Journal of Applied Science and Education, 2025,
Vol. 05, Iss. 02, S. No. 102, pp. 1-13
ISSN (Online): 2583-1372

Deep Learning in Borel Measure

Sakina Rizvi[1], Dr. Chitra Singh[2]

[1]Research Scholar, Rabindranath Tagore University, Bhopal (Madhya Pradesh)
[2]Professor, Rabindranath Tagore University, Bhopal (Madhya Pradesh)
rizvisakoo@gmail.com ,drchitrasingh123@gmail.com

How to cite this paper: S. Rizvi, C. Singh, "Deep Learning in Borel Measure," *Journal of Applied Science and Education (JASE)*, Vol. 05, Iss. 02, S. No. 102, pp 1-13, 2025.

https://doi.org/10.54060/a2zjournals.jase.103

Received: 12/02/2025
Accepted: 15/06/2025
Online First: 14/07/2025
Published: 14/07/2025

Abstract

Deep learning has had a profound impact on computer science in recent years, with applications to image recognition, language processing, bioinformatics, and more. Recently, Cohen et al. provided theoretical evidence for the superiority of deep learning over shallow learning. We formalized their mathematical proof using Isabelle/HOL. The Isabelle development simplifies and generalizes the original proof, while working around the limitations of the HOL type system. To support the formalization, we developed reusable libraries of formalized mathematics, including results about the matrix rank, the Borel measure, and multivariate polynomials as well as a library for tensor analysis.

Keywords

Isabelle/HOL • Deep learning • Machine learning • Convolutional arithmetic circuits • Formalization • Tensors

1. Introduction

Deep learning algorithms enable computers to perform tasks that seem beyond what we can program them to do using traditional techniques. In recent years, we have seen the emergence of unbeatable computer go players, practical speech recognition systems, and self- driving cars. These algorithms also have applications to image recognition, bioinformatics, and many other domains. Yet, on the theoretical side, we are only starting to understand why deep learning works so well. Recently, Cohen et al. [6] used tensor theory to explain the superiority of deep learning over shallow learning for one specific learning architecture called convolutional arithmetic circuits (CACs).

Machine learning algorithms attempt to model abstractions of their input data. A typical application is image recogni-

tion, e.e., classifying a given image in one of several categories, depending on what the image depicts. The algorithms are usually learned from a set of data points, each specifying an input (the image) and a desired output (the category). This learning process is called training. The algorithms generalize the sample data, allowing them to imitate the learned output on previously un-seen input data.

CACs impose the structure of the popular convolutional neural networks (CNNs) onto sum–product networks, using alternating convolutional and pooling layers, which are realized as collections of sum nodes and product nodes, respectively. These networks can be shallower or deeper—i.e., consist of few or many layers—and each layer can be arbitrarily small or large, with low or high arity sum nodes. CACs are equivalent to similar networks, which have been demonstrated to perform at least as well as CNNs [5].

As an exercise in mechanizing modern research in machine learning, we developed formal proof of the funda-mental theorem with and without weight sharing using the Isabelle/HOL proof assistant [17,18]. To simplify our work, we recast the original proof into a more modular version, which generalizes the result as follows: S is not only a Lebesgue null set, but also a subset of the zero set of a nonzero multivariate polynomial. This stronger theorem gives a clearer picture of the expressiveness of deep CACs.

The formal proof builds on general libraries that we either developed or enriched. We created a library for tensors and their operations, including products, CP-rank, and matricization. We added the matrix rank and its properties to Thiemann and Yamada's matrix library [27], generalized the definition of the Borel measure by Hölzl and Himmelmann [13], and extended Lochbihler and Haftmann's polynomial library [12] with various lemmas, including the theorem stating that zero sets of nonzero multivariate polynomials are Lebesgue null sets. For matrices and the Lebesgue measure, an issue we faced was that the definitions in the standard Isabelle libraries have too restrictive types: the dimensionality of the matrices and of the measure space is parameterized by types that encode numbers, whereas we needed them to be on terms.

2. ISABELLE/HOL

Isabelle [17,18] is a generic proof assistant that supports many object logics. The metalogic is based on an intuitionistic fragment of Church's simple type theory [4]. The types are built from type variables α, β… and n-ary type constructors, normally written in postfix notation

(e.g., α *list*). The infix type constructor $\alpha \Rightarrow \beta$ is interpreted as the (total) function space from α to β. Function applications are written in a curried style (e.g., f *x y*). Anonymous functions $x \rightarrow y_x$ are written $\lambda x.\ y_x$. The notation $t :: \tau$ indicates that term t has type τ.

Isabelle's architecture follows the tradition of the theorem prover LCF [11] in implementing a small inference kernel that verifies the proofs. Trusting an Isabelle proof involves trusting this kernel, the formulation of the main theorems, the assumed axioms, the compiler and runtime system of Standard ML, the operating system, and the hardware. Specification mechanisms help us define important classes of types and functions, such as inductive datatypes and recursive functions, without introducing axioms. Since additional axioms can lead to inconsistencies, it is generally good style to use these mechanisms.

