Convolutional Neural Networks: Theory, Implementation and Application (Draft 0.01)

by

Arthur Redfern and Tarek Lahlou

Contents

Part I

Introduction

Chapter 1

Motivation

This book provides an introduction to CNNs covering theory, implementation and application from the viewpoint that all 3 parts are important.

The theory portion of the book is motivated by the realization that many information extraction problems can be reduced to a classification problem, NNs are universal approximators and CNNs are an efficient NN structure for images and other multidimensional data. In the implementation portion, software and hardware requirements for CNNs are described with an emphasis on graph specification and optimization, memory sizing, data movement with formatting and fundamental primitives for computation. Theory and implementation are demonstrated and expanded on in the application portion of the book.

The 1st section in this chapter briefly discusses how CNNs, ML and AI impact science and engineering. The 2nd section briefly discusses the skills that a person needs to make progress in CNNs. Many more words could be written in each, but a person who picked up this book in the 1st place probably doesn't need to be convinced further so they'll remain brief.

1.1 | Technology

1.1.1 | Engineering

There's a trend towards the complete instrumentstion of all:

- Spaces
- Objects
- People

Extracting information from data and deciding what to do with it leads to a sense → compute → act style processing flow. CNNs achieve start of the art performance on all sorts of information extraction and decision making problems making them a fundamental part of engineered systems.

1.1.2 | Science

Somewhere in the 2000s all sciences got a computational cousin: biology got computational biology, chemistry got computational chemistry, . . . with system modeling and simulation being the key use of the computation. Why? Real systems frequently have complex interactions that humans can't write down in a succinct set of equations with a closed form solution. Simulations built from underlying principles provide a conveinent mechanism for studying problems and understanding potential solutions. Computation is used to replace tedious mechanical work for a person.

At the time, it was a big deal to introduce computation into science, now no one thinks twice about it. Scientists apply theory to model systems where theory makes sense and apply computation to model systems where computation makes sense. Computational science is not an either or proposition, it's fine to blend the use of both.

Today all sciences are getting a new relative that goes by the name of ML: ML for biology, ML for chemistry, ... with analysis of results being the key use of ML. Why? Again, it goes back to the complexity of real systems. Experiments frequently produce large amounts of data with complex relationships that are difficult for a human to analyze. In these cases, it's natural to augment or replace a human with ML and train algorithms directly from data. Note that this is a bit different than the computational case that can be thought of as assisting or replacing the human's body. Here, ML can be thought of as assisting or replacing the human's brain.

Are there concerns with this? Sure, phrased as above it sounds sort of creepy. And maybe the resulting solution is not easily understandable by humans, maybe the data used by ML to train the algorithm fails to capture something important in the system, ... These are reasonable concerns and there are more. However, sufficient successes will force the hand of most scientists to at least understand ML and view it as an additional tool that can be applied. As with computational science, ML for science is not an either or proposition, it's fine to blend the use of both.

1.2 | People

Without theory you're lost, without implementation you're impractical and without application you're pointless.

Can a person make progress in CNNs only knowing 1 of these 3 parts? Probably. Companies can likely hire many specialists and only a few generalists and still do amazing things, in some sense overwhelming siloing problems with numbers of people and a small amount of glue. But not every company has that luxury and frequently there's a much smaller pool of people grouped together with CNN knowledge.

In both the absense and presence of overwhelming numbers, it's useful to have people who while potentially specialists in 1 area still know a little bit about all 3.

The hypothesis is that better ideas will be created via an understanding of theory, they'll be practical with an understanding of implementation and they'll focus on problems that matter with an understanding of application.

This book seeks to provide a self contained treatment that ties all 3 parts together and allows for a test of the proposed hypothesis.

Chapter 2

Plan

2.1 | Organization

Perhaps not surprisingly, this book has 3 main parts:

- Theory
- Implementation
- Application

1 part to provide a refresher on the math that's used throughout the book

- Background

and 3 parts to satisfy the need for a book to have a beginning, ending and junk drawer:

- Introduction
- Conclusion
- Appendix

This is the final chapter in the Introduction part so it should already be clear as to what the Introduction contains minus 1 section.

Background divides math into linear algebra, calculus and probability chapters. The math chapters are not comprehensive, instead, they focus on a limited set of topics to make the book mostly self contained. These topics are generally useful to many fields and will be familiar to a reader with a math, science or engineering background. However, it's likely that some parts of the treatment will be new and useful.

Theory is split into machine learning and CNN chapters. The machine learning chapter introduces a framework for thinking about machine learning that sets up the CNN chapter such that CNNs are understood within the larger context of machine learning. This includes a look into the design, training and performance of CNNs. A simple example is provided for reference, additional examples will be included in the applications.

Implementation is broken into hardware and software chapters. Both chapters focus on what is needed to make CNNs run at a high level of performance. For the hardware chapter, this means SoC designs with a lot of memory, efficient data movement and primitive compute operations. An argument is made for simple hardware that does what the software tells it to do. For the software chapter, this means a low level computational graph that describes the network and individual nodes including data movement and compute matched to the hardware. Intelligence and optimization are moved to software, the appropriate place from an efficiency perspective. Understanding what hardware and software can and can't do well provides a bridge from theory to application.

Application allocates 1 chapter to each reasonably well covered application and 1 chapter as a temporary parking place for all other applications. Currently, this means there are vision, speech, language, game and other chapters in the application portion. These will all likely be continual works in progress, with material added to existing chapters and new chapters added with material migrating from other and

expanding. If an application is in other or not listed at all does it mean that the application is not important? Yes is does. Just kidding, likely it's currently not it's own chapter becasuse of a lack of author time or knowledge.

Conclusion includes summary and next chapters. Summary includes a few highlights from each of the different parts of the book, perhaps useful for making sure that some big topic wasn't missed in the rare event that the book wasn't read word by word cover to cover. The next chapter includes both likely to be correct and likely to be incorrect sections, the former being interpolation and small perturbations around current CNNs, the latter being extrapolating into the future.

Appendix is for items that typically go in an appendix. Unlike the part of human anatomy, this appendix is meant to be relatively useful with a mix of timeless and timely information. Chapters include notation, practice, history, references and about. The practice chapter is basically a suboptimal but workable current implementation that allows application to test theory.

2.2 | Style

From a language perspective, this is not the most elegant or formal book ever written. Introductions to chapters are a little more conversational, bodies of chapters are a little more bare bones.

The style was chosen to make the book compact but sufficient. It's difficult to find time to read unnecessary words or to search outside for information after every other page. This book is a distillation of disparate info that attempts to keep what's important and create a unified story from start to finish.

CNNs are a field with a lot of attention and with it new ideas all the time. While this book covers a lot, there is a lot more that it doesn't cover. Hopefully, however, after reading the book it will be easier to pick up related ideas outside of the book's

direct scope.

Before proceeding to the 1st useful part of the book, a disclaimer: it's possible that a different book resonates better with a specific reader for whatever reason. That's cool, no big deal, this book has no ego. Use it if it's useful, don't if it's not.

Either way, good luck!

Part II

Background

Chapter 3

Linear Algebra

Convolutional neural networks are fundamentally compositions of nonlinear functions mapping inputs to outputs. The core computations are dominated by linear operations that can be reduced to various forms of matrix multiplication. Depending on the particular formulation, matrix multiplication has cubic computational complexity and square data movement complexity thereby enabling high performance hardware implementations to be possible.

This chapter begins by reviewing vector spaces and basic matrix operations in part to establish notational conventions. Matrix decompositions, solutions of linear systems of equations and matrix transformations complete the chapter and provide the foundation of linear algebra tools used in the analysis, design and implementation of CNNs in subsequent chapters.

Before proceeding, 3 disclaimers that are applicable in spirit to all the background math chapters in this book:

- Reducing linear algebra to a low double digit number of pages means a more accurate title for the chapter would be "A Small Number Of Linear Algebra Facts Refresher". Linear algebra was and still is the focus of many lifetimes of work by many amazing people, this is just scratching the surface.

- It's the intention to make the book mathematically precise without being mathematically burdensome. As such, cases that are not obviously applicable to CNNs and complicate the presentation of material are explicitly removed. It's not that the material is unimportant, it's just that this is not the place for it's presentation.
- This is not a great 1st introduction to linear algebra. Liberties are taken with respect to the organization and emphasis of the material to make it work better for a person who has already seen some of it before.

All 3 disclaimers lead to the same comment: for a jumping off point to a more complete, precise and more beautiful treatment of a topic, please consult 1 of the references.

3.1 | Vector Spaces

3.1.1 | Elements

Set

A set is a collection of distinct objects. Sets are generally described by either enumerating elements or providing inclusion rules. For example, the set S of non-negative integers can be written as $S = \{0, 1, 2, \dots\}$ or $S = \{n \colon n \text{ is an integer}, n \geq 0\}$.

Field

A field is a set with well-defined addition ($+$) and multiplication ($\cdot$) operations. More formally, a field $\mathbb{F}$ is a set for which any arbitrary elements $a, b, c \in \mathbb{F}$ satisfy the field axioms:

- associativity: $a + (b + c) = (a + b) + c$ and $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
- commutitivity: $a + b = b + a$ and $a \cdot b = b \cdot a$

- additive identity: an element $0 \in \mathbb{F}$ exists such that $a + 0 = a$
- multiplicative identity: an element $1 \in \mathbb{F}$ exists such that $1 \cdot a = a$
- additive inverse: there exists an element $-a \in \mathbb{F}$ such that $a + (-a) = 0$
- multiplicative inverse: there exists an element $a^{-1} \in \mathbb{F}$ such that $a \cdot a^{-1} = 1$
- distributivity: $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$

Elements of fields are generally referred to as scalars. Common examples of fields include the real ($\mathbb{R}$) and complex ($\mathbb{C}$) numbers.

Function

A function $f\colon \mathcal{X} \to \mathcal{Y}$ is a rule mapping each element of the domain set $\mathcal{X}$ to an element of the co domain set $\mathcal{Y}$. The range of f refers to the subset of $\mathcal{Y}$ produced by applying f to the entire domain. For the expression $y = f(x)$, y is referred to as the image of x. The inverse image of y is then the set of elements $\{x\}$ in the domain such that the image of those elements is y.

Common designations for functions with special relationships between the domain and co domain are:

- Injective or one-to-one: every $y \in \mathcal{Y}$ is produced by at most one $x \in \mathcal{X}$
- Surjective or onto: every $y \in \mathcal{Y}$ is produced by at least one $x \in \mathcal{X}$
- Bijective or one-to-one and onto: both injective and surjective

Bijective functions are also referred to as invertible.

Vector

A vector is an K-tuple of scalars arranged into a 1-dimensional array for some positive integer K. In detail, a vector $\mathbf{x}$ is written according to

$$\mathbf{x} \;=\; \begin{bmatrix} x(0) \\ \vdots \\ x(K-1) \end{bmatrix} \;\in\; \begin{bmatrix} \mathbb{F} \\ \vdots \\ \mathbb{F} \end{bmatrix} \;=\; \mathbb{F}^K \tag{3.1}$$

where the k-th entry of $\mathbf{x}$ is the scalar $x(k) \in \mathbb{F}$. Note the zero-based indexing convention of vector entries.

Matrix

A matrix is an $M \times K$ rectangular array comprising M rows and K columns of scalars. In detail, a matrix $\mathbf{A}$ is written according to

$$\mathbf{A} \;=\; \begin{bmatrix} a(0,0) & \cdots & a(0,K-1) \\ \vdots & & \vdots \\ a(M-1,0) & \cdots & a(M-1,K-1) \end{bmatrix}. \tag{3.2}$$

Each column of $\mathbf{A}$ is itself an M-element vector $\mathbf{a}$. Using this observation, (3.2) can be expressed using a collection of K such vectors as

$$\mathbf{A} \;=\; \begin{bmatrix} \mathbf{a}_0 & \cdots & \mathbf{a}_{K-1} \end{bmatrix} \tag{3.3}$$

where $\mathbf{a}_k = \mathbf{A}(:,k)$ and $i:j$ indicates all indices from i to j (inclusive) and : indicates all indices in the specified position. This perspective will later be useful when considering the column space and right null space of $\mathbf{A}$.

Each row of $\mathbf{A}$ is an K-element transposed vector $\mathbf{a}^T$. Using this definition of transposition, (3.2) can be expressed using a collection of M such transposed vectors

as

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_0^T \\ \vdots \\ \mathbf{a}_{M-1}^T \end{bmatrix} \tag{3.4}$$

where $\mathbf{a}_m^T = \mathbf{A}(m,:)$. This perspective will later be useful when considering the row space and left null space of $\mathbf{A}$. Note the vectors in (3.3) and (3.4) are not the same.

Tensor

A tensor is an $K_0 \times \cdots \times K_{D-1}$ array of scalars. Tensors are D-dimensional arrays in the sense that vectors and matrices are 1 and 2 dimensional arrays, respectively. For example, let $\mathbf{T}$ denote a 3-dimensional tensor with MNK entries organized into an $M \times K \times N$ array. Various slices of $\mathbf{T}$ provides familiar vectors and matrices:

$\mathbf{f}(:,k,n)$	M-element vectors	$\mathbf{f}(:,:,n)$	$M \times K$-element matrices
$\mathbf{f}(m,:,n)$	K-element vectors	$\mathbf{f}(:,k,:)$	$M \times N$-element matrices .
$\mathbf{f}(m,k,:)$	N-element vectors	$\mathbf{f}(m,:,:)$	$K \times N$-element matrices

3.1.2 | Vector Spaces

Vector Space

A vector space is a set of vectors and all linear combinations of those vectors. More formally, a vector space $\mathcal{V}$ over a field $\mathbb{F}$ is a set for which any arbitrary vectors $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathcal{V}$ and scalars $a, b \in \mathbb{F}$ satisfy the vector space axioms:

- associativity: $\mathbf{x} + (\mathbf{y} + \mathbf{z}) = (\mathbf{x} + \mathbf{y}) + \mathbf{z}$
- commutativity: $\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$
- additive identity: a vector $\mathbf{0} \in \mathcal{V}$ exists such that $\mathbf{x} + \mathbf{0} = \mathbf{x}$
- additive inverse: a vector $-\mathbf{x} \in \mathcal{V}$ exists such that $\mathbf{x} + (-\mathbf{x}) = \mathbf{0}$

- multiplicative compatibility: $a(b\mathbf{x}) = b(a\mathbf{x})$
- multiplicative identity: $1\mathbf{x} = \mathbf{x}$ for $1 \in \mathbb{F}$
- distributivity: $(a+b)(\mathbf{x}+\mathbf{y}) = a\mathbf{x} + a\mathbf{y} + b\mathbf{x} + b\mathbf{y}$

Common examples of vector spaces include K-dimensional vectors with real ($\mathbb{R}^K$) and complex ($\mathbb{C}^K$) scalars. Moreover, the collection of all $K_0 \times \cdots \times K_{D-1}$ tensors with real ($\mathbb{R}^{K_0 \times \cdots \times K_{D-1}}$) and complex ($\mathbb{C}^{K_0 \times \cdots \times K_{D-1}}$) scalars form bona fide vector spaces as well.

Span

The span of a set of vectors is the set of all finite linear combinations of the vectors. More formally, the span of the set $\{\mathbf{a}_0, \ldots, \mathbf{a}_{K-1}\}$ is denoted span$\{\mathbf{a}_0, \cdots, \mathbf{a}_{K-1}\}$ and contains all vectors $\mathbf{a}$ of the form

$$\mathbf{a} \quad = \quad \alpha_0 \mathbf{a}_0 + \cdots + \alpha_{K-1} \mathbf{a}_{K-1} \tag{3.5}$$

where the α_k are arbitrary scalars. The span of any set of vectors is a vector space.

