Deep Learning
lecture 1

Yi Wu, IIIS

Spring 2024

Mar-1

About the Course

- (Hopefully) English Teaching
- Friday 1:30 pm to 16:05 pm @ 舜德/经管西楼301
  - Special: 9th lecture will be rearranged. Pay attention to the week before Labor Day Festival (劳动节).
- Offline class required
  - For special cases, contact TA
- Office Hour
  - FIT 1-202, Thu 1pm to 2pm (TA will be always there)
    - **Special: Next week's OH is from 3pm to 4pm**
  - Email me if you want a 1-on-1 conversation with me
- **Use 钉钉 for announcements and questions**

Staff

吴翼

Office: 清华科技园C19

jxwuyi@mails.tsinghua.edu.cn

Research Directions:

Human-AI Interaction; Human-Robot Interaction; Deep Reinforcement Learning

Multi-Agent Learning; Distributed Learning System

Jiaxuan Gao (TA)

高嘉煊

E-mail:

gaojx23@mails.tsinghua.edu.cn

Huaijie Wang (TA)

王怀颉

E-mail:

wang-hx23@mails.tsinghua.edu.cn

Syllabus

- 8 writing homework (due in 1 week) + 4 coding project + final project (2~3 members/team) + final exam

#	Date	Topic	Homework	Note
1	3/1	Overview		
2	3/8	Supervised Learning (1)		CP1 release
3	3/15	Supervised Learning (2)	HW1	CP1 due; CP2 release
4	3/22	Energy-Based Model	HW2	
5	3/29	Normalizing Flow	HW3	CP2 due
6	4/7	Variational Autoencoder	HW4	CP3 release
7	4/12	Generative Adversarial Network	HW5	
8	4/19	Sequence Modeling (1)		CP 4 release
9	**TBA**	**Sequence Modeling (2)**	**HW6**	
10	4/26	Advanced Generative Model & Frontier	HW7	CP3 due; Project proposal due;
11	5/10	Learning Structured Data	HW8	CP4 due
12	**5/24**	Advanced Topics: Self-Supervised Learning, Meta-Learning, AutoML		
13	5/31	Security, Explainability and Fairness		
14	6/7	Frontier and Applications (Guest Lecture)		
15	6/14	Final Project Poster Session		Project report due TBA

A lot of work!

Grading

- Overall: Homework 20% + Coding 40% + Project 20% + Final 20%
  - Coding: 10% each
    - backpropagation; classification; generative models; sequence modeling;
  - Homework:
    - We will convert all the homework scores into 20 pts
    - No late submission allowed
  - Project: 12~15min presentation; grade based on contributions
- Bonus (up to 5% + 中文笔记/调研额外加分)
  - Coding & Project
  - Lecture Notes / Survey (starting from Lecture 4, details announced later)
  - 中文笔记 (达到要求的笔记可以获得额外至多3%的加分)
  - 中文笔记要求：语言通顺，大一同学光看笔记可以看懂 ☺

About the Course

- Prerequisites
  - Linux and Python coding
  - Calculus & basic Machine Learning
  - Mentally ready for **heavy** engineering work ☺

Convergence Analysis of Two-layer Neural Networks with ReLU Activation

Yuanzhi Li
Computer Science Department
Princeton University
yuanzhil@cs.princeton.edu

Yan
Computer Sci
Cornell
yangyuan@c

November 1, 2017

Abstract

In recent years, stochastic gradient descent (SGD) based techniques has
training neural networks. However, formal theoretical understanding of why SG
practice is largely missing.

In this paper, we make progress on understanding this mystery by providing
on a rich subset of two-layer feedforward networks with ReLU activations. T
special structure called "identity mapping". We prove that, if input follows fr
standard $O(1/\sqrt{d})$ initialization of the weights, SGD converges to the global n
of steps. Unlike normal vanilla networks, the "identity mapping" makes our ne
global minimum is unique. To complement our theory, we are also able to show
networks with this mapping have better performance compared with normal va

Our convergence theorem differs from traditional non-convex optimization
converges to optimal in "two phases": In phase I, the gradient points to the wron
function g gradually decreases. Then in phase II, SGD enters a nice one point convex region and converges. We
also show that the identity mapping is necessary for convergence, as it moves the initial point to a better place
for optimization. Experiment verifies our claims.

C Phase I: The Decreasing Potential Function

As we saw in Theorem 3.3, if $\|\mathbf{W}\|_2, \|\mathbf{W}^*\|_2$ is bounded by a constant $\gamma = \frac{1}{100}$, and the potential function $g \le 0.1$, $\mathrm{L}(\mathbf{W})$ is 0.03-one point convex, which will give us convergence guarantee according to Lemma 2.5. However, g could be larger than 0.1 initially, and as we run SGD, $\|\mathbf{W}\|_2$ might be larger than $\frac{1}{100}$ as well.

In this section, we address both problems by analyzing the dynamics of SGD, thus prove Theorem 3.2. The proofs can be found in Appendix H. Before proceeding to the interesting stuff, we need a simpler form of $\nabla \mathrm{L}(\mathbf{W})$

negative gradient of $\mathrm{L}(\mathbf{W})$ *is approximately*

$+ (\mathbf{W}^* - \mathbf{W})^\top - 2\mathrm{Diag}(\mathbf{W}^* - \mathbf{W}) + g\mathbf{I} + \mathbf{W}$

ror is $\|\mathbf{Q}(\mathbf{W}) - [-\nabla \mathrm{L}(\mathbf{W})]\|_2 \le 61\gamma^2$.

l norm.

$\nabla \mathrm{L}(\mathbf{W})\|_2 \le 6d\gamma$.

We use subscript t under each variable to denote its value at the
$_t \triangleq (\mathbf{W}^* - \mathbf{W}_t)u$. We first compute the updating rule for g_t.

≥ 100, $\eta \le \frac{\gamma^2}{Cd^2}$, *then* $|g_{t+1}| \le (1 - 0.95\eta d)|g_t| + 86\eta d\gamma^2 +$

ge, so we also need to compute its updating rule:

$\|s_{t+1}\|_2 \le \left(1 - \eta\frac{(d+1)\gamma}{2}\right)\|s_t\|_2 + \eta(6.61\gamma + 1.03|g_t| + \varepsilon)\sqrt{d}$.

Show that g_t will shrink, conditioned on that $\|\mathbf{W}_t\|_2$ is bounded

$\mathbf{W}^*\|_2 \le \gamma \le \frac{1}{100}, d \ge 100, \eta \le \frac{\gamma^2}{Cd^2}, \varepsilon \le \gamma^2$, *then* $|g_t|$ *will keep*
, until $|g_{t_1}| \le 197\gamma^2$ *for* $t_1 \le \frac{1}{10\eta}$.

ays bounded by γ during the process described in Lemma C.5.

such that if $\|\mathbf{W}_0\|_2, \|\mathbf{W}^*\|_2 \le \gamma_0$, $d \ge 100$, $\eta \le \frac{\gamma^2}{Cd^2}$, $\varepsilon \le \gamma^2$,
always have $\|\mathbf{W}_T\|_2 \le \gamma \le \frac{1}{100}$ *for any* $T > 0$.

and $\|\mathbf{W}_T\|_2 \le \gamma$, which means we are in Phase II. The next
er.

such that if $\|\mathbf{W}_0\|_2, \|\mathbf{W}^*\|_2 \le \gamma_0$, $d \ge 100$, $\eta \le \frac{\gamma^2}{Cd^2}$, $\varepsilon \le \gamma^2$,
then after $|g_{t_1}| \le 197\gamma^2$, *Phase I ends and Phase II starts. That is, for every* $T > t_1$, $\|\mathbf{W}_T\|_2 \le \gamma$ *and* $|g_T| \le 0.1$.

