KSBi-BIML 2026

Bioinformatics & Machine Learning(BIML) Workshop for Life Scientists

생명정보학 & 머신러닝 워크샵 (온라인)

딥러닝(Deep neural networks)의 이해

이제근 _ 숭실대학교

KSBI KOREAN SOCIETY FOR BIOINFORMATICS | 한국생명정보학회

본 강의 자료는 한국생명정보학회가 주관하는 BIML 2026 워크샵을 목적으로
제작된 것으로 해당 목적 이외의 다른 용도로 사용할 수 없음을 분명하게 알립니다.

이를 다른 사람과 공유하거나 복제, 배포, 전송할 수 없으며 만약 이러한 사항을 위반할 경우
발생하는 **모든 법적 책임은 행위자 본인에게 있음**을 알립니다.

KSBi-BIML 2026

Bioinformatics & Machine Learning (BIML) Workshop for Life Scientists

한국생명정보학회가 주최하는 BIML-2026 동계 Bioinformatics &Machine Learning 교육 워크숍에 여러분을 초대합니다.

BIML 워크숍은 생명정보학 연구자들이 최신 AI바이오 분야의 인공지능 기반 분석 기술과 바이오 데이터 분석 기법을 이론과 실습을 통해 체계적으로 배울 수 있는 전문 교육 프로그램입니다. 2015년에 시작된 BIML 워크숍은 올해로 12년 차를 맞이하며, 국내 생명정보학 분야의 최초이자 최고 수준의 교육 프로그램으로 자리 잡았습니다. 이번 워크숍은 크게 인공지능바이오(AI바이오) 분야와 디지털바이오 분야, 두 분야로 구성됩니다.

AI바이오 분야에서는 생명정보 분석에 폭넓게 응용되고 있는 다양한 인공지능 기반 자료 모델링 기법을 다룰 예정입니다. 특히, 인공지능 심층학습을 활용한 단백질 구조 예측, 유전체 분석, 신약 개발에 대한 이론 및 실습 강의가 진행됩니다.

또한 디지털바이오 분야에서는 단일세포오믹스, 공간오믹스, 멀티오믹스, 메타오믹스에 대한 강의도 마련되어 있어, 연구자들의 분석 역량 강화에 실질적인 도움을 줄 것으로 기대됩니다.

또한 2024년부터 추가된 의료정보 자료 분석을 다루는 강의를 올해도 지속해서 운영하고자 합니다. 이는 최근 의료정보 자료 분석에 관한 연구 수요 증가를 반영한 것으로, 관련 연구를 수행하는 의과학자 및 의료정보 연구자들에게 유용한 지침을 제공할 것입니다.

또한, 올해도 생명정보학 기술의 다양화에 발맞춰 온라인 강좌를 대폭 확대했습니다. 올해는 무료 강좌 10개를 포함한 총 40개 이상의 강좌가 개설되며, 연구 주제에 맞는 강좌 추천과 강연료 할인 혜택도 제공합니다.

BIML-2026는 국내 주요 연구 중심 대학의 전임 교수 및 각 분야 최고 전문가들의 강의로 구성되어 있으며, 기초 이론부터 최신 연구 동향까지 아우르는 심도 있는 교육의 장이 될 것으로 확신합니다.

여러분의 많은 관심과 참여를 기대합니다!

2026년 2월

한국생명정보학회장 류 성 호

강의개요

딥러닝(Deep Neural Networks)의 이해

본 강의는 인공지능의 핵심 기술로 자리잡은 딥러닝(deep neural networks)의 기본 개념부터 심화 내용까지 체계적으로 학습하여 딥러닝에 대한 깊이 있는 이해를 도모하는 것을 목표로 한다. 인공신경망(artificial neural networks)의 시초인 단순 퍼셉트론(Simple Perceptron)부터 현대의 심층 신경망에 이르기까지의 발전 과정을 살펴보며, 각 모델의 특징과 각각의 방법론들을 이해한다. 또한 인공신경망과 심층신경망의 기본 구조와 학습 원리를 심도 있게 학습하여, 현대 딥러닝에서 널리 활용되는 다양한 학습 기법들을 소개함으로써, 실제 문제 해결에 딥러닝을 적용할 수 있는 이론적 기반을 마련한다.

* 교육생준비물: 없음.

* 강의 난이도: 중급

* 강의: 이제근 교수 (숭실대학교 의생명시스템학부)

Curriculum Vitae

Speaker Name: Je-Keun Rhee, Ph.D.

▶ **Personal Info**

Name	Je-Keun Rhee
Title	Associate Professor
Affiliation	Soongsil University

▶ **Contact Information**

Address	369, Sangdo-ro, Dongjak-gu, Seoul, Republic of Korea
Email	jkrhee@ssu.ac.kr

Research Interest

Cancer genomics, Epigenomics, Machine learning

Educational Experience

2004	B.S. in Life Science, Korea University, Korea
2004	B.S. in Computer Science & Engineering, Korea University, Korea (double major)
2014	Ph.D. in Bioinformatics, Seoul National University, Korea

Professional Experience

2011-2011	Visiting Scholar, School of Informatics and Computing (SoIC), Indiana University, USA
2014-2018	Post-doctoral Fellow/Research Assistant Professor, Catholic University of Korea College of Medicine
2018-2019	Assistant Professor, Pusan National University
2019-	Assistant Professor/Associate Professor, Soongsil University

Selected Publications (3 maximum)

([§]Corresponding author)

1. Dong-Yeon Nam, **Je-Keun Rhee**[§], Identifying microRNAs associated with tumor immunotherapy response using an interpretable machine learning model, *Scientific Reports,* 14:6172, 2024.
2. Bonil Koo, **Je-Keun Rhee**[§], Prediction of tumor purity from gene expression data using machine learning, *Briefings in Bioinformatics,* 22(6):bbab163, 2021.
3. Yeongjoo Kim, Ji Wan Kang, Junho Kang, Eun Jung Kwon, Mihyang Ha, Yoon Kyoung Kim, Hansong Lee, **Je-Keun Rhee**[§], Yun Hak Kim§, Novel deep learning-based survival prediction for oral cancer by analyzing tumor-infiltrating lymphocyte profiles through CIBERSORT, *OncoImmunology,* 10(1):e1904573, 2021

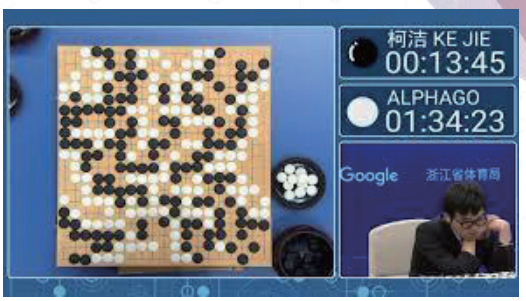
KSBI | 한국생명정보학회

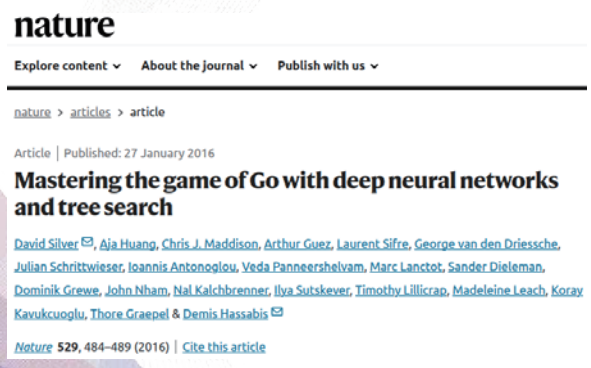
Deep neural network의 이해

숭실대학교 의생명시스템학부

이제근(Je-Keun Rhee)

Mastering the game of GO





nature

Explore content · About the journal · Publish with us

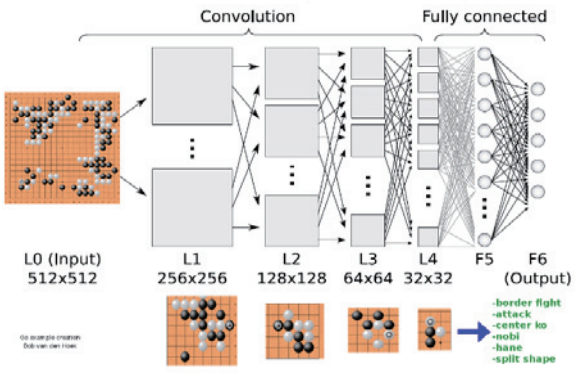
nature > articles > article

Article | Published: 27 January 2016

Mastering the game of Go with deep neural networks and tree search

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel & Demis Hassabis

Nature **529**, 484–489 (2016) | Cite this article

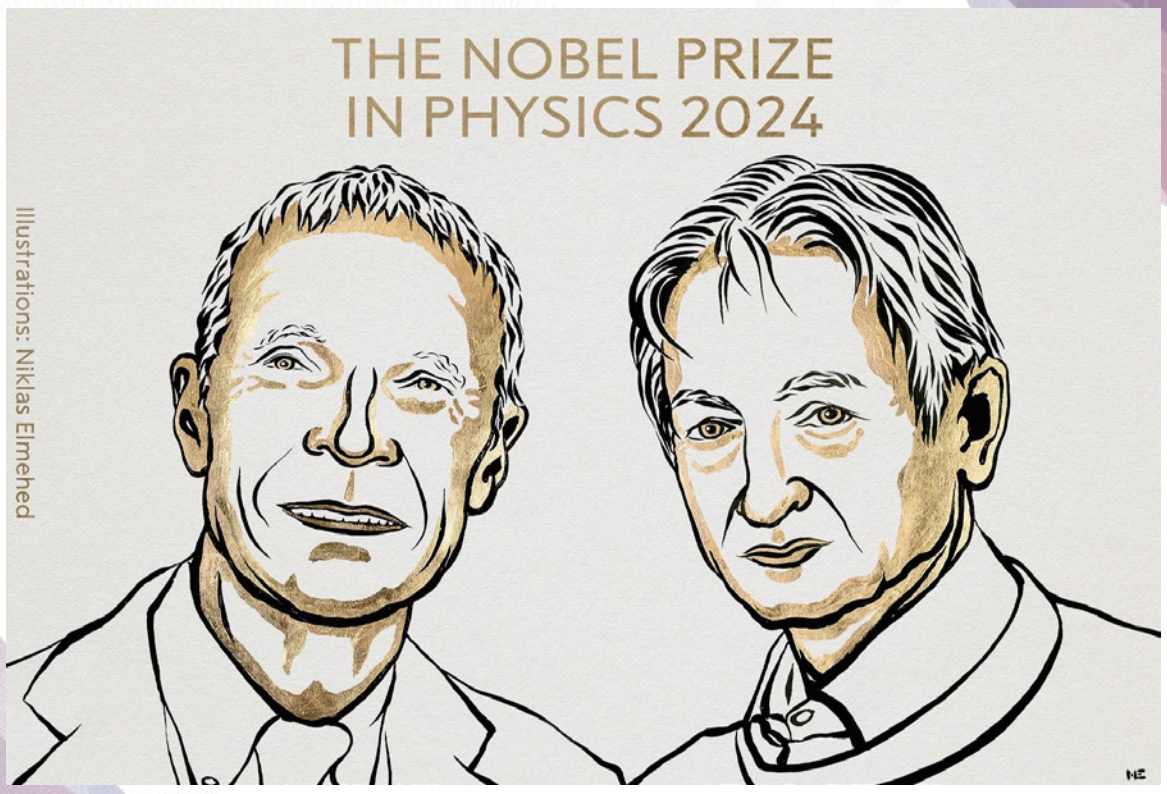


Nobel prize in chemistry 2024

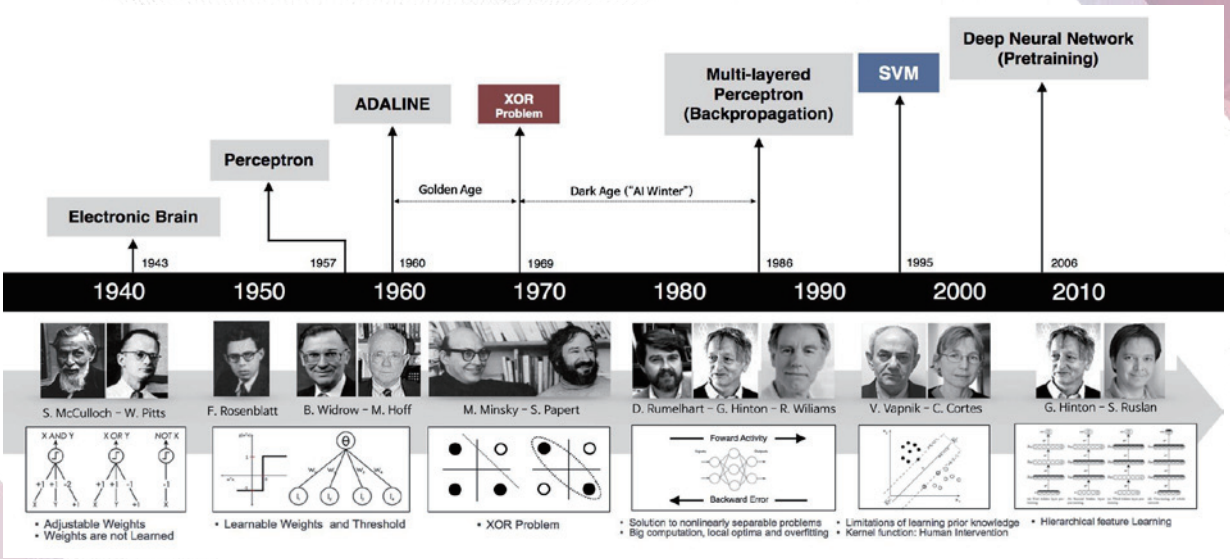
David Baker. Ill. Niklas Elmehed © Nobel Prize Outreach

