

．國 二 目U 昌

隨著 AlphaGo 以 3:1 的成績戰勝李世石，人們對人工智慧的熱情如 爆炸式增 長，你也許對人工智慧充滿興趣， 嚮往著了解機器學習，特別是深度學習，那麼 本書剛好能夠由當及裸地帶你 進入課度學習這個世界。

講課度學習的書有很多， 課度學習的架構也有很多， 本書將以 PyTorch 為 工具從基礎的蟬性回歸開始， 講到時下最先進的產生對抗網路， 並在其中穿插 PyTorch 的教學，所以本書不僅是深度學習 的入門指南，同時也是 PyTorch 的入門 教學。

本書針對的對象是對課度學習有所了解、 用過一些深度學習架構 （ 如使用 TensorFlow 跑過簡單的模型）， 但是希望能夠用 PyTorch 進行深度學習研究和學習 的入門者 回 閱讀本書並不需要太多的數學基暉， 但是需要一定的 Python 基礎。本 書中的數學推導不多，厲覺困難的讀者可以跳過， 這對了解全書的主要內容不會 造成影響 ，

本書的主要內容包含：

第 1 章，深度學習介紹﹔第 2 章，海度學習架構﹔ 第 3 章，多層全連接神鱷網路﹔

第 4 章，艇積神經網路﹔

第 5 章，循環神經網路﹔

第 6 章，產生對抗網路﹔

第 7 章，課度學習貫戰。

建議讀者按照本書的內容順序學習， 因為能面的內容會以前面的內容為基 暉，另外本書的全部程式放在了 https://github.com/Elin24/learning\_pyT orch\_with\_ SherlockLiao 中 ，讀者可以前往下載。

本書針對的對象是初學者， 學習完本書之後， 讀者能夠大致 7解釋度學習的 基躍，基本掌握 PyTorch 的使用方法，知道如何根據實際問題架設對應的深層網 路結構，並能鉤進行調喜獲得較好的結果。 當然本書只是一本入門譚物， 如果希 望以接從事設頡域的研究， 僅靠此書是不夠的， 需要閱讀更多專業的書籍和學術 論文。

在本書的創作過程離不開很多人對我的幫助， 書中的一部分內容參考了李飛 飛教授在史丹佛大學開設的課程 cs23 ln’以及台大教授李宏毅開始的 MLDS，除 此之外還參考了網路上的一些圖例， 因為大多找不到出處，所以無法一一列出進 行感謝。

除此之外，還成謝在寫書的過程中我的家人對我的鼓勵和信任， 正是他們的 支援讓我能夠堅持寫完整本書。

最後，威謝電子工業出版社給我這次機會讓我能夠出版此書， 同時也戚謝孫 學瑛編輯全程對我的幫助。

由於本人水準有限， 書中存在的稅漏，歡迎大家向我指出， 我也很高興收到 大家的意見和建議， 不勝感激。

廖星字 中國科學技術大學數學系

E-mail : sherl。[ckliao01@gmail.c](mailto:ckliao01@gmail.c) 。m

-==== 目富豪

第 1章 深度學習介紹

1.1 人工智慧 ····································4•••• 4•• ....... . .........................................1 l

1.2 資料揖擷、機器學習與深度學習. 1-3

1.2.1 資料採擷 ·····················一. 1-3

1.2.2 機器學習 ··················································， 1-3

1.2.3 深度學習. 1-4

1.3 學習資輝興建議 1-10

第 2 章 深度學習架構

2.1 深度學習架構介紹.........................................................一 ··········· ...2-1

2.2 PyTorch 介紹 2-4

2.2.1 什麼是 PyTorch 2-4

2.2.2 屆何要使用 PyTorch .. .....................................................’，2-5

2.3 設定 PyTorch 課度學習環揖. 2-6

2.3.1 作業系統的選擇 .................................. ．．．．一．．．． 2-6

2.3 .2 Python 開發環境的安裝 ································· ．．．．．．． 2-7

2.3.3 PyTorch 的安裝.....................................···················· 2-9

第 3 章 多層全連接神經網路

3.1 熱身：PyTorch 基睡 .........’............... ··············· · 3-1

3.1.1 Tensor （ 張量）．．．． 3-1

3.1.2 Variable （ 撞數 ） .......... ．．．．．．......................................... ．．．．．..3-3

3.1.3 Dataset （ 資料集）．．．．.....................................，’ 3-5

3 .1.4 nn.Module （ 模組 ） ................................................ ........ ........ ．．．．． 3-6

iii

3.I 5 torch optim （ 最佳化 ） ． 3-7

3.1.6 模型的儲存和載入 － － ．， 3-9

3 2 線性模型........................ . . .. ’，． 3-9

3 2 I 問題介紹 －． ．．．．．” 3-9

3 2 2 一維線性回歸 .... ··· · · · ···· · · ········· ... . . .. .... . 3 11

3 2.3 多維線性回歸．．. .. .... ..............”....... ........ ，－ ...3-12

3.2.4 一維線性回歸的程式實現．．．．．．．．． 3-13

3.2.5 多項式回歸 ．’ 3-16

3.3 分類問題 3-20

3 3 I 問題介紹 ..............，． ．． ．－ ．． 3-20

3 3 2 Logistic 起源 ... ...…” 3 20

3.3 3 Logistic 分佈 .. .. .... .. ............... .. . ... ...3剖

3.3.4 二分類的 Logistic 回歸．．．． 3-21

3.3.5 模型的參數估計 ． － .. .............. .. ........ 3-23

3.3.6 Logistic 回歸的程式實現 3-24

3.4 簡單的多層全連接正向網路， 3 28

3.4 I 模擬神經元． ．． －－﹒ .............. ............3必

3.4.2 單層神經網路的分類器 ．－－ － ..................3”29

3 4.3 敢動函數 ．．．．’.... .. . ... . ................，.3-30

3.4.4 神經網路的結構. 3-34

3.4.5 模型的表示能力與容量. 3 35

3.5 深度學習的基礎：反向傳播演算法 － ··························· 3-37

3.5.l 鏈式法則－ － － ．．．．．．．．．．－ －… ..........3『 38

3.5.2 反向傳播演算法 ·········· ................ .. －－－ 3-38

3 5.3 Sigmoid 函數舉例. 3-39

3.6 各種最佳化演算法的變式...······· ······ ···· ············· 3-40

3 6.1 梯度下降法 ． .. ... ..... .... ． .......” 3-40

3 6 2 梯度下降法的變式 3-43

3.7 處理資料和訓練模型 的技巧 3-45

3.7.1 資料前置處理 ． 3-46

3.7.2 加權初始化........... ......“．．．． 3-48

3.7 3 防止過擬合 － ．．－ ．............................ . . . 3-50

3.8 多層全連接神經網路實現 MNIST 手寫數字分類 3-52

3.8.1 簡單的三層全連接神經網路 ... ....... .... ....﹒......”. 3-52

3.8 2 增加敢動函數 3-53

3.8.3 增加批次標準化． 3-54

3 8 4 訓練網路.....”.....”................ ．．．．． 3 54

第 4 章 旋積神經網路

4.1 主要工作及起源 ． ．． ．． ”．．............4白l

4.2 旋積神經網路的原理和結構 4-2

4.2.1 旋積層 4-5

4 2.2 池化層．．．”，’”...... ........... .. ........ . .. 4-10

4 2 3 全連接層． ....刊 .......”．”，......................... .. e ﹒ 4-12

4 2 4 旋積神經網路的基本形式 ........... ........ ．”，’” 4-13

4.3 PyTorch 旋積模組 － － ．． ．．．．．． ． ．．．” 4-15

4.3.1 旋積層．．．．．．． 4-15

4 3 2 池化層 － ............... . .... ..... .. 4 16

4 3 3 分析層結樁 ．..... ................” 4-18

4.3.4 如何分析參數及自訂初始化 － － .......... “...4-19

4.4 旋積神經網路案例分析－ － － 4-20

V

4 4.1 LeNet . . ... ﹒－ ．．．．．．．．．．．．．．’，－ 4 20

4 4.2 AlexNet . . . . . . . . ... ’ 4 22

4 4 3 VGGNet . . . ................. .. .﹒ 4-23

4.4.4 GoogLeNet .. ..................... . . ... .. . ” 4-26

4.4.5 ResNet “........... . . .’， 4-28

4.5 再實現 MNIST 手寫數字分類 4-31

4.6 影像增強的方法． 4-33

4 7 實現 cifarl 0 分類 4 35

第 5 章 循環神經網路

5.1 循環神經網路 － · ··· . .. . . . ............ .......... . . 5-l

5 l l 問題介紹 ． 5 2

5.1.2 循環神經網路的基本結構 ． ．.......... .... .. ···· ·· ..5-2

5.1.3 存在的問題－ － － ’ 5-5

5.2 循環神經網路的變式： LSTM 與 GRU 5-7

5 2 l LSTM . 。”， 5-7

5 2.2 GRU. 5 11

5.2.3 收斂性問題 ．．．．．． ． 5-12

日 循環神經網路的 PyTorch 實現. 5-14

5.3.1 PyTorch 的循環網路模組－－－ ．．··· ·· · ·· ··· ····· .. 5-15

5 3 2 實例介紹 “........ . . ...... . ...... ．．．” 5-20

5.4 自然語言處理的應用 5 24

5 4.1 詞餃入．． 5-24

5.4.2 詞最入的 PyTorch 實現. 5-27

5 4.3 N Gram 模型 .. . ...... ........ ....... ．． 5-28

5.4.4 單字預測的 PyTorch 實現． ．．． 5-28

5.4.5 詞性判斷 ．． ． 5-30

5.4.6 詞性判斷的 PyTorch 實現 5-31

5.5 循環神經網路的更多應用 5 34

5 5 I Many to one . . . . . . . . . ............................... . . . ... ’，5-35

5 5 2 Many to Many ( shorter ) . . . . . . .’，...............”.........“5-35

5.5 3 Seq2seq 5-36

5.5.4 CNN+RNN . .. .. .... .. .. .... . . . . . . .. .. . . . .. .. . 5」7

第 6 章 產生對抗網路

6 I 產生模型 .. . . . .............................. .. . . . .. ... 6 I

6 I I 自動編碼器 ．． ． － 6-2

6.1 2 變分自動編喝器 ． .... . .... .................” 6-7

6.2 產生對抗網路 ...... ... ........... ..... ... ... ...... 6-10

6.2.1 何為產生對抗網路． ......... ................... .. ....6』

6.2.2 產生對抗網路的數學原理”......... .. .. ... .. ....6 17

6.3 Improvmg GAN 6-22

6 3 I Wasserstem GAN . . . . . . . . . . .......”， 6-22

6.3.2 Improving WGAN . . . . . . . ........ ．．． 6-26

6.4 應用介紹................ ............ ............ . . ． ． ι27

6 4.1 Condit10nal GAN ....” 6 27

6 4.2 Cycle GAN . . . . . . ’， 6-30

第 7 章 深度學習實戰

7.1 實例一一一貓狗大戰：運用預訓練旋積神經網路進行特徵分析與預測 叫

71 I 背景介紹 ． 7-2

7 1.2 原理分析 ．．. . . . . .... .... ..＂ ．．．．” 7-2

7 1.3 程式實現 ． ， 7-5

7.1.4 歸納 ，.................. .... . . ... e ﹒ 7-12

7.2 實例三一一Deep Dream ：探素旋積神經網路眼中的世界 ... ... ... .7 12

7 2 I 原理介紹．．. ..... ............”......”．．．．．．． 7-13

7 2 2 預備知識 backward 7-14

7.2 3 程式實現－ ．．．．．．．．． ． － 7-20

7.2.4 歸納．． ．． . .... .... . .. .. . ... ............7必

7.3 實例三一－Neural-Style ：使用 PyTorch 進行風格移轉 7-26

7.3.1 背景介紹 7 26

7 3.2 原理分析． .. .. .. ...﹒－．． 7-27

7 3.3 程式實現 . .. . .. ..... ... ........................7回到

7.3.4 歸納 ． ’ 7-36

7.4 實例四一－Seq2seq ：透過 RNN 實現簡單的 Neural Ma chine Translation

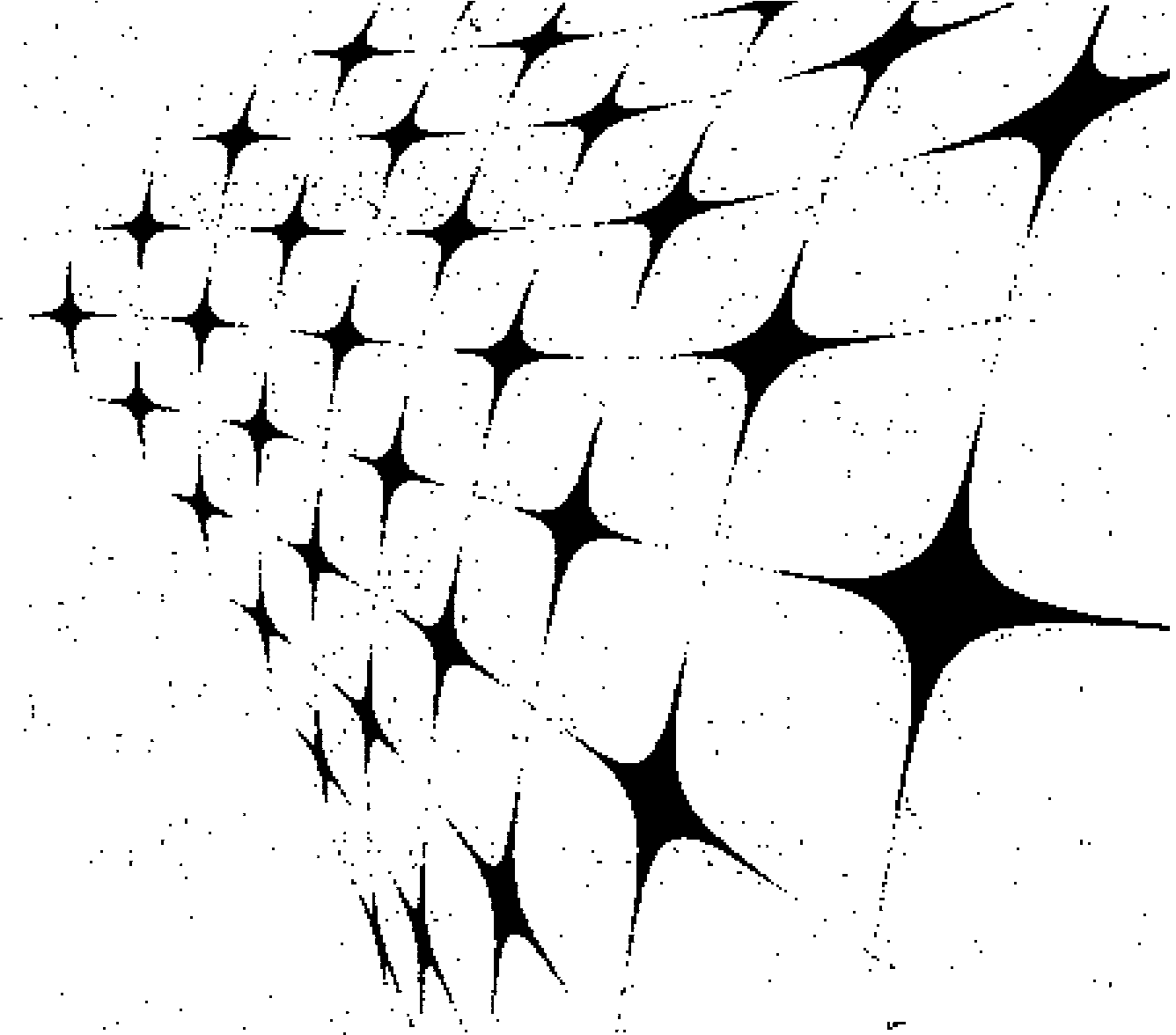
... 7 36 7 4 I 背景介紹 .. ..... － ......。”.............. ....... 7-37

7 4.2 原理分析． ． ．．．．．．．．．－ － － －﹒ ．．． 7-37

7 4.3 程式實現 ，－ ... .. ... . .. .. .... ... － － －．． 7-41

7.4.4 歸納 7-53

第 1章



###### 深度學習介紹

自古以來，發明家們都夢想著能夠創造有思想的機器人， 當電腦問世之後， 人們一直想知道它們是否可以變得更加智慧。 如今人工智慧成為一個具有無限商 業價值和研究價值的報捕， 人們也看到了人工智慧在影偉、 語音 、自然語言上取 梧的極大成功 ，

這一章，我們將簡要地介紹一下人工智慧及其重要性， 以及什麼是資料揖 擷、機器學習和i罪度學習，並介紹它們值此之間的聯繫與區別， 其中會重點介紹 深度學習，最佳列出一些合理可行的學習建議， 幫助大家從零開抬進入深度學習 這個充滿魔力的讀域 ，

1 1.1 I 人工智慧

人工智慧 （ Artificial Intelligence ）， 也稱為機器智慧， 是指由人工製造出來的 系統所表現的智慧， 所謂的智慧，即指可以觀察周圍環揖並蟑此做出行動以達到 目的。

在人工智慧的早期，那些對人額智力東說非常國難、 但對電腦來說相對簡單 的問題迅速獲得解訣， 舉側來說 ，那些可以透過一系列形式化的數學姐則來描述 的問題 也 AI 的真正挑戰在於解訣那些對人來說很容屬執行、 但很難形式化描述的 工作，舉側來說， 識別人們所說的話或影像中 的艙。對於這些問題，我們人額常 常可以憑藉直覺輕易地解訣， 因為我們已經在上萬年的進化中形成了這些直覺性 的能力，但是機器卻很難找到實現的 方法 ，

長久以來人們一直相信人工智慧是存在的， 但是卻不知道如何實現 以前的 科幻電影總會融入人工智慧， 倒賣日 ﹛星際大戰﹜ 《魔鬼終結者﹜ 《眩客帝國 卜等

等。電影的繪製使我們總覺得人工智慧缺乏真實感 ，或總將人工智慧和機器人聯 繫在一起。其實我們身邊早已實現了一些弱人工智慧， 只是因為人工智慧聽起來 很神秘，所以我們常常沒有意識到。

首先，不要一提到人工智慧就想到機器人。 機器人只是人工智慧的一種容 器，如果將人工智慧比作大腦 ，那麼機器人就好似身體一 然而這個身體卻不是 必需的，例如現在很火的 AlphaGo •其背後充滿著軟體 、演算法和資料，它下圍 棋是一種人格化的表現， 然而其本身並沒有 「 機器人」 這個硬體形式。

人工智慧的概念很寬泛，現在根據人工智蟹的實力將它分成三大類。

( l ） 弱人工智慧 （ Artificial Na叮ow Intelligence ’AN!) 弱人工智慧是擅長於單一方面的人工智慧。 例如戰勝世界圍棋冠軍的人工智 慧 Al phaGo ，它只會下圓棋，如果你讓他辨識一下貓和狗，它就不知道怎麼 做了。我們現在實現的幾乎全是 弱人工智慧 。

( 2 ） 強人工智慧 （ Artificial General Intelligence ’AG!) 這是類似人類等級的人工智慧。 強人工智慧是指在各方面都能和 人類比肩的 人工智慧，人類能幹的腦力活，它都能幹。創造強人工智慧比創造弱人工智 慧難得多，我們現在還做不到。 Linda Gottfredson 教授把智慧定義為 「 一種 寬泛的心理能力 ，能夠進行思考、計畫、解決問題、抽象思維、了解複雜理 念、快速學習和從經驗中學習等操作。J 強人工智慧在進行這些操作時應該和 人類一樣得心應手。

( 3 ） 超人工智慧 （ Artificial Superintelligen間，AS! )

牛津哲學家、知名人工智慧思想家 Nick Bostrom 把超級智蠶定義為 「 在幾乎 所有領域都比最聰明的人類大腦都聰明很多，包含科學創新、通識和社交技 能。」 超人工智慧可以是各方面都比人類強一點 ，也可以是各方面都比人類強 兆倍的 。

我們現在處於一個充滿弱人工智慧的世界， 例如垃圾郵件分類系統， 是一 個可以幫助我們篩選垃圾郵件的弱人工智慧﹔ Google 翻譯是一個可以幫助我們

l 2

翻譯英文的弱人工智慧， AlphaGo 是一個可以戰勝世界圍棋冠軍的弱人工智慧， 等等。這些弱人工智慧演算法不斷地加強創新， 每一個弱人工智慧的創新， 都是 在替通往強人工智慧和超人工智慧的旅途添磚加瓦。 正如人工智慧科學家 Aaron Saenz 所說，現在的弱人工智慧就像地球早期軟泥中 的氮基酸，可能突然之間就形 成了生命。

1 1.2 1 資料採擷、機器學習與深度學習

大數據的興起使得資料科學家身為新生職業被提出， 資料研究進階科學家 Rachel Schutt 將其定義為 「 電腦科學家、軟體工程師和統計學家的混合體」。 資 料採擷作為一個學術領域， 橫跨多個學科， 1函蓋但不限於統計學、數學、機器學 習和資料庫，此外還運用在各種專業領域， 例如油田 、電力、海洋生物、歷史文 字、影像、電子通訊等。

1.2.1 資料探擷

簡單來說，資料採擷就是在大型 的資料庫中發現有用的資訊，並加以分析的 過程，也就是人們所說的 KDD ( knowledge discovery in database ）。 一個資料的處 理過程，就是從輸入資料開始，對資料進行前置處理， 包含特徵選擇、規範化、 降低維數、資料提升等，然後進行資料的分析和採擷， 再經過處理，例如模式識 別、視覺化等，最後形成可用資訊的全過程。

所以說資料採擷只是一種概念， 即從資料中採擷到有意義的資訊， 從大量的 資料中尋找資料之間的特性。

1.2.2 機器學習

機器學習算是實現人工智慧的一種途徑， 它和資料採擷有一定的相似性，也 是一種多領域交換學科， 有關機率論、統計學、逼近論、凸分析、計算複雜性理 論等多門學科 。比較於資料採擷從大數據之間找相 互特性而言 ，機器學習更加注

重演算法的設計， 讓電腦能夠自動地從資料中 「 學習」 規律，並利用規律對未知 數據進行預測。因為學習演算法有關了大量的統計學理論，與統計推斷聯繫尤為 緊密 ，所以也被稱為統計學習方法。

機器學習可以分為以下五個大類：

( I） 監督學習：從指定的訓練資料集中學習 出一個函數，當新的資料到來時，可 以根據這個函數預測結果。 監督學習的訓練集要求是輸入和輸出， 也可以說 是特徵和 目標。訓練集中的目標是由人標記的 。常見的監督學習演算法包含 回歸與分類。

( 2 ） 無監督學習：無監督學習與監督學習相 比，訓練集沒有人為標記的結果。 常 見的無監督學習演算法有分群等。

( 3 ） 半監督學習：這是一種介於監督學習與無監督學習之間的方法。

( 4 ） 移轉學習：將已經訓練好的模型參數移轉到新的模型來幫助新模型訓練資 料集。

( 5 ） 增強學習：透過觀察周圍環境來學習 。每個動作都會對環境有所影響，學習 對象根據觀察到的周圍項境的 回饋來做出判斷。

傳統的機器學習演算法有以下幾種：線性回歸模型、logistic 回歸模型、品 臨 近演算法、決策樹、隨機森林、支援向量機、類神經網路、 EM 演算法、機率圖 模型等。

1.2.3 深度學習

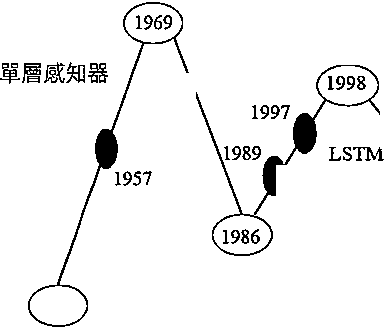
深度學習的最初級版本是類神經網路， 這是機器學習的分支，其試圖模擬人 腦 ，透過更加複雜的結構 自動分析資料特徵。

在深度學習發展起來之前， 機器學習雖然發展了幾十年， 但還是存在很多人 工智慧沒法良好解決的問題，例如影像識別、語音辨識、 自然語言處理等，而深 度學習的出現則極佳地解決了 這些領域的一部分問題。正因為大數據的興起和高 佳能 GPU 的出現，使得更加複雜的網路模型成為可能，這也促使深度學習有了進 一步的發展。

1 4

油 1. 深度學習的歷史 首先透過一張圖來概括一下深度學習的歷史浪潮，如圖l 所示。

在 Minsky 和 S即mcur Pape忱 的專著 Perceptrons 中，他們說明了單層戚 知器不能解決 XOR 問題



LeNet

BN

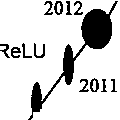
Faste『R-CNN

Residual Net

2015

GoogLeNet VGGNet R-CNN

Dropout Alex Net



2010

萬能逼近定理 吧且

Xavier

深度信念網路

反向傳播算法

圖 1.1 深度學習的發展史

透過這張圖，我們可以很明顯地看出深度學習從發展到喊起經歷了兩個低 谷，這兩個低谷也將深度學習的發展分為了三個不同的階段， 下面分別說明這三 段歷史。

第一代神經網路 （ 1958年－1969年 ）

最早的神經網路的思想起源於 1943 年的 MCP 類神經元模型，當時人們希望 能夠用電腦來模擬人的神經元反應的 過程，該模型將神經元簡化為三個過程： 輸 入訊號線性加權，求和，非線性敢動 （ 設定值法 ）， 如圖 1.2 所示。

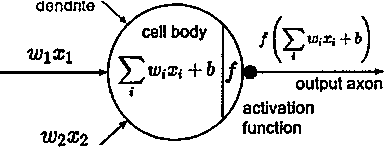
*Wa*



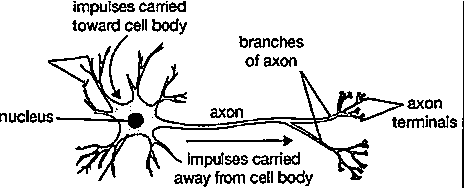
目onnom a 刪*roe* 、

a 片 synapse

包*1oxo*



叫出dY



dend而＇＂

圖 1.2 神經元獲型

第一次將 MCP 用於機器學習 （ 分類） 的當屬 1958 年 Rosenblatt 發明的感知 器 （ per－閃閃on ） 演算法。該演算法使用 MCP 模型對輸入的多維資料進行二分 類，且能夠使用梯度下降法從訓練樣本中 自動學習更新權重。1962 年，該方法被 證明為能夠收斂，它的理論與實作效果引發了第一次神經網路的浪潮 。

然而學科發展的歷史不總是一帆風順的。

1969 年，美國數學家及人工智慧先驅 Minsky 在其著作中證明了戚知 器本質 上是一種線性模型，只能處理線，生分類問題，就連最簡單的 XOR （ 亦或） 問題都 無法正確分類。這等於直接宣判了戚知器的死刑，神經網路的研究也陷入了近 20 年的停滯。

第二代神經網路 （ 1986 年－1998年 ）

第一次打破非線性詛咒的當屬現代深度學習 高手 Hin個n ，他在 1986 年發明了 遁詞於多層感知器 （ MLP ） 的 BP 演算法，並採用 Sigmoid 進行非線性對映， 有 效解決了非線性分類和學習 的問題。該方法引發了神經網路的第二次熱潮。

1989 年，Robert Hecht-Nielsen 證明了 MLP 的萬能逼近定理，即對於任何閉 區間內的連續函數 f ，都可以用含有一個隱含層的 BP 網路來逼近。該定理的發 現，相當大地鼓舞了神經網路的研究人員。

也是在 1989 年，LeCun 發明了旋積神經網路一一 L刮目，並將其用於數字識 別，且獲得了較好的成績 ，不過當時並沒有引起足夠的注意。

值得強調的是， 1989 年以後由於沒有提出特別突出的方法， 且神經網路一 直缺少對應的嚴格的數學理論支援， 神經網路的熱潮漸漸冷淡下去。 冰點發生 於 1991 年，BP 演算法被指出存在梯度消失問 題，即在誤差梯度反向傳遞的 過程 中 ，後層梯度以乘性方式覆蓋到前層， 由於 Sigmoid 函數的飽和特性，後層梯度 本來就小，誤差梯度傳到前層時幾乎為 0，因此無法對前層進行有效的學習，該 發現對此時的神經網路發展無異於雪上加霜。

1997 年，LSTM 模型被發明，儘管該模型在序列建模上表現出的特性非常突 出，但由於正處於神經網路的下坡期，也沒有引起足夠的重視。

統計學習方法的春天 （ 1986 年－2006 年 ）

1986 年，決策樹方法被提出，很快 ID3 • ID4 • CART 等改進的決策樹方法相 繼出現 （ 到目前仍然是非常常用的機器學習方法），這些方法也是符號學習 方法的 代表。正是由於這些方法的出現， 使得統計學習開始進入人們的視野， 迎來統計 學習方法的春天。

1995 年，統計學家 Vapnik 提出線性 SVM 。自於它有非常完美的數學理論推 導做支撐 （ 統計學與凸最佳化等），並且非常符合人的直觀戚受 （ 最大間隔）， 逐 漸成為當時的主流演算法， 更重要的是它在線性分類的問題上獲得了當時最好的 成績，這使得神經網路更陷入無人問津的境地。

1997 年，AdaBo叫被提出，該方法是 PAC ( Probably Approximately Correct ) 理論在機器學習貫作上的代表， 也催生了整合學習 （ Ensemble Learning ） 這一種 方法的誕生，在回歸和分類工作上獲得了非常好的效果。 該方法透過一系列的弱 分類器整合，達到強分類器的效果。 現在整合學習仍然活躍在傳統機器學習方法 中 ，在很多比賽中依舊大放異彩。

2000 年，Kernel SVM 被提出，核心化的 SVM 透過一種巧妙的方式將原空間 線性不可分的問題，透過 Kernel 對映成高維空間的線性可分問題，成功解決了非 線性分類的問題，旦分類效果非常好。至此也更加終結了神經網路時代， 在具有 如此多完美理論支持的方法中 ，神經網路似乎被宣告了死刑。

2001 年，隨機森林被提出， 這是整合方法的另一代表， 該方法的理論紮簣， 比 AdaBoost 能更進一步地抑制過擬合問 題，實際效果也非常不錯 ，使得機器學習 更向前邁進一步。

2001 年，一種新的統一架構一一圖模型被提出，該方法試圖統一機器學習混 亂的方法，如單純貝氏 、SVM 、隱瑪律可夫模型等，為各種學習方法提供一個統 一的描述架構，希望實現大一統的理論架構。

第三代神經網路深度學習 （ 2006 年至今 ）

該階段又分為兩個時期﹒ 快速發展期 （ 2006 年一切12 年 ） 與爆發期 （ 2012

年至今）。

﹒快速發展期 （ 2006 年－2012 年 ）

2006 年，深度學習元年。這一年，Hinton 提出了深層網路訓練中梯度消失問 題的解決方案：「 無監督預訓練對權重進行初始化＋ 有監督訓練微詞 」。其主 要思想是先透過自我學習的方法學習到訓練資料的結構 （自動編喝器 ）， 然後 在該結構上進行有監督訓練微詞。 但是由於沒有特別有效的實驗驗證， 該論 文並沒有引起重視。

2011 年，ReLU 殷勤函數被提出， 該敢動函數能夠有效地抑制梯度消失的 問題。

2011 年，微軟第一次將深度學習應用在語音辨識上， 取得了重大突滅。

﹒爆發期 （ 2012 年至今）

2012 年，Hinton 課題組為了證明深度學習的潛力 ，第一次參加 ImageNet 影 像識別比賽，透過建置 CNN 網路 AlexNet 一舉奪得冠軍，並且礦壓了第二名 ( SVM 方法 ） 的分類效能。也正是由於該比賽，CNN 吸引了許多研究者的 注意。

AlexNet 的創新點主要有以下幾點：

( I ） 第一次採用 ReLU 敢動函數，相當大加快了收斂速度，並且從根本上解決了 梯度消失的問題﹔

( 2 ） 由於 ReLU 方法可以極佳地抑制梯度消失問題， AlexNet 拋棄了 「 預訓練＋ 微調」 的方法，完全採用有監督訓練。 也正因為如此，現在深度學習的主流 學習方法也變成了純粹的有監督學習 ﹔

( 3 ） 擴充了 LeNet5 結繕 ，增加 Dropout 層減小過擬合，LRN 層增強泛化能力／ 減 小過擬合，

( 4 ） 第一次探用 GPU 對計算進行加速。

2013 年 、2014 年、2015 年，透過 ImageNet 影像識別比賽，深度學習因其網 路結構、訓練方法、GPU 硬體不斷進步，在其他領域也不斷地徵服戰場。

2015 年，Hinton ’LeCun • Bengio 論證了局部極值問題對於深度學習的影 響，獲得的結果是 Loss 的局部極值問題對於深層網路的影響可以忽略， 該論斷 也消除了籠罩在神經網路上的局部極值問題的陰霾。 實際原因是深層網路雖然局 部極值非常多，但是透過深度學習的 Batch Gradient Descent 最佳化方法很難陷進 去，而且就算陷進去，其局部極小值點與全域極小值點也是非常接近的。 而淺層 網路卻雖然擁有較少的局部極小值點， 但是卻很容易陷進去， 且這些局部極小值 點與全域極小值點相差較大。 對於這一點，論述原文其實沒有嚴格地證明， 只是 簡單敘述。

2015 年， 何愷明提出 Deep Residual Net 。分層預訓練時，ReLU 和 Batch Normalization 都是為了解決深度神經網路最佳化時 的梯度消失或爆炸問題，但是 在對更深層的神經網路進行最佳化時， 又出現了新的 Degradation 問題，「 即通常 來說 ，如果在 VGG16 後面加上許多個單位對映，網路的輸出特性將和 VGG16 一 樣，這說明更深層的網路的潛在分類效能只 可能大於等於 VGG16 的效能，不可 能變壤，然而實際效果卻是： 如果簡單地加深 VGG16 的話，分類效能就會下降

（ 不考慮、模型過擬合問題）」。

Residual 網路認為這說明深度學習網路在學習單位對映方面有困難， 因此設 計了一個對於單位對映 （ 或接近單位對映） 有較強學習能力的深度學習網路，相 當大地增強了深度學習網路的表達能 力。使用此方法能夠經鬆地訓練高 達 150 層 的網路，使得深度學習成為了名副其實的 「深度」 學習 。

油 2. 深度學習結構

隨著神經網路的發展， 目前比較流行的網路結構分別有： 深度神經網路 ( DNN ）、 旋積神經網路 （ CNN ）、 循環遞迴神經網路 （即心）、 產生對抗網路 ( GAN ）， 等等，本書會逐一詳細介紹，在此不再贅述。

/ 1.3 ／ 學習資源與建議

隨著近年深度學習的興起，很多研究者都投入這個領域當中 ，由於各個大學 都將自己的課程放到了網上，出現了很多學習資源和網路課程，而且很多大公司 如 Google • Facebook 都將自己的開放原始碼架構放到了 Github 上，使得深度學 習的入門越來越簡單， 很多麻煩的重複操作已經透過架構簡 化了，這也讓我們每 個人都能有機會接觸深度學習 。

網上有各種各樣的學習經驗分享， 有的人注重理論知識的累積 ，看了很多 書，但是動手實作的經驗為 0﹔也有一些人熱衷於程式的實現， 每天學習別人已 經寫好的程式。對於這兩種情況， 我認為都是不好的，深度學習是理論和工程相 結合的領域，不僅需要寫程式的能力強， 也需要有理論知識能夠看懂論文， 實現 論文提出來的新想法，所以我們的學習路線應該是理論與程式相結合，平衡兩邊 的學習工作，不能出現只管 邊而不學另外一邊的情況， 因為只有理論與程式兼 顧才不至於一旦學習深入，就會發現自己會有很多知識的漏洞 。

在學習本書之前，需要一定的 Python 語言基礎和微積分、線性代數的基礎， 下面將列出開始閱讀本書之 前的學習建議，以及讀完本書後繼續深入了解深度學 習領域的學習建議。

在開始本書之前 ，對於微積分和線性代數需要掌握的知識並不多， 對於微積 分只需要知道導數和偏導數，對於線性導數只需要知道矩陣乘法就可以了。

對於 Python 語言，列出三個學習資源，學完第一個學習資源之後就可以沒有 障礙地閱讀完本書，後績的兩個學習資源幫助你更深入地了解 Python 語言及其數 值計算。

( I ） ﹛笨方法學 Python﹜ （*Learn Python the Hard* 肌O')

本書針對零基礎的學者， 透過一系列簡單的實例快速入門 Python 的基本 操作。

( 2 ） 廖雲峰的 Python 入門

這個系列教學可用來更全面地學習 Python，掌握前幾章的 Python 基礎即可， 後面講授的部分是更為專業的 Web 開發的內容，對於機器學習而言不需要學 握這些部分。

( 3 ) *Edx: Introduction to Computer Science and Programm* 峙 的*ing Python*

這是 MIT 的公開課，以 Python 作為入門語言 ，簡潔、全面地說明了電腦科 學的肉容，適合更進一步的學習。

以上的 Python 入門課程只需要看完第一個學習資源便可開始 本書的閱讀。下 面列出讀完本書，以及在本書的閱讀過程 中的一些參考學習資源 。

( I） 線性代數

線性代數相當於深度學習的基礎， 深度學習裡面具有大量 的矩陣還算，而且 線性代數的一些矩陣分解的思想也被參考到了機器學習 中，所以必須熟練掌 握線性代激。可參考以下資源學習：

《線性代數應該這樣學｝ *(Linear A*信*Bbra done right)*

MIT 的線性代數公開課

*Coding The M*σ*, trix*

( 2 ） 機器學習基礎雖然深度學習現在很火， 但是也需要掌握其根本 ，即機器學 習 ，這才是本質與核心。這裡的學習資源從易到難排列 ：

Coursera 上 Andrew Ng 的機器學習入門課程 林軒固的機器學習基礎和機器學習技法 Udacity 的機器學習奈米學位

周志華著的 《機器學習﹜ 李航著的 ﹛統計學習方法﹜

*Pattern Recognition and Machine Learning*

一一－第 1章 深度學習介紹

( 3 ） 深度學習

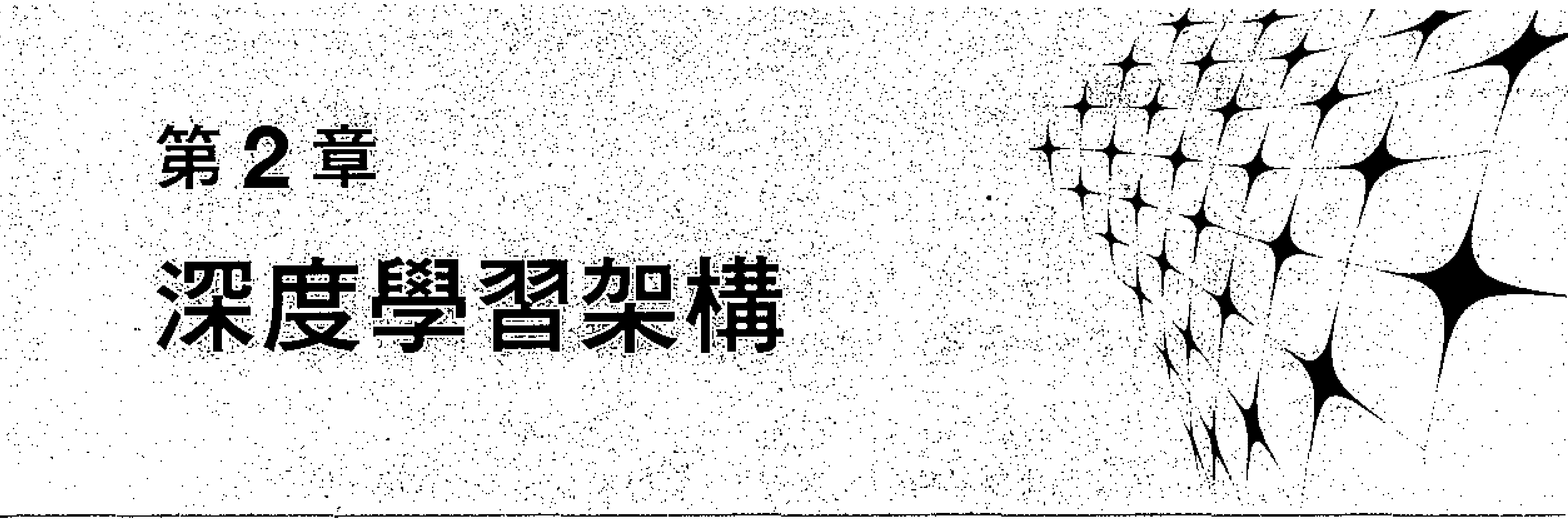
這是最近幾年最為活躍的研究領域， 爆發了很多革命性的突破， 很多前端的 學習資源，如 ：

Udacity 的兩個深度學習課程

Coursera 的 *Neural Networks* 戶r *Machine Learning*

Stanford 的 cs23 I n Stanford 的 cs224n

以上的學習資源很多， 可以學習及參考， 很多課程都是理論與實作相結合 的，透過上面資源的學習，可以完成深度學習領域絕大部分的知識儲備。



在開始深度學習專案前，選擇一個合適的架構是非常重要的事情， 因為選擇 一個合適的架構能讓你事半功倍。 目前研究者使用了各種不闊的架構來達到他們 的研究目的，也從倒面印證了如今深度學習頡域 E花齊放 ，

這一章，我們將介紹一下各個大企司的課度學習開放原始喝架構， 隨後注重 介紹本書的主角 PyTorch’最佳介紹一下如何安裝 PyTorch 將你的電腦設定成一 台能調進行揮度學習的機器。

1 2.1 I 深度學習架構介紹

在深度學習初始階段， 每個深度學習研究者都需要寫大量的重種程式 e 為了 加強工作效率，這些研究者就將這些程式寫成了一個架構放到網上讓所有研究者 一起使用，接著網上就出現 7不同的架構，隨著時間的演撞，最為好用的聽個架 構棋大量的人使用進一步流行 7起來，首先讓我們來介紹一下目前全世界最為流 行的轟夫課度學習架構。

油 Tensorllow

首先要介紹的當然是 Google 開放原始晦的 TensorFlow’這是一款使用 C++ 語言開發的開放原始碼數學計算軟體， 使用資料流程圖 （ Data Flow Graph ) 的形式進行計算，圍中的節點代表數學運算， 而圖中的線筆表示步推資科陣到

( tensor ） 之間的互動。TensorFlow 靈活的架構可以部署在一個或多個 CPU 、

GPU 的桌上型及伺服器中 ，或使用單一的 API 應用

在行動裝置中 。TensorFlow 最初是由研究人員和 Google Brain 團隊針對機器 學習和深度神經網路進行研究而開發的， 目前開放原始碼之後幾乎可以在各種領 域適用。

目前 Tensorflow 是全世界使用人數最多 、社區最為龐大的架構， 因為 Google 公司出品，所以維護與更新也比較頻繁，並且具有 Python 和 C＋＋的介面，教學也 非常增強，同時很多論文複現的第一個版本都是以 Tensor咀ow 寫為基礎的，所以 是深度學習界架樁預設的老大。

由於其語言過於底層， 目前有很多以 Tensor咀ow 為基礎的協力廠商抽象函數 庫將 Tensor咀ow 的函數進行封裝，使其變得簡潔， 目前比較有名的幾個是 Keras ’ Tfleam • tfslim’以及 TensorLayer •

Caffe

和 Tensor封ow 名氣一樣大的是深度學習架構 Ca伍 ，由加州大學柏克萊的 Phd 賈揚清開發，全稱是 Convolutional Architecture for Fast Feature Embedding 是一 個清晰而高效的開放原始碼深度學習 架構，目前由柏克萊視覺學中心 （ Berkeley Vision and Leaming Center BVLC ） 進行維護。

從它的名字就可以看出其對於旋積網路的支援特別好 ，同時也是用 C＋＋ 寫 的，但是並沒有提供 Python 介面，只提供了 C＋＋的介面。

Ca位 之所以流行，是因為之前很多 ImageNet 比寮裡面使用的網路都是用 Caffe 寫的，所以如果你想使用這些比賽裡面的網路模型就只 能使用 Ca悔，這也 就導致了很多人直接轉到 Ca能 這個架構下面。

Ca低 的缺點是不夠靈活， 同時記憶體佔用高，只提供了 C＋＋的介面，目前 Ca臨的升級版本 Caffe2 已經開放原始碼了，修復了一些問題，同時工程水準獲 得了進一步加強。

Theano

Theano 於 2008 年誕生於蒙特宴理工學院，其衍生出了大量深度學習 Python 軟體套件，最著名的包含 Blocks 和 Keras 。Theano 的核心是一個數學運算式的編 譯器，它知道如何取得你的結構，並使之成為一個使用 numpy 、高效本機函數庫 的高效程式，如 BLAS 和本機程式 （ C＋＋） 在 CPU 或 GPU 上盡可能快地執行。 它是為深度學習中處理大型神經網路演算法所需的計算而專門設計的， 是這種別 庫的首創之一 （ 發展始於 2007 年 ）， 被認為是深度學習研究和 開發的業界標準 。

但是目前開發 Theano 的研究人員大多去了 Google 參與 Tensorflow 的開發， 所以某種程度來講 Tensorflow 就像 Th間的 的孩子。

己過 Torch

Torch 是一個有大量機器學習演算法支援的科學計算架構， 其誕生已經有十 年之久，但是真正起勢得益於 Facebook 開放原始喝了大量 Torch 的深度學習模組 和擴充。Torch 的特點在於特別靈活， 但是另外一個特殊之處是採用了程式語言 Lua ＇在目前深度學習大部分以 Python 為程式語言的大頭境之下， 一個以 Lua 為 程式語言的架構具有更多的劣勢 ，這一項小眾的語言增加了學習使用 Torch 這個 架構的成本。

本書的主角 PyTorch 的前身便是 Torch’其底層和 Torch 架構一樣，但是使用

Python 重新寫了很多內容，不僅更加靈活，支援動態圈 ，也提供了 Python 介面 。

MXNet

MXNet 的主要作者是李沐’最早就是幾個人抱著純粹對技術和 開發的熱情做 起來的興趣專案，如今成為了亞馬遜的官方架構，具有非常好的分散式支援， 而 且效能特別好，佔用顯存低， 同時其開放的語言介面不僅有 Python 和 C抖 ，還有 R’Matlab Scala JavaScri阱，等等，可以說能夠滿足使用任何語言的人。

但是 MXNet 的缺點也很明顯，教學不夠增強 ，使用的人不多導致社區不大， 同時每年很少有比賽和論文是以 Mxt寸et 實現為基礎的，這就使得 MXNet 的推廣 力道和知名度不高 。

I 2.2 I PyTorch 介紹

2.2.1 什麼是 PyTorch

透過前面一節我們已經了解到 7錢款主流的課度學習架構，下面我們將重點 介紹本書的主角一－PyTorch •

P）晶rch 是 Torch7 團障閱讀的，從它的名字就可以看出， 其與 Torch 的不同 之處在於 PyTorch 便用7 Python 作為開發語言。所謂 「 P抖100 宜rst J’同樣說明它 是一個以 Python 優先的深度學習架構，不僅能夠實現強大的 GPU 加速 ，同時還 支援動態神經網緒，這是現在很多主混架構冊如 Tensorflow 等都不支援的。

PyTorch 既可以視為加入了 GPU 支握的 numpy 同時也可以看成一個攘有自 動求導功能的強大的深度神經網路， 除了 Facebook 之外，它還己種植 Twitter 、

CMU 和 Salesforce 等機構採用，如圖 2.1 *filr*示 ，

Companies & Universities Using PyTorch

facebook W 也nVIDIA •

art ch '11o1e

晶 ﹔

﹜'i'. :tl眉 r:> 唯諾甜， Uninirsily

Digi I

雄性也自 -=R間soning

蟬單訓2＆間問 Z卓＝： 悔

品也p

Am…

圖 2.1 PyTorch 使用廠商

* + 1. 為何要使用 PyTorch

面對如此多的深度學習 架構，我們為何要選擇 PyTorch 呢？ Tensorflow 不是 深度學習架構預設的老大嗎， 為什麼不直接選擇 Tensor咀ow 而是要選擇 PyTorch 呢 ？下面分 4 個方面來介紹為何要使用 PyTorch •

( I ） 掌握一個架構並不能一勞永逸， 現在深度學習並沒有誰擁有絕對的壟斷地 位，就算是 Google 也沒有，所以只學習 Tensorflow 並不夠。同時現在的研究 者使用各個架構的都有， 如果你要去看他們實現的程式， 至少也需要了解他 們使用的架構，所以多學一個架構， 以備不時之需 固

( 2 ) Tensor咀ow 與 Ca位 都是指令式的程式語言，而且是靜態的，首先必須建置一 個神經網路，然後一次又一次使用同樣的結構， 如果想要改變網路的結構， 就必須從頭開始。但是對於 PyTorch’透過一種反向自動求導的技術 ’可以 讓你零延遲地任意改變神經網路的行為， 儘管這項技術不是 Py Torch 獨有， 但目前為止它實現是最快的，能夠為你任何瘋狂想法的實現獲得最高 的速度 和最佳的靈活性， 這也是 PyTorch 比較 Tensorflow 最大的優勢。

( 3 ) PyTorch 的設計想法是線性、直觀且易用的， 當你執行一行程式時，它會忠 實地執行，並沒有非同步的世界觀，所以當你的程式出現 Bug 的時候，可以 透過這些資訊輕鬆快速地找到出錯的程式， 不會讓你在 Debug 的時候因為錯 誤的指向或非同步和不透明 的引擎浪費太多的時間。

( 4 ) PyTorch 的程式相對於 Tensor宜 w 而言，更加簡潔直觀， 同時對於 Tensor宜ow 高度工業化的很難看懂的底層程式 ，PyTorch 的原始程式碼就要友善得多， 更容易看懂。深入 AP’了解 PyTorch 底層一定是一件令人高興的事。 一個 底層架構能夠看懂的架構， 你對其的了解會更深。

最後，我們簡要歸納一下 PyTorch 的特點：

﹒支援 GPU ﹔

﹒動態神經網路﹔

* + - * Python 優先﹔

﹒指令式體驗﹔

E 輕鬆擴充 。

擁具有如此多優點的 .PyTorch 也具有它的缺點，首先 PyTorch 於 2017 年 3 月 開放原始喝發佈， 目前還是 beta 測試版，沒有發佈正式版本，所以可能有一些小 的 Bug﹔其次因為這款架構比較新，所以使用的人也就 比較少，這也就使得它的 社區沒有那麼強大， 但是 PyTorch 提供了一個官方的討論區，大多數碰到的問題 都可以去裡面搜索， 裡面的答案一般都是由作者或其他 PyTorch 使用者提供的， 討論區的更新也特別頻繁，同時也可以去 Github 上面提 Issue ，一般很快就會獲得 開發者的回應 ，也算是某種程度上解決了社區的 問題。

I 2.3 I 設定 Pyli。rch 深度學習環境

透過上面兩節，我們簡要地了解了現今深度學習的主流架橋，以及 PyTorch ’ 下面就開始打造 PyTorch 的深度學習環境。

2.3.1 作業系統的選擇

首先需要選擇電腦的作業系統， 現在一般有三種作業系統： Windows 、Mac 和 Linux 這才里只能在 Mac 和 Linux 之間選擇，這是因為 目前 PyTorch 只支援這 兩種作業系統，Windows 作業系統的支援應該會在未來實現， 但是現在還不支 援，所以如果要學習 PyTorch’就只能選擇 Mac 和 Linux 。

如果你有一台蘋果電腦 ，那麼你就可以直接開始下面 Python 環境的設定了 ﹔ 如果你是 台Windows 筆記型電腦或桌上型電腦，這裡有兩種選擇：第一，安裝 Linux 虛擬機器﹔第二，安裝一個雙系統。這裡推薦安裝雙系統， 因為虛擬機器 下的體驗不是很好， 雙系統也不需要佔用太多的電腦 空間。對於 Linux 系統，推 薦使用 Ubuntu，在網上能夠找到很多的安裝教學， 同時使用者較多，如果遇到問 題，容易在網上找到很多解答辦法。

2.3.2 Python 開發環攝的安裝

接下來我們需要一個好用的 Python 的聞發環境，對於 Python 而言，比較 頭疼的就是不同版本環揖管理與套件管理。 針對這些問題，有不少發行版本的 Python ，倒如 WinPython 和 Anaconda 等，這些發行版本將 Python 常用的包包裝， 方便使用者直接使用而不需要再重新安裝，這裡推薦使用 Anaconda 軟體管理包。

那麼 Anaconda 是什麼呢？ Anaconda 是一個用於科學計算的 Python 發行版 本，主揖 Linxu 、Mac 和 Windows 系統，提棋了套件管理與環揖管理的功能 ，可 以很方便地解訣多版本 Python 並存、切換，以及各種協力廠商包安裝的 問題。

Anaconda 的下載可以進入其官網如圖 2.2 所示的 https://www.continuum 品／ downloads •

.MICGl'ld 口回d E曲mw,11t41:旭a -se.ird! Cc,n扭＂

o自也扭曲

PROOU （祖 !i.'LIPP.回國T&S0C.UTION$ to訕MUNIIY Al!IOIJf IIISDU眩目

.,

DOWNLOAD ANACONDA NOW

- ....

叫＂＝＝－區A]

* + - * + *A.*

GET SUPERPOWERS ：〉

， Whldl.鴨E刮目曲曲＂＇d 兩間. Ml 恤tllll M山晶晶....祖祖uun(llnm叫自扭扭咽回祖 llllil:lf

WITH ANACONDA t:,.‘ 快

...扭曲ln.iml.il刮目吋間隔......扭曲祖國國官

甜甜輛自扭曲 岫a 『四咱”們曲.，.；酬 帥... ..........冉曲...,,. d曲咽師團叫岫岫E即由啊 甜甜..... ..山揖

h ‘－ 可

”n 圓圓間lionol'AII圓圓＇＂間fonn;1intl!',distrlbut:lain.11f ，開 個前 4個凡... 叫圓..,....臨k肉陣boo andR1ndiiKlu由巴研討t四ofth1m個k 阻＂＂＂阿甘曲＂＇＂ k血間 呵輯 n!l'$k!們.

可 可

，即由臼區司MO.

『F申u d回l"tlantimHJdid：崢斟 叮...酬...

配，；.回 ..扭曲岫胃圖暐起個帥 「＇ ＂＇旬 輔 W,ta,i 個，；.. ..,... 祖國＂＇棚 峙Mini«!咱叫抽間恤圖＂＇＇＇

’，*A.*

四.，. 圓圓再叫lrl!ntNl'lll吋•輯隘 咱也個咽 『'ICY ill唔 r-a,nm研官－田間 ＂ 開呵曲直..內臨an.TIM!nin 由 ，，u』＂＇叫晶晶 『F

祖國lslrid1;1曲dlnANi由吋a 缸＃出•o><"OH;,a呵 dwi曲M 祖祖H甜甜唱 阻岫曲，.，.陣申＂＇甜可..扭曲扭曲!Mllncl. *4'"*

......... 祖＇＇＂＂

可＇ .'

*-q* J戶 ，

圖 2.2 Anaconda 官網

點擊圖 2.2 上的框東選擇 Mac 或 Linux • 進入之接可以看到如圖 2.3 所示的介 面。在圖 2.3 上的框中選擇 Python 版本，注意這裡是預設系統為 64 位元版本的， 一艦現在的電腦都是 64 位版本，但是可能會有 32 位版本的電腦，也可以在中間 找到對應的版本。

對於 Python 版本，可以選擇 2.7 版本，也可以選擇 3.6 版本，這裡推薦使用

Python 3.6 版本。本書中的全部程式都是以向咄on 3.6 版本來寫起基睦的。如果你

使用 2.7 版本，可能會出現一些小的問題。雖然 Python 2.7 是穩定的版本，但是隨

著 3.6 版本的不斷更新， 不久之後 2.7 版本一定會被淘掉，版本更是是未來的趨

興。

oown岫甜 fot WindOI阻

I Anaconda 4.4.0

i cr lim』X

Downl 回dfot,o缸OS : 00叫 loadf0<Linu,

－

門thon 3.6version

目IT(POWER8﹜ INSTALLER （扭曲帶

︱a輛也』圓圈 h個d酬帥 闢間開m扭.，...國岫團由

I －曹自巾， ldfarndi'llti』.....

64

I ll>"golog

1 z1.日Drow.nload 叫自回，；ty .，；曲 MDSor

＂圳咐岫h

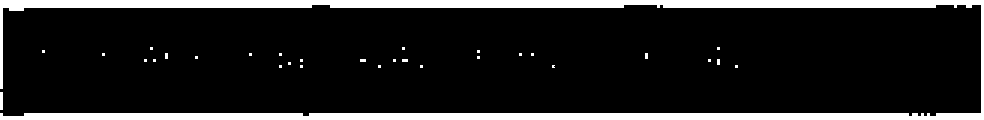
酬 '56 Moro

32 茵TlN訂AL旺R(42BMI

1.，，，，，峙＂＂＇崎1.,;ndow 旬阻。......晶晶 l酬a咽lollowtht!

i阻回國。間 '

' P)'lh•臨u....r••



thonZ.7 四.. .

i

內的on 2.7時時Jon

圈圈圈噩噩團軍區圖E

I ---- I I 64-BIT （叫酬間TALL帥m

i .......啞lt;ld•W"IIMh."mm＂＇＂呵研個W 帽”..到呵 扭曲曲a且 自

I I 32·前TIN訂A心血R(4時間

###### L一 一 一一 一一 一 一一一」

圖 2.3 選擇 Python 版本

選擇好之後點擊對應的圖示就能夠開始下載了。 選擇框中的 Anaconda 選項，進入如圖 2.5 所示的介面。

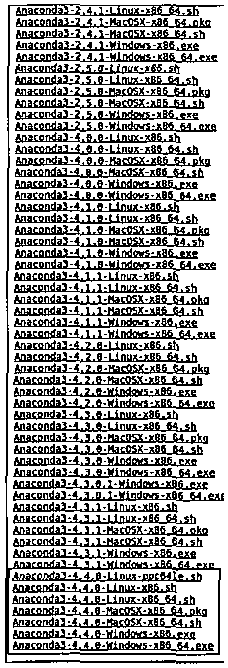
|  |  |  |
| --- | --- | --- |
| Index of /anaconda/ | | |
| E且坦!L I | 81-Jun-2817 15:SB | ． |
| 且!!Iinic!nda/ | 23-Jan-2817 14:SB  自1-Jun-2617 16:28  84-Apr-2916 89:118 |  |
| 且fad aackages. txt | 22-Mar-2817 18:16 | 2618 |

圖 2.5 Anaconda 單單

2.3 設定 PyTorch 深度學習環璋 圖 －

點擊圖 2.5 所示框中的 archive 就能夠看到所有的 Anaconda 安裝套件了，滑 動滑鼠到最下面能夠看到最新 版的安裝程式，如圖 2.6 所示。

C 7015 2 血＂＂＇



.. ＂一m

r

叩瞄 M 帥”， 開心目，，“ 血m

a 恥＂，，個 呵呵

，恥叫別2 3叩叩

＂抽－281& 21 “ 翩翩叩

…岫叫 ”啪”

岫－

2816 21<'1 lll72746J

昀 祖＇＂21 ““5圓圓

F

圈 岫剖16 21:U J1H6l4酬

叫叫叫 JJ2&•n•

2叫叫16 16:15 *un•*蝴

叫C 抽 血 祖向

＂6 18'16 叫“

詢 祖 －

et& 16:16 阻m岫

扭岫2816 16 28 M"2•o.49J，U，

n

柚

…” 16:28 叫車的

n岫胡6 16:28 扭扭岫

祖岫－2<116 16:28 n抽血

＂叫2悼16 16:28 Z酪制”

叫

2祖 祖時也

e,5

Jut-,n• 1•:a 4 間

間

．心 ：n 叩瞄

‘

ul-2816 1•:21 J132lnl詔

”－Jul-2818 16現 ,0011..2

．國 16:21 178955728

2,1.,010

27-""P 2016 28:58 J9叫“

品Sep 2816 28:58 明＂＇＇

,, S隨時p-2816 28:58 Z甜”岫

扭曲

，， ＇ 叫2妞”

肘，

2日叫口扭曲 叩開

2日叫 且 扭曲 叫臨

別a ·'"" ai:S, ＂嘲

M叫＂＇＇ 甜的闢．

"F的一拙 1悶 悶糊

油恥1c-Ml1 1且7 “

a筒 ＂＇

的…叫 “…

”m 口曰喝m “… ”n .，； z叫帥

口 ＂閻明凰

”m”lac-2'11n7 口”“＝....

恥凹

叫，2811

19 13 ,..叫

19:2' 叫由＂

18-May 2017 叫 叫岫

祠 啪”也5 叫，，，，，

m骨間 1叫 扭曲M

祠 拙

E li:26 37979叩

＂峭間 車 到 4間間

圖 2.6 所有 Anaconda 安裝套件

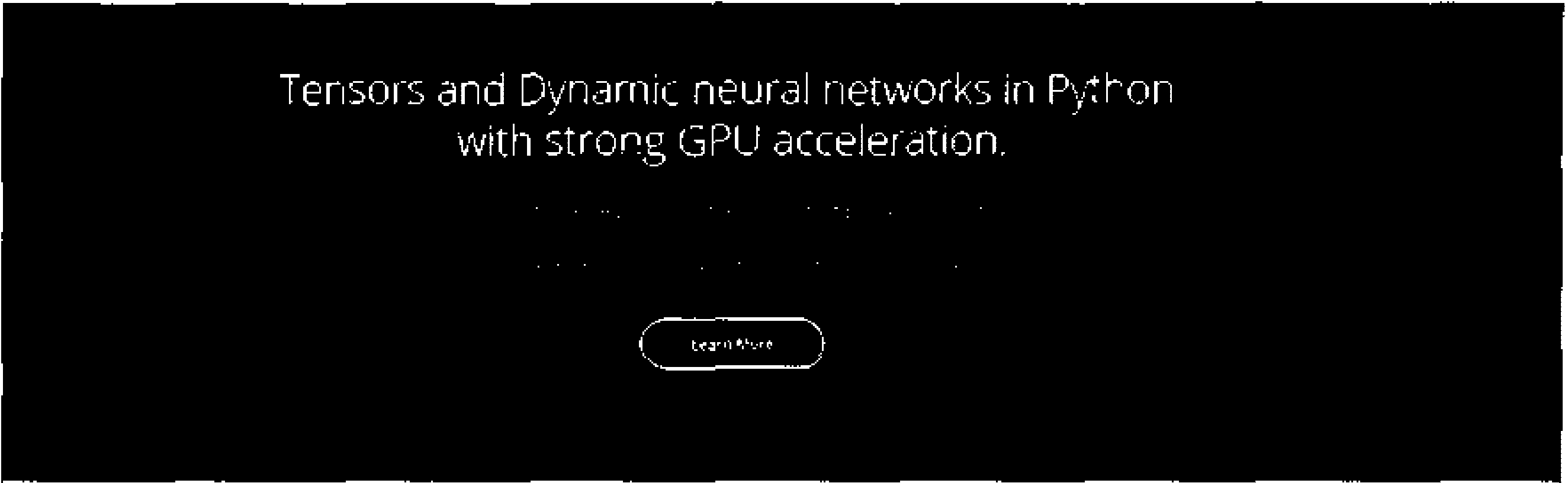
選擇圖 2.6 中框中對應系統的版本就能夠下載最新的 Anaconda 了，等待下載 完成，就可以開始安裝了。 之後如果有需要的套件而 Anaconda 沒有的話，可以 透過 pip 或 conda 來安裝，例如要安裝 numpy 就在終端裡面輸入指令 pip install numpy 或 conda install numpy 即可，像 numpy 這樣常用的套件 Anaconda 裡面是附 帶的。

2.3.3 PyTorch 的安裝

安裝完 Python 環境之後，我們就進入了深度學習函數庫 PyTorch 的安裝。首 先進入 PyTorch 的官方網站，輸入網址 [http://pytorch.o](http://pytorch.o/)嗯，可以進入如圓 2.7 所示 的介面。

在框中可以選擇要安裝的版本，注意如果你的電腦沒有主握的顯示卡可以進 行 GPU 加速，那麼 Cuda 這個部分就選擇 None 韓、後在終端中執行左下角框中的 教述，這樣就可以順利安裝 7 。

如果你有對應的顯示卡， 那麼我們需要先安裝 Cuda 才能鉤使用安裝 GPU 版 本的 PyTorch 0



Get Started. 自． 亡三王二3

Se'ew,,"'O'd 們Cc!., lc- r;.,er叮

叮Torct, nt.1’

(Ofl'.－，唱

p:ii,晶揖 e阻urel山＂＇＂『eon t!UI! 旭＂＂＂＂d

n叫間申y ，...， 屑

An.:lζc咽3旭間Hecomme,附＇ ＂＂ －™＇＂

呵呵 －c王三亡三D

.,.... C王：＝） －亡三三 叫 亡三：＝）－

圓圓圓 叫岫叫“呵呵... .,

圖 2.7 PyTorch 官網首頁

對於 GPU 版本的安裝，會稍橄樓雜一點，大概需要以下聽個步聽：

( 1） 安裝 Nvidia Cuda ﹔

(2 ） 安裝 CuDNN ﹔

( 3 ） 安醫 GPU 版本 PyTorch ﹔

( 4 ） 測試。

1.安裝 Nvidia Cuda

首先需要確認你的電腦顯示卡已經安裝好了驅動並且是支援 Cuda 的，對於 Linux ，顯卡驅動的安裝非常簡單， 進入系統設定 ，選擇圖 2.8 中框中的系統設 定。然鐘就可以看到如圖 2.9 所示的介面 ，

2.3 設定 PyTorch j單度學習環瑋 圓 －

p........,••

．

國 軍 Ii 連串 IP

Appe.aral\ce Brightness& Language Online E間＆ Te暐en甘y Lack Suppart Accounts Privacy

Ha『'llw甜E

＠ 哥哥 理軍 區區 已 國 會

t

Bluetaath c:alor Display軍 K旬board Mome& Netw。rk Power

。

晶 亡＝＝

Touchp剖

Print•『S Sound Wac。mr.iblet

System

I

I 結』

＠



Backups Deta11s



so!&

＠ 品

Universal User

Access Accounts

圖 2.8 系統設定

的U耐晶伽圖阻 自由er甜岫re Up血te5 Auth叫世世j噩噩孟噩噩噩世些壇巨旦控阻

t, NVIDIACOrporatlon: Unknown

。

甘由 d•吋回＂由自由＂但＇曲酬祖耐e<ldriver.

Using NVIDIAblnaiydriver叫rslon:J扭曲from nvldla·37S (pr呻巾，taiy,t甜甜﹜ QUslngx.o『g x server-Nouv回udisp旭，y driver from 盟erver-•org-vld自吶間V曲u(op開source)

*Q* Unknown: Unknown

。，開 曲 峙

Th dewf i,ulingan eroat d M!f.

。

u,i曲 Proc眉目mlao曲de "rmware forIntelCPUs fromIntel耐aocode (proprietar諦

Donotuse帥 d酬1個

Vou 叫個「臨時出E叫P蜘to comp恤 thed抑自nge5. 區亟百 亟亟區函互i

Apropr個呵d巾etlias 帥血，ode 血E Ubl岫車間kip,,扭曲，屯，啤w•『1npr......個fity圖doth 『啊at..臨血pendenton 闆體 driwr 酬曲，．

仁亟己

圖 2.9 選擇顯示卡驅動

然撞在裡面選擇對應提供的顯示卡驅動等待安裝完成 即可 。

接著需要安裝 Nvidia Cud且 ，首先進入 https://developer.nvidi在.com/cuda-downloads

這個網址，然扭扭著重日圖 2.10 所示的介面選擇下載按鈕就可 以等待下載7 ·

．回－ 第 2 章 深度學習架構

下載完成接進入到下載檔案 的目錄，撞著按照圖 2.10 所示的框指令安裝就可以 了。然能進入指令介面跟著指示執行即可，但是要、注意圖2.11 所示的這兩個地方。

I O O Q I

”，＝、Comp """'', CUDAT lkot> CUDAT，開岫恥...，nc岫d

s,L«t '""'•"'"""' below Lo d,..,..,•••1呵l叫，

11!!!19 ””譚嗯 冒司 ．．

Cilek oo the Y＇＂＂恤lion, Iha! dmrib< ，.「 t呵 ，p個Llono. D,ly ,uppood pC.tfoon, wiO b• ,.,w民

圓圓 圓圓 E圓

岫＂帥”。 圓圓 圓圓

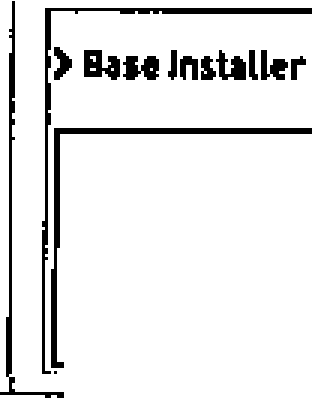
E圖 －圓 圈 圓圓 噩噩 圖．

．回國

叫

Ill 且 圓圓

n,,....'"'''""；，剖面扭曲峙，111rdowr叫回d *,,,,w.*



h訓 1妞”。＇＂＇＂＇＂＇d＇回＂＇

•.•,11

＇·＂＇＂＇四曲sbcu血－ ···＂－＂＇·＂－＇＇＂揖仙”可

＿’

*ho* <0mm,.d•LI" p間

mpl•

圖 2.10 聞抽下載

re,,o El,L ＇

— u it

口口 , :,u d(( ept t he p r >?nou sl./ ac cep士. dec l me ac <: e仁t

＇＂·＇閉·F咀 －·.... －－－圖喧－－－－－－－ －n•圓圓區 圓圓圓－－..:a-...---n.....國＿＿..

，

↓

Jef a, l lS

a \ 1. ti· 亡 （；pc.n 亡 L ; 卓.： 、句 吧J .

, es ﹞

DQ ';OCI

· , , es

, ,·, , o

‘I

θ 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| .,2.1•, |  |  | 芋 斗 － | ·st |
|  | * q | | ，υl t | ﹝ |

’O’』

E

t

，弓 $ s

The p 「＜？ e,1st L

門

＼弓，1-Jt

t e u

于 h 斗$ 「 J.1.1 υl oa te

i.s 1.1 ,e•J

Th1:. ·)P弋斗 2肉 早 ho1J

te吶 叭 呵 οr, f ι正i叫 a哇 L口0 斗 1. e Z亡l t h 主t t h 告 ？」 ι 9 貫 三＝叭 才 9u ,-.a t 1c內 卡工 1 e ι，1l. l 仁佇 t ac ,·eci 4 扎

ti

＇

：

.

ν

：0

ms

* .t

、企

主芒•］ .釘 主） S t e•S t,o a 電 ，·e也 • t11 " a c•., s tocr

，

月

DRIii

,: -,n t 斗S·-• ra tion su :1' a s 古· s te

, , • es r. ·:.. , ＂＂＇＂電 ﹝ def a,.1.t 1,

祖國眉圖盟國圓圓宙

E n te,· 明凸 l • Ct 」二三主哇斗弓 ，，

。

片 r 1., l. 哇 λp 1, 主＂－ ,,e,c J,:, t s

﹞

﹛ 苦企 ↑a,..1t 斗 S u s ,·. 1\_.:,c al. :,\_,,j司 令占 (I )

1.-J－祖國•－－＝1.·－－－－－ 噩噩噩 －－圖用••iii圈圈 團 圈圈-=:!II•

Ill E圓圓圓－

.., 5 t

亡 ，·.a ,

l 去：有 1

圖 2.11 指令介面

2.3 設定 PyTorch 深度學習環攬

一定不要安裝 Open GL ’這個地方要選 r 否則有可能在安裝完成接無法敢 動圓形化桌面，之後重新敢動就可以了。

接著增加環境變數，在根目攝下開敢終端，接臨下面指示進行輸入：

1 串 串udo vim ～／ .bashyrofile

2

3 在開廠的文字尾端加入：

1. export LD\_LIBR且RY\_PATH="$LD\_LIB阻RY＿阻四 ／ usr/local/cu也／lib64:/usr/local/ cuda/extras/CUPTI/lib64 ”
2. export CUD且＿HOME=/usr/local/cuda

這樣 Nvidia Cuda 就安裝完成了，接下來需要安裝 cuDNN •

2過 2. 安裝 cuDNN

進入 加ps://developer.nvidia.com/rdp/cudnn-download 這個位址進行下載， 下 載前需要註冊登入，註冊完成之後登入可以進入如圖 2.12 所示的介面。

I

nv1

…

叫。 PcR

Hc"'t • Ccim u1eW目rk• • De.pt且也m;•o • s ，.，，1• cu'CIN:N DIM-咽·岫·

cu DN N Download

NVIDIA cu CNN js l!I G間 ollCCl!Lli!甜甜 U抽句叫 1•lmJ1•"'"'"'n間nlnll!'twarlcl.

國 1,,...τ s,1,..,u.. ..''"'"'"

....... 。臨•＂"'"",,,..'""""