Proof for Theorem 3.2. We immediately get Theorem 3.2 by combining the above three lemmas. They show that g_t will decrease to a small value in Phase I (Lemma C.5), $\|\mathbf{W}_t\|_2$ will keep small during this process (Lemma C.6), and they all keep small afterwards (Lemma C.7). □

- What will you learn?
  - Not about "how to import python packages"!
  - Not about "theorem proving"
- Formal Mission
  - How to train a neural net
    - To solve practical problems

About the Course

- Prerequisites
  - Linux and Python coding
  - Calculus & basic Machine Learning
  - Mentally ready for **heavy** engineering work ☺

Convergence Analysis of Two-layer Neural Networks with ReLU Activation

Yuanzhi Li
Computer Science Department
Princeton University
yuanzhil@cs.princeton.edu

November 1, 2017

Abstract

In recent years, stochastic gradient descent (SGD) based techniques has
training neural networks. However, formal theoretical understanding of why SG
practice is largely missing.

In this paper, we make progress on understanding this mystery by providing
on a rich subset of two-layer feedforward networks with ReLU activations. T
special structure called "identity mapping". We prove that, if input follows fr
standard $O(1/\sqrt{d})$ initialization of the weights, SGD converges to the global m
of steps. Unlike normal vanilla networks, the "identity mapping" makes our ne
global minimum is unique. To complement our theory, we are also able to show
networks with this mapping have better performance compared with normal va

Our convergence theorem differs from traditional non-convex optimization
converges to optimal in "two phases": In phase I, the gradient points to the wron
function g gradually decreases. Then in phase II, SGD enters a nice one point convex region and converges. We
also show that the identity mapping is necessary for convergence, as it moves the initial point to a better place
for optimization. Experiment verifies our claims.

C Phase I: The Decreasing Potential Function

As we saw in Theorem 3.3, if $\|\mathbf{W}\|_2, \|\mathbf{W}^*\|_2$ is bounded by a constant $\gamma = \frac{1}{100}$, and the potential function $g \le 0.1$, $\mathrm{L}(\mathbf{W})$ is 0.03-one point convex, which will give us convergence guarantee according to Lemma 2.5. However, g could be larger than 0.1 initially, and as we run SGD, $\|\mathbf{W}\|_2$ might be larger than $\frac{1}{100}$ as well.

In this section, we address both problems by analyzing the dynamics of SGD, thus prove Theorem 3.2. The

negative gradient of $\mathrm{L}(\mathbf{W})$ *is approximately*

$+ (\mathbf{W}^* - \mathbf{W})^\top - 2\mathrm{Diag}(\mathbf{W}^* - \mathbf{W}) + g\mathbf{I} + \mathbf{W}$

ror is $\|\mathbf{Q}(\mathbf{W}) - [-\nabla \mathrm{L}(\mathbf{W})]\|_2 \le 61\gamma^2$.

l norm.

$\nabla \mathrm{L}(\mathbf{W})\|_2 \le 6d\gamma$.

We use subscript t under each variable to denote its value at the
$_t \triangleq (\mathbf{W}^* - \mathbf{W}_t)u$. We first compute the updating rule for g_t.

≥ 100, $\eta \le \frac{\gamma^2}{Cd^2}$, *then* $|g_{t+1}| \le (1 - 0.95\eta d)|g_t| + 86\eta d\gamma^2 +$

ge, so we also need to compute its updating rule:

$\|s_{t+1}\|_2 \le \left(1 - \eta\frac{(d+1)\gamma}{8}\right)\|s_t\|_2 + \eta(6.61\gamma + 1.03|g_t| + \varepsilon)\sqrt{d}$.

Show that g_t will shrink, conditioned on that $\|\mathbf{W}_t\|_2$ is bounded

$\mathbf{W}^*\|_2 \le \gamma \le \frac{1}{100}, d \ge 100, \eta \le \frac{\gamma^2}{Cd^2}, \varepsilon \le \gamma^2$, *then* $|g_t|$ *will keep*
, until $|g_{t_1}| \le 197\gamma^2$ *for* $t_1 \le \frac{1}{10\eta}$.

ays bounded by γ during the process described in Lemma C.5.

such that if $\|\mathbf{W}_0\|_2, \|\mathbf{W}^*\|_2 \le \gamma_0$, $d \ge 100$, $\eta \le \frac{\gamma^2}{Cd^2}$, $\varepsilon \le \gamma^2$,
always have $\|\mathbf{W}_T\|_2 \le \gamma \le \frac{1}{100}$ *for any* $T > 0$.

and $\|\mathbf{W}_T\|_2 \le \gamma$, which means we are in Phase II. The next
er.

such that if $\|\mathbf{W}_0\|_2, \|\mathbf{W}^*\|_2 \le \gamma_0$, $d \ge 100$, $\eta \le \frac{\gamma^2}{Cd^2}$, $\varepsilon \le \gamma^2$,
then after $|g_{t_1}| \le 197\gamma^2$, *Phase I ends and Phase II starts. That is, for every* $T > t_1$, $\|\mathbf{W}_T\|_2 \le \gamma$ *and* $|g_T| \le 0.1$.

Proof for Theorem 3.2. We immediately get Theorem 3.2 by combining the above three lemmas. They show that g_t will decrease to a small value in Phase I (Lemma C.5), $\|\mathbf{W}_t\|_2$ will keep small during this process (Lemma C.6), and they all keep small afterwards (Lemma C.7). □

- What will you learn?
  - Not about "how to import python packages"!
  - Not about "theorem proving"
- Formal Mission
  - **How to train a neural net**
    - To solve practical problems

Some Terminologies

Data

Neural Network (model, f)

Goal : Learn a neural network from data

The Problem

$X \longrightarrow Y$

Discriminative Model (classification)

$Y \longrightarrow X$

Generative Model (generation)

(detailed) Goal: learn a discriminative/generative neural network from data

About the Course

- Prerequisites
  - Linux and Python coding
  - Calculus & basic Machine Learning
  - Mentally ready for **heavy** engineering work ☺

- What will you learn?
  - How to design a neural network **model** (*network architecture*)
  - **Algorithms** and **intuitions** to learn a good model for $X \rightarrow Y / Y \rightarrow X$
  - How to apply a learned model to solve different **real-world problems**
  - Broad **understanding** of active domains and important problems

A whole class of "*tools*" and "*case studies*" for practical problems

Additional References

- Deep Learning
  - https://mitpress.mit.edu/books/deep-learning
- Dive into Deep Learning
  - https://d2l.ai/
- Use Google!
  - DeepMind and OpenAI resources
  - CMU & Stanford Courses
  - Some wonderful BLOGs
    - https://towardsdatascience.com/
    - https://colah.github.io/
    - https://lilianweng.github.io/lil-log/

Lecture 1: Overview

- Part 1: A comprehensive history recap of deep learning
  - with many lessons

- Part 2: some programming basics

Deep Learning is Everywhere

Deep learning

Yann LeCun[1,2], Yoshua Bengio[3] & Geoffrey Hinton[4,5]

Neural Networks

Representation Learning

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech.

Machine-learning technology powers many aspects of modern society: from web searches to content filtering on social networks to recommendations on e-commerce websites, and it is increasingly present in consumer products such as cameras and smartphones. Machine-learning systems are used to identify objects in images, transcribe speech into text, match news items, posts or products with users' interests, and select relevant results of search. Increasingly, these applications make use of a class of techniques called deep learning.

Conventional machine-learning techniques were limited in their ability to process natural data in their raw form. For decades, constructing a pattern-recognition or machine-learning system required careful engineering and considerable domain expertise to design a feature extractor that transformed the raw data (such as the pixel values of an image) into a suitable internal representation or feature vector from which the learning subsystem, often a classifier, could detect or classify patterns in the input.

intricate structures in high-dimensional data and is therefore applicable to many domains of science, business and government. In addition to beating records in image recognition[1–4] and speech recognition[5–7], it has beaten other machine-learning techniques at predicting the activity of potential drug molecules[8], analysing particle accelerator data[9,10], reconstructing brain circuits[11], and predicting the effects of mutations in non-coding DNA on gene expression and disease[12,13]. Perhaps more surprisingly, deep learning has produced extremely promising results for various tasks in natural language understanding[14], particularly topic classification, sentiment analysis, question answering[15] and language translation[16,17].