Demis Hassabis. Ill. Niklas Elmehed © Nobel Prize Outreach

John Jumper. Ill. Niklas Elmehed © Nobel Prize Outreach



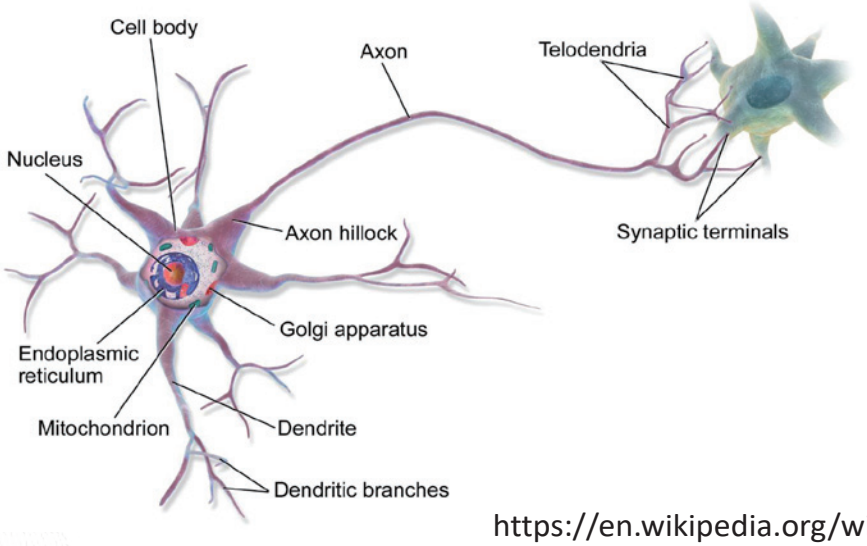
Milestones in the development of neural networks



http://beamlab.org/deeplearning/2017/02/23/deep_learning_101_part1.html

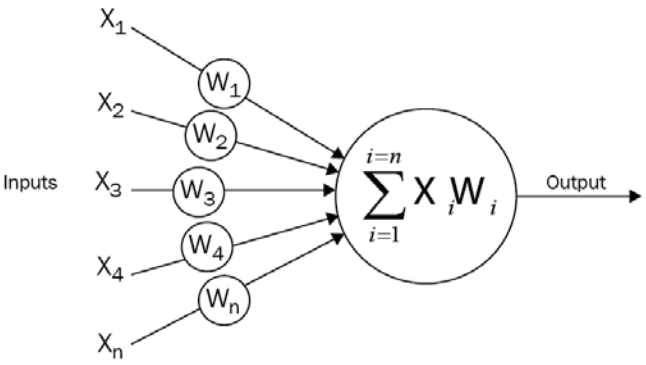
Neuron

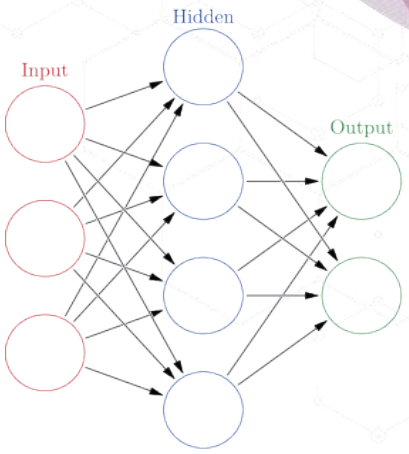
- Information processing unit
  - Dendrite accepts signals from other neurons
  - Axon carries electronical impulses away from the cell body



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

Artificial neural networks





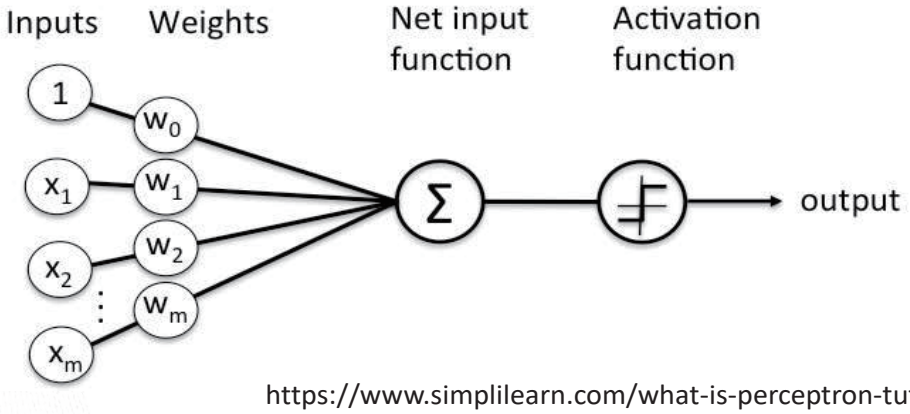
https://subscription.packtpub.com/book/data/9781788392303/1/ch01lvl1sec10/building-blocks-of-a-neural-network

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

(Simple) Perceptron

- A neural net with a single neuron
  - Input: a vector of real values
  - Output: 1 or -1 (binary)
  - Activation function: threshold function

$$f(\mathbf{x}) = w_0 + w_1 x_1 + \cdots + w_m x_m$$



https://www.simplilearn.com/what-is-perceptron-tutorial

Basic perceptron learning rule

Given Training Data:

$D = \{(x^{(i)}, y^{(i)}) \mid x^{(i)} \in \mathbb{R}^m, y^{(i)} \in \{0,1\}, i = 1,\ldots,n\}$

Algorithm:

1. Initialize weights:

   $w \leftarrow 0^m$ (including bias term)

2. For each epoch:

   For each training sample $(x^{(i)}, y^{(i)})$:

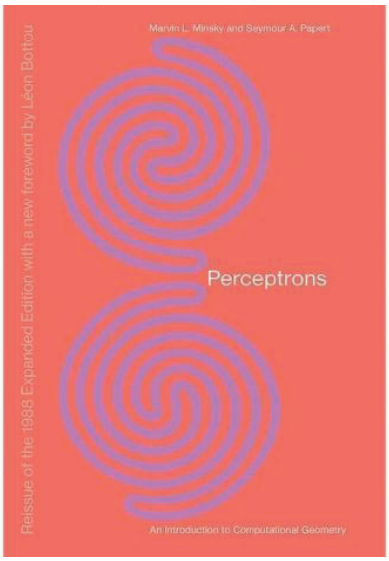
   a) $\hat{y}^{(i)} = \sigma(x^{(i)T}w)$

   b) $err = y^{(i)} - \hat{y}^{(i)}$

   c) $w \leftarrow w + \eta \cdot err \cdot x^{(i)}$ (η: learning rate)

Perceptrons: an introduction to computational geometry (1969)

- by Marvin Minskey, founder of the MIT AI lab
- Perceptrons could not solve XOR problems!



Logic gate

AND

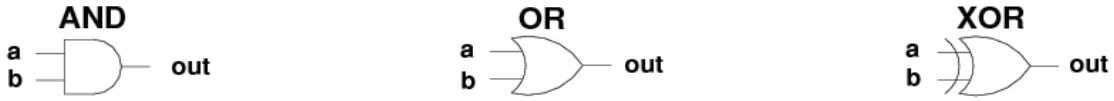
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

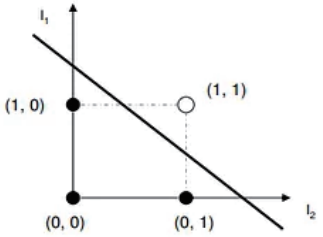
XOR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

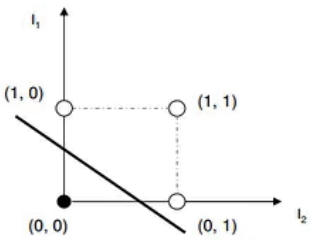


XOR

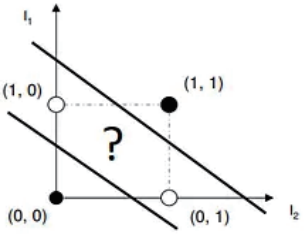
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1

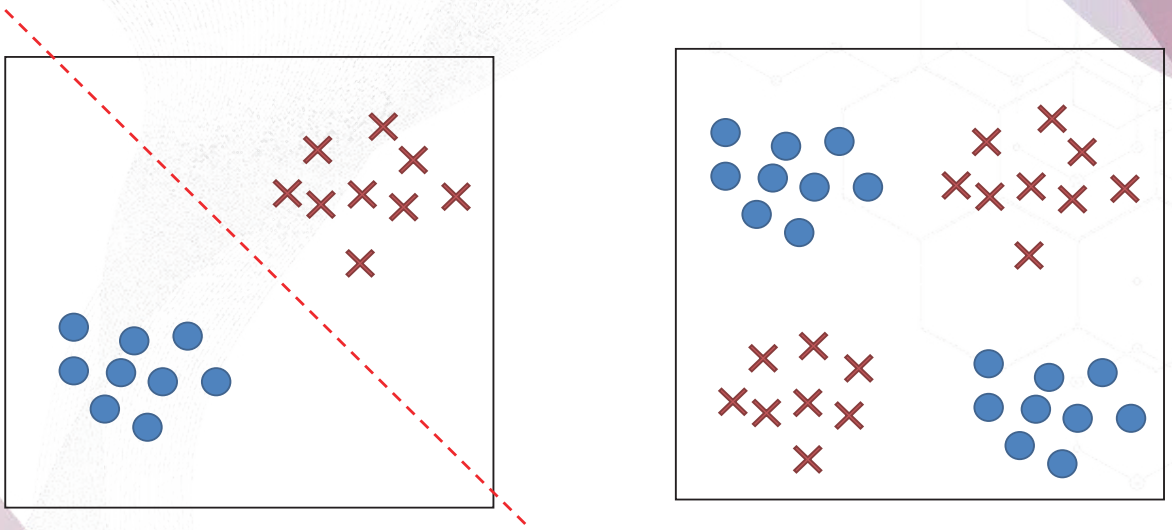


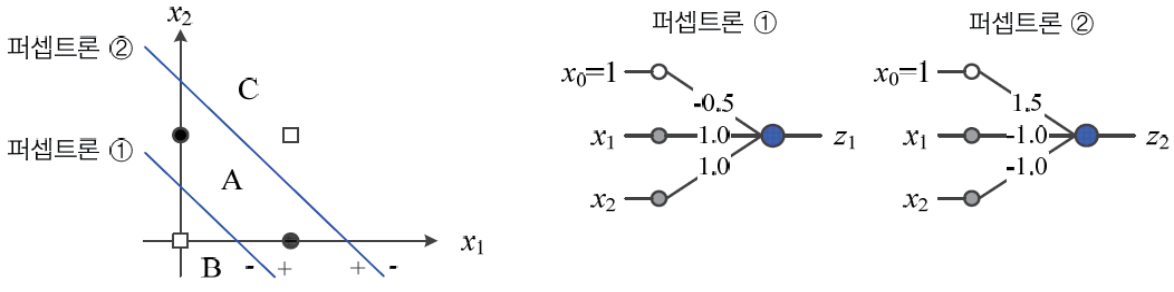
XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0



https://medium.com/@lucaspereira0612/solving-xor-with-a-single-perceptron-34539f395182

Linearly separable *vs.* nonseparable



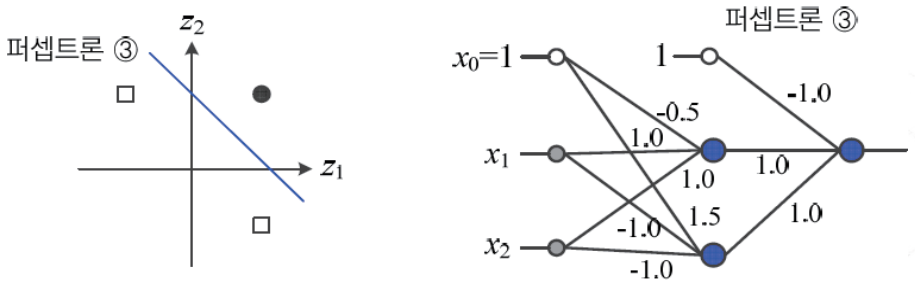


(a) 퍼셉트론 2개를 이용한 공간분할 (b) 퍼셉트론 2개

그림 3-8 XOR 문제의 해결

오일석, 기계학습, 한빛미디어

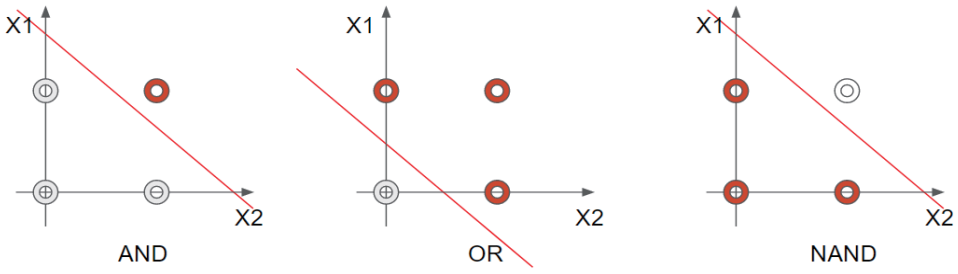
Multilayer perceptron

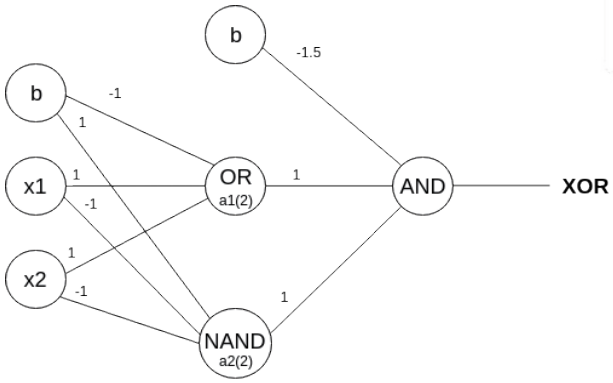


(a) 새로운 특징 공간에서 분할 (b) 퍼셉트론 3개를 결합한 다층 퍼셉트론

그림 3–10 다층 퍼셉트론

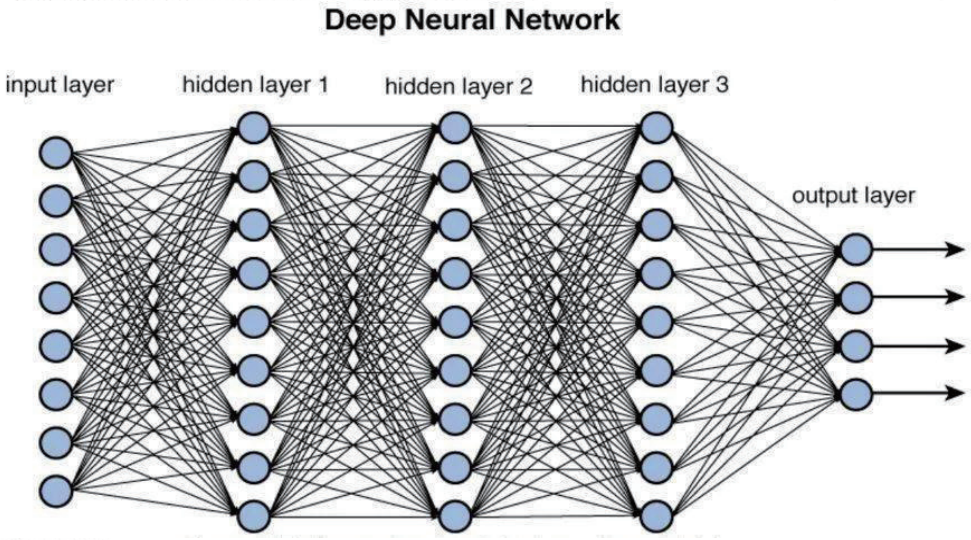
오일석, 기계학습, 한빛미디어





https://www.i2tutorials.com/exploring-or-xorand-gate-in-neural-network/

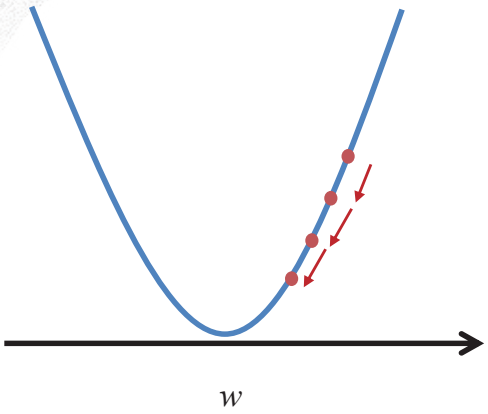
Deep neural networks



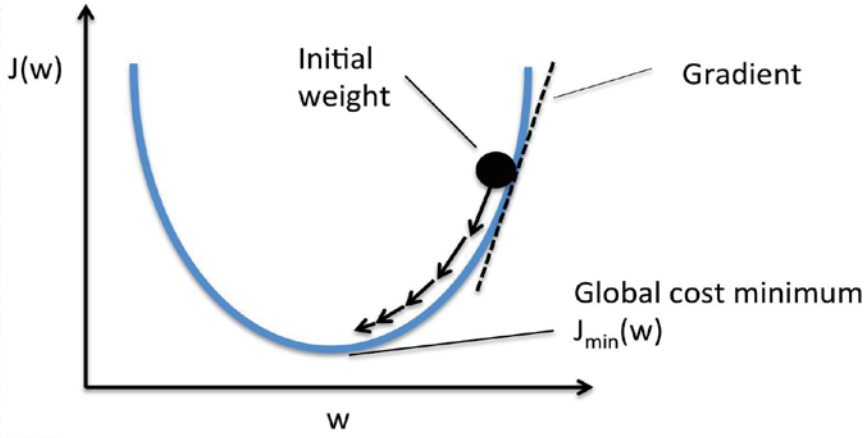
https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

