DEEP LEARNING

ERNEST YEUNG ERNESTYALUMNI@GMAIL.COM

From the beginning of 2016, I decided to cease all explicit crowdfunding for any of my materials on physics, math. I failed to raise *any* funds from previous crowdfunding efforts. I decided that if I was going to live in *abundance*, I must lose a scarcity attitude. I am committed to keeping all of my material **open-sourced**. I give all my stuff *for free*.

In the beginning of 2017, I received a very generous donation from a reader from Norway who found these notes useful, through *PayPal*. If you find these notes useful, feel free to donate directly and easily through PayPal, which won't go through a 3rd. party such as indiegogo, kickstarter, patreon. Also, you can donate to me on Venmo, Otherwise, under the *open-source MIT license*, feel free to copy, edit, paste, make your own versions, share, use as you wish.
gmail : ernestyalumni
linkedin : ernestyalumni
tumblr : ernestyalumni
twitter : ernestyalumni
youtube : ernestyalumni

Contents

Abstract. Everything about Machine Learning, Deep Learning.

Part 1. Neural Networks

1. Loss functions

1.1. **L1.** From Pytorch Docs, torch.nn, L1Loss Pytorch

Date: 22 November 2023.
Key words and phrases. Machine Learning, statistical inference, statistical inference learning, deep learning.

$$\ell(x,y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = |x_n - y_n|$$

where N is the batch size. If 'reduction' is not ''none'' (default ''mean''), then:

$$\ell(x,y) = \begin{cases} \text{mean}(L), & \text{if reduction} = \text{'mean'}; \\ \text{sum}(L), & \text{if reduction} = \text{'sum'}. \end{cases}$$

Instead, consider the following: let $\mathbf{x}, \mathbf{y} \in \mathbf{R}^N$ where $\mathbf{R}$ is some ring (integers and floating point types in computer sciecne are at least rings).

Let $\mathbf{x}$ be the predicted values and $\mathbf{y}$ be the true values.

If

$$l(x,y) = L = \{l_1, \dots l_N\}^T, \, l_n = |x_n - y_n|$$

then

$$\frac{\partial}{\partial x^i} L = \frac{\partial}{\partial x^i} \begin{cases} x_j - y_j & \text{if } x_j \geq y_j \\ y_j - x_j & \text{if } x_j < y_j \end{cases} = \begin{cases} 1 & \text{if } x_j \geq y_j \\ -1 & \text{if } x_j < y_j \end{cases}$$

If

$$l(x,y) = L = \frac{1}{N} \sum_{j=1}^N l_j$$

Then

$$\frac{\partial}{\partial x_i} L = \frac{1}{N} \begin{cases} 1 & \text{if } x_j \geq y_j \\ -1 & \text{if } x_j < y_j \end{cases}$$

If

$$l(x,y) = L = \sum_{j=1}^N l_j$$

Then

$$\frac{\partial}{\partial x_i} L = \begin{cases} 1 & \text{if } x_j \geq y_j \\ -1 & \text{if } x_j < y_j \end{cases}$$

Part 2. CUDA

Keep in mind that CUDA won't be the end-all, be-all for optimized deep learning. Consider Open XLA and consider LLVM.

2. Threads, Blocks, Grids

cf. Chapter 5 Thread Cooperation, Section 5.2. Splitting Parallel Blocks of Sanders and Kandrot (2010) [42].

Consider first a 1-dimensional block.

- `threadIdx.x` $\Longleftarrow M_x \equiv$ number of threads per block in x-direction. Let $j_x = 0 \dots M_x - 1$ be the index for the thread. Note that $1 \leq M_x \leq M_x^{\max}$, where $M_x^{\max} = 1024$ is the maximum number of threads per block that can be allocated, which is a hard hardware limit (I presume).
- `blockIdx.x` $\Longleftarrow N_x \equiv$ number of blocks in x-direction. Let $i_x = 0 \dots N_x - 1$
- `blockDim` stores number of threads along each dimension of the block M_x.

Then if we were to "linearize" or "flatten" in this x-direction,

$$k = j_x + i_x M_x \tag{1}$$

where k is the kth thread. $k = 0 \dots N_x M_x - 1$.

Take a look at heattexture1.cu which uses the GPU texture memory. Look at how `threadIdx`/`blockIdx` is mapped to pixel position.