We think that deep learning will have many more successes in the near future because it requires very little engineering by hand, so it can easily take advantage of increases in the amount of available computation and data. New learning algorithms and architectures that are currently being developed for deep neural networks will only accelerate this progress.

What is deep learning?

- Deep Learning is a class of ***machine learning*** methods that use ***neural networks*** to ***learn representations*** from raw data.

What makes DL so special?

Deep learning

Yann LeCun[1,2], Yoshua Bengio[3] & Geoffrey Hinton[4,5]

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech.

Machine-learning technology powers many aspects of modern society: from web searches to content filtering on social networks to recommendations on e-commerce websites, and it is increasingly present in consumer products such as cameras and smartphones. Machine-learning systems are used to identify objects in images, transcribe speech into text, match news items, posts or products with users' interests, and select relevant results of search. Increasingly, these applications make use of a class of techniques called deep learning.

Feature Engineering

Conventional machine-learning techniques were limited in their ability to process natural data in their raw form. For decades, constructing a pattern-recognition or machine-learning system required careful engineering and considerable domain expertise to design a feature extractor that transformed the raw data (such as the pixel values of an image) into a suitable internal representation or feature vector from which the learning subsystem, often a classifier, could detect or classify patterns in the input.

intricate structures in high-dimensional data and is therefore applicable to many domains of science, business and government. In addition to beating records in image recognition[1–4] and speech recognition[5–7], it has beaten other machine-learning techniques at predicting the activity of potential drug molecules[8], analysing particle accelerator data[9,10], reconstructing brain circuits[11], and predicting the effects of mutations in non-coding DNA on gene expression and disease[12,13]. Perhaps more surprisingly, deep learning has produced extremely promising results for various tasks in natural language understanding[14], particularly topic classification, sentiment analysis, question answering[15] and language translation[16,17].

We think that deep learning will have many more successes in the near future because it requires very little engineering by hand, so it can easily take advantage of increases in the amount of available computation and data. New learning algorithms and architectures that are currently being developed for deep neural networks will only accelerate this progress.

What is deep learning?

- Deep Learning is a class of ***machine learning*** methods that use ***neural networks*** to ***learn representations*** from raw data.

- What makes deep learning so special?
  - Conventional machine learning (shallow learning)

$$y = f\big(\phi(x)\big) \text{ w. } \phi(x) \text{ fixed hand designed features}$$

  - Deep Learning (end-to-end learning)

$$y = f\big(NN(x;\theta)\big) \text{ w. } NN(\cdot) \text{ a parameterized neural network}$$

Neural Network is a universal function approximator

What is Neural Network?

- Artificial Neural Network (ANN, or NN) is *a computing system* based on a collection of *connected units or nodes* called artificial neurons, which *loosely* model the neurons in a biological brain.
  - Connections are also called edges
  - Neurons and edges typically has *weights* associated
  - A weight adjusts as learning proceeds
  - The output of each neuron is computed by some non-linear function of the sum of its inputs

- How is it invented?

A feedforward NN

The Rise of Connectionism

- The human brain is a connectionist machine
  - Bain, A. (1873). Mind and body. The theories of their relation. London: Henry King.
  - Ferrier, D. (1876). The Functions of the Brain. London: Smith, Elder and Co.
- The processing/capacity of the brain is a function of these connections

Many neurons connect in to each neuron
Each neuron connects out to many neurons
Neurons can be activated

Connectionist Machines

- Connectionist machines emulate brain structure
  - A computation network of processing elements
- All world *knowledge* is stored in the *connections* between the elements
- Neural networks are connectionist machines

  - The machine has many non-linear processing units
  - Cyclic Connections may also define memory

Turing's Connectionist Machines

- B-type machine
  - A unit is a NAND node (NOT+AND)
  - An edge can modify the signal
  - Learning: edge types are determined by an A-type machine
- Intelligent Machinery
  - Alan Turing, as early as 1948
- Unfortunately, it was dismissed by his employer and went unpublished until 1968, 14 years after his death.

Intelligent Machinery

A. M. Turing
[1912—1954]

Abstract

The possible ways in which machinery might be made to show intelligent behaviour are discussed. The analogy with the human brain is used as a guiding principle. It is pointed out that the potentialities of the human intelligence can only be realized if suitable education is provided. The investigation mainly centres round an analogous teaching process applied to machines. The idea of an unorganized machine is defined, and it is suggested that the infant human cortex is of this nature. Simple examples of such machines are given, and their education by means of rewards and punishments is discussed. In one case the education process is carried through until the organization is similar to that of an ACE.

Before his "Turing Test" paper in 1950

McCulloch-Pitts Neuron

- The first mathematical model of neuron
- The model was specifically targeted as a computational model of the "nerve net" in the brain.
- Boolean unit
  - $x_n \in \{0,1\}$
  - $g(x) = \sum_i x_i$
  - $y = f\big(g(x)\big) = \mathbb{I}[g(x) > \theta]$ (or flip)

- Represent any Boolean functions

McCulloch-Pitts Neuron

- The first artificial neuron.
  - *A Logical Calculus of the Ideas Immanent in Nervous Activity, 1943*
  - McCulloch: Neurophysiologist, Prof at U. Illinois at Chicago
  - Pitts: 20 year-old homeless wannabe logician who arrived at his door
  - After five years of unofficial studies, the University of Chicago awarded Pitts an Associate of Arts (his only earned degree) for his work on the paper

Hebbian Learning

- The first learning algorithm for neural networks
  - “Organization of behavior”, Donald Hebb, 1949
- Principle: *Cells that fire together wire together.*
  - *If neuron x_i repeatedly triggers neuron y, the synaptic knob (神经突触) connecting x_i to gets larger*
- Mathematical model:

$$w_i = w_i + \eta\, x_i\, y$$

w_i is the weight between x_i and y

- The basis of many learning algorithms in Machine Learning!

The First Implementation Attempt

- SNARC (Stochastic Neural Analog Reinforcement Calculator)
  - 1951 summer
  - 40 units with random wires
  - An operator will press a button to give reward to the machine

- Marvin Minsky
  - 1969 Turing Award
  - Founder of MIT CSAIL

The First Spring of AI

- Perceptron, 1958 by Frank Rosenblatt
  - First practical implementation of learning neural networks
- Mark 1 perceptron: a machine for image recognition
  - 400 photocells, randomly connected to the "neurons".
  - Weights were encoded in potentiometers (继电器)
  - Weight updates during learning
- *“the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.”* --- New York Times, 1958

The MIT Summer Vision Project

- Marvin Minsky
  - 1969 Turing Award
  - Founder of MIT CSAIL
- The project at 1966 summer
  - *“The summer vision project is an attempt to* ***use our summer workers*** *effectively in the construction of a significant part of* ***a visual system****. The particular task was chosen partly* ***because it can be segmented into sub-problems*** *which allow individuals to* ***work independently*** *and yet participate in the construction of a system complex enough to be real* ***landmark in the development of ‘pattern recognition ’“”***
- Summary
  - “奋斗一个夏天，解决模式识别”

The Perceptron

- Simplified Mathematical Formula
  - $y = \begin{cases} 1 & \text{if } \sum_i w_i x_i - T > 0 \\ 0 & \text{else} \end{cases}$
  - w weights;
- Learning rule
  - $w = w + \eta(d(x) - y(x))x$
  - Update the weights whenever the perceptron output is wrong
  - Convergence on linearly-separable case
  - It can mimic any Boolean gates (AND, OR, NOT)

Values shown on edges are weights, numbers in the circles are thresholds

The Perceptron

- The XOR problem
  - The book, *Perceptrons*, at 1969 by Marvin Minsky

- Multi-layer Perceptron!
  - Also suggested by Marvin Minsky at 1969

- A Multi-layer perceptron can compose arbitrarily complicated Boolean functions!

