

# Expressiveness of Deep Learning

Alexander Bentkamp

December 7, 2022

## Abstract

Deep learning has had a profound impact on computer science in recent years, with applications to search engines, image recognition and language processing, bioinformatics, and more. Recently, Cohen et al. [2] provided theoretical evidence for the superiority of deep learning over shallow learning. For my master's thesis [1], I formalized their mathematical proof using Isabelle/HOL. This formalization simplifies and generalizes the original proof, while working around the limitations of the Isabelle type system. To support the formalization, I developed reusable libraries of formalized mathematics, including results about the matrix rank, the Lebesgue measure, and multivariate polynomials, as well as a library for tensor analysis.

## Contents

<b>1</b>	<b>Tensor</b>	<b>2</b>
<b>2</b>	<b>Subtensors</b>	<b>5</b>
<b>3</b>	<b>Tensor Addition</b>	<b>7</b>
<b>4</b>	<b>Tensor Scalar Multiplication</b>	<b>10</b>
<b>5</b>	<b>Tensor Product</b>	<b>11</b>
<b>6</b>	<b>Unit Vectors as Tensors</b>	<b>13</b>
<b>7</b>	<b>Tensor CP-Rank</b>	<b>14</b>
<b>8</b>	<b>Tensor Matricization</b>	<b>16</b>
<b>9</b>	<b>CP-Rank and Matrix Rank</b>	<b>18</b>
<b>10</b>	<b>Matrix to Vector Conversion</b>	<b>19</b>
<b>11</b>	<b>Deep Learning Networks</b>	<b>20</b>

12 Concrete Matrices	24
13 Missing Lemmas of Finite_Set	26
14 Deep Network Model	26
15 Polynomials representing the Deep Network Model	33
16 Alternative Lebesgue Measure Definition	35
17 Lebesgue Measure of Polynomial Zero Sets	37
18 Shallow Network Model	37
19 Fundamental Theorem of Network Capacity	38

## 1 Tensor

```
theory Tensor
imports Main
begin
```

```
typedef 'a tensor = {t::nat list × 'a list. length (snd t) = prod-list (fst t)}
⟨proof⟩
```

```
definition dims::'a tensor ⇒ nat list where
  dims A = fst (Rep-tensor A)
```

```
definition vec::'a tensor ⇒ 'a list where
  vec A = snd (Rep-tensor A)
```

```
definition tensor-from-vec::nat list ⇒ 'a list ⇒ 'a tensor where
  tensor-from-vec d v = Abs-tensor (d,v)
```

```
lemma
assumes length v = prod-list d
shows dims-tensor[simp]: dims (tensor-from-vec d v) = d
and vec-tensor[simp]: vec (tensor-from-vec d v) = v
⟨proof⟩
```

```
lemma tensor-from-vec-simp[simp]: tensor-from-vec (dims A) (vec A) = A
⟨proof⟩
```

```
lemma length-vec: length (vec A) = prod-list (dims A)
⟨proof⟩
```

```
lemma tensor-eqI[intro]:
```

**assumes**  $\text{dims } A = \text{dims } B$  **and**  $\text{vec } A = \text{vec } B$   
**shows**  $A=B$   
 $\langle \text{proof} \rangle$

**abbreviation**  $\text{order}::'a \text{ tensor} \Rightarrow \text{nat}$  **where**  
 $\text{order } t == \text{length } (\text{dims } t)$

**inductive**  $\text{valid-index}::\text{nat list} \Rightarrow \text{nat list} \Rightarrow \text{bool}$  (**infix**  $\triangleleft 50$ ) **where**  
 $\text{Nil}: [] \triangleleft [] \mid$   
 $\text{Cons}: is \triangleleft ds \Longrightarrow i < d \Longrightarrow i \# is \triangleleft d \# ds$

**inductive-cases**  $\text{valid-indexE}[\text{elim}]: is \triangleleft ds$   
**inductive-cases**  $\text{valid-index-dimsE}[\text{elim}]: is \triangleleft \text{dims } A$

**lemma**  $\text{valid-index-length}: is \triangleleft ds \Longrightarrow \text{length } is = \text{length } ds$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{valid-index-lt}: is \triangleleft ds \Longrightarrow m < \text{length } ds \Longrightarrow is!m < ds!m$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{valid-indexI}$ :  
**assumes**  $\text{length } is = \text{length } ds$  **and**  $\bigwedge m. m < \text{length } ds \Longrightarrow is!m < ds!m$   
**shows**  $is \triangleleft ds$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{valid-index-append}$ :  
**assumes**  $is1\text{-valid}:is1 \triangleleft ds1$  **and**  $is2\text{-valid}:is2 \triangleleft ds2$   
**shows**  $is1 @ is2 \triangleleft ds1 @ ds2$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{valid-index-list-all2-iff}: is \triangleleft ds \longleftrightarrow \text{list-all2 } (<) is ds$   
 $\langle \text{proof} \rangle$

**definition**  $\text{fixed-length-sublist}::'a \text{ list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ list}$  **where**  
 $\text{fixed-length-sublist } xs \ l \ i = (\text{take } l \ (\text{drop } (l*i) \ xs))$

**fun**  $\text{lookup-base}::\text{nat list} \Rightarrow 'a \text{ list} \Rightarrow \text{nat list} \Rightarrow 'a$  **where**  
 $\text{lookup-base-Nil}: \text{lookup-base } [] \ v \ [] = \text{hd } v \mid$   
 $\text{lookup-base-Cons}: \text{lookup-base } (d \# ds) \ v \ (i \# is) =$   
 $\text{lookup-base } ds \ (\text{fixed-length-sublist } v \ (\text{prod-list } ds) \ i) \ is$

**definition**  $\text{lookup}::'a \text{ tensor} \Rightarrow \text{nat list} \Rightarrow 'a$  **where**  
 $\text{lookup } A = \text{lookup-base } (\text{dims } A) \ (\text{vec } A)$

**fun**  $\text{tensor-vec-from-lookup}::\text{nat list} \Rightarrow (\text{nat list} \Rightarrow 'a) \Rightarrow 'a \text{ list}$  **where**  
 $\text{tensor-vec-from-lookup-Nil}: \text{tensor-vec-from-lookup } [] \ e = [e \ []] \mid$   
 $\text{tensor-vec-from-lookup-Cons}: \text{tensor-vec-from-lookup } (d \# ds) \ e = \text{concat } (\text{map}$   
 $(\lambda i. \text{tensor-vec-from-lookup } ds \ (\lambda is. e \ (i \# is))) \ [0..<d])$

**definition** *tensor-from-lookup*:: $\text{nat list} \Rightarrow (\text{nat list} \Rightarrow 'a) \Rightarrow 'a$  **tensor where**  
*tensor-from-lookup ds e = tensor-from-vec ds (tensor-vec-from-lookup ds e)*

**lemma** *concat-parts-leq*:

**assumes**  $a * d \leq \text{length } v$

**shows**  $\text{concat } (\text{map } (\text{fixed-length-sublist } v \ d) \ [0..<a]) = \text{take } (a*d) \ v$

*<proof>*

**lemma** *concat-parts-eq*:

**assumes**  $a * d = \text{length } v$

**shows**  $\text{concat } (\text{map } (\text{fixed-length-sublist } v \ d) \ [0..<a]) = v$

*<proof>*

**lemma** *tensor-lookup-base*:

**assumes**  $\text{length } v = \text{prod-list } ds$

**and**  $\bigwedge is. is \triangleleft ds \implies \text{lookup-base } ds \ v \ is = e \ is$

**shows**  $\text{tensor-vec-from-lookup } ds \ e = v$

*<proof>*

**lemma** *tensor-lookup*:

**assumes**  $\bigwedge is. is \triangleleft \text{dims } A \implies \text{lookup } A \ is = e \ is$

**shows**  $\text{tensor-from-lookup } (\text{dims } A) \ e = A$

*<proof>*

**lemma** *concat-equal-length*:

**assumes**  $\bigwedge xs. xs \in \text{set } xss \implies \text{length } xs = l$

**shows**  $\text{length } (\text{concat } xss) = \text{length } xss * l$

*<proof>*

**lemma** *concat-equal-length-map*:

**assumes**  $\bigwedge i. i < a \implies \text{length } (f \ i) = d$

**shows**  $\text{length } (\text{concat } (\text{map } (\lambda i. f \ i) \ [0..<a])) = a*d$

*<proof>*

**lemma** *concat-parts*:

**assumes**  $\bigwedge xs. xs \in \text{set } xss \implies \text{length } xs = d$  **and**  $i < \text{length } xss$

**shows**  $\text{fixed-length-sublist } (\text{concat } xss) \ d \ i = xss \ ! \ i$

*<proof>*

**lemma** *concat-parts'*:

**assumes**  $\bigwedge i. i < a \implies \text{length } (f \ i) = d$

**and**  $i < a$

**shows**  $\text{fixed-length-sublist } (\text{concat } (\text{map } (\lambda i. f \ i) \ [0..<a])) \ d \ i = f \ i$

*<proof>*

**lemma** *length-tensor-vec-from-lookup*:

$\text{length } (\text{tensor-vec-from-lookup } ds \ e) = \text{prod-list } ds$

*<proof>*

**lemma** *lookup-tensor-vec*:  
**assumes**  $is \triangleleft ds$   
**shows**  $lookup\text{-base } ds \ (tensor\text{-vec-from-lookup } ds \ e) \ is = e \ is$   
 $\langle proof \rangle$

**lemma** *lookup-tensor-from-lookup*:  
**assumes**  $is \triangleleft ds$   
**shows**  $lookup \ (tensor\text{-from-lookup } ds \ e) \ is = e \ is$   
 $\langle proof \rangle$

**lemma** *dims-tensor-from-lookup*:  $dims \ (tensor\text{-from-lookup } ds \ e) = ds$   
 $\langle proof \rangle$

**lemma** *tensor-lookup-cong*:  
**assumes**  $tensor\text{-from-lookup } ds \ e_1 = tensor\text{-from-lookup } ds \ e_2$   
**and**  $is \triangleleft ds$   
**shows**  $e_1 \ is = e_2 \ is \ \langle proof \rangle$

**lemma** *tensor-from-lookup-eqI*:  
**assumes**  $\bigwedge is. is \triangleleft ds \implies e_1 \ is = e_2 \ is$   
**shows**  $tensor\text{-from-lookup } ds \ e_1 = tensor\text{-from-lookup } ds \ e_2$   
 $\langle proof \rangle$

**lemma** *tensor-lookup-eqI*:  
**assumes**  $dims \ A = dims \ B$  **and**  $\bigwedge is. is \triangleleft (dims \ A) \implies lookup \ A \ is = lookup \ B \ is$   
**shows**  $A = B \ \langle proof \rangle$

**end**

## 2 Subtensors

**theory** *Tensor-Subtensor*  
**imports** *Tensor*  
**begin**

**definition** *subtensor*:: $'a \ tensor \Rightarrow nat \Rightarrow 'a \ tensor$  **where**  
 $subtensor \ A \ i = tensor\text{-from-vec} \ (tl \ (dims \ A)) \ (fixed\text{-length-sublist} \ (vec \ A) \ (prod\text{-list} \ (tl \ (dims \ A)))) \ i$

**definition** *subtensor-combine*:: $nat \ list \Rightarrow 'a \ tensor \ list \Rightarrow 'a \ tensor$  **where**  
 $subtensor\text{-combine} \ ds \ As = tensor\text{-from-vec} \ (length \ As \ \# \ ds) \ (concat \ (map \ vec \ As))$

**lemma** *length-fixed-length-sublist[simp]*:  
**assumes**  $(Suc \ i) * l \leq length \ xs$   
**shows**  $length \ (fixed\text{-length-sublist} \ xs \ l \ i) = l$   
 $\langle proof \rangle$

**lemma** *vec-subtensor*[simp]:  
**assumes**  $\text{dims } A \neq []$  **and**  $i < \text{hd } (\text{dims } A)$   
**shows**  $\text{vec } (\text{subtensor } A \ i) = \text{fixed-length-sublist } (\text{vec } A) (\text{prod-list } (\text{tl } (\text{dims } A))) \ i$   
 ⟨proof⟩

**lemma** *dims-subtensor*[simp]:  
**assumes**  $\text{dims } A \neq []$  **and**  $i < \text{hd } (\text{dims } A)$   
**shows**  $\text{dims } (\text{subtensor } A \ i) = \text{tl } (\text{dims } A)$   
 ⟨proof⟩

**lemma** *subtensor-combine-subtensor*[simp]:  
**assumes**  $\text{dims } A \neq []$   
**shows**  $\text{subtensor-combine } (\text{tl } (\text{dims } A)) (\text{map } (\text{subtensor } A) [0..\text{hd } (\text{dims } A)]) = A$   
 ⟨proof⟩

