

**ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH**



**LUẬN VĂN TỐT NGHIỆP ĐẠI HỌC
PHÂN LOẠI ẢNH
BẰNG KỸ THUẬT DEEP LEARNING**

HỘI ĐỒNG: CÔNG NGHỆ PHẦN MỀM

GVHD: ThS. Nguyễn Cao Trí

GVPB: PGS. TS. Quản Thành Thor

---00---

SVTH1: Vũ Ngọc Linh 51201929

SVTH2: Nguyễn Đức Hoàn 51201191

TP. HỒ CHÍ MINH, 05/2017

LỜI CAM ĐOAN

Chúng tôi cam đoan toàn bộ nội dung của báo cáo đều do nhóm chúng tôi nghiên cứu, thực hiện và biên soạn, không sao chép từ bất kì tài liệu nào khác. Các thông tin tham khảo trong báo cáo đều được nêu rõ nguồn gốc. Chúng tôi sẽ chịu trách nhiệm nếu có bất cứ sai phạm nào so với lời cam đoan.

Tp. Hồ Chí Minh, ngày 16/05/2017

Nhóm sinh viên thực hiện đề tài
Vũ Ngọc Linh, Nguyễn Đức Hoàn

LỜI CẢM ƠN

Đầu tiên, chúng tôi xin gửi lời cảm ơn chân thành nhất đến ThS. Nguyễn Cao Trí, người đã hướng dẫn tận tình, cung cấp tài liệu và đóng góp những lời khuyên hết sức quý báu từ những ngày đầu nhóm chúng tôi thực hiện đề tài.

Chúng tôi xin gửi lời cảm ơn tới tất cả các thầy cô trong khoa Khoa học và Kỹ thuật Máy tính nói riêng và các thầy cô trong trường Đại học Bách Khoa nói chung đã hết lòng truyền dạy những kiến thức cũng như kinh nghiệm thực tế trong suốt quãng thời gian chúng tôi học tập tại trường.

Chúng tôi cảm ơn tác giả, đồng tác giả của những quyển sách, bài báo, tài liệu, khóa học... được chúng tôi sử dụng làm tài liệu tham khảo.

Chúng tôi xin cảm ơn gia đình và bạn bè đã hỗ trợ, quan tâm và đồng hành cùng chúng tôi trong suốt khoảng thời gian học tập tại trường.

Một lần nữa chúng tôi xin chân thành cảm ơn.

Tp. Hồ Chí Minh, ngày 16/05/2016

Nhóm sinh viên thực hiện đề tài

Vũ Ngọc Linh, Nguyễn Đức Hoàn

TÓM TẮT LUẬN VĂN

Với sự phát triển của Internet và các thiết bị công nghệ, lượng dữ liệu hình ảnh được tạo ra và lưu trữ trở nên không lồ và đa dạng. Trước yêu cầu tận dụng, khai thác nguồn dữ liệu giá trị này, một số lĩnh vực của Khoa học máy tính là Thị giác máy tính (Computer Vision) và Học máy (Machine Learning) đã có sự phát triển mạnh trong thời gian gần đây. Thị giác máy tính hiện diện mọi nơi trong cuộc sống, từ các ứng dụng như tìm kiếm bằng hình ảnh, tự động gán thẻ trên facebook, nhận diện khuôn mặt, bản đồ..., đến ứng dụng trong y khoa, nông nghiệp, các công việc tự động hóa (máy bay không người lái, ô tô tự điều khiển). Bản chất của các ứng dụng trên là việc giải quyết các bài toán phân loại, định vị và phát hiện hiện hình ảnh. Với sự phát triển gần đây của Deep Learning, vấn đề giải quyết các bài toán trên đã dễ dàng và có hiệu suất cải thiện rõ rệt so với các phương pháp khác.

Trong bài báo cáo này, chúng tôi sẽ giới thiệu về Deep Learning từ các kiến thức cơ bản đến các mô hình được áp dụng trong bài toán phân loại hình ảnh. Nhận thấy được những ưu điểm của mô hình Convolutional Neural Network, chúng tôi quyết định áp dụng mô hình này để giải quyết bài toán cụ thể: Phân loại ảnh nông nghiệp. Qua việc giải quyết bài toán phân loại ảnh nông nghiệp, chúng tôi muốn thực nghiệm độ chính xác và hiệu quả của mô hình Convolutional Neurol Network, qua đó xây dựng và phát triển một ứng dụng có thể áp dụng vào thực tế.

ABSTRACT

With the development of the Internet and technology devices, the amount of image data created and stored has become enormous and varied. Prior to using this valuable data in the most effective way, some fields of Computer Science such as Computer Vision and Machine Learning have developed rapidly recently. Computer vision is everywhere in life, from applications such as image search, facebook tagging, face recognition, mapping, etc. to medical and agricultural applications... Automation work (unmanned aircraft, self-driven cars). The nature of the above applications is solving problems of image classification, localization and detection. With the recent development of Deep Learning, it has been easier to solve these problems and has significant improvement compared to other methods.

In this paper, we will introduce Deep Learning from the basic knowledge to the models applied in the image classification problem. Recognizing the advantages of the Convolutional Neural Network model, we decided to apply this model to solve the specific problem: Agricultural photo classification. By solving the problem of agricultural imaging classification, we want to experiment on the efficiency and effectiveness of the Convolutional Neurol Network, thereby building and developing an application that can be applied in practice.

BỘ CỤC LUẬN VĂN

Nội dung của Luận văn gồm 4 chương, nội dung từng chương như sau:

Chương 1. Giới thiệu Giới thiệu sơ lược về đề tài thực hiện, lí do chọn đề tài và mục tiêu của đề tài.

Chương 2. Cơ sở lý thuyết Trình bày cơ sở lý thuyết để thực hiện đề tài, từ các bài toán cơ bản của Machine Learning đến Convolutional Neural Network, một trong những mô hình Deep Learning tiên tiến phổ biến trong bài toán phân loại ảnh thời gian gần đây.

Chương 3. Ứng dụng CNN trong phân loại ảnh Nông nghiệp Giới thiệu về việc nghiên cứu và kết quả làm việc của nhóm với bài toán phân loại ảnh Nông nghiệp.

Chương 4. Tổng kết Chỉ ra những nội dung làm được, chưa làm được và hướng phát triển của đề tài.

MỤC LỤC

Chương 1. GIỚI THIỆU	1
1.1 Giới thiệu đề tài	1
1.2 Mục tiêu đề tài	3
1.3 Phạm vi đề tài	3
Chương 2. CƠ SỞ LÝ THUYẾT	5
2.1 Machine Learning	5
2.1.1 Định nghĩa	5
2.1.2 Phân loại	6
2.2 Bias và Variance thông qua Linear Regression	7
2.2.1 Linear Regression	7
2.2.2 Mô hình biểu diễn của Linear Regression	7
2.2.3 Bias và Variance	8
2.3 Logistic Regression và Neural Network	9
2.3.1 Logistic Regression	9
2.3.2 Neural Network	15
2.4 Các yếu tố ảnh hưởng đến việc huấn luyện	16
2.4.1 Tiền xử lý dữ liệu	16
2.4.2 Khởi tạo tham số	17
2.4.3 Gradient Descent	18
2.4.4 Regularization	29
2.4.5 Learning rate	30
2.5 Convolution Neural Network	33
2.5.1 Kiến trúc tổng quan	33
2.5.2 Các lớp cấu tạo nên Convolutional Neural Network	34
2.5.3 Một số mô hình kiến trúc Convolutional Neural Network	41
Chương 3. ÚNG DỤNG CNN TRONG PHÂN LOẠI ẢNH NÔNG NGHIỆP	46
3.1 Giới thiệu bài toán	46
3.2 Giới thiệu tập dữ liệu	46

3.3	Kết quả	47
3.3.1	Training với SGD	48
3.3.2	Training với Nesterov Accelerated Gradient	50
3.3.3	Training với Adam.....	52
3.3.4	Training với Adam không có Dropout.....	54
3.3.5	Training với chỉ 1 Convolutional Layer	56
	CHƯƠNG 4. TỔNG KẾT	58
4.1	Tổng kết.....	58
4.1.1	Những việc đã làm được	58
4.1.2	Những việc chưa làm được	58
	TÀI LIỆU THAM KHẢO	59
	PHÂN CÔNG CÔNG VIỆC.....	60

MỤC LỤC HÌNH

Hình 1.1 Số lượng các tổ chức phối hợp đầu tư cùng NVIDIA trong lĩnh vực deep learning	2
Hình 1.2 Số lượng các bài báo học thuật về Deep Learning từ năm 2007 đến năm 2015... .	3
Hình 2.1 Phân loại Machine Learning	6
Hình 2.2 Bias và Variance	8
Hình 2.3 Hàm phi tuyến Sigmoid	10
Hình 2.4 Hàm phi tuyến Tanh.....	10
Hình 2.5 Rectified Linear Unit	11
Hình 2.6 Bài toán donut với 2 biến đầu vào x_1, x_2	12
Hình 2.7 Bài toán donut với biến đầu vào x_1, x_2 và x_1x_2	13
Hình 2.8 Bài toán donut với biến đầu vào x_1, x_2, x_{12}, x_{12}	14
Hình 2.9 Neural network với 1 hidden layer - 3 neurons	15
Hình 2.10 Neural Network với 1 hidden layer và Neural Network với 2 hidden layer....	16
Hình 2.11 Các dạng tiền xử lý dữ liệu phổ biến	16
Hình 2.12 Vanilla gradient descent trên ngôn ngữ python	19
Hình 2.13 Gradient descent trên mặt cắt của bề mặt lồi và gradient trên mặt cắt của mặt không lồi	20
Hình 2.14 Vanilla Minibatch Gradient Descent	20
Hình 2.15 Biến động SGD	22
Hình 2.16 Hiện thực SGD bằng numpy	23
Hình 2.17 Mô phỏng hàm loss SGD	23
Hình 2.18 Góc nhìn vật lý SGD với momentum	24
Hình 2.19 SGD không sử dụng Momentum	24
Hình 2.20 SGD có sử dụng Momentum	25
Hình 2.21 Minh họa thuật toán SGD với Momentum và NAG	25
Hình 2.22 Cập nhật Nesterov	26
Hình 2.23 Thuật toán Adagrad.....	27
Hình 2.24 Thuật toán RMSprop	28
Hình 2.25 Thuật toán Adam.....	28
Hình 2.26 So sánh sự tối ưu SGD trên các thuật toán tối ưu	29
Hình 2.27 Neural Network trước và sau khi áp dụng Dropout.....	30
Hình 2.28 Learning rate quá cao.....	31
Hình 2.29 Learning rate cao.....	31
Hình 2.30 Learning rate quá nhỏ	32
Hình 2.31 Đánh giá learning rate qua đồ thị hàm loss.....	33
Hình 2.32 Convolutional Neural Network.....	34
Hình 2.33 Filter và Convolved Feature trong Convolutional Layer	36
Hình 2.34 Filter và Convolved Feature trong Convolutional Layer sau khi thực hiện phép trượt với Stride = 1	37