The Perceptron

- Perceptron on real inputs
  - Construct a perceptron for the following decision boundary

The Perceptron

- Perceptron on real inputs
  - Construct a perceptron for the following decision boundary

The Perceptron

- Perceptron on real inputs
  - A more complex case

MLPs can represent arbitrarily complex decision (classification) boundaries

MLPs are universal Boolean functions

The Perceptron

- How many layers are necessary for Boolean functions?
  - *Two-layered MLPs are universal Boolean functions*
- How many perceptrons in hidden layers are required for a N-variable function?
  - The XOR function over N variables
  - $O(2^{N-1})$ perceptrons
- Lower-bound for the parity function (Håstad, 1987)
  - depth-k circuits require size $\exp\left(\Omega\left(N^{\frac{1}{k-1}}\right)\right)$
- A ***deep*** network is preferred!

$O = W \oplus X \oplus Y \oplus Z$

The Perceptron

- How to learn the weights of a ***multi-layer*** perceptron?
  - An intuitive example
  - Low-level: linear boundaries
  - Hidden-level: convex regions
  - Exponential search?

- *Training a 3-node neural network is NP-complete*
  - *Avrim L. Blum & Ronald L. Rivest, Neural Networks, 1992*
  - Assumption: a 2-layered network with n-variable input and thresholding functions

The First AI Winter

- The perceptron cannot represent the XOR problem
  - Marvin Minsky, 1969
- The notion of NP-completeness
  - Levin-Cook Theorem, 1971
  - Karp's 21 NP-complete problems, 1973
- Sad Fact: Frank Rosenblatt died in July 1971 on his 43rd birthday ☹

The First AI Winter

- The Lighthill report, 1973
  - *Artificial Intelligence: A General Survey* by James Lighthill
  - Evaluation of the state of AI research in the United Kingdom

3 Past disappointments

Most workers in AI research and in related fields confess to a pronounced feeling of disappointment in what has been achieved in the past twenty-five years. Workers entered the field around 1950, and even around 1960, with high hopes that are very far from having been realised in 1972. In no part of the field have the discoveries made so far produced the major impact that was then promised.

The disappointment felt may be analysed into two kinds: work in the categories A and C of section 2 has some respectable achievements to its credit (and achievement in such categories of work with rather clear aims is clearly discernible), but to a disappointingly smaller extent than had been hoped and expected, while progress in category B has been even slower and more discouraging, tending (as explained in section 2) to sap confidence in whether the field of research called AI has any true coherence. In the meantime, claims and predictions regarding the potential results of AI research had been publicised which went even farther than the expectations of the majority of workers in the field, whose embarrassments have been added to by the lamentable failure of such inflated predictions.

These general statements are expanded in a little more detail in the rest of section 3, which has been influenced by the views of large numbers of people listed in section 1 but which like the whole of this report represents in the last analysis only the personal view of the author. Before going into such detail he is inclined, as a mathematician, to single out one rather general cause for the disappointments that have been experienced: failure to recognise the implications of the combinatorial explosion. This is a general obstacle to the construction of a self-organising system on a large knowledge base which results from the explosive growth of any combinatorial expression, representing numbers of possible ways of grouping elements of the knowledge base according to particular rules, as the base's size increases.

The First AI Winter

Summary:

- **MLP is Universal Function Approximator**
- **But hopeless to find the *optimal* solution**

- The Lighthill report, 1973
  - *Artificial Intelligence: A General Survey* by James Lighthill
  - Evaluation of the state of AI research in the United Kingdom

- DARPA Funding Cuts, 1974
  - DARPA's own study (the American Study Group) suggested that most AI research was unlikely to produce anything truly useful in the foreseeable future

- A decade-long AI winter until the success of expert system in 1980s
  - And then followed by the second (longer) AI winter in 1990s ☺

Differentiable Functions

- Why is training so hard in classical multi-layer perceptrons?
  - Perceptron training: adjust weights when error
  - You can alter weights a lot without changing the error
  - No indication of improvement → combinatorial optimization

$\sigma(z) =$

Differentiable Functions

- The differentiable approximation of thresholding

Differentiable Functions

- A probability interpretation of sigmoid function
  - Differentiable function → we now have gradient!
  - The direction of improvement

Differentiable Functions

- The *Backpropagation* Algorithm
  - *Learning representations by back-propagation errors, Nature, 1986*
  - *By David Rumelhart, Geoffrey Hinton, Ronald Williams*
  - Key idea: update weights by back-propagating gradients

Learning representations by back-propagating errors

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

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

Fig. 2 Two isomorphic family trees. The information can be

Fig. 4 The weights from the 24 input units that represent people to the 6 units in the second layer that learn distributed representations of people. White rectangles, excitatory weights; black rectangles, inhibitory weights; area of the rectangle encodes the magnitude of the weight. The weights from the 12 English people are in the top row of each unit. Unit 1 is primarily concerned with the distinction between English and Italian and most of the other units ignore this distinction. This means that the representation of an English person is very similar to the representation of their Italian equivalent. The network is making use of the isomorphism between the two family trees to allow it to share structure and it will therefore tend to generalize sensibly from one tree to the other. Unit 2 encodes which generation a person belongs to, and unit 6 encodes which branch of the family they come from. The features captured by the hidden units are not at all explicit in the input and output encodings, since these use a separate unit for each person. Because the hidden features capture the underlying structure of the task domain, the network generalizes correctly to the four triples on which it was not trained. We trained the network for 1500 sweeps, using $\varepsilon = 0.005$ and $\alpha = 0.5$ for the first 20 sweeps and $\varepsilon = 0.01$ and $\alpha = 0.9$ for the remaining sweeps. To make it easier to interpret the weights we introduced 'weight-decay' by decrementing every weight by 0.2% after each weight change. After prolonged learning, the decay was balanced by $\partial E/\partial w$, so the final magnitude of each weight indicates its usefulness in reducing the error. To prevent the network needing large weights to drive the outputs to 1 or 0, the error was considered to be zero if output units that should be on had activities above 0.8 and output units that should be off had activities below 0.2.

Differentiable Functions

- The *Backpropagation* Algorithm
  - *Learning representations by back-propagation errors, Nature, 1986*
  - *By David Rumelhart, Geoffrey Hinton, Ronald Williams*
  - Key idea: update weights by back-propagating gradients

- For the first time, it shows such a backward gradient computation method can automatically learn features
  - Although the idea can be traced back to Paul Werbos's dissertation at 1974
  - *Remark: Still **NO** guarantee for optimal solution*
  - ***It is particularly important to make an idea work!***

Revival of Connectionism

- Geoffrey E. Hinton
  - PhD in AI at 1978, U. Edinburgh (short of funding)
  - Worked at UCSD and CMU (backprop)
  - Now a Prof at U. Toronto
  - Inventor of Boltzmann machine, distributed representations, mixtures of experts
  - The beginning of DL era in 2012
  - Turing Award at 2018
- Comments
  - 1978 ~ 2012 → two AI winter and multiple waves
  - Many ideas has been invented for decades
  - The perspective of doing great science

Fun story: Geoffrey **Everest** Hinton
- great-great-grandson of logician George Boole
- middle name is from another relative, George Everest

Revival of Connectionism (1990s)

- The convolutional neural network (CNN)
  - *Backpropagation applied to handwritten zip co*
  - *Neural Computation 1989*
  - *By Yann LeCun, et al., from AT&T Bell Lab*
  - Remark: idea can be traced back to K. Fukushim
- LeNet-5 at 1998
  - The first commercial application of CNN
  - The foundation of modern computer vision

LeNet-1 at 1993
Remark: 32-year-old Yann LeCun
Fun fact: A pos-doc of Hinton

10-year effort to make CNN work!