**lemma**  
**assumes**  $\bigwedge A. A \in \text{set } As \implies \text{dims } A = ds$   
**shows** *subtensor-combine-dims*[simp]:  $\text{dims } (\text{subtensor-combine } ds \ As) = \text{length } As$   
 $\# \ ds$  (is ?D)  
**and** *subtensor-combine-vec*[simp]:  $\text{vec } (\text{subtensor-combine } ds \ As) = \text{concat } (\text{map } \text{vec } As)$  (is ?V)  
 ⟨proof⟩

**lemma** *subtensor-subtensor-combine*:  
**assumes**  $\bigwedge A. A \in \text{set } As \implies \text{dims } A = ds$  **and**  $i < \text{length } As$   
**shows**  $\text{subtensor } (\text{subtensor-combine } ds \ As) \ i = As \ ! \ i$   
 ⟨proof⟩

**lemma** *subtensor-induct*[case-names order-0 order-step]:  
**assumes** *order-0*:  $\bigwedge A. \text{dims } A = [] \implies P \ A$   
**and** *order-step*:  $\bigwedge A. \text{dims } A \neq [] \implies (\bigwedge i. i < \text{hd } (\text{dims } A) \implies P \ (\text{subtensor } A \ i)) \implies P \ A$   
**shows**  $P \ B$   
 ⟨proof⟩

**lemma** *subtensor-combine-induct*[case-names order-0 order-step]:  
**assumes** *order-0*:  $\bigwedge A. \text{dims } A = [] \implies P \ A$   
**and** *order-step*:  $\bigwedge As \ ds. (\bigwedge A. A \in \text{set } As \implies P \ A) \implies (\bigwedge A. A \in \text{set } As \implies \text{dims } A = ds) \implies P \ (\text{subtensor-combine } ds \ As)$   
**shows**  $P \ A$   
 ⟨proof⟩

**lemma** *lookup-subtensor1*[simp]:  
**assumes**  $i \# is \triangleleft \text{dims } A$   
**shows**  $\text{lookup } (\text{subtensor } A \ i) \ is = \text{lookup } A \ (i \# is)$   
 ⟨proof⟩

**lemma** *lookup-subtensor*:

**assumes**  $is \triangleleft dims A$   
**shows**  $lookup A is = hd (vec (fold (\lambda i A. subtensor A i) is A))$   
 $\langle proof \rangle$

**lemma** *subtensor-eqI*:  
**assumes**  $dims A \neq []$   
**and** *dims-eq*:  $dims A = dims B$   
**and**  $\bigwedge i. i < hd (dims A) \implies subtensor A i = subtensor B i$   
**shows**  $A=B$   
 $\langle proof \rangle$

**end**

### 3 Tensor Addition

**theory** *Tensor-Plus*  
**imports** *Tensor-Subtensor*  
**begin**

**definition** *vec-plus*  $a b = map (\lambda(x,y). plus x y) (zip a b)$

**definition** *plus-base*:: $'a::semigroup-add tensor \Rightarrow 'a tensor \Rightarrow 'a tensor$   
**where** *plus-base*  $A B = (tensor-from-vec (dims A) (vec-plus (vec A) (vec B)))$

**instantiation** *tensor*:: $(semigroup-add) plus$   
**begin**

**definition** *plus-def*:  $A + B = (if (dims A = dims B)$   
 $then plus-base A B$   
 $else undefined)$

**instance**  $\langle proof \rangle$

**end**

**lemma** *plus-dim1*[*simp*]:  $dims A = dims B \implies dims (A + B) = dims A$   $\langle proof \rangle$

**lemma** *plus-dim2*[*simp*]:  $dims A = dims B \implies dims (A + B) = dims B$   $\langle proof \rangle$

**lemma** *plus-base*:  $dims A = dims B \implies A + B = plus-base A B$   $\langle proof \rangle$

**lemma** *fixed-length-sublist-plus*:

**assumes**  $length xs1 = c * l$   $length xs2 = c * l$   $i < c$

**shows**  $fixed-length-sublist (vec-plus xs1 xs2) l i$   
 $= vec-plus (fixed-length-sublist xs1 l i) (fixed-length-sublist xs2 l i)$   
 $\langle proof \rangle$

**lemma** *vec-plus*[*simp*]:

**assumes**  $dims A = dims B$

**shows**  $vec (A+B) = vec-plus (vec A) (vec B)$   
 $\langle proof \rangle$

**lemma** *subtensor-plus*:  
**fixes**  $A::'a::\text{semigroup-add tensor}$  **and**  $B::'a::\text{semigroup-add tensor}$   
**assumes**  $i < \text{hd} (\text{dims } A)$   
**and**  $\text{dims } A = \text{dims } B$   
**and**  $\text{dims } A \neq []$   
**shows**  $\text{subtensor } (A + B) i = \text{subtensor } A i + \text{subtensor } B i$   
 $\langle \text{proof} \rangle$

**lemma** *lookup-plus[simp]*:  
**assumes**  $\text{dims } A = \text{dims } B$   
**and**  $is \triangleleft \text{dims } A$   
**shows**  $\text{lookup } (A + B) is = \text{lookup } A is + \text{lookup } B is$   
 $\langle \text{proof} \rangle$

**lemma** *plus-assoc*:  
**assumes**  $\text{dims } A = ds$  **and**  $\text{dims } B = ds$  **and**  $\text{dims } C = ds$   
**shows**  $(A + B) + C = A + (B + C)$   
 $\langle \text{proof} \rangle$

**lemma** *tensor-comm[simp]*:  
**fixes**  $A::'a::\text{ab-semigroup-add tensor}$   
**shows**  $A + B = B + A$   
 $\langle \text{proof} \rangle$

**definition**  $\text{vec0 } n = \text{replicate } n \ 0$

**definition**  $\text{tensor0}::\text{nat list} \Rightarrow 'a::\text{zero tensor}$  **where**  
 $\text{tensor0 } d = \text{tensor-from-vec } d (\text{vec0 } (\text{prod-list } d))$

**lemma** *dims-tensor0[simp]*:  $\text{dims } (\text{tensor0 } d) = d$   
**and** *vec-tensor0[simp]*:  $\text{vec } (\text{tensor0 } d) = \text{vec0 } (\text{prod-list } d)$   
 $\langle \text{proof} \rangle$

**lemma** *lookup-is-in-vec*:  $is \triangleleft (\text{dims } A) \implies \text{lookup } A is \in \text{set } (\text{vec } A)$   
 $\langle \text{proof} \rangle$

**lemma** *lookup-tensor0*:  
**assumes**  $is \triangleleft ds$   
**shows**  $\text{lookup } (\text{tensor0 } ds) is = 0$   
 $\langle \text{proof} \rangle$

**lemma**  
**fixes**  $A::'a::\text{monoid-add tensor}$   
**shows** *tensor-add-0-right[simp]*:  $A + \text{tensor0 } (\text{dims } A) = A$   
 $\langle \text{proof} \rangle$

**lemma**  
**fixes**  $A::'a::\text{monoid-add tensor}$   
**shows** *tensor-add-0-left[simp]*:  $\text{tensor0 } (\text{dims } A) + A = A$



*<proof>*

**definition** *listsum*::*nat list*  $\Rightarrow$  *'a::monoid-add tensor list*  $\Rightarrow$  *'a tensor* **where**  
*listsum ds As* = *foldr (+) As (tensor0 ds)*

**definition** *listsum'*::*'a::monoid-add tensor list*  $\Rightarrow$  *'a tensor* **where**  
*listsum' As* = *listsum (dims (hd As)) As*

**lemma** *listsum-Nil*: *listsum ds []* = *tensor0 ds* *<proof>*

**lemma** *listsum-one*: *listsum (dims A) [A]* = *A* *<proof>*

**lemma** *listsum-Cons*: *listsum ds (A # As)* = *A + listsum ds As*  
*<proof>*

**lemma** *listsum-dims*:  
**assumes**  $\bigwedge A. A \in \text{set } As \implies \text{dims } A = ds$   
**shows** *dims (listsum ds As)* = *ds*  
*<proof>*

**lemma** *subtensor0*:  
**assumes** *ds*  $\neq []$  **and** *i* < *hd ds*  
**shows** *subtensor (tensor0 ds) i* = *tensor0 (tl ds)*  
*<proof>*

**lemma** *subtensor-listsum*:  
**assumes**  $\bigwedge A. A \in \text{set } As \implies \text{dims } A = ds$   
**and** *ds*  $\neq []$  **and** *i* < *hd ds*  
**shows** *subtensor (listsum ds As) i* = *listsum (tl ds) (map (\lambda A. subtensor A i) As)*  
*<proof>*

**lemma** *listsum0*:  
**assumes**  $\bigwedge A. A \in \text{set } As \implies A = \text{tensor0 } ds$   
**shows** *listsum ds As* = *tensor0 ds*  
*<proof>*

**lemma** *listsum-all-0-but-one*:  
**assumes**  $\bigwedge i. i \neq j \implies i < \text{length } As \implies As!i = \text{tensor0 } ds$   
**and** *dims (As!j)* = *ds*  
**and** *j* < *length As*  
**shows** *listsum ds As* = *As!j*  
*<proof>*

**lemma** *lookup-listsum*:  
**assumes** *is*  $\triangleleft$  *ds*  
**and**  $\bigwedge A. A \in \text{set } As \implies \text{dims } A = ds$   
**shows** *lookup (listsum ds As) is* =  $(\sum A \leftarrow As. \text{lookup } A \text{ is})$

*<proof>*

**end**

## 4 Tensor Scalar Multiplication

**theory** *Tensor-Scalar-Mult*  
**imports** *Tensor-Plus Tensor-Subtensor*  
**begin**

**definition** *vec-smult*::'a::ring  $\Rightarrow$  'a list  $\Rightarrow$  'a list **where**  
*vec-smult*  $\alpha$   $\beta$  = *map* ((\*)  $\alpha$ )  $\beta$

**lemma** *vec-smult0*: *vec-smult* 0 *as* = *vec0* (*length as*)  
*<proof>*

**lemma** *vec-smult-distr-right*:  
**shows** *vec-smult* ( $\alpha + \beta$ ) *as* = *vec-plus* (*vec-smult*  $\alpha$  *as*) (*vec-smult*  $\beta$  *as*)  
*<proof>*

**lemma** *vec-smult-Cons*:  
**shows** *vec-smult*  $\alpha$  (*a* # *as*) = ( $\alpha * a$ ) # *vec-smult*  $\alpha$  *as* *<proof>*

**lemma** *vec-plus-Cons*:  
**shows** *vec-plus* (*a* # *as*) (*b* # *bs*) = (*a+b*) # *vec-plus* *as* *bs* *<proof>*

**lemma** *vec-smult-distr-left*:  
**assumes** *length as* = *length bs*  
**shows** *vec-smult*  $\alpha$  (*vec-plus as bs*) = *vec-plus* (*vec-smult*  $\alpha$  *as*) (*vec-smult*  $\alpha$  *bs*)  
*<proof>*

**lemma** *length-vec-smult*: *length* (*vec-smult*  $\alpha$  *v*) = *length v* *<proof>*

**definition** *smult*::'a::ring  $\Rightarrow$  'a tensor  $\Rightarrow$  'a tensor (**infixl**  $\cdot$  70) **where**  
*smult*  $\alpha$  *A* = (*tensor-from-vec* (*dims A*) (*vec-smult*  $\alpha$  (*vec A*)))

**lemma** *tensor-smult0*: **fixes** *A*::'a::ring *tensor*  
**shows** 0  $\cdot$  *A* = *tensor0* (*dims A*)  
*<proof>*

**lemma** *dims-smult[simp]*:*dims* ( $\alpha \cdot A$ ) = *dims A*  
**and** *vec-smult[simp]*: *vec* ( $\alpha \cdot A$ ) = *map* ((\*)  $\alpha$ ) (*vec A*)  
*<proof>*

**lemma** *tensor-smult-distr-right*: ( $\alpha + \beta$ )  $\cdot$  *A* =  $\alpha \cdot A$  +  $\beta \cdot A$   
*<proof>*

**lemma** *tensor-smult-distr-left*:  $\text{dims } A = \text{dims } B \implies \alpha \cdot (A + B) = \alpha \cdot A + \alpha \cdot B$

*<proof>*

**lemma** *smult-fixed-length-sublist*:

**assumes**  $\text{length } xs = l * c \ i < c$

**shows**  $\text{fixed-length-sublist } (\text{vec-smult } \alpha \ xs) \ l \ i = \text{vec-smult } \alpha \ (\text{fixed-length-sublist } xs \ l \ i)$