Hình 2.35 Minh họa tính toán kích thước output volume.....	38
Hình 2.36 Toán tử downsampling	40
Hình 2.37 Minh họa Pooling Layer	40
Hình 2.38 Kiến trúc LeNet.....	41
Hình 2.39 Kiến trúc AlexNet.....	41
Hình 2.40 Kiến trúc ZF Net	42
Hình 2.41 6 kiến trúc khác nhau của VGG Net.....	43
Hình 2.42 Kiến trúc GoogleLeNet.....	44
Hình 2.43 Kiến trúc Microsoft ResNet.....	45
Hình 3.1 Ba loại ảnh được sử dụng trong bài toán phân loại ảnh nông nghiệp.....	46
Hình 3.2 Kiến trúc sử dụng trong mô hình nhóm sử dụng	47
Hình 3.3 Đồ thị hàm loss khi training với SGD	49
Hình 3.4 Đồ thị hàm loss khi training với NAG.....	51
Hình 3.5 Đồ thị hàm loss khi training với Adam.....	53
Hình 3.6 Đồ thị hàm loss khi traning với Adam không có Dropout.....	55
Hình 3.7 Đồ thị hàm loss khi training với chỉ 1 Convolutional Layer	57

MỤC LỤC BẢNG

Bảng 3-1 Training với SGD.....	48
Bảng 3-2 Training với NAG	50
Bảng 3-3 Training với Adam	52
Bảng 3-4 Training với Adam không có Dropout.....	54
Bảng 3-5 Training với chỉ 1 Convolutional Layer.....	56

DANH MỤC THUẬT NGỮ

Deep Learning: Học sâu

Machine Learning: Học máy

Supervised Learning: Học có giám sát

Unsupervised Learning: Học không giám sát

Reinforcement Learning: Học tăng cường

Regression Problem: Bài toán hồi quy

Classification Problem: Bài toán phân lớp

Polynomial regression: Hồi quy đa thức

Lagrange Interpolating Polynomial: Nội suy đa thức Lagrange

Neural Network: Mạng nơ-ron

Neural: Nơ-ron

Loss function (or Cost function, or Objective function): Hàm mục tiêu

Hypothesis function: Hàm lý thuyết

Convolutional Neural Network: Mạng nơ-ron tích chập

Learning rate: Tỉ lệ học

Layer: Lớp, tầng

Epoch: Một chu kỳ duyệt qua toàn bộ mẫu dữ liệu huấn luyện

Chương 1. GIỚI THIỆU

1.1 Giới thiệu đề tài

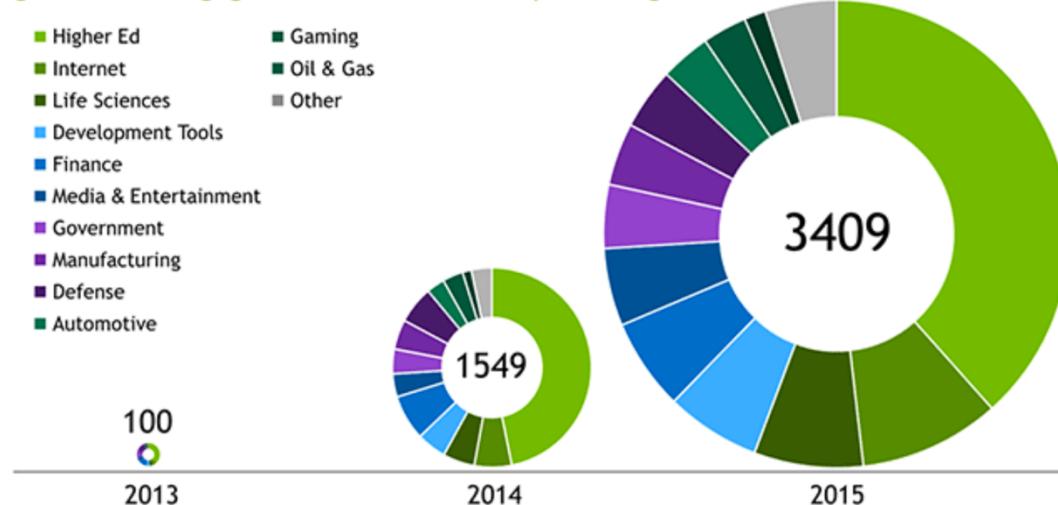
Trên cơ sở sự phát triển mạnh mẽ, toàn diện và vượt bậc của cuộc cách mạng công nghiệp lần thứ 3, đặc biệt là sự phát triển của công nghệ số, phần cứng máy tính, phần mềm và Internet, đời sống xã hội cũng như nền kinh tế toàn cầu đã có những biến đổi sâu sắc. Đó cũng là cơ sở cho sự ra đời của cuộc cách mạng được nhắc tới khá nhiều và phổ biến trong thời gian gần đây: Cuộc cách mạng công nghiệp lần thứ 4.

Trong kỷ nguyên của cuộc cách mạng mới này, có thể coi lĩnh vực Trí tuệ nhân tạo (Artificial Intelligence) là một lĩnh vực đóng vai trò cốt lõi và quan trọng. Trong lĩnh vực Trí tuệ nhân tạo, một lĩnh vực con của nó là Machine Learning đã có sự phát triển mạnh mẽ. Machine Learning cho phép máy tính có thể tự học thay vì được lập trình để thực hiện một tác vụ cụ thể nào đó. Mục tiêu của Machine Learning là nghiên cứu và xây dựng một hệ thống có thể tự “học” tự động từ dữ liệu đã có để giải quyết một vấn đề cụ thể. Một thuật toán Machine Learning là một thuật toán có thể học qua huấn luyện. Thuật toán về cơ bản được khởi tạo bằng cách nhận một tập dữ liệu với output đã biết, so sánh sự khác biệt giữa output mà thuật toán tính ra và output chính xác đã biết, sau đó chỉnh sửa các tham số input để tăng độ chính xác của output mà thuật toán tính ra. Đó là một đặc điểm của các thuật toán Machine Learning: chất lượng của kết quả sẽ được cải thiện qua kinh nghiệm mà thuật toán học được. Chúng ta càng cung cấp nhiều dữ liệu, một hệ thống dự đoán chính xác hơn sẽ được tạo ra.

Có rất nhiều cách tiếp cận với Machine Learning, mỗi cách sử dụng một thuật toán khác nhau để tối ưu kết quả dựa vào dữ liệu mà ta có được. Mỗi cách tiếp cận sẽ có ưu điểm, nhược điểm riêng và có thể kết hợp nhiều cách tiếp cận khác nhau. Thuật toán để giải quyết bài toán cụ thể phụ thuộc vào nhiều yếu tố, trong đó bao gồm tính chất của tập dữ liệu. Trong số các cách tiếp cận để giải quyết một bài toán cụ thể trong Machine Learning, Deep Learning là một lĩnh vực mới, trở nên phổ biến trong vòng một thập kỷ trở lại đây.

EVERY INDUSTRY WANTS INTELLIGENCE

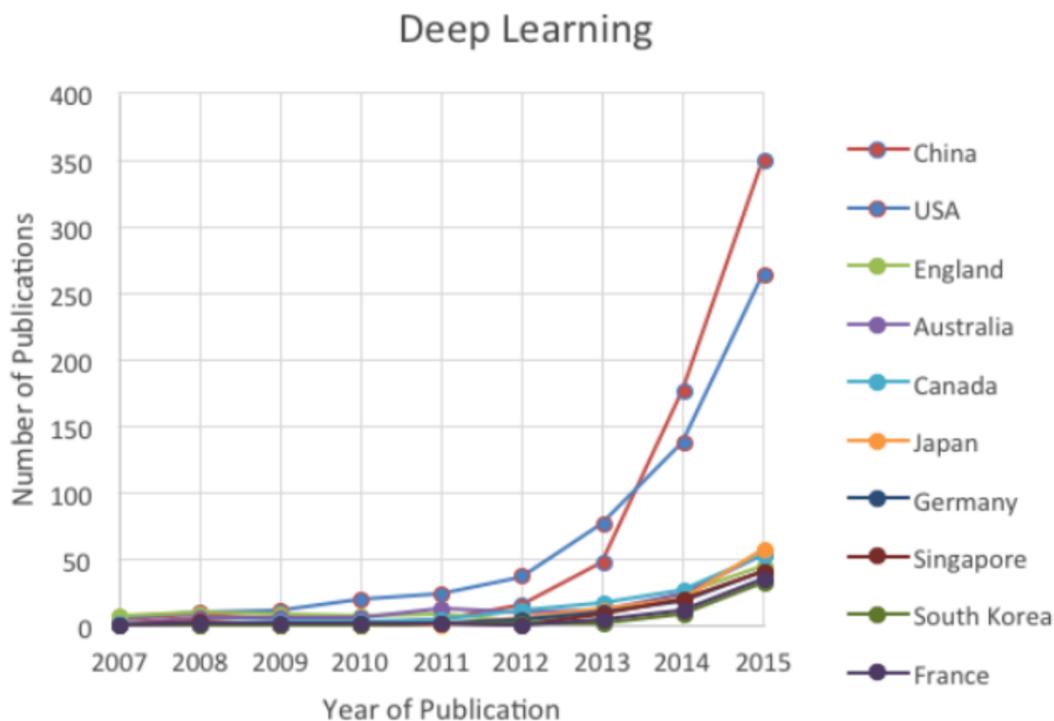
Organizations engaged with NVIDIA on deep learning



Hình 1.1 Số lượng các tổ chức hợp tác cùng NVIDIA trong lĩnh vực deep learning¹

Thuật toán của Deep Learning lấy cảm hứng từ mô hình hoạt động của não bộ con người, nó tăng độ hiệu quả trong việc xử lý các bài toán nhận dạng các phần tử trong hình ảnh, dịch ngôn ngữ thời gian thực, sử dụng giọng nói điều khiển thiết bị, các bài toán trong di truyền học, phân tích ngữ nghĩa, xác định khối u trong y khoa và rất nhiều ứng dụng khác. Tuy Deep Learning không phải là cách phù hợp để giải quyết mọi vấn đề vì nó yêu cầu một tập dữ liệu lớn cho việc huấn luyện, cần sức mạnh xử lý lớn cho việc huấn luyện và chạy mô hình, nhưng đây cũng là một cách tiếp cận giải quyết các bài toán Machine Learning có nhiều ưu điểm và hiệu năng vượt trội.

¹ Source: <https://www.slideshare.net/LuMa921/deep-learning-the-past-present-and-future-of-artificial-intelligence>



Hình 1.2 Số lượng các bài báo học thuật về Deep Learning từ năm 2007 đến năm 2015²

Trước xu hướng thực tế và mong muốn tìm hiểu kiến thức mới, nhóm đã tìm hiểu về kỹ thuật Deep Learning, cụ thể là mô hình Convolutional Neural Network và áp dụng vào bài toán phân loại hình ảnh trong Nông nghiệp.

1.2 Mục tiêu đề tài

Mục tiêu của đề tài là tìm hiểu về Deep Learning, bao gồm các công việc:

- Tìm hiểu về Deep Learning và các kiến thức liên quan để thực hiện bài toán phân loại ảnh.
- Sử dụng một open source – TensorFlow để hiện thực bài toán phân loại ảnh trên công cụ đã tìm hiểu.

