessarily complex dynamics into the model.

Figure 3.8: Results in terms of validation accuracy on the sentiment classification task as the number of recurrent units is decreased

Results			
Model	Accuracy	Trainable recurrent parameters	Recurrent units
Bidirectional-LSTM	**0.8602**	98816	64 Units
Bidirectional-LSTM	0.8566	66048	32 Units
Bidirectional-LSTM	0.8469	24832	16 Units
Bidirectional-LSTM	0.8323	10368	8 Units
Bidirectional-LSTM	0.8266	1040	1 Unit
Bidirectional-GRU	**0.8780**	74496	64 Units
Bidirectional-GRU	0.8749	31104	32 Units
Bidirectional-GRU	0.8589	14016	16 Units
Bidirectional-GRU	0.8552	6624	8 Units
Bidirectional-GRU	0.8552	786	1 Unit
Bidirectional-LMU	**0.8648**	63832	64 Units
Bidirectional-LMU	0.8580	26072	32 Units
Bidirectional-LMU	0.8480	13880	16 Units
Bidirectional-LMU	0.8606	8168	8 Units
Bidirectional-LMU	0.8598	1220	1 Unit
Bidirectional-LegS	0.8630	31232	64 Units
Bidirectional-LegS	0.8470	14784	32 Units
Bidirectional-LegS	0.8530	8096	16 Units
Bidirectional-LegS	0.8542	5136	8 Units
Bidirectional-LegS	**0.8798**	2756	1 Unit

Figure 3.9: Results in terms of validation accuracy on the sentiment classification task as the dimension of the space we project to is decreased. The number of units is fixed to 64

Results			
Model	Accuracy	Trainable recurrent parameters	Order
Bidirectional-LMU	**0.8694**	25092	1
Bidirectional-LMU	0.8614	25228	2
Bidirectional-LMU	0.8540	26128	8
Bidirectional-LMU	0.8598	31668	32
Bidirectional-LegS	0.8598	1220	1
Bidirectional-LegS	**0.8700**	25232	2
Bidirectional-LegS	0.8670	26144	8
Bidirectional-LegS	0.8630	31232	32

Chapter 4

Neuromorphic computing and spiking neural network

In this section we will briefly introduce an alternative spiking neuron model closer to its biological counterpart. Then we move to a brief introduction of a formal framework for describing architectures of spiking neurons. Finally, we will introduce the software and hardware tools related to this framework.

4.1 Izikevich neuron model

Spiking Neural Networks are said to be the third neural network generation [63], they have been developed to closely imitate natural neural networks, meaning they are close to their biological counterparts in terms of their internal dynamics. Spiking neural networks follow the same trend as computational neuroscience. A popular model that features both efficiency and biologically realistic behavior is the Izhikevich model[64]. This model is not only biologically plausible, it is also similar to the first model of this kind. A model inspired by the flow of current in biological nerves introduced in 1952 in [65] called Hodgkin-Huxley model, but also it is computationally as efficient as an integrate-and-fire model. The Izhikevich model is also capable of simulating large-scale spiking neurons in real-time [66]. Izhikevich model can be described by two differential equations:

$$v' = 0.04v^2 + 5v + 140 - u + I$$
$$u' = a(bv - u),$$

and a function, that restarts neuron internal state after a spike is generated:

$$\text{if } v \geq 30 \text{ m}V \text{ , then } \begin{cases} v \leftarrow c \\ u \leftarrow u + d. \end{cases}$$

This kind of neuron is using an internal membrane potential to accumulate the signal and decide whether the potential crosses the threshold and a spike

should be emitted and sent out via an output connection (this is called an axon in biological terminology). Arrival of spikes on the input connections of the neuron increases its potential. In the absence of any input signal, the potential is decaying [67].

4.2 Spiking neural networks

Spiking Neural Networks (SNNs) are a promising new paradigm for efficient processing of data streams. SNNs have inspired the design and can take advantage of the emerging class of neuromorphic processors like Intel Loihi. These novel hardware architectures expose a variety of constraints that affect firmware, compiler, and algorithm development alike. To enable the fast and flexible development of SNN algorithms on Loihi and similar neuromorphic processors, there are tools that make the transition from a simulator on a traditional CPU to an actual spiking implementation on a piece of neuromorphic hardware easier. One of the main reasons one may find spiking neural networks interesting is their order of magnitude lower energy cost per inference, which important for both edge devices with low power supply and large simulations which nowadays require enormous energy supply [9].

4.3 Neural engineering framework

Neural Engineering framework describes three principles that govern the construction of models of neurobiological systems. It unifies control (and dynamic systems) theory with biologically plausible spiking models of neural function. It provides a general way to generate circuits that have analytically determined synaptic weights that provide a desired functionality. Promotes the formulation of specific hypotheses about circuit function and about key design constraints.

The neural engineering framework presented in [39] consists of three mathematical principles used to describe neural computation. The NEF is most commonly applied to building recurrent spiking neural networks, but also applies to nonspiking and feedforward networks. Its primary use case is providing an efficient way of engineering spiking neural models, and programming neuromorphic hardware, it enables the implementation of linear dynamical systems, but it also extends to nonlinear dynamical systems as well. The framework is consistent with the Nengo simulator which will also be presented in this section. Three basic principles will be introduced in this section as described in [39], the most important of them being the third one which closely relates to the dynamic systems introduced in the second chapter.

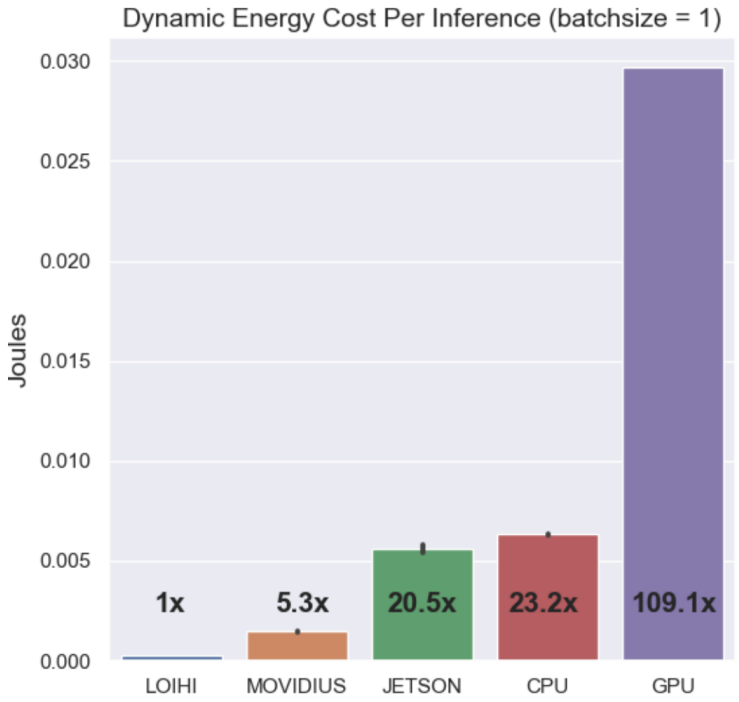


Figure 4.1: Dynamic energy cost per inference across hardware devices. Movidius and Jetson are edge computing devices, which are not neuromorphic chips [9].