曲＂＇＂＇＂酬 兩d阻menl1t區on 1ode1Hmlnlll the- f"tllc:aimm 曹甜甜”ni1:1nC1tc11 酬M

..，..，們. u, ,g 回DNN with, Pmol JGT寓，由眉目 0Tll l010I, wn;., S *Ot* IOler ls o,qulo,O

If叫叫川刷27,2Gl7l:\_'\_'.'.:0UDA0.0

﹝古；互司1月三二二一

巴亟亟 王二二二二二二三二

..耐

l凡酬 ；，， lJ，叫別吼叫山間 且 *!*

L .......＿自一 一－－ － － － － － －－－ －－一－－一－一圓圓一”一『－－－ －－』且 』－曰－﹞ －」

°＇..＇＇.＇.：.＂＇.M ＇.士－ 坐直至二－－····－····－－－－·－·－－二二二二二 一 一一一＿\_:j

［＂：＇.一一些 1哩＇. ＇：＿ 互了三三三三二二二二 日一 一二二二﹞

鬥叩門叩門也已乎已 －﹔ 二二二一二三于三三二二二i

Aπhi'llf'CI 直旭日＂ '"'

圖 2.12 下載 cuDN N

然接選擇下載 cuDNN v5. l(Jan 20, 2017） 如 CUDA 8.0 ，接著選擇如圖 2.13

*J'ifr* 示框中的部好進行下載。

下載下來的是 deb 套件，然值執行 dpkg -i 就可以安裝了。

圓圓圓 － 第 2 章 深度學習架構

3. GPU 版 PyTorch

安裝 GPU 版 PyTorch 跟 CPU 版本差不多，也是先進入宮網，只不過最佳一 行選擇 CUDA 8.0 ，接照這命令列說明就可以安裝了。

凶 4. 測試

聞敵終端，輸入 python 進入到 python 介面，然接按照固2.14 輸入。

口 帆ntoad ,uONN ,5.1 [Jan 20, W17I, •or CUDA B.O

E虹口NN UmG,.do

＂口＂＂＇＂旭110υ』de

<UONN ,5 \ l;b,a,y ,o, Unux

<,ONN vS 1 l;b「司可，o,Pow 『目

＇＂心NN,5.1 l;b呵呵 lorw;ndows 7

＇＂口NN 苟且 lllb呵呵 1orw;noo＂岫

'"""" ,s., L;b,a,y!o,osx ewONN v5.1 Release Notes

＇＂必NN ,5.1 Ronhrne LiOra 可怕r ＂可＇＂＂04 I目前1

.口NN 苟且 1 Developer LiO,a呵？他「 Ut ,n!, 4 04 IOebl

c:t1DNN 兩.1 Code Sarnplos and u,er '""' [Oobl

l叫N 咱圳枷C 山a可 lor U仙仙也 Po明時附l

,uONN v5 I Omlopo, Libra 可 or Ubontul&.04 PowerB [Debi

.口NN 苟且 1 Code Samp,os and uso, a,;o, PowerB !Debi

圍 2.13 選擇 cuDNN 版本

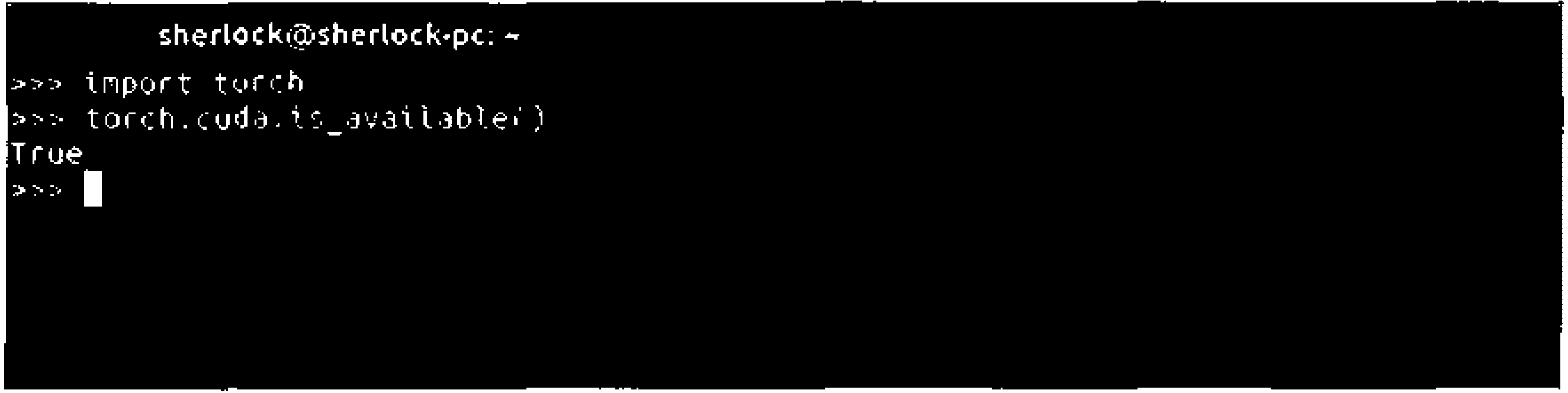


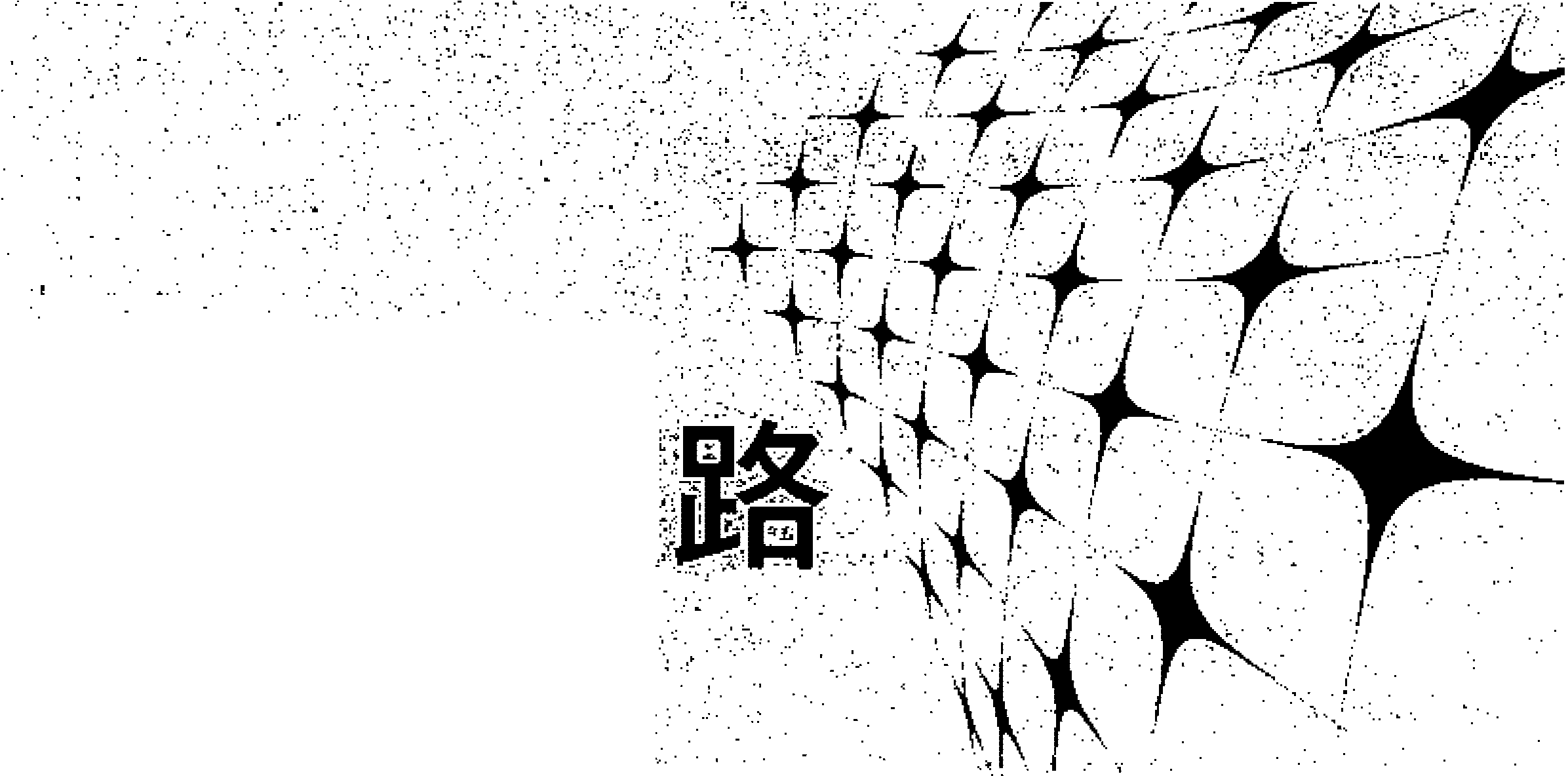
圖 2.14 揖端介E

如果最佳一行出來的結果如闡 2.14 所示的是 True • 那麼恭喜你 GPU 版本的

PyTorch 安裝成功了。

到這裡我們將按完成前導知識的介紹 ，同時也設定好了環境，下面就讓我們 一起進入 PyTorch 的世界。

第3章



多層全連接神經網

課度學習的前身便是多層全連接神體網路， 神經網路頡域最開始主要是用來 模擬人腦神經元系統， 但是隨佳運漸發展成了一項機器學習技惘。 多屆全連接神 經網路是現在深度學習 中各種裡雜網路的基礎， 了解它能詢幫助我們更進一步地 學習之後的內容。

這一章我們將先從 PyTorch 基體入手，介紹 PyTorch 的處理物件、 運算操 作、 自動＊導，以及資料處理方法， 扭著提甜 生模型開始進入機器學習的內容， 然後由 Logistic 回歸引用分頭問題，撞著介紹多層全連接神經網路、 反自傳播 擴算法、各種以梯度為基體的優化演算法、 資料前置處理和訓練技巧， 最佳用 PyTorch 實現多層全連接神經網路。

[ 3.1 I 熱身：PyTorch 基硨

首先我們會介紹一下 PyTorch 裡面的一些基體知識，有了這些基礎知輯之控 我們才能做出更多複雜的暨形。

3.1.1 Tensor （ 張量 ）

PyTorch 裡面處理的最基本的操作物件就是 Tensor ’Tensor 是張量的英文， 表示的是一個多維的矩陣， 側重日霉維就是一個點，一維就是向量 ，三誰就是一臨 的矩陣，多誰就相當於一個多誰的陣列， 造和 numpy 是對應的，而且 PyTorch 的 Tensor 可以和 numpy 的 ndarray 相互轉換，唯一不間的是 PyTorch 可以在 GPU 上

執行，而 且umpy 的 ndarray 只能在 CPU 上執行。

一 第 3章

多層全連接神經網路

我們先介紹一下一些常用的不間資料類型的 Tensor ，有 32 位浮點數 torch. Float Tensor 、64 位浮點數 torch.Double Tensor 、16 位整數 torch.Shor !Tensor 、32 位整數 torch.IntTensor 和 64 位整數 torch.LongTensor 。

我們可以透過下面這樣的方式來定義一個三行兩列指 定元素的矩障， 並且顯 示出矩陣的元素和大小：

a a 位 rch .Tens r ﹛ ﹝ ﹝ 2, 3﹞，﹝ ，8﹞ ，﹝ 7, 9﹞ ﹞ ﹜

1 2 3

print ﹛’a is ﹛ ｝ ' .f rmat ﹛a﹜ ﹜

print （ a size is ﹛ ﹜ ’主 rmat ﹝a.size (I ﹜ ﹜

# a .size () a 3, 2

需要注意的是 torch.Tensor 預設的是 torch.FloatTensor 資料類型，也可以定義 我們想要的資料類型， 就像下面這樣 ：

1 b a t rch LongTens 主 （ ﹝ ﹝ 2, 3﹞ ，﹝ ，8﹞，﹝ 7, 9﹞ ﹞ ﹜

2 prin乞 （ ＇b is ：﹛ ﹜ ’f rmat ﹛b﹜ ﹜

當然也可以建立一個全是 0 的空 Tensor 或取一個正態分佈作為隨機初值 ：

C a 七 rch .zer s ( ( 3, 2) )

1 2 3 4 5

print ﹛ Y zero tens r, ﹛ ﹜’.format (cl ﹞

d a t。rch.ran 由1﹛ ﹝ 3, 2) )

print ﹛ n rmal randon is ：﹛ I ' .f rmat ﹛ di I

我們也可以像 numpy 一樣透過東引的方式取得其中的元素，同時也可以改變 它的值 ，例如將 a 的第一行第二列改變為 100 °

1 a﹝ O, 1﹞ a 100

2 print （ changed a is ．﹛ ） ' .format ﹝a） ﹜

除此之外，還可以在 Tensor 與 numpy.ndarray 之間相互轉換：

numpy\_b a b.numpy ()

1 2 3 4 5 6 7 8

print ﹛ Y conver t numpy is \n ﹛ ﹜ ＇.f 口阻t (numpy b﹜ ）

e a np.array （ ﹝ ﹝ 2, 3﹞，﹝ ，5﹞ ﹞ ）

。

torch e a torch.fr m numpy ﹝e﹜

prin乞（’fr m numpy t t rch Tens r 1S ﹝ ﹜’.f rmat (torch e) I f torch e a t。rch e fl。at ﹛﹜

print ﹛ change data type t fl at tens r , ｛ ﹜ ’f rmat {f torch\_e﹜ ﹜

3句2

3.1 熱身 PyTorch 基礎 －間

透過簡單的 bnumpy（）， 就能將 b 轉為 numpy 資料類型 ，同時使用 torch. from\_ num py（） 就能將 numpy 轉為 tensor’如果需要更改 tensor 的資料類型，只 需要在轉換後的臼nsor 後面加上你需要的類型，例如想將 a 的類型轉換成 宜 鼠 ， 只需 a.float（） 就可以了。

如果你的電腦支援 GPU 加速，還可以將 Tensor 放到 GPU 上。

首先透過 torch.cuda.is\_available （） 判斷一下是否支援 GPU ，如果想把 tensor a

放到 GPU 上，只需 acuda（） 就能夠將 tensor a放到 GPU 上了。

1 if t。rch cuda .is available (I ,

1. a cuda - a .cuda ﹝﹜
2. print (a cuda﹜

3.1.2 Variable （ 變數 ）

接著要講的概念就是 Variable ，也就是變數，這個在 numpy 裡面就沒有了， 是神經網路計算圓裡特有的概念， 就是 Variable 提供了自動求導的功能，之前如 果了解過 Tensor咀ow 的讀者應該清楚神經網路在做還算的時候需要先建置一個計 算圖譜，然後在裡面進行正 向傳播和反向傳播 。

Variable 和 Tensor 本質上沒有區 5lU ，不過 Variable 會被放入一個計算圍中 ， 然後進行正向傳播 ，反向傳播 ，自動求導。

首先 Variable 是在 torch.autograd.Variable 中，要將一個 臼nsor 變成、variable 也非常簡單，例如想讓一個 tensor a 變成 Variable ，只需要 Variable(a） 就可以了。 我們可以透過圖 3.1 了解到 Variable 的屬性。

I 叫叫.Va岫恥 l

︱白白 i r::. ：！

i ： 悶： l

L＿ ＿ ＿二二二三三二二三 ·：... i

圖 3.1 結構圖

一 第 3章

多居全連接神經網路

從國 3.1 中可以看出 Variable 有三個比較重要的組成屬性：data • grad 和 grad fu 0 透過 data 可以取出 Variable 裡面的 tensor 數值 ，grad fn 表示的是獲得這 個 Variable 的操作 ，例如透過加減還是乘除來得到的， 最後 grad 是這個 Variabel 的反向傳播梯度，下面透過實例來實 際說明一下：

# Create Variable

。 。

1 2 3 4 5 6 7 8 9 1 1 1 1 1

1. - Variable ﹛t rch Tens r （ ﹝1﹞ ） , requires grad-True﹜ w a Vanable (torch.Tens r ﹝ ﹝ 2﹞ I, requires grad-True) b a Variable （七 rch .Tens x﹛ ﹝ 3﹞ I, requires\_grad-True)

。

# Build a c。mputati。nal graph . y a w \* x + b # y - 2 \* x + 3

# C。mpute gradients

。

1. backward （﹜ ＃ same as y backward (t rch .Fl atTens r ﹝ ﹝ l] II

。

# Print ut the gradients print (x grad) # x grad a 2 print ﹛w grad) # w grad " 1 print ﹛b grad﹜ I b.grad a 1

1 2 3 4

建置 Variable，要注意得傳入一個參數 requires grad=True ’這個參數表示是 否對這個變數求梯度，預設的是 False ，也就是不對這個變數求梯度，這裡我們希 望獲得這些變數的梯度，所以需要傳入這個參數。

從上面的程式 中，我們注意到了一行 y.backward （）， 這一行程式就是所謂的自 動求導，這其實相等於 y.backward(torch.FloatTensor （﹝ l﹜））， 只不過對於純量求導裡 面的參數就可以不寫了 ，自動求導不需要你再去明確地寫明哪個函數對哪個函數 求導，直接透過這行程式就能對所有的需要梯度的變數進行求導 ，獲得它們的梯 度，然後透過 x.grad 可以獲得 x 的梯度。

上面是純量的求導，同時也可以做矩陣求導，例如 ：

X a t。rch. randn ( 3)

1 2 3 4 5 6 7 8

x a Variable (x, requ斗res grad-True)

y a X \* 2

pr工nt ﹛ y)

y.backward ﹝t。rch.Fl。atTensor ﹝ ﹝ 1，。1, 0.01﹞ ） ）

print (x.grad)

3.1 熱身：PyT。c rch 基礎 團－

相當於列出了一個 3D 向量去做運算，這時候獲得的結果 y 就是一個向量， 這裡對這個向暈求導就不能直接寫成 y.backward（）， 這樣程式是會顯示出錯的 。這 個時候需要傳導入參數宣告，例如早backward(torch.FloatTensor （﹝ I , I, I﹞））， 這樣獲 得的結果就是它們每個分量的梯度 ，或可以傳入 y.backward(torch.FloatTensor （﹝I, O. l , 0.01﹞））， 這樣獲得的梯度就是它們原本的梯度分別乘上 I • 0.1 f日 0.01 °

* + 1. Dataset （ 資料集 ）

在處理任何機器學習問題之前都需要資料讀取， 並進行前置處理。PyTorch

提供了很多工具使得資料的讀取 和前置處理變得很容易。

torch.utils.data.Dataset 是代表這一資料的抽象類別 ，你可以自己定義你的

資料類別繼承和重新定義這個抽象類別， 非常簡單， 只需要定義 Jen F日

一且etitem一這兩個函數：

* + - 1. class myDataset{Dataset ) :
      2. def init {self, csv file, txt file, r。七 dir’。ther file﹜
      3. self ,csv data - pd .read csv ﹛ csv file)

。

* + - 1. with pen{ txt file’，主，I as f ·
      2. data list - f.readlines {I
      3. self.txt data - data list
      4. self .r 。t dir - root dir

8

1. def len {self) :
2. return len ﹛ self csv data )

11

1. def getitem {self, idx) .
2. data - {self .csv data ﹝ idx﹞，self txt data ﹝ idxl I
3. re乞urn data

透過上面的方式， 可以定義我們需要的資料類別， 可以透過反覆還算的方式 來取得每一個資料， 但是這樣很難實現取 batch ' shuffle 或是多執行緒去讀取資 料，所以 PyTorch 中提供了一個簡單的辦法來做這個事情， 透過 torch.utils.data. DataLoader 來定義一個新的反覆運算器，如下 ：

1. datai七er a DataLoader ﹛myDataset, batch\_size-32, shuffle-True,

。

1. collate fna刊default c llate﹞

一第 3章

多層全連接神經網路

裡面的參數都特別清楚， 只有 collate fu 是表示如何取樣本的，我們可以定義 自己的函數來準確地實現想要的蹦喔，預設的函數在一般情況下都是可以使用的。

另外在 torchvision 這個套件中還有一個更進階的有關於電腦視覺的資料讀取 類別：

ImageFolder，主要功能是處理圖片，且要求圖片是下面這種儲存形式：

root/d。g/xxx.png x。ot/d。g/xxy.png ro。t/d。g/xxz.png

1 2 3

4 5

r。。t/cat/123 png z。ot/cat/asd png

6

root/cat/zxc.png

7

之後這樣來呼叫這個類別

1 dset - ImageF。lder ﹝z。。t-' r。。t path，transform-N。ne, loader-default l。ader﹜

其中的 root 需要是根目錄，在這個目錄下有幾個資料夾，每個資料夾表示一 個類別：transform 和 target\_transform 是圖片增強，之後我們會詳細諦， loader 是 圖片讀取的辦法， 因為我們讀取的是圖片的名字，然後透過 loader 將圖片轉換成 我們需要的圖片類型進入神經網路。

3.1.4 nn.Module （ 模組 ）

在 PyTorch 裡面撰寫神經網路， 所有的層結構和損失函數都來自 torch.nn ，所 有的模型建置都是從這個基礎類別 m.Module 繼承的，於是有了下面這個範本。

class net name ﹛nn M。dule) :

－1

def init (self ’。ther arguments) :

2 3 4 5 6 7 8 9

super ﹝ net name, self ) in斗t (I

，。

self .convl - nn Conv2d ﹛ in channels ut channels, kernel size﹜

＃。ther netw 。rk layer

def forward (self , x) x - self .c。nvl (x﹜

return x

3.1 熱身：PyTorch 基礎 圓圓圓圓－

這樣就建立了一個計算圖，並且這個結構可以重複使用多次， 每次呼叫就相 當於用該計算圖定義的相同參數做一次正 向傳播，這得益於 PyTorch 的自動求導 功能，所以我們不需要自 己撰寫反向傳播 ，而所有的網路層都是由 m這個套件獲 得的，例如線性層 rm.Linear’等之後使用的時候我們可以詳細地介紹每一種網路 對應的結構，以及如何呼叫 。

定義完模型之後，我們需要透過 nn 這個套件來定義損失函數。 常見的損失 函數都已經定義在了 nn 中，例如均方誤差、多分類的交叉婿，以及二分類的交叉 焰，等等，呼叫這些已經定義好的損失 函數也很簡單：

1 criterion nn .CrossEnt玄 pyLoss ﹛）

。 。

2 1 ss 宣 criteri n ﹛ output, target)

這樣就能求得我們的輸出和其實目標之間的損失 函數了。

3.1.5 torch.optim （ 最佳化 ）

在機器學習或深度學習 中，我們需要透過修改參數使得損失函數最小化 （ 或 最大化），最佳化演算法就是一種調整模型參數更新的策略。

最佳化演算法分為兩大類。

油 1一階最佳化演算法

這種演算法使用各個參數的梯度值來更新參數， 最常用的一階最佳化演算法 是梯度下降 。所謂的梯度就是導敏的多變數運算式， 函數的梯度形成了一個向量 場，向時也是一個方向 ，這個方向上方向導數最大，且等於梯度。梯度下降的功 能是透過尋找最小值， 控制方差，更新模型參數，最後使模型收斂，網路的參數 更新公式是：

1

而‘（J

州一切

AV

*(B)*

）

可

其中可 是學習率， 了百一是函數的梯度，我們可以透過圖 3.2 具體地說明一下

該方法。這是深度學習裡面最常用的最佳化方法， 我們之後會詳細說明 它的各種 變式。

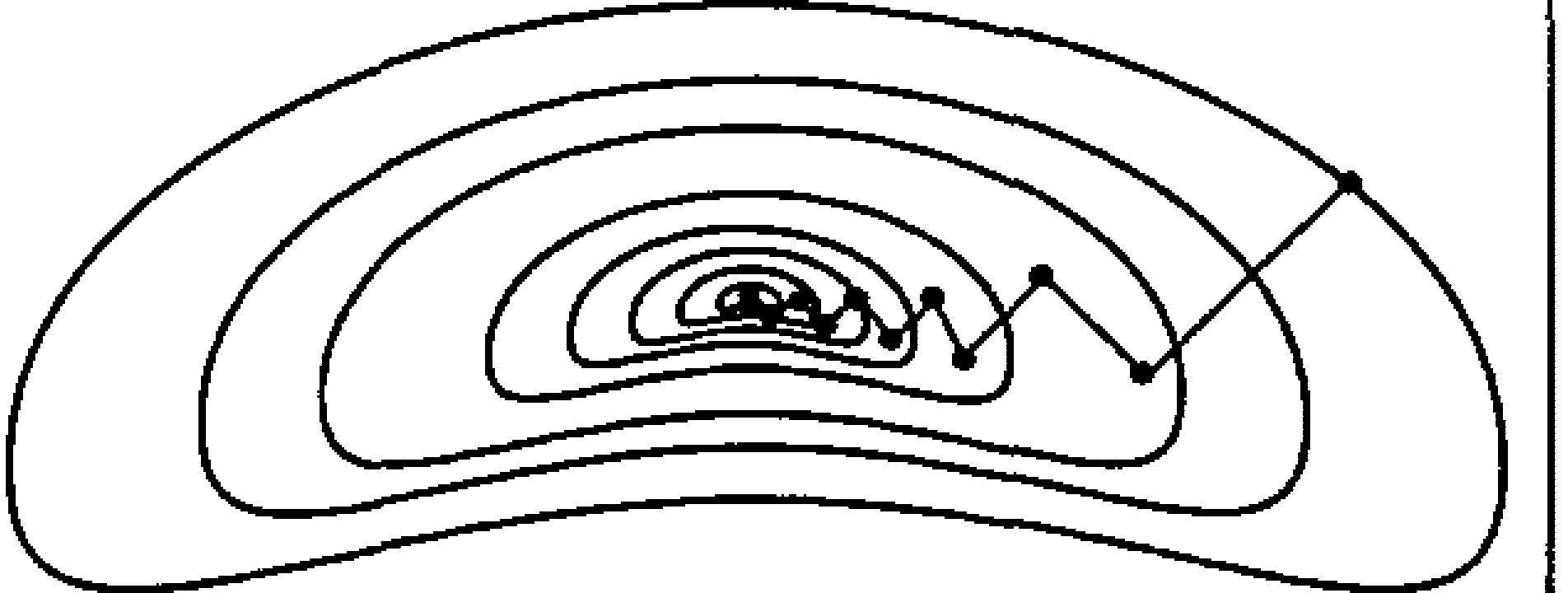
－第 3 章 多層全直接神經網路

油 2. 二階最佳化演算法

二階最佳化讀算法使用 7二階導數 （ 也叫做 Hessian 方法） 來最小化議最大 化損失函數，主要以牛頓法，但是由於二階導數為基確的計算成本很高， 所以這 種方法並沒有廣泛使用。 torch.optim 是一個實現各種最佳化讀算法的套件， 大多 數常見的讀算法都能夠直接遺過這個套件來呼叫， 'Wll如隨機梯度下降，以及增加 動量的隨機梯度下降， 自我調整學習章等。

在呼叫的時帳將需要最佳化的事數傳入，這些吾吾數都品置是 Variable 然雄傳 入一些基本的設定，倒要日學習率和動量等。

2



。

·1·

’2~~·~~

·10 .5 自 5 10

圖 3.2 一階最佳化讀算法

下面舉一個實例：

1 optimizer = torch.optim.SGD ﹛model.parameters () , lr=0.01, momentu恥＝0.9﹜

這樣我們就將模型的妻數作為需要更新的書數傳入最佳化器， 設定學習 單是 0.01 ，動量是 0.9 的隨攝輔度下降，在最佳化之前需要先將樺度歸霉 ，即 optimiz肘 zeros（）， 然後遺過 loss.bac』cward（） 反向傅擂， 自動＊導獲得每個學數的

梯度，最佳只需要 optimizer.step（） 就可以透過梯度做一步審數更新。

3.1.6 模型的儲存和載入

在 PyTorch 裡面使用 torch.save 來儲存模型的結構和參數， 有兩種儲存方式：

( I ） 儲存整個模型的結構資訊和參數資訊，儲存的物件是模型 model ﹔

( 2 ） 儲存模型的參數， 儲存的物件是模型的狀態 model.state diet() 0 可以這樣儲 存，S盯e 的第一個參數是儲存物件， 第三個參數是儲存路徑及名稱： torch. save(model ，.／model抖的

t。rch.save ﹛m 。del. state diet ﹛﹜，’／m。del\_state.pth ）

載入模型有兩種方式對應 於儲存模型的方式：

( 1 ） 載入完整的模型結構和參數資訊 ，使用 load model - torch.load （’model. pth') •

在網路較大的時候載入的時間比較長，同時儲存空間也比較大﹔

( 2 ） 載入模型參數資訊，需要先匯入模型的結構，然後透過 model.load state die (torch.load（’model\_state.pth ＇）） 來匯入。

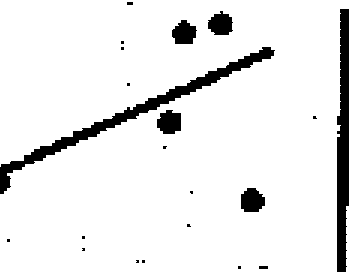
I 3.2 I 險性模型

這一節將從機器學習最簡單的線性模型入手， 看看 PyTorch 如何解決這個間 題。

3.2.1 問題介紹

說起線性模型，大家對它都很熟悉了 ，一般來說就是指定很多個資料點， 希 望能夠找到一個函數來擬合這些資料點使其誤差最小， 例如最簡單的一元線性模 型就可以用圖 3.3 來表示。

1 i i －－



..... ,·

so

苗。

間 3.3 列出了一系列的點， 找一陣直線，值得直線盡可能與這些點接近， 也

就是這些點到直蟬的距離之和量可能 *lj\*.用數學語言來嚴格表達， 即指定由 d 個

屬性描述的範側 *x* ＝（爪，為 *X3 , ···, xd* ） ，其中 X；表示 x 在第 i個屬性上面的設定值， 線性模型就是試圖學習一個透過屬性的騙性組合來進行預制的區數， 即

。.2)

(3.3)

其中 *w* ＝（叭，叫，...*, wd)* • *w* 和 b 都是需要學習的參數，模型透過不斷地調整

w 和 b，最值就能夠撞得一個最佳的模型。

騙性模型形式簡單、 屬於建模，卻孕育著機器學習頡釀中重要的基本思想，

同時騙性模型還具有特別好的解釋性， 因為加權的大小說直接可以表示這個屬性 的重要程度 ，

i ．，－ i

— i ．﹒i t i －

．1 ﹒i

E﹒巴i i i I L －

i 戶 i id F－ ．． i i ι

40 ·

﹒j ＼a r t e l

－ － －－

．

＼

*f (x)* == *w1x1* + w2x2 *+···+wdxd +b*

－－

i i

圖 3.3 一元臨性草草型

i i i

·30

i i

i

ii i －

m a－L

i

j i J I

一 i i

（

l －／…i H γ

一 － ．o i i

﹒一 i i 戶 i －

一 J －

－ f r －i－i1 …－

’

．

*f (x)""' wrx+b*

首先我們從最簡單的一雄線性回歸入手。

i i i

I﹔ j

i i y ‘ l

－i i戶

I J i i－u

－－－戶，

i f －i

i i － AF －－ i i

i i J i

i i

i － o

l i i L ii i －i r

－i ZE

一i i 一 ﹒iii I l i ．4氓，﹒F

i i ’i l i－－i i

a i i i i

．

八

I i i… i i

j

l －u －u dr

i ～ i i i

[l](#_TOC_250000)

h

－

一艘可以用向量的形式來表達：

i ‘←i i

一i i i i i i i l

l

i y i i i r i

1. i － iI

i ：i

：－i

1. i ＼i －巴－i ﹒i －． i f I i ﹒ i i ’I －II

iL E E

－i ii －i M

i －F i 、d

－ l － J 1

h i ＼ －﹒－胃．F ．a － 1

i i

川 ﹒ ’．…．i

4

1. i 』 U

1 i i i ii i i i

j

I i ？ i

i i i

i i

ii i

i ：一i i

J －－ o

J

R h J

e f －－

」 Ii

E l i － i．，－ ﹒i 叫 i 一

i

屯 i i

咱i 』i I l i i J i － F司 i﹒i i

il i i固

－i l i i－ O i i－i －i ﹒i一i S －﹔ i J 一

多居全連接神經網路

15

’﹒

r i 一

』i

l ii i i I J i i f － i l i J i i a F ii

1. j ii i

i i i i…i

i －l

i － － －

a t －一 l －

－

－

i

： 』 － －

i

ι

i

i y L

i F

l i r

i i F i

（i … 一九 ﹒ F

i i

u O

… ： d － 4

八

i

－

i－i i

I I i

i l J i

i

i i

－l t

4

’－ －

－i －i ii

－u －l i ii

﹒i i ，－：－a

l i i i l i

－a i i

i i i i －e I

I ’u

-20

i

— －li

團

軍 3 章

卜， J

－

3-10

3.2.2 一維線性回歸

指定資料集 D斗的，*Y 1),* （巧. ，*y,).* （.巧 *Y,).* • • •, *(xm,Ym*）﹜’線性回歸希望能夠優化 出一個好的函數 *f( x*） ，使得 *f(x;) -wx;+by*；能夠與 *y* ，盡可能接近。

如何才能學習到參數 w 和 b 呢？很簡單，只需要確定如何衡量 *f(x* ） 與y 之間 的差別，就像之前講的一樣，可以用它們之間的距離差異 *Loss =* ）*f ( x*，） 一 *y*，）＇來 衡量誤差，取平方是因為距離有正有負， 我們希望能夠將它們全部變成正的。這 也就是著名的均方誤差，要做的事情就是希望能夠找 到 J和 f ，使得

山﹒） ＝哼哼U州 几）＇ (3.4)

＝略min L(y，叫一*b)'*

*w,b f•I*

(3.5)

均方誤差非常直觀， 也具有很好的幾何意義，對應了常用的歐幾里德距離， 以均方誤差最小化來進行模型求解幸福基礎的辦法也稱 為 「 最小平方法 J 0

求解辦法其實非常簡單，如果求這個連續函數的最小值， 那麼只需要求它的 偏導敏，讓它的偏導數等於 0 來估計它的參數，即 ：

二且也哥*OS*孔拉」

*m* 2

*m*

平

*2( wx* ﹜

*(y,-b)x,)* = 0 (3.6)

*ar* η*•ss* A

，→

一一」且*2.= 2( mb -*z::(y *vx,))* = 0

*8b ti'*

透過求解式 （ 3.6） 和式 （ 3.7）， 我們就可以獲得 w 和b 的最佳解：

*w= L;:y*」*x,- x)*

(3.7)

*L:1 xf* 主cL:1冉）＇ (3.8)

*b* ＝ 主*(y*，一 *wx,)*

唱， J

其中 寸L:1x，即為 x 的平均值。

(3.9)

－－－－－ 第 3章 多層全連接神經網路

3.2.3 多維線性回歸

更一般的情況是多維線性回歸， 例如像前文描述的，我們有 d個屬性，試圖 學得最佳的函數 f（.可）：

*f ( x;) =wrx1 +b* (3. I 0)

使得z: , (/(xJ一九）＇最小’這也稱為 「 多元線性回歸」， 同樣可以用最小平方 法對 *w fa b* 進行估計，為了方便計算，可以將 w 和 d 寫進同一個矩障，將資料集 D表示成一個 m× （*d+* I ） 的矩陣X，每行前面 d個元素表示 d個屬性值，最後一個 元素設為 1，即

*(* X11 *X12* ..• *Xld ()* （ λ「 11

X 一 巧1 高2 *x,d* 'IJl x; 1 1•

-- .一

0 • • • I I P

I •

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| • | • |  | * I | I , I |
| *x,,2* | ... | *x,,d* | *1)* | lx;. *I )* |

、叫

將目標y 也寫乘向量的形式 *y* ＝仇 *Y* ，·，*y.*） 那麼我們就能夠獲得．

*w" = arg min.(y -Xw)' ( y -Xw)* (3.11)

同樣對其求導，令它等於 0 。

些主＝*2X r (Xw- y )= O*

*ow*

(3.12)

因為上面有關了矩陣的逆還算，所以需要 *x rx* 是一個滿秩矩陣或正定矩障， 那麼我們可以獲得：

*w' =(X' X ) 'X' y* (3.13)

所以線性回歸模型可以寫成：

f的） *=x; (X' xr• xr y* (3.14)

3.2 線性模型 一

然而在現實工作中 ，*x r x* 常常不是滿秩矩障，就算是滿秩矩障，求解逆的過 程也比較慢，所以我們一般可以使用梯度下降法去求解這個最小平方問 題。

3.2.4 一維線性回歸的程式實現

講了這麼多原理，下面用 PyTorch 來求解一下一維線性回歸問題。 首先我們 隨便列出一些點：

x train a np array （ ﹝ ﹝ 3.3﹞ ，﹝ 4 .4﹞ ，﹝ 5 5﹞ ，﹝ 6 .71﹞ ，﹝ 6.93﹞ ，﹝ 4 .168﹞ ，

1

﹝ 9 .779﹞ ，﹝ 6 182﹞ ，﹝ 7.59﹞ ，﹝ 2.167 ﹞ ，﹝ 7 042 ﹞ ，

。

2 3

﹝ 10 .791﹞ ，﹝ 5.313﹞ ，﹝ 7 997﹞ ，﹝ 3.1﹞ ﹞ F d乞ypeanp.f1 at32J

，

1』

3

7

5

．

1

『L 1』 ny

’

2

﹞

﹝

* d

1 ，

﹝

’

﹞ 2

3

﹝

’﹞

﹞

9

0 2

r

2

﹝

﹞

．

2

﹞

．

﹝﹝ ﹝3 ﹝3

﹝

V4

a r z a

o－

n

＝

n

l

a r t

丸

4 5 7

透過 matplotlib 畫出來就是這個樣子， 如圖 3.4 所示。我們想要做的事情就是 找一條直線去逼近這些點， 也就是希望這條宜線離這些點的距離之和 最小，先將 numpy.array 轉換成 Tensor 因為 PyTorch 裡面的處理單元是 Tensor ，按上一章諧 的方法，這就特別簡單了：

）

2

qd t a 0

且

，

7 n

＝

8 e

M 2 叩

6 t

﹞ ，

﹞

1

﹞

2

3

9 ． ．

1 1 1

．﹝ ﹝

’ ，

﹞

3 4

．

5 0

9

L 令

﹝

，

﹞ ，

， ﹞

6

9 5

7 ．．

6 5 6

2 1

﹝ ﹝’﹝’

， ﹞ ﹞

6

5

6

7 6

3 4

1 － －

6

1 x\_train a t。rch.f r。m numpy ﹛ x\_train﹞

2 y﹛tra工n = torch frαn numpy IY. train)

接著需要建立模型， 根據上一節 PyTorch 的基礎知識，這樣來定義一個簡單 的模型：

class LinearRegressi。n (nn Module) .

1 2 3 4 5 6 7 8

def init (self ﹜．

super (LinearRegression, self ) init ﹛﹜

。

self .linear a nn Linear (1, 1﹜ ＃ input and utput is 1 dimension def f。rward (self , x) ·

out a self linear (x)

】return out

if t。rch. cuda is ava斗lable 11 :

。

9 1 1 1 1

m 。del a LinearRegressi 。n ﹛ J .cuda II

1 2 3

else

。

model a LinearRegressi n ﹛﹜

3-13

3.5 1

2.5

． ．．

．．

．

．

．

2.0 ．

#### ．．．．

．

．

2 4 6 自 1口

圖 3.4 matplotlib 畫出的圖

這裡我們就定義了一個超級簡單的模型 *Y ""WX+b*，輸入參數是一維， 輸出吾吾 數也是一雄， 這就是一條直線，當然這裡可以根據你想要的輸入輸出雄度進行更 故，我們希望去最佳化書數 w 和 b 能鉤使得這條直線盡可能接近這些點， 如果支 援 GPU 加速，可以透過 model.cuda（） 將模型放到 GPU 上。

然接定義損失函數和最佳化函數， 這裡使用均方誤聾作為最佳化函數， 使用 梯度下降進行最佳化：

1. criterion = nn .MSELoss （ ﹜
2. optimizer = optim.SGD (model.parameters ﹝ ） , lr=le-3)

接著就可以開始訓練我們的模型了：

1. num＿已pochs = 1000
2. for epoch in range (num epochs) :
3. if torch.cuda .is available ﹛ ） !
4. inputs = Variable ﹛耳＿ train) .cuda （﹞
5. target = Variable ﹛ y\_train﹜ .cuda ﹛）
6. else:
7. inputs = Variable (x\_train﹜

B target = Variable ﹛ y\_train) 9

3.2 線性模型 －－

1. # forward
2. out m。del ﹛ inputs﹜
3. 1。ss = cr1ter1。n （。ut, target)
4. # backward

。 。

1. ptimirer .rer grad （﹜

。

1. 1 ss.backward ﹛﹜

。

1. ptimirer .step (I

17

。

18 if (ep ch+l) % 20 aa O ·

。 。

19 print ﹛’Ep ch ﹝ ﹛ ﹜／ ﹛ ﹜ ﹞，l ss. ﹛ · 6f ｝ ’

。 。 。

20 .format (ep ch+l, num ep chs, 1 ss.data ﹝ O﹞ ） ）

定義好我們要跑的 epoch 個數，然後將資料變成 Variable 放入計算園， 然 後透過 out=model(inputs） 須得網路正向傳播獲得的結果， 透過 loss-criterion(out, target） 獲得損失函數，然後歸零梯度，做反向傳播和更新參數， 特別要注意的 是，每次做反向傳播之前都要歸零梯度，optimizer.zero\_grad （）。 不然梯度會累加 在一起，造成結果不收斂。在訓練的過程中隔一段時間就將損失函數的值列印 出來看看，確保我們的模型誤差越來越小 。注意 loss.data﹝呵， 首先 loss 是一個 Variable’所以透過 loss.data 可以取出一個 Tensor 再透過 loss.data﹝0﹞ 獲得一個 int 或 float 類型的資料，這樣我們才能夠列印出對應的資料。

做完訓練之後可以預測一下結果。

1. model. eval (I
2. m。del cpu ﹛﹜
3. predict a m 。del (Variable ﹝ x\_train﹜ ）
4. pred工ct a predict. data numpy (I

。

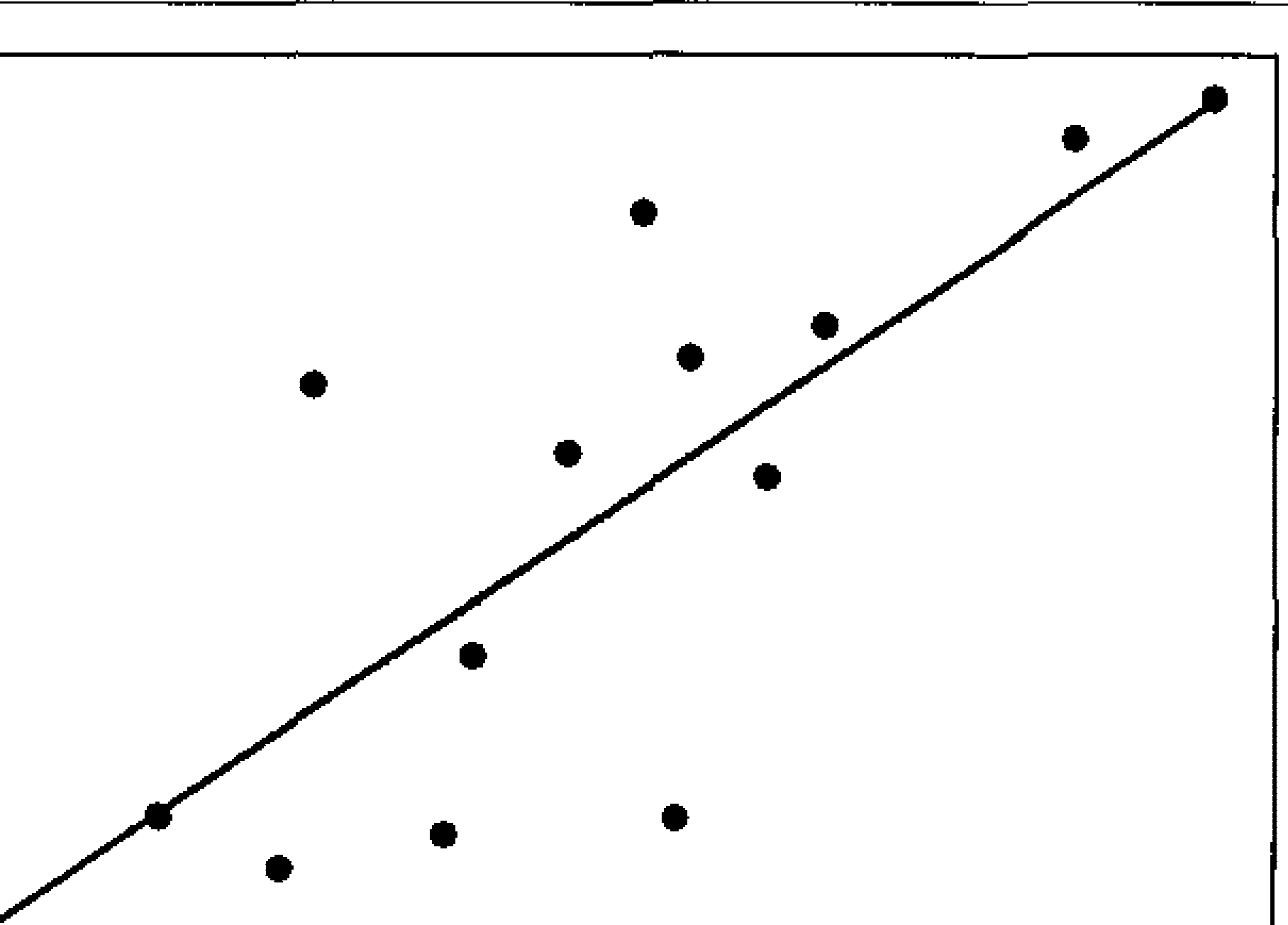
1. plt .pl t ﹝x train .numpy (I , *y* train numpy (I , ' ro ', labela Original data ）

。

1. plt pl t (x train .numpy (I , predict, labela' Fitting Line E ﹜
2. plt .show (I

首先需要透過 model.eval（） 將模型變成測試模式 ，這是因為有一些層操作，例 如 Dropout 和 BatchNormalization 在訓練和測試的時候是不一樣的，所以我們需要 透過這樣一個操作來轉換這些不一樣的層操作。 然後將測試資料放入網路做正向 傳播獲得結果 ，最後畫出的結果如 圖 3.5 所示。

章 5



3.0

2.5

2.0

1.5

2．．

4 6 8 10

圖 3.5 一元回歸

這樣我們就透過 PyTorch 解快了一個簡單的一元回歸問題， 撞得了一儸直線 去盡可能逼近這些離散的點。

3.2.5 多項式回歸

對於一值的騙性回歸， 由於該函數擺台出來的是一棵直線， 所以精度欠佳， 我們可以考慮多項式回歸， 也就是加強每個屬性的次數， 而不再是只使用一次去 回歸目標函數，

原理和之前的線性回歸是一樣的， 只不過這裡用的是高次多項式而非簡單的 一次總世多項式。

首先列出我們想要提合的 方程式：

*y=* 0.9+ 0.5xx+3x x2 + 2.4x x3

然種可以設定吾吾數方程式 ：

*y =b+w1 x x+ w1 x x2* ＋門 *x x3*

3.2 線性模型 一

我們希望每一個參數都能夠學習到和其實參數很接近的結果 。下面來看看如 何用 PyTorch 實現這個簡單的工作 。

首先需要前置處理資料，也就是需要將資料變成一個矩陣的形式：

*X* 2 *X* 3 、I

I I I

2 3 I

冉 冉︱

*x*2*, x*3*,)*I

在 PyTorch 裡面使用 torch.cat（） 函數來實現 Tensor 的連接：

def make f eatures ﹝ x) '

。

1 2 3

”””Builds features i e. a matrix with c lumns ﹝ X, X勻，x'3﹞ ．””H

x x unsqueeze ( l﹜

。 。

return t rch.cat （ ﹝ X \*\* 1 f r 立 in range ( l, 4 ﹜ ﹞ ，1)

4

對於輸入的 nf固資料，我們將其擴充成上面矩陣所示的樣子 。 然後定義好真實的函數：

W\_target torch.Fl。atTens。r I ﹝ 0 .5, 3, 2 .4﹞ ） unsqueeze ﹝ 1) b target a t。rch.FloatTens 。r I ﹝ 0.9﹞ ）

1

2

3

def f (X) ,

。

4 5 6

’，”，Approximated functi n ”。”，

return x .mm ﹛W\_target ﹜ ＋ b\_target ﹝ O﹞

這裡的加權已經定義 unsqueeze(l ） 是將原來的 tensor 大小由 3 變成 （3, !) • x.mm(W\_target） 表示做矩陣乘法， f(x） 就是每次捕入一個 x 獲得一個y 的其實函 數。

在進行訓練的時候我們需要取樣一些點， 可以隨機產生一些數來得到每次的 訓練集 ：

: 1 def get batch (batch\_sizea32) ,

1. ”””Bcilds a batch i e E ﹛x，主﹝x﹜ ） pa工r. ”””

。

1. rand m a torch .randn (batch size)

。

1. x a make features ﹛ rand m)

一 第 3章

多屆全連接神經網路

y - f (x﹜

。

5 6 7 8 9

if t rch cuda is available ()

return Variable (xi .cuda ﹛﹜，Variable ﹛ y) .cuda () else :

return Variable ﹝x﹜ ，Variable ﹛ y)

透過上面這個函數我們每次取 batch size 這麼多個數據點，然後將其轉換成 矩陣的形式，再把這個值透過函數之後的結果也傳回作為真實的目標。

然後可以定義多項式模型：

J Defi立 e m。del

1 2 3 4 5

c工ass poly model ﹝nn .M。dule) :

def 一＿rnit←＿ (self ) :

super ﹛p。ly\_m。del, self ) . init ﹛﹜

self .poly a nn Linear ( 3, 11

：

） （

VA Y

工

ι

1 ．

e

﹛ e

d s

w

f

f

e

d

6

if t。rch.cuda.is available () . m 。del a poly model () .cuda ﹛﹜

）

x

伊

f t

。

s l u

z n

a E Z

u

z t t

。

o u e

z

。

1

2

1

7 8 9 1 1

else·

3 4

1 1

m。del a p。ly\_model ﹝）

這裡的模型輸入是 3 維，輸出是 I 維，跟之前定義的線性模型只有很小的差 別。然後我們定義損失函數和 最佳化器：

1 criteri。n a nn MSELass ()

。 。 。

2 ptimizer a

pt im SGD ﹛m del.parameters 口，lrale-3﹜

同樣使用均方誤差來衡量模型的好壞， 使用隨機梯度下降來最佳化模型， 然 後開始訓練模型 ：

epoch a O while True:

1 2 3 4 5 6

J Get data

batch x, batch\_y a get batch ()

J Forward pass

。 。

utput a m del (batch xi

1. loss = criterion ﹛ output, batch\_y)
2. print\_loss 旨 loss.data ﹝ O J
3. # Reset gradients
4. 口ptimizer .zero\_grad ()
5. j/ Backward pass
6. las昌 backward ﹛﹜
7. JI update parameters
8. optimizer .step （ ﹜
9. epoch += 1
10. if print\_loss < le-3:
11. break

這裡我們希望模型能夠不斷地最佳化， 直到實現我們設立的條件， 取出的 32

個點的均方誤差能鉤小於 0.001 °

執行程式可以獲得如圖 3.6 所示的結果。

L叭 （） .0正1，）叫 .;f電

﹔

佇 d f υη

cc .• L 佇 3』 o （ ti.O'I • •/ C 咐 91 ＋ 這 4':•. ＋ 句*9'," 2* + 2.41 九、古︱

。

0

l

|  |  |  |  |
| --- | --- | --- | --- |
| ＋ 自 | $ λ + | ，作旬 ，···，. | *2* |

已〉 Mct ual fu n 付＼ 0仇 :, C 1) • 9句

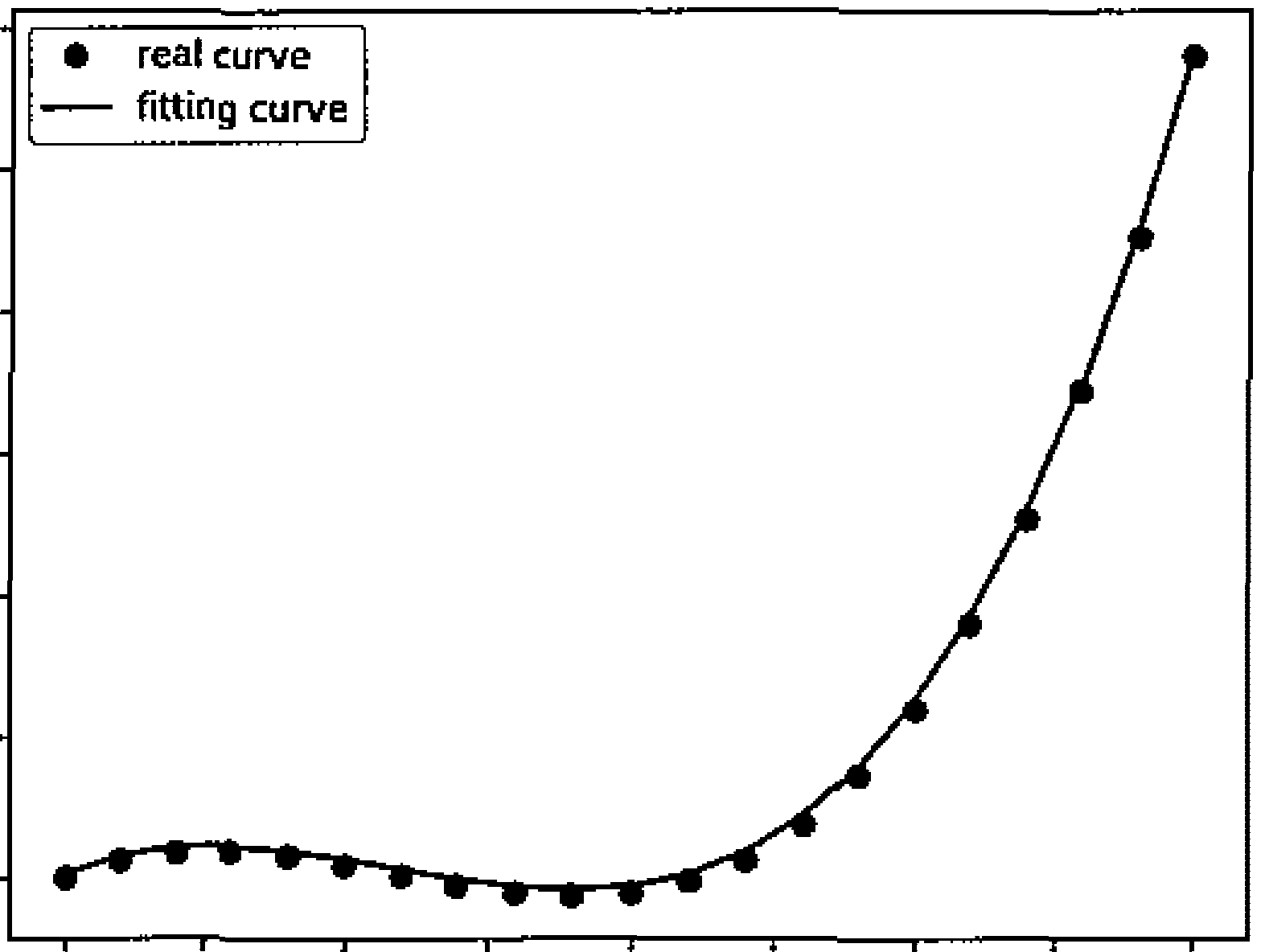
圖 3.6 程式執行結果

4,)+:,'3 1

將真實函數的資料點和擺台的多項式畫在同一張圖上， 我們可以獲得如圖 3.7

所示的結果。

〉



7

6

5

、

4

3

2

l

-1.00 --0.75 --0.50 -0.25 日.00 0.25 0.50 0.75 l.O自

X

圖 3.7 多頂式回歸

從兩個結果來看，我們已經很接近真實的函數了。 現實世界中很多問題都不 是簡單的線性回歸，有關很多複雜的非線性模型， 而線性模型是機器學習 中最重 要的模型之一，它的統計思想及其非常直觀的解釋性仍然可以給我們一些敢發。

J 3.3 J 分類問題

下面這節我們將開始介紹機器學習領域裡面的另一個 問題：分類問題。

3.3.1 問題介紹

機器學習中的監督學習主要分為回歸問題和分類問題，我們之前已經講過回 歸問題了 ，它希望預測的結果是連續的， 那麼分類問題所預測的結果就是離散的 類別 。這時輸入變數可以是離散的， 也可以是連續的，而監督學習從資料中學習 一個分類模型或分類決策函數， 它被稱為分類器 （ classifier ）。 分類器對新的輸入 進行輸出預測，這個過程即稱為分類 （ classification ）。 舉例來說，判斷郵件是否 為垃圾郵件，醫生判斷病人是否生病，或預測明天天氣是否下雨等 。同分時類問 題中包含有二分類和多分類問題，我們下面先講一下最著名的二分類演算法一一 Logistic 回歸。首先從 Logistic 回歸的起源說起。

* + 1. Logistic 起源

Logistic 起源於對人口數量增長情況的研究，後來又被應用到了對於微生物生 長情況的研究，以及解決經濟學相關的問題，現在作為回歸分析的分支來處理分 類問題，先從 Logistic 分佈入手，再出 Logistic 分佈推出 Logistic 回歸。

* + 1. Logistic 分佈

設X是連續的隨機變數， 服從 Logistic 分佈是指X的累積分佈函數和密度函 數如下：

*F*（恥*P( X* 三 *x*）＝ (3.15)

，－（*x-µ)lr*

*f (x)= r(l* － 」b□□*y*賢

*+e*「「削巾

(3.16)

其中 μ影響中心對稱點的位置， y 越小中心點附近的增長速度越快。 下一 節會講到在深度學習中常用的非總性轉換 Sigmoid 函數是 Logistic 分佈函數中 *y=* 1,µ=0 的特殊陪式。

(sigmoid)Logistic 函數的表達形式如下：

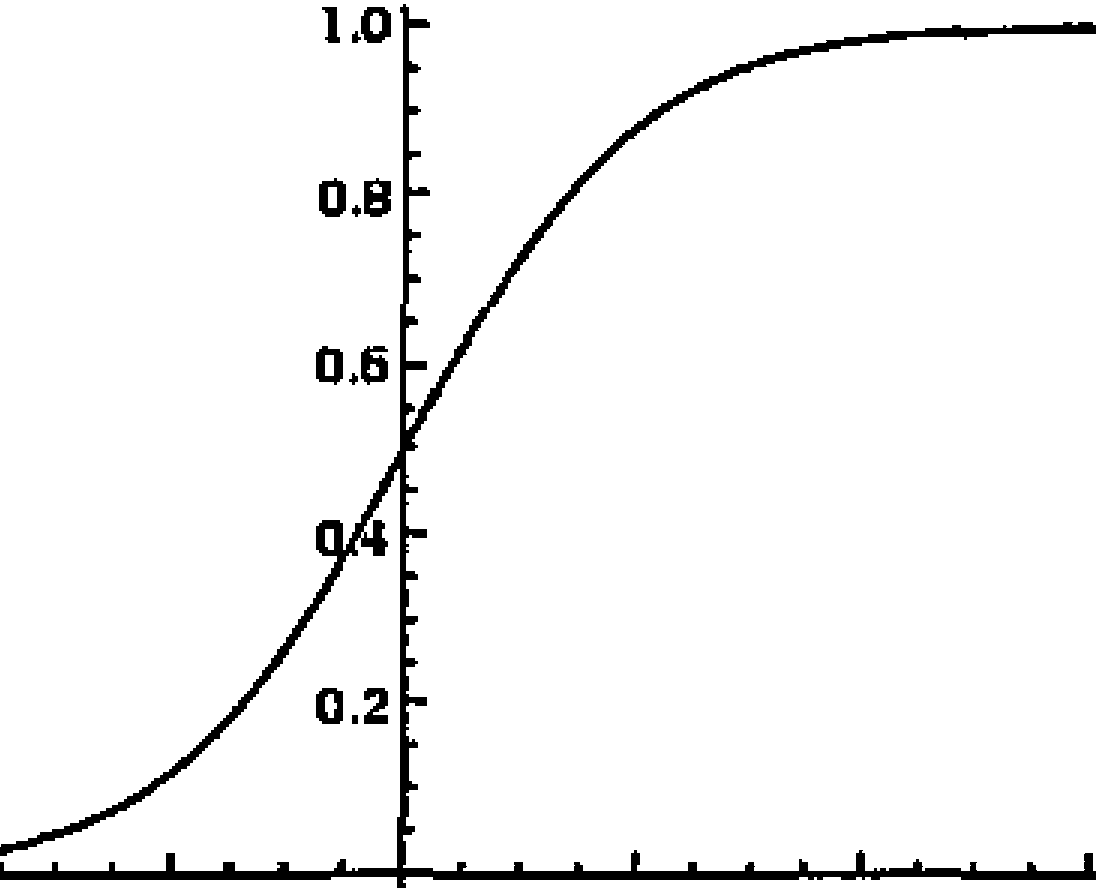
*p( x*）＝」士

1－← e

(3.17)

其函數影像真日圖 3.8 所示，由於函數很像 「 S」 形，所以鼓函數叉口到 Sigmoid

函數。



-6 -4 -2 2 4 5

圖 3.8 Sigmoid 函數影像

* + 1. 二封類的 Logistic 回歸

Logistic 回歸不僅可以解訣二分顯問題，也可以解決多好額問題，但是三分額 問題最為常見同時也具有良好的解釋性。對於二卦輯問題，Logistic 回歸的目標是 希望找到一個區分度足夠好的訣策邊界， 能夠將兩種極佳地告聞 。

假設輸入的資料的特徵 向量把 R＂，那麼決策邊界可以表示為主 三，恨，＋*b=O* ﹔ 假設存在一個樣本點使得 *h,( x)* = I;;.,*w,x, +b* > 0 ，那麼可以判斷它的類別是 1 ﹔

如果*h.( x)=* I: i叫*x,+b < O* ，那麼可以判斷其類別是 0 。這個過程其實是一個成知 機的過程，透過決策函數的符號來判斷其屬於哪一種。 而 Logistic 回歸要更進一

步，透過找到分類機率 *P( Y=*I） 與輸入變數x 的直接關係，然後透過比較機率值來 判斷類別，簡單來說就是透過計算下面兩個機率分佈：

*P(Y* = 0 */ x)* ＝于占了

“.. (3.18)

J+e

*\_w·x+b*

*P(Y = l / x*） ＝－＝－－τ立T

l + e''"'"

(3.19)

其中 w 是加權，b 是偏置。現在介紹 Logistic 模型的特點，先引用一個概 念：一個事件發生的機率 （ odds ） 是指該事件發生的機率與不發生的機率的比 值，例如一個事件發生的機率是 p 那麼該事件發生的機率是 p’該事件的對數機 率或 logit 函數是：

l咿咿） = log:-1:-

*! - p*

對於 Logistic 回歸而言，我們由式 （ 3.16 ） 和式 （ 3.17 ） 可以獲得：

(3.20)

*P(Y =l / x)* c

log一一一－一一一一一＝ • *b*

*1 -P(Y = l / x)*

(3.21)

這也就是說在 Logistic 回歸模型中 ，輸出 *Y= I* 的對教機率是輸入 x 的線性函 數，這也就是 Logistic 回歸名稱的原因 。如果觀察式 （ 3.17 ）， 則可以獲得另外一 種 Logistic 回歸的定義，即線性函數的值越接近正無限大， 機率值就越接近 1﹔線 性函數的值越接近負無限大，機率值越接近 0 。因此 Logistic 回歸的想法是先擬合 決策邊界 （ 這裡的決策邊界不侷限於線性，還可以是多項式）， 在建立這個邊界和

分類機率的關f系，進一步獲得三分類情況下的機率。

3.3.5 模型的參數估計

上面我們簡單地介紹了 Logistic 回歸模型的建立，下面透過相當大似然估計 來求出模型中的參數。

對於指定的訓練集資料 *T*＝﹛（*X1 ,X 1)* ／巧，*x,) ,* ...，（丸 *x*＂） ﹜ ’其中X;E R" ,Y; E ﹛0, I ﹜ ’ 假設 *X;E P( Y=* !I x）＝π｛x）， 那麼 *P( Y=O* I x）＝ 一π｛x）， 所以似然函數可以有以下的表 達：

曰•I﹝π（冉）﹞刊1 π（x，）﹞I *Y,* (3.22)

取對教之後的對數似然函數為：

*L( w*）＝ 主﹝*y,* logll(x,) +(I *y,)* log(l -ll(x,l｝﹞ (3.23)

'C' *ll( X )*

＝主﹝Y; log一一」÷ log(! *ll*（冉））﹞

(3.24)

*';;(* I-*ll( X; )*

＝乏﹝*y*，（ ，÷ *b*） 一 log(!+ e＂·刊）﹞ (3.25)

對 *L( w*） 求極大值就能夠獲得 w 的估計值，可以使用最簡單的梯度下降法來求 得。用 *L*（ 少） 對 W求導，可以獲得：

啥*;2= ty,x,* t古 *x,* (3.26)

= 4*(y,-/ogit* （的科t (3.27)

帶＝令一2古L (3.28)

=4弘一*logit* （她，）） (3.29)

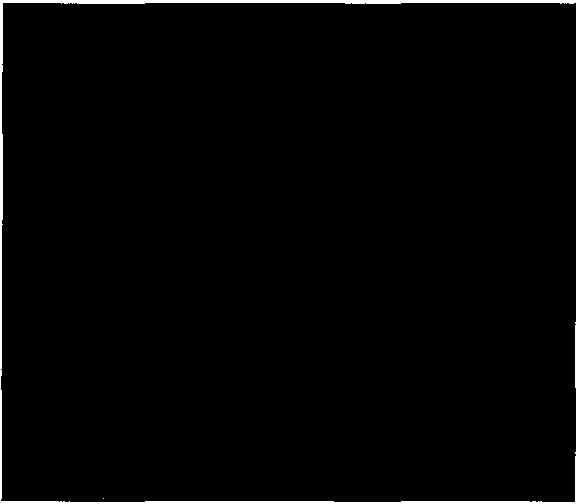
求出梯度之後就可以使用反覆還算的梯度下降來求解。

一 第 3章

多居全連接神經網路

3.3.6 Logistic 回歸的程式實現

首先我們開敢 txt 檔案，可以看到資料儲存的方式，如圖 3.9 所示。



圓 3.9

資料儲存

每個數據點是一行， 每一行中前面兩個資料表示 x 座標和y 座標，最後一個 資料表示其類別。

我們先從 data.txt 檔案中讀取資料，使用非常簡單的 python 讀取 txt 的方法就 能夠實現。

with open ﹛ data txt’，’r’﹜ as f :

1 2 3 4 5

data list f readlines （﹜

data\_list z ﹝ *i.*split （ ＼ n’﹜ ﹝O ﹞ for *i* in data\_list﹞ data list E ﹝ i.split ﹛ Y，） f r l 工n da七a\_list﹞

。

。 。

data ＝ ﹝ （ float ﹛ i﹝ 0﹞ ）， float ( i﹝ 1﹞ ） 'fl at ﹛i﹝ 2﹞ ﹞ ﹜ f X *i* in data list﹞

然後透過 matplotlib 能夠簡單地將資料畫出來。

xO list (filter (lar曲da X, X ﹝－1﹞ －－ 0 O, data） ﹜

1

xl - list ﹝filter (lambda x, x ﹝－1﹞ aa 1.0, data) ) pl t\_xO O ＂﹝土﹝ O﹞ for *i* in xO ﹞

。

2

3 4 5

pl。t\_xO\_l ＂﹝ 1﹝ 1﹞ for iin xO ﹞ pl。t x l Q a ﹝ i﹝ O﹞ f。x *i* in xl﹞

6 plot xl 1 ＝ ﹝ i﹝ 1﹞ f or i in xl ﹞

7

1. plt .plot (plot\_xO一日，plot\_xO\_l ’' ro’，label＝’x\_O ’）
2. plt .plot (plot\_xl\_O, plot\_x工＿ 1，’bo’，label= 'x\_l ＇﹜
3. plt .legend ( loc且，best＇﹜

首先將兩個類別分間， 然接將所有的資料點畫出就能夠獲得圖 3.10 。

1日0 →

．．． ．

．．．自

9日才 ．．．．．＇’．

BO 才

．．．．．

．．．．．、﹒．．

70 -l

．．．．．

．．．、．．．．

60 -l ．．．．．． ．

．

自-l • ．

*I* ．．.．．

,.、﹒

4日才

30

．．、－．．ξ ．．

．

30 40 50 60 70 BO 9日 100

圖 3.10 資料點

從圖 3.10 中我們可以明顯看出這些資料點被分為兩個額別： 一種用紅色的 點，一種用藍色的點， 我們希望透過 Logistic 回歸將它們分聞。

接下來定義 Logistic 回歸的模型，以及二分頡問題的損失曲數和最佳化方法。

1. class LogisticRegression (nn .Module) :
2. def init ﹛ self ﹜：
3. super ﹛ LogisticRegression, self ) . init ()
4. self .lr *=* nn.Linear ﹝ 2, 1)
5. self .sm *=* nn. Sigmoid ﹝）

6

7 def f口rward ﹛ self , x) : B x *=* self .lr ﹝耳﹜

9 x *=* self .sm ﹝ x)

1日 return x

一一－ 第3章 多層全連接神經網路

1. 1。gistic\_model 巨 Log工 sticRegressi。n ﹛）

11

1. if t。rch.cuda .is available ﹛）．
2. 1。gistic m 。del.cuda I﹜
3. criteri。n "" nn .BCEL。ss ﹛﹜

15

1. 。ptimizer t。rch.。ptim. SGD (1。gistic一m。del.parameters 日 lrle 3, 18 m。nentum O 9)

這裡 nn.BCELoss 是二分類的損失函數， torch.optim.SGD 是隨機梯度下降最 佳化函數。

然後訓練模型，並且間隔一定的反覆運算次數輸出結果。

1. f。r ep。ch in range (50000) ,
2. if t。rch .cuda .is available ﹝） ．
3. x Variable (x data﹜ cuda I)
4. y Variable ﹝y data ) .cuda ﹛）
5. else.
6. x Variable (x data)
7. y 巨 Variable (y data)

。

8 t ＝＝＝＝＝＝＝＝＝＝＝＝＝＝士＝＝f rward======-===-====

9 out ＝ 工。gistic\_m。del (x) 10 l。ss = criteri。n （。ut, y)

11 print\_l。ss = 1。ss data ﹝ O﹞ 12 mask z 。ut .ge﹛ 0.5) .float I ﹞

1. correct ＝ ﹛mask y) • sum ﹛）

。

1. ace = c rrect data ﹝ O﹞ *I* x .size ( O )

15 t ＝＝－＝＝＝＝＝＝＝＝一＝三backward ＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝三

。

1. pt工mizer zero grad ﹝）

。

1. 1 ss backward I)

。

1. ptimizer. step I)

。

19 if (ep ch+l) % 1000 = O · 20 print ﹛ ＊’10)

1. print I 'epoch ﹛ ﹜’.f。rmat ﹝ep。ch÷1))
2. print I 'loss is ( , .4f ） .f。rmat ﹛print l。ss） ﹜
3. print I acc is ﹛ 4f ） .f 主mat (ace） ﹞

其中 mask=out.ge(0.5).float （） 是判斷輸出結果如果大於 0.5 就等於 1，小於 0.5

就等於 0 ，透過這個來計算模型分類 的準確率。

3.3 分類問題 一

訓練完成我們可以獲得國 3.11 所示的 loss 和準確率。因為資料相對簡單，同 時我們使用的是也是簡單的線性 Logistic 回歸，loss 已經降得相對較低，同時也有 91% 的準確率。

epoc h 4S0c>O loss 1.s 0 2 549

a c e ,,. ,•, 91仇。

.年 全 ＊ \* ＊長. ＊＊＊

epoc h 49000 los,s, 1.s 0 . 2 5 3討

司（ （ i. s

府 Q

看 看 看 脅，令 食 可是 1般 是 企

H> O

epoc ,, 50000 loss i. s, O 2 5之 7

ac e i.s t1 91府。

圖 3.11 loss 和準確率

我們可以將這條直線畫出來，因為模型中學習的參數 叭，門和日b 其實組成了 一條直線 W戶＋*w,Y =b=O*，在直線上方是一種，在直線下方又是一種。我們可以透 過下面的方式將模型的參數取出來，並將直線畫出來，如圖 3.12 所示。

wO, wl 1。gistic\_model lr weight ﹝ O﹞

1 2 3 4 5 6 7 8

wO 巨 wO .data ﹝ O﹞

wl 巨 wl.data ﹝ O﹞

。 。

b a 1 gistic\_m del lr.bias.data ﹝ O﹞ plot\_x " np.arange (30, 100’。.1)

pl t y z ﹛－wa \* pl 乞 x - b) *I* wl plt pl 乞﹛pl t\_x, pl t\_y)

plt show ﹛﹜

透過國 3.12 我們可以看出這條直線基本上將這兩種資料都分開了。

以上我們介紹了分類問題中的二分類問題和 Logistic 回歸演算法，一般來 說，Lo伊tic 回歸也可以處理多分類問題，但最常見的還是應用在處理二 分類問題 上，下面我們將介紹一下使用神經網路演算法來處理多分類問 題。

申 － 第 3 章

．

’

．

－

恤，

多居全連接神經網路

100

．

90

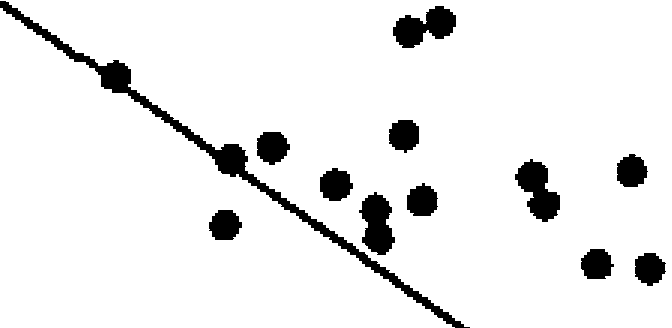
－

．

．

．、．

BO 70



－

＼ e －

，

、

．

．．．

．．．．

60

一叫－－H一﹒

一

一

「︱︱」’

．’．

，

‘up

．

．

－

．

．．．

h

－

』

J

＼

．

．

．

、

＼

－

．

．

．．

50

，

’

．

40

3日 ．

20

．．．

30 40

－

．

﹒

． ．

z ．

50 60

70 BO

90 100

圖 3.12 草帽至于闆兩種資料點

I 3.4 I 簡單的多居全連接正向網路

前面我們介紹了機器學習頡域中兩個最基本的讀算法： 一個是聽性回歸， 一 個是 Logistic 回歸，下面的部份將聚焦於機器學習中的深度學習。 我們知道課度 學習的前身就是神經網路， 但是由於之前的運算能力與資料量不足、 傳統機器學 習方法如主握向量權完美的數學解釋等原因， 使得深度學習一直沒有發展起來， 宜到最近聽年才有了爆發式的發展， 首先從它們之間的過噓一一神經網路誼算法 入手。

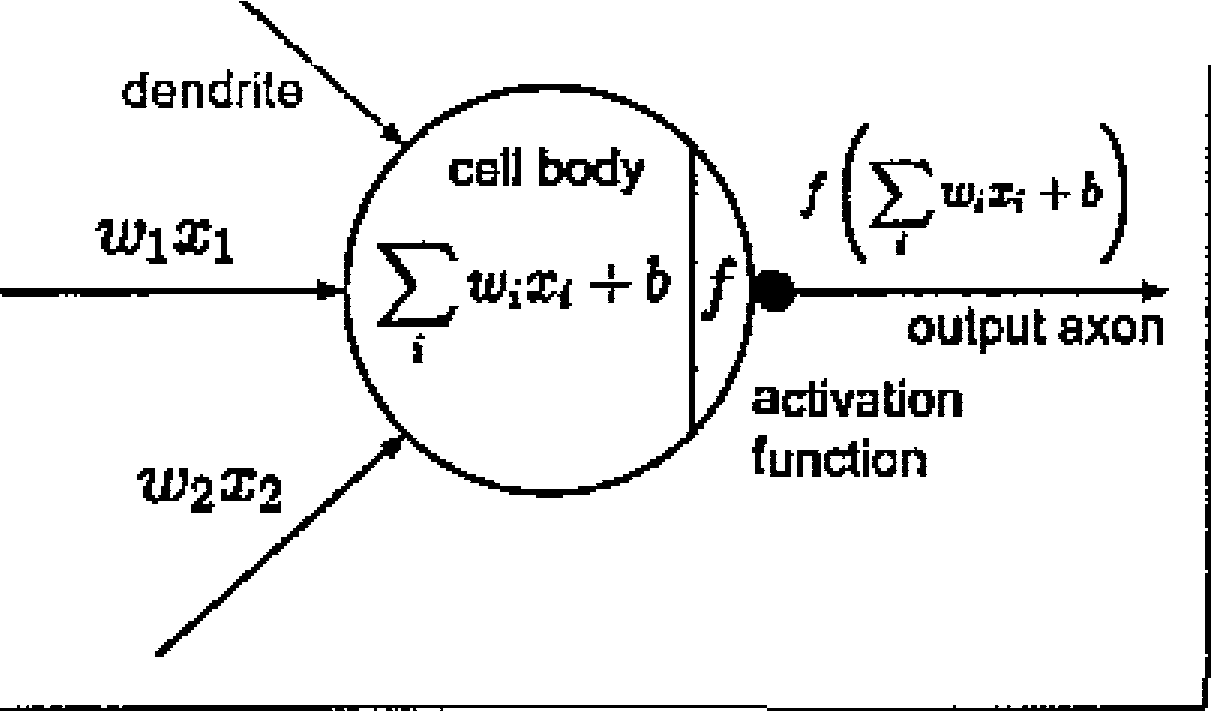
3.4.1 模擬神經元

神經網路最開始是受到了模擺腦神經元的 麓 ，但是現在已經發展成了攝器 學習中的重要演算法，我們先簡要地介紹腦神經科學，接著介紹神權網路。

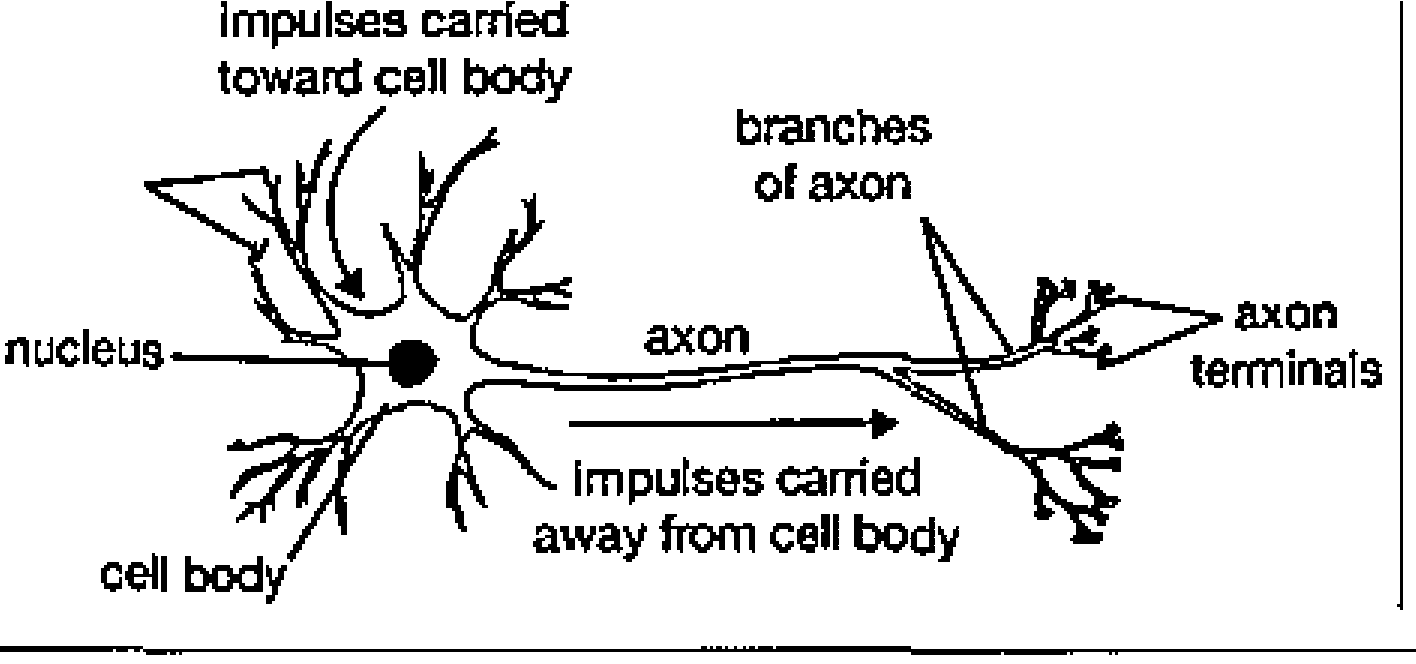
腦中的計算單元是一個簡單的腦神經元， 在人腦的神經系統中 ，大的有 8.6 X 1010 個神經元，它們被 1014 ～ 1015 個更關相連，圖 3.13 能夠簡要地展示一下 腦神經元與神經網路的相似之處。

zo 叩。

帥；，；；； 區函 syn睛，



叩o:i:o



<landrita•

圖 3.13 腦神經完與神經網路相個之處

腦神經元收到一個輸入的訊號 ，透過不同的寞觸，訊號進入神種元， 接著透 過神經元內部的敢動處理， 最佳站著神經元的軸哭產生一個輸出訊號， 這個軸實 透過下一個神經元的突觸相連將輸出訊號傳到下一個神經元。

在神經網路的計算模型中 ，輸入訊號就是我們的輸入資料， 模型的書數競 相當於寶觸，僻、復輸入訊號傳入神經元就像是輸入的資料和模型事數進行騙性組 合，然後經過敢動函數， 最佳傳出模型。

這裡的模型書數透過學習可以控制輸入資料傳入神經元的強度， 毆動函數就 是神經元內部的敢動處理， 最佳將結果輸出變成第二層網路的輸入。 我們之前講

過的 Sigmoid 函數，也就是 logistic 函數σ（x｝＝一上一就是一種毆動函數， 它能夠

l+ e-•

將輸入的資料轉化到 0 ～ 1之間。

但以上只是一個粗糙的腦神經元簡述 ，因為在大腦裡面，有很多不同類型的 神經元，每種神經元的性質都不一樣， 同時神經觸還是一個種雜的動態系統， 而且神經元的輸出也是有時效性的， 輔之道只是一個簡化的神經系統， 但是可以 給我們一些關於神經網路的肢體。

3.4.2 單層神經網路的封類器

一個簡單的一層神經網路的數學形式是特別簡單的， 可以看成一個瞬盟運算 再加上一個敢動函數， 正如前面講過的， 一個神經元可以對一個輸入進行不同的 操作，可以是 「 喜歡」 （ 毆動聲大） 或是 「 不喜歡」 （由動龔小），正是由於鹿動函

－第 3章 多屆全連接神經網路

數的作用，所以我們可以將一層神經網路用做分類 器，正樣本就讓敢動函數敢動 變大，負樣本就讓敢動函數毆動變小 。

例如我們可以使用 Sigmoid 敢動函數做一個二分類問題，這個時候使用數學

形式可以獲得 σ（ I;.J *w*凡*.*

＋*b*） 表示 *P(Y1- l* I *x1 ,*w）， 即輸出為 l類別的機率，那麼輸

出1詣 大類的機率為 *l -P(y1= l* l x1,w）， 是不是看著很熟悉呢 ？將 *P(Y ;=* !I抖，w） 展 開 ，即

*P( y1* = l l x，w） ＝σ（*L:.1 W;X; +b)* (3.30)

I

l + e-(w,,,+b) (3.31)

e"'"

l+e川崎 ＋*b*

(3.32)

這就是前面講過的 Logistic 回歸，所以 Logistic 回歸只是一個使用了 Sigmoid

作為敢動函數的一層神經網路。

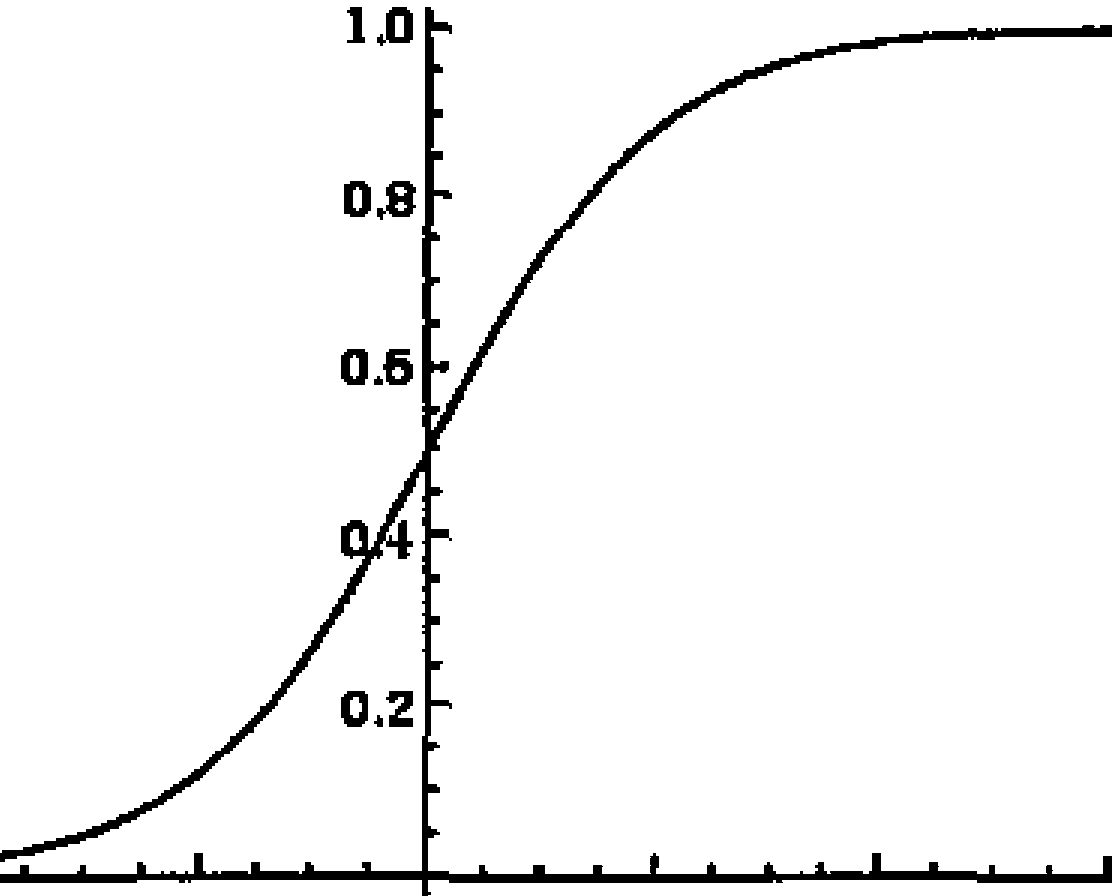
3.4.3 敢動函數

上面我們知道了使用 Sigmoid 敢動函數的一層神經網路就是 Logistic 回歸， 可以看到敢動函數也佔據重要的地位， 下面我們將介紹一下神經網路中 各種常用 的敢動函數。

::.i 1.Sigmoid

Sig血oid 非卡線J帥動函數的數學運算式是 σ（*x*

目前我們知道 Sigmoid 毆動函數是將一個實數輸入轉化到 0 ～ l 之間的輸出，實 際來說也就是將越大的負數轉化到越接近 0 ，越大的正數轉化到越接近 1 。歷史上 Sigmoid 函數頻繁地使用，因為其具有良好的解釋性。