As an exercise, let's again rewrite the code in mathematical notation:

- `threadIdx.x` $\Longleftarrow j_x$, $0 \leq j_x \leq M_x - 1$
- `blockIdx.x` $\Longleftarrow i_x$, $0 \leq i_x \leq N_x - 1$
- `blockDim.x` $\Longleftarrow M_x$, number of threads along each dimension (here dimension x) of a block, $1 \leq M_x \leq M_x^{\max} = 1024$
- `gridDim.x` $\Longleftarrow N_x$, $1 \leq N_x$

resulting in

- $k_x = j_x + i_x M_x \Longrightarrow$

```
int x =  threadIdx.x + blockIdx.x * blockDim.x ;
```

- $k_y = j_y + i_y M_y \Longrightarrow$

```
int y =  threadIdx.y + blockIdx.y * blockDim.y ;
```

and so for a "flattened" thread index $J \in \mathbb{N}$,

$$J = k_x + N_x \cdot M_x \cdot k_y$$

$\Longrightarrow$

```
offset = x + y * blockDim.x * gridDim.x ;
```

Suppose vector is of length N. So we *need* N parallel threads to launch, in total.

e.g. if $M_x = 128$ threads per block, $N/128 = N/M_x$ blocks to get our total of N threads running.

Wrinkle: integer division! e.g. if $N = 127$, $\frac{N}{128} = 0$.

Solution: consider $\frac{N+127}{128}$ blocks. If $N = l \cdot 128 + r$, $l \in \mathbb{N}$, $r = 0 \dots 127$.

$$\begin{aligned} \frac{N+127}{128} &= \frac{l \cdot 128 + r + 127}{128} = \frac{(l+1)128 + r - 1}{128} = \\ &= l + 1 + \frac{r-1}{128} = \begin{cases} l & \text{if } r = 0 \\ l+1 & \text{if } r = 1 \dots 127 \end{cases} \end{aligned}$$

$$\begin{aligned} \frac{N + (M_x - 1)}{M_x} &= \frac{l \cdot M_x + r + M_x - 1}{M_x} = \frac{(l+1)M_x + r - 1}{M_x} = \\ &= l + 1 + \frac{r-1}{M_x} = \begin{cases} l & \text{if } r = 0 \\ l+1 & \text{if } r = 1 \dots M_x - 1 \end{cases} \end{aligned}$$

So $\frac{N+(M_x-1)}{M_x}$ is the smallest multiple of M_x greater than or equal to N, so $\frac{N+(M_x-1)}{M_x}$ **blocks are needed or more than needed to run a total of N threads.**

Problem: Max. grid dim. in 1-direction is 65535, $\equiv N_i^{\max}$.

So $\frac{N+(M_x-1)}{M_x} = N_i^{\max} \Longrightarrow N = N_i^{\max} M_x - (M_x - 1) \leq N_i^{\max} M_x$. i.e. number of threads N is limited by $N_i^{\max} M_x$.

Solution.

- number of threads per block in x-direction $\equiv M_x \Longrightarrow$ `blockDim.x`
- number of blocks in grid $\equiv N_x \Longrightarrow$ `gridDim.x`
- $N_x M_x$ total number of threads in x-direction. Increment by $N_x M_x$. So next scheduled execution by GPU at the $k = N_x M_x$ thread.

Sanders and Kandrot (2010) [42] made an important note, on pp. 176-177 Ch. 9 Atomics of Section 9.4 Computing Histograms, an important *rule of thumb* on the number of blocks.

First, consider N^{threads} total threads. The extremes are either N^{threads} threads on a single block, or N^{threads} blocks, each with a single thread.

Sanders and Kandrot gave this tip:

number of blocks, i.e. `gridDim.x` $\Longleftarrow N_x \sim 2\times$ number of GPU multiprocessors, i.e. twice the number of GPU multiprocessors. In the case of my GeForce GTX 980 Ti, it has 22 Multiprocessors.

2.1. **global thread Indexing: 1-dim., 2-dim., 3-dim.** Consider the problem of *global thread indexing*. This was asked on the NVIDIA Developer's board (cf. Calculate GLOBAL thread Id). Also, there exists a "cheatsheet" (cf. CUDA Thread Indexing Cheatsheet). Let's consider a (mathematical) generalization.