*<proof>*

**lemma** *smult-subtensor*:

**assumes**  $\text{dims } A \neq [] \ i < \text{hd } (\text{dims } A)$

**shows**  $\alpha \cdot \text{subtensor } A \ i = \text{subtensor } (\alpha \cdot A) \ i$

*<proof>*

**lemma** *lookup-smult*:

**assumes**  $is < \text{dims } A$

**shows**  $\text{lookup } (\alpha \cdot A) \ is = \alpha * \text{lookup } A \ is$

*<proof>*

**lemma** *tensor-smult-assoc*:

**fixes**  $A :: 'a :: \text{ring tensor}$

**shows**  $\alpha \cdot (\beta \cdot A) = (\alpha * \beta) \cdot A$

*<proof>*

**end**

## 5 Tensor Product

**theory** *Tensor-Product*

**imports** *Tensor-Scalar-Mult Tensor-Subtensor*

**begin**

**instantiation** *tensor*:: (*ring*) *semigroup-mult*

**begin**

**definition** *tensor-prod-def*:  $A * B = \text{tensor-from-vec } (\text{dims } A \ @ \ \text{dims } B) \ (\text{concat } (\text{map } (\lambda a. \text{vec-smult } a \ (\text{vec } B)) \ (\text{vec } A)))$

**abbreviation** *tensor-prod-otimes* ::  $'a \ \text{tensor} \Rightarrow 'a \ \text{tensor} \Rightarrow 'a \ \text{tensor}$  (**infixl**  $\otimes$  70)

**where**  $A \otimes B \equiv A * B$

**lemma** *vec-tensor-prod[simp]*:  $\text{vec } (A \otimes B) = \text{concat } (\text{map } (\lambda a. \text{vec-smult } a \ (\text{vec } B)) \ (\text{vec } A))$  (**is** ?V)

**and** *dims-tensor-prod[simp]*:  $\text{dims } (A \otimes B) = \text{dims } A \ @ \ \text{dims } B$  (**is** ?D)

*<proof>*

**lemma** *tensorprod-subtensor-base*:

**shows**  $\text{concat} (\text{map } f (\text{concat } xss)) = \text{concat} (\text{map} (\lambda xs. \text{concat} (\text{map } f xs)) xss)$   
 ⟨proof⟩

**lemma** *subtensor-combine-tensor-prod*:  
**assumes**  $\bigwedge A. A \in \text{set } As \implies \text{dims } A = ds$   
**shows**  $\text{subtensor-combine } ds \ As \otimes B = \text{subtensor-combine} (ds @ \text{dims } B) (\text{map} (\lambda A. A \otimes B) As)$   
 ⟨proof⟩

**lemma** *subtensor-tensor-prod*:  
**assumes**  $\text{dims } A \neq []$  **and**  $i < \text{hd} (\text{dims } A)$   
**shows**  $\text{subtensor} (A \otimes B) i = \text{subtensor } A \ i \otimes B$   
 ⟨proof⟩

**lemma** *lookup-tensor-prod[simp]*:  
**assumes** *is1-valid*:  $is1 \triangleleft \text{dims } A$  **and** *is2-valid*:  $is2 \triangleleft \text{dims } B$   
**shows**  $\text{lookup} (A \otimes B) (is1 @ is2) = \text{lookup } A \ is1 * \text{lookup } B \ is2$   
 ⟨proof⟩

**lemma** *valid-index-split*:  
**assumes**  $is \triangleleft ds1 @ ds2$   
**obtains**  $is1 \ is2$  **where**  $is1 @ is2 = is$   $is1 \triangleleft ds1$   $is2 \triangleleft ds2$   
 ⟨proof⟩

**instance** ⟨proof⟩

**end**

**lemma** *tensor-prod-distr-left*:  
**assumes**  $\text{dims } A = \text{dims } B$   
**shows**  $(A + B) \otimes C = (A \otimes C) + (B \otimes C)$   
 ⟨proof⟩

**lemma** *tensor-prod-distr-right*:  
**assumes**  $\text{dims } A = \text{dims } B$   
**shows**  $C \otimes (A + B) = (C \otimes A) + (C \otimes B)$   
 ⟨proof⟩

**instantiation** *tensor* :: (ring-1) monoid-mult  
**begin**  
**definition** *tensor-one-def*:  $1 = \text{tensor-from-vec} [] [1]$

**lemma** *tensor-one-from-lookup*:  $1 = \text{tensor-from-lookup} [] (\lambda-. 1)$   
 ⟨proof⟩

**instance** ⟨proof⟩

**end**

**lemma** *order-tensor-one*:  $order\ 1 = 0$  *<proof>*

**lemma** *smult-prod-extract1*:  
**fixes**  $a::'a::comm-ring-1$   
**shows**  $a \cdot (A \otimes B) = (a \cdot A) \otimes B$   
*<proof>*

**lemma** *smult-prod-extract2*:  
**fixes**  $a::'a::comm-ring-1$   
**shows**  $a \cdot (A \otimes B) = A \otimes (a \cdot B)$   
*<proof>*

**lemma** *order-0-multiple-of-one*:  
**assumes**  $order\ A = 0$   
**obtains**  $a$  **where**  $A = a \cdot 1$   
*<proof>*

**lemma** *smult-1*:  
**fixes**  $A::'a::ring-1\ tensor$   
**shows**  $A = 1 \cdot A$  *<proof>*

**lemma** *tensor0-prod-right[simp]*:  $A \otimes tensor0\ ds = tensor0\ (dims\ A\ @\ ds)$   
*<proof>*

**lemma** *tensor0-prod-left[simp]*:  $tensor0\ ds \otimes A = tensor0\ (ds\ @\ dims\ A)$   
*<proof>*

**lemma** *subtensor-prod-with-vec*:  
**assumes**  $order\ A = 1\ i < hd\ (dims\ A)$   
**shows**  $subtensor\ (A \otimes B)\ i = lookup\ A\ [i] \cdot B$   
*<proof>*

**end**

## 6 Unit Vectors as Tensors

**theory** *Tensor-Unit-Vec*  
**imports** *Tensor-Product*  
**begin**

**definition** *unit-vec*:: $nat \Rightarrow nat \Rightarrow 'a::ring-1\ tensor$   
**where**  $unit-vec\ n\ i = tensor-from-lookup\ [n]\ (\lambda x. if\ x=[i]\ then\ 1\ else\ 0)$

**lemma** *dims-unit-vec*:  $dims\ (unit-vec\ n\ i) = [n]$  *<proof>*

**lemma** *lookup-unit-vec*:  
**assumes**  $j < n$

**shows** *lookup* (*unit-vec* *n* *i*) [*j*] = (*if* *i=j* *then* *1* *else* *0*)  
 ⟨*proof*⟩

**lemma** *subtensor-prod-with-unit-vec*:

**fixes** *A*::'a::ring-1 *tensor*

**assumes** *j*<*n*

**shows** *subtensor* (*unit-vec* *n* *i*  $\otimes$  *A*) *j* = (*if* *i=j* *then* *A* *else* (*tensor0* (*dims* *A*)))  
 ⟨*proof*⟩

**lemma** *subtensor-decomposition*:

**assumes** *dims* *A*  $\neq$  []

**shows** *listsum* (*dims* *A*) (*map* ( $\lambda i.$  *unit-vec* (*hd* (*dims* *A*)) *i*  $\otimes$  *subtensor* *A* *i*)  
 [*0*..*hd* (*dims* *A*)] = *A* (**is** ?*LS* = *A*)  
 ⟨*proof*⟩

**end**

## 7 Tensor CP-Rank

**theory** *Tensor-Rank*

**imports** *Tensor-Unit-Vec*

**begin**

**inductive** *cprank-max1*::'a::ring-1 *tensor*  $\Rightarrow$  *bool* **where**

*order1*: *order* *A*  $\leq$  *1*  $\Longrightarrow$  *cprank-max1* *A* |

*higher-order*: *order* *A* = *1*  $\Longrightarrow$  *cprank-max1* *B*  $\Longrightarrow$  *cprank-max1* (*A*  $\otimes$  *B*)

**lemma** *cprank-max1-order0*: *cprank-max1* *B*  $\Longrightarrow$  *order* *A* = *0*  $\Longrightarrow$  *cprank-max1*  
 (*A*  $\otimes$  *B*)

⟨*proof*⟩

**lemma** *cprank-max1-order-le1*: *order* *A*  $\leq$  *0*  $\Longrightarrow$  *cprank-max1* *B*  $\Longrightarrow$  *cprank-max1*  
 (*A*  $\otimes$  *B*)

⟨*proof*⟩

**lemma** *cprank-max1-prod*: *cprank-max1* *A*  $\Longrightarrow$  *cprank-max1* *B*  $\Longrightarrow$  *cprank-max1*  
 (*A*  $\otimes$  *B*)

⟨*proof*⟩

**lemma** *cprank-max1-prod-list*:

**assumes**  $\bigwedge B. B \in \text{set } Bs \Longrightarrow$  *cprank-max1* *B*

**shows** *cprank-max1* (*prod-list* *Bs*)

⟨*proof*⟩

**lemma** *cprank-max1-prod-listE*:

**fixes** *A*::'a::comm-ring-1 *tensor*

**assumes** *cprank-max1* *A*

**obtains** *Bs* *a* **where**  $\bigwedge B. B \in \text{set } Bs \Longrightarrow$  *order* *B* = *1* *a*  $\cdot$  *prod-list* *Bs* = *A*

⟨*proof*⟩

**inductive** *cprank-max* :: *nat*  $\Rightarrow$  '*a*::*ring-1 tensor*  $\Rightarrow$  *bool* **where**

*cprank-max0*: *cprank-max* 0 (*tensor0* *ds*) |

*cprank-max-Suc*: *dims* *A* = *dims* *B*  $\Rightarrow$  *cprank-max1* *A*  $\Rightarrow$  *cprank-max* *j* *B*  $\Rightarrow$   
*cprank-max* (*Suc* *j*) (*A+B*)

**lemma** *cprank-max1*: *cprank-max1* *A*  $\Rightarrow$  *cprank-max* 1 *A*

*<proof>*

**lemma** *cprank-max-plus*: *cprank-max* *i* *A*  $\Rightarrow$  *cprank-max* *j* *B*  $\Rightarrow$  *dims* *A* = *dims*  
*B*  $\Rightarrow$  *cprank-max* (*i+j*) (*A+B*)

*<proof>*

**lemma** *cprank-max-listsum*:

**assumes**  $\bigwedge A. A \in \text{set } As \Rightarrow \text{dims } A = ds$

**and**  $\bigwedge A. A \in \text{set } As \Rightarrow \text{cprank-max } n \ A$

**shows** *cprank-max* (*n\*length* *As*) (*listsum* *ds* *As*)

*<proof>*

**lemma** *cprank-maxE*:

**assumes** *cprank-max* *n* *A*

**obtains** *BS* **where** ( $\bigwedge B. B \in \text{set } BS \Rightarrow \text{cprank-max1 } B$ ) **and** ( $\bigwedge B. B \in \text{set } BS$   
 $\Rightarrow \text{dims } A = \text{dims } B$ ) **and** *listsum* (*dims* *A*) *BS* = *A* **and** *length* *BS* = *n*

*<proof>*

**lemma** *cprank-maxI*:

**assumes**  $\bigwedge B. B \in \text{set } BS \Rightarrow \text{cprank-max1 } B$

**and**  $\bigwedge B. B \in \text{set } BS \Rightarrow \text{dims } B = ds$

**shows** *cprank-max* (*length* *BS*) (*listsum* *ds* *BS*)

*<proof>*

**lemma** *cprank-max-0E*: *cprank-max* 0 *A*  $\Rightarrow$  *A* = *tensor0* (*dims* *A*) *<proof>*

**lemma** *listsum-prod-distr-right*:

**assumes** ( $\bigwedge C. C \in \text{set } CS \Rightarrow \text{dims } C = ds$ )

**shows** *A*  $\otimes$  *listsum* *ds* *CS* = *listsum* (*dims* *A* @ *ds*) (*map* ( $\lambda C. A \otimes C$ ) *CS*)

*<proof>*

**lemma** *cprank-max-prod-order1*:

**assumes** *order* *A* = 1

**and** *cprank-max* *n* *B*

**shows** *cprank-max* *n* (*A* $\otimes$ *B*)

*<proof>*

**lemma** *cprank-max-upper-bound*:

**shows** *cprank-max* (*prod-list* (*dims* *A*)) *A*

*<proof>*

**definition** *cprank* :: '*a*::*ring-1 tensor*  $\Rightarrow$  *nat* **where**

$cprank\ A = (LEAST\ n.\ cprank-max\ n\ A)$

**lemma** *cprank-upper-bound*:  $cprank\ A \leq prod-list\ (dims\ A)$   
*<proof>*

**lemma** *cprank-max-cprank*:  $cprank-max\ (cprank\ A)\ A$   
*<proof>*

**end**

## 8 Tensor Matricization

**theory** *Tensor-Matricization*

**imports** *Tensor-Plus*

*Jordan-Normal-Form.Matrix Jordan-Normal-Form.DL-Missing-Sublist*

**begin**

**fun** *digit-decode* ::  $nat\ list \Rightarrow nat\ list \Rightarrow nat$  **where**  
*digit-decode* [] [] = 0 |  
*digit-decode* (d # ds) (i # is) = i + d \* *digit-decode* ds is