4.3.1 Principle 1 - Representation

Simply put, it says that neural representations are defined by the combination of nonlinear encoding and weighted linear decoding. It is possible can formalize this principle in the following way. Let $\mathbf{x}(t) \in \mathbb{R}^q$ denote a q -dimensional time-varying signal that is to be represented by a population of n spiking neurons. By formally describing this representation, it is feasible to define a nonlinear encoding and a linear decoding that together determine how neural activity relates to the represented vector. First, encoders need to be chosen

$$E = [\mathbf{e}_1, \ldots, \mathbf{e}_n]^\top \in \mathbb{R}^{n \times q},$$

gains $\alpha_i > 0$, and biases

$$\beta_i, i = 1 \ldots n,$$

as parameters for the encoding, which map $\mathbf{x}(t)$ to neural activities. These parameters are fit from neuroanatomical data, these include but are not limited to neuron tuning curves, preferred directions, firing rates, sparsity) or randomly sampled from distributions constrained by the domain of $\mathbf{x}(t)$ and the

dynamic range of the neuron models. In either case, the encoding is defined by

$$a_i^{\mathrm{x}}(t) = G_i \left[\alpha_i \left\langle \mathbf{e}_i, \mathbf{x}(t) \right\rangle + \beta_i\right], \quad i = 1 \ldots n,$$

where $a_i^{\mathrm{x}}(t)$ is the neural activity generated by the i th neuron encoding the vector $\mathbf{x}(t)$ at time t, $\langle \cdot, \cdot \rangle$ denotes a dot product, and $G_i[\cdot]$ is the nonlinear dynamical system for a single neuron (e.g., a leaky integrate-and-fire, neuron, a conductance-based neuron). Then

$$a_i^{\mathrm{x}}(t) = \sum_m \delta\left(t - t_{i,m}\right),$$

where $\delta(\cdot)$ is the Dirac delta and $\{t_{i,m}\}$ is the sequence of spike times generated.

Having defined an encoding, it is possible to introduce a postsynaptic filter $h(t)$, which acts as the synapse model by capturing the dynamics of a receiving neuron's synapse. In particular, this filter models the postsynaptic current (PSC) triggered by action potentials arriving at the synaptic cleft. It is possible to fix

$$h(t) = \frac{1}{\tau} e^{-\frac{i}{t}},$$

to be an exponentially decaying PSC with time constant τ, which is equivalent (in the Laplace domain) to the canonical first-order low-pass filter (also known as a leaky integrator). This is the conventional choice of synapse in the NEF, since it strikes a convenient balance between mathematical simplicity and biological plausibility [39].

Now it is possible to characterize the decoding of the neural response, which determines the information extracted from the neural activities encoding $\mathbf{x}(t)$. Let $D = [\mathbf{d}_1, \ldots, \mathbf{d}_n]^\top \in \mathbb{R}^{n \times q}$ be the decoding matrix that decodes $\mathbf{x}(t)$ from the population's activities $(a_i^{\mathrm{x}}(t))$ at time t. This linear decoding is described by

$$(\mathbf{x} * h)(t) \approx \sum_{i=1}^{n} \left(a_i^{\mathbf{x}} * h\right)(t) \mathbf{d}_i,$$

where $*$ is the convolution operator that is used to apply the synaptic filter. The equation takes a linear combination of the filtered activities to recover a filtered version of the encoded signal. To complete the characterization of the neural representation, they solve for the optimal linear decoders D. This optimization is identical for principles 1 and 2, as discussed below [10].

4.3.2 Principle 2 - Transformation

Simple explanation of the Principle 2 is that transformations of neural representations are functions of variables that are represented by neural populations. Transformations are determined using an alternately weighted linear decoding. The second principle of the NEF addresses the issue of computing transformations of the represented signal. The encoding remains defined

in the same way as in the Principle 1. However, to decode some desired function of $\mathbf{x}(t), \mathbf{f} : \mathbb{R}^q \to \mathbb{R}^q$,[2] by applying an alternate matrix of decoders $D^f = \left[\mathbf{d}_1^f, \ldots, \mathbf{d}_n^f\right]^\top \in \mathbb{R}^{n \times q}$ to the same activities:

$$(\mathbf{f}(\mathbf{x}) * h)(t) \approx \sum_{i=1}^{n} \left(a_i^{\mathbf{x}} * h\right)(t) \mathbf{d}_i^f.$$

For both principles 1 and 2 , the optimization for D^f over the domain of the signal, $S = \{\mathbf{x}(t) : t \geq 0\}$, which is typically the unit q -ball $\{\mathbf{v} \in \mathbb{R}^q : \|\mathbf{v}\|_2 \leq 1\}$ or the unit q -cube $[-1, 1]^q$. To determine these decoders, first let $r_i(\mathbf{v})$ be the limiting average firing rate of the i the neuron under the constant input $\mathbf{v} \in S$:

$$r_i(\mathbf{v}) = \lim_{t \to \infty} \frac{1}{t} \int_0^t a_i^{\mathbf{v}}\left(t'\right) dt'.$$

To account for the variance introduced by neural spiking and other sources of uncertainty, white noise term is introduced $\eta \sim \mathcal{N}\left(0, \sigma^2\right)$ The optimality criterion for D^f is therefore

$$D^{\mathrm{f}} = \underset{D \in \mathbb{R}^{xx}}{\arg\min} \int_S \mid \mathbf{f}(\mathbf{v}) - \sum_{i=1}^{n} \left(r_i(\mathbf{v}) + \eta\right) \mathbf{d}_i \|^2 d^q \mathbf{v}.$$

Note that this optimization depends on only $r_i(\mathbf{v})$ for $\mathbf{v} \in S$, as opposed to depending on the signal $\mathbf{x}(t)$. In other words, the optimization is determined strictly by the distribution of the signal, and not according to its particular dynamics. Furthermore, this is a convex optimization problem, which may be solved by uniformly sampling S and applying a standard regularized least-squares solver to the sampled data. Monte Carlo sampling introduces $\mathcal{O}\left(\frac{1}{\sqrt{m}}\right)$ error into the integral where m is the number of samples, but this can be improved to $\widetilde{\mathcal{O}}\left(\frac{1}{m}\right)$ - effectively squaring $m-$ by the use of quasi-Monte Carlo methods [39]. The accuracy of this approach relies on $r_i(\mathbf{v})$ being a suitable proxy for $a_i^{\mathrm{x}}(t)$ whenever $\mathbf{x}(t) = \mathbf{v}$. This zeroth-order approximation clearly holds in the steady state for constant $\mathbf{x}(t)$ and turns out to be ideal in practice for low frequency $\mathbf{x}(t)$, and likewise for $h(t)$ that filter out high frequencies (when the synaptic time-constant τ is large).

The derivation can be done in the following way with a connection weight matrix between layers to implicitly compute the desired function f(x) within the latent vector space $\mathbb{R}^q$. Specifically, the weight matrix $W = [\omega_{ij}] \in \mathbb{R}^{n \times n}$, which maps activities from the j the presynaptic neuron to the i the postsynaptic neuron, is given by

$$\omega_{ij} = \left\langle \mathbf{e}_i, \mathbf{d}_j^f \right\rangle.$$

Consequently, the matrices E and D^f are a low-rank factorization of W. In other words, the process of decoding and then encoding is equivalent to taking the dot product of the full-rank weight matrix W with the neural activities.

This factorization has important consequences for the computational efficiency of neural simulations. The crucial difference between the factorized and nonfactorized forms is that it takes $\mathcal{O}(qn)$ operations per simulation time step to implement this dot product [10].

4.3.3 Principle 3 - Dynamics

This principle says that neural dynamics are characterized by considering neural representations as control theoretic state variables. Thus, the dynamics of neurobiological systems can be analyzed using control theory.

The third principle is a method of harnessing the dynamics of the synapse model for network-level information processing. NEF dynamics for the neural implementation of continuous linear time-invariant (LTI) systems have to following form:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t)$$
$$\mathbf{y}(t) = C\mathbf{x}(t) + D\mathbf{u}(t),$$

where the time-varying signal $\mathbf{x}(t)$ represents the system state and $\dot{\mathbf{x}}(t)$ represents the time derivative. We also need to to account for the output $\mathbf{y}(t)$ and the input $\mathbf{u}(t)$. These together with the time-invariant matrices (A, B, C, D) fully describe the system [33]. This form of an LTI system is also called the state-space model, but there are many other equivalent forms.