-6 -4 -2 2 4 Ii

圖 3.14 Sigmoid 函數置酷

但是最近蟲年， Sigmoid 敢動函數已經越來越少地被人使用了， 主要是固為

Sigmoid 函數有以下兩大缺點 ，

( 1) Sigmoid 函數會造成梯度消失。一個非常不好的特點就是 Sigmoid 函數在接近 l 和 0 的兩端時，梯度會聽乎變成 0，我們前面講過梯度下降法透過梯度乘上 學習車來更新書數， 因此如果梯度接近 0 ，那麼挂有任何賣部來更新書數， 這樣就會造成模型不耽斂 另外，如果使用 Sigmoid 函數，那麼需要在初始 化加權的時儷也必讀非常小心。 如果初始化的時帳加權太大， 那麼經過毆動 函數也會導至立大多數神經元變得飽和， 挂有辦法更新吾吾數。

( 2 ) Sigmoid 輸出不是以 為平均值，這就會導致經過 Sigmoid 毆動國數之佳的 輸出，作搞接面一層網路的輸入的時帳是非 0 平均值的，這個時帳如果輸入 進入下一層神經元的時﹛晨全是正的，這就會導.樺度全是正的，那麼在更新 吾吾數的時（民永遷都是正梯度，怎麼了解呢 ？倒如進入下一層神經元的輸入是

X’吾吾數是 w 和 b，那麼輸出就是 *f=wx+b* ，這個時棋 *'vf(w*﹜ ＝*x*’所以如果 x

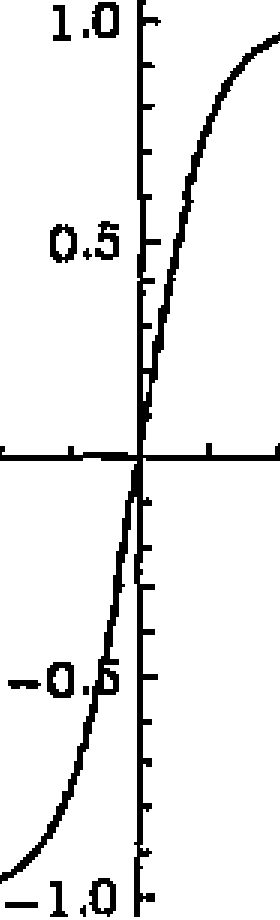
是 0 平均值的資料，那麼梅度就會有正有負。但是這個問題並不是太嚴重， 因為一艘神經網路在訓練的時帳都是按 batch （ 批） 進行訓朧的，這個時棋可 以在某種程度上緩解這個問題， 所以說雖然 0 平均值這個問題會產生一些不 好的影響，但是整體來講跟上一個缺點： 梯度消失相比還是要好很多 ，

．－ 第 3 章 多圖畫連接神經網路

2過 2. Tanh

Tanh 毆動函數是上面 Sigmoid 敢動函數的變形， 其數學表達 2孟 加*nh(x);,;*

2σ（2.x）一 I ，圓形如圖 3.15 所示。



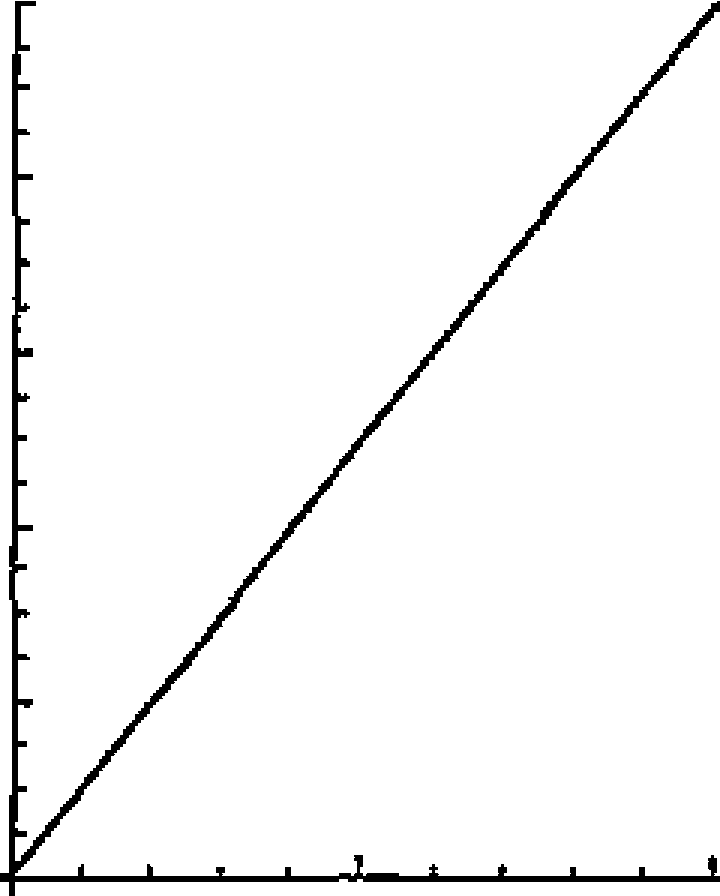
-10 *-5 5* 10

圖 3.15 Tan h 函數圖酷

它將輸入的資料轉化到 一 l ～ 1之間，可以遭遇影像看出它將輸出變成 7 0 平均值，在某種程度上解決了 Sigmoid 函數的第二個問題，但是它的然存在梯度 消失的問題。因此實際上 Tanh 敢動函數總是比 Sigmoid 歐動函數更好。

3. ReLU

ReLU 毆動函數 （ Recti面ed Linear Unit ） 近聽年變得越來越流行，它的數學運 算式為*f (w) =max(O* ,x｝， 換句話說，這個由動函數只是簡單地將大於 0 的部分保 留，將小於 0 的部分變成 0，它的圖形如圖 3.16 所示。



10

自

眉

4

2

-JO *-5 5* 10

圖 3.16 ReLU l函數圖形

下面我們簡單來介紹一下 ReLU 敢動函數的優缺點。

ReLU 的優點

( 1） 相比於 Sigmoid 敢動函數和 Tanh 敢動函數，ReLU 敢動函數能夠相當大地 加速隨機梯度下降法的收斂速度， 這因為它是線性的，且不存在梯度消失的 問題。

( 2 ） 相比於 Sigmoid 殷勤函數和 Tanh 敢動函數的複雜計算而言， ReLU 的計算方 法更加簡單，只需要一個設定值過濾就可以獲得結果， 不需要進行一大堆複 雜的運算。

ReLU 的缺點

訓練的時候很脆弱， 例如一個很大的梯度經過 ReLU 敢動函數，更新參數之 後，會使得這個神經元不會對任何資料有敢動現象。 如果發生這種情況之後， 經 過 ReLU 的梯度永遠都會是 0 ，也就表示參數無法再更新了 ，因為 ReLU 敢動函數 本質上是一個不可逆的過程， 因為它會直接去掉輸入小於 0 的部分。在實際操作 中可以透過設定比較小的學習率來避免這個小問題。

:11 4. Leaky ReLU

Leaky ReLU 敢動函數是 ReLU 歐動函數的變式 ，主要是為了修復 ReLU 殷勤 函數中訓練比較脆弱的這個缺點，不將 柄 的部分變成 0 ，而給它一個很小的負 的斜率，例如 0.01 ，它的數學形式可以表現為／（ w) *=I( x<O* ） （ 即）＋*I ( x 2e: O)( x*），其 中α 是一個很小的常數， 這樣就可以使得輸入小於 0 的時候也有一個小的梯度。 關於 Leaky ReLU 敢動函數的效果， 眾說紛耘，一些實驗證明很好， 一些實驗證 明並不好。

同時也有人提出可以對 α 進行參數化處理，也就是說可以在網路的訓練過程 中對 α也進行更新 ，但是否對所有情況都有效， 目前也不清楚。

．四－ 第 3章 多屬全直接神經網路

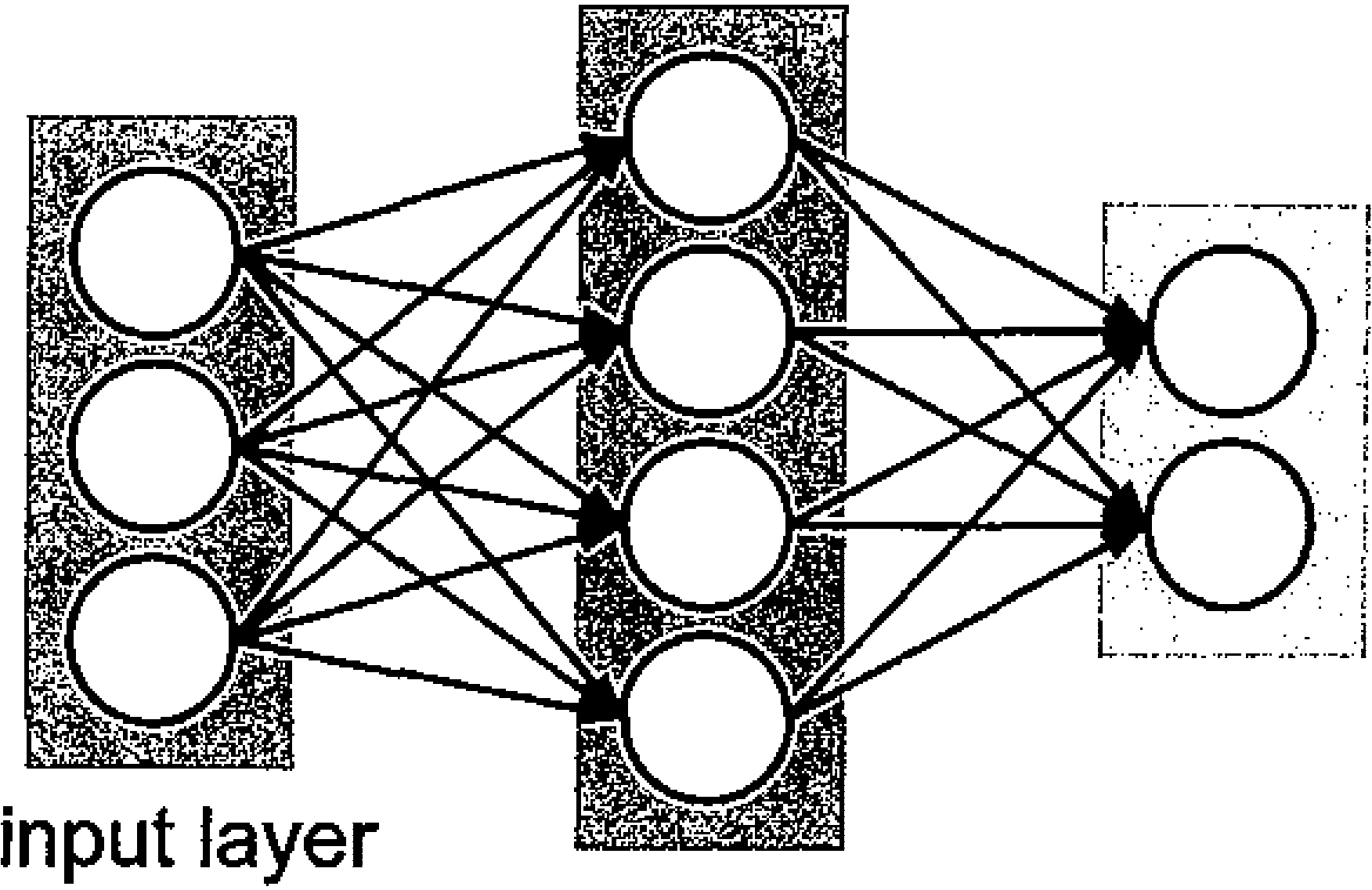
：過 5. Maxout

另外一種毆動函數的額型並不是 *f(wx+b* ） 作用在一種輸出結果的形式， 而是 *max(w1x* ＋丸，wi,:t + b2） 這種 Maxout 的頡型，可以發現 ReLU 由動函數只是 Max­ out 中w1 =0, b1 = 0 的特殊形式。因此 Maxout 既具有 ReLU 毆動函數的種點， 同時 也避免了 ReLU 毆動函數訓練脆弱的缺點 不過，它也有一個缺點，那說是它加 倍了模型的事數， 導致了模型的儲存盤大。

遺過上面的部分我們簡單地介紹了一些敢動函數的優缺點，在實際我們使用 較多的還是 ReLU 毆動函數，但是需要注意學習辜的設定不要太大了﹔ 一定不要 使用 Sigmoid 毆動函數，可以試試 Tanh 毆動函數，但是一般它的效果都比 ReLU 和 Maxout 差 最佳一點，我們在實際使用中也很少使用混合輯型的毆動函數， 也 就是說一艘在同一個網絡中我們都使用同一種輯型的毆動函數。

3間 4.4 神經網路的結構

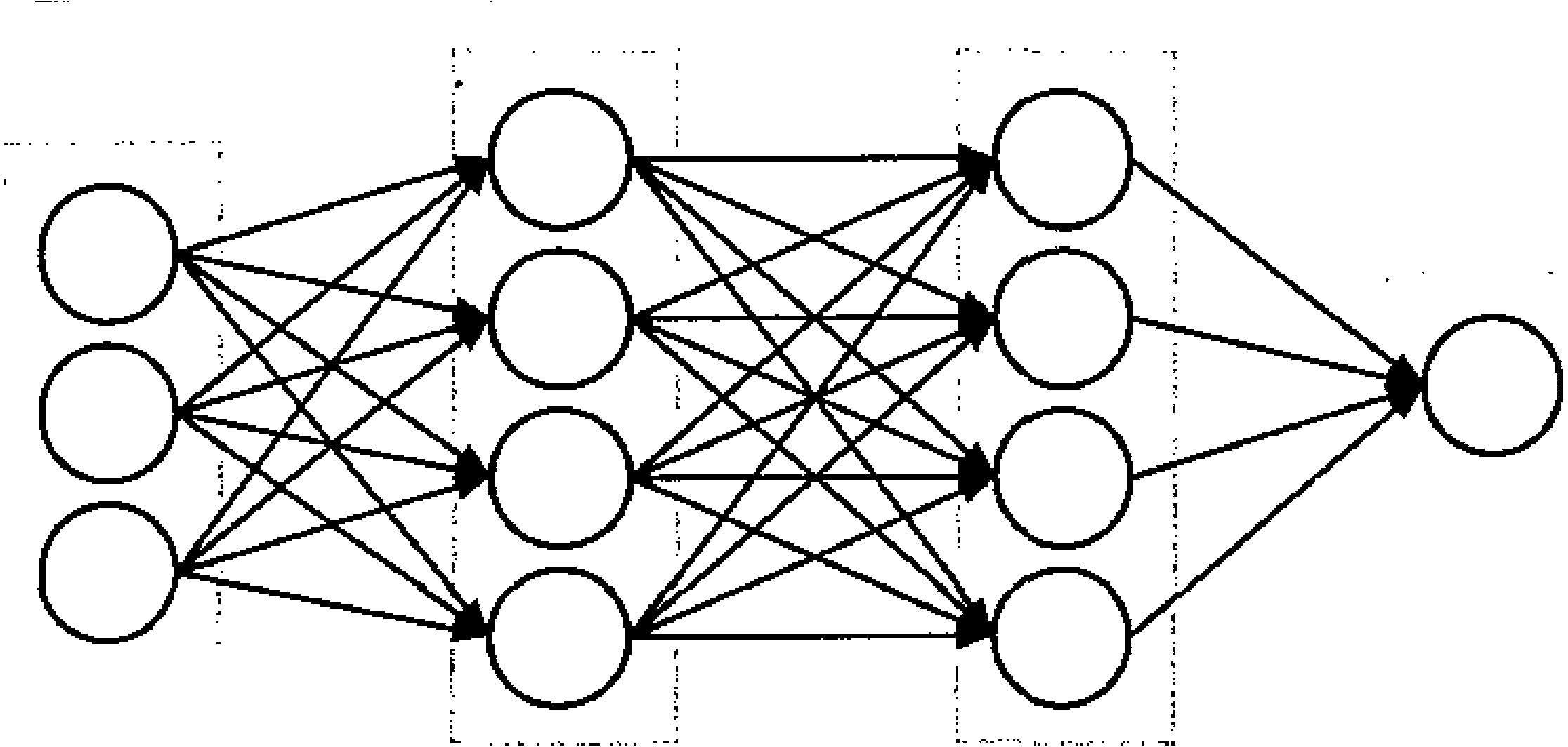
神經網路是一個由神鱷元組成的無環圖， 揖句話說一些神鱷元的輸出會變成 另外一些神經元的輸入， 環是不被允許的， 因為這樣會造成網路中的無限循環。 同時神經網路一般是以層來組織的， 最唔 見的神經網路就是全連接神經網路， 其 中兩個相鄰層中每一個層的所有神經元和另外一個層的所有神經元相 謹，每個層 內部的神經元不相謹，如圖 3.17 和圖 3.18 所示 。



output layer

hidden layer

圖 3.17 hidden layer



output layer

input layer

hidden layer 1 hidden layer 2

圖 3.18 hidden layer 1 揖 hidden layer 2

在之前的線性模型和 Logistic 回歸中 ，我們已體接觸到了 nn.Linear （扭，out) •

它就是在 PyTorch 裡用來表示一個全連接神經網路層的函數， 倒如輸入層 4 個

節 點， 輸 出 2 個 節 點， 可以用 nn.Line盯（4, 2） 來表 示， 同時 nn.Linear(i且，o祉， bias=False） 可以不使用偏置，預設是 True •

一般而言，N層神經網路並不會把輸入層算進去， 因此一個一層的神經網路 是指控有躊躇層、只有輸入層和輸出層的神經網路，Logistic 回歸就是一個一層的 神經網路。

輸出層一般是沒有殷勤函數的， 因為輸出層通常表示一個錯別的得好或回歸 的一個實值的 目標，所以輸出層可以是任意的實數。

3.4.5 模型的表示能力與容量

前面我們通過腦神經結構引出了神經網路的層結構， 如果從數學的角度來解 釋神經網路，那麼神鱷網路就是由 網路中的事數決定的函數釀。所謂的函數聽說 是一系列的函數， 這些函數都是由網路的事數決定的 e 提出了函數簇之後，我們 就想明確這個函數麓的表達能力，也就是說們想知道是否有函數是這個函數接無 法表達的？

這個問題在 1989 年就被人證明過， 擁有至少一個隱藏層的神經網路可以逼 近任何的連續函數。如果一個只有一層隱藏層的神經網路就能夠 逼近任何連續函 數，為什麼我們還要使用更多層的網路呢 ？

這個問題可以這麼去了解，理論上講增加的網路層可以看成是一系列值等轉 換的網路層，也就是說這些網路層對輸入不做任何轉換， 這樣這個深層的網路結 構至少能夠達到與這個淺層網路相同的效果 同時在實際使用中我們也發現更深 層的網路具有更好的表現力， 同時具有更好的最佳化結果。

在實際中 ，我們可能會發現一個三層的全連接神經網路 比一個兩層的全連接 神經網路表現更好， 但是更深的網路結構，例如 4 層、5 膚、6 層等對全連接神經 網路效果提升就不大了。 而與此比較的是旋積神經網路的深層結構則會有更好的 效果，下一章我們會詳細介紹。

知道了多層網路具有 比較好的表現能力之後，如何來設定網路的參數呢 ？例 如我們應該設計為幾層，每層的節點又應該設計為多少個等。首先需要注意的是 如果我們增大網路的層數和每層的節點數， 相當於在增大網路的容量，容量的增 大表示網路具有更大的潛在表現能力，可以用圖 3.19 來說明一下不同網路容量訓 練之後的效果。

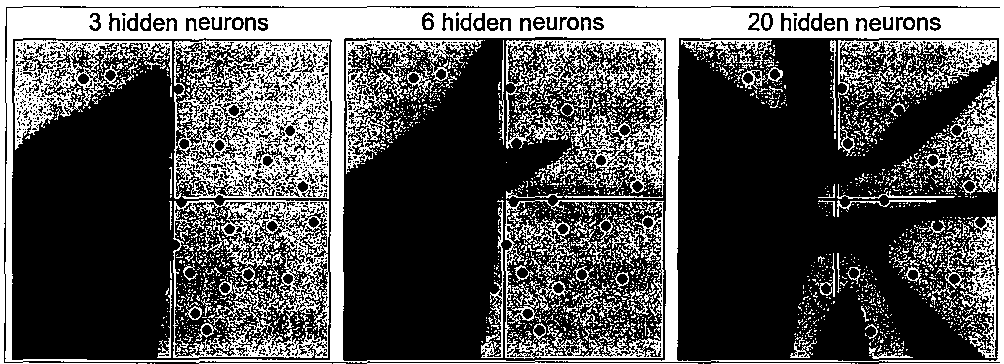


圖 3.19 不同網路容量訓練結果 上面三張圖分別是三個網路模型做二分類獲得的結果， 每個網路模型都是一

個隱藏層，但是每個隱藏層的節點數目 不一樣，從左到右分別是 3 個、6 個和 20

個隱藏節點，這三個模型訓練之後獲得的結果完全不一樣 ，可以看到隱藏節點越 多的模型能夠表示更加複雜的模型， 然而根據我們想要的結果，其實最左邊的模 型才是最好的， 最右邊的模型雖然具有更加複雜的形狀， 但是它忽略了潛在的資 料關係，將雜訊的干擾放大了，這種效果被稱為過擬合 C ove珀社ing ）。

從最右邊的圖片我們可以看出雖然模型成功地將紅色的點和綠色的點完全分 閉，但是卻付出了很大的代價， 將區域分割成了很多分離的區域 ，最左邊的圖雖 然模型無法將所有的點都 完全分閱，但是形成了兩個大區域， 在實際應用中，這 樣的結果抗噪能力和泛化能力反而更強。

透過上面的討論，我們知道了如果資料不太複雜，那麼容量較小的模型反而 具有更好的效果， 是不是我們可以用小的模型去處理呢 ？答案並非如此，首先透 過這樣的方式沒有辦法極佳地衡 量到底多小的模型才算是小的模型 ，其次小的模 型在使用如梯度下降法等訓練的時候通常更難。

因為神經網路的損失函數一般是非凸 的，容量小的網路更容易陷入局部極小 點而遠不到最佳的效果， 同時這些局部最小點的方差特別大， 換句話說 ，也就是 每個局部最優點的差異都特別大， 所以你在訓練網路的時候訓練 10 次可能獲得的 結果有很大的差異。 但是對於容量更大的神經網路，它的局部極小點的方差特別 小，也就是說訓練多次雖然可能陷入不同的局部極小點， 但是它們之間的差異是 很小的，這樣訓練就不會完全依靠隨機初始化。

所以我們更希望使用大容量的網路去訓練模型 ，同時運用一些方法來控制網 路的過擬合，這些方法在之後的 3.7 節會詳細說明。

J 3.5 J 深度學習的基礎： 反向傳播演算法

前一章我們介紹了正向傳播的多層全連接神經網路， 一個核心的問題就是列 出了損失函數f 我們需要更新參數那就需要算出 f的梯度 *'Vf(x* ） ，那麼我們如何 能夠有效地求出這個梯度呢 ？

反向傳播演算法就是一個有效地求解梯度的演算法， 本質上其實就是一個鏈 式求導法則的應用，然而這個如此簡單而且顯而易見的方法卻是在 Roseblatt 提出 成長日器演算法後將近 30 年才被發明和普及的，對此 Bengi 這樣說道：「 很多看似 顯而易見的想法只有在事後才變得顯而易見。」

下面我們就來詳細將一講什麼是反 向傳播演算法。

3.5.1 鏈式法則

首先來簡單地介紹一下鏈式法則，考慮一個簡單的函數，例如f恥 *y ,z)= (x+y)*

？我們當然可以直接求出這個函數的微分， 但是這裡我們要使用鏈式法則， 令

*q* ＝句 ，那麼*f=qz*，對於這兩個式子， 分別求出它們的晰 ，-f *=z* 多＝*q •*

同時 q是 x 和y 的棚 ，所以能夠獲得 年＝I年＝I。我們關心棚題是 4

4 *\_gf\_* 鏈式法則告訴我們如何來計算出它們 的值。 －

*iJv* ’*oz*

*\_gf\_* ＿ ι (3.33)

*ox*

*OX 8q*

*\_gf\_* = *\_}j £\_*ι （3.34)

*iJy iJq iJy*

*8{*

*az =q* (3.35)

透過鏈式法則知道，如果需要對其中的元素求導 ，那麼可以一層一層求導， 然後將結果乘起來，這就是鏈式法則的核心，也是反向傳播演算法的核心。

3.5.2 反向傳播演算法

了解了鏈式法則，我們就可以開始介紹反 向傳播演算法，本質上反向傳播i寅 算法只是鏈式法則的應用。 還是使用之前那個相同的實例 *q=x+y ,f=qz* ，透過計 算圖可以將這個計算過程表達出來，如圖 3.20 所示。

3『38

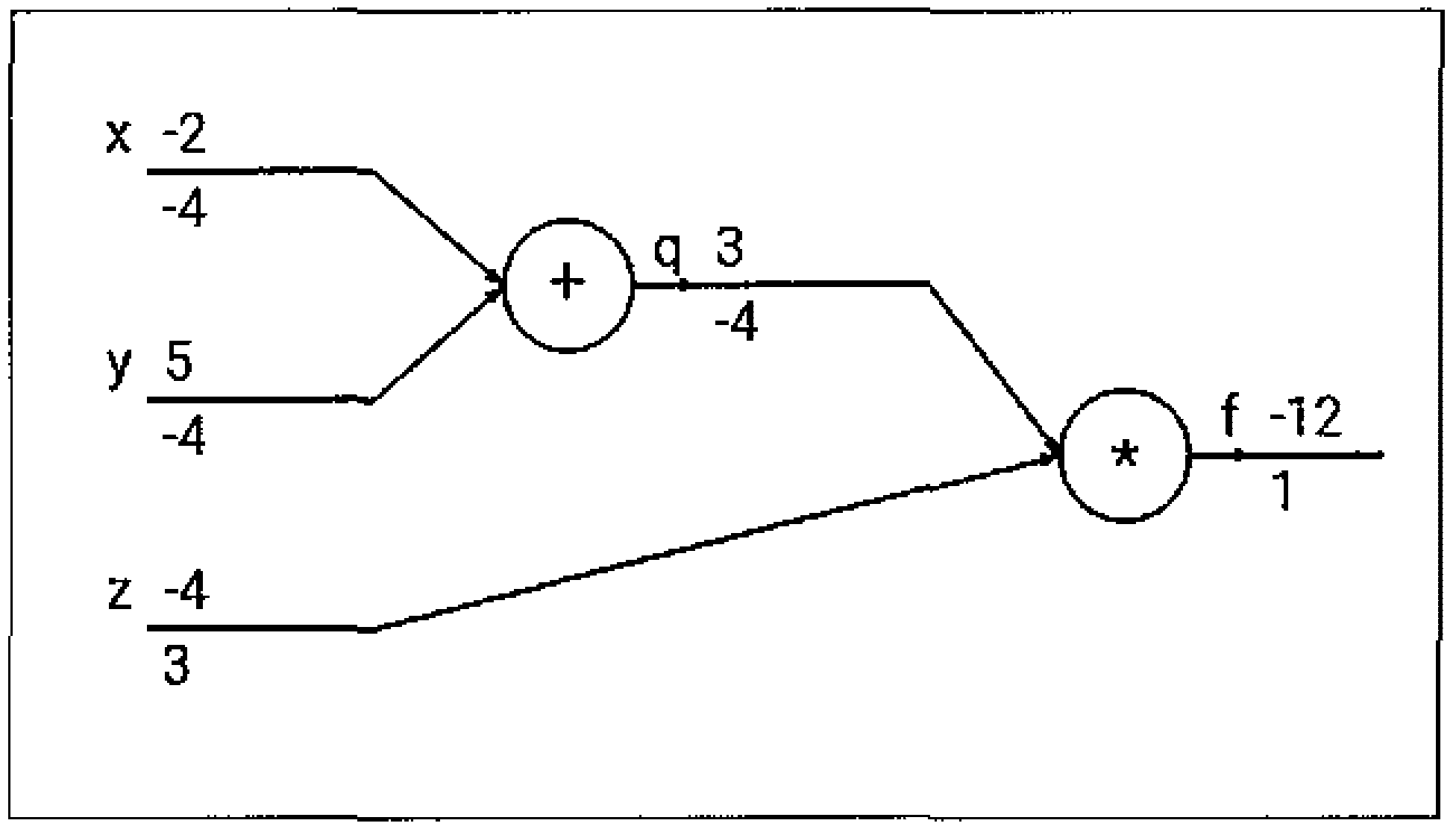


圖 3.20 計算圖

上面的數字表示其數值 ，下面的數字表示求出的梯度， 我們可以一步一步地 看看反向傳播潰算法的實現，首先從最後開始， 梯度當然是 1 ，然接計算 者＝*z* 斗，哥＝戶，接著計算主＝主主 斗X l = -4 *\_g[\_* ＝主主

*OX* δq 缸 ，句 句 δy

= -4 × l = -4，這樣一步一步地就求出了 'vji缸，*y ,z*） 。

直觀上看反向傳播i寅算法是一個優雅的局部過程， 每次求導只是對目前的運 算求導，求解每層網路的吾吾數都是譚過鏈式怯則將前面的結果求出不斷反覆運算 到這一層的，所以說這是一個傳播過程，

* + 1. Sigmoid 函數舉例

下面我們鐘過 Sigmoid 函數來示範反向傳播過程在一個權雜的區數上是如 何 進行的。

*f* （叩）＝*I +*「*e* 、J叮 ....呵呵 可1P (3.36)

，

f!P

我們需要求解出去，者，，首先將這個函數抽象成一個計算區來表示，

*f(x* ）＝ 一一 (3.37)

*X*

*fc( x*﹜ ＝*c+x* {3.38)

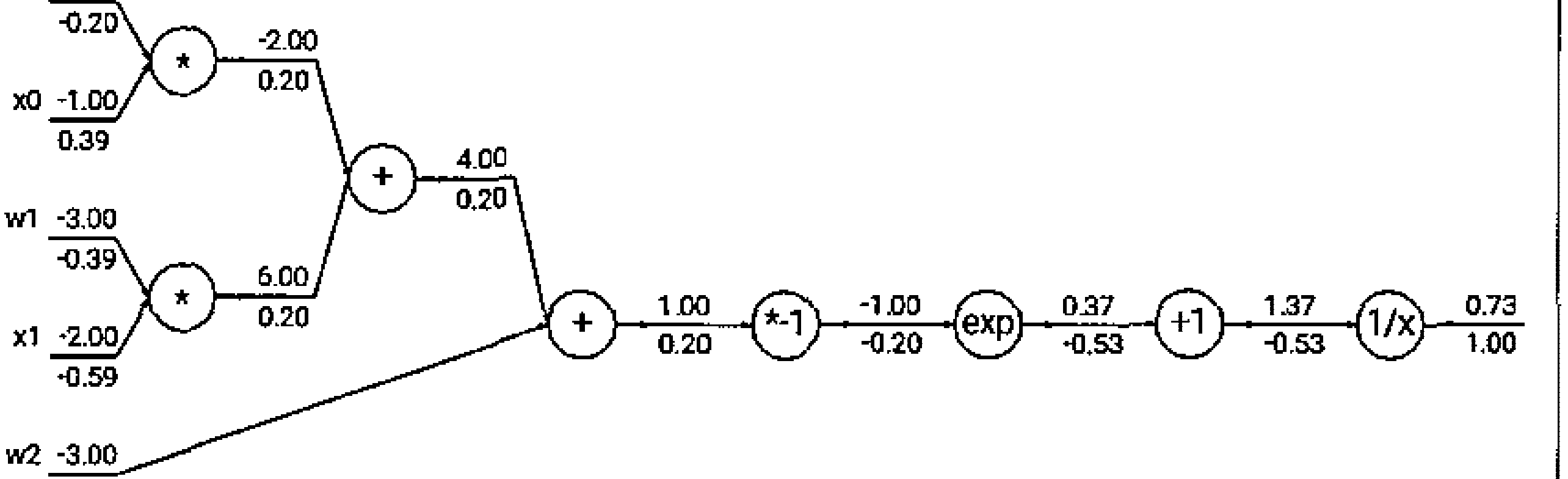
*f (x)=ex* (3.39)

j （*x)=ax* (3.40)

圖圓圓圓－ 第 3章 多居全連接神，極網路

這樣就能鉤畫出如圖 3.21 所示的計算圖。

WO Z口。



口20

圖 3.21 計算圖

同樣圖中的數字表示數值， 下面的數字表示梯度， 從接往前計算一下各個 參數的梯度 首先最佳面的梯度是’，然接經過 土 這個函數，這個函數的蟬度

， 芷

是 一斗，所以 l x一 .l ＝一0 53 往前傳播的梯度是 lx」 一間，然接＋1

*X* 1.372 . 1.37

這個操作，梯度不鹽，撞著是 *0.2ex* 這個運算，它的梯度就是 －0.53e-1=-0.2 ，這 樣不斷往能傳播就能掏求得每個書數的梯度。

邁過鰱式法則我們引用 7反向傳播讀算法， 從上面的實倒詳細地運算了一下 反向傳播揖算詣，這是深度學習中最佳化潰算法的棋心， 因為所有以梯度為基體 的最佳化揖算法都需要計算每個吾吾數的梯度， 下面詳細講一下各種最佳化讀算法 的變式。

[ 3固自 各種最佳化演算法的變式

在前面講 torch.optim 的部分，我們簡單介紹7樟度下降法，下面會先講一講 梯度下降怯的原理，再詳細地介紹各種以梯度為基礎的最佳化方法。

3.6.1 梯度下降法

首先，我們知道梯度下降怯的更新公式如下：

θi＝θ』 *i \_ri'i/L(B;-1)* (3.41)

現在希望透過實際理論來證明這樣更新參數能夠達到最佳的效果。 重新表述一下問題，希望求解下面這個方程式﹒

。＊＝argmin, *L( 6)* (3.42)

也就是說，我們希望更新參數之後有下面的結果：

*L( B0 )> L( B1)> L( B2 )>* ··• (3.43)

在解決這個問題之前，我們需要先回顧一下泰勒級數 （ Taylor Series ）。 對於任何一個無限可微函數 *h(x*），在一個點 *x=x0* 附近，有以下的泰勒級數：

*hl''(x,)*

＝主τ「

*h( x)* （ *x* 有l' (3.44)

*h( x,)* ,

*= h( x0 )+h( x0 )( x* 刊）＋才＇＂－（ *x x0* ）＇÷ (3.45)

當x 足夠接近布，有下面的近似：

*h(x*） 也 *h(x0)* +*h'( x0 )(x-x0 )* (3.46)

對於多元泰勒級數，有以下的公式：

峙，*y )= h( x0 ,y0* ）＋哈拉（*x x* ）＋ 呵且（*y y,)+·* (3.47)

同樣當x F日y 足夠接近 *x*，和*Yo* 的時候，有以下的近似：

月*h( x,, y,) Bh( x, y,)*

*h( x,y)" h( x0 ,y0* ）＋－古」一（刊）＋一步＇..！！＂－ （*y -y,)* (3.48)

現在假設參數 有兩個變數 1 鈍，那麼由上面的二元泰勒級數知道，對於一 個點 仰，*b*） ，對於一個足夠小的範圈，有以下近似 ：

*L( B)=L*紗，*b*）＋ 旦旦*a*包*e1J!l ce1 -a*）＋ 主主*a*已*e*J*,* !lce, bJ (3.49)

．－圓圓 第 3章 多層全連接神經網路

令 *s-L( a ,b) ,u*一 旦*laEe.1,.]jl\_* ,*v*＝笠B包鈍J!2.. ，那麼 *L ( 8* ） 有以下的簡單表達 ：

*L(8*） 句＋*u( 81* α）＋ v(e, *b)* (3.50)

我們知道 *s • u* 和 v 都是常數，所以希望找到 i*,e*，使得 *L( 8* ） 最小 ’同時 *81,*

也 都是在 （α b） 的小範圍之內，所以要滿足以下條件 ：

|  |  |  |
| --- | --- | --- |
| *(81 a)2+( 82-b)2* 三；；*6'*  再換元，將 （*81 -a*）－血肉，（*8, b)2-l'.8*，，同時由於 s是一個與 *el*’ | | (3.51)  2 沒有關係的 |
| 常敏，所以可以去掉它 ，這樣就能夠獲得下面的式子 ． |  |  |
| *L(8)* ""'° *ul'.81 +vl'.82* |  | (3.52) |
| 血 ＋品 2 ,;*6'* |  | (3.53) |

對於上面這個式子， 可以將 （*u* ν） 視為一個向量 ，同時 （*l'.81* 血，） 也是一個向 量，所以 （*u,* v)?(I'.吭 血 *2)-ul'.81* ＋尬 2 。那麼怎麼能夠求到這個內積的最小值呢 ？ 很簡單，只需要保障 （血肉，且句 是 紗，*v*） 的反方向，也就是 （*l'.81 ,l'.8* ）－ 一（*u* ν）， 但 是由於需要把 A吭，占4 都限制在一個小的範圍內，所以要對其範數做一個限制， 也就是：

*(l'.81* ,l'.8,) = 句俏 v) (3.54)

特 （品 1 血，）＝紗，*b*） 一句紗 *v)* (3.55)

所以只要在一個足夠小的範圍 內，更新參數之後就能取得一個更小的值，更 新公式如下所示 ：

*(81* ）＝（a , *b)-17*（ 主但在 皇位J!2..) (3.56)

*881* ’ 2

所以只要取一個特別小的學習率 旬， 就能夠保障 吭，鳥 在一個足夠接近 仰，b)

的範圍內，這就實現了梯度下降法 。 下面介紹一些常見的梯度下降法的變式 。

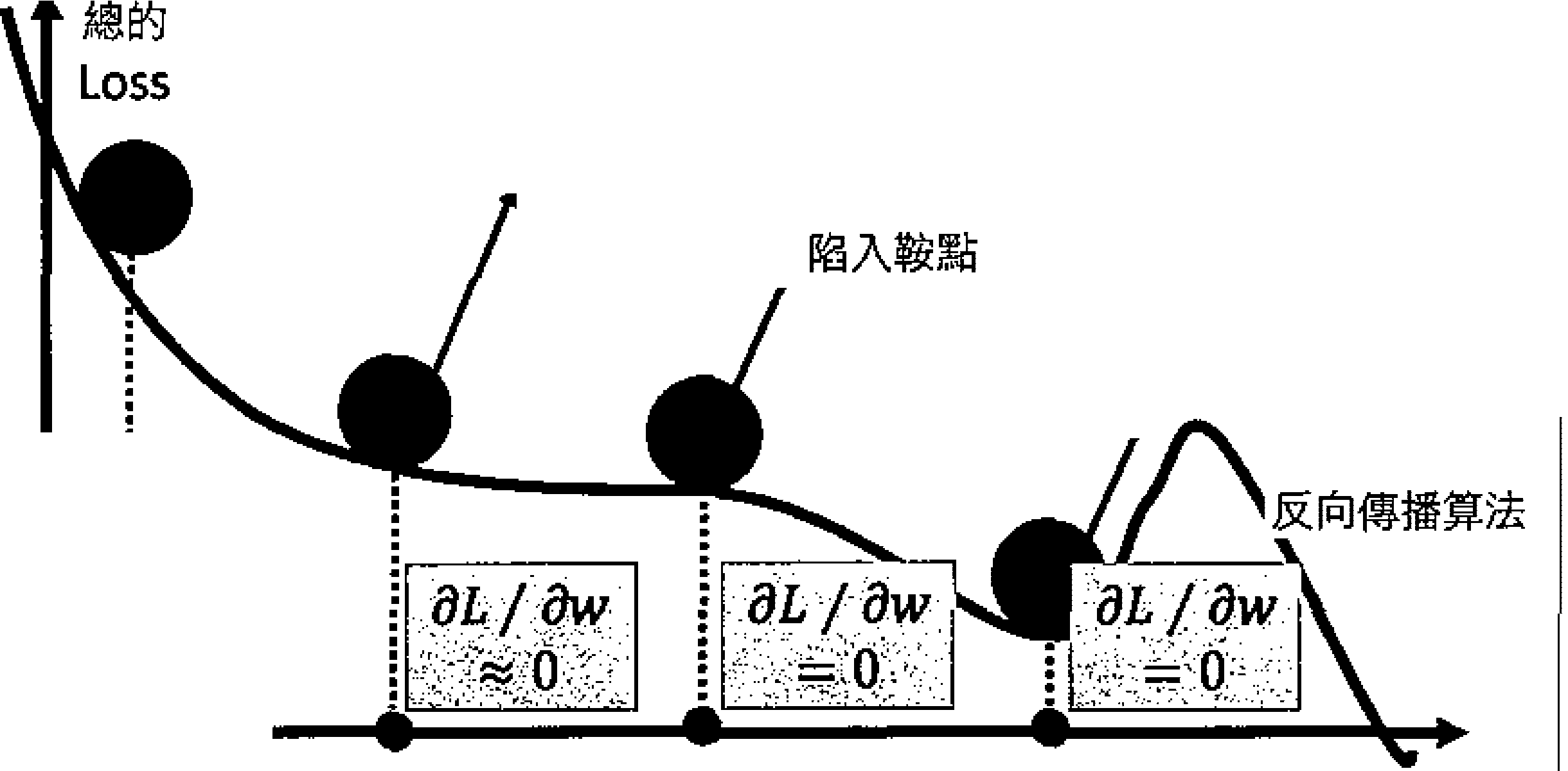
3.6.2 梯度下降法的變式

* + - 1. SGD

隨機梯度下降法是梯度下降怯的小變悟， 就是每次使用一批 （ batch ） 資料 進行梯度的計算，而非計算全部資料的梯度， 因聶現在深度學習的資料量都特別 大，所以每次都計算所有資料的梯度是不現實的 ，這樣會導致運算時間特別長， 同時每次都計算全部的梯度還矢去了一些隨時睦，容易陷入局部誤差，所以使用 幢樓梯度下降法可能每次都不是朝著真正最小 的方向 ，但是這樣反而容易跳出局 部極小點。

* + - 1. Momentum

第二種最佳化方法就是在隨機梯度下降的間時， 增加動量 （ Momentum ）。 這 來自實體中的概念，可以想像揖失函數是一個山谷， →國球健山谷滑下來，在一 個平坦的地勢，球的滑動速度就會慢下來， 可能陷入一些鞍點或局部極小值點 ， 如圖 3.22 所示。



在梯度平穩的地方

非常慢

陷入局部極小值點

--IJt,:

網路參數的數值

圖 3.22 增加動量

－－－－－ 第 3章 多屆全直接神經網路

這個時使替它增加動量就可以讓它能高處滑落時的勢能轉為平地的動龍 ，相 當於慣性增加了小球在平地滑動的速度， 進一步幫助其跳出體點聶局部極小點。

動量怎麼計算呢 ？動量的計算以前面梯度， 也就是說書數更新不僅以目前為 基體為基礎的梯度， 也以之前為基礎的梯度，可以用圓 3.23 來簡單地說明 ，

．＋ 棉度

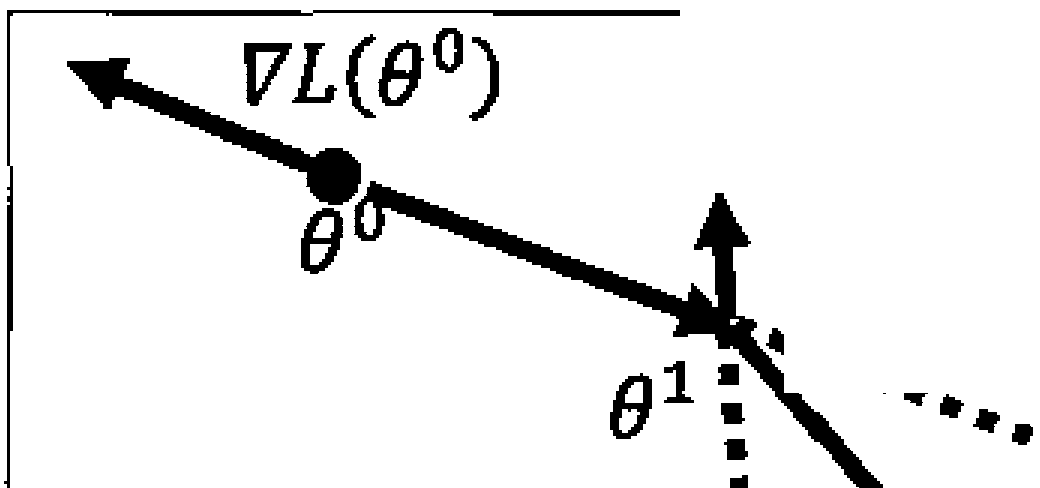
．＋ 移劫方向

..上一步的移

劫芳向

＝ 仰，

q 耍 叫83)



*VL(81)*

﹔ －



*I* ;

圖 3.23 動量的計算

記住我們更新的是梯度的負方 向，紅色表示梯度，藍色表示更新芳 向，圖中 綠色的虛線就是動量，可以看到也就是之前一次梯度的負 方向自

除此之外，對於動量還有一個變形， RP ？、fosterov 。我們在更新書數的時帳需 要計算梯度，傳統的動量方法是計算 目前位置的梯度，但是 Nesterov 的方法是計 算輕過動量更新之後的位置的梯度。

* + - 1. Adagrad

這是一種自我調整學習車 （adaptive） 的方法，它的公式是：

W+I ← w' －τ＝主＝ *g'*

J})g')2 ＋占

鐘過式 （ 3.57 ）， 我們可以看到學習車在不斷變小 ，且受每計算出來的梯 度影響，對於梯度比較大的吾吾數， 它的學習車就會變得相對更小，裡面的根號特

別重要，沒有這個根號演算法表現非常差。 同時 是一個平順參數，通常設定為

10 4 ～ 10-8 ，這是為了避免分母為 0 0

自我調整學習翠的缺點就是在某些情況下一直遞減的學習率並不好， 這樣會 造成學習過早停止。

:11 4. RMSprop

這是一種非常有效的 自我調整學習率的改進方法，它的公式是：

*cache'* ＝α＊*c*α*che'* 1 + (l 一 α）（ g ）＇ (3.58)

＂←

*w'* w' －宇、」－，！＇

／ *cache'* +o (3.59)

這裡多了一個 α ，這是一個衰減率，也就是說 RMSprop 不再會將前面所有的 梯度平方求和，而是透過一個衰減率將其變小， 使用了一種滑動平均的方式 ，越 靠前面的梯度對自我調整的學習率影響越小， 這樣就能更加有效地避免 Adagrad 學習率一直遞減太多的問題，能夠更快地收斂。

:11 5. Adam

這是一種綜合型的學習 方法，可以看成是 RMSprop 加上動量 （ Momentum )

的學習方法，達到比RMSProp 更好的效果。

以上介紹了多種以梯度為基礎的參數更新方法， 實際中我們可以使用 Adam 作為預設的最佳化演算法， 常常能夠達到比較好的效果， 同時 SGD+Momentum 的芳法也值得嘗試。

I 3.7 I 處理資料和訓練模型的技巧

在開始訓練網路之前，良好的資料前置處理和參數初始化能夠很快達到事半 功倍的結果。在模型訓練中採用一些訓練技巧， 能夠使得模型最後達到 state-of­ art 的效果，這一節我們講一講處理資料和訓 練模型的技巧。

間圓圓圓－ 第 3章 多居全連接神值網路

3.7.1 資料前置處理

道 1. 中，oft

資料前置處理中一個最常見的處理辦法就是每個特徵緯度都攝去對廳 的平均 值實現中心化，這樣可以使得資料變成 0 平均值，特別對於一些圖像資料，為了 方使我們將所有的資料都減去一個相同 的值。

油 2. 標準化

在值得資料都變成 0 平均值之後，還需要使用標車化的做怯讓資料不同的特 橄維度都具有相 同的揖模。有兩種常用的方法：一種是除以標準差， 這樣可以使 得新資料的分布接近標車高斯分佈 ﹔還有一種做法就是讓每個特徵緯度的最大值

和最小值按比側縮放到 一 l～ I之間 ，

如果知道輸入不同特徵具有不同的揖模 ，那就需要使用標準化的做法讓它們 處於同一個現模下面， 這對於機器學習讀算法而言是非常重要的 。

我們可以透過圖 3.24 看看每種做法處理完資料之麓 的故果 e

。吋ginal data zero-centered data normar12ed da隘

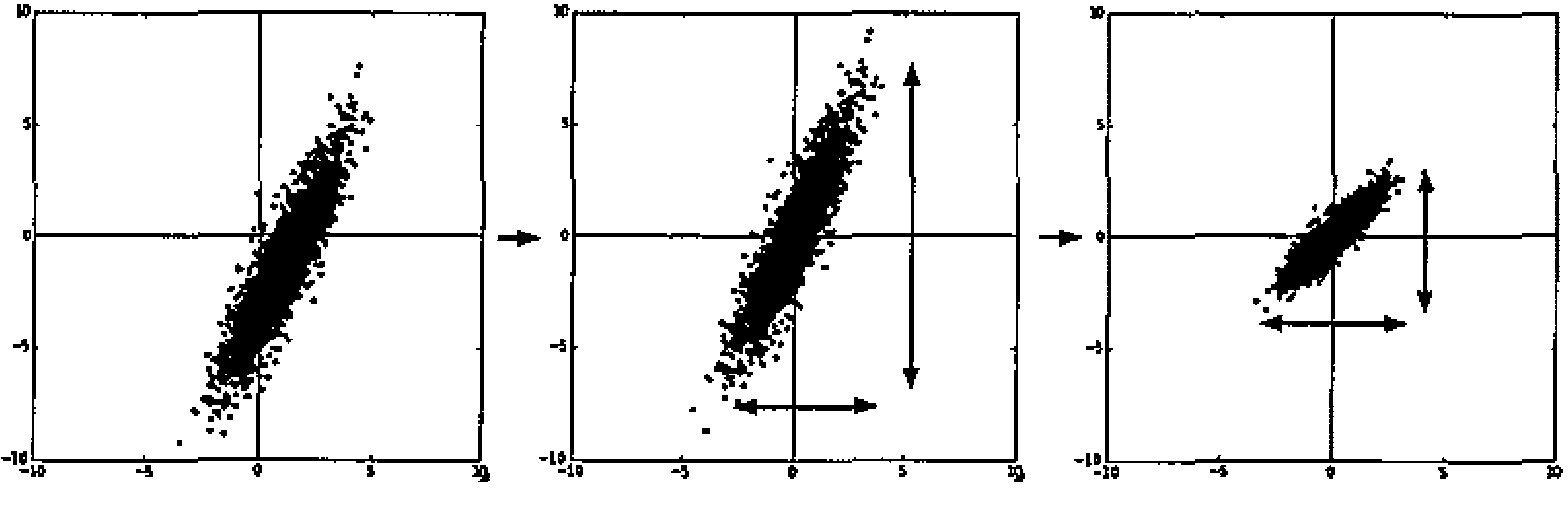


圖 3.24 賣料處理觀果

3. PCA

PCA 是另外一種處理資料的方法 ，在進行這一步以前， 首先會將資料中心 化，然接計算資料的協方差矩陣， 這一步特別簡單， 但設輸入是 *X=N X D* ，那

3.7 處理賣輯和首 II褲模型的揖巧 回

麼遭遇 于能夠獲得這個協雄喧陣， 可以瞬一下這個結果的正確性， 而 且這個協方差矩陣是對稱半正定的， 可以透過這個協方差矩陣來進行奇異值分解 ( SVD ）， 然接對資料進行去相關性， 將其投影到一個特擻空間， 我們能揖取一些 較大的、主要的特徵向量來降｛*It*資料的維數，去掉一些沒有方聾的維度，這也叫 做主成分分析 （ PCA ）。

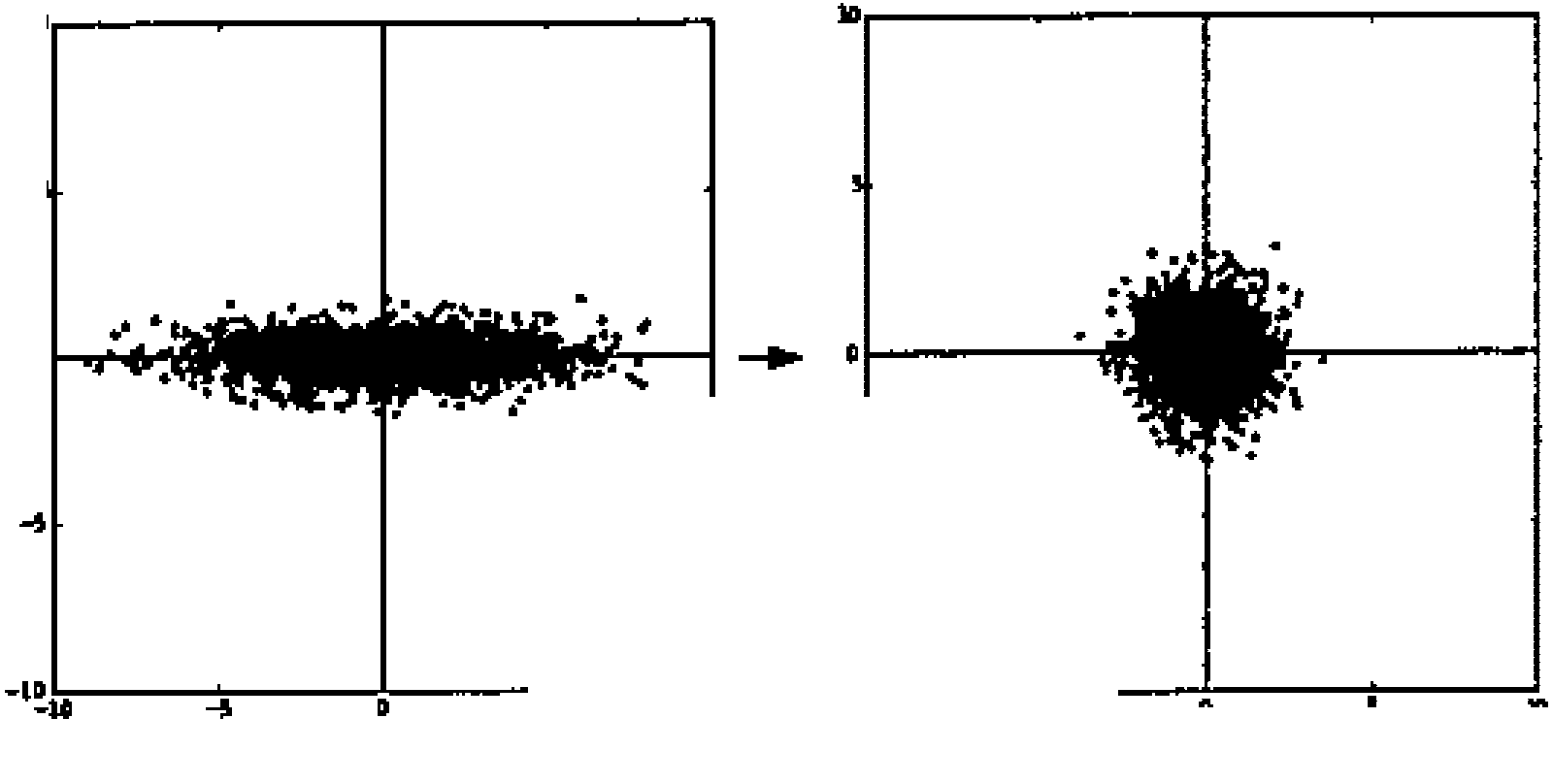
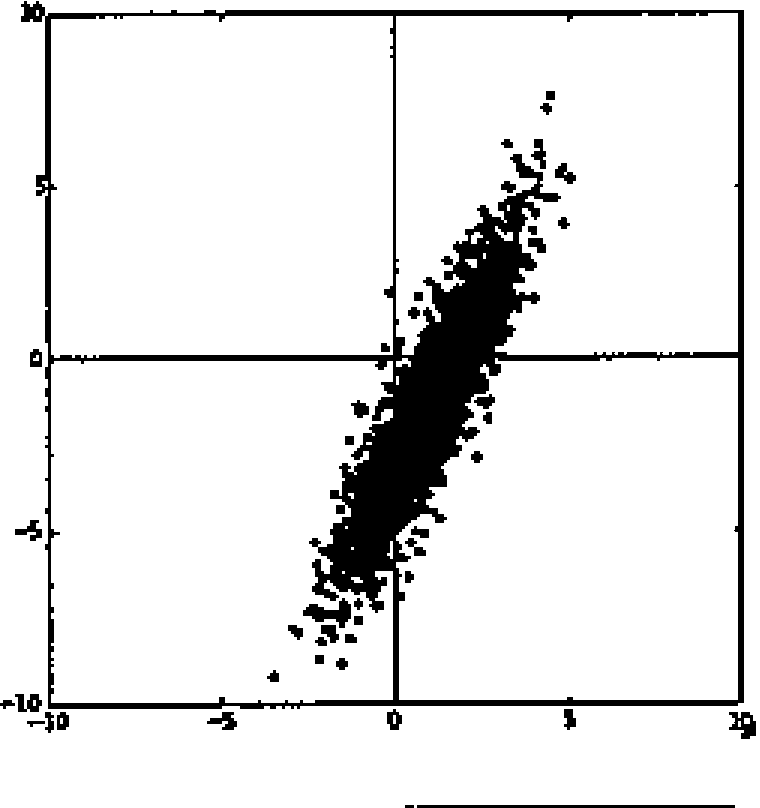
這個揖作對於一些線性模型和 神經網路，都能取得良好的教果。

'.:lJ 4. 自色雜訊

白色雜訊也是一種處理資料的方式，首先會跟 PCA 一樣將資料投影到一個特 骰空間，然接每個維度除以特擻值來標準化這些資料， 直觀上就是一個多元高斯 卦佈轉化到了一個 0 平均值，協方差矩陣主告 I 的多元高斯分佈。

實際的 PCA 和白色雜訊處理之聾的故果如 圖 3.25 所示。

。『iginal data decorrelated data whitened data



．－ －

種 -"!J;-一一一....，.

圖 3.25 PCA 和自色雜訊處理觀果

圖 3.25 具體地展示了 PCA f日自色雜訊處理之佳的效果， 但是自色雜訊的處 理會增強資料中的雜訊，因為其增強了資料中的所有維度，包含了一些芳差很小 的不相關的維度 ，

在實際處理資料中 ，中心化和標準化都特別重要。 我們計算訓練集的統計量 側重日平均值，然接將這些統計量應用到制試集和聽證集當中 。但是 PCA 和白色雜 訊在提積網路中基本不使用，因為雄積網路可以自動學習如何分析這些特徵而不 需要人工再去對其進行干頭 ，

3.7.2 加權初始化

前面介紹了資料的前置處理， 在進入網路訓練之前， 需要做參數的前置處 理。下面介紹前置處理的策略。

油 1全 0 初始化 首先從一個最直觀，但是不應該採用的策略入手， 那就是將參數全部初始化

為 0 。

解釋一下為什麼不能採用這一種策略。 首先，我們並不知道訓練之後的網路 最後加權更新的是多少， 但是知道資料在進入網路之前經過了合適的前置處理， 所以我們可以假設最後的加權有一半是正的， 一半是負的，所以將參數全部初始 化為 0 似乎是一個非常好的選擇。 但這是不對的，因為如果神經網路中每個加權 都被初始化成相 同的值，那麼每個神經元就會計算出相 同的結果，在反向傳播的 時候也會計算出相同的梯度， 最後導致所有權重都會有相 同的更新。換句話說， 如果每個加權都被初始化成相同 的值，那麼加權之間失去了不對稱性。

油 2. 隨機初始化

目前知道我們希望加權初始化的時候能夠儘量接近 0 ，但是不能全都等於 0 ' 所以可以初始化加權為一些接近 0 的亂數，透過這個方式可以打破對稱性。 這裡 面的核心想法就是神經元最開始都是隨機的 、唯一的，所以在更新的時候也是作 為獨立的部分，最後一起合成在神經網路當中 。

一般的隨機化策略有高斯隨機化、 均勻隨機化等， 需要注意的是並不是越 小的隨機化產生的結果越好， 因為加權初始化越小，反向傳播中關於加權的梯度 也越小，因為梯度與參數的大小是成 比例的，所以這會相當大地減弱梯度流的訊 號，成為神經網路訓練中的隱憂。

這個初始化策略還會有一個問 題就是網路論出分佈的方差會隨著輸入維度的 增加而增大，可以用下面的數學式子來說明 ：

均*r( s)* = *Var(4w*凡） (3.60)

= *I;var* （ 冉冉） (3.61)

＝主﹝*E* （川）﹞*2Var( x,)+ E*﹝（*x,)]'Va*咖，）＋ *Var( x,)Var*（叫） (3.62)

= *4Var (x,) Var( w1 )* (3.63)

*=( n*均*r*（刊•））*Var(x)* (3.64)

其中假設輸入和加權都是 0 平均值的，也就是說 E﹝*x*，﹞ ＝*E* ﹝*w*，﹞＝ O ，但這並不是 一般的情況，例如經過了 ReLU 殷勤函數之後輸出就會是一個正的平均值， 這裡

做這樣的假設是為了方便計算。 我們可以看到輸出的結果 S比輸入x 的方差增大 了 *nr*色列*w*） 倍，如果網路越來越深，就會導致方差越來越大，所以希望 *nVar( w*） 盡 可能接近 1，也就是 枷（w） 斗，這可以獲得 均*r(,*

需要除以、后 。

訕 3. 稀疏初始化

另外一種初始化的芳法就是稀疏初始化， 將加權全部初始化為 0，然後為了 打破對稱性在裡面隨機挑選一些參數附上一些隨機值。 這種方法的好處是參數佔 用的記憶體較少，因為裡面有較多的 0 ，但是實際中使用較少。

訕 4. 初始化偏置 （ bias )

對於偏置 ias）， 通常是初始化為 0 ，因為加權已經打破了對稱性， 所以使用

。來初始化是最簡單的。

油 5. 批次標準化 （ Batch Normalization )

最近興起的一項技術叫做批次標準化， 它的核心想法就是標準化這個過程 是可微的，減少了很多不合理初始化的問題， 所以我們可以將標準化過程應用到

神經網路的每一層中做正向傳播和反向傳播，通常批次標準化應用在全連接層後 面、非線性層前面。

實際中批次標準化已經變成了神經網路中 的標準技術，特別是在旋積神經網 路中，它對於很壞的初始化有很強的堅固性， 同時還可以加快網路的收斂速度。 另外，批次標準化還可以視為在網路的每一層前面都會做資料的前宣處理。

3.7.3 防止過擬合

之前在 3.4.5 節中我們講到了如果網路容量過大會造成過擬合，但是防止過擬 合的最佳辦法並不是減少網路容量 ，那麼如何防止過擬合呢 ？下面我們會講幾個 實際的辦法來防止過擬合。

2過 1正規化

*L2* 正規化是正規化 （ regularization ） 中比較常用的形式，它的想法是對於加 權過大的部分進行懲罰， 也就是直接在損失函數中增加加權的二範數量級，也就 是 去恥 其中λ 是正規化強度， 通常使用 5，因為對於 J的梯度是 *2w*，使用 去 就能使得梯度是 λw 而非 以w 。所以使用 *L2* 正規化可以看成是加權更新在原來

的基礎上再 －*AW*’這樣可以讓參數更新之後更加接近 0 。

*LI* 正規化是另外一種正規化方法， 其在損失函數中增加加權的 1範數，也就 是 λ 圳，我們也可以把 *LI* 正規化和 *L2* 正規化結合起來，如λ1 lwl+l2w' 。*LI* 正規 化相對於 *L2* 正規化的優勢是在最佳化的過程中可以讓加權變得更加稀疏，換句話 說，也就是在最佳化結束的時候， 加權只會取一些與最重要的輸入有關的加權， 這就使得與雜訊相闋的加權被盡可能降為 0 。*L2* 正規化的優勢在於最後的效果會 比*LI* 正規化更加發散，加權也會被限制得更小 。

除此之外，還有一種正規化方法叫做最大範數限制， 其迫使加權在更新的過 程中範數有一個上界，也就是 llwll 咐，這種辦法可以使得當學習率設定太高的時 候網路不會 「 爆炸」， 因為更新總是有界的 。

在實際中對於正規化的選擇通常使用 *L2* 正規化，其使用更加常見。

1. Dropout

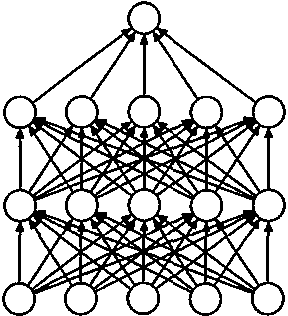
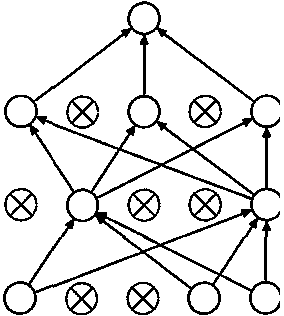
現在介紹一種非常有效、 簡單、同時也是現在深度學習使用最為廣泛 的防止 過擬合的方法 Dropout 。其核心想法就是在訓練網路的時候依機率 P保留每 個神經元，也就是說每次訓練的時候有些神經元會被設定為 0 ，其簡單示意圖如 3.26 所示。

透過圓 3.26 我們可以看到每次訓練都有某些神經元並沒有參與到網路中， 但 是在預測的時候不再這樣處理， 這也很好了解，如果預測應用 Dropout，由於隨機 性，每次預測出來的結果都不一樣，這樣預測的時候完全靠運氣，這顯然是不行 的。所以會保留網路全部的加權，取代應用 Dropout，在每層網路的輸出上應用 P 的縮放。這個想法是很重要的， 因為如果我們不做任何處理， 那麼網路的行為在 預測時和訓練時就會不同，這不是所希望的，所以需要應用縮放。

那麼為什麼在網路輸出部分應用 P縮放可以達到相同的效果呢 ？考慮一個神 經元在應用 Dropout 之前的輸出是 ν 那麼應用 Dropout 之後它的輸出期望值就是

*Px+( l -P*） ，所以在預測的時候，如果保留所有的加權，就必須調整 x → *Px* 來保

障其輸出與期待相同。

* 1. S回dard Ne叫Net (b） 岫E applying 世opout.

圖 3.26 簡單神經元

還有一種了解 Dropout 的想法就是把 Dropout 看作是整合的學習方法，如圖

3.27 所示。

I ﹔

m i 們 i

I

mi ni atch

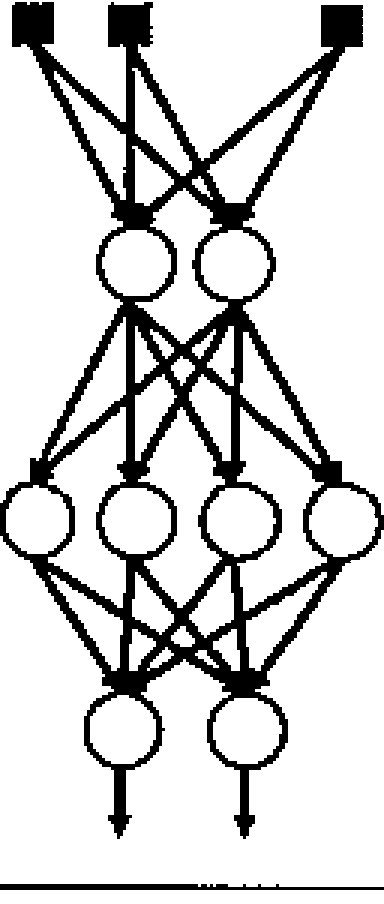
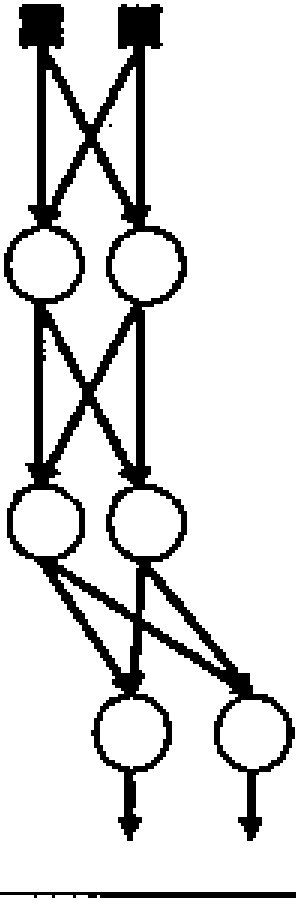
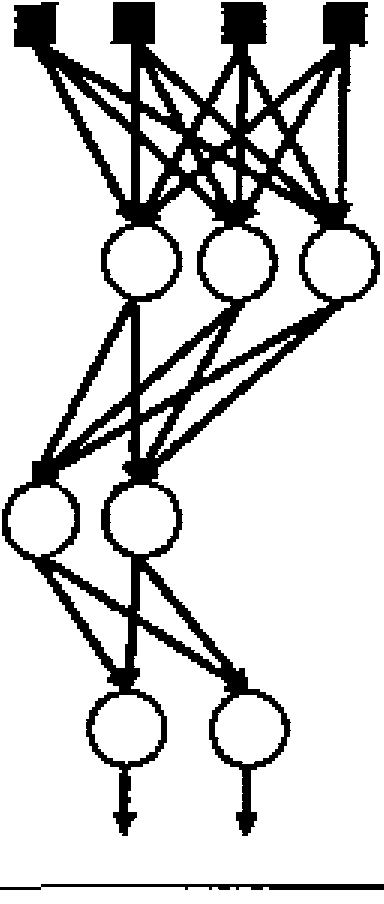
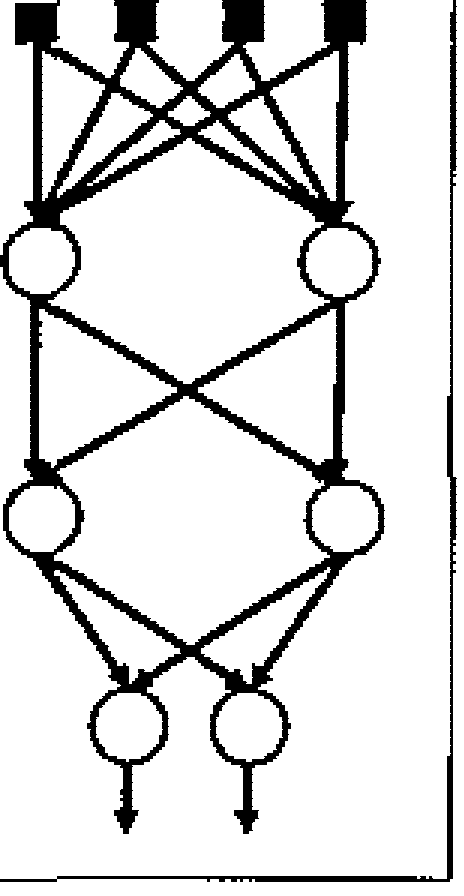
   

圖 3.27 聲音的學習 E法

每一次訓輔 Dropout 之後就可以看作是一個新的模型， 擇手、接訓練了很多次之 後就可以看成是這些模型的整舍 ，

上面什紹了多種防止過擺舍的辦法，在實際應用中，一股探用全蟻加權的 *L2*

正揖化搭配 Dropout 來肪止過攝舍，

I 3.B I 多層全連接神經網路實現 MNIST 手寫數 字分類

前面我們已經講完 7很多理論知識，「 Talk is cheap’show me the code 」 是時

i度來實作一下我們的擴算括了。

先用 PyTorch 實現最簡單的三層全連接神經網路， 然後增加歐動層樟祖試驗 結果，最後再加上批次標準化驗證是否能夠更加有效。

3.8.1 簡單的三層全連接神經網路

在 PyTorch 裡面可以很簡單地定義三屆全連接神經網路﹒

1 class simpleNet ﹛ nn.Module﹜ z

2 def 一＿，，init,..,,,.., (self , in.,,.dim, ncc:hidden,,), n.,,,hidden.,.,2, out,...,dim﹜ z

1. super (simpleNet, self ﹜ init ﹛）
2. self . layerl - nn .Linear ﹛ in dim, n hidden 1)
3. self layer2 - nn Linear ﹛ n\_hidden 1, n hidden ,)
4. self .layer3 - nn .Linear ﹛ n\_hidden\_2 ’。ut\_dim ﹜

B def f。rward (self , x) :

7

1. x - self .layerl ﹝ x)
2. x - self layer2 (x)
3. x - self . layer3 ﹛ X)
4. return x

對於這個三層網路， 需要傳遞進去的參數包含 輸入的維度、 第一層網路 的神經元個敏、第二層網路神經元的個數， 以及第三層網路 （ 輸出層 ） 神經元的 個數。

3.8.2 增加敢動函數

接著改進一下網路，增加敢動函數增加網路的非線性，方法也非常簡單。

工 class Activati。n Net (nn M。dule) :

1. def init (self , in dim, n hidden 1, n hidden 2, out dim) :
2. super (NeuralNetw。rk, self ) init ()
3. self layerl - nn .Sequential ﹝
4. nn .Linear ( in dim, n hidden 1) , nn .ReLU (True))
5. self .layer2 a nn .Sequential (
6. nn .Linear (n hidden 1, n hidden 2) , nn ReLU ﹛ True) )

，。

1. self layer3 " nn. Sequential (nn Linear ﹝ n\_hidden 2 ut\_dim) )
2. def f。rward (self , x) .