- How can we determine the weights from training data?

Gradient descent



Gradient descent



https://hackernoon.com/gradient-descent-aynk-7cbe95a778da

The learning rate and steepness of the gradient indicates the direction to the minimum point and how much we update

Backpropagation

Learning representations by back-propagating errors

David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams

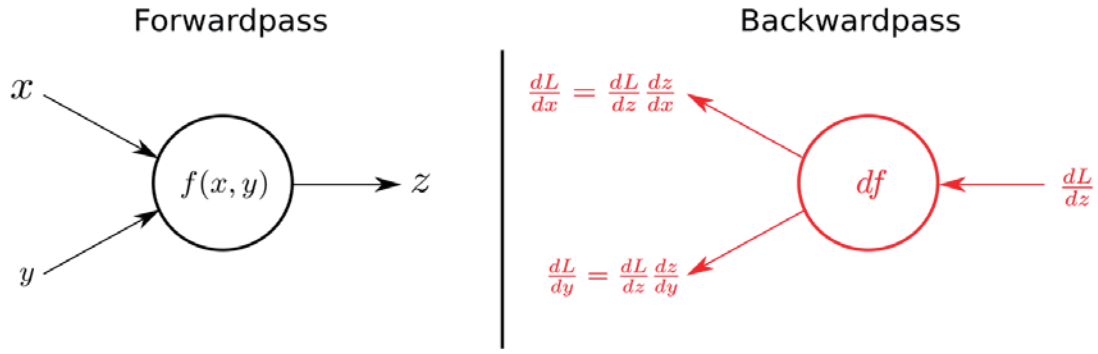
Nature **323**, 533–536(1986) | Cite this article

29k Accesses | **8351** Citations | **172** Altmetric | Metrics

Abstract

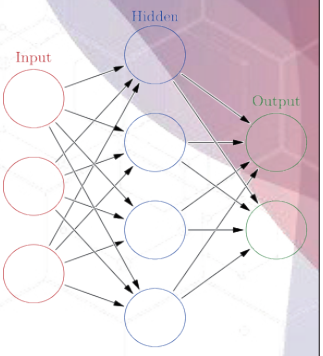
We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

Feed-forward & Error propagation



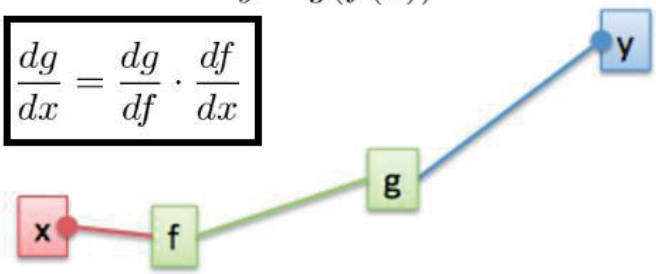
https://github.com/Vercaca/NN-Backpropagation

Chain rule



$$y = g(f(x))$$

$$\frac{dg}{dx} = \frac{dg}{df} \cdot \frac{df}{dx}$$



https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/backpropagation.html

$$y = f(g(x))$$

$$y' = ?$$

$$y = f(g(x))$$

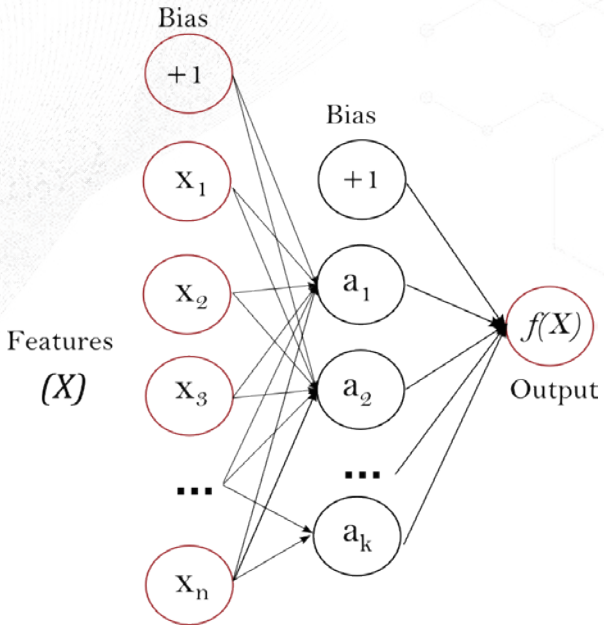
$$y' = f'(g(x))g'(x)$$

$$\frac{d}{dx}\left[f(g(x))\right] = \frac{df}{dg} \cdot \frac{dg}{dx}$$

$$F(x) = f(g(h(u(v(x)))))$$

$$\begin{aligned}\frac{dF}{dx} = \frac{d}{dx}F(x) &= \frac{d}{dx}f(g(h(u(v(x))))) \\ &= \frac{df}{dg} \cdot \frac{dg}{dh} \cdot \frac{dh}{du} \cdot \frac{du}{dv} \cdot \frac{dv}{dx}\end{aligned}$$

Training with gradient descent



https://scikit-learn.org/stable/modules/neural_networks_supervised.html

Review for the perceptron learning

Given Training Data:

$D = \{(x^{(i)}, y^{(i)}) \mid x^{(i)} \in \mathbb{R}^m, y^{(i)} \in \{0,1\}, i = 1,\ldots,n\}$

Algorithm:

1. Initialize weights:

   $w \leftarrow 0^m$ (including bias term)

2. For each epoch:

   For each training sample $(x^{(i)}, y^{(i)})$:

   a) $\hat{y}^{(i)} = \sigma(x^{(i)T}w)$

   b) $err = y^{(i)} - \hat{y}^{(i)}$

   c) $w \leftarrow w + \eta \cdot err \cdot x^{(i)}$ (η: learning rate)

Batch mode

$D = \{(x^{(i)}, y^{(i)}) \mid x^{(i)} \in \mathbb{R}^m, y^{(i)} \in \{0,1\}, i = 1,\ldots,n\}$

1. Initialize:

   $w := 0^{m-1}$, $b := 0$

2. For each training epoch:

   A. Initialize updates:

      $\Delta w := 0$, $\Delta b := 0$

   B. For each training sample $(x^{(i)}, y^{(i)})$:

      (a) Compute output: $\hat{y}^{(i)} = \sigma(w \cdot x^{(i)} + b)$

      (b) Calculate error: $err^{(i)} = y^{(i)} - \hat{y}^{(i)}$

      (c) Update $\Delta w := \Delta w + err^{(i)} \cdot x^{(i)}$, $\Delta b := \Delta b + err^{(i)}$

   C. Update parameters: $w := w + \Delta w$, $b := b + \Delta b$

Stochastic learning

$D = \{(x^{(i)}, y^{(i)}) \mid x^{(i)} \in \mathbb{R}^m, y^{(i)} \in \{0,1\}, i = 1,\ldots,n\}$

1. Initialize:

   $w := 0^{m-1}$, $b := 0$, $\eta := learning\_rate$

2. For each training epoch:

   A. Randomly shuffle training data D

   B. For each training sample $(x^{(i)}, y^{(i)})$:

      (a) Compute output: $\hat{y}^{(i)} = \sigma(w \cdot x^{(i)} + b)$

      (b) Calculate error: $err^{(i)} = y^{(i)} - \hat{y}^{(i)}$

      (c) Immediate update:

      $w := w + \eta \cdot err^{(i)} \cdot x^{(i)}$

      $b := b + \eta \cdot err^{(i)}$

Minibatch stochastic learning

$D = \{(x^{(i)}, y^{(i)}) \mid x^{(i)} \in \mathbb{R}^m, y^{(i)} \in \{0,1\}, i = 1,\ldots,n\}$

1. Initialize:

   $w := 0^{m-1}$, $b := 0$, $\eta :=$ learning_rate, $B :=$ batch_size

2. For each training epoch:

   A. Randomly shuffle training data D

   B. For each mini-batch D_β of size B:

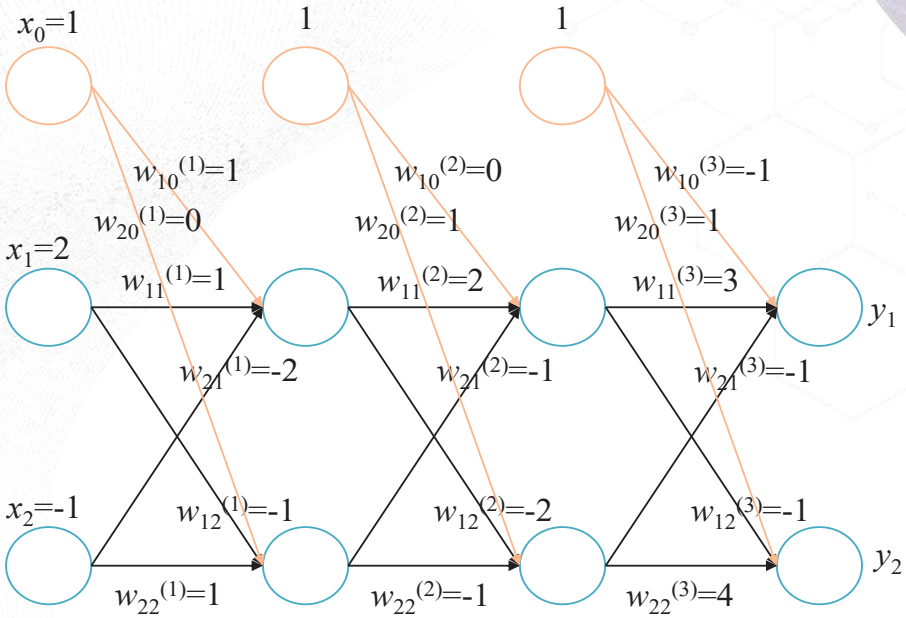
      (a) Initialize: $\Delta w := 0$, $\Delta b := 0$

      (b) For each $(x^{(i)}, y^{(i)})$ in D_β:

      - Compute $\hat{y}^{(i)} = \sigma(w \cdot x^{(i)} + b)$
      - $err^{(i)} = y^{(i)} - \hat{y}^{(i)}$
      - $\Delta w := \Delta w + err^{(i)} \cdot x^{(i)}$, $\Delta b := \Delta b + err^{(i)}$

      (c) Update: $w := w + (\eta/B) \cdot \Delta w$, $b := b + (\eta/B) \cdot \Delta b$

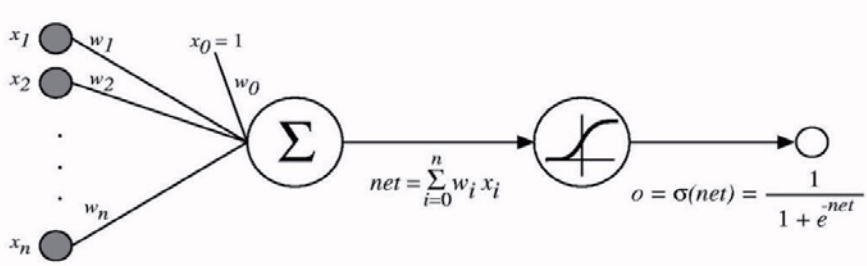
DNN forward pass



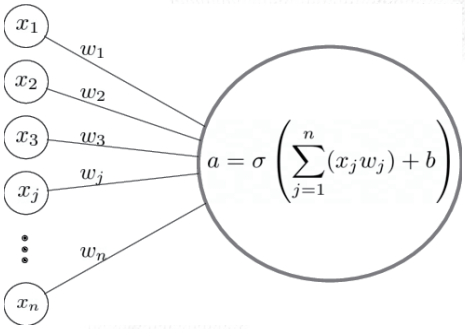
Backpropagation

- Find the optimal parameter minimize the total cost (MSE) $C = \frac{1}{2}\sum_{n=1}^{N}(o^{(n)} - y^{(n)})^2$
- Iteratively updating the parameter

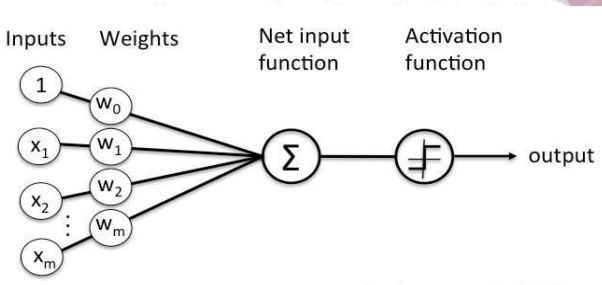
$$w_i(t+1) = w_i(t) - \eta \frac{\partial C}{\partial w_i}$$