We also need a nonlinearity and for the LTI systems, the dynamical primitive that is, the source of the dynamics - is the integrator. However, the dynamical primitive available is the leaky integrator, given by the canonical first-order low-pass filter modeling the synapse

$$h(t) = \frac{1}{\tau} e^{-\frac{t}{t}} = \mathcal{L}^{-1}\left\{\frac{1}{\tau s + 1}\right\},$$

where $\mathcal{L}^{-1}(\cdot\}$ denotes the inverse Laplace transform. To be more precise, their approach is to represent the state vector $\mathbf{x}(t)$ in a population of spiking neurons such that this vector is obtained by filtering some linearly decoded spike trains with a leaky integrator. Thus, the goal of principle 3 is to determine the transformations required to implement the equation

$$r_i(\mathbf{v}) = \lim_{t\to\infty} \frac{1}{t} \int_0^t a_i^{\mathbf{v}}\left(t'\right) dt', \tag{4.1}$$

given that $\mathbf{x}(t)$ is obtained by some convolution with a leaky integrator rather than the perfect integrator.

Principle 3 states that to implement the equation in a population of neurons that represents $\mathbf{x}(t)$, tocompensate for the effect of "replacing" the integrator with a leaky integrator , that is, by driving the synapse with $\tau\dot{\mathbf{x}}(t)+\mathbf{x}(t)$ instead of only $\dot{\mathbf{x}}(t)$. This compensation is achieved as follows: implement the recurrent transformation $(\tau A + I)\mathbf{x}(t)$ and the input transformation $\tau B\mathbf{u}(t)$,

but use the same output transformation $\mathrm{Cx}(t)$, and the same pass-through transformation $D\mathbf{u}(t)$. Specifically, this may be implemented in a spiking dynamical network by representing $\mathbf{x}(t)$ via principle 1 and then using principle 2 to decode the needed transformations.

The correctness of this "mapping" procedure relies on three assumptions:

1. Usage the synapse model equation described in principle 2
2. The network is simulated in continuous time (or the discrete time step is sufficiently small)
3. All the necessary representations and transformations are sufficiently accurate such that the approximation error $\mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$ is negligible. In other words, assuming n is sufficiently large, the architecture using leaky integrator is equivalent to the architecture with the ideal integrator but using the leaky integrator instead of an integrator as the dynamical primitive [39, 10].

Consequently, both systems compute the exact same signals $\mathbf{x}(t)$ and $\mathbf{y}(t)$. The work [10] provides a novel proof of this equivalence. This framework can be extended to remove the first and second assumptions.

Principle 3 is useful for accurately implementing a wide class of dynamical systems (e.g., integrators, oscillators, attractor networks) to solve specific problems that frequently arise in neural modeling. Furthermore, the class of state-space models is isomorphic to the class of all finite-dimensional causal linear filters or, equivalently, all rational (finiteorder) proper transfer functions, which is a large and useful class of dynamical systems employed widely in control applications [33]. Given the ability of Principle 2 to compute nonlinear functions, Principle 3 also naturally generalizes to nonlinear dynamical systems [10].

4.4 Spiking delay network

A fundamental dynamical system is the continuous-time delay line of θ seconds, expressed as

$$y(t) = (u * \delta_{-\theta})(t) = u(t - \theta), \quad \theta > 0, \tag{4.2}$$

where $\delta_{-\theta}$ denotes a Dirac delta function shifted backward in time by θ. This system takes a time-varying scalar signal, $u(t)$, and outputs a purely delayed version, $u(t - \theta)$. The task of computing this function both accurately and efficiently in a biologically plausible spiking dynamical network is a significant theoretical challenge that was unsolved until the recent publication of [3] which shows that the continuous-time delay is worthy of detailed consideration for several reasons.

- It is nontrivial to implement using continuous-time spiking dynamical primitives. Equation 4.2 requires maintaining a rolling window of length θ (the history of $u(t)$, going θ seconds back in time). Thus, computing a delay of θ seconds is just as hard as computing every delay of length θ' for all $0 \leq \theta' \leq \theta$. Since any finite interval of $\mathbb{R}$ contains an uncountably infinite number of points, an exact solution for arbitrary $u(t)$ requires maintaining an uncountably infinite amount of information in memory.
- Moreover, the delay provides one with a window of input history from which to compute arbitrary nonlinear functions across time.
- Delays introduce a rich set of interesting dynamics into large-scale neural models. [68].
- Delay lines can be coupled with a single nonlinearity to construct a network displaying many of the same benefits as reservoir computing [69, 41].
- Examining the specific case of continuous-time delay introduces several methods and concepts that can be used to extend the NEF [10].

4.5 Neuromorphic hardware

The human brain has been estimated to consume as little as 10–20 fJ. On average 10 per synaptic event [70, 71], totalling approximately 20 W across its approximately 10^{15} neurons and 10^{14} synapses [72]. After more than 30 years of work [73], the state-of-the-art in neuromorphic computing is close to this level of performance, with Braindrop, Loihi, and TrueNorth soon to be commercially available [74, 75]. Such potential energy savings seem very promising. Having this kind of energy efficiency is the prerequisite to reach the exascale level of computation achieved by the human brain without requiring an insane amount of energy to run it [76, 77].

4.5.1 SpiNNaker

The SpiNNaker is a processor that is optimized for simulating of neural networks. It is implemented using many ARM cores as an integrated system architecture optimized for communication and fast memory access. To take advantage of the asynchronous subcomputations of biological neurons, each core simulates neurons independently and communicates via a lightweight, spike-optimized asynchronous communication protocol. Neurons are simulated for a certain timestep, and then activity patterns exchanged between cores, on the assumption that time models itself, for example, an exchange of activities every millisecond is assumed to represent biological real time. This way they are able to perform an energy efficient simulation of neural network

models in real time, with SpiNNaker, significantly outperforming conventional high performance computing. The first generation of SpiNNaker has been designed by the University of Manchester and is currently operational at its intended maximum system size, which consists of 1 Million ARM processors, as well as in the form of smaller boards for IoT and mobile applications. One Million cores is enough to create models that represent roughly one percent of the human brain capacity. Technische Universitt Dresden and the University of Manchester have been jointly developing the next generation SpiNNaker2 system in the framework of the EU flagship Human Brain Project. They aim for a tenfold increase in the number of cores while keeping the same power consumption of the whole system that means 10 times better power efficiency. They expect to create a system with 50 times bigger capacity [78].

4.5.2 Braindrop

Braindrop claims to be the first neuromorphic system designed to be programmed at a high level of abstraction. Previous neuromorphic systems were programmed at the neurosynaptic level and required expert knowledge of the hardware to use. Braindrop's computations are specified as coupled nonlinear dynamical systems and synthesized on the hardware by an automated procedure, which means that a clean abstraction is presented to the user. Fabricated in a 28-nm FDSOI process, Braindrop integrates 4096 neurons in 0.65 square milimeters . It includes two innovations, the first is sparse encoding through analog spatial convolution and weighted spike-rate summation trough digital accumulative thinning, that leads to a significant reduction in digital traffic and reduction of the energy Braindrop consumes per synaptic operation to 381 fJ for typical network configurations [79].

4.5.3 Intel Loihi

Intel's Loihi research chip is an asynchronous, compute-in-memory neuromorphic processor optimized for the execution of Spiking Neural Networks. Fabricated in Intel's standard 14 nm CMOS process, Loihi consists of 128 neurocores, each of which supports up to 1024 neurons. Three embedded x86 processors per chip enable off-chip data encoding and interaction with the neurocores. Loihi's asynchronous network-on-chip communication infrastructure allows it to be seamlessly scaled up to various form factors, ranging from 2-chip USB devices, Kapoho Bay to the 768-chip rack Pohoiki Springs. [80]. Spiking neurons on Loihi are stateful dynamical systems that support a wide range of features. These include synaptic plasticity, variable numeric precision of synaptic weights up to 9 bits, multicompartment models, threshold adaptation for homeostasis, and configurable synaptic, axon and refractory time constants. To support hierarchical and repeated connectivity as in CNNs, Loihi provides a connection-sharing mechanism. The dynamical equa-

tions that underlie the Loihi neuron model are approximated in discrete time. The transition between algorithmic time steps is not driven by a fixed global clock but mediated through a barrier synchronization protocol between all participating neurocores that is very different from what we are used to from traditional computing. As part of this protocol, neurocores signal the completion of the workload for the current time step to their neighbors, resulting in a workload-dependent duration of each time step. Neuron updates contribute on the order of microseconds to the duration of each time step. Spike traffic typically dominates the overall time step duration. This is the reason spatially and temporally sparsely communicating SNNs promise the greatest level of efficiency on such architectures [11].