Consider again (cf. 2) the following notation:

- `threadIdx.x` $\Longleftarrow i_x$, $0 \leq i_x \leq M_x - 1$, $\quad i_x \in \{0 \dots M_x - 1\} \equiv I_x$, of "cardinal length/size" of $|I_x| = M_x$
- `blockIdx.x` $\Longleftarrow j_x$, $0 \leq j_x \leq N_x - 1$, $\quad j_x \in \{0 \dots N_x - 1\} \equiv J_x$, of "cardinal length/size" of $|J_x| = N_x$
- `blockDim.x` $\Longleftarrow M_x$
- `gridDim.x` $\Longleftarrow N_x$

Now consider formulating the various cases, of a grid of dimensions from 1 to 3, and blocks of dimensions from 1 to 3 (for a total of 9 different cases) mathematically, as the CUDA Thread Indexing Cheatsheet did, similarly:

- *1-dim. grid* of *1-dim. blocks*. Consider $J_x \times I_x$. For $j_x \in J_x$, $i_x \in I_x$, then $k_x = j_x M_x + i_x$, $k_x \in \{0 \dots N_x M_x - 1\} \equiv K_x$.
  The condition that k_x be a valid global thread index is that K_x has equal cardinality or size as $J_x \times I_x$, i.e.

$$|J_x \times I_x| = |K_x|$$

(this must be true). This can be checked by checking the most extreme, maximal, case of $j_x = N_x - 1$, $i_x = M_x - 1$:

$$k_x = j_x M_x + i_x = (N_x - 1)M_x + M_x - 1 = N_x M_x - 1$$

and so k_x ranges from 0 to $N_x M_x - 1$, and so $|K_x| = N_x M_x$.
Summarizing all of this in the following manner:

$$\begin{aligned} J_x \times I_x &\longrightarrow K_x \equiv K^{N_x M_x} = \{0 \dots N_x M_x - 1\} \\ (j_x, i_x) &\longmapsto k_x = j_x M_x + i_x \end{aligned}$$

For the other cases, this generalization we've just done is implied.

- *1-dim. grid* of *2-dim. blocks*

$$\begin{aligned} J_x \times (I_x \times I_y) &\longrightarrow K^{N_x M_x M_y} \equiv \{0 \dots N_x M_x M_y - 1\} \\ (j_x, (i_x, i_y)) &\longmapsto k = j_x M_x M_y + (i_x + i_y M_x) = j_x |I_x \times I_y| + (i_x + i_y M_x) \in \{0 \dots N_x M_x M_y - 1\} \end{aligned}$$

The "most extreme, maximal" case that can be checked to check that the "cardinal size" of $K^{N_x M_x M_y}$ is equal to $J_x \times (I_x \times I_y)$ is the following, and for the other cases, will be implied (unless explicitly written or checked out):

$$k = j_x M_x M_y + (i_x + i_y M_x) = (N_x - 1)M_x M_y + ((M_x - 1) + (M_y - 1)M_x) = (N_x M_x M_y - 1)$$

The thing to notice is this emerging, general pattern, what could be called a "global view" of understanding the threads and blocks model of the GPU (cf. njuffa's answer:

total number of threads $=$ block index (Id) $\cdot$ total number of threads per blcok $+$ thread index on the block

But as we'll see, that's not the only way of "flattening" the index, or transforming into a 1-dimensional index.

- *1-dim. grid* of *3-dim. blocks*

$$\begin{aligned} J_x \times (I_x \times I_y \times I_z) &\longrightarrow K^{N_x M_x M_y M_z} \\ (j_x, (i_x, i_y, i_z)) &\longmapsto k = j_x(M_x M_y M_z) + (i_x + i_y M_x + i_Z M_x M_y) \in \{0 \dots N_x M_x M_y M_z - 1\} \end{aligned}$$

- *2-dim. grid* of *1-dim. blocks*

$$\begin{aligned} (J_x \times J_y) \times I_x &\longrightarrow L^{N_x N_y} \times I_x \longrightarrow K^{N_x N_y M_x} \\ ((j_x, j_y), i_x) &\longmapsto ((j_x + N_x j_y), i_x) \longmapsto k = (j_x + N_x j_y) \cdot M_x + i_x \in \{0 \dots N_x N_y M_x - 1\} \end{aligned}$$