Activation function

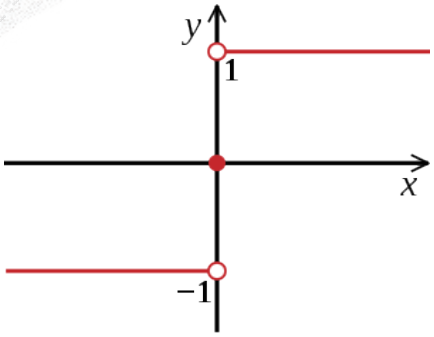


https://www.researchgate.net/figure/The-perceptron-processes-the-inputs-and-regularizes-them-with-an-activation-function_fig1_357301768



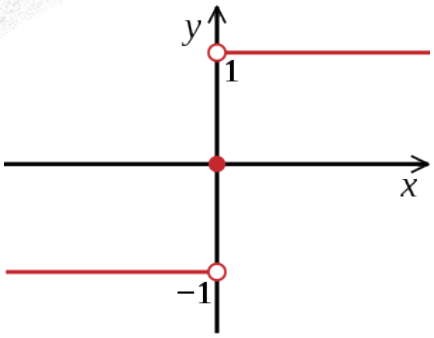
https://www.simplilearn.com/what-is-perceptron-tutorial

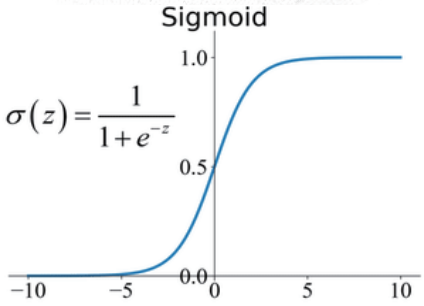
sign function

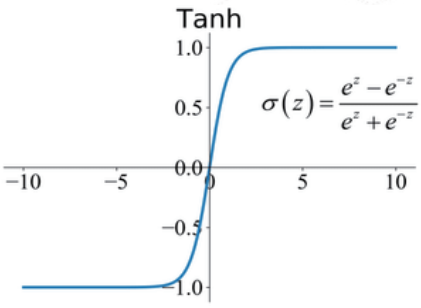


Activation function

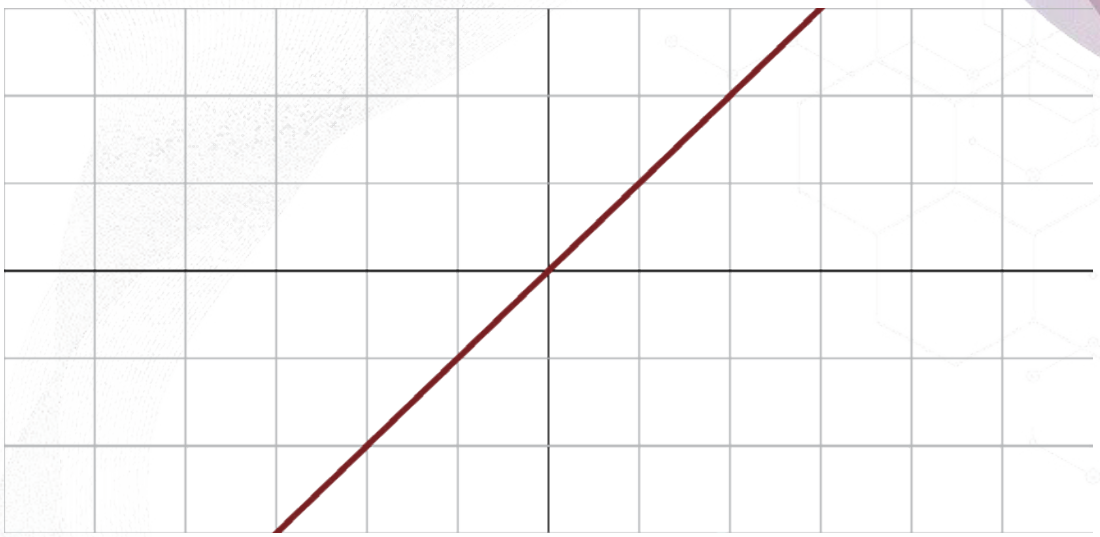
- How do I calculate derivative of sign function?







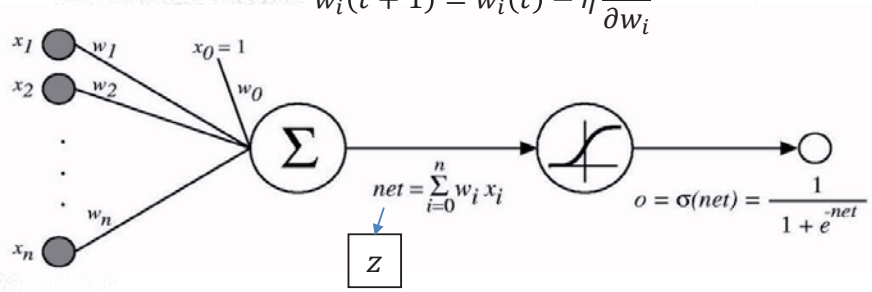
Why should we use non-linear activation function in the artificial neural network?



Backpropagation

- Find the optimal parameter minimize the total cost (MSE) $C = \frac{1}{2}\sum_{n=1}^{N}(o^{(n)} - y^{(n)})^2$
- Iteratively updating the parameter

$$w_i(t+1) = w_i(t) - \eta \frac{\partial C}{\partial w_i}$$



$$\frac{\partial C}{\partial w_i} = \sum_{n=1}^{N} \frac{\partial z}{\partial w_i}\frac{\partial o}{\partial z}\frac{\partial C}{\partial o} =$$

- Can you calculate derivative of the sigmoid function (activation function)?

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Derivative of sigmoid function

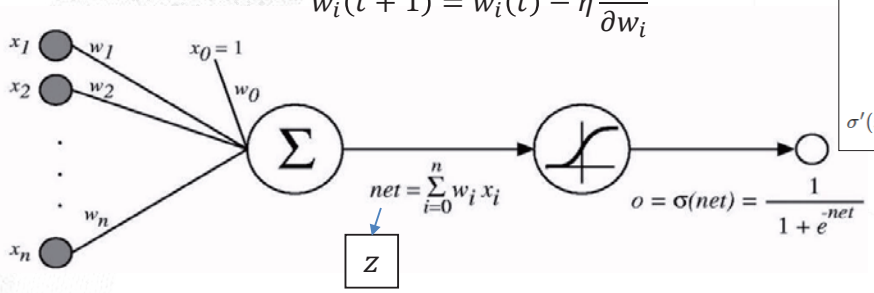
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\begin{aligned}
\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] = \frac{d}{dx}(1+e^{-x})^{-1} \\
&= -1 * (1+e^{-x})^{-2}(-e^{-x}) \\
&= \frac{-e^{-x}}{-(1+e^{-x})^2} \\
&= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{1}{1+e^{-x}}\frac{e^{-x}}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}}\frac{e^{-x}+(1-1)}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}}\frac{(1+e^{-x})-1}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}}\left[\frac{(1+e^{-x})}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right] \\
&= \frac{1}{1+e^{-x}}\left[1 - \frac{1}{1+e^{-x}}\right] \\
&= \sigma(x)(1-\sigma(x))
\end{aligned}$$

Backpropagation

- Find the optimal parameter minimize the total cost (MSE) $C = \frac{1}{2}\sum_{n=1}^{N}(o^{(n)} - y^{(n)})^2$
- Iteratively updating the parameter

$$w_i(t+1) = w_i(t) - \eta\frac{\partial C}{\partial w_i}$$



$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma'(x) = \frac{d}{dx}\sigma(x) = \sigma(x)(1-\sigma(x))$$

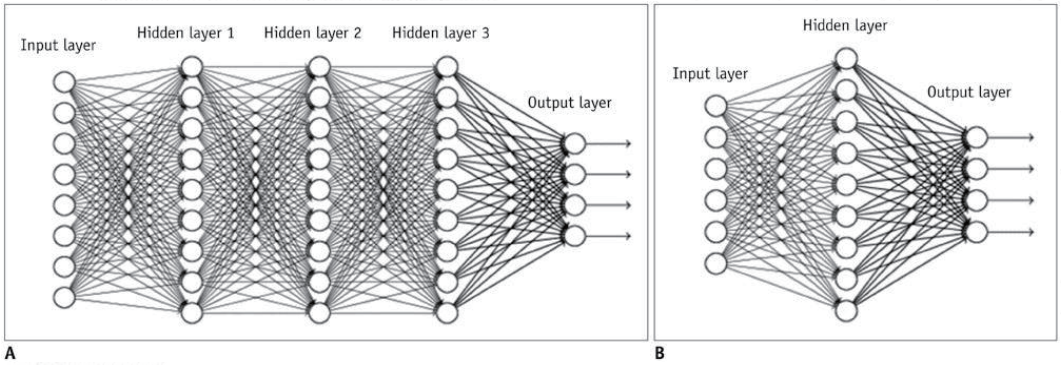
$$\frac{\partial z}{\partial w_i} = x_i \qquad \frac{\partial o}{\partial z} = o(1-o)$$

$$\frac{\partial C}{\partial w_i} = \sum_{n=1}^{N}\frac{\partial z}{\partial w_i}\frac{\partial o}{\partial z}\frac{\partial C}{\partial o} = \sum_{n=1}^{N} x_i o(1-o)(o-y)$$

What about multiple layers?

- You can compute the derivative of the cost function at a similar way

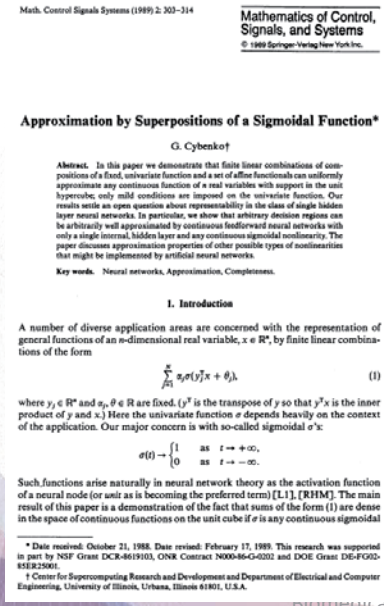
Are more layers better in a neural network?



https://doi.org/10.3348/kjr.2017.18.4.570

Universal approximation theorem

- Neural networks with one hidden layer can represent any continuous function
  - The approximation can be improved by increasing the number of hidden nodes. (with many hidden nodes and nonlinear activation function)



Math. Control Signals Systems (1989) 2: 303–314

Mathematics of Control, Signals, and Systems

© 1989 Springer-Verlag New York Inc.

Approximation by Superpositions of a Sigmoidal Function*

G. Cybenko†

Abstract. In this paper we demonstrate that finite linear combinations of compositions of a fixed, univariate function and a set of affine functionals can uniformly approximate any continuous function of n real variables with support in the unit hypercube; only mild conditions are imposed on the univariate function. Our results settle an open question about representability in the class of single hidden layer neural networks. In particular, we show that arbitrary decision regions can be arbitrarily well approximated by continuous feedforward neural networks with only a single internal, hidden layer and any continuous sigmoidal nonlinearity. The paper discusses approximation properties of other possible types of nonlinearities that might be implemented by artificial neural networks.

Key words. Neural networks, Approximation, Completeness.

1. Introduction

A number of diverse application areas are concerned with the representation of general functions of an n-dimensional real variable, $x \in \mathbb{R}^n$, by finite linear combinations of the form

$$\sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j), \qquad (1)$$

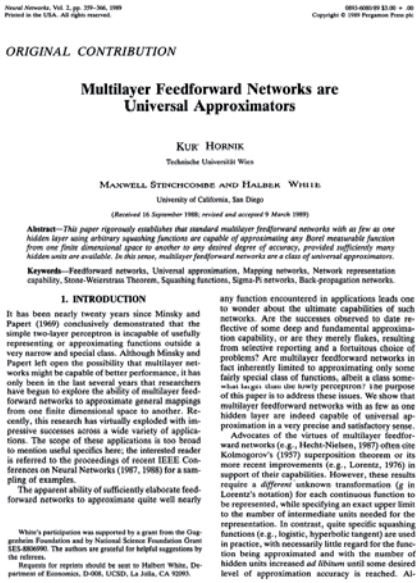
where $y_j \in \mathbb{R}^n$ and $\alpha_j, \theta \in \mathbb{R}$ are fixed. (y^T is the transpose of y so that $y^T x$ is the inner product of y and x.) Here the univariate function σ depends heavily on the context of the application. Our major concern is with so-called sigmoidal σ's:

$$\sigma(t) \to \begin{cases} 1 & \text{as } t \to +\infty, \\ 0 & \text{as } t \to -\infty. \end{cases}$$

Such functions arise naturally in neural network theory as the activation function of a neural node (or *unit* as is becoming the preferred term) [LI], [RHM]. The main result of this paper is a demonstration of the fact that sums of the form (1) are dense in the space of continuous functions on the unit cube if σ is any continuous sigmoidal

* Date received: October 21, 1988. Date revised: February 17, 1989. This research was supported in part by NSF Grant DCR-8619103, ONR Contract N000-86-G-0202 and DOE Grant DE-FG02-85ER25001.

† Center for Supercomputing Research and Development and Department of Electrical and Computer Engineering, University of Illinois, Urbana, Illinois 61801, U.S.A.



Neural Networks, Vol. 2, pp. 359–366, 1989
Printed in the USA. All rights reserved.

ORIGINAL CONTRIBUTION

Multilayer Feedforward Networks are Universal Approximators

KURT HORNIK
Technische Universität Wien

MAXWELL STINCHCOMBE AND HALBERT WHITE
University of California, San Diego

(Received 16 September 1988; revised and accepted 9 March 1989)

Abstract—*This paper rigorously establishes that standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available. In this sense, multilayer feedforward networks are a class of universal approximators.*

Keywords—Feedforward networks, Universal approximation, Mapping networks, Network representation capability, Stone-Weierstrass Theorem, Squashing functions, Sigma-Pi networks, Back-propagation networks.

1. INTRODUCTION

It has been nearly twenty years since Minsky and Papert (1969) conclusively demonstrated that the simple two-layer perceptron is incapable of usefully representing or approximating functions outside a very narrow and special class. Although Minsky and Papert left open the possibility that multilayer networks might be capable of better performance, it has only been in the last several years that researchers have begun to explore the ability of multilayer feedforward networks to approximate general mappings from one finite dimensional space to another. Recently, this research has virtually exploded with impressive successes across a wide variety of applications. The scope of these applications is too broad to mention useful specifics here; the interested reader is referred to the proceedings of recent IEEE Conferences on Neural Networks (1987, 1988) for a sampling of examples.

