A Formal Proof of the Expressiveness of Deep Learning

Alexander Bentkamp[1,3(✉)], Jasmin Christian Blanchette[1,2], and Dietrich Klakow[3]

[1] Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
{a.bentkamp,j.c.blanchette}@vu.nl
[2] Max-Planck-Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany
jasmin.blanchette@mpi-inf.mpg.de
[3] Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany
{s8albent@stud.,dietrich.klakow@lsv.}uni-saarland.de

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.

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. [13] 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 recognition—i.e., classifying a given image in one of several categories, depending on what the image depicts. Algorithms usually learn 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 unseen input data.

CACs are based on sum–product networks (SPNs), also called arithmetic circuits [26]. An SPN is a rooted directed acyclic graph with input variables as leaf nodes and two types of inner nodes: sums and products. The incoming edges of sum nodes are labeled with real-valued weights, which are learned by training.

CACs impose the structure of the popular convolutional neural networks (CNNs) onto SPNs, 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 similarity networks, which have been demonstrated to perform as well as CNNs, if not better [12].

Cohen et al. prove two main theorems about CACs: the fundamental and the generalized theorem of network capacity (Section 3). The generalized theorem states that CAC networks enjoy complete depth efficiency: In general, to express a function captured by a deeper network using a shallower network, the shallower network must be exponentially larger than the deeper network. By "in general," we mean that the statement holds for all CACs except for a Lebesgue null set S in the weight space of the deeper network. The fundamental theorem is a special case of the generalized theorem, where the expressiveness of the deepest possible network is compared with the shallowest network. Cohen et al. present both theorems in a variant where weights are shared across the networks and a more flexible variant where they are not.

As an exercise in mechanizing modern research in machine learning, we developed a formal proof of the fundamental theorem for networks with nonshared weights using the Isabelle/HOL proof assistant [23]. To simplify our work, we recast the original proof into a more modular version (Section 4), 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 (Section 5). We created a library for tensors and their operations, including product, CP-rank, and matricization. We added the matrix rank and its properties to Thiemann and Yamada's matrix library [29], generalized the definition of the Borel measure by Hölzl and Himmelmann [16], and extended Lochbihler and Haftmann's polynomial library [15] 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 types that are too restrictive: The dimensionality of the matrices and of the measure space is parametrized by types that encode numbers, whereas we needed them to be terms.

Building on these libraries, we formalized the fundamental theorem for networks with nonshared weights (Section 6). CACs are represented using a datatype that is flexible enough to capture networks with and without concrete weights. We defined tensors and a polynomial to describe these networks, and used the datatype's induction principle to show their properties and deduce the fundamental theorem.

Our formalization is part of the *Archive of Formal Proofs* [2] and is described in more detail in Bentkamp's M.Sc. thesis [3]. It comprises about 7000 lines of Isabelle proofs, mostly in the declarative Isar style [30], and relies only on the standard axioms of higher-order logic, including the axiom of choice.

2 Mathematical Preliminaries

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 $\mathscr{A} \in \mathbb{R}^{M_1 \times \cdots \times M_N}$ of order N and dimension M_i in mode i contains values $\mathscr{A}_{d_1,\ldots,d_N} \in \mathbb{R}$ for $d_i \in \{1,\ldots,M_i\}$.

Like for vectors and matrices, addition $+$ is defined as componentwise addition for tensors of identical dimensions. The product $\otimes$ is a binary operation on two arbitrary tensors that generalizes the outer vector product. The canonical polyadic rank, or CP-rank, associates a natural number with a tensor, generalizing the matrix rank. The matricization $[\mathscr{A}]$ of a tensor $\mathscr{A}$ is a matrix obtained by rearranging $\mathscr{A}$'s entries using a bijection between the tensor and matrix entries. It has the following property:

Lemma 1 *Given a tensor $\mathscr{A}$, we have* $\operatorname{rank}[\mathscr{A}] \leq \text{CP-rank}\ \mathscr{A}$.

Lebesgue 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 a large amount of subsets of $\mathbb{R}^n$, including all closed and open sets, although it is impossible to design a measure that caters for all subsets of $\mathbb{R}^n$ 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 $\mathbb{R}^n$ except for a null set, the property is said to hold *almost everywhere*.

The following lemma [11] about polynomials will be useful for the proof of the fundamental theorem of network capacity.

Lemma 2 *If $p \not\equiv 0$ is a polynomial in d variables, the set of points $\mathbf{x} \in \mathbb{R}^d$ with $p(\mathbf{x}) = 0$ is a Lebesgue null set.*

3 The Theorems of Network Capacity

Figure 1 gives the formulas for evaluating a CAC and relates them to the network's hierarchical structure. The $*$ operator denotes componentwise multiplication. The inputs $\mathbf{x}_1, \ldots, \mathbf{x}_N$ of a CAC are N real vectors of length M, where N must be a power of 2. The output $\mathbf{y}$ is a vector of length Y. The network's depth d can be any number between 1 and $\log_2 N$. The first $d-1$ pooling layers consist of binary nodes. The last pooling layer consists of a single node with an arity of $N/2^{d-1} \geq 2$.

The calculations depend on the learned weights, which are organized as entries of a collection of real matrices $W_{l,j}$, where l is the index of the layer and j is the position in that layer where the matrix is used. Matrix $W_{l,j}$ has dimensions $r_l \times r_{l-1}$ for natural numbers $r_{-1}, \ldots, r_d$ with $r_{-1} = M$ and $r_d = Y$. The *weight space* of a CAC is the space of all possible weight configurations. For a given weight configuration, the network expresses the function $(\mathbf{x}_1, \ldots, \mathbf{x}_N) \mapsto \mathbf{y}$.