**fun** *digit-encode* ::  $nat\ list \Rightarrow nat \Rightarrow nat\ list$  **where**  
*digit-encode* [] a = [] |  
*digit-encode* (d # ds) a = a mod d # *digit-encode* ds (a div d)

**lemma** *digit-encode-decode[simp]*:  
**assumes**  $is \triangleleft ds$   
**shows**  $digit-encode\ ds\ (digit-decode\ ds\ is) = is$   
*<proof>*

**lemma** *digit-decode-encode[simp]*:  
**shows**  $digit-decode\ ds\ (digit-encode\ ds\ a) = a\ mod\ (prod-list\ ds)$   
*<proof>*

**lemma** *digit-decode-encode-lt[simp]*:  
**assumes**  $a < prod-list\ ds$   
**shows**  $digit-decode\ ds\ (digit-encode\ ds\ a) = a$   
*<proof>*

**lemma** *digit-decode-lt*:  
**assumes**  $is \triangleleft ds$   
**shows**  $digit-decode\ ds\ is < prod-list\ ds$   
*<proof>*

**lemma** *digit-encode-valid-index*:  
**assumes**  $a < prod-list\ ds$   
**shows**  $digit-encode\ ds\ a \triangleleft ds$   
*<proof>*



**lemma** *length-digit-encode*:  
**shows**  $\text{length } (\text{digit-encode } ds \ a) = \text{length } ds$   
 ⟨*proof*⟩

**lemma** *digit-encode-0*:  
 $\text{prod-list } ds \ \text{dvd } a \implies \text{digit-encode } ds \ a = \text{replicate } (\text{length } ds) \ 0$   
 ⟨*proof*⟩

**lemma** *valid-index-weave*:  
**assumes**  $is1 \triangleleft (\text{nths } ds \ A)$   
**and**  $is2 \triangleleft (\text{nths } ds \ (-A))$   
**shows**  $\text{weave } A \ is1 \ is2 \triangleleft ds$   
**and**  $\text{nths } (\text{weave } A \ is1 \ is2) \ A = is1$   
**and**  $\text{nths } (\text{weave } A \ is1 \ is2) \ (-A) = is2$   
 ⟨*proof*⟩

**definition** *matricize* ::  $\text{nat set} \Rightarrow 'a \ \text{tensor} \Rightarrow 'a \ \text{mat}$  **where**  
 $\text{matricize } rmodes \ T = \text{mat}$   
 ( $\text{prod-list } (\text{nths } (\text{Tensor.dims } T) \ rmodes)$ )  
 ( $\text{prod-list } (\text{nths } (\text{Tensor.dims } T) \ (-rmodes))$ )  
 ( $\lambda(r, c). \ \text{Tensor.lookup } T \ (\text{weave } rmodes$   
   ( $\text{digit-encode } (\text{nths } (\text{Tensor.dims } T) \ rmodes) \ r$ )  
   ( $\text{digit-encode } (\text{nths } (\text{Tensor.dims } T) \ (-rmodes)) \ c$ )  
 )))

**definition** *dematricize*:: $\text{nat set} \Rightarrow 'a \ \text{mat} \Rightarrow \text{nat list} \Rightarrow 'a \ \text{tensor}$  **where**  
 $\text{dematricize } rmodes \ A \ ds = \text{tensor-from-lookup } ds$   
 ( $\lambda is. \ A \ \$\$ \ (\text{digit-decode } (\text{nths } ds \ rmodes) \ (\text{nths } is \ rmodes),$   
    $\text{digit-decode } (\text{nths } ds \ (-rmodes)) \ (\text{nths } is \ (-rmodes)))$ )  
 )

**lemma** *dims-matricize*:  
 $\text{dim-row } (\text{matricize } rmodes \ T) = \text{prod-list } (\text{nths } (\text{Tensor.dims } T) \ rmodes)$   
 $\text{dim-col } (\text{matricize } rmodes \ T) = \text{prod-list } (\text{nths } (\text{Tensor.dims } T) \ (-rmodes))$   
 ⟨*proof*⟩

**lemma** *dims-dematricize*:  $\text{Tensor.dims } (\text{dematricize } rmodes \ A \ ds) = ds$   
 ⟨*proof*⟩

**lemma** *valid-index-nths*:  
**assumes**  $is \triangleleft ds$   
**shows**  $\text{nths } is \ A \triangleleft \text{nths } ds \ A$   
 ⟨*proof*⟩

**lemma** *dematricize-matricize*:  
**shows**  $\text{dematricize } rmodes \ (\text{matricize } rmodes \ T) \ (\text{Tensor.dims } T) = T$   
 ⟨*proof*⟩

**lemma** *matricize-dematricize*:  
**assumes**  $\dim\text{-row } A = \text{prod-list } (nths \ ds \ rmodes)$   
**and**  $\dim\text{-col } A = \text{prod-list } (nths \ ds \ (-rmodes))$   
**shows**  $\text{matricize } rmodes \ (\text{dematricize } rmodes \ A \ ds) = A$   
 $\langle \text{proof} \rangle$

**lemma** *matricize-add*:  
**assumes**  $\text{dims } A = \text{dims } B$   
**shows**  $\text{matricize } I \ A + \text{matricize } I \ B = \text{matricize } I \ (A+B)$   
 $\langle \text{proof} \rangle$

**lemma** *matricize-0*:  
**shows**  $\text{matricize } I \ (\text{tensor0 } ds) = 0_m \ (\dim\text{-row } (\text{matricize } I \ (\text{tensor0 } ds))) \ (\dim\text{-col } (\text{matricize } I \ (\text{tensor0 } ds)))$   
 $\langle \text{proof} \rangle$

**end**

## 9 CP-Rank and Matrix Rank

**theory** *DL-Rank-CP-Rank*  
**imports** *Tensor-Rank Jordan-Normal-Form.DL-Rank Tensor-Matricization*  
*Jordan-Normal-Form.DL-Submatrix Jordan-Normal-Form.Missing-VectorSpace*  
**begin**

**abbreviation**  $mrank \ A == \text{vec-space.rank } (\dim\text{-row } A) \ A$

**no-notation** *normal-rel* (**infixl**  $\triangleleft 60$ )

**lemma** *lookup-order1-prod*:  
**assumes**  $\bigwedge B. B \in \text{set } Bs \implies \text{Tensor.order } B = 1$   
**assumes**  $is \triangleleft \text{dims } (\text{prod-list } Bs)$   
**shows**  $\text{lookup } (\text{prod-list } Bs) \ is = \text{prod-list } (\text{map } (\lambda(i,B). \text{lookup } B \ [i]) \ (\text{zip } is \ Bs))$   
 $\langle \text{proof} \rangle$

**lemma** *matricize-cprank-max1*:  
**fixes**  $A :: 'a :: \text{field} \ \text{tensor}$   
**assumes** *cprank-max1*  $A$   
**shows**  $mrank \ (\text{matricize } I \ A) \leq 1$   
 $\langle \text{proof} \rangle$

**lemma** *matrix-rank-le-cprank-max*:  
**fixes**  $A :: ('a :: \text{field}) \ \text{tensor}$   
**assumes** *cprank-max*  $r \ A$   
**shows**  $mrank \ (\text{matricize } I \ A) \leq r$   
 $\langle \text{proof} \rangle$

**lemma** *matrix-rank-le-cp-rank*:

**fixes**  $A :: ('a::field) \text{ tensor}$   
**shows**  $\text{mrank } (\text{matricize } I \ A) \leq \text{cprank } A$   
 $\langle \text{proof} \rangle$   
**end**

## 10 Matrix to Vector Conversion

**theory** *DL-Flatten-Matrix*  
**imports** *Jordan-Normal-Form.Matrix*  
**begin**

**definition** *extract-matrix*  $:: (\text{nat} \Rightarrow 'a) \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ mat}$  **where**  
*extract-matrix*  $a \ m \ n = \text{mat } m \ n \ (\lambda(i,j). a \ (i*n + j))$

**definition** *flatten-matrix*  $:: 'a \text{ mat} \Rightarrow (\text{nat} \Rightarrow 'a)$  **where**  
*flatten-matrix*  $A \ k = A \ \$\$ \ (k \ \text{div} \ \text{dim-col } A, \ k \ \text{mod} \ \text{dim-col } A)$

**lemma** *two-digit-le*:  
 $i * n + j < m * n$  **if**  $i < m \ j < n$  **for**  $i \ j :: \text{nat}$   
 $\langle \text{proof} \rangle$

**lemma** *extract-matrix-cong*:  
**assumes**  $\bigwedge i. i < m * n \implies a \ i = b \ i$   
**shows**  $\text{extract-matrix } a \ m \ n = \text{extract-matrix } b \ m \ n$   
 $\langle \text{proof} \rangle$

**lemma** *extract-matrix-flatten-matrix*:  
 $\text{extract-matrix } (\text{flatten-matrix } A) \ (\text{dim-row } A) \ (\text{dim-col } A) = A$   
 $\langle \text{proof} \rangle$

**lemma** *extract-matrix-flatten-matrix-cong*:  
**assumes**  $\bigwedge x. x < \text{dim-row } A * \text{dim-col } A \implies f \ x = \text{flatten-matrix } A \ x$   
**shows**  $\text{extract-matrix } f \ (\text{dim-row } A) \ (\text{dim-col } A) = A$   
 $\langle \text{proof} \rangle$

**lemma** *flatten-matrix-extract-matrix*:  
 $\text{flatten-matrix } (\text{extract-matrix } a \ m \ n) \ k = a \ k$  **if**  $k < m * n$   
 $\langle \text{proof} \rangle$

**lemma** *index-extract-matrix*:  
**assumes**  $i < m \ j < n$   
**shows**  $\text{extract-matrix } a \ m \ n \ \$\$ \ (i,j) = a \ (i*n + j)$   
 $\langle \text{proof} \rangle$

**lemma** *dim-extract-matrix*:  
**shows**  $\text{dim-row } (\text{extract-matrix } a \ m \ n) = m$   
**and**  $\text{dim-col } (\text{extract-matrix } a \ m \ n) = n$   
 $\langle \text{proof} \rangle$

end

## 11 Deep Learning Networks

**theory** *DL-Network*

**imports** *Tensor-Product*

*Jordan-Normal-Form.Matrix Tensor-Unit-Vec DL-Flatten-Matrix*

*Jordan-Normal-Form.DL-Missing-List*

**begin**

This symbol is used for the Tensor product:

**no-notation** *Group.monoid.mult* (**infixl**  $\otimes_1$  70)

**notation** *Matrix.unit-vec* ( $unit_v$ )

**hide-const** (**open**) *Matrix.unit-vec*

**datatype** *'a convnet* = *Input nat* | *Conv 'a 'a convnet* | *Pool 'a convnet 'a convnet*

**fun** *input-sizes* :: *'a convnet*  $\Rightarrow$  *nat list* **where**

*input-sizes* (*Input M*) = [*M*] |

*input-sizes* (*Conv A m*) = *input-sizes m* |

*input-sizes* (*Pool m1 m2*) = *input-sizes m1* @ *input-sizes m2*

**fun** *count-weights* :: *bool*  $\Rightarrow$  (*nat*  $\times$  *nat*) *convnet*  $\Rightarrow$  *nat* **where**

*count-weights shared* (*Input M*) = 0 |

*count-weights shared* (*Conv (r0, r1) m*) = *r0 \* r1 + count-weights shared m* |

*count-weights shared* (*Pool m1 m2*) =

(*if shared*

*then max (count-weights shared m1) (count-weights shared m2)*

*else count-weights shared m1 + count-weights shared m2*)

**fun** *output-size* :: (*nat*  $\times$  *nat*) *convnet*  $\Rightarrow$  *nat* **where**

*output-size* (*Input M*) = *M* |

*output-size* (*Conv (r0,r1) m*) = *r0* |

*output-size* (*Pool m1 m2*) = *output-size m1*

**inductive** *valid-net* :: (*nat*  $\times$  *nat*) *convnet*  $\Rightarrow$  *bool* **where**

*valid-net* (*Input M*) |

*output-size m = r1*  $\Longrightarrow$  *valid-net m*  $\Longrightarrow$  *valid-net (Conv (r0,r1) m)* |

*output-size m1 = output-size m2*  $\Longrightarrow$  *valid-net m1*  $\Longrightarrow$  *valid-net m2*  $\Longrightarrow$  *valid-net (Pool m1 m2)*

**fun** *insert-weights* :: *bool*  $\Rightarrow$  (*nat*  $\times$  *nat*) *convnet*  $\Rightarrow$  (*nat*  $\Rightarrow$  *real*)  $\Rightarrow$  *real mat convnet* **where**