4.5.4 Other hardware platforms and commercial solutions

There are other hardware platforms available which are not compatible with Nengo at the moment or the information available on them is limited. They will be listed and briefly described for completeness.

- GrAI Matter Labs has their own chip and toolset built around it. They have a few different approaches to increase sparsity as the basic idea of spiking neural networks is that changes in the real world don't happen everywhere, or all at once.
- Synsense technology has the DYNAP-SE2, each Chip features 1k redesigned adaptive exponential integrate-and-fire analog ultra low-power spiking neurons and 65k enhanced synapses with configurable delay, weight and short-term plasticity.
- Applied Brain Research offers their own implementation of NER spiking neural networks as a FPGA.
- BrainChip's Akida integrated circuit technology is an ultra-low power, high performance, minimum memory footprint, event domain neural processor targeting Edge AI applications. In addition, because the architecture is based upon an event domain processor, leveraging fundamental principles from biological SNN the processor supports incremental learning
- Inivation is selling their Extreme Machine Vision High-performance neuromorphic vision systems for demanding real-time applications.

4.6 Implementation of SNN

The Spiking Neural Network (SNN) is a promising approach to energy efficient computing, since its activation levels are quantized into temporally

sparse, one-bit values called spikes, which additionally converts the sum over weight-activity products into a simple addition of weights (one weight for each spike). However, the goal of maintaining a competitive accuracy when moving from a nonspiking model to a spiking one has for a long time been unsolved, mainly because spikes have only a single bit of precision. The LMU can be implemented on a spiking neural network [3], and on neuromorphic hardware [75], while consuming several orders less energy than traditional computing architectures [9, 10]. By reducing the amount of communication and converting weight multiplies into additions, spikes can trade precision for energy efficiency on neuromorphic hardware [9]. Moreover, this can be accomplished while preserving the optimizations that are found by deep learning [56].

4.6.1 Neural precision

The dynamical system for the memory cell can be implemented by mapping each state variable onto the postsynaptic currents of d individual populations of p Poisson spiking neurons with fixed heterogeneous tuning curves [3]. Considering the error between the ideal input to the classical neuron representing some dimension, versus the weighted summation of spike events representing the same dimension. The following was proved in [10] about scaling of feed-forward precision in the Neural Engineering Framework . Let $\mu(t) = \sum_{i=1}^{n} (a_i * h)(t)\mathbf{d}_i^{\mathrm{f}} - \sum_{i=1}^{n} (r_i(\mathbf{x}) * h)(t)\mathbf{d}_i^{\mathrm{f}}$ be the spiking noise. If the ISIPs, g_i, are uniformly and independently distributed, then:

1. $\mathbb{E}[\mu(t)] = 0$; hence the expected spike noise is exactly zero at all times (no systematic bias).

2. $\sqrt{\mathbb{W}[\mu(t)]} = \mathcal{O}\left((\tau\sqrt{n})^{-1}\right)$ is the standard error, or root-mean-squared error; hence the precision scales as $\mathcal{O}(\tau\sqrt{n})$ at all times.

Simply put, this error has a variance of $\mathcal{O}(1/p)$. By the variance sum law, repeating this for d independent populations yields an overall RMSE of $\mathcal{O}(\sqrt{d/p})$. Letting $m = pd$ be the total number of neurons, now it is clear that the error scales as $\mathcal{O}(d/\sqrt{m})$. It is now clear that m is a hyperparameter that trades precision for energy efficiency, while scaling to the original network in the limit of large d. [10].

4.6.2 Neuromorphic implementation of LMU

The LMU spiking neural network has been implemented on neuromorphic hardware including Braindrop [75] and Loihi [10]. Each population is coupled to one another to implement equation 2.4 by converting the postsynaptic filters into integrators [3]. This results in a specific connectivity pattern, shown in Figure 4.6.2 that exploits the alternating structure of equation 2.5. An ideal implementation of this system requires m nonlinearities, $\mathcal{O}(m)$ additions, and

d state variables. Spiking neurons may also be used to implement the hidden state of the LMU by nonlinearly encoding the memory vector [3]. Since this scales linearly in time and memory with square root precision, the LMU offers a promising architecture for low-power RNNs.

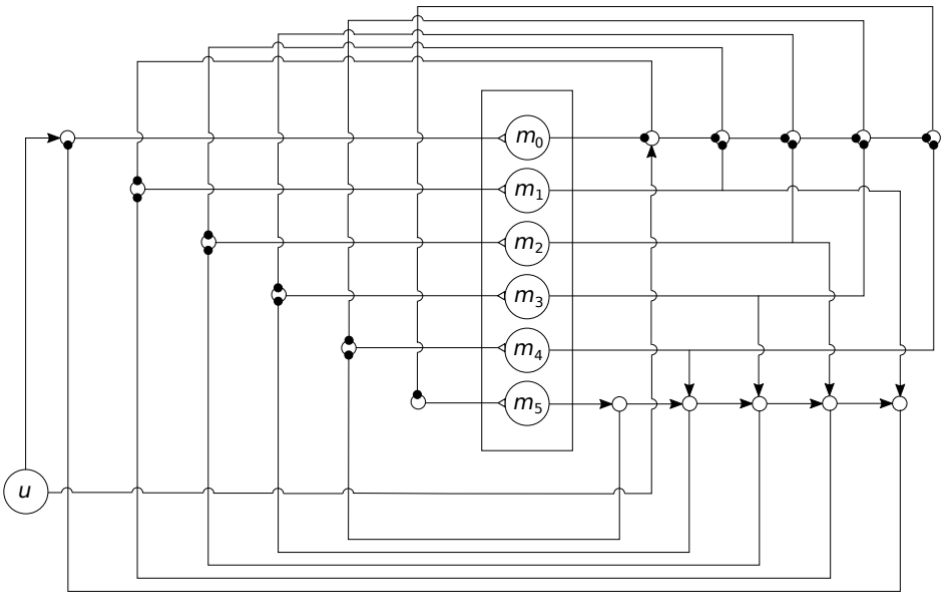


Figure 4.2: Connection structure as presented in [4] where ($d = 6$) and is adapted from an earlier work [10]. Forward arrow heads indicate addition, circular heads indicate subtraction. The i^{th} state variable continuously integrates its input with a gain of $(2i + 1)\theta^{-1}$ [4].

4.7 Adjustable power efficiency

A useful property of SNNs is their ability to trade off accuracy with computational cost. As the classification result is obtained by accumulating spikes at the output layer across a certain period of time, one may shorten the run time of the network to reduce the inference delay and energy cost at the potential risk of misclassifying some samples [10].

4.8 Software tools

There is many different software tools which are able to simulate spiking networks. For the experiment with spoken digit recognition, NengoDL was used.

Multiple other tools that were considered will however be covered in this section as finding comprehensive up-to-date information on these tools is rather difficult.

4.8.1 Nengo

Nengo is a simulator for Nengo models. That means it takes a Nengo network as input and allows the user to simulate that network using some underlying computational framework, in this case it is TensorFlow. The nice thing about Nengo is that it supports some of the hardware platforms mentioned earlier in this chapter, specifically Intel Loihi and SpiNNaker. In practice, what that means is that the code for constructing a Nengo model is the same as it would be for the standard Nengo simulator. All that changes is that a different simulator class is used to execute the model. The experiment conducted on classifying the spoken digits was replicated using Nengo and achieved the same results during the simulation of spiking hardware on classical hardware - even though no neuromorphic hardware was commercially available at the time of writing of this work, it should not be necessary to modify the code that describes the model. To change the backend to Loihi, just switch the Nengo backend to, for example, an Intel Loihi implementation and we should be good to go [56].