The above definitions are all we need to state the main result proved by Cohen et al.:

Theorem 3 (Generalized Theorem of Network Capacity) *Consider two CACs with identical N, M, and Y parameters: a deeper network of depth d_1 with weight matrix dimensions $r_{1,l}$ and a shallower network of depth $d_2 < d_1$ with weight matrix dimensions $r_{2,l}$. Let $r = \min\{M, r_{1,0}, \ldots, r_{1,d_2-1}\}$ and assume*

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

Let S be the set of configurations in the weight space of the deeper network that express functions also expressible by the shallower network. Then S is a Lebesgue null set.

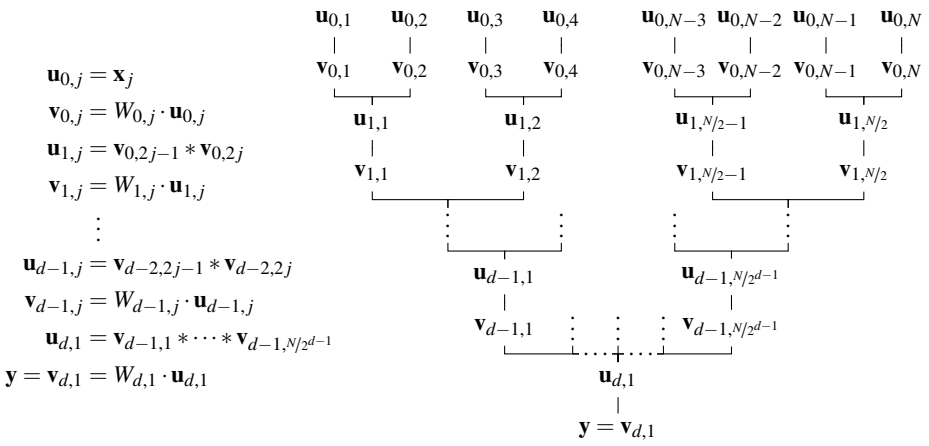


Fig. 1: Definition and hierarchical structure of a CAC with d layers

Intuitively, to express the same functions as the deeper network, almost everywhere in the weight space of the deeper network, r_{2,d_2-1} must be at least $r^{N/2^{d_2}}$, which means the shallower network needs exponentially larger weight matrices than the deeper network.

The special case of this theorem where $d_1 = \log_2 N$ and $d_2 = 1$ is called the fundamental theorem of network capacity. This is the theorem we formalized. Cohen et al. extended the result to CACs with an initial representational layer that applies a collection of nonlinearities to the inputs before the rest of the network is evaluated. Independently, they also showed that the fundamental and generalized theorems hold when the same weight matrix is applied within each layer l—i.e., $W_{l,1} = \cdots = W_{l,N/2^l}$.

4 Restructured Proof of the Theorems

The proof of either theorem of network capacity depends on a connection between CACs and measure theory, using tensors, matrices, and polynomials along the way. Briefly, the CACs and the functions they express can be described using tensors. Via matricization, these tensors can be analyzed as matrices. Polynomials bridge the gap between matrices and measure theory, since the matrix determinant is a polynomial, and zero sets of polynomials are Lebesgue null sets (Lemma 2).

The proof by Cohen et al. is structured as a monolithic induction over the deep network structure. It combines tensors, matrices, and polynomials in each induction step. Before launching Isabelle, we restructured the proof into a more modular version that cleanly separates the mathematical theories involved, resulting in the following sketch:

I. We describe the function expressed by a CAC for a fixed weight configuration using tensors. We focus on an arbitrary entry y_i of the output vector $\mathbf{y}$. If the shallower network cannot express the output component y_i, it cannot represent the entire output either. Let $\mathscr{A}_i(w)$ be the tensor that represents the function $(\mathbf{x}_1, \ldots, \mathbf{x}_N) \mapsto y_i$ expressed by the deeper network with a weight configuration w.

II. We define a function φ that reduces the order of a tensor. The CP-rank of $\varphi(\mathscr{A})$ indicates how large the shallower network must be to express a function represented by a tensor $\mathscr{A}$: If the function expressed by the shallower network is represented by $\mathscr{A}$, then $r_{2,d_2-1} \geq \text{CP-rank}\,(\varphi(\mathscr{A}))$.

III. We construct a multivariate polynomial p, mapping the weights configurations w of the deeper network to a real number $p(w)$. It has the following properties:
   (a) If $p(w) \neq 0$, then $\text{rank}\,[\varphi(\mathscr{A}_i(w))] \geq r^{N/2^{d_2}}$. Hence $\text{CP-rank}\,(\varphi(\mathscr{A}_i(w))) \geq r^{N/2^{d_2}}$ by Lemma 1.
   (b) The polynomial p is not the zero polynomial. Hence its zero set is a Lebesgue null set by Lemma 2.

By properties IIIa and IIIb, the inequation $\text{CP-rank}\,(\varphi(\mathscr{A}_i(w))) \geq r^{N/2^{d_2}}$ holds almost everywhere. By step II, almost everywhere we need

$$r_{2,d_2-1} \geq r^{N/2^{d_2}}$$

for the shallower network to express functions the deeper network expresses.