Revival of Connectionism (2000s)

- Neural Networks for Natural Language Processing
  - *A neural probabilistic language model, NIPS 2000* (neural language model)
  - *By Bengio, Yoshua, Réjean Ducharme, and Pascal Vincent*
  - *Neural Machine Translation by Jointly Learning to Align and Translate, ICLR 2015* (attention)
  - *By Dzmitry Bahdanau, KyungHyun Cho, Yoshua Bengio*

a simple PCFG

1.0 S → NP VP
0.3 NP → Adj Noun
0.7 NP → Det Noun
1.0 VP → Vb NP
-
0.2 Adj → fruit
0.2 Noun → flies
1.0 Vb → like
1.0 Det → a
0.4 Noun → banana
0.4 Noun → tomato
0.8 Adj → angry

Example

$1 * 0.3 * 0.2 * 0.7 * 1.0 * 0.2 * 1 * 1 * 0.4 = 0.0033$

Revival of Connectionism (2000s)

- Neural Networks for Natural Language Processing
  - *A neural probabilistic language model, NIPS 2000* (neural language model)
  - *By Bengio, Yoshua, Réjean Ducharme, and Pascal Vincent*
  - *Neural Machine Translation by Jointly Learning to Align and Translate, ICLR 2015* (attention)
  - *By Dzmitry Bahdanau, KyungHyun Cho, Yoshua Bengio*
- The foundation of deep NLP models
  - Prof. Yoshua Bengio, Turing Award 2018
  - 15 years of works, finally revolutionize the NLP field

Revival of Connectionism: The New Spring

- **Speech Recognition, 2012**
  - Led by G. Hinton; Joint work with Google, MSR and IBM
  - Outperforms existing methods with a huge gap (1.3% ~ 10%)

[TABLE 3] A COMPARISON OF THE PERCENTAGE WERs USING DNN-HMMs AND GMM-HMMs ON FIVE DIFFERENT LARGE VOCABULARY TASKS.

TASK	HOURS OF TRAINING DATA	DNN-HMM	GMM-HMM WITH SAME DATA	GMM-HMM WITH MORE DATA
SWITCHBOARD (TEST SET 1)	309	18.5	27.4	18.6 (2,000 H)
SWITCHBOARD (TEST SET 2)	309	16.1	23.6	17.1 (2,000 H)
ENGLISH BROADCAST NEWS	50	17.5	18.8	
BING VOICE SEARCH (SENTENCE ERROR RATES)	24	30.4	36.2	
GOOGLE VOICE INPUT	5,870	12.3		16.0 (>> 5,870 H)
YOUTUBE	1,400	47.6	52.3	

Revival of Connectionism: The New Spring

- Speech Recognition, 2012
- **Google's Brain Recognizes Cat Videos, 2012**
  - Led by Andrew Ng and Jeff Dean, at Google X Lab
  - A neural network of 16,000 CPUs trained on YouTube videos

Revival of Connectionism: The New Spring

- Speech Recognition, 2012
- Google's Brain Recognizes Cat Videos, 2012
- **AlexNet in ImageNet Challenge, 2012**
  - By Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton

2012 is a magical year!

Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

Table 1: Comparison of results on ILSVRC-2010 test set. In *italics* are best results achieved by others.

Revival of Connectionism: The New Spring

- Speech Recognition, 2012
- Google's Brain Recognizes Cat Videos, 2012
- AlexNet in ImageNet Challenge, 2012
- **Deep Q-Learning, 2014**
  - By DeepMind
  - First time apply deep learning in reinforcement learning

Revival of Connectionism: The New Spring

- Speech Recognition, 2012
- Google's Brain Recognizes Cat Videos, 2012
- AlexNet in ImageNet Challenge, 2012
- Deep Q-Learning, 2014
- **AlphaGo, 2016**
  - by DeepMind
  - Defeat 李世乭 by 4:1, March 2016

Revival of Connectionism: The New Spring

- Speech Recognition, 2012
- Google's Brain Recognizes Cat \
- AlexNet in ImageNet Challenge
- Deep Q-Learning, 2014
- AlphaGo, 2016
- **Google's Machine Translation System, 2016**
  - *It reduces translation errors by 60% compared to Google's phrase-based production system*
  - Only 2~3 years of development

Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi
yonghui,schuster,zhifengc,qvl,mnorouzi@google.com

Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, Jeffrey Dean

Revival of Connectionism: The New Spring

- Speech Recognition, 2012
- Google's Brain Recognizes Cat Videos, 2012
- AlexNet in ImageNet Challenge, 2012
- Deep Q-Learning, 2014
- AlphaGo, 2016
- Google's Machine Translation System, 2
- **OpenAI's RL Projects, 2019**
  - **OpenAI Five defeated OG in Dota II (2:0)**

Revival of Connectionism: The New Spring

- Speech Recognition, 2012
- Google's Brain Recognizes Cat Videos, 2012
- AlexNet in ImageNet Challenge, 2012
- Deep Q-Learning, 2014
- AlphaGo, 2016
- Google's Machine Translation System, 2016
- **OpenAI's RL Projects, 2019**
  - **OpenAI Five defeated OG in Dota II (2:0)**
  - **First time to use pure deep RL to solve Rubik's cube manipulation**

Revival of Connectionism: The New Spring

Playground Load a preset...

Is it important to study deep learning as a college student?

Yes, definitely. Deep learning is one of the most important research areas in computer science today. If you want to learn about artificial intelligence, machine learning, and deep learning, then it's good to study them as a college student.

Is it important to learn deep learning as a college student?

Definitely. Artificial intelligence is the future. It's going to be in every industry from medicine to finance to marketing. And deep learning is a huge part of that.

I just saw a beautiful girl sitting next to me, but I was too nervous to say anything to her. Now, I notice that she is going to leave but I don't want to miss her. So, I grab a piece of paper and pen and I write: I think you're beautiful and would love to get to know you better. Then I slip the note into her bag. I hope she gets it.

shot learner

Revival of Connectionism: The New Spring

- Speech Recognition, 2012
- Google's Brain Recognizes Cat Videos, 2012
- AlexNet in ImageNet Challenge, 2012
- Deep Q-Learning, 2014
- AlphaGo, 2016
- Google's Machine Translation System, 2016
- OpenAI's RL Projects, 2019
- GPT-3, 2020
- **AlphaFold, 2020**
  - A new trend of "AI for science"

T1037 / 6vr4
90.7 GDT
(RNA polymerase domain)

T1049 / 6y4f
93.3 GDT
(adhesin tip)

Experimental result
Computational prediction

Revival

- Speech Re
- Google's E
- AlexNet ir
- Deep Q-Le
- AlphaGo,
- Google's M
- OpenAI's I
- GPT-3, 20
- AlphaFold
- **AlphaCode, 2021**

1553_H. XOR and Distance | C++ | pass | Layer 18 | play

Head 1 | Head 2 | Head 3 | Head 4 | Head 5 | Head 6 | Head 7 | Head 8 | Head 9 | Head 10 | Head 11 | all | none

Problem Description

```
// RATING: 1200
// TAGS: brute force,greedy
// LANGUAGE IS cpp
// CORRECT SOLUTION
// You are given an array a consisting of n distinct elements and an integer k.
// Each element in the array is a non-negative integer not exceeding 2^k-1.
//
// Let's define the XOR distance for a number x as the value of
//
// $$$f(x) = min_{i = 1}^{n} min_{j = i + 1}^{n} |(a_i ⊕ x) - (a_j ⊕ x)|,$$$
//
// where ⊕ denotes [the bitwise XOR
// operation](https://en.wikipedia.org/wiki/Bitwise_operation#XOR).
//
// For every integer x from 0 to 2^k-1, you have to calculate f(x).
//
// Input
//
// The first line contains two integers n and k (1 ≤ k ≤ 19; 2 ≤ n ≤ 2^k).
//
// The second line contains n integers a_1, a_2, ..., a_n (0 ≤ a_i ≤ 2^k-1). All
// these integers are distinct.
//
// Output
//
// Print 2^k integers. The i-th of them should be equal to f(i-1).
//
// Examples
//
// Input
//
//
// 3 3
// 6 0 3
//
//
// Output
//
//
// 3 1 1 2 2 1 1 3
```

Solution

```
#include <bits/stdc++.h>
using namespace std;
int n, k;
int a[1 << 19];
int main() {
  cin >> n >> k;
  for (int i = 0; i < n; i++) {
    cin >> a[i];
  }
  for (int x = 0; x < (1 << k); x++) {
    int ans = (1 << 30);
    for (int i = 0; i < n; i++) {
      for (int j = i + 1; j < n; j++) {
        int t = a[i] ^ x;
        int t2 = a[j] ^ x;
        ans = min(ans, abs(t - t2));
      }
    }
    cout << ans << " ";
  }
  return 0;
}
```

Revival of Connectionism: The New Spring

- Speech Recogni
- Google's Brain R 2
- AlexNet in Imag
- Deep Q-Learnin
- AlphaGo, 2016
- Google's Machin 2
- OpenAI's RL Projects, 2019
- GPT-3, 2020
- AlphaFold, 2020
- AlphaCode, 2021

- **AIGC, 2022**

Revival of Connectionism: The New Spring

User: help me write a short note to introduce myself to my neighbor

ChatGPT: Hi there,

My name is [Your Name] and I just moved in next door. I wanted to reach out and introduce myself. I'm looking forward to getting to know you and the neighborhood. If you ever need anything or just want to chat, please don't hesitate to stop by or give me a call.