1.3 Phạm vi đề tài

Trong phạm vi đề tài, nhóm chỉ nghiên cứu và áp dụng những kiến thức liên quan đến Deep Learning có thể áp dụng cho bài toán phân loại ảnh theo mục tiêu của đề tài đưa ra. Các kiến thức tìm hiểu chỉ để chạy trên một máy tính cá nhân duy nhất phục vụ cho mục

² Source: <https://www.slideshare.net/LuMa921/deep-learning-the-past-present-and-future-of-artificial-intelligence>

tiêu nghiên cứu tìm hiểu đơn giản mà không áp dụng cho các hệ thống file lớn, phân tán trong các mô hình công nghiệp.

Chương 2. CƠ SỞ LÝ THUYẾT

Deep Learning (còn gọi là Neural Network) là một phần quan trọng trong Machine Learning. Trong chương này, trước tiên chúng ta sẽ nói sơ qua về định nghĩa và phân loại của Machine Learning, khái niệm “learning” nghĩa là gì và “learning” như thế nào thông qua thuật toán cơ bản nhất là Linear Regression. Ở đây chúng ta sẽ tiếp xúc luôn với vấn đề bias và variance có ảnh hưởng như thế nào đối với bài toán cần giải quyết. Kế tiếp, chúng ta sẽ đến với Logistic Regression phân tích những ưu điểm và nhược điểm của thuật toán này trong các bài toán phân loại để có thể thấy được Deep Learning cơ bản là một bản nâng cấp từ thuật toán này. Sau đó, xem xét các yếu tố ảnh hưởng đến kết quả đầu ra của thuật toán như tiền xử lý dữ liệu, khởi tạo tham số, một số hàm kích hoạt phổ biến được sử dụng để cho kết quả trong loss function, gradient descent dùng trong “learning” giảm thiểu giá trị loss function, các thuật toán tối ưu gradient descent, regularization để giảm high variance và đánh giá thay đổi learning rate thông qua biểu đồ loss function để đạt kết quả tối ưu nhất. Cuối cùng, chúng ta sẽ có cái nhìn tổng quan về Convolution Neural Network, thuật toán hiệu quả nhất về phân loại ảnh thời điểm hiện tại.

2.1 Machine Learning

2.1.1 Định nghĩa

Machine Learning là một lĩnh vực của Trí tuệ nhân tạo liên quan đến việc nghiên cứu và xây dựng các thuật toán tạo nên một hệ thống có thể “học” tự động từ dữ liệu để giải quyết vấn đề cụ thể. Thông thường, khi nói đến một hệ thống Machine Learning là nói đến một hệ thống có thể học để giải quyết vấn đề tốt hơn trong tương lai dựa trên những kinh nghiệm trong quá khứ. Một điểm cần phải nhấn mạnh nữa của Machine Learning là tính tự động. Nói cách khác, mục tiêu của Machine Learning là đưa ra một thuật toán có thể học một cách tự động mà không cần phải có sự can thiệp hay sự trợ giúp của con người. Dưới góc nhìn của một lập trình viên, cơ chế của Machine Learning có thể được xem như là việc lập trình qua các mẫu. Với một bài toán cụ thể, thay vì lập trình để giải quyết bài toán một cách trực tiếp, Machine Learning tìm kiếm một cách thức giải quyết bài toán này thông qua những mẫu được cung cấp sẵn.

Có rất nhiều định nghĩa về Machine Learning. Dưới góc nhìn Trí tuệ nhân tạo, có 2 định nghĩa đắt xuât được biết đến nhiều nhất.

Arthur Samuel, một trong những người tiên phong trong lĩnh vực Machine Learning, đã mô tả về Machine Learning: “Là lĩnh vực cho máy tính khả năng học mà không phải lập trình một cách trực tiếp.”. Đây là một định nghĩa cũ, không chính thống.

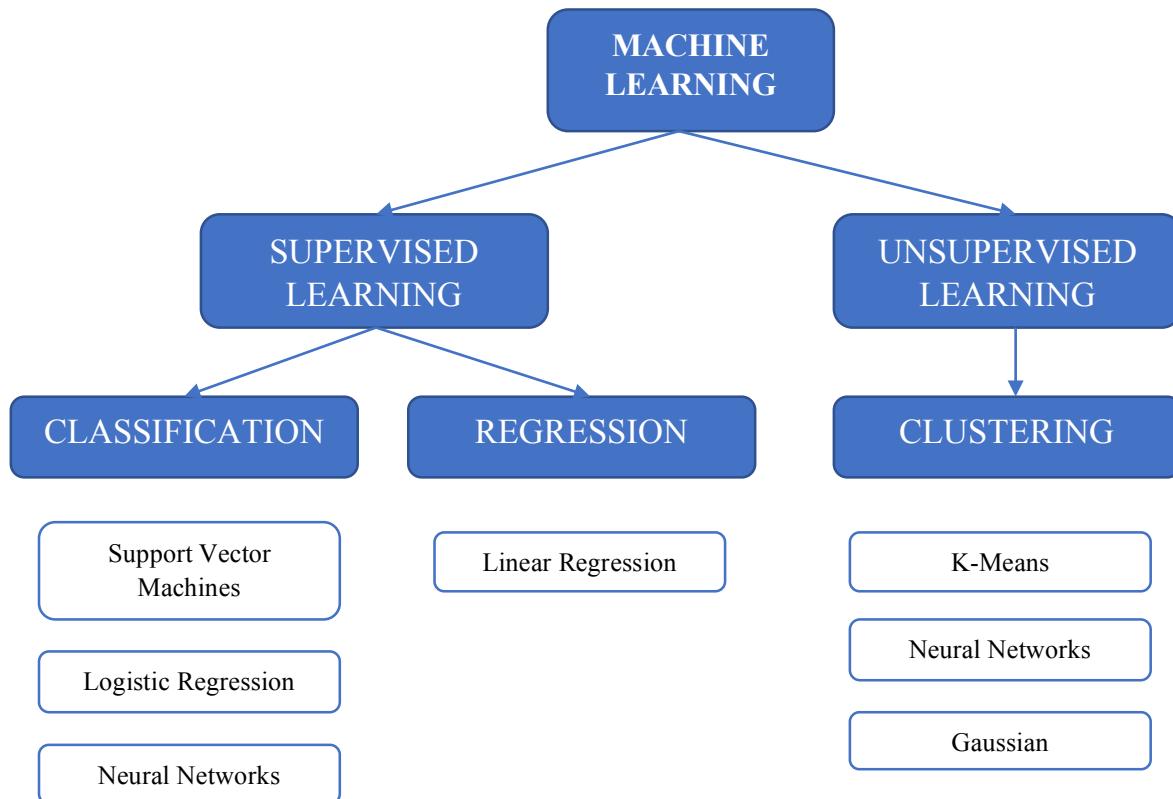
Tom Mitchell, giáo sư đầu ngành trong lĩnh vực Machine Learning định nghĩa cụ thể và chuẩn mực hơn: “Máy tính được gọi là học từ kinh nghiệm (dữ liệu) E với tác vụ T và được đánh giá bởi độ đo P nếu máy tính khiếu tác vụ T này cải thiện được độ chính xác P thông qua kinh nghiệm (dữ liệu) E cho trước.”

Để hiểu hơn về định nghĩa Machine Learning của Tom Mitchell, ta có ví dụ cụ thể sau: Giả sử ta cần một hệ thống Machine Learning để thực hiện một tác vụ xác định một tin nhắn có phải là spam hay không. Theo định nghĩa của Tom Mitchell:

- Tác vụ T: Xác định 1 tin nhắn có phải là spam hay không.
- Kinh nghiệm E: Xem lại những tin nhắn đánh dấu là spam xem có những đặc tính gì để có thể xác định nó là spam.
- Độ chính xác P: Là tỷ lệ số tin nhắn spam được phân loại đúng.

Khi đó, thông qua càng nhiều kinh nghiệm E, tức là càng có nhiều dữ liệu về những tin nhắn đánh dấu là spam, độ chính xác P của tác vụ T sẽ càng được cải thiện tốt hơn^[1].

2.1.2 Phân loại



Hình 2.1 Phân loại Machine Learning

Machine Learning được chia thành nhiều loại trong đó 2 loại được biết đến nhiều nhất là Supervised Learning và Unsupervised Learning, ngoài ra còn có Reinforcement Learning.

Trong phạm vi của đề tài này chúng ta sẽ chỉ nói về Supervised Learning.

Supervised Learning là một kỹ thuật học máy để học tập từ tập dữ liệu được gán nhãn cho trước. Tập dữ liệu cho trước sẽ chứa nhiều bộ dữ liệu. Mỗi bộ dữ liệu có cấu trúc theo

cặp $\{x, y\}$ với x được xem là dữ liệu thô và y là nhãn của dữ liệu đó. Nhiệm vụ của Supervised Learning là dự đoán đầu ra mong muốn dựa vào giá trị đầu vào. Để nhận ra, Supervised Learning là học dựa vào sự trợ giúp của con người, hay nói cách khác con người dạy cho máy học và giá trị đầu ra mong muốn được định trước bởi con người. Tập dữ liệu huấn luyện hoàn toàn được gán nhãn dựa vào con người. Tập càng nhỏ thì máy tính học càng ít.

Supervised Learning cũng được áp dụng cho 2 nhóm bài toán chính là Regression Problem và Classification Problem.

Thực chất “learning” trong supervised learning là học để xây dựng một hàm có thể xuất ra giá trị đầu ra tương ứng với tập dữ liệu. Ta gọi hàm này là hàm $h(x)$ (hypothesis function) và mong muốn hàm này xuất ra đúng giá trị y với một hoặc nhiều tập dữ liệu mới khác với dữ liệu được học. Hàm $h(x)$ cần các loại tham số học khác nhau tùy thuộc với nhiều bài toán khác nhau. Việc học từ training dataset cũng chính là tìm ra bộ tham số học cho hàm $h(x)$.^[2]

2.2 Bias và Variance thông qua Linear Regression

2.2.1 Linear Regression

Với Regression Problem, chúng ta nhận các biến đầu vào và cố gắng tạo ra kết quả đầu ra phù hợp với các nhãn ban đầu dựa trên một hàm xuất kết quả tuyến tính hay nói cách khác là cố gắng tìm một đường phù hợp nhất đi qua các điểm được cho. Các dạng bài toán này có thể được giải bằng Linear Regression.

Một ví dụ đơn giản của Linear Regression là Linear Regression với một biến còn được biết đến như là “univariate linear regression”. Chúng ta có thể giải bài toán này bằng lý thuyết toán học như Lagrange Interpolating Polynomial hoặc một cách chung đối với các bài toán supervised learning là tối ưu hóa objective function bằng gradient descent để tìm tham số thích hợp cho hypothesis function $h(x)$. Cách thứ nhất chỉ có thể với trường hợp đặc biệt này của bài toán (một biến), còn cách hai sử dụng tổng quát cho tất cả các trường hợp.

Cách sử dụng cách một có thể được xem thêm trong phần tài liệu kham khảo, ở phạm vi của đề tài này chúng ta sẽ chỉ dùng cách hai để giải quyết các bài toán.

2.2.2 Mô hình biểu diễn của Linear Regression

Đối với mỗi giải thuật trong supervised learning đều sẽ có một cặp hypothesis và objective function tương ứng khác nhau.