- *2-dim. grid* of *2-dim. blocks*

$$\begin{aligned} (J_x \times J_y) \times (I_x, I_y) &\longrightarrow L^{N_x N_y} \times (I_x, I_y) \longrightarrow K^{N_x N_y M_x} \\ ((j_x, j_y), (i_x, i_y)) &\longmapsto ((j_x + N_x j_y), (i_x, i_y)) \longmapsto k = (j_x + N_x j_y) \cdot M_x M_y + i_x + M_x i_y \end{aligned}$$

But this *isn't the only way of obtaining* a "flattened index." Exploit the commutativity and associativity of the Cartesian product:

$$\begin{aligned} J_x \times J_y \times I_x \times I_y = (J_x \times I_x) \times (J_y \times I_y) &\longrightarrow K^{N_x M_x} \times K^{N_y M_y} \longrightarrow K^{N_x N_y M_x M_y} \\ ((j_x, j_y, i_x, i_y) = ((j_x, i_x), (j_y, i_y)) &\longmapsto (i_x + M_x j_x, i_y + M_y j_j) \equiv (k_x, k_y) \longmapsto \begin{gathered} k = k_x + k_y N_x M_x = \\ = (i_x + M_x j_x) + (i_y + M_y j_y) M_x N_x \end{gathered} \end{aligned}$$

Indeed, checking the "maximal, extreme" case,

$$k = k_x + k_y N_x M_x = M_x N_x - 1 + (M_y N_y - 1)(N_x M_x) = M_y M_y N_x M_x - 1$$

and so k ranges from 0 to $M_y M_y N_x M_x - 1$.

- *3-dim. grid* of *3-dim. blocks*

$$\begin{aligned} \begin{gathered} (J_x \times J_y \times J_z) \times (I_x \times I_y \times I_z) = \\ = (J_x \times I_x) \times (J_y \times I_y) \times (J_z \times I_z) \end{gathered} &\longrightarrow K^{N_x M_x} \times K^{N_y M_y} \times K^{N_z M_z} \longrightarrow K^{N_x N_y N_z M_x M_y M_z} \\ \begin{gathered} ((j_x, j_y, j_z), (i_x, i_y, i_z)) = \\ = ((j_x, i_x), (j_y, i_y), (j_z, i_z)) \end{gathered} &\longmapsto \begin{gathered} (i_x + M_x j_x, i_y + M_y j_j, i_z + M_z j_z) \equiv \\ \equiv (k_x, k_y, k_z) \end{gathered} \longmapsto k = k_x + k_y N_x M_x + k_z N_x M_x N_y M_y \end{aligned}$$

Indeed, checking the "extreme, maximal" case for k:

$$\begin{gathered} k = k_x + k_y N_x M_x + k_z N_x M_x N_y N_y = \\ = (N_x M_x - 1) + (N_y M_y - 1) N_x M_x + (N_z M_z - 1) N_x M_x N_y M_y = N_x N_y N_z M_x M_y M_z - 1 \end{gathered}$$

2.2. **With Stride.** Consider this code for the L1 loss function and L2 loss function. It introduces the idea of a "stride." Let's figure out what the stride means mathematically.

Given Eq. 1, recall that

$$k = j_x + i_x M_x, \quad \begin{gathered} 0 \leq j_x < M_x \\ 0 \leq i_x < N_x \end{gathered}, \quad 0 \leq k < N_x M_x$$

Let $S \equiv$ stride.

Following the code we mentioned above for L1 and L2 loss functions, it seems that the following 2 indices were introduced, with inter roughly meaning "outside" or "outward" and intra meaning "within" or "internal", presumably:

$$\begin{gathered} i_{\text{intra}} = k \mod S \in \{0, 1, \dots S - 1\} \\ i_{\text{inter}} = \lfloor k/S \rfloor \in \{0, 1, \dots (N_x M_x - 1)/S\} \end{gathered}$$

Now let $D =$ number of dimensions.

Continuing with the code, it seems that $i_{\text{intra}} < D$ is what's expected. Otherwise, if $i_{\text{intra}} \geq D$, then we effectively don't compute.

$$l \equiv i_{\text{inter}} D + i_{\text{intra}}, \quad 0 \leq l \leq \left(\frac{N_x M_x - 1}{S} \right) D + S - 1$$

Let's try some examples. If $S = 1$, $i_{\text{intra}} = 0$, always (the modulus of a number by 1 is always 0 because any number can be divided by 1 "evenly" (no remainder)), and $i_{\text{inter}} = k$. So $l = kD$. If $D = 1$ we get a 1 to 1 mapping which is what I'd typically expect if given arrays. If $D = 2$, then l would be all the even indices it seems implied that the "target" array that's using l as an index would have at least twice the size of the input array!