Best,
[Your Name]

User: can you make it more formal?

ChatGPT: Dear Neighbor,

I am writing to introduce myself. My name is [Your Name] and I have recently moved in next door. I wanted to take a moment to reach out and say hello. I am looking forward to getting to know you and the neighborhood.

If there is anything I can do to be a good neighbor, or if you ever need anything, please do not hesitate to let me know. I am happy to help in any way that I can.

Sincerely,
[Your Name]

User: can you send it for me?

ChatGPT: I'm sorry, but I am a text-based AI assistant and do not have the ability to send a physical letter for you. I can only provide suggestions and advice on what to write. It is up to you to take that advice and send the letter yourself.

User: this code is not working like i expect — how do i fix it?

```
resultWorkerErr := make(chan error)
defer close(resultWorkerErr)
go func() {
        defer cancel()
        resultWorkerErr <- b.resultWorker(ctx)
}()

err := b.worker(ctx)
cancel()
if err == nil {
        return <-resultWorkerErr
}
return multierror.Append(err, <-resultWorkerErr)
```

ChatGPT: It's difficult to say what's wrong with the code without more context. Can you provide more information about what the code is supposed to do and what isn't working as expected? Also, is this the entire code or just a part of it?

- **ChatGPT, 2022**

Revival of Connectionism: The New Spring

- GPT-3, 2020
- AlphaFold, 2020
- AlphaCode, 2021

- **GPT4-V, 2023**

Re

- Sp
- G
- Al
- De
- Al
- G
- O
- G
- AlphaFold, 2020
- AlphaCode, 2021

- GPT4-V, 2023
- **Sora, 2024**

The Era of Deep Learning

- Rapid Advances
  - Image Classification
  - Larger/Deeper models

The Era of Deep Learning

- BIG data + Huge Model in NLP
  - Without any use of human knowledge

The Era of Deep Learning

- Turing Award at 2018
  - Geoffrey Hinton, inventor of modern backpropagation and many fundamental ideas
  - Yann LeCun, Inventor of convolutional neural networks, foundation of modern pattern recognition
  - Yoshua Bengio, Inventor of neural language model and attention

The Era of Deep Learning

A Historical Perspective

- The history of deep learning
  - Original over-optimism in 1960s and then AI winter
  - A “cold” research area for decades (1970s ~ 2000s)
  - Magic year of 2012 ---- but “Rome wasn't built in a day”
- What’s next?
- We can we learn? **Get ready to solve real-world challenges!**
  - Be determined for great science!
  - Be respectful to prior works ---- most ideas have been explored
  - Be aware of entire literature beyond those hot topics!
  - Deep Learning is the science of engineering …
  - **and it is critical to make things work!**
    - Idea is cheap. Show me the code/model!

Coding Time!

- You need to
  - Use Google! (so try to get your VPN set up)
  - Be ready to use Python and PyTorch
  - Be prepared to write a well-written report in English (Grammarly recommended!)
  - Learn to use cloud compute (no need for HW1, more instructions coming)
- Computation
  - Your laptop for debugging and coding
  - Google CoLab for light tuning
  - Tencent-Cloud (腾讯云) for heavy training

Google Colab Tutorial

Deep Learning 2024 Spring

Huaijie Wang, 2023/03/01

Purpose

- Introduce the usage of Google Colab
- Help the ones who are not familiar with Python or PyTorch

Google Account

- https://accounts.google.com/SignUp?hl=en

Google Colab

- Colaboratory is a Google research project created to help disseminate machine learning education and research.
- It’s a Jupyter notebook environment that requires no setup to use and runs entirely in the cloud.
- It allows you to use free Tesla T4 GPU it also gives you a total of 16GB of RAM, and you can use it up to 12 hours in row (You need to restart the session after 12 hours)

欢迎使用 Colaboratory

文件 修改 视图 插入 代码执行程序 工具 帮助

分享

目录

使用入门

数据科学

机器学习

更多资源

精选示例

部分

+ 代码 + 文本 复制到云端硬盘

连接 修改

欢迎使用 Colab!

如果您已经熟悉 Colab，请观看此视频，了解互动表格、已执行代码的历史记录视图和命令面板。

什么是 Colab?

借助 Colaboratory（简称 Colab），您可在浏览器中编写和执行 Python 代码，并且：

- 无需任何配置
- 免费使用 GPU
- 轻松共享

无论您是一名**学生**、**数据科学家**还是 **AI 研究员**，Colab 都能够帮助您更轻松地完成工作。您可以观看 Colab 简介了解详情，或查看下面的入门指南！

使用入门

您正在阅读的文档并非静态网页，而是一个允许您编写和执行代码的交互式环境，称为 **Colab 笔记本**。

例如，以下**代码单元格**包含一个简短的 Python 脚本，该脚本会计算值、将其存储在变量中并输出结果：

```
[ ] seconds_in_a_day = 24 * 60 * 60
    seconds_in_a_day
```

```
86400
```

How to Upload Your Code

- From local (our coding projects)
  - File -> Upload Notebook -> Choose File from local
- From Google Drive (or other existing Colab notebooks)
  - Don’t run codes directly!
- From GitHub
  - File -> Open Notebook

欢迎使用 Colaboratory

文件 修改 视图 插入 代码执行程序 工具 帮助

分享

目录

使用入门

数据科学

机器学习

更多资源

精选示例

+ 部分

+ 代码 + 文本 复制到云端硬盘

连接 Colab AI

欢迎使用 Colab!

（新功能）试用 Ger

- Generate a Gemini API
- Talk to Gemini with the
- Compare Gemini with
- More notebooks

如果您已经熟悉 Colab，请观

[ ] 开始借助 AI 编写或生成代

什么是 Colab?

借助 Colaboratory（简称 Col

- 无需任何配置
- 免费使用 GPU
- 轻松共享

无论您是一名**学生**、**数据科学家**还是 **AI 研究员**，Colab 都能够帮助您更轻松地完成工作。您可以观看 Colab 简介了解详情，或查看下面的入门指南！

使用入门

打开笔记本

示例

最近打开过

Google 云端硬盘

GitHub

上传

搜索笔记本

标题	上次打开时间	首次打开时间
欢迎使用 Colaboratory	22:06	2022年10月27日
Basic_MetaDrive_Usages.ipynb	1月24日	1月24日
Kaggle.ipynb	2023年12月27日	2023年12月26日
tensorboard_in_notebooks.ipynb	2023年12月26日	2023年12月26日
word_embeddings_tutorial.ipynb	2023年12月21日	2023年12月21日
TPUs in Colab	2023年10月27日	2023年10月27日
Bot Of Legends Failed To Download Could N...	2023年10月7日	2023年10月7日