*insert-weights shared* (*Input M*) *w* = *Input M* |

*insert-weights shared* (*Conv (r0,r1) m*) *w* = *Conv*

```

    (extract-matrix w r0 r1)
    (insert-weights shared m ( $\lambda i. w (i+r0*r1)$ )) |
insert-weights shared (Pool m1 m2) w = Pool
    (insert-weights shared m1 w)
    (insert-weights shared m2 (if shared then w else ( $\lambda i. w (i+(count-weights shared m1))$ ))))

```

**fun** *remove-weights* :: *real mat convnet*  $\Rightarrow$  (*nat*  $\times$  *nat*) *convnet* **where**  
*remove-weights* (*Input* *M*) = *Input* *M* |  
*remove-weights* (*Conv* *A* *m*) = *Conv* (*dim-row* *A*, *dim-col* *A*) (*remove-weights* *m*) |  
*remove-weights* (*Pool* *m1* *m2*) = *Pool* (*remove-weights* *m1*) (*remove-weights* *m2*)

**abbreviation** *output-size'* == ( $\lambda m. output-size (remove-weights m)$ )  
**abbreviation** *valid-net'* == ( $\lambda m. valid-net (remove-weights m)$ )

**fun** *evaluate-net* :: *real mat convnet*  $\Rightarrow$  *real vec list*  $\Rightarrow$  *real vec* **where**  
*evaluate-net* (*Input* *M*) *inputs* = *hd inputs* |  
*evaluate-net* (*Conv* *A* *m*) *inputs* = *A* \*\_v *evaluate-net m inputs* |  
*evaluate-net* (*Pool* *m1* *m2*) *inputs* = *component-mult*  
 (*evaluate-net* *m1* (*take* (*length* (*input-sizes* *m1*)) *inputs*))  
 (*evaluate-net* *m2* (*drop* (*length* (*input-sizes* *m1*)) *inputs*))

**definition** *mat-tensorlist-mult* :: *real mat*  $\Rightarrow$  *real tensor vec*  $\Rightarrow$  *nat list*  $\Rightarrow$  *real tensor vec*  
**where** *mat-tensorlist-mult* *A* *Ts* *ds*  
 = *Matrix.vec* (*dim-row* *A*) ( $\lambda j. tensor-from-lookup ds (\lambda is. (A *_v (map-vec (\lambda T. Tensor.lookup T is) Ts)) \$j)$ )

**lemma** *insert-weights-cong*:  
**assumes** ( $\bigwedge i. i < count-weights s m \implies w1 i = w2 i$ )  
**shows** *insert-weights* *s* *m* *w1* = *insert-weights* *s* *m* *w2*  
 <proof>

**lemma** *dims-mat-tensorlist-mult*:  
**assumes**  $T \in set_v (mat-tensorlist-mult A Ts ds)$   
**shows** *Tensor.dims* *T* = *ds*  
 <proof>

**fun** *tensors-from-net* :: *real mat convnet*  $\Rightarrow$  *real tensor vec* **where**  
*tensors-from-net* (*Input* *M*) = *Matrix.vec* *M* ( $\lambda i. unit-vec M i$ ) |  
*tensors-from-net* (*Conv* *A* *m*) = *mat-tensorlist-mult* *A* (*tensors-from-net* *m*) (*input-sizes* *m*) |  
*tensors-from-net* (*Pool* *m1* *m2*) = *component-mult* (*tensors-from-net* *m1*) (*tensors-from-net* *m2*)

**lemma** *output-size-correct-tensors*:  
**assumes** *valid-net'* *m*  
**shows** *output-size'* *m* = *dim-vec* (*tensors-from-net* *m*)  
 <proof>

**lemma** *output-size-correct*:  
**assumes** *valid-net' m*  
**and** *map dim-vec inputs = input-sizes m*  
**shows** *output-size' m = dim-vec (evaluate-net m inputs)*  
*<proof>*

**lemma** *input-sizes-remove-weights*: *input-sizes m = input-sizes (remove-weights m)*  
*<proof>*

**lemma** *dims-tensors-from-net*:  
**assumes**  $T \in \text{set}_v$  (*tensors-from-net m*)  
**shows** *Tensor.dims T = input-sizes m*  
*<proof>*

**definition** *base-input* :: *real mat convnet*  $\Rightarrow$  *nat list*  $\Rightarrow$  *real vec list* **where**  
*base-input m is = (map ( $\lambda(n, i). \text{unit}_v n i$ ) (zip (input-sizes m) is))*

**lemma** *base-input-length*:  
**assumes**  $is \triangleleft \text{input-sizes } m$   
**shows** *input-sizes m = map dim-vec (base-input m is)*  
*<proof>*

**lemma** *nth-mat-tensorlist-mult*:  
**assumes**  $\bigwedge A. A \in \text{set}_v Ts \implies \text{dims } A = ds$   
**assumes**  $i < \text{dim-row } A$   
**assumes** *dim-vec Ts = dim-col A*  
**shows** *mat-tensorlist-mult A Ts ds \$ i = listsum ds (map ( $\lambda j. (A \text{ $$ } (i, j)) \cdot Ts$  \$ j) [0..*dim-vec Ts*])*  
*(is - = listsum ds ?Ts')*  
*<proof>*

**lemma** *lookup-tensors-from-net*:  
**assumes** *valid-net' m*  
**and**  $is \triangleleft \text{input-sizes } m$   
**and**  $j < \text{output-size}' m$   
**shows** *Tensor.lookup (tensors-from-net m \$ j) is = evaluate-net m (base-input m is) \$ j*  
*<proof>*

**primrec** *extract-weights::bool*  $\Rightarrow$  *real mat convnet*  $\Rightarrow$  *nat*  $\Rightarrow$  *real* **where**  
*extract-weights-Input: extract-weights shared (Input M) = ( $\lambda x. 0$ )*  
| *extract-weights-Conv: extract-weights shared (Conv A m) =*  
*( $\lambda x. \text{if } x < \text{dim-row } A * \text{dim-col } A \text{ then flatten-matrix } A x$*   
*else extract-weights shared m (x - dim-row A \* dim-col A))*  
| *extract-weights-Pool: extract-weights shared (Pool m1 m2) =*  
*( $\lambda x. \text{if } x < \text{count-weights shared (remove-weights m1)}$*

then *extract-weights shared*  $m1$   $x$   
 else *extract-weights shared*  $m2$  ( $x - \text{count-weights shared (remove-weights } m1)$ ))

**inductive** *balanced-net::(nat × nat) convnet ⇒ bool where*  
*balanced-net-Input: balanced-net (Input M)*  
 | *balanced-net-Conv: balanced-net m ⇒ balanced-net (Conv A m)*  
 | *balanced-net-Pool: balanced-net m1 ⇒ balanced-net m2 ⇒*  
*count-weights True m1 = count-weights True m2 ⇒ balanced-net (Pool m1 m2)*

**inductive** *shared-weight-net::real mat convnet ⇒ bool where*  
*shared-weight-net-Input: shared-weight-net (Input M)*  
 | *shared-weight-net-Conv: shared-weight-net m ⇒ shared-weight-net (Conv A m)*  
 | *shared-weight-net-Pool: shared-weight-net m1 ⇒ shared-weight-net m2 ⇒*  
*count-weights True (remove-weights m1) = count-weights True (remove-weights m2) ⇒*  
 ( $\bigwedge x. x < \text{count-weights True (remove-weights m1)} \Rightarrow \text{extract-weights True m1 } x = \text{extract-weights True m2 } x$ )  
 $\Rightarrow \text{shared-weight-net (Pool m1 m2)}$

**lemma** *insert-extract-weights-cong-shared:*  
**assumes** *shared-weight-net m*  
**assumes**  $\bigwedge x. x < \text{count-weights True (remove-weights m)} \Rightarrow f x = \text{extract-weights True m } x$   
**shows**  $m = \text{insert-weights True (remove-weights m) } f$   
 ⟨*proof*⟩

**lemma** *insert-extract-weights-cong-unshared:*  
**assumes**  $\bigwedge x. x < \text{count-weights False (remove-weights m)} \Rightarrow f x = \text{extract-weights False m } x$   
**shows**  $m = \text{insert-weights False (remove-weights m) } f$   
 ⟨*proof*⟩

**lemma** *remove-insert-weights:*  
**shows**  $\text{remove-weights (insert-weights } s \text{ m } w) = m$   
 ⟨*proof*⟩

**lemma** *extract-insert-weights-shared:*  
**assumes**  $x < \text{count-weights True m}$   
**and** *balanced-net m*  
**shows**  $\text{extract-weights True (insert-weights True m } w) x = w$   
 ⟨*proof*⟩

**lemma** *shared-weight-net-insert-weights: balanced-net m ⇒ shared-weight-net (insert-weights True m w)*  
 ⟨*proof*⟩

**lemma** *finite-valid-index: finite {is. is < ds}*

*<proof>*

**lemma** *setsum-valid-index-split*:

$(\sum is \mid is < ds1 \ @ \ ds2. f \ is) = (\sum is1 \mid is1 < ds1. (\sum is2 \mid is2 < ds2. f \ (is1 \ @ \ is2)))$

*<proof>*

**lemma** *prod-lessThan-split*:

**fixes**  $g :: nat \Rightarrow real$  **shows**  $prod \ g \ \{..<n+m\} = prod \ g \ \{..<n\} * prod \ (\lambda x. g \ (x+n)) \ \{..<m\}$

*<proof>*

**lemma** *evaluate-net-from-tensors*:

**assumes** *valid-net' m*

**and**  $map \ dim\_vec \ inputs = input\_sizes \ m$

**and**  $j < output\_size' \ m$

**shows**  $evaluate\_net \ m \ inputs \ \$ \ j$

$= (\sum is \in \{is. is < input\_sizes \ m\}. (\prod k < length \ inputs. inputs \ ! \ k \ \$ \ (is!k)) * Tensor.lookup \ (tensors\_from\_net \ m \ \$ \ j) \ is)$

*<proof>*

**lemma** *tensors-from-net-eqI*:

**assumes** *valid-net' m1 valid-net' m2 input-sizes m1 = input-sizes m2*

**assumes**  $\bigwedge inputs. input\_sizes \ m1 = map \ dim\_vec \ inputs \Longrightarrow evaluate\_net \ m1 \ inputs = evaluate\_net \ m2 \ inputs$

**shows**  $tensors\_from\_net \ m1 = tensors\_from\_net \ m2$

*<proof>*

**end**

## 12 Concrete Matrices

**theory** *DL-Concrete-Matrices*

**imports** *Jordan-Normal-Form.Matrix*

**begin**

The following definition allows non-square-matrices, `mat_one (mat_one n)` only allows square matrices.

**definition** *id-matrix*:: $nat \Rightarrow nat \Rightarrow real \ mat$

**where**  $id\_matrix \ nr \ nc = mat \ nr \ nc \ (\lambda(r, c). \ if \ r=c \ then \ 1 \ else \ 0)$

**lemma** *id-matrix-dim*:  $dim\_row \ (id\_matrix \ nr \ nc) = nr \ dim\_col \ (id\_matrix \ nr \ nc) = nc$  *<proof>*

**lemma** *row-id-matrix*:

**assumes**  $i < nr$

**shows**  $row \ (id\_matrix \ nr \ nc) \ i = unit\_vec \ nc \ i$

*<proof>*



**lemma** *unit-eq-0[simp]*:  
**assumes**  $i: i \geq n$   
**shows**  $\text{unit-vec } n \ i = 0_v \ n$   
 $\langle \text{proof} \rangle$

**lemma** *mult-id-matrix*:  
**assumes**  $i < nr$   
**shows**  $(\text{id-matrix } nr \ (\text{dim-vec } v) *_{\mathbb{R}} v) \ \$ \ i = (\text{if } i < \text{dim-vec } v \ \text{then } v \ \$ \ i \ \text{else } 0) \ (\text{is } ?a \ \$ \ i = ?b)$   
 $\langle \text{proof} \rangle$

**definition** *all1-vec::nat  $\Rightarrow$  real vec*  
**where**  $\text{all1-vec } n = \text{vec } n \ (\lambda i. 1)$

**definition** *all1-matrix::nat  $\Rightarrow$  nat  $\Rightarrow$  real mat*  
**where**  $\text{all1-matrix } nr \ nc = \text{mat } nr \ nc \ (\lambda(r, c). 1)$

**lemma** *all1-matrix-dim*:  $\text{dim-row } (\text{all1-matrix } nr \ nc) = nr \ \text{dim-col } (\text{all1-matrix } nr \ nc) = nc$   
 $\langle \text{proof} \rangle$

**lemma** *row-all1-matrix*:  
**assumes**  $i < nr$   
**shows**  $\text{row } (\text{all1-matrix } nr \ nc) \ i = \text{all1-vec } nc$   
 $\langle \text{proof} \rangle$

**lemma** *all1-vec-scalar-prod*:  
**shows**  $\text{all1-vec } (\text{length } xs) \cdot (\text{vec-of-list } xs) = \text{sum-list } xs$   
 $\langle \text{proof} \rangle$