9

1. x a self layerl (x)
2. x a self . layer2 ﹛ x)
3. X a self .layer3 ﹛ x)
4. return x

這 裡 只需 要 在每 層 網 路 的輸 出部分增加敢 動 函數 就可以 了 ，用 到 nn.Sequential （）， 這個函數是將網路的層組合到一起， 例如上面將 nn.Lin開r（） 和 nn.ReLU（） 組合到一起作為 self.layer 。注意最後一層輸出層不能增加敢動函數，因 為輸出的結果表示的是實際的得分。

3.8.3 增加批次標準化

最後增加一個加快收斂速度的方法一一批次標準化 。

* 1. class Batch Net ﹛nn Module) :
  2. def init (self , in dim, n hidden 1, n hidden 2’。ut d工m)
  3. super (Ba乞ch Net, self ) init （ ﹜
  4. self .layerl - nn Sequential (
  5. nn Linear ﹝ rn dim, n\_hidden 1) ,
  6. nn.BatchNormld (n hidden 11 , nn ReLU ﹛ True) )
  7. self .layer2 - nn.Sequential (
  8. nn .Lrnear (n hidden\_l, n hidden 2) ,

。

* 1. nn .BatchN rmld ﹝n hidden 2) , nn ReLU ﹛ True） ﹜
  2. self .layer3 - nn .Sequential ﹛ nn .Lrnear (n hidden\_2, out\_dim﹜ ﹜

11

1. def forward (self , x)
2. x - self .layerl ﹛xi
3. x - self.layer2 (xi
4. x - self layer3 ﹛x﹜
5. return x

同樣使用 no.Sequential （） 將 nn.BatchNormld （） 組合到網路層中，注意批標准 化一般放在全連接層的後面、非線性層 （ 殷勤函數） 的前面。

3.8.4 訓練網路

網路的定義特別簡單， 現在用 MNIST 資料集訓練網路並測試一下每種網路的 結果。

MNIST 資料集是一個手寫字型資料集，包含 0 到 9 這 10 個數字，其中有 55000 張訓練集， 10000 張測試集，5000 張驗證集，圖片大小是 28 ×泌 的灰階 圖，如圖 3.28 所示 。

圓圓圈圍！

圖 3.28 訓練圖序

3.8 多層全連接神經網路實現 MNIST 手寫數字分類 一

首先需要匯入一些要用的套件：

import t。rch

1 2 3 4 5 6 7

fr m torch import nn ptim

。 。

from t rch aut grad import Variable from t 主ch.utils.data imp。rt DataL。ader

fr m t rchvisi ,n imp rt datasets, transf rms imp。rt net

最下面的 import net 是之前定義網路的 Python 檔案。然後可以定義一些超參 數，如 batch size 、learning\_rate 還有 num epoches 等。

＃ 超參數 ﹝Hyperparameters ﹜

1 2 3 4

batch size - 64 lea主nrng\_rate - le-2

num\_ep。ches 互 20

接著需要進行資料前置處理， 就像之前介紹的，需要將資料標準化，這裡運 用到的函數是 torchvision.transforms’它提供了很多圖片前置處理的方法。這裡使 用兩個方法 第一個是 transforms.To Tensor（）， 第二個是 trans臼nns.Nonnali且（）。

transfonn.ToTensor （） 很好了解，就是將圖片轉換成 PyTorch 中處理的物件 Tensor，在轉化的過程中 PyTorch 自動將圖片標準化了，也就是說 Tensor 的範圍 是 0 ～ 1。接著我們使用 transforms.Normalize （）， 需要傳入兩個參數： 第一個參數

是平均值，第三個參數是方差，做的處理就是滅平均值，再除以方差。

data\_tf - transf 主ms.C mpose (

。 。

1

﹝ transf rms T Tensor ﹛） ，

。

2

transf rms .Normalise （ ﹝ 0 .5﹞ ﹝ 0 .5﹞ ） ﹞ ）

3

這裡 transforms.Compose （） 將各種前宣處理操作組合到一起， transforms.Nor malize（﹝0呵，﹝0.5﹞） 表示減去 5 再除以 0.5 ，這樣將圖片轉化到了 l ～ 1之間，

注意因為圖片是灰階圖，所以只有一個通道，如果是彩色的圖片，有三通道，那 麼用transforms.Normali ze（包 b, C﹞ ﹝d, e 可）來表示每個通道對應的平均值和方差。

然後讀取資料集。

一 第 3章

多層全連接神經網路

＃ 下載訓練集間IST 手寫數字訓練集

1 2 3 4 5 6 7 8

train dataset 呈 datasets MNIST ﹛

。 。 。。

r ta’／ data' , trainaTrue, transf rm-data\_tf , d wnl adaTrue)

test dataset a datasets 間IST (roata ’／ data trainaFalse, transf 目司data tf ﹜

train 1。ader " DataL。ader ﹛ train dataset, batch sizeabatch size, shuffleaTrue) test l。ade r a DataL。ader ( test dataset, batch sizeabatch size, shuffleaFalse﹜

透過 PyTorch 的內建函數 torchvision.datasets. 前置處理，前面介紹了如何定義自己的資料集，之後會用實際的實例說明。接著 使用 torch.utils.da旭DataLoader 建立一個資料反覆還算器， 傳入資料集和 batch\_ si血，透過 shuffie=True 來表示每次反覆還算資料的時候是否將資料打亂。

接著匯入網路，定義損失函數和最佳化方法。

m。del a net.simpleNet (28 • 28, 300, 100, 10 ) if torch .cuda .is available ﹝）

。

1 2 3

model a m del.cuda (I

。criten。n " 。nn .Cr。ssEntr。opyLoss (I

5 6

4

ptimizer a ptim SGD (m del par臼neters (I , lralearning\_ rate﹜

net.simpleNet 是定義的簡單三層網路， 裡面的參數是 28 X 詣，300, 100, 10 , 其中輸入的維度是 28 × 2日，因為輸入圖片大小是 28 × 訝，然後定義兩個隱藏層 分別是 300 和 100 ，最後輸出的結果必須是 10 ，因為這是一個分類問題， 一共有

0 ～9 這 10 個數字，所以是 10 分類。損失函數定義為分類問題中最常見的損失

函數交叉婿，使用隨機梯度下降來最佳化損失函數。

接著開始訓練網路， 流程基本和之前一致， 這裡就不再贅述。最後訓練完網 路之後需要測試網路， 透過下面的程式來測試。

m。del eval ﹛）

1 2 3 4 5 6

eval loss " O

eval ace a O

f 主 data in test l ader,

img, label a data

img a img v斗ew ﹛ img size ( O J , 1) if t rch , cuda is available (I ,

。

7

3.8 多層全連接神經網路實現 MNIST 手寫數字分類 －

1. img 三 Variable ﹛ img, V。latile豈True﹜ .cuda ﹛）
2. label Variable ( label, v。latileTrue) .cuda （ ﹜
3. else

。

1. 1mg variabel (img, v latileTrue﹞

。

1. label Variable ﹝ label, v latileTrue)
2. 。ut madel (img )
3. lass cnterian ﹝ ut, label﹜

。 。

1. eval l ss ＋旦 l ss.data ﹝ O﹞ ＊ label.size (O)

。 。

16 , pred t rch max I ut, 1)

1. num一correct (pred 巨 label) .sum （ ﹞
2. eval acc ta num correct .data ﹝ O﹞
3. print I ’Test L ss：﹛ .6f ﹜ Acc. ﹛ .6f ﹜ 『 主 rmat (
4. eval lass *I* ( len ( test dataset) I ,
5. val ace *I* (len ﹛ test dataset ) 1 1 1

這裡需要注意的一點就是 img=Variable(im耳，volatile=True）， 裡面的 vol atile= True 表示正向傳播時不會保留快取，因為對於測試集，不需要做反向傳播，所以 可以在正向傳播時釋放掉記憶體，節省記憶體空間。

最後看一下每種網路的測試結果， 如圖 3.29 、圓 3.30 、圖 3.31 所示。

國 3.29 、圖 3.30 和圖 3.31 依次是簡單三層網路、增加敢動層的網路和 增加了 批次標準化的網路，可以從中看出網路的準確率越來越高 ，這也證明了之前所介 紹的內容的正確性， 同時還可以進入網路中增加網路的泛化能力，如 Dropout 、正 規化等，還可以修改隱藏層的神經元個數， 增加隱藏層層敏等， 這裡留給讀者自 己做實驗去練習。



圖 3.29 簡單三層網路測試結果

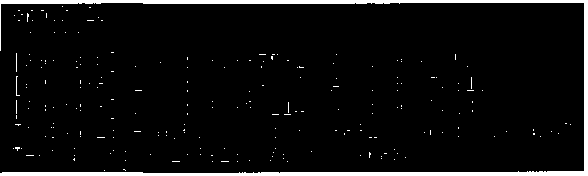


圖 3.30 增加歐動層的網路測試結果

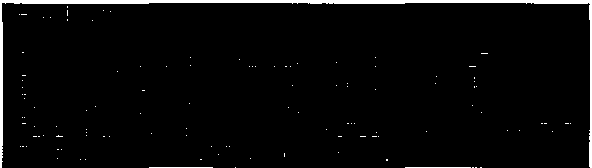
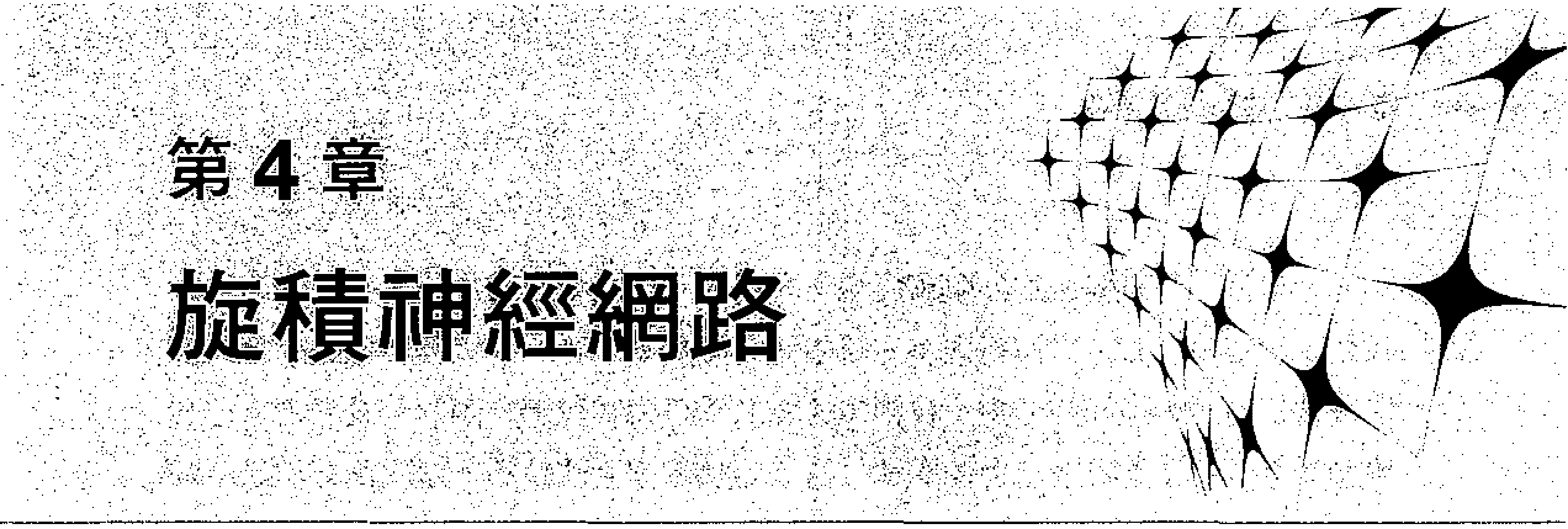


圖 3.31 增加類比標準化的網路測試結果

至此便介紹完簡單的多層全連接神經網路， 最後也用 PyTorch 實現了不同結 構的網路，透過在 MNIST 資料集上的訓練，比較了它們之間的差異。

下一章將正式進入時下最流行的深度學習領域， 從電腦視覺入手，引用特別 適合做影像處理的網路模型一一旋積神經網路。



影像分顯問題是電腦視覺中的核心問題， 雖然問題描述很簡單， 部具有很 廣龍的實用價值， 很多獨立的電腦視覺工作如目 標槍甜、分割等，都可以簡化 為影像分頭問題。旋種神鱷網路於 1998 年由 Yann Lecun 提出。2012 年，Alex Krizhevsky 憑藉它贏得了 ImageNet 挑戰賽，震驚世界，如今旋積神經網路已經成 為電腦視覺頡域最具影響力的一部 分。

這一章將從電腦祖覺的工作起輝開始引用旋積神經網路， 介招艦種神經網路 的原理和基礎，然接介紹 PyTorch 的旋積模組，接著介紹琨在應用最廣泛的錢個 雄積神經網路模型， 然接用雄積神經網路再次實現上一章的 MNIST 手寫數字分 頡’並介紹影像增強的挂巧，運用它在新的資料集 cifarlO 上進行測試，最佳介招 龍積神經網路的逆過程： 反雄積神經網路 ，

I 4.1 I 主要工作晨起源

人頭取得外界資訊，主要依靠祖覺、聽覺、觸覺、嗔覺和味覺等j萬覺器官， 其中 80% 的資訊取得都來自祖覺， 而且祖覺取得的竇訊也是最豐富、 最權雜的。 人的生理建置訣定了我們能掏看清楚並了解身邊的場景， 而要讓電腦看懂這個世 界卻是一件非常困難的事情， 即使在很多人看來現在的電腦搜捕已經足夠先進 了，但是要達到看懂並自主分析各種權雜資訊的程度， 還有很長的一段路要是， 這也是電腦祖覺這們學科要解缺的事情，

電腦祖覺的棋心工作之一是影像輯到， 我們總是在想能不能教會攝器東完 成影像識別這個工作。 人頭對於圖片的識別相當容易，然而機器卻面臨了很多問 題，如角度轉換、光蕭條件、背景干擾、物體變形等，正是由於這些問題的干擾 使得電腦在聽別圖片的時帳單確率蟬，是太恆 e

如何寫一個演算法來分類圖片呢 ？和排序不同，分類圖片並沒有那麼簡單， 我們不可能自己制定一個規則決定哪張圖 片屬於哪一種，所以需要透過學習演算 法讓機器自己知道如何分類。我們將給電腦提供每種類別的圖片，讓機器自己去 學習其中的特徵並形成一個演算法， 這就是機器學習的核心 。這些演算法是依賴 於資料集的，所以也稱為資料騷動的演算 法。

在旋積神經網路流行起來之 前，影像處理使用的都是一些傳統的方法， 例如 分析影像中的邊緣、紋理、線條、邊界等特徵，依據這些特徵再進行下一步的處 理，這樣的處理不僅效率特別低， 準確率也不高 。隨著電腦視覺的快速發展 ，如 今在某些影像集上機器的識別準確率已經超過了 人類，這一切都要歸功於旋積神 經網路。

下面就來介紹什麼是旋積神 經網路。

I 4.2 I 旋積神經網路的原理和結構

在介紹旋積神經網路之前 ，先提出三個觀點，正是這三個觀點使得旋積神經 網路能夠真正有作用 。

油 1. 局部性

對於一張圖片而言，需要檢測圖片中的特徵來決定圖片的類別，大部分的情 況下這些特徵都不是 由整張圖片決定的，而是由一些局部的區域決定的。

例如圖 4.1 中的鳥喙，該特徵只存在於圖片的局部中 。

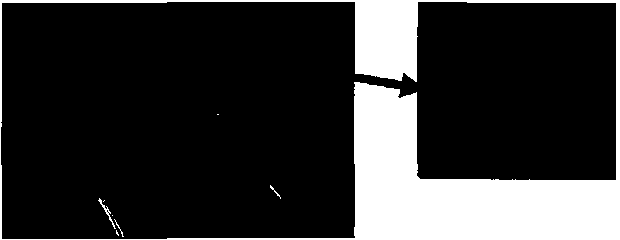
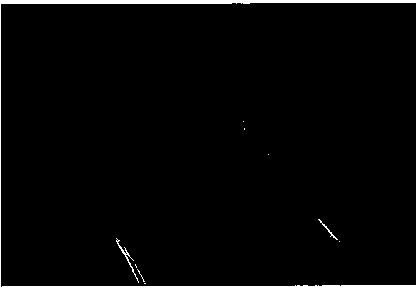


圖 4.1 圖片特徵只在局部

2. 相同性

對於不同的圖片 ，如果它們具有同樣的特徵’這些特徵會出現在圖片不同的 位置，也就是說可以用同樣的檢測模式去檢測不同圖片的相同特徵，只不過這些 特徵處於圖片中不同的位置，但是特徵檢測所做的操作幾乎一樣。

圖 4.2 中兩張圖片的鳥喙處於不間的位置， 但是可以用相同的檢測模式去檢 測。



丹 亨 、

主 垮＼、

﹒

！” ‘

今. ＇～

：

d詛  *.-:f.*呵..

”

；

圓 4.2 圖片特徵在不同的位置

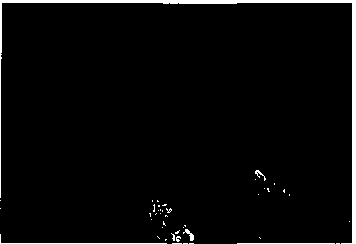
過 3. 不變性

對於一張大圖片，如果我們進行下取樣， 那麼圖片的性質基本保持不變。 圖 4.3 經過下取樣還是能夠看出來是一張鳥的圖片。

回 閉目團 第 4 章 旋積神經網路

bird

bird



圓圈，

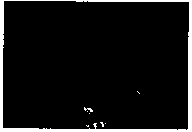


圖 4.3 下取樣

上面的三個性質分別對應著旋積神經網路中 的三種思想，接下來介紹網路的 層結構。

第 3 章介紹了一般的全連接神經網路，旋積神經網路和它是相似的，也是由 一些神經元組成的， 如圖 4.4 所示。這些神經元中具有需要學習的參數， 透過網 路輸入，最後輸出結果，透過損失函數來最佳化網路中 的參數。旋積神經網路與 其不同之處在於網絡的層結構是不同的。 園 4.4 是全連接神經網路， 由一系列隱 藏層組成，每個隱蔽層由許多個神經元組成， 其中每個神經元都和前一層的所有 神經元相連結，但是每一層中的神經元是相互獨立的 。

＼﹜／ j

input layer

hidden laye「 1 hidden layer 2

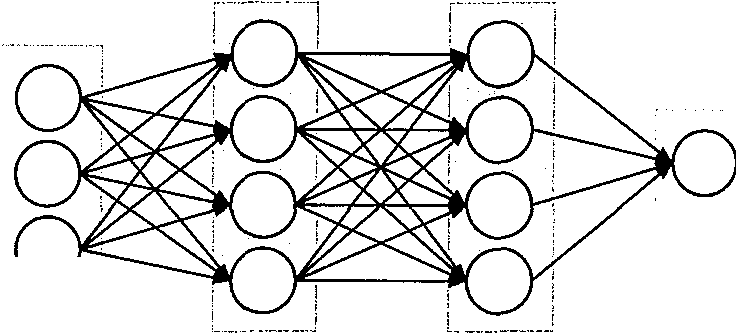


圖 4.4 全連接神經網路

output layer

這樣的神經網路在處理圖片上 存在什 麼問題呢 ？例如在 MNIST 資料

集上， 圖片大小是 28 × 恕，那麼第一個隱藏層的單一神經元的加權數目就

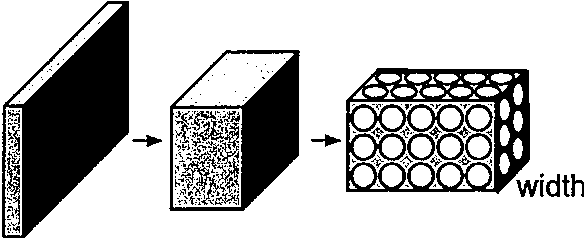
是 28 X 28 = 784 ，這似乎還不是特別大， 但這只是 一張小園片，且是 灰 階 圈。對於 一張較大的圖片而言，例 如 200 X 200 × 3 ，就會 導致加權數目是 200 × 200 X 3 = 120000，如果設定幾個隱藏層中的神經元數目 ，就會導致參數增 加特別快。其實這樣的圖片在現實中並不算大圖片 ，所以全連接神經網路對於處

理影像並不是一個好的選擇。

圓 4.5 所示的是旋積神經網路的處理過程， 不同於一般的全連接神經網路， 旋積神經網路是一個 3D 容量的神經元，也就是說神經元是以三個維度來排列 的：寬度、高度和深度。例如輸入的圖片是 32 × 32 = 3 ，那麼這張圖片的寬度就 是 32，高度也是 32，深度是 3 。後面會詳細地介紹旋積神經網路是如 何計算的， 以及為什麼它被這樣設計並取得如 此好的效果。

旋積神經網路中的主要層結構有三個：旋積層、池化層和全連接屑， 透過堆 疊這些層結構形成了一個完整的旋積神經網路結構。 旋積神經網路將原始圖片轉 化成最後的類別得分 ，其中一些層包含參數， 一些層沒有包含參數， 例如旋積層 和全連接層擁有參數， 而敢動層和池化層不含參數。 這些參數透過梯度下降法來 更新，最後使得模型盡可能正確地識別出圖片類別。

depth



height

＋芒三三軍

圖 4.5 旋積神經網路的處理過程 接下來實際介紹每一種層的連接方式和它們的超參數。

4.2.1 旋積層

旋積層是旋積神經網路的核心， 大多數計算都是在旋積層中進行的。

－第 4 章 旋積神經網路

油 1. 概述

首先介紹旋積神經網路的參數。 這些參數是由一些可學習的濾波器集合組成 的，每個濾波器在空間上 （寬度和高度） 都比較小，但是深度和輸入資料的深度 保持一致。舉例來說，旋積神經網路的第一層旋積一個典型的濾波器 的尺寸可以 是 5 X 5 X 3 （ 寬和高都是 5 ）， 或是 3 X 3 X 3 （ 寬和高都是 3 ）， 這裡的寬度和高 度可以任意定義，但是深度必須是 3 ，因為深度要和輸入一致，而輸入的圖片是 3 通道的。在正向傳播的時候，讓每個濾波器都在輸入資料的寬度和 高度上滑動

（ 旋積 ）， 然後計算整個濾波器和輸入資料任意一處的內積。

當濾波器沿著輸入資料的寬度和高度滑動時， 會產生一個二維的殷動圈， 敢 動圖上的每個空間位置表示了原圖片對於該濾波器的反應。直觀來看，網路會讓 濾波器學習到當它看到某些類型的視覺特徵的時候就殷勤 ，實際的視覺特徵可以 是邊界、顏色、輪廓、甚至可以是網路更高層上的蜂巢狀或車輪狀 圖案。

在每個旋積層上，會有一整個集合的濾波器， 例如 20 個，這樣就會形成 20 張二維的、不同的敢動間，將這些殷勤圖在深度方向上層疊起來就形成了旋積層 的輸出。

如果用大腦和生物神經元做比喻，那麼輸出的 3D 資料中的每個資料都可以 看成是神經元的輸出， 而該神經元只是觀察輸入資料中的一種特徵， 並且和空間 上左右兩邊的所有神經元共用參數 （ 因為這些輸出都是使用同一個濾波器獲得的 結果）。下面介紹旋積神經網路中的神經元連接，它們在空間中的排列，以及它們 參數共用的模式。

油 2. 局部連接 在處理影像這樣高維度輸入的時候，讓每個神經元都與它那一層中 的所有神

經元進行全連接是不現實的。相反，讓每個神經元只與輸入資料的局部區域連接

是可行的，為什麼可以這樣做呢 ？其實這是因為圖片特徵的局部性， 所以只需要 透過局部就能分析出對應的特徵。

與神經元連接的空間 大小叫做神經元的厲受野 （ receptive 益eld ）， 它的大小是 一個人為設定的超書數， 這其實就是爐控器的寬和高 。在深度方向上，其大小總 是和輸入的深度相等。 最佳強調一下，對待空間推度 （寬和高） 和深度雖度是不 同的，連接在空間上是局部的，但是在深度上輯是和 輸入的資料深度保持一致 e

圖4.6 具體地展示了風受野在空間和深度上的大小，左邊表示輸入的賣料， 中 間是屬安野，右邊每個小小數點表示一個神經元 。下面舉一個實際的實例來說明一 下。側如輸入的資料尺寸為 32 X 32 × 3 ，如果聽受野 （ 爐控器尺寸） 是 sxs ，旋

積層中每個神經元會有輸入資料中 5 × 5 X 3 區域的加權 ，一共 5 ×5 × 3 =75 個加

權，這裡再次強調厲受野深度的大小必績是 卜和輸入資料保持一致。*-wa*如輸入資 料體尺寸是 16 X 16 X 20，厲受野是 3 × 3 ，麓積層中每個神經元和輸入資料體之間 說有 3 × 3 × 20 = 180 個連接，這裡的深度必讀是 20 ，和輸入資料一致。

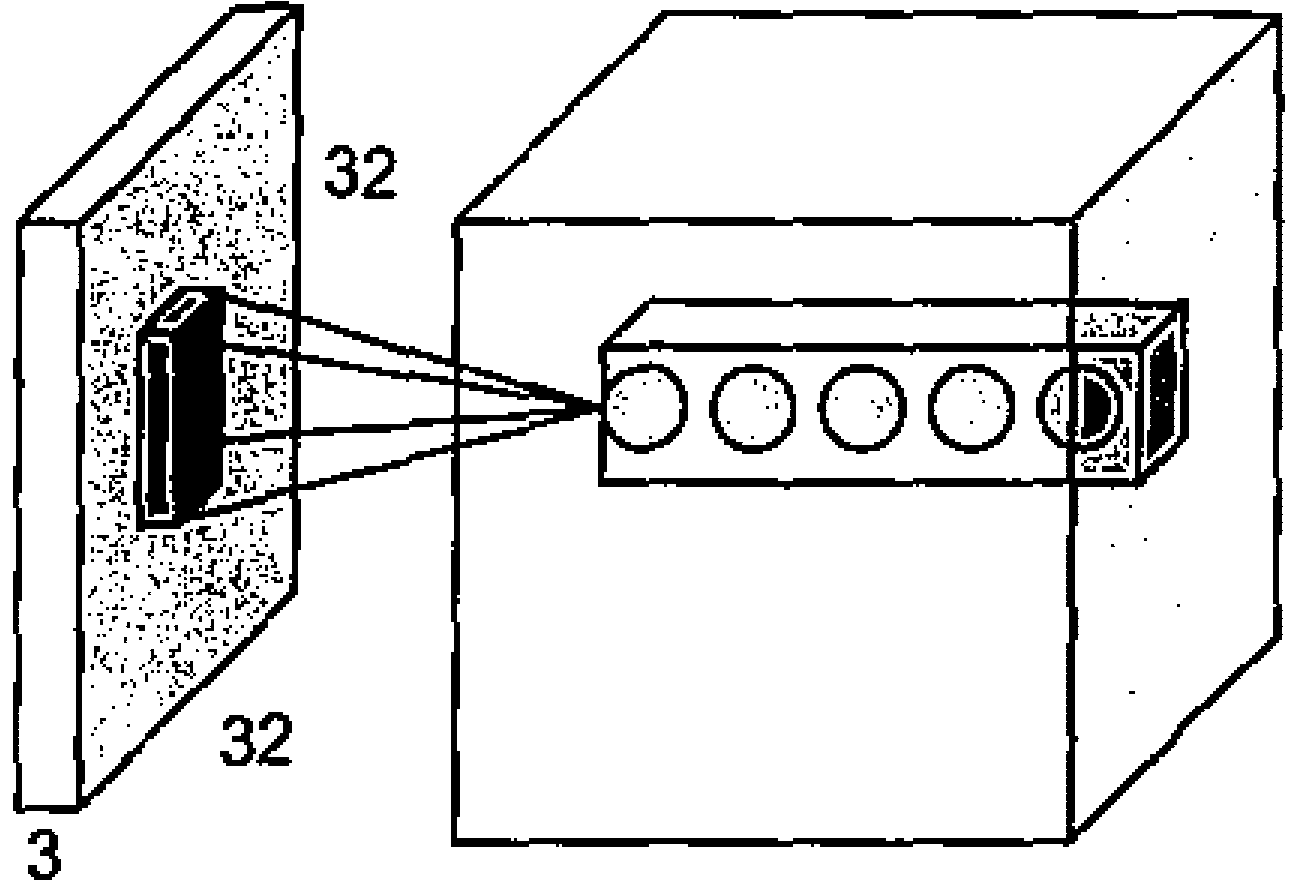


圖 4.6 厲曼野

凶 3. 空間排列

前面介紹了每個神經元只需要與輸入資料的局 部區蟻相連接，但是沒有介紹 旋積層中神鱷元的數量和它們的排列方式、 輸出深度、滑動步進值，以及邊界壤 充控制著雄積盾的空間排列。

首先，旋積層的輸出揮度是一個超喜數， 它與使用的權控器數量一致 ，每種 爐控器所做的就是在輸入資料中 尋找一種特徵。比如說輸λ一張原始圖片，旋積 層輸出的深度是 20 ，這說明有 20 個權控器對資料進行處理， 每種爐控器尋找一 種特館進行敢動。

．圓圓圓 第 4 章 旋積神經網路

其次，在滑動濾波器的時候， 必須指定步進值。例如步進值為 1，說明濾波 器每次移動 l 個像素點。當步進值為 2 的時候，濾波器會滑動 2 個像素點。滑動 的操作會使得輸出的資料在空間上變得更小 ﹒

最後介紹邊界填充，可以將輸入資料用 在邊界進行填充，這裡將 0 填充的 尺寸作為一個超參數， 有一個好處就是， 可以控制輸出資料在空間上的尺寸 ，最 常用來保障輸入和輸出在空間上尺寸一致。

輸出的尺寸到底是多少呢 ？其實可以用一個公式來計算 ，就是

*W-F+2P*

一一了一一＋I'

其中 W表示輸入的資料大小，F表示旋積層中神經元的成受野尺寸， S表示步進 值，p表示邊界填充 0 的數量 。例如輸入是 7 × 7 ，濾波器是 3 X 3 ，步進值是 I,

填充的數量是 0，那麼根據公式，就亂包獲得，7 3+2 x o

一一－－－－－，一一＋

1-5 ，即輸出的空間大

小是 5 × 5 ，如果步進值是 2

，7 3+2 X 0 ！＝3 ，輸出的空間大小就是 3 X 3 。

2

一一一一一＋

可以用圓 4.7 所示的這個一維的實例來質際說明。

右上角表示神經網路的加權， 其中輸入資料的大小為 5 ，感受野的大小為 3 ﹔

左邊表示滑動步進值為 1，且填充也為 1，右邊表示滑動步進值為 2 ，填充為 1。

目

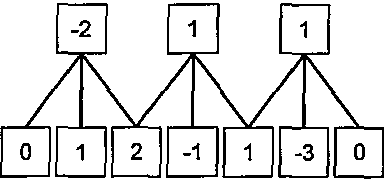
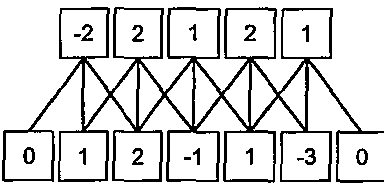


圖 4.7 維空間

油 4. 零填禿的使用

從上面的實例中 ，我們看到輸入的維度是 5，輸出的維度也是 5 。之所以如 此，是因為成受野是 3，並且使用了 l 的零填充。如果不使用零填充， 那麼輸出

一一一？一一＋

資料的維度也

5 3+ 2 X 0 ！＝3 。一般來說，當步進值 S= l 峙，零填充的值為

F I .

*P= 2*一 這樣就能夠保障輸入的資料和輸出的資料具有相 同的空間尺寸。

油 5. 步進值的限制

透過上面的公式我們知道步進值的選擇是有所限制的， 舉例來說，當輸入尺 寸 *w*巨 *10* 的時候 ，如果不使用零填充， 即*P=O* ，濾波器尺寸 *F=3* ，這樣步進值

S= 2 就行

10-3+0

一 一＋

＝4.5，結果不是一個整數， 這就說明神經元不能整齊對稱

地滑過輸入資料體 ，這樣的超參數設定是無效的，使用 PyTorch 的時候就會顯示 出錯，可以使用零填充讓設定變得合理。 在後面旋積神經網路的結構設計中 ，需 要合理地設計網路的尺寸， 使得所有維度都能夠正常執行， 這件事並沒有看上去 那麼容易。

油 6. 參數共用

在旋積層使用參數共用可以有效地減少參數的個數， 這樣之所以能夠行得 遇，是因為之前介紹的特徵的相同性， 也就是說相同的濾波器能夠檢測出不同位 置的相同特徵。

比如說一個旋積層的輸出是 20 × 20 X 32 ，那麼其中神經元的個數就是 20 X 20 X 32 = 12800，如果視窗大小是 3 X 3 ，而輸入的資料體深度是 10 ，那麼每 個神經元就有 3 X 3 X !0 = 900 個參數，這樣合起來就有 12800 × 900 = 11520000 個 參數，單單一層旋積就有這麼多參數， 這樣還算速度顯然是特別慢的。

根據之前介紹的，一個濾波器檢測出一個空閑位置 （吭 *Y* ，） 處的特徵，那麼也 能夠有效檢測出 （巧 *y* ） 位置的特徵，所以就可以用相同的濾波器來檢測相 同的特 徵’基於這個假設， 我們就能夠有效減少參數個數。 例如上面這個實例， 一共有

32 個濾波器，這使得輸出體的厚度是 32 ，每個濾波器的參數為 3 × 3 X !0= 900 •

總共的參數就有 32 X 900 =28800 個，相當大減少了參數的個毀 。

由參數共用我們知道輸出體資料在深度切片上所有的加權都使用同一個加權 向量，那麼旋積層在向前傳播的過程中 ，每個深度切片都可以看成是神經元的加 權對輸入資料體做旋積， 這也就是為什麼要把這些 3D 的加權集合稱為濾波器， 或旋積核心 。

需要注意的是，參數共用之所以能夠有效， 是因為一個特徵在不同位置的 表現是相同的， 例如一個濾波器檢測到了水平邊界這個特徵， 那麼這個特徵具有 平移不變性，所以在其他位置也能夠檢測出來。 但是有時候這樣的假設可能是沒 有意義的，特別是當旋積神經網路的輸入影像呈現的是一些明確的中 心結構的時 候，希望在圖片的不同位置學習到不同的特徵。 一個實際的實例就是人臉識別， 人臉一般位於圖片的中心，我們希望不同的特徵能夠在不同 的位置被學習到，例 如眼睛特徵或頭髮特徵， 正是由於這些特徵在不同的地方， 才能夠對人臉進行識 別。

凶 7. 歸納

最後歸納一下旋積層的一些性質。

( I ） 輸入資料體的尺寸是 *W1 X H1* × *D1* 。

( 2 ) 4 個超參數：濾波器數量 K ，濾波器空間尺寸 F’滑動步進值 S 零填充的數

量p。

*W, -F* 斗。*P fl -F* 斗。p

× ， *w*，巨

( 3 ） 輸出資料體的尺寸為 丹 X H2 *D* ，其中 」寸一一＋ I，只＝斗了一

+ I *• D2= K* 。

(4 ） 由於參數共用，每個濾波器包含的加權數目 為 F×F×*D1* ，旋積層一共有

*F X F X D1 X K* 個加權和 K個偏置。

( 5 ） 在輸出體資料中，第 d個深度切片（ 空間尺寸是 *w* × *F* ） ，用第 d個濾波器 和輸入資料進行有效旋積運算的結果， 再加上第 d個偏置。

對於旋積神經網路的一些超參數，常見的設定是 *F =3 • S = l • P = I* ，同時這 些超參數也有一些約定俗成的價1研日經驗，在之後的章節會介紹。

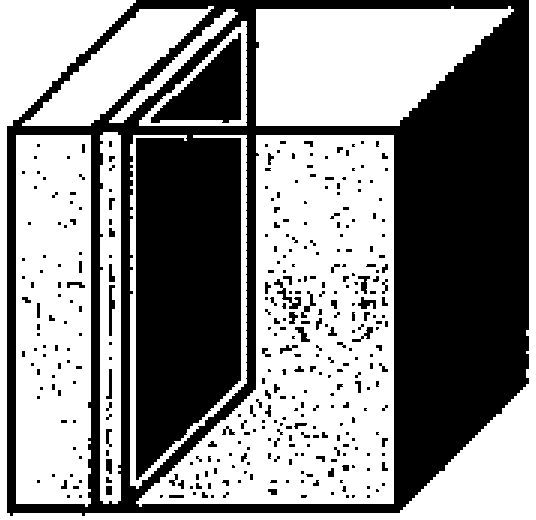
4.2.2 池化層

上一部分介紹完旋積神經網路中 最核心的內容一一旋積層， 下面來介紹一下 第二種層結構一一池化層。

通常會在聽積層之間週期性插入一個地化層， 其作用是逐漸降低資料體的空 間尺寸，這樣就能掏誡少網路中學數的數量 ，誠少運算資 耗費，同時也能興有 兢地控制過揖台 。

下面先來介紹到底什麼是油化屑。油化層干01.在積層一樣也有一個空間觀窗， 通常探用的是取這些觀窗中 的最大值作為輸出結果， 然後不斷滑動祖窗，對輸入 資料體每一個深度切片單獨處理，揖少它的空間尺寸， 如圖4.8 所示 。

224x224x64

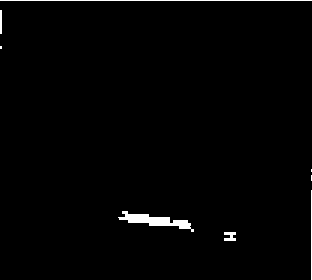


…個 ！即

pool

－－－－－－－ ．

224



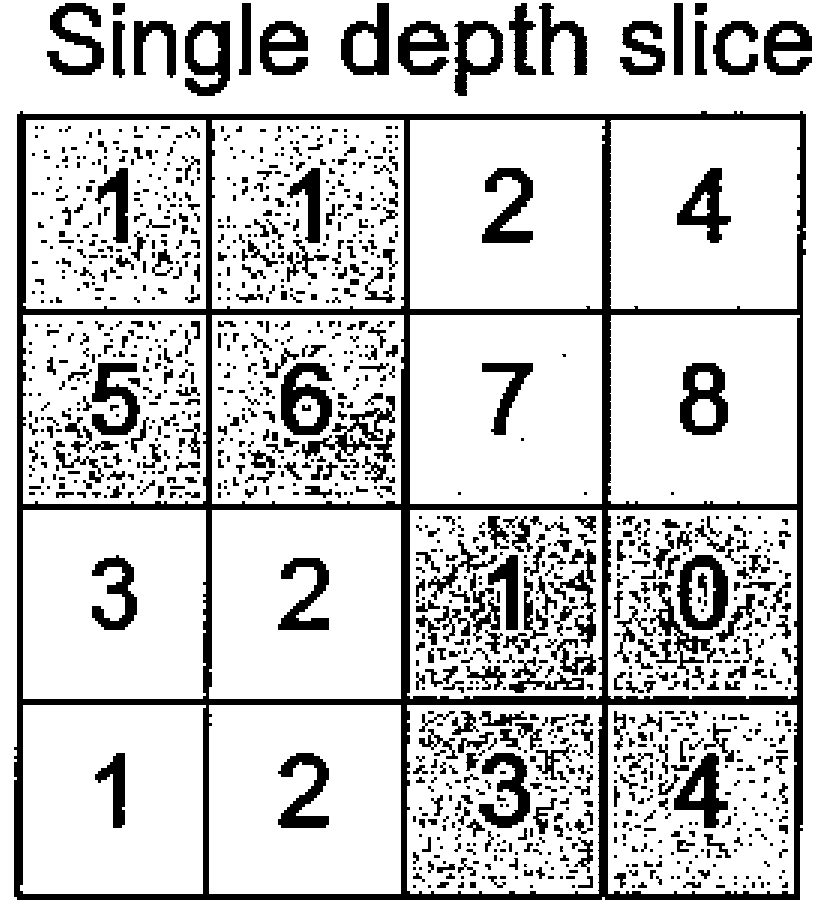
224

downsampling

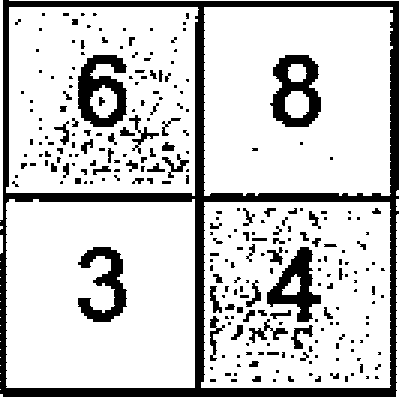
112

圖 4.8 油化層處理碩果

從圖 4.8 能鉤看出油化層能夠有強 降低資料體空間的大小， 圖4.9 具體地說明 了視窗大小是 2，滑動步進值是 2 的最大值油化是如何計算的 ：每次都從 2 ×2 的 祖窗中選擇最大的數值， 同時每次滑動 2 個步進值進入新的祖窗。



X max pool with 2x2 filte時



and stride 2

y

圖 4.9 油化層計算

池化層之所以有效， 是因為之前介紹的 圖片特徵具有不變性， 也就是透過下 取樣不會遺失圖片擁有的特徵， 自於這種特性，我們可以將圖片縮小再進行旋積 處理，這樣能夠大幅降低旋積運算的時間。

最常用的池化層形式是尺寸為 2 × 2 的視窗 ，滑動步進值為 2，對影像進行下

取樣，將其中 75% 的敢動資訊都丟掉，選擇其中最大的保留下來， 這其實是因為 我們希望能夠更加敢動裡面的數值大的特徵，去除一些雜訊資訊。

池化層有一些和旋積層類似的性質。

C I ） 輸入資料體的尺寸是 *WI* × *H1* × *D1* 。

( 2 ） 有兩個需要設定的超參數，空間大小 F和滑動步進值 S o

*W, F* ff *Ti'*

( 3 ） 輸出資料體的尺寸是 你做丸，其中 院＝守一＋ I *,H2* ＝」了＋ *1 ,D2=D1* 。

( 4 ） 對輸入進行固定函數的計算，沒有參數引用。

( 5 ） 池化層中很少引用零填充。

在實際中 ，有兩種方式：一種是 *F = 3 ' S = 2* ，這種池化有重壘，另外更常用 的一種是 *F =2 • S = 2*，一般來說應該謹慎使用比較大的池化視窗，以免對網路有 破壞性。

除了最大值池化之外，還有一些其他的池化函 數，例如平均池化，或口范數 池化。在實際中證明，在旋積層之間引用最大池化的效果是最好的， 而平均池化 一般放在旋積神經網路的最後一層。

4.2.3 全連接層

全連接層和之前介紹的一般的神經網路的結構是一樣的 ，每個神經元與前一 層所有的神經元全部 連接，而旋積神經網路只和輸入資料中的局部區域連接，並 且輸出的神經元每個深度切片共用參數。

一般經過了一系列的旋積層和池化層之後， 分析出圖片的特徵圖，比如說特 徵圓的大小是 3× 3 X 512 ，這個時候，將特徵圍中的所有神經元變成全連接層的

樣子，直觀上也就是將一個 30 的立方體重新排列，變成一個全連接層， 裡面有

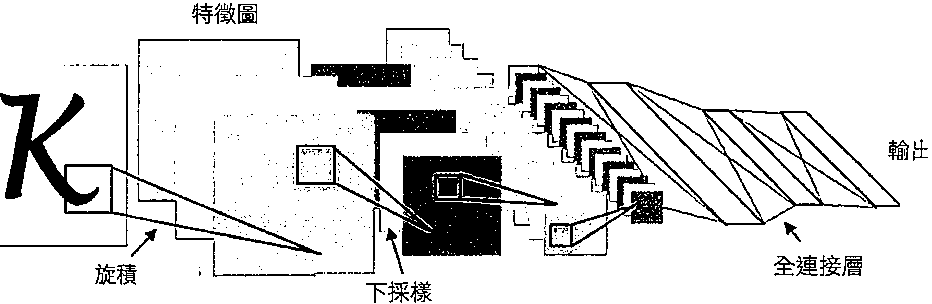
3× 3 × 512-4608 個神經元，再經過幾個隱藏層，最後輸出結果。 在這個過程 中為了防止過擬合會引用 Dropout o 最近的研究表明， 在進入全連

接層之前，使用全域平均池化能夠有效地降低 過擬合。

4.2.4 旋積神經網路的基本形式

旋積神經網路中通常是由上面介紹的三種層結構所組成 ，上一章還介紹過引 用激活函數增加模型的非線性，所以旋積神經網路最常見的形式就是將一些旋積 層和 ReLU 層放在一起，有可能在 ReLU 層前面加上批次標準化屑， 隨後緊接著 池化層，再不斷重複，直到影像在空間上被縮小到一個足夠小的尺寸， 然後將特 徵圍展闕，連接幾層全連接層，最後輸出結果，例如分類評分等。

圖 4.10 就是一種旋積神經網路的基本形式。



輸入

圖 4.10 旋積神經網路的基本形式

油 1. 小濾波器的有奴性

一般而言 ，幾個小濾波器旋積層的組合比一個大濾波器旋積層要好，例如層 層堆疊了 3 個 3 X 3 的旋積層，中問含有非線性敢動層，在這種排列下面，第一個 旋積層中每個神經元對輸入資料的感受野是 3× 3，第二層旋積層對第一層旋積層 的感受野也是 3 X 3 ，這樣對於輸入資料的戚受野就是 5 X 5 ，岡樣，第三層旋積層 上對第二層旋積層的感受野是 3×3，這樣第三層旋積層對於第一層輸入資料的成 受野就是 7 x 7 •

假設這裡不使用 3 個 3 × 3 的感受野，直接單獨使用一個 7× 7 大小的旋積 層，那麼所有神經元的成受野也是 7× 7，但是這樣會有一些缺點。多個旋積層首 先與非線性敢動層交替的結構， 比單一旋積層的結構更能分析出深層的特徵﹔ 其 次，假設輸入資料體的深度是 C ，輸出體的深度也是 c，那麼單獨的 7 X 7 的旋積 層會有 7× 7 X ex C=49 X C' 的參數個數，而使用 3 個 3 ×3 的旋積層的組合，僅

含有 3 ×（3 X 3 X C× C)=27 × c 的參數。宜觀來說，選擇小濾波器的旋積組合能 夠對輸入資料表達出更有力的特徵， 同時使用參數也更少。 唯一的不足是反向傳 播更新參數的時候，中間的旋積層可能會佔用更多的記憶體。

油 2. 網路的尺寸

對於旋積神經網路的尺寸設計， 沒有嚴格的教學證明 ，這是根據經驗制定出 來的規則。

( I ） 輸入層：一般而言，輸入層的大小應該能夠被 2 整除很多次，常用的數字包 含 32 • 64 ’96 和 224 。

(2 ） 旋積層：旋積層應該盡可能使用小尺寸的濾波器， 例如 3 X 3 或 5 × 5 ，滑動 步進值取 l。還有一點就是需要對輸入資料體進行零填充， 這樣可以有效地 保障旋積層不會改變輸入資料體的空間尺寸。 如果必須要使用更大的濾波器 尺寸，例如 7 X 7，通常用在第一個面對原始影像的旋積層上。

( 3 ） 池化層：池化層負責對輸入的資料空間維度進行下取樣， 常用的設定使用

2 X 2 的感受野做最大值池化， 滑動步進值取 2 。另外一個不常用的設定是使

用 3 × 3 的戚受野，步進值設定為 2 。一般而言池化層的感受野大小很少超過 3 ，因為這樣會使得池化過程 過於激烈，造成資訊的遺失，這通常會造成演算 法的效能變差。

( 4 ） 零填充：零填充的使用可以讓旋積層的輸入和輸出在空間上的維度保持一 致 ，除此之外，如果不使用零填充，那麼資料體的尺寸就會略微減少，在不 斷進行旋積的過程中，影像的邊緣資訊會過快地損失掉。

上面介紹了旋積神經網路中最重要的三種層結構： 旋積層、池化層和全連接 層，下面會介紹每一種層結構在 PyTorch 中是如何實現的。

I 4.3 I PyTorch 旋積模組

PyTorch 作為一個深度學習函數庫， 旋積神經網路是其中一個最為基礎的模 組，旋積神經網路中所有的層結構都可以透過 nn 這個套件呼叫，下面實際介紹如 何呼叫每種層結構，以及每個函數中的參數。

4.3.1 旋積層

nn.Conv2d（） 就是 PyTorch 中的旋積模組了 ，裡面常用的參數有 5 個，分別是 in\_chan n訟，out\_channels, kernel\_size, stride, padding 除此之外還有參數 dilati帥， groups, bias 。下面來解釋每個參數的含義。

in channels 對應的是輸入資料體的深度﹔ out channels 表示輸出資料髓的深 度﹔kernel size 表示濾波器 （ 旋積核心） 的大小，可以使用一個數字來表示高和 寬相同的旋積核心，例如 kernel size=3 ，也可以使用不同的數字來表示高和寬不 同的旋積核心 ，例如 kernel\_size=(3, 2) ; stride 表示滑動的步進值﹔ padding=O 表 示四周不進行零填充， 而 padding＝］ 表示四周進行 l 個像素點的零填充﹔bias 是 一個布林值，預設 bias=True，表示使用偏置﹔groups 表示輸出資料體深度上和 輸入資料體深度上的聯繫 ’預設 groups＝］，也就是所有的輸出和輸入都是相連結 的，如果 groups弓，這表示輸入的深度被分割成兩份， 輸出的深度也被分割成兩 份，它們之間分別對應起來，

所以要求輸出和輸入都必須要能被 groups 整除，dilation 表示旋積對於輸入資 料體的空間間隔，預設 dilation＝］，可以直觀地用圖 4.11 表示。

# 展可

圓 4.11 資料體的空間間隔

4 15

4.3.2 池化層

nn.M阻Pool2d （） 表示網路中的最大值池化，其中的參數有 kemel\_size 、stride 、

padding 、dilation 、return\_indices 、ceil mode ，下面解釋一下它們各自的含義。

* kemel\_size, stri缸，padding, dilation 之前旋積層已經介紹過了，是相同的含義﹔
* return indices 表示是否傳回最大值所處的案引，預設 return indices=False ﹔

• ceil mode 表示使用一些方格代替層結構，預設 ceil\_mode=False，一般都不會 設定這些參數。

• nn.AvgPool2d（） 表示平均值池化， 裡面的參數和 nn.MaxPool2d （） 類似 ，但多 一個參數 count incl ude pad ，這個參數表示計算平均值的時候是否包含零填 充，預設 count include pad=True 0

一般使用較多的就是 m.MaxPool2d（） 和 mAvgPool2d （）， 另外 PyTorch 還提 供了一些別的池化層，如 nn.LPPool2d （）、 nn.AdaptiveMaxPool2d （） 等，這些運用 較少，有興趣的讀者可以去檢視官方文件 。

全連接層和敢動函數上一章已經介紹過， 接下來建置一個簡單的多層旋積神 經網路。

1. class SimpleCNN ﹝ nn M。dule)
2. def init (self ) ,
3. super (SimpleC＜咽 self ) . init （ ﹞ ＃ b, 3, 32, 32
4. layerl - nn .Sequential ()

。 。

1. layer1.add m 。dule ( 'c nvl ，nn c nv2d (3, 32, 3, 1, padding-11 I

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | # b, 32, 32, 32 |  |  |
| 7 | layerl.add\_module ( ' relul， | nn .ReLU (True） ﹜ |
| 8 | layer1.add m aule ﹛’p。。11’， | nn .MaxP 。12d ﹝ 2, 2 ﹜ I | # b, 32, 16, 16 |
| 9 | self layerl - layerl |  |  |
| 10 |  |  |  |
| 11 | layer2 - nn Sequential ﹛﹜ |  |  |
| 12 | layer2. add m。dule （ c。nv2 ， | nn.C nv2d (32, 64 , 3, | 1, paddrng- 1) ) |
| 13 | # b, 64 , 16, 16 |  |  |
| 14 | layer2. add module ﹛ ＇relu2 ， | nn .ReLU ﹛ True) ) |  |
| 15 | layer2.add m。dule （’po 12 ， | nn.MaxP 。12d (2, 21 I | # b, 64 , a, a |
| 16 | self .layer2 - layer2 |  |  |

4.3 PyTorch 位積模組 －

17

1. layer3 - nn Sequential ﹛﹜
2. layer3. add\_module ﹛’conv3 ，nn .C nv2d ﹛ 64 , 128, 3, 1, padding- 1) )

20 tb, 128, 8, 8

21 layer3 add\_modole ﹛ ＇relu3 ，nn .ReLU (True) I

22 layer3.add\_module ( 'p 13 ，nn .MaxP。12d ﹛ 2, 2) ) #b, 128，也，4

23 self .layer3 - laye主3 24

1. layer4 - nn. Sequen七ial (I
2. layer4 .add m 。αule （’fcl ' , nn .Linear (2048, 512) )
3. layer4 .add\_mcdule （’fc relul’，nn .ReLU ﹛ True) I
4. layer4 .add\_m dule ﹝’fc2 ，nn Linear (512, 64 ) )
5. layer4 .add module ﹝’fc\_relu2 ' , nn.ReLU ﹛ True) )
6. layer4 .add module ﹛ ＇fc3 ，nn .Linear ( 64, 10) )
7. self .layer4 - layer4 32

。

1. def f rward (self , x) ,

。

1. C nvl - self .layerl (x)

。 。

1. C nv2 - self .layer2 (c nvl)
2. C。nv3 - self layer3 ﹛ conv2)
3. fc input = c。nv3.view (conv3 size (O﹜ ，1)
4. fc一 ut = self .layer4 ( fc\_inp"t﹜
5. return fc 。ut
6. m。del = SimpleCNN ﹝）

在上面的定義中 ，我們將旋積層、敢動層和池化層組合在一起組成了一個層 結構，定義了 3 個這樣的層結構， 最後定義了全連接層， 輸出 10 。強烈建議在建 立旋積層和池化層時透過參數計算一下輸出的資料體大小， 然後在程式旁邊寫出 詮釋，這樣在定義很複雜的網路結構時就不容易出錯。

同時可以在 forward 中re阻m 增加中間層的輸出結果，這樣能很方便地獲得網 路的中崗層輸出。

還可以透過 print(model） 顯示網路中定義了哪些層結構。

從圖 4.12 中可以看到這些層結構正如之前定義的一樣，括號裡面表示它的名 字。下面會介紹如何分析網路中指定的層結構、參數，以及如何對參數進行自訂 的初始化。

－－－－－ 第 4 章 能積神經網路

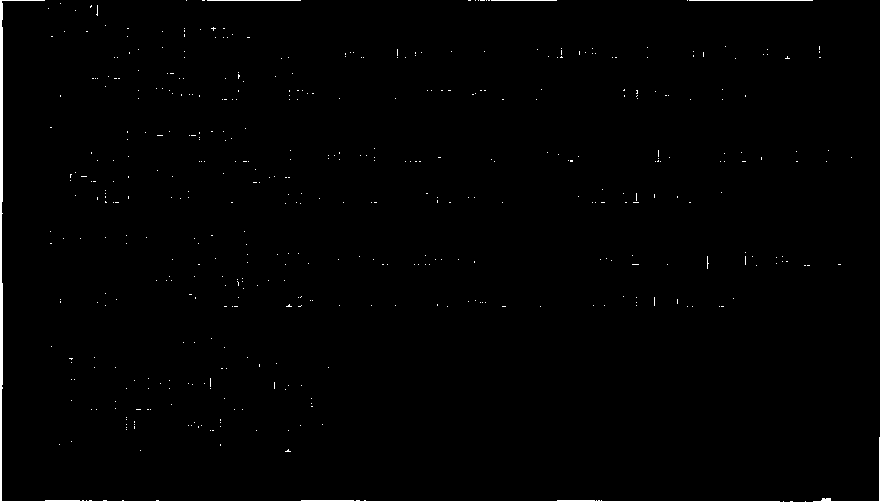


圖 4.12 層結構

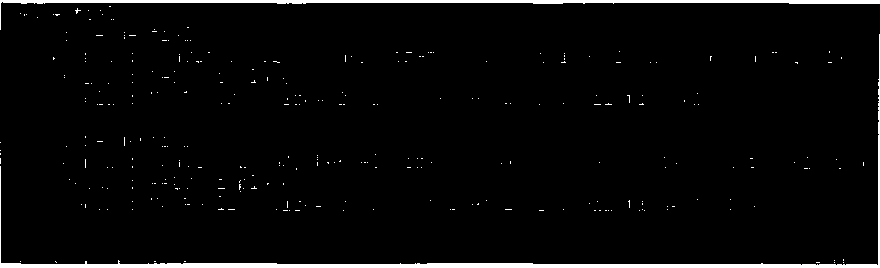
4.3.3 分析層結構

對於一個指定的模型，如果不想要模型中所有的層結構，只希望能夠分析網 路中的某一層或幾層，應該如何來賞現呢 ？

首先看看 nn.Module 的幾個重要屬性。 第一個是 children（）， 這個會傳回 下一級模組的反覆運算器， 例如上面這個模型， 它只會傳困在 self.layer!, self. layer2, self.layer3，以及 self.layer4 上的反覆運算器，不會傳回它們內部的東西﹔ modules（） 會傳回模型中所有模組的反覆還算器， 這樣就有了一個好處，即它能 夠存取到最內層，例如 self.layer! .convl 這個模組，還有一個與它們相對應的是 named children（） 屬性以及 named\_modules（）， 這兩個不僅會傳回模組的反覆運算 器，還會傳回網路層的名字。

下面來分析網路 中我們需要的層，如果希望能夠分析出前面兩層，那麼可以 透過下面的辦法來實現，獲得如圖 4.13 所示的結果。

1 new π，odel - nn Sequential ﹝＊list (m。del.children ﹝） ） ﹝ 2﹞ ）



圓 4.13 分析層的結果

如果希望分析出模型 中所有的旋積層，可以像下面這樣操作：

1. f。r layer in model named\_modules ﹛），
2. if isinstance ﹛ layer ﹝ l﹞，nn .C。nv2d﹜
3. C。nv\_mcdel add\_mcdule (layer ﹝ O﹞，layer ﹝ 1﹞ ）

使用 isinstance 可以判斷這個模組是不是所需要的類型實例， 這樣就分析出了 所有的旋積模組，獲得如圓 4.14 所示的結果。

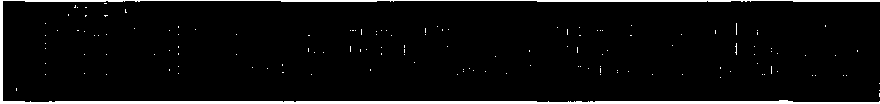


圖 4.14 分析所有旋積模組

4”3.4 如何分析參數及 自訂初始化

有時候分析出的層結構並不夠， 還需要對裡面的參數進行初始化， 那麼如何 分析出網路的參數並初始化呢 ？

首先 m.Module 裡面有兩個特別重要的關於參數的屬性， 分別是 named p盯amet ers（） 和 parameters（）。 named\_para meters（） 是列出網路層的名字和參數的 反覆運算器，parameter s（） 會列出一個網路的全部參數的反覆運算器 。

1. f。r param in m。del .named\_par卸neters ﹛）
2. print ﹝param ﹝ O﹞ ）

可以獲得每一層參數的名字， 如圖 4.15 所示。



圖 4.15 獲得每 層參數的名字

如何對加權做初始化呢？非常簡單，因為加權是一個 Variable ，所以只需要取 出其中的 data 屬性，然後對它進行所需要的處理就 可以了。

for m in model m。dules ﹛）

1 2 3 4 5 6 7

if isrnstance ﹛m, nn c。nv2d) ,

rnit normal (m.weight .data﹜

。

1n1.t xav1er n rmal (m weight data) rnit. kaimrng\_normal ﹛m ﹒we斗ght.data)

m.bias.data .fill ﹛ 0)

elif isinstance (m, nn L工near) · m weight .data .n rmal＿﹛）

。

8

透過上面的操作，對將旋積層中使用 PyTorch 裡面提供的方法的加權進行初 始化，這樣就能夠使用任意我們想使用的初始化， 甚至我們可以 自己定義初始化 方法並對加權進行初始化。

I 4.4 I 旋積神經網路案例分析

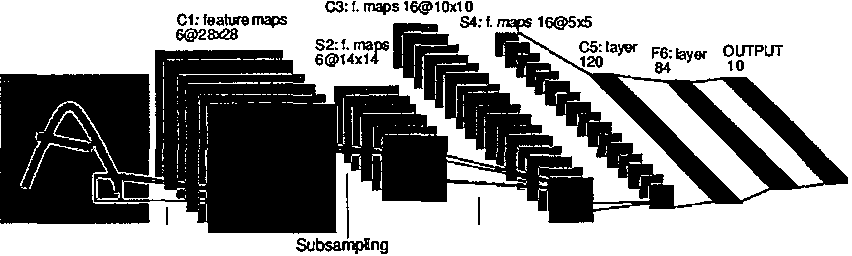
上面部分介紹了 PyTorch 中的旋積模組，接下來將介紹幾個旋積神經網路的 案例，透過案例入手來介紹旋積神經網 路的結構設計。

* + 1. LeNet

LeNet 是整個旋積神經網路的開山之作，1998 年由 LeCun 提出，它的結構特 別簡單，我們能夠由此入芋，一步一步地進入時下最為流行的旋積神經網路結構。

首先，LeNet 的網路結構如圖 4.16 所示。

制PITT



32X32

Convo 凶＂＇＇

l 叫側的＇＂ I 面 甜間酬叫間

c !'Ml ullons 勘＂＂＂叫略 目II connection

。

圖 4.16 LeNet 網路結構

從圓 4.16 可以看出整個網路結構特別清晰， 一共有 7 層，其中 2 層旋積和 2

層池化層交替出現， 最後輸出 3 層全連接層獲得整體的結果 。 針對這個簡單的網路，可以自己操作 。

1. class Lenet ﹛ nn Module﹜ ：
2. def init (self ) .
3. super ﹝ Lenet, self ) init II

4

1. layerl nn Sequential II
2. layerl add\_m。dule ﹛’c。nvl ，nn .Conv2d ( l, 6, 3, padding lJ J
3. layerl add\_m。dule I’paall ，nn .MaxP 。12d (2, 2 ） ﹜

B self layerl layerl 9

10 layer2 a nn .Sequential II

11 layer2 .add module ﹛’c。nv2’，nn .Conv2d (6, 16, 5﹜ ﹜

1. layer2 .add module ﹛ ＇poo2 ，nn MaxP。12di 2, 21 I
2. self layer2 a layer2

14

1. layer3 a nn Sequential ﹝）
2. layer3 add\_module ( ' fcl’，nn .Linear ( 400, 120 ） ﹜ 17 layer3 add\_m dule ﹝’fc2 ，nn .Linear (120, 84) I 18 layer3. add m dule （’fc3 ，nn .Linear (84 , 10 ） ﹜

19 self layer3 a layer3 20

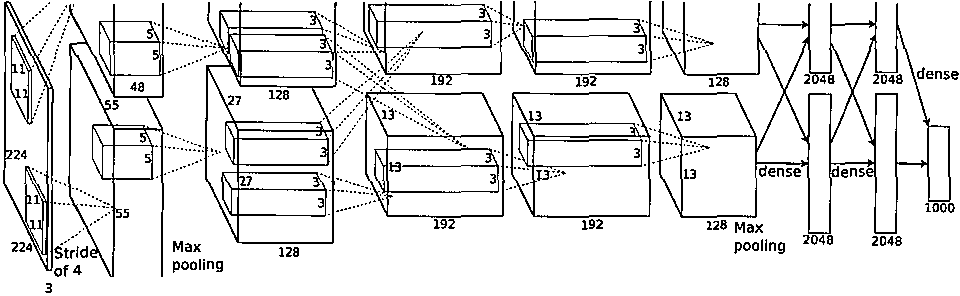
1. def forward (self , x) .
2. x a self.layerl (x﹜
3. x a self.layer2 (x﹜
4. X a X .View ﹛x size ﹛ OJ , 1﹜
5. x a self layer3 (x﹜
6. return x

這樣就實現了 LeNet 網路，可以發現網路的層數很淺，也沒有增加殷勤層。

* + 1. AlexNet

接下來要介紹 2012 年在 ImageNet 競賽上面大放異彩的 AlexNet’它以領先 第二名 10% 的準確率奪得冠軍，並且成功地向世界展示了深度學習的威力。

首先看看 AlexNet 的網路結構，如圖 4.17 所示。



p"o"。og

可E

圖 4.17 Alex Net 網路結椅

園 4.17 可能讓你看得眼花撩亂， 其實這是因為當時 GPU 還算能力不強，而 AlexNet 又比較複雜，所以 Alex 使用了兩個 GPU 平行來做運算，現在已經完全可 以用一個 GPU 代替了。

AlexNet 網路相對於 LeN仗，層數更深，同時第一次引用了敢動層 ReLU ，在 全連接層引用了 Dropout ｝冒防止過擬合。

實現 AlexNet 的網路結構如下。

class AlexNet ﹛ nn Module ）﹒

1

def init一一 ﹛ self , num classes﹜ super ﹝AlexNet, self ) . init （﹜ self .features - nn .Sequential ﹛

2 3 4 5 6 7 8 9

nn .Conv2d (3, 64 , ke主nel\_she-11, stride-4, padding-2) , nn ReLU ( inplace-True) ,

nn .MaxP。。12d (kernel sizea3, stnde-2) ,

nn. C。nv2d (64, 192, kernel size-5, padding-2) ,

nn ReLU (rnplace-True﹜ ，

1. nn MaxPo。12d ﹛ kernel size-3, stride-2) ,
2. nn .C。nv2d (192, 384 , kernel size呈 3, padding- 1﹜ ，
3. nn .ReLU ﹝ inplaceaT主ue) ,

。

1. nn .C nv2d ( 384, 256, kernel\_size•3, padding•l) ,
2. nn .ReLU ( inplace•True) ,
3. nn Conv2d ( 256, 256, kernel\_s工 ze•3, padding•l) ,
4. nn ReLU I inplace•True I ,

17 nn MaxP。。12d ﹛ kernel size•3, stride•2) , I

18 self.classifrer • nn Sequential I 19 nn .Dr p ut （﹜ ，

。。

20 nn .Linear ﹛ 256 • 6 • 6, 4096) ,

1. nn .ReLU ﹛ inplace•True﹜ ，
2. nn Dropout II,

23 nn Linear 14096, 4096) ,

1. nn .ReLU ( inplace•True) ,
2. nn Linear ( 4096, num\_classes ﹜，）

26

。

1. def f rward (self , x﹜ ．
2. x • self features ﹛ x)

29 X • X. V斗ew (x size ( O ) , 256 • 6 • 6﹜

1. x 貫 self.classifier ﹝ xi
2. return x

這是 mageNet 競賽史上第一次以旋積神經網路為基礎的模型獲得冠軍， 從此 掀起了深度學習在電腦！視覺上的革命。

4.4.3 VGGNet

VGGNet 是 ImageNet 2014 年的亞軍，歸納起來就是它使用了更小的濾波器， 同時使用了更深的結構，AlexNet 只有 8 層網路，而 VGGNet 有 的 層～ 19 層網 路，也不像 AlexNet 使用 11× 11 那麼大的濾波器，它只使用 3 X 4 的旋積濾波器 和 2 X 2 的大池化層。圖 4.18 是 AlexNet 和 VGGNet 的比較圖。

區噩噩噩盟 C主豆軍立2 亡三噩噩iL::J C歪歪iiCJ 亡三至

E歪歪歪歪司 亡豆豆豆莖3 亡:I!!室主S

己

CZ莖墨iiEl2] 亡豆豆 仁主歪歪歪莖3

三3

AlexNet

圖 4.18

區噩噩－盟

c:::J:!lJ:l!!!l三3

E軍噩噩E噩噩 E二草草草二3 亡三盟軍司 Cl§l§l![己 C畫室主2 E墨噩噩莖盟 E三豆豆E三2 C五5互D E至還歪歪司 E草互立露3 C豆豆豆這已

己 E豆豆豆互E

互E E噩噩噩單詞 E歪歪 E這歪歪 E歪歪自己 E莖 E幸這;;;;:]iI:] E豆豆

C還互歪歪3 E室主主諮3 E室還 E歪歪墨畫室EJ E歪歪]lD

C豆豆己 E豆豆豆軍司 自主噩噩ID C:噩噩噩E C歪歪孟立豆2 E玄歪歪主主3 C豆豆豆豆D E豆豆至2 c:!iI五E三3 E翠軍 C主歪歪歪主3 亡三盡量主軍司 C三豆豆E莖3 E主玄草草主3 C:豆豆三2

VGG16 VGG19

AlexNet 和 VGGNet 比較圖

它之所以使用很多小的濾波器 ，是因為層蠱很多小的濾波器 的感受野和一個 大的濾波器的感受野是相同的，還能減少參數， 同時有更深的網路結構。

同樣可以實現如下。

class VGG (nn .Mcdule) ,

1

def init＿一 ﹛ self , num classes) . super ﹛VGG, self ) init I ﹜ self.features - nn .Sequential ﹛

2 3 4 5 6 7

nn.Conv2d (3, 64 , kernel size-3, paddingal) ,

nn .ReLU (True﹜ ，

nn c。nv2d (64 , 64 , kernel\_size-3, paddingal) ,

8

nn .ReLU (True) ,

nn .Ma.xP。。12d (kernel size-2, stride-2) ,

。

9 1

nn c。nv2d (64, 128, kernel\_size-3, paddingal ) ,

1 2 3 4

1 1 1 1

nn ReLU (True) ,

。

nn c nv2d ( l28, 128, kernel\_size-3, paddinqal) , nn .ReLU ﹛ True) ,

nn.MaxP。12d ﹛ kernel size-2, stride-2) ,

1. nn Conv2d (128, 256, kernel\_size-3, paddingal ﹜ ，
2. nn .ReL心 （ True) ,
3. nn .c。nv 2d (256, 256, kernel\_sizea3, padding- 1) ,
4. nn ReLU ﹛ True ) ,
5. nn .c。nv2d (256, 256, kernel\_size-3, padding- 1) ,
6. nn .ReLU (True﹜ ，
7. nn MaxP。。12d (kernel size-2, stride-2) ,
8. nn c。nv2d ﹛ 256, 512, kernel sizea3, padding- 1) ,
9. nn .ReLU (True) ,

。

1. nn .c nv2d (512, 512, kernel\_size-3, paddingal) ,
2. nn .ReLU ﹛ True) ,
3. nn Conv2d (512, 512, kernel\_size-3, paddingal) ,
4. nn .ReLU (True) ,
5. nn MaxP。。12d ﹛ kernel sizea2, stridea2) ,
6. nn.c。nv2d ﹛ 512, 512, kernel size-3, padding- 1) ,
7. nn .ReLU ﹛ True) ,

。

1. nn .C nv2d (512, 512, kernel\_size-3, pa ddingal) ,
2. nn ReLU ﹝ True) ,

。

1. nn .C nv2d (512, 512, kernel\_sizea3, padding- 1) ,
2. nn .ReLU (True﹜ ，
3. nn .MaxP。。12d ﹛ kernel size-2, stnde-2) , )
4. self classi丘er a nn.Sequential ﹛

37 nn .Linear (512 \* 7 \* 7, 4096) ,

1. nn .ReLU (True) ,

。。

1. nn Dr p ut () ,

40 nn .Linear ( 4096, 4096) ,

1. nn. ReLU (True﹜ ，
2. nn Dr。pout () ,

4 3 nn .Linear ﹛ 4096, num classes﹜ ，）

44 self . initialize weights () 45

。

4 6 def f rward ﹛ self , x) ·

1. x a self .features (x)
2. x a x.view (x size (O ) , -1)
3. X a self.classihe主 ﹛x﹜

其實可以看出 VGG 只是對網路層進行不斷的堆壘， 並沒有進行太多的創新，

而增加深度確實可以某種程度改善模型效果。

4.4.4 GoogleNet

GoogLeNet 也叫 InceptionN剖，是在 2014 年被提出的，如今已經進化到了 v4

版本，下面介紹它最核心的部分。

GoogLeNet 採取了比 VGGNet 更深的網路結構， 一共有 22 層，但是它的參 數卻比AlexNet 少了 12 倍，同時有很高的計算效率，因為它採用了一種很有效的 Inception 模組，而且它也沒有全連接層，是 2014 年比賽的冠軍。

先看看 GoogLeNet 的網路結構和其中最為創新的 Inception 模組，如圖 4.19

所示。

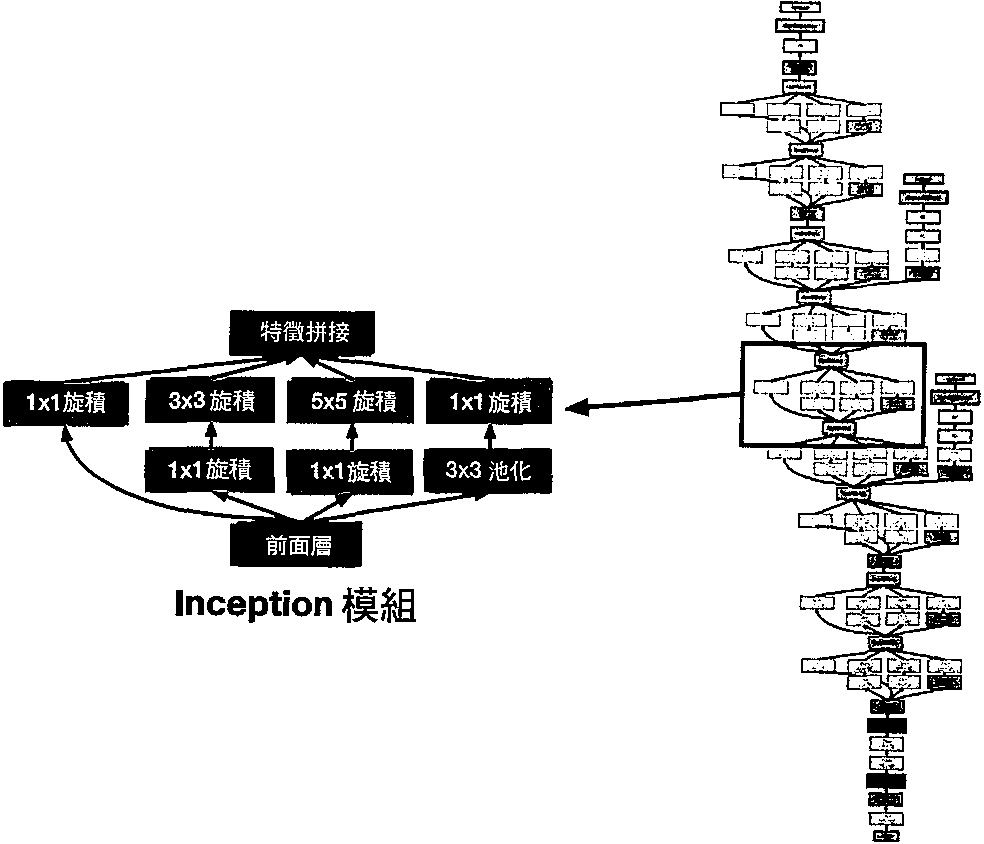


圖 4.19 I nception 模組

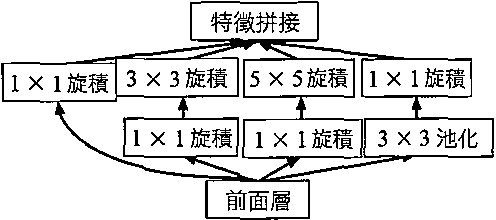
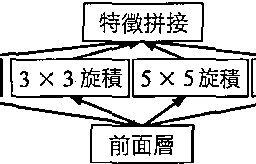
4.4 旋積神經網路案例分析 －

Inception 模組設計了一個局部的網路拓撲結構， 然後將這些模組堆疊在一起 形成一個抽象層網路結構。 實際來說就是運用幾個平行的濾波器對翰入進行旋積 和池化，這些濾波器有不同的戚受野， 最後將輸出的結果按深度連接在一起形成 輸出層。

這樣的網路結構非常新穎，正是由於這種網路結構，GoogLeNet 才能夠取得 如此大的成功，但是這種網路結構有一個小 問題，就是參數太多，導致計算複雜。

為了解決這個問題， GoogLeNet 又推出了下一個版本，這個版本對 Inception

模組有了新的設計，如圖 4.20 所示。



I X ］ 隨積 3 × 3 池化

簡單的 Inception 模組

緯度滅小的 Inception 模組

圖 4.20 Inception 模組的新設計

這個模組增加了一些 l X l 的旋積層來降低輸入層的維度， 使網路參數減少， 進一步減少了網路的複雜性。

下面實現 GoogLeNet 中的 Inception 棍組，整個 GoogLeNet 都是由這些 Inception

模組成的。

1. class BasicCcnv2d (nn .Modcle﹜ 回
2. def init (sel主，in channels，ut channels, \*\*kwargs) ,

。

1. super (BasicC nv2d, self ﹞ . init ()
2. self .conv nn. Conv2d ﹛ rn\_channels，ut channels, biasFalse, \*\* kwargs)
3. self bn a nn .BatchNorm2d （ ut channels, epsaO .001)
4. def f。rward {self, x) ,

6

1. X a self .C。nv I,也）
2. X a Self .bn (X)
3. return F. relu Ix, inplaceaTrue﹜

11

* 1. class Inception I即，.Module﹜ ：
  2. def in ct一一 ﹛ self , in channels, p。。l features) .

。

* 1. super I工 ncepti n, self ) init （﹜
  2. self .branchlxl BasicConv2d ( in channels, 64 , kernel s土 zel) 16

。

17 self .branchSxS 1 巨 BasicC nv2d (in channels, 48, kernel size司l)

。

18 self.branchSxS 2 BasicC nv2d (48, 64, kernel sizeaS, paddinga2) 19

1. self .branch3x3dbl\_l a BasicConv,d ﹛ in\_channels, 64 , kernel sizeal)
2. self .branch3x3dbl 2 a BasicConv2d (64, 96, kernel\_sizea3, padd斗ngal)

。

1. self .branch3x3dbl\_3 a BasicC nv2d ( 96, 96, kernel sizea3, paddingal) 23
2. self .branch*\_p*。al a BasicC。nv2d ﹛
3. in\_channels, pool\_features, kernel\_sizeal) 26

。

1. def f rward (self , x) .
2. branchlxl a self .branchlxl (x) 29
3. branchSxS a self.branchSxS 1(x)
4. branchSxS a self branchSxS 2 (branchSxS) 32
5. branch3x3dbl a self .branch3x3dbl 1(xi
6. branch3x3dbl a self .branch3x3dbl 2 (branch3x3dbl)
7. branch3x3dbl a self branch3x3dbl 3 (branch3x3dbl ﹜

36

1. branch\_p 1 a F.avg\_p。。12d ﹛ x, kernel\_sizea3, strideal, paddingal)
2. branch\_p。ol a self.branch\_pc。1(branch\_p 。1) 39
3. 。utputs ＝ ﹝ branchlxl, branchSxS, branch3x3dbl, branch\_p。。1﹞
4. return t。rch cat （。utputs, 11

首先定義一個最基礎的旋積模組， 然後根據這個模組定義了 l X l ’3 X 3 和 5 × 5 的模組和一個池化層，最後使用 torch.cat（） 將它們按深度連接起來， 獲得輸 出結果。

* + 1. ResNet

ResNet 是 2015 年 ImageNet 競賽的冠軍， 由微軟研究院提出，透過殘差模組 能夠成功地訓練高達 152 層深的神經網路。

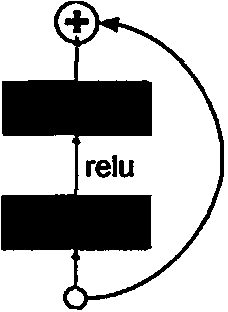
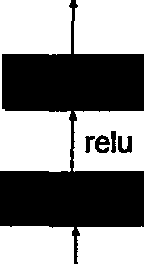
4.4 旋積神經網路案例分析 －

ResNet 最初的設計靈戚來自這個問題： 在不斷加深神經網路的時 候，會出現 一個 Degradati凹，即準確率會先上升然後達到飽和，再持續增加深度則會導致模 型準確率下降。

這並不是過擬合的問題， 因為不僅在驗證集上誤差增加，訓練集本身誤差也 會增加。假設一個比較淺的網路達到了飽和的準確率，那麼在後面加上幾個區等 對映層，誤差不會增加，也就說更深的模型起碼不會使得模型效果下降 。

這裡提到的使用值等對映直接將前一層輸出傳 到後面的思想，就是 Re刮目的 靈感來源 。假設某個神經網路的輸入是 v 期望輸出是 f司令）， 如果直接把輸入 x 傳到輸出作為初始結果， 那麼此時需要學習的目標就是 *F(x)=H(x* ） 弓，也就是下 面這個殘差模組， 如圖 4.21 所示。

*H(x) F(x*﹜*+x*



*X*

“Plain”layers

*FC*可）

*X*

Resiaua1 olock

圖 4.21 普通網路與 ResNET 的殘差學習單元

*X*

identity

圖 4.21 左邊是一個普通的網路， 右邊是一個 ResNet 的殘差學習單元， ResNet 相當於將學習 目標改變了，不再是學習一個完整的輸出 H(x）， 而是學習輸 出和輸入的差別 *H(x)-x*，即殘差。

ResNet 因為殘差模組的存在，使整個網路可以訓練高達 152 層，下面只用

ResNet 網路中的殘差模組舉例。

．國－ 第 4 章 提積神經網路

* + - 1. def c。nv3x3 (in\_ylanes, out\_ylanes, stdde-1)
      2. ”3x3 C。nv。luti。n with padding”
      3. return nn. C。nv2d ﹛
      4. in\_ylanes,
      5. out\_ylanes,
      6. kernel size-3,
      7. stride-stride,
      8. padd工ngal,
      9. bias-False) 10

11

1. class BasicBlock (nn.Module) .