Beyond simplifying the formalization, the restructured proof allows us to state a stronger property than Cohen et al.: The set S from Theorem 3 is not only a Lebesgue null set, but also a subset of the zero set of the polynomial p. This fact can be used to derive further properties of S. Zero sets of polynomials are well studied in algebraic geometry, where they are known as algebraic varieties. This generalization partially addresses an issue that arises when applying the theorem to actual implementations of CACs: Cohen et al. assume that the weight space of the deeper network is a Euclidean space, but in practice it will always be discrete. They also show that S is a closed null set, but since these can be arbitrarily dense, this gives no information about the discrete counterpart of S.

We can estimate the size of this discrete counterpart of S using our generalization in conjunction with a result from algebraic geometry [10, 20] that allows us to estimate the size of the ε-neighborhood of the zero set of a polynomial. The ε-neighborhood of S is a good approximation of the discrete counterpart of S if ε corresponds to the precision of computer arithmetic. Unfortunately, the estimate is trivial, unless we assume $\varepsilon < 2^{-170000}$, which largely exceeds the precision of modern computers. Thus, shallow CACs are perhaps more expressive than Theorem 3 suggests. On the other hand, our analysis is built upon inequalities, which only provide an upper bound. A mathematical result estimating the size of S with a lower bound would call for an entirely different approach.

5 Formal Libraries

Matrices We had several options for the choice of a matrix library, of which the most relevant were Isabelle's analysis library and Thiemann and Yamada's matrix library [29]. The analysis library fixes the matrix dimensions using type parameters, a technique that appears to have been introduced by John Harrison. The advantage of this approach is that the dimensions are part of the type and need not be stated as conditions. Moreover, it makes it possible to instantiate type classes depending on the type arguments. However, this approach is not practical when the dimensions are specified by terms. Therefore, we chose Thiemann and Yamada's library, which uses a single type for matrices of all dimensions and includes a rich collection of lemmas.

We extended the library in a few ways. 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* $\Rightarrow$ *nat* **where**
rank A = vectorspace.dim F (span_vs (set (cols A)))

Moreover, we defined submatrices and proved that the rank of a matrix is larger than any submatrix with nonzero determinant, and that the rank is the maximum amount of linearly independent columns of the matrix.

Tensors The *Tensor* entry of the *Archive of Formal Proofs* [27] might seem to be a good starting point for a formalization of tensors. However, despite its name, this library does not contain a type for tensors. Instead, it introduces the Kronecker product, which is equivalent to the tensor product but operates on the matricizations of tensors.

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* = $\{(ds :: nat\ list, as :: \alpha\ list).\ \mathsf{length}\ as = \prod ds\}$

We formalized addition, multiplication by scalars, product, matricization, and the CP-rank. We instantiated addition as a semigroup (*semigroup_add*) and 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 $a+c=b+c \Longrightarrow a=b$.

For proving properties of addition, scalar multiplication, and product, we devised a powerful induction principle on tensors that uses tensor slices. The induction step amounts to showing a property for a tensor $\mathscr{A} \in \mathbb{R}^{M_1 \times \cdots \times M_N}$ assuming it holds for all slices $\mathscr{A}_i \in \mathbb{R}^{M_2 \times \cdots \times M_N}$, which are obtained by fixing the first index $i \in \{1,\ldots,M_1\}$.

Matricization rearranges the entries of a tensor $\mathscr{A} \in \mathbb{R}^{M_1 \times \cdots \times M_N}$ into a matrix $[\![\mathscr{A}]\!] \in \mathbb{R}^{I \times J}$. This rearrangement can be described as a bijection between $\{0,\ldots,M_1-1\} \times \cdots \times \{0,\ldots,M_N-1\}$ and $\{0,\ldots,I-1\} \times \{0,\ldots,J-1\}$, assuming that indices start at 0. The operation is parametrized by a partition of the set of tensor indices into two sets $\{r_1 < \cdots < r_K\} \uplus \{c_1 < \cdots < c_L\} = \{1,\ldots,N\}$. The proof of Theorem 3 uses only standard matricization, which partitions the indices into odd and even numbers, but we formalized the more general formulation [1]. The matrix $[\![\mathscr{A}]\!]$ has $I = \prod_{i=1}^{K} r_i$ rows and $J = \prod_{j=1}^{L} c_j$ columns. The rearrangement function is

$$(i_1,\ldots,i_N) \mapsto \left(\sum_{k=1}^{K} \left(i_{r_k} \cdot \prod_{k'=1}^{k-1} M_{r_{k'}}\right),\ \sum_{l=1}^{L} \left(i_{c_l} \cdot \prod_{l'=1}^{l-1} M_{c_{l'}}\right)\right)$$

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, respectively. This is perhaps more obvious if we expand the sum and product operators and factor out the bases M_i:

$$\begin{aligned}(i_1,\ldots,i_N) \mapsto \big(&i_{r_1} + M_{r_1} \cdot (i_{r_2} + M_{r_2} \cdot \ldots \cdot (i_{r_{K-1}} + M_{r_{K-1}} \cdot i_{r_K}) \ldots)),\\ &i_{c_1} + M_{c_1} \cdot (i_{c_2} + M_{c_2} \cdot \ldots \cdot (i_{c_{L-1}} + M_{c_{L-1}} \cdot i_{c_L}) \ldots))\big)\end{aligned}$$