Isabelle locales are a convenient mechanism for structuring large proofs. A locale fixes types, constants, and assumptions within a specified scope. For example, an informal math-ematical text stating "in this section, let *A* be a set of natural numbers and *B* a subset of *A*" could be formalized by introducing a locale *AB_subset* as follows:

locale *AB_subset* = **fixes** A B :: *nat set* **assumes** B ⊆ A

Definitions made within the locale may depend on A and B, and lemmas proved within the locale may use the assumption that B ⊆ A. A single locale can introduce arbitrarily many types, constants, and assumptions. Seen from the out-

side, the lemmas proved in a locale are polymorphic in the fixed type variables, universally quantified over the fixed constants, and conditional on the locale's assumptions. It is good practice to provide at least one interpretation after defining a locale to show that the assumptions are consistent. For example, we can interpret the above locale using the empty set for both A and B by proving that $\emptyset \subseteq \emptyset$:

interpretation *AB_subset_empty*: *AB_subset* $\emptyset\,\emptyset$
using *AB_subset_def* **by** *simp*

3. Mathematical Premliminaries

3.1. Tensors

Tensors can be understood as multidimensional arrays, with vectors and matrices as the one and two-dimensional cases. Each index corresponds to a *mode* of the tensor. For matrices, the modes are called row" and "column". The number of modes is the *order* of the tensor. The number of values an index can take in a particular mode is the *dimension* in that mode. Thus, a real-valued tensor $A \in \mathrm{R}^{M_1 \times \cdots \times M_N}$ of order N and dimension M_i in mode i contains $A_{d_1,\ldots,d_N} \in \mathrm{R}$ for $d_i \in \{1,\ldots, M_i\}$.

The matricization [A] of a tensor A is a matrix obtained by rearranging A's entries using a bijection between the tensor and matrix entries. It has the following property:

Lemma 1 *Given a tensor A, we have* rank $[A] \leq$ CP-rank A.

3.2. Lesbegue Measure

The Lebesgue measure is a mathematical description of the intuitive concept of length, surface, or volume. It extends this concept from simple geometrical shapes to many subsets of R^n, including all closed and open sets, although it is impossible to design a measure that caters for all subsets of R^n while maintaining intuitive properties. The sets to which the Lebesgue measure can assign a volume are called *measurable.* The volume that is assigned to a measurable set can be a nonnegative real number or ∞. A set of Lebesgue measure 0 is called a *null set.* If a property holds for all points in R^n except for a null set, the property is said to be held almost everywhere.

Lemma 2 *If $p \neq 0$ is a polynomial in d variables, the set of points* $\mathbf{x} \in \mathrm{R}^d$ *with* $p(\mathbf{x}) = 0$ *is lebesgue null set.*

$$
\begin{aligned}
u_{0,j} &= x_j \in \mathbb{R}^M \\
v_{0,j} &= w_{0.j} \,.\, u_{0.j} \in \mathbb{R}^{r_0} \\
u_{1.j} &= v_{0,2j-1} * v_{0,2j} \in \mathbb{R}^{r_0} \\
v_{1,j} &= w_{1,j} \,.\, u_{1.j} \in \mathbb{R}^{r_1} \\
&\vdots \\
v_{d-1,j} &= v_{d-2,2j-1} * v_{d-2,2j} \\
&\qquad \in \mathbb{R}^{r_{d-2}} \\
v_{d-1,j} &= w_{d-1,j} \,.\, u_{d-1,j} \in \mathbb{R}^{r_{d-1}} \\
u_{d,1} &= u_{d-1,1}{}^{*\ldots} v_{d-1,N/2^{d-1}} \in \mathbb{R}^{r_{d-1}} \\
y &= v_{d,1} = w_{d,1.u_{d,1}} \in \mathbb{R}^y
\end{aligned}
$$

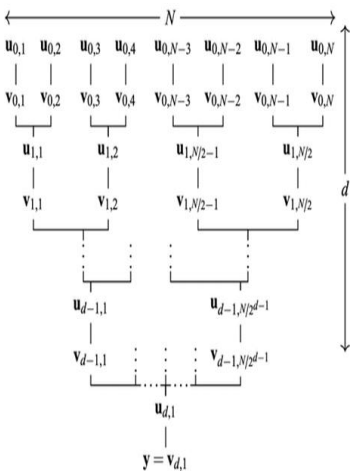


Figure 1. Definition and hierarchical structure of CAC with d layers

4. The Theorems of Network Capacity

A CAC is defined by the following parameters: the number of input vectors N, the depth d, and the dimensions of the weight matrices $r_{-1}, \ldots, r_d$. The number N must be a power of 2 and d can be any number between 1 and $\log_2 N$. The size of the input vectors is $M = r_{-1}$ and the size of the output vector is $Y = r_d$.

The evaluation of a CAC—i.e., the calculation of its output vector given the input vectors depends on learned weights. The results by Cohen et al. are concerned only with the expressiveness of these networks and are applicable regardless of the training algorithm. The weights are organized as entries of a collection of real matrices $W_{l,j}$ of dimension $r_l \times r_{l-1}$, where l is the index of the layer and j is the position in that layer where the matrix is used. A CAC has *shared weights* if the same weight matrix is applied within each layer l—i.e., $W_{l,1} = \cdots = W_{l,N/2^l}$. The *weight space* of a CAC is the space of all possible weight configurations.

