This text is expected to appear in January 2019
You could express interest by a note to learningfromdata1@gmail.com
This would not be an order – it would bring updated information
Thank you

LINEAR ALGEBRA AND LEARNING FROM DATA

GILBERT STRANG
Massachusetts Institute of Technology

WELLESLEY - CAMBRIDGE PRESS
Box 812060 Wellesley MA 02482

Linear Algebra and Learning from Data

ISBN 978-0-692-19638-0

LaTeX typesetting by Ashley C. Fernandes (info@problemsolvingpathway.com)

Printed in the United States of America 9 8 7 6 5 4 3 2 1

Other texts from Wellesley - Cambridge Press

Introduction to Linear Algebra, 5th Edition, Gilbert Strang ISBN 978-0-9802327-7-6
Computational Science and Engineering, Gilbert Strang ISBN 978-0-9614088-1-7
Wavelets and Filter Banks, Gilbert Strang and Truong Nguyen ISBN 978-0-9614088-7-9
Introduction to Applied Mathematics, Gilbert Strang ISBN 978-0-9614088-0-0
Calculus Third edition (2017), Gilbert Strang ISBN 978-0-9802327-5-2
Algorithms for Global Positioning, Kai Borre & Gilbert Strang ISBN 978-0-9802327-3-8
Essays in Linear Algebra, Gilbert Strang ISBN 978-0-9802327-6-9
Differential Equations and Linear Algebra, Gilbert Strang ISBN 978-0-9802327-9-0
An Analysis of the Finite Element Method, 2017 edition, Gilbert Strang and George Fix ISBN 978-0-9802327-8-3

Wellesley - Cambridge Press
Box 812060
Wellesley MA 02482 USA
www.wellesleycambridge.com

linearalgebrabook@gmail.com (orders)
math.mit.edu/∼gs
phone (781) 431-8488
fax (617) 253-4358

The website for this book is **math.mit.edu/learningfromdata**
That site will link to 18.065 course material and video lectures on YouTube and OCW

The cover photograph shows a neural net on Inle Lake. It was taken in Myanmar.
From that photograph Lois Sellers designed and created the cover.

Linear Algebra is included in MIT's OpenCourseWare site **ocw.mit.edu**
This provides video lectures of the full linear algebra course 18.06 and 18.06 SC

Deep Learning and Neural Nets

Linear algebra and probability/statistics and optimization are the mathematical pillars of machine learning. Those chapters will come before the architecture of a neural net. But we find it helpful to start with this description of the goal: *To construct a function that classifies the training data correctly, so it can generalize to unseen test data.*

To make that statement meaningful, you need to know more about this learning function. That is the purpose of these three pages—to give direction to all that follows.

The inputs to the function F are vectors or matrices or sometimes tensors—one input $\boldsymbol{v}$ for each training sample. For the problem of identifying handwritten digits, each input sample will be an image—a matrix of pixels. We aim to classify each of those images as a number from 0 to 9. Those ten numbers are the possible outputs from the learning function. In this example, the function F learns what to look for in classifying the images.

The MNIST set contains $70,000$ handwritten digits. We train a learning function on part of that set. By assigning weights to different pixels in the image, we create the function. The big problem of optimization (the heart of the calculation) is to choose weights so that the function assigns the correct output $0, 1, 2, 3, 4, 5, 6, 7, 8,$ or 9. And we don't ask for perfection! (One of the dangers in deep learning is *overfitting the data*.)

Then we validate the function by choosing unseen MNIST samples, and applying the function to classify this test data. Competitions over the years have led to major improvements in the test results. Convolutional nets now go below 1% errors. In fact it is competitions on known data like MNIST that have brought big improvements in the structure of F. That structure is based on the architecture of an underlying neural net.

Linear and Nonlinear Learning Functions

The inputs are the samples $\boldsymbol{v}$, the outputs are the computed classifications $\boldsymbol{w} = F(\boldsymbol{v})$. The simplest learning function would be linear: $\boldsymbol{w} = A\boldsymbol{v}$. The entries in the matrix A are the weights to be learned: not too difficult. Frequently the function also learns a *bias vector* $\boldsymbol{b}$, so that $F(\boldsymbol{v}) = A\boldsymbol{v} + \boldsymbol{b}$. This function is "*affine*". Affine functions can be quickly learned, but by themselves they are too simple.

More exactly, linearity is a very limiting requirement. If MNIST used Roman numerals, then II might be halfway between I and III (as linearity demands). But what would be halfway between I and XIX ? Certainly affine functions $A\boldsymbol{v} + \boldsymbol{b}$ are not always sufficient.

Nonlinearity would come by squaring the components of the input vector $\boldsymbol{v}$. That step might help to separate a circle from a point inside—which linear functions cannot do. But the construction of F moved toward "sigmoidal functions" with S-shaped graphs. It is remarkable that big progress came by inserting these standard nonlinear S-shaped functions between matrices A and B to produce $A(S(B\boldsymbol{v}))$. Eventually it was discovered that the smoothly curved logistic functions S could be replaced by the extremely simple ramp function now called $\mathbf{ReLU}(\boldsymbol{x}) = \mathbf{max}\,(\mathbf{0}, \boldsymbol{x})$. The graphs of these nonlinear "*activation functions*" R are drawn in Section VII.1.