To formalize the matricization operation, we defined a function calculating the digits of a number n in a given mixed-based numeral system:

```
fun encode :: nat list ⇒ nat ⇒ nat list where
  encode [] n = []
| encode (b # bs) n = (n mod b) # encode bs (n div b)
```

We then defined matricization as

```
definition matricize :: nat set ⇒ α tensor ⇒ α mat where
  matricize R 𝒜 = mat (∏ sublist (dims 𝒜) R) (∏ sublist (dims 𝒜) (−R))
    (λ(r, c). lookup 𝒜 (weave R
      (encode (sublist (dims 𝒜) R) r)
      (encode (sublist (dims 𝒜) (−R)) c)))
```

The matrix constructor mat takes as arguments the matrix dimensions and a function that returns each matrix entry given the indices r and c. Defining this function amounts to finding the corresponding indices of the tensor, which are essentially the mixed-base encoding of r and c, but the digits of these two encoded numbers must be interleaved in an order specified by the set $R = \{r_1, \ldots, r_K\}$.

To merge two lists of digits in the right way, we defined a function weave. This function is the counterpart of sublist from the standard library, which reduces a list to those entries whose indices belong to a set I:

```
lemma weave_sublists: weave I (sublist as I) (sublist as (−I)) = as
```

The main concern when defining such a function is to determine how it should behave in corner cases—in our scenario, when $I = \{\}$ and $xs \neq []$. We settled on a definition such that the property length (weave I xs ys) = length xs + length ys holds unconditionally:

```
definition weave :: nat set ⇒ α list ⇒ α list ⇒ α list where
  weave I xs ys = map (λi. if i∈I then xs ! |{a∈I. a < i}| else ys ! |{a∈−I. a < i}|)
    [0 ..< length xs + length ys]
```

(The ! operator returns the list element at a given index.) This definition allows us to prove lemmas about weave I xs ys ! a and length (weave I xs ys) very easily. Other properties, such as the *weave_sublists* lemma above, are justified using an induction over the length of a list, with a case distinction in the induction step on whether the new list element is taken from xs or ys.

Another difficulty arises with the rule rank $[\mathscr{A} \otimes \mathscr{B}]$ = rank $[\mathscr{A}]$ · rank $[\mathscr{B}]$ for standard matricization and tensors of even order, which seemed tedious to formalize. Restructuring the proof eliminates one of its two occurrences (Section 4). The remaining occurrence is used to show that rank $[\mathbf{a}_1 \otimes \cdots \otimes \mathbf{a}_N] = 1$, where $\mathbf{a}_1, \ldots, \mathbf{a}_N$ are vectors and N is even. A simpler proof relies on the observation that the entries of $[\mathbf{a}_1 \otimes \cdots \otimes \mathbf{a}_N]$ can be written as $f(i) \cdot g(j)$, where f depends only on the row index i, and g depends only on the column index j. Using this argument, rank $[\mathbf{a}_1 \otimes \cdots \otimes \mathbf{a}_N] = 1$ can be shown for generalized matricization and an arbitrary N, which we used to prove Lemma 1:

```
lemma matrix_rank_le_cp_rank:
  fixes A :: (α :: field) tensor
  shows mrank (matricize R A) ≤ cprank A
```

Lebesgue Measure Isabelle's analysis library defines the Borel measure on $\mathbb{R}^n$ 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 has more measurable sets. The proof by Cohen et al. allows us to show that the set S defined in Theorem 3 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) formulate the theorem using the almost-everywhere quantifier ($\forall_{\text{ae}}$) instead of the null set predicate.

We chose the third approach, because it seemed simpler. Theorem 3, as expressed in Section 3, defines the set S as set of configurations in the weight space of the deeper network that express functions also expressible by the shallower network, and then states that S is a null set. In the formalization, we state it as follows: Almost everywhere in the weight space of the deeper network, the deeper network expresses functions 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* **where**
  lborel = distr ($\prod_{\mathsf{M}} b \in$ Basis. interval_measure ($\lambda x.\ x$)) borel
    ($\lambda f.\ \sum b \in$ Basis. $f\ b *_{\mathsf{R}} b$)

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 provide a new definition of the Borel measure whose type does not fix the number of dimensions. The multidimensional Borel measure is the product measure ($\prod_{\mathsf{M}}$) of the one-dimensional Borel measure (lborel :: *real measure*) with itself:

definition $\text{lborel}_{\mathsf{f}}$ :: *nat* $\Rightarrow$ (*nat* $\Rightarrow$ *real*) *measure* **where**
  $\text{lborel}_{\mathsf{f}}\ n = (\prod_{\mathsf{M}} b \in \{..<n\}.$ lborel)

The argument n specifies the dimension of the measure space. Unlike with lborel, the measure space of $\text{lborel}_{\mathsf{f}}\ n$ is not the entire universe of the type: Only functions that map to a default value for numbers $\geq n$ are contained in the measure space, which is available as space ($\text{lborel}_{\mathsf{f}}\ n$). With the above definition, we could prove the main lemmas about $\text{lborel}_{\mathsf{f}}$ from the corresponding lemmas about lborel with little effort.