If $S = 2$, $i_{\text{intra}} \in \{0, 1\}$, $i_{\text{inter}} \in \{0, 1, \dots \lfloor \frac{N_x M_x - 1}{2} \rfloor \}$ and so $l \in \{0, 1, \dots (\frac{N_x M_x - 1}{2}) D + 1\}$.

3. Row-Major ordering vs. Column Major ordering, as flatten

So-called row-major ordering and column major ordering should be formalized, to deal with continguous memory access in reading or writing to a matrix, or lack thereof.

Given

$$\begin{gathered} A \in \text{Mat}_{\mathbb{R}}(m, n) \\ A : \{1, 2, \dots m\} \times \{1, 2, \dots n\} \to \mathbb{R} \\ A : (i, j) \mapsto A(i, j) \in \mathbb{R} \\ \text{or} \begin{aligned} & A : \{0, 1, \dots m-1\} \times \{0, 1, \dots n-1\} \to \mathbb{R} \\ & A : (i, j) \mapsto A(i, j) \in \mathbb{R} \end{aligned} \end{gathered}$$

Consider isomorphism "flatten":

$$\begin{gathered} \text{Mat}_{\mathbb{R}}(m, n) \xrightarrow{\text{flatten}} \mathbb{R}^{mn} \\ \{1, 2, \dots m\} \times \{1, 2, \dots n\} \to \{1, 2, \dots mn\} \\ \{0, 1, \dots m-1\} \times \{0, 1, \dots n-1\} \to \{0, 1, \dots mn-1\} \end{gathered} \tag{2}$$

There are 2 kinds of flatten:

Row-major ordering is the one we're (psychologically) used to, if we read continguously from left to right, horizontally, along a row.

Definition 1 (row-major ordering)**.**

$$\begin{gathered} \{0, 1, \dots m-1\} \times \{0, 1, \dots n-1\} \to \{0, 1, \dots mn-1\} \\ (i, j) \mapsto in + j \\ \{1, 2, \dots m\} \times \{1, 2, \dots n\} \to \{1, 2, \dots mn\} \\ (i, j) \mapsto (i-1)n + j \end{gathered} \qquad \begin{gathered} \{0, 1, \dots mn-1\} \to \{0, 1, \dots m-1\} \times \{0, 1, \dots n-1\} \\ k \mapsto (k/n, k \mod n) \\ \{1, 2, \dots mn\} \to \{1, 2, \dots m\} \times \{1, 2, \dots n\} \\ k \mapsto (\lceil k/n \rceil, k \mod n) \end{gathered} \tag{3}$$

Definition 2 (column-major ordering)**.**

$$\begin{gathered} \{0, 1, \dots m-1\} \times \{0, 1, \dots n-1\} \to \{0, 1, \dots mn-1\} \\ (i, j) \mapsto i + jm \\ \{1, 2, \dots m\} \times \{1, 2, \dots n\} \to \{1, 2, \dots mn\} \\ (i, j) \mapsto i + (j-1)m \end{gathered} \qquad \begin{gathered} \{0, 1, \dots mn-1\} \to \{0, 1, \dots m-1\} \times \{0, 1, \dots n-1\} \\ k \mapsto (k \mod m, k/m) \\ \{1, 2, \dots mn\} \to \{1, 2, \dots m\} \times \{1, 2, \dots n\} \\ k \mapsto (k \mod m, \lceil k/m \rceil) \end{gathered} \tag{4}$$

Part 3. Learning: Gradient

4. GaLore

Zhao et. al. (2024) [43].