Với Linear Regression, hypothesis function là một hàm tuyến tính theo tất cả các biến đầu vào được cung cấp, còn objective function chính là hàm tính tổng bình phương độ

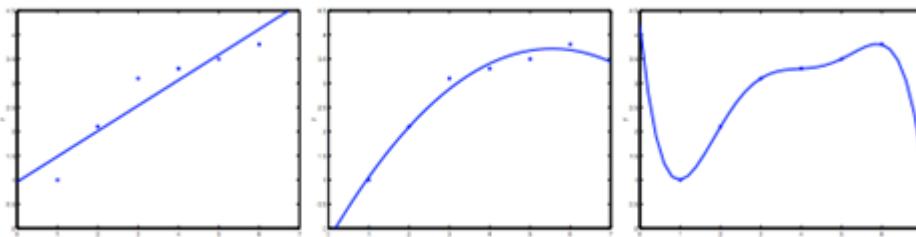
lệch giữa kết quả đầu ra với các nhãn được cung cấp hay còn được gọi là squared error function hoặc mean squared error.

Qua đó chúng ta có thể thấy, “learning” cụ thể là dùng gradient descent nhằm làm giảm giá trị objective function đến mức thấp nhất để đạt được bộ tham số tối ưu sử dụng trong hypothesis function để đưa ra được kết quả phù hợp với nhãn dán mong muốn.

2.2.3 Bias và Variance

Trở lại với bài toán một biến đầu vào, ta có thể thấy nếu dữ liệu phân bố trên một đường cong thì heuristic function bậc một với biến đầu vào sẽ cho ta một đường best-fit là một đường thẳng thì không thực sự tốt. Để tăng cường sự phù hợp thì việc tăng số lượng biến có thể xem là một cách, để tăng biến mà không ảnh hưởng đến kết quả bài toán ta có thể tạo ra biến bằng cách kết hợp những biến cũ đã có (combine features) hoặc thay đổi hành vi của hypothesis function từ hàm bậc một thành một bậc khác bất kì (polynomial regression).

Việc áp dụng hoặc không áp dụng combine hoặc polynomial có thể dẫn đến các trường hợp dưới đây:



Hình 2.2 Bias và Variance³

Giả sử ta có bài toán dự đoán giá trị y từ một giá trị $x \in \mathbb{R}$ cho trước. Đồ thị ngoài cùng bên trái của hình 2.2 sẽ hiển thị kết quả đường best-fit của tập dữ liệu là đường thẳng $y = w_0 + w_1$. Chúng ta thấy rằng dữ liệu không thật sự nằm trên đường thẳng đó, vì vậy đường thẳng trên không phù hợp để giải quyết việc dự đoán giá trị y . Trường hợp này được gọi là underfitting (High Bias).

Thay vào đó, nếu chúng ta thêm một đặc tính x^2 , khi đó ta có đường best-fit $y = w_0 + w_1x + w_2x^2$, được biểu bằng đồ thị ở giữa của hình 2.2, chúng ta sẽ nhận được một đường thể hiện mối quan hệ giữa x và y tốt hơn.

Nhưng nếu quá nhiều đặc tính được thêm vào, như đồ thị ngoài cùng bên phải hình 2.2 là kết quả của đường best-fit với hàm thể hiện mối quan hệ giữa x và y là $y = \sum_{j=0}^5 w_i x^i$ là đa thức bậc 5. Ta thấy mặc dù dữ liệu nằm trên đường này một cách hoàn toàn, tức dữ

³ Source: <https://www.coursera.org/learn/machine-learning/supplement/VTe37/the-problem-of-overfitting>

liệu đã cho hoàn toàn khớp với hàm này, nhưng trong thực tế chưa chắc hàm này đã dự đoán chính xác giá trị y từ giá trị x cho trước. Trường hợp này được gọi là overfitting (High Variance).^[1]

2.3 Logistic Regression và Neural Network

2.3.1 Logistic Regression

2.3.1.1 Định nghĩa

Đối với Classification Problem, ta có gắng tiên đoán kết quả với output rời rạc. Nói cách khác, chúng ta có gắng ánh xạ biến đầu vào thành các danh mục rời rạc.

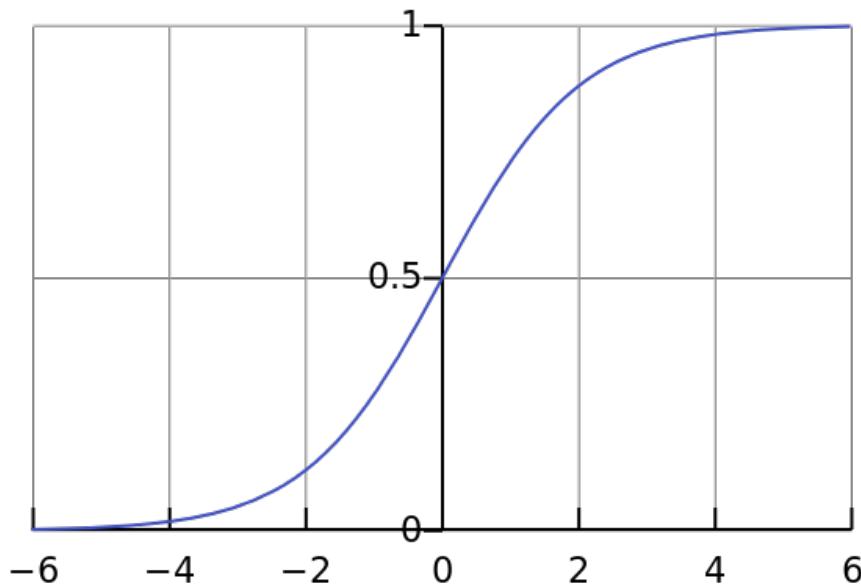
Classification Problem cũng giống như Regression Problem, ngoài trừ việc chúng ta muốn dự đoán giá trị y là các giá trị rời rạc.

Chúng ta có thể tiếp cận bài toán phân loại mà không quan tâm đến việc y là các giá trị rời rạc và sử dụng Linear Regression để dự đoán x từ y. Tuy nhiên, dễ dàng nhận thấy những ví dụ mà phương pháp này có kết quả không tốt. Ví dụ như không có nghĩa gì khi hàm $h_0(x)$ cho giá trị lớn hơn 1 hoặc nhỏ hơn 0 khi chúng ta biết rằng x thuộc $(0, 1)$. Chúng ta sẽ tìm cách thay đổi dạng của hàm $h_0(x)$ để thỏa mãn $0 \leq h_0(x) \leq 1$. Chúng ta thực hiện điều này bằng cách đặt hàm tuyến tính vào hàm kích hoạt.

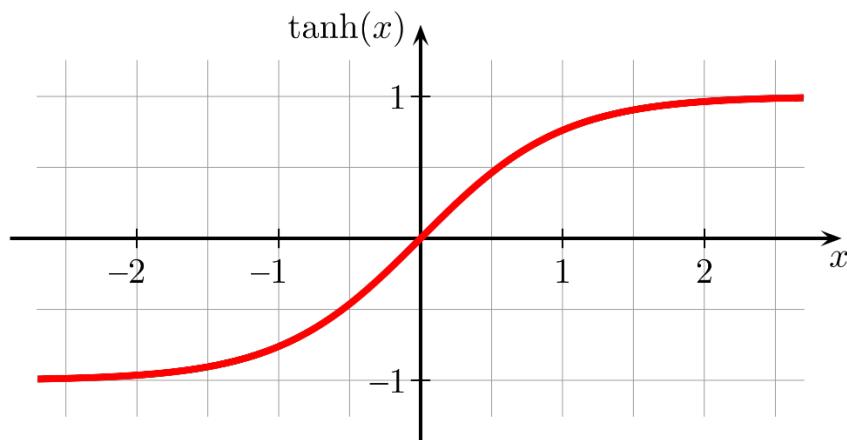
2.3.1.2 Hàm kích hoạt

Một số hàm kích hoạt cơ bản

Hàm Sigmoid. Hàm Sigmoid có dạng công thức toán học là $\sigma(x) = 1/(1 + e^{-x})$. Hàm này có tham số x là một số thực bất kỳ và tính ra giá trị hàm là một giá trị có giá trị nằm trong đoạn từ 0 tới 1. Với trường hợp tham số x là một giá trị âm cực lớn, hàm trả ra giá trị bằng 0. Với trường hợp tham số x là một giá trị dương cực lớn, hàm trả ra giá trị bằng 1. Hàm Sigmoid đã từng được sử dụng nhiều trong quá khứ nhờ biểu diễn của nó tương tự như việc dẫn truyền xung thần kinh của nơron thần kinh. Trong thực tế hiện nay, hàm phi tuyến Sigmoid đã trở nên lỗi thời và hiếm khi được sử dụng. Nhược điểm chính của hàm Sigmoid là làm mất hệ số góc và giá trị của hàm không đổi xứng qua giá trị 0.

Hình 2.3 Hàm phi tuyến Sigmoid⁴

Hàm Tanh. Hàm Tanh có dạng công thức toán học đơn giản là $\tanh(x) = 2\sigma(2x) - 1$ với σ là hàm Sigmoid. Hàm này có tham số x là các số một số thực bất kỳ và giá trị hàm là một giá trị thực nằm trong đoạn từ -1 tới 1. Hàm Sigmoid vẫn còn một số nhược điểm của hàm Sigmoid nhưng đã loại bỏ được tính chất giá trị của hàm không đổi xứng qua giá trị 0 (vì giá trị hàm nằm trong đoạn từ -1 tới 1. Vì vậy, trong thực tế người ta thường sử dụng hàm phi tuyến Tanh hơn hàm phi tuyến Sigmoid.

Hình 2.4 Hàm phi tuyến Tanh⁵

⁴ Source: https://en.wikipedia.org/wiki/Sigmoid_function#/media/File:Logistic-curve.svg

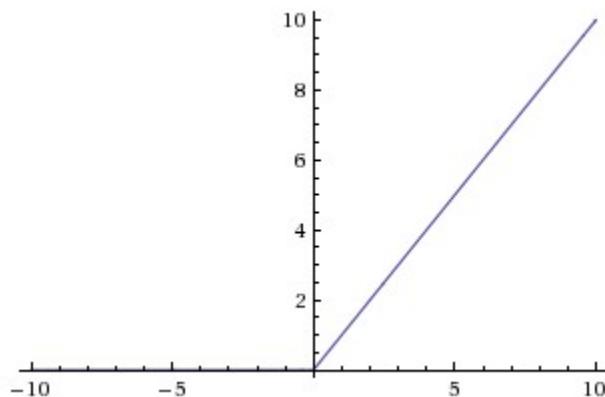
⁵ Source https://commons.wikimedia.org/wiki/File:Hyperbolic_Tangent.svg

ReLU

Rectified Linear Unit trở nên phổ biến trong một vài năm trở lại đây, sử dụng công thức $f(x) = \max(0, x)$.

Ưu điểm: Tăng tốc độ huấn luyện, hiện thực đơn giản. (hội tụ nhanh hơn nhiều so với hàm sigmoid/tanh trong thực tiễn (khoảng 6 lần), hiệu quả tính toán cao hơn, không bão hòa trên miền dương).