。 。

1. def in工乞←一 (self , inplanes, planes, stnde-1, d wnsample-N ne) ,

。

1. super ﹝ BasicBl ck, self ﹜．斗nit （﹜

。 。

1. self .c nvl a c nv3x3 ( inplanes, planes, stride)
2. self .bnl a nn BatchN。rm2d (planes﹜

17 self .relu - nn .ReLU ﹛ inplace-True)

。 。

1. self .c nV2 a C nv3x3(planes, planes ﹜
2. self .bn2 - nn BatchN。rm2d ﹝planes)
3. self d。wnsample - downsample
4. self .stride a stride 22
5. def forward ﹛ self , x)
6. residual a X

25

。

1. ut a self .convl (x )
2. 。ut a self.bnl ﹝。ut)
3. 。ut a self .relu ﹛。ut)
4. 。ut a self c。nv2 （。ut )

29

1. 。ut - self bn2 （。ut)

32

1. if self .ct “nsample is n。t None.
2. residual - self .down sample ﹝x) 35

－

t

。

u

（

l u

a l

u e

1 ．

d z

s f

e l u

r e

s

z

＋ z

t t

叫 叩

6 7 8 9

3

3 3 3

從 forward 的最後一行，能夠看出網路將最開始的 x 加到了輸出當 中 ，形成 了殘差結糟。

。

t

n

x

u

t

開

除了這些比較出名的網路之外， 旋積神經網路的世界中還有很多別的網路， 例如 Network in Network 、HighwayNetwork 等，大家可以自己去看相關的論文。

另外並不需要重復造輪子，PyTorch 中早就為我們實現了上面介紹過的 這些網路， 都在 torchvision. model 裡面，同時大部分網路都有預訓練好的參數， 這些預訓練 好的網路為後面介紹的移轉學習和微調做了很好的準 備。

1 4回 5 1 再實現 MNIST 手寫數字分類

至今為止，已經介紹了很多很進階的網路結構， 是不是躍躍欲試了呢 ？記得 上一節我們使用簡單的多層全連接神經網路在 MNIST 資料上訓練 ，最後驗證集 達到了 98% 的準確率，能不能使用旋積神經網路來進一步提升網路的準確率呢 ？ 答案是一定的，下面就來寫一個多層旋積神經 網路，使它在 MNIST 資料集上達到 99% 的驗證集準確率。

1. class CNN (nn .M。dule﹜ ：
2. def init (self ) ,
3. super (CNN, self ) . init II
4. self layerl - nn Sequential ﹝

5 nn .C。nv2d ( l, 16, kernel S斗ze-3) , # b, 16, 26, 26

1. nn .BatchNorm2d (16﹜ ，
2. nn ReLU ( inplaceaTrue) I

8

9 self layer2 " nn Sequential (

。

。

nn .C nv2d (l6, 32, kernel\_size-3) , # b, 32, 24, 24 nn .BatchN rm2d ( 32﹜ ，

。

1

1 1 1 1 1

nn ReLU (inplaceaTrue) ,

2

nn.MaxP。。12d ( kernel\_size-2, stridea2) #b, 32, 12, 12

3 4

15 self layer3 a nn. Sequential ﹛

16 nn.C。nv2d ﹛ 32, 64, kernel size-3﹜ ，＃b, 64 , 10, 10

1. nn.BatchN。rm2d ( 641 ,
2. nn.ReLU ﹛ inplaceaTrue﹜ ）

19

20 self .layer4 " nn Sequential ﹝

21 nn Conv2d （ 駒 128, kernel\_size-3) , #b, 128, 8, 8

22 nn BatchN。rm2d (128) ,

1. nn ReLU (inplaceaTrue) ,
2. nn MaxP。。12d ﹛ kernel sizea2, stridea2) #b, 128, 4, 4

25

26

27 self fc - nn .Sequent ial (

28 nn Linear ( l28 \* 4 \* 4 , 1024) ,

29 nn.ReLU ﹝ inplaceaTrue﹜ ，

30 nn .Linear (l024 , 128) ,

31 nn ReLU ﹝ rnplaceaTrue ﹜ ，

32 nn .L斗near (l28, 10) )

33

1. def f。rward ﹛ self , x) .
2. X a self .layerl (X﹜
3. x a self .layer2 ﹛ x )
4. X a self .layer3 ﹛x﹜
5. x a self layer4 (xi
6. x a x .view (x size ﹛ O ) , 1)
7. X a self .f c (x)
8. return x

上面這個簡單的旋積神經網路是運用之前所學到的知識來建立的， 裡面有 4 層旋積，2 層最大池化，旋積之後使用批次標準化加快收斂速度， 使用 ReLU 殷 勤函數增加非線性，最後使用全連接層輸出分類得分。 最後再測試一下網路的結 果，可以看到圖 4.22 所示的測試集準確率已經達到了 99.31%’比之前使用的 3 層 全連接神經網路要高 。可以看到透過增加網路的深度和複雜化網路的結構加強網 路的準確率是可行的，下面我們將從資料方面出發來加強網路的準確率。

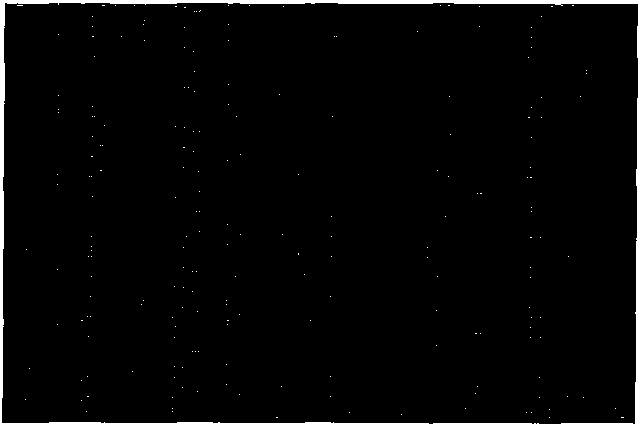


圖 4.22 測試集準確率

4.6 影像增強的方法－

圓圓

1 4圓＆ I 影像增強的方法

前面的部好專注於雄積神經網路的層結構介紹， 同時還介租了到臣前為止比 較出名的龍種神經網路， 接著使用比較握雜的旋積神經網路加強了 MNIST 資料集 的車確率。下面將從另外的角度一一影像增強的方面入手， 加強權型的車確率和

龍化能力。

一直以來，影像識別這一電腦視覺的核心問題都面臨很多挽戰， 間一個物體

在不同情前下都會得出不同的結論。

對於一張照片，我們看到的是一些物體， 而對於電腦而言，它看到的是一些

像章點，如圖 4.23 所示。

圖 4.23

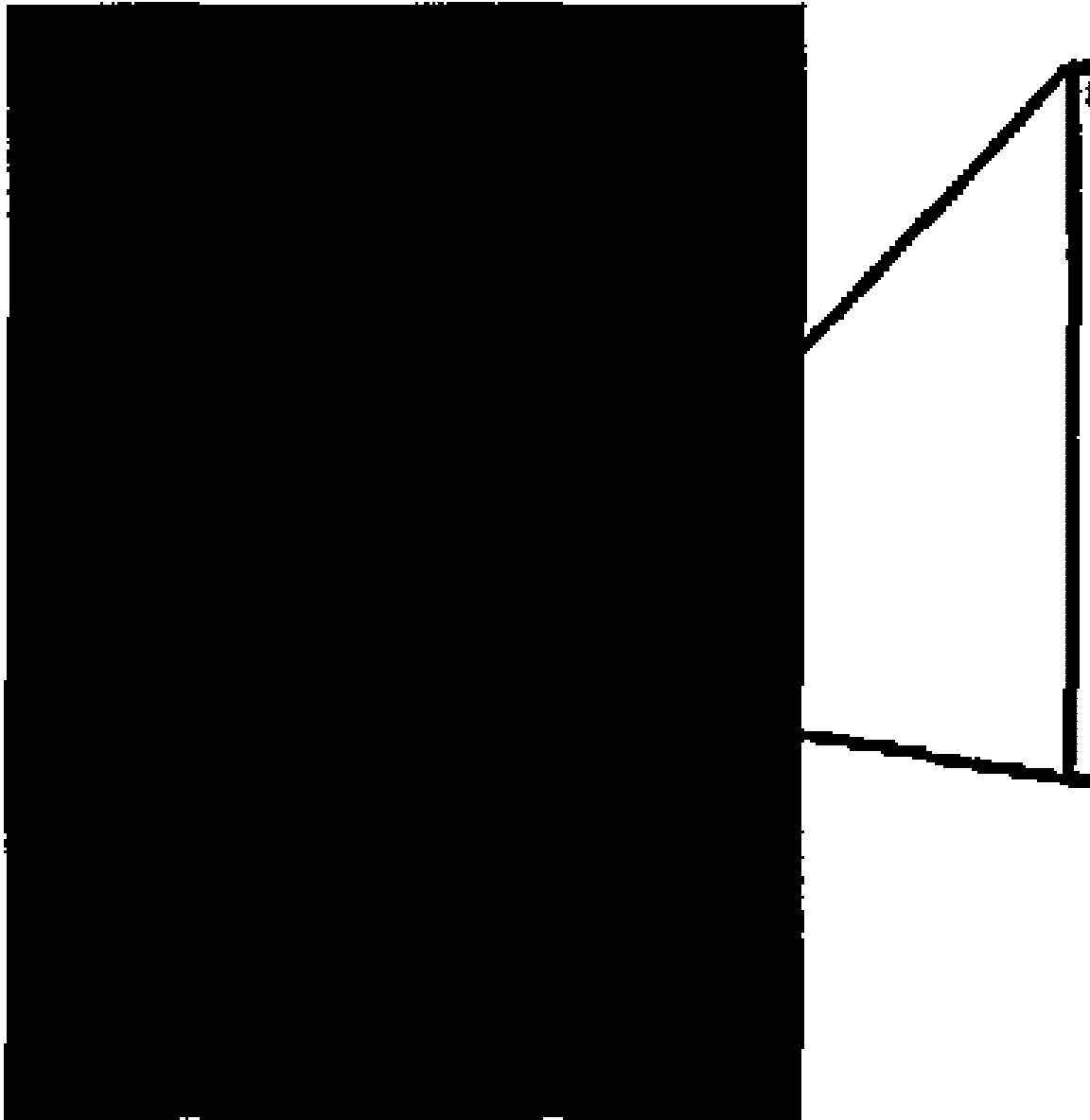
電腦識別的圖片

如果拍攝照片的照相攝位置龍生了改變， 那麼拍得的圖片對於我們而言，變

化很小，但是對於電腦而言， 圖片的像素變化是很大的。拍照時的光輯悔件也是

很重要的一個影響因素： 光輝太弱，照片裡的物體會和背景區直屬一體， 它們的像 章點就會很接近，電腦就無法正確識別出物體， 如圖 4.24 所示。

4-33



間闢刪叫樹叫回闖闖叫 剖 開 甜 甜 叫制別 扭

” ”

缸

，甜甜值”hn”恥”

．．

吼

－

趾恤，“

n 租”MW抽m MMm m ”－m m值”

“幢幢 區開凰”甜”

珊岫圓圓宙間

山間祖祖馳”” ”” ”

盟 “ 圓圓 M HH四 四 驢”

” ””帥”

珈山扭扭岫開曲”圓圓”u m m叫mmu 幢幢WE” 珊

甜

珊曲間抽間闢m Mmm 圓圓mmm m ”缸”mmmmm

m 咽 “時恥

”

”MRHha m 瑚曲曲曲祖a M Wm 四”削”一

””” ”

Mm扭扭帥 祖mmmmm Ha m閻明酬m Mm

－

” ” ””圓”” ”

曲 啪 mm 田間祖國 圓mm m帽”Mn

棚 甜甜 mmm 團曲曲回Mm 圓闡闖關m曲m Hn

”” ”

”n M棚 “圓 四

”

” 闡明世nm mm 祖祖國四酬m 曲”

叩開叫mm啪 叫間叫叫 ””” m m帥”u m

曲削凹的輛“岫叫四M 四

”圓圓圓圓”” 回 ””叮

即 四 叮－

m 岫闢剖開M W山山田四個開阻mnu 叫四酬閻明嘲叮叮一

”“” 詞圓圓世間一

“ 闡四 圓間四咀祖國m

個n M阻圓圓 －

m

m z

m 酬 摺一個田間

圖個圓圓 M

－

m M呻恤岫岫岫岫

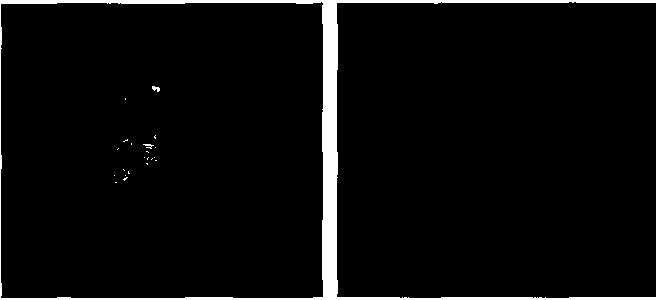


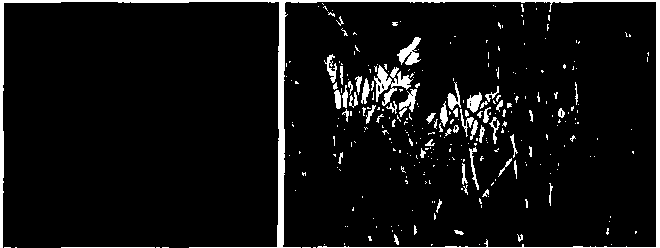
圖 4.24 光源對電腦話別的影響

除此之外，物體本身的變形也會對電腦識別造成障礙， 例如一隻貓是趴著 的，計算機能夠識別它， 但如果貓換了一個姿勢， 變成躺著的狀態，那麼電腦就 無法識別了，如圖 4.25 所示。



圖 4.25 姿勢對電腦識別的影響 最後，物體本身會隱藏在一些遮蔽物中，這樣物體只呈現局部的資訊，電腦

也難以識別，如圖 4.26 所示。



也包也呵呵 且也抽血

圖 4.26 是否有遮蔽物對電腦識別的影響

針對這些問題，我們希望能夠對原始圖片進行增強，在某種程度上解決部分 問題。在 PyTorch 中已經內建了一些影像增 強的方法，不需要再繁瑣地去實現， 只需要簡單的呼叫。

torchvision.transforms 包含所有影像增強的方法。 ”第一個函數是 Scale ，對圖片的尺度進行縮小和放大﹔

團 第二個函數是 CenterCrop ，對影像正中心進行指定大小的修改﹔

國 第三個函數是 RandomCrop，對圖片進行指定大小的隨機修改﹔

團 第四個函數是 RandomHorizaontalFlip，對圖片進行機率為 0.5 的隨機水平翻

轉﹔

國 第五個函數是 RandomSizedCrop ’首先對圖片進行隨機尺寸的修改， 然後對 修改的圖片進行一個隨機比例的縮放， 最後將圖片變成指定的大小， 這在 Inception Net 中比較流行﹔

”最後一個是 Pad，對圖片進行邊界零填充。

上面介紹了 PyTorch 內建的一些影像增強的方法，還有更多的增強方法，可 以使用 OpenCV 或 PIL 等協力廠商圓形函數庫實現。 在網路的訓練中影像增強是 一種常見、預設的做法，對多工進行影像增強之後都能夠在某種程度上提升工作 的準確率。

I 4.7 I 實現 cifar10 分類

前面我們處理過 MN!ST 手寫數字資料集， 這些資料集比較簡單， 大小是 28 X 泌 的灰階圈，沒有什麼太多的特徵’雖然實現了比較高的準確率，但是並沒 有太大的說服力 ，我們希望能夠在更加複雜的資料集上有更好的表現。

cifar!O 資料集有 60000 張圖片 ，每張圖片的大小都是 32 X 32 的三通道的彩 色圈，一共是 10 種類別 ，每種類別有 6000 張圖片，如圖 4.27 所示。

螂lane 圓圓圓悶 － 矗

automo蜘 盟國圖＝ 圓圓圓 ．

bird 團圓 國圖E頁 圓圓圓

cat 圓圓圓圓圓圓 團

deer 圖 圓圓

dog 圓 圍觀圖個國囡囡區司圈

frog

圓 ．

horse 國團團圓圓圓圖圖 ship 回國區圓圓圓圓圓圓圓圓 truck 團團圓圓圓圈圓圓圓

圖 4.27 cifar 資料集

使用前面講過的殘差結構來處理 叫做10 資料集，可以實現比較高的準確率。 首先進行影像增強，使用前面介紹的增強方式。

1. train\_transf 。rm = trans£。rms c。mp。se ﹛ ﹝
2. transf。rms .Scale ( 40 ﹜ ，
3. transf 。rms Rand。，mH。riz。ntalFlip II ,

*4* transf 。rms.Rand。mer。p (32) ,

5 transf 。rms.ToTens 。r （﹜ ，

6 transf 。rms.N•。rmalize ﹛ ﹝ 0 .5, 0 5, 0 .5﹞，﹝ 0 .5’。.5, 0 .5﹞ ）

7 ﹞ ）

8

1. test\_transform - transf 。rms Comp。se （ ﹝
2. transforms ToTens。x ﹛） ，

11 transf 。rms .N。rmalize ﹛ ﹝ 0 .5, 0 S, 0 .5﹞ ﹝ 0 S, 0 5, 0.5﹞ ）

12 ﹞ ﹜

注意只對訓練圖片進行影像增強，加強其泛化能力， 對於測試集，僅對其中 心化，不做其他的影像增強。

下面先定義好 resnet 的基本模組。

1. def c。nv3x3 ﹛ in\_channels ，。ut\_channels, stnde-1)
2. return nn.Ccnv2d (

4.7 實現 cifar10 分類 一

in channels, out channels, kernel sizea3, strideastride, paddrngal,

3 4 5 6 7 8 9 1

biasaFalse ﹜

# Residual Bl。ck

。

class ResidualBlock (nn .M。dule) :

1 2 3

1 1 1 1 1 1

def init ﹛ self , in channels，ut\_channels, strideal, d。wnsampleaN。ne) :

supe主 ﹛Residua工Bl。ck, self ) init I ﹜

4 5

self c nvl *==* c nv3x3 ﹛ in channels，ut channels, stride) self bnl " nn .BatchNorrn2d （ ut channels)

6 7 8 9

self .relu " nn ReLU ( inplaceaTrue)

1 1

self c nv2 a C nv3x3 （ ut channels，ut channels﹜ self .bn2 " nn .BatchNorrn2d （ ut channels﹜

。

1 2 2 2 2 2 2 2 2 2 2 3 3 3 3

self .ct wnsample " downsample def f。rward (self , x) ·

0 1 2 3 4 5 7 8 9 0 1 2 3

residual a X

out a self .convl (xi

。ut a self .bnl （ ut)

。ut a se工f.relu （ ut ) out a self.c nv2 ﹛ ut ) ut " self .bn2 ( out﹜

。

6

if self downsamp工e.

residual " self downsample (x) out += resiαual

。ut a self .relu﹛ ut﹜

return out

和前面介紹的內容一樣，先定義殘差模組，再將殘差模組連接起來， 注意其 中的維度變化。

class ResNet Inn .Module)

1 2 3 4

def init (self , block, layers, num\_classesalQ ) : super (ResNet, self ﹜ . init ﹛）

self .工n channels a 16

。

self .conv a c nv3x3 (3, 16) self .bn " nn.BatchNcrrn2d (16)

5 6 7

self .relu " nn ReLU ﹛ inplaceaTrue﹜

* 1. self layerl 呈 self make\_ layer ﹝bl。ck, 16, layers ﹝ O﹞ ）
  2. self.layer2 self .ma ke\_layer (bl。ck, 32, layers ﹝ O﹞，2)
  3. self .layer3 巨 self.make layer (bl。ck, 64 , layers ﹝ 1﹞，2)
  4. self avg\_pa 。1 nn .AvgP。。12d ﹛ 8)
  5. self .f c nn .Linear ( 64 , num classes﹜

13

1. def make layer (self , bl。ck’。ut\_channels, blacks, stridel﹜ ：
2. d。wnsample N。ne
3. if (stride ！1） r (self .in channels ， aut channels﹜
4. dawnsample nn .Sequential (
5. canv3x3 ﹛ self .in channels, aut channels, strideastride) ,
6. nn .BatchN。rm2d ﹝。u乞 channels﹜ ﹜
7. layers ＝ ﹝ ﹞
8. layers.append ﹝
9. bl。ck (self in\_channels ’。ut\_channels, stride, dawnsample﹜ ）
10. self in channels ＝ 。ut channels
11. f。r i in range (l, bl。cks﹜．
12. layers. append (bl。ck ﹛。ut\_channels, aut\_channels) )
13. return nn .Sequential ﹛＇layers﹜

27

。

1. def f rward (self , x ) ,

。 。

1. ut a self .c nv ﹛ x)
2. 。ut a self bn （。ut )
3. 。ut " self relu ﹛ aut )
4. aut " self layerl (aut)
5. 。ut a self layer2 （。ut)
6. 。ut " self .layer3 ﹛。ut )
7. 。ut a self .avg\_p。l (aut﹜
8. 。u t a 。ut view ( aut.size ( O ) , -1)
9. 。ut a self fc ﹛。ut )
10. return 。ut

最後在白色r!O 的資料集上跑 到 個 叩och ，實現 93.6% 的訓練集準確率， 86.23% 的驗證集準確率，因為這裡只跑了 50 次，所以還有一定的提升空間。 同 時使用更深的殘差和 更多的訓練技巧能實現更好的實驗結果， 如圖 4.28 所示。

4.7 實現 cifar10 分類 四 眉目

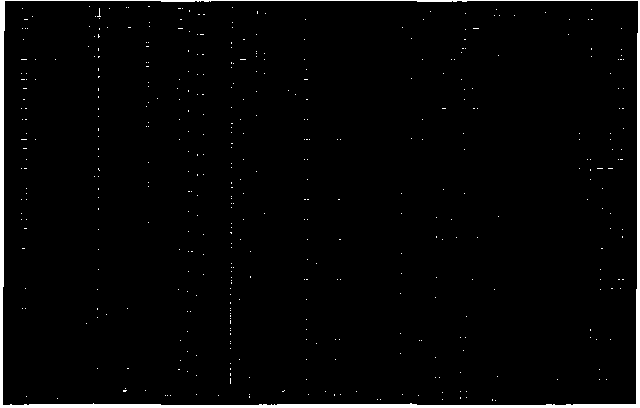


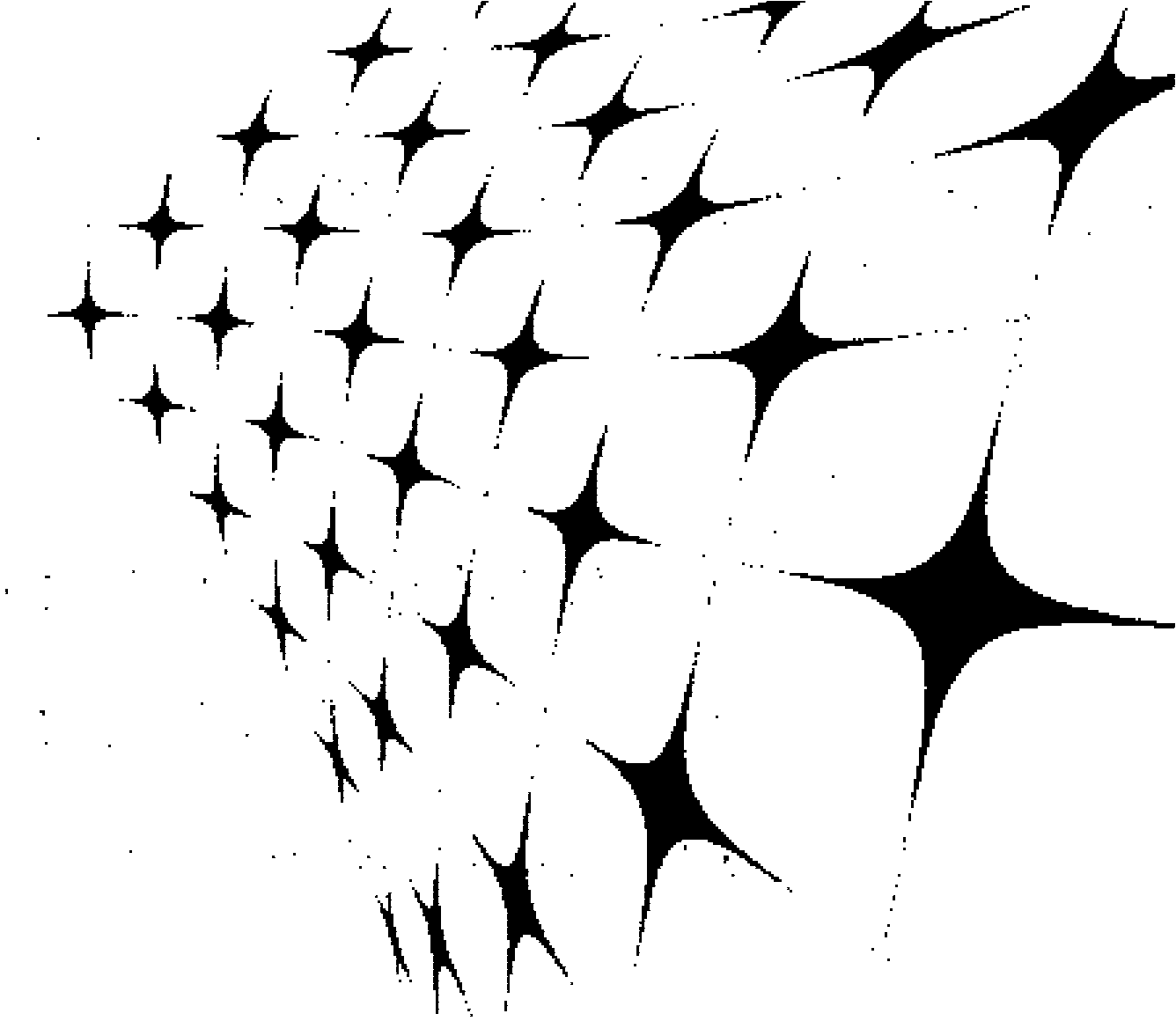
圖 4.28 實驗結果

上面的所有小節中，從電腦視覺的工作入手， 引用了旋積神經網路，介紹了 旋積神經網路的基本層結構， 接著介紹了時下非常出名的旋積神經網路案例， 最 後引用影像增強的技巧，使用 PyTorch 進行網路架設和訓練。

下一章將介紹一種非常適合處理序列資料和自 然語言問題的神經網路一一循 環神經網路。

M E M 日

第 5章



###### 循環神經網路

對於人類而言，以前見過的事物會在腦海裡面留下記塘， 雖然隨後記體會 慢慢消失，但是每當經過提醒，人們常常能夠重捨記倍。 在神經網路的研究中 ， 讓模型充滿記億力的研究很早便開始了 ，Saratha Sathasivam 於 1982 年提出了霍 普菲爾德網路，但是由於它實現困難， 在提出的時棋也沒有很好的應用場景， 所 以逐漸被遺忘。深度學習的興起又讓人們重新開始研究價環神經網路 （ Recurrent Neural Network ）， 並在序列問題和 自然語言處理等頡域取得很大的成功 e

這一章將從循環神經網路的基本結構出聲 ，介紹價環神經網路的基本模型和 存在的間題，接著介紹楣環神經網路目 前使用最多的兩種變式： LSTM 和 GRU • 然接說明一個使用 P）柚rch 處理時序問題的實例， 接著介紹福環神經網路在自 然 語言處理中的應用及其 PyTorch 實現，最佳介紹它在機器翻譯、 語音辨識等頡場 中的應用。

1 s.1 I 循環神經網路

前一章介紹了旋積神經網路， 旋積神經網路相當於人頓的視覺 ，但是它並握 有記幢能力 ，所以它只能處理一種特定的觀覺工作， 沒辦法根攝以前的記憶、來處 理新的工作 那麼記體力對網路而言到底是不是必要的呢 ？很顯然在某些問題上 是必要的，串倒來說 ，在一場電影中推斷下一個時間點的場景， 這個時儂僅依輯 於現在的情景並不鉤，需要位輯於前面發生的情節 對於這樣一些不僅依賴於目 前情沮，還依賴於過去情祝的問題，傳統的神經網路結構無法極佳地處理， 所以 以記惜品基礎的網路模型是必不可少的。

循環神經網路的提出便是以記憶模型為基礎的想法， 期望網路能夠記住前面 出現的特徵，並依據特徵推斷後面的結果， 而且整體的網路結構不斷循環， 因為 得名循環神經網路。

5.1.1 問題介紹

前面介紹過多層全連接網路和旋積神經網路， 可以發現這些網路不需要記憶 的特性也能處理對應的工作，下面舉一個實際的實例引出循環神經網路。

首先我們來看看下面這兩句話， 如圖 5.1 與圖 5.2 所示。

.） )2gjj 。n Novem ber 2°a ]

! ! ! ! !

other ggst other time time

圖 5.1 到達北京

i固有星。n

2°a ]

er

place 口f depa rtu re

圖 5.2 離開北京

第一句話表達到達北京的意思， 第二句話表達離開北京的意思。 對於這樣一 個工作，如果網路只輸入 「 beijing 」 這個詞而沒有記憶的特性， 那麼對於這兩句 話，網路會輸出相同的結果。但是如果網路能夠記憶 「 beijing 」 前面的詞，它就 會預測出不同的結果，所以說神經網路需要記憶的特性。

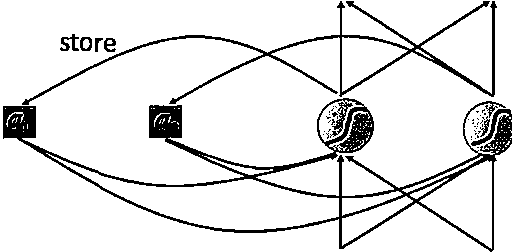
5.1.2 循環神經網路的基本結構

循環神經網路的基本結構特別簡單， 就是將網路的輸出儲存在一個記憶單元 中 ，這個記憶單元和下一次的輸入一起進入神經網路 中 。使用一個簡單的兩層網路 作為示範，在它的基礎上擴充為循環神經網路的結構，我們用圖 5.3 簡單地表示。

可以看到網路在輸入的時候會聯合記憶單元一起作為輸入， 網路不僅輸出結 果，還會將結果儲存到記憶單元中 ，圖 5.3 就是一個最簡單的循環神經網路在一 次輸入峙的結構示意圖。

從上面的原理中可以發現，輸入序列的順序改變 ，會改變網路的輸出結果， 這是因為記憶單元的存在， 使得兩個序列在順序改變之後記憶單元中 的元素也改 變了，所以會影響最後的輸出結果。

會 發



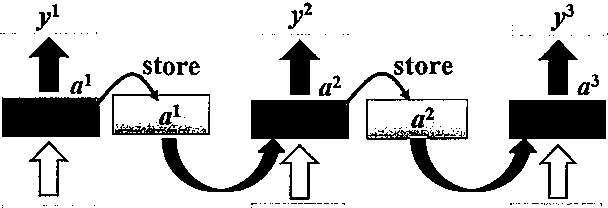
國 團

團 5.3 將一個數接點傳入網路

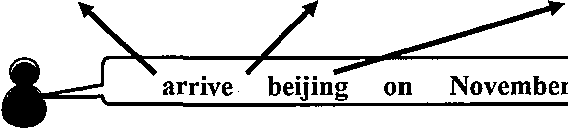
圖 5.3 是序列中一個數據點傳入網路的示意園， 那麼整個序列車日何傳入網路

呢？

將序列中的每個數據點依次傳入網路 即可，如圓 5.4 所示。



*x• x' x3*



2nd

圖 5.4 將整個序列傳入網路

無論序列有多長，都能不斷輸入網路， 最後獲得結果。可能看到這捏，讀者 會有一些疑問 ，圖 5.4 中每一個網路是不是都是獨立的加權 ？對於這個問題，先 考慮一下如果是不同的序列， 那麼圖 5.4 中格子的數目就是不同的，對於一個網 路結繕，不太可能出現這種參數目變化的情況。

事實上，這裡再次使用了參數共用的概念， 也就是說雖然上面有三個格子， 其實它們都是同一個格子， 而網路的輸出依賴於輸入和記憶單元， 可以周圍 5.5 表示。

如圖 5.5 所示，左邊就是循環神經網路實際的網路流， 右邊是將其展開的結 果，可以看到網路中具有循環結構， 這也是循環神經網路名字的由來。 同時根據 循環神經網路的結構也可以看出它在處理序列類型的資料上具有天然的優勢， 因 為網路本身就是一個序列結構， 這也是所有循環神經網路最本質的結構。

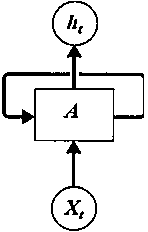
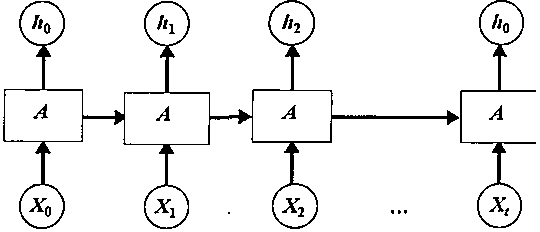
 

圖 5.5 網路的輸入和記憶單元

循環神經網路也可以有很深的網路層結構，如圖 5.6 所示。

|  |  |  |
| --- | --- | --- |
| *y'* | *y"* | *y*卅 |
| 一位 | 台 | 合 |

． ．．．

．．：

．．．

﹛企f ,,

于 ＇ .，，刊 i

A

圖 5.6 深層網路結構

．．．

可以看到網路是單方向的， 這代表網路只能知道單側的資訊，有的時候序列 的資訊不只是單邊有用，雙邊的資訊對預測結果也很重要， 例如語音訊號，這時 候就需要看到兩倒資訊的循環神經網路結構。 這並不需要用兩個循環神經網路分 別從左右兩邊開始讀取序列輸入， 使用一個雙向的循環神經網路就能完成這個工 作，如圖 5.7 所示。

x’ 刊 土仟*.z\_ \_ i*

：： 。圓－.－：：

們－＼ 門、 C n. •

' *y' y+'* :. *pz*

*u i*

u ' .• u ／ 」

..!

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ．．． | ←＇ | 企一.. | 企叮  c *x!'L.!* | 」企－  .2\_ j | ．． |
|  |  | 圖 5.7 | 雙向循環神經網路 |  |  |

使用雙向循環神經網路， 網路會先從序列的正方向讀取資料， 再從反方向讀 取資料，最後將網路輸出的兩種結果合在一起形成網路的最後輸出結果。

5.1.3 存在的問題

根據前面介紹的內容可以了解到循環神經網路具有特別好的記憶 特性，能夠 將記憶內容應用到目前↑背景下，但是隨後人們發現網路的記憶能力並沒有想像中 那麼有效。

記憶最大的問題在於它有遺忘性， 我們總是更加清楚地記得最近發生的事情 而遺忘很久之前發生的事情， 循環神經網路同樣有這樣的問題。 如果一項工作需 要依賴近期的資訊來預測結果， 循環神經網路常常具有比較好的表現，例如列出 下面一句話 「 我住在中國 ，我會講中文 J ，使用循環神經網路就能夠依據前面內容 中的 「中國」 來預測後面的 「中文 J 0 循環神經網路能夠極佳地解決這種短時依賴 的問題，如圖 5.8 所示。

．－－－ 第 5 章 循環神經網路

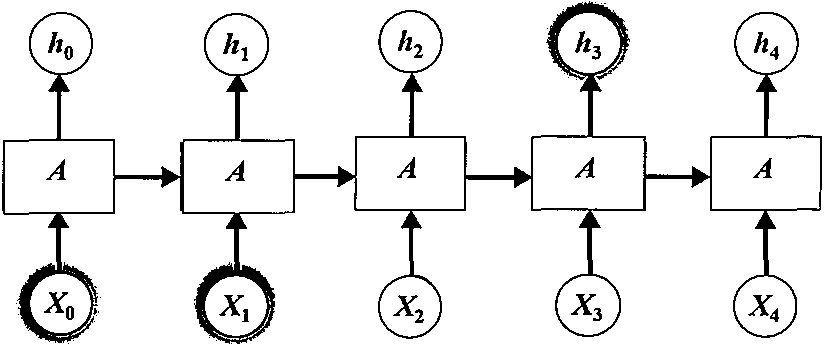


圖 5.8 循環神經網路解決短時依賴問題

但是對於長時依賴的問題， 循環神經網路的表現就不再那麼盡如 人意。例如 將上一個問題中的這句話拆開，在一篇文章的關頭寫上 「 我住在中國J 希望循環 神經網路在文章的尾端能夠預測 「 我會講中文」’這時網路就沒辦法極佳地預測這 個結果，因為記憶的資訊和預測位置之間的跨度太大， 網路常常不能記憶這麼長 時間的資訊，而且隨著時間跨度越來越大， 循環神經網路也越來越難以學習這些 資訊，要日圓 5.9 所示。

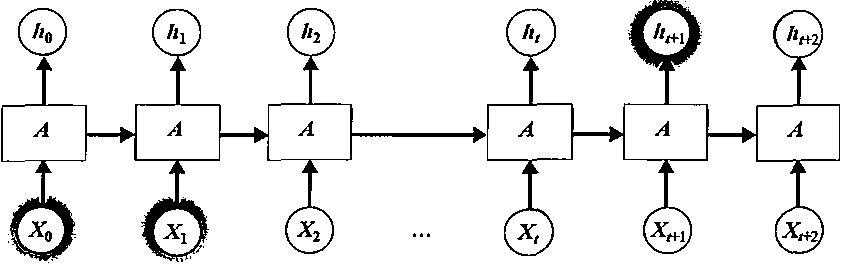


圖 5.9 循環神經網路記憶長時間資訊 這些問題導致循環神經網路在實際應用上一直沒辦法取得很好的效果， 但是

這些問題能不能解決呢？ 從理論上說，循環神經網路是完全能夠解決這種長時依 賴問題的，可以針對實際的問題，人為挑選一些特定的參數來解決， 但這並不具 有普遍性，因為循環神經網路沒有辦法自 己決定該挑選哪些參數。 1994 年 Bengio 甚至發表論文論述長時依賴問題的原因以及為什麼它難以解決。

早期循環神經網路的發展存在長時依賴問題， 後來依據循環神經網路的基本 結構，有人提出了一些特定的變式， 這些變式解決了長時依賴的問題，並在此後 循環神經網路的應用 中成為主流，下面就來介紹目前循環神經網路最為流行的兩 種結構﹒LSTM 和 GRU 。

1 s.2 1 循環神經網路的變式： LSTM 與 GRU

上一節介紹了因為循環神經網路的問題， 出現了兩種網路結構的變式：一種 叫 LSTM’一種叫 GRU’這兩種變式都能夠極佳地解決長時依賴的問題。 首先介 紹 LSTM 0

* + 1. LSTM

LSTM 是 Long Short Term Memory Networks 的縮寫，按字面翻譯就是長的短 時記憶網路，從字面意思知道它解決的仍然是短時記憶的問 題，只不過這種短時 記憶比較長，能在某種程度上解決長時依賴的 問題。LSTM 的網路結構是 1997 年 由Hochreiter 和 Schmidhuber 提出的，隨後這種網路結構變得非常流行， 有很多 人跟進相關的工作解決了很多實際的 問題，現在 LSTM 仍然被廣泛地使用 。

循環神經網路的結構都是鏈式循環的網路結構， LSTM 的網路結構大致上也 是這樣的結構， 不過 LSTM 在網路的內部具有更加複雜的結構， 所以它能夠處理 長時依賴的問題。

LSTM 的抽象網路結構示意圖如 圖 5.10 所示。從國 5.10 中可以看出 LSTM 由 三個門來控制，這三個門分別是輸入門 、遺忘門和輸出門 。顧名思義，輸入門控 制著網路的輸入，遺忘門控制著記憶單元 ，輸出門控制著網路的輸出。 這其中最 重要的就是遺忘門，遺忘門的作用是決定之前的哪些記憶將被保 留，哪些記憶將 被去掉，正是由於遺忘門的作用，使得 LSTM 具有了長時記憶的功能，對於指定 的工作，遺忘門能夠自己學習保留多少以前的記憶，這使得不再需要人為平擾， 網路就能夠自主學習 。

輸出門訊號捏制

軸λ門訊號控制

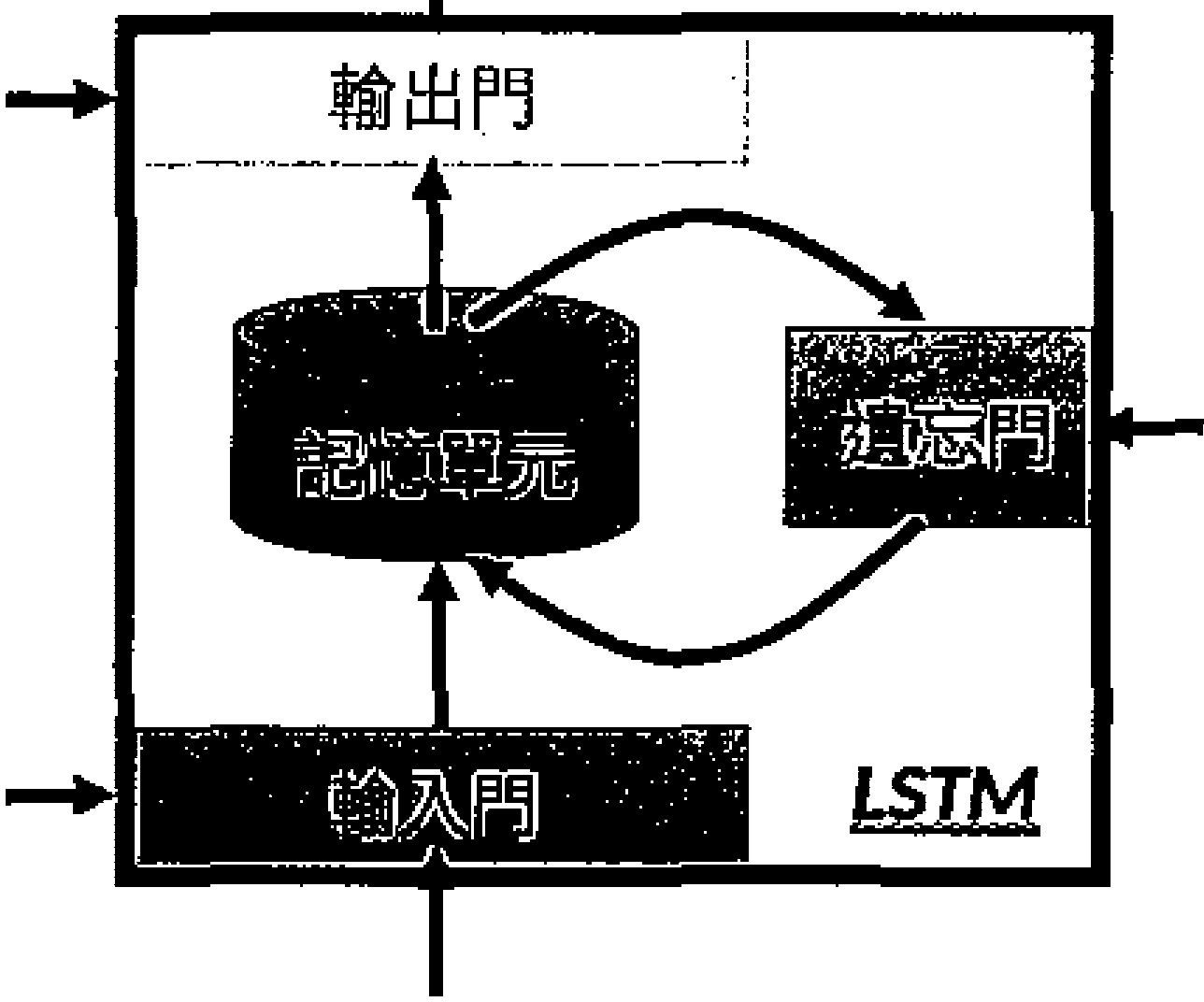
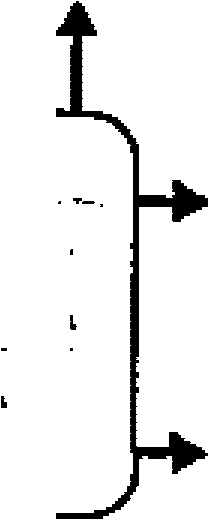
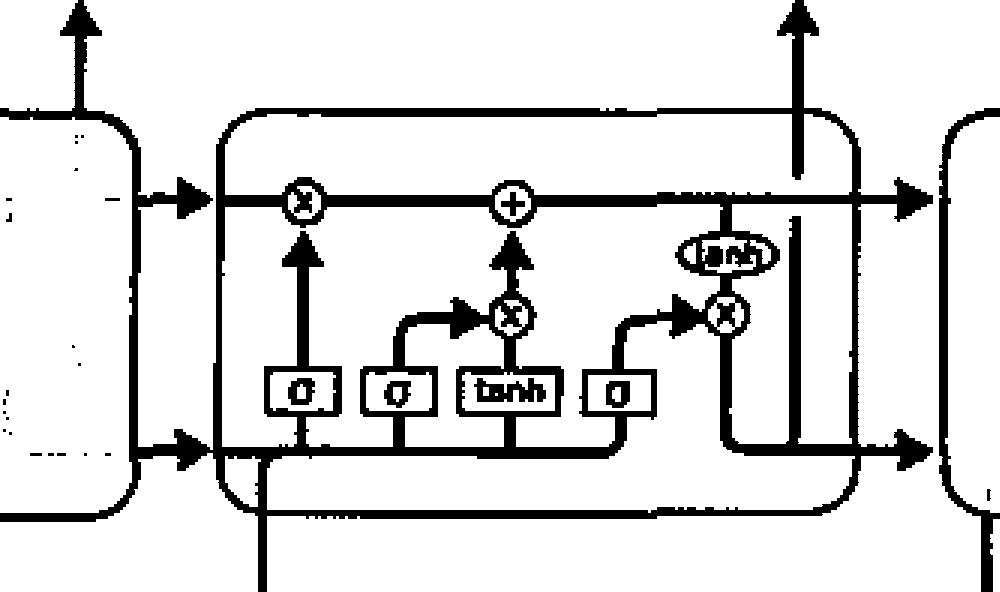
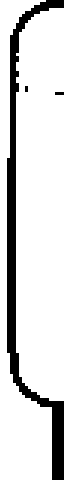


圖 5.10 LSTM 的抽象網路結構

遺忘鬥訊單控制

下面從實際的內部結構來解釋 LSTM 的網路流過程，首先列出 LSTM 內部的 結構示意圖，如圖 5.11 所示 ，

＠ ＠ ＠



且

h 且

＠ ＠ ＠

圖 5.11 LSTM 內部結構

*ct.*」 作為上一步 t-1 時刻網路中的記憶單元，傳入 t 時割的網路之後，第一 步操作是接定它的遺忘程度 ，將 t 時刻前面的記憶狀態乘上一個 0 ～ l 的係數進

行衰誠，接著加上 t 時刻學到的記憶作為更新之值的記憶傳出網路， 作品 t+ l 時 刻網路的記』臆單元。其中← 1 時刻網路記噫的衰誡係數是透過 r 時刻網路的輸入 和 t一 1網路的輸出來確定的， t 時刻網路學到的記憶也是根攝 t 時刻網路的輸入和 t-1 時刻網路的輸出獲得的， 下面會實際介紹如何計算出結果 。

標車的楣環神經網路內部只 有一個簡單的層結構， 而 LSTM 向部有 4 個層結 構，下面來位次解釋它們內部的運算方式以及如何表示上面所說的輸入門、遺忘 門和輸出門。

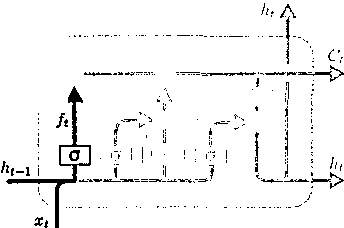
5.2 循環神經網路的變式 ：LSTM 與 G R U 一

首先是整個網路結構 中最重要的遺忘門，如圖 5.12 所示 。

圖 5.12

網路結緝的遺忘門

首先介紹記憶的衰減係數是如何獲得的 ，計算過程如圖 5.13 所示。



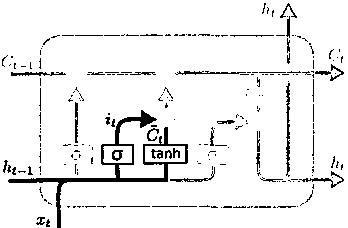
五 ＝σ（ *Wr*﹝*h,* 1 的I + *b1*l

區 5.13

衰減係數計算過程

首先將 ← 1時刻的網路輸出 九 1 和這一步的網路諭入 *x*，結合起來，然、後作用 線性轉換 時*1·* ﹝*h,.1 ,X* 」＋旬 ，最後再經過 sigmoid 敢動函數，將結果對映到 0 ～ I 作 為記憶的衰減係敏， 記作J呵 。可以看到網路實際要保留多少記憶是由前一時刻的 輸出和這一時刻的輸入共同決定的。

按著介紹 t 時刻學到的記憶是如何計算的，過程控日固 5.14 所示。



i, ＝σ （ W﹜ ［h, 1, x,] + *b,)*

*C,* = tanh(Wc ﹝*h,\_,, x,]* + *be)*

圖 5.14

t 時刻學到記憶的計算過程

首先對該時刻學到的記憶 ，對它應用一個衰減係數，這個係數應用跟上面的 方式相同，再使用線性轉換，然後使用 sigmoid 敢動函數，將結果對映到 0 ～ l 之間，這個結果作為目前學習到記憶的衰減係數， 記作 z 。目前狀態學習到的記

憶C 是透過線性轉換 月九﹒﹝l仙 x﹞ ＋*be* 和 tanh 敢動函數獲得的。

最後將 *I* I時刻的衰減係數*f,*乘 f一 l時刻的記憶 C, 1 加上該時刻 t 下學到的 記憶 c，乘以它對應的衰減係數 1，這樣便獲得了 t 時刻下的記憶狀態 C，可以用 圖 5.15 來顯示實際的計算過程。

*c,\_ ,* 一一一－ J 一一 一九

咕 ：［ …

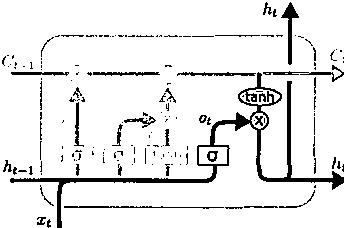
]:f

圖 5.15 C計算過程

上面的過程就是 LSTM 中的輸入門和遺忘門，最後介紹輸出門。

目前時刻t 的網路輸出 *h*，取決於目前時刻 t 的記憶狀態 C和 r 時刻的輸入

冉、t l 時刻的輸出 *h,* 1，實際的計算過程如圖 5.16 所示。



*o,* ＝σ（師已 ［ ht-1’的﹞ ＋ *bo)*

*h,* = *o,* •tanh (C,)

圖 5.16 *0* 與 h 的計算過程 首先使用類似計算記憶衰減係數的方法計算獲得輸出門的係數 ，這個係數

決定了輸出的多少，最後網路的輸出由 *h,=o,*X *tanh(C*，） 獲得，這就是輸出門如何 控制網路輸出的原理。

上面介紹了 LSTM 內部的網路結構和計算方式，它和傳統循環神經網路最大 的不同在於它使用了輸入門 、遺忘鬥和輸出門來控制網路實現長時記憶的功能 。 實際上在每篇使用 LSTM 的論文中 ，作者都使用了和標準 LSTM 稍微不同的版 本，這些變式的網路結構並不存在哪一個是最好的說法， 針對特定的工作 ，特定 的網路結構常常表現得更好。

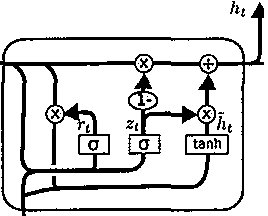
接下來介紹和 LSTM 比，具有更大區別的網路結構一－GRU 0

* + 1. GRU

GRU 是 Gated Recurrent Unit 的縮寫，由Ch 在 2014 年提出。GRU 和 LSTM 最大的不同在於 GRU 將遺忘門和輸入門合成了一個 「 更新F恥，同時網路不再額 外列出記憶狀態 C’而是將輸出結果 *h*，作為記憶狀態不斷向後循環傳遞 ，網路的 輸入和輸出都變得特別簡單。

GRU 實際的計算過程可以用國 5.17 來表示。

,,,\_, *z,* ＝σ （ *Wz [h,* 1, *x,])*



*r,* ＝σ （ l仇 ［ h門，叫）

*h,* = tanh *(W* · [r, \* h,\_1，叫）

*h,* = (1 品） *• h, 1 + z,• h,*

*x,*

圖 5.17 G RU 計算過程

上面列出了 GRU 實際的計算過程，就不再贅述了。它和 LSTM 本質上是相 同的，將上一時刻 f一 1的輸出 h「1和目前 f 時刻的輸入 *x*，結合起來計算各種衰減 係數，略微不同的地方是，線性轉換沒有使用偏置， 由於記憶狀態也是 *h*，一 1，所

以直接對它進行更新就可以了 ，最後輸出網路的結果 丸，這個結果也是網路的記 憶狀態 。

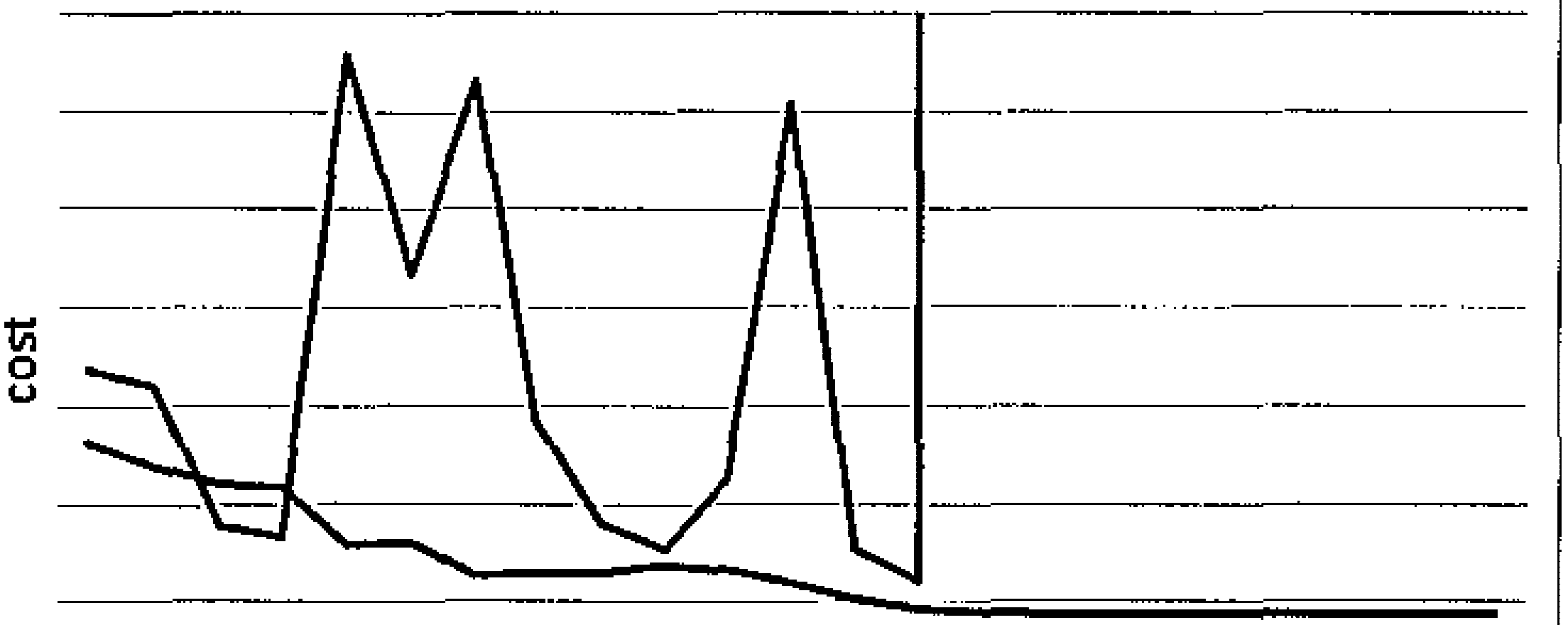
以上了解了 目前使用最為廣泛的兩種植環神極網路變式： LSTM 和 GRU’它 們為循環神經網路的發展做出了 重要貢獻，而研究者預計福環神經網路的下一步發 展將是挂意力攝制， 本書的實戰部分將介紹如何將注意力體制應用到攝器翻譯中 。

5.2.3 收敢性問題

雖然措環神經網路由最初的標車形式發展出了很多變式， 個如 LSTM 和

GRU ’但是這些網路都無可避免地存在一個問題，那就是故蝕性。

如果寫了一個簡單的 LSTM 網路去訓練資料， 你會發現 loss 並不會接聽想﹛摯 的*1i*式下降，如果運氣好的話能夠獲得一直下降的 loss’但是大多數4情況下 Loss 都是在亂跳薯的，如圖 5.18 所示。



1 2 3 4 5 6 7 8 9 1日 11 12 13 14 15 16 17 18 19 20 21 22 23

epoch

圖 5.18 loss 亂臨 運氣好的話能夠獲得藍色的下降曲 線，運氣不好的話就會就獲得糖色的下降

曲線，這在楣環神經網路的種展初期形成 7很大的阻礙， 因為這種不種定的訓練 過程是控辦法使用的。 同時人們也屬到疑惑，為什麼以提積為基體的神經網路不 會出現遣種跳躍的 loss 而以循環為基體的神罐網路就會出現這樣的情況，在隨 後的研究中，人們發現出現這種情前的根本原因是因為 恥凹的誤差曲面瞌睡不 平，如圖 5.19 所示。

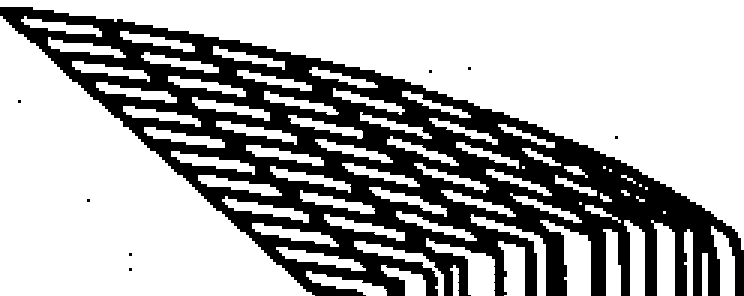
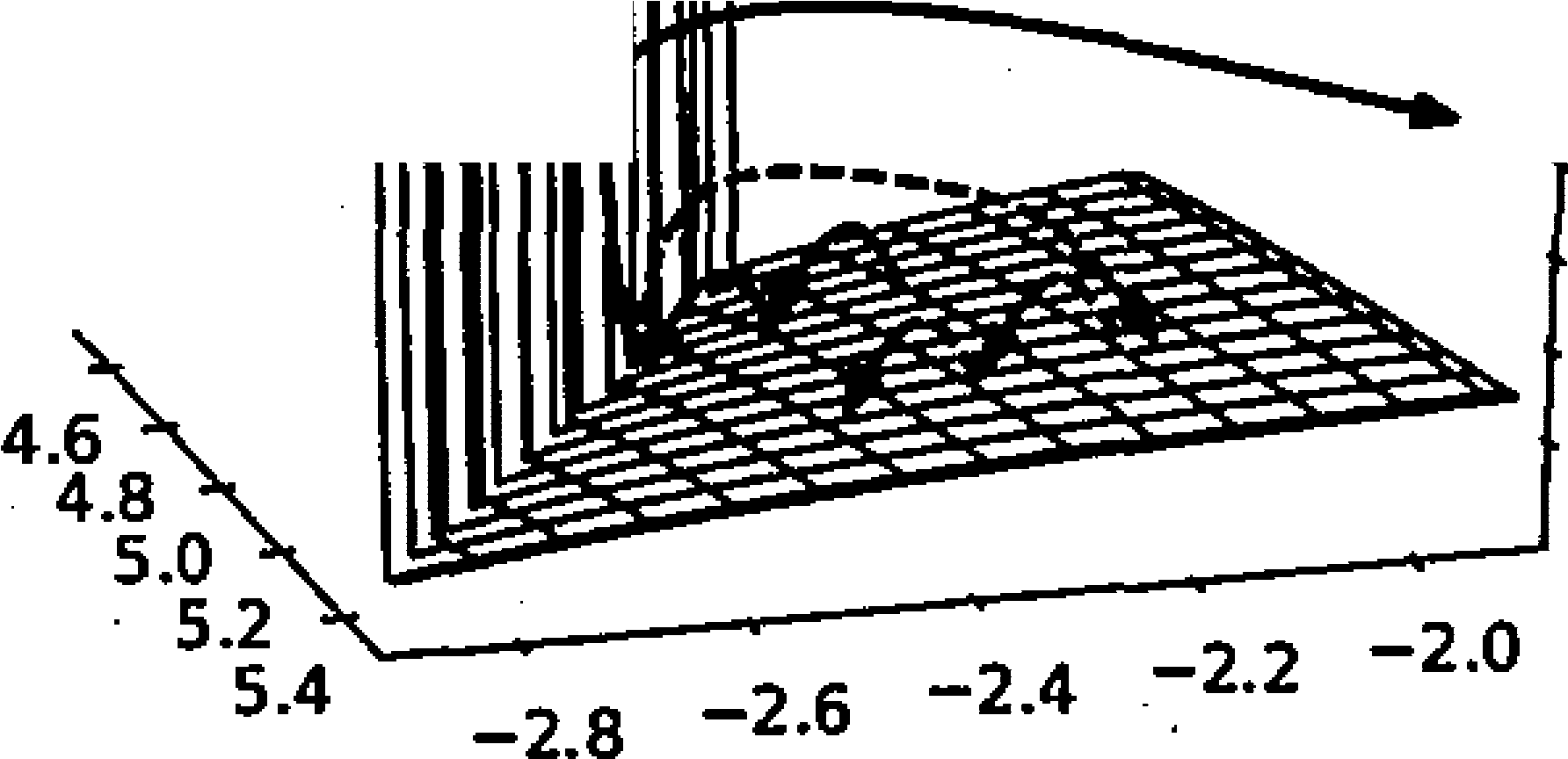


圖 5.19 RNN 誤差曲面祖睡不平



Ed nu Ed nu

zd nU EJ

332211o

oooaaaa

誤差曲面上存在很多陡峭的斜壤， 斜坡上誤差的變化率特別大， 正是這個原 因導致道路 loss 曲線在不斷地跳躍。對這種以循環為基確的網路結構的不揖則 、 健峭的斜坡曲面， 能夠採用什麼方法對網路進行訓練呢 ？可能有的人會說只需使 用比較小的學習率就可以7 ，下面用一個實個來說明 這種方法是不可行的。

首先引用一個非常簡單的循環網路， 如圖 5.20 所示，隱蔽狀態的加權是 *w •* 其備加權都是 1 ，網路輸入的序列長度是 1000 ，除了第一個是 1 之外，其餘的都 是 0，網路上的／ •••yJOOO 表示輸出，計算樹暑序列最佳的輸出是 w憫 。

nu

au

m

1

yflI

H

ff

咱I

Z

yaEEl

．4 A

．

－

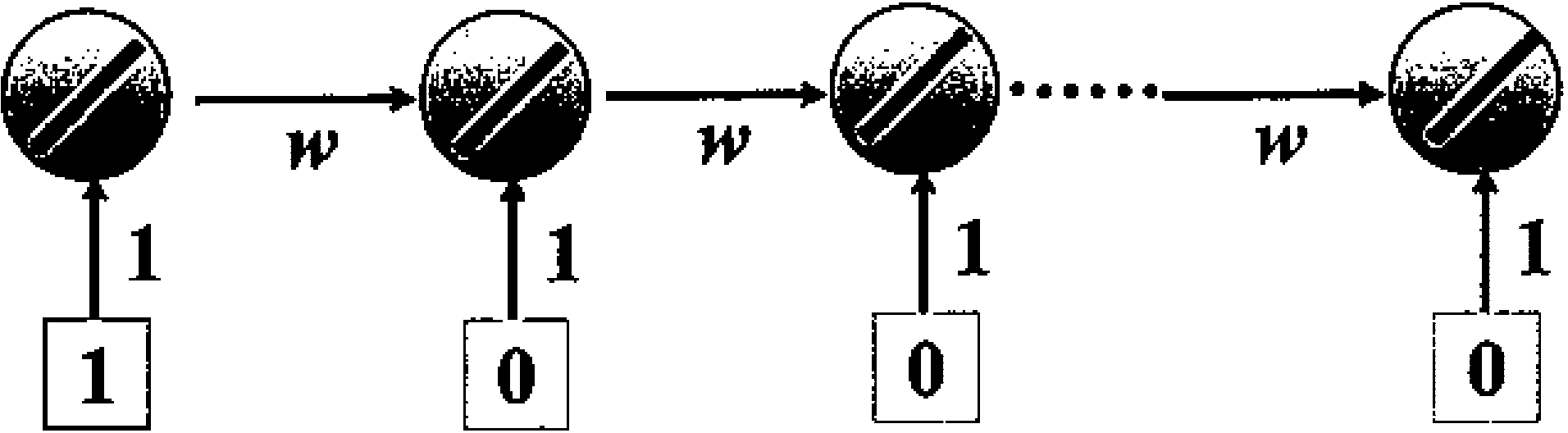
A

V ＃

11

H

圖 5.20 簡單的櫃環網路



－－－－－ 第 5章 循環神經網路

對於這樣一個結構，如果 *w= I*，那麼輸出 *y* ＇刪 ＝*y*卿＝ I，如果 *w=* 1.01’那麼 輸出 *y'* 酬＝ I.OJ 卿＂＇20000，所以計算出它的數值梯度如下：

*BL* ～ *11L* \_ 20000 一 I

*aw 11w* 1.01一 l (5.1)

可以看到，這裡的梯度非常大，所以需要非常小的學習率才能夠收斂。 但是 ，如果 *w=O.* ，那麼y岫＝0.99999 叫，獲得的數值梯度如下 ：

*BL* ～ 血L 一 。一 I

(5.2)

*aw 11w* 0 99一 l

這樣獲得的梯度又非常小， 需要一個很大的學習率才能 收斂。

這個問題的原因很明顯，就是2日權在網路中循環的結構裡面會不斷地被重複 使用，那麼梯度的微小變化在經過循環的結樁之後都被放大。 正是因為網路在訓 練的過程中梯度的變化範圍很大， 所以設定一個固定的學習率並不能有效收斂 ， 同時梯度的變化並沒有規律，所以設定衰滅的學習率也不能滿足條件。

是否沒有辦法來解決這個問題呢 ？其實是有辦法的，說出來你可能會覺得太 簡單了，但是確實有效果，那就是梯度修改 （ gradient clipping ）。 使用梯度修改能 夠將大的梯度裁掉，這樣就能夠在某種程度上避 免收斂不好的問題。

當然，現在能用很多別的辦法解決這個問題， 這裡只是提出循環神經網路確 實存在著收斂性問題，有興趣的讀者可以查閱一些近年的論文。

I 5.3 I 循環神經網路的 PyTorch 實現

前一節介紹了循環神經網路的理論和實 際計算，這些復雜的計算並不需要從 頭去寫，PyTorch 中早已整合好了一切供我們呼叫，下面這個部分將介紹循環神經 網路的 PyTorch 實現。

首先先介紹 PyTorch 呼叫循環神經網路的 API •接著用兩個簡單的實例實際 說明 PyTorch 中循環神經網路的實作方式。

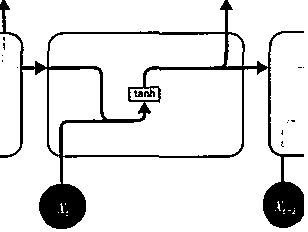
5.3.1 PyTorch 的循環網路模組

下面會分別介紹標準即州 、LSTM 和 GRU 說組在 PyTorch 中的呼叫。

己過 1.標準 RNN

先列出標準 RNN 的示意圖 ，如圓 5.21 所示，按照圖 5.21 來說明 PyTorch 中 的 AP 。

＠ ＠ ＠



*A*

＠



*A*

圖 5.21 標準 RN N

從圖 5.21 中可以看到在標準即們 的內部網路中，計算公式如下：

*h,-tanh( w,,,•x,+b*的 ＋*w,,,\*h,\_,* +*b*叫 (5.3)

在 PyTorch 中的呼叫也非常簡單，使用 nn.肘 可0 即可呼叫，下面依次介紹其 中的參數。

input\_size 表示輸入冉 的特徵維度﹔hidden size 表示輸出 *h* 的特徵維度﹔ num\_laye 間 表示精路層數，預設是 I 層﹔nonlinearity 表示非線性敢動函數的選 擇，預設是 旭恤， 可以選擇 relu﹔bias 表示是否使用偏置， 預設是 True﹔batch first 這個參數是決定網路輸入的維度順序， 預設網路輸入是按照 （ s呵，batch, feature） 輸入的，也就是序列長度放在最前面，然後是批次，最後是特徵維度，如 果這個參數設定為 True ，那麼順序就變為 （batch, seq, feature）﹔ dropout 這個參數 接受一個 0 ～I 的數值，會在網路中除了最後一層之外的其他輸出 層加上 dropout 層﹔bidirectional 預設是 False 如果設定為 True’就是雙向循環神經網路的結構。

接著再介紹網路接收的輸入和輸出。 網路會接收一個序列輸入品 和記憶輸 入 J旬，冉 的維度是 （seq, batch, feature）， 分別表示序列長度、 批次和輸入的特徵 維度，hO 也叫隱藏狀態 ，它的維度是 （ layers direction, batch, hidden）， 分別展現 層數次方向 （ 如果是單向 ，就是 1 ，如果是雙向就是 2 ）、 批次和輸出的維度。網

路會輸出 output 和丸，output 表示網路實際的輸出， 維度是 （s呵 ，batch ’hidden

direction）， 分別表示序列長度、 批次和瀚出維度乘上方向，*h*，， 表示記憶單元，維 度是 （ layer direction ’batch ’hidden）， 分別展現層數次方向 、批次和輸出維度。

下面來解釋幾個令人感到疑惑 的地方。

﹒第一個要注意的地方就是網路的輸入和前面講過的旋積網路有些不同 ，因為 旋積神經網路的輸入將 batch 放在前面，而在循環神經網路 中將 batch 放在中 間，當然可以使用 batch first-True 讓 batch 放在前面。

﹒第二個要注意的地方就是網路的輸出是 （s呵，batch, hidden 吋1rection）， 這裡 direction= I 或 2 ，前面也介紹過， 如果是雙向的網路結構，相當於網路從左 往右計算一次， 再從右往左計算一次， 這樣會有兩個結果， 將兩個結果按最 後一維連接起來，就是上面的結果。

﹒第三個要注意的地方就是隱藏狀態的網路大小 、輸入和輸出都是 (layer

\*direction, batch, hidden）， 因為如果網路有多層，那麼每一層都有一個新 的記 憶單元，而雙向網路結構在每一層左右會有兩個不同的記憶單元， 所以維度 的第一位是 layer \*direction 。

下面用 PyTorch 實現簡單的循環神經網路， 並驗證上面所介紹的內容。 首 先建立一個簡單的循環神經網路： 輸入維度是 20 、輸出維度是 50 、兩層的單向 網路。

1 basic\_rnn 旦 nn. 由叩﹛input\_size20, hidden\_size SO, num\_layers 旦2)

對於式 （ 5.1 ） 中的每個加權，都能夠透過如圖 5.22 所示的方式直接存取。 對於定義好的 則憫，可以透過 weight ih\_10 來存取第一層中的 圳，因為輸入

*x*，是 20 維，輸出是 50 維，所以 *W;1,* 是一個 50 ×泌 的向量 ，另外要存取第二層網

路可以使用 weight\_ih\_ll •對於 w ，，＇可以用 weight\_hh\_IO 來存取，而 丸，即可以用 bias ih 10 來存取。當然可以對它進行自訂的初始化，只需要記得這些學數都是 Variable’取出它們的 data 對它進行自定的初胎化即可。

|  |
| --- |
| ．．』．． ---· . -· ----· --·· -·- -  basic\_rnn.weigbt\_ih\_lO |
| Parameter containing,  -4.4217e胃口2 -1.1岳36e-02 -l.0950e-Dl • • , J.0989e『02 -5.355 e 曰：2 l.3470e-D2 2,Dl99e-02 2.6346e-02 -l.2236e-01 ... J, D369e-02 3 .1B75e 目2 7.5414e『02  -7 ,J592e-02 -1.0765e-Ol -6.4064e 2 ... -1. l&le-02 -2.7480e-D2 -7. B86le-02  ．’，． ． ．．．  1.3970e-01 s.zzsae『02 -1.5994e-oz ... 7.24lJe-D2 -l.ll46e”。1 ”4.zoose-oz  l.9304e-D2 l.2416e-01 4.3513e-02 ... -6,990Je-02 -l,9!187e-02 9粵 1069e-02  -l.2357e”。1 l.3164e-Ol 4 ，64e-02 ... -1.2016e-01 l.1797e-Ol -7.6756e-02  [torch.Float'rensor f sbe 50x2DJ |
| Ibasic\_rnn.we岫，t\_hh\_lo  Parueter containing,  。.0022 0.0818 0.0834 ... ”。.1024 -D.0085 -0.0421  -0.0973 0,1049 -0’旦 自 ..• -0.0427 0.0391 -0.0953  -0.0930 -0.0776 0.1303 ••• 。.0230 -0.1142 0,1007  ．．． ．．．  -0.0447 -0.1351 -0.0234 , .• 0.094目 。.0350 D.0472  。.0477 0.1165 -0.0886 ... -0.0022 0.1095 曲 自由51  -o.0309 o.0166 o.0746 , , ，”。.0695 0,0498 0.0692  (torch. Float'rens r of she 50當50] |

圖 5.22 直接存取古式

接著l時輸入傳入網路， 驗證獲得的輸出是否如前面介紹的一樣。 首先隨機初 始化輸入和隱藏狀態如 下，輸入是一個長為 100 、批次為 32 、緯度為 20 的張量 ， 隱藏狀態的緯度按照網路的需求定聶如下：

1. toy\_input = Variable (torch.randn (lOO, 32, 20﹜ ﹜
2. h\_O = Variable (torch .randn (2，泣，50） ﹜ ＊ layer \* direction, batch, hidden\_size

然值將輸入手日厝藏狀態傳入網路，獲得輸出和更新之後的隱藏狀態， 輸出的 長度是 100 、批次是 32 、緯度是 呵，和前面介紹的一致， 而更新之後的隱藏狀態 和輸λ的體聽狀態也是大小相同的。

1. toy\_output, h\_n = basic\_rnn (toy\_input, h\_O﹜
2. print (toyιoutput .size ﹛ ） ） ,t seq, batch, hidden\_size

*3* print (h\_n .size （﹜ ） \* layer \* direction, batch, hidden\_size

獲得它們的維度分別是 （ 100, 32, 50） 和 （2 泣，50）。 如果在傳入網路的時使不特別註明隱藏狀態 仇，那歷朝始的隱藏狀態預設事

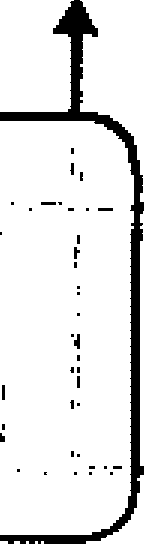
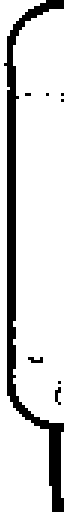
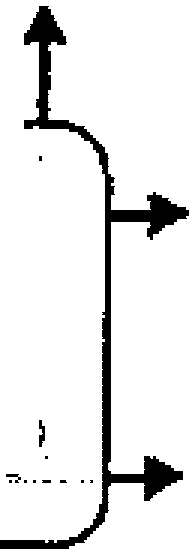
數全是 0，當然也可以用上面的方式來自 訂隱藏狀態的初始化，

2. LSTM

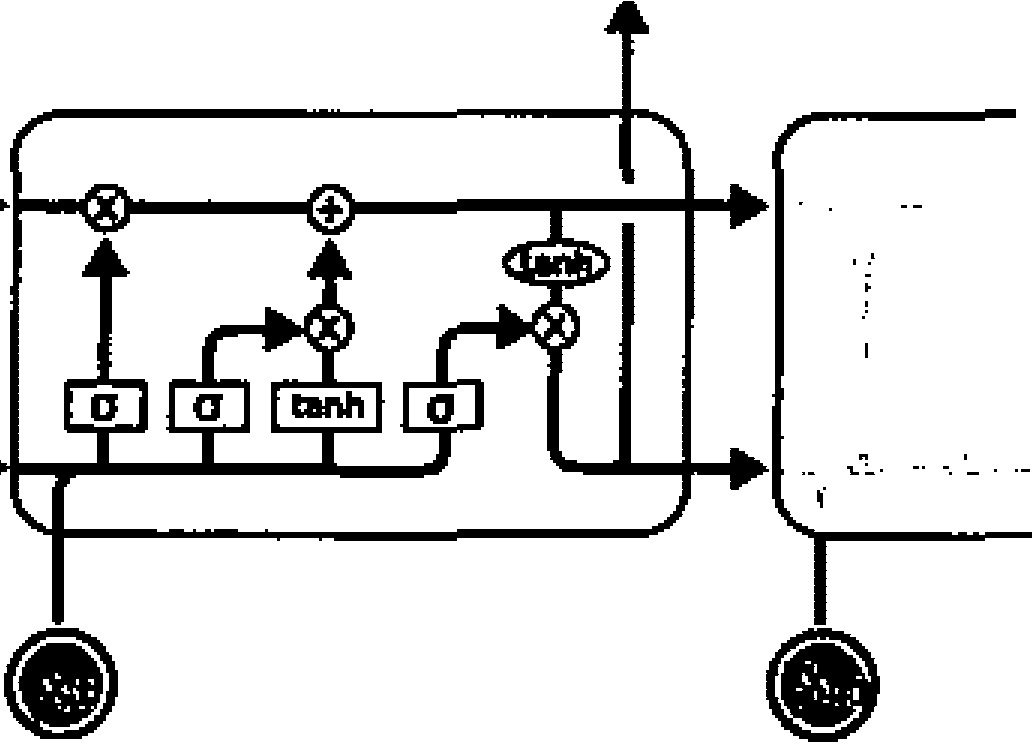
LSTM 在本質上和標車 RNN 是一樣的， 只不過 LSTM 內部的計算更加種 雜、書數更多、輸入和輸出的數目也更多 ，

圖 5.23 所示的就是 LSTM 的計算過程，在 P）晶rch 中呼叫也非常簡單，使用 nn. LSTM（） 即可，裡面的事數和標車 RNN 中的參數一模一樣，就不再贅述 ，下 面介紹 LSTM 與標單 阻啊 不同的地方。

＠ ＠ ＠



？ 是



在 1

＠

圖 5.23 LSTM 的計算過程

首先，LST 的書數比標車 RNN 多，但是存取的方式的然是相同的，使用 weight\_ih\_lO 即可，只是裡面的維度和標車 RNN 不再相間，它是標單 RNN 緯度 的 4 倍，因為 LSTM 中間比標單 E刑 多了三個騙性轉晶 ，多的三個輯性轉換的 加權拼在一起，所以一共是 4 倍，同理偏置也將是 4 倍。換句話說，LSTM 裡面 做了 4 個額做標車 間耐 所做的運算，所以參數個數是標車 郎恥 的 4 倍。

其次，LSTM 的輸入也不再只有序列輸入和體曬狀態 ，從圖 5.23 也可以看 出，隱曬狀態除了 ho 以外，還多了一個 旬 ，它們合在一起成為網路的隱藏狀態， 而且它們的大小完全一樣，就是 (layer \*direction, batch, hidden）， 當然輸出也會有 *ho* 菲:Q Co 0

下面定義一個和標車 RNN 輸入輸出相同的 LSTM 如下 ：

1 lstm = nn.LS叫﹛in!'ut\_,,,size=20, hidden,.,.,size 目50, num\_layers=2 ﹜

然後可以存取其中的書數，使用 weight\_ih\_lO 獲得如圖 5.24 所示的結果。

lstm.weight\_ih\_lO Pa:rameter containing:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 .0648 輛0.0786 -0 .1359 |  | 0.1099 | 0.0977 -0.1010 |
| -0.0857 -0.1410 -0 .0689 |  | -0.0929 | -0.0275 -0.0984 |
| -0.0641 0.1151 -0 .07日1 |  | -0.0754 | 0.1070 0.1384 |
| 0.0431 -0 .0﹔398 0.1186 | .•• | 0 .0470 | 0.1370 -0.0892 |
| -0.0578 -0.1063 0.1066 | • • . | -0 .0205 | -0.0217 0.1318 |
| -0.0685 0.1297 -0.0677 | • • • | 0.0851 | 0.1152 -0.0856 |

[torch.FloatTensor of size 200x20]

圖 5.24 LSTM 的計算結果

可以看到’事數的大小變成了 （50 x 4, 20）， 確實變成了標車 甜、凹 的 4 倍。 傳入輸入，這次沒有轉入攝藏狀態，那麼預設就會傳導入吾吾數全是 0 的隱藏

狀態 ho 和 旬，同樣可以獲得結果如下：

1. lstm\_out, (h＿；戶 c\_n﹜ E lstm ﹛ toy\_input﹜
2. print (lstm\_out .size O ﹜
3. print (h\_n.size （﹜ ﹜
4. print (c\_n.size （） ﹜

這樣就可以獲得它們的維度分別是 （ 100, 32, SO）、 （2 泣，SO） 和 位 32, SO）。

'.:lil 3.GRU

GRU 本質上和 LSTM 是一樣的，它的網路計算圖前面也實際介紹7，這裡簡 單介紹它與 LSTM 不同的地方。

首先它的隱藏狀態書數不再是標準即刑 的 4 倍，而是 3 倍，這是由於它內部 計算結構確定的。 同時網路的體聽狀態也不再是 ho 和 旬，而是只有 ho’這裡從網 路的計算圍中也能看出來，其餘部分和 LSTM 完全一樣，就不再贅述。

除此之外，PyTorch 中還提供了 RNNCell 、.LSTMCell 、GRUCell’這三個與 前面介紹的有什麼區別呢？

這三個背別是上面介紹的三個函數的單步版本， 也就是說它們的輸入不再是 一個序列，而是一個序列中的一步，也可以說是福單神躍網路的宿環， 在序列的

應用上更加靈活， 因為序列中每一步都是手動實現的， 能夠在基礎上增加更多自 訂的操作，這個部分就不再介紹，有興趣的讀者可以自行查閱文件。

5.3.2 實例介紹

前面的部分介紹了 心削、臼TM 和 GRU 的基本概念，以及在 PyTorch 中如 何呼叫，下面用兩個實例來熟悉整體的操作。

凶圖片分類

循環神經網路特別適用於序列資料， 那麼對於閻片類型的資料，是不是循環 神經網路就無辦法處理了呢 ？其實不是這樣的，仍然可以用循環神經網路進行圖 片分類，下面就對 MNIST 手寫數字進行分類。

首先需要將圖片資料轉化為一個序列資料， MNIST 手寫數字的圖片大小是 28 X 尬，那麼可以將每張圖片看作是長為 泌 的序列，序列中的每個元素的特徵維 度是 況，這樣就將圖片變成了一個序列。 同時考慮到循環神經網路的記憶性， 所 以圖片從左往右輸入網路的時候， 網路可以記憶住前面觀察到的東西， 也就是說 一張圖片雖然被切割成了 28 份，但是網路能夠透過記住前面的部分，同時和後面 的部分結合獲得最後預測數字的輸出結果， 所以從理論上而言是行得通的 。

下面介紹如何定義一個用於圖片分類的循環神經網路。

* 1. class Rnn (nn .Mcdule﹜ ：
  2. def init ﹛ self , in dim, hidden dim, n\_layer, n class) .
  3. super (Rnn, self ) . init II
  4. self .n\_layer - n\_layer
  5. self .hidden dim - hidden dim
  6. self lstm a nn LSTM ( in\_dim, hidden dim, n\_layer, batch\_frrst-True ﹜
  7. self .classifrer a nn .Linear (hidden dim, n class﹞ 8

1. def forward ﹝ self , x) :

。

1. J hO " Var斗able (torch zer s(self.n\_layer, x .size ﹝ 11 ,

|  |  |  |
| --- | --- | --- |
| 11 | # | self .hidcjen\_dim) ) .cuda ﹛） |
| 12 | # cO " Variable I 仁。rch .zer s ( | |
| 13 | # | self .hidden dim) I . cuda ﹛） |

self n layer, x size (l) ,

14 。ut, - self .lstm (x)

15 out －。ut ﹝ ，－1，：﹞

16 。ut - self classifier ﹛。u七）

17 return 。ut

上面的定義中網路主要由 LSTM 網路和線性網路組成， LSTM 網路接受圖 片序列，線性網路將它輸出成最後的機率 向量 。因為處理的是圖像資料， 常常是 batch 放在前面，所以在 LSTM 的定義中 ，使用了 batch first-True ＇這樣網路的輸 出也是 batch 在前面。另外在 forward 要注意一個細節， out=out﹝，l：﹞’這是因為 循環神經網路的輸出也是一個序列， 這一行程式是取出輸出序列中 的最後一個， 在應用線性層作為最後的輸出結果。

經過上面的介紹， 整個過程就變得清晰了，而網路的整個訓練過程跟之前一 模一樣，就不再贅述。最後訓練 20 次，獲得如圖 5.25 所示的結果。

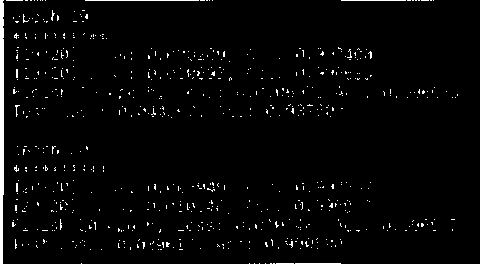


圖 5.25 訓練結果

可以看到在 MNIST 資料集上能夠達到 99% 的驗證集準確率，應該是比較高 了。雖然在一個簡單的圖片資料集上能夠達到相對滿意的效果， 但是循環神經網 路還是不適合處理圖片類型的資料：

﹒第一個原因是圖片並沒有很強的序列關係’圖片中的資訊可以從左往右看， 也可以從右往左看， 甚至可以跳著隨機看，不管是什麼樣的方式都能夠完整 地了解圖片資訊 ﹔

﹒第二個原因是循環神經網路傳遞的時候， 必須前面一個資料計算結束才能進 行後面一個資料的計算，這對於大圖片而言是很慢的，但是旋積神經網路並 不需要這樣，因為它能夠平行，在每一層旋積中 ，並不需要等待第一個旋積 做完才能做第二個旋積，整體是可以同時進行的 。