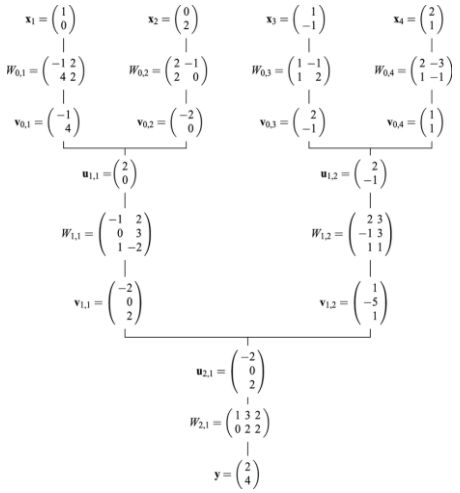


Figure 2. A CAC with concrete weights evaluated on concrete input vectors

Theorem 3 (Fundamental theorem of network capacity) We consider two CACs with identical N, M, and Y parameters: a deep network of depth d = $\log_2$ N with weight matrix dimensions $r_{1,l}$ and a shallow network of depth d = 1 with weight matrix dimensions $r_{2,l}$. Let r = min ($r_{1,0}$, M) and assume $r_{2,0} < r^{N/2}$. Let S be the set of configurations in the weight space of the deep network that express functions also expressible by the shallow network. Then S is a Lebesgue null set. This result holds for networks with and without shared weights.

$$
\begin{aligned}
\mathbf{u}_{0,j} &= \mathbf{x}_j && \in \mathbb{R}^M \\
\mathbf{v}_{0,j} &= W_{0,j} \cdot \mathbf{u}_{0,j} && \in \mathbb{R}^{r_{2,0}} \\
\mathbf{u}_{1,1} &= \mathbf{v}_{0,1} * \cdots * \mathbf{v}_{0,N} && \in \mathbb{R}^{r_{2,0}} \\
\mathbf{y} = \mathbf{v}_{1,1} &= W_{1,1} \cdot \mathbf{u}_{1,1} && \in \mathbb{R}^Y
\end{aligned}
$$

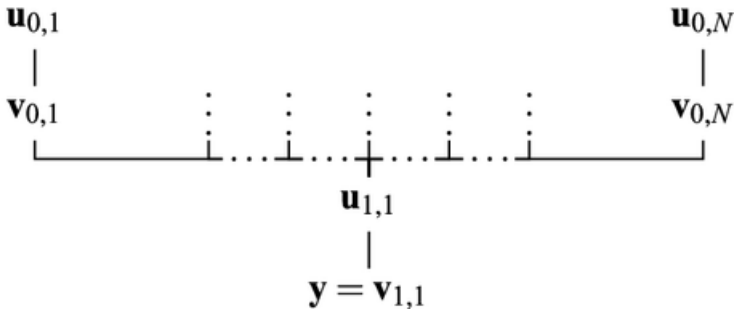


Figure 3. Evaluation formulas and hierarchical structure of a shallow CAC

Theorem 4 (Generalized Theorem of Network Capacity) Consider two CACs with identical N, M and Y parameters: a deeper network of depth d_1 and weight matrix dimension $r_{1,l}$ and a shallower network of depth $d_2 < d_1$ with weight matrix dimension

$r_{2,l}$. Let r = min{M, *($r_{1,0}$,… $r_{1,}$ d_2-1) and assume*

$r_{2,}$ d_2 -1< $r^{N/2\ d2}$.

Let S be the set of configurations in the weight space of the deep network that express functions also expressible by the shallow network. Then S is a Lebesgue null set. This result holds for networks with and without shared weights

5. Formal Libraries

Our proof requires basic results in matrix, tensor, polynomial, and measure theory. For matrices and polynomials, Isabelle offers several libraries, and we chose those that seemed the most suitable. We adapted the measure theory from Isabelle's analysis library and developed a new tensor library.

5.1. Matrices

We contributed a definition of the matrix rank, as the dimension of the space spanned by the matrix columns:

definition *(***in** *vec_space)* rank::*α mat* ⇒ *nat* **where**

rank *A* = vectorspace.dim F *(*span_vs *(*set *(*cols *A)))*

5.2. Tensors

The *Tensor* entry [24] of the *Archive of Formal Proofs* might seem to be a good starting point for the formalization of tensors. However, despite its name, this library does not contain a type for tensors. It introduces the Kronecker product, which is equivalent to the tensor product but operates on the matricizations of tensors.

The *Group-Ring-Module* entry [16] of the *Archive of Formal Proofs* could have been another potential basis for our work. Unfortunately, it introduces the tensor product in a very abstract fashion and does not integrate well with other Isabelle libraries. Instead, we introduced our own type for tensors, based on a list that specifies the dimension in each mode and a list containing all of its entries:

typedef *α tensor* = {*(ds* ::*nat list, as* ::*α list).* length *as* = $\prod ds$}

We formalized addition, multiplication by scalars, product, matricization, and the CP- rank. We instantiated addition as a semigroup (*semigroup_add*) and tensor product as a monoid (*monoid_mult*). Stronger type classes cannot be instantiated: their axioms do not hold collectively for tensors of all sizes, even though they hold for fixed tensor sizes. For example, it is impossible to define addition for tensors of different sizes while satisfying the cancellation property $a + c = b + c \rightarrow$ =b. We left the addition of tensors of different sizes underspecified. For proving properties of addition, scalar multiplication, and product, we devised a powerful induction principle on tensors, which relies on tensor slices. The induction step amounts to showing a property for a tensor $A \in R^{M1 \times \cdots \times MN}$ assuming it holds for all slices