+ 新建笔记本

取消

Set Up Environment

- Hardware environment
  - Connect to hosted runtime by clicking “Connect”
  - Enable GPU (TPU):
    - Edit -> Notebook Settings
- You can run shell command with a “!” prefix

```
[1] !nvidia-smi

Wed Feb 16 12:26:29 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   40C    P8    27W / 149W |      0MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Untitled1.ipynb

文件 修改 视图 插入 代码执行程序 工具 帮助 已保存所有更改

相当于连接到了一台服务器上

评论 分享

+ 代码 + 文本

重新连接 Colab AI

开始借助 AI 编写或生成代码。

版本说明

Please follow our blog to see more information about new features, tips and tricks, and featured notebooks such as Analyzing a Bank Failure with Colab.

2024-02-21

- Try out Gemma on Colab!
- Allow unicode in form text inputs
- Display documentation and link to source when displaying functions
- Display image-like ndarrays as images
- Improved UX around quick charts and execution error suggestions
- Released Marketplace image for the month of February (GitHub issue)
- Python package upgrades
  - bigframes 0.19.2 -> 0.21.0
  - regex 2023.6.3 -> 2023.12.25
  - spacy 3.6.1 -> 3.7.4
  - beautifulsoup4 4.11.2 -> 4.12.3
  - tensorflow-probability 0.22.0 -> 0.23.0
  - google-cloud-language 2.9.1 -> 2.13.1
  - google-cloud-aiplatform 1.39.0 -> 1.42.1
  - transformers 4.35.2 -> 4..37.2
  - pyarrow 10.0.1 -> 14.0.2

2024-01-29

- New Kaggle Notebooks <> Colab updates! Now you can:
  - Import directly from Colab without having to download/re-upload
  - Upload via link, by pasting Google Drive or Colab URLs
  - Export & run Kaggle Notebooks on Colab with 1 click
- Try these notebooks that talk to Gemini:
  - Gemini and Stable Diffusion
  - Learning with Gemini and ChatGPT
  - Talk to Gemini with Google's Speech to Text API
  - Sell lemonade with Gemini and Sheets
  - Generate images with Gemini and Vertex
- Python package upgrades
  - google-cloud-aiplatform 1.38.1 -> 1.39.0
  - bigframes 0.18.0 -> 0.19.2
  - polars 0.17.3 -> 0.20.2
  - gdown 4.6.6 -> 4.7.3 (GitHub issue)
  - tensorflow-hub 0.15.0 -> 0.16.0
  - flax 0.7.5 -> 0.8.0
- Python package inclusions
  - sentencepiece 0.1.99

2024-01-08

Untitled1.ipynb

文件 修改 视图 插入 代码执行程序 工具 帮助 已保存所有更改

选择使用 GPU 还是 CPU

评论 分享

+ 代码 + 文本

RAM 磁盘 Colab AI

开始借助 AI 编写或生成代码。

版本说明

连接到托管的运行时
更改运行时类型
连接到自定义 GCE 虚拟机
连接到本地运行时
查看资源
管理会话
断开连接并删除运行时
显示代码执行历史记录
将光标放在上次运行的单元格上

Please follow our blog to see more info ... tured notebooks such as Analyzing a Bank Failure with Colab.

2024-02-21

- Try out Gemma on Colab!
- Allow unicode in form text inputs
- Display documentation and link t
- Display image-like ndarrays as im
- Improved UX around quick charts
- Released Marketplace image for
- Python package upgrades
  - bigframes 0.19.2 -> 0.21.0
  - regex 2023.6.3 -> 2023.12.25
  - spacy 3.6.1 -> 3.7.4
  - beautifulsoup4 4.11.2 -> 4.12.3
  - tensorflow-probability 0.22.0 -> 0.23.0
  - google-cloud-language 2.9.1 -> 2.13.1
  - google-cloud-aiplatform 1.39.0 -> 1.42.1
  - transformers 4.35.2 -> 4..37.2
  - pyarrow 10.0.1 -> 14.0.2

2024-01-29

- New Kaggle Notebooks <> Colab updates! Now you can:
  - Import directly from Colab without having to download/re-upload
  - Upload via link, by pasting Google Drive or Colab URLs
  - Export & run Kaggle Notebooks on Colab with 1 click
- Try these notebooks that talk to Gemini:
  - Gemini and Stable Diffusion
  - Learning with Gemini and ChatGPT
  - Talk to Gemini with Google's Speech to Text API
  - Sell lemonade with Gemini and Sheets
  - Generate images with Gemini and Vertex
- Python package upgrades
  - google-cloud-aiplatform 1.38.1 -> 1.39.0
  - bigframes 0.18.0 -> 0.19.2
  - polars 0.17.3 -> 0.20.2
  - gdown 4.6.6 -> 4.7.3 (GitHub issue)
  - tensorflow-hub 0.15.0 -> 0.16.0
  - flax 0.7.5 -> 0.8.0
- Python package inclusions
  - sentencepiece 0.1.99

2024-01-08

已连接到"Python 3 Google Compute Engine 后端"

Set Up Environment

- Code environment
  - Install necessary packages with “!pip install”

```
!pip install torch torchvision

Requirement already satisfied: torch in /usr/local/lib/python3.7/dist-packages (1.10.0+cu111)
Requirement already satisfied: torchvision in /usr/local/lib/python3.7/dist-packages (0.11.1+cu111)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from torch) (3.10.0.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from torchvision) (1.21.5)
Requirement already satisfied: pillow!=8.3.0,>=5.3.0 in /usr/local/lib/python3.7/dist-packages (from torchvision) (7.1.2)
```

Set Up Environment

- Code environment
  - Where am I?

```
[2] !pwd
/content
```

  - Upload/Mount nece

Mount your google drive in Colab notebook

1. Open your Google drive

https://drive.google.com/drive/my-drive

Mount your google drive in Colab notebook

2. Create a new folder in My Drive

Mount your google drive in Colab notebook

3. Run the code in your colab

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

4. Click the link and enter your authorization code

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&red

Enter your authorization code:
```

5. success

```
[1] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

6. Now you can access your google drive in Colab notebook in /content/drive/MyDrive

```
[ ] !ls /content/drive/MyDrive
```

```
'Colab Notebooks'   deep_learning
```

Run Your Code

- Fill out the code in your homework
- Execute the code one-by-one in the notebook
  - When you select any other code segments, then you should be able to see a number in [ ] area
  - If there is no number within [ ], then you need to re-run that segment

Run your code in google drive

1. Upload your code to you google drive (linear.py)

Run your code in google drive

2. Mount your google drive

```
[1] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[ ] !ls /content/drive/MyDrive
```

```
'Colab Notebooks'   deep_learning
```

```
[ ] !nvidia-smi
```

```
Mon Feb 22 08:03:05 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.39       Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   33C    P8     9W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Run your code in google drive

3. run your code in the google drive

```
[ ] !python /content/drive/MyDrive/deep_learning/linear.py
```

```
cuda:0
(1000, 1) (1000,) 2.6731389771201597
target paremeters, weight 2.673, bias: 10.000:
[1,     1] loss: 118208.547, weight: 0.643, bias: -0.978
[1,     2] loss: 108463.602, weight: 0.646, bias: -0.957
[11,     1] loss: 103885.477, weight: 0.828, bias: 0.410
[11,     2] loss: 94150.102, weight: 0.841, bias: 0.502
[21,     1] loss: 86701.719, weight: 1.082, bias: 2.234
[21,     2] loss: 78251.109, weight: 1.094, bias: 2.320
[31,     1] loss: 74544.781, weight: 1.315, bias: 3.803
[31,     2] loss: 67236.242, weight: 1.325, bias: 3.874
[41,     1] loss: 66530.805, weight: 1.517, bias: 5.076
[41,     2] loss: 60108.305, weight: 1.526, bias: 5.133
[51,     1] loss: 61304.531, weight: 1.690, bias: 6.098
[51,     2] loss: 55559.105, weight: 1.698, bias: 6.144
[61,     1] loss: 57895.461, weight: 1.839, bias: 6.917
[61,     2] loss: 52669.000, weight: 1.846, bias: 6.953
[71,     1] loss: 55665.801, weight: 1.965, bias: 7.573
[71,     2] loss: 50839.309, weight: 1.971, bias: 7.602
[81,     1] loss: 54202.312, weight: 2.072, bias: 8.099
[81,     2] loss: 49685.758, weight: 2.077, bias: 8.122
[91,     1] loss: 53237.641, weight: 2.163, bias: 8.520
[91,     2] loss: 48962.387, weight: 2.167, bias: 8.538
[101,     1] loss: 52598.625, weight: 2.240, bias: 8.857
```

Download Your Checkpoints

- Save checkpoints in Google Drive frequently
- Download your checkpoints immediately after finishing the experiment
  - File -> Download -> Download .ipydb

