# Preface and Acknowledgments

My deepest gratitude goes to Professor Raj Rao Nadakuditi of the University of Michigan. On his sabbatical in 2017, Raj brought his EECS 551 course to MIT. He flew to Boston every week to teach 18.065. Thanks to Raj, the students could take a new course. He led the in-class computing, he assigned homeworks, and exams were outlawed.

This was linear algebra for signals and data, and it was alive. 140 MIT students signed up. Alan Edelman introduced the powerful language *Julia*, and I explained the four fundamental subspaces and the Singular Value Decomposition. The labs from Michigan involved rank and SVD and applications. We were asking the class for *computational thinking*.

That course worked, even the first time. It didn't touch one big topic : **Deep learning**. By this I mean the excitement of creating a learning function on a neural net, with the hidden layers and the nonlinear activation functions that make it so powerful. The system trains itself on data which has been correctly classified in advance. The optimization of weights discovers important features—the shape of a letter, the edges in an image, the syntax of a sentence, the identifying details of a signal. Those features get heavier weights—*without overfitting the data* and learning everything. Then unseen test data from a similar population can be identified by virtue of having those same features.

The algorithms to do all that are continually improving. Better if I say that they are *being improved*. This is the contribution of computer scientists and engineers and biologists and linguists and mathematicians and especially optimists—those who can optimize weights to minimize errors, and also those who believe that deep learning can help in our lives.

You can see why this book was written :

1. To organize central methods and ideas of **data science**.
2. To see how the language of **linear algebra** gives expression to those ideas.
3. Above all, to show how to **explain and teach** those ideas—to yourself or to a class.

I certainly learned that projects are far better than exams. Students ask their own questions and write their own programs. From now on, **projects !**

## Linear Algebra and Calculus

The reader will have met the two central subjects of undergraduate mathematics : Linear algebra and calculus. For deep learning, it is linear algebra that matters most. We compute "weights" that pick out the important features of the training data, and those weights go into matrices. The form of the learning function is described on page iv. Then calculus shows us the *direction to move*, in order to improve the current weights $\boldsymbol{x}_k$.

From calculus, it is partial derivatives that we need (and not integrals) :

**Reduce the error $L(x)$ by moving from $x_k$ to $x_{k+1} = x_k - s_k \nabla L$.**

That symbol $\boldsymbol{\nabla L}$ stands for the first derivatives of $L(\boldsymbol{x})$. Because of the minus sign, $\boldsymbol{x}_{k+1}$ *is downhill from* $\boldsymbol{x}_k$ on the graph of $L(\boldsymbol{x})$. The stepsize $\boldsymbol{s_k}$ (also called the learning rate) decides how far to move. You see the basic idea : Reduce the loss function $L(\boldsymbol{x})$ by moving in the direction of fastest decrease. $\boldsymbol{\nabla L = 0}$ at the best weights $\boldsymbol{x}^*$.

The complication is that the vector $\boldsymbol{x}$ represents thousands of weights. So we have to compute thousands of partial derivatives of $L$. And $L$ itself is a complicated function depending on several layers of $\boldsymbol{x}$'s as well as the data. So we need the chain rule to find $\boldsymbol{\nabla L}$.

The introduction to Chapter VI will recall essential facts of multivariable calculus.

By contrast, *linear algebra is everywhere in the world of learning from data*. This is the subject to know ! The first chapters of this book are essentially a course on applied linear algebra—the basic theory and its use in computations. I can try to outline how that approach (to the ideas we need) compares to earlier linear algebra courses. Those are quite different, which means that there are good things to learn.

**Basic course**

1. Elimination to solve $\boldsymbol{Ax = b}$
2. Matrix operations and inverses and determinants
3. Vector spaces and subspaces
4. Independence, dimension, rank of a matrix
5. Eigenvalues and eigenvectors

If a course is mostly learning definitions, that is not linear algebra in action. A stronger course puts the algebra to use. The definitions have a purpose, and so does the book.

**Stronger course**

1. $\boldsymbol{Ax = b}$ in all cases : square system—too many equations—too many unknowns.
2. Factor $A$ into $\boldsymbol{LU}$ and $\boldsymbol{QR}$ and $\boldsymbol{U\Sigma V^{\mathrm{T}}}$ and $\boldsymbol{CMR}$: *Columns times rows.*
3. *Four fundamental subspaces* : dimensions and orthogonality and good bases.
4. Diagonalizing $A$ by eigenvectors and by left and right singular vectors.
5. Applications : Graphs, convolutions, iterations, covariances, projections, filters, networks, images, matrices of data.

Linear algebra has moved to the center of machine learning, and we need to be there.