$A_i \in R^{M_2 \times \cdots \times M_N}$, which are obtained by fixing the first index $i \in \{1, \ldots, M_1\}$.Matricization rearranges the entries of a tensor $A \in R^{M_1 \times \cdots \times M_N}$ into a matrix $[A] \in R^{I \times J}$ for some suitable I and J. This rearrangement can be described as a bijection between $\{0, \ldots, M_1 - 1\} \times \cdots \times \{0, \ldots, M_N - 1\}$ and $\{0, \ldots, I - 1\} \times \{0, \ldots,$ $\quad, J$ *-1)*two sets $\{r_1 < \cdots < r_K\} \uplus \{c_1 < \cdots < c_L\} = \{1, \ldots, N\}$. The proof of Theorem 4 uses only standard matricization, which partitions the indices into odd and even numbers, but we

formalized the more general formulation [**1**]. The matrix [*A*] has *I* = $\prod_{i=1}^{k} r_i$ rows and J = $\prod_{J=1}^{L} c_i$ columns. The rearrangement function is $(i_1, \ldots\ldots\ i_N) \rightarrow \sum_{k=1}^{K}(i_{r_k} \prod_{k'=1}^{k-1} M_{r_{k'}}), (\sum_{l=1}^{L}(i_{c_l} \prod_{l'=1}^{l-1} M_{c_{l'}}))$

The indices $i_{r_1}, \ldots, i_{r_k}$ *and* $i_{c_1}, \ldots, i_{c_L}$ *serve as digits in a mixed* -base numeral system to specify the row and the column in the matricization. Expand the sum and product operators and factor out the bases M_i

$(i_1, \ldots\ldots\ i_N) \rightarrow (i_{r_1}+M_{r_1} .(i_{r_2}+M_{r_2} . \ldots\ldots i_{r_{K-1}}+M_{r_{K-1}}. i_{r_K}) \ldots)),$

$(i_{c_1}+M_{c_1} .(i_{c_2}+M_{c_2} . \ldots\ldots i_{c_{L-1}}+M_{c_{L-1}}. i_{c_K}) \ldots)),$

5.3. Lebesgue Measure

At the time of our formalization work, Isabelle's analysis library defined only the Borel measure on R^n but not the closely related Lebesgue measure. The Lebesgue measure is the completion of the Borel measure. The two measures are identical on all sets that are Borel measurable, but the Lebesgue measure can measure more sets. Following the proof by Cohen et al., we can show that the set *S* defined in Theorem 4 is a subset of a Borel null set. It follows that *S* is a Lebesgue null set, but not necessarily a Borel null set.

To resolve this mismatch, we considered three options: (1).Prove that *S* is a Borel null set, which we believe is the case, although it does not follow trivially from *S*'s being a subset of a Borel null set; (2)Define the Lebesgue measure, using the already formalized Borel measure and measure completion.; (3)Use the Borel measure whenever possible and use the almost-everywhere quantifier ($\forall_{ae}$) otherwise. We chose the third approach, which seemed simpler. Theorem 4, defines *S* as the set of configurations in the weight space of the deeper network that express functions also expressible by the shallower network, and then asserts that *S* is a null set. In the formalization, we state this as follows: almost everywhere in the weight space of the deeper network, the deeper network expresses function not expressible by the shallower network. This formulation is equivalent to asserting that *S* is a subset of a null set, which we can easily prove for the Borel measure as well.

There is, however, another issue with the definition of the Borel measure from Isabelle's analysis library:

definition lborel ::*(α ::euclidean_space) measure* **were**

lborel = distr (∏M *b* ∈ Basis. interval_measure *(λx. x))* borel

(λf. ∑ b∈Basis. f b∗R^b)

The type *α* specifies the number of dimensions of the measure space. In our proof, the measure space is the weight space of the deeper network, and its dimension depends on the number *N* of inputs and the size r_l of the weight matrices. The number of dimensions is a term in our proof. We described a similar issue with Isabelle's matrix library already.

The solution is to introduce a new notion of the Borel measure whose type does not fix the number of dimensions. This multidimensional Borel measure is the product measure (∏M) of the one-dimensional Borel measure (lborel ::*real measure*) with itself:

definition lborel_f ::*nat ⇒ (nat ⇒ real) measure* **where**

lborel_f *n* = (∏M *b* ∈{. .< *n*}. lborel*)*

5.4. Multivariate Polynomials

Several multivariate polynomial libraries have been developed to support other formalization projects in Isabelle. Sternagel and Thiemann [26] formalized multivariate polynomials designed for execution, but the equality of polynomials is a custom predicate, which means that we cannot use Isabelle's simplifier to rewrite polynomial expressions. Immler and Maletzky [13] formalized an axiomatic approach to multivariate polynomials using type classes, but their focus is not on the evaluation homomorphism, which we need. Instead, we chose to extend a previously unpublished multivariate polynomial library by

Lochbihler and Haftmann [11]. We derived induction principles and properties of the evaluation homomorphism and of nested multivariate polynomials. These were useful to formalize Lemma:

lemma *lebesgue_mpoly_zero_set*:
fixes p :: *real mpoly*
assumes $p \neq 0$ **and** vars $p \subseteq \{..<n\}$
shows $\{x \in$ space $(\text{lborel}_f\ n).$ insertion $x\ p = 0\} \in$ null_sets $(\text{lborel}_f\ n)$

6. Formalization of the Fundamental Theorem

With the necessary libraries in place, we undertook the formal proof of the fundamental theorem of network capacity, starting with the CACs. A recursive datatype is appropriate to capture the hierarchical structure of these networks:

datatype $\alpha\ cac =$ Input *nat* | Conv α (α *cac*) | Pool (α *cac*) (α *cac*)

To simplify the proofs, Pool nodes are always binary. Pooling layers that merge more than two branches are represented by nesting Pool nodes to the right. The type variable α can be used to store weights. For networks without weights, it is set to *nat* × *nat*, which associates only the matrix dimension with each Conv node. For networks with weights, α is *real mat*, an actual matrix. These two network types are connected by insert_weights, which inserts weights into a weightless network, and its inverse extract_weights :: *bool* ⇒ *real mat cac* ⇒ *nat* ⇒ *real*, which retrieves the weights from a network containing weights.