Rank

The rank of a matrix $\mathbf{A}$ is the dimension of the vector space generated by the span of the column vectors forming the matrix. As will be discussed later in this chapter, the dimension of the vector space formed by the column vectors is also the dimension of the space spanned by the row vectors forming $\mathbf{A}$.

Linear Independence

A set of vectors are linearly dependent if at least one of them is a linear combination of the others. On the other hand, a set of vectors $\{\mathbf{a}_k\}$ are linearly independent if no vector in $\{\mathbf{a}_k\}$ can be written as a linear combination of the others.

Basis

A basis for a vector space $\mathcal{V}$ is any linearly independent set of vectors spanning $\mathcal{V}$. Bases are often interpreted as coordinate systems through which to navigate vector spaces. Indeed, the dimension of a vector space $\mathcal{V}$ is the number of vectors required to form a basis. This book focuses exclusively on finite dimensional vector spaces.

Normed Vector Space

Normed vector spaces are vector spaces endowed with a notion of distance. More formally, a norm $\|\cdot\|$ is a map taking a vector to a non-negative real number for which any arbitrary vectors $\mathbf{x}, \mathbf{y} \in \mathcal{V}$ and scalars $a \in \mathbb{F}$ satisfy the norm axioms:

- Nonnegativity: $\|\mathbf{x}\| \geq 0$, and $\|\mathbf{x}\| = 0$ if and only if $\mathbf{x} = \mathbf{0}$
- Absolute scalability: $\|a\mathbf{x}\| = |a|\|\mathbf{x}\|$
- Triangle inequality: $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$

Different norms are appropriate under different circumstances. A few common norms referred to throughout the book are provided below.

The ℓ_p norm of a length K vector $\mathbf{a}$ is defined for $1 \leq p \leq \infty$ as

$$\|\mathbf{a}\|_p \quad = \quad \left(\sum_{k=0}^{K-1} |a(n)|^p \right)^{\frac{1}{p}}. \tag{3.6}$$

The ℓ_1 norm is the sum of absolute values of the entries in $\mathbf{a}$ and is also referred to as the Manhattan norm. The ℓ_2 norm is the square root of the sum of squares of the entries in $\mathbf{a}$ and is also referred to as the Euclidean norm. The ℓ_∞ norm reduces to the largest absolute entry in $\mathbf{a}$, i.e. $\|\mathbf{a}\|)_\infty = \max_k |a(k)|$. The expression in (3.6) for $0 < p < 1$ violates the triangle inequality and is therefore not a proper norm. However, it is still of general interest in mathematics and is referred to as a seminorm.

The matrix norm induced by the ℓ_p vector norm for an $M \times K$ matrix $\mathbf{A}$ is

$$\|\mathbf{A}\|_p = \sup_{\mathbf{x} \neq \mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|_p}{\|\mathbf{x}\|_p}. \tag{3.7}$$

The ℓ_1 induced matrix norm is the maximum absolute column sum of $\mathbf{A}$. The ℓ_2 induced matrix norm is the largest singular value of $\mathbf{A}$. The ℓ_∞ induced matrix norm is the maximum absolute row sum of $\mathbf{A}$.

The matrix norm expressed as a vector norm applied first across matrix columns then to the resulting vector is defined as

$$\|\mathbf{A}\|_{p,q} = \left(\sum_{k=0}^{K-1} \left(\sum_{m=0}^{M-1} |a(m,k)|^p \right)^{\frac{q}{p}} \right)^{\frac{1}{q}} \tag{3.8}$$

for $1 \leq p, q \leq \infty$. The matrix norm for $p = q = 1$ is the sum of the absolute value of all matrix entries. The matrix norm for $p = q = 2$ is the square root of the sum of the squares of all matrix entries and is referred to as the Frobenius norm. The matrix norm for $p = q = \infty$ is the maximum of the absolute value of all matrix entries.

Inner Product Space

Inner product spaces are vector spaces endowed with geometric notions of distance and angle. More formally, an inner product $\langle \cdot, \cdot \rangle$ is a map taking two vectors to an element of the underlying field for which any arbitrary vectors $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathcal{V}$ and scalar $a \in \mathbb{F}$ satisfy the inner product axioms:

- Positive definiteness: $\langle \mathbf{x}, \mathbf{x} \rangle \geq 0$, and $\langle \mathbf{x}, \mathbf{x} \rangle = 0$ if and only if $\mathbf{x} = \mathbf{0}$
- Conjugate symmetry: $\langle \mathbf{x}, \mathbf{y} \rangle = \overline{\langle \mathbf{y}, \mathbf{x} \rangle}$ where $\overline{a}$ is the complex conjugate of a
- Linearity: $\langle a\mathbf{x}, \mathbf{y} \rangle = a\langle \mathbf{x}, \mathbf{y} \rangle$ and $\langle \mathbf{x} + \mathbf{y}, \mathbf{z} \rangle = \langle \mathbf{x}, \mathbf{z} \rangle + \langle \mathbf{y}, \mathbf{z} \rangle$

The vector inner product utilized in this book is the dot product unless specifically noted otherwise. In particular, for length K vectors $\mathbf{a}$ and $\mathbf{b}$ the dot product is

$$\begin{aligned} \langle \mathbf{a}, \mathbf{b} \rangle &= \mathbf{a} \cdot \mathbf{b} && (3.9) \\ &= \mathbf{a}^H \mathbf{b} && (3.10) \\ &= \sum_{k=0}^{K-1} \overline{a}(k) b(k) && (3.11) \end{aligned}$$

where $\mathbf{a}^H = \overline{\mathbf{a}}^T$. The dot product is also commonly referred to as the standard inner product. Moreover, it provides a means to determine the angle θ between vectors $\mathbf{a}$ and $\mathbf{b}$ as

$$\cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|_2 \|\mathbf{b}\|_2}. \quad (3.12)$$

The close relationship between the dot product and the ℓ_2 norm in (3.12) is not coincidental. Indeed, this relationship generalizes to the fact that inner products induce norms on vector spaces, i.e. specifying an inner product on a vector space implies the norm as well. Not all norms are consistent with an inner product, e.g. there is no inner product that induces the ℓ_∞ norm.

For $M \times K$ matrices $\mathbf{A}$ and $\mathbf{B}$, the matrix inner product is the Frobenious inner product given by

$$\langle \mathbf{A}, \mathbf{B} \rangle_F = \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} \overline{a}(m,k) b(m,k). \quad (3.13)$$

Note that if we flattened or vectorized the matrices by stacking their rows or columns end to end, e.g. using the vectorize operator defined shortly), the Frobenius inner product is equivalent to the vector dot product.

3.2 | Matrix Operations

3.2.1 | Matrix Transpose

The transpose or adjoint of a real-valued matrix produces a new matrix with rows and columns swapped. Transposition, more generally referred to as the Hermitian adjoint, extends to complex valued matrices through an additional conjugate of the individual matrix entries. In this book, we overload the term transpose for both cases with the understanding that conjugation should be applied when the underlying field is complex. More formally, the transpose of an $M \times K$ matrix $\mathbf{A}$ with entires $a(m, k)$ is the $K \times M$ matrix $\mathbf{A}^T$ with entries $\overline{a}(k, m)$.

For matrices $\mathbf{A}$ and $\mathbf{B}$ with compatible dimensions so that the operations below are well-defined, the following properties are often useful in manipulating equations:

$$(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T \tag{3.14}$$

$$(\mathbf{A} + \mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T. \tag{3.15}$$

3.2.2 | Matrix Addition

The addition of two matrices is performed by adding corresponding entries together. Let $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ denote $M \times K$ matrices. The expression $\mathbf{C} = \mathbf{A} + \mathbf{B}$ is written

explicitly as

$$\begin{bmatrix} c(0,0) & \cdots & c(0,K-1) \\ \vdots & \ddots & \vdots \\ c(M-1,0) & \cdots & c(M-1,K-1) \end{bmatrix} = \begin{bmatrix} a(0,0) & \cdots & a(0,K-1) \\ \vdots & \ddots & \vdots \\ a(M-1,0) & \cdots & a(M-1,K-1) \end{bmatrix} + \begin{bmatrix} b(0,0) & \cdots & b(0,K-1) \\ \vdots & \ddots & \vdots \\ b(M-1,0) & \cdots & b(M-1,K-1) \end{bmatrix} \quad (3.16)$$

where the right hand side simplifies to

$$\begin{bmatrix} a(0,0)+b(0,0) & \cdots & a(0,K-1)+b(0,K-1) \\ \vdots & \ddots & \vdots \\ a(M-1,0)+b(M-1,0) & \cdots & a(M-1,K-1)+b(M-1,K-1) \end{bmatrix} \quad (3.17)$$

consistent with the rule $c(m,k) = a(m,k) + b(m,k)$.

3.2.3 | Matrix Scalar Multiplication

The multiplication of a matrix by a scalar is performed by multiplying the scalar by each matrix entry. Let a denote a scalar and $\mathbf{B}$ and $\mathbf{C}$ denote $M \times K$ matrices. The expression $\mathbf{C} = a\mathbf{B}$ is written explicitly as

$$\begin{bmatrix} c(0,0) & \cdots & c(0,K-1) \\ \vdots & \ddots & \vdots \\ c(M-1,0) & \cdots & c(M-1,K-1) \end{bmatrix} = a \begin{bmatrix} b(0,0) & \cdots & b(0,K-1) \\ \vdots & \ddots & \vdots \\ b(M-1,0) & \cdots & b(M-1,K-1) \end{bmatrix} \quad (3.18)$$

$$= \begin{bmatrix} ab(0,0) & \cdots & ab(0,K-1) \\ \vdots & \ddots & \vdots \\ ab(M-1,0) & \cdots & ab(M-1,K-1) \end{bmatrix} \quad (3.19)$$

consistent with the rule $c(m,k) = ab(m,k)$.

3.2.4 | Matrix Vector Multiplication

The multiplication of a vector by a matrix produces a new vector through linear combinations of the original vectors entries. To see this, let $\mathbf{A}$ denote an $M \times K$ matrix and $\mathbf{x} \in \mathbb{F}^K$ and $\mathbf{y} \in \mathbb{F}^M$ vectors. Then, the entries of $\mathbf{y}$ produced by the matrix vector product $\mathbf{Ax}$ in terms of the expression

$$\begin{bmatrix} y(0) \\ \vdots \\ y(M-1) \end{bmatrix} = \begin{bmatrix} a(0,0) & \cdots & a(0,K-1) \\ \vdots & \ddots & \vdots \\ a(M-1,0) & \cdots & a(M-1,K-1) \end{bmatrix} \begin{bmatrix} x(0) \\ \vdots \\ x(K-1) \end{bmatrix} \tag{3.20}$$

are generated according to the rule

$$y(m) = \sum_{k=0}^{K-1} A(m,k)x(k). \tag{3.21}$$

For matrix vector products to be well defined, it is important that the number of columns of the matrix $\mathbf{A}$ to be equal to the number of entires in the vector $\mathbf{x}$.

Looking closely at (3.21) and the definition of a dot product, it follows that the scalar $y(m)$ is a dot product between the vector $\mathbf{x}$ and the m-th row of the matrix $\mathbf{A}$ when $\mathbf{A}$ is real valued, i.e. $y(m) = A(m,:) \cdot \mathbf{x} = A(m,:)^T\mathbf{x}$. Therefore, $\mathbf{y}$ can be interpreted as the collection of scalars corresponding to the inner product of $\mathbf{x}$ with each row vector in $\mathbf{A}$.

3.2.5 | Matrix Matrix Multiplication

The multiplication of one matrix by another produces a third matrix through effectively several matrix vector multiplications. To see this, consider the expression $\mathbf{AB}$ for an $M \times K$ matrix $\mathbf{A}$ and a $K \times N$ matrix $\mathbf{B}$. Then, the entires of the $M \times N$

matrix $\mathbf{C}$ produced by the matrix matrix product $\mathbf{AB}$ in terms of the expression

$$\begin{bmatrix} c(0,0) & \cdots & c(0,N-1) \\ \vdots & & \vdots \\ c(M-1,0) & \cdots & c(M-1,N-1) \end{bmatrix} = \begin{bmatrix} a(0,0) & \cdots & a(0,K-1) \\ \vdots & & \vdots \\ a(M-1,0) & \cdots & a(M-1,K-1) \end{bmatrix} \begin{bmatrix} b(0,0) & \cdots & b(0,N-1) \\ \vdots & & \vdots \\ b(K-1,0) & \cdots & b(K-1,N-1) \end{bmatrix} \tag{3.22}$$

are generated according to the rule

$$\begin{aligned} c(m,n) &= \sum_{k=0}^{K-1} a(m,k)b(k,n) && (3.23) \\ &= \mathbf{A}(m,:) \cdot \mathbf{B}(:,n). && (3.24) \end{aligned}$$

For a matrix matrix product to be well defined, it is important that the number of columns of $\mathbf{A}$ and the number rows of $\mathbf{B}$ are the same integer K. Generally speaking, matrix matrix multiplication is not commutative. Note that the column vectors in $\mathbf{C}$ are directly the matrix vector product of $\mathbf{A}$ with the column vectors in $\mathbf{B}$.

Inner Product Based

Inner product based implementations of matrix matrix multiplication can be thought of as computing the individual entries in $\mathbf{C}$ matrix either sequentially or in parallel. This follows directly from (3.24) by observing that the scalar $c(m,n)$ is the standard inner product between the m-th row of $\mathbf{A}$ and the n-th column of $\mathbf{B}$. In row major order this is conceptually computed sequentially in m then n with an inner loop over k to perform the dot product. An example of this procedure written in pseudocode is provided in Listing 3.1.

Listing 3.1: Pseudocode for inner product based matrix matrix multiplication organized in row major order.

```
1 for (m = 0; m < M; m++) {
2     for (n = 0; n < N; n++) {
3         c(m, n) = 0;
4         for (k = 0; k < K; k++) {
5             c(m, n) += a(m, k)*b(k, n);
6         }
7     }
8 }
```

The assignment $c(m, n) = 0$ in line 3 could be removed if the first iteration of the inner loop used = rather than + =. When both **A** and **B** are stored in row major format, memory accesses of **A** are contiguous whereas memory accesses of **B** are not. Producing each scalar $c(m, n)$ requires loading two length K vectors from memory and K multiply accumulate (MAC) operations.

Outer Product Based

Outer product based implementations of matrix matrix multiplication can be thought of as partially computing all elements in the **C** matrix at the same time. With the partial updates in row major order, this is conceptually computed sequentially with an outer loop over the index k as described by the pseudocode in Listing 3.2.

Listing 3.2: Pseudocode for outer product based matrix matrix multiplication.

```
1 C = 0;
2 for (k = 0; k < K; k++) {
3     for (m = 0; m < M; m++) {
4         for (n = 0; n < N; n++) {
5             c(m, n) += a(m, k)*b(k, n);
6         }
7     }
8 }
```

The assignment $C = 0$ in line 1 could be removed if the first iteration of the k

loop used = rather than + =. Opposite the inner product based implementation, memory accesses of $\mathbf{B}$ are contiguous whereas memory accesses of $\mathbf{A}$ are not when both matrices are stored in row major format. Each partial update affects all MN entries in the $\mathbf{C}$ matrix and requires 1 length M column of $\mathbf{A}$ and 1 length N row of $\mathbf{B}$ to be loaded from memory and MN MACs.

Loop Ordering

The above exposition highlights an important principle that runs throughout the entire book. Mathematical statements of an operation can often be translated to an implementation in several different ways, all of which produce correct results but whose subtle differences may in turn yield significant differences in practical metrics such as speed and memory usage. For example, the difference between the inner and outer product methods for implementing matrix matrix multiplication result from a simple reordering of loops, both of which are consistent with (3.23).