Generate training data

```
[1] import torch
    import numpy as np
    from sklearn.datasets import make_regression
    import matplotlib.pyplot as plt
    import math
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    print(device)
```

```
cuda:0
```

```
[2] bias = 10
    X, y, coef = make_regression(n_samples=1000, n_features=1, n_informative=1,
                                 coef=True, bias=bias, noise=10)
    X, y = X.astype(np.float32), y.astype(np.float32)
    plt.scatter(X, y)
    plt.plot(X, X.reshape(-1) * coef + bias, color='red')
    X_all, y_all = torch.from_numpy(X).to(device), torch.from_numpy(y).to(device)
    print(X.shape, y.shape, coef)
```

```
(1000, 1) (1000,) 71.81313920366837
```

Define a Network

```
[3] import torch.nn as nn
    import torch.nn.functional as F


    class Net(nn.Module):
        def __init__(self):
            super(Net, self).__init__()
            self.fc = nn.Linear(1, 1)

        def forward(self, x):
            predict = self.fc(x)
            return predict


    net = Net()
    net.to(device)
```

```
Net(
  (fc): Linear(in_features=1, out_features=1, bias=True)
)
```

Train the network

```
[5] bsz = 500
    print("target paremeters, weight %.3f, bias: %.3f:" % (coef, bias))
    for epoch in range(301):  # loop over the dataset multiple times

        running_loss = 0.0
        for i, data in enumerate(zip(torch.split(X_all, bsz), torch.split(y_all, bsz))):
            # get the inputs; data is a list of [inputs, labels]
            # inputs, labels = data
            inputs, labels = data[0], data[1]

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = net(inputs)
            # print(outputs.shape, outputs)
            loss = criterion(outputs.view(-1), labels)
            loss.backward()
            optimizer.step()

            # print statistics
            running_loss += loss.item()
            if epoch % 10 == 0:
                weight_ = net.fc.weight.item()
                bias_ = net.fc.bias.item()
                print('[%d, %5d] loss: %.3f, weight: %.3f, bias: %.3f' % (epoch + 1, i + 1, running_loss, weight_, bias_))
                running_loss = 0.0

    print('Finished Training')
```

```
[6] PATH = './linear.pth'
    torch.save(net.state_dict(), PATH)
```

The training progress

```
target paremeters, weight 71.813, bias: 10.000:
[1,     1] loss: 2705857.500, weight: -0.320, bias: -0.436
[1,     2] loss: 2699893.000, weight: -0.183, bias: -0.415
[11,     1] loss: 2113961.250, weight: 8.660, bias: 0.927
[11,     2] loss: 2084677.250, weight: 9.260, bias: 1.018
[21,     1] loss: 1409867.000, weight: 20.470, bias: 2.712
[21,     2] loss: 1399013.750, weight: 21.024, bias: 2.796
[31,     1] loss: 918827.750, weight: 30.618, bias: 4.231
[31,     2] loss: 921905.625, weight: 31.073, bias: 4.299
[41,     1] loss: 600871.125, weight: 38.840, bias: 5.449
[41,     2] loss: 611270.500, weight: 39.205, bias: 5.502
[51,     1] loss: 397890.875, weight: 45.431, bias: 6.413
[51,     2] loss: 411304.188, weight: 45.724, bias: 6.455
[61,     1] loss: 268777.438, weight: 50.704, bias: 7.175
[61,     2] loss: 282736.000, weight: 50.938, bias: 7.208
[71,     1] loss: 186803.750, weight: 54.921, bias: 7.776
[71,     2] loss: 199999.219, weight: 55.109, bias: 7.803
[81,     1] loss: 134855.828, weight: 58.294, bias: 8.252
[81,     2] loss: 146672.781, weight: 58.444, bias: 8.273
[91,     1] loss: 102010.633, weight: 60.991, bias: 8.627
[91,     2] loss: 112233.250, weight: 61.111, bias: 8.643
[101,     1] loss: 81303.500, weight: 63.147, bias: 8.923
[101,     2] loss: 89936.211, weight: 63.243, bias: 8.936
[111,     1] loss: 68297.453, weight: 64.872, bias: 9.157
[111,     2] loss: 75456.961, weight: 64.949, bias: 9.167
[121,     1] loss: 60167.836, weight: 66.251, bias: 9.341
[121,     2] loss: 66020.109, weight: 66.313, bias: 9.349
[131,     1] loss: 55118.484, weight: 67.354, bias: 9.487
[131,     2] loss: 59842.617, weight: 67.404, bias: 9.493
[141,     1] loss: 52008.531, weight: 68.236, bias: 9.601
[141,     2] loss: 55777.508, weight: 68.276, bias: 9.606
[151,     1] loss: 50114.676, weight: 68.942, bias: 9.692
```

Visualize the weight and bias of trained model

```
[7] net = Net()
    net.load_state_dict(torch.load(PATH))
    net.to(device)
    weight_ = net.fc.weight.item()
    bias_ = net.fc.bias.item()
    print("parameters in model:")
    print("weiths: ", weight_, "bias: ", bias_)
    print("target paremeters :")
    print("weiths: ", coef, "bias: ", bias)

    fig2 = plt.figure()
    plt.scatter(X, y)
    plt.plot(X, X.reshape(-1) * weight_ + bias_, color='green')
```

```
parameters in model:
weiths:  71.66061401367188 bias:  10.018959999084473
target paremeters :
weiths:  71.81313920366837 bias:  10
[<matplotlib.lines.Line2D at 0x7ff322becf98>]
```

PyTorch Example

- Load dataset

```
#load the data
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=args['batch_size'], shuffle=True, **kwargs)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=False, transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=args['test_batch_size'], shuffle=True, **kwargs)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ../data/MNIST/raw/train-images-idx3-ubyte.gz
9913344/? [00:00<00:00, 18851363.58it/s]
Extracting ../data/MNIST/raw/train-images-idx3-ubyte.gz to ../data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ../data/MNIST/raw/train-labels-idx1-ubyte.gz
29696/? [00:00<00:00, 484374.71it/s]
Extracting ../data/MNIST/raw/train-labels-idx1-ubyte.gz to ../data/MNIST/raw
```

PyTorch Example

- Define the network

```
[11]

class Net(nn.Module):
    #This defines the structure of the NN.
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()  #Dropout
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        #Convolutional Layer/Pooling Layer/Activation
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        #Convolutional Layer/Dropout/Pooling Layer/Activation
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        #Fully Connected Layer/Activation
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        #Fully Connected Layer/Activation
        x = self.fc2(x)
        #Softmax gets probabilities.
        return F.log_softmax(x, dim=1)
```

PyTorch Example

- Define the dataflow

```
def train(epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        if args['cuda']:
            data, target = data.cuda(), target.cuda()
        #Variables in Pytorch are differenciable.
        data, target = Variable(data), Variable(target)
        #This will zero out the gradients for this batch.
        optimizer.zero_grad()
        output = model(data)
        # Calculate the loss The negative log likelihood loss. It is useful to train a classification p
        loss = F.nll_loss(output, target)
        #dloss/dx for every Variable
        loss.backward()
        #to do a one-step update on our parameter.
        optimizer.step()
        #Print out the loss periodically.
        if batch_idx % args['log_interval'] == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.data[0]))
```

Notice:

- Save your data and model in google drive, otherwise, they may be lost when you disconnect.

- If you want to train your model using Colab for a long time, make sure you save your training states frequently and you can recover your training from these training states.

The example links

- PyTorch linear:
- https://colab.research.google.com/drive/1yS5kvHC_TQuWesPmon7W7hrD978tOJ2i?usp=sharing

- Mount and run your code in google drive:
- https://colab.research.google.com/drive/163LrsNRHC48tCCGHhBfgqCyGCODXRmaR?usp=sharing

- linear.py
- https://drive.google.com/file/d/1jdwTgo6Qf4PoPz5JEQwy-KJoPncb30yt/view?usp=sharing

- PyTorch MNIST
- https://colab.research.google.com/github/rpi-techfundamentals/fall2018-materials/blob/master/10-deep-learning/04-pytorch-mnist.ipynb

Lunch Time!