fun insert_weights :: *bool* ⇒ *(nat* × *nat) cac* ⇒ *(nat* ⇒ *real)* ⇒ *real mat cac* **where**
insert_weights *shared (*Input *M) w* = Input *M*
| insert_weights *shared (*Conv *(r, c) m) w* =
Conv *(*extract_matrix *w r c) (*insert_weights *shared m (λi. w (i* + *r* ∗ *c)))*
| insert_weights *shared (*Pool *m1 m2) w* =
Pool *(*insert_weights *shared m1 w) (*insert_weights *shared m2*
*(*if *shared* then *w* else *(λi. w (i* + count_weights *shared m1))))*
fun extract_weights :: *bool* ⇒ *real mat convnet* ⇒ *nat* ⇒ *real* **where**
extract_weights *shared (*Input *M) i* = 0
| extract_weights *shared (*Conv *A m) i* =
if *i* < $\dim_r A$ ∗ $\dim_c A$ then flatten_matrix *A i*
else extract_weights shared *m (i* − $\dim_r A$ ∗ $\dim_c A$*)*
| extract_weights *shared (*Pool m_1 m_2*) i* =
if *i* < count_weights *shared* m_1 then extract_weights *shared* m_1 *i*
else extract_weights *shared* m_2 *(i* − count_weights *shared* m_1*)*

The first argument of these two functions specifies whether the weights should be shared among the Conv nodes of the same layer. The weights are represented by a function *w*, of which only the first *k* values *w* 0, *w* 1, . . . , *w* (*k* − 1) are used. Given a matrix, flatten_matrix creates such a function representing the matrix entries. Sets over *nat* ⇒ *real* can be measured using lborel_f. The count_weights function returns the number of weights in a network.

The next function describes how the networks are evaluated:

fun evaluate_net :: *real mat cac* ⇒ *real vec list* ⇒ *real vec* **where**
evaluate_net *(*Input *M) is* = hd *is*
| evaluate_net *(*Conv *A m) is* = *A* $\otimes_{mv}$ evaluate_net *m is*

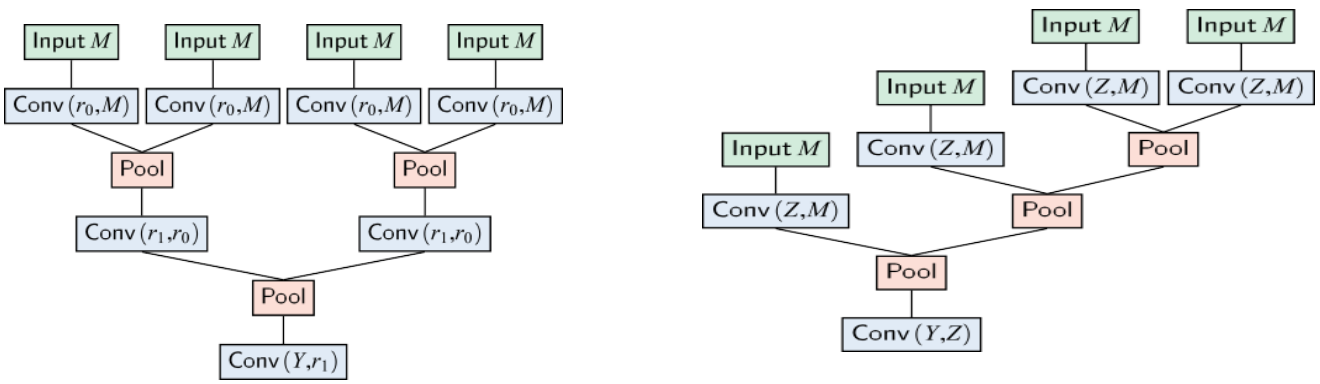
| evaluate_net (Pool m_1 m_2) *is* = component_mult
(evaluate_net m_1 (take (length (input_sizes m_1)) *is*))
(evaluate_net m_2 (drop (length (input_sizes m_1)) *is*))

where $\otimes_{mv}$ multiplies a matrix with a vector, and component_mult multiplies vectors component wise.

The *cac* type can represent networks with arbitrary nesting of Conv and Pool nodes, going beyond the definition of CACs. Moreover, since we focus on the fundamental theorem, it suffices to consider a deep model with $d_1 = \log_2 N$ and a shallow model with $d_2 = 1$. These are specified by generating functions:

fun
deep_model$_0$ :: *nat* ⇒ *nat list* ⇒ *(nat* × *nat) cac* **and**
deep_model :: *nat* ⇒ *nat* ⇒ *nat list* ⇒ *(nat* × *nat) cac*
where
deep_model$_0$ *Y* [] = Input *Y*
| deep_model$_0$ *Y (r # rs)* = Pool (deep_model *Y r rs)* (deep_model *Y r rs)*
| deep_model *Y r rs* = Conv *(Y, r)* (deep_model$_0$ *r rs)*

fun shallow_model$_0$ :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *(nat* × *nat) cac* **where**
shallow_model$_0$ *Z M* 0 = Conv *(Z, M)* (Input *M)*
| shallow_model$_0$ *Z M* (Suc *N)* = Pool (shallow_model$_0$ *Z M* 0) (shallow_model$_0$ *Z M N)*



a) deep _model *Y r*1 [*r*0, *M*] b) shallow_model *Y Z M* 3

Figure 4. A deep and shallow network represented using the CAC datatype

definition shallow_model :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *(nat* × *nat) cac* **where**
shallow_model *Y Z M N* = Conv *(Y, Z)* (shallow_model$_0$ *Z M N)*

Two examples are given in Fig. **4. F**or the deep model, the arguments *Y # r # rs* correspond to the weight matrix sizes [$r_{1,d}$ *(= Y),* $r_{1,d-1}$, …, $r_{1,0}$, $r_{1,-1}$ *(= M)*]. For the shallow model, the arguments *Y* , *Z*, *M* correspond to the parameters $r_{2,1}$ *(= Y)*, $r_{2,0}$, $r_{2,-1}$ *(= M)*, and *N* gives the number of inputs minus 1.