**lemma** *mult-all1-matrix*:  
**assumes**  $i < nr$   
**shows**  $((\text{all1-matrix } nr \ (\text{dim-vec } v)) *_{\mathbb{R}} v) \ \$ \ i = \text{sum-list } (\text{list-of-vec } v) \ (\text{is } ?a \ \$ \ i = \text{sum-list } (\text{list-of-vec } v))$   
 $\langle \text{proof} \rangle$

**definition** *copy-first-matrix::nat  $\Rightarrow$  nat  $\Rightarrow$  real mat*  
**where**  $\text{copy-first-matrix } nr \ nc = \text{mat } nr \ nc \ (\lambda(r, c). \text{if } c = 0 \ \text{then } 1 \ \text{else } 0)$

**lemma** *copy-first-matrix-dim*:  $\text{dim-row } (\text{copy-first-matrix } nr \ nc) = nr \ \text{dim-col } (\text{copy-first-matrix } nr \ nc) = nc$   
 $\langle \text{proof} \rangle$

**lemma** *row-copy-first-matrix*:  
**assumes**  $i < nr$

**shows** *row (copy-first-matrix nr nc) i = unit-vec nc 0*  
⟨*proof*⟩

**lemma** *mult-copy-first-matrix:*

**assumes**  $i < nr$  **and**  $dim-vec\ v > 0$

**shows**  $(copy-first-matrix\ nr\ (dim-vec\ v)\ *_v\ v)\ \$\ i = v\ \$\ 0$  (**is**  $?a\ \$\ i = v\ \$\ 0$ )  
⟨*proof*⟩

**end**

## 13 Missing Lemmas of Finite\_Set

**theory** *DL-Missing-Finite-Set*

**imports** *Main*

**begin**

**lemma** *card-even[simp]:*  $card\ \{a \in Collect\ even.\ a < 2 * n\} = n$   
⟨*proof*⟩

**lemma** *card-odd[simp]:*  $card\ \{a \in Collect\ odd.\ a < 2 * n\} = n$   
⟨*proof*⟩

**end**

## 14 Deep Network Model

**theory** *DL-Deep-Model*

**imports** *DL-Network Tensor-Matricization Jordan-Normal-Form.DL-Submatrix DL-Concrete-Matrices*  
*DL-Missing-Finite-Set Jordan-Normal-Form.DL-Missing-Sublist Jordan-Normal-Form.Determinant*

**begin**

**hide-const**(**open**) *Polynomial.order*

**hide-const** (**open**) *Matrix.unit-vec*

**fun** *deep-model* **and** *deep-model'* **where**

*deep-model'*  $Y\ [] = Input\ Y\ |$

*deep-model'*  $Y\ (r\ \# rs) = Pool\ (deep-model\ Y\ r\ rs)\ (deep-model\ Y\ r\ rs)\ |$

*deep-model*  $Y\ r\ rs = Conv\ (Y,r)\ (deep-model'\ r\ rs)$

**abbreviation** *deep-model'-l*  $rs == deep-model'\ (rs!0)\ (tl\ rs)$

**abbreviation** *deep-model-l*  $rs == deep-model\ (rs!0)\ (rs!1)\ (tl\ (tl\ rs))$

**lemma** *valid-deep-model:* *valid-net (deep-model Y r rs)*  
⟨*proof*⟩

**lemma** *valid-deep-model':* *valid-net (deep-model' r rs)*  
⟨*proof*⟩

**lemma** *input-sizes-deep-model'*:  
**assumes**  $\text{length } rs \geq 1$   
**shows**  $\text{input-sizes } (\text{deep-model}'\text{-l } rs) = \text{replicate } (2^{\wedge}(\text{length } rs - 1)) \text{ (last } rs)$   
 $\langle \text{proof} \rangle$

**lemma** *input-sizes-deep-model*:  
**assumes**  $\text{length } rs \geq 2$   
**shows**  $\text{input-sizes } (\text{deep-model}\text{-l } rs) = \text{replicate } (2^{\wedge}(\text{length } rs - 2)) \text{ (last } rs)$   
 $\langle \text{proof} \rangle$

**lemma** *evaluate-net-Conv-id*:  
**assumes**  $\text{valid-net}' m$   
**and**  $\text{input-sizes } m = \text{map } \text{dim-vec } \text{input}$   
**and**  $j < nr$   
**shows**  $\text{evaluate-net } (\text{Conv } (\text{id-matrix } nr \text{ (output-size}' m)) m) \text{ input } \$ j$   
 $= (\text{if } j < \text{output-size}' m \text{ then } \text{evaluate-net } m \text{ input } \$ j \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma** *tensors-from-net-Conv-id*:  
**assumes**  $\text{valid-net}' m$   
**and**  $i < nr$   
**shows**  $\text{tensors-from-net } (\text{Conv } (\text{id-matrix } nr \text{ (output-size}' m)) m) \$ i$   
 $= (\text{if } i < \text{output-size}' m \text{ then } \text{tensors-from-net } m \$ i \text{ else } \text{tensor0 } (\text{input-sizes } m))$   
 $(\text{is } ?a \$ i = ?b)$   
 $\langle \text{proof} \rangle$

**lemma** *evaluate-net-Conv-copy-first*:  
**assumes**  $\text{valid-net}' m$   
**and**  $\text{input-sizes } m = \text{map } \text{dim-vec } \text{input}$   
**and**  $j < nr$   
**and**  $\text{output-size}' m > 0$   
**shows**  $\text{evaluate-net } (\text{Conv } (\text{copy-first-matrix } nr \text{ (output-size}' m)) m) \text{ input } \$ j$   
 $= \text{evaluate-net } m \text{ input } \$ 0$   
 $\langle \text{proof} \rangle$

**lemma** *tensors-from-net-Conv-copy-first*:  
**assumes**  $\text{valid-net}' m$   
**and**  $i < nr$   
**and**  $\text{output-size}' m > 0$   
**shows**  $\text{tensors-from-net } (\text{Conv } (\text{copy-first-matrix } nr \text{ (output-size}' m)) m) \$ i =$   
 $\text{tensors-from-net } m \$ 0$   
 $(\text{is } ?a \$ i = ?b)$   
 $\langle \text{proof} \rangle$

**lemma** *evaluate-net-Conv-all1*:  
**assumes**  $\text{valid-net}' m$   
**and**  $\text{input-sizes } m = \text{map } \text{dim-vec } \text{input}$   
**and**  $i < nr$

**shows** *evaluate-net* (*Conv* (*all1-matrix* *nr* (*output-size'* *m*)) *m*) *input* \$ *i*  
 = *Groups-List.sum-list* (*list-of-vec* (*evaluate-net* *m* *input*))  
 ⟨*proof*⟩

**lemma** *tensors-from-net-Conv-all1*:

**assumes** *valid-net'* *m*

**and**  $i < nr$

**shows** *tensors-from-net* (*Conv* (*all1-matrix* *nr* (*output-size'* *m*)) *m*) \$ *i*  
 = *listsum* (*input-sizes* *m*) (*list-of-vec* (*tensors-from-net* *m*))  
 (**is** ?*a* \$ *i* = ?*b*)  
 ⟨*proof*⟩

**fun** *witness* **and** *witness'* **where**

*witness'* *Y* [] = *Input* *Y* |

*witness'* *Y* (*r* # *rs*) = *Pool* (*witness* *Y* *r* *rs*) (*witness* *Y* *r* *rs*) |

*witness* *Y* *r* *rs* = *Conv* ((*if* *length* *rs* = 0 *then* *id-matrix* *else* (*if* *length* *rs* = 1 *then* *all1-matrix* *else* *copy-first-matrix*)) *Y* *r*) (*witness'* *r* *rs*)

**abbreviation** *witness-l* *rs* == *witness* (*rs!*0) (*rs!*1) (*tl* (*tl* *rs*))

**abbreviation** *witness'-l* *rs* == *witness'* (*rs!*0) (*tl* *rs*)

**lemma** *witness-is-deep-model: remove-weights* (*witness* *Y* *r* *rs*) = *deep-model* *Y* *r* *rs*  
 ⟨*proof*⟩

**lemma** *witness'-is-deep-model: remove-weights* (*witness'* *Y* *rs*) = *deep-model'* *Y* *rs*  
 ⟨*proof*⟩

**lemma** *witness-valid: valid-net'* (*witness* *Y* *r* *rs*)  
 ⟨*proof*⟩

**lemma** *witness'-valid: valid-net'* (*witness'* *Y* *rs*)  
 ⟨*proof*⟩

**lemma** *shared-weight-net-witness: shared-weight-net* (*witness* *Y* *r* *rs*)  
 ⟨*proof*⟩

**lemma** *witness-l0'*: *witness'* *Y* [*M*] =  
 (*Pool*  
 (*Conv* (*id-matrix* *Y* *M*) (*Input* *M*))  
 (*Conv* (*id-matrix* *Y* *M*) (*Input* *M*))  
 )  
 ⟨*proof*⟩

**lemma** *witness-l1*: *witness* *Y* *r0* [*M*] =  
*Conv* (*all1-matrix* *Y* *r0*) (*witness'* *r0* [*M*])  
 ⟨*proof*⟩

**lemma** *tensors-ht-l0*:

**assumes**  $j < r0$

**shows**  $\text{tensors-from-net } (\text{Conv } (\text{id-matrix } r0 \ M) \ (\text{Input } M)) \ \$ \ j$

$= (\text{if } j < M \ \text{then } \text{unit-vec } M \ j \ \text{else } \text{tensor0 } [M])$

$\langle \text{proof} \rangle$

**lemma**  $\text{tensor-prod-unit-vec}$ :

$\text{unit-vec } M \ j \ \otimes \ \text{unit-vec } M \ j = \text{tensor-from-lookup } [M, M] \ (\lambda is. \ \text{if } is = [j, j] \ \text{then } 1 \ \text{else } 0) \ (\text{is } ?A = ?B)$

$\langle \text{proof} \rangle$

**lemma**  $\text{tensors-ht-l0}'$ :

**assumes**  $j < r0$

**shows**  $\text{tensors-from-net } (\text{witness}' \ r0 \ [M]) \ \$ \ j$

$= (\text{if } j < M \ \text{then } \text{unit-vec } M \ j \ \otimes \ \text{unit-vec } M \ j \ \text{else } \text{tensor0 } [M, M]) \ (\text{is } - = ?b)$

$\langle \text{proof} \rangle$

**lemma**  $\text{lookup-tensors-ht-l0}'$ :

**assumes**  $j < r0$

**and**  $is \triangleleft [M, M]$

**shows**  $(\text{Tensor.lookup } (\text{tensors-from-net } (\text{witness}' \ r0 \ [M]) \ \$ \ j)) \ is = (\text{if } is = [j, j] \ \text{then } 1 \ \text{else } 0)$

$\langle \text{proof} \rangle$

**lemma**  $\text{lookup-tensors-ht-l1}$ :

**assumes**  $j < r1$

**and**  $is \triangleleft [M, M]$

**shows**  $\text{Tensor.lookup } (\text{tensors-from-net } (\text{witness } r1 \ r0 \ [M]) \ \$ \ j) \ is$

$= (\text{if } is!0 = is!1 \ \wedge \ is!0 < r0 \ \text{then } 1 \ \text{else } 0)$

$\langle \text{proof} \rangle$

**lemma**  $\text{length-output-deep-model}$ :

**assumes**  $\text{remove-weights } m = \text{deep-model-l } rs$

**shows**  $\text{dim-vec } (\text{tensors-from-net } m) = rs \ ! \ 0$

$\langle \text{proof} \rangle$

**lemma**  $\text{length-output-deep-model}'$ :

**assumes**  $\text{remove-weights } m = \text{deep-model}'\text{-l } rs$

**shows**  $\text{dim-vec } (\text{tensors-from-net } m) = rs \ ! \ 0$

$\langle \text{proof} \rangle$

**lemma**  $\text{length-output-witness}$ :

$\text{dim-vec } (\text{tensors-from-net } (\text{witness-l } rs)) = rs \ ! \ 0$

$\langle \text{proof} \rangle$

**lemma**  $\text{length-output-witness}'$ :

$\text{dim-vec } (\text{tensors-from-net } (\text{witness}'\text{-l } rs)) = rs \ ! \ 0$

$\langle \text{proof} \rangle$

**lemma** *dims-output-deep-model*:  
**assumes**  $\text{length } rs \geq 2$   
**and**  $\bigwedge r. r \in \text{set } rs \implies r > 0$   
**and**  $j < \text{rs!}0$   
**and** *remove-weights*  $m = \text{deep-model-l } rs$   
**shows**  $\text{Tensor.dims } (\text{tensors-from-net } m \ \$ \ j) = \text{replicate } (2^{\wedge}(\text{length } rs - 2)) \ (\text{last } rs)$   
 $\langle \text{proof} \rangle$