Nhược điểm: Output không zero-centered và các đơn vị ReLU unit có thể bị yếu trong suốt quá trình train dẫn đến việc “chết” (dead). Ví dụ, hệ số góc lớn qua một ReLU neuron có thể là nguyên nhân mà các tham số được cập nhập theo một cách mà neuron này sẽ không bao giờ được kích hoạt vào điểm dữ liệu đó nữa. Đó là việc mà các đơn vị ReLU có thể bị “chết” suốt quá trình huấn luyện. Ví dụ, có thể khoảng 40% của network có thể bị “chết” nếu learning rate quá cao.



Hình 2.5 Rectified Linear Unit⁶

Leaky ReLU. Một biến thể của ReLU giải quyết vấn đề “dying ReLU”. Thay vì giá trị của hàm bằng 0 khi $x < 0$, leaky ReLU sẽ có một hệ số góc dương nhỏ (khoảng 0.01). Công thức toán học của hàm ReLU sẽ là $f(x) = 1(x < 0)(\alpha x) + 1(x \geq 0)(x)$ với α là một hằng số có giá trị nhỏ.

Maxout. Loại đơn vị cuối cùng được đề xuất không có dạng hàm số $f(w^T x + b)$.

Maxout là tổng quát hóa của ReLU và leaky ReLU. Một Maxout neuron có dạng $\max(w_1^T x + b_1, w_2^T x + b_2)$. Cả ReLU và leaky ReLU đều là trường hợp đặc biệt của Maxout. Maxout vừa có các ưu điểm của ReLU, vừa không có các nhược điểm mà ReLU có. Tuy nhiên nhược điểm của Maxout là nó nhân đôi số tham số cho mỗi neuron, dẫn tới số lượng tham số cao.

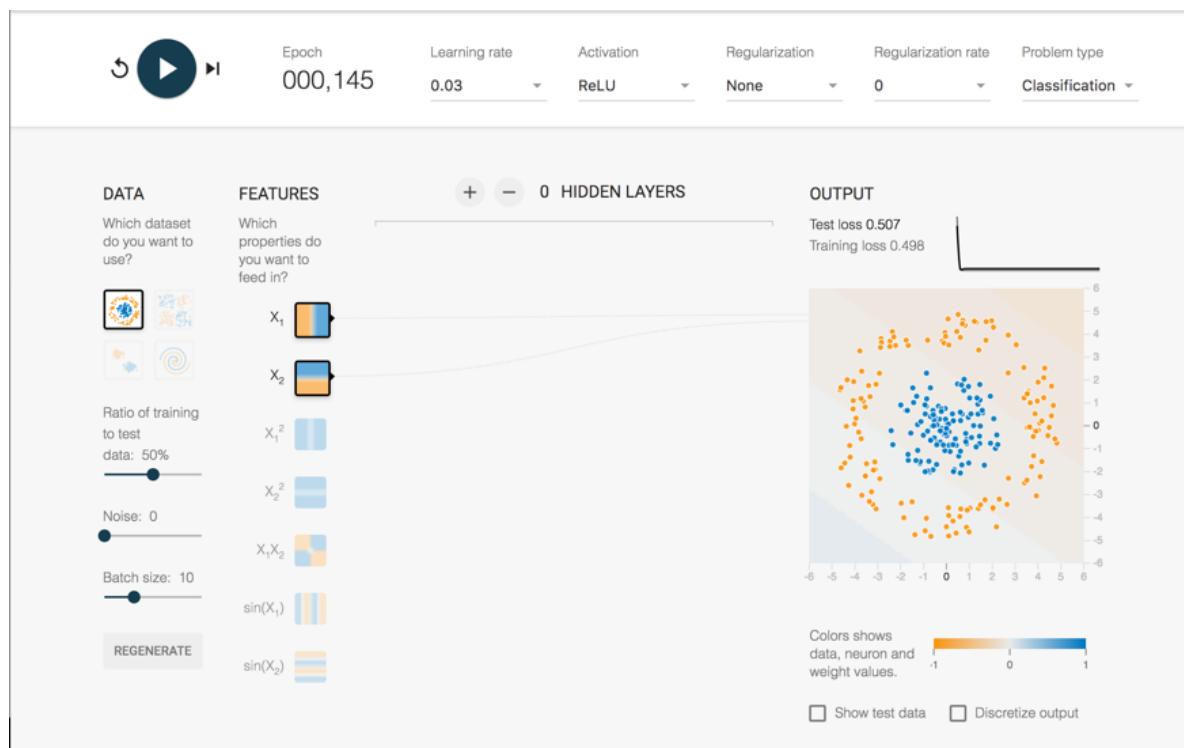
⁶ Source: <http://cs231n.github.io/neural-networks-1/>

Rất hiếm khi trộn lẫn và dùng các loại hàm activation trong một network, mặc dù về cơ bản không có vấn đề gì khi làm như vậy.^[4]

2.3.1.3. Nhược điểm Logistic Regression:

Cũng giống như Linear Regression, để tăng độ chính xác cho giải thuật với một số bài toán như donut, xor... chúng ta không thể đơn giản xây dựng mô hình cho giải thuật với các biến đầu vào để mong muốn ra một kết quả chính xác.

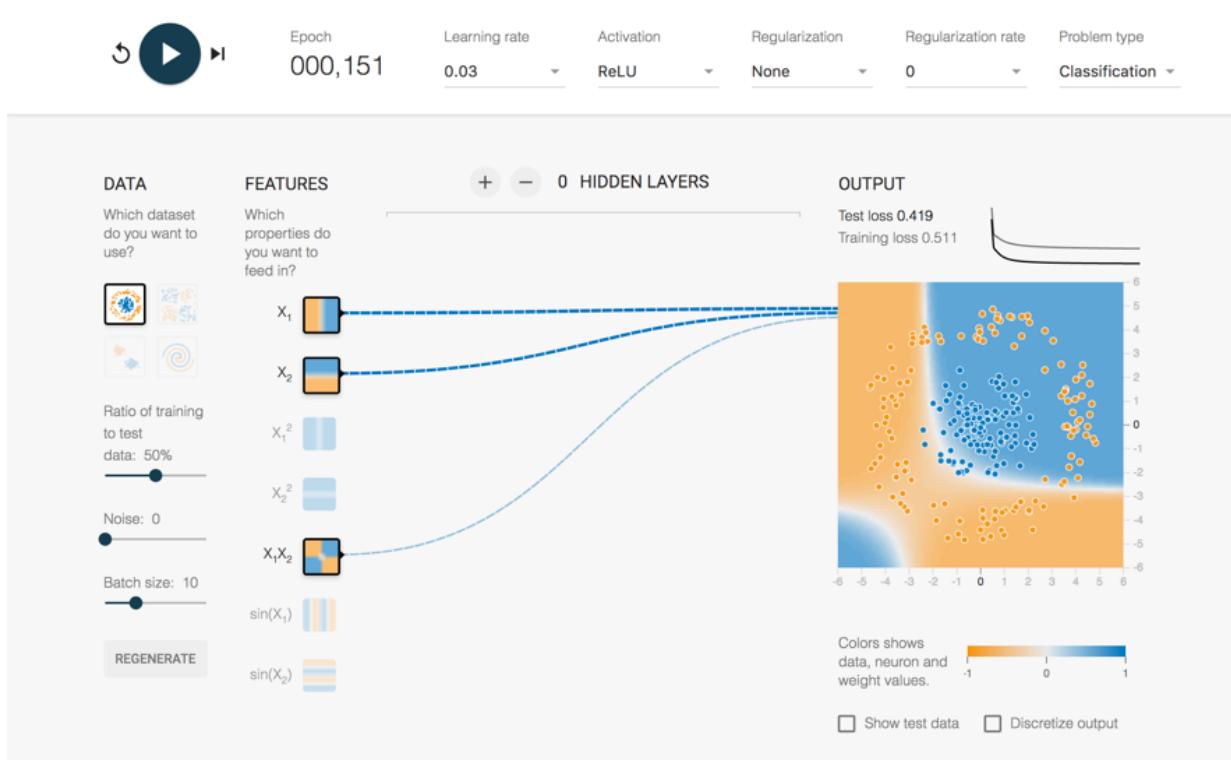
Ví dụ bài toán donut chỉ dùng biến đầu vào x_1, x_2 chúng ta sẽ không thể tìm được các tham số thích hợp cho hypothesis function mà phải áp dụng polynomial để tăng đặc điểm của mô hình này.



Hình 2.6 Bài toán donut với 2 biến đầu vào x_1, x_2 ⁷

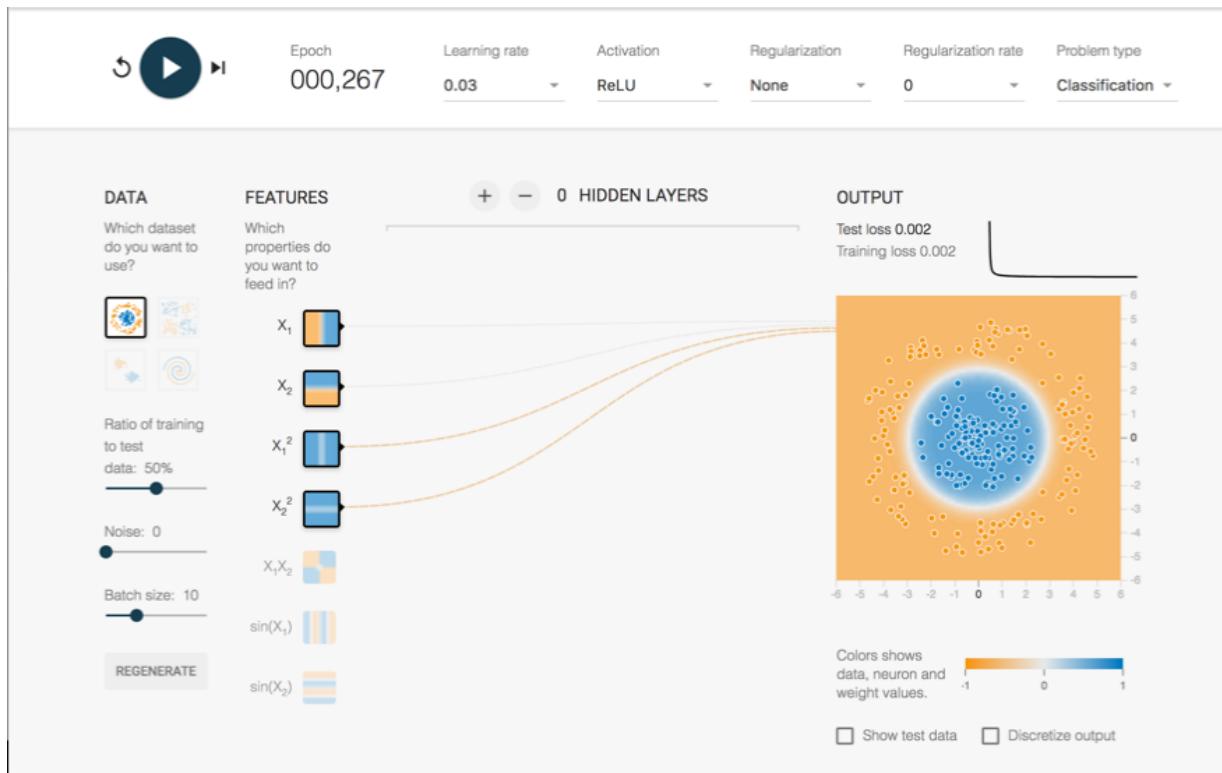
Việc chọn đặc điểm cũng phải phụ thuộc vào kinh nghiệm, như ở đây ta thấy việc tăng thêm đặc điểm x_1x_2 sẽ không giúp cải thiện cho ra một lời giải chính xác, trong khi đó tăng thêm đặc điểm x_1^2, x_2^2 sẽ cho ra một kết quả hoàn hảo.