Step I The following operation computes a list, or vector, of tensors representing a net- work's function, each tensor standing for one component of the output vector:
fun tensors_from_net :: *real mat cac* ⇒ *real tensor vec* **where**
tensors_from_net (Input *M)* = Matrix.vec *M (λi.* unit_vec *M i)*

| tensors_from_net (Conv *A m*) =
mat_tensorlist_mult *A* (tensors_from_net *m*) (input_sizes *m*)
| tensors_from_net (Pool m_1 m_2) =
Component_mult (tensors_from_net m_1) (tensors_from_net m_2)

lemma *tensors_from_net_eqI*:
assumes valid_net' m_1 **and** valid_net' m_2 **and** input_sizes m_1 = input_sizes m_2
and ∀*is*. input_correct *is* ⟶ evaluate_net m_1 *is* = evaluate_net m_2 *is*
shows tensors_from_net m_1 = tensors_from_net m_2

The fundamental theorem is a general statement about deep networks. It is useful to fix the deep network parameters in a locale:

locale deep_model_correct_params
fixes rs :: *nat list* **and** shared :: *bool*
assumes length rs ≥ 3 **and** ∀*r* ∈ set rs. *r* > 0

The list rs completely specifies one specific deep network model:

abbreviation deep_net :: (*nat* × *nat*) *cac* **where**
deep_net = deep_model (rs ! 0) (rs ! 1) (tl (tl rs))

The parameter shared specifies whether weights are shared across Conv nodes within the same layer. The other parameters of the deep network can be defined based on rs and shared:

definition r :: *nat* **where** r = min (last rs) (last (butlast rs))
definition N_half :: *nat* **where** N_half = $2^{\text{length rs} - 3}$
definition weight_space_dim :: *nat* **where**
weight_space_dim = count_weights shared deep_net

The shallow network must have the same input and output sizes as the deep network to express the same function as the deep network. This leaves only the parameter $Z = r_{2,0}$, which specifies the matrix weight sizes in the Conv nodes and the size of the vectors multiplied in the Pool nodes of the shallow network:

abbreviation shallow_net :: *nat* ⇒ (*nat* × *nat*) *cac* **where**
shallow_net *Z* = shallow_model (rs ! 0) *Z* (last rs) (2 ∗ N_half − 1)

Following the proof sketch, we consider a single output component y_i. We rely on a second locale that introduces a constant i for the index of the considered output component. We provide interpretations for both locales.

locale deep_model_correct_params_output_index = deep_model_correct_params +
fixes i :: *nat*
assumes i < rs ! 0

Then we can define the tensor A_i, which describes the behavior of the function expressed by the deep network at the output component y_i, depending on the weight configuration *w* of the deep network:

definition A_i :: (*nat* ⇒ *real*) ⇒ *real tensor* **where**
A_i *w* = tensors_from_net (insert_weights shared deep_net *w*) ! i

We want to determine for which *w* the shallow network can express the same function and is hence represented by the same tensor.

Step II We must show that if a tensor *A* represents the function expressed by the shallow network, then $r_{2,d_2} - 1 \geq$ CP-rank $(\phi(A))$.

For the fundamental theorem, ϕ is the identity and $d_2 = 1$. Hence, it suffices to prove that $Z = r_{2,0} \geq$ CP-rank (A):

lemma *cprank_shallow_model*:

cprank (tensors_from_net (insert_weights shared *w* (shallow_net *Z*)) ! i) ≤ *Z*

This lemma can be proved easily from the definition of the CP-rank.

Step III We define the polynomial *p* and prove that it has properties IIIa and IIIb. Defining *p* as a function is simple:

definition p_{func} :: *(nat ⇒ real) ⇒ real* **where**

p_{func} *w* = det (submatrix [*A* ¡ *w*] rows_with_1 rows_with_1)

where [*A* ¡ *w*] abbreviates the standard matricization matricize {*n.* even *n* (*A* ¡ *w*), and rows_with_1 is the set of row indices with 1s in the main diagonal for a specific weight configuration *w* defined in step III b. Our aim is to make the sub-matrix as large as possible while maintaining property that *p* is not the zero polynomial. The bound-on *Z* in the statement of the final theorem is derived from the size of this submatrix.