------

第 5 章

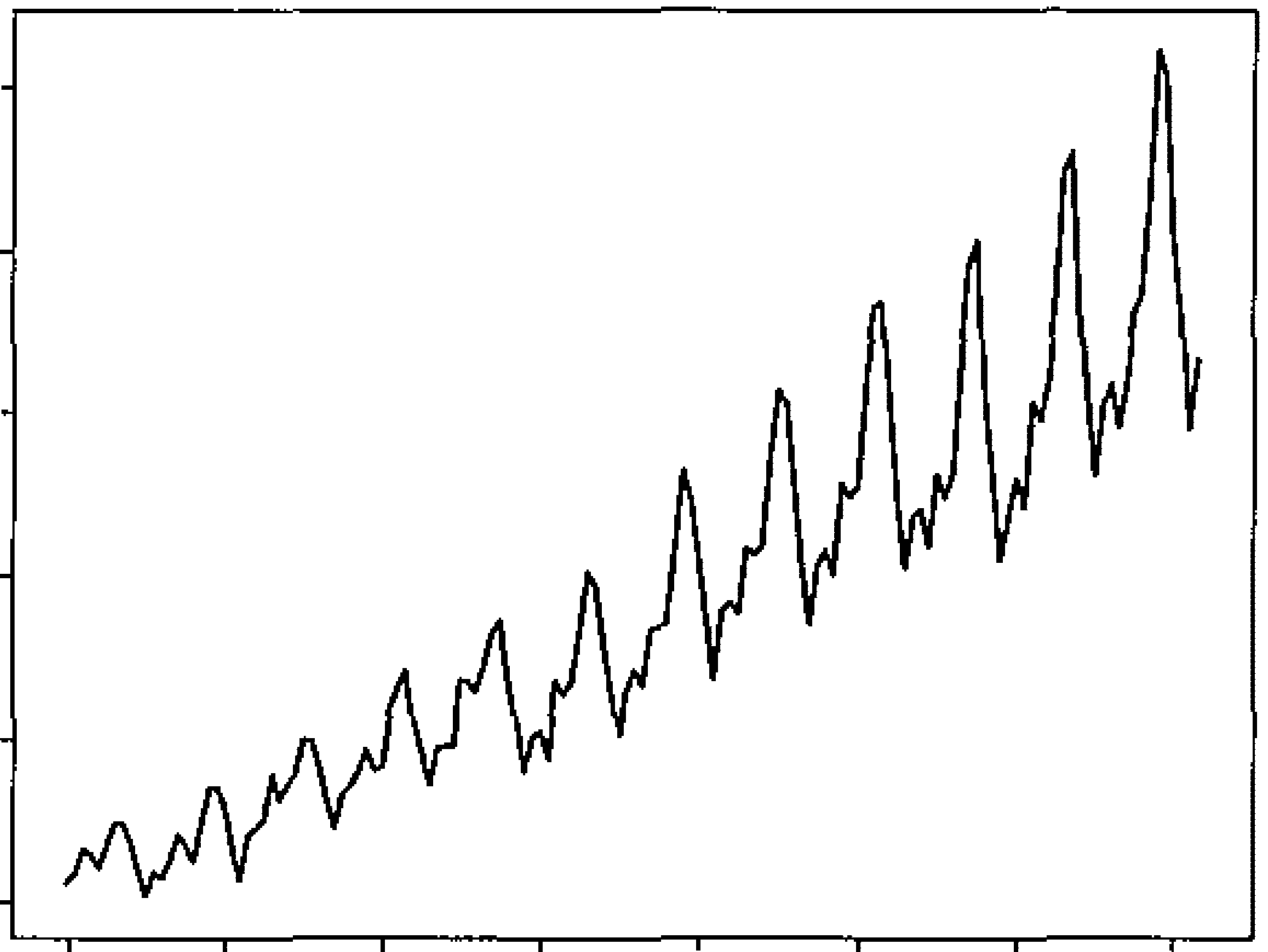
幅最神經網路

下面就來介紹福環神經網路真正適用的場景一一序列賈棚。

對序列資料而言， 因為它具有時序性，即前面的資料對接面資料有影響， 所 以LSTM 的記憶性能鉤適用於這種場景。

讀取的資料是 2010 年的飛機月流量，視覺化龍獲得如圖 5.26 所示的結果。

E口口



5口。

400

3口口

200

100

。 20 40

區。 BO

100

120

140

圖 5.26 2010 年的飛揖月流量

首先需要將資料標準化到 0 ～ I 之間，這是前置處理的標單步驟，接著開始 建立資料集，因為列出的是純資料，沒有任何特擻，所以希望使用前聽個月 的流 量預制目前月的流量，所以可以建立以下資料集：

def create dataset (datase七 look hack•尊2﹜： dataX, dataY ＝ ﹝﹞ ，﹝﹞

可占l

呵，－ qJ

for i in range (len ﹛dataset) - look\_back﹜： a = dataset ﹝ i: ( i + look back﹜ ﹞

A－－

dataX .append ﹝a﹜

RJ

dataY.append ﹛ dataset ﹝ i + look\_back﹞ ﹜

EV門，

return np.array (dataX) , np.array (dataY﹜

5.3 循環神經網路的 PyTorch 實現 圓圓－－－

這裡使用前兩個月的流量資料預測目前月的流量 ，也可以設定為前三個月。 然後需要將資料分為訓練集和測試集， 這裡簡單地將前面幾年的資料作為訓練 集，後面兩年的資料作為測試集。 接著需要再處理資料， 因為網路讀取的資料維 度是 （ s呵，batch, input）， 所以重新排列資料就可以了 ，這裡的 batch 是 l，因為只 有一個序列，input 就是預測依據的月份數， 這裡設定為 2 ，序列長度就是前面剖 分好的訓練集的序列長度。

按著建立簡單的循環神經網路， 這裡使用非常流行的 LSTM :

1. class lstm ﹛ nn .M。dule)
2. def init (self , input\_size-2, hidden\_size-4 ，。utput\_size- 1,
3. num\_layer 旦2J ,
4. super (lstm, self ) . init II
5. self .layerl = nn LS啞1(input\_size, h斗dden\_size, num﹜layer)
6. self .layer2 - nn Linear ﹛ hidden\_size ’。utput\_s 立 ze﹜

7

1. def forward (self , x﹜ ：
2. x, - self layerl ﹝ x) # seq, batch, hidden
3. s, b, h - x .size ()

11 x - x .view﹛ s \* b, hi

12 x = self.layer2 ﹛ x)

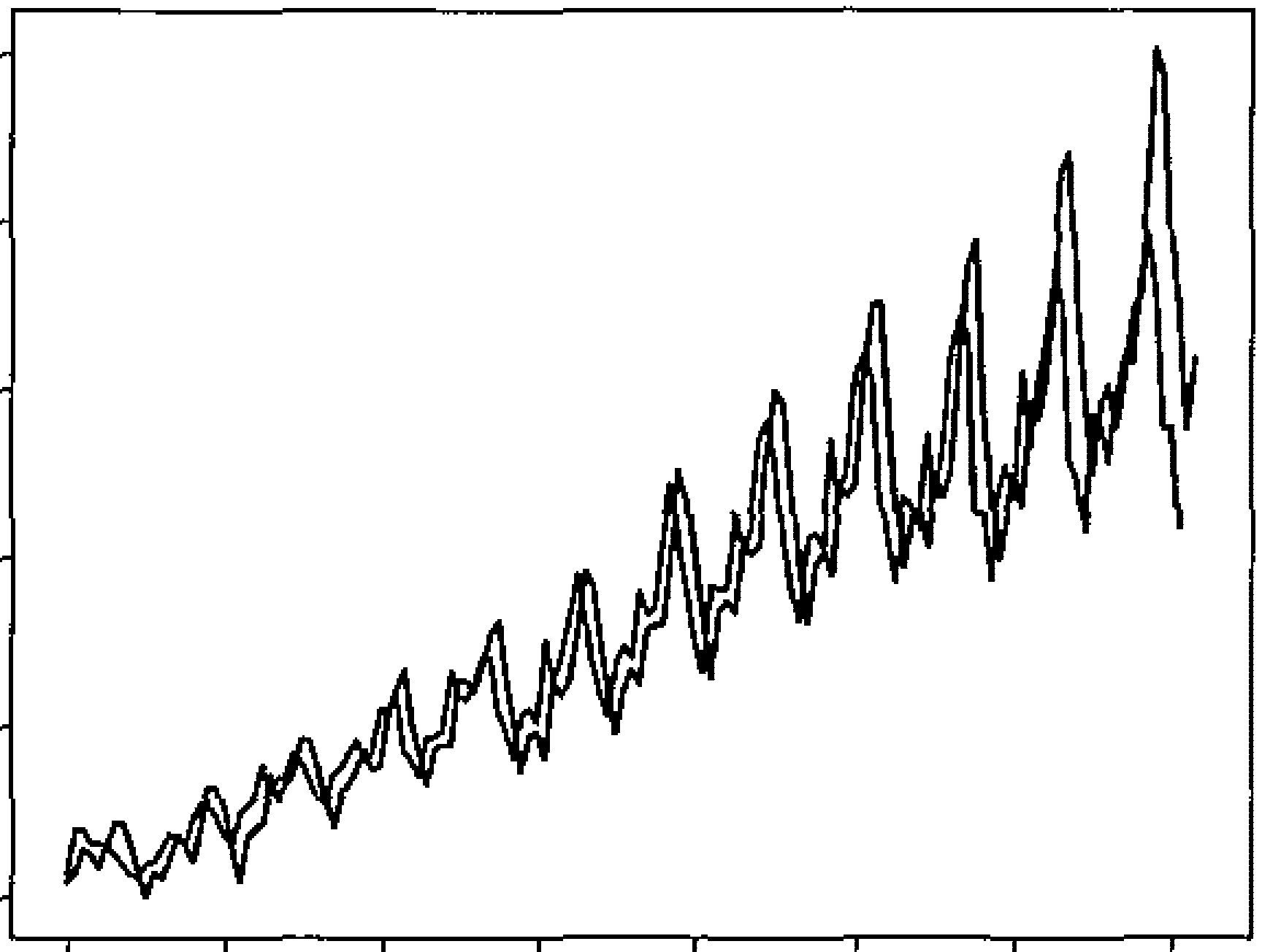
13 X - X V工ew ﹛ s, b, 1)

14 return x

網路的定義特別簡單，有兩層：一層是 LSTM ，一層是線性層， 主要介紹向 前傳播的部分。LSTM 會傳回隱藏狀態，而我們並不需要隱藏狀態 ，所以可以不 儲存這個狀態， 按著需要使用 view 來重新排列，因為 nn.Linear 不接受 3D 的輸 入，只接受兩維的輸入 ，所以我們可以將前面兩維先合併到一起， 然後經過線性 層之後再把它們分闊， 最後輸出結果。 另外，網路輸入的維度是根據前面建立資 料來確定的，建立資料時依賴前面兩個月的流量來預測第三個月 的流量，那麼輸 入的維數就是 2 • LSTM 網路的輸出可以任意定義， 和後面的線性層轉換即可。

最後使用 Adam 訓練 100 次，獲得了如圖 5.27 所示的結果。

1.2



。 20

40

60

80

100 uo

140

l.O

0.8

日E

口 4

0.2

圖 5.27 Adam 割 蟬 100 次的結果

其中藍色是真實的資料，紅色是頭測的資料，可以看到使用 LSTM 能夠獲得 相對近似的預測結果， 如果只使用騙性回歸， 並不能達到比較理想的效果， 這個 實倒也說明了循環神經網路相比序列資料效能更好。

I 5.4 I 自然語言處理的應用

前面的部分實陳介紹了如何在 PyTorch 中呼叫價環神經網路， 同時列舉了兩 個實倒：一個是 MNIST 手寫數字分額，另一個是飛機流量的時間序列，還過這兩 個實側說明循環神經網路的簡單應用， 但是循環神經網路目前在自然語言處理中 應用最為火熱，所以這一小節將介紹自然語言處理中如何使用循環神鱷網路。

5.4.1 詞嚴入

首先介紹自然語言處理中的第一個概念一一詞攝入 （ word embedding ）， 也可 以稱為詞向量。

5.4 自然語言處理的應用 －

影像分類問題會使用 one-hot 編碼，例如一共有五種，那麼屬於第兩種的話， 它的編碼就是 （0, I , 0, 0, 0）， 對於分類問題，這樣當然特別簡明。但是在自然語言 處理中，因為單字的數目過多，這樣做就行不通了，例如有 10000 個不同的詞， 那麼使用 one-hot 這樣的方式來定義，效率就特別低，每個單字都是 10000 維的向 量，其中只有一位是 1，其餘都是 0 ，特別佔用記憶體。除此之外，也不能表現單 字的詞性，因為每一個學字都是 one-hot ，雖然有些單字在語義上會更加接近， 但 是 one-hot 沒辦法表現這個特點， 所以必須使用另外一種方式定 義每一個單字，這 就引出了詞嚴入。

詞餃入到底是什麼意思呢 ？其實很簡單，對於每個詞，可以使用一個高維向 量去表示它，這裡的高維向量和 one-hot 的區別在於，這個向量不再是 0 和 1的形 式，向量的每一位都是一些實數， 而這些實數隱含著這個單字的某種屬性。 這樣 解釋可能不太直觀， 先舉四個實例，下面有 4 段話：

( I ) The cat likes playing ball.

( 2 ) The kitty likes playing wool. ( 3 ) The dog likes playing ball.

( 4 ) The boy likes playing ball.

重點分析裡面的 4 個詞，cat 、kit句、dog 和 boy o 如果使用 one-hot’那麼 cat 就可以表示成 (l, 0, 0, 0) • kitty 就可以表示成 （ 0, I , 0, 0）， 但是 cat 和 kitty 其實都 表示小貓，所以這兩個詞語義是接近的，但是 one hot 並不能表現這個特點。

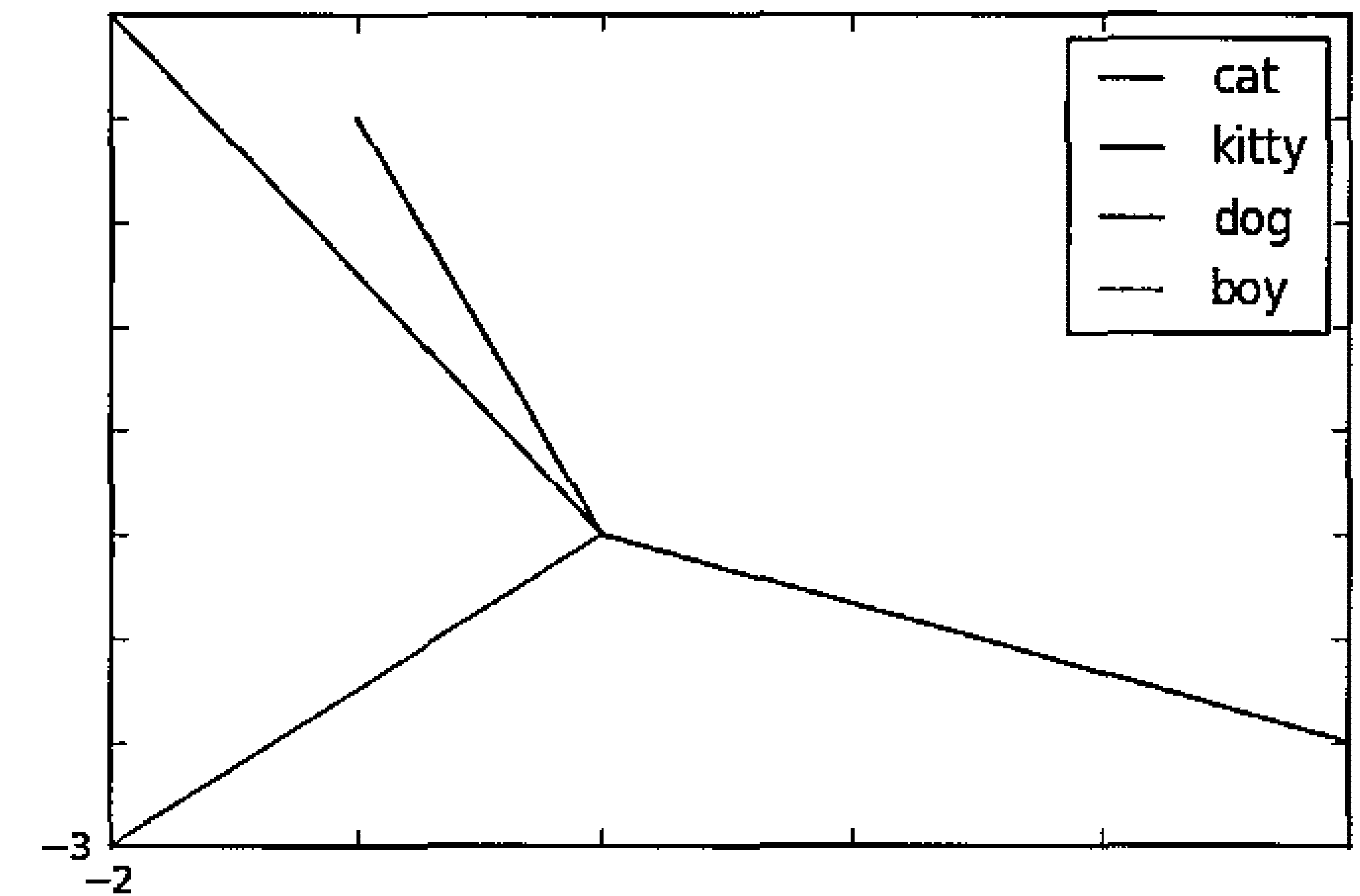
下面使用詞餃入的方式來表示這 4 個詞，假如使用一個二維 向量 低 b） 來表示 一個詞，其中 a b 分別代表這個詞的一種屬性，例如 a 代表是否喜歡玩球， b 代 表是否喜歡玩毛線， 並且這個數值越大表示越喜歡， 這樣就能夠定義每一個詞的 詞餃入，並且透過這個來區分語義， 下面來解釋一下原因。

對於 c前，可以定義它的詞般入是 抖，4）， 因為它不喜歡玩球，喜歡玩毛線﹔ 而對於 kitty’它的詞嚴入可以定義為 （－2, 5）﹔ 那麼對於 dog 它的詞依入就是 （ 3,

-2）， 因為它喜歡玩球，不喜歡玩毛線﹔最後對於 boy’它的詞向量就是 （－2, -3) •

因為這兩樣東西他都不喜歡。定義好了這樣的詞嚴入， 怎麼去定義它們之間的語 義相似度呢 ？可以透過詞向量之間的夾角來定義它們的相似度。 下面先將每個詞 向量都在座標系中表示出來，如圖 5.28 所示 。

5



4

3

2

l

。

-1

-2

-1 。 l 2 3

圖 5.28 不同詞向量的東角

圖 5.28 就顯示出了不同詞向量之間的英角， 可以發現 kitty 和 cat 的夾角更 小，所以它們更加相似的，而 dog 和 boy 之間夾角很大，所以它們不相蝕。

透過這樣一個簡單的實倒能掏看出詞竄入對於單字的表示具有很好的優勢， 但是問題來7，對於一個詞，怎麼知道如何去定義它的詞嚴入？ 如果向量的維數 只有 5 稚，可能還能定義出來， 如果向量的維數是 100 膛，那麼怎麼知道每一維 體是多少呢 ？

這個問題可以交給神經網路去解訣， 只需要定義我們想要的緯度， 倒如 100 雄，神經網路就會自己去更新每個詞嚴入中 的元素。而之前介紹過詞嚴λ的每個 元素表示一種屬性， 當然對於維數比較低的時帳， 可能我們能夠推斷出每一推實 際的屬性含義，然而緯度比較高之後， 我們並不需要關心每一推到底代表著什 麼含義，因為每一維都是網路自己學習出來的屬性， 只需要知道詞向量的夾角越 小，表示它們之間的語義更加接近就可以了。 這有重日提積網路會對一張圖片分析

5.4 自然語言處理的應用 一

出很厚的特徵園，並不需要關心網路分析出來的特徵到底是什麼， 只需要知道抽 象的特徵能夠幫助我們分類影像就可 以了。

5.4.2 詞骰入的 PyTorch 實現

詞飯入在 PyTorch 中是如何實現的呢 ？下面來實作 *15*式一下。

PyTorch 中的詞般入是透過函數 on.Embedding徊，n） 來實現的，其中 m 表示 所有的單字數日 ，n表示詞般入的維度，下面舉一個實例：

w。rd\_to\_ix a ﹛ T hello ' , 0’，w。rld ' , 11

1 2 3

err甘beds " nn Embedding ( 2, 5)

hello idx a t rch LongTens r ﹛ ﹝ word\_t。ix ﹝ hell ’﹞ ﹞ ﹜

。

hello idx " Var工able (hell idx} hell embed • e 曲eds (hell idx} print (hello embed﹜

。 。

5 6

4

上面就是輸出的 hello 的詞餃入，下面來解釋一下程式。 首先需要給每個單字 建立一個對應索引，這樣每個單字都可以用一個數字去表示，例如需要 hello 的時 候，就可以用 來表示，用這種方式，存取每個詞會特別方便。 按著是詞餃入的 定義 nn.Embedding(2, 5）， 如上面介紹過的，表示有兩個詞， 每個詞向量是 5 維，

也就是一個 2 × 5 的矩障，

只不過矩陣中的元素是可以被學習更新的，所以如果有 1000 個詞，每個詞向 重希望是 100 維，就可以這樣定義詞依入 m.Embedding(!000, I 00）。 存取每一個 詞的詞向量需要將 tensor 轉換成 Variable ，因為詞向量也是網路中更新的參數， 所以在計算園中 ，需要透過 Variable 去存取。另外這裡的詞向量只是初始的詞向 量，並沒有經過學習更新，需要建立神經網路最佳化更新， 修改詞向量裡面的參 數使得詞向量能夠表示不同的詞，且語義相近的詞能夠有更小 的夾角 。

以上介紹了詞餃入在 PyTorch 中是如何實現的，下一節將介紹詞般入是如 何 更新的，以及它如何結合 N Gram 語言模型進行預測。

－－－－－ 第 5章 循環神經網路

* + 1. N Gram 模型

首先介紹 NGram 模型的原理和它要解決的問 題。在一篇文章中，每一句話 都是由很多單字組成的， 而且這些單字的排列順序也是非常重要的。 在一句話 中 ，是否可以由前面幾個詞來預測這些詞後面的單字 ？例如在 r Ilived in France

for I O years, I can speak \_ .J 這句話中，我們希望能夠預測最後這個詞 是 French 0

知道想要解決的問題後，就可以引出N Gram 語言模型了。對於一句話 T ，它 由叫，*w,..·w.*這 n個詞組成，可以獲得下面的公式：

*P(T )=P(w1 )P(w2* I w1 )P(w3 Iw2w1 ).. *P(w"* I*w"* 1w" ,..·w2w1) (5 4)

但是這樣的模型存在著一些缺陷 ，例如參數空間過大，預測一個詞需要前面 所有的詞作為條件來計算條件機率， 所以在實際中沒辦法使用。為了解決這個問 題，引用了馬可夫假設，也就是說這個單字只與前面的幾個詞有關係’並不是和 前面所有的詞都有關係， 有了這個假設，就能夠在實際中使用 NGram 模型了。

對於這個條件機率， 傳統的方法是統計語料中每個單字出現的頻率， 據此來 估計這個條件機率， 這裡使用詞骰入的辦法，直接在語料中計算這個條件機率， 然後最大化條件機率進一步最佳化詞 向量，據此進行預測。

* + 1. 單字預測的 PyTorch 實現

首先列出一段文章作為訓練集

1. CONTEXT SIZE - 2
2. E>但EDDING DIM a 10

。

1. # We will use Shakespeare s nnet 2
2. test sentence ＝ ”，＇＂When f。rty winters shall besiege thy brow,

And dig deep trenches in thy beauty's field,

5

1. Thy y。uth ’s pr。ud livery s。gazed 。n n。w,
2. W工 11 be a totter ’d weed 。f sma ll w。rth held·
3. Then being asked, where all thy beauty lies,

。

1. Where all the treasure f thy lusty days﹔

。

1. T say, within thine own deep sunken eyes,
2. Were an all eating shame, and thrif tless praise

。

1. How much m re praise deserv' d thy beauty's use,
2. If thou couldst answer 'This fa工r child of mine 14 Shall sum my count, and make my ld excuse, '

。

。

1. Pr ving his beauty by succession thine '
2. This were t。be new made when th。u art 。ld,
3. And see thy bl。。d warm when thou feel’st it cold．”””.split ﹛）

CONTEXT SIZE 表示想由前面的幾個單字來預測這個單字， 這裡設定為 2 , 就是說我們希望透過這個單字的前兩個單字來預測這一個單字， EMBEDDING DIM 表示詞骰入的維數。

按著建立訓練集，檢查所有語料來建立， 將資料整理好，需要將單字分三個 組，每個組前兩個作為傳入的資料，而最後一個作為預測的結果。

1. trigr缸＂－ ﹝ ﹛ ﹝ test\_sentence ﹝ l﹞，test sentence ﹝ i+l﹞ I , test sentence ﹝ i+2﹞ ）
2. f。x 1 in range ( len ﹛ test sentence ﹜ 21 ]

將每個單 字編碼， 即用數字來表 示 每個學 字， 只有這樣才能夠傳入

nn.Embedding 變得詞向量 。

1. vocb a set ﹛ test sentence﹜ ＃ 遞過 set 將重複的單字去掉
2. w。rd t。斗dx a ﹛ w。rd: l. f。r 1’，ord in enumerate (vocb﹜ ﹜
3. idx\_t。w。rd a ﹛ w。rd\_t。＿idx ﹝ word ﹞ word for word 斗n w。rd\_t。＿1由毛﹜

然後可以定義 N Gram 模型如下：

1. class Ngra叫1odel Inn .Module﹜ ．

。 。

1. def init ﹛ self , *v* cb size, c ntext\_size, n\_d斗m﹜ －
2. supe主 ﹛ Ngrn訟4。del, self ) init II
3. self.n w。rd a vocb size
4. self embedding a nn Embedding (self n\_w。rd, n\_d工m)
5. self .hnearl - nn .Linear ( c。ntext size•n dim, 128)
6. self .linear2 - nn .Linear (128, self n一 rd﹜
7. def f。rward (self , x﹜ －

8

1. emb a self .embedding (x﹜
2. emb a emb view ( l, -1)

。

1. ut a self linearl (emb)
2. 。ut a F.relu ﹝。ut)
3. 。ut a self linear2 ﹛。ut)
4. 1。g prob a F l。q s。f tmax （。ut)
5. return l。g prob

一 第 5 章

循環神經網路

模型需要傳入的參數有三個， 分別是所有的單字數、 預測單字所依賴的單 字數、即 CONTEXT SIZE f日詞向量的維度。網路在向前傳播中，首先傳入單字 獲得詞向量，模型是根據前面兩個詞預測第三個詞的， 所以需要傳入兩個詞 ，獲 得的詞向量是 （2, 100）， 然後將詞向量展開成 (I, 200）， 按著經過線性轉換， 經過

relu 毆動函數，再經過一個線性轉換， 輸出的維數是單字總數，最後經過一個 log softmax 敢動函數獲得機率分佈， 最大化條件機率，可以用下面的公式表示 ：

*-/og( p( w1* I *C ))= -log* （ 崢*max( A( Lw ee, )* + b))

(5.5)

在網路的訓練中，不僅會更新線性層的參數， 還會更新詞餃入中的參數，訓 練 100 次模型，可以發現 loss 已經降到了 37 ，也可以透過預測來檢測模型是否 有效：

w。rd, label trigram ﹝ 3﹞

1 2 3 4 5 6

w。rd Variable ﹛t。rch .L。ngTensor ﹛ ﹝ word t。idx ﹝ i﹞ f。r i in word﹞ ） ）

。ut ngr前間 del (word)

一，predict\_label 七 rch .max ﹛ ut, 1)

predict\_w 。rd 巨 idx\_t ＿w。rd ﹝ predict label data ﹝ O﹞ ﹝ O﹞ ﹞

print ﹛ a real wcrd is I ﹜ pred斗ct w。rd is ﹛ ） ＇format ﹛ label, predict w。rd) I

執行上面的程式，可以發現其實的單字跟預測的單字都是一樣的， 雖然這是 在訓練集上，但是在某種程度上也說明這個小模型能夠處理 N Gram 模型的問題。

上面介紹了如何透過最簡單的單邊 N Gram 模型預測單字，還有一種複雜一點 的N Gram 模型透過雙邊的單字來預測中間的單字，這種模型有個專門的名字，叫 Continuous Bag of-Words model ( CBOW ）， 實際內容差別不大，就不再贅述。

5.4.5 詞性判斷

上面只使用了詞般入和 N Gram 模型進行自然語言處理， 還沒有真正使用循 環神經網路，下面介紹 RNN 在自然語言處理中的應用。在這個實例中 ，我們將使 用 LSTM 做詞性判斷， 因為同一個單字具有不同的詞性， 例如 book 可以表示名 詞，也可以表示動詞，所以需要結合前後文列出實際的判斷。 先介紹使用凶TM 做詞性判斷的原理。

油 1. 基本原理

定義好一個 LSTM 網路，然後列出一個由很多個詞組成的句子，根據前面的 內容，每個詞可以用一個詞向 量表示，這樣一句話就可以視為是一個序列 ，序列 中的每個元素都是一個高維向 量，將這個序列傳入 LSTM’可以獲得與序列等長 的輸出，每個諭出都表示為對詞性的判斷， 例如名詞、動詞等。從本質上看，這 是一個分類問題，雖然使用了 LSTM ，但實際上是根據這個詞前面的一些詞來對 它進行分類，看它是屬於幾種詞性 中的哪一種。

思考一下為什麼 LSTM 在這個問題裡面起注重要的作用。 如果完全孤立地對 一個詞做詞性的判斷， 常常無法獲得比較準確的結果， 但是透過 LSTM，根據它 記憶的特性，就能夠透過這個單字前面記億的一些詞語來對它做一個判斷， 例如 前面的單字如果是 my’那麼它緊接的詞很有可能就是一個名詞， 這樣就能夠充分 地利用上文來處理這個問 題。

油 2. 字元增強

還可以透過引用字元來增強表達，這是什麼意思呢 ？就是說一些單字存在著 字首或副檔名，例如 句這種副檔名很可能是一個副詞， 這樣我們就能夠在字元水 平上對詞性進行進一步判斷， 把兩種方法整合起來， 能夠獲得一個更好的結果。

在實現上還是用 LSTM ，只是這次不再將句子作為一個序列， 而是將每個單 字作為一個序列。每個單字由不同的字母組成，例如 apple 由a p p 1 e 組成，給這 些字元建立詞向量，形成了一個長度為 5 的序列，將它傳入 LSTM 網路，只取最 後輸出的狀態層作為它的一種字元表達， 不需要關心分析出來的字元表達到底是 什麼樣，它身為抽象的特徵，能夠更進一步地預測結果。

* + 1. 詞性判斷的 PyTorch 實現

作為示範 ，使用一個簡單的訓練資料，下面有兩句話，每句話中的每個詞都 列出了詞性：

一 一 第 5 章 循環神經網路

* + - 1. training\_data － ﹝
      2. ﹛”The d。q ate the apple”spl斗℃（） ，﹝”DET ”，”NN ”，”V ＂ ”DET ””NN ”﹞ ） ，
      3. ﹛”Everybody read that ba 。k".split ll，﹝”NN ””V ＂ ”DET ＂ ”NN ”﹞ ）

4

按著對單字和詞性由 a 到 z 的字元進行編嗎 ：

1. w。rd ta 斗dx － ﹛ ﹜
2. tag ta i也是 ＝ ﹛ l
3. far context, tag in training\_data ,
4. f r w rd in c ntext ﹒
5. if w rd n t in w。rd t idx.
6. ward t idx ﹝ ward﹞ a len (w。rd t idx﹜
7. f，z label in tag,
8. if label n t in tag\_t 一立dx
9. tag\_t ＿idx ﹝ label] a len ( tag\_t ＿idx) 10

。

1. alphabet ＝’abcdefghi jklmn pqrstuvwxyz ’
2. character t。工dx a ﹛ ﹜
3. f，r i in range ( len ﹛alphabet﹜ ） ．
4. character\_t ＿idx ﹝ a工phabet ﹝ L﹞ ﹞ a i

接著先定義字元水準上的 LSTM，定義方式和之前類似 ：

1. class CharLSTM ﹝ nn .Module) ,
2. def init (self , n﹜char, char dim, char\_hidden ﹜：
3. super (CharLSTM, self ) ini七 ﹛﹜
4. self .char\_embedding a nn.Embedding (n\_char, char\_dim﹜
5. self char lstm a nn .LSTM (char dim, char hidden, batch first-True)
6. def f。rward ﹛ self , x﹜：

6

1. X a Self.ch缸 自由eddi呵（x)
2. , h a self char lstm (X)
3. return h ﹝ O﹞

定義兩層結構：第一層是詞絞入，第二層是臼TM 。在網路的正向傳播中 ， 先將單字的 n 個字元傳入網路， 再透過 nn.Embedding 獲得詞向量 ，接著傳入 LSTM 網路，獲得隱藏狀態輸出 h ，然後透過 h﹝O﹞ 獲得想要的輸出狀態 。對於每 個單字，都可以透過 CharLSTM 用對應的字元表示。

接著完成 目標，分析每個單字的詞性，首先定義好詞性的 LSTM 網路：

1. class LSTMTagger ﹛nn.M。dule) ,
2. def init ﹛ self , n\_word, n char, char dim, n dim, char\_hidden,
3. n\_hidden, n tag﹜

*4* super (LSTMTagger, self ) init I)

。

1. self.word embedding - nn .Embedding (n w rd, n dim﹞
2. self.char lstm - CharLSTM ﹛n char, char\_d1m, char\_hidden)
3. self lstm - nn LSTM ﹝n dim+char hidden, n hidden, batch first-True)
4. self .linearl - nn.Linear (n hidden, n tag﹜
5. def f。rward (self , x, word data﹜ －

9

1. w。主d a ﹝ 1 f。r i in w。rd data﹞
2. char - t。rch .Fl。atTens。r ﹛）
3. f。r each in w。rd·

14 w。rd 11.s乞 ＝ ﹝﹞

1. f。r letter in each
2. word list .a即end (character\_to\_idx ﹝ letter .1。wer ﹛） ﹞ ）
3. w。rd list a t。rch .LongTensor ﹛w 。rd 11.s乞﹜
4. w。rd\_list - w。rd list unsqueeze ( 0)
5. tempchar - self char\_lstm (Variable ﹛w。rd list ) .cuda I ﹜ ）
6. tempchar - tempchar . squeeze (0)

。

1. char - t rch cat （ ﹛ char, tempchar cpu () data) , 0)
2. char - char .squeeze (1)
3. char - Variable ﹛ char) .cuda ﹛）

*24* x - self word\_embedding ﹛ x)

。

1. X - t rch .cat ﹝ Ix, char) , 1)
2. x a x .unsqueeze ( O ﹜
3. x, " self lstm ﹛x﹜
4. x a x.squeeze (O )
5. x " self .linearl﹛ x)

。。

1. y a F .1 q s ftmax ﹛x﹜
2. return y

看著有點複雜，慢慢來介紹。首先使用 n word 和 n dim 定義單字的詞向量 矩陣的維度，n char f日char dim 定義字元的詞向量給度，char hidden 表示字元水

準上的 LSTM 輸出的維度，n hidden 表示每個單字作為序列輸入 LSTM 的輸出維 度，最後 n一tag 表示輸出的詞性分類。 介紹完裡面參數的含義， 下面實際介紹其 中網路的向前傳播。

學習過 PyTorch 的動態圖結構，網路的向前傳播就非常簡單了。 因為要使用 字元增強，所以在傳入一個句子作為序列的同時， 還需要傳入句子中 的單字，用 word da旭 表示。動態圖結構使得正向傳播 中可以使用 for 循環將每個單字都傳入 CharLSTM，獲得的結果和單字的詞向量拼在 起作為新的序列輸入，將它傳入 LSTM 中 ，最後接一個全連接層，將輸出維數定義為詞性的 數目。

這是基本的想法，就不實際解釋每句話的含義了 ，只是要注意程式裡面有 unsqueeze 和 squeeze 的操作，原因前面介紹過，LSTM 的輸入要帶上 batch size • 所以需要將維度擴大。

網路訓練經過了 300 次，loss 降到了 0.16 左右。為了驗證模型的準確性， 可以預測 「 Everybody ate the apple 」 這句話中每個詞的詞性， 一共有三種詞： DET 、NN 、v 。最後獲得的結果如 闊 5.29 所示。

結果是一個 4 行 3 列的向量 ，每一行表示一個單字， 每一列表示一種詞性， 從 左到右的詞性分別是 DET 、NN 、v 。從每行裡面取最大值 ，那麼第一個詞的詞性就 是 NN ，第二個詞是 v，第三個詞是 DET，第四個詞是向4 ，與想要的結果相符。

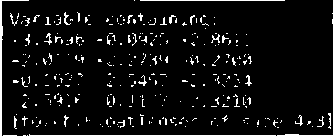


圖 5.29 網路訓練結果 以上，透過幾個簡單的實例介紹了循環神經網路在自 然語言處理中的應用 ，

當然真正的應用會更多， 同時也更加複雜，這裡就不再深入介紹了，對自然語言 處理有興趣的讀者可以 進行更深入地深入。

J 5.5 I 循環神經網路的更多應用

前面介紹了循環神經網路在時間序列資料和 自然語言處理中的簡單應用，循 環神經網路的應用不止於此，下面簡單地介紹現實中循環神經網路更多的應用。

5.5 衝環神經網路的更多應用 一

* + 1. Many to one

循環神經網路不僅能夠輸入序列、 輸出序列、還能夠輸入序列， 輸出單一向 量。只需要在輸出的序列裡面取其中 的就可以，通常是取最後一個。 這樣的結構 被稱為 Many to O間 ，那麼這種結構能夠做什麼工作呢 ？

第一個工作是情成分析， 將一句話作為序列輸入網路， 輸出只取最後一個， 根據輸出判斷這句話的態度是消極的還是積極的﹔ 第二個工作是關鍵字分析， 原 理也是一樣的， 將一句話作為序列輸入網路， 輸出只取最後一個， 不同的是用它 來表示這句話的關鍵字。 實際的結構可以周圍 5.30 所示。



國一團一圈一 一國一國一

圖 5.30

Many t。。ne

* + 1. Many to Many ( shorter )

第二種結構就是輸入和輸出都是序列， 但是輸出的序列比輸入的序列短。這 種類型的結構通常會在語音辨誡中 遇到，因為一段話如果用語音表達常常會比這 段話更長，可以看看圖 5.31 。

園摺盟國盟國體圖

o

d

．．

間

．

．

＋1

．

．

↑

＋

l

．－

．．

l

f

．

．

↑

．

．

＋l

．

．

＋

l

1

u

nv

n

圖 5.31 Many to Many

e

pu

凹

間

．

如

咿

．

州

— W

M門

L

Lf

叮

喇

u

吋

4

這種情況需要使用 CTC 演算法解決重複的問題，CTC 就是將輸出的所有可能 列列出來，然後透過去重複、去空格的方式來選擇最大的機率，這裡就不再贅述。

* + 1. Seq2seq

第三種情況是輸出的長度不確定， 這種情況一般是在機器翻譯的工作中出 現，將一句中文翻譯成英文，那麼這句英文的長度有可能會比中文短，也有可能 會比中文長，所以這時候輸出的長度就不確定了， 需要用序列到序列的模型來解 決這個問題，實際如圖 5.32 所示。

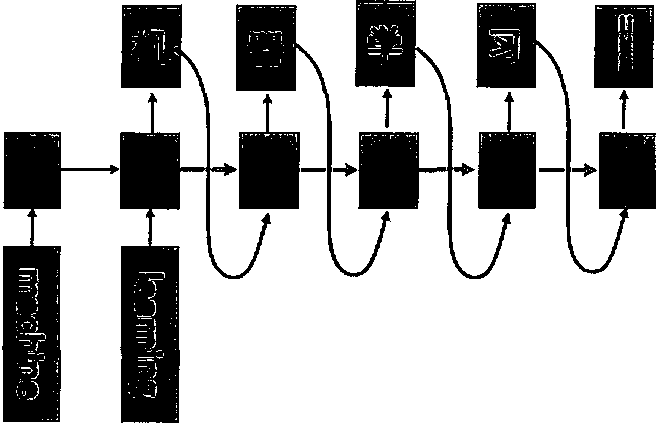


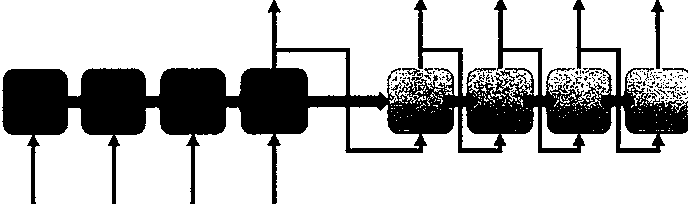
圖 5.32 序列到序列模型

在網路結構中 ，輸入一個英文序列，然後輸出它對應的中文爾譯，輸出的部 分透過前面的結果預測後面，根據上面的實例，也就是先輸出 「 機」’將它作為下 一次的翰入，接著輸出 「器」 這樣就能夠輸出任意長的序列。

聊天機器人和問答系統也都是同樣的原理， 將句子編入，輸出是根據前面的 輸入來得到，如圖 5.33 所示。

5.5 循環神經網路的更多應用 圓

w am fine <E。L>



How a『e *yoo* <E ＂抄

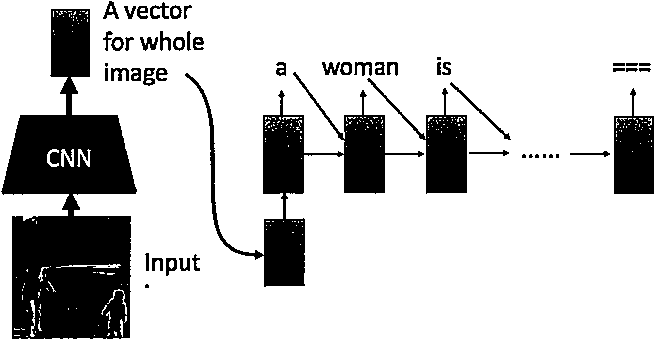
LSTM Encoder LSTM Dec。der

圖 5.33 聊天機器人原理

除此之外還可以引用注意力機制強化模型效果， 這裡就不再贅述， 本書的實 戰部分會詳細地介紹機器翻譯和 注意力機制。

* + 1. CNN+RNN

恥們 早日 CNN 還能夠聯合在一起完成影像描述工作， 簡而言之，就是透過預 訓練的旋積神經網路分析圖片特徵，接著透過循環網路將特徵變成文字描述， 實 際細節就不再多作說明了， 如圖 5.34 所示。



Input

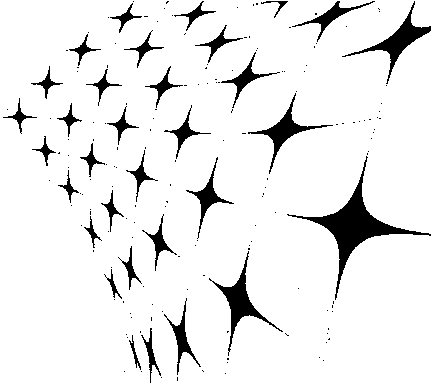
image

圖 5.34 RNN 與 CNN 共同完成影像描述工作

本章首先介紹循墳前申經網路的基礎，其次介紹流行的網路結構： LSTM 和 GRU’接著用 PyTorch 實現了幾個實例，然後介紹循環神經網路是如何應用於自 然語言處理的，最後簡單介紹循環神經網路在 現實中的應用工作。下一章將介紹 時下非常流行的產生對抗網路。

M E M 日

第 6 章



產生對抗網路

長久以來，人們都希望機器能夠充滿創造力， 不僅能完成重複的機械工作， 還能完成一些創造性的工作，例如畫畫、寫詩、創作歌詞等。這些一直是人工智 慧長久以來的夢想 ，隨著自動編碼器和變分編碼器的提出，這一夢想慢慢變成了 現實，到了 2014 年，Ian Goodfellow 提出了產生對抗網路 （ Generative Adversarial Networks, GANs ） 這一概念，推進了整個無監督學習的發展處理程序。

這一章將從最簡單的產生模型 （ Generative Model ） 入手，包含自動編碼器 ( Autoen- coder ） 和變分自動編碼器 （ Variational AutoEncoder, VAE ）， 接著詳細 地介紹產生對抗網路的創新和原理，以及為何產生對抗網路能夠成為現在最熱門 的研究領域。最後介紹產生對抗網路的變式和應用， 以及用 PyTorch 實現簡單的 產生對抗網路。

I &.1 I 產生模型

產生模型 （ Generative Model ） 這一概念屬於機率統計和機器學習 ，是指一 系列用於隨機產生可觀測資料的模型。 簡而言之 ，就是 「 產生 」 的樣本和 「 真 實 」 的樣本盡可能地相似 。產生模型的兩個主要功能就是學習一個機率分佈 Pmodel(X） 和產生資料，這是非常重要的，不僅能夠用在無監督學習 中 ，還可以 用在監督學習中 。

前面已經介紹過了適用於影像的旋積神經網路、 適用於序列的循環神經網 路，但是要知道 Lecun 提出第一代旋積神經網路 L閉目的時間是 1998 年，而循環 神經網路提出的時間更旱，是在 1986 年。這些網路在當時並沒有流行起來， 如今 隨著運算能力的加強 ，資料集的增多，深度學習逐漸流行了起來， 隨著越來越多

人的研究，各種各樣的神經網路都在不斷進步， CN!可 裡面出現了 inception net 、 resnet 等，心刑 演變了 LSTM f日 GRU 等不同的長短時間記憶的網路， 雖然神經 網路在不斷發展，但本質上仍然是在 CNN 和 RNN 的基礎上做著監督學習的工 作，無監督學習的發展一直 比較緩慢，產生模型希望能夠讓無監督學習 取得比較 大的進展。

6.1.1 自動編碼器

自動編碼器 （ AutoEncoder ） 最開始身為資料的壓縮方法，其特點有

( I） 跟資料相關程度很高，這表示自動編碼器只能壓縮與訓練資料相似的資料， 因為使用神經網路分析的特徵一般是高 度相關於原始的訓練集 ，使用人臉訓 練出來的自動編碼器在壓縮自然界動物的圖片時表現就會比較差， 因為它只 學習到了人峻的特徵，而沒有學習到自然界圖片的特徵。

( 2 ） 壓縮後資料是有損的， 這是因為在降維的過程中不可避免地要遺失資訊。 到 了 2012 年，人們發現在旋積神經網路中使用自動編磚器做逐層預訓練可以訓 練更深層的網路，但是人們很快發現， 良好的初始化策略要 比複雜的逐層預 訓練有效得多，2014 年出現的 Batch Normalization 技術也使得更深的網路能 夠被有效訓練，到了 2015 年年底，透過殘差 （ R臼Net ） 基本可以訓練任意深 度的神經網路。

所以現在自動編碼器主要應用在兩個方面： 第一是資料去噪，第二是進行l視 覺化降維。自動編碼器還有一個功能， 即產生資料。

首先列出自動編碼器的一般結構，如圖 6.1 所示。

As close as possible

3島 國惜 圈，其

圖 6.1 自動編碼器結構

6.1 產生模型 －

從圓 6.1 中能夠看到兩個部分﹒第一個部分是編碼器 （ Encoder ）， 第二個部分 是解碼器 （ Decoder ）， 編碼器和解碼器都可以是任意的模型， 通常使用神經網路 模型作為編碼器和解碼器。輸入的資料經過神經網路降維到一個編 喝 （ code ）， 按 著又透過另外一個神經網路去解碼獲得一個與輸入原資料一模一樣的產生資料， 然後透過比較這兩個資料，最小化它們之間的差異來訓練這個網路中 編喝器和解 碼器的參數。當這個過程

訓練完之後，拿出這個解碼器， 隨機傳入一個編喝 （ code ）， 透過解碼器能夠 產生一個和原資料差不多的資料，圖 6.2 就是希望能夠產生一張差不多的圖片。

：您 sgc";de ate 盟 圈 I mage ,

圖 6.2 編碼與解碼

這件事情能不能實現呢 ？其實是可以的，下面會用 PyTorch 來簡單地實現一 個自動編碼器。

首先建置一個簡單的多層全 連接網路。

對 MNIST 資料裡面的大小做一下轉換， 使用 transforms.Normalize （） 使得圖 片的大小變為 1～ l 之間，這是為了輸入變成一個比較對稱的分佈’訓練更加容 易收斂。按著透過 DataLoader 產生一個讀取圖片的反覆運算器。

* + - 1. class aut。enc。der ﹝ nn .Mcdule﹜ －
      2. def init (self ) .

。。

* + - 1. super (aut enc der, self ﹜ init ﹛）
      2. self encode r a nn Sequential ﹛

5 nn.Linear (28\*28, 128﹜ ，

1. nn ReLU (True﹜ ，
2. nn Linear (l28, 64 ) ,
3. nn.ReLU (True) ,
4. nn Linear (64, 12) ,
5. nn.ReLU ﹛ True) ,
6. nn.Linear ( l2, 3)

12

6戶3

13 self .decoder 呈 no.Sequential ( 14 on.Linear ﹛ 3, 12) ,

15 n口 ReLU (True﹞，

16 on .Linear (12, 64) ,

17 nn .ReLU ﹛ True) ,

18 nn .Linear (64, 128﹜，

19 nn .ReLU ﹝ True) ,

20 nn .Linear ( l28, 28\*28) ,

21 nn .Tanh (I

22

23

1. def forward ﹝ self , x) ,
2. x self .encoder ﹝x﹜
3. x self .decoder (x)
4. return x

定義一個簡單的 4 層網路作為編碼器， 中間使用 ReLU 敢動函數，最後輸出 的維度是 3D 的，定義的解碼器，輸入 3D 的編藹，輸出一個 28 ×泌 的圖像資 料，特別要注意最後使用的敢動函數是 Tanh，這個殷勤函數能夠將最後的輸出轉

換到 1 ～ I 之間，這是因為輸入的圖片已經轉換到了 l ～ 1之間，這裡的輸出

必須和它對應。

在開始訓練網路的時候， 每次編碼器的輸入要展開乘 28 × 28-784 的大小， 然後輸入編碼器獲得編碼， 再輸入解碼器獲得產生的圖片 ，然後和原圖片去比 較，透過 save image（） 這個函數輸出網路獲得的結果。

看看自動編馮器的資料結果到底是怎麼樣的，如圖 6.3 所示 B

.. . ,,

92 月／ d s i dJ回 A J 7

J

7 1 J

A －d o

－J

也

，＿， ＇司 ，’，

M J 5 3 4

Z J Y J

E

已 4

F F－

，

’

FU d

r e

t 9

" 叫

，

’

'-' '

' (.. ' '

f J

dk

t

d ，

e

t ／

h

’ ，

F f

’

f

r ，

e

f l y

／ 7，

1 ，

圖 6.3 資料結果

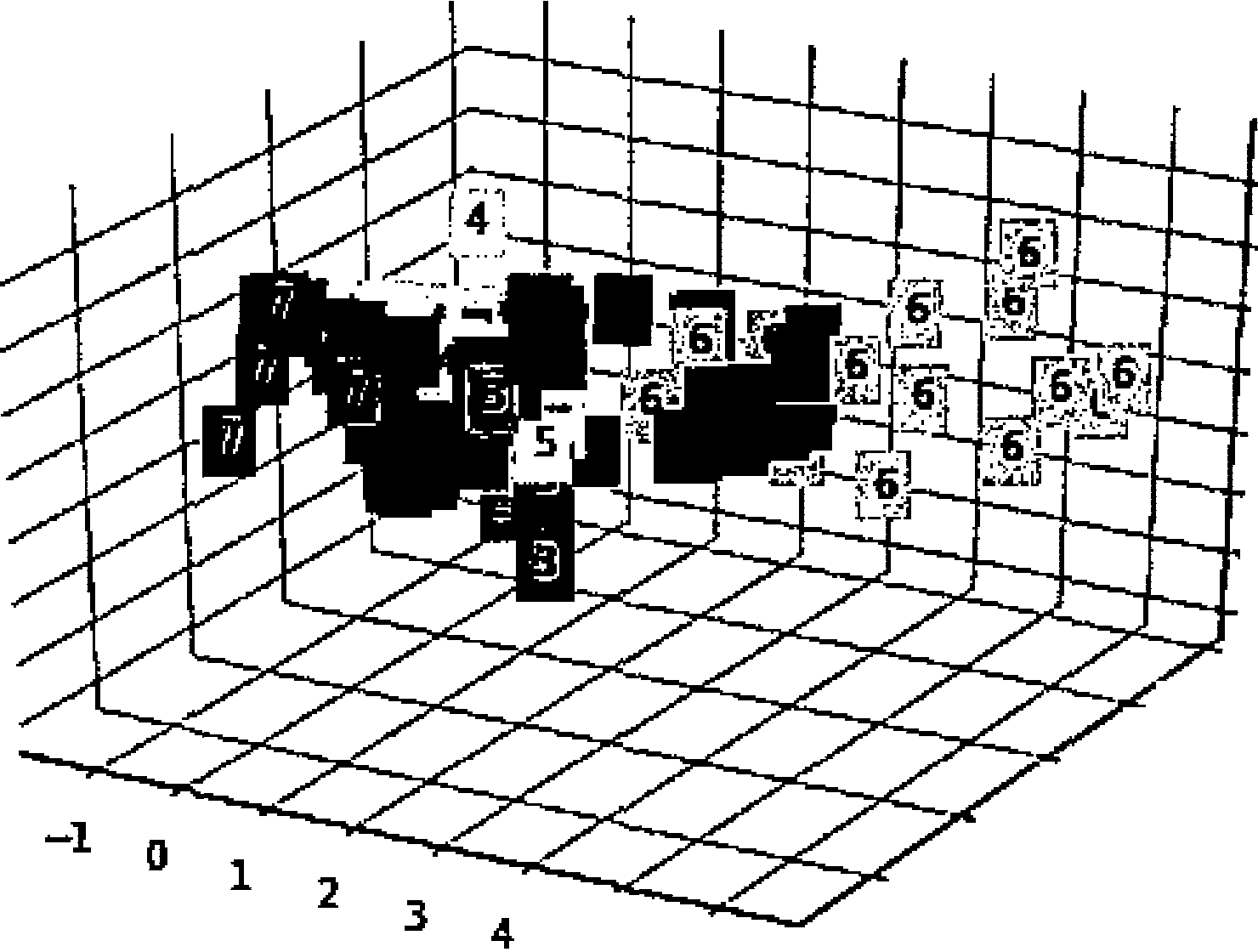
6.1 產生模型

．．

圖 6.3 左邊是剛訓練 20 次獲得的效果， 圖片還比較模糊。閏 6.3 右邊是 40 次 之後的效果，可以看到圖片已經有點清晰了，但輪廓還是有些模糊， 所以說多層 國知器能鉤做得比較有限，產生的圖片還是比較模糊的。

可以將蝙碼的好佈視覺化出來， 實際看看隨機給一個 泊的編碼，能鉤產生 的圖片的分佈，如國 6.4 與圖 6.5 所示。

5 6



20

LS LO 0.5

0.0

-0.5

-10

-1.5

2

0

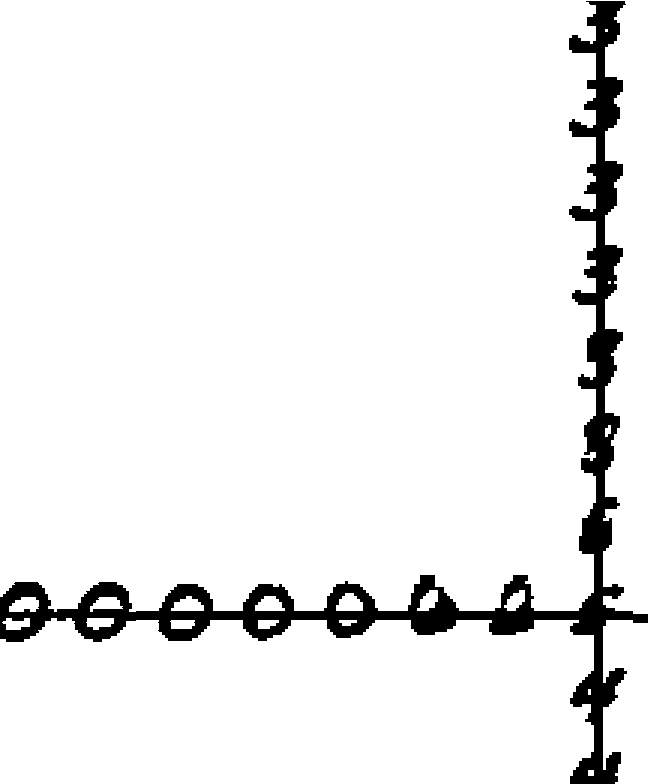
-2

-4

-6

圖 6.4 30 母佈

圖 6.4 是一個 3D 的分佈，圖 6.5 是一個二雄的分佈，可以看出產生的圖片是 如何根據所列出的隨機編晦而改聾的。



’’’’

－－＋

／ ／ ／JI

／ ／ ／ ／’

，r

f

333SSE－

53SE－－za

55SSS巴 巴

55SSSOO

555SOOO

。

。

566eooo

6

6

ooo

圖 6.5 二雄主子佈

EJ

咱A

，f ，d，，s ，1．s－z－＋．E．E－．．．．．E ．E－－RE

／ ／ ／JIll－＋l11111t

’

l111’

．

，，

．，

11

＋

11

1

J／ ／ ／frt－TF77蝠，q Tq

JJJrrrr177777qqq

JJ32ggrET77qqqqq

aqqqq 可可

ooqqq 叮 叮

OOOO 句 叮 叮

OOOOOO 可

OOOOOOO

ooooooo ooooooo

Fb

．44

接下來希望使用一個更加複雜的網路做自 動編碼器，這樣能夠使產生的圖片 效果更好，而對於圖片而言 ，最好的影像處理網路就是旋積神經網路， 所以可以 建置一個旋積神經網路做自動編碼器 。

工 class CCaut。encoder (nn .M。duleJ .

1. def init ﹛ self ﹜：

。。

1. super ﹛D 己 aut enc der, self ) init （ ﹜
2. self .enc。der 巨 nn .Sequential ﹛

5 nn .C。nv2d (l, 16, 3, stricte3, padding ）， # b, 16, 10, 10

6 nn ReLU ﹛ True﹜ ，

。

7 nn MaxP 12d ﹛ 2, stricte豆2﹜ ，＃ b, 16, 5, 5

1. nn .Conv2d (16, 8, 3, stride2, padding ）， # b, 8, 3, 3
2. nn ReLU (True﹜ ，
3. nn MaxP。。12d (2, stridelJ # b, 8, 2, 2

11

12 self decoder = nn .Sequential ﹛

13 nn .C。nvTransp 。se2d (8, 16, 3, stride=2) , # b, 16, 5, 5

14 nn ReLU ﹛ True﹜ ，

15 nn .C。nvTransp。se2d (16, 8, 5, stride=3, padding,-1) , # b, 8, 15, 15

1. nn .ReLU (True﹜ ，

。

1. nn .ConvTransp se2d (8, 1, 2, stnde=2, padding=l) , # b, 1, 28, 28
2. nn .Tanh ﹛）

19

20

1. def f。rward (self , x﹜ ：
2. x = self .enc。de r (x)
3. x = self.dee。der (x)
4. return x

這裡的編碼器使用了多層旋積神經網路，解碼器使用了 nn.ConvTranspose2d0 '

看作旋積的反操作，可以在某種意義上看成是反旋積。 使用旋積神經網路作為自動編碼器和前面使用多層感知器定義的自動編聽

器一樣，最後獲得的結果會 比簡單的多層感知器好，可以看如圖 6.6 所示的結果

比較。

圖 6.6 左邊是多層全連接神經網路的結果， 圖 6.6 右邊是旋積神經網路的結 果，可以看出其實兩者之間的差別不大，只不過多層戚知器的結果會稍微模糊一

是色。

6.1 產生模型 國－

*I* ＇， *1 7* I · . 3 1 , , I

：

*3* ':; l 弓 I :: *I* 吋 *1 1 .:- 1 1*

I

抖 也 *71* ？ 可 ＿ *1* 1 , *0* I

。

﹒

*I* , ,*i* I ＇ ﹔τ · *J* I

3 a

ι

I；：

于

I *' I* 己

;

b 心卡，*I*

o I :

I *t* 斗

*1*

汁

γ 7

*! 5- , r O , I* 「 ？丘，i可 叫 q ，-I

*(* J 雪 凶 ， 1 - *3* ＼ 品 7 7 1

，

*1*

*' t* ;

*1* 1 1 .· 0 3 :. <:. 7 ,I

*·-1 7* , ;;i ., I

。

|  |  |
| --- | --- |
| *i ,'* | |
|  | ’， |
| 7 · | *1 I* '  I *I* |
| 7 |

- .' f 三︱

；

|  |  |  |
| --- | --- | --- |
| ι | ， | I |

· .：， , C 1

句 守

, *·* 泣 － *0 '-f / 7 1*

，

I *f* 7

1 I -, / C 7 1 I ::; I

圖 6.6 結果比較

6.1.2 變封自動編碼器

變分自動編碼器 （ Variational AutoEncoder ） 是自動編碼器的升級版本， 它的 結構跟自動編碼器是相似的，也是由編碼器和解喝器組成的。

在自動編碼器中 ，需要輸入一張圖片 ，然後將一張圖片編E庸之後發得一個隱 含向量，這比隨機取一個隨機向量好，因為這包含著原圖片的資訊 ，然後將隱含 向量解1ley獲得與原圖片對應的照片 。

但是這樣其實並不能任意產生圖片 ，因為沒辦法自己去建置隱藏向量，需要 透過一張圖片輸入編碼才知道E獲得的隱含向量是什麼， 這時就可以透過變分自動 編碼器解決這個問題。

其實原理特別簡單，只需要在開發過程替它增加一些限制，迫使它產生的隱 含向量能夠粗略地遵循一個標準正態分佈， 這就是它與一般的 自動編碼器最大的 不同 。

這樣產生一張新圖片就很簡單了，只需要給它一個標準正態分佈 的隨機隱含 向量，透過解碼器就能夠產生想要的圖片 ，而不需要先給它一張原始圖 片編碼。

在實際情況中 ，需要在模型的準確率上與隱含 向量服從標準正態分佈之間做 一個權衡，所謂模型的準確率就是指解碼器產生的圖片 與原圖片的相似程度。可

一 第 6 章

產生對抗網路

以讓神經網路自己做這個決定， 只需要將這兩者都做一個 l.oss’然後再將它們求 和作為整體 l.oss ，這樣網路就能夠自 己選擇如何做才能使得這個整體 l.oss 下降。 另外要衡量兩種分佈的相似程度， 需要引用一個新的概念， 也 divergence’這是 用來衡量兩種分佈相似程度的統計量，它越小，表示兩種機率分佈越接近。

對於離散的機率分佈， 定義如下：

且（P II Q） 口"）':P(i)loPg(i)-

(6.1)

于 Q(i)

對於連續的機率分f布，定義如下：

Dn (P l[ Q） 口"） 'P(i)lo*P*g*(i*-*)*

(6.2)

U γ Q(i)

這裡就是用 KL divergence 表示隱含向量與標準正態分佈之間差異的 loss ，另 外一個 loss 仍然使用產生圖片與原圖片的均方誤差來表示。

這裡變分編碼器使用了一個技巧一一 「 重新參數化」 來解決 也 divergence 的 計算問題。

這時不再是每次產生一個隱含向 量，而是產生兩個向量：一個表示平均值， 一個表示標準差，然後透過這兩個統計量合成隱含向 量，用一個標準正態分佈先 乘標準差再加上平均值就行了， 這裡預設編碼之後的隱含向量是服從一個正態分 佈的。這個時候要讓平均值盡可能接近 0 ，標準差盡可能接近 1＇論文裡面有詳細 的推導介紹如何獲得這個 loss 的計算公式，有興趣的讀者可以去 https://arxiv.org/ pdf/1606.05908.pdf 看看推導過程。

這個問題可以使用 PyTorch 輕鬆地實現﹒

rec。nstructi。n function - nn BCEL。55 ﹛ 51ze\_average-Fal5e) def l。55 functi。n (rec。n x, x, mu, l。gvar﹜．

1

2 3 4 5 6 7

．冒﹒

reccn\_x· generating 工mages

。

X, rig斗n images mu , latent mean

# mse l。55

6.1 產生模型 圓圓

8 1。qvar, latent l。q variance

10 BCE - rec 。nstructi。n functi。n ( rec。n X, X﹜

9

11 t 1。ss - 0 5 金 sum ( l + 1。g (sigma《2﹜ － mu"2 - sigma"2)

1. KLD\_element - mu .pow (2) .add一（ 1。gvar exp ﹝ 1 1 .mul (-11 .add ﹛ 11 add 11。gvarl
2. KLD - t。rch .sum (KLD element l .mul ( 0 51
3. t KL divergence
4. return BCE + KLD

另外變分編碼器除了可以隨機產生隱含變數， 還能夠加強網路的泛化能力。 最後是 VAE 網路定義的程式實現：

1. class V且﹛nn .M。dulel
2. def init ﹛ self I
3. super (VAE, self I . init ﹛﹜

4

1. self fcl - nn .Linear (784 , 4001
2. self. f c21 - nn Linear ﹛ 400, 201
3. self. fc22 - nn. Linear (400, 201
4. self.fc3 - nn .Linear (20 , 4001
5. self .fc4 - nn .Linear (400, 7841

。

1. def enc de (self , xi ,

11 hl - F relu ﹛ self .f cl (xi I

12 return self .fc21 (hll , self .fc22 (hll 13

1. def reparametrize ﹛ self , mu, logvar1 ,

。

1. std - 1 gvar.mul ﹛ O SI .exp （﹜

。

1. if t rch.cuda is available ﹛）：

。

1. eps - torch .cuda.FloatTensor (std.size l) l .n rmal一﹛）
2. else.

。 。 。 。

1. eps - t rch .Fl atT ens z ﹛ std.size l) I .n rmal II
2. eps - Variable (epsl
3. return eps mul (stdl .add ﹛mu ﹜

22

1. def decode (self , zl .
2. h3 - F.relu ﹛ self f c3 (zl ﹜
3. return F.sigmoid (self .f c4 ﹛ h31 I 26

。

1. def f rward (self , xi ,
2. mu, 1。gvar - self enc。de ﹛ xi
3. z - self reparametrize ﹛mu, 1。gvarl
4. return self.dee。de ( z﹜ ，mu, 1 那，ar

一－－－ 第 6 章 產生對抗網路

VAE 的結果比普通的 自動編碼器要好很多， 可以將它和旋積神經網路做的標 準自動編碼器獲得的結果進行 比較。

圖 6.7 左邊是標準自動編碼器，右邊是變分編碼器， 可以看到右邊的結果明 顯要比左邊的結果清晰。

1 汀

己 I j I *i 7* . 4 υ 品

＇、

司

*0 t*

*J* 3 a

？也’ 叫 f ,, "

*)* 7

且

＆ I

I 且.

C'{ ＇＂

*S* ,;- *.:'* "

*:* .,

'· "

*.- a ::, ·. <:.*

*·1 ·:. f 7*

I

'i /

J

7 守 *I* I:, ，.： ·, •· I *O J*

、

O *I*

., *)*

: , '

「

*I* 已 （

T ι *:*

可 ）

.\_,

:·

之 *0 7 '1 7*

*7* ，》 *I*

,

*5* ·1 *I* 守 。

？ 去 l I

1. 4

*·*

，

*I* \_ .• */* '. *'i*

”

：

ι

”，

7 *J. "!o* D

q J.

|  |  |
| --- | --- |
| · 丸 | 斗 |
| 三 O | |

＇智 *d* ι

’: 7 2: 句 r., S'" ι

’吧 ／

弓 3 .:,

圖 6.7 VAE 與標準自動編碼器結果比較

雖然變分編晦器有這些優點， 但是它同樣有自動編碼器的缺點 ，那就是直 接計算產生圖片和原始圖片的均方誤差來作為損失函數， 使得產生的圖片會有點 模糊。

接下來介紹產生對抗網路是如何解決這個問題的。

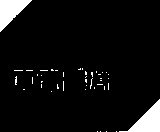
[ 6.2 I 產生對抗網路

2014 年，深度學習三巨頭之一 Ian Goodfellow 提出了產生對抗網路 （ Generative Adversarial Networks, GANs ） 這一概念，剛開始並沒有引起轟動 ，直到 2016 年， 學界、業界對它的興趣如 「 爆炸」 一樣爆發，多篇重磅文章陸續發表，Lecun 這 樣形容 GANs 「 adversarial training is the coolest thing since sliced bread 」。 2016 年 12 月NIPS 大會上 ，Goodfellow 做了關於 GANs 的專題報告，使得 G必 s 成為了當 今最熱門的研究領域之一 ，接下來介紹如今深度學習界的明星一一產生對抗網路。

6.2.1 何為產生對抗網路

產生對抗網路，根據它的名字，可以推斷這個網路由兩部分組成：第一部分 是產生，第二部分是對抗。 這個網路的第一部分是產生模型， 就像之前介紹的自 動編碼器的解碼部分﹔ 第二部分是對抗模型， 嚴格來說它是一個判斷真假圖片的 判別器。產生對抗網路最大的創新在此， 這也是產生對抗網路與自 動編碼器最大 的區別。簡單來說，產生對抗網路就是讓兩個網路相 互競爭，透過產生網路來產 生假的資料，對抗網路透過判別器判別真偽， 最後希望產生網路產生的資料能夠 以假亂真騙過判別器。

過程如圓 的所示。

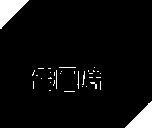
D－ 維  II柯 ＼JI ﹝ 預測棲霞

．．

：

噪音向量

I



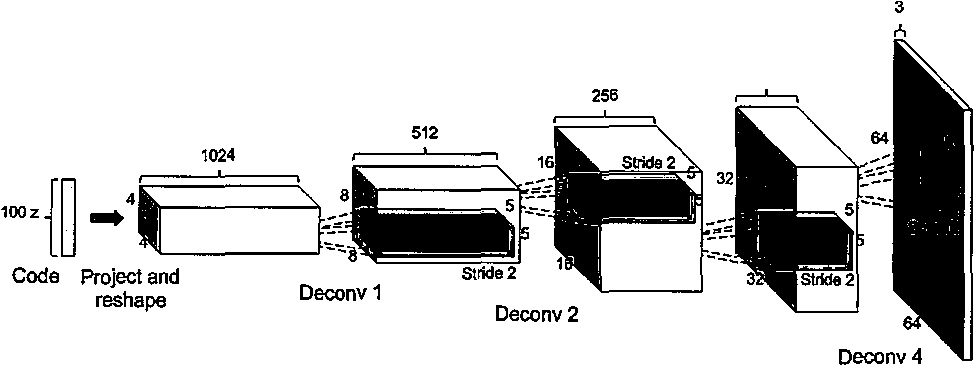
．

圓 6.8 產生對抗網路產生資料過程 下面依次介紹產生模型和對抗模型。

油 1. 產生模型

首先看看產生模型， 前一節自動編碼器其實已經列出了一般的產生模型。 在 產生對抗網路中 ，不再是將圖片輸入編碼器獲得隱含向 量然後產生圖片，而是隨 機初始化一個隱含向量 ，根據變分自動編碼器的特點，初始化一個正態分佈的隱 含向量，透過類似解碼的過程， 將它對映到一個更高 的維度，最後產生一個與輸 入資料相似的資料， 這就是假的圖片。這時自動編碼器是透過 比較兩張圖片之間 每個像素點的差異計算損失函數的， 而產生對抗網路會透過對抗過程來計算出這 個損失函數，如圖 6.9 所示。

－第 6 章 產生對抗網路



128

「一」「

Deconv 3

Image

圖 6.9 產生模型

油 2. 對抗模型

重點來介紹對抗過程，這個過程是產生對抗網路相對於之前的產生模型如 自 動編碼器等最大的創新。

對抗過程簡單來說就是一個判斷真假的判別器， 相當於一個二分類問題，輸 入→張真的圖片希望判別器輸出的結果是 1，輸入一張假的圖片希望判別器輸出 的結果是 0 。這跟原圖片的 label 沒有關係，不管原圖片到底是一個多少類別的圖 片，它們都統一稱為真的圖片 ，輸出的 label 是 l’則表示是真實的﹔而產生圖片 的 label 是 0 ，則表示是假的。

在訓練的時候，先訓練判別器，將假的資料和真的資料都輸入給判別模型， 這個時候最佳化這個判別模型， 希望它能夠正確地判斷出真的資料和假的資料， 這樣就能夠獲得一個比較好的判別器。

然後開始訓練產生器， 希望它產生的假的資料能夠騙過現在這個 比較好的判 別器。實際做法就是將判別器的參數固定， 透過反向傳播最佳化產生器的參數， 希望產生器獲得的資料在經過判別器之後獲得的結果能盡可能地接近 1，這時只 需要調疆一下損失函數就可以了，之前在最佳化判別器的時候損失函數是讓假的 資料盡可能接近 0 ，而現在訓練產生器的損失函數是讓假的資料盡可能接近 I o

這其實就是一個簡單的二分類問 題，這個問題可以用前面介紹過的很多方法 去處理，例如 Logistic 回歸、多層感知器、旋積神經網路、循環神經網路等。

上面是產生對抗網路的簡單解釋， 可以透過程式更清晰地展示整個過程。 跟 自動編碼器一樣，先使用簡單的多層感知器來實現：

1. class discriminator ﹝ nn .Module I ,
2. def init ﹝ self ) ,
3. super (discriminator, self ) . init （）
4. self .dis nn.Sequential ﹝

5 nn.Linear ( 784 , 256﹜，

6 nn.LeakyReLU ﹝ 0 .2) ,

7 nn.Linear ( 256, 256﹜，

8 nn.LeakyReLU ( 0.21 ,

9 nn .Linear ﹛ 256, 1) ,

10 nn .Sigmoid （ ﹜

11

12

13 def forward ﹛ self , x﹜：

14 x 呈 self .dis (x)

15 return x

上面是判別器的結構， 中間使用了斜率設為 0.2 的 LeakyReLU 敢動函數， 最後需要使用 nn.Sigmoid（） 將結果對映到 0 ～l 之間機率進行真假的二分類。 這 裡之所以用 LeakyReLU 敢動函數而不使用 ReLU 敢動函數，是因為經過實驗，

LeakyReLU 的表現更好。

1. class generat。r (nn Mcdule﹜ ．
2. def 一＿init一＿ (self • input\_size ） 回
3. super ﹛ generat 主，self ﹜ init 11
4. self .gen nn Sequential ﹛
5. nn .Linear ﹛ input\_size, 256) ,
6. nn ReLU ﹛ True) ,

7 nn .Linear (256, 256) ,

8 nn .ReLU ﹛ True﹜ ，

9 nn .Linear (256, 784) ,

10 nn .Tanh﹛）

11

12

。

1. def f rward (self , x)
2. x self gen ﹝X﹜
3. return x

圓圓圓圓－ 第 6 章 產生對抗網路

這就是產生器的結構， 跟自動編碼器中的解碼器是類似的， 最後需要使用 no.Tanh（）， 將資料分佈到 一 l ～ l 之間 ，這是因為輸入的圖片會規範化到 1～ l 之間。

接著需要定義損失函數和最佳化函數．

* 1. criten。n *=* nn .BCEL。ss ﹛ I # Binary Cr。ss Entr。PY
  2. d一。ptimizer - t。rch. 。ptim .Adam ( D.parameters ﹛ I, lr-O .00031
  3. g\_optimizer - t。rch 。ptim.Adam (G .par部neters II , lr-0 0003﹜

這裡使用二分類的損失函數 mBCELoss（）’使用 Adam 最佳化函數，學習率 設定為 0.0003 。

接著是最為重要的訓練過程， 這個過程分為兩個部分． 一個是判別器的訓 練，→個是產生器的訓練 。

首先來看看判別器的訓練 。

l 工mg = 1mg.v1ew (num \_1mg, -1)

1. real img = Variable (img ) cuda II

。。

1. real\_label = variable ﹝t rch nes (num\_img) J cuda II
2. fake\_label = Variable （立。rch . zer。s (num\_img ﹜ I cuda ﹛）
3. # compute l。ss 。f real img

5

1. real＿ ut = D ﹝ real\_img )
2. d loss real = criten。n ( real 。ut, real label)
3. real sc。res = real out

10

1. # C。mpute l。ss 。f fake\_img
2. z = Variable （七。rch.randn ﹛num 1mg , z\_dimensi。n) J .cuda ﹛）
3. fake\_img = G ﹛ z)
4. fake一。ut = D ﹛ f ake\_img﹜
5. d loss fake = cnteri。m ﹛ fake out, fake label)
6. fake scores = f ake *out*

17

。

1. # bp and ptimize
2. d l。ss = d 1。ss real + d l。ss fake
3. d 。ptimizer . zer。grad II
4. d l。ss.backward ﹛）
5. d optimizer. step I ﹞

開始需要自己建立 label 真實的資料是 l，產生的假的資料是 0，然後將其 實的資料輸入判別器獲得 loss 將假的資料登錄判別器獲得 loss，將這兩個 loss 加起來獲得整體 loss’然後反向傳播去更新參數就能夠獲得一個最佳化好的判 別器。

接下來是產生模型的訓練：

1. J C。mpote l。ss of f ake\_img
2. z - variable I七 rch ran曲﹛num\_i呵 z d工mensi n)) .cuda I ﹜ ＃ 獲得隨機雜訊

3 fake img - G ( z﹜ ＃ 產生假的圖片

。

1. otput a D (fake\_img) ＃ 經過判別器獲得結果

。 ﹛。 。

1. g\_l ss = er斗terion utput, real\_label) ＃ 獲得假的圖片與真實圖片label 的 1 ss
2. # bp and 。ptimize

6

1. g\_optimizer. zero\_grad ﹝﹜ ＃ 歸。梯度
2. g\_loss.backward l) ＃ 反向傳播
3. g optimizer step I ﹜ ＃ 更新產生網路的參數

一個隨機隱含向量透過產生網路1獲得了一個假的資料， 然後希望假的資料經 過判別模型後盡可能和真實 label 接近，透過 E loss=criterion(output, real label） 實 現，然後反向傳播去最佳化產生器的參數，在這個過程中 ，判別器的參數不再發 生變化，否則產生器永遠無法騙過最佳化的判別器。

除了使用簡單的多層I認知器外，也可以在產生模型和對抗模型 中使用更加複 雜的旋積神經網路，定義十分簡單。

工 class disc笠iminat。r ﹛ nn Module ﹜ －

1. def init (self ﹜ ：

。

1. super (discriminat r, self ) . init ﹝）

。

1. self .c nvl a nn .Sequential (

。

5 nn c nv2d ( l, 32, 5, paddinga2) , # batch, 32, 28, 28

6 nn LeakyReLU ﹛ 0.2, True) ,

。

7 nn AvgP 12d (2, stride-2) , # batch, 32, 14, 14

8

9 self ccnv2 a nn . Sequential ﹛

10 nn .C。nv2d ( 32, 64 , 5, padding-2) , i batch, 64, 14 , 14

nn .LeakyReLU ﹝ 0 . 2, True) ,

1

1

nn AvgPool2d (2, stride-2) # batch, 64, 7, 7

2 3

1

1

一間 一 第 6 章 產生對抗網路

14 self fc nn Sequential (

15 nn Linear ( 64 \*7會 7, 1024 ) ,

16 nn .LeakyReLU ( 0 2, True) ,

17 nn Lrnear (1024, 1) ,

18 nn.Sigmoid （﹜

19

20

。

21 def f rward (self , x﹞ 22

23 x, batch, width, heigh乞 channell

24 ’，

1. x self ccnvl (x)
2. x self .conv2 (xi
3. x a x view (x size ( O J , 1﹜
4. X a self fc (X﹜
5. return x

30

1. class generat 主 ﹛nn.Mcdule ﹜ ．
2. def in斗匕 ﹝ self , input size, num feature﹜ －
3. super (generat。r, self ﹞ init （ ﹜
4. self .fc a nn Linear ﹛ input\_size, num feature) # batch, 3136-lx56x56
5. self .br a nn Sequential (

。

1. nn .BatchN rrn2d (11 ,
2. nn ReLU ﹛ Tr口e) 38
3. self .ct。wnsample 工 a nn Sequentia﹔1 (
4. nn .C。nv2d ( l, 50, 3, str工deal, paddingal) , # batch, 50, 56, 56
5. nn .BatchN。rrn2d (50) ,
6. nn ReLU (True) 43

。

1. self .ct wnsample2 a nn .Sequential ﹛
2. nn.C。nv2d ﹝ 50, 25, 3, strideal, paddingal) , # batch, 25, 56, 56

4 6 nn .BatchNorrn2d (25﹜ ，

47 nn ReLU (True)