Block Based

Multiplying two large matrices can be broken into several multiplications through a procedure called tiling without introducing any extra MACs. Tiling is an example of a more general computer science pattern referred to as divide and conquer. For

simplicity, consider a large matrix matrix multiplication $\mathbf{C} = \mathbf{AB}$ written as

$$\begin{bmatrix} \mathbf{C}(0,0) & \cdots & \mathbf{C}(0,N-1) \\ \vdots & \ddots & \vdots \\ \mathbf{C}(M-1,0) & \cdots & \mathbf{C}(M-1,N-1) \end{bmatrix} = \begin{bmatrix} \mathbf{A}(0,0) & \cdots & \mathbf{A}(0,K-1) \\ \vdots & \ddots & \vdots \\ \mathbf{A}(M-1,0) & \cdots & \mathbf{A}(M-1,K-1) \end{bmatrix} \begin{bmatrix} \mathbf{B}(0,0) & \cdots & \mathbf{B}(0,N-1) \\ \vdots & \ddots & \vdots \\ \mathbf{B}(K-1,0) & \cdots & \mathbf{B}(K-1,N-1) \end{bmatrix} \tag{3.25}$$

where $\mathbf{C}(m,n)$, $\mathbf{A}(m,k)$ and $\mathbf{B}(k,n)$ are all $T \times T$ matrices referred to as tiles and

$$\mathbf{C}(m,n) = \sum_{k=0}^{K-1} \mathbf{A}(m,k)\mathbf{B}(k,n). \tag{3.26}$$

The dimensions of the tiled case are the same as the non tiled case scaled by the tile size T. Generalizations to non square tilings are straightforward by specifying tile sizes for the rows (T_{row}) and columns (T_{col}) separately. Often, zero-padding the initial large matrices can be used to produce manageable tile sizes.

Matrix matrix multiplication of the individual tiles as well as the overal large matrix matrix multiplication can be implemented using inner and outer product methods. If an inner product ordering is used and split across P processors, $1/P$-th of $\mathbf{C}$, $1/P$-th of $\mathbf{A}$ (or $\mathbf{B}$) and all of $\mathbf{B}$ (or $\mathbf{A}$) have to be distributed to all P processors, requiring excess memory and data movement for $\mathbf{B}$ (or $\mathbf{A}$). If an outer product ordering is used, $\mathbf{C}$ partial results, $1/P$-th of $\mathbf{A}$ and $1/P$-th of $\mathbf{B}$ have to be distributed to all P processors, requiring excess memory and data movement to re assemble to P partial products of $\mathbf{C}$.

Reducing Multiplications

A general strategy for reducing the total cost of computation when different types of computations have different associated costs, e.g. multiplications likely have higher cost than additions in hardware, is to decompose the full computation into a number of partial computations that can be reassembled to equal the total computation but at a reduced total cost. This strategy is consistent with the divide and conquer pattern mentioned previously and often underlies design choices made in mathematical software development.

An early example of this is Gauss' trick for multiplying two complex scalars together. Let a, b, c, d denote real scalars and consider the complex multiplication

$$(a + ib)(c + id) \quad = \quad (ac - bd) + i(ad + bc) \tag{3.27}$$

which requires four real multiplications and two real additions when computed as depicted. Alternatively, consider first forming the real scalars u, v, x, y, and z as

$$u = ac \qquad v = bd \qquad x = a + b \qquad y = c + d \qquad z = xy \ . \tag{3.28}$$

The cost of computing (3.28) is three real multiplications and two real additions. By appropriately rearranging these terms the desired result is produced as

$$ac - bd \quad = \quad u - v \tag{3.29}$$

$$ad + bc \quad = \quad z - u - v \tag{3.30}$$

which requires three more real additions and no multiplications. Therefore, the total cost of Gauss' method is three real multiplications and five real additions.

Following a similar conceptual strategy, large matrix matrix multiplications can

be broken into smaller ones via a few different methods with fewer multiplications than that of tile based inner or outer product methods. Strassen's algorithm, one of the most widely known, is illustrated next with large square matrices of size $M = N = K$ being a power of 2 but can be adapted to arbitrary sizes through zero row or column padding and/or the use of conventional matrix matrix multiplication. To demonstrate, consider the multiplication $\mathbf{C} = \mathbf{AB}$. Strassen's algorithm is based on the fact that tiled matrix multiplication can also be computed using the procedure:

1. Partition the $N \times N$ problem into a tiled problem using $(N/2) \times (N/2)$ tiles as

$$\begin{bmatrix} \mathbf{C}(0,0) & \mathbf{C}(0,1) \\ \mathbf{C}(1,0) & \mathbf{C}(1,1) \end{bmatrix} = \begin{bmatrix} \mathbf{A}(0,0) & \mathbf{A}(0,1) \\ \mathbf{A}(1,0) & \mathbf{A}(1,1) \end{bmatrix} \begin{bmatrix} \mathbf{B}(0,0) & \mathbf{B}(0,1) \\ \mathbf{B}(1,0) & \mathbf{B}(1,1) \end{bmatrix}$$

2. Define seven $(N/2)x(N/2)$ partial product matrices $\mathbf{S}_1, \ldots, \mathbf{S}_7$ using $(N/2) \times (N/2)$ matrix multiplications

$$\begin{aligned} \mathbf{S}_1 &= (\mathbf{A}(0,0) + \mathbf{A}(1,1))(\mathbf{B}(0,0) + \mathbf{B}(1,1)) \\ \mathbf{S}_2 &= (\mathbf{A}(1,0) + \mathbf{A}(1,1))\mathbf{B}(0,0) \\ \mathbf{S}_3 &= \mathbf{A}(0,0)(\mathbf{B}(0,1) - \mathbf{B}(1,1)) \\ \mathbf{S}_4 &= \mathbf{A}(1,1)(\mathbf{B}(1,0) - \mathbf{B}(0,0)) \\ \mathbf{S}_5 &= (\mathbf{A}(0,0) + \mathbf{A}(0,1))\mathbf{B}(1,1) \\ \mathbf{S}_6 &= (\mathbf{A}(1,0) - \mathbf{A}(0,0))(\mathbf{B}(0,0) + \mathbf{B}(0,1)) \\ \mathbf{S}_7 &= (\mathbf{A}(0,1) - \mathbf{A}(1,1))(\mathbf{B}(1,0) + \mathbf{B}(1,1)) \end{aligned}$$

3. Compute the four output tiles as additions of different linear combinations of the seven partial products

$$\begin{bmatrix} \mathbf{C}(0,0) & \mathbf{C}(0,1) \\ \mathbf{C}(1,0) & \mathbf{C}(1,1) \end{bmatrix} = \begin{bmatrix} \mathbf{S}_1 + \mathbf{S}_4 - \mathbf{S}_5 + \mathbf{S}_7 & \mathbf{S}_3 + \mathbf{S}_5 \\ \mathbf{S}_2 + \mathbf{S}_4 & \mathbf{S}_1 - \mathbf{S}_2 + \mathbf{S}_3 + \mathbf{S}_6 \end{bmatrix}$$

The partial product matrices required only one smaller matrix matrix multiplication, hence seven in total are required versus the eight required by conventional tiling methods. This seven versus eight advantage of Strassen versus conventional is extended in a straightforward way by recursive application to each of the seven partial product matrix multiplications. Carrying the recursion argument through to a base step, Strassen results in a multiplication complexity $\mathcal{O}(N^{\log_2 7}) \approx \mathcal{O}(N^{2.81})$ versus the $\mathcal{O}(N^3)$ exhibited by inner and outer product based methods. This difference in complexity becomes increasingly apparent as N becomes large.

While at first glance it may seem that Strassen matrix multiplication is advantageous, there are issues pertaining to memory, data movement, the number of required additions and numerical stability that often make it a less than ideal candidate. Moreover, additional algorithms such as Coppersmith-Winograd based methods have complexities on the order $\mathcal{O}(N^{2.37})$ that offer different trade offs for consideration. The details of these methods are outside the scope of this book.

Options

The exposition above highlights that conventional and Strassen based methods can be used to compute large matrix matrix multiplications from tiles of smaller matrix matrix multiplications. Different decompositions trade between memory, data movement and compute but accomplish the same goal. More broadly, different variations of algorithms are often optimal under different conditions and thus simply because an algorithm is optimal in some sense does not mean it is the right choice in all circumstances.

Consider a scenario wherein a conventional or Strassen based algorithm decomposes a larger matrix matrix multiplication into smaller tiles and uses either inner or outer product based matrix multiplication for the smaller tile size matrix matrix multiplications. This approach to decomposing large linear operations forshadows

later chapters wherein we decompose key CNN operations (large matrix matrix multiplications) to be implemented on computational hardware primitives (smaller tile size matrix matrix multiplications).

3.2.6 | Inversion

Square Matrix

Given linear systems of equations in the form $\mathbf{y} = \mathbf{A}\mathbf{x}$, the inverse matrix associated with $\mathbf{A}$, when it exists, allows for vectors $\mathbf{x}$ to be uniquely determined from vectors $\mathbf{y}$ through simple matrix vector multiplication. More formally, a square $K \times K$ matrix $\mathbf{A}$ has an associated inverse matrix $\mathbf{B}$ when the column vectors comprising $\mathbf{A}$ are linearly independent. In this case, the inverse is both a left and right inverse, i.e. $\mathbf{AB} = \mathbf{BA} = \mathbf{I}_K$ where $\mathbf{I}_K$ is an $K \times K$ identity matrix with ones on the principal diagonal and zeros elsewhere. The inverse of $\mathbf{A}$ is typically denoted $\mathbf{B} = \mathbf{A}^{-1}$. Indeed, the column independence condition above is equivalent to many other conditions including the rank of $\mathbf{A}$ being K, the row vectors forming $\mathbf{A}$ being linearly independent, etc. When a matrix is invertible, it is also referred to as nonsingular. Any matrix that is its own inverse, i.e. $\mathbf{A}^2 = \mathbf{I}_K$, is referred to as involutory.

For nonsingular square matrices $\mathbf{A}$ and $\mathbf{B}$ with compatible dimensions, the following properties are often useful in manipulating equations:

$$(\mathbf{A}^T)^{-1} = (\mathbf{A}^{-1})^T \tag{3.31}$$

$$(\mathbf{A}^{-1})^{-1} = \mathbf{A} \tag{3.32}$$

$$(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}. \tag{3.33}$$

If $\mathbf{A}$ is a diagonal matrix, then it is invertible if the diagonal entries are non-zero, i.e. $a(k,k) \neq 0$. The inverse matrix $\mathbf{B}$ is formed by taking the scalar inverse of each diagonal entry of $\mathbf{A}$, i.e. $b(k,k) = \frac{1}{a(k,k)}$.

Non Square Matrix

A non square $M \times K$ matrix $\mathbf{A}$ has a left or right inverse if the rank of $\mathbf{A}$ is M or K. In more detail, when the rank of $\mathbf{A}$ is M then $\mathbf{A}$ has a right inverse $\mathbf{B}$ such that $\mathbf{AB} = \mathbf{I}_M$. When the rank of $\mathbf{A}$ is K then $\mathbf{A}$ has a left inverse $\mathbf{B}$ such that $\mathbf{BA} = \mathbf{I}_K$.

Unitary Matrix

A unitary matrix has columns that are orthogonal (pairwise dot products equal zero) and each have unit norm. Said another way, the columns of a unitary matrix form an orthonormal basis. More formally, an $K \times K$ matrix $\mathbf{U}$ is referred to as unitary if the conjugate transpose $\mathbf{U}^H$ is the inverse matrix $\mathbf{U}^{-1}$, i.e.

$$\mathbf{U}^H\mathbf{U} \quad = \quad \mathbf{U}\mathbf{U}^H \quad = \quad \mathbf{I}_K. \tag{3.34}$$

Unitary matrices have the property that they preserve inner products in the sense that, for any vectors $\mathbf{x}$ and $\mathbf{y}$ and unitary matrix $\mathbf{U}$, the their inner products satisfy

$$\langle \mathbf{U}\mathbf{x}, \mathbf{U}\mathbf{y} \rangle \quad = \quad \langle \mathbf{x}, \mathbf{y} \rangle. \tag{3.35}$$

Orthogonal Matrix

Orthogonal matrices are essentially unitary matrices with only real valued entries. Indeed, an $K \times K$ real matrix $\mathbf{Q}$ is referred to as orthogonal if the transpose $\mathbf{Q}^T$ is the inverse matrix $\mathbf{Q}^{-1}$, i.e.

$$\mathbf{Q}^T\mathbf{Q} \quad = \quad \mathbf{Q}\mathbf{Q}^T \quad = \quad \mathbf{I}_K. \tag{3.36}$$

Orthogonal matrices also preserve inner products in the sense that, for any vectors $\mathbf{x}$ and $\mathbf{y}$ and orthgonal matrix $\mathbf{Q}$, their inner products satisfy

$$\langle \mathbf{Qx}, \mathbf{Qy} \rangle \quad = \quad \langle \mathbf{x}, \mathbf{y} \rangle. \tag{3.37}$$

3.2.7 | Hadamard Product

The Hadamard or Schur product allows for matrices to be multiplicatively combined through entry-wise operations. More formally, a Hadamard product $\odot$ is a map combining two $M \times K$ matrices $\mathbf{A}$ and $\mathbf{B}$ according to

$$\mathbf{A} \odot \mathbf{B} \quad = \quad \begin{bmatrix} a(0,0)b(0,0) & \cdots & a(0,K-1)b(0,K-1) \\ \vdots & \ddots & \vdots \\ a(M-1,0)b(M-1,0) & \cdots & a(M-1,K-1)b(M-1,K-1) \end{bmatrix} \tag{3.38}$$

Among other places, Hadamard products are instrumental in a variety of Fast Fourier Transform (FFT) algorithms in handling twiddle factor multiplication.

3.2.8 | Kronecker Product

The Kronecker product generalized vector outer products to matrix outer products. More formally, a Kronecker product $\otimes$ combines an $M \times K$ matrix $\mathbf{A}$ and a $L \times N$ matrix $\mathbf{B}$ to produce an $ML \times KN$ matrix $A \otimes B$ according to

$$\mathbf{A} \otimes \mathbf{B} \quad = \quad \begin{bmatrix} a(0,0)\mathbf{B} & \cdots & a(0,K-1)\mathbf{B} \\ \vdots & & \vdots \\ a(M-1,0)\mathbf{B} & \cdots & a(M-1,K-1)\mathbf{B} \end{bmatrix} \tag{3.39}$$

The Kronecker product can be thought of as the matrix of tensor product with respect to the standard basis and is sometimes easier to work with from a notational

perspective than the full tensor product. Importantly, Kronecker products are not commutative, i.e. $\mathbf{A} \otimes \mathbf{B} \neq \mathbf{B} \otimes \mathbf{A}$ for general $\mathbf{A}$ and $\mathbf{B}$.

3.2.9 | Vectorization

Vectorization produces a vector by flattening a matrix or tensor. More formally, vectorization is a linear transform that stacks all K columns of a $M \times K$ matrix $\mathbf{A}$ to produce an $MK \times 1$ vector according to

$$\text{vec}(\mathbf{A}) = \begin{bmatrix} \mathbf{A}(:,0) \\ \vdots \\ \mathbf{A}(:,K-1) \end{bmatrix}. \tag{3.40}$$

Vectorization is a useful tool in expression some matrix derivatives, frequently in conjunction with the Kronecker product. In anticipation of this, for matrices $\mathbf{A}, \mathbf{B}$, and $\mathbf{C}$ with compatible dimensions, the following identities are often useful in manipulating equations:

$$\begin{aligned} \text{vec}(\mathbf{ABC}) &= (\mathbf{C}^T \otimes \mathbf{A})\text{vec}(\mathbf{B}) && (3.41) \\ &= (\mathbf{I}_N \otimes \mathbf{AB})\text{vec}(\mathbf{C}) && (3.42) \\ &= (\mathbf{B}^T\mathbf{C}^T \otimes \mathbf{I}_K)\text{vec}(\mathbf{A}). && (3.43) \end{aligned}$$

3.2.10 | Trace

For an $K \times K$ matrix $\mathbf{A}$, the trace is the sum of the elements on the principal diagonal. The trace also equals the sum of the eigenvalues of $\mathbf{A}$ and is closely related to the determinant of $\mathbf{A}$.