A book was needed for the 18.065 course. It was started in the original 2017 class, and a first version went out to the 2018 class. I happily acknowledge that this book owes its existence to Ashley C. Fernandes. Ashley receives pages scanned from Boston and sends back new sections from Mumbai, ready for more work. This is our seventh book together and I am extremely grateful.

Students were generous in helping with both classes, especially William Loucks and Claire Khodadad and Alex LeNail and Jack Strang. The project from Alex led to his online code **alexlenail.me/NN – SVG/** to draw neural nets (an example appears on page v). The project from Jack on **http://www.teachyourmachine.com** learns to recognize handwritten numbers and letters drawn by the user : open for experiment. See Section VII.2.

MIT's faculty and staff have given generous and much needed help :

Suvrit Sra gave a fantastic lecture on stochastic gradient descent (now an 18.065 video)

Alex Postnikov explained when matrix completion can lead to rank one (Section IV.8)

Tommy Poggio showed his class how deep learning generalizes to new data

Jonathan Harmon and Tom Mullaly and Liang Wang contributed to this book every day

Ideas arrived from all directions and gradually they filled this textbook.

## The Content of the Book

This book aims to explain the mathematics on which data science depends : *Linear algebra, optimization, probability and statistics.* The weights in the learning function go into matrices. Those weights are optimized by "stochastic gradient descent". That word stochastic (= random) is a signal that success is governed by probability not certainty. The law of large numbers extends to the law of large functions : If the architecture is well designed and the parameters are well computed, there is a high probability of success.

Please note that this is not a book about computing, or coding, or software. Many books do those parts well. One of our favorites is *Hands-On Machine Learning* (2017) by Aurélien Géron (published by O'Reilly). And online help, from Tensorflow and Keras and MathWorks and Caffe and many more, is an important contribution to data science.

Linear algebra has a wonderful variety of matrices : symmetric, orthogonal, triangular, banded, permutations and projections and circulants. In my experience, *positive definite symmetric matrices* $S$ are the aces. They have positive eigenvalues $\lambda$ and orthogonal eigenvectors $\boldsymbol{q}$. They are combinations $S = \lambda_1 \boldsymbol{q}_1 \boldsymbol{q}_1^{\mathrm{T}} + \lambda_2 \boldsymbol{q}_2 \boldsymbol{q}_2^{\mathrm{T}} + \cdots$ of simple rank-one projections $\boldsymbol{q}\boldsymbol{q}^{\mathrm{T}}$ onto those eigenvectors. And if $\lambda_1 \geq \lambda_2 \geq \ldots$ then $\lambda_1 \boldsymbol{q}_1 \boldsymbol{q}_1^{\mathrm{T}}$ is the most informative part of $S$. For a sample covariance matrix, that part has the greatest variance.

**Chapter I** In our lifetimes, the most important step has been to extend those ideas from symmetric matrices to all matrices. Now we need **two sets of singular vectors, $\boldsymbol{u}$'s and $\boldsymbol{v}$'s**. Singular values $\sigma$ replace eigenvalues $\lambda$. The decomposition $A = \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^{\mathrm{T}} + \sigma_2 \boldsymbol{u}_2 \boldsymbol{v}_2^{\mathrm{T}} + \cdots$ remains correct (this is the SVD). With decreasing $\sigma$'s, those rank-one pieces of $A$ still come in order of importance. That "Eckart-Young Theorem" about $A$ complements what we have long known about the symmetric matrix $A^{\mathrm{T}}A$: For rank $k$, stop at $\sigma_k \boldsymbol{u}_k \boldsymbol{v}_k^{\mathrm{T}}$.

**II** The ideas in Chapter I become algorithms in Chapter II. For quite large matrices, the $\sigma$'s and $\boldsymbol{u}$'s and $\boldsymbol{v}$'s are computable. For very large matrices, we resort to randomization: Sample the columns and the rows. For wide classes of big matrices this works well.

**III-IV** Chapter III focuses on low rank matrices, and Chapter IV on many important examples. We are looking for properties that make the computations especially fast (in III) or especially useful (in IV). The *Fourier matrix* is fundamental for every problem with constant coefficients (not changing with position). That discrete transform is superfast because of the **FFT:** the Fast Fourier Transform.

**V** Chapter V explains, as simply as possible, the statistics we need. The central ideas are always **mean and variance:** The *average* and the *spread* around that average. Usually we can reduce the mean to zero by a simple shift. Reducing the variance (the uncertainty) is the real problem. For random vectors and matrices and tensors, that problem becomes deeper. It is understood that the *linear algebra of statistics* is essential to machine learning.

**VI** Chapter VI presents two types of optimization problems. First come the nice problems of linear and quadratic programming and game theory. Duality and saddle points are key ideas. But the goals of deep learning and of this book are elsewhere: *Very large problems with a structure that is as simple as possible*. "Derivative equals zero" is still the fundamental equation. The second derivatives that Newton would have used are too numerous and too complicated to compute. Even using all the data (when we take a descent step to reduce the loss) is often impossible. That is why we choose only a minibatch of data, in each step of stochastic steepest descent.