The apparent ability of sufficiently elaborate feedforward networks to approximate quite well nearly any function encountered in applications leads one to wonder about the ultimate capabilities of such networks. Are the successes observed to date reflective of some deep and fundamental approximation capability, or are they merely flukes, resulting from selective reporting and a fortuitous choice of problems? Are multilayer feedforward networks in fact inherently limited to approximating only some fairly special class of functions, albeit a class somewhat larger than the lowly perceptron? The purpose of this paper is to address these issues. We show that multilayer feedforward networks with as few as one hidden layer are indeed capable of universal approximation in a very precise and satisfactory sense.

Advocates of the virtues of multilayer feedforward networks (e.g., Hecht-Nielsen, 1987) often cite Kolmogorov's (1957) superposition theorem or its more recent improvements (e.g., Lorentz, 1976) in support of their capabilities. However, these results require a *different* unknown transformation (g in Lorentz's notation) for each continuous function to be represented, while specifying an exact upper limit to the number of intermediate units needed for the representation. In contrast, quite specific squashing functions (e.g., logistic, hyperbolic tangent) are used in practice, with necessarily little regard for the function being approximated and with the number of hidden units increased *ad libitum* until some desired level of approximation accuracy is reached. Al-

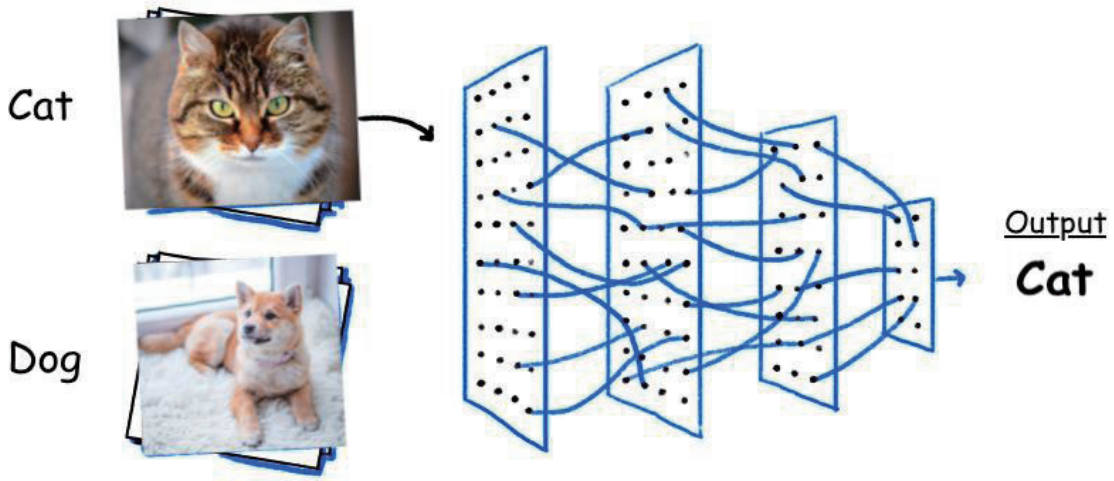
White's participation was supported by a grant from the Guggenheim Foundation and by National Science Foundation Grant SES-8806990. The authors are grateful for helpful suggestions by the referees.

Requests for reprints should be sent to Halbert White, Department of Economics, D-008, UCSD, La Jolla, CA 92093.

- Deeper neural networks may be generally better than shallow networks.

- But..
  - Training datasets were too small
  - Computing power was not enough
  - Initialization of the weights in a wrong way
  - Improper nonlinear activation functions
  - Etc.

Image classification (convolutional neural networks)



Backpropagation Applied to Handwritten Zip Code Recognition

Y. LeCun
B. Boser
J. S. Denker
D. Henderson
R. E. Howard
W. Hubbard
L. D. Jackel
AT&T Bell Laboratories Holmdel, NJ 07733 USA

The ability of learning networks to generalize can be greatly enhanced by providing constraints from the task domain. This paper demonstrates how such constraints can be integrated into a backpropagation network through the architecture of the network. This approach has been successfully applied to the recognition of handwritten zip code digits provided by the U.S. Postal Service. A single network learns the entire recognition operation, going from the normalized image of the character to the final classification.

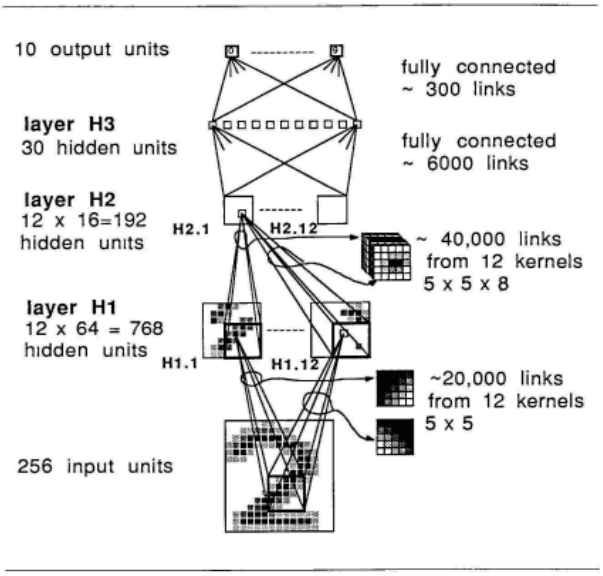
1 Introduction

Previous work performed on recognizing simple digit images (LeCun 1989) showed that good generalization on complex tasks can be obtained by designing a network architecture that contains a certain amount of a prior knowledge about the task. The basic design principle is to reduce the number of free parameters in the network as much as possible without overly reducing its computational power. Application of this principle increases the probability of correct generalization because it results in a specialized network architecture that has a reduced entropy (Denker *et al* 1987; Patarnello and Carnevali 1987, Tishby *et al.* 1989; LeCun 1989), and a reduced Vapnik–Chervonenkis dimensionality (Baum and Haussler 1989).

In this paper, we apply the backpropagation algorithm (Rumelhart *et al* 1986) to a real-world problem in recognizing handwritten digits taken from the U S Mail. Unlike previous results reported by our group on this problem (Denker *et al* 1989), the learning network is directly fed with images, rather than feature vectors, thus demonstrating the ability of backpropagation networks to deal with large amounts of low-level information.

Neural Computation **1, 541–551** (1989)

- The Yann LeCun et al. (1989) paper
- Backpropagation Applied to Handwritten Zip Code Recognition



Recurrent neural network (RNN) & LSTM

Long Short-Term Memory

Sepp Hochreiter
Fakultät für Informatik, Technische Universität München, 80290 München, Germany

Jürgen Schmidhuber
IDSIA, Corso Elvezia 36, 6900 Lugano, Switzerland

Learning to store information over extended time intervals by recurrent backpropagation takes a very long time, mostly because of insufficient, decaying error backflow. We briefly review Hochreiter's (1991) analysis of this problem, then address it by introducing a novel, efficient, gradient-based method called long short-term memory (LSTM). Truncating the gradient where this does not do harm, LSTM can learn to bridge minimal time lags in excess of 1000 discrete-time steps by enforcing constant error flow through constant error carousels within special units. Multiplicative gate units learn to open and close access to the constant error flow. LSTM is local in space and time; its computational complexity per time step and weight is $O(1)$. Our experiments with artificial data involve local, distributed, real-valued, and noisy pattern representations. In comparisons with real-time recurrent learning, back propagation through time, recurrent cascade correlation, Elman nets, and neural sequence chunking, LSTM leads to many more successful runs, and learns much faster. LSTM also solves complex, artificial long-time-lag tasks that have never been solved by previous recurrent network algorithms.

1 Introduction

In principle, recurrent networks can use their feedback connections to store representations of recent input events in the form of activations (short-term memory, as opposed to long-term memory embodied by slowly changing weights). This is potentially significant for many applications, including speech processing, non-Markovian control, and music composition (Mozer, 1992). The most widely used algorithms for learning what to put in short-term memory, however, take too much time or do not work well at all, especially when minimal time lags between inputs and corresponding teacher signals are long. Although theoretically fascinating, existing methods do not provide clear practical advantages over, say, backpropagation in feed-forward nets with limited time windows. This article reviews an analysis of the problem and suggests a remedy.

Neural Computation **9, 1735–1780** (1997)

Enhancement of the neural networks

- Many enhancements where developed to make neural networks perform better and solve new problems
  - ReLU
  - BatchNorm
  - Dropout
  - Weight initialization
  - And many more

Rectified linear units

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th international conference on machine learning (ICML-10) (pp. 807-814).

Batch normalization

Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning (pp. 448-456). pmlr.

Dropout

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958.

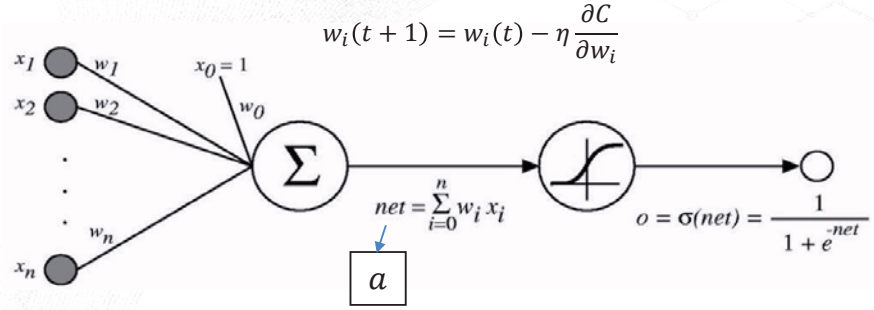
He initialization

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034).

Vanishing gradient

- However, the gradient can decrease exponentially by multiplying of the small numbers.

$$w_i(t+1) = w_i(t) - \eta \frac{\partial C}{\partial w_i}$$

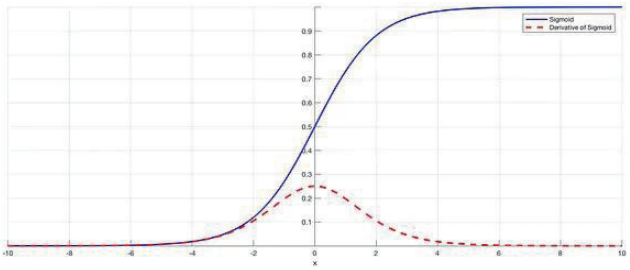


$$\frac{\partial a}{\partial w_i} = x_i \qquad \frac{\partial o}{\partial z} = o(1-o) \qquad 0 \le o \le 1$$

$$\frac{\partial C}{\partial w_i} = \sum_{n=1}^{N} \frac{\partial a}{\partial w_i}\frac{\partial o}{\partial a}\frac{\partial C}{\partial o} = \sum_{n=1}^{N} x_i o(1-o)(o-y)$$

- Assume we have the largest gradient;

E.g. for 10 layer: $0.25^{10} \approx 10^{-6}$



https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e

ReLU (Rectified Linear Unit) (2010)

Rectified Linear Units Improve Restricted Boltzmann Machines

Vinod Nair VNAIR@CS.TORONTO.EDU
Geoffrey E. Hinton HINTON@CS.TORONTO.EDU
Department of Computer Science, University of Toronto, Toronto, ON M5S 2G4, Canada

Abstract

Restricted Boltzmann machines were developed using binary stochastic hidden units. These can be generalized by replacing each binary unit by an infinite number of copies that all have the same weights but have progressively more negative biases. The learning and inference rules for these "Stepped Sigmoid Units" are unchanged. They can be approximated efficiently by noisy, rectified linear units. Compared with binary units, these units learn features that are better for object recognition on the NORB dataset and face verification on the Labeled Faces in the Wild dataset. Unlike binary units, rectified linear units preserve information about relative intensities as information travels through multiple layers of feature detectors.

1. Introduction

Restricted Boltzmann machines (RBMs) have been used as generative models of many different types of data including labeled or unlabeled images (Hinton et al., 2006), sequences of mel-cepstral coefficients that represent speech (Mohamed & Hinton, 2010), bags of words that represent documents (Salakhutdinov & Hinton, 2009), and user ratings of movies (Salakhutdinov et al., 2007). In their conditional form they can be used to model high-dimensional temporal sequences such as video or motion capture data (Taylor et al., 2006). Their most important use is as learning modules that are composed to form deep belief nets (Hinton et al., 2006).

Appearing in *Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel, 2010.

1.1. Learning a Restricted Boltzmann Machine

Images composed of binary pixels can be modeled by an RBM that uses a layer of binary hidden units (feature detectors) to model the higher-order correlations between pixels. If there are no direct interactions between the hidden units and no direct interactions between the visible units that represent the pixels, there is a simple and efficient method called "Contrastive Divergence" to learn a good set of feature detectors from a set of training images (Hinton, 2002). We start with small, random weights on the symmetric connections between each pixel i and each feature detector j. Then we repeatedly update each weight, w_{ij}, using the difference between two measured, pairwise correlations

$$\Delta w_{ij} = \epsilon(<v_i h_j>_{data} - <v_i h_j>_{recon}) \quad (1)$$

where ϵ is a learning rate, $<v_i h_j>_{data}$ is the frequency with which visible unit i and hidden unit j are on together when the feature detectors are being driven by images from the training set and $<v_i h_j>_{recon}$ is the corresponding frequency when the hidden units are being driven by reconstructed images. A similar learning rule can be used for the biases.

Given a training image, we set the binary state, h_j, of each feature detector to be 1 with probability

$$p(h_j = 1) = \frac{1}{1 + \exp(-b_j - \sum_{i \in \text{vis}} v_i w_{ij})} \quad (2)$$

where b_j is the bias of j and v_i is the binary state of pixel i. Once binary states have been chosen for the hidden units we produce a "reconstruction" of the training image by setting the state of each pixel to be 1 with probability

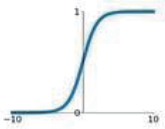
$$p(v_i = 1) = \frac{1}{1 + \exp(-b_i - \sum_{j \in \text{hid}} h_j w_{ij})} \quad (3)$$

The learned weights and biases implicitly define a probability distribution over all possible binary images via the energy, $E(\mathbf{v}, \mathbf{h})$, of a joint configuration of the

Activation function

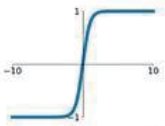
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



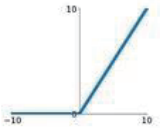
tanh

$\tanh(x)$



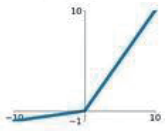
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

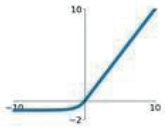


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



ReLU

- ReLU is probably the most popular activation function (simple to compute, fast, good results)

- But ReLU neurons might "die" during training
- Not necessarily bad, can be considered as a form of regularization
- compared to sigmoid/Tanh, ReLU suffers less from vanishing gradient problem but can more easily "explode"

Overfitting

- Collecting more data would be helpful to reduce overfitting
- Also, data augmentation is helpful (E.g. for images: rotation, crop, translation, an so on)
- In addition, reducing the model capacity is necessary.
  - Smaller architecture: fewer hidden layers & hidden nodes, dropout, etc.
  - Smaller weights: early stopping, norm penalties
  - Adding noise
  - Etc.