Part 4. Embeddings

5. FAISS

References

[1] Trevor Hastie, Robert Tibshirani, Jerome Friedman. **The Elements of Statistical Learning: Data Mining, Inference, and Prediction**, Second Edition (Springer Series in Statistics) 2nd ed. 2009. Corr. 7th printing 2013 Edition. ISBN-13: 978-0387848570. https://web.stanford.edu/~hastie/local.ftp/Springer/OLD/ESLII_print4.pdf

[2] Jared Culbertson, Kirk Sturtz. *Bayesian machine learning via category theory.* arXiv:1312.1445 [math.CT]

[3] John Owens. David Luebki. *Intro to Parallel Programming. CS344.* **Udacity** http://arxiv.org/abs/1312.1445 Also, https://github.com/udacity/cs344

[4] CS229 Stanford University. http://cs229.stanford.edu/materials.html

[5] Richard Fitzpatrick. "Computational Physics." http://farside.ph.utexas.edu/teaching/329/329.pdf

[6] LISA lab, University of Montreal. Deep Learning Tutorial. http://deeplearning.net/tutorial/deeplearning.pdf September 2015.

[7] Kurt Hornik. "Approximation Capabilities of Muitilayer Feedforward Networks." **Neural Networks**, Vol. 4, pp. 251-257. 1991

[8] Kurt Hornik. Maxwell Stinchcombe and Halbert White. "Multilayer Feedforward Networks are Universal Approximators." **Neural Networks**, Vol. 2, pp. 359-366, 1989.

[9] Rafal Jozefowicz, Wojciech Zaremba, Ilya Sutskever. "An Empirical Exploration of Recurrent Network Architectures." *Proceedings of the 32 nd International Conference on Machine Learning*, Lille, France, 2015. JMLR: W&CP volume 37.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. **Deep Learning** (Adaptive Computation and Machine Learning series). The MIT Press (November 18, 2016).

[11] Jacob C. Bridgeman, Christopher T. Chubb. *Hand-waving and Interpretive Dance: An Introductory Course on Tensor Networks.* https://arxiv.org/abs/1603.03039 [quant-ph]

[12] David Stutz. *Understanding Convolutional Neural Networks.* Seminar Report. August 30, 2014. Advisor: Lucas Beyer. Fakultät für Mathematik, Informatik und Naturwissenschaften; Lehr- und Forschungsgebiet Informatik VIII; Computer Vision; Prof. Dr. Bastian Leibe. http://davidstutz.de/wordpress/wp-content/uploads/2014/07/seminar.pdf.

[13] Andrew Lavin, Scott Gray. *Fast Algorithms for Convolutional Neural Networks* arXiv:1509.09308 [cs.NE]

[14] Thomas Nowak. "Implementation and Evaluation of a Support Vector Machine on an 8-bit Microcontroller." Univ.Ass. Dipl.-Ing. Dr.techn. Wilfried Elmenreich Institut für Technische Informatik Fakultät für Informatik Technische Universität Wien. Juli 2008. https://www.lri.fr/~nowak/misc/bakk.pdf

[15] J. Shawe-Taylor and N. Cristianini, Support Vector Machines and other kernel-based learning methods, Cambridge University Press (2000).

[16] Edwin K. P. Chong and Stanislaw H. Zak. **An Introduction to Optimization**. 4th Edition. Wiley. (January 14, 2013). ISBN-13: 978-1118279014

[17] Lecture by Harikrishna Narasimhan. *Optimization Tutorial 3: Projected Gradient Descent, Duality.* **E0 270 Machine Learning**. Jan 23, 2015. http://drona.csa.iisc.ernet.in/~e0270/Jan-2015/Tutorials/lecture-notes-3.pdf

[18] Christopher M. Bishop. **Pattern Recognition and Machine Learning** (Information Science and Statistics). Springer (October 1, 2007). ISBN-13: 978-0387310732

[19] Bertrand Clarke, Ernest Fokoue, Hao Helen Zhang. **Principles and Theory for Data Mining and Machine Learning** (Springer Series in Statistics) Springer; 2009 edition (July 30, 2009). ISBN-13: 978-0387981345

[20] Hubert Nguyen. **GPU Gems 3**. Addison-Wesley Professional (August 12, 2007). ISBN-13: 978-0321515261. Also made available in its entirety online at https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_pref01.html

[21] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, **JMLR 12**, pp. 2825-2830, 2011.