3.2.11 | Determinant

The determinant of an $K \times K$ matrix $\mathbf{A}$ can be computed as the product of that matrices eigenvalues. Geometrically, the determinant of a matrix is a measure of the volume of the polytope defined by the column vectors of $\mathbf{A}$.

3.3 | Transformations

3.3.1 | Linear Transform

Let $\mathcal{X}$ and $\mathcal{Y}$ denote vector spaces and $T\colon \mathcal{X} \to \mathcal{Y}$ a linear map between them. From the definition of linearity it follows that for all vectors $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ and scalars a the map T satisfies the following properties:

- Additivity: $T(\mathbf{x} + \mathbf{z}) = T(\mathbf{x}) + T(\mathbf{z})$
- Homogeneity: $T(a\mathbf{x}) = aT(\mathbf{x})$

One of the most powerful perspectives in understanding matrix properties is to view them through the lens of linear transformations or maps. This perspective is appropriate since every linear map can be represented by a matrix and every matrix represents a linear map.

There are four fundamental subspaces used to capture the behavior of a linear map T. The first two deal with special sets of inputs and outputs of T whereas the second two deal with companion sets of inputs and outputs of T^T.

- The range or image or column space of T is the vector subspace of $\mathcal{Y}$ comprising all vectors T can produce and is denoted $\text{range}(T) = \{T(\mathbf{x}) \in \mathcal{Y}\colon \mathbf{x} \in \mathcal{X}\}$.
- The null space or kernel of T is the vector subspace of $\mathcal{X}$ comprising all vectors T maps to $\mathbf{0}$ and is denoted $\text{null}(T) = \{\mathbf{x} \in \mathcal{X}\colon T(\mathbf{x}) = \mathbf{0}\}$.
- The row space or coimage of T is the vector subspace of $\mathcal{X}$ comprising all vectors T^T can produce and is denoted $\text{range}(T^T) = \{T^T(\mathbf{y}) \in \mathcal{X}\colon \mathbf{y} \in \mathcal{Y}\}$.

- The left null space or cokernel of T is the vector subspace of $\mathcal{Y}$ comprising all vectors T^T maps to $\mathbf{0}$ and is denoted $\text{null}(T^T) = \{\mathbf{y} \in \mathcal{Y} \colon T^T(\mathbf{y}) = \mathbf{0}\}$.

As mentioned above, linear transformations T between finite dimensional vector spaces can always be expressed as matrices and implemented as matrix vector multiplication of the form

$$\mathbf{y} = \mathbf{A}\mathbf{x} \tag{3.44}$$

where $\mathbf{A}$ is the $M \times K$ matrix representation of T, $\mathbf{x}$ is the length K input and $\mathbf{y}$ is the length M output. The range of A is then the vector space formed by the span of the set of column vectors forming $\mathbf{A}$. The number of linearly independent columns or equivalently the dimesion of that span is the rank of $\mathbf{A}$ and satisfies $\text{rank}(\mathbf{A}) \leq \min(M, K)$.

Linear transformations implemented as matrix multiplication and various generalizations to handling multiple inputs and producing multiple outputs are often used in the body of a CNN mapping inputs to features as well as in the head of the CNN mapping the features into classes.

3.3.2 | Affine Transform

Affine transformations are closely related to linear transformations in the sense that they can be implemented as a linear transformation followed by an offset or bias. To see this, consider the affine transformation from $\mathbf{x}$ to $\mathbf{y}$ given by

$$\mathbf{y} \;=\; \mathbf{A}\mathbf{x} + \mathbf{b} \tag{3.45}$$

where $\mathbf{A}$ denotes the linear transformation and $\mathbf{b}$ the bias. Rearranging terms to augment $\mathbf{A}$ with an additional column and $\mathbf{x}$ an additional row yields

$$\begin{aligned} \mathbf{y} &= [\mathbf{A} \ \mathbf{b}] \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} && (3.46) \\ &= \mathbf{A}_{\text{aug}} \mathbf{x}_{\text{aug}} && (3.47) \end{aligned}$$

where the inner matrix dimension of the augmented linear transformation has increased from K to $K + 1$.

Several important CNN layers including convolution and fully connected layers with bias addition are implemented using this type of augmentation, typically generalized to multiple inputs and outputs, with a complexity increase of one MAC and one input data movement per output scalar.

3.3.3 | Compositions

The composition of linear transformations reduces to a single linear transformation. More concretely, consider the successive application of S linear transformations $\mathbf{A}_s$ summarized into the transformation $\mathbf{A}$ as

$$\mathbf{A} = \mathbf{A}_0 \cdots \mathbf{A}_{S-1}. \quad (3.48)$$

Without some form of nonlinear transformation between linear transformations, the effective number of linear transformations is one. This important observation relates to the concept of depth in neural networks and why directly repeating linear transformations does not correspond to producing deeper networks.

3.3.4 | Principal Component Analysis

Principal component analysis (PCA) determines a collection of uncorrelated variables best describing strong variations in experimental data. In particular, let $\mathbf{X}$ denote a data matrix with M rows each corresponding to a different experimental trial and K zero mean possibly linearly correlated columns each corresponding to a different measurement from the experiment. The PCA transformation refers to the $K \times K$ orthogonal linear transformation $\mathbf{Q}$ that converts the data matrix $\mathbf{X}$ into an $M \times K$ matrix of principal components $\mathbf{Y}$ according to

$$\mathbf{Y} = \mathbf{XQ} \tag{3.49}$$

where each column of $\mathbf{Y}$ is orthogonal to the other columns and ordered from largest to smallest variance.

The PCA transformation matrix $\mathbf{Q}$ is readily accessible from the singular value decomposition (discussed in the matrix decompositions section) of $\mathbf{X}$. To see this, let the SVD of $\mathbf{X}$ be $\mathbf{X} = \mathbf{USV}^T$ and let $\mathbf{Q} = \mathbf{V}$. Then $\mathbf{Y} = \mathbf{XQ} = \mathbf{USV}^T\mathbf{V} = \mathbf{US}$ where the columns of $\mathbf{U}$ are the column vectors in $\mathbf{Y}$ and the corresponding variances are along the principal diagonal of the $\mathbf{S}$ matrix. Principal component analysis can be used for dimensionality reduction by keeping only the $L < K$ most significant columns of $\mathbf{Q}$.

3.3.5 | Discrete Fourier Transform

Discrete Fourier Transform (DFT)

The Discrete Fourier Transform (DFT) produces frequency domain descriptions of vectors. Said another way, the DFT transforms the basis used to represent a vector to harmonically related complex exponentials. In detail, the Discrete Fourier Transform

is an $K \times K$ unitary matrix $\mathbf{F}_K$ defined by

$$\mathbf{F}_K = \frac{1}{\sqrt{K}} \begin{bmatrix} e^{-i\frac{2\pi}{K}(0)(0)} & \dots & e^{-i\frac{2\pi}{K}(0)(K-1)} \\ \vdots & \ddots & \vdots \\ e^{-i\frac{2\pi}{K}(K-1)(0)} & \dots & e^{-i\frac{2\pi}{K}(K-1)(K-1)} \end{bmatrix}. \tag{3.50}$$

The inverse DFT matrix $\mathbf{F}_K^{-1} = \mathbf{F}_K^T$ which follows from the property that the transpose of a unitary matrix is the inverse. For K-dimensional vectors $\mathbf{x}$ whose entries are interpretable with units of t, the DFT matrix transforms $\mathbf{x}$ to the K-dimensional vector $\mathbf{y}$ according to

$$\mathbf{y} = \mathbf{F}_K \mathbf{x} \tag{3.51}$$

where the entries of $\mathbf{y}$ have units interpretable as $1/t$. In a variety of applications, the interpretation of t often pertains to time or space.

Many real world vectors including audio signals and images have non-uniform distributions of information in this $1/t$ domain, thereby enabling the DFT to be used for dimensionality reduction by compressing the vector by keeping the $L < K$ elements of $\mathbf{y}$.

Fast Fourier Transform (FFT) Algorithms

The FFT is an efficient algorithm for the computation of the DFT. Most variants use a divide and conquor style approach with a combination of smaller matrix and pointwise matrix multiplications to convert from $O(K^2)$ operations to $O(K \log(K))$ operations.

3.3.6 | Filtering

To bridge into the CNN chapter, the filtering section will use subscripts r and c for vector and matrix lengths. To help with intuition, think of r as rows and c as columns.

1D

The convolution of a length N_r vector $\mathbf{x}$ and a length F_r vector $\mathbf{h}_{\text{conv}}$ is a length $N_r + F_r - 1$ vector with elements

$$y_{\text{conv}}(n_r) = \sum_{\tau_r=0}^{F_r-1} h_{\text{conv}}(\tau_r)x(n_r - \tau_r) \tag{3.52}$$

1D filtering constrains convolution to the indices of total overlap between $\mathbf{x}$ and $\mathbf{h}_{\text{conv}}$ resulting in a length $N_r - F_r + 1$ vector composed of indices $n_r = F_r - 1, \ldots, N_r - 1$ from the above equation.

The correlation of a length N_r vector $\mathbf{x}$ and a length F_r vector $\mathbf{h}_{\text{corr}}$ is a length $N_r + F_r - 1$ vector

$$y_{\text{corr}}(n_r) = \sum_{\tau_r=0}^{F_r-1} h_{\text{corr}}(\tau_r)x(n_r + \tau_r) \tag{3.53}$$

Constraining correlation to the indices of total overlap between $\mathbf{x}$ and $\mathbf{h}_{\text{corr}}$ results in a length $N_r - F_r + 1$ vector composed of indices $n_r = 0, \ldots, N_r - F_r$ from the above equation.

If $\mathbf{h}_{\text{corr}} = \text{flip}(\mathbf{h}_{\text{conv}})$ then

$$y_{\text{corr}}(n_r) = y_{\text{conv}}(n_r + F_r - 1) \tag{3.54}$$

As such, convolution and correlation are equivalent with an appropriate ordering of $\mathbf{h}$ and selection of indices n_r.

In general, CNNs designs will say convolution but mean correlation of indices of total overlap. The remainder of the book will follow the same convention and drop

the subscript corr unless it's explicitly stated that another convention is being used.

Correlation with total overlap can be written in terms of vector matrix multiplication. Using the subscript 1D to keep track of the filter type and defining the length $N_r - F_r + 1$ output vector as

$$\mathbf{y}_{1\mathrm{D}}^T = [y(0) \cdots y(N_r - F_r)] \tag{3.55}$$

and the length F_r filter vector as

$$\mathbf{h}_{1\mathrm{D}}^T = [h(0) \cdots h(F_r - 1)] \tag{3.56}$$

the output can be written as

$$\begin{aligned} \mathbf{y}_{1\mathrm{D}}^T &= \mathbf{h}_{1\mathrm{D}}^T \begin{bmatrix} x(0) & \cdots & x(N_r - F_r) \\ \vdots & & \vdots \\ x(F_r - 1) & \cdots & x(N_r - 1) \end{bmatrix} && (3.57) \\ &= \mathbf{h}_{1\mathrm{D}}^T \mathbf{X}_{1\mathrm{D}} && (3.58) \end{aligned}$$

with a F_r x $(N_r - F_r + 1)$ input filtering matrix $\mathbf{X}_{1\mathrm{D}}$ (a subset of a Toeplitz matrix). Note that there are a few ways of doing this that result in different orderings of the output. The choice here sets up the output of CNN style 2D convolution to have each output feature map on a separate row.

2D

The 2D correlation with indices of total overlap between a N_r x N_c matrix $\mathbf{X}$ and a F_r x F_c matrix $\mathbf{H}$ is

$$y_{\mathrm{corr}}(n_r, n_c) = \sum_{\tau_r=0}^{F_r-1} \sum_{\tau_c=0}^{F_c-1} h_{\mathrm{corr}}(\tau_r, \tau_c) x(n_r + \tau_r, n_c + \tau_c) \tag{3.59}$$

where $n_r = 0, \ldots, N_r - F_r$ and $n_c = 0, \ldots, N_c - F_c$.

As in the 1D case, 2D correlation with total overlap can be written in terms of vector matrix multiplication via vectorizing the inner product operations of correlation. Using the subscript 2D to keep track of the filter type and defining the length $(N_r - F_r + 1)(N_c - F_c + 1)$ output vector as

$$\mathbf{y}_{2\mathrm{D}}^T = [y(0,0) \cdots y(0, N_c - F_c) \cdots y(N_r - F_r, 0) \cdots y(N_r - F_r, N_c - F_c)] \tag{3.60}$$

and the length $F_r F_c$ filter vector as

$$\mathbf{h}_{2\mathrm{D}}^T = [h(0,0) \cdots h(0, F_c - 1) \cdots h(F_r - 1, 0) \cdots h(F_r - 1, F_c - 1)] \tag{3.61}$$

the output can be written as

$$\begin{aligned}
\mathbf{y}_{2\mathrm{D}}^T &= \mathbf{h}_{2\mathrm{D}}^T \begin{bmatrix}
x(0,0) & \cdots & x(0, N_c - F_c) & \cdots & x(N_r - F_r, 0) & \cdots & x(N_r - F_r, N \\
\vdots & & \vdots & & \vdots & & \vdots \\
x(0, F_c - 1) & \cdots & x(0, N_c - 1) & \cdots & x(N_r - F_r, F_c - 1) & \cdots & x(N_r - F_r, N \\
\vdots & & \vdots & & \vdots & & \vdots \\
x(F_r - 1, 0) & \cdots & x(F_r - 1, N_c - F_c) & \cdots & x(N_r - 1, 0) & \cdots & x(N_r - 1, N_c \\
\vdots & & \vdots & & \vdots & & \vdots \\
x(F_r - 1, F_c - 1) & \cdots & x(F_r - 1, N_c - 1) & \cdots & x(N_r - 1, F_c - 1) & \cdots & x(N_r - 1, N
\end{bmatrix} \\
&= \mathbf{h}_{2\mathrm{D}}^T \mathbf{X}_{2\mathrm{D}}
\end{aligned}$$

with a $F_r F_c$ x $(N_r - F_r + 1)(N_c - F_c + 1)$ input filtering matrix $\mathbf{X}_{2\mathrm{D}}$.

CNN Style 2D

CNN style 2D convolution generalizes 2D convolution to N_i input feature maps (matricies), N_o output feature maps (matricies) and $N_i N_o$ filters, 1 connecting each input

feature map to each output feature map. N_i 2D convolutions, 1 for each input feature map, are summed together to produce each output feature map.

Adopting the matrix notation from the previous subsection and adding subscripts to index input and output feature maps, let $\mathbf{y}^T_{\text{2D:}n_o}$ be the n_oth output feature map, $\mathbf{h}^T_{\text{2D:}n_o,n_i}$ be the filter connecting input feature map n_i to output feature map n_o and $\mathbf{X}_{\text{2D:}n_i}$ be the n_ith input feature map. CNN style 2D convolution is

$$\begin{bmatrix} \mathbf{y}^T_{\text{2D:}0} \\ \vdots \\ \mathbf{y}^T_{\text{2D:}N_o-1} \end{bmatrix} = \begin{bmatrix} \mathbf{h}^T_{\text{2D:}0,0} & \cdots & \mathbf{h}^T_{\text{2D:}0,N_i-1} \\ \vdots & & \vdots \\ \mathbf{h}^T_{\text{2D:}N_o-1,0} & \cdots & \mathbf{h}^T_{\text{2D:}N_o-1,N_i-1} \end{bmatrix} \begin{bmatrix} \mathbf{X}_{\text{2D:}0} \\ \vdots \\ \mathbf{X}_{\text{2D:}N_i-1} \end{bmatrix} \tag{3.64}$$

$$\mathbf{Y}_{\text{CNN}} = \mathbf{H}_{\text{CNN}}\mathbf{X}_{\text{CNN}} \tag{3.65}$$

Re capping dimensions, the final equation shows N_0 x $(N_r - F_r + 1)(N_c - F_c + 1)$ matrix $\mathbf{Y}_{\text{CNN}}$, N_0 x $F_rF_cN_i$ matrix $\mathbf{H}_{\text{CNN}}$ and the $F_rF_cN_i$ x $(N_r - F_r + 1)(N_c - F_c + 1)$ matrix $\mathbf{X}_{\text{CNN}}$.

