

DEEP LEARNING

JOHN D. KELLEHER

The MIT Press Essential Knowledge Series

A complete list of the titles in this series appears at the back of this book.

The MIT Press | Cambridge, Massachusetts | London, England

CONTENTS

Series Foreword	vii
Preface	ix
Acknowledgments	xi

© 2019 The Massachusetts Institute of Technology

All rights reserved. No part of this book may be reproduced in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher.

This book was set in Chaparral Pro by Toppan Best-set Premedia Limited.
Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Names: Kelleher, John D., 1974- author.

Title: Deep learning / John D. Kelleher.

Description: Cambridge, MA : The MIT Press, [2019] | Series:

The MIT press essential knowledge series | Includes bibliographical references and index.

Identifiers: LCCN 2018059550 | ISBN 9780262537551 (pbk. : alk. paper)

Subjects: LCSH: Machine learning. | Artificial intelligence.

Classification: LCC Q325.5 .K454 2019 | DDC 006.3/1—dc23 LC record available at <https://lcn.loc.gov/2018059550>

1098765

1	Introduction to Deep Learning	1
2	Conceptual Foundations	39
3	Neural Networks: The Building Blocks of Deep Learning	65
4	A Brief History of Deep Learning	101
5	Convolutional and Recurrent Neural Networks	159
6	Learning Functions	185
7	The Future of Deep Learning	231

Glossary	251
Notes	257
References	261
Further Readings	267
Index	269

the network using large datasets. By contrast, hyperparameters are the parameters of a model (in these cases, the parameters of a neural network architecture) and/or training algorithm that cannot be directly estimated from the data but instead must be specified by the person creating the model, either through the use of heuristic rules, intuition, or trial and error. Often, much of the effort that goes into the creation of a deep learning network involves experimental work to answer the questions in relation to hyperparameters, and this process is known as hyperparameter tuning. The next chapter will review the history and evolution of deep learning, and the challenges posed by many of these questions are themes running through the review. Subsequent chapters in the book will explore how answering these questions in different ways can create networks with very different characteristics, each suited to different types of tasks. For example, recurrent neural networks are best suited to processing sequential/time-series data, whereas convolutional neural networks were originally developed to process images. Both of these network types are, however, built using the same fundamental processing unit, the artificial neuron; the differences in the behavior and abilities of these networks stems from how these neurons are arranged and composed.

A BRIEF HISTORY OF DEEP LEARNING

The history of deep learning can be described as three major periods of excitement and innovation, interspersed with periods of disillusionment. Figure 4.1 shows a timeline of this history, which highlights these periods of major research: on threshold logic units (early 1940s to the mid 1960s), connectionism (early 1980s to mid-1990s), and deep learning (mid 2000s to the present). Figure 4.1 distinguishes some of the primary characteristics of the networks developed in each of these three periods. The changes in these network characteristics highlight some of the major themes within the evolution of deep learning, including: the shift from binary to continuous values; the move from threshold activation functions, to logistic and tanh activation, and then onto ReLU activation; and the progressive deepening of the networks, from single layer,

without using layer-wise pretraining. In the mid-2000s, researchers began to appreciate that the vanishing gradient problem was not a strict theoretical limit, but was instead a practical obstacle that could be overcome. The vanishing gradient problem does not cause the error gradients to disappear entirely; there are still gradients being backpropagated through the early layers of the network, it is just that they are very small. Today, there are a number of factors that have been identified as important in successfully training a deep network.

Weight Initialization and ReLU Activation Functions

One factor that is important in successfully training a deep network is how the network weights are initialized. The principles controlling how weight initialization affects the training of a network are still not clear. There are, however, weight initialization procedures that have been empirically shown to help with training a deep network. Glorot initialization¹⁰ is a frequently used weight initialization procedure for deep networks. It is based on a number of assumptions but has empirical success to support its use. To get an intuitive understanding of Glorot initialization, consider the fact that there is typically a relationship between the magnitude of values in a set and the variance of the set: generally the larger the values in a set, the larger the variance of the set. So, if the variance calculated on a set of gradients propagated through a layer

In the mid-2000s,
researchers began to
appreciate that the
vanishing gradient
problem was not a strict
theoretical limit, but
was instead a practical
obstacle that could be
overcome.