**lemma** *dims-output-witness*:  
**assumes**  $\text{length } rs \geq 2$   
**and**  $\bigwedge r. r \in \text{set } rs \implies r > 0$   
**and**  $j < \text{rs!}0$   
**shows**  $\text{Tensor.dims } (\text{tensors-from-net } (\text{witness-l } rs) \ \$ \ j) = \text{replicate } (2^{\wedge}(\text{length } rs - 2)) \ (\text{last } rs)$   
 $\langle \text{proof} \rangle$

**lemma** *dims-output-deep-model'*:  
**assumes**  $\text{length } rs \geq 1$   
**and**  $\bigwedge r. r \in \text{set } rs \implies r > 0$   
**and**  $j < \text{rs!}0$   
**and** *remove-weights*  $m = \text{deep-model'-l } rs$   
**shows**  $\text{Tensor.dims } (\text{tensors-from-net } m \ \$ \ j) = \text{replicate } (2^{\wedge}(\text{length } rs - 1)) \ (\text{last } rs)$   
 $\langle \text{proof} \rangle$

**lemma** *dims-output-witness'*:  
**assumes**  $\text{length } rs \geq 1$   
**and**  $\bigwedge r. r \in \text{set } rs \implies r > 0$   
**and**  $j < \text{rs!}0$   
**shows**  $\text{Tensor.dims } (\text{tensors-from-net } (\text{witness'-l } rs) \ \$ \ j) = \text{replicate } (2^{\wedge}(\text{length } rs - 1)) \ (\text{last } rs)$   
 $\langle \text{proof} \rangle$

**abbreviation**  $\text{ten2mat} == \text{matricize } \{n. \text{even } n\}$   
**abbreviation**  $\text{mat2ten} == \text{dematricize } \{n. \text{even } n\}$

**locale** *deep-model-correct-params* =  
**fixes** *shared-weights*::*bool*  
**fixes** *rs*::*nat list*  
**assumes**  $\text{deep:length } rs \geq 3$   
**and** *no-zeros*: $\bigwedge r. r \in \text{set } rs \implies 0 < r$   
**begin**

**definition**  $r = \text{min } (\text{last } rs) \ (\text{last } (\text{butlast } rs))$

**definition**  $N\text{-half} = 2^{\wedge}(\text{length } rs - 3)$

**definition** *weight-space-dim* = *count-weights* *shared-weights* (*deep-model-l* *rs*)

**end**

**locale** *deep-model-correct-params-y* = *deep-model-correct-params* +  
**fixes** *y::nat*  
**assumes** *y-valid:y < rs ! 0*  
**begin**

**definition** *A* :: (*nat*  $\Rightarrow$  *real*)  $\Rightarrow$  *real tensor*  
**where** *A ws* = *tensors-from-net (insert-weights shared-weights (deep-model-l rs) ws) \$ y*

**definition** *A'* :: (*nat*  $\Rightarrow$  *real*)  $\Rightarrow$  *real mat*  
**where** *A' ws* = *ten2mat (A ws)*

**lemma** *dims-tensor-deep-model:*

**assumes** *remove-weights m = deep-model-l rs*

**shows** *dims (tensors-from-net m \$ y) = replicate (2 \* N-half) (last rs)*  
 $\langle$ *proof* $\rangle$

**lemma** *order-tensor-deep-model:*

**assumes** *remove-weights m = deep-model-l rs*

**shows** *order (tensors-from-net m \$ y) = 2 \* N-half*  
 $\langle$ *proof* $\rangle$

**lemma** *dims-A:*

**shows** *Tensor.dims (A ws) = replicate (2 \* N-half) (last rs)*  
 $\langle$ *proof* $\rangle$

**lemma** *order-A:*

**shows** *order (A ws) = 2 \* N-half*  $\langle$ *proof* $\rangle$

**lemma** *dims-A':*

**shows** *dim-row (A' ws) = prod-list (nth (Tensor.dims (A ws)) {n. even n})*

**and** *dim-col (A' ws) = prod-list (nth (Tensor.dims (A ws)) {n. odd n})*  
 $\langle$ *proof* $\rangle$

**lemma** *dims-A'-pow:*

**shows** *dim-row (A' ws) = (last rs) ^ N-half* *dim-col (A' ws) = (last rs) ^ N-half*  
 $\langle$ *proof* $\rangle$

**definition** *Aw* = *tensors-from-net (witness-l rs) \$ y*

**definition** *Aw'* = *ten2mat Aw*

**definition** *witness-weights* = *extract-weights shared-weights (witness-l rs)*

**lemma** *witness-weights:witness-l rs = insert-weights shared-weights (deep-model-l rs) witness-weights*

*<proof>*

**lemma** *Aw-def'*:  $Aw = A$  witness-weights *<proof>*

**lemma** *Aw'-def'*:  $Aw' = A'$  witness-weights *<proof>*

**lemma** *dims-Aw*:  $Tensor.dims Aw = replicate (2 * N-half) (last rs)$   
*<proof>*

**lemma** *order-Aw*:  $order Aw = 2 * N-half$   
*<proof>*

**lemma** *dims-Aw'*:  
 $dim-row Aw' = prod-list (nth (Tensor.dims Aw) \{n. even n\})$   
 $dim-col Aw' = prod-list (nth (Tensor.dims Aw) \{n. odd n\})$   
*<proof>*

**lemma** *dims-Aw'-pow*:  $dim-row Aw' = (last rs) \wedge N-half$   
 $dim-col Aw' = (last rs) \wedge N-half$   
*<proof>*

**lemma** *witness-tensor*:

**assumes**  $is \triangleleft Tensor.dims Aw$

**shows**  $Tensor.lookup Aw is$

$= (if\ nth\ is\ \{n. even\ n\} = nth\ is\ \{n. odd\ n\} \wedge (\forall i \in set\ is. i < last\ (butlast\ rs))\ then\ 1\ else\ 0)$

*<proof>*

**lemma** *witness-matricization*:

**assumes**  $i < dim-row Aw'$  and  $j < dim-col Aw'$

**shows**  $Aw' \$\$ (i, j)$

$= (if\ i=j \wedge (\forall i0 \in set\ (digit-encode\ (nth\ (Tensor.dims\ Aw)\ \{n. even\ n\})\ i). i0 < last\ (butlast\ rs))\ then\ 1\ else\ 0)$

*<proof>*

**definition** *rows-with-1* =  $\{i. (\forall i0 \in set\ (digit-encode\ (nth\ (Tensor.dims\ Aw)\ \{n. even\ n\})\ i). i0 < last\ (butlast\ rs))\}$

**lemma** *card-low-digits*:

**assumes**  $m > 0 \wedge d. d \in set\ ds \implies m \leq d$

**shows**  $card\ \{i. i < prod-list\ ds \wedge (\forall i0 \in set\ (digit-encode\ ds\ i). i0 < m)\} = m \wedge (length\ ds)$

*<proof>*

**lemma** *card-rows-with-1*:  $card\ \{i \in rows-with-1. i < dim-row Aw'\} = r \wedge N-half$   
*<proof>*



**lemma** *infinite-rows-with-1*: *infinite rows-with-1*  
<proof>

**lemma** *witness-submatrix*: *submatrix Aw' rows-with-1 rows-with-1 = 1<sub>m</sub> (r<sup>^</sup>N-half)*  
<proof>

**lemma** *witness-det*: *det (submatrix Aw' rows-with-1 rows-with-1) ≠ 0* <proof>

**end**

**interpretation** *example* : *deep-model-correct-params False [10,10,10]*  
<proof>

**interpretation** *example* : *deep-model-correct-params-y False [10,10,10] 1*  
<proof>

**end**

## 15 Polynomials representing the Deep Network Model

**theory** *DL-Deep-Model-Poly*

**imports** *DL-Deep-Model Polynomials.More-MPoly-Type Jordan-Normal-Form.Determinant*  
**begin**

**lemma** *polyfun-det*:

**assumes**  $\bigwedge x. (A\ x) \in \text{carrier-mat } n\ n$

**assumes**  $\bigwedge x\ i\ j. i < n \implies j < n \implies \text{polyfun } N\ (\lambda x. (A\ x)\ \S\S\ (i,j))$

**shows** *polyfun*  $N\ (\lambda x. \text{det } (A\ x))$

<proof>

**lemma** *polyfun-extract-matrix*:

**assumes**  $i < m\ j < n$

**shows** *polyfun*  $\{.. < a + (m * n + c)\} (\lambda f. \text{extract-matrix } (\lambda i. f\ (i + a))\ m\ n\ \S\S\ (i,j))$

<proof>

**lemma** *polyfun-mult-mat-vec*:

**assumes**  $\bigwedge x. v\ x \in \text{carrier-vec } n$

**assumes**  $\bigwedge j. j < n \implies \text{polyfun } N\ (\lambda x. v\ x\ \$\ j)$

**assumes**  $\bigwedge x. A\ x \in \text{carrier-mat } m\ n$

**assumes**  $\bigwedge i\ j. i < m \implies j < n \implies \text{polyfun } N\ (\lambda x. A\ x\ \S\S\ (i,j))$

**assumes**  $j < m$

**shows** *polyfun*  $N\ (\lambda x. ((A\ x) *_v (v\ x))\ \$\ j)$

<proof>

**lemma** *polyfun-evaluate-net-plus-a*:

**assumes**  $\text{map dim-vec inputs} = \text{input-sizes } m$   
**assumes**  $\text{valid-net } m$   
**assumes**  $j < \text{output-size } m$   
**shows**  $\text{polyfun } \{..<a + \text{count-weights } s\ m\} (\lambda f. \text{evaluate-net } (\text{insert-weights } s\ m$   
 $(\lambda i. f\ (i + a)))\ \text{inputs } \$ j)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{polyfun-evaluate-net}$ :  
**assumes**  $\text{map dim-vec inputs} = \text{input-sizes } m$   
**assumes**  $\text{valid-net } m$   
**assumes**  $j < \text{output-size } m$   
**shows**  $\text{polyfun } \{..<\text{count-weights } s\ m\} (\lambda f. \text{evaluate-net } (\text{insert-weights } s\ m\ f)$   
 $\text{inputs } \$ j)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{polyfun-tensors-from-net}$ :  
**assumes**  $\text{valid-net } m$   
**assumes**  $is \triangleleft \text{input-sizes } m$   
**assumes**  $j < \text{output-size } m$   
**shows**  $\text{polyfun } \{..<\text{count-weights } s\ m\} (\lambda f. \text{Tensor.lookup } (\text{tensors-from-net } (\text{insert-weights}$   
 $s\ m\ f)\ \$ j)\ is)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{polyfun-matricize}$ :  
**assumes**  $\bigwedge x. \text{dims } (T\ x) = ds$   
**assumes**  $\bigwedge is. is \triangleleft ds \implies \text{polyfun } N (\lambda x. \text{Tensor.lookup } (T\ x)\ is)$   
**assumes**  $\bigwedge x. \text{dim-row } (\text{matricize } I\ (T\ x)) = nr$   
**assumes**  $\bigwedge x. \text{dim-col } (\text{matricize } I\ (T\ x)) = nc$   
**assumes**  $i < nr$   
**assumes**  $j < nc$   
**shows**  $\text{polyfun } N (\lambda x. \text{matricize } I\ (T\ x)\ \$\$ (i,j))$   
 $\langle \text{proof} \rangle$

**lemma**  $(\neg (a::nat) < b) = (a \geq b)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{polyfun-submatrix}$ :  
**assumes**  $\bigwedge x. (A\ x) \in \text{carrier-mat } m\ n$   
**assumes**  $\bigwedge x\ i\ j. i < m \implies j < n \implies \text{polyfun } N (\lambda x. (A\ x)\ \$\$ (i,j))$   
**assumes**  $i < \text{card } \{i. i < m \wedge i \in I\}$   
**assumes**  $j < \text{card } \{j. j < n \wedge j \in J\}$   
**assumes**  $\text{infinite } I\ \text{infinite } J$   
**shows**  $\text{polyfun } N (\lambda x. (\text{submatrix } (A\ x)\ I\ J)\ \$\$ (i,j))$   
 $\langle \text{proof} \rangle$

**context**  $\text{deep-model-correct-params-}y$   
**begin**

**definition**  $\text{witness-submatrix}$  **where**

*witness-submatrix f = submatrix (A' f) rows-with-1 rows-with-1*

**lemma** *polyfun-tensor-deep-model:*

**assumes** *is < input-sizes (deep-model-l rs)*

**shows** *polyfun {..*weight-space-dim*}*

*(λf. Tensor.lookup (tensors-from-net (insert-weights shared-weights (deep-model-l rs) f) \$ y) is)*

*<proof>*

**lemma** *input-sizes-deep-model: input-sizes (deep-model-l rs) = replicate (2 \* N-half)*

*(last rs)*

*<proof>*

**lemma** *polyfun-matrix-deep-model:*

**assumes** *i < (last rs) ^ N-half*

**assumes** *j < (last rs) ^ N-half*

**shows** *polyfun {..*weight-space-dim*} (λf. A' f \$\$ (i,j))*

*<proof>*

**lemma** *polyfun-submatrix-deep-model:*

**assumes** *i < r ^ N-half*

**assumes** *j < r ^ N-half*

**shows** *polyfun {..*weight-space-dim*} (λf. witness-submatrix f \$\$ (i,j))*

*<proof>*

**lemma** *polyfun-det-deep-model:*

**shows** *polyfun {..*weight-space-dim*} (λf. det (witness-submatrix f))*

*<proof>*

**end**

**end**

## 16 Alternative Lebesgue Measure Definition

**theory** *Lebesgue-Functional*

**imports** *HOL-Analysis.Lebesgue-Measure*

**begin**

Lebesgue\_Measure.lborel is defined on the typeclass euclidean\_space, which does not allow the space dimension to be dependent on a variable. As the Lebesgue measure of higher dimensions is the product measure of the one dimensional Lebesgue measure, we can easily define a more flexible version of the Lebesgue measure as follows. This version of the Lebesgue measure measures sets of functions from nat to real whose values are undefined for arguments higher than n. These "Extensional Function Spaces" are defined in HOL/FuncSet.