4.8.2 NengoDL

NengoDL is a software framework designed to combine the strengths of neuromorphic modelling and deep learning. NengoDL allows users to construct biologically detailed neural models, intermix those models with deep learning elements (such as convolutional networks), and then efficiently simulate those models in an easy-to-use, unified framework. In addition, NengoDL allows users to apply deep learning training methods to optimize the parameters of biological neural models. However, NengoDL adds a number of unique features, such as:

- Optimizing the parameters of a model through deep learning training methods (using the Keras API)
- Faster simulation speed on both CPU and GPU
- Automatic conversion from Keras model to Nengo networks
- Inserting TensorFlow code (individual functions or whole network architectures) directly into a Nengo model

This is the simulator that was used to implement the Spoken Digit Recognition experiment. Running the code on the actual neuromorphic hardware should be as easy as changing one line, defining which simulator will be used

for running the network and possibly converting the network using the bundled convertor if Keras layers were used, which was not the case here. Instead of invoking the simulator like in the example below, which is used to simulate a spiking network on classical hardware.

```
import nengo_dl

with nengo_dl.Simulator(
        net, minibatch_size=100, unroll_simulation=16) as sim:
    sim.compile(
        loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
        optimizer=tf.optimizers.Adam(),
        metrics=["accuracy"],
    )
    sim.evaluate(X_test, y_test)
    sim.fit(X_train, y_train, epochs=25)
    sim.save_params("./weights")
    sim.evaluate(X_test, y_test"]
```

After training the network using the NengoDL simulator and possibly converting the network using the bundled convertor, we would replace the line defining the simulator to be used from `nengo_dl.Simulator` to something like this

```
import nengo_loihi
with nengo_loihi.Simulator(net, model=model) as sim:
```

Some additional finetuning may be however, required to achieve optimal performance on neuromorphic hardware. Earlier work from the same author [81] suggests that we can first train the model on classical hardware and then transfer the weight to the spiking implementation for inference. The NengoDL framework provides methods for converting a trained Keras model into a Nengo one while keeping the trained weights. The goal of NengoDL is to provide a tool that unites deep learning and neuromorphic modelling methods [56].

4.8.3 KerasSpiking

KerasSpiking is a companion project to NengoDL that has a more minimal feature set but integrates even more transparently with the Keras API. KerasSpiking provides tools for training and running spiking neural networks directly within the Keras framework. The main feature is `SpikingActivation`, which can be used to transform any activation function into a spiking equivalent. For example, translating a non-spiking feedforward model into a a model with spiking activation function by using spiking activation layer. Models with spiking activation layers can be optimized and evaluated in the same way as

any other Keras model. The only thing needed is adding the time-dimension input as all spiking neurons need to will automatically take advantage of KerasSpiking's spiking aware training: using the spiking activations on the forward pass and the non-spiking (differentiable) activation function on the backwards pass[56].

4.8.3.1 Neuromorphic chip energy use estimation

When we have successfully developed a model using Nengo or using Keras and successfully converted it to a Nengo model. We may be interested in the estimated energy use of the current implementation on a real neuromorphic device like Intel Loihi. Estimation of the energy use of the model which provides useful information about total energy per inference on both CPU and neuromorphic chips is available as `keras_spiking.ModelEnergy`.

4.8.4 NxTF

NxTF is a programmatic interface based on Keras that is designed to map spiking neural networks onto Intel's Loihi neuromorphic architecture. It is based on a collaboration between Intel and researchers from the neuroinformatics field. As the team explains, NxTF trades generality for an application focus towards deep SNNs. The objective is achieved by inheriting from the Keras Model and Layer interface and providing a specialized DNN compiler. They add that to compress large DNNs with Loihi, the compiler exploits the regular structure of typically DNN topologies like convolutional neural networks in combination with Loihi's ability to share connective states.

The NkSDK software stack and workflow to configure a deep SNN on Loihi is roughly the following. Starting from a trained or converted SNN, the user defines a model in Python using the Keras-derived NxTF interface. The network is then partitioned and mapped onto Loihi by the DNN compiler via the register-level NxCore API. The NxDriver and NxRuntime components are responsible for the interaction and execution of the SNN with Loihi. See the figures 4.8.4 and 4.8.4 taken from [**?**].

4.8.5 SNN toolbox

This toolbox automates the conversion of a pretrained analog to spiking neural networks (ANN to SNN), and provides tools for testing the SNNs in spiking neuron simulators or neuromorphic hardware. It supports both PyTorch and Keras as backends used for development and training. The supported hardware includes Intel Loihi, and SpiNNaker [82].

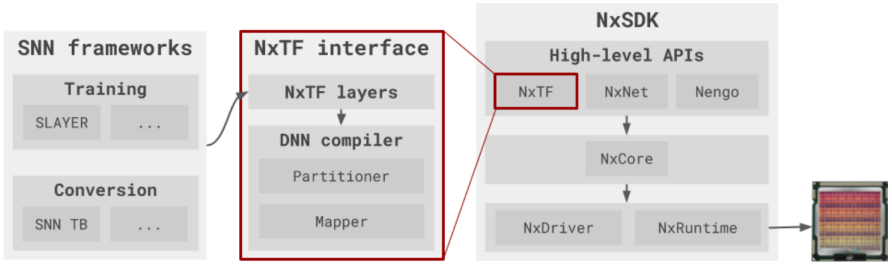


Figure 4.3: NkSDK software stack and workflow to configure a deep SNN on Loihi [11].

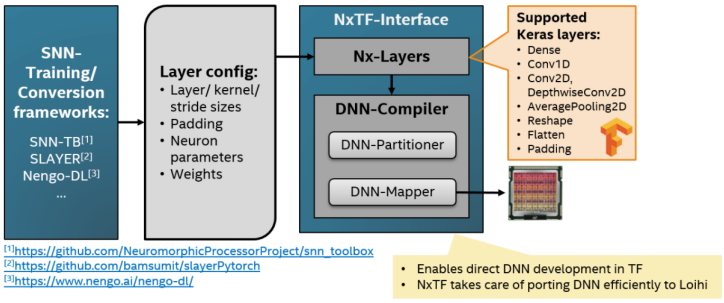


Figure 4.4: NxTF layer configuration and supported keras layers [11].

Chapter 5

Discussion

5.1 Conclusion

In this thesis, limitations of the basic RNN were discussed architecture as well as the attempts to avoid these limitations using the LSTM and GRU architectures including but not limited to the problem of vanishing/exploding gradient. Before diving into the novel approaches, a few topics from complex analysis and control theory were briefly introduced to help the formalization of projecting arbitrary functions onto the space of orthogonal polynomials. After that, novel approach called Legendre Memory Unit is explained, which is based on projections into a space of a lower dimension of orthogonal polynomials. We discuss a recent work which introduced the HiPPO framework, that shows that the problem of maintaining memory representations of sequential data can be tackled by specifying and solving continuous-time formalisms. Finally, we discuss how the HiPPO framework based recurrent neural networks generalize the previously introduced architecture LMU and explain the mechanisms used in GRU and LSTM as using 1-dimensional projections.

In the practical part of this thesis, we then use these novel architectures in the experiments where we compare them to the typically used RNN architectures, LSTM and GRU. We introduce a version of Sequential Spoken Digit Classification where we clearly hit the limits of the LSTM and GRU architectures as they fail to learn anything while HiPPO based models perform very well at the task. We then move to a bigger dataset concerned with environmental sound classification where we confirm that this approach works also on a similar dataset bigger in both complexity and size. We continue with experiments in another domain, specifically in the field of natural language processing (NLP). We succesfully test HiPPO based architectures on tasks where LSTM and GRU perform fairly well and determine that they usually perform on par with these in some cases even better while enjoying better theoretical properties. We confirmed that both LMU(HiPPO-LegT) and HiPPO-LegS can indeed be treated as a drop in replacement for LSTM and GRU in many

cases. It should be noted, that at least in the NLP domain, currently one would probably still be better off using a transformer based model for those tasks. Those models however suffer from other limitations compared to RNNs.