at one point in the network is similar to the variance for the set of gradients propagated through another layer in a network, it is likely that the magnitude of the gradients propagated through both of these layers will also be similar. Furthermore, the variance of gradients in a layer can be related to the variance of the weights in the layer, so a potential strategy to maintain gradients flowing through a network is to ensure similar variances across each of the layer in a network. Glorot initialization is designed to initialize the weight in a network in such a way that all of the layers in a network will have a similar variance in terms of both forward pass activations and the gradients propagated during the backward pass in backpropagation. Glorot initialization defines a heuristic rule to meet this goal that involves sampling the weights for a network using the following uniform distribution (where w is the weight on a connection between layer j and $j+1$ that is being initialized, $U[-a,a]$ is the uniform distribution over the interval $(-a,a)$, n_j is the number of neurons in layer j , and the notation $w \sim U$ indicates that the value of w is sampled from distribution U)¹¹:

$$w \sim U \left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}} \right]$$

Another factor that contributes to the success or failure of training a deep network is the selection of the

activation function used in the neurons. Backpropagating an error gradient through a neuron involves multiplying the gradient by the value of the derivative of the activation function at the activation value of the neuron recorded during the forward pass. The derivatives of the logistic and tanh activation functions have a number of properties that can exacerbate the vanishing gradient problem if they are used in this multiplication step. Figure 4.6 presents a plot of the logistic function and the derivative of the logistic function. The maximum value of the derivative is 0.25. Consequently, after an error gradient has been multiplied

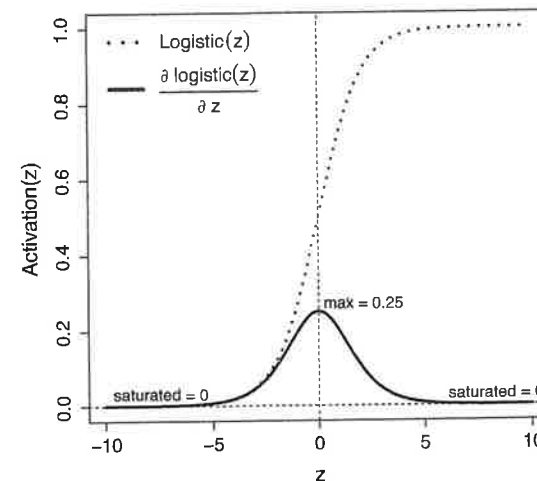


Figure 4.6 Plots of the logistic function and the derivative of the logistic function.

by the value of the derivative of the logistic function at the appropriate activation for the neuron, the maximum value the gradient will have is a quarter of the gradient prior to the multiplication. Another problem with using the logistic function is that there are large portions of the domain of the function where the function is *saturated* (returning values that very close to 0 or 1), and the rate of change of the function in these regions is near zero; thus, the derivative of the function is near 0. This is an undesirable property when backpropagating error gradients because the error gradients will be forced to zero (or close to zero) when backpropagated through any neuron whose activation is within one of these saturated regions. In 2011 it was shown that switching to a rectified linear activation function, $g(x) = \max(0, x)$, improved training for deep feedforward neural networks (Glorot et al. 2011). Neurons that use a rectified linear activation function are known as rectified linear units (ReLU). One advantage of ReLUs is that the activation function is linear for the positive portion of its domain with a derivative equal to 1. This means that gradients can flow easily through ReLUs that have positive activation. However, the drawback of ReLUs is that the gradient of the function for the negative part of its domain is zero, so ReLUs do not train in this portion of the domain. Although undesirable, this is not necessarily a fatal flaw for learning because when backpropagating through a layer of ReLUs the gradients

can still flow through the ReLUs in the layers that have positive activation. Furthermore, there are a number of variants of the basic ReLU that introduce a gradient on the negative side of the domain, a commonly used variant being the leaky ReLU (Maas et al. 2013). Today, ReLUs (or variants of ReLUs) are the most frequently used neurons in deep learning research.

The Virtuous Cycle: Better Algorithms, Faster Hardware, Bigger Data

Although improved weight initialization methods and new activation functions have both contributed to the growth of deep learning, in recent years the two most important factors driving deep learning have been the speedup in computer power and the massive increase in dataset sizes. From a computational perspective, a major breakthrough for deep learning occurred in the late 2000s with the adoption of graphical processing units (GPUs) by the deep learning community to speed up training. A neural network can be understood as a sequence of matrix multiplications that are interspersed with the application of nonlinear activation functions, and GPUs are optimized for very fast matrix multiplication. Consequently, GPUs are ideal hardware to speed up neural network training, and their use has made a significant contribution to the development of the field. In 2004, Oh and Jung reported a twentyfold performance increase using a GPU