**definition**  $lborel-f :: nat \Rightarrow (nat \Rightarrow real) \text{ measure where}$   
 $lborel-f\ n = (\Pi_M\ b \in \{..<n\}. lborel)$

**lemma** *product-sigma-finite-interval*:  $product-sigma-finite (\lambda b. interval-measure (\lambda x. x))$   
 $\langle proof \rangle$

**lemma** *l-borel-f-1*:  $distr (lborel-f\ 1) lborel (\lambda x. x\ 0) = lborel$   
 $\langle proof \rangle$

**lemma** *space-lborel-f*:  $space (lborel-f\ n) = Pi_E\ \{..<n\} (\lambda-. UNIV) \langle proof \rangle$

**lemma** *space-lborel-f-subset*:  $space (lborel-f\ n) \subseteq space (lborel-f\ (Suc\ n))$   
 $\langle proof \rangle$

**lemma** *space-lborel-add-dim*:  
**assumes**  $f \in space (lborel-f\ n)$   
**shows**  $f(n := x) \in space (lborel-f\ (Suc\ n))$   
 $\langle proof \rangle$

**lemma** *integral-lborel-f*:  
**assumes**  $f \in borel-measurable (lborel-f\ (Suc\ n))$   
**shows**  $integral^N (lborel-f\ (Suc\ n))\ f = \int^+ y. \int^+ x. f (x(n := y))\ \partial lborel-f\ n$   
 $\langle proof \rangle$

**lemma** *emeasure-lborel-f-Suc*:  
**assumes**  $A \in sets (lborel-f\ (Suc\ n))$   
**assumes**  $\bigwedge y. \{x \in space (lborel-f\ n). x(n := y) \in A\} \in sets (lborel-f\ n)$   
**shows**  $emeasure (lborel-f\ (Suc\ n))\ A = \int^+ y. emeasure (lborel-f\ n)\ \{x \in space (lborel-f\ n). x(n := y) \in A\}\ \partial lborel-f\ n$   
 $\langle proof \rangle$

**lemma** *lborel-f-measurable-add-dim*:  $(\lambda f. f(n := x)) \in measurable (lborel-f\ n) (lborel-f\ (Suc\ n))$   
 $\langle proof \rangle$

**lemma** *sets-lborel-f-sub-dim*:  
**assumes**  $A \in sets (lborel-f\ (Suc\ n))$   
**shows**  $\{x. x(n := y) \in A\} \cap space (lborel-f\ n) \in sets (lborel-f\ n)$   
 $\langle proof \rangle$

**lemma** *lborel-f-measurable-restrict*:  
**assumes**  $m \leq n$   
**shows**  $(\lambda f. restrict\ f\ \{..<m\}) \in measurable (lborel-f\ n) (lborel-f\ m)$   
 $\langle proof \rangle$

**lemma** *lborel-measurable-sub-dim*:  $(\lambda f. restrict\ f\ \{..<n\}) \in measurable (lborel-f\ (Suc\ n)) (lborel-f\ n)$

*<proof>*

**lemma** *measurable-lborel-component* [*measurable*]:

**assumes**  $k < n$

**shows**  $(\lambda x. x k) \in \text{borel-measurable } (\text{lborel-f } n)$

*<proof>*

**end**

## 17 Lebesgue Measure of Polynomial Zero Sets

**theory** *Lebesgue-Zero-Set*

**imports**

*Polynomials.More-MPoly-Type*

*Lebesgue-Functional*

*Polynomials.MPoly-Type-Univariate*

**begin**

**lemma** *measurable-insertion* [*measurable*]:

**assumes**  $\text{vars } p \subseteq \{..<n\}$

**shows**  $(\lambda f. \text{insertion } f p) \in \text{borel-measurable } (\text{lborel-f } n)$

*<proof>*

This proof follows Richard Caron and Tim Traynor, "The zero set of a polynomial" <http://www1.uwindsor.ca/math/sites/uwindsor.ca.math/files/05-03.pdf>

**lemma** *lebesgue-mpoly-zero-set*:

**fixes**  $p::\text{real mpoly}$

**assumes**  $p \neq 0 \text{ vars } p \subseteq \{..<n\}$

**shows**  $\{f \in \text{space } (\text{lborel-f } n). \text{insertion } f p = 0\} \in \text{null-sets } (\text{lborel-f } n)$

*<proof>*

**end**

## 18 Shallow Network Model

**theory** *DL-Shallow-Model*

**imports** *DL-Network Tensor-Rank*

**begin**

**fun** *shallow-model'* **where**

*shallow-model' Z M 0 = Conv (Z,M) (Input M) |*

*shallow-model' Z M (Suc N) = Pool (shallow-model' Z M 0) (shallow-model' Z M*

*N)*

**definition** *shallow-model* **where**

*shallow-model Y Z M N = Conv (Y,Z) (shallow-model' Z M N)*

**lemma** *valid-shallow-model'*: *valid-net (shallow-model' Z M N)*  
⟨*proof*⟩

**lemma** *output-size-shallow-model'*: *output-size (shallow-model' Z M N) = Z*  
⟨*proof*⟩

**lemma** *valid-shallow-model*: *valid-net (shallow-model Y Z M N)*  
⟨*proof*⟩

**lemma** *output-size-shallow-model*: *output-size (shallow-model Y Z M N) = Y*  
⟨*proof*⟩

**lemma** *input-sizes-shallow-model*: *input-sizes (shallow-model Y Z M N) = replicate (Suc N) M*  
⟨*proof*⟩

**lemma** *balanced-net-shallow-model'*: *balanced-net (shallow-model' Z M N)*  
⟨*proof*⟩

**lemma** *balanced-net-shallow-model*: *balanced-net (shallow-model Y Z M N)*  
⟨*proof*⟩

**lemma** *cprank-max1-shallow-model'*:

**assumes**  $y < \text{output-size (shallow-model' Z M N)}$

**shows** *cprank-max1 (tensors-from-net (insert-weights s (shallow-model' Z M N) w) \$ y)*  
⟨*proof*⟩

**lemma** *cprank-shallow-model*:

**assumes**  $m = \text{insert-weights s (shallow-model Y Z M N) w}$

**assumes**  $y < Y$

**shows** *cprank (tensors-from-net m \$ y) ≤ Z*  
⟨*proof*⟩

end

## 19 Fundamental Theorem of Network Capacity

**theory** *DL-Fundamental-Theorem-Network-Capacity*

**imports** *DL-Rank-CP-Rank DL-Deep-Model-Poly Lebesgue-Zero-Set*

*Jordan-Normal-Form.DL-Rank-Submatrix HOL-Analysis.Complete-Measure DL-Shallow-Model*

**begin**

**context** *deep-model-correct-params-y*

**begin**

**definition** *polynomial-f w = det (submatrix (matricize {n. even n} (A w)) rows-with-1*

*rows-with-1*)

**lemma** *polyfun-polynomial*:

**shows** *polyfun*  $\{..<weight-space-dim\}$  *polynomial-f*  
*<proof>*

**definition** *polynomial-p* = (*SOME* *p*. *vars* *p*  $\subseteq \{..<weight-space-dim\}$   $\wedge (\forall x$ . *insertion* *x* *p* = *polynomial-f* *x*))

**lemma**

*polynomial-p-not-0*: *polynomial-p*  $\neq 0$  **and**  
*vars-polynomial-p*: *vars* *polynomial-p*  $\subseteq \{..<weight-space-dim\}$  **and**  
*polynomial-pf*:  $\bigwedge w$ . *insertion* *w* *polynomial-p* = *polynomial-f* *w*  
*<proof>*

**lemma** *if-polynomial-0-rank*:

**assumes** *polynomial-f* *w*  $\neq 0$   
**shows**  $r \wedge N\text{-half} \leq \text{cprank } (A \ w)$   
*<proof>*

**lemma** *if-polynomial-0-evaluate*:

**assumes** *polynomial-f* *wd*  $\neq 0$   
**assumes**  $\forall$  *inputs*. *input-sizes* (*deep-model-l* *rs*) = *map dim-vec inputs*  $\longrightarrow$  *evaluate-net* (*insert-weights shared-weights* (*deep-model-l* *rs*) *wd*) *inputs*  
= *evaluate-net* (*insert-weights shared-weights* (*shallow-model* (*rs* ! 0) *Z* (*last* *rs*)  
( $2 * N\text{-half} - 1$ )) *ws*) *inputs*)  
**shows**  $Z \geq r \wedge N\text{-half}$   
*<proof>*

**lemma** *if-polynomial-0-evaluate-notex*:

**assumes** *polynomial-f* *wd*  $\neq 0$   
**shows**  $\neg(\exists$  *weights-shallow* *Z*.  $Z < r \wedge N\text{-half} \wedge (\forall$  *inputs*. *input-sizes* (*deep-model-l*  
*rs*) = *map dim-vec inputs*  $\longrightarrow$   
*evaluate-net* (*insert-weights shared-weights* (*deep-model-l* *rs*) *wd*) *inputs*  
= *evaluate-net* (*insert-weights shared-weights* (*shallow-model* (*rs* ! 0) *Z* (*last* *rs*)  
( $2 * N\text{-half} - 1$ )) *ws*) *inputs*))  
*<proof>*

**theorem** *fundamental-theorem-network-capacity*:

*AE* *x* in *lborel-f weight-space-dim*.  $r \wedge N\text{-half} \leq \text{cprank } (A \ x)$   
*<proof>*

**theorem** *fundamental-theorem-network-capacity-v2*:

**shows** *AE* *wd* in *lborel-f weight-space-dim*.  
 $\neg(\exists$  *ws* *Z*.  $Z < r \wedge N\text{-half} \wedge (\forall$  *inputs*. *input-sizes* (*deep-model-l* *rs*) = *map*  
*dim-vec inputs*  $\longrightarrow$   
*evaluate-net* (*insert-weights shared-weights* (*deep-model-l* *rs*) *wd*) *inputs*  
= *evaluate-net* (*insert-weights shared-weights* (*shallow-model* (*rs* ! 0) *Z* (*last* *rs*))

$(2 * N - \text{half} - 1)$  ws) inputs))  
<proof>

**abbreviation** lebesgue-f **where** lebesgue-f n  $\equiv$  completion (lborel-f n)

**lemma** space-lebesgue-f: space (lebesgue-f n) =  $Pi_E \{.. < n\} (\lambda \cdot UNIV)$   
<proof>

**theorem** fundamental-theorem-network-capacity-v3:

**assumes**

$S = \{wd \in \text{space (lebesgue-f weight-space-dim)}.$

$\exists ws Z. Z < r \wedge N - \text{half} \wedge (\forall \text{inputs. input-sizes (deep-model-l rs) = map dim-vec inputs} \rightarrow$

$\text{evaluate-net (insert-weights shared-weights (deep-model-l rs) wd) inputs}$

$= \text{evaluate-net (insert-weights shared-weights (shallow-model (rs ! 0) Z (last rs) (2 * N - \text{half} - 1)) ws) inputs)\}$

**shows**  $S \in \text{null-sets (completion (lborel-f weight-space-dim))}$

<proof>

**end**

**end**

## References

- [1] A. Bentkamp. An Isabelle Formalization of the Expressiveness of Deep Learning. Master's thesis, Universität des Saarlandes, 2016. [http://matryoshka.gforge.inria.fr/bentkamp\\_msc\\_thesis.pdf](http://matryoshka.gforge.inria.fr/bentkamp_msc_thesis.pdf).
- [2] N. Cohen, O. Sharir, and A. Shashua. On the expressive power of deep learning: A tensor analysis. In V. Feldman, A. Rakhlin, and O. Shamir, editors, *Conference on Learning Theory (COLT 2016)*, volume 49 of *JMLR Workshop and Conference Proceedings*, pages 698–728. JMLR.org, 2016.