⁷ Source: <http://playground.tensorflow.org/>



Hình 2.7 Bài toán donut với biến đầu vào x_1, x_2 và x_1x_2 ⁸

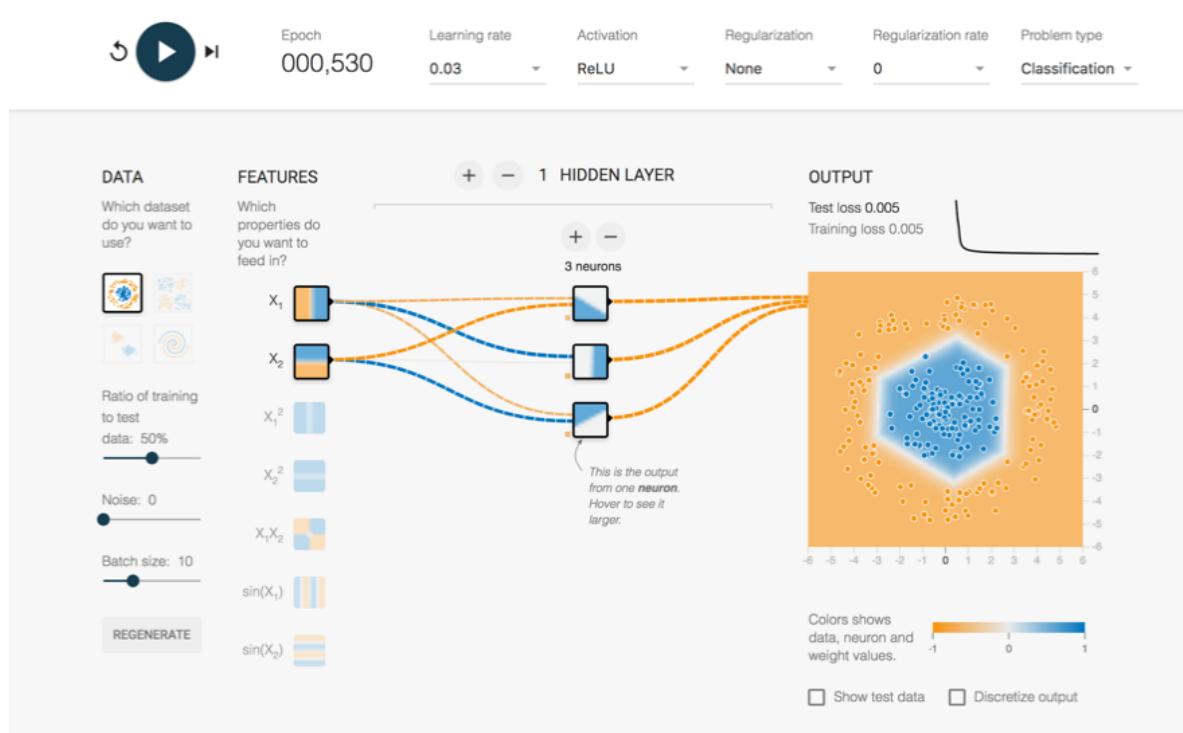
⁸ Source: <http://playground.tensorflow.org/>



Hình 2.8 Bài toán donut với biến đầu vào x_1, x_2, x_1^2, x_2^2 ⁹

Việc xác định thêm những tính chất nào không phải là một chuyện đơn giản. Nhưng chúng ta cũng có thể thấy rằng việc thêm một layer trung gian để layer này đảm nhiệm việc tính toán các đặc điểm cần thiết mới cho bài toán là một cách giải quyết hay và tổng quát cho mọi bài toán phân loại này, đây cũng chính là Neural Network.

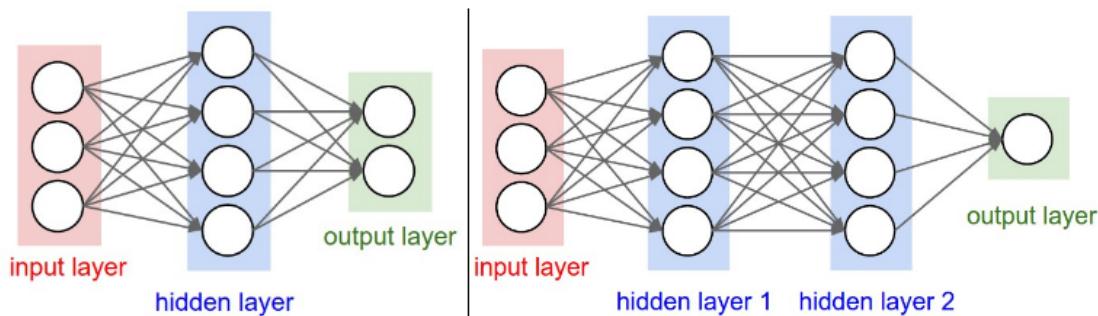
⁹ Source: <http://playground.tensorflow.org/>

Hình 2.9 Neural network với 1 hidden layer - 3 neurons¹⁰

2.3.2 Neural Network

Neural Network được mô hình như một tập hợp các neuron được kết nối với nhau trong một đồ thị không vòng. Nói cách khác, output của một neuron có thể là input của một neuron khác. Chu trình không được cho phép bởi vì nó có thể kéo theo vòng lặp vô hạn trong network. Mô hình Neural Network thường được tổ chức thành các lớp các neuron. Loại phổ biến nhất là các lớp được kết nối đầy đủ (fully-connected layer) trong đó các neuron giữa 2 lớp liền kề được liên kết đầy đủ từng đối một với nhau, nhưng các neuron trong một lớp thì không có liên kết với nhau. Hình 2.10 minh họa 2 ví dụ về cấu trúc hình học của Neural Network sử dụng fully-connected layers.

¹⁰ Source: <http://playground.tensorflow.org/>



Left: A 2-layer Neural Network (one hidden layer of 4 neurons (or units) and one output layer with 2 neurons), and three inputs.
Right: A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer. Notice that in both cases there are connections (synapses) between neurons across layers, but not within a layer.

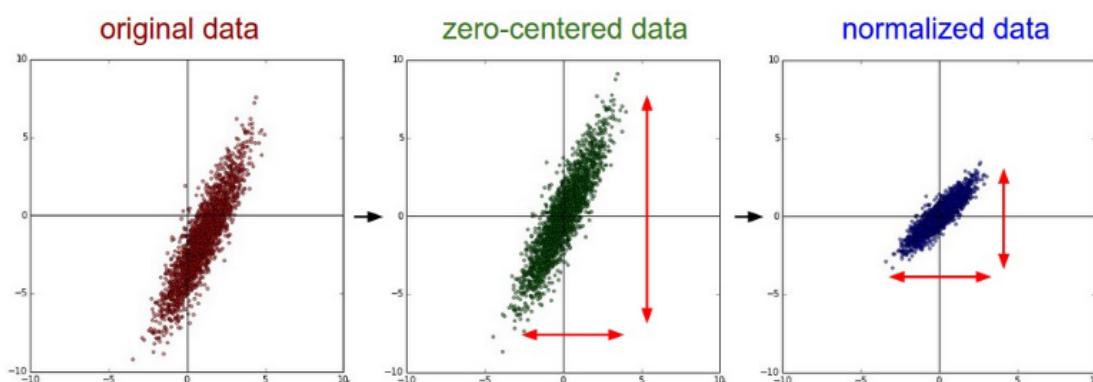
Hình 2.10 Neural Network với 1 hidden layer và Neural Network với 2 hidden layer¹¹

2.4 Các yếu tố ảnh hưởng đến việc huấn luyện

2.4.1 Tiền xử lý dữ liệu

Có 3 dạng tiền xử lý dữ liệu của một ma trận dữ liệu X, giả sử X có kích thước $[N \times D]$ (N là số dữ liệu, D là kích thước của nó).

Mean Subtraction là dạng phổ biến nhất của tiền xử lý dữ liệu. Cách hiện thực của dạng này là với mỗi feature của dữ liệu, ta lấy giá trị này trừ đi giá trị trung bình, như vậy ta sẽ tập trung dữ liệu qua các gốc tọa độ.



Common data preprocessing pipeline. Left: Original toy, 2-dimensional input data. Middle: The data is zero-centered by subtracting the mean in each dimension. The data cloud is now centered around the origin. Right: Each dimension is additionally scaled by its standard deviation. The red lines indicate the extent of the data - they are of unequal length in the middle, but of equal length on the right.

Hình 2.11 Các dạng tiền xử lý dữ liệu phổ biến¹²

¹¹ Source: <http://cs231n.github.io/neural-networks-1/>

¹² Source: <http://cs231n.github.io/neural-networks-2/>

Chuẩn hóa (Normalization) là chuẩn hóa kích thước dữ liệu sao cho chúng có cùng tỷ lệ kích thước. Có hai cách phổ biến để thực hiện chuẩn hóa. Một cách là chia mỗi chiều cho độ lệch chuẩn, một khi nó đã được zero-centered. Một dạng chuẩn hóa khác của cách tiền xử lý này là chuẩn hóa sao cho mỗi chiều có giá trị nhỏ nhất và lớn nhất tương ứng là -1 và 1. Cách tiền xử lý dữ liệu này chỉ có ý nghĩa nếu như chúng ta biết được các input feature có các tỷ lệ (đơn vị) khác nhau, và lúc này Normalization cũng đóng một vai trò rất quan trọng trong thuật toán học. Trong trường hợp hình ảnh, đơn vị các pixel input là như nhau nên không quá cần thiết để thực hiện bước tiền xử lý này. PCA và whitening là một dạng tiền xử lý khác. Trong quá trình xử lý này, trước tiên dữ liệu được tập trung như được miêu tả ở trên, sau đó chúng ta có thể tính ma trận hiệp phương sai giúp chúng ta nhận biết tính tương quan về cấu trúc của dữ liệu. Sau khi thực hiện PCA, chúng ta giảm tập dữ liệu gốc kích thước [NxD] thành [Nx100], giữ 100 chiều của dữ liệu chứa phương sai lớn nhất. Chúng ta có thể đạt hiệu suất tốt khi huấn luyện phân loại tuyến tính hoặc neural network trong tập dữ liệu được qua PCA, tiết kiệm cả về thời gian lẫn dung lượng lưu trữ.

Phép biến đổi cuối cùng có thể gặp trong thực tế là whitening. Phép whitening lấy dữ liệu từ eigenbasis và chia số mỗi chiều cho eigenbasis để chuẩn hóa đơn vị. Chúng ta đã đề cập đến PCA/Whitening nhưng những phép biến đổi này không được dùng với CNN. Tuy nhiên, việc zero-center dữ liệu rất quan trọng, việc chuẩn hóa từng pixel cũng rất phổ biến và quan trọng.