In the last part of this work, we show how the control theory used to derive the LMU unit relates to neuroscience and the spiking neural networks. We implement the spoken digit recognition experiment using NengoDL simulator of spiking neural networks. Brief explanation of the essential concepts in the field of neuromorphic computation is given. In context of this thesis we explain how being able to implement recurrent networks on neuromorphic hardware is beneficial because of considerable energy consumption reduction. Additionally, we showcase some software tools which can be used for the development of such networks. We also briefly go over hardware platforms that could be used to run such networks as they should be soon commercially available. The field of neuromorphic computation is going trough rapid development in the recent years, even thought it may not live up to the expectations of the business oriented users in the next few years, the research in this field seems to be laying the foundation for exascale computation at a reasonable energy cost. There is already a few neuromorphic solutions available on the market and the toolkit around those platforms is being very actively developed.

5.2 Future work

5.2.1 Parallel training of LMU

The training of both LMU(LegT) and HiPPO RNNs is not very fast, another work [41] is concerned with the possibility of training such RNNs in a parallel manner. They suggest that most weights in the LMU can be kept constant without negatively impacting the performance of the network. Surprisingly, this includes the recurrent connections in the LMU. Constant recurrent weights can be replaced by a set of static feedforward Finite Impulse Response filters arranged in a basis transformation matrix H. This facilitates parallel training, leading to significant speed-ups. The basis transformation matrix H can be interpreted as a discrete function basis. They also show that this technique is applicable to arbitrary polynomial bases, but there usually exists no closed-form equation describing the state transition matrix.

5.2.2 Discrete Function Bases and Temporal Convolutions

Discrete function bases are further explored in the work [83] with a particular focus on the discrete basis derived from the Delay Network proposed in [10]. They characterize the performance of these bases in a delay computation task and attempt to formulate them as fixed temporal convolutions in neural networks. Main Results of the work include a numerically stable algorithm for constructing a matrix of Discrete Legendre Orthogonal Polynomials

in $O(qN)$ The work also demonstrated that the Delay Network can be used to form a discrete function basis with a basis transformation matrix H in $O(qN)$. They also prove that linear-time invariant systems similar to the DN can be constructed using many discrete function bases, however, the DN system is superior in terms of having a finite impulse response. From the application focused point of view, the interesting part of this work is neural network experiments, which suggest that fixed temporal convolutions can outperform learned convolutions. Networks using fixed temporal convolutions are conceptually simple and yield state-of-the-art results in tasks such as psMNIST [83]. Another work is concerned with Continuous Kernel Convolution For Sequential Data by formulating convolutional kernels in CNNs as continuous functions. The resulting Continuous Kernel Convolution (CKConv) allows them to model arbitrarily long sequences in a parallel manner, within a single operation, and without relying on any form of recurrence [84].

5.2.3 Simple derivation of DN used in LMU

The Delay Network, originally proposed in [10], is a recurrent neural network capable of delaying an input signal $u(t)$ by θ seconds as discussed in the previous chapters. The impulse response of the linear time-invariant system underlying the delay network traces out the shifted Legendre polynomials. For the rest of this section, it will be referred to as the Delay Network, and to the LTI system underlying the DN as the DN system. The DN has been derived from the Padé approximants of a Laplace domain delay $e^{-\theta s}$ and a subsequent conditioning coordinate transformation. From this perspective, the relationship to the Legendre polynomials is rather surprising. And the HiPPO framework has proposed LTI systems similar to the DN system with other polynomial bases and various window functions. These systems are derived in the opposite direction of the original approach in [4]. Given a polynomial basis and a window function, HiPPO framework is able to construct an LTI system that realizes this basis with the desired weighting applied. Another similar approach exists, which has been developed independently, but it differs mainly in the way of presentation. Its main goal is to provide a simple derivation assuming a minimal mathematical background. In essence, they derive the DN system $\hat{\mathbf{A}}, \hat{\mathbf{B}}$ in two steps.

- They start with a LTI system $\overline{\mathbf{A}}, \overline{\mathbf{B}}$ that traces out the Legendre polynomials as its impulse response over the interval $[0, \theta]$.
- Then a matrix $\overline{\mathbf{\Gamma}}$ that decodes a delayed signal $u(t - \theta)$ from the state vector $\mathbf{m}(t)$ and re-encodes this delayed function in terms of the Legendre basis is derived.

This matrix $\overline{\mathbf{\Gamma}}$ is called the delay re-encoder. Subtracting $\overline{\mathbf{\Gamma}}$ from $\overline{\mathbf{A}}$ results in a dampened impulse response. The DN system is simply given as $\hat{\mathbf{A}} = \overline{\mathbf{A}} - \overline{\mathbf{\Gamma}}$ and $\hat{\mathbf{B}} = \overline{\mathbf{B}}$ [83].

Bibliography

[1] Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.*, volume 9, no. 8, Nov. 1997: p. 1735–1780, ISSN 0899-7667, doi: 10.1162/neco.1997.9.8.1735. Available from: https://doi.org/10.1162/neco.1997.9.8.1735

[2] Staudemeyer, R. C.; Morris, E. R. Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks. 2019, arXiv:1909.09586.

[3] Voelker, A.; Eliasmith, C. Improving Spiking Dynamical Networks: Accurate Delays, Higher-Order Synapses, and Time Cells. *Neural Computation*, volume 30, 03 2018: pp. 569–609, doi:10.1162/neco_a_01046.

[4] Aaron R. Voelker, C. E., Ivana Kajic. Legendre Memory Units: Continuous-Time Representation in Recurrent Neural Networks. https://papers.nips.cc/paper/2019/hash/952285b9b7e7a1be5aa7849f32ffff05-Abstract.html, (Accessed on 03/06/2021).

[5] Gu, A.; Dao, T.; et al. HiPPO: Recurrent Memory with Optimal Polynomial Projections. 2020, arXiv:2008.07669.

[6] Gu, A.; Dao, T.; et al. HiPPO: Recurrent Memory with Optimal Polynomial Projections. (Accessed on 03/06/2021). Available from: https://hazyresearch.stanford.edu/hippo

[7] Jain, R. Improving performance and inference on audio classification tasks using capsule networks. *CoRR*, volume abs/1902.05069, 2019, 1902.05069. Available from: http://arxiv.org/abs/1902.05069

[8] Petasis, G.; Cucchiarelli, A.; et al. Automatic adaptation of Proper Noun Dictionaries through cooperation of machine learning and probabilistic methods. 01 2000, pp. 128–135, doi:10.1145/345508.345563.

[9] Blouw, P.; Choo, X.; et al. Benchmarking Keyword Spotting Efficiency on Neuromorphic Hardware. In *Proceedings of the 7th Annual Neuro-Inspired Computational Elements Workshop*, NICE '19, New York, NY, USA: Association for Computing Machinery, 2019, ISBN 9781450361231, doi: 10.1145/3320288.3320304. Available from: https://doi.org/10.1145/3320288.3320304

[10] Voelker, Aaron Russell. *Dynamical Systems in Spiking Neuromorphic Hardware*. Dissertation thesis, 2019. Available from: http://hdl.handle.net/10012/14625

[11] Rueckauer, B.; Bybee, C.; et al. NxTF: An API and Compiler for Deep Spiking Neural Networks on Intel Loihi. *CoRR*, volume abs/2101.04261, 2021, 2101.04261. Available from: https://arxiv.org/abs/2101.04261

[12] Mehlig, B. Artificial Neural Networks. *CoRR*, volume abs/1901.05639, 2019, 1901.05639. Available from: http://arxiv.org/abs/1901.05639

[13] Dettmers, T. Deep Learning in a Nutshell: Sequence Learning — NVIDIA Developer Blog. https://developer.nvidia.com/blog/deep-learning-nutshell-sequence-learning/, (Accessed on 03/06/2021).

[14] Education, I. C. What are Recurrent Neural Networks? — IBM. https://www.ibm.com/cloud/learn/recurrent-neural-networks, (Accessed on 03/06/2021).