Multivariate Polynomials Multivariate polynomial libraries have been developed to support other formalization projects in Isabelle. Sternagel and Thiemann [28] 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 [17] 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 [15]. We derived induction principles and properties of the evaluation homomorphism and of nested multivariate polynomials. These were useful to formalize Lemma 2:

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

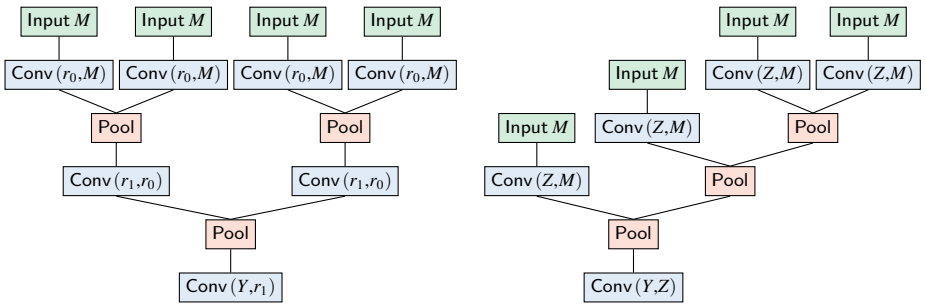
The type variable α can be used to store weights. For networks without weights, it is set to $nat \times nat$, which associates only the matrix dimension with each Conv node. For networks with weights, α is *real mat*, an actual matrix. These two network types are connected by insert_weights :: $(nat \times nat)\ cac \Rightarrow (nat \Rightarrow real) \Rightarrow real\ mat\ cac$, which inserts weights into a weightless network. The weights are specified by the second argument f, of which only the first values $f\,0$, $f\,1$, …, $f\,(k-1)$ are used, until the necessary number of weights, k, is reached. Sets over $nat \Rightarrow real$ can be measured using lborel$_f$.

The following function describes how the networks are evaluated, where $\otimes_{mv}$ multiplies a matrix with a vector and component_mult multiplies vectors componentwise:

fun evaluate_net :: *real mat cac* $\Rightarrow$ *real vec list* $\Rightarrow$ *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))

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 of network capacities, 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$_0$ :: $nat \Rightarrow nat\ list \Rightarrow (nat \times nat)\ cac$ **and**
  deep_model :: $nat \Rightarrow nat \Rightarrow nat\ list \Rightarrow (nat \times nat)\ cac$
where



(a) deep_model Y r_1 $[r_0, M]$ (b) shallow_model Y Z M 3

Fig. 2: A deep and a shallow network represented using the *cac* datatype

$\mathsf{deep\_model}_0\ Y\ [] = \mathsf{Input}\ Y$
$\mid \mathsf{deep\_model}_0\ Y\ (r\ \#\ rs) = \mathsf{Pool}\ (\mathsf{deep\_model}\ Y\ r\ rs)\ (\mathsf{deep\_model}\ Y\ r\ rs)$
$\mid \mathsf{deep\_model}\ Y\ r\ rs = \mathsf{Conv}\ (Y,r)\ (\mathsf{deep\_model}_0\ r\ rs)$

fun $\mathsf{shallow\_model}_0 :: nat \Rightarrow nat \Rightarrow nat \Rightarrow (nat \times nat)\ cac$ **where**
$\mathsf{shallow\_model}_0\ Z\ M\ 0 = \mathsf{Conv}\ (Z,M)\ (\mathsf{Input}\ M)$
$\mid \mathsf{shallow\_model}_0\ Z\ M\ (\mathsf{Suc}\ N) =$
$\mathsf{Pool}\ (\mathsf{shallow\_model}_0\ Z\ M\ 0)\ (\mathsf{shallow\_model}_0\ Z\ M\ N)$

definition $\mathsf{shallow\_model} :: nat \Rightarrow nat \Rightarrow nat \Rightarrow nat \Rightarrow (nat \times nat)\ cac$ **where**
$\mathsf{shallow\_model}\ Y\ Z\ M\ N = \mathsf{Conv}\ (Y,Z)\ (\mathsf{shallow\_model}_0\ Z\ M\ N)$

Two examples are given in Figure 2. For the deep model, the arguments Y # r # rs correspond to the weight matrix sizes $[r_{1,d}\,(=Y), r_{1,d-1}, \ldots, r_{1,0}, r_{1,-1}\,(=M)]$. 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.

The rest of the formalization follows the proof sketch presented in Section 4.

Step I The following operation computes a list, or *vec*tor, of tensors representing a network's function, each tensor standing for one component of the output vector:

fun $\mathsf{tensors\_from\_net} :: real\ mat\ cac \Rightarrow real\ tensor\ vec$ **where**
$\mathsf{tensors\_from\_net}\ (\mathsf{Input}\ M) = \mathsf{Matrix.vec}\ M\ (\lambda i.\ \mathsf{unit\_vec}\ M\ i)$
$\mid \mathsf{tensors\_from\_net}\ (\mathsf{Conv}\ A\ m) =$
$\mathsf{mat\_tensorlist\_mult}\ A\ (\mathsf{tensors\_from\_net}\ m)\ (\mathsf{input\_sizes}\ m)$
$\mid \mathsf{tensors\_from\_net}\ (\mathsf{Pool}\ m_1\ m_2) =$
$\mathsf{component\_mult}\ (\mathsf{tensors\_from\_net}\ m_1)\ (\mathsf{tensors\_from\_net}\ m_2)$

For an Input node, we return the list of unit vectors of length M. For a Conv node, we multiply the weight matrix A with the tensor list computed for the subnetwork m, using matrix–vector multiplication. For a Pool node, we compute, elementwise, the tensor products of the two tensor lists associated with the subnetworks m_1 and m_2. If two networks express the same function, the representing tensors are the same:

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 $\forall is$. input_correct $is \longrightarrow$ evaluate_net m_1 is = evaluate_net m_2 is
shows tensors_from_net m_1 = tensors_from_net m_2

The fundamental theorem fixes an arbitrary deep network. It is useful to fix the deep network parameters in a locale—a sectioning mechanism that fixes variables and assumptions on them across definitions and lemmas:

locale deep_model_correct_params =
fixes rs :: *nat list*
assumes *deep*: length rs ≥ 3
and *no_zeros*: $\bigwedge r.\ r \in$ set rs $\Longrightarrow r > 0$

The list rs completely specifies one specific deep network model:

abbreviation deep_net = deep_model (rs ! 0) (rs ! 1) (tl (tl rs))

The other parameters of the deep network can be defined based on rs:

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

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

abbreviation 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 do so using a second locale that introduces a constant i for i.

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

Then we can define the tensor $\mathscr{A}_i$, 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 $\mathscr{A}_{\text{i}}$ w = tensors_from_net (insert_weights deep_net w) ! i

We want to analyze 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 $\mathscr{A}$ represents the function expressed by the shallow network, then $r_{2,d_2-1} \geq$ CP-rank $(\varphi(\mathscr{A}))$. For the fundamental theorem of network capacity, φ is the identity and $d_2 = 1$. Hence, it suffices to prove that $Z = r_{2,0} \geq$ CP-rank $(\mathscr{A})$:

lemma *cprank_shallow_model*:
cprank (tensors_from_net (insert_weights w (shallow_net Z)) ! i) $\leq Z$

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 $\mathsf{p_{func}}\ w = \mathsf{det}\ (\mathsf{submatrix}\ [\mathscr{A}_\mathsf{i}\ w]\ \mathsf{rows\_with\_1}\ \mathsf{rows\_with\_1})$

where $[\mathscr{A}_\mathsf{i}\ w]$ abbreviates the standard matricization $\mathsf{matricize}\ \{n.\ \mathsf{even}\ n\}\ (\mathscr{A}_\mathsf{i}\ w)$, and rows_with_1 is the set of row indices with 1s in the main diagonal for a specific weight configuration w that will be defined in Step IIIb. We try to make the submatrix as large as possible while maintaining the 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 $\mathsf{p_{func}}$ must be shown to be a polynomial function. We introduce a predicate polyfun, which is true if a function is a polynomial function:

definition $\mathsf{polyfun}\ N\ f = (\exists p.\ \mathsf{vars}\ p \subseteq N \wedge (\forall x.\ \mathsf{insertion}\ x\ p = f\ x))$

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*: $\mathsf{polyfun}\ \{..<\mathsf{weight\_space\_dim}\}\ \mathsf{p_{func}}$

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

lemma *if_polynomial_0_rank*:
  assumes $\mathsf{p_{func}}\ w \neq 0$
  shows $\mathsf{r^{N\_half}} \leq \mathsf{cprank}\ (\mathscr{A}_\mathsf{i}\ w)$
proof −
  have $r^{\mathsf{N\_half}} = \mathsf{dim_r}\ (\mathsf{submatrix}\ [\mathscr{A}_\mathsf{i}\ w]\ \mathsf{rows\_with\_1}\ \mathsf{rows\_with\_1})$
    by calculating the size of the submatrix
  also have $\cdots \leq \mathsf{mrank}\ [\mathscr{A}_\mathsf{i}\ w]$
    using the assumption and the fact that the rank is higher than a submatrix with nonzero determinant
  also have $\cdots \leq \mathsf{cprank}\ (\mathscr{A}_\mathsf{i}\ w)$
    using Lemma 1
  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 main diagonal and 0s elsewhere (eye_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**
  $\mathsf{copy\_first\_matrix}\ nr\ nc = \mathsf{mat}\ nr\ nc\ (\lambda(r,c).\ \text{if } c = 0 \text{ then } 1 \text{ else } 0)$

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

lemma *mult_copy_first_matrix*:
  assumes $i < nr$ **and** $\mathsf{dim_v}\ v > 0$
  shows $(\mathsf{copy\_first\_matrix}\ nr\ (\mathsf{dim_v}\ v) \otimes_{\mathsf{mv}} v)\ !\ i = v\ !\ 0$

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*
where
witness$_0$ *Y* [] = Input *Y*
| witness$_0$ *Y* (*r* # *rs*) = Pool (witness *Y r rs*) (witness *Y r rs*)
| witness *Y r rs* = Conv ((if length *rs* = 0 then eye_matrix else
if length *rs* = 1 then all1_matrix else copy_first_matrix) *Y r*) (witness$_0$ *r 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, such 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.

The witness weights can be extracted from the witness network as follows:

definition witness_weights :: *nat* ⇒ *real* **where**
witness_weights =
(*εw*. witness (rs ! 0) (rs ! 1) (tl (tl rs)) = insert_weights deep_net *w*)

This could also be achieved without using Hilbert's choice operator, by defining a recursive function that extracts the weights from weighted networks.