Điểm quan trọng nữa khi nó về tiền xử lý dữ liệu là bất kỳ thông kê tiền xử lý nào (ví dụ như giá trị trung bình) phải được tính toán ở tập dữ liệu huấn luyện rồi mới được áp dụng vào tập dữ liệu validation/test.^[5]

2.4.2 Khởi tạo tham số

Chúng ta đã tìm hiểu cách làm thế nào để tiền xử lý dữ liệu. Trước khi đi vào tìm hiểu mô hình kiến trúc Convolutional Neural Network, chúng ta sẽ tìm hiểu các cách khởi tạo tham số cho mô hình.

Chúng ta không thể biết tất cả các giá trị cuối cùng của các tham số sẽ có trong mạng được huấn luyện, nhưng với cách chuẩn hóa dữ liệu thích hợp, ta có thể giả sử rằng xấp xỉ một nửa tham số sẽ có giá trị dương và một nửa có giá trị âm. Một ý tưởng nghe có vẻ hợp lý là khởi tạo các tham số bằng 0. Tuy nhiên cách khởi tạo này sẽ gây ra lỗi, vì khi đó mỗi neuron của network sẽ tính ra output giống nhau, nó cũng sẽ tính toán ra hệ số góc như nhau trong quá trình lan truyền ngược và cập nhập các tham số như nhau. Nói cách khác, sẽ không có sự bất đối xứng giữa các neuron nếu các tham số được khởi tạo với giá trị giống nhau.

Chúng ta vẫn muốn khởi tạo các tham số rất gần với 0, nhưng chắc chắn không phải chính xác bằng 0 vì như vậy sẽ không có sự bất đối xứng như đã nói ở trên. Giải pháp là

khởi tạo tham số là các số ngẫu nhiên rất nhỏ để phá vỡ sự bất đối xứng. Cách hiện thực thông thường là mỗi vector tham số của neuron được khởi tạo là một vector ngẫu nhiên được sinh ra dựa trên phân phối chuẩn nhiều chiều. Vector ngẫu nhiên cũng có thể được sinh ra dựa trên phân phối đều liên tục, nhưng trong thực tế sẽ có một chút ảnh hưởng đến hiệu suất.

Một vấn đề với cách làm trên là phân bố xác suất của output từ các neuron được khởi tạo ngẫu nhiên có phương sai càng lớn theo số lượng input. Chúng ta có thể chuẩn hóa phương sai của mỗi output của các neuron với 1 bằng cách chia vector được sinh ra ngẫu nhiên cho căn bậc 2 của số lượng input. Điều này đảm bảo tất cả các neuron trong mạng có cùng phân bố output một cách xấp xỉ và cải thiện tốc độ hội tụ. Trong thư viện numpy của ngôn ngữ Python, vector w là vector ngẫu nhiên sẽ được khởi tạo $w = np.random.randn(n) / sqrt(n)$ với $np.random.randn(n)$ vector ngẫu nhiên 1 chiều với n tham số, n là số tham số.

Trong một bài báo cáo khoa học gần đây về chủ đề khởi tạo tham số, *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification* của He và các cộng sự, đề xuất một cách khởi tạo đặc thù cho ReLU neuron, kết luận rằng phương sai của các neuron trong mạng nên là $2.0/n$. Khi đó, theo cách viết của thư viện Numpy, vector ngẫu nhiên được khởi tạo là $w = np.random.randn(n) * sqrt(2.0/n)$.^[5]

2.4.3 Gradient Descent

Gradient descent là một trong những thuật toán phổ biến nhất để thực hiện tối ưu hóa và là cách thông thường nhất cho đến nay để tối ưu hóa các Neural Network. Đồng thời, mỗi thư viện Deep Learning hiện đại đều có hiện thực các thuật toán khác nhau để tối ưu hóa gradient descent (ví dụ: tài liệu của lasagne, caffe và keras). Tuy nhiên, các thuật toán này thường được sử dụng như các trình tối ưu hóa black-box, vì những lời giải thích thực tế về điểm mạnh và điểm yếu của chúng khó có thể đi qua.

Gradient descent là một cách để giảm thiểu một hàm mục tiêu được tham số bởi các tham số của mô hình bằng cách cập nhật các tham số theo chiều ngược lại của gradient hàm mục tiêu đến các tham số. Learning rate xác định kích thước của các bước mà chúng ta thực hiện để đạt được mức tối thiểu (cục bộ). Nói cách khác, chúng ta đi theo hướng độ dốc của bề mặt được tạo ra bởi hàm mục tiêu xuống dốc cho đến khi chúng ta đến được một thung lũng.

2.4.3.1 Các biến thể gradient descent

Có ba biến thể của gradient descent, dựa vào khác nhau về số lượng dữ liệu chúng ta sử dụng để tính gradient của hàm mục tiêu. Tùy thuộc vào lượng dữ liệu, chúng ta thực hiện một sự cân bằng giữa độ chính xác của cập nhật tham số và thời gian cần để thực hiện cập nhật.

2.4.3.1.1 Vanilla gradient descent

Batch (hoặc full-batch, hoặc classic) gradient descent. Bây giờ chúng ta có thể tính gradient của hàm loss, thủ tục lặp đi lặp lại hiện thực gradient và sau đó thực hiện một cập nhật tham số được gọi là Gradient Descent. Phiên bản vanilla của nó trông như sau:

```
# Vanilla Gradient Descent

while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += - step_size * weights_grad # perform parameter update
```

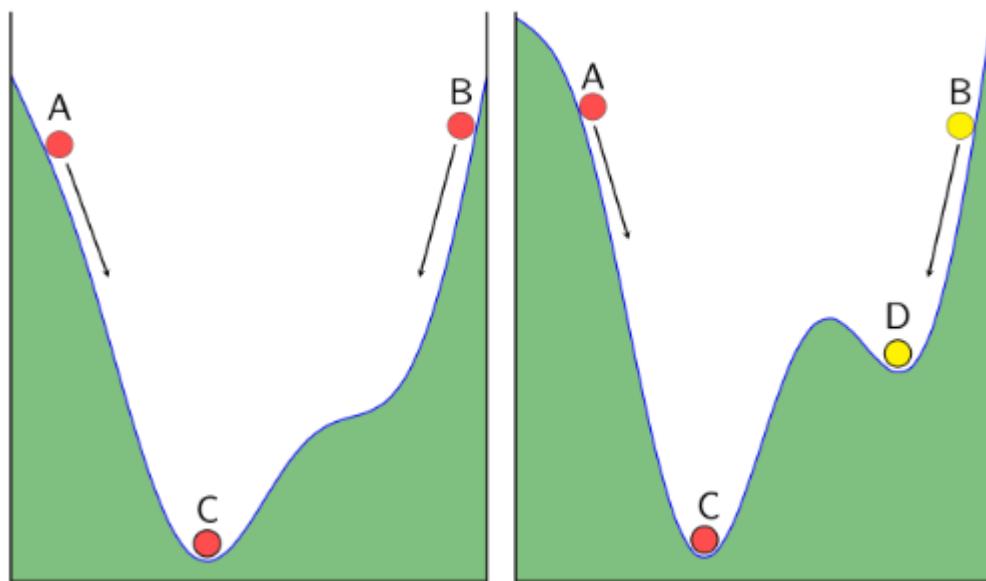
Hình 2.12 Vanilla gradient descent trên ngôn ngữ python¹³

Vòng lặp đơn giản này là cốt lõi của tất cả các thư viện Neural Network. Có những cách khác thực hiện tối ưu hóa (ví dụ như LBFGS), nhưng Gradient Descent hiện là phương pháp phổ biến nhất và được thiết lập để tối ưu hóa các hàm loss Neural Network.

Khi chúng ta cần phải tính toán gradient cho toàn bộ tập dữ liệu để thực hiện chỉ một lần cập nhật, batch gradient descent có thể rất chậm và khó thực hiện đối với các bộ dữ liệu không vừa trong bộ nhớ. Batch gradient descent cũng không cho phép chúng ta cập nhật trực tuyến, tức là với các mẫu mới luôn luôn được thêm vào

Với số epoch được định nghĩa trước, chúng ta đầu tiên tính gradient vector weight_grad của hàm loss cho toàn bộ tập dữ liệu. Sau đó, chúng ta cập nhật tham số của chúng ta theo hướng của gradient vector với learning rate dùng xác định độ lớn của bản cập nhật chúng ta thực hiện. Batch gradient descent được bảo đảm hội tụ đến mức tối thiểu toàn cầu (global minimum) cho bề mặt lồi và tối thiểu ở địa phương (local minimum) đối với bề mặt không lồi.

¹³ Source: <http://cs231n.github.io/optimization-1/>



Hình 2.13 Gradient descent trên mặt cắt của bìe mặt lồi và gradient trên mặt cắt của mặt không lồi¹⁴

2.4.3.1.2 Mini-batch gradient descent

Mini-batch gradient descent. Trong các ứng dụng quy mô lớn (như ILSVRC challenge), dữ liệu huấn luyện có thể lên đến hàng triệu mẫu. Do đó, nó có vẻ lãng phí để tính toán các toàn bộ hàm loss trong toàn bộ tập huấn luyện để thực hiện chỉ một cập nhật các thông số đơn. Cách tiếp cận rất phổ biến để giải quyết thách thức này là tính toán gradient trên các batches của dữ liệu huấn luyện. Ví dụ: trong ConvNets hiện đại, một batch điển hình chứa 256 mẫu từ toàn bộ bộ đào tạo là 1,2 triệu. Sau đó, batch này được sử dụng để thực hiện một cập nhật thông số:

```
# Vanilla Minibatch Gradient Descent

while True:
    data_batch = sample_training_data(data, 256) # sample 256 examples
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
    weights += - step_size * weights_grad # perform parameter update
```

Hình 2.14 Vanilla Minibatch Gradient Descent¹⁵

Lý do nó hoạt động tốt là các mẫu trong dữ liệu huấn luyện có tương quan với nhau. Để xem điều này, hãy xem xét trường hợp cụ thể, trong đó tất cả 1,2 triệu hình ảnh trong ILSVRC thực sự được tạo thành từ bản sao chính xác của 1000 hình ảnh độc nhất ban đầu (một cho mỗi lớp, hoặc nói cách khác là 1200 bản sao giống nhau của mỗi hình ảnh). Rõ ràng rằng gradient chúng ta sẽ tính cho tất cả 1200 bản sao giống hệt nhau, và khi chúng ta

¹⁴ Source: <http://machinelearningcoban.com/2017/01/16/gradientdescent2/>

¹⁵ Source: <http://cs231n.github.io/optimization-1/>

trung bình dữ liệu loss trên tất cả 1,2 triệu hình ảnh chúng ta sẽ có được thông số loss tương tự như thế chúng ta chỉ đánh giá trên một tập con nhỏ 1000. Trên thực tế, tập dữ liệu sẽ không chứa các hình ảnh trùng lặp, gradient từ một mini-batch sẽ trả về một kết quả tốt xấp xỉ so với full-batch. Do đó, sự hội tụ nhanh hơn có thể đạt được trong thực tế bằng cách sử dụng mini-batch gradient để thực hiện các cập nhật thông số thường xuyên hơn. Mini-batch gradient descent thỉnh thoảng có thể với một vài bài toán chạy nhanh hơn stochastic gradient descent.