48

4 9 self .downsample3 " nn .Sequential (

。

。

nn c nv2d ( 25, 1, 2, stridea2J , # batch, 1, 28, 28 nn Tanh ﹝﹜

1 2

5

5

5

53

1. def forward ﹛ self , x﹜ ：
2. x a self .fc(x)
3. x a x .view (x size (O J , 1, 56, 56)
4. *x* self br ﹝ x)

。

1. *x* self d wnsamplel (xi
2. *x* self .dawnsample2 (x)
3. x - self. dawnsample3 (x)
4. return *x*

圖 6.10 左邊是多層I竄知器的產生對抗網路， 右邊是旋積產生對抗網路， 右邊的圖片比左邊的圖片雜訊明顯更少。 在旋積神經網路裡引用了批次標準化 ( Batchnormalization ） 來穩定訓練， 同時使用了 LeakyReLU 和平均池化來進行訓

練。產生對抗網路的訓練其實是很困難的， 因為這是兩個對偶網路在相互學習 ， 所以需要增加一些訓練技巧才能使訓練更加穩定。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | *1* 3 J | | |
| ﹜ | 吉 | |
| *1l* | *9* | *S* | | *'1* | *3* |

圖 6.10 產生對抗網路比較結果

*3*

I *i*

‘＇

, ; :, q 5 :, ;

,

'

I

*?* 3

7 1 *'i .S* 可 9 *1*

＆ 可 1

／

*t*

,

*I*

’

*'>*

*I*

·,

*I/ I* I *t 'I* 長 I S- *.*

*:, J J l -' I I* ？

'1

τ

*t*

*·1 ·i*

*Y*

*1*

于

*1* . 1

主 f *9*

？*s·* 3

I *s* .; "' *1*I*'1 6*

I *5 :, c; 1 , t 1*

*3* γ *7 'f 1 '.i*

*7*

/ 7

*I*

*I*

” I

i /

·' 7

＜

7

,

' 1 5

ilLf

•I f

'' *i* 主 的 ＇／ "*'i*

叮 *l I*

- tl

*I* 7

*4*

γ 2 ? <'· 9 'l 1

、？？﹜ B

*I* J *4*

' *?* 7 i

I

*I*

市

.:

以上介紹了產生對抗網路的簡單原理和訓練流程， 但是對產生對抗網路而 言，它其實並沒有真正地學習到它要表示的物體 ，透過對抗的過程，它只是產生 了一張盡可能真的圖片 ，這就表示沒辦法決定用哪種雜訊能夠產生想要的圖片 ， 除非把初始分佈都試一遍。 所以在產生對抗網路提出之後， 有很多以標準產生對 抗網路為基礎的變式來解決各種各樣的問題，我們會在 6.3 節介紹。

6.2.2 產生對抗網路的數學原理

上一節介紹了什麼是產生對抗， 這一節將用嚴格的數學語言證明產生對抗網 路的合理性。

團圓圓圓 第 6 章 產生對抗網路

首先需要一點預備知識， KL divergence’這是統計中的概念，用於衛量兩種 機率分佈的相似程度 ，數值越小，表示兩種攝率分佈越接近。 離散的機率分佈， 定義如下 ：

*DKL(P* II *Q)* = *LP(i)log* 旦旦

*Q(i)*

連續的攝率分﹛布，定義如下：

*p(*

『I閻 －*x )*

*DKL(P II Q)=* p(x)log 一 *dx*

恥 J\_, *q(x)*

根據之前介紹的 內容，要做的操作如圖 6.11 所示。

(6.3)

(6.4)

想要將一個隨機高斯雜訊 z 鐘過一個產生網路 G 獲得一個和真的資料分怖 *PdataC x*） 差不多的產生分佈 *Pa( X*﹔*0*） ，其中參數 是網路的書數訣定的， 希望找到 。 使得 *po( x 0*） 和 *Pd* ， （ *x*） 盡可能接近。

*PG (x 8)*

*G(z)* = *X* generated distribution

*PdataCx)*

true eta恆 dis甘ibution

區司 日

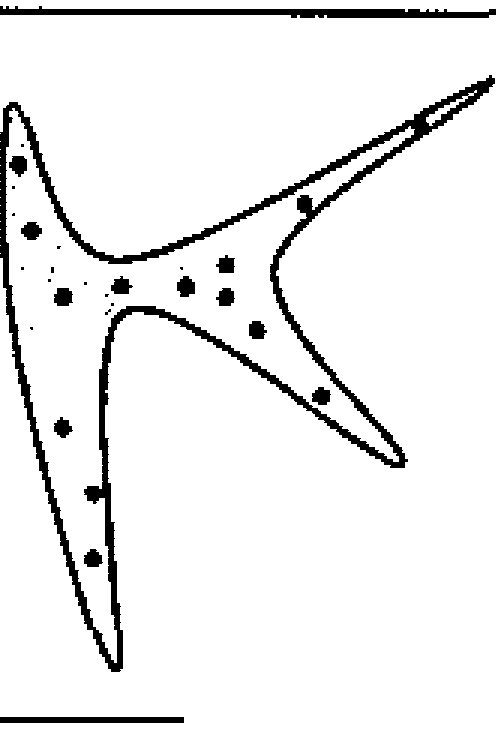
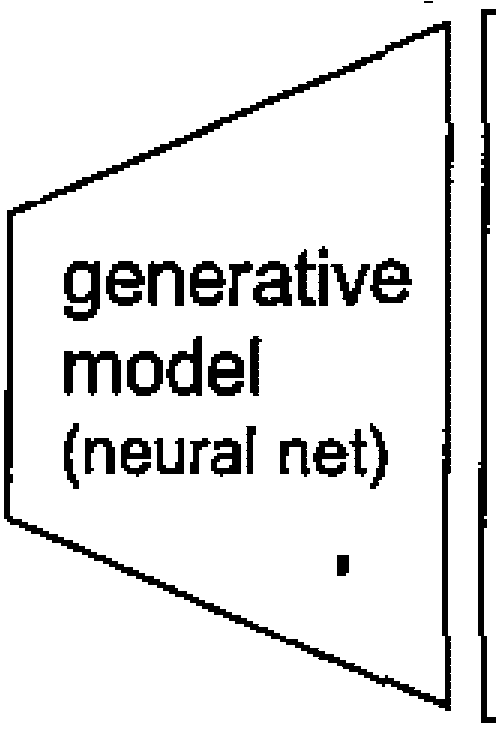


image space Image spa由

圖 6.11 操作過程

從其實資料分佈 *Pd.,.( x*） 裡面取樣 m 個點，﹛x1 *,x2 ,"' ·,x*勻，根據指定的事數 θ

可以計算以下的機車 *Pa( x 0*） ，那麼產生 m 個樣本資料的似無 （ likelihood ） 就是：

L = II7 1九 （*x IJ)* (6.5)

找到 伊 東最大化這個似然估計：

*9°* = arg m缸 rr:1Pa(xi θ） 特 缸gm血 log II ／已 （*x* （）） (6局 （6.6)

= arg m叫主log凡（*x B)* (6.7)

局 arg m血 *E* 也弘﹝log *Pa*令，*B*）﹞ (6.8)

特 缸g max0 J,P帥（x) log九（月*B)dx -L P*帥令） log 弘（*x)dx* (6.9)

r *P-( x* ：趴

J －

= arg m缸 一九a Jx)log 」一」 *dx* ' (6 10)

h 丸。（*x)*

＝缸g min8 *KL* （凡＂＇

(6.12)

而*Pa( x* ；的 重則可算出來呢 ？

九＜*x)=* Lι°＇(z）／［叫l彷

(6.13)

裡面的 表示示性函數，也就是﹒

*I* = J O *G( z ) ,t, x*

*ac,,-,* [1 G(z) = *x* （丘 14)

這樣其實根本沒辦法求出這個 *Pa( x*） ，這就是產生模型的基本想法。

Generator G 是一個產生器， 指定先驗分佈 *Pp* 間（ z）， 希望獲得產生分佈

*Pa( x*） ，這裡很難透過相當大似然估計獲得結果。

Discriminator D 中D 是一個函數，用來衡量 *Pa( x*） 與 *P,,., ( x* ） 之間的差距，可 用來取代相當大似然估計。

首先定義函數 *V(G,D*） 如下：

*V(G,D)=E,\_p*帥﹝log D（均﹞ ＋*Ex-PG* ﹝log(l -D（均）﹞ （6.15)

可以透過下面的式子求得最佳的產生模型：

*G* = arg min max *V(G,D)*

*G D*

(6.16)

－－－－－ 第 6 章 產生對抗網路

為什麼定義了一個 P扭，*D*） 然後透過求 max 和 min 就能夠取得最佳的產生模 型呢 ？首先我們只考慮 max0 *V( G,D*） ，看它表示什麼含義。

在指定 G 的前提下，取一個合適的 D 使得 阿〔G 的能夠取得最大值，這就是 簡單的微積分。

*V =E* ＿恥﹝*log D( X* ）﹞ ＋*E* 而﹝log(! *D( x*））﹞ （ 6.17)

*=L* 弘（*x)log D( x)d*吋，*Pc( x) log(I -D( x*））申 （6.叫

= *f)*圳特 *log D( X* ）﹞喘（特 *log( I -D( x*））﹞枕 (6.19)

對於這個積分， 要取它的最大值， 希望對於指定的 x’積分裡面的項是最大 的，也就是希望取到一個最佳的 *D* ＊ 最大化下面這個式子 ：

*P,,* ,(x)log *D( x)+* PG (x)log ( 1-D*( x))* (6.20)

在資料指定、G指定的前提下，*P,,"(x*） 與 *PG ( x*） 可以看作常敏，分別用 r *b*

來表示它們，這樣就可以獲得以下的式子：

/(D) = a log(D) + b log(l *D) df ( D)*

(6.21)

*dD*

*= ax*

I *+bx*

1 × （ I) = 0

*D 1 -D* (6.22)

I . I

*ax* －－ ？

*-:= D* × －－－

*D 1 -D* (6.23)

特 *a x (I-D* ）＝的 D﹒ (6.24)

*D ( x)=*

,,.,(x)

*P,,.,( x)+ Pa ( x)*

(6.25)

這樣就求得了在指定 G 的前提下 ，能夠使得 *V(D*） 取得最大值的 D ，將 D 代

回原來的 *V(G,D*），獲得以下的結果

m阻*V(G,D)* = *V(G,D* ） (6.2郁的

＝丸』﹝log 可靠前］＋弘﹝log 忍石研﹞ (6.27)

*p* h、 *Pa( x)*

*=L* lι

了如J,

1ι（ x)

前你

弘（特 log 互訪布

*Pa( x)log* 瓦話可

(6.羽）

2 2

*P ( x)+ P. ( x*、 *P* (x)+ *P. ( x*、

= -2log 2 ＋紅（*P,,,,( x)* II」Lτ*....:2.*一）＋ *KL* （九（*x)* 11 一也Lτ*....:2.*一） (6.2月

= -2 log 2 + *2JSD* （九，*,( x)* II *Pa( x))* (6.30)

這裡引用了一個新的概念，JS Divergence’定義如下：

*JSD*

透過上面的定義知道 KL Divergence 其實是不對稱的，而 JS Divergence 是對 稿的，它們都能夠用於衡量兩種分佈之間的差異。 看到這裡其實就已經推導出了 為什麼這麼衡量是有意義的， 因為取D 使得 *V(G* 的取得 max 值 ，這個時候這個 max i直是由兩個 KL divergence 組成的，相當於這個 max 的值就是衡量 *Pa (x*） 與 p帥 ） 的差異程度，所以這個時候取：

a屯（mina max0 *V(G,D))* (6.32)

就能夠取到 G使得這兩種分佈的差異性最小， 這樣自然就能夠產生一個和原 分佈盡可能接近的分佈，同時也擺脫了計算相當大似然估計，所以 GAN 的本質是 透過改變訓練的過程來避免煩瑣的計算。

以上詳細地介紹了產生對抗網路的數學推導， 略微有些煩瑣 ，但是有助了解 產生對抗網路的本質。

－第 6 章 產生對抗網路

I 6.3 I Improving GAN

這一節將介紹改善的產生對抗網路， 因為產生對抗網路存在很多問 題，所以 人們研究是否可透過改善網路結構或損失函數來解決這些問題。

* + 1. Wasserstein GAN

Wasserstein GAN 是 G必司 的一種變式，我們知道 GAN 的訓練是非常麻煩 的，需要很多訓練技巧，而且在不同的資料集上，由於資料的分佈會發生變化， 也需要重新調整參數，不僅需要小心地平衡產生器和判別器的訓練處理程序， 同 時產生的樣本還缺乏多樣性。 除此之外最大的問題是沒辦法衡量這個產生器到底 好不好，因為沒辦法透過判別器的 loss 去判斷這個事情。 雖然 DC GAN 依靠對產 生器和判別器的結構進行列舉， 最後找到了一個比較好的網路設定，但還是沒有 從根本上解決訓練的問題。

WGAN 的出現，徹底解決了下面這些困難：

( I ） 徹底解決了訓練不穩定的問題，不再需要設計參數去平衡判別器和產生器 ﹔

( 2 ） 基本解決了 collapse mode 的問題，確保了產生樣本的多樣性﹔

( 3 ） 訓練中有一個向交叉娟、準確率的數值指標來衡量訓練 的處理程序，數值越 小代表 GAN 訓練得越好，同時也就代表著產生的 圖片品質越高 ﹔

( 4 ） 不需要精心設計網路結構， 用簡單的多層感知器就能夠取得比較好的效果。 下面先介紹為什麼 GAN 會有這些缺點，然後解釋 WGAN 是透過什麼辦法解 決這些問題的。

* + - 1. GAN 的限制

根據之前介紹的，有下面的式子

max *V( G,D)-V( G,D\* )--2log 2+2.JSD( Pd* ， （*x)* II *PG(x))* (6.33)

從式 （ 6.33 ） 我們知道原始的 GAN 是透過最佳判別器下的 JS Divergence 來 衡量兩種分之間的差異的， 而且最佳判別器下 JS Divergence 越小，就說明兩種分 佈越接近，但 S Divergence 有一個嚴重的問題，那就是如果兩種分佈完全沒有重 疊部分，或說壘部分可忽略，那麼 JS Divergence 將值等於常數 log2 。換句話說， 就算兩種分佈很接近 ，但是只要它們沒有重疊，那麼 JS Divergence 就是一個常 敏，這就使得網路沒辦法透過這個損失函數去學習 ，因為它沒辦法知道它是否做 得好，這就會導致梯度消失， 同時這也使得我們沒有辦法衡量這兩種分佈到底有 多接近。

而真實分佈與產生的分佈沒有重疊部分的機率有多大呢 ？其實是非常大的， 直觀來講，真實分佈是一個高維分佈’而產生的分佈來自一個低維分佈， 所以其 實很有可能產生分佈和其質分佈之間 就沒有重疊的部分。 除此之外，不可能真正 去計算兩個分佈’只能近似取樣，所以也導致了兩種分佈沒有重疊部分。 如果判 別器訓練得太好，那麼產生的分佈和原來分佈基本沒有重疊部分 ，這就導致了梯 度消失﹔如果判別器訓練得不好，這樣產生器的梯度又不准， 就會出現錯誤的最 佳化方向 。如果要使得 GAN 能夠完美地收斂，那麼需要判別器的訓練不好也不 壞，而這個度是很難把握的， 況且這還依賴資料的分佈等條件， 所以 GAN 才這麼 難訓練 。

1. Wasserstein 距離

既然 GAN 存在的問題都是由於 JS Divergence 引起的，那麼能不能換一種度 量方式去衡量兩種分佈之間的差異，而不使用 JS Divergence ？答案是一定的，這 就是 WGAN 中提出的解決辦法。

首先介紹一種新的度量方式去度量兩種分佈之間的差異 Wasserstein 距 離，也稱為 Earth Mover 距離，定義如下

*W* （月 *P.)=* inf *E,.*V﹝ll x -yl﹞

' • y-Il(P, ,P (6.34)

） 的，

看上去可能比較複雜，數學解釋如下．對於兩種分佈 *P*，和f言 ，它們的聯合分 佈是TI *(P,,Pg*），換句話說TI *(P,,P* ），中每一個聯合分佈的邊緣分佈就是 *P*，或 尺 。

E且－－－ 第 6 章 產生對抗網路

那麼對每一個聯合分佈而言， 從裡面取權 x 和y ’並計算 x 和y 的距離，然值取遍 所有的 x 和y 計算一下期望，接著取這些期望裡面最小的作為 W距離的定義 ＠

如果上面的解釋不夠清楚， 也可以通俗地解釋， 固為它還有一個別名叫 Earth mover 距離，也就是推土機距離，這是什麼意思呢？可以把兩種封佈想像成兩堆 士，然接想想、如何用推土機將一種分佈暨成另外一種分佈的樣子， 會有很多種移 動方案，裡面最小消耗的那種方案就是最佳的方案，也就是這個距離的定義。

w 距離與 JS Divergence 相比有什麼好處呢？最大的好處就是不管兩種封佈是 否有重壘，它都是連續轉換的而非龔蟹的， 可以用下面這個實倒東說明一下， 如 圖 6.12 所示。

a t a d

p

。

圓圓圓圓”mMM揖

圓圓圓圓

。nu－

p

a t a

“

A

P

什lI

叫

ι

nMW

s

auu

p

a

a

搏”

P

向 －

o

nb

p

|  |  |  |
| --- | --- | --- |
| *JS( Pc*口， *Pdata )*  ::: *1092* | *JS( Pc*呵 *Pda*個）  = *log2* | *JS( Pc,00 ,\_Pdata )*  = u |
| *w( Pc Pdata )* | *w( P.*甸 *pdllla )* | *W(Pc,oo•\_Pdllla )* |
| = do | -- «-so | －一- |

圖 e.12 w m離實側

透過上面這個示範可以發現， 雖然兩種分佈更接近，但 JS Divergence 仍然是

1og2 • *W* 距離就能夠連續而有致地衛量兩種分佈之間 的差異 ，

1. WGAN

W距離有很好的優越性， 把它拿來作為兩種分佈的度量最佳化產生器， 但是 W 距離裡面有一個 infy ～H恨，*Pg*） 是沒辦法求解的。 作者 Martin 在論文附錯捏面 邁過定理將這個問題轉變成 7一個新的問題，具有以下形式：

*W*（九九）＝ -b 呻 *E,,.r,*﹝*f (x*）﹞ －*E,,.p\_[ f (x*﹜﹞

- *.I\.* I叭1,SK •

(6.35)

這裡引用了一個新的概念一 －Lipschitz 連續。如果函數 f 滿足 Lipschitz 連續 條件，那麼它就滿足下面的式子：

1 /(x,) /(x,)I主 *K l x*，一高 ︱ (6.36)

我們不希望函數的變化太快，希望函數 f 變化能比較平緩。那麼可以將上面 的式子改成 GAN

﹒

*W(P,.* 毛） 也 max叫D.1," *E* ＿，.﹝*D( X* ）﹞ －*E* 呵﹝*D( x*）﹞ (6.37)

也就是說建置一個神經網路 D作為判別器，希望 D輸出的變化比較平緩，在 實際計算中限制 D 中的參數大小不超過某個範圈，這樣就使得關於輸入的樣本， D 的輸出變化基本不會超過某個範園， 所以就能夠基本滿足 Lipschitz 連續條件。

所以最後建置一個判別器 D，滿足：

*L* = *E* ＿ ，，﹝*D( x*）﹞ －*E* 叮﹝*D( x*）﹞ (6.38)

盡可能取到最大， 同時D 還要滿足 Lipschitz 連續條件，獲得的 L 可以近似 為真實分佈和產生分佈的 Wasserstein 距離。原始的 GAN 做的是二分類的工作 ， 也就是對於真假圖片進行二分類，而 WGAN 做的是回歸問題， 相當於近似擬合 Wasserstein 距離。

最後最佳化產生器的時候希望最小化 L，這時候需要滿足 Lipschitz 連續條 件，所以需要做加權的修改，由於 W距離的優越性，不再需要擔心梯度消失的問 題，這樣就能夠獲得 WGAN 的整個訓練過程。

歸納一下，WGAN 與原始 GAN 相比，只改了以下四點：

( I ） 判別器最後一層去掉 sigmoid ﹔

( 2 ） 產生器和判別器的 loss 不取 log ﹔

( 3 ） 每次更新判別器的參數之後把它們的絕對值修改到不超過一個固定常數的 數﹔

一一－第 6 章 產生對抗網路

( 4 ） 不要用以動量為基礎的最佳化演算法 （ 例如 momentuem 和 Adam ）， 推薦使 用 RMSProp 0 前三點都是從理論分析獲得的結果， 第 （ 4 ） 點是作者從實驗 中發現的。對於 WGAN’論文作者做了不少實驗，獲得了幾個結論﹒ 第一 ， WGAN 如果使用類似 DCGAN 的結構，那麼和 DCGAN 產生的圖片差不多， 但是 WGAN 的優勢就在於不用 DCGAN 的結構，也能產生效果比較好的圖 片，但是把 DCGAN 的 Batch Normalization 拿掉的話，DCGAN 就不能產生 圖片了，第二，WGAN 和原始的 GAN 都是用多層全連接網路的話， WGAN 產生的圖片品質會變得差一些， 但是原始的 GAN 不僅晶質很差， 還有多樣 性不足的問題。

* + 1. Improving WGAN

WGAN 的提出成功地解決了 GAN 的很多問題，最後需要滿足一階 Lipschitz

連續性條件，所以在訓練的時候加了一個限制一一加權修改。 然而加權的修改只是一種簡單的做法， 不是最好的做法， 所以隨後有人提出

了一些新的辦法來解決這個問題。

首先提出一個定理： 一個可微函數如果滿足 I階 Lipschitz 連續，相等於它的 梯度範數處小於 1。用式子來表示就是﹒

II *D* IIL:5 I峙II *11,D( x)* II歪 I *for all x* (6.39)

有了這個定理，就能夠近似地這樣去表達 W 距離

*W*（ ，*P,*） 起 maxv ﹛*E* ＿，，﹝*D( x*）﹞ 互有﹝*D( x*）﹞

λ*E* ＿弘，，*max(0,11 11,D( x))* II 一I）﹜

(6.40)

(6.41)

不需要在整個分佈上都滿足 Lipschitz 條件，只需要沿著一些直線上的點滿 足這些，結果就已經很同時在實際中 採用的策略也不是取 m缸，因為不希望 IIti, *D(x)* II 太小，所以做的是最小化 （ II *11,D( x)* II 1)2，最後改進的 WGAN 就是：

*W*（月，*P,*） 也 m缸D﹛*E,-e.*﹝*D( x*）﹞ *E,-P* ﹝*D( x*）﹞

λ*E* ＿仙。I *ti,D( x))* II 一 1)'}

(6.42)

(6.43)

改進後的 WGAN 和改進前的 WGAN 相比，訓練更加穩定，產生的圖片效果 也更好。

I &.4 1 應用介紹

GAN 被提出兩年以後， 很多想法都被研究者們提出、 探索並實作，因此催生 出了 GAN 的很多應用，下面就來介紹其中的兩個應用 Conditonal GAN 和 Cycle GAN •

6.4.1 Conditional GAN

在前面的產生對抗網路學習 中，我們發現它能夠產生以假亂真的 圖片，但是 這些圖片並不一定是我們想要的 。例如手寫字型的圖片產生中，想要產生 4 ，卻 產生了 5 或者 9 ，這使得產生網路並沒有實際的作用， 因為它產生的圖片都是隨 機的。隨著這個問題的出現， 研究者開始研究如何能夠按照意願產生圖片 ，這便

有了 Conditional GAN 。

Conditional GAN 的應用是做文字產生圖片，例如列出一段文字 「 a dog is run­ ning 」’希望網路產生的 圖如圖 6.13 所示。Image caption 就是透過旋積神經網路和 循環神經網路給圖片加字幕，上面這個過程是它的逆過程。 這個逆過程與 image caption 相比難在什麼地方呢 ？

Image caption 透過預訓練的旋種神經網路分析圖片特徵，然後據此特徵透過 循環神經網路產生文字， 例如一張火車的照片，不管火車是什麼樣的， 預訓練的 旋積神經網路都能夠分析有效的特徵， 據此能夠獲得文字資訊 「 train 」。 但是透過 文字產生圖片就面臨著一個很大的問題一一圖片的多樣性，例如列出 「 train *1* 這 個單字，需要產生一張火車的 圖片。

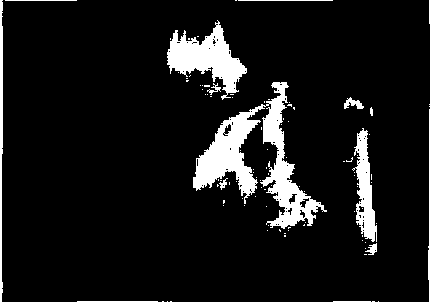


圖 6.13 奔跑的狗

圖 6.14 所示的是 4 張火車的圖片。

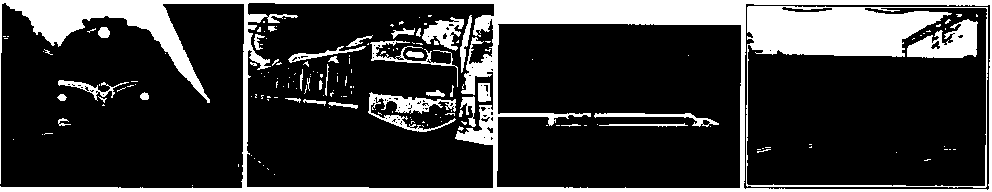


圖 6.14 火車

這樣並沒有很好的辦法定義損失函數， 如果單純地將圖片作為目標，那麼最 後的結果就是圓 6.14 的平均，全是馬賽克，沒有實際的內容。

使用產生對抗網路能夠比較好地解決問題，這就是 Conditional GAN 。原理 非常簡單，正如它的名字一樣， 就是普通的產生對抗網路， 只是加上一個條件， 例如要產生火車的影像， 就是將 「 train 」 作為一個條件放到普通的產生對抗網路 中 ，可以用圖 6.15 抽象地表示。

附晶tribt』t ’．



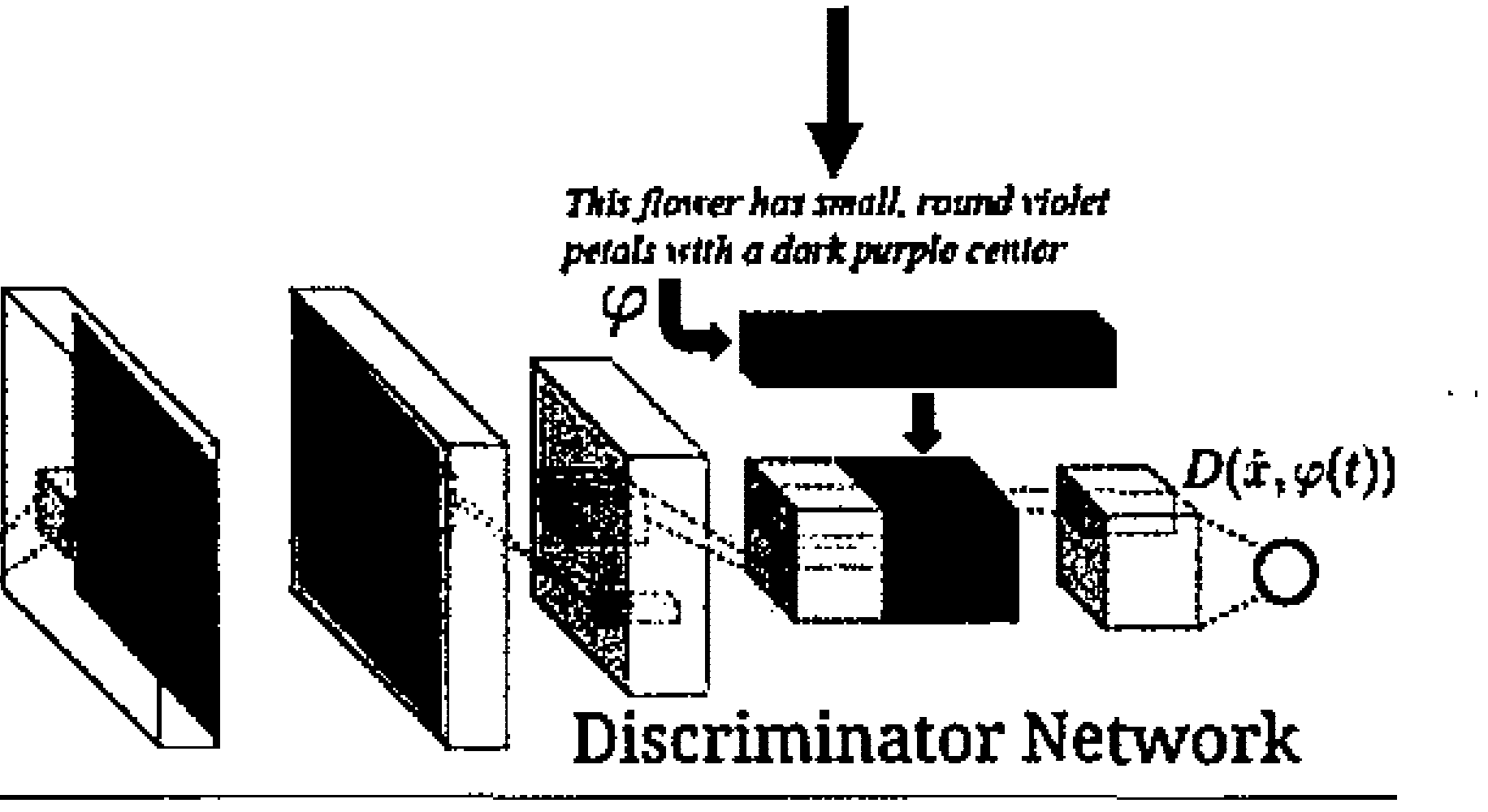
區壘司 ．

圖 6.15 c。r】 diti。nal GAN 的基本思想

以上表示*T* Conditional GAN 的基本思想，圖 6.16 引自於 NIPS 20闊 的論文

「 Learn- i時 What and Where 個 Draw 」 這張圖片展示了實際的產生網路和判別 網路。在產生網路中 ，將列出的文字資訊和幢樓向量拼在一起作搞產生網路的輸 入，接著產生一張圖片 在判別網路中 ，也會將文字資訊和 中間分析的特徵圖拼 起來，最後輸出結果表示真假。 其中正樣本就是文字和它對應的圖片，負樣本是 文字和產生的圖片 ，以及真實的闡片和不對廳的文字 。

文字內容 文字內容



m曲S闕 他叫個且剛..， "" "'

﹛

戶1帥 叫 a .；，，，. 岫 是 E U ，ip(t))

IJ1

rp... .,p(I﹜ h

S 咐I）直：＝駐

Generator Network

圖 6.16 產生網路和判巨u輔臣

訓練完 Conditional GAN 之後，只需取出產生網路， 輸入文字作為悔件就可 以產生想要的圖片 ，效果如圖 6.17 所示。

圓圓一位置團團

圓圓一回墮國﹒ 圓圓圓圓圓圓 圓圓一臨國團團 團圓圓開團團 圓圓一個團團團 向圈圈團一包圓圓



圖 6.17 產生的圖片

6『29

6.4.2 Cycle GAN

Cycle GAN 之所以被提出，是因為人們能夠根據一個人的作品， 想像他畫出 的其他場景會是什麼樣的， 也就是說雖然沒有對應的資料， 但是卻能獨實現這種 對應的對映。

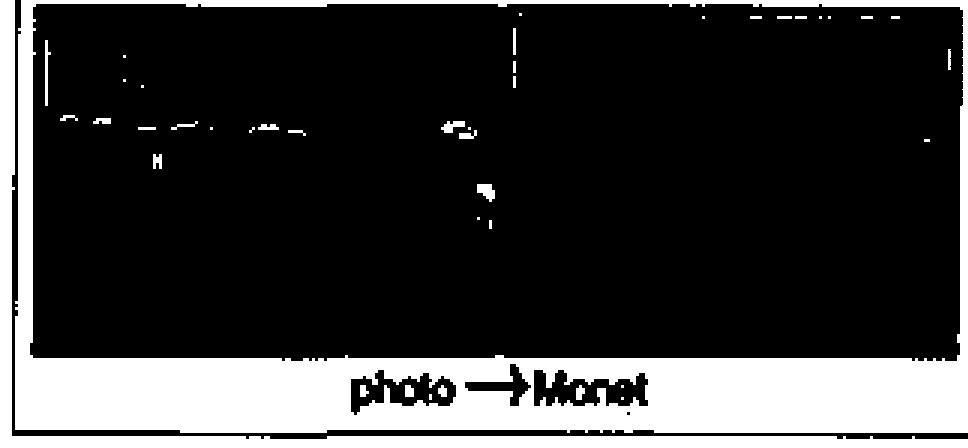
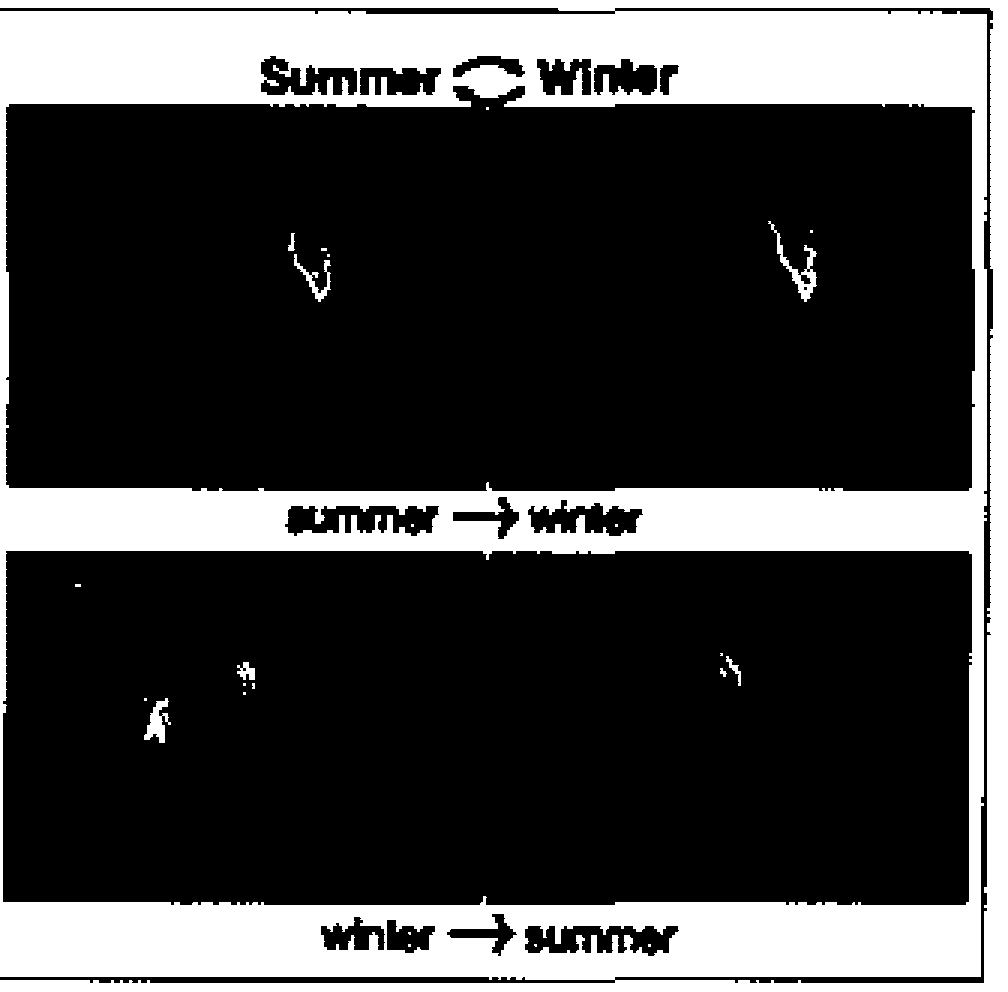
那麼機器是否可做到這一點呢？

人額之所以能夠做到這件事， 是因為在兩個頡域 恆，Y） 之間 ，存在著一種底 層的關係’對於同一個事物，如果有兩種不同的對映之後的表達， 那麼這隔種表 達之間就是針對同一個事物的一種連箱。 如果僅使用一種連結對它進行單向對 映，那麼就無法保障這個對映是單一的， 也就是說對於同一個事物，可能會將它 對映到兩個不同的表達。

針對這個間題，Cycle GAN 提出了一種自然的解訣方法， 不僅要求單向對 映，還要求一種雙向的對映， 倒如 G : X→ *Y,F : Y* → X兩個對映，我們希望 *F(G(X))::::X ,G(F(Y )):::: Y* •

上面就是 Cycle GAN 提出的原理和動機， 先放幾張 Cycle GAN 的故果間， 如圖 6.18 所示。從圖 6.18 可以看出，Cycle GAN 將冬天和夏天互換，將薩馬和馬 進行相互轉換。其實這個想法並不是 Cycle GAN 獨有的，同時期的 Disco GAN 和

的重點有所不同 。



Dual GAN 都有著相做的思想，只是它們

﹔可

-

cPho

岫－

二

z.tw.C 枷

a曲1

閉

ι

」

....國－＋且聽間

圖 6.18 Cycle GAN 強果圖

占

目

7

那麼實際 Cycle GAN 的網路結構是怎樣的呢？ 下面先從 GAN 開始，逐漸發 展到 Cycle GAN °

根據最基本的產生對抗網路， 可以透過圖 6.19 所示的結構來完成所需要的 工作。

····

··

··

μ

》h

i p

—p

軍 事

b

h

I l

戶 h i

們 f

w

（

圖

！

X

—

V

IJ

—

：



－「P l： 1 一

～－一＼伊甸－

是一

a －

E

f

＆

－

a

－

q

p

－i

J

I

Fii～j

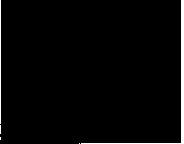
一戶可nvrLr巴

．

→

．FU－－

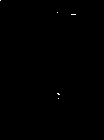
：

國

Y

···

1



．．D．

－

－

』

－

t

』

一

一自

Y．．

．

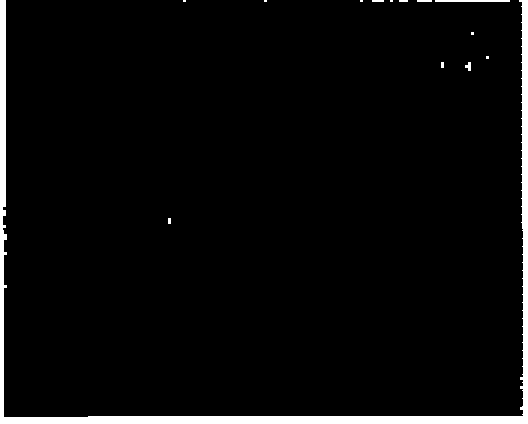
圓 6.19 產生對抗網路的工作示意圖

將原始圖片輸入網路，然後透過判別器的判斷，使得它的風格盡可能接近我 們想要的風格。但是這樣會出現問題， 在產生器不斷訓練的過程中，原始圖片透 過產生網路會失去資訊， 變成別的圖片 。例如圖 6.19 輸入的是一張河道的圖片， 可能透過產生網路之後會變成一張梵古的畫像， 這正是因為單項對映無法保障單 一性。

那麼 Cycle GAN 是如何做的呢？正如前面介紹的， 建立一個雙向對映如圖 6.20 所示。透過建立兩個產生網路進一步建立一個雙向 對映，這樣就避免了資訊 的遺失，使得原始圖片能夠被對映到想要的效果並保 留主要資訊。

產生對抗網路作為近年來深度學習 中升起的新星，近年來被大量的研究者研 究，它的變式和應用不僅如此，有興趣的讀者可以進一步閱讀相關論文。

|  |  |
| --- | --- |
|  |  |
| ---e  I '- *:* ：圓心目 | as possible |

國

as clos

*T*

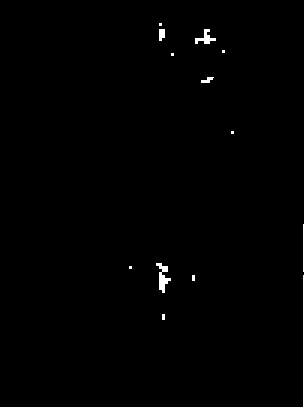
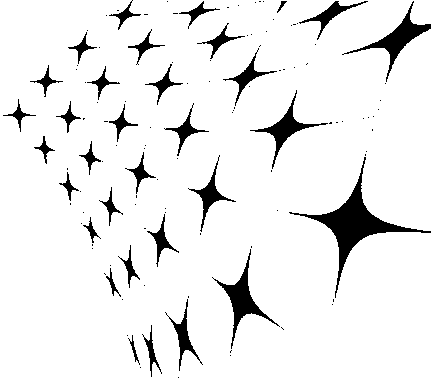


圖 6.20 單向對映

本章從產生對抗網路的基本思想開始， 接著介紹了數學原理， 從數學原理 出發推導出它的限制，進一步引用改善的產生對抗網路， 最佳介紹了產生對抗網 路的一些變式和應用， 下一章將透過賈戰說明實際介紹深度學習在實際問題中的 應用。

第 7 章



###### 深度學習實戰

機器學習對現實世界中的很多問 題都實現了突破性的進展，而機器學習中發 展最為迅速的深度學習無疑領導著整個人工智慧領域的變革， 也許人類因此能夠 製造出越來越聰明的機器。

前面介紹了深度學習 中的基礎知識，也實現了一些賞例，但是真正要掌握的 是：如何在現實中運用深度學習的技術解決實際問題。 從資料的收集、 模型的架 設、接著不斷調參訓練模型、 最後獲得結果，經歷完整的過程對深刻地了解深度 學習大有梅益。透過實際場景的應用，更加明白深度學習能夠處理什麼類型的資 料，了解其演算法的侷限 ，知道如何根據不同的資料集調整網路結構， 同時也能 夠加強工程能力 。這一章透過四個實戰練習介紹深度學習在實際 中的運用。

1 1.1 I 實例一一一貓狗大戰： 運用預訓練旋積神 經網路進行特徵分析與預測

這是 Kaggle 在 2013 年的比賽，在這個比饗中，需要寫一個演算法來分辨 圖 7.1

中的動物是貓還是狗 ，這對於人類而言很容易，但是對於機器而言，也許並不簡單。

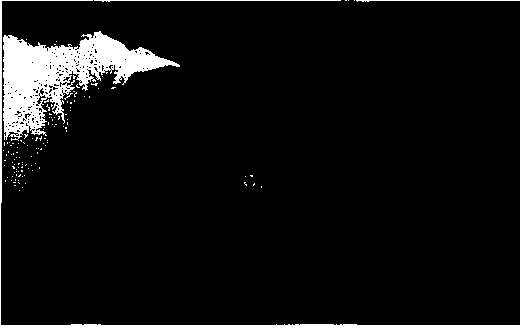


圖 7.1 貓與狗

7.1.1 背景介紹

網路保護通常面臨一個難題， 就是它應該讓人能夠很容易地識別， 而讓機器 無法識別，例如驗證碼，它可以有效地減少垃圾郵件，還能夠防止使用者的密碼 被惡意破解。

Asirra 是一個影像識別機制的驗證碼， 需要人們根據圖片來辨認是貓還是 狗，這件事情對於電腦而言是特別難的，但是對於人而言是非常簡單且準確的， 很多人甚至認為這件事很有趣 。Asirra 有很多不同的貓和狗的照片，因為其合作 者是一家致力於1話流浪動物尋找家園的網站， 它們為 Asirra 提供了三百萬張貓和 狗的圖片，接下來要使用它的子集作為訓練集。

對於該網站的攻擊，隨機猜測是其中 種，但是依據影像識別演算法能夠使 得猜測的準確率變得更高 。而圖片有無窮多種變化，例如背景、角度、光線等， 這使得識別變得非常困難。 很多年前，電腦視覺的專家列出單張的準確率只有 60% ，而且提升齣旱越來越困難。

到了現在，傳統的機器學習演算法能夠達到 80% 的準確率，但是這並沒有達 到極限，深度學習中旋積神經網路的出現使得現在的識別工作有更大的提升， 下 面就以 80% 作為基準，使用旋積神經網路達到更好的效果。

7.1.2 原理至于析

對這個問題，根據之前學習的旋積神經網路， 可以寫一個簡單的網路模型， 但是會發現自己寫的網路效果並沒有那麼好， 不僅準確率不高，甚至有可能收 斂速度還很懂， 或不收斂。這個時候，使用一些成熟的模型， 例如 VggNet 、 GoogleNet 、ResNet 等就可以幫助我們解決問題。這些成熟的網路都是由主管全球 深度學習技術的實驗室做了無數次實驗而實現的優秀的網路， 而且透過 ImageNet 比賽獲得過冠亞軍的成績，所以使用這些網路， 對於結果有一定保障。 現在深度 學習的門檻越來越低，一方面得益於現在的架構讓寫網路變得很簡單 另外一方 面就是這些實驗室願意開放原始碼共用他們 的模型和實驗結果。

可以使用現成的模型直接來訓練的資料， 從已有的模型出發再不斷地微詞。 但是這會帶來一些運算資源的問題，因為對大數據集而言，每跑一次實驗結果就 需要耗費很多運算資源， 作為學習者，沒有那麼強大的資源供我們使用， 這個時 候是不是就沒有辦法了呢 ？其實辦法是有的，那就是移轉學習。透過移轉學習 ， 讓沒有太多運算資源的人也告別順利質現深度學習 中複雜模型的訓練。

移轉學習

在機器學習的經典監督學習場景中，如果要針對一些工作 A 訓練一個模型， 會通過提供工作 A 的資料和標籤來訓練，現在已經在指定的資料集上訓練了一個 模型 A ，並且期望它在間一個工作和未知數據上表現 良好。在另外一種情況下， 當指定一些工作 B的資料和標籤時，也可以根據工作 B來訓練我們的模型 B ，這 些都是很合理的，如圓 7.2 所示。

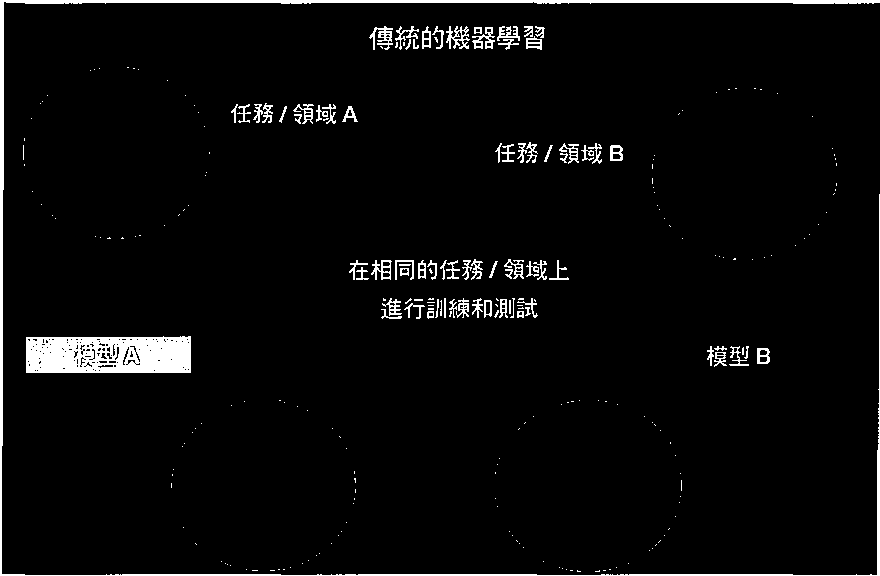


圖 7.2 傳統的機器學習 但是對於一個特定的工作， 如果沒有來自該工作足夠的資料集， 傳統的監督

學習就無法支援了。移轉學習允許透過借用已經存在 的一些相關工作的標籤資料 來處理這些場景，把解決相關工作時獲得的知識儲存下來，並將它應用在我們有 興趣的 目標工作中，如圖 7.3 所示。

－－－－－ 第 7 章 深度學習實戰

正是由於這個原因 ，我們便可以使用 ImageNet 中預訓練好的網路進行移轉學 習了，因為網路透過完成 ImageNet 比賽，能夠獲得知識，也就是網路中的參數， 而 ImangeNet 中也有很多貓和狗的圖片，所以我們認為網路在 ImageNet 中獲得的 知識能夠進行移轉。

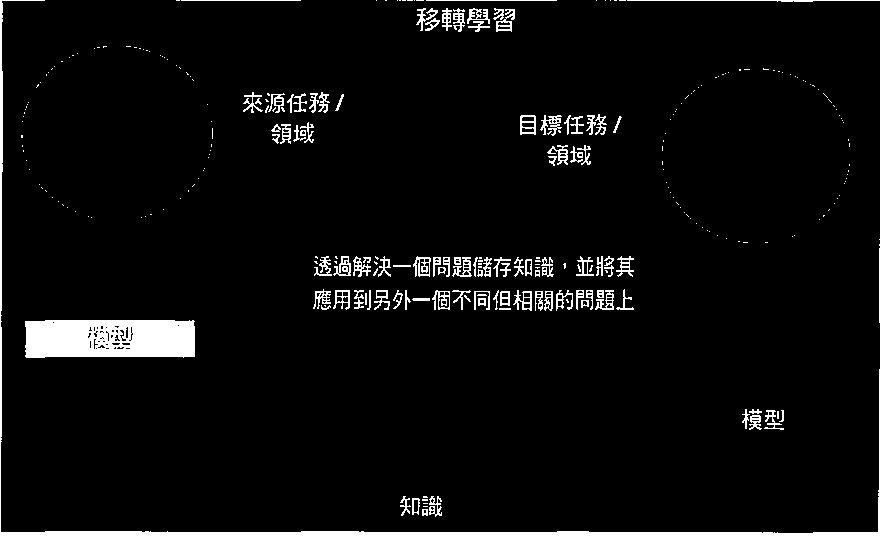


圖 7.3 移轉學習

旋積神經網路可以視為兩個部分： 前面的旋積部分和後面的分類部分， 而前 面的旋積部分主要做的事就是分析圖片特徵，而預訓練好的網路對於圖片的特徵 分析效果是非常好的， 這是因為網路學到了需要的參數，我們可以直接用預訓練 的網路旋積部分來分析我們自 己圖片的特徵，而對於我們自己的工作，也就是貓 狗二分類 ，就用我們自己的分類全連接層就可以了 。

以上就是移轉學習在影像識別中 的一種應用，是不是特別簡單呢 ？歸納起來 就是將預訓練的網路移轉過來，然後訓練過程中只更新最後的全連接層部分的參 數，實現最後我們自己工作的分類目的 。

最後強調一下，移轉學習並不是任何時候都能夠使用的， 前面我們提過，需 要它們完成的工作是相關的，所以移轉學習在相似資料集上的應用效果才是 良好 的，例如你用的預訓練的參數是自然景物的圖片分類獲得的，那麼使用這些參數

來做人臉的識別，效果可能就沒有那麼因為人臉的特徵分析和自 然景物的特徵分 析是不同的，所以對應的參數訓練後也是不同的。

實現方法

在程式實現中，我們會層層遞進。 第一種方法是匯入預訓練的旋積網路， 將 最後的全連接層改成我們自 己設計的全連接屑，然後更新整個網路的參數，最後 能特別快地達到收斂﹔第二種方法是鎖定前面旋積層的參數， 讓網路訓練只更新 最後全連接的參數，這樣可以讓我們訓練時間大幅減少﹔前面兩種方法都是用一 種預訓練好的網路， 第三種方法中我們使用多個預訓練好的網路， 將它們並聯在 一起，圖片經過每個網路都會獲得特徵圈 ，我們將這些特徵圖連接在一起進入最 後的全連接層。

7.1.3 程式實現

道 1資料前置處理

資料集可以去 ht恥：／／[www.kaggle.com/c/dogs](http://www.kaggle.com/c/dogs) 耐－cats/data 下載，並解壓整個檔 案，因為裡面的貓和狗的圖片是混在一起的，所以我們需要透過前置處理將它們 分成兩個資料夾。

我們先建立兩個資料夾：一個是訓練集資料夾， 一個是測試集資料夾。 這兩 個資料夾內部都有兩個資料夾： 一個資料夾中放狗的圖片 ，一個資料夾中放貓的 圖片。

按著我們將貓和狗的圖片分別移動到對應的資料夾中 ：

1. data file 。S listdir ( ' /media/she rl。ck/Files/kaggle d。g\_vs\_cat/zip’﹜
2. ct。g\_file list (filter (1 da x, x ﹝ ，3﹞＝＝’d。9’，data\_file ﹜ ﹜
3. cat £,le a list (filter ﹛ lambda x. x ﹝ 3﹞＝＝＇cat’，data file） ﹜

我們將解壓之後所有的圖片放到 zip 資料夾中，第一行程式是分析出資料夾 中的所有圖片的名稱，然後第二行和第三行程式分別是分析出狗和貓的圖片，其 中使用了 創t缸，這個函數能夠將滿足條件的資料分析出來， 上面就是將圖片名字 是 dog 和 cat 的圖片分別取出來，存為兩個 list 。

．圖阻 草 7 章 深度學習實戰

接著我們將貓和狗的圖片分別啟動到訓蟬集和驗證集中，其中 90% 的竇料作 為訓練集，10% 的圖片作為驗證集，使用 shutil.move（） 來移動圖片。

* 1. root ＝’／media/sherlock/Files/kaggle\_dog\_vs\_cat/ '
  2. f口z i in range ( len ﹛ dog\_file﹞ ﹜：
  3. pie\_path = root + • zip/ ' + dog\_file ﹝ i﹞
  4. if i < len (dog file﹜＊0.9:
  5. obj\_path = train＿目前 ＋ I/dog／ ’＋ dog\_file ﹝ ii
  6. else:
  7. obj\_path = val\_root ＋’／ dog／冒 ＋ dog\_file[iJ
  8. shutil .move (pic\_path’，bj\_path) 9

1. f or i in range ( len (cat\_file﹜ ﹜ z
2. pie\_path = root ＋﹒zip／’ ＋ cat\_file ﹝ i﹞ 12 if i < 1自n (dog\_file﹜＊ 0.9:

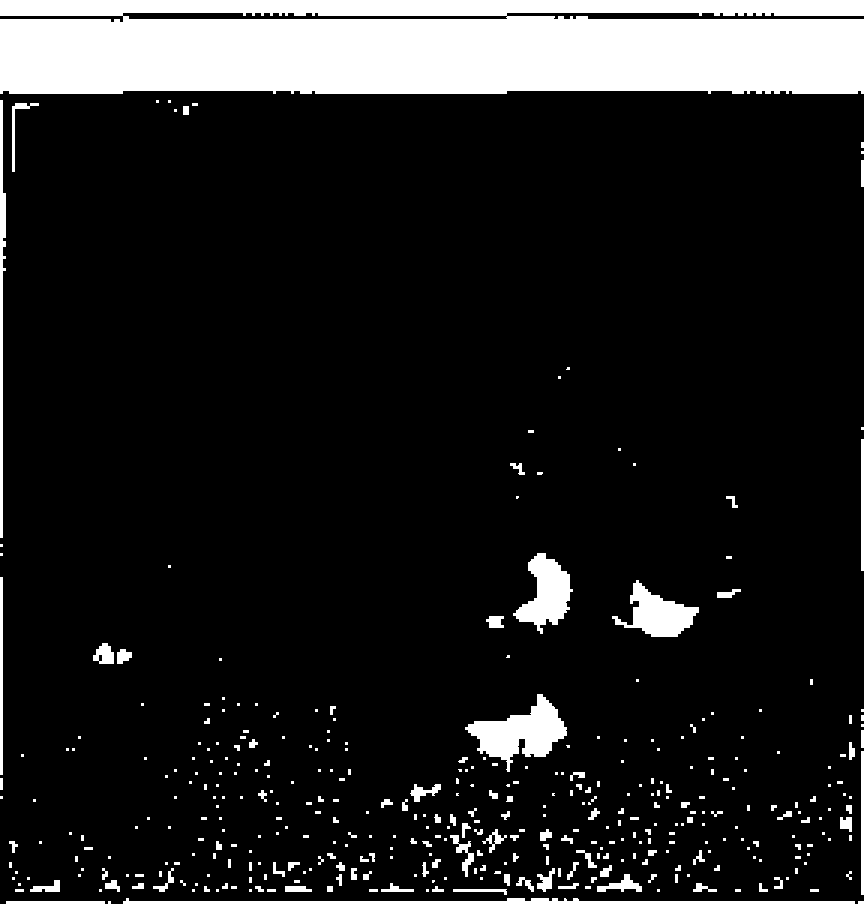
13 obj\_path = train\_root + • /cat ／ ’＋ cat\_file ﹝ i) 14 else:

15 obj\_path = val\_root + • /cat ／﹒ ＋ cat\_file ﹝ i] 16 shut斗 move (pic\_path。bj\_path ﹜

這樣就完成資料前置處理部分了， 可以用視覺化圖片來幫助我們了解剖 褲資 料，如圖 7.4 所示 ，

接下來我們就可以開始移轉學習的模型訓練 7 。

。 。



|  |  |
| --- | --- |
| 100 | 50 |
|  | 100 |
| 200 | 150 |
|  | 200 |
| 300 | 250 |
|  | 300 |

400

。100 200 300 400

35口

0 100 200 300 400

圖 7.4 串轉學習的模型訓練

7”6

道 2. 方法一

首先我們使用第一種方法進行移轉學習，對於預訓練加權的模型， PyTorch 將常見的模型和預訓練的參數都已經存在了網上， 所以我們不需要自己再去寫網 路在 Ima geNet 上訓練模型了，所有預訓練的模型都在 torchvision.models 裡面， 可以透過下面這樣定義預訓練 的模型。

1. transfer rncdel 呈 rn。dels. resnetlB (pretramed巨True)
2. dim 斗n 旦 transf er rn。del. fc in fea七ures
3. transfer rn。del fc nn Linear ﹝ dirn\_in, img\_classes)

這裡使用的網路是一個簡單的 18 層的殘差網路 （ ResNet ）， 將最後的全連接 層換成了適用於我們資料集分類的全 連接層。

接著就可以開始訓練了，可以發現經過一次資料反覆運算， 訓練集準確率就 達到了 88% 左右，驗證集準確率都達到了 97% 左右，使用沒有預訓練的網路第一

次反覆運算之後準確率應該 只有 60% 左右，從中可以看出使用移轉學習的好處， 能夠相當大地減少運算資源。最後經過 10 次反覆運算，驗證集能夠達到 98.7% 的 準確率，如果希望實現更高的準確率，

則可以使用更複雜的預訓練模型，然後調整全連接層的參數，並且使用更加 複雜的影像增強，結果如圖 7.5 所示。

圓圓 ， 吵

，，

E

E

C

—

、

R

1

4

←

3

1

、

r

v

4

5

—

l

言

—

U

‘

4

H

‘

H

O

ι

4

4

3

1

“v

F

g

r

J

I

J

ι

j

T 罰

J

〈

c

r

r

r，一，

’

'

”

、、

r O 吵 吵 吵 才 有 、』

』

凶

”ι。

o

：三 一

l

＆－

！

唸』3 3 J巾 J

－

，－

－，

、3 J ‘

～

』』

』』

i

、’

』

“q 芯， 3 b

F

司‘‘怒之、U、v 削U

－－ d

一吋－〉－一〉－－＝干

『

：

l r

’”

叮 、

d F

圖 7.5 方法 的訓練結果

川

O

h

u

i

1

F

♂

三

1

r

n

1

—

n

—

『

d

t

f

f

之

i

1

，J

司

已—

o

r t

e

t

y

f

t

3

t

戶

Y

T

I t o B Q Q

— 〕 l i i I 1 ny

8 月

I

－4 3 J JJV一司 司 u－7 3

l 1 1

“心

C 〈 〈 〈 〈 ι i「 ：

、

』

凶 】

。。

”

t

t － ，

－ 2 」 J一、3 3 3

tJ

1 1 1 l i i － 3 3

n、！td

1

－

一

、一 －

d 一 －

3 斗－

d

〈

－ ←

l

2

r

→t

心，

、、－

－』，－’、、3 、3

』們

一－

”

4

4司

－ ， 一

》－－

L －

吋4 d背 34﹜ J J J 、吋

λH O 叫 HV 川Vυou－ 0

3

C

﹜

Pt P FF

－t－

－』←－J J J 可心 “

．－－ 第 7 章 深度學習實戰

誼 3. 方法二

方法二跟方法一本質上是相同的， 只是方法二會固定住旋積層的參數， 只更 新全連接層的參數，這在 PyTorch 中的實現也很簡單。

* 1. f。z param in transfer\_m。del.parameters () .
  2. param.requires grad 詰 False

4 。ptimizer E 。ptim.SGD ( transfer\_model fc.parameters ﹝ I , lPle-3)

3

首先分析出模型中的參數，然後將其設定為不求梯度，最後在最佳化器中設 定需要最佳化的參數只有全連接層的參數， 要注意不能將旋積層的參數也放進最 佳化器中 ，不然會顯示出錯，這樣就只需更新最後的全連接層參數。

因為不需要更新旋積層的參數， 所以訓練的時間可以大為縮短， 同時驗證集 的準確率一開始就能夠達到 96%’但是有得必有失，網路更新的參數變少，這導 致經過 10 次反覆還算驗證集的精度只有 97%’沒有所有參數都進行更新的精度 高，結果如圖 7.6 所示。

1/1心 1方／ 10

T r<.1t o

圓

t:I

L ,r·.

L .<

, ,-,

句＂1

9t·V,

＇:'I ＇（） 3

1·－，》

Ill!

。

，呵 7 'i)

j S(,')C.f可 丸屯

*(,* l 文C叫＇ 可﹒（ (

村 》 1'· *(,*

*I)* •Jl, j

l *•x.* 】 、 0 9 ］、q

l *f*

0

，）

O ;R--l40<,

1 l '·

• 方心 I ，于4

.:.,,

＿.， － 《

」＂

•)

仇

,:; －寸

l﹜ 守 1～.. •

( :,, ii ＇＂、*3*

' I\ 峙 ？、仆n

*\_(* '· ' *,:,* Jι

l t',"i 4ii:

,,- (

o) . 向1 "

Lυ

仇 l ''

L o · ,1 '.d j,) ι哭 ＇ ， 可 CC•'

l

﹔

，

t .?13.:' 3

汁 可

L I﹜

"' ( •）.（，•什υ

l•. t﹜ 「 ﹜吵 1 戶《 （ ﹛J 可 *n*廿

1

|  |  |  |
| --- | --- | --- |
| 」4 「← | 主. | ,,,. ,. |
| 吉 JC 汁川，＿ |  | ，＿.，，＿ ι |

le• f勻 '.-I L豈

＇刁

叫

i

i引 3勻、于司 J ← ﹛ 〈 、可 q ,

L u I i 4

、

L﹛ L﹛

L －

九. ＼

＂志 、”，句I 一（《 仆 ＂

.-., ,

, ＞九＇.＇4 J 4 心 ＇ ＇ 之--17 T ＼ 「

，

H,,t 川﹜

ι仁（

l i r

←'...

L﹛ ＇） 13V.•1

... ，：＿ ，～

圖 7.6 方法二的訓練結果

油 4. 方法三

在第三種方法中，我們組合了多個預訓練網路， 都將它們的旋積層參數固 定，只更新最後的全連接層的參數， 相當於是方法二的衍生， 在每一次反覆運算 中 ，都需要將圖片前向傳播，透過旋積層，到全連接層，最後輸出結果，按著進 行反向傳播更新全連接層的參數。 如果資料集特別大，這樣做效率特別低，因為 旋積層的參數沒有更新，所以每次反覆運算中資料集正向傳播經過旋積層的結果

是一樣的，所以沒有必要每次都進行正向傳播，只需要將資料集一次反覆還算中 正向傳播經過旋積網路的結果儲存起來就可 以了，這個結果成為特徵向量 。

首先定義好特徵分析的預訓練網路如下：

1. class feature ne七 ﹛ nn M。dule) ,
2. def init一一 ﹛ self , m。del) ·
3. super (f eature\_net, self ) imt ﹛）
4. 斗f m。del ＝＝ ’vgg’．

4

1. vgg m。dels vgg19 ﹛pretrained True)
2. self .feature a nn .Sequential ( •list ﹛vgg children () ） ﹝ －1﹞ ）
3. self.feature add\_module （’gl。hal average’，nn AvgP。。12d (9))
4. ellf model aa ’incept 立 nv3'
5. incepti。n a m。dels incept 斗。n\_v3 (pretrainedaTrue﹜

11 self .feature a nn Sequential ﹝ •1ist ( inception .children （） ） ﹝ l﹞ ）

1. self .feature . modules pop ﹝ ＇13，﹜
2. self . feature add\_mcdule ﹝’global average ，nn 』vgP。。12d ( 35))

14 elif model aa ’resnetl52 '

1. resnet a m。dels resnet152 ﹝pretrainedaTrue)
2. self f eature a nn Sequential ('list ( resnet children （） ） ﹝ 1﹞ ）

17

。

18 def f rward (self , x) , 19

20 m。del includes vggl9, incepti。nv3, resnet152

21

1. x a self . feature (x﹜
2. x a x .view (x size ﹝ 0) , 1)
3. return x 25

26

1. class classif,er (nn .M。dule) .
2. def init ﹛ self , dim, n classes) ,
3. super (classifier, self ) init ()
4. self .fc a nn Sequential (
5. nn . Linear ﹛ dim, 1000) ,
6. nn ReLU (True) ,

。

1. nn . Dr pout ﹛ 0 .5) ,
2. nn .Linear (lOOO, n classes﹞ 35

36

。

1. def f rward ﹛ self , x﹜ ：
2. X a self .f C (X﹜
3. return x

feature net 就是特徵分析的網路， 接受的參數 model 有 V眩，inceptionv3 • r目net 紹，分別表示用的是 19 層 Vgg 網路，Inceptionv3 網路或 152 層的殘差網 路作為預訓練的網路， 然後將網路的全連接層去掉， 最後在特徵分析的旋積層的 最後一層加上一個平均池化層， 將結果轉化為特徵向量。

classifier 是對於我們自己的資料集的全連接分類層， 最後的分類結果是兩 種，其中使用 Dropout 防止過擬合。

然後就可以將資料登錄到網路中 進行正向傳播獲得特徵向 量，對於訓練集和 驗證集都需要進行正向傳擂， 一共要使用三個預訓練的模型，最後可以將結果存 成 的 檔案。

* 1. featurenet - feature net (mcdel)
  2. if use gpu ﹒
  3. featurenet. cucla (I

。 。

* 1. feature map a t rch Fl atTensor ﹛）
  2. label map - t。rch LongTens。r (I
  3. f。z data in tqdm (datal。ader ﹝phase ﹞ ）：
  4. 1mg, latel a data
  5. if use gpu
  6. 1mg a Variable (1mg, volatile-True) .cuda (I
  7. else,
  8. 1mg - Variable (1mg, volatile-True) 12 out a featurenet (img﹜

1. feature map a torch cat （ ﹝ feature map ut.cpu () data﹜ ，0﹜
2. label\_map ＝ 七arch.cat ﹛ （ label map, label) , 0)
3. feature map - f eature map.numpy ()
4. label\_map a label map numpy (I
5. frle name E ’feature ﹛ ） hd5「 .f。rmat (mcdel﹞
6. h5\_path a 。s.path . j 。in ﹛ outputPath, phase) + file\_name
7. with h5py. File ﹛ h5\_path’，w’） as h.
8. h create dataset ﹛’data ’，data-feature map)
9. h create\_dataset ﹛ ＇label ，dataalabel\_map)

feature net 是前面定義的特徵分析網路， 將每種網路輸出的特徵向 量儲存到 的 檔案中 ，注意在分析特徵向量的時候，資料是不能隨機打亂的， 因為使用多個 模型，每次都隨機打亂就會造成標籤混亂，然後將 h5 檔案儲存到 目前的目錄下， 最後透過下面的方式將特徵分析出來，定義一個由特徵向量組成的資料集。

1 class hSDataset ﹝ Dataset ) ·

2

1. def rnit一一 ﹝ self , hSpy list﹜﹒
2. label\_f,le hSpy File (hSpy\_lis立 ﹝ O﹞ ，主，）
3. self .label torch.f rom nw睡y ( label file ﹝ label ＇﹞ .value ﹜
4. self nS帥ples a self .label size (O )
5. temp\_dataset ＂七。rch .Fl。atTens。r ﹛）
6. 主 r file in hSpy list ,
7. hS frle " hSpy. File (file, '玄，﹜

。 。

1. dataset " t rch fr m\_numpy (h5\_f'1e ﹝ v data’﹞ .value﹜

。

11 temp\_dataset " t rch.ca七（ ﹛ temp dataset, dataset﹜，1) 12

13 self.dataset " temp\_dataset

14

1. def len ﹛ self ) ,
2. return self .nSamples

17

1. def getitem (self , index) .
2. assert index < len ﹛ self ﹜ ，index range error ’
3. data 呈 self .dataset ﹝ index﹞
4. label a self label ﹝ index﹞
5. return ﹛ data, label﹜

這個資料集的定義在前面 PyTorch 基礎部分已經講過了， 如果忘了，可以去 前面的部分看一下， 它是繼承於 torch.utils.data.Dataset 這個基礎類別，只需要簡 單地將每個 的檔案開段，然後分析出每張圖片使用不同的預訓練網路獲得的對應 的特徵向量，將它們連接成一個特徵向量就可以了。 每種網路大概需要 10 到 20 分鐘匯出特徵向量，如果不能使用 GPU 加速，那麼時間還會更長，電腦規格不高 的讀者可以下載我已經匯出的特徵向量， 在百度網路硬碟中，地址是 https ://pan.

baidu.com/s/1 c2lci IU 0

最後我們使用這些特徵向量來訓練一個 2 層的全連接網路就可以了， 可以發 現收斂特別快，而且獲得的結果特別好， 一次反覆還算之後在訓練集上的準確率 是 95%’在驗證集上能夠達到 99.6% 的準確率，因為驗證集上的資料較少，所以 獲得的準確率也相當高， 最後在訓練完 20 次反覆運算之後， 訓練集的準確率是 99.52%’單車證集的準確率是 99.68%’這大大好於之前只是屑一個小型預訓練網路 的結果，同時訓練所需要的時間也大幅減少， 每次反覆還算只需要 l秒鐘就能夠

完成，這個可以看作將三個訓練好的網路整合在一起組成一個大的網路， 結果如 圖 7.7 所示。

!l.m:..:..,

—

4

4

J

4

o

2

電

—

〈

t

t

f

l

A

R

司

九

3

r

J

b

曰

‘

9

：

9

e

6

1

q

t

3

♂

一刀

一、

r

o

O

O

r

S

Z

O

3

9

lH

叭叭

的

h

m

2

r

4

」、

，

1

1

1

7

1

1

1

1

1

h

l

〉

i

s

f

l

、、

可

3 lE

’

、

h 3 J

－

J

J O h

P

” ”

怖

。 3 叫

“

3 0

、 〈 （ （

1

、 A

MV 可 司 ，1

1 9

、

1 1

0 J A Q A叮 O N

i

1ik

l F

1 ） J P －

J J

\* 、 心J iJJ

J

T L L Lt L I Ll

1

。可

4 甸 、

CG －3 4 h－E E AV

OHVO O 。。

9 CJ hy U

、

C 〈

4 r

〈

— v hHR H 戶

、

令 ：

。 可

s o

7 1

l J J I － 1 1 1huo

o o o oo o oo to

d

c 〈 〈

〈

O O

IL－ － L L LLVL

圖 7.7 方法三的訓練結果

R

開

1

H

I

—

h

H

O

b

l

wn

O

H

H V

M O

O

J

U V

V

A V

V

v

也

3

7

4

3、

1

1

恥

1

d

t

G

l

o

o

o

d

O

戶

的 內 ω

、 仇。

付 1

J

4

L O

E

圳 叫 川W MH

v

o o

“

— ι

c

－

4咱 叫 咱 的 只

’

－

句

。。

叭 。

”

j

I

t

hu H h 吟。。

h － e

n u

”

q

o

－－

已

倒

！ ！ t C A

t、 《 〈

戶 μ

。

1 8

拈 制 制

帥

t

’ 〈 rJ

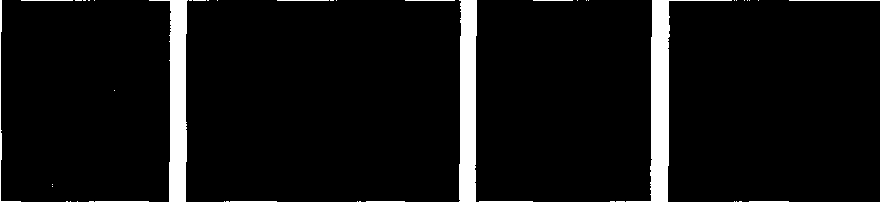
p

7.1.4 歸納

透過這個實例的學習 ，我們了解到如何從資料開始進行分析， 然後到模型的 建立，最後進行結果分析的整個建模流程。 我們掌握了移轉學習的基本思想，同 時學會了如果使用預訓練的網路進行微詞來加強網路 準確率，以及如何使用旋積 網路的對資料進行特徵分析， 也真切地成受到了移轉學習對運算資源的節省 。

/ 7.2 ／ 實例二一一Deep Dream ：探索旋積神經 網路眼中的世界

在前面一個實例中我們使用了預訓練的旋積神經網路實現了移轉學習，而預 訓練的旋積網路應用不僅於此， 2015 年 Google 發佈了一個很有意思的東西， 叫 做 Deep Dream ，網上瞬間掀起了 Deep Dream 的熱潮，各種各樣具有 Deep Dream 效果的圖片漫天飛，就算你沒有聽過 Deep Dream ，你也一定已經看過下面 的圖片 了，如圖 7.8 所示 。



”Admi ral Dog！” ，可he P咱－Snail"" 'The Camel Bird” ”The Dog-Fish""

圍 7.8 雲朵夢境

這些圖片都是天空中雲朵的圖片，但是卻呈現出幻覺和夢境， 所以這個演算 法被稱為 Deep Dream e 這個演算法其實是意外獲得的結果。

7.2.1 原理介紹

我們知道神經網路在影像分類上獲得了顯著的進展， 但是由於深度學習網路 中參數太多了，導致這個演算法是一個黑盒子，雖然能夠達到 良好的效果，但是 人們仍然對其內部知之甚少，所以人們希望能夠窺探一下網路裡面的內部。

道 1. 反向神經網路

一個神經網路讀取一張圖片，透過多層網路，最後輸出一個分類的結果， 但 是我們僅知道一個結果並不夠， 神經網路的挑戰是耍了解在每一層到底都發生了 什麼事。我們知道經過訓練之後，每一層網路逐步分析越來越進階的影像特徵， 直到最後一層將這些特徵比較做出分類的結果。例如前面幾層也許在尋找邊緣和 拐角的特徵’中間幾層分析整體的輪廓特徵’這樣不斷的增加層數就可以發展出 越來越多的複雜特徵 ’最後幾層將這些特徵要素 組合起來形成完整的解釋， 這樣 到最後網路就會對非常複雜的東西， 例如樹葉 、小貓等圖片有了反應。

為了了解神經網路是如何學習的，我們必須要了解特徵是如 何被分析和識別 的。如果分析一些特定層的輸出， 就可以發現當它識別到了一些特定的模式， 它 就會將這些特徵顯著地增強，而且層數越高，識別的模式就越複雜。當我們分析 這些神經元的時候，輸入很多圖片 ，然後去了解這些神經元到底檢測出了 什麼特 徵是不現實的， 因為很多特徵人眼是很難識別 的。一個更好的辦法是將神經網路

－－ － 第 7 章 深度學習實戰

顛倒一下，不是輸入一些圖片去測試神經元分析的特徵，而是我們選出一些神經 元，看它能夠模擬出最可能的圖片是什麼，將這些資訊反向傳回網路， 每個神經 元將顯示出它想增強的模式或特徵。

例如從圓 7.8 中我們能夠看出不同的神經元模擬出了不同的增強特徵和模 式，有一些是狗 ，有一些是蝸牛，還有一些是魚 。

2. Deep Drea m

透過上面的過程我們會迫使神經網路在圖片中產生一些本來不存在 的東西， 這也就產生了類似夢境和幻覺， 其實上這些夢境強調了網路到底學習到了什麼 ， 這種技術給我們提供了一種對抽象層次的定性戚受， 雖然這和現實中的夢境沒有 太大的關係，這也就是 Deep Dream 最早提出的軍戚。

實際上 Deep Dream 在上面的基礎上使用了更多的技術，如果我們將此演算法 反覆地應用在本身的輸出上， 也就是不斷地反覆運算 ，並在每次反覆還算後應用 一些縮放，這樣我們就能夠不斷地敢動特徵， 獲得無盡的新效果，例如最開始網 路的一些神經元模擬出了一張有狗的輪廓的圖 ，透過不斷的反覆運算， 網路就會 越來越相信這是一隻狗，圖片中狗的樣子也就會越來越明顯。

7.2.2 預備知輯：backwa rd

前面講了 Deep Dream 的基本原理，在實現之前，我們需要掌握一個預備知 識，就是 PyTorch 中的 backward 0

前面我們已經寫了很多網路， 也對網路進行了多次訓練， 這個過程都特別簡 單、直觀，但是一個網路訓練過程中 的操作一直沒有仔細去考慮過， 那就是 loss. backward （）， 看到這個大家一定都很熟悉，loss 是網路的損失函數，是一個純量 ， 但是如果 loss 不是一個純量，而是一個向量，那麼 loss.backward （） 是什麼結果 呢？

7.2 實例二一－Deep Dream：探索旋積神經網路眼中 的世界 一

我們可以去試試，寫一個簡單的程式 ：

imp 。rt torch

1 2 3 d 5

fr。m torch.aut 。grad imp。rt Variab工e

x - variable ﹝t。rch 。nes (2, 2﹜，主equires grad-True﹜

Y - X + 1

y.backward II

執行一下程式 ，會顯示錯誤如圖 7.9 所示。

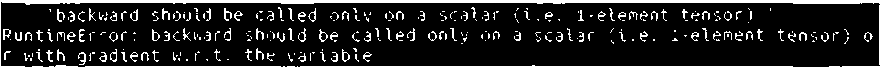


圖 7.9 執行錯誤

我們來讀一讀這個錯誤是什麼意思 。backward 只能被應用在一個純量上， 也 就是一個 lX l的張量上，或在 backward 中傳入跟變數相關的梯度。

之前一直都是將 backward 應用在一個純量 ，對於向量我們還沒有碰到過。 為什麼這裡我們需要知道 backward 是如何作用在向畫上的呢 ？這是因為在 Deep Dream 中需要對向量進行反向傳播。

首先複習一下純量的反向傳播 固

# simple gradient

。 。

1 2 3 4 5 6 7 8 9 1 1 1 1 1

a - Variabel ( torch Fl atTens r （ ﹝ 2, 3﹞ ﹜，requires\_gradaTrue) b a a ÷ 3

C a b ' b ' 3

。。ut = c.mean ﹛）

ut .backward ﹛﹜

print ﹝ 『＊’•101

print ﹛’－－－－－simple gradient－－－－’﹜ print （ input ' )

。

print ﹛ a.data)

。

print ﹝ c mp ute result is’﹜

1

print ﹛ ut .data ﹝ 0﹞ ﹜

2 3 4

print ﹛ Y input gradients are' I print (a .grad.data)

－第 7 章 深度學習實戰

上 面這個過 程運算 很 簡單，可以把 數學運算式寫出來， 傳 入的參數 *X1 = 2* 冒犯＝卜特別注意 Variable 裡面預設的參數 requires\_grad=False ，所以這裡要 重新傳入 requires\_grad=True 讓它成為計算圍 中的葉子節點。

α ＝*( xi,x,)* (7.1)

*b =( x1 +3*，λ2 + 3) (7.2)

*C* = (3× （x1 + 3)2,3×（也＋ 3)') (7.3)

*out =* 3×（（x1 + 3)2 ＋（也＋3)')

2

(7.4)

那麼我們對其求偏導也很簡 單：

*aout*

a;:;- = 3(xl + 3)1,, c,= 15 (7.5)

*aout*

τ

：于＝

M心2

3(x, +3) f,,c3= l8

(7.6)

我們就這樣依靠簡單的微積分知識算出結果， 執行一下程式，獲得如圖 7.10

所示的結果，和程式結果是一致的。

下面我們來學習一下如何能夠對非純量的情況下使用 backward 0

1 m - v （七 .Fl。atTens。r ﹛ ﹝ ﹝ 2, 3﹞ ﹞ I , requires grad-True)