[22] Bogusł aw Cyganek. J. Paul Siebert. **An Introduction to 3D Computer Vision Techniques and Algorithms**. Wiley. 2009. ISBN 978-0-470-01704-3

[23] Richard Hartley. Andrew Zisserman. **Multiple View Geometry in Computer Vision** Second Edition. *Cambridge University Press.* 2003. ISBN-13 978-0-511-18618-9 eBook (EBL)

[24] John Lee, **Introduction to Smooth Manifolds** (Graduate Texts in Mathematics, Vol. 218), 2nd edition, Springer, 2012, ISBN-13: 978-1441999818

[25] Jeffrey M. Lee. **Manifolds and Differential Geometry**, *Graduate Studies in Mathematics* Volume: 107, American Mathematical Society, 2009. ISBN-13: 978-0-8218-4815-9

[26] C.-C. Chang and C.-J. Lin. *LIBSVM : a library for support vector machines.* **ACM Transactions on Intelligent Systems and Technology**, 2:27:1–27:27, 2011.

[27] J. Platt. *Fast training of support vector machines using sequential minimal optimization.* In A. Smola B. Schölkopf, C. Burges, editor, **Advances in Kernel Methods: Support Vector Learning**. MIT Press, Cambridge, MA, 1998.

[28] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. *A Practical Guide to Support Vector Classification.* http://www.ee.columbia.edu/~sfchang/course/spr/papers/svm-practical-guide.pdf

[29] Rada Mihalcea and Paul Tarau. "TextRank: Bringing Order into Texts." https://web.eecs.umich.edu/~mihalcea/papers/mihalcea.emnlp04.pdf

[30] David F. Gleich. *PageRank Beyond the Web.* **SIAM Review**. Vol. 57, No. 3, pp. 321-363 2015

[31] Page, Lawrence and Brin, Sergey and Motwani, Rajeev and Winograd, Terry. "The PageRank Citation Ranking: Bringing Order to the Web." Technical Report. Stanford InfoLab. 1998. http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf

[32] Prof. Dr. Michael Bader, with Alexander Pöppl, Valeriy Khakhutskyy (Tutorials). High Performance Computing (HPC) - Algorithms and Applications - Winter 16/17. Informatics V - Scientific Computing. Technical University of Munich (TUM) https://www5.in.tum.de/wiki/index.php/HPC_-_Algorithms_and_Applications_-_Winter_16

[33] Andrew Ng. Machine Learning. coursera

[34] Kilian Stoffel and Abdelkader Belkoniene. *Parallel k/h-Means Clustering for Large Data Sets.* Euro-Par'99, LNCS 1685, pp. 1451-1454, 1999. Springer-Verlag Berlin Heidelberg 1999 https://grid.cs.gsu.edu/~wkim/index_files/papers/parkh.pdf

[35] Theano Development Team. "Theano: A Python framework for fast computation of mathematical expressions".

[36] Joseph J. Rotman, **Advanced Modern Algebra** (Graduate Studies in Mathematics) 2nd Edition, American Mathematical Society; 2 edition (August 10, 2010), ISBN-13: 978-0821847411

[37] Jeffrey M. Lee. **Manifolds and Differential Geometry**, *Graduate Studies in Mathematics* Volume: 107, American Mathematical Society, 2009. ISBN-13: 978-0-8218-4815-9

[38] Lawrence Conlon. **Differentiable Manifolds** (Modern Birkhäuser Classics). 2nd Edition. Birkhäuser; 2nd edition (October 10, 2008). ISBN-13: 978-0817647667

[39] The Sage Development Team. Sage Reference Manual: Category Framework. Release 7.6. Mar. 25, 2017.

[40] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo,Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis,Jeffrey Dean,Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow,Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia,Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster,Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens,Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker,Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas,Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke,Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Software available from tensorflow.org.

[41] Rafal Jozefowicz, Wojciech Zaremba, Ilya Sutskever. "An Empirical Exploration of Recurrent Network Architectures." *Proceedings of the 32nd International Conference on Machine Learning*, Lille, France, 2015. JMLR: W&CP volume 37. http://www.jmlr.org/proceedings/papers/v37/jozefowicz15.pdf

[42] Jason Sanders, Edward Kandrot. **CUDA by Example: An Introduction to General-Purpose GPU Programming** 1st Edition. Addison-Wesley Professional; 1 edition (July 29, 2010). ISBN-13: 978-0131387683

[43] Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, Yuandong Tian. "GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection." arXiv: 2403.03507v1. 6 Mar 2024. https://github.com/jiaweizzhao/GaLore

[44] freeCodeCamp.org Vector Search RAG Tutorial – Combine Your Data with LLMs with Advanced Search . https://youtu.be/JEBDfGqrAUA?si=han3HbnkLP5lH3b4