Regularization for neural networks

$$L2 - Regularized - Cost_{\mathbf{w},\mathbf{b}} = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(y^{[i]},\hat{y}^{[i]}) + \frac{\lambda}{n}\sum_{l=1}^{L}||\mathbf{w}^{(l)}||_F^2$$

- Regular gradient descent update:

$$w_{i,j} := w_{i,j} - \eta\frac{\partial\mathcal{L}}{\partial w_{i,j}}$$

- Gradient descent update with L2 regularization:

$$w_{i,j} := w_{i,j} - \eta\left(\frac{\partial\mathcal{L}}{\partial w_{i,j}} + \frac{2\lambda}{n}w_{i,j}\right)$$

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava NITISH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Alex Krizhevsky KRIZ@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
10 Kings College Road, Rm 3302
Toronto, Ontario, M5S 3G4, Canada.

Editor: Yoshua Bengio

Abstract

Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time. Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different "thinned" networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods. We show that dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology, obtaining state-of-the-art results on many benchmark data sets.

Keywords: neural networks, regularization, model combination, deep learning

1. Introduction

Deep neural networks contain multiple non-linear hidden layers and this makes them very expressive models that can learn very complicated relationships between their inputs and outputs. With limited training data, however, many of these complicated relationships will be the result of sampling noise, so they will exist in the training set but not in real test data even if it is drawn from the same distribution. This leads to overfitting and many methods have been developed for reducing it. These include stopping the training as soon as performance on a validation set starts to get worse, introducing weight penalties of various kinds such as L1 and L2 regularization and soft weight sharing (Nowlan and Hinton, 1992).

With unlimited computation, the best way to "regularize" a fixed-sized model is to average the predictions of all possible settings of the parameters, weighting each setting by

Improving neural networks by preventing co-adaptation of feature detectors

G. E. Hinton*, N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov
Department of Computer Science, University of Toronto,
6 King's College Rd, Toronto, Ontario M5S 3G4, Canada

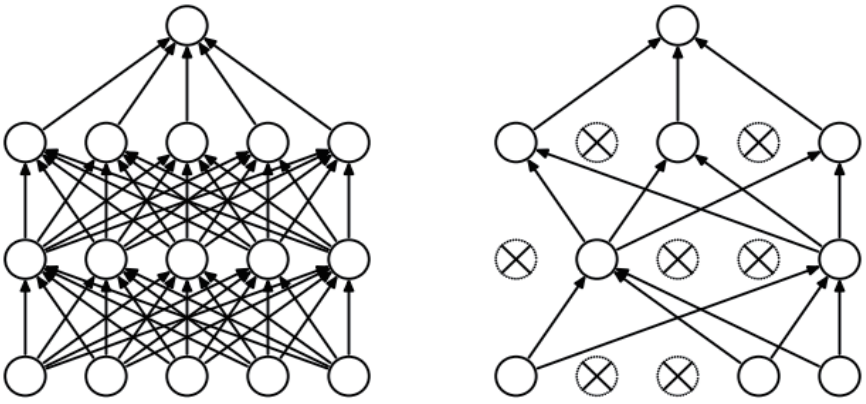
*To whom correspondence should be addressed; E-mail: hinton@cs.toronto.edu

When a large feedforward neural network is trained on a small training set, it typically performs poorly on held-out test data. This "overfitting" is greatly reduced by randomly omitting half of the feature detectors on each training case. This prevents complex co-adaptations in which a feature detector is only helpful in the context of several other specific feature detectors. Instead, each neuron learns to detect a feature that is generally helpful for producing the correct answer given the combinatorially large variety of internal contexts in which it must operate. Random "dropout" gives big improvements on many benchmark tasks and sets new records for speech and object recognition.

A feedforward, artificial neural network uses layers of non-linear "hidden" units between its inputs and its outputs. By adapting the weights on the incoming connections of these hidden units it learns feature detectors that enable it to predict the correct output when given an input vector (*1*). If the relationship between the input and the correct output is complicated and the network has enough hidden units to model it accurately, there will typically be many different settings of the weights that can model the training set almost perfectly, especially if there is only a limited amount of labeled training data. Each of these weight vectors will make different predictions on held-out test data and almost all of them will do worse on the test data than on the training data because the feature detectors have been tuned to work well together on the training data but not on the test data.

Overfitting can be reduced by using "dropout" to prevent complex co-adaptations on the training data. On each presentation of each training case, each hidden unit is randomly omitted from the network with a probability of 0.5, so a hidden unit cannot rely on other hidden units being present. Another way to view the dropout procedure is as a very efficient way of performing model averaging with neural networks. A good way to reduce the error on the test set is to average the predictions produced by a very large number of different networks. The standard

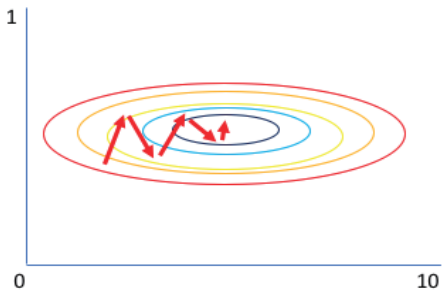
Dropout



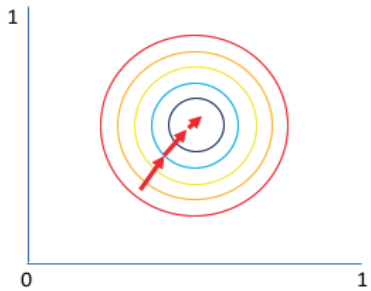
(a) Standard Neural Net

(b) After applying dropout.

Why do we normalize inputs for gradient descent?



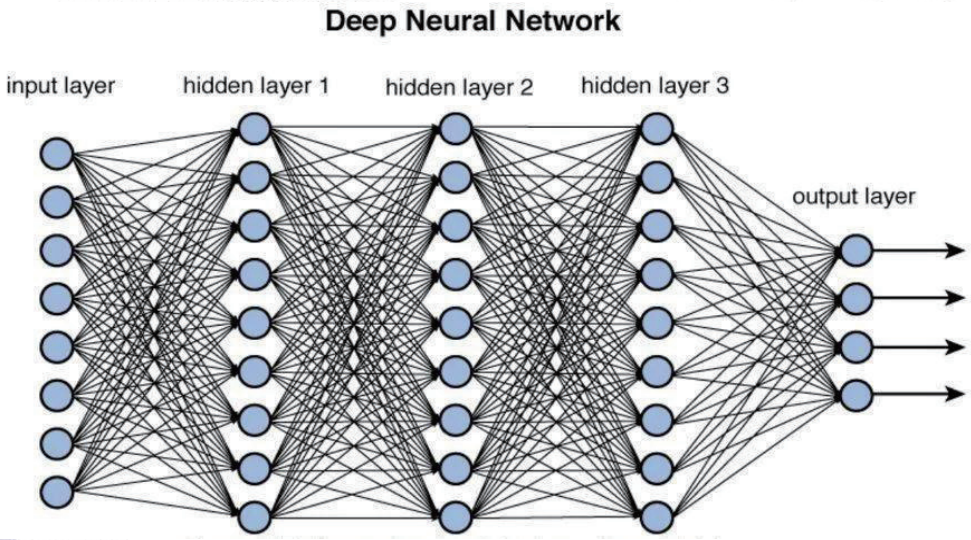
Gradient of larger parameter dominates the update



Both parameters can be updated in equal proportions

https://www.jeremyjordan.me/batch-normalization/

- However, normalizing the inputs only affects the first hidden layer.
- Is it enough? What about the other hidden layers?



https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe
Google Inc., *sioffe@google.com*

Christian Szegedy
Google Inc., *szegedy@google.com*

Abstract

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout. Applied to a state-of-the-art image classification model, Batch Normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by a significant margin. Using an ensemble of batch-normalized networks, we improve upon the best published result on ImageNet classification: reaching 4.9% top-5 validation error (and 4.8% test error), exceeding the accuracy of human raters.

1 Introduction

Deep learning has dramatically advanced the state of the art in vision, speech, and many other areas. Stochastic gradient descent (SGD) has proved to be an effective way of training deep networks, and SGD variants such as momentum (Sutskever et al., 2013) and Adagrad (Duchi et al., 2011) have been used to achieve state of the art performance. SGD optimizes the parameters Θ of the network, so as to minimize the loss

$$\Theta = \arg\min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} \ell(x_i, \Theta)$$

where $x_{1...N}$ is the training data set. With SGD, the training proceeds in steps, and at each step we consider a *mini-batch* $x_{1...m}$ of size m. The mini-batch is used to approximate the gradient of the loss function with respect to the parameters, by computing

$$\frac{1}{m} \frac{\partial \ell(x_i, \Theta)}{\partial \Theta}.$$

Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a batch can be much more efficient than m computations for individual examples, due to the parallelism afforded by the modern computing platforms.

While stochastic gradient is simple and effective, it requires careful tuning of the model hyper-parameters, specifically the learning rate used in optimization, as well as the initial values for the model parameters. The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers – so that small changes to the network parameters amplify as the network becomes deeper.

The change in the distributions of layers' inputs presents a problem because the layers need to continuously adapt to the new distribution. When the input distribution to a learning system changes, it is said to experience *covariate shift* (Shimodaira, 2000). This is typically handled via domain adaptation (Jiang, 2008). However, the notion of covariate shift can be extended beyond the learning system as a whole, to apply to its parts, such as a sub-network or a layer. Consider a network computing

$$\ell = F_2(F_1(u, \Theta_1), \Theta_2)$$

where F_1 and F_2 are arbitrary transformations, and the parameters Θ_1, Θ_2 are to be learned so as to minimize the loss ℓ. Learning Θ_2 can be viewed as if the inputs $x = F_1(u, \Theta_1)$ are fed into the sub-network

$$\ell = F_2(x, \Theta_2).$$

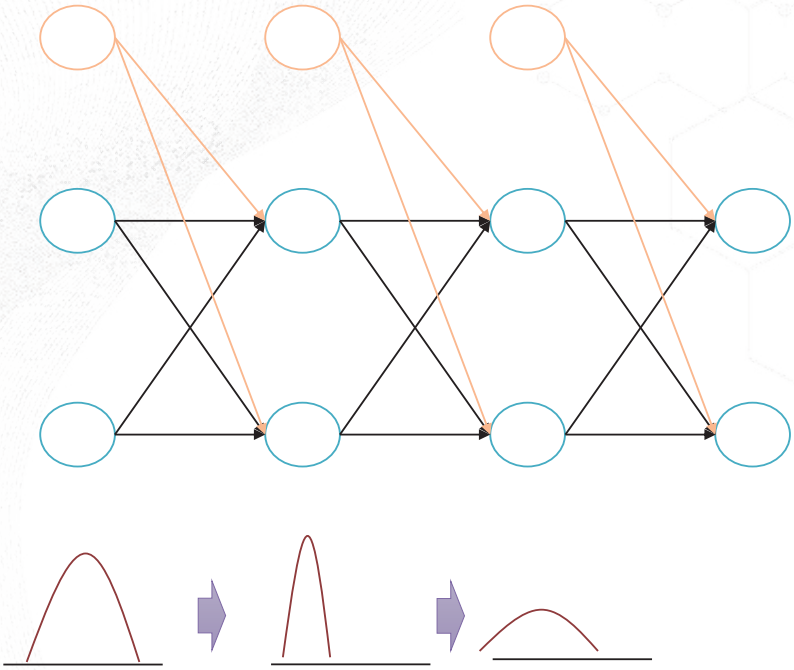
For example, a gradient descent step

$$\Theta_2 \leftarrow \Theta_2 - \frac{\alpha}{m} \sum_{i=1}^{m} \frac{\partial F_2(x_i, \Theta_2)}{\partial \Theta_2}$$

(for batch size m and learning rate α) is exactly equivalent to that for a stand-alone network F_2 with input x. Therefore, the input distribution properties that make training more efficient – such as having the same distribution between the training and test data – apply to training the sub-network as well. As such it is advantageous for the distribution of x to remain fixed over time. Then, Θ_2 does

1

Internal covariant shift



Batch normalization

- Normalization of inputs for hidden layers
- Increasing training speed
- Reducing sensitivity for weight initialization and hyperparameter setting
- Regularization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

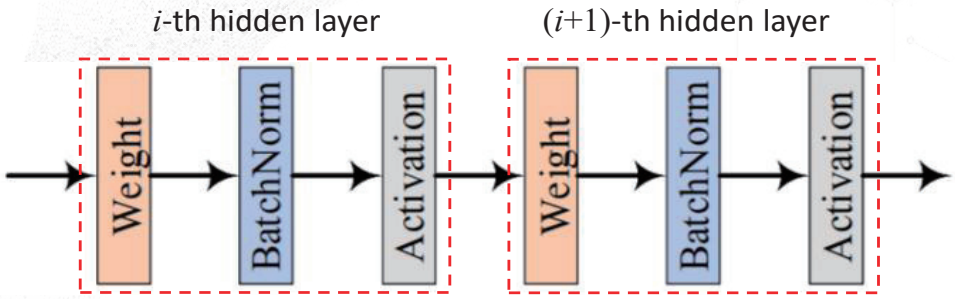
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

- r and β are learnable parameters.

Batch normalization



How Does Batch Normalization Help Optimization?

Shibani Santurkar* MIT shibani@mit.edu
Dimitris Tsipras* MIT tsipras@mit.edu
Andrew Ilyas* MIT ailyas@mit.edu
Aleksander Mądry MIT madry@mit.edu

Abstract

Batch Normalization (BatchNorm) is a widely adopted technique that enables faster and more stable training of deep neural networks (DNNs). Despite its pervasiveness, the exact reasons for BatchNorm's effectiveness are still poorly understood. The popular belief is that this effectiveness stems from controlling the change of the layers' input distributions during training to reduce the so-called "internal covariate shift". In this work, we demonstrate that such distributional stability of layer inputs has little to do with the success of BatchNorm. Instead, we uncover a more fundamental impact of BatchNorm on the training process: it makes the optimization landscape significantly smoother. This smoothness induces a more predictive and stable behavior of the gradients, allowing for faster training.

1 Introduction

Over the last decade, deep learning has made impressive progress on a variety of notoriously difficult tasks in computer vision [16, 7], speech recognition [5], machine translation [29], and game-playing [18, 25]. This progress hinged on a number of major advances in terms of hardware, datasets [15, 23], and algorithmic and architectural techniques [27, 12, 20, 28]. One of the most prominent examples of such advances was batch normalization (BatchNorm) [10].