Neural Nets and the Structure of $\boldsymbol{F(v)}$

The functions that yield deep learning have the form $F(\boldsymbol{v}) = L(R(L(R(\ldots(L\boldsymbol{v})))))$. This is a *composition* of affine functions $L\boldsymbol{v} = A\boldsymbol{v} + \boldsymbol{b}$ with nonlinear functions R—which act on each component of the vector $L\boldsymbol{v}$. The matrices A and the bias vectors $\boldsymbol{b}$ are the **weights in the learning function $\boldsymbol{F}$**. It is the A's and $\boldsymbol{b}$'s that must be learned from the training data, so that the outputs $F(\boldsymbol{v})$ will be (nearly) correct. Then F can be applied to new samples from the same population. If the weights (A's and $\boldsymbol{b}$'s) are well chosen, the outputs $F(\boldsymbol{v})$ from the unseen test data should be accurate. More layers in the function F will typically produce more accuracy in $F(\boldsymbol{v})$.

Properly speaking, $F(\boldsymbol{x}, \boldsymbol{v})$ depends on the input $\boldsymbol{v}$ and the weights $\boldsymbol{x}$ (all the A's and $\boldsymbol{b}$'s). The outputs $\boldsymbol{v_1} = \mathrm{ReLU}(A_1\boldsymbol{v} + \boldsymbol{b_1})$ from the first step produce the **first hidden layer in our neural net**. The complete net starts with the input layer $\boldsymbol{v}$ and ends with the output layer $\boldsymbol{w} = F(\boldsymbol{v})$. The affine part $L_k(\boldsymbol{v}_{k-1}) = A_k\boldsymbol{v}_{k-1} + \boldsymbol{b}_k$ of each step uses the computed weights A_k and $\boldsymbol{b_k}$.

All those weights together are chosen in the giant optimization of deep learning :

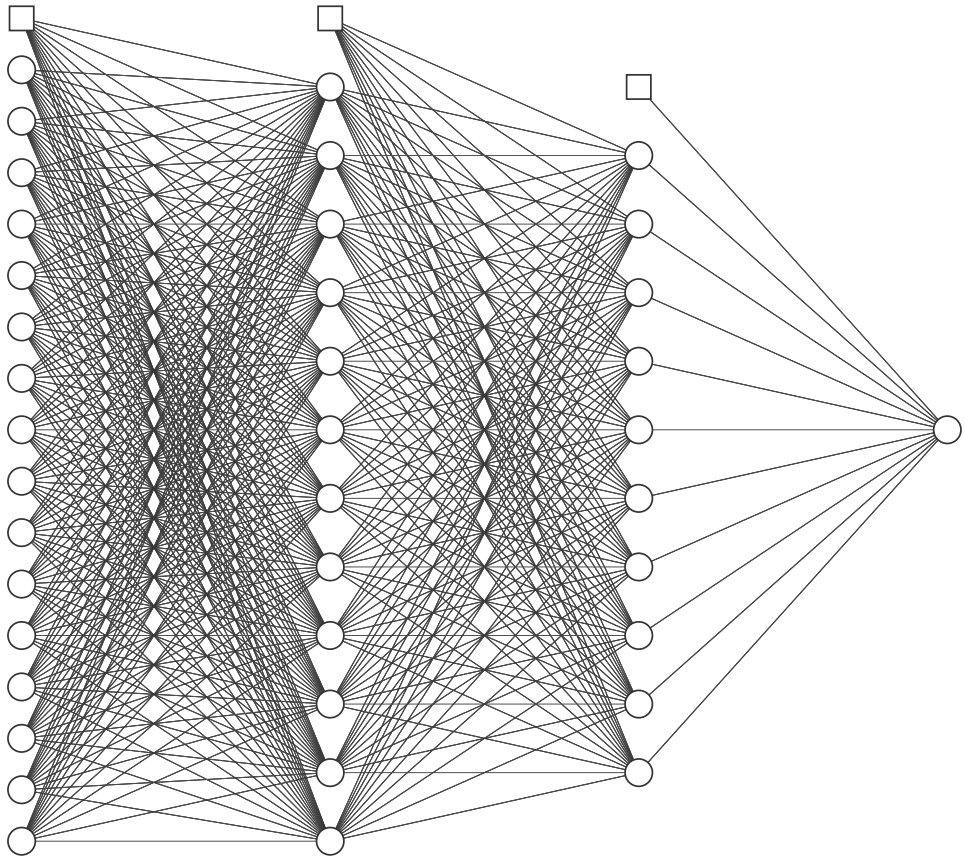
Choose weights $\boldsymbol{A_k}$ and $\boldsymbol{b_k}$ to minimize the total loss over all training samples.

The total loss is the sum of individual losses on each sample. The loss function for least squares has the familiar form $||F(\boldsymbol{v}) - \text{true output}||^2$. Often least squares is not the best loss function for deep learning.

One input $\boldsymbol{v} =$  **One output $\boldsymbol{w} = 2$**

Here is a picture of the neural net, to show the structure of $F(\boldsymbol{v})$. The input layer contains the training samples $\boldsymbol{v} = \boldsymbol{v}_0$. The output is their classification $\boldsymbol{w} = F(\boldsymbol{v})$. For perfect learning, $\boldsymbol{w}$ will be a (correct) digit from 0 to 9. **The hidden layers add depth to the network**. It is that depth which has allowed the composite function F to be so successful in deep learning. In fact the number of weights A_{ij} and b_j in the neural net is often larger than the number of inputs from the training samples $\boldsymbol{v}$.

This is a feed-forward fully connected network. For images, a *convolutional* neural net (**CNN**) is often appropriate and weights are shared—the diagonals of the matrices A are constant. Deep learning works amazingly well, when the architecture is right.



Input sample Hidden Layer Hidden Layer Output

Each diagonal in this neural net represents a weight to be learned by optimization. Edges from the squares contain bias vectors $\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_3$. The other weights are in A_1, A_2, A_3.