Mini-batch gradient descent cuối cùng lấy tốt nhất của cả hai và thực hiện một bản cập nhật cho mỗi mini-batch của n training examples:

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

Với cách này, nó a) làm giảm sự biến thiên giữa các lần cập nhật tham số, có thể dẫn đến hội tụ ổn định hơn; và b) có thể tận dụng các sự tối ưu hóa ma trận tối ưu mà phổ biến đối với các thư viện deep learning hiện đại để thực hiện tính toán gradient qua một mini-batch rất hiệu quả.

Các kích thước mini-batch thông thường khoảng từ 50 đến 256, nhưng có thể thay đổi cho các ứng dụng khác nhau. Mini-batch gradient descent thường là thuật toán được lựa chọn khi huấn luyện một neural network và thuật ngữ SGD thường được sử dụng khi dùng các mini-batch. Chú ý: Trong sự điều chỉnh SGD trong phần còn lại của bài viết này, chúng ta bỏ qua các tham số $x^{(i:i+n)}, y^{(i:i+n)}$ cho đơn giản.

2.4.3.1.3 Stochastic gradient descent

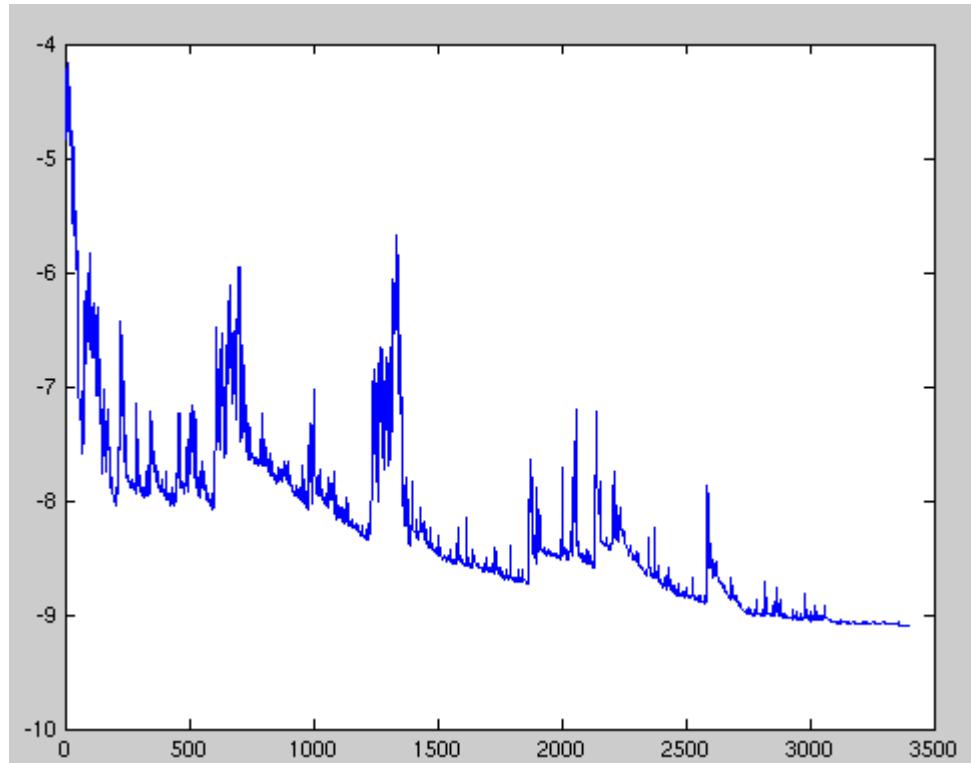
Trường hợp đặc biệt của mini-batch gradient descent này là một thiết lập mà mini-batch chỉ chứa một mẫu duy nhất. Quá trình này được gọi là Stochastic Gradient Descent (SGD) (hoặc đôi khi on-line gradient descent). Stochastic gradient descent là một sự thay thế cho batch (hoặc classic) gradient descent và có khả năng mở rộng cho các bộ dữ liệu lớn. Do nó sẽ chỉ thử một mẫu huấn luyện tại một thời điểm. Bằng cách này, chúng ta có thể chạy tiến trình trong gradient descent mà không cần phải quét tất cả các mẫu huấn luyện. Stochastic gradient sẽ không hội tụ ở global minimum và thay vào đó sẽ ngẫu nhiên vòng quanh khu vực đó, nhưng thường mang lại kết quả gần đủ. Stochastic gradient xuôi thường sẽ mất 1-10 lần đi qua bộ dữ liệu của bạn để đạt được cực tiểu toàn cục.

Stochastic gradient descent (SGD) tương phản thực hiện cập nhật thông số cho mỗi mẫu huấn luyện $x(i)$ và nhãn $y(i)$:

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i)}; y^{(i)})$$

Batch gradient descent thực hiện tính toán dư thừa cho các tập dữ liệu lớn, vì nó tính lại gradients cho các mẫu tương tự trước khi cập nhật từng tham số. SGD không có sự thừa này bằng cách thực hiện một cập nhật tại một thời điểm. Do đó thường nhanh hơn nhiều và cũng có thể được sử dụng để học trực tuyến.

SGD thực hiện các cập nhật thường xuyên với độ biến thiên cao khiến cho hàm mục tiêu dao động mạnh mẽ như trong hình:



Hình 2.15 Biến động SGD¹⁶

Trong khi batch gradient descent hội tụ tại cực tiểu của thung lũng nơi mà tham số được khởi tạo tại, sự biến động của SGD, một mặt cho phép nó nhảy tới một điểm cực tiểu mới và có tiềm năng tốt hơn, mặt khác khi đã đạt được điểm cực tiểu mong muốn nó vẫn có thể sẽ đi qua.

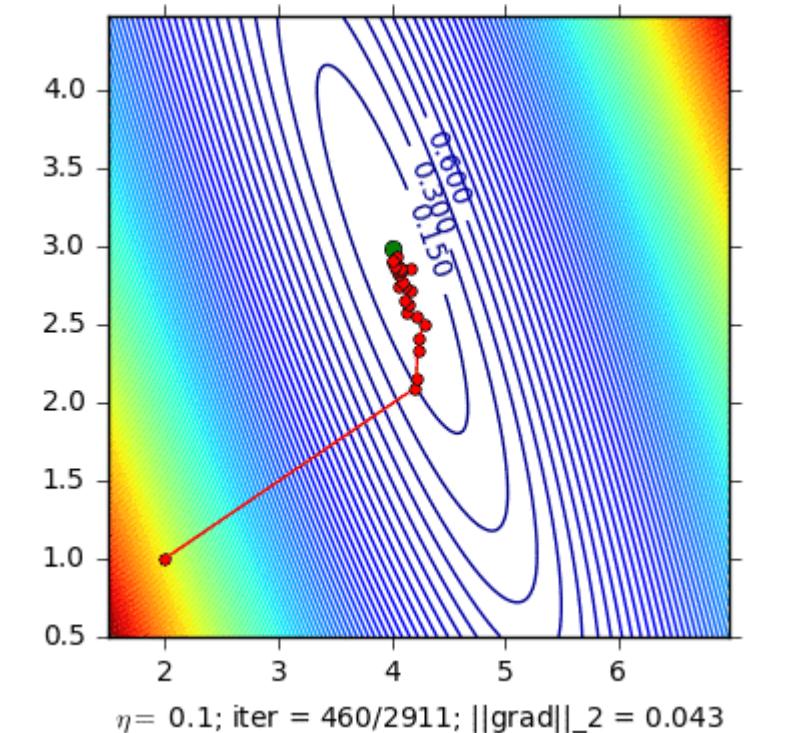
Tuy nhiên, nó đã chỉ ra rằng khi chúng ta từ từ giảm learning rate, SGD cho thấy hành vi hội tụ tương tự như batch gradient descent, hầu như chắc chắn hội tụ đến một local hoặc global minimum cho bề mặt lồi hoặc không lồi tương ứng.^{[3][8]}

Đoạn code của nó chỉ đơn giản là thêm một vòng lặp qua tập huấn luyện và đánh giá gradient qua mỗi mẫu:

¹⁶ Source: <https://en.wikipedia.org/wiki/File:Stogra.png>

```

for i in range(nb_epochs):
    np.random.shuffle(data)
    for example in data:
        params_grad = evaluate_gradient(loss_function, example, p)
        params = params - learning_rate * params_grad
    
```

Hình 2.16 Hiện thực SGD bằng numpy¹⁷Hình 2.17 Mô phỏng hàm loss SGD¹⁸

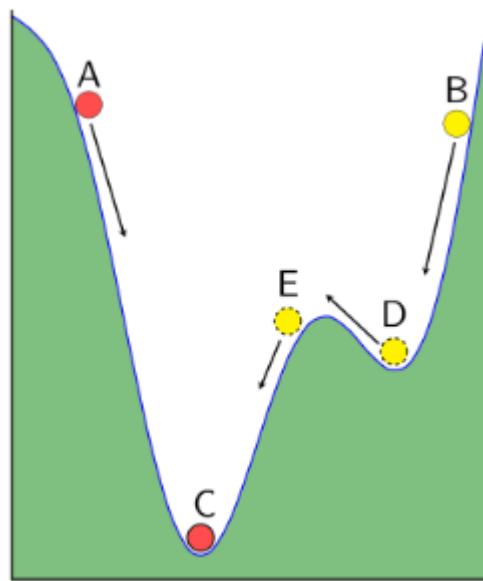
2.4.3.2 Các thuật toán tối ưu gradient descent

Sau đây, chúng ta sẽ phác thảo một số thuật toán được cộng đồng deep learning sử dụng rộng rãi

¹⁷ Source: <http://sebastianruder.com/optimizing-gradient-descent/>

¹⁸ Source: <http://machinelearningcoban.com/2017/01/16/gradientdescent2/>

2.4.3.2.1 Momentum



Hình 2.18 Góc nhìn vật lý SGD với momentum¹⁹

SGD gặp sự cố khi điều hướng các khe núi, nghĩa là các khu vực có bề mặt cong nhiều hơn ở một chiều hơn so với ở một vùng khác, phô biến quanh vùng tối ưu cục bộ. Trong những hoàn cảnh này, SGD dao động qua các sườn dốc của khe núi trong khi chỉ có sự tiến tới chậm chạp theo hướng tối ưu cục bộ như trong Hình 2.19.

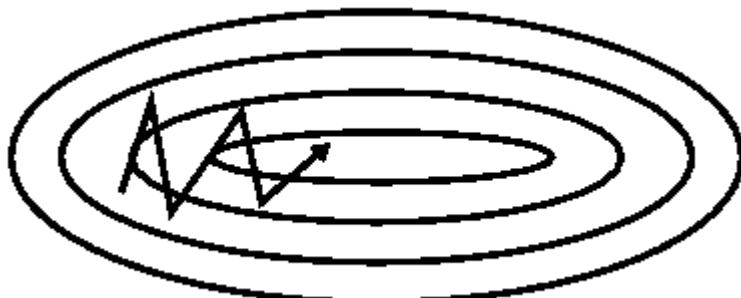


Hình 2.19 SGD không sử dụng Momentum²⁰

Momentum là một phương pháp giúp tăng SGD theo hướng có liên quan và làm giảm dao động có thể thấy trong Hình 2.20. Nó làm điều này bằng cách thêm một phần γ của vector cập nhật của bước nhảy trước đây vào vector cập nhật hiện tại:

¹⁹ Source: <http://machinelearningcoban.com/2017/01/16/gradientdescent2/#-momentum>