At a high level, BatchNorm is a technique that aims to improve the training of neural networks by stabilizing the distributions of layer inputs. This is achieved by introducing additional network layers that control the first two moments (mean and variance) of these distributions.

The practical success of BatchNorm is indisputable. By now, it is used by default in most deep learning models, both in research (more than 6,000 citations) and real-world settings. Somewhat shockingly, however, despite its prominence, we still have a poor understanding of what the effectiveness of BatchNorm is stemming from. In fact, there are now a number of works that provide alternatives to BatchNorm [1, 3, 13, 31], but none of them seem to bring us any closer to understanding this issue. (A similar point was also raised recently in [22].)

Currently, the most widely accepted explanation of BatchNorm's success, as well as its original motivation, relates to so-called *internal covariate shift* (ICS). Informally, ICS refers to the change in the distribution of layer inputs caused by updates to the preceding layers. It is conjectured that such continual change negatively impacts training. The goal of BatchNorm was to reduce ICS and thus remedy this effect.

Even though this explanation is widely accepted, we seem to have little concrete evidence supporting it. In particular, we still do not understand the link between ICS and training performance. The chief goal of this paper is to address all these shortcomings. Our exploration lead to somewhat startling discoveries.

*Equal contribution.

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

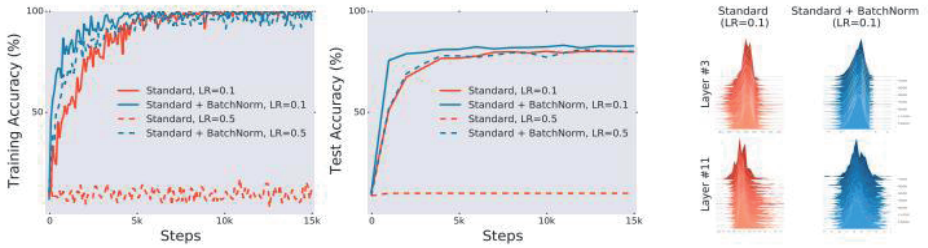


Figure 1: Comparison of (a) training (optimization) and (b) test (generalization) performance of a standard VGG network trained on CIFAR-10 with and without BatchNorm (details in Appendix A). There is a consistent gain in training speed in models with BatchNorm layers. (c) Even though the gap between the performance of the BatchNorm and non-BatchNorm networks is clear, the difference in the evolution of layer input distributions seems to be much less pronounced. (Here, we sampled activations of a given layer and visualized their distribution over training steps.)

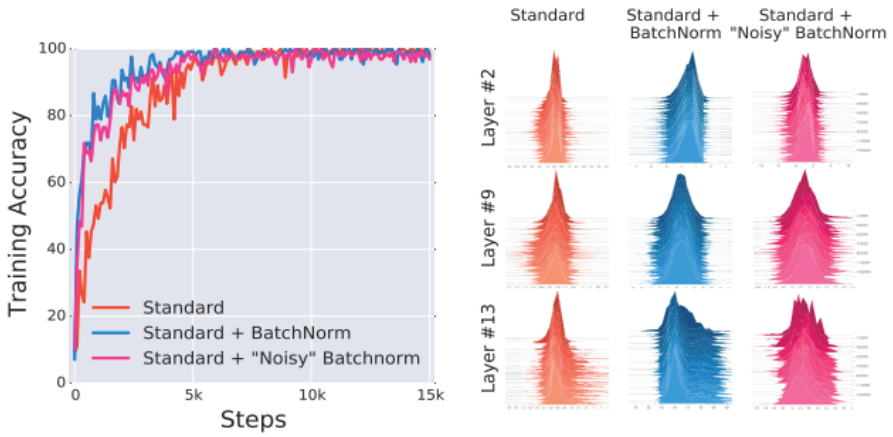


Figure 2: Connections between distributional stability and BatchNorm performance: We compare VGG networks trained without BatchNorm (Standard), with BatchNorm (Standard + BatchNorm) and with explicit "covariate shift" added to BatchNorm layers (Standard + "Noisy" BatchNorm). In the later case, we induce distributional instability by adding *time-varying*, *non-zero* mean and *non-unit* variance noise independently to each batch normalized activation. The "noisy" BatchNorm model nearly matches the performance of standard BatchNorm model, despite complete distributional instability. We sampled activations of a given layer and visualized their distributions (also cf. Figure 7).

(a) loss landscape (b) gradient predictiveness (c) "effective" β-smoothness

Figure 4: Analysis of the optimization landscape of VGG networks. At a particular training step, we measure the variation (shaded region) in loss (a) and ℓ_2 changes in the gradient (b) as we move in the gradient direction. The "effective" β-smoothness (c) refers to the maximum difference (in ℓ_2-norm) in gradient over distance moved in that direction. There is a clear improvement in all of these measures in networks with BatchNorm, indicating a more well-behaved loss landscape. (Here, we cap the maximum distance to be $\eta = 0.4\times$ the gradient since for larger steps the standard network just performs worse (see Figure 1). BatchNorm however continues to provide smoothing for even larger distances.) Note that these results are supported by our theoretical findings (Section 4).

Weight initialization

- What happens if we initialize all $\mathbf{W}=0$?
  - All outputs are 0, all gradients are the same! No “symmetry breaking”

- What about small random numbers?
  - Eg.
    - Random sampling from a uniform distribution in range [0,1] or [-0.5, 0.5]
    - Random sampling from a Gaussian distribution with zero mean and small variance (0.1 or 0.01?)

  - It works okay for small networks, but problems with deeper networks.

Understanding the difficulty of training deep feedforward neural networks

Xavier Glorot **Yoshua Bengio**
DIRO, Université de Montréal, Montréal, Québec, Canada

Abstract

Whereas before 2006 it appears that deep multi-layer neural networks were not successfully trained, since then several algorithms have been shown to successfully train them, with experimental results showing the superiority of deeper vs less deep architectures. All these experimental results were obtained with new initialization or training mechanisms. Our objective here is to understand better why standard gradient descent from random initialization is doing so poorly with deep neural networks, to better understand these recent relative successes and help design better algorithms in the future. We first observe the influence of the non-linear activations functions. We find that the logistic sigmoid activation is unsuited for deep networks with random initialization because of its mean value, which can drive especially the top hidden layer into saturation. Surprisingly, we find that saturated units can move out of saturation by themselves, albeit slowly, and explaining the plateaus sometimes seen when training neural networks. We find that a new non-linearity that saturates less can often be beneficial. Finally, we study how activations and gradients vary across layers and during training, with the idea that training may be more difficult when the singular values of the Jacobian associated with each layer are far from 1. Based on these considerations, we propose a new initialization scheme that brings substantially faster convergence.

1 Deep Neural Networks

Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. They include learning methods for a wide array of *deep architectures*, including neural networks with many hidden layers (Vincent et al., 2008) and graphical models with many levels of hidden variables (Hinton et al., 2006), among others (Zhu et al., 2009; Weston et al., 2008). Much attention has recently been devoted to them (see (Bengio, 2009) for a review), because of their theoretical appeal, inspiration from biology and human cognition, and because of empirical success in vision (Ranzato et al., 2007; Larochelle et al., 2007; Vincent et al., 2008) and natural language processing (NLP) (Collobert & Weston, 2008; Mnih & Hinton, 2009). Theoretical results reviewed and discussed by Bengio (2009), suggest that in order to learn the kind of complicated functions that can represent high-level abstractions (e.g. in vision, language, and other AI-level tasks), one may need *deep architectures*.

Most of the recent experimental results with deep architecture are obtained with models that can be turned into deep supervised neural networks, but with initialization or training schemes different from the classical feedforward neural networks (Rumelhart et al., 1986). Why are these new algorithms working so much better than the standard random initialization and gradient-based optimization of a supervised training criterion? Part of the answer may be found in recent analyses of the effect of unsupervised pre-training (Erhan et al., 2009), showing that it acts as a regularizer that initializes the parameters in a “better” basin of attraction of the optimization procedure, corresponding to an apparent local minimum associated with better generalization. But earlier work (Bengio et al., 2007) had shown that even a purely supervised but greedy layer-wise procedure would give better results. So here instead of focusing on what unsupervised pre-training or semi-supervised criteria bring to deep architectures, we focus on analyzing what may be going wrong with good old (but deep) multi-layer neural networks.

Our analysis is driven by investigative experiments to monitor activations (watching for saturation of hidden units) and gradients, across layers and across training iterations. We also evaluate the effects on these of choices of activation function (with the idea that it might affect saturation) and initialization procedure (since unsupervised pre-training is a particular form of initialization and it has a drastic impact).

Appearing in Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy. Volume 9 of JMLR: W&CP 9.

Xavier initialization

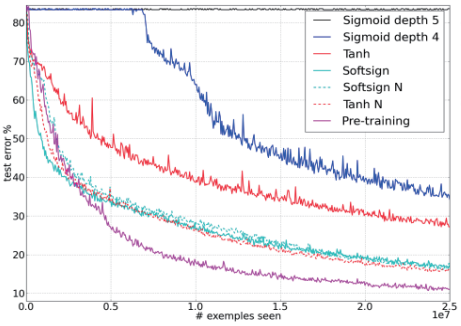


Figure 11: *Test error during online training on the* **Shapeset-**3×2 *dataset, for various activation functions and initialization schemes (ordered from top to bottom in decreasing final error). N after the activation function name indicates the use of normalized initialization.*

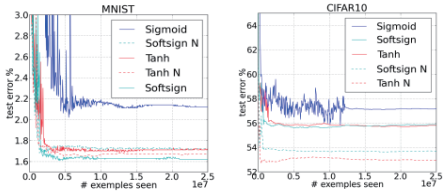


Figure 12: *Test error curves during training on MNIST and CIFAR10, for various activation functions and initialization schemes (ordered from top to bottom in decreasing final error). N after the activation function name indicates the use of normalized initialization.*

Xavier initialization

- Initialize weights from Gaussian or uniform distribution
- Then, scale the weights proportional to the number of inputs to the layer

$$\mathbf{W}^{(l)} := \mathbf{W}^{(l)} \cdot \sqrt{\frac{1}{m^{(l-1)}}}$$

where m is the number of input units to the next layer

Sometimes, "fan in" + "fan out" are used

e.g.,

$$W_{i,j}{}^{(l)} \sim N(\mu = 0, \sigma^2 = 0.01)$$

(or uniform distr. in a fixed interval, as in the original paper)

Xavier initialization

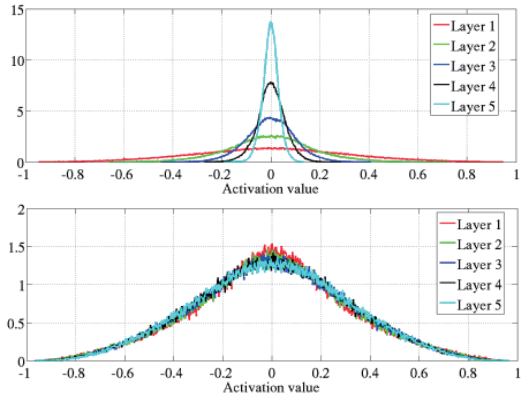


Figure 6: *Activation values normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized initialization (bottom). Top: 0-peak increases for higher layers.*

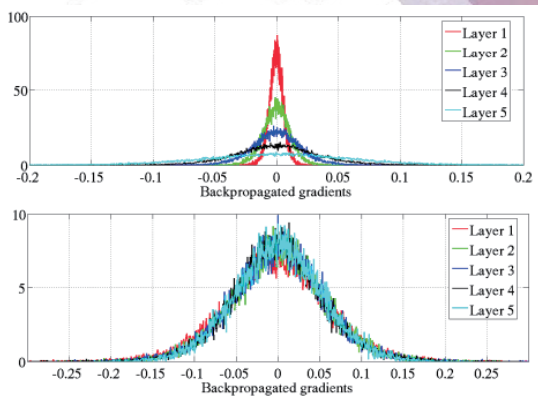


Figure 7: *Back-propagated gradients normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized (bottom) initialization. Top: 0-peak decreases for higher layers.*

Delving Deep into Rectifiers:
Surpassing Human-Level Performance on ImageNet Classification

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research

Abstract

Rectified activation units (rectifiers) are essential for state-of-the-art neural networks. In this work, we study rectifier neural networks for image classification from two aspects. First, we propose a Parametric Rectified Linear Unit (PReLU) that generalizes the traditional rectified unit. PReLU improves model fitting with nearly zero extra computational cost and little overfitting risk. Second, we derive a robust initialization method that particularly considers the rectifier nonlinearities. This method enables us to train extremely deep rectified models directly from scratch and to investigate deeper or wider network architectures. Based on the learnable activation and advanced initialization, we achieve ***4.94%*** *top-5 test error on the ImageNet 2012 classification dataset. This is a 26% relative improvement over the ILSVRC 2014 winner (GoogLeNet, 6.66% [33]). To our knowledge, our result is the first*[1] *to surpass the reported human-level performance (5.1%, [26]) on this dataset.*

1. Introduction

Convolutional neural networks (CNNs) [19, 18] have demonstrated recognition accuracy better than or comparable to humans in several visual recognition tasks, including recognizing traffic signs [3], faces [34, 32], and handwritten digits [3, 36]. In this work, we present a result that surpasses the human-level performance reported by [26] on a more generic and challenging recognition task - the classification task in the 1000-class ImageNet dataset [26].

In the last few years, we have witnessed tremendous improvements in recognition performance, mainly due to advances in two technical directions: building more powerful models, and designing effective strategies against overfitting. On one hand, neural networks are becoming more capable of fitting training data, because of increased complexity (*e.g.*, increased depth [29, 33], enlarged width [37, 28], and the use of smaller strides [37, 28, 2, 29]), new nonlinear activations [24, 23, 38, 22, 31, 10], and sophisticated layer designs [33, 12]. On the other hand, better generalization is achieved by effective regularization techniques [13, 30, 10, 36], aggressive data augmentation [18, 14, 29, 33], and large-scale data [4, 26].

Among these advances, the rectifier neuron [24, 9, 23, 38], *e.g.*, Rectified Linear Unit (ReLU), is one of several keys to the recent success of deep networks [18]. It expedites convergence of the training procedure [18] and leads to better solutions [24, 9, 23, 38] than conventional sigmoid-like units. Despite the prevalence of rectifier networks, recent improvements of models [37, 28, 12, 29, 33] and theoretical guidelines for training them [8, 27] have rarely focused on the properties of the rectifiers.