CNN style 2D convolution is matrix multiplication with a filtering matrix $\mathbf{X}_{\text{CNN}}$. If F_r x F_c = 1 x 1 then there's no data repetition in $\mathbf{X}_{\text{CNN}}$ and CNN style 2D convolution reduces to traditional matrix multiplication without the data transformation complexity of the filtering matrix. For fully grouped CNN style 2D convolution $\mathbf{H}_{\text{CNN}}$ is block diagonal and CNN style 2D convolution reduces to $N_i = N_o$ traditional 2D convolutions.

Frequency Domain

Deptnding on filter sizes and hardware memory, data movement and compute capabilities, it's occasionally more efficient (from a practical perspective) to implement filtering in the frequency domain. This relies on circular convolution in the time domain being equivalent to pointwise multiplication in the frequency domain and the FFT/IFFT is an efficient method for transforming between domains.

In the 1D case, length N_r vector $\mathbf{x}$ and length F_r vector $\mathbf{h}$ are both zero padded to length $N_r + F_r - 1$, transformed to the frequency domain via the FFT, pointwise multiplied, then transformed back to the time domain via the IFFT. Typically, this works best if multiple inputs are each filtered via the same $\mathbf{h}$, so the process requires 2 FFTs vs 3. Extensions to 2D and CNN style 2D convolution are straightforward.

For systems with large numbers of small filters applied to large inputs, the additional memory and associated data movement required for filter storage can end up dominating the performance. As such, for CNNs the most likely place to benefit from this is early in the network where the product $N_i N_o$ is smaller.

Reducing Multiplications

As in the case of matrix multiplication, filtering implementation exist with less than the standard number of multiplications. This should be evident from the realization that CNN style 2D convolution is actually matrix multiplication and as such Strassen's algorithm applies.

Beyond a generic application of Strassen, however, it's possible to use a version optimized for the specific convolution strucutre.

3.4 | Matrix Decompositions

Matrix decompositions are commonly used in transformations and solutions of linear systems of equations.

3.4.1 | Eigen Decomposition

An eigenvector $\mathbf{v}$ of a K x K square matrix $\mathbf{A}$ is a non zero vector that satisfies

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v} \tag{3.66}$$

where λ is a scalar referred to as the assodiated eigenvalue. In other words, $\mathbf{A}$ is a linear transformation of an eigenvector to a scaled version of itself. If $\mathbf{A}$ has K linearly independent eigenvectors, then $\mathbf{A}$ can be factored as

$$\mathbf{A} = \mathbf{QDQ}^{-1} \tag{3.67}$$

where $\mathbf{Q}$ is an orthogonal matrix with eigenvectors as columns and $\mathbf{D}$ is a diagonal matrix with associated eigenvalues as diagonal elements. This is referred to as the eigen decomposition of $\mathbf{A}$ and is commonly computed via a power method and deflation.

Given an eigen decomposition, the inverse of the matrix $\mathbf{A}$ can be found easily as

$$\begin{aligned} \mathbf{A}^{-1} &= (\mathbf{QDQ}^{-1})^{-1} && (3.68) \\ &= \mathbf{QD}^{-1}\mathbf{Q}^{-1} && (3.69) \end{aligned}$$

taking advantage of simple inversion formulas for orthogonal and diagonal matricies and products of matricies.

3.4.2 | Singular Value Decomposition

The SVD of an arbitrary M x K matrix $\mathbf{A}$ is a weighted outer product of 2 matricies

$$\mathbf{A} = \mathbf{USV}^H \tag{3.70}$$

where $\mathbf{U}$ is a M x M orthogonal matrix, $\mathbf{S}$ is a M x K diagonal matrix of singular values and $\mathbf{V}^H$ is a K x K orthogonal matrix.

The columns of $\mathbf{U}$ are the eigenvectors of $\mathbf{AA}^H = \mathbf{USV}^H\mathbf{VS}^H\mathbf{U}^H = \mathbf{USS}^H\mathbf{U}^H$. To see this let $\mathbf{Q} = \mathbf{U}$, $\mathbf{D} = \mathbf{SS}^H$ and note $\mathbf{Q}^{-1} = \mathbf{U}^H$ in the eigen decomposition. Initial columns corresponding to the non zero singular values span the column space

of $\mathbf{A}$. Last columns corresponding to zero singular values span the left null space of $\mathbf{A}$.

The number of non zero singular values is the rank of $\mathbf{A}$ and the ratio of the largest to the smallest singular value is the condition number.

The columns of $\mathbf{V}^H$ are the eigenvectors of $\mathbf{A}^H\mathbf{A} = \mathbf{V}\mathbf{S}^H\mathbf{U}^H\mathbf{U}\mathbf{S}\mathbf{V}^H = \mathbf{V}\mathbf{S}^H\mathbf{S}\mathbf{V}^H$. To see this let $\mathbf{Q} = \mathbf{V}$, $\mathbf{D} = \mathbf{S}^H\mathbf{S}$ and note $\mathbf{Q}^{-1} = \mathbf{V}^H$ in the eigen decomposition. Last columns corresponding to zero singular values span the null space of $\mathbf{A}$. Initial columns corresponding to the non zero singular values span the row space of $\mathbf{A}$.

The SVD can be used for low rank approximations and pseudo inverse calculations.

3.5 | Optimization

3.5.1 | Linear Systems Of Equations

Consider the linear system of equations $\mathbf{y} = \mathbf{A}\mathbf{x}$ where $\mathbf{y}$ and $\mathbf{A}$ are known and the goal is to solve for $\mathbf{x}$. There are at least 2 levels to think about solutions to equations of this form

- A theoretical level to understand if a solution exists, what form it takes and how to mathematically write the solution
- A computational level to understand how to accurately and efficiently numerically compute the solution

This section is a exceedingly brief look into the theoretical level from the perspective of an optimization problem to connect with later material in the book.

Uniquely Determined

Over Determined

Under Determined

Chapter 4

Calculus

This chapter spends a little time on integrals so the continual random variable section hangs together and a lot of time on derivatives because they're at the core of learning. While it would be nice to have balance between the integral and derivative sections for aesthetic and theoretical reasons, that will have to wait for a future version.

The following items complicate the presentation of information on derivatives:

- It's nice to use vector notation for vector functions of multiple variables
- Different options for dimensions are possible for derivatives of vector functions of multiple variables
- The choice of dimensions affects the resulting derivatives and associated rules
- The choice of dimensions affects the use of the derivatives in implementations of machine learning algorithms

So basically a lot of stuff is hanging on getting this correct and as clean as possible.

4.1 | Derivatives

Derivatives provide information about the sensitivity of functions and are useful tools in finding extrema such as local and global minimums, maximums and saddle points.

In this section, we explore properties of derivatives in the single and multi variable cases.

4.1.1 | Single Variable

We first turn our attention to the definition and properties of derivatives of scalar valued functions $f, g, h \colon \mathbb{R} \to \mathbb{R}$ mapping real-valued inputs to real-valued outputs.

Differentiable Functions

Derivatives are linear operators mapping functions to functions. In particular, the derivative of the function f at the point x is defined according to

$$\frac{d}{dx} f(x) \quad = \quad \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{4.1}$$

and has the visual interpretation of being the slope of the tangent line to the graph of f and x. Intuitively, the derivative represents the rate of change of the function f at x. At values of x for which the limit in (4.1) exists, f is said to be differentiable at x. Likewise, at values of x where the limit in (4.1) does not exist, f is said to not be differentiable at x.

Not Differentiable Functions

Functions having one or more points where they are not differentiable come into play during training for some layers commonly used in CNNs. If there are a countable number of discrete points where the derivative is not defined then the numerical workaround tends to be to approximate the derivative by a point just to the left or right of the non-differentiable point or to use the average of the left and right limits in (4.1). Such approximations are consistent with the use of subgradients in convex analysis. In more complex settings, differential approximations to non differentiable

layers are also used. Convolutional neural networks tens to work best when trained end to end and layer differentiability is important to this.

Rules

Several rules pertaining to derivative operators are useful in manipulating and organizing equations into forms amenable to implementation. As was mentioned above, the derivative operator is a linear operator. Therefore, the derivative of a weighted sum of functions is the weighted sum of the derivatives of the individual functions. As an example of this, the derivative of the weighted sum of two functions reduces to

$$\frac{d}{dx}\left(c_1 f(x) + c_2 g(x)\right) = c_1 \frac{d}{dx} f(x) + c_2 \frac{d}{dx} g(x) \tag{4.2}$$

where c_1 and c_2 are constant scalars. The product rule is useful for taking derivatives of products of functions. For the case of two functions, the derivative of their product reduces in a straightforward manner to

$$\frac{d}{dx}\left(f(x)g(x)\right) = \left(\frac{d}{dx} f(x)\right) g(x) + f(x) \left(\frac{d}{dx} g(x)\right). \tag{4.3}$$

Closed-form expressions of the product rule applied to arbitrarily many functions immediately follows from iterative application of (4.3). The quotient rule is useful for taking derivatives of rational functions. In more detail, the derivative of the ratio of two functions $\frac{f(x)}{g(x)}$ is

$$\frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = \frac{\left(\frac{d}{dx} f(x)\right) g(x) - f(x) \left(\frac{d}{dx} g(x)\right)}{g^2(x)} \tag{4.4}$$

An important special case of the quotient rule occurs when the numerator $f(x) = 1$ which results in the expression

$$\frac{d}{dx}\left(\frac{1}{g(x)}\right) = -\frac{\frac{d}{dx}g(x)}{g^2(x)}. \tag{4.5}$$

The chain rule is useful for taking derivatives of compositions of functions. Namely, let $h(x) = f(g(x)) = f \circ g(x)$ and define $u = g(x)$, then

$$\frac{d}{dx}h(x) = \frac{d}{du}f(u)\frac{d}{dx}u(x). \tag{4.6}$$

The chain rule is used in training CNNs for back propagating error information from layer outputs to layer inputs as well as providing the direction for parameter updates.

Examples

The use of scalar-valued, nonlinear functions is pervasive throughout the machine learning discipline and notably in the design of CNN architectures. Common examples of these functions and their derivatives include

$$\begin{aligned}
\text{sigmoid}(x) &= \frac{1}{1+e^{-x}} & \frac{d}{dx}\text{sigmoid}(x) &= \text{sigmoid}(x)(1-\text{sigmoid}(x)) \\
\tanh(x) &= \frac{e^x - e^{-x}}{e^x + e^{-x}} & \frac{d}{dx}\tanh(x) &= 1 - \tanh^2(x) \\
\text{ReLU}(x) &= \text{maximum}(0, x) & \frac{d}{dx}\text{ReLU}(x) &= \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases}
\end{aligned}$$

Taking the derivative of many functions including hyperbolic or trigonometric functions often follow from first representing them using a power series expansion and then taking advantage of the linearity of the derivative operator.

4.1.2 | Multiple Variables

In this section, we generalize the derivative properties of scalar valued functions to the corresponding results for real-valued functions $f\colon \mathbb{R}^N \to \mathbb{R}^M$ mapping N-dimensional vectors to M-dimensional vectors. The treatmenet in this section begins with the important special case of $M = 1$ before developing into cases with $M > 1$.

Partial Derivatives

Partial derivatives of scalar valued functions of multiple input variables encode the sensitivities of those functions to a particular variable where all other variables are treated as scalar constants. In more detail, the partial derivative of the multivariate function $f\colon \mathbb{R}^N \to \mathbb{R}$ at the point $\mathbf{x} \in \mathbb{R}^N$ with regard to the n-th variable $\mathbf{x}_n$ is

$$\frac{\partial}{\partial \mathbf{x}_n} f(\mathbf{x}) \quad = \quad \lim_{\epsilon \to 0} \frac{f(\mathbf{x} + \epsilon \mathbf{e}_n) - f(\mathbf{x})}{\epsilon}. \tag{4.7}$$

The value returned by the partial derivative is the slope of the function f in the coordinate direction $\mathbf{e}_n$ at the point $\mathbf{x}$. Said another way, this can be viewed as taking the regular derivative of a new function of a single variable where all variables x_k for $k \neq n$ become constants.

Gradient

Gradients generalize derivatives of scalar functions to encode the sensitivities of functions of multiple variables with respect to all input variables. In particular, the gradient of a function $f\colon \mathbb{R}^N \to \mathbb{R}$ at the point $\mathbf{x} \in \mathbb{R}^N$ is defined as

$$\nabla f(\mathbf{x}) \quad = \quad \begin{bmatrix} \frac{\partial}{\partial \mathbf{x}_1} f(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial \mathbf{x}_N} f(\mathbf{x}) \end{bmatrix}. \tag{4.8}$$

Note that the gradient of the function f with respect to $\mathbf{x}$ is a vector of the same shape and dimensionality as $\mathbf{x}$.

Gradient operators can generally be defined in a variety of ways, all of which being mathematically rigorous but for which careful bookkeeping must be kept for notational consistency. The definition above and used throughout this book is the so-called denominator layout notation. In short, this means gradient of scalar-valued functions of N-dimensional column vector inputs will produce N-dimensional column vectors whereas numerator layout formats would produce transposed configurations.

Directional Derivatives

Directional derivatives generalize partial derivatives to provide the slope of a function in the direction of an arbitrary unit vector. In particular, the directional derivative of the function $f\colon \mathbb{R}^N \to \mathbb{R}$ at the point $\mathbf{x} \in \mathbb{R}^N$ in the unit direction $\mathbf{u} \in \mathbb{R}^N$ is written using the notation $D_{\mathbf{u}} f(\mathbf{x})$ and defined according to

$$D_{\mathbf{u}} f(\mathbf{x}) \;=\; \nabla f(\mathbf{x}) \cdot \mathbf{u} \;=\; \nabla f(\mathbf{x})^T \mathbf{u}. \tag{4.9}$$

Note that the partial derivative is a special case of the directional derivative when the unit vector $\mathbf{u}$ is chosen to be the coordinate direction $\mathbf{e}_n$, i.e. $\frac{\partial}{\partial \mathbf{x}_n} f(\mathbf{x}) = D_{\mathbf{e}_n} f(\mathbf{x})$

Jacobian

Jacobians encode the sensitivities of multivariate functions producing multiple output values using all possible partial derivatives between inputs and outputs. In particular, the Jacobian of a function $f\colon \mathbb{R}^N \to \mathbb{R}^M$ at the point $\mathbf{x} \in \mathbb{R}^N$ is the matrix of partial derivatives where the (m, n) entry is the partial derivative of the m-th output $f_m(\mathbf{x})$ with respect to the n-th input variable $\mathbf{x}_n \in \mathbb{R}^N$ in the coordiante direction $\mathbf{e}_n \in \mathbb{R}^N$

and can be computed according to

$$J_f(\mathbf{x}) = \begin{bmatrix} \nabla f_1(\mathbf{x})^T \\ \vdots \\ \nabla f_M(\mathbf{x})^T \end{bmatrix} \tag{4.10}$$

$$= \begin{bmatrix} \frac{\partial}{\partial \mathbf{x}_1} f_1(\mathbf{x}) & \cdots & \frac{\partial}{\partial \mathbf{x}_N} f_1(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial \mathbf{x}_N} f_M(\mathbf{x}) & \cdots & \frac{\partial}{\partial \mathbf{x}_N} f_M(\mathbf{x}) \end{bmatrix}. \tag{4.11}$$

The Jacobian is interpreted as the optimal linear approximation of a vector valued function of multiple variables near the point $\mathbf{a}$ in the sense

$$f(\mathbf{x}) \approx f(\mathbf{a}) + J_F(\mathbf{a})(\mathbf{x} - \mathbf{a}). \tag{4.12}$$

The Jacobian matrix is sometimes referred to as the total derivative matrix and is used in the definition of total derivatives.