The function p_{func} must be shown to be a polynomial function. We introduce a predicate polyfun that determines whether a function is a polynomial function:

definition polyfun :: *nat set ⇒ ((nat ⇒ real) ⇒ real) ⇒ bool* **where**

polyfun *N f* ⇔ ∃*p.* vars *p* ⊆ *N* ∧ ∀*x.* insertion *x p* = *f x*

This predicate is preserved from constant and linear functions through the tensor representation of the CAC, matricization, choice of submatrix, and determinant:

lemma *polyfun_p*:

polyfun {..< weight_space_dim} p_{func}

Step IIIa We must show that if *p(w)* ≠ 0, then CP-rank $(A_i(w)) \geq r^{N/2}$. The Isar proof is sketched below:

lemma *if_polynomial_0_rank*:

assumes p_{func} *w* ≠ 0

shows r^{N_half} ≤ cprank (*A*¡ *w*)

Proof -

have r^{N_half} = dim_r (submatrix [*A*¡ *w*] rows_with_1 rows_with_1)

by calculating the size of the submatrix

also have ⋯ ≤ mrank [*A*¡ *w*]

also have ⋯ ≤ cprank (*A*¡ *w*) using Lemma (Given a tensor A , we have rank [A] ≤CP-rank A.

Finally, show? thesis.

qed

Step IIIb To prove that *p* is not the zero polynomial, we must exhibit a witness weight configuration where *p* is nonzero. Since weights are arranged in matrices, we define concrete matrix types: matrices with 1s on their diagonal and 0s elsewhere (id_matrix), matrices with 1s everywhere (all1_matrix), and matrices with 1s in the first column and 0s elsewhere (copy_first_matrix). For example, the last matrix type is defined as follows:

definition copy_first_matrix :: *nat ⇒ nat ⇒ real mat* **where**

copy_first_matrix *nr nc* = mat *nr nc (λ(r, c).* if *c* = 0 then 1 else 0)

For each matrix type, we show how it behaves under multiplication with a vector:

lemma *mult_copy_first_matrix*:

assumes *i* < *nr* **and** dim_v *v* > 0

shows (copy_first_matrix *nr* (dim_v *v*) $\otimes_{mv}$ *v*) ! *i* = *v* ! 0

Using these matrices, we can define the deep network containing the witness weights:

Fun

witness$_0$ :: *nat* ⇒ *nat list* ⇒ *real mat cac* **and**

witness:: *nat* ⇒ *nat* ⇒ *nat list* ⇒ *real mat cac*

were

witness$_0$ *Y* [] = Input *Y*

| witness$_0$ *Y (r # rs)* = Pool *(*witness *Y r rs) (*witness *Y r rs)*

| witness *Y r* [] = Conv *(*id_matrix *Y r) (*witness$_0$ *r* []*)*

| witness *Y r* [*a*] = Conv *(*all1_matrix *Y r) (*witness$_0$ *r* [*a*]*)*

| witness *Y r (a# b # rs)* = Conv *(*copy_first_matrix *Y r) (*witness$_0$ *r (a # b # rs))*

The network's structure is identical to deep_model. For each Conv node, we carefully choose one of the three matrix types we defined, so that the representing tensor of this network has as many 1s as possible on the main diagonal and 0s elsewhere. This in turn ensures that its matricization has as many 1s as possible on its main diagonal and 0s elsewhere. The rows_with_1 constant specifies the row indices that contain the 1s.

We extract the weights from the witness network using the function extract_weights:

definition witness_weights :: *nat* ⇒ *real* **where**

witness_weights = extract_weights shared deep_net

We prove that the representing tensor of the witness network, which is equal to the tensor A_i witness_weights, has the desired form. This step is rather involved: we show how the defined matrices act in the network and perform a tedious induction over the witness network. Then we can show that the submatrix characterized by rows_with_1 of the matricization of this tensor is the identity matrix of size r^{N_half}:

lemma *witness_submatrix*:

submatrix [A_i witness_weights] rows_with_1 rows_with_1 = id_matrix r^{N_half} r^{N_half}

As a consequence of this lemma, the determinant of this submatrix, which is the definition of p_{func}, is nonzero. Therefore, *p* is not the zero polynomial:

lemma *polynomial_not_zero*:

p_{func} witness_weights ≠ 0

Fundamental Theorem The results of steps II and III can be used to establish the fundamental theorem:

Theorem *fundamental_theorem_of_network_capacity*:

$\forall_{ae}$ w_d w.r.t. lborel$_f$ weight_space_dim. $\nexists w_s$ *Z*. $Z < r^{N\_half}$ ∧

∀*is*. input_correct *is* −→

evaluate_net *(*insert_weights shared deep_net w_d*) is* =

evaluate_net *(*insert_weights shared *(*shallow_net *Z) w*$_s$*) is*

Here, '$\forall_{ae}$ *x* w.r.t. *m*. P_x ' means that the property P_x holds almost everywhere with respect to the measure *m*. The r^{N_half} bound corresponds to the size of the identity matrix in the *witness_submatrix.*

7. Conclusion

We applied for a proof assistant to formalize recent results in a field where they have been little used before, namely machine learning. We found that the functionality and libraries of a modern proof assistant such as Isabelle/HOL were mostly up to the task. Beyond the formal proof of the fundamental theorem of network capacity, our main contribution is a general library of tensors. Admittedly, even the formalization of short pen-and-paper proofs can require a lot of work, partly because of the need to develop and extend libraries. On the other hand, not only does the process led to a computer verification of

the result, but it can also reveal new ideas and results. The generalization and simplifications we discovered illustrate how formal proof development can be beneficial to research outside the small world of interactive theorem proving.

8. Acknowledgment

I would like to express my sincere gratitude to Dr. Chitra Singh for her invaluable guidance, encouragement, and insightful feedback, which greatly contributed to the success of this study. My appreciation also goes to Rabindranath Tagore University, Department of Mathematical Sciences, for providing the necessary resources and support throughout the course of this research.