2 n a v (t.zer。s ﹝ 1, 2 ） ﹜

3 n ﹝ 0。﹞ a m﹝ 0, 0﹞ 2

4 n﹝ O, 1﹞ a m﹝ O, 1﹞ ＊＊ 3

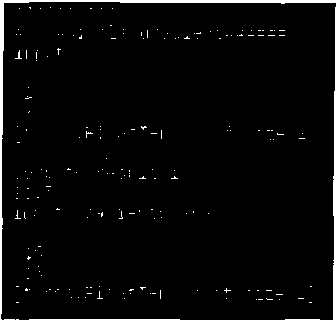


圖 7.10 純量的反向傳播結果

7.2 實例二一一Deep Dream：探索旋積神經網路眼 中的世界 一

首先定義好輸入 m ＝（λ1 ,x2) = (2 ,3）， 然後做的操作就是 n ＝（泣 后）， 這樣就定 義好了一個向量輸出，獲得的結果第一項只和 *X1* 有闕，第二項只和 *x*，有闕，那麼

*an,* 2

求解這個梯度，我們知道二立＝以 ＝4. 一＝3:< = 27 ，下面開始深入如何能夠讓它

，

*axl* 且 *ax* ι

呼叫 backward •

首先想到的是裡面的參數是要求梯度的物件， 我們這樣呼叫 n.backward(m. da呦，會獲得如圖 7.11 所示的結果。

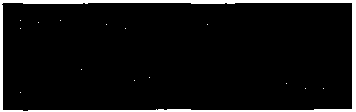


圖 7.11 非純量呼叫 backward 結果

獲得的結果跟之前計算出來的結果不一樣， 我們計算獲得的結果是 4 和 27 • PyTorch 反向傳播獲得的結果是 s f日 81，傳入的參數是 m.data ，數值是 （2, 3） 的 向量，我們希望獲得的梯度結果是 （4, 27）， 注意到 4 X 2 = 8, 27 X 3 = 81 ，其實 backward 將傳入的參數 m.data 每個元素分別乘上了獲得的梯度， 所以我們傳入 n.backward(t.FloatTensor （﹝﹝ I , I﹞﹞））， 可以獲得如圓 7.12 所示的結果。

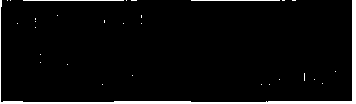


圖 7.12 結果校正 這就跟我們計算的結果一樣了。

下面來試試另外一種情況：

m Variable ﹛t。rch.Fl。atTens。r I ﹝ ﹝ 2, 3﹞ ﹞ ） , requires\_grad True)

1 2 3 4 5 6

j torch.zer。s (2 , 2)

k 貫 Variable ﹝t。rch zer。s﹝ 1, 2) )

m.grad.data .zer （﹜

k ﹝ O, O ﹞ m﹝0。﹞ ＊會 2 + 3 \* m﹝ 0 ,1﹞ k [O, 1﹞ m ﹝ 0, 1﹞ ＊＊ 2 + 2 \* m﹝ 0，。﹞

一第 7 章

深度學習實戰

將上面的程式寫成數學運算式就是 *m* （*x1 2* 也＝3), *k=(x;+3x2 ,x* ＋2x1）， 直 接對 k 反向傳擂，傳導入參數 k.backward(t.FloatTensor（﹝﹝ I, l﹞﹞）， 可以獲得如圖 7.13 所拍甘結果。

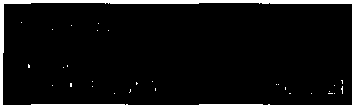


圖 7.13 直接對 k 反向傳播

下面我們手動算一算結果 是憾 。立專2主2 = 2x1 = 4 且牢控＝3

0.,:1 α丸2

*8(x;* + 2叫） \_ *8(x* ＋2叫）

一－；；－－一－ 2，一－－－－－；；－一一＝ 2x2 = 6，手動算出來是上面四個結果，這和上面輸出的

*OX1 OX2*

結果是不一樣的，上面只輸出了兩個結果， 而且數值還不對。其實這是因為我們 沒有正確了解傳導入參數數是如 何作用的，下面來解釋一下 PyTorch 是如何獲得 上面這個結果的。

已經知道獲得的 許的，）， 以及傳入的參數是 l 和 1，其中第一個結果是透過

I x *dk,* ＋Ix *dk*

＝*2x1* + 2 = 6獲得的，第一-個結果是透過 I x *dk.* + J ×*d*」*k*

＝3+ 2石 ＝9 獲得

*dxl dx*， 晶2 *dx,*

竺L

的，這樣就了解了 PyTorch 中backward 是如何處理傳入的參數。

下面透過 PyTorch 來求出上面計算所得到的 4 個結果，這 4 個結果又稱為

jacobian 矩障。

# jacobian

。

1 2 3 4 5 6 7

j = torch .zer s (2 , 2﹜

。

k = Variable (torch.zer s (l, 21 ) m.grad.data.zer 一﹛﹜

k﹝ O, 0﹞ ＝ m ﹝ O, 0﹞ ＊費 2 + 3 ' m ﹝ 0 ,1﹞

k ﹝ 0, 1﹞ ＝ m ﹛ 0, 1﹞＂2 + 2 金 m ﹝ 0。﹞

。 。

k backward (t Fl atTens r （ ﹝ ﹝ 1，。﹞ ﹞ I , reta1n\_vanables=True) 1﹝ ，0﹞ ＝ m. grad data

8

m.grad data .zer （）

。

9 1

k backwa主d ﹛ t.FloatTensor （ ﹝ ﹝ O, 11 I ﹜ 1 1﹝ ，1﹞ ＝ m.g玄ad.data

1

1 1 1

print ﹛’jacob1an matrix is’）

2 3

print ( j ﹜

透過執行上面的程式，可以獲得如圖 7.14 所示的結果。

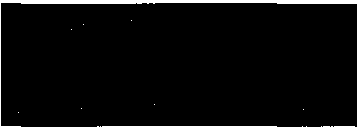


圖 7.14 jacobian 短陣

這裡我們使用了 backward （） 裡面另外的參數 retain variables=True ，這個參 數預設是 False ，表示的意思是反向傳播之後這個計算圓的記憶體會被釋放， 如 果計算圓被釋放掉，就沒辦法進行第二次反向 傳播了，所以我們需要將其設定為 True ，因為這裡我們需要進行兩次反 向傳播求得 jacobian 矩障。

最後再舉一個矩陣乘法的實 例檢驗一下計算結果， 下面是程式，留給讀者自 己去學習以掌握本小節的內容。

1. x - torch .Fl atTensor （ ﹝ 2, 1﹞ ﹜ .view (l, 2 ﹜
2. x - Variable (x, requires grad-True﹜
3. y - Variabel (torch.Fl atTens r ﹝ ﹝ ﹝ 1, 2﹞ ，﹝ 3, 4 ﹞ ﹞ ） ﹜
4. z - t。rch .mm (x 的

4

1. j acobian - t。rch.zer。s （ ﹛ 2, 2 ） ﹜
2. z back>咀rd ﹛t rch.Fl atTens r ﹛ ﹝ ﹝ 1, 0﹞ ﹞ I, retain\_variables-True) \* dzl/dxl, dz2／由d
3. jacobian ﹝ ，O J - x grad.data
4. x .grad data .zer 一﹛﹜
5. z backward (t rch Fl atTens r ﹝ ﹝ ﹝ O, 1﹞ ﹞ ﹜ ﹜ ＊ dzl/dx2, dz2/dx2
6. jacobian ﹝ ，1﹞ － x grad.data

12 print ﹛’－－－－－－－－－jac。bian－－－－－ －’）

1. print ﹛’x﹒）
2. print (x .data﹜
3. pr斗nt ﹛’y ）
4. print (y.data﹜
5. print （ compute result ）
6. print (z .data﹜
7. print （’jac bian matrix is ）
8. print ( jacobian ﹜

7.2.3 程式實現

前面兩個小節已經講了 Deep Dream 的基本原理和預備知識 backward 了，下 面為了更清晰地了解 Deep Dream，我們用程式來實現一下。

油 1. 實現原理

預訓練的網路對特定的分類工作有 良好的效果，為了深入它到底學到了什 麼，直接了解神經元分析的特徵是很困難的， 這時可以將一些與工作無闋的 圖片 輸入，希望透過網路對其分析特徵 ’然後反向傳播的時候不再更新網路的參數， 而是更新圖片中的像素點，不斷地反覆運算讓網路越來越相信這張圖片 屬於分類 工作中的某一種。這就是 Deep Dream 實現的原理，看上去是非常簡單的，但是實 際運用中要應用一些訓練技巧才能達到更好的效果。

油 2. 預訓練網路

首先需要一個預訓練的網路，同時還需要對預訓練網路 中的 foward 函數進行 調整，因為每次我們並不是對整個網路進行正向傳播， 而是獲得網路的中間輸出 結果，同時希望能夠很自由地控制輸出層。在 PyTorch 中這個實現是很簡單的， 我們使用預訓練的 50 層的殘差網路 （ ResNet ) • PyTorch 已經有 50 層殘差網路的 實現，只需要對其做一些修改就能滿足要求。

* 1. class B。ttleneck ﹝ nn .M。dule﹜
  2. expans斗 n 4

3

1. def \_init (self , inplanes, planes, stride＝工，d。wnsample=None)
2. super (Bottleneck, self ﹜ init I ﹜
3. self .c。nvl = nn .C。nv2d (inplanes, planes, kernel\_size=l, b工as=False)
4. self .bnl = nn .BatchN。rm2d (planes﹜
5. self c。nv2 = nn .C。nv2d (planes, planes, kernel size=3, stride=str工de’
6. padding=l, b1as=False)
7. self.bn2 = nn .BatchNorm2d (planes)
8. SE' 工 f .conv3 = nn c。nv2d ﹛planes, planes \* 4, kernel\_size=l, bias=False) 12 self bn3 = nn .BatchN。rm2d (planes \* 4 )

13 self relu = nn.ReLU ﹛ inplace=True﹜

1. self .downsample - d。wnsample
2. self .stride 己 stride 16
3. def f 主ward ﹝ self , x) ,
4. residual 巨 x 19

。

1. ut self convl ﹛x)

。

1. ut 旦 self.bnl (out)
2. 。ut self .relu ﹛。ut )

23

1. 。ut a self .conv2 （。ut )
2. 。ut a self bn2 ﹛ out)
3. 。ut a self . relu ﹛ out)

27

1. 。ut a self c。nv3 （。ut )
2. 。u乞 a self bn3 (out)
3. if self .d。wnsample is n。t None ﹒

30

1. residual a self .d。wnsample ﹛ x)

33

。

1. ut +a residual
2. 。ut a self .relu ﹛ out )

36

。

37 return ut

1. class Cust。mResNet (m。dels.resnet.ResNet ) ,

38

1. def f。rward ﹛ self , x, end layer﹜

41

。

42 end\_layer range fr m 1 to 4 43

。

1. X a Self c nvl (x)
2. X a self .bnl (X)

4 6 X 百 sel至. relu (x)

47 x a self .maxpool (x) 48

1. layers E ﹝ self .layerl, self .layer2, self .layer3, self .layer4﹞
2. E。r 1 in range (end\_layer) ,
3. x a layers ﹝ i﹞ （ X﹜
4. return x 53

54

55 def resnet50 ﹛pretrainedaFalse, 會 kwargs) ·

1. m。del - Cust mResNet ﹛B。ttleneck ，﹝ 3 告，6, 3﹞ ，＊＇kwargs ﹜
2. if pretrained,
3. model l ad state diet ﹛model z。load url (model urls ﹝ ＇resnet50 ＇﹞ ） ）
4. return m。del

首先定義一個最基本的殘差模組， 然後定義個性化的 ResNet 模組，這個模組 是繼承於 PyTorch 中的ResNet 模組，不需要改變其初始化， 讓它去繼承 ResNet 的初始化，只需要重新定義 forward 就可以 ，其中我們加入了結束層， 也就是我 們希望網路輸出第幾層的結果。

道 3. 訓練 Deep Dream

定義好了預訓練的網路，接下來就可以開始訓練模型， 再次說明 ，網路中的 參數是預訓練的參數，不發生改變，更新的是圖片中的像素點。

對於分類問題，使用交叉矯作為損失函數，那麼在 Deep Dream 中使用什麼作 為損失函數呢 ？非常簡單，之前講過 Deep Dream 希望能夠在反覆還算中不斷地讓 網路更加確定這個圖片屬於某一種，所以損失函數就是網路結束層輸出的特徵向 量的 *L2* 範數，目標是最大化 *L2* 範數來使得圖片經過網路之後分析的特徵更像網 路希望分析的特徵 。

看著可能有點難以了解， 舉個實例，一朵雲分析特徵之後有一點點像狗的 圖片分析的特徵 ’因為網路的參數不會改變，所以圖片如果不改變，那麼再次輸 入網路之後獲得的特徵還是跟之前一樣， 只是有一點點像狗的 圖片分析出來的特 徵。Deep Dream 希望透過反向傳播更新圖片的像素點，使得分析出來的特徵更 大，也就是使得分析出來的特徵跟狗的 圖片分析出來的特徵更像，那麼用 *L2* 范數 作為損失函數，網路不斷更新圖片的像素點來最大化 *L2* 範數，最後使得分析的特 徵越來越大，我們將多次更新之後的圖片輸出，也就獲得了 Deep Dream 效果之後 的圖片。

為了獲得更好的圖片 ，在訓練中需要應用一些小的技巧，否則獲得的圖片可 能會存在很多雜訊，或需要很長的時間才能 達到滿意的效果。

首先，對於輸入的圖片，我們需要對其做一些隨機抖動，下面是範例程式。

7.2 實冊三一一Deep Dream ：攝車撞撞神體網路眼中 的世界 －－－－－

1. shif t\_x, shift\_y "' np.random.randint (-max\_jitter, max\_jitter + 1, 2﹜
2. *img* = np.roll 【np.roll (img, shif t x, -1) , shif t y, -2﹜

m阻」i前er 是一個整數，表示抖動的範圈， 隨機能中取出兩個整數表示 x 軸和

y 軸的抖動程度，然接使用 np.roll 對陣列沿著一個維度進行平移， 上面的程式首 先對圖片的第三個維度進行平移 shift\_x 然接對第二個緯度進行平靜 shift\_y 。

如何進行反向傳播呢， 是不是需要計算出 *L2* 範數呢 ？其實不需要，*L2* 範獸 的公式如下：

*ll x llL*，＝」*x:+ x.* ＋..﹒＋*x*﹔ (7.7)

對這個結果進行求導是很麻煩的， 因為裡面有根號，非常影響計算效率，而 且目標是最大化 *L2* 範數，所以去掉根號不影響計算的結果。 去掉棍號之後＊導就 變得容昂了，可以獲得下面的式子：

llxll ：＝：*x;* ＋茍 ＋..*·+x* (7.8)

每個事數 x 都是圖片像章點的函數，記圖片中像章點為*P* ，品．吼，那麼

*xi=xi( P* ，函，...A）， 所以對其京偏導，可以獲得下面的結果：

叫*I x* 11;, ＿ ι δIX,

一τ一」－＝ *£ 1* 抖了4

*opi* 五T *opj*

(7.9)

前面的係數 2 是常數，可以去掉，這樣就撞得了反向傳播時悵的梯度的計算 方法。這裡就要用到上一節講的 PyTorch 中backward’可以透過下面的方式獲得 所有待更新的偉素點卸的梯度。

1. act\_value = model.forward ﹛ img\_variable, end\_layer﹜
2. act value .backward (act value.data ﹜

*\_ n ax* --

主

act\_value.backward(act value.data） 就袁軍 *X* －；：主 ，這就是上面 *L2* 範數反向

- *i=I* 吟j

傳播的企式。

第二個訓練拉巧是對學習車進行一些限制， 首先將所有事數梯度的括對值求 平均值，靜、使用學習草除以平均值，這樣就獲得了實酷用的學習宰 。

* 1. rati。里 np.abs (img\_variable.grad.data .cpu () .numpy ﹛） ) mean ﹛）
  2. learning\_rate\_use learning\_rate *I* rati。
  3. img\_vanable data .add ( img\_vanable.grad data \* learning rate use﹜

最後還需要使用一個小技巧， 就是使用多尺度的圖片進行計算，如果一直使 用原始的圖片 ，可能收斂速度會比較慢，所以我們先將圖片縮小進行更新，再放 大進行更新 。

1. for 1 in range ﹛ ctave n - 1) ,
2. 。ctaves.append (nd.z nm （ ctaves ﹝－1﹞ ，（ 1, 1, 1.0 *I* octave\_scale, 1 0 *I* octave\_ scale） ，rder旦 l) )

octave n 表示有多少張小圖片 ，使用 scipy.ndimage.zoom 進行圖片的縮放， 每次縮放的比例是 1/。clave scale ，這樣就可以獲得大小遞減的小圖片 ，然後從小

到大依次對圖片的像素點進行更新，最後獲得 Deep Dream 的圖片。 最後做完一次更新需要將圖片做逆抖動，開始的時候圖片經過平移，現在將

圖片反向平移回來，間時需要對圖片的像素點進行修改以控制大小。

1 1mg np.roll (np r。11（立mg, -shif t\_x，一11 , -shif t\_y, 2)

2 1mg ﹝ 0’，：，．﹞ 詰 np.clip ﹛ 1mg ﹝ 0’，’，﹞ ，－mean *I* std, (1 - mean) *I* std)

2過 4. Deep Drea m 結果

對一張雲的圖片輸入做 Deep Dream，經過 Deep Dream 的效果之後變成了圖

7.15 ，可以看到圖片中多了一些狗頭，還有一些眼睛，中間左邊有一個明顯的塔。

「



I 圖 7.15 對雲做 Deep Dream

7.2 實例二一一Deep Dream 探索旋積神經網路眼中 的世界 －

除此之外，還可以控制 Deep Dream 中的夢境，也就是說可以控制圖片中出現 的東西。要實現其實很簡單，之前最大化特徵向量的 *L2* 範數，這導致了圖片中出 現了各種各樣的圖片 ，所以要實現夢境控制，我們需要修改一下目標函數。

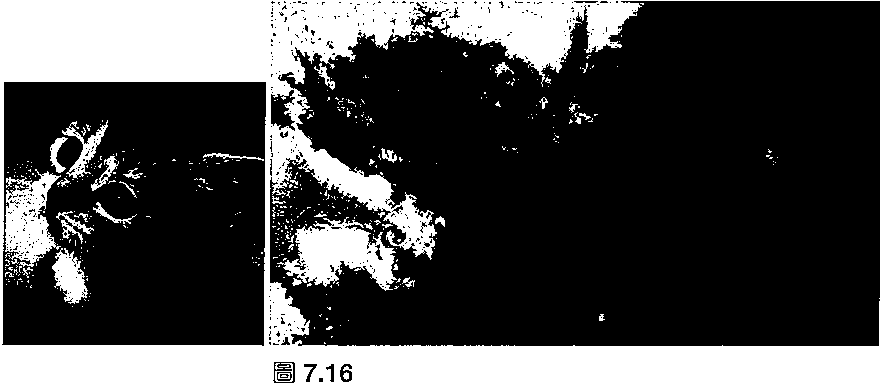
首先需要輸入一張圖片作為夢境的控制圖片 ，將控制圖片透過網路正向傳播 獲得其特徵向量，然後將原始圖片輸入網路也獲得原始圖片的特徵向量，這兩個 特徵向量的大小不同 ，所以先將它們重新排列成新的矩障，然後做矩陣乘法，最 後選擇矩陣乘法裡面最大的索引，這些索引對應於原始圖片和控制圖片最為符合 的特徵向量，將這些最符合的特徵向 量分析出來作為新的特徵向量進行最佳化， 最後獲得的結果越來越像控制圖片 ，這樣就可以獲得控制的夢境， 實際可以看看 下面的程式實現。

1 def 。bjective\_guide (dst, gu工de\_features)

2 X a dst .data ﹝ O﹞ .cpu （﹜ .numpy （﹜ .c。py ()

1. y a guide features data[ O ﹞ cpu ﹛ I nu呻y （ ﹜
2. ch, W, h a X shape
3. x a x .reshape (ch,-11
4. y a y.reshape ( ch,-1﹜
5. λ a x.T dot ﹝ y) # C叩pute the matrix of dot pr。ducts with guide features
6. result a y ﹝ A argmax (1） ﹞ ＃ select 。nes that match best
7. resclt a torch, T， s。r (np.缸ray （ ﹝扭曲lt.resha戶（曲 w, h﹞ ﹞ dtY!浩『可p float! I cc血的
8. return result

如果輸入的控制圖片是一張小貓的圖片 ，最後透過 Deep Dream 能夠獲得圖 7.16 所示的圖片，可以看到圖片中有一些貓的頭 ，貓的眼睛和貓的鼻子 ，是不是 很神奇呢 ？



對貓做 Deep Dream

7.2.4 歸納

透過上面的實例， 我們不僅再次了解到了預訓練網路的有用之處， 同時學會 了不僅只有網路的參數能夠更新，輸入的圖片也能進行更新。除此之外，還了解 到了如何將 backward 作用於純量上，如何手動更新參數，以及如何製作出 Deep Dream 效果之後的圖片。

I 7.3 I 實例三一－Neural-Style ：使用 Pyli rch

進行風格移轉

在上面兩個實例中，我們都使用了預訓練的旋積神經網路： 一個是用來做移 轉學習，節省運算資源，另外一個是用它來產生 Deep Dream 效果的圖片，這一節 我們將使用預訓練的旋積神經網路來實現藝術家的風格移轉， 讓我們自己的圖片 看著像是被藝術家畫出來的一樣。

7.3.1 背景介紹

前段時間有一個叫做 prisma 的 App 很火，就算各位讀者沒有用過，也一定看 見別人使用過這個軟體，下面是用這個軟體獲得的效果圖， 如圖 7.17 所示。

li'..il

4、一輛

F

也

日國 固

矗矗..

、

—q

E

幸

H寸 J芷

i

－

i

本JN唱

A時 ，

一

叭，、J

、F

圖 7.17 prisma 紋果圖

7.3 實例三 Neu ral-Style：使用 PyTc rch 進行風格移轉 －

App 宣傳的賣點是讓你的照片擁有名家風格，例如畢卡索、梵古等。其實這 個軟體背後的原理就是這裡我們要講的風格移轉， 這個演算法是 2015 年由 Leon A.Ga酬，Alexander S.Ecker 等人在其論文中提出的，有興趣的讀者可以去閱讀一 下論文 （ https://arxiv.org/abs/1508.06576 ）， 這篇論文提出的演算法就叫做 Neural Style 裡面主要有關的是旋積神經網路。

Neural-Style 也叫做 Neural-Transfer ，是一個神奇的演算法， 輸入一張國片 ， 然後選擇一種藝術風格圖片 ，演算法會將它們兩者融合在一起， 讓輸入的圖片達 到這種藝銜的效果， 如圈 7.18 所示﹒我們將一張海龜的圖片輸入，然後選擇一種 抽象派的風格，最後透過演算法融合，獲得了一張抽象的海龜圖片 。

可可向

區 -: ） 「

包、 、＼

可

』

毛1

＇ι

、訓、

可吟 ，＼ ν

" *,(* 0 三且且 且士 《ι

4

、

圖 7.18 融合藝術妓果

7.3.2 原理至于析

透過前面對於背景知識的介紹，可以知道需要實現的東西很清楚， 就是要將 兩張圖片融合在一起，但是這只是一種抽象的表達，我們需要準確地定義怎麼樣 才算融合在一起。 一張園主要由兩個部分組成﹒一是內容：二是風格，所以論文 中提出了兩個度量方式：第一個就是在內容上相似程度的度量﹔第二個是風格上 相似的度量 。透過這樣的定義，我們就明確地知道需要最佳化的 目標是什麼了。 首先需要讓融合圖片和原始圖片有盡可能高的相似度，或說盡可能低的差異性﹔ 同時也需要讓融合圖片和風格圖片在風格上盡可能相近。雖然我們知道了需要最 佳化的 目標是內容上的相似度和風格上的相似度 ，但是仍然沒有實際的量化方式 來處理，所以接下來就需要實際定義內容上相似和風格上相似的數學表達。

．－－ 第 7 章 深度學習實戰

油 1. 內容差異

內容的差異性該如何定義呢 ？最簡單的想法就是將兩張圖片每個像素點逐 進行比較，可以看作是均方誤差， 也就是求一下差，然後計算平方的和。但是透 過第 6 章 「 產生對抗網路 」 了解到這種定義圖片相似度誤差的方式是很不好的， 因為這樣把擾的其實是單一像素點的影響， 而弱化了整體圖片的影響，所以我們 需要有更好的方式來定義內容誤差。

前面一個實例中我們知道 Deep Dream 透過將圖片輸入預訓練的網路來分析圖 片的特徵’那麼我們可不可以 比較圖片透過預訓練網路分析的特徵來代替直接 比 較圖片呢 ？答案當然是可以的， 特徵可以視為是更高維度的圖片，所以比較特徵 比直接比較圖片有更好的效果。

我們讓 C ，，表示一個隻合前面旋積分析特徵部分的預訓練網路， 讓X表示任 何輸入的圖片，那麼 C，，，，伏） 表示輸入圖片經過預訓練網路每層分析出來的特徵圓 的集合，每個特徵圖都是一個 泊的矩障，讓 F也 E C,,,,(X） 表示第 L 層網路分析 出來的特徵圖 ，大小是 *h x w x d*’我們可以將這個矩陣展開成一維的向量， 那麼 這個向量的大小是 伊叭d，可以認為這張輸入的圖片X在網路第 L層的內容就是

*Fx,* ,

我們需要比較兩張圖片的內容差異，這兩張圖片大小應該是相同的，例如 Y

是另外一張圖片，就可以定義這兩張圖片在 L 層的內容差異如下：

*Di ( X ,Y )* =II *Fx* 一九 「＝ *L(FXL(i)* 几(i))' (7.10)

其中 *Fx,( i* ） 表示網路第 L 層輸出的特徵圖展開的向重的第 1個元素。我們該 如何定義風格的差異性呢 ？這並沒有內容差異性定義得那麼直觀 ，這也是這篇 論文提出的創新點， 透過引用 Gram 矩陣來表示圖片的風格，據此再來計算風格 的差異，首先我們介紹如何用 Gram 矩陣來定義圖片風格，接著講一下如何透過 Gram 矩陣來定義風格差異。

1. Gram 矩陣 首先各位讀者都能想到要做的第一件事情就是將圖片輸入一個預訓練的旋

積神經網路分析出特徵圈 ，將圖片空間投影到高維，這是前面使用過很多次的芳

法，因為直接在原始圖片空間上做處理會損失掉很多的特徵。 前面我們用 *F,i(i )* 表示網路第 L層輸出的特徵圓展闊的向量 中的第 1個元素，如果特徵圍不展闕， 我們知道這是一個大小為 h川*x d* 的矩障，所以我們用 此表示這個大小為 *h x w x d* 特徵園中第 k層厚度的矩障，大小為 *h x w* 。

那麼如何來定義 Gram 矩陣呢 ？首先 Gram 矩陣的大小是由特徵圓的厚度 d決 定的，等於 *d x* d ，每一個 Gram 矩陣中的元素，也就是 *Gram( i,j* ） 等於多少呢 ？

先把特徵圍中厚度第 1層和厚度第Jj哥取出來，這樣就獲得了兩個 h X w 的矩障， 分別表示成 *FJa,* 和 F卒 ，然後將這兩個矩陣對應元素相乘求和，這樣就獲得了 *Gram(i,j* ） 可以用下面的公式來表示 ：

*GXL(i,J )=< FJ,,F*立＞＝*L F1 ( k )?Fj, (k)*

*k XL*

(7.11)

透過上面這個方式可以定義 Gram 矩陣中所有的元素，這樣 Gram 矩陣中每個 元素都可以了解為由特徵園中的 1層干日j j富矩陣相關性的表示，用其定義為圖片在 網路為 L 層輸出的風格。

迫 3.風格差異

定義完了圖片的風格之後，風格的差異就很簡單了，就是兩幅圓的 Gram 矩 陣的差異，就像內容的差異的計算方法一樣，可以用下面的公式計算 L 層風格的 差異。

D （ *X ,Y* ） 斗I *GXL GYL* II'=*Z:<F.*，伏，I） 几阱 ／））＇

'·'

(7.12)

在風格轉換中我們需要最小化幾層內容差異 *Dc (X* 和幾層風格差異 *Ds*

i玄 旬，所以我們的目標函數是這兩者的和，對其求梯度可以用下面的公式濃示：

*V( X ,S,C)= Lic WciJV* （ *X ,C)+ L* 叫宵 ·（*X ,S )* (7.13)

間圓圓圓－ 第 7 章 深度學習實戰

其中心和 *Ls* 表示內容和風格分別需要幾層的輸出， 這是一個可以根據所需 要的效果任意設定的參數， *WcLc* 和 *WsLs* 表示內容上和風格上指定的加權， 也是可 以根據需要的效果任意設定。

最後透過反向傳播和梯度下降法能夠最佳化圖片中的像素點，實現風格轉換 的效果。

X← *X -aV(X ,S,C)* (7.14)

7.3.3 程式實現

前面的部分有太多的數學公式， 如果你仍然沒有太了解風格轉換的原理， 沒 有關係’透過程式實現的部分會能夠重新 了解那些用數學符號表達的公式。

::,i 1. 內容差異

首先我們實現內容差異的程式， 這個部分原理比較簡單，程式也相對容易。

* 1. class Content Loss ﹝ nn.M。dule﹞
  2. def 一＿init\_ (self , target, weight﹜
  3. super (C。ntent L。ss, self ﹜ . init (I
  4. self , weight weight
  5. self .target target .detach ﹝﹜ •self weight

。 。

* 1. self .criteri n a nn .MSEL ss (I

7

1. def forward (self , input) ·

。 。

1. self .l ss a self .criten n (input • self weight, self , target)

。

1. ut a input. clone ﹝）

。

1. return ut 12
2. def backward ﹝ self , retain vanabelsaTrue ) ,
3. self .loss.backward ﹛ retain variab工esaretain variabels) 15 return self 1。ss

這裡注意，定義的 loss 不像我們之前定義的 loss 一樣，因為之前的 loss 在 PyTorch 中都已經實現了，但是這裡我們需要自訂新的 loss 所以如果想要像之前 樣定義，需要重新定義 backward 。這裡使用網路模型的定義方式，把這個部分

當成一個網路層，下面我們來完整地解釋 整個定義方式。

圖片經過預訓練的旋積層獲得 L層的特徵圖 *Fn*，最後諭出與基準內容圖 片的 加權內容距離 *Wc,-D.* 伏，CJ ，所以初始化中需要傳入基準闆片和加權，上面的程式 中傳入的基準圖片為 ta喀肘，加權為 weight 0

接著在初始化中，透過 tar且et.detach（） 將基準圖片從計算圖中分離出來， 視 為一個常數。定義 nn.MSELoss （） 為了計算 L 層輸出的兩個特徵圓的距離 *II Fxc Fn* 11 2 。

將上面定義的內容差異視為是一個網路層結構， 在自orw缸d 的實現中 ，將輸 入的特徵圍和基準特徵園求出內容距離儲存為 self.loss 然後用 clone 將輸入複製 成輸出，傳出網路。可以看到特徵圖傳到這一層網路結構之後再傳出， 輸入和輸 出是完全相同的，只是經過這層網路之後，會計算出一個內容的差異被儲存起來。

最後定義 backward’定義這一層網路是如 何進行反向傳播的 。該層網路相當 於把輸入直接轉換成輸出， 然後進入下一層網路， 如果不重新定義反向傳播，那 麼這一層就相當於一個值等轉換， 是沒有辦法將計算出來的內容差異反向 傳播回 去更新參數的。

在重新定義的 backward 中，我們將內容距離，也就是 self.loss 進行反向傳 播，這裡注意一個小細節， retain variables-True •正如前面講過的，這個變數是 為了在反向傳播之後保留計算圖，因為網路中會有多層的內容差異和風格差異， 所以還會做多次反 向傳播。

油 2. 風格差異

首先需要定義好 Gram 矩障，根據 7.11 的公式我們可以將特徵圖 F且 重新排 列成 阱，m× *n*），用 K來表示，那麼上面這個公式可以改寫成下面這樣：

*Gram(i,J ) -"I,K(i,s)* ·*K(j,s)* (7.15)

= *L,K* （悅 (7.16)

*=K ·KT (i,J )* (7.17)

間圖－第 7 章 深度學習實戰

所以可以獲得 *Gram =K ' Kr*，獲得了這個結論 ，就能很容易地寫出 Gram 矩 陣的定義。

1. class Gram (nn.M。dule) ,
2. def init (self ) ,
3. super (Gram, self ﹜ . rnit I ﹜

也

1. def f。rward ﹛ self , input ) .
2. a, b, c, d 工 nput .size （﹜
3. feature 巨 input view (a • b, c • di
4. gram 呈 torch.πm ﹛ f eature, f eature .t﹛ I I
5. gram *I* (a • b • c • di
6. return gram

在 forward 中，我們將輸入的特徵用 view 重新排列，然後用 torch.mm 做矩陣 乘法，最後標準化獲得 Gram 矩障。

定義好了 Gram 矩障，我們就可以定義風格差異了 ，方法和內容差異幾乎一 模一樣，程式如下所示。

1. class Style\_L。ss (nn M。dule﹜
2. def init\_ (self , target, weightl ·
3. super (Style Loss, self l . init ﹛）
4. self .we工ght 呈 weigh七
5. self .target target detach （﹜ •self weight
6. self gram Gram ﹝）
7. self criter斗on = nn MSEL 。ss 11

8

1. def f。rward (self , mpu℃﹜ －
2. G 巨 self gram ﹝ rnput l 有 self .weight

11 self .loss self .enter工 n ﹛ G, self target l 12 。ut 巨 斗nput .clone ﹛）

13 return 。ut

14

1. def backward ﹛ self , retain variabels巨True﹜
2. self loss backward ﹛ retain variablesretarn variabelsl 17 return self .l ss

。

* 1. 實例三一－Neural-Style 使用 PyTorch 進行風格移轉 －眉目－

訕 3. 建立模型

有了前面內容差異和風格差異的定義， 就可以據此來建立模型了。 首先，會使用預訓練的 19 層 VGG 網路，去掉最後的全連接層，只保留前面

的旋積層。

1 vgg 旦 m 。dels.vgg19 (pretrainedTrue) . features

接著，需要確定哪幾層的網路輸出作為內容和風格的輸出層， 透過下面的程 式來定義。

1. C。ntent\_layers\_default ﹝ c。nv\_4 ＇﹞
2. style\_layers default ＝ ﹝’c nv 1 ，v c nv 2 ＇，conv\_3，，c nv\_4 ，，c nv\_S ＇﹞

這裡我們只使用第四層旋積層作為內容層的輸出， 使用 l ～5 層的旋積層輸 出作為風格層的輸出，需要重構我們的網路模型， 將內容差異和風格差異加入到 原先的網路層當中，可以透過下面的程式來達到所需要的 目的。

1. 主 r layer 工n cnn.
2. if isrnstance ( layer, nn.Conv2d) ,
3. name 'conv ’＋ str ﹛斗）
4. m 。del.add m 。αule (name, layer﹜
5. i主 name 工n c。ntent\_layers\_default ,

5

1. target 互 me del I c。ntent img)
2. content l。ss = l。ss Content\_L 。ss (target, content\_weight)
3. m del.add m。dule I c。ntent l ss ’＋ str ﹝ i) , content l ss) 10 C ntent loss list .append (c ntent loss)

。 。

11

1. if name rn style\_layers\_default ,
2. target 巨 m。del (style img﹜
3. target gram (target﹜

。 。

1. style loss 1 ss Style\_L ss ( target, style\_weight) 16 m del.add\_module ﹛ style l ss ' + str ( i) , style loss﹜ 17 style\_l ss\_list.append ﹝ style\_loss﹜

。 。

。

18

19 i ＋ 1

1. if isinstance ﹛ layer, nn .MaxP。。12d) ,
2. name ’pool＿’ ＋ str I ii

一第 7 章

深度學習實戰

mcdel.add\_module (name, layer)

2 3 4 5 6

2

2

if isinstance (layer, nn ReLU) · n凹e a ’relu ' + str ( i)

2 2 2

model.add m。dule (n訓e, layer)

我們檢查網路 中的每一層，然後判斷一下， 如果這一層使我們要求的內容輸 出層或是風格輸出層，我們就可以將前面定義的 ContentLoss 和 Style\_loss 增加到 其後面，這樣檢查完整個網路結構之後， 便建置出了我們需要的網路結構。

油 4. 執行程式

建立完模型就可以反向傳播更新參數了 ，需要注意的是更新的參數並不是網 路中的參數，而是輸入圖片的像素，所以首先需要申明一下參數。

def get\_input\_param一optimier ﹛ input\_img) : input\_img is a Variable

1 2 3 4 5

”””

～

input param a nn Parameter ﹛ input\_img. data)

。

ptimizer a optim.LBFGS ﹛ ﹝ input\_param﹞ ）

6 7

return 工nput\_param，ptimizer

這裡使用 nn.Parameter 將輸入圖片變成計算園中的葉子節點，進一步能夠進 行梯度下降更新參數， 然後論文推薦使用二階收斂方法 L-BFGS，不太了解的讀 者可以自行查閱相關資料，這裡就不再展開說。

按著就可以進行模型的訓練了 。

def run\_style\_transfer (c。ntent\_img, style\_img, input\_img, num epcchesa3QO ) .

1 2 3 4 5 6 7 8 9 1

print （’Building the style transfer model ．）

m。del, style l。ss list, content loss list a get style model and loss ﹛

style\_img, content\_img ﹜

input\_param’。ptimizer a get input\_pararr ＿ ptimier ﹛ input\_1mg ﹞

print （ Opimizing . ' ) epoch ＝ ﹝ O J

while epoch ﹝ O﹞ ＜ num epoches:

。

1. def cl。sure ﹛）：
2. input\_param.data .clamp (0, 1)

13

1. 恥 del ( input\_param)
2. style score - 0

。

1. C ntent scare t旱。

17

1. 。,pt1m1zer.zer。＿grad （﹜
2. 主 r sl in style l ss list ,
3. style\_sc。re +- sl backward (I
4. f。E cl in c。ntent l。ss list .
5. content sc。re +- cl backward ﹛）

23

24 epoch ﹝ D﹞ ＋－ 1

25 if ep。ch ﹝ O﹞ ＼老 50 -- 0 ,

1. print ﹝’叩n ( ) ' .format ﹛ epoch） ﹜
2. print ﹛’Style L。ss· ﹛ ·.4f ﹜ Content Loss：﹛ 4主﹜ .format (
3. style score .data ﹝ O﹞ c。ntent\_sc。re.data ﹝ O﹞ I I
4. print ()

30

。

31 return style\_score + c ntent score

33 。ptimizer .step (cl。sure I

32

34

35 日put\_param.data .clamp ( 0, 11 36

37 return input\_par臼n.data

網路透過對所有的內容差異和風格差異進行多次反向傳播來更新參數， 注意 一下裡面定義了 closure，這是因為我們使用 L-BFGS 演算法，所以更新參數的時 候需要在 s但p裡面呼叫。最後還需要注意 的一點就是每次我們都會將更新之後的

參數，也就是最後要輸出的風格融合的圖片 中的資料的大小進行重新修改，因為 更新的過程中可能會超過圖片資料允許的大小，一般是 0 ～255 。

訕 5. 結果

我們輸入內容圖片和風格圖片 ，經過我們定義的網路進行參數更新之後獲得 如圖 7.19 所示的圖片。

．－ 第 7 章 深度學習實戰

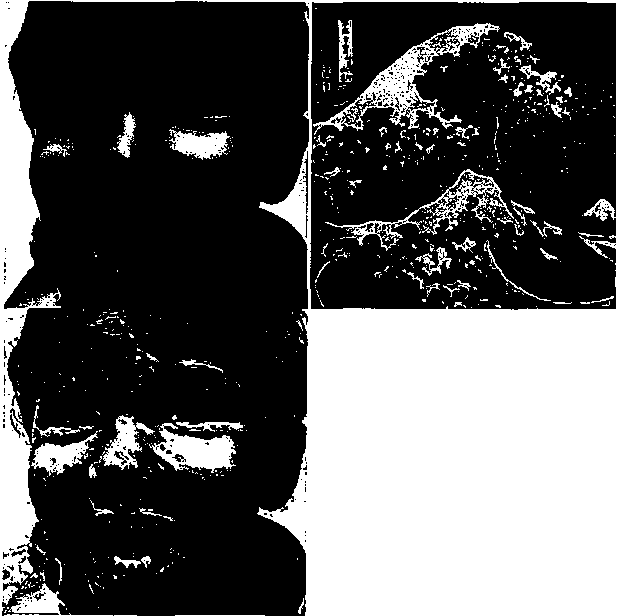


圖 7.19 圖片風格融合結果

7.3.4 歸納

透過這一節，我們利用 PyTorch 實現了基本的風格傳輸演算法， 透過預訓練 的旋積網路分析出更高維度的圖片的內容和風格，最後透過定義內容損失函數和 風格損失函數進行反向傳播更新參數， 實現了圖片風格融合的演算法。 從該j寅算 法中，我們也了解到了很多獨特的定義方式，以及預訓練網路的神奇之處。

1 7.4 I 實例四一 －Seq2seq ：透過 RNN 實現簡 單的 Neural Machine Translati n

。

前面三個實例都是講旋積神經網路在不同 方面的應用 ，最後一個實例我們講 一下循環神經網路如何應用在自然語言處理中的機器翻譯上，介紹如何教會神經 網路進行法語和英文的翻譯。

7.4.1 背景介紹

隨著深度學習的掘起，以神經網路為基礎的機器翻譯技術 （ Neural Machine Translation’NMT ） 也獲得了長足的進步， 迅速替代了之前的主流翻譯技術。

NMT 技術使用了點對點的方式， 直接透過神經網路建立來源語言和目的語言的對 應關係，省去了傳統方法煩瑣 的前宣處理流程，同時也相當大地加強了機器翻譯 的準確率。

而引發這一切成功的是一個簡單卻又強大的想法， 即序列到序列的網路 ( sequence to sequence network）， 使用兩個循環神經網路，將一個語言序列直接轉 換到另外一個語言序列，下面我們就來講講其原理。

7.4.2 原理分析

前面講過循璟神經網路因為具有記憶性， 所以能夠根據前面的狀態來預測後 面的狀態，這對於語言模型是非常重要的， 因為和一個詞在不同的語境下面具有 不一的意思，所以循項神經網路特別適合應用在語言模型中 。

序列到序列的模型是循環神經網路的升級版 ，其聯合了兩個循環神經網路： 一個神經網路負責接收要源句子﹔另外一個循環神經網路負責將句子輸出成翻譯 的語言。這兩個過程分別被稱為編碼 （ encoder ） 和解碼 （ decoder ） 的過程。如果 還記得 6.1.1 「 自動編碼器」 那一節的內容，你就會記得那裡也有一個編鵑和解碼 的過程，它們的原理是相似的，下面我們來講講這兩個過程。

:,i 1.編碼 （ encoder )

開發過程實際上使用了循環神經網路記憶的功能 ，透過上下文的序列關係， 將詞向量依次輸入網路。 對於循環神經網路， 我們知道每一次網路都會輸出一 個結果，但是編碼 （ encoder ） 不同之處在於，其只保留最後一個隱藏狀態，相 當於將整句話濃縮在一起， 將其存為一個內容向量 （ context ） 供後面的解碼器 ( decoder ） 使用。編碼可以由如圖 7.20 所示的示意圖更加直觀地表示。

油 2. 解碼 （ deocder )

解碼和編碼網路結構轟乎是一樣的， 唯一不同的就是在解碼過程中， 是根 據前面的結果來得到接面的結果， 什麼意思呢 ？因為在開發過程輸入一句話， 這 一句話就是一個序列，而且序列中的每個詞都是已知的，而解碼過程相當於什麼 都不知道，首先需要一個識別符號表示一句話的開始， 然後將其輸入網路獲得第 一個輸出作為這句話的第一個詞， 接著透過獲得的第一個詞作為網路的下一個輸 入，獲得的輸出作為第二個詞， 不斷措環，透過這種方式來得到最佳網路輸出的 一句話。如果厭覺有些接惑，可以透過圖 7.21 所示的示意圖束了解 e

”le chat est noir" c::EOS>

[ 02 85 03 扭 曲 ﹞

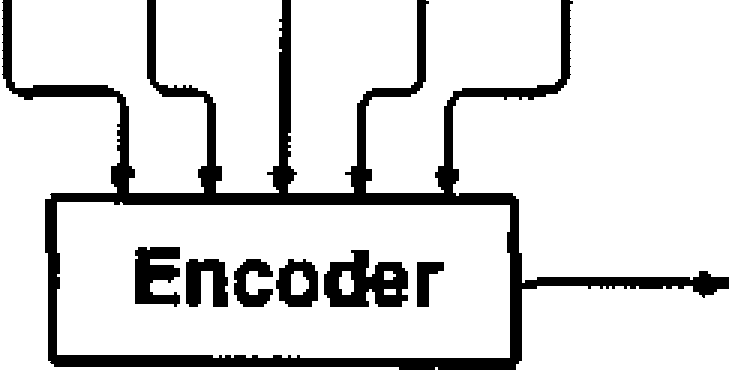
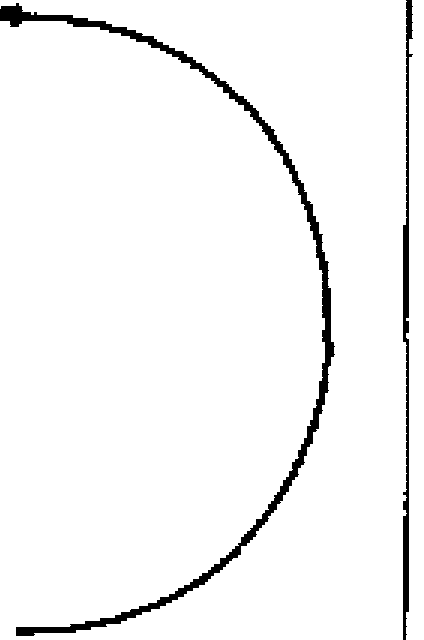
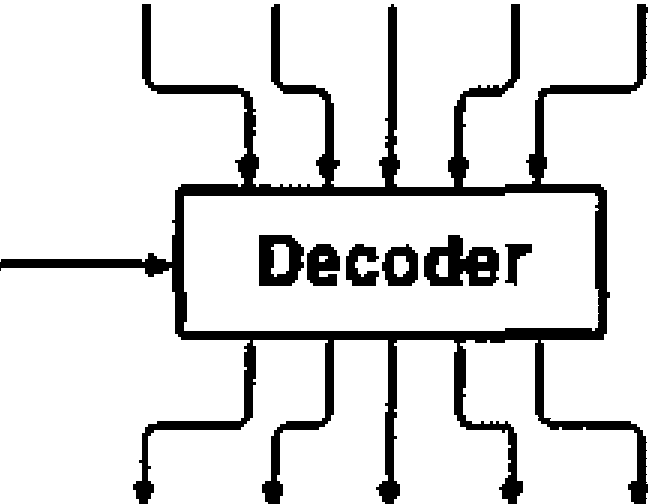
 Context

圖 7.20 攝碼

<SOS＞可he cat is black"

[ 00 42 82 16 04 )



Context

﹝ 42 82 16 04 99 ]

"the cat is black＇ ＜EOS>

圖 7.21 解囑

從 6.1.1 節我們知道，旋積神經網路也可以作為一個編碼器和解碼器， 這裡使 用循環神蘊網路作為編碼器和解碼器的好處是什麼呢 ？還是因為數述存在序列關 係’對於同一個詞，其前面的詞語不同，那麼這個詞所代表的含義也將不同， 而 宿環神經網路有記憶的 這一特性能夠極佳地處理這種關保。

那麼使用序列到序列 （ seq2seq ） 這種網路結構的原因又是什麼呢 ？第一， 翻譯的每句話的輸入長度和輸出長度一般來講都是不同的， 而序列到序列的網路 結構的優勢在於不同長度的輸入序列能夠獲得任意長度的輸出序列﹔第二，在翻 譯中，雖然有一些詞能夠一一對應，例如中文中的 「 貓」 對應英文中的 r cat 」， 但是對於一句話，每個詞並不一定是一一對應的， 例如一句中文 「 你今天吃了什 麼？」’翻譯成英文就是 「 what do you eat today？」’這裡整個敘述的對應關係就是 顛倒的，所以使用序列到序列的模型，首先將一句話所有的內容壓縮成一個肉容 向量 ，然後透過一個循環網路不斷地內容提取 出來，形成一句新的話。

同時序列到序列的模型不僅能夠用在機器翻譯上， 這個想法還能夠用在圖片 字幕上，將圖片透過旋積網路分析特徵， 然後使用序列模型獲得圖片中的內容描 述，本質上跟上面的編喝解喝過程是一致的， 只是使用了旋積神經網路作為編嗎 器。這樣的模型本質上都是編碼和解碼的網路結構。

凶 3. 注意力機制

透過編喝早日解碼形成的序列到序列的模型建置了機器翻譯的基本架構， 但是 在實際中表現常常不夠好，這裡面很大的原因是因為在開發過程會將一句話的內 容壓縮成一個固定大小的內容向量，如果這一句話比較長，這個壓縮的過程就容 易失去這句話中的資訊，同時如果兩句話比較相似，只有一兩個詞不同，壓縮成 內容向量過於相似，會導致在解碼的過程表現不夠好。

因為上面問題的存在，我們希望能夠找到一個機制來解決這個問 題。在之前 的序列到序列的模型中，我們將最後的隱藏狀 態儲存成一個內容向量 ，而每一步 的輸出我們並沒有使用，那麼能不能將每一步的輸出都利用起來呢 ？答案當然是 可以的，這就是下面要講的注意力機制。

注意力機制 （ 甜ention） 是由 Bahdanau 在 2014 年提出的，聽名字就知道， 注意力機制就是讓網路在解碼的時候能夠 「 集中注意力」 在編碼輸出的某些部分 上，而不僅依賴於簡單的內容 向量，解碼的每一步都能夠視為是在考慮 輸入序列 的不同部分。

下面我們可以邁過簡單的圖 7.22 解釋一下注意力攝制 ，

編碼輸出

編碼輸λ

片 片 r n · 「

口口口口口

•[\_J [J 口 ［J 口」（ 伽

注意力

﹔CJ •[J LJ 戶 口

cat

編碼輸出

［ J 仁J •[J CJ 口一→ is

解碼輸出

["J i-::-J [] ［］ 口 申

） 口。口

汀 c:· L.

black

EOS

圖 7.22

注意力攝制

圖 7.22 就是注意力攝制的簡單圖示， 對於每個解碼的輸出， 都是由編碼的輸 出和加權一起決定的， 圖上藍色表示編碼的輸出， 橙色表示加權，顏色越深表示 加權越大。可以看出第一行中最深的顏色出現在最左邊， 說明第一個解碼的輸出 更多地依賴於編碼輸出的前面部 分。

所以說注意力攝制就是要指定加權， 那麼加權是如何獲得的呢 ？如權並不 是預設的，而是鐘過網路學習去更新的 。解碼過程的每一步都會有一個加權，力日 權的計算依賴於解碼過程的輸入手 日曙藏狀態來計算的，並不依賴於開發過程的 輸出，而開發過程的輸出需要和加權結合起東輸出結果， 可以周圍 7.23 來簡單 表示。

7

8

6

。

。

l

2

4

2

3

4

5

6

．．

．．

．

B 山

圖 7.23 與加權結合的注軍力攝制

解碼過程中每一步都將輸入和曙聽狀態產生注意力加權， 由此和聞發過程的 輸出結合起來形成注意力攝制。

訕 4. teacher forcing

最佳提一點訓練中的技巧能夠加快肢敵速度 ，那就是 teacher forcing •這個 方法也特別簡單， 就是在解碼的過程， 不再是把前一步獲得的輸出當做下一步的 輸入，而是將正確的詞作為輸入， 因為前一步的輸出並不一定是正確的 ，所以使 用 teacher forcing 能夠保障作為輸入的永遺是正確的詞， 這樣就能夠加快敢做的 速度，

7.4.3 程式實現

前面的原理分析已經完全地說明了這一節內容背後的原理， 下面會遭遇程式 來加擇了解 ，

四 － 第 7 章 深度學習實戰

油 1. 資料前置處理

首先透過這個網址下載資料集： https://download.pytorch.org/tutorial/data.z 中 ， 然後將其解壓放到家 目錄下面。每個單字沒有辦法直接作為網路的輸入， 所以需 要給每個不同的學字一個索引作為它的標籤，可以透過下面的程式實現 u

1. class Lang ﹛ bject﹜ ﹒
2. def init (self , name) ,
3. self name - name
4. self w。rd2index a ﹛ ﹞
5. self .word2caunt a ﹝ ﹜
6. self.index2w 。rd a ｛ 。”sos”1 ”EOS ”﹜
7. self .n w。rds a 2 i Count SOS and EOS

8

1. def addSentence ﹛ self , sentence)

。。

1. f z w rd in sentence split ﹛ ’，﹜：

。 。

1. self addW rd (w rd) 12

。

1. def addW rd ﹝ self , word) ,

。 。

1. if w rd not in self .w rd2index,
2. self .w。rd2index ﹝w rd﹞ a self .n w。rds
3. self w。rd2caunt ﹝ ward﹞ a 1
4. self rndex2word ﹝ self .n\_words﹞ a W。rd
5. self n w。rds +a 1
6. else:
7. self w。rd2c unt ﹝w rd﹞ ＋＝ 工

上面的部分首先建立一個字典類別， r sos1 和 「 EOS 」 表示一句話的開始 標示和結束標示， 這兩個字元先給它們 0 和 l 的標籤，然後透過下面兩個函數 addSentence 和 addWord 去分析每句話，如果一句話中出現了沒有標記過的詞，就 在字典中建立一個新的對應關係。

為了簡化，我們透過上面的程式將 Unicode 的字元都轉化成 ASCII ’議所有 的字母都是小寫，去掉所有的標點。

1. def unic。deToAscii ( s﹜
2. return’，1。in ﹝
3. C f。r c in unic。dedata .normalize （ NFn ', s)
4. if unic。dedata .categ。ry (c) !a ’M泣，﹜
   1. 實例四一－Seq2seq：透過 R NN 實現簡單的 Neural Machine Translation 一

def normalizeString (s) ,

。 。

5 6

s a unicodeT AsCl.l. ﹛ s.l wer ﹛ I .strip II ﹜

7 8 1

s a re sub ﹝r”（ ﹝ ＇？﹞ ） ”，r" \1”，s﹜

s a re sub （主”﹝《 a-zA Z I ﹞＋” 主” ”，s) return s

nu

9

因為下載下來的資料集是一對一對地放在 txt 文件中的 ，所以需要定義一個函 數來讀取阻t 文件中的內容並儲存，我們可以定義下面 的函數來完成這件事：

def readLangs ﹛ langl, lanq2, reverseaFalse﹜ ．

1 2 3 4 5 6 7 8 9 1 1 1 1 1

print ﹛＂Reading lines．．”）

\* Read the frle and split into lines

lines a pen ﹛’data ／＼老s ＼老s.乞X仁 ，＼倍 （ langl, lang2) , enc dinga utf-8 ' ) .\

read () strip () split ﹛’＼n ＇﹜

# Split every line into pairs and normalize

pairs a ﹝ ﹝ n。rmalizeString ﹝ s) for s in 1 split （ ＼ t ﹜ ﹞ f r 1 in lines﹞

。

# Reverse pairs, make Lang instances

1 2 3 4 5 6

if reverse,

pa irs z ﹝ list ﹛ reversed (p﹜ ﹜ rnput\_lang " Lang ﹛ lanq2) output lang a Lang ﹛ langl)

1 1 1 1 2

else,

rnput\_lang " Lang ( langl﹜ output\_lang a Lang (lang2)

8 9 0

7

1

f。r p in pairs﹞

return input lang，。utput\_lang, pa工rs

為了簡化計算 ，我們不將整個資料集作為訓練 集，因為這樣需要的訓練時間 太長，只保留一部分以某些字首開頭的句子和 長度小於 JO 的句子作為訓練集進行 訓練 ，所以可以透過下面的方式過濾掉不滿足條件的資料集。

eng\_pref 血es a ﹛”i am ～ ＂i m ”，”he is”，＂he s ”，”she is”，盯she s”， ”y u are”，”y u re ”，”we are”，”we re ”，’，they are”，

1 2 3

”they re ＂﹜

4

def f,lterPair (pl .

5

return len (p﹝ O﹞. split （ ’’） I < MAX\_LENGTH and \

6

len (p ﹝ 11 split （ ’，I I < M由＜ LENGTH and \ p﹝ l﹞ startswith ﹛ eng\_pref 血es)

。

8 9

7

def filterPairs ﹛pairs) ·

。

1 1

return ﹝pair f r pair in pairs if frlterPair (pair） ﹞

1

接著將前面定義的函數組合在一起組成一個新的函數， 這個函數首先從文 字中讀取資料，然後過濾掉不滿足條件的資料， 按著將要進行翻譯的兩種語言一 次放在不同的字典下， 然後將所有出現過的詞與一個標籤對應起來， 儲存成兩個 字典。

def prepareData ( langl, lang2, reverse-False) ,

1 2 3 4 5 6 7 8 9 1 1

input lang, output lang, pairs - readLangs ( langl, lang2, reverse﹜

print ﹛”Read ＼電s sentence pairs ”＼老 len ﹛pairs) ) pa irs - filterPairs (pa工rs)

print ﹝”Trimmed t ＼ %s sentence pairs ”＼老 len (pairs) J

。

print （ 『Counting w rds ”）

主 r pair in pairs ﹒

input\_lang .addSentence (pair ﹝ 0﹞ ）

。utput\_lang addSentence (pair ﹝ l﹞ ）

。

print ﹛”Counted words ＂﹜

pr斗nt ( input lang.name, inpu七＿lang. n words) print ﹛ utput lang name utput\_lang.n w rds) P主int ( random.choice (pairs) )

1

3 4

1 1

2

1

return input\_lang。utput lang, pairs

透過三個函數依次對輸入的一句話做前宣處理： 第一個函數將一句話裡面 出 現的單字都轉化成對應的標籤﹔ 第二個函數在第一個函數的基礎上， 將一句話轉 換成對應的標籤序列﹔第三個函數依據於前面兩個定義好的函數， 將一組對應的 翻譯語言分別轉化成兩個標籤序列。

def 1ndexesFromSentence (lang, sentence) ,

1 2 3 4 5 6 7 8

return ﹝ lang.word2index ﹝w rd﹞ f r w rd in sentence.spl斗乞（’’﹜ ﹞

def tens rFr ，msentence (lang, sentence) ,

indexes a rndexesFr。nSentence (lang, sentence﹜

indexes append (ECS token﹜

result - torch.L。ngTens。r ﹝ indexes﹜

return result

7.4 實例囚一－Seq2seq 透過 R N N 實現簡單的 Neural Machine Translati。n -

1. def tens。rFr。mPair ( input lang’。utput\_lang, pair) :

9

1. rnput\_tensor - tens。rFr。mSentence ﹛ input lang, pair ﹝ O﹞ ）
2. target\_tensor - tensorFr。nSentence ﹛。utput l ang, pair ﹝ 1﹞ ）
3. return input tens。r, target\_tens。r

最後結合前面所有的函數， 自訂一個資料集繼承於 Da扭扭et 基礎類別，定義 好 ＿getitem＿ 和一len一兩個函數。

1. lass TextDataset ﹝ Dataset) :

。

1. def rnit一－ ﹛ self , datal ad-prepareData, langa ﹝ Y eng ”，fra’﹞ ）：
2. self , input lang, self ．。utp"t lang, self .pairs " datal。ad (
3. lang ﹝ 0﹞，lang ﹝ 1﹞，reverse-True)

。 。

1. self.rnput\_lang w rds " self.input\_lang n\_w rds

。

1. self.output\_lang\_w rds a self output\_lang.n words

7

1. def getitem (self , index) ·
2. return tens rF主。mPair ﹛ self .input lang, self.output lang,
3. self pairs ﹝ index﹞ ﹜

11

1. def len ﹝ self ) :
2. return len (self .pairs﹜

透過執行上面的程式，可以獲得圓 7.24 所示的結果。



圖 7.24 執行結果

可以發現透過過濾將資料大小從 135842 減少到了 10853，獲得英文和法語的 單字數 目分別是 2925 和 44 ，最後列出了一組英文和法語翻譯的文字。

油 2. 模型

下面建立序列到序列的模型。 先從最基本的編碼解碼模型 開始，建立兩個簡 單的循環神經網路。

首先建立一個編碼器 （ encoder ）。

class Enc。derRNN (nn .Mcdule)

1 2 3 4 5 6 7 8 9 1 1 1 1 1 1 1 1 1 1

def init＿一 （ self , input\_size, hidden\_size, n layers-1)

。

super (Enc derRNN, self ) . init () self .n layers - n layers self.hidden size hidden size

self .e 曲edd ing - nn .Embedding ﹛ input\_size, hidden size﹜ self .gru - nn .GRU (hidden\_size, hidden\_size)

。

def forward ﹛ self , input, hidden) · input - input .unsqueeze (1)

1 2 3 4 5 6 7 8 9 0 1 2 3

embedded a self embedding (input﹜ ＃ batch, hidden

。

utput - embedded .permute ( l, 0, 2﹜ f z i in range ﹛ self n layers) ,

。

。utput, hidden a self .gru （。utput, hidden﹜ return 。utput, hidden

def initHidden (self )

。 。

result a Variable (t rch.zer s﹛ 1, 1, self .hidden size﹞ ﹜

if use cuda ,

2 2 2

return result. cuda ﹛）

else: return result

2

初始化中定義了隱藏狀態的大小及網路層數， 但是這裡的網路層數與之前 我們講的網路層數不太相間， 這裡的層數不是定義在循環神經網路中， 而是定義 在網路之外的。換句話說，如果定義在循環神經網路 中，那麼有多少層就有多少 個隱藏狀態，而定義在循環網路外面， 那麼只有一個隱藏狀態，這個隱藏狀態會 繼續往後面的網路層中傳遞。在正向傳播中，將輸入的標籤序列轉化為詞向 量序 列，按著進入網路輸出結果和隱藏狀態，這裡的循環表示定義在 網路外的層數。

接著定義好解磚器 （ decoder ）。

class Dec。derRNN (nn Module) ·

1 2 3 4 5

def 一＿init一－ ﹝ self , hidden\_size ，。utput\_size, n layers-1)

super (DecoderRNN, self ) . 斗nit ﹛）

self .n\_layers - n\_layers

self .hidden size ζ hidden size

7白46

7.4 賣個四一－Seq2seq ：還過 RNN 實現簡單的 Neural Machine Translation -

6

*1* self .embedding = nn.Embedding ﹛ output\_size, hidden\_size﹜

且 self .gru = nn.GRU ﹛ hidden\_size, hidden\_size ﹜

‘9 self .out = nn.Linear ﹛ hidden\_size, output\_size﹜

10 self .softmax = nn. LogSoftmax ()

11

1. def forward ﹛ self , input, hidden﹜：
2. output = self .embedding ( input) i batch, 1, hidden 14 output = output. permute (1，口 2﹜ t 1, batch, hidden 工 5 for i in range ﹝ self .n\_layers) :

16 output = F. relu ﹛ output】

. 17 output, hidden = self .gru ﹛ output, hidden﹜

1. output = self .softmax ﹛ self .out (output [O﹞ ﹜ ﹜
2. return output, hidden

20

1. def initHidden (self ) :
2. result = variable ﹛ torch.zeros (l, 1, self .hidden size﹜ ﹜

. 23 if use cuda:

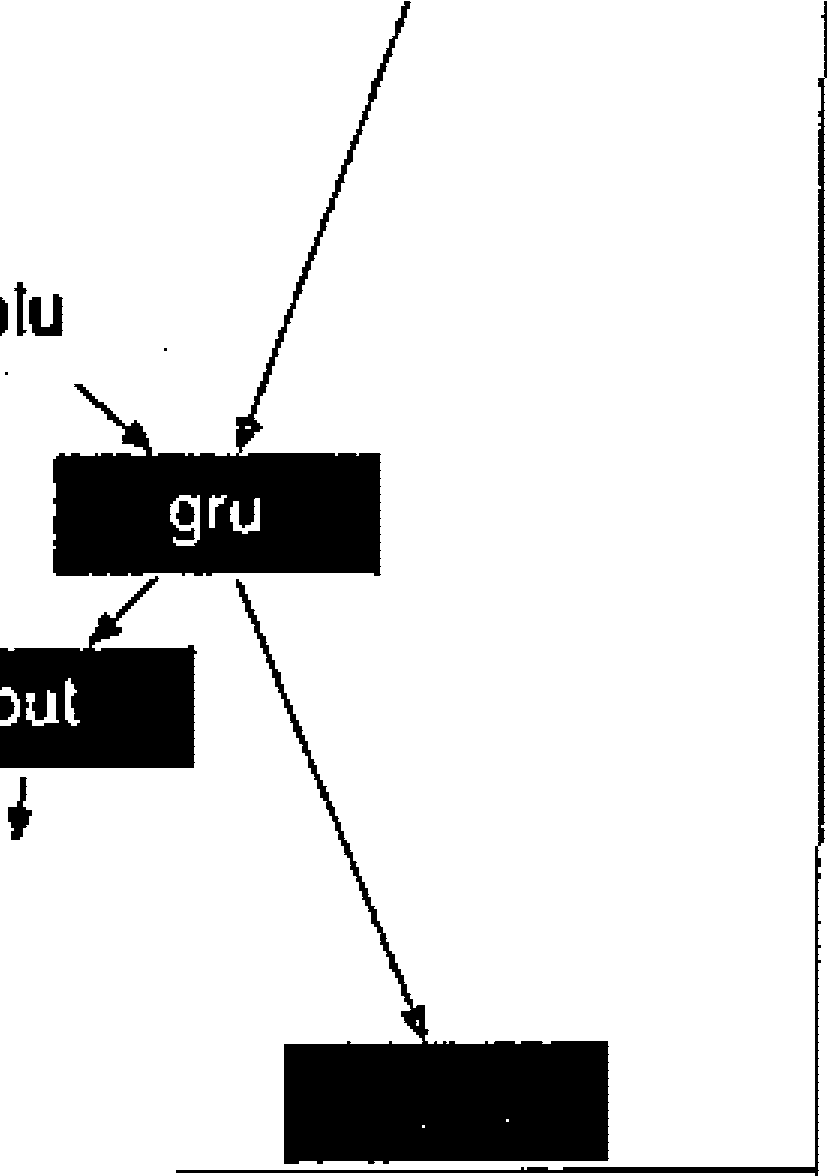
1. return result.cuda ()
2. else:

26 return result

解碼的過程跟編碼的過程聽乎是一樣的， 唯一不同的是會多定義一個騙性層 作為輸出層，將推度轉換到單字的數目，因為最佳是根據所有輸出 中的最高權車 來確定輸出的。

編碼和解碼的整個過程可以表示成圖 7.25 。

．．



•&1mi!ili!'i1H1.t•

re

..-.·111．softmax

自 ．

圖 7.25 騙碼和解瞌

一一一 第 7 章 深度學習實戰

左邊是開發過程，右邊是解碼過程， 都是由簡單的循環神經網路所組成的 。 建立完最簡單的模型之後， 我們可以將注意力機制增加到解碼器中建立更複

雜的模型。

因為注意力機制是增加到解碼器中的，所以編碼器不需要做任何修改， 只需 要重新建立一個含有注意力機制的解碼器。

工 class AttnDec。derRNN ﹝nn.Module﹜ ：

1. def init ﹝ self ,
2. hidden size,

。

1. ctpct\_size,
2. n layersal,

。。

1. dr p ct\_paO .1,
2. max\_ lengtha坦白｛＿LENGTH)

。

1. super ﹛AttnDec derRNN, self ）， 工nit 11
2. self.hidden size a hidden size
3. self .output\_size a output\_size

11 self .n layers a n layers

。。

1. self .dropout\_p a dr p ut\_p
2. self .max\_length a max\_length

14

。

1. self .embedding a nn .Embedding ﹛ self . utput\_size, self hidden\_size)
2. self attn a nn Linear (self .hidden size ' 2, self max\_length)
3. self .attn c。mbine a nn L斗near (self hidden size ' 2, self hidden size)
4. self .dropout a nn .Dropout (self .dr。pout\_p)
5. self gru a nn .GRU ﹛ self hidden size, self .hidden size﹜

。 。

1. self . ut a nn .Linear (self hidden\_s工 ze, self utpct size) 21

22 def f rward (self , input, hidden, encoder一。ctputs) ·

23

1. input , batch, 1
2. hidden 1, batch, hidden

。

1. enc der\_outputs· length, hidden

27

1. embedded a self .embedd斗ng ﹛ input) # batch, 1, hidden
2. en甘bedded a self .drop。ut (embedded)
3. embedded a en甘bedded.squeeze (1﹜ ＃ batch, hidden 31

。

1. attn\_weights a F s ftmax ﹝

。

1. self .attn ﹛t rch.ca七 I Iembedded, hidden ﹝ O﹞ I , 11 ﹜ ）

7.4 實例四一－Seq2seq ：透過 R N N 實現簡單的 Neural Machine Translati。n -

1. \* batch, max length
2. encoder cutputs 呈 enc。der 。utputs unsqueeze ( 0)
3. \* ba七ch, max\_length, hidden
4. attn\_applied 呈 t rch brr,n (attn\_weights unsqueeze ﹛ 1) , encode芝一 utputs)
5. \* batch, 1, hidden
6. 。utput a to玄ch cat ( (en甘bedded, attn\_applied squeeze ﹝ 1) ) , 1)
7. \* batch, 2xhidden

甲

、叫

a

p叩

43

耐

b

1. for 1 in range ﹝ self .n\_layers﹜ －
2. 。utput a F. relu （。utput ﹜

4 6 。utput, hidden a self .gru ( output, hidden)

47

48 。utput a F.log\_sof 七max ﹝ self out ﹛。utput. squeeze (0﹜ ﹜ ）

4 9 return output, hidden, attn\_weights

50

1. def initHidden (self )

。

1. result a Variable ( t rch.zeros ( l, 1, self hidden size﹜ ）
2. if use cuda
3. return result cuda I)
4. else﹒
5. return result

上面就是含有注意力機制的解碼器改 良版，可以和前面定義的最基本的解碼 器比較一下，可以發現它們之間存在一些相同的部分。

初始化中引用了一些其他的網路層都是為了定義注意力機制， 而最重要的注 意力機制在上面的程式中 已經增加好了詮釋， 這些註釋是為了方便了解每一步之 後的資料大小的，可降低錯誤機率。

我們直接考慮 forward 裡面的過程，首先將網路輸入轉化為詞向量 ，然後將 詞向量和隱藏狀態連接在一起， 按著透過線性層加 softmax 敢動層輸出固定長度 的序列。這個序列就是注意力序列，每個數的大小表示注意力的重要程度， 然後 使用 torch.hmm 將開發過程的輸出和注意力加權透過批矩陣乘法變得輸出結果 ， 最後將這個結果和網絡輸入連接在一起， 透過一個線性層將維度轉化為循環神經 網路接受的維度，將其作為網路的輸入， 最後透過網路獲得最後的輸出。

透過上面的方式，我們就在解碼的過程中增加了注意力機制。

－第 7 章 深度學習實戰

油 3.模型訓練

建立好網路之後就可以 開始模型訓練了，得益於 PyTorch 的動態園和指令式 程式設計，所以在模型的訓練中可以增加 for 循環，非常方便。

首先是編碼的過程，先定義一個零序列， 每一步編碼之後的輸出都填入這個 零序列當中，最後將隱藏狀態儲存為解碼過程的初始隱藏狀態 。

* 1. enc der＿ u乞puts - Variable I
  2. t rch.zer s﹝MAX LENGTH, enc der hidden s工ze) )

。

* 1. if t rch.cuda is available ﹛）
  2. enc。der一 utputs - encoder一 utputs cuda ﹛）

。 。

* 1. enc der hidden a enc der 工nitHidden I)
  2. f。r ei in range ( in lang.size (l)) .
  3. enc der＿ utput, enc der\_hidden a enc der ﹝
  4. in lang ﹝ ，ei﹞，encoder\_hidden ﹜
  5. enc der utputs ﹝ ei﹞ ＝ enc der utput ﹝ O﹞ ﹝ O﹞ 10

11 dee。der hidden a enc。der hidden

按著定義解碼過程， 這裡分為兩種情況：一種是不使用注意力機制， 一種是 使用注意力機制。

對於不使用注意力機制的訓練過程，非常簡單，每次將輸入和隱藏狀態傳入 解碼器中 ，將獲得的結果再傳入解碼器的下一步就可以了 ，在循環中判斷一下， 如果輸入的是結束符號，那麼這句話就結束了， 跳出這個循環。

1. f。z di in range ﹛。ut lang.size ( l)) ·
2. dee der＿ utput, decoder\_hidden a dee der (
3. dee。der input, decoder\_hidden ﹜
4. 1 ss +- cnterion (decoder output ut\_lang ﹝ ，di﹞ 1
5. topv, topi a dee der utput data.t pk ( l)
6. ni a t pi ﹝ O﹞ ﹝ O﹞

7

1. dee der\_input a Vanable (t rch.LongTens r （ ﹝ ﹝ ni﹞ ﹞ ﹜ ﹜
2. if torch.cuda .is available ﹛），
3. dee。der input a dee。der input.cuda ﹛）
4. if ni aa EDS t。ken
5. break

如果使用注意力攝制， 只需在解碼的過程中傳入騙碼的輸出，其餘部好是一 樣的。

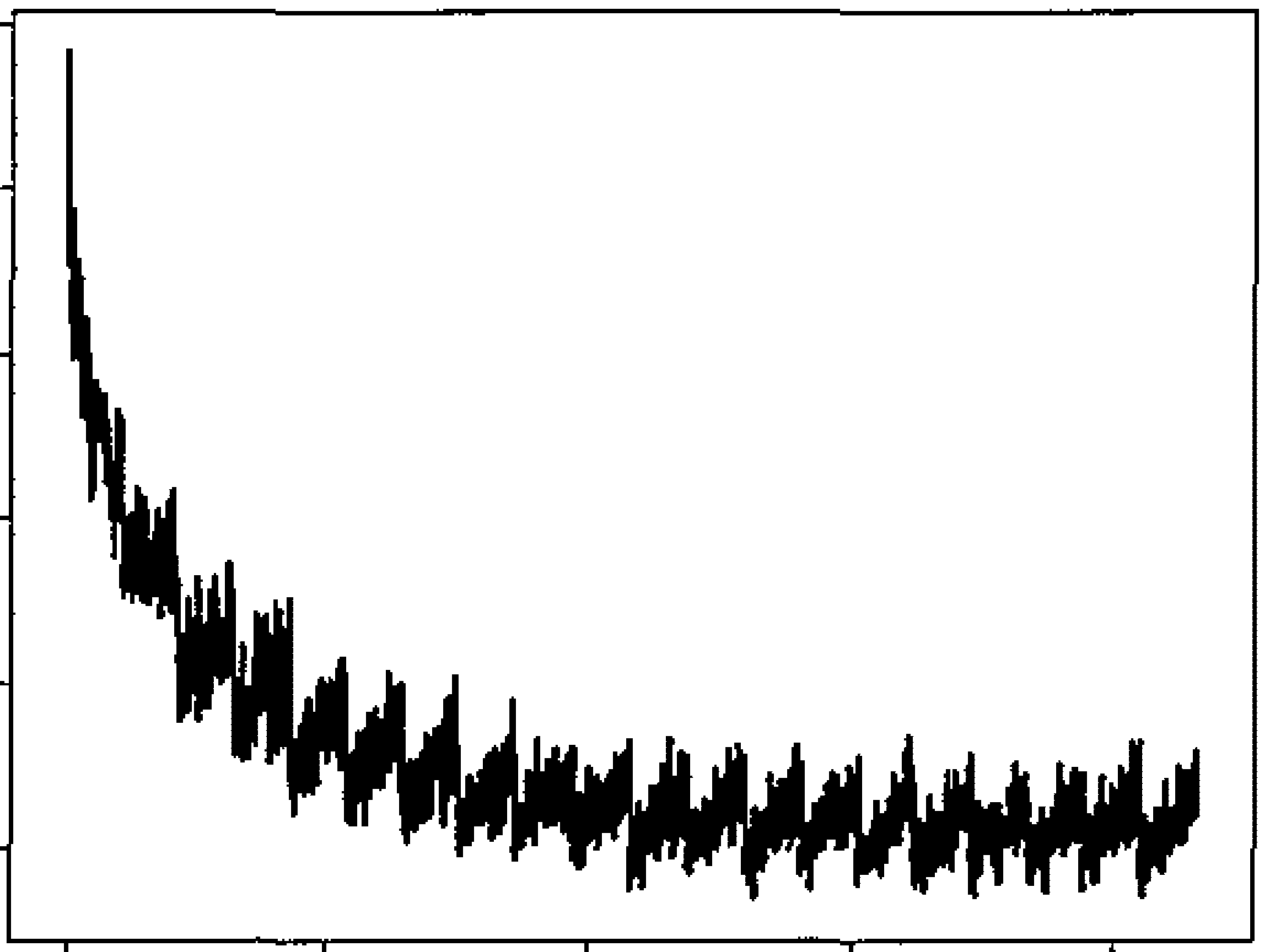
1. for di in range (out\_lang.size ( l） ﹜ Z
2. decoder\_output, decoder\_hidden, decoder\_attention = attn\_decoder (
3. decoder\_input, decoder\_hidden, enc口der\_outputs﹜
4. loss += criterion ﹝ decoder\_output, out\_lang ﹝：，di﹞ ﹜
5. topv, t口pi = decoder output.data.topk ﹛ 1)
6. ni = topi ﹝ D﹞ ｛ 0﹞

7

1. decoder input = Variable ﹛ torch.LongTensor ( [[ni﹞ ﹞ ﹜ ﹜
2. if torch. cuda. is available ﹝ ﹜ t
3. decoder\_input = decoder\_input.cuda ()
4. if ni == EOS token:
5. break

網路使用注意力機制，並訓練了 20 次，能夠獲得圖 7.26 所示的說差效果園 ，

*30*



25

20

15

10

5

口 500 1000 1500 2000

圖 7.26 誤差強果圖

如果希望能鉤獲得更好的效果， 可以使用 teacher forcing’以及更加複諱的網 路結構和挂意力攝制的多種變式。

圓圓圓 團 第 7 章 深度學習實戰

油 4. 模型評估

訓練獲得的模型需要評估其效果如 何，而評估的過程跟網路的訓練過程是類 似的，不斷將前一步網路獲得的結果作為網路下一步的輸入， 直到最後輸出一句 話的終止標示或是長度超過最大長度限制。

透過隨機從文字中搜尋幾句話，能夠獲得園 7.27 所示的翻譯效果， 其中第一 句話是符翻譯的法文， 第二句話是真實的英文翻譯， 第三句話是機器翻譯獲得的 結果，可以發現透過簡單序列到序列的模型，機器翻譯能夠取得一定的效果。

1111輛矗國輯斟姐

祖師臨國輛圓圓

團1回國國輛輛

嘲鵬翻祖祖團

a

e

e

e

S

l

e

e s s

t

t

t

C

「 o

t

t t u

dp

ncu t c at rap thu

tc

n

。cq 9

n

5 1 1

1 3 S U U U S

nn

〕1 1

\* Z 〈

圈 1國團團祖棚圈圈

S

n

〉

E

〈

p u

c

up

耳聞蘭幅圖祖祖

國幢幢闊姐擱

—冒聶聾回祖祖帽圓圓

觀蘭輔輔國團

圖 7.27 翻譯效果

同時還可以將注意力的加權視覺化，檢視一下網路在翻譯每個單字的時候到 底注意力更加集 中在哪些部分，如圖 7.28 所示。

可以看到在翻譯一句話的時帳， 控意力也是不斷集中於編碼輸出的不同區 雄。同時注意力機制還有一個很好的性質， 即具有很強的解釋性，能夠幫助我們 了解網路到底從哪裡學習、 學到了什磨。

。 2 4 6 自

。

1

2

3

4

5

6

7

自

圖 7.28 注意力的加權視覺化

7.4.4 歸納

透過上面的部分， 我們學會了如何使用序列到序列的模型去建置簡單的機器 翻譯系統， 同時也了解了注意力機 I制的原理，以及如何使用注意力攝制來改善 模型效果，有興趣的單者可以嘗試其他的語言翻譯， 例如中文到英文。

序列到序列模型的應用不僅限於此， 還能夠用於對話系統、 問答系統等，如 果請者有興趣， 可以嘗試更多的序列到序列的模型應用。 同時注意力機制近來也 變得非常流行，Google 最近發表了一篇論文， 只使用注意力攝制而非用任何實際 的網路結構來做機器翻譯， 達到了目前業界的最好水準， 可以看出注意力攝制有 很多值得探尋的地方， 期待讀者們帶著問題去閱讀相闊論文， 找到更多有意義的 應用。