Similar to the notational freedom discussed surrounding gradients, a similar choice can be made in defining Jacobians. In contrast with the previous choice, we proceed to make use of Jacobians in numerator format. The reasoning behind this choice will become clear in the later sections surrounding neural network training as having gradients in denominator layout and Jacobians in numerator layout keeps optimization procedures notationally clean.

Hessian

Other

Layout

There's a complication with defining derivative rules and examples for vector functions with multiple input variables in the subsequent sub sections: we have a choice.

The gradient is in what is referred to as denominator layout, with the dimensions of the resulting derivative being denominator (input variables) dimension x numerator (function output, in the typical case 1) dimension. The Jacobian is in what is referred to as numerator layout, with the dimensions of the resulting derivative being numerator (function output) dimension x denominator (input variables) dimension.

Why does this matter? Denominator layout and numerator layout are transposes of each other and the transpose affects the chain rule ordering and the resulting derivatives. So a choice needs to be made for consistency within this book.

As mentioned above, this book will use the denominator layout for all derivatives other than the Jacobian.

Why this choice? The primary use of vector function of multiple variable derivatives is for learning parameters in CNNs, and the key derivatives in this process are all gradients which use the denominator layout.

A reasonable question to ask would be why not define the Jacobian in denominator layout to match? The reason is is the Jacobian is special enough and enough theorems include it in numerator layout form that it feels like it would introduce more confusion to a casual reader to re define it from the default. But it could be and everything could be made to work out fine.

Rules

Examples

4.2 | Integrals

4.2.1 | Single Variable

Definite

The definite integral of a real valued function $f(x)$ over the interval $[a, b]$ is written as

$$\int_a^b f(x)dx \tag{4.13}$$

and has the intuitive interpretation of being the signed area of the graph of $f(x)$ over the interval from $[a, b]$.

Indefinite

An indefinite integral is a linear operator that maps a real valued function $f(x)$ to a differentiable function $F(x)$ whose derivative is equal to $f(x)$. Let f be a continuous real valued function defined on the closed interval $[a, b]$ and let F be a function defined for all x in [a, b] by

$$F(x) = \int_a^x f(t)dt \tag{4.14}$$

Fundamental Theorem Of Calculus

Then by the fundamental theorem of calculus, F is uniformly continuous on $[a, b]$, differentiable on (a, b) and

$$\frac{dF(x)}{dx} = f(x) \tag{4.15}$$

for all x in $[a, b]$. A corollary to the fundamental theorem allows for definite integrals to be computed in terms of indefinite integrals

$$F(b) - F(a) = \int_a^b f(t)dt \tag{4.16}$$

4.2.2 | Multiple Variables

4.3 | Approximation

The Taylor series of a real or complex function $f(x)$ that is infinitely differentiable at a is

$$\sum_{n=0}^{\infty} \frac{d^n f(a)}{dx^n} \frac{(x-a)^n}{n!} \tag{4.17}$$

4.4 | Optimization

4.4.1 | Closed Form Solutions

4.4.2 | Iterative Methods

Chapter 5

Probability

5.1 | Probability Spaces

5.1.1 | Probability Spaces

A probability space consists of a sample space S of all possible outcomes, an event space E where each event is a set of 0 or more outcomes, and probability measure function $P : E \to [0, 1]$ that maps events to probabilities and satisfies the following axioms

1. $P(A) \geq 0$ for all events $A \in E$
2. $P(S) = 1$
3. $P(\cup_{i=0}^{\infty} A_i) = \sum_{i=0}^{\infty} P(A_i)$ for mutually exclusive events A_i

5.1.2 | Events

Event probabilities will be be considered from the perspective of 1 event A, 2 events A and B or N events $\{A_0, \ldots, A_{N-1}\}$ all in the same event space. This is motivated by definitions, clarity and subsequent use.

Single

The probability of event A occuring is $P(A) \in [0, 1]$. The probability of event A' denoting not A occuring is $P(A') = 1 - P(A)$.

Joint

The probability of events A and B occurring, $P(A \cap B)$, is referred to as the joint probability of A and B and is also written as $P(A, B)$. If $A \subseteq B$, then $P(A, B) = P(A)$. If the events A and B are independent, then the occurance of 1 event does not affect the probability of the other and $P(A, B) = P(A)P(B)$.

Joint probability can be extended to N events as $P(A_0, \ldots, A_{N-1})$ with the interpretation of being the probability of all N events occurring. As in the case of 2 events, if events $\{A_0, \ldots, A_{N-1}\}$ are independent then the occurance of 1 event does not affect the probability of another event and $P(A_0, \ldots, A_{N-1}) = P(A_0) \cdots P(A_{N-1})$.

Union

The probability of event A or B occuring is $P(A \cup B) = P(A) + P(B) - P(A, B)$ where the subtraction can be thought of as preventing the intersection of outcomes from being double counted. If events A and B are mutually exclusive, then there are no outcomes occuring in both A and B and the sets are disjoint, so $P(A, B) = 0$ and $P(A \cup B) = P(A) + P(B)$.

The probability of the union of N events occuring is $P(A_0 \cup \cdots \cup A_{N-1})$ is straightforward to understand but a little messy to write in the general form as all multiply counted intersections need to be removed. However, when the N events are mutually exclusive, this simplifies considerably and $P(A_0 \cup \cdots \cup A_{N-1}) = P(A_0) + \cdots + P(A_{N-1})$.

Conditional

The probability of event A occuring given event B occured

$$P(A|B) = \frac{P(A,B)}{P(B)} \tag{5.1}$$

is referred to as the conditional probability of A given B. Multiplying this out and taking advantage of the joint probability being order independent results in $P(A,B) = P(A|B)P(B) = P(B|A)P(A)$.

The last 2 equations can be rearranged to form Bayes' Theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{5.2}$$

In this equation, $P(A)$ is the prior belief in A, $P(A|B)$ is the posterior belief in A having accounted for B and $P(B|A)/P(B)$ is the support that B provides for A. If events A and B are independent then event B doesn't provide support for event A and $P(A|B) = P(A)$.

The chain rule of probability relates joint probabilities of N random variables to conditional probabilities

$$P(A_0, \ldots, A_{N-1}) = P(A_0|A_1, \ldots, A_{N-1})P(A_1, \ldots, A_{N-1}) \tag{5.3}$$

The chain rule can be recursively applied to the 2nd term on the right hand size of the equation. If all N events are independent, then $P(A_0, \ldots, A_{N-1}) = P(A_0) \cdots P(A_{N-1})$.

Law Of Total Probability

Let $\{B_1, \ldots, B_n\}$ be a set of disjoint events whose union is the full event space and the event space is the same as A. The law of total probability states that

$$P(A) = \sum_n P(A, B_n) = \sum_n P(A|B_b)P(B_n) \tag{5.4}$$

$P(A)$, the occurance of a single event, is also referred to as the marginal probability of A as it is not conditioned on other events.

5.1.3 | Combinatorics

5.2 | Random Variables

A random variable X is a function that maps events s from the sample space S to numbers $x \in \mathbb{R}$

$$X(s) = x \tag{5.5}$$

where x is referred to as a realization of X. Note that a random variable is neither random or a variable.

5.2.1 | Discrete

If the distribution of the range of X is finite or countably infinite, the random variable is discrete and described by a probability mass function $p_X(x_i)$ that assigns a probability to every value in the range

$$p_X(x_i) = P(X(s) = x_i) \tag{5.6}$$

with $s \in S$, $x_i \in R$ and $\sum_i p_X(x_i) = 1$. Frequently, (s) is dropped from the notation but it's included here to make the connection that a random variable is a function explicit and to help keep track of arguments when random processes are discussed in a subsequent section. A cumulative distribution function $F_X(x_i)$ is defined for a discrete random variable as the probability that the random variable X will be less than x

$$F_X(x_i) = P(X(s) \leq x_i) \tag{5.7}$$

and can be found via $\sum_{x_j \leq x_i} p_X(x_j)$.

Joint, conditional and marginal cumulative distribution functions and probability mass functions can be defined according to the earlier definitions. For 2 random variables X and Y

$$p_{X,Y}(x_i, y_j) = p_{X|Y}(x_i|y_j)p_Y(y_j) = p_{Y|X}(y_j|x_i)p_X(x_i) \tag{5.8}$$

relates the joint and conditional probability mass functions and

$$p_X(x_i) = \sum_j p_{X,Y}(x_i, y_j) = \sum_j p_{X|Y}(x_i|y_j)p_Y(y_j) \tag{5.9}$$

is the marginal probability mass function. If X and Y are independent then $p_{X,Y}(x_i, y_j) = p_X(x_i)p_Y(y_j)$ and $p_{X|Y}(x_i|y_j) = p_X(x_i)$.

For N random variables $X = \{X_0, \ldots, X_{N-1}\}$ with $X_{a:b}$ used to refer to elements from a to b the joint and conditional probability mass functions are related by

$$p_X(x_{i_0}, \ldots, x_{i_{N-1}}) = p_{X_0|X_{1:N-1}}(x_{i_0}|x_{i_1}, \ldots, x_{i_{N-1}})p_{X_{1:N-1}}(x_{i_1}, \ldots, x_{i_{N-1}}) \tag{5.10}$$

If $\{X_0, \ldots, X_{N-1}\}$ are independent then $p_X(x_{i_0}, \ldots, x_{i_{N-1}}) = p_{X_0}(x_{i_0}) \cdots p_{X_{N-1}}(x_{i_{N-1}})$.

5.2.2 | Continuous

If the distribution of the range is uncountably infinite, then the random variable is continuous and described by a cumulative distribution function that is identical in definition to the discrete case. If the cumulative distribution function is absolutely continuous, then the continuous random variable has a probability density function $p_X(x)$

$$\int_a^b p_X(x)dx = P(a \leq X(s) \leq b) \tag{5.11}$$

with $\int_{-\infty}^{\infty} p_X(x)dx = 1$. In this case, the cumulative distribution function and probability density function can be written in terms of each other as

$$F_X(x) = \int_{-\infty}^{x} p_X(t)dt \tag{5.12}$$

Note that many books will use a letter other than p for the probability density function to emphasize that it's not the same as the probability mass function, but this text will use the same letter.

For 2 random variables X and Y with defined probability density functions

$$p_{X,Y}(x,y) = p_{X|Y}(x|y)p_Y(y) = p_{Y|X}(y|x)p_X(x) \tag{5.13}$$

relates the joint and conditional probability density functions and

$$p_X(x) = \int_{-\infty}^{\infty} p_{X,Y}(x,y)dy = \int_{-\infty}^{\infty} p_{X|Y}(x|y)p_Y(y)dy \tag{5.14}$$

is the marginal probability density function. If X and Y are independent then $p_{X,Y}(x,y) = p_X(x)p_Y(y)$ and $p_{X|Y}(x|y) = p_X(x)$.

For N random variables $X = \{X_0, \ldots, X_{N-1}\}$ with $X_{a:b}$ used to refer to elements

from a to b the joint and conditional probability density functions are related by

$$p_X(x_0, \ldots, x_{N-1}) = p_{X_0|X_{1:N-1}}(x_0|x_1, \ldots, x_{N-1}) p_{X_{1:N-1}}(x_1, \ldots, x_{N-1}) \tag{5.15}$$

If $\{X_0, \ldots, X_{N-1}\}$ are independent then $p_X(x_0, \ldots, x_{N-1}) = p_{X_0}(x_0) \cdots p_{X_{N-1}}(x_{N-1})$.

5.2.3 | Expected Value

Operator

The expected value operator E is a linear operator that maps random variables to a probability weighted average of all events. To simplify the presentation, the change of variable formula will be included in the expectation formula and common expectations will become different choices for the function f.

For a discrete random variable X the expected value of $f(X(s))$ is

$$E[f(X(s))] = \sum_i f(x_i) p_X(x_i) \tag{5.16}$$

Likewise, for a continuous random variables X with a probability density function the expected value of $f(X(s))$ is

$$E[f(X(s))] = \int_{-\infty}^{\infty} f(x) p_X(x) dx \tag{5.17}$$

Scalar

$E[X(s)] = \mu_X$ is the mean of X. Other common expectations are the nth order moments about the mean $E[(X(s) - \mu_X)^n]$. If $n = 2$ then $E[(X(s) - \mu_X)^2] = \sigma_X^2$ is the variance of X and σ_X is the standard deviation of X.

The expected value operator can be applied to functions of different random variables. For random variables X and Y, the covariance is $cov(X, Y) = E[(X(s) -$

$\mu_X)(Y(s) - \mu_Y)] = E[X(s)Y(s)] - E[X(s)]E[Y(s)] = E[X(s)Y(s)] - \mu_X\mu_Y$ with units of X times Y. For cases where a unit less quantity is required, the correlation is divided by the product of the means to form the correlation coefficient $corr(X, Y) = cov(X, Y)/(\mu_X\mu_Y)$. The correlation coefficient takes on values from [-1, 1], where -1 indicates a perfect inverse relationship and +1 indicates a perfect linear relationship. If X and Y are independent, then $E[X(s)Y(s)] = E[X(s)]E[Y(s)] = \mu_X\mu_Y$ and both $cov(X, Y)$ and $corr(X, Y)$ are 0. Note that 0 covariance or correlation does not imply independence of X and Y.

Vector And Matrix

The expected value operator applied to a vector or a matrix of random variables is simlpy the operator applied to all individual elements of the vector or matrix.

Let $\mathbf{x} = [X_0(s), \ldots, X_{N-1}(s)]^T$ be a vector of N random variables defined on the same probability space. Then the mean vector is

$$E[\mathbf{x}] = \begin{bmatrix} E[X_0(s)] \\ \vdots \\ E[X_{N-1}(s)] \end{bmatrix} \tag{5.18}$$

$$= \boldsymbol{\mu}_{\mathbf{x}} \tag{5.19}$$

and the covariance matrix is

$$E[(\mathbf{x} - \boldsymbol{\mu}_{\mathbf{x}})(\mathbf{x} - \boldsymbol{\mu}_{\mathbf{x}})^T] = \begin{bmatrix} E[(X_0(s) - \mu_{X_0})(X_0(s) - \mu_{X_0})] & \cdots & E[(X_0(s) - \mu_{X_0})(X_{N-1}(s) - \\ \vdots & & \vdots \\ E[(X_{N-1}(s) - \mu_{X_{N-1}})(X_0(s) - \mu_{X_0})] & \cdots & E[(X_{N-1}(s) - \mu_{X_{N-1}})(X_{N-1}(s \end{bmatrix}$$

$$= \boldsymbol{\Sigma}_{\mathbf{x},\mathbf{x}}$$

Law Of Large Numbers

The law of large numbers allows the expected value of a random variable with fininte mean to be estimated from it's sample average.

Let $X_0(s), X_1(s), \ldots$ be an infinite sequence of independent identically distributed random variables with $E[X_i(s)] = \mu_X$ and let the sample average be

$$\overline{X}_{N-1}(s) = \frac{X_0(s) + \cdots + X_{N-1}(s)}{N} \tag{5.22}$$

The law of large numbers states that $\overline{X}_{N-1}(s)$ converges to μ_X as $N \to \infty$ in probability for the weak law and almost surely for the strong law. Note that the sample average is a random variable and the law of large numbers states how close a relaization of the sample average is to the true mean.