[15] Chandar, S.; Sankar, C.; et al. Towards Non-saturating Recurrent Units for Modelling Long-term Dependencies. 2019, arXiv:1902.06704.

[16] Werbos, P. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, volume 78, 11 1990: pp. 1550 – 1560, doi: 10.1109/5.58337.

[17] Williams, R. J.; Zipser, D. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*, volume 1, no. 2, 1989: pp. 270–280, doi:10.1162/neco.1989.1.2.270.

[18] Mozer, M. Induction of Multiscale Temporal Structure. In *NIPS*, 1991.

[19] Gers, F.; Schmidhuber, J.; et al. Learning to Forget: Continual Prediction with LSTM. *Neural computation*, volume 12, 10 2000: pp. 2451–71, doi: 10.1162/089976600300015015.

[20] Pérez-Ortiz, J.; Gers, F.; et al. Kalman filters improve LSTM network performance in problems unsolvable by traditional recurrent nets. *Neural networks : the official journal of the International Neural Network Society*, volume 16, 04 2003: pp. 241–50, doi:10.1016/S0893-6080(02)00219-8.

[21] Gers, F.; Schraudolph, N.; et al. Learning Precise Timing with LSTM Recurrent Networks. *Journal of Machine Learning Research*, volume 3, 01 2002: pp. 115–143, doi:10.1162/153244303768966139.

[22] Rumelhart, D. E.; McClelland, J. L. *Learning Internal Representations by Error Propagation*. 1987, pp. 318–362.

[23] Williams, R. J.; Zipser, D. *Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity*. USA: L. Erlbaum Associates Inc., 1995, ISBN 0805812598, p. 433–486.

[24] Bengio, Y.; Simard, P.; et al. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, volume 5, no. 2, 1994: pp. 157–166, doi:10.1109/72.279181.

[25] Hochreiter, S.; Schmidhuber, J. LSTM Can Solve Hard Long Time Lag Problems. In *Proceedings of the 9th International Conference on Neural Information Processing Systems*, NIPS'96, Cambridge, MA, USA: MIT Press, 1996, p. 473–479.

[26] Informatik, F.; Bengio, Y.; et al. Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies. *A Field Guide to Dynamical Recurrent Neural Networks*, 03 2003.

[27] Cho, K.; van Merriënboer, B.; et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. 06 2014, doi:10.3115/v1/D14-1179.

[28] Jozefowicz, R.; Zaremba, W.; et al. An Empirical Exploration of Recurrent Network Architectures. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, JMLR.org, 2015, p. 2342–2350.

[29] Royden, H. L. *Real analysis / H.L. Royden, Stanford University, P.M. Fitzpatrick, University of Maryland, College Park.* Pearson modern classic, New York, NY: Pearson, fourth edition [2018 reissue]. edition, 2018 - 2010, ISBN 9780134689494.

[30] Rudin, W. *Real and Complex Analysis, 3rd Ed.* USA: McGraw-Hill, Inc., 1987, ISBN 0070542341.

[31] Gupta, M. R. A Measure Theory Tutorial (Measure Theory for Dummies). May 2006. Available from: https://vannevar.ece.uw.edu/techsite/papers/documents/UWEETR-2006-0008.pdf

[32] Kolmogorov, A.; Fomin, S. *Introductory Real Analysis*. Dover Books on Mathematics, Dover Publications, 1975, ISBN 9780486612263. Available from: https://books.google.cz/books?id=z8IaHgZ9PwQC

[33] Brogan, W. L. *Modern Control Theory (3rd Ed.)*. USA: Prentice-Hall, Inc., 1991, ISBN 0135897637.

[34] Tao, T. FUNCTION SPACES. Mar 2008. Available from: https://terrytao.wordpress.com/

[35] Wang, A. LEBESGUE MEASURE AND L2 SPACE. Available from: http://www.math.uchicago.edu/~may/VIGRE/VIGRE2011/REUPapers/WangA.pdf

[36] Cobzas, S. *Lipschitz functions*. Cham, Switzerland: Springer, 2019, ISBN 978-3-030-16489-8.

[37] Heinonen, J.; Heinonen, A. *Lectures on Analysis on Metric Spaces*. Universitext (Berlin. Print), Springer New York, 2001, ISBN 9780387951041. Available from: https://books.google.cz/books?id=4slOGmnwlyEC

[38] Brezinski, C. *Computational aspects of linear control*. Dordrecht Boston: Kluwer Academic Publishers, 2002, ISBN 978-1-4613-0261-2.

[39] Eliasmith, C.; Anderson, C. H. *Neural Engineering (Computational Neuroscience Series): Computational, Representation, and Dynamics in Neurobiological Systems*. Cambridge, MA, USA: MIT Press, 2002, ISBN 0262050714.

[40] Voelker, A. R.; Eliasmith, C. Methods for applying the Neural Engineering Framework to neuromorphic hardware. 2017, arXiv:1708.08133.

[41] Chilkuri, N.; Eliasmith, C. Parallelizing Legendre Memory Unit Training. 2021, arXiv:2102.11417.

[42] Partington, J. Some frequency-domain approaches to the model reduction of delay systems. *Annual Reviews in Control*, volume 28, 01 2004: pp. 65–73, doi:10.1016/S1367-5788(04)90008-9.

[43] Padé, H. Sur la représentation approchée d'une fonction par des fractions rationnelles. *Annales scientifiques de l'École Normale Supérieure*, volume 3e série, 9, 1892: pp. 3–93, doi:10.24033/asens.378. Available from: www.numdam.org/item/ASENS_1892_3_9__S3_0/

[44] Legendre, A. *Recherches sur l'attraction des spheroïdes homogènes*. Mémoires de l'Académie Royale de Sciences, 1790. Available from: https://books.google.cz/books?id=xb8RywEACAAJ

[45] RODRIGUES. *DE L'ATTRACTION DES SPHIROIDES*. Place of publication not identified: HACHETTE LIVRE - BNF, 2018, ISBN 2019220997.

[46] Chen, R. T. Q.; Rubanova, Y.; et al. Neural Ordinary Differential Equations. 2019, arXiv:1806.07366.

[47] Virtanen, P.; Gommers, R.; et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, volume 17, 2020: pp. 261–272, doi:10.1038/s41592-019-0686-2.

[48] Zhang, J.; Lin, Y.; et al. Learning Long Term Dependencies via Fourier Recurrent Units. 2018, arXiv:1803.06585.

[49] Gencoglu, O.; van Gils, M. J.; et al. HARK Side of Deep Learning - From Grad Student Descent to Automated Machine Learning. *CoRR*, volume abs/1904.07633, 2019, 1904.07633. Available from: http://arxiv.org/abs/1904.07633

[50] Szegő, G.; G, S.; et al. *Orthogonal Polynomials*. American Math. Soc: Colloquium publ, American Mathematical Society, 1975, ISBN 9780821810231. Available from: https://books.google.cz/books?id=ZOhmnsXlcY0C

[51] Gu, A.; Gulcehre, C.; et al. Improving the Gating Mechanism of Recurrent Neural Networks. 2020, arXiv:1910.09890.

[52] Pascanu, R.; Mikolov, T.; et al. On the difficulty of training Recurrent Neural Networks. 2013, arXiv:1211.5063.

[53] Majeed, S.; HUSAIN, H.; et al. Mel frequency cepstral coefficients (Mfcc) feature extraction enhancement in the application of speech recognition: A comparison study. *Journal of Theoretical and Applied Information Technology*, volume 79, 09 2015: pp. 38–56.

[54] Jakobovski. Jakobovski/free-spoken-digit-dataset. (Accessed on 03/06/2021). Available from: https://github.com/Jakobovski/free-spoken-digit-dataset/tree/v1.0.8

[55] Bekolay, T.; Bergstra, J.; et al. Nengo: a Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, volume 7, no. 48, 2014: pp. 1–13, ISSN 1662-5196, doi:10.3389/fninf.2013.00048.

[56] Rasmussen, D. NengoDL: Combining deep learning and neuromorphic modelling methods. 2019, arXiv:1805.11144.