Unlike traditional sigmoid-like units, *ReLU is not a symmetric function*. As a consequence, the mean response of ReLU is always no smaller than zero; besides, even assuming the inputs/weights are subject to symmetric distributions, the distributions of responses can still be asymmetric because of the behavior of ReLU. These properties of ReLU influence the theoretical analysis of convergence and empirical performance, as we will demonstrate.

In this paper, we investigate neural networks from two aspects particularly driven by the rectifier properties. First, we propose a new extension of ReLU, which we call *Parametric Rectified Linear Unit* (PReLU). This activation function adaptively learns the parameters of the rectifiers, and improves accuracy at negligible extra computational cost. Second, we study the difficulty of training rectified models that are very deep. By explicitly modeling the nonlinearity of rectifiers (ReLU/PReLU), we derive a theoretically sound initialization method, which helps with convergence of very deep models (*e.g.*, with 30 weight layers) trained directly from scratch. This gives us more flexibility to explore more powerful network architectures.

On the 1000-class ImageNet 2012 dataset, our network leads to a single-model result of 5.71% top-5 error, which surpasses all multi-model results in ILSVRC 2014. Further, our multi-model result achieves **4.94%** top-5 error on the test set, which is a 26% relative improvement over the ILSVRC 2014 winner (GoogLeNet, 6.66% [33]). To the best of our knowledge, our result surpasses for the first time the reported human-level performance (5.1% in [26]) of a dedicated individual labeler on this recognition challenge.

[1] reported in Feb. 2015.

He initialization

- Assuming activations with mean zero at Xavier Initialization (which is reasonable for tanH)
- For ReLU, this is different, as the activations are not centered at zero.
- He initialization added a scaling factor of $2^{0.5}$

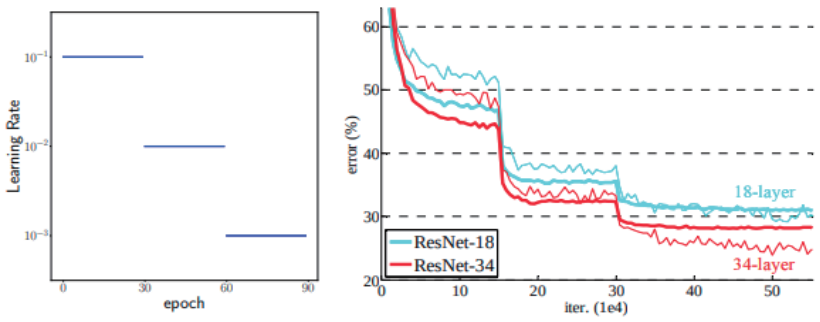
$$\mathbf{W}^{(l)} := \mathbf{W}^{(l)} \cdot \sqrt{\frac{2}{m^{[l-1]}}}$$

For Leaky ReLU with negative slope alpha:

$$\mathbf{W}^{(l)} := \mathbf{W}^{(l)} \cdot \sqrt{\frac{2}{(1+\alpha^2) \cdot m^{[l-1]}}}$$

Learning rate decay

- Learning rate decay is a technique in which a network starts training with a large learning rate and then slowly decreases it.



(a) Learning rate decay strategy (b) Figure taken from He et al. (2016)

Figure 1: Training error in (b) is shown by thin curves, while test error in (b) by bold curves.

https://arxiv.org/pdf/1908.01878

Momentum

$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}})$$

weight increment — learning rate — weight gradient

$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}}) + (\gamma * \Delta w_{ij}^{t-1})$$

momentum factor — weight increment, previous iteration

PERGAMON

Neural Networks 12 (1999) 145–151

Neural Networks

Contributed Article

On the momentum term in gradient descent learning algorithms

Ning Qian[1]

Center for Neurobiology and Behavior, Columbia University, 722 W. 168th Street, New York, NY 10032, USA

Received 4 November 1997; revised 6 August 1998; accepted 6 August 1998

Abstract

A momentum term is usually included in the simulations of connectionist learning algorithms. Although it is well known that such a term greatly improves the speed of learning, there have been few rigorous studies of its mechanisms. In this paper, I show that in the limit of continuous time, the momentum parameter is analogous to the mass of Newtonian particles that move through a viscous medium in a conservative force field. The behavior of the system near a local minimum is equivalent to a set of coupled and damped harmonic oscillators. The momentum term improves the speed of convergence by bringing some eigen components of the system closer to critical damping. Similar results can be obtained for the discrete time case used in computer simulations. In particular, I derive the bounds for convergence on learning-rate and momentum parameters, and demonstrate that the momentum term can increase the range of learning rate over which the system converges. The optimal condition for convergence is also analyzed. © 1999 Elsevier Science Ltd. All rights reserved.

Keywords: Momentum; Gradient descent learning algorithm; Damped harmonic oscillator; Critical damping; Learning rate; Speed of convergence

1. Introduction

Connectionist neural network models have been successfully applied to a wide range of problems (Rumelhart and McClelland, 1986; McClelland and Rumelhart, 1986; Anderson et al., 1990; Churchland and Sejnowski, 1992). Although there are many different varieties of learning algorithms available, the majority of them—including the popular back-propagation learning algorithm—are of the gradient descent type. For a given network architecture, one usually starts with an error function which is parameterized by the weights (the connection strengths between units) in the network. The gradient of the error function with respect to each weight is then computed and the weights are modified along the downhill direction of the gradient in order to reduce the error. Let $E(\mathbf{w})$ be the error function, where $\mathbf{w}$ is a vector representing all the weights in the network, the simplest gradient descent algorithm, known as the steepest descent, modifies the weights at time step t according to:

$$\Delta \mathbf{w}_t = -\epsilon \nabla_w E(\mathbf{w}_t) \quad (1)$$

where ∇_w represents the gradient operator with respect to the weights, and ϵ is a small positive number known as the learning rate.

It is well known that such a learning scheme can be very slow. The inclusion of a momentum term has been found to increase the rate of convergence dramatically (Rumelhart et al., 1986). With this method, Eq. (1) takes the form:

$$\Delta \mathbf{w}_t = -\epsilon \nabla_w E(\mathbf{w}) + p \Delta \mathbf{w}_{t-1} \quad (2)$$

where p is the momentum parameter. That is, the modification of the weight vector at the current time step depends on both the current gradient and the weight change of the previous step. Intuitively, the rationale for the use of the momentum term is that the steepest descent is particularly slow when there is a long and narrow valley in the error function surface. In this situation, the direction of the gradient is almost perpendicular to the long axis of the valley. The system thus oscillates back and forth in the direction of the short axis, and only moves very slowly along the long axis of the valley. The momentum term helps average out the oscillation along the short axis while at the same time adds up contributions along the long axis (Rumelhart et al., 1986).

Other methods have also been proposed for improving the speed of convergence of gradient descent learning algorithms. For example, the conjugate gradient method has been shown to be superior to the steepest descent in most

[1] Tel.: +1-212-543-5213; Fax: +1-212-543-5161; E-mail: nq6@columbia.edu

On the importance of initialization and momentum in deep learning

Ilya Sutskever[1] ILYASU@GOOGLE.COM
James Martens JMARTENS@CS.TORONTO.EDU
George Dahl GDAHL@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU

Abstract

Deep and recurrent neural networks (DNNs and RNNs respectively) are powerful models that were considered to be almost impossible to train using stochastic gradient descent with momentum. In this paper, we show that when stochastic gradient descent with momentum uses a well-designed random initialization and a particular type of slowly increasing schedule for the momentum parameter, it can train both DNNs and RNNs (on datasets with long-term dependencies) to levels of performance that were previously achievable only with Hessian-Free optimization. We find that both the initialization and the momentum are crucial since poorly initialized networks cannot be trained with momentum and well-initialized networks perform markedly worse when the momentum is absent or poorly tuned.

Our success training these models suggests that previous attempts to train deep and recurrent neural networks from random initializations have likely failed due to poor initialization schemes. Furthermore, carefully tuned momentum methods suffice for dealing with the curvature issues in deep and recurrent network training objectives without the need for sophisticated second-order methods.

1. Introduction

Deep and recurrent neural networks (DNNs and RNNs, respectively) are powerful models that achieve high performance on difficult pattern recognition problems in vision, and speech (Krizhevsky et al., 2012; Hinton et al., 2012; Dahl et al., 2012; Graves, 2012).

Although their representational power is appealing, the difficulty of training DNNs has prevented their widespread use until fairly recently. DNNs became the subject of renewed attention following the work of Hinton et al. (2006) who introduced the idea of greedy layerwise pre-training. This approach has since branched into a family of methods (Bengio et al., 2007), all of which train the layers of the DNN in a sequence using an auxiliary objective and then "fine-tune" the entire network with standard optimization methods such as stochastic gradient descent (SGD). More recently, Martens (2010) attracted considerable attention by showing that a type of truncated-Newton method called Hessian-free Optimization (HF) is capable of training DNNs from certain random initializations without the use of pre-training, and can achieve lower errors for the various auto-encoding tasks considered by Hinton & Salakhutdinov (2006).

Recurrent neural networks (RNNs), the temporal analogue of DNNs, are highly expressive sequence models that can model complex sequence relationships. They can be viewed as very deep neural networks that have a "layer" for each time-step with parameter sharing across the layers and, for this reason, they are considered to be even harder to train than DNNs. Recently, Martens & Sutskever (2011) showed that the HF method of Martens (2010) could effectively train RNNs on artificial problems that exhibit very long-range dependencies (Hochreiter & Schmidhuber, 1997). Without resorting to special types of memory units, these problems were considered to be impossibly difficult for first-order optimization methods due to the well known vanishing gradient problem (Bengio et al., 1994). Sutskever et al. (2011) and later Mikolov et al. (2012) then applied HF to train RNNs to perform character-level language modeling and achieved excellent results.

Recently, several results have appeared to challenge the commonly held belief that simpler first-order methods are incapable of learning deep models from random initializations. The work of Glorot & Bengio (2010), Mohamed et al. (2012), and Krizhevsky et al. (2012) reported little difficulty training neural networks with depths up to 8 from certain well-chosen

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

[1] Work was done while the author was at the University of Toronto.

The momentum method (Polyak, 1964), which we refer to as classical momentum (CM), is a technique for accelerating gradient descent that accumulates a velocity vector in directions of persistent reduction in the objective across iterations. Given an objective function $f(\theta)$ to be minimized, classical momentum is given by:

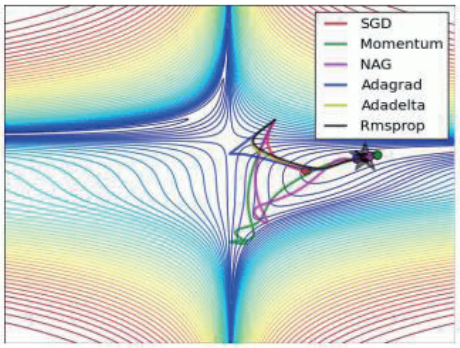
$$v_{t+1} = \mu v_t - \varepsilon \nabla f(\theta_t) \quad (1)$$
$$\theta_{t+1} = \theta_t + v_{t+1} \quad (2)$$

where $\varepsilon > 0$ is the learning rate, $\mu \in [0, 1]$ is the momentum coefficient, and $\nabla f(\theta_t)$ is the gradient at θ_t.

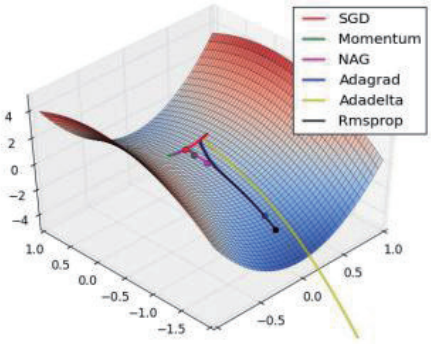
Nesterov's Accelerated Gradient (abbrv. NAG; Nesterov, 1983) has been the subject of much recent attention by the convex optimization community (e.g., Cotter et al., 2011; Lan, 2010). Like momentum, NAG is a first-order optimization method with better convergence rate guarantee than gradient descent in certain situations. In particular, for general smooth (non-strongly) convex functions and a deterministic gradient, NAG achieves a global convergence rate of $O(1/T^2)$ (versus the $O(1/T)$ of gradient descent), with constant proportional to the Lipschitz coefficient of the derivative and the squared Euclidean distance to the solution. While NAG is not typically thought of as a type of momentum, it indeed turns out to be closely related to classical momentum, differing only in the precise update of the velocity vector v, the significance of which we will discuss in the next sub-section. Specifically, as shown in the appendix, the NAG update may be rewritten as:

$$v_{t+1} = \mu v_t - \varepsilon \nabla f(\theta_t + \mu v_t) \quad (3)$$
$$\theta_{t+1} = \theta_t + v_{t+1} \quad (4)$$

Gradient descent optimization algorithms



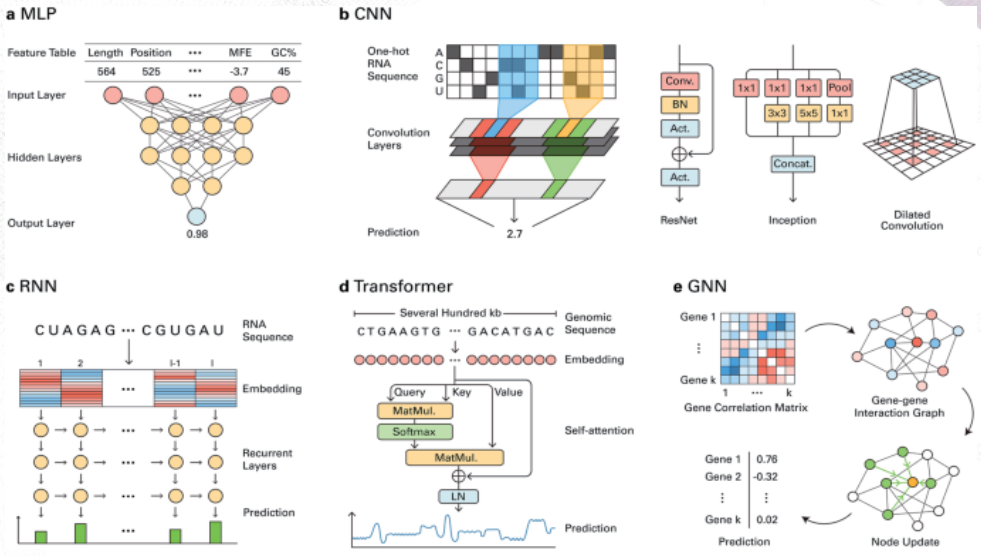
(a) SGD optimization on loss surface contours



(b) SGD optimization on saddle point

https://arxiv.org/pdf/1609.04747

Deep neural network architectures



Exp Mol Med 56, 1293–1321 (2024)

Thank you!