The success of large scale learning comes from the wonderful fact that *randomization often produces reliability*—where there are thousands or millions of variables.

**VII** Chapter VII begins with the architecture of a neural net. An input layer is connected to hidden layers and finally to the output layer. For the training data, input vectors $\boldsymbol{v}$ are known. Also the correct outputs are known (often $\boldsymbol{w}$ is the correct classification of $\boldsymbol{v}$). **We optimize the weights $x$ in the learning function $\boldsymbol{F}$ so that $\boldsymbol{F}(x, v)$ is close to $w$ for almost every training input $v$.**

Then $\boldsymbol{F}$ is applied to *test data*, drawn from the same population as the training data. If $\boldsymbol{F}$ learned what it needs (*without overfitting*: we don't want to fit $100$ points by $99$th degree polynomials), the test error will also be low. The system recognizes images and speech. It translates between languages. It may follow designs like ImageNet or AlexNet, winners of major competitions. A neural net defeated the world champion at Go.

The function $F$ is often *piecewise linear*—the weights go into matrix multiplications. Every neuron on every hidden layer also has a nonlinear "activation function". The ramp function **ReLU**$(\boldsymbol{x})$ = (**maximum of 0 and** $\boldsymbol{x}$) is now the overwhelming favorite.

There is a growing world of expertise in designing the layers that make up $F(\boldsymbol{x}, \boldsymbol{v})$. We start with *fully connected* layers—all neurons on layer $n$ connected to all neurons on layer $n+1$. Often CNN's are better—*Convolutional neural nets* repeat the same weights around all pixels in an image: a very important construction. Other layers are different. A *pooling layer* reduces the dimension. *Dropout* randomly leaves out neurons. *Batch normalization* resets the mean and variance. All these steps create a function that closely matches the training data. Then $F(\boldsymbol{x}, \boldsymbol{v})$ is ready to use.

## Acknowledgments

Above all, I welcome this chance to thank so many generous and encouraging friends:

Pawan Kumar and Leonard Barrado and Mike Giles and Nick Trefethen in Oxford
Ding-Xuan Zhou and Yunwen Lei in Hong Kong
Alex Townsend and Heather Wilber at Cornell
Nati Srebro and Srinadh Bhojanapalli in Chicago
Tammy Kolda and Thomas Strohmer and Trevor Hastie and Jay Kuo in California
Bill Hager and Mark Embree and Wotao Yin, for help with Chapter III
Stephen Boyd and Lieven Vandenberghe, for great books
Alex Strang, for creating the best figures
Ben Recht in Berkeley, especially.

Your papers and emails and lectures and advice were wonderful.

# THE MATRIX ALPHABET

| | | | |
|---|---|---|---|
| $\boldsymbol{A}$ | Any Matrix | $\boldsymbol{Q}$ | Orthogonal Matrix |
| $\boldsymbol{C}$ | Circulant Matrix | $\boldsymbol{R}$ | Upper Triangular Matrix |
| $\boldsymbol{C}$ | Matrix of Columns | $\boldsymbol{R}$ | Matrix of Rows |
| $\boldsymbol{D}$ | Diagonal Matrix | $\boldsymbol{S}$ | Symmetric Matrix |
| $\boldsymbol{F}$ | Fourier Matrix | $\boldsymbol{S}$ | Sample Covariance Matrix |
| $\boldsymbol{I}$ | Identity Matrix | $\boldsymbol{T}$ | Tensor |
| $\boldsymbol{L}$ | Lower Triangular Matrix | $\boldsymbol{U}$ | Upper Triangular Matrix |
| $\boldsymbol{L}$ | Laplacian Matrix | $\boldsymbol{U}$ | Left Singular Vectors |
| $\boldsymbol{M}$ | Mixing Matrix | $\boldsymbol{V}$ | Right Singular Vectors |
| $\boldsymbol{M}$ | Markov Matrix | $\boldsymbol{X}$ | Eigenvector Matrix |
| $\boldsymbol{P}$ | Probability Matrix | $\boldsymbol{\Lambda}$ | Eigenvalue Matrix |
| $\boldsymbol{P}$ | Projection Matrix | $\boldsymbol{\Sigma}$ | Singular Value Matrix |

**Video lectures**: OpenCourseWare ocw.mit.edu and YouTube (**Math 18.06 and 18.065**)
**Introduction to Linear Algebra** (5th ed) by Gilbert Strang, Wellesley-Cambridge Press
**Book websites**: **math.mit.edu/linearalgebra** and **math.mit.edu/learningfromdata**