[57] Salamon, J.; Jacoby, C.; et al. A Dataset and Taxonomy for Urban Sound Research. In *Proceedings of the 22nd ACM International Conference on Multimedia*, MM '14, New York, NY, USA: Association for Computing Machinery, 2014, ISBN 9781450330633, p. 1041–1044, doi:10.1145/2647868.2655045. Available from: https://doi.org/10.1145/2647868.2655045

[58] LaPorte, J. Rigid Designators. Feb 2016, (Accessed on 03/06/2021). Available from: https://plato.stanford.edu/entries/rigid-designators/

[59] Nadeau, D.; Sekine, S. A Survey of Named Entity Recognition and Classification. *Lingvisticae Investigationes*, volume 30, 08 2007, doi: 10.1075/li.30.1.03nad.

[60] Kripke, S. *Naming and necessity*. Oxford: Blackwell, 1980, ISBN 978-0631128014.

[61] Bos, J.; Basile, V.; et al. The Groningen Meaning Bank. In *Handbook of Linguistic Annotation*, volume 2, edited by N. Ide; J. Pustejovsky, Springer, 2017, pp. 463–496.

[62] Maas, A. L.; Daly, R. E.; et al. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. Available from: http://www.aclweb.org/anthology/P11-1015

[63] Maass, W. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, volume 10, no. 9, 1997: pp. 1659–1671, ISSN 0893-6080, doi:https://doi.org/10.1016/S0893-6080(97)00011-7. Available from: https://www.sciencedirect.com/science/article/pii/S0893608097000117

[64] Izhikevich, E. Simple model of Spiking Neurons. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, volume 14, 02 2003: pp. 1569–72, doi:10.1109/TNN.2003.820440.

[65] Hodgkin, A. L.; Huxley, A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, volume 117, no. 4, 1952: pp. 500–544, doi:https://doi.org/10.1113/jphysiol.1952.sp004764, https://physoc.onlinelibrary.wiley.com/doi/pdf/10.1113/jphysiol.1952.sp004764. Available from: https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1952.sp004764

[66] Hojjatinia, S.; Aliyari Shoorehdeli, M.; et al. Improvement of the Izhikevich model based on the rat basolateral amygdala and hippocampus neurons, and recognition of their possible firing patterns. *Basic and Clinical Neuroscience Journal*, Nov 2019, ISSN 2008-126X, doi: 10.32598/bcn.9.10.435. Available from: http://dx.doi.org/10.32598/bcn.9.10.435

[67] Khun, J.; Novotný, M.; et al. High-Performance Spiking Neural Network Simulator. In *2019 8th Mediterranean Conference on Embedded Computing (MECO)*, 2019, pp. 1–4, doi:10.1109/MECO.2019.8760291.

[68] Roxin, A.; Brunel, N.; et al. Role of Delays in Shaping Spatiotemporal Dynamics of Neuronal Activity in Large Networks. *Phys. Rev. Lett.*, volume 94, Jun 2005: p. 238103, doi:10.1103/PhysRevLett.94.238103. Available from: https://link.aps.org/doi/10.1103/PhysRevLett.94.238103

[69] Appeltant, L.; Soriano, M.; et al. Information processing using a single dynamical node as complex system. *Nature communications*, volume 2, 09 2011: p. 468, doi:10.1038/ncomms1476.

[70] Boahen, K. A Neuromorph's Prospectus. *Computing in Science Engineering*, volume 19, 03 2017: pp. 14–28, doi:10.1109/MCSE.2017.33.

[71] Cassidy, A. S.; Alvarez-Icaza, R.; et al. Real-Time Scalable Cortical Computing at 46 Giga-Synaptic OPS/Watt with 100× Speedup in Time-to-Solution and 100,000× Reduction in Energy-to-Solution. *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014: pp. 27–38.

[72] Koch, C.; Poggio, T. Biophysics of Computation: Neurons, Synapses and Membranes. 10 2004. Available from: http://hdl.handle.net/1721.1/6414

[73] Cassidy, A. S.; Georgiou, J.; et al. Design of silicon brains in the nano-CMOS era: Spiking neurons, learning synapses and neural architecture optimization. *Neural Networks*, volume 45, 2013: pp. 4–26, ISSN 0893-6080, doi:https://doi.org/10.1016/j.neunet.2013.05.011, neuromorphic Engineering: From Neural Systems to Brain-Like Engineered Systems. Available from: https://www.sciencedirect.com/science/article/pii/S0893608013001597

[74] Merolla, P.; Arthur, J.; et al. ARTIFICIAL BRAINS A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science (New York, N.Y.)*, volume 345, 08 2014: pp. 668–673, doi:10.1126/science.1254642.

[75] Neckar, A.; Fok, S.; et al. Braindrop: A Mixed-Signal Neuromorphic Architecture With a Dynamical Systems-Based Programming Model. *Proceedings of the IEEE*, volume 107, no. 1, 2019: pp. 144–164, doi:10.1109/JPROC.2018.2881432.

[76] Furber, S. To build a brain. *Spectrum, IEEE*, volume 49, 08 2012: pp. 44–49, doi:10.1109/MSPEC.2012.6247562.

[77] Benjamin, B. V.; Gao, P.; et al. Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations. *Proceedings of the IEEE*, volume 102, no. 5, 2014: pp. 699–716, doi:10.1109/JPROC.2014.2313565.

[78] Mayr, C.; Hoeppner, S.; et al. SpiNNaker 2: A 10 Million Core Processor System for Brain Simulation and Machine Learning. 2019, arXiv:1911.02385.

[79] Neckar, A.; Fok, S.; et al. Braindrop: A Mixed-Signal Neuromorphic Architecture With a Dynamical Systems-Based Programming Model. *Proceedings of the IEEE*, volume 107, no. 1, 2019: pp. 144–164, doi:10.1109/JPROC.2018.2881432.

[80] Frady, E. P.; Orchard, G.; et al. Neuromorphic Nearest-Neighbor Search Using Intel's Pohoiki Springs. *CoRR*, volume abs/2004.12691, 2020, 2004.12691. Available from: https://arxiv.org/abs/2004.12691

[81] Hunsberger, E.; Eliasmith, C. Training Spiking Deep Networks for Neuromorphic Hardware. *CoRR*, volume abs/1611.05141, 2016, 1611.05141. Available from: http://arxiv.org/abs/1611.05141

[82] Spiking neural network conversion toolbox. (Accessed on 03/06/2021). Available from: https://snntoolbox.readthedocs.io/en/latest/guide/intro.html

[83] Stöckel, A. Discrete Function Bases and Convolutional Neural Networks. 2021, arXiv:2103.05609.

[84] Romero, D. W.; Kuzina, A.; et al. CKConv: Continuous Kernel Convolution For Sequential Data. 2021, arXiv:2102.02611.

Appendix A

Acronyms

GUI Graphical user interface

DN Delay Network

DA Data Augumentation

LTI Linear-Time-Invariant

CPU Central Processing Unit

GPU Graphics processing unit

API Application Programming Interface

SNN Spiking Neural Network

CNN Convolutional Neural Network

RNN Recurrent Neural Network

LMU Legendre Memory Unit

RMSE Root Mean Squared Error

CMOS Complementary Metal–Oxide–Semiconductor

FDSOI Fully Depleted Silicon On Insulator

XML Extensible markup language

NEF Neural Engineering Framework

GRU Gated Recurrent unit

LSTM Long-Short-Term-Memory

CEC Constant Error Carousel

LagT Laguerre Translated measure in HiPPO framework

LegT Legendre Translated measure in HiPPO framework

LegS Legendre Scaled measure in HiPPO framework

Appendix B

Contents of enclosed SD card

- `README.md` ..... basic info and instructions
- `notebooks` ..... directory with experiments as Jupyter notebooks
- `src` ..... directory with Python code used for preprocessing
- `thesis` ..... the directory of LaTeX source codes of the thesis
  - `thesis.pdf` ..... the thesis text in PDF format