We prove that the representing tensor of the witness network, which is equal to $\mathscr{A}_{\mathsf{i}}$ 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 $\mathsf{r}^{\mathsf{N\_half}}$:

lemma *witness_submatrix*:
submatrix [$\mathscr{A}_{\mathsf{i}}$ witness_weights] rows_with_1 rows_with_1 =
eye_matrix $\mathsf{r}^{\mathsf{N\_half}}$ $\mathsf{r}^{\mathsf{N\_half}}$

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

lemma *polynomial_not_zero*: $\mathsf{p}_{\mathsf{func}}$ witness_weights ≠ 0

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

theorem *fundamental_theorem_of_network_capacity*:
$\forall_{\mathsf{ae}}\, w_{\mathrm{d}}$ w.r.t. $\mathsf{lborel}_{\mathsf{f}}$ weight_space_dim. $\nexists w_{\mathrm{s}}\, Z$.
$Z < \mathsf{r}^{\mathsf{N\_half}} \wedge$
∀*is*. input_correct *is* ⟶
evaluate_net (insert_weights deep_net w_{d}) *is* =
evaluate_net (insert_weights (shallow_net Z) w_{s}) *is*

The $r^{\mathsf{N\_half}}$ bound corresponds to the size of the identity matrix in *witness_submatrix*.

The theorem statement is independent of the tensor library, and is therefore correct regardless of whether the library faithfully captures tensor-related notions.

7 Discussion and Related Work

Sledgehammer and SMT To discharge proof obligations, we used Sledgehammer [24] extensively. This Isabelle tool invokes external automatic theorem provers and, in case of success, returns short Isar proof texts that can be inserted in the formalization. In the best-case scenario, Sledgehammer quickly produces a one-line proof text consisting of an invocation of the *metis* proof method [25], Isabelle's internal superposition prover. Unfortunately, Sledgehammer sometimes returns only cryptic structured Isar proofs [7] or, if all else fails, proofs that depend on the *smt* method [9].

The *smt* method relies on the SMT solver Z3 [21] to find a proof, which it then replays using Isabelle's inference kernel. Relying on a highly heuristic third-party prover is fragile; some proofs that are fast with a given version of the prover might time out with a different version, or be unreplayable due to some incompleteness in *smt*. As a result, entries in the *Archive of Formal Proofs* cannot use it. Sledgehammer often generates *smt* proofs, especially in goals about sums and products of reals, existential quantifiers, and λ-expressions. We ended up with over 60 invocations of *smt*, which we later replaced one by one with structured Isar proofs, a tedious process. The following equation on reals is an example that can only be proved by *smt*, with suitable lemmas:

$$\sum_{i\in I}\sum_{j\in J} a\cdot b\cdot f(i)\cdot g(j) = \Big(\sum_{i\in I} a\cdot f(i)\Big)\cdot\Big(\sum_{j\in J} b\cdot g(j)\Big)$$

We could not solve it with other proof methods without engaging in a detailed proof involving multiple steps. This particular example relies on *smt*'s partial support for λ-expressions through λ-lifting, an instance of what we would call "easy higher-order."

Similar Results for Other Deep Learning Architectures CACs are relatively easy to analyze but little used in practice. In a follow-up paper [14], Cohen et al. connected their tensor analysis of CACs to the frequently used CNNs with rectified linear unit (ReLU) activation. Unlike CACs, ReLU CNNs with average pooling are not universal—that is, even shallow networks of arbitrary size cannot express all functions a deeper network can express. Moreover, ReLU CNNs do not enjoy complete depth efficiency; the analogue of the set S for those networks has a Lebesgue measure greater than zero. This leads Cohen et al. to conjecture that CACs could become a leading approach for deep learning, once suitable training algorithms have been developed.

Related Formal Proofs We are aware of a few other formalizations of machine learning algorithms, including hidden Markov models [19], perceptrons [22], expectation maximization, and support vector machines [6]. To our knowledge, our work is the first formalization about deep learning.

Some of the mathematical libraries underlying our formalizations have counterparts in other systems, notably Coq. For example, the Mathematical Components include comprehensive matrix theories [5], which are naturally expressed using dependent types. The tensor formalization by Boender [8] restricts itself to the Kronecker product on matrices. Bernard et al. [4] formalized multivariate polynomials and used them to show the transcendence of e and π. Kam formalized the Lebesgue measure as part of a formalization of the Lebesgue integral, which in turn was used to state and prove Markov's inequality [18].

8 Conclusion

We applied a proof assistant to formalize a recent result 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 fairly 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 lead 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.

Acknowledgment We thank Robert Lewis, Anders Schlichtkrull, and Mark Summerfield for suggesting many textual improvements. The work has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation program (grant agreement No. 713999, Matryoshka).

References

[1] Bader, B.W., Kolda, T.G.: Algorithm 862: MATLAB tensor classes for fast algorithm prototyping. ACM Trans. Math. Softw. 32(4), 635–653 (2006)

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

[3] Bentkamp, A.: An Isabelle Formalization of the Expressiveness of Deep Learning. M.Sc. thesis, Universität des Saarlandes (2016)

[4] Bernard, S., Bertot, Y., Rideau, L., Strub, P.: Formal proofs of transcendence for e and π as an application of multivariate and symmetric polynomials. In: Avigad, J., Chlipala, A. (eds.) Certified Programs and Proofs (CPP 2016). pp. 76–87. ACM (2016)

[5] Bertot, Y., Gonthier, G., Biha, S.O., Pasca, I.: Canonical big operators. In: Mohamed, O.A., Muñoz, C.A., Tahar, S. (eds.) Theorem Proving in Higher Order Logics (TPHOLs 2008). vol. 5170, pp. 86–101. Springer (2008)