9. References

[1.] B. W. Bader, T. G. Kolda, "Algorithm 862: MATLAB tensor classes for fast algorithm prototyping," *ACM Trans. Math. Softw.* vol. 32, no. 4, pp. 635–653, 2006.

[2.] A. Bentkamp, *Expressiveness of deep learning. Archive of Formal Proofs*. 2016. http://isa-afp.org/entries/Deep_Learning.shtml

[3.] A. Bentkamp, *An Isabelle formalization of the expressiveness of deep learning*. 2016. http://matryoshka.gforge.inria.fr/pubs/bentkamp_msc_thesis.pdf

[4.] A. Church, "A formulation of the simple theory of types," *J. Symb. Log.* vol. 5, no. 2, pp. 56–68, 1940.

[5.] N. Cohen, O. Sharir, and A. Shashua, "Deep SimNets," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[6.] N. Cohen, O. Sharir, and A. Shashua, "On the expressive power of deep learning: A tensor analysis," *arXiv [cs.NE]*, 2015)

[7.] N. Cohen and A. Shashua, "Convolutional rectifier networks as generalized tensor decompositions," *arXiv [cs.NE]*, 2016.

[8.] N. Cohen, A. Shashua, "Inductive bias of deep convolutional networks through pooling geometry," *CoRR arXiv:1605.06743*, 2016.

[9.] N. Cohen, R. Tamari, A. Shashua A, "Boosting dilated convolutional networks with mixed tensor decompositions," *CoRR arXiv: 1703.06846,* 2017.

[10.] M. J. C. Gordon, R. Milner, and C. P. Wadsworth, "Edinburgh LCF: A Mechanised Logic of Computation," *LNCS*, vol. 78, 1979.

[11.] F. Haftmann, A. Lochbihler, and W. Schreiner, "Towards abstract and executable multivariate polynomials in Isabelle," in *Isabelle Workshop*, T. Nipkow, L. Paulson, and M. Wenzel, Eds. 2014.

[12.] J. Hölzl and A. Heller, "Three chapters of measure theory in Isabelle/HOL," in *Interactive Theorem Proving*, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 135–151, 2011.

[13.] F. Immler and A. Maletzky, "Formal proof development," in *Gröbner bases theory. Archive of Formal Proofs*, 2016. http://isa-afp.org/entries/Groebner_Bases.shtml

[14.] R.Kam,*Case studies proof checking*. 2007. http:// scholar works.sjsu.edu/cgi/viewcontent.cgi?context=etd_projects&article=1149

[15.] K. Kawaguchi, "Deep learning without poor local minima," *arXiv [stat.ML]*, 2016.

[16.] L. Liu, V. Aravantinos, O. Hasan, and S. Tahar, "On the formal analysis of HMM using theorem proving," in *Formal Methods and Software Engineering*, Cham: Springer International Publishing, pp. 316–331, 2014.

[17.] M. Lotz, "On the volume of tubular neighborhoods of real algebraic varieties," *Proc. Am. Math. Soc.* vol. 143,

no. 5, pp. 1875–1889, 2015.

[18.] C. Murphy, P. Gray, and G. Stewart, “Verified perceptron convergence theorem,” in *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 43-50, 2017

[19.] T. Nipkow and G. Klein, *Concrete Semantics: With Isabelle/HOL*. Berlin: Springer, 2014.

[20.] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, vol. 2283. Berlin: Springer, 2002.

[21.] L. Paulson, “Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers,” 2018.

[22.] L. C. Paulson and K. W. Susanto, “Source-level proof reconstruction for interactive theorem proving,” in *Lecture Notes in Computer Science*, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 232–245, 2007.

[23.] H. Poon and P. Domingos, “Sum-product networks: A new deep architecture,” in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, 2011.

[24.] T. V. H. Prathamesh, *Tensor product of matrices. Archive of Formal Proofs*. 2016. http://isa-afp.org/entries/Matrix_Tensor.shtml, Formal proof development

[25.] D. Selsam, P. Liang, and D. L. Dill, “Developing bug-free machine learning systems with formal mathematics,” *arXiv [cs.SE]*, 2017.

[26.] C. Sternagel and R. Thiemann, *Executable multivariate polynomials. Archive of Formal Proofs*. 2010. http://isa-afp.org/entries/Polynomials.shtml, Formal proof development.

[27.] R. Thiemann and A. Yamada, *Matrices, Jordan normal forms, and spectral radius theory. Archive of Formal Proofs*. 2015. http://isa-afp.org/entries/Jordan_Normal_Form.shtml, Formal proof development.

[28.] N. Tishby and N. Zaslavsky, “Deep learning and the information bottleneck principle,” in *2015 IEEE Information Theory Workshop (ITW)*, 2015.

[29.] M. Wenzel, “Isar-A generic interpretative approach to readable formal proof documents,” in *Theorem Proving in Higher Order Logic (TPHOLs ’99)*, vol. 1690, Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin-Mohring, and L. Théry, Eds. Springer, pp. 167–184, 1999.

[30.] S. Rizvi and C. Singh, “Hausdorff Measures in Caratheodary Construction, Vital Covering Theorem,” in *Steiner Symmetrization & Isodiametric Inequality"ICTSGA-1*, Taylor and Francis ISC2022: CRC Press, pp. 6–34.

[31.] S. Rizvi and C. Singh, “Disital point for Borel Measure,” vol. 12, 2024

[32.] S. Rizvi and C. Singh, “ON EXISTENCE OF THE SUPPORT OF A BOREL MEASURE AND THEIR REMARKABLE PROPERTIES,” vol. 11, 2024.