Additional variants of the law of large numbers exist, including a version where the independence constraint of the weak law is removed and replaced by a constraint on the variance of $X_i(s)$.

Examples

5.2.4 | Sums

Convolution

Central Limit Theorem

DFT

Matrix Multiplication

5.3 | Random Processes

A random process $X(s,t)$ maps events s from the sample space to functions $x(t)$ where the domain of the function is referred to as the index set and the range of the function is referred to as the state space. In this setting t will be referred to as time as it's frequently used to represent discrete or continuous time, though this is not a requirement. Note that

- $X(s,t)$ is a random variable at a fixed t; by considering all times t this leads to the observation that a random process can be considered a collection of random variables $\{X(s,t)\}$
- $X(s,t)$ is a deterministic function of t for a fixed s; this is referred to as a realization of the random process and the set of all possible functions is referred to as the ensemble
- $X(s,t)$ is a number for a fixed s at a fixed t

If t refers to multiple dimensions such as the width and height of an image, $X(s,t)$ is referred to as a random field. Conditional random fields are frequently used in computer vision applications.

5.3.1 | Expected Value

The expected value of a random process is found by viewing the random process as a random variable at a fixed t and applying the expected value operator as before. Conceptually, it operates across many realizations s of a random process at a single t. The mean, variance, higher order moments about the mean and other expectations are then defined as in the case of a random variable.

For a discrete random process at time t, let $p_X(x_i, t) = P(X(s,t) = x_i)$ be the probability that $X(s,t) = x_i$. The expected value of $X(s,t)$ is

$$E[f(X(s,t))] = \sum_i f(x_i) p_X(x_i, t) \tag{5.23}$$

which in general is a function of t.

For a continuous random process at time t, let $\int_a^b p_X(x,t)dx = P(a \leq X(s,t) \leq b)$ be the probability that $a \leq X(s,t) \leq b$. The expected value of $X(s,t)$ is

$$E[f(X(s,t))] = \int_{-\infty}^{\infty} f(x) p_X(x,t) dx \tag{5.24}$$

which in general is also a function of t.

5.3.2 | Time Average

The time average of a random process is found by viewing the random process as a deterministic function for a fixed s and applying the time average operator. Conceptually, it operates across 1 realization s of a random process at many points in time t. Different time averages are defined similar to that of the expected value of a random variable.

5.3.3 | Stationarity

In general, the pdf and pmf of a random process are arbitrary functions of time and the random variable is considered non stationary. However, for many practical cases of interest random processes exhibit statistics that are not arbitrary functions of time.

Cyclo Stationary

Strictly Stationarity

Wide Sense Stationarity

5.3.4 | Markov

5.4 | Information Theory

5.4.1 | Entropy

5.4.2 | Mutual Information

5.4.3 | Divergence

5.4.4 | Compression

Depending on memory and system configurations, CNNs frequently need to move around large amounts of data. Data communication can easily take more power and more time than computation, so it's important to be be efficient when moving data. Compression is a technique used to reduce the amount of data that needs to be moved.

Compression can be broadly broken into 2 categories: lossless and lossy. As their names imply, the original input can be recovered after lossless compression / de compression and the original input can't be recovered after lossy compression / de compression. The extra degree of freedom that loss provides typically allows lossy

compression methods to achieve much higher compression ratios than lossless compression methods. However, there are many occasions where losses are not tolerable and lossless methods must be used.

Lossless

Lossless compression is possible when the number of bits per symbol is > than the entropy per symbol. Entropy codes assign shorter code words to more likely events and longer code words to less likely events. 2 common types of entropy codes are Huffman codes and arithmetic codes.

Lossy

In addition to taking advantange of data > information, lossy compression takes advantage of the statistics and intended use of the data to allow errors (noise) to be introduced in the de compressed output. For example, the noise that JPEG indroduces into compressed / decompressed images and the noise that MP3 introduces into compressed / decompressed audio to minimize it's impact to the human perceptual system. Lossy compression methods tend to be data type dependent.

5.5 | Optimization

5.5.1 | Closed Form Solutions

5.5.2 | Iterative Methods

Part III

Theory

Chapter 6

Machine Learning

Placeholder

6.1 | Data Generation

6.1.1 | Natural Data

Collection

Labeling

Cleaning

6.1.2 | Synthetic Data

Hand Engineered

Learned

6.1.3 | Data Augmentation

Placeholder

6.2 | Information Extraction

6.2.1 | Pre Processing

6.2.2 | Feature Extraction

Hand Engineered

Learned

6.2.3 | Prediction

6.2.4 | Post Processing

Placeholder

6.3 | Learning

6.3.1 | Initialization

6.3.2 | Training

Supervised

Unsupervised

Semi Supervised

Reinforcement

6.3.3 | Evaluation

Placeholder

6.4 | Examples

6.4.1 | Vision

6.4.2 | Speech

Chapter 7

Convolutional Neural Networks

So the medicine has been taken (linear algebra, calculus and probability), the warm up stretches are over (machine learning) and now it's time to start of the main event: the convolutional neural network chapter. In all seriousness, depending on background and interest, it's not clear that this is the most important chapter in the book but it's probably in the top few.

The machine learning chapter broke the information extraction problem into pre processing, feature extraction, prediction and post processing. This chapter will look at the design of CNNs for feature extraction and prediction. Layers that make up CNNs will be described in terms of how space, channel and time are mapped from input to output and a number of example network configurations will be shown in a historical-ish order.

Likewise, the machine learning chapter considered a number of different methods for learning. This chapter will focus on supervised learning using initialization, forward propagation, loss, back propagation and weigh update via a learning schedule. Various methods for regularization to improve generalization and parallel training to reduce training time will be described.

Departing from the 1 to 1 mapping with the machine learning chapter, the per-

formance (speed not accuracy) of CNNs will be considered from the perspective of complexity and various methods for reducing complexity (quantization, compression and simplification). This will be used with a model for computation in the implementation chapter to optimize hardware and software for running CNNs.

Finally, the CNN chapter returns to paralleling the machine learning chapter with the vision and speech examples repeated using a CNN for feature extraction and prediction. Spoiler alert: accuracy will improve. Spoiler: it doesn't really matter for this sort of toy problem, but it will when more complex problems are considered in the application chapters.

7.1 | Information Extraction

Placeholder

7.1.1 | Network Design Strategy

Pre Processing

Feature Extraction

Prediction

Post Processing

Placeholder

7.1.2 | Common Layers

Channel And Space

Channel

Space

Element

Time

Rearrangement

Training

Placeholder

7.1.3 | Example Networks

Linear

Parallel

Dense

Residual

ML Optimized Designs

Placeholder

7.2 | Learning

7.2.1 | Initialization

7.2.2 | Training

Data

Forward Propagation

Loss

Backward Propagation

Parameter Update

Placeholder

7.3 | Performance

Performance in this book refers to the time required to run a specified network on a specified piece of hardware (the word accuracy will be used to describe how well the network predicts outputs).

It's pointless to talk about performance without a model for hardware (and software that can practically realize the hardware capability). As such, this section will be more about general bookkeeping items that will serve as inputs for optimizing hardware and software designs for performance.

7.3.1 | Complexity

7.3.2 | Quantization

Data Formats

Parameters

Feature Maps

Without Training

With Training

7.3.3 | Compression

Feature Maps

Filter Coefficients

Gradients

7.3.4 | Simplification

Placeholder

7.4 | Approximation

Placeholder

7.5 | Examples

7.5.1 | Vision

7.5.2 | Speech

Part IV

Implementation

Chapter 8

Hardware

If you're a hardware designer reading this a warning: you're going to be mad. Why? You're likely a smart person and want to build hardware that's also smart. However, for the big compute portion of CNNs (and many other problems), this is exactly the wrong thing to do. What's needed is smart software that does (effectively) offline optimization and dumb as dirt hardware that does exactly what the software says.

Here's another reason you're going to be mad as a hardware designer reading this chapter: you probably want to think about your hardware running applications and some of the interesting algorithm work making it's way into your hardware. As such, you're going to be disappointed when you learn that all we want from you are a bunch of core primitives (multiplying matrices, sorting vectors, ...) that you probably learned in high school.

So what's the good news? We're going to look at the future (present?) of big compute so you understand why hardware smarts (caching, branch prediction, ...) is not needed (answer = optimality) and how by building low level computational primitives the hardware you create has the potential for immortality (or at least multiple years of near optimal use) vs needing to be changed with the daily whims of an algorithm designer. So your life became easier and the things you create become

more meaningful. On balance, that feels like a reasonable tradeoff.

If it wasn't already apparent, this is an opinionated hardware chapter. The future will see a bifurcation in hardware:

- Host like general hardware that does everything ok and adapts ok to dynamic requirements via hardware decision making; lots of work goes into making ok as good as possible but realistically it never rises much above ok
- Big compute specific hardware that does certain things orders of magnitude better than general hardware and contains almost 0 hardware decision making, all optimization is performed ahead of time by software and the hardware along for the ride

The question is then how to make this apply to as many things as possible vs become a point solution that's out of date the day it comes out of the fab? The answer to this is to put low level primitives into gates vs high level algorithms. So this section could describe lots of different existing architectures but it won't. Instead it will focus on what they will all evolve to.

Placeholder

8.1 | Device

Placeholder

8.2 | Domain Specific Architecture

8.2.1 | Control

8.2.2 | Memory

8.2.3 | Communication

8.2.4 | Computation

Placeholder

8.3 | Configurations

8.3.1 | Node

8.3.2 | Network

Chapter 9

Software

Placeholder

9.1 | Application

9.1.1 | Specification

9.1.2 | Optimization

Placeholder

9.2 | Runtime

9.2.1 | Context Creation

9.2.2 | Compilation

9.2.3 | Graphs

Bootstrap

Initialization

Execution

9.2.4 | Accelerator Instructions

DMA

Matrix

Sort

9.2.5 | Modification Tables

Placeholder

9.3 | Performance

Part V

Application

Chapter 10

Vision

Placeholder

10.1 | Data Generation

Placeholder

10.2 | Pre Processing

Placeholder

10.3 | Feature Extraction

Placeholder

10.4 | Prediction

10.4.1 | Classification

10.4.2 | Detection

10.4.3 | Segmentation

10.4.4 | Depth

10.4.5 | Motion

Placeholder

10.5 | Post Processing

Chapter 11

Speech

Placeholder

11.1 | Data Generation

Placeholder

11.2 | Pre Processing

Placeholder

11.3 | Feature Extraction

Placeholder

11.4 | Prediction

11.4.1 | Keyword Spotting

11.4.2 | Recognition

Placeholder

11.5 | Post Processing

Chapter 12

Language

Placeholder

12.1 | Data Generation

Placeholder

12.2 | Pre Processing

Placeholder

12.3 | Feature Extraction

Placeholder

12.4 | Prediction

12.4.1 | Sentiment Analysis

12.4.2 | Document Summarization

12.4.3 | Question Answering

12.4.4 | Translation

Placeholder

12.5 | Post Processing

Chapter 13

Games

Placeholder

13.1 | Data Generation

Placeholder

13.2 | Board Games

Placeholder

13.3 | Video Games

Placeholder

13.4 | Puzzles

Chapter 14

Other

Hey, there should be a lot more chapters in this part of the book with descriptive names like radar, controls, EEG, genomics, ... So why does the application part end with an 'Other' chapter?

Simple mundane life answer: we ran out of time for this version of the book. For some background, we put the book together in a relatively short period of time to support the teaching of a class and decided to focus on arguably the most important applications which are also the application we know best. It's not to say that other applications aren't important. They are. We'll just gradually split them out of this chapter and into their own stand alone chapters with expanded coverage in subsequent revisions of the book when and if they're ready.

So if this was 1998 here would be a great place to include some under construction text with an anomated GIF showing a construction person shoveling.

But it's not, so we've decided to organize this chapter slightly different than the previous chapters. Previous chapters considered applications in the context of a complete data generation or information extraction flow. Here we'll organize around a laundry list of different signal types and provide some information and references on how each can be used with CNNs. This will naturally allow us to individually

split them off into their own chapter as our treatment of them expands.

So thankfully, this is 2018 and this book is online, so revisions can happen at any time and are relatively easy from a logistical point of view.

14.1 | Laundry List

Part VI

Conclusion

Chapter 15

Summary

Thank you for reading the whole book linearlly from start to almost finish and making your way here! Just kidding - we know you didn't - and that's cool.

Time sometimes gives perspective and when we look back on our classes not every detail is remembered, but hopefully the key items stand out. Sort of like when you take a math class, at the end you should probably remember the fundamental theorem associated with that class and a little more, but it's fine that you don't remember every single theorem. There's the internet for that.

View this chapter as a fundamental theorem plus for this book that hopefully won't take too much time to read and will also serve as a later reference. As a local advertisement, the next chapter in the conclusion part is more interesting and you definately want to read that.

15.1 | Theory

15.2 | Implementation

15.3 | Application

Chapter 16

Next

16.1 | Interpolation

16.2 | Extrapolation

Part VII

Appendices

Appendix A

Notation

A.1 | Conventions

A.2 | Symbols

Appendix B

Practice

B.1 | Hardware

B.2 | Software

B.3 | Workflow

B.4 | Data

B.5 | Network Design

B.6 | Training

Appendix C

History

Version	Date	Comments
0.01	2018-08-13	This is the initial version of a book in an early draft stage; a logical organization for most chapters is shown, hidden are associated notes in each section

Table C.1: Revision history

Appendix D

References

Titles and links are provided, formal citations and associated links in text will be added later.