[6] Bhat, S.: Syntactic foundations for machine learning. Ph.D. thesis, Georgia Institute of Technology (2013)

[7] Blanchette, J.C., Böhme, S., Fleury, M., Smolka, S.J., Steckermeier, A.: Semi-intelligible Isar proofs from machine-generated proofs. J. Autom. Reasoning 56(2), 155–200 (2016)

[8] Boender, J., Kammüller, F., Nagarajan, R.: Formalization of quantum protocols using Coq. In: Heunen, C., Selinger, P., Vicary, J. (eds.) Workshop on Quantum Physics and Logic (QPL 2015). EPTCS, vol. 195, pp. 71–83 (2015)

[9] Böhme, S., Weber, T.: Fast LCF-style proof reconstruction for Z3. In: Kaufmann, M., Paulson, L.C. (eds.) Interactive Theorem Proving (ITP 2010). LNCS, vol. 6172, pp. 179–194. Springer (2010)

[10] Bürgisser, P., Cucker, F., Lotz, M.: The probability that a slightly perturbed numerical analysis problem is difficult. Math. Comput. 77(263), 1559–1583 (2008)

[11] Caron, R., Traynor, T.: The zero set of a polynomial. Tech. rep., University of Windsor (2005)

[12] Cohen, N., Sharir, O., Shashua, A.: Deep SimNets. In: Computer Vision and Pattern Recognition (CVPR 2016). pp. 4782–4791. IEEE Computer Society (2016)

[13] Cohen, N., Sharir, O., Shashua, A.: On the expressive power of deep learning: A tensor analysis. In: Feldman, V., Rakhlin, A., Shamir, O. (eds.) Conference on Learning Theory (COLT 2016). JMLR Workshop and Conference Proceedings, vol. 49, pp. 698–728. JMLR.org (2016)
[14] Cohen, N., Shashua, A.: Convolutional rectifier networks as generalized tensor decompositions. In: Balcan, M., Weinberger, K.Q. (eds.) International Conference on Machine Learning (ICML 2016). JMLR Workshop and Conference Proceedings, vol. 48, pp. 955–963. JMLR.org (2016)
[15] Haftmann, F., Lochbihler, A., Schreiner, W.: Towards abstract and executable multivariate polynomials in Isabelle. In: Nipkow, T., Paulson, L., Wenzel, M. (eds.) Isabelle Workshop 2014 (2014)
[16] Hölzl, J., Heller, A.: Three chapters of measure theory in Isabelle/HOL. In: van Eekelen, M.C.J.D., Geuvers, H., Schmaltz, J., Wiedijk, F. (eds.) Interactive Theorem Proving (ITP 2011), LNCS, vol. 6898, pp. 135–151. Springer (2011)
[17] Immler, F., Maletzky, A.: Gröbner bases theory. Archive of Formal Proofs (2016), http://isa-afp.org/entries/Groebner_Bases.shtml, Formal proof development
[18] Kam, R.: Case Studies in Proof Checking. Master's thesis, San Jose State University (2007), http://scholarworks.sjsu.edu/cgi/viewcontent.cgi?context=etd_projects&article=1149
[19] Liu, L., Aravantinos, V., Hasan, O., Tahar, S.: On the formal analysis of HMM using theorem proving. In: Merz, S., Pang, J. (eds.) International Conference on Formal Engineering Methods (ICFEM 2014). LNCS, vol. 8829, pp. 316–331. Springer (2014)
[20] Lotz, M.: On the volume of tubular neighborhoods of real algebraic varieties. Proc. Amer. Math. Soc. 143(5), 1875–1889 (2015)
[21] de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer (2008)
[22] Murphy, T., Gray, P., Stewart, G.: Certified convergent perceptron learning, http://oucsace.cs.ohiou.edu/~gstewart/papers/coqperceptron.pdf, Unpublished draft
[23] Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
[24] Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: Sutcliffe, G., Schulz, S., Ternovska, E. (eds.) IWIL-2010. EPiC, vol. 2, pp. 1–11. EasyChair (2012)
[25] Paulson, L.C., Susanto, K.W.: Source-level proof reconstruction for interactive theorem proving. In: Schneider, K., Brandt, J. (eds.) Theorem Proving in Higher Order Logics (TPHOLs 2007). LNCS, vol. 4732, pp. 232–245. Springer (2007)
[26] Poon, H., Domingos, P.M.: Sum–product networks: A new deep architecture. In: Cozman, F.G., Pfeffer, A. (eds.) Uncertainty in Artificial Intelligence (UAI 2011). pp. 337–346. AUAI Press (2011)
[27] Prathamesh, T.V.H.: Tensor product of matrices. Archive of Formal Proofs (2016), http://isa-afp.org/entries/Matrix_Tensor.shtml, Formal proof development
[28] Sternagel, C., Thiemann, R.: Executable multivariate polynomials. Archive of Formal Proofs (2010), http://isa-afp.org/entries/Polynomials.shtml, Formal proof development
[29] Thiemann, R., Yamada, A.: 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
[30] Wenzel, M.: Isar—A generic interpretative approach to readable formal proof documents. In: Bertot, Y., Dowek, G., Hirschowitz, A., Paulin-Mohring, C., Théry, L. (eds.) Theorem Proving in Higher Order Logics (TPHOLs '99). LNCS, vol. 1690, pp. 167–184. Springer (1999)