D.1 | Introduction

D.1.1 | General

D.1.2 | Motivation

D.1.3 | Plan

D.2 | Background

D.2.1 | General

D.2.2 | Linear Algebra

General

Notes

- The matrix cookbook

  https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf

- Linear algebra review and reference

  http://cs229.stanford.edu/section/cs229-linalg.pdf

- Linear algebra review

  http://people.csail.mit.edu/dsontag/courses/ml12/notes/linear_algebra.pdf

Websites

- Math insight

  https://mathinsight.org/thread/vector_algebra

- Mathonline

  http://mathonline.wikidot.com/linear-algebra

Books

- Linear algebra

  http://joshua.smcvt.edu/linearalgebra/book.pdf

- Introduction to applied linear algebra: vectors, matrices, and least squares

  https://web.stanford.edu/~boyd/vmls/vmls.pdf

- Linear algebra

  https://www.math.ucdavis.edu/~linear/linear-guest.pdf

Courses

- GA Tech Math 1553 Introduction to linear algebra

  http://people.math.gatech.edu/~jrabinoff6/1718F-1553/

- A first course in linear algebra

  http://linear.ups.edu/html/fcla.html

Specific

Affine transformations

- Affine transformations

  https://eli.thegreenplace.net/2018/affine-transformations/

CNN style 2D convolution

- A guide to convolution arithmetic for deep learning

  https://arxiv.org/abs/1603.07285
- High performance convolutional neural networks for document processing

  https://hal.inria.fr/inria-00112631/document

Strassen's algorithm

- Designing Strassen's algorithm

  https://arxiv.org/abs/1708.09398
- Minimizing computation in convolutional neural networks

  https://vast.cs.ucla.edu/sites/default/files/publications/CNN_ICANN14.pdf

D.2.3 | Calculus

General

Notes

- University of Singapore MA 1104 Multivariable calculus lecture notes

  http://www.math.nus.edu.sg/~matwyl/multivariable_calculus_notes.pdf
- National University of Singapore CS 5240 Matrix differentiation

  https://www.comp.nus.edu.sg/~cs5240/lecture/matrix-differentiation.pdf

- Caltech course notes

  http://www.math.caltech.edu/~dinakar/dr_notes.html

- Stanford CS 224n Review of differential calculus theory

  https://web.stanford.edu/class/cs224n/readings/review-differential-calculus.pdf

- Stanford CS 224n Computing neural network gradients

  https://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

- Stanford CS 231n Vector, matrix, and tensor derivatives

  http://cs231n.stanford.edu/vecDerivs.pdf

- Matrix calculus

  https://ccrma.stanford.edu/~dattorro/matrixcalc.pdf

Web sites

- Math insight

  https://mathinsight.org/thread/calculus_refresher

- Math insight

  https://mathinsight.org/thread/multivar

- Mathonline

  http://mathonline.wikidot.com/calculus

- Multivariable calculus

  https://people.math.gatech.edu/~cain/notes/calculus.html

- Wikipedia Matrix calculus

  https://en.wikipedia.org/wiki/Matrix_calculus

- Wikipedia Vector calculus identities

  https://en.wikipedia.org/wiki/Vector_calculus_identities

- Matrix reference manual

  http://www.psi.toronto.edu/matrix/calculus.html

- Vector and matrix differentiation

http://fourier.eng.hmc.edu/e176/lectures/algebra/node17.html

Courses

- Mechanics lecture notes part III: foundations of continuum mechanics
  http://homepages.engineering.auckland.ac.nz/~pkel015/SolidMechanicsBooks/Part_III/
- UC Boulder ASEN 5007 Introduction to finite element methods
  https://www.colorado.edu/engineering/CAS/courses.d/IFEM.d/
- UC San Diego ECE 275A Vector derivatives, gradients, and generalized gradient descent algorithms
  http://dsp.ucsd.edu/~kreutz/PEI-05%20Support%20Files/ECE275A_Viewgraphs_5.pdf

Specific

Automatic differentiation

- Automatic differentiation in machine learning: a survey
  https://arxiv.org/abs/1502.05767
- Wikipedia Automatic differentiation
  https://en.wikipedia.org/wiki/Automatic_differentiation

Chain rule

- Linear maps, the total derivative and the chain rule
  http://www.math.columbia.edu/department/lipshitz/teaching/Linearization.pdf
- The chain rule of calculus
  https://eli.thegreenplace.net/2016/the-chain-rule-of-calculus/

Taylor series

- Taylor's theorem in one and several variables

  https://www.rose-hulman.edu/~bryan/lottamath/mtaylor.pdf

- Wikipedia Taylor series

  https://en.wikipedia.org/wiki/Taylor_series

- Taylor series expansion

  http://fourier.eng.hmc.edu/e176/lectures/NM/node45.html

Optimization

- Convex optimization

  https://web.stanford.edu/~boyd/cvxbook/

  https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf

- Convex optimization overview

  http://cs229.stanford.edu/section/cs229-cvxopt.pdf

  http://cs229.stanford.edu/section/cs229-cvxopt2.pdf

D.2.4 | Probability

General

Notes

- Review of probability theory

  http://cs229.stanford.edu/section/cs229-prob.pdf

- Review: probability and statistics

  http://people.csail.mit.edu/dsontag/courses/ml12/notes/probx.pdf

Web sites

- Random

  http://www.randomservices.org/random/index.html

- StatLect

  https://www.statlect.com/fundamentals-of-probability/

- Introduction to probability, statistics and random processes

  https://www.probabilitycourse.com

Books

- An introduction to statistical signal processing

  https://ee.stanford.edu/~gray/sp.pdf

Courses

- Stanford EE 278 Introduction to statistical signal processing

  https://web.stanford.edu/class/archive/ee/ee278/ee278.1114/handouts.html

D.3 | Theory

D.3.1 | General

D.3.2 | Machine Learning

General

Books

- Python machine learning book

  https://github.com/rasbt/python-machine-learning-book

  https://github.com/dmitriydligach/PyMLSlides

Courses

- Stanford CS 229 Machine learning

  http://cs229.stanford.edu

- University of Oxford Machine learning

  https://www.cs.ox.ac.uk/people/nando.defreitas/machinelearning/

- Carnegie Mellon University Introduction to machine learning

  http://www.cs.cmu.edu/~10701/lectures.html

- Classnotes in machine learning

  https://www.ics.uci.edu/%7Ewelling/classnotes/classnotes.html

- NYU Machine learning and pattern recognition

  https://cilvr.nyu.edu/doku.php?id=mlpr:start#course_material

- MIT Introduction to machine learning

  http://people.csail.mit.edu/dsontag/courses/ml13/

Specific

Cognative science

- Computational cognitive modeling

  https://brendenlake.github.io/CCM-site/

- Advancing AI through cognitive science

  https://brendenlake.github.io/AAI-site/

D.3.3 | Convolutional Neural Networks

General

Notes

- IJCAI 2018 Tutorial deep learning for AI

  http://www.iro.umontreal.ca/~bengioy/talks/IJCAI2018-DLtutorial.html

- NIPS 2015 Tutorial deep learning

  http://www.iro.umontreal.ca/~bengioy/talks/DL-Tutorial-NIPS2015.pdf

Websites

- UFLDL Tutotual deep learning

  http://ufldl.stanford.edu/tutorial/

- VGG convolutional neural networks practical

  https://www.robots.ox.ac.uk/~vgg/practicals/cnn/

Books

- Deep learning

  https://www.deeplearningbook.org

- Neural networks and deep learning

  http://neuralnetworksanddeeplearning.com/index.html

- Linear algebra and learning from data

  http://math.mit.edu/~gs/learningfromdata/

Courses

- Stanford CS 231n Convolutional neural networks for visual recognition

  http://cs231n.github.io

- Stanford CS 231n Convolutional neural networks for visual recognition

  http://cs231n.stanford.edu

- Virginia Tech ECE 6504 Deep learning for perception

  https://computing.ece.vt.edu/~f15ece6504/

- NYU Deep learning

  https://cilvr.nyu.edu/doku.php?id=courses:deeplearning2017:start

- NYU Mathematics of deep learning

  https://joanbruna.github.io/MathsDL-spring18/

- NYU Deep generative models

  https://cs.nyu.edu/courses/spring18/CSCI-GA.3033-022/

Specific

Appraisal

- Deep learning: a critical appraisal

  https://arxiv.org/abs/1801.00631

Approximation

- Wikipedia Universal approximation theorem

  https://en.wikipedia.org/wiki/Universal_approximation_theorem
- A visual proof that neural networks can compute any function

  http://neuralnetworksanddeeplearning.com/chap4.html
- IFT 6084 Theoretical principles for deep learning

  http://mitliagkas.github.io/ift6085-dl-theory-class/
- Understanding deep learning requires rethinking generalization

  http://mitliagkas.github.io/ift6085/student_slides/IFT6085_Presentation_Rethinking.pdf
- Neural networks part 1: a simple proof of the universal approximation theorem

  https://blog.goodaudience.com/neural-networks-part-1-a-simple-proof-of-the-unive
- The universal approximation theorem for neural networks

  http://mcneela.github.io/machine_learning/2017/03/21/Universal-Approximation-The

  html

Biological connection of neurons

- Dendritic computation

  https://neurophysics.ucsd.edu/courses/physics_171/annurev.neuro.28.061604.135703.pdf

Generative models

- Ian Goodfellow presentations

  http://www.iangoodfellow.com/slides/

  http://www.iangoodfellow.com/slides/2018-06-22-gan_tutorial.pdf
- NIPS 2016 Tutorial generative adversarial networks (GANs)

  https://media.nips.cc/Conferences/2016/Slides/6202-Slides.pdf
- Stanford CS 231n Lecture 13 generative models

  http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Network visualization

- Netscope CNN analyzer

  https://dgschwend.github.io/netscope/quickstart.html

Training (need to categorize)

- Learning representations by back-propagating errors

  https://www.nature.com/articles/323533a0
- Efficient backprob

  http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf
- Gradient-based learning applied to document recognition

  http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
- Introduction to multi-layer perceptrons (feedforward neural networks)

  http://www.iro.umontreal.ca/~pift6266/H10/notes/mlp.html
- Flow graphs, chain rule and backpropagation: efficient computation of the gradient

  http://www.iro.umontreal.ca/~pift6266/H10/notes/gradient.html#flowgraph
- Large-scale machine learning with stochastic gradient descent

  http://leon.bottou.org/publications/pdf/compstat-2010.pdf
- Convolutional neural networks backpropagation: from intuition to derivation

  https://grzegorzgwardys.wordpress.com/2016/04/22/8/

- Back propagation in convolutional neural networks - intuition and code

  https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intui

- Backpropagation applied to handwritten zip code recognition

  http://yann.lecun.com/exdb/publis/pdf/lecun-89e.pdf

- Original approach for the localisation of objects in images

  https://ieeexplore.ieee.org/document/318027/

- Forward And backpropagation in convolutional neural network

  https://medium.com/@2017csm1006/forward-and-backpropagation-in-convolutional-neu

- The softmax function and its derivative

  https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/

- Understanding softmax and the negative log-likelihood

  https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log

Training (initialization)

- On the importance of initialization and momentum in deep learning

  http://www.cs.toronto.edu/~fritz/absps/momentum.pdf

Training (batch size)

- Don't decay the learning rate, increase the batch size

  https://arxiv.org/abs/1711.00489

- Revisiting small batch training for deep neural networks

  https://arxiv.org/abs/1804.07612

Training (back propagation)

- 3 blue 1 brown

  https://www.youtube.com/watch?v=aircAruvnKk

  https://www.youtube.com/watch?v=IHZwWFHWa-w&t=26s

  https://www.youtube.com/watch?v=Ilg3gGewQ5U

  https://www.youtube.com/watch?v=tIeHLnjs5U8

- Deep learning summer bootcamp

  http://courses.cs.tau.ac.il/Caffe_workshop/Bootcamp/pdf_lectures/Lecture%203%20CNN%20-%20backpropagation.pdf

Training (solvers)

- The marginal value of adaptive gradient methods in machine learning

  https://arxiv.org/pdf/1705.08292.pdf
- ADAM: a method for stochastic optimization

  https://arxiv.org/pdf/1412.6980.pdf
- On the convergence of ADAM and beyond

  https://openreview.net/pdf?id=ryQu7f-RZ
- Neural optimizer search with reinforcement learning

  https://arxiv.org/abs/1709.07417
- Deep learning summer bootcamp

  http://courses.cs.tau.ac.il/Caffe_workshop/Bootcamp/pdf_lectures/Lecture%204%20CNN%20-%20optimization.pdf
- Why momentum really works

  https://distill.pub/2017/momentum/
- On the importance of initialization and momentum in deep learning

  http://www.cs.toronto.edu/~fritz/absps/momentum.pdf

Training (parameter update schedule)

- Train longer, generalize better: closing the generalization gap in large batch training of neural networks

  https://arxiv.org/abs/1705.08741

Training (convergence)

- Batch normalization: accelerating deep network training by reducing internal

covariate shift

https://arxiv.org/pdf/1502.03167.pdf

- Batch renormalization

  http://papers.nips.cc/paper/6790-batch-renormalization-towards-reducing-minibatc

- Group normalization

  https://arxiv.org/abs/1803.08494

- When is a convolutional filter easy to learn?

  https://research.fb.com/wp-content/uploads/2018/04/when-is-a-convolutional-filte

  pdf?

Training (regularization)

- Regularization for deep learning: a taxonomy

  https://arxiv.org/abs/1710.10686

- Sensitivity and generalization in neural networks: an emperical study

  https://openreview.net/pdf?id=HJC2SzZCW

- Robust large margin deep neural networks

  https://arxiv.org/abs/1605.08254

- Dropout: a simple way to prevent neural networks from overfitting

  https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

Training (PyTorch)

- Automatic differentiation package - torch.autograd

  https://pytorch.org/docs/stable/autograd.html

- Autograd

  https://pytorch.org/tutorials/beginner/former_torchies/autograd_tutorial.

  html

- PyTorch - variables, functionals and autograd

  https://jhui.github.io/2018/02/09/PyTorch-Variables-functionals-and-Autograd/

- Automatic differentiation in PyTorch

  https://openreview.net/pdf?id=BJJsrmfCZ

Training (TensorFlow)

- Back propagation with TensorFlow

  http://blog.aloni.org/posts/backprop-with-tensorflow/

Performance (quantization)

- Quantization and training of neural networks for efficient integer-arithmetic-only inference

  https://arxiv.org/abs/1712.05877

- Estimating or propagating gradients through stochastic neurons for conditional computation

  https://arxiv.org/abs/1308.3432

- Caffe Ristretto

  Links

  http://lepsucd.com/?page_id=621

  http://lepsucd.com/?page_id=637

  Ristretto: a framework for empirical study of resource-efficient inference in convolutional neural networks

  https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8318896

  Hardware-oriented approximation of convolutional neural networks

  https://arxiv.org/abs/1604.03168

- Training deep neural networks with low precision multiplications

  https://arxiv.org/abs/1412.7024

- Quantized neural networks: training neural networks with low precision weights and activations

  https://arxiv.org/abs/1609.07061

- Mixed precision training

  https://arxiv.org/abs/1710.03740

- Quantization and training of neural networks for efficient integer-arithmetic-only inference

  https://arxiv.org/abs/1712.05877

- Mixed precision training of convolutional neural networks using integer operations

  https://arxiv.org/abs/1802.00930

- TensorFlow

  https://www.tensorflow.org/performance/quantization

  https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/quantize

  https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md

Performance (simplification)

- Optimal brain damage

  http://yann.lecun.com/exdb/publis/pdf/lecun-90b.pdf

- Optimal brain surgeon and general network pruning

  https://authors.library.caltech.edu/54981/1/Optimal%20Brain%20Surgeon%20and%20general%20network%20pruning.pdf

- Learning efficient convolutional networks through network slimming

  https://arxiv.org/abs/1708.06519

- Accelerating very deep convolutional networks for classification and detection

  https://arxiv.org/abs/1505.06798

- Efficient and accurate approximations of nonlinear convolutional networks

  https://arxiv.org/abs/1411.4229

- To prune or not to prune: exploring the efficacy of pruning for model compres-

sion

https://openreview.net/pdf?id=Sy1iIDkPM

D.4 | Implementation

D.4.1 | General

D.4.2 | Hardware

D.4.3 | Software

D.5 | Application

D.5.1 | General

D.5.2 | Vision

D.5.3 | Speech

D.5.4 | Language

D.5.5 | Games

D.5.6 | Other

D.6 | Conclusion

D.6.1 | General

D.6.2 | Summary

D.6.3 | Next

Appendix E

About

E.1 | Arthur

Arthur J. Redfern received a B.S. in 1995 from the University of Virginia and a M.S. and Ph.D. in 1996 and 1999, respectively, from the Georgia Institute of Technology, all in electrical engineering. Following his thesis work on nonlinear systems modeled by the Volterra series, Arthur joined Texas Instruments where he currently manages the Machine Learning Lab. His activities at TI have spanned the areas of machine learning (convolutional neural network based automotive and industrial applications, software libraries and hardware design), high performance computing (software libraries), signal processing for analog systems (ADCs, amplifiers, DACs, design optimization, speakers and touch screens) and communication system design (DSL, DTV and SerDes). He has over 20 papers published in refereed conferences and journals and has been granted over 20 US patents.

E.2 | Tarek

Tarek A. Lahlou received a B.S. in Electrical and Computer Engineering in 2011 from George Mason University and a S.M. and Ph.D. in Electrical Engineering and

Computer Science in 2013 and 2016, respectively, from the Massachusetts Institute of Technology. His thesis work blended semi-algebraic geometry with variational analysis for the purpose of understanding large-scale optimization in asynchronous and decentralized settings. Following his thesis work, Tarek joined Texas Instruments where he currently is a member of the Machine Learning Lab focused on the design, training and deployment of convolutional neural networks on TI embedded devices. He has over 10 papers published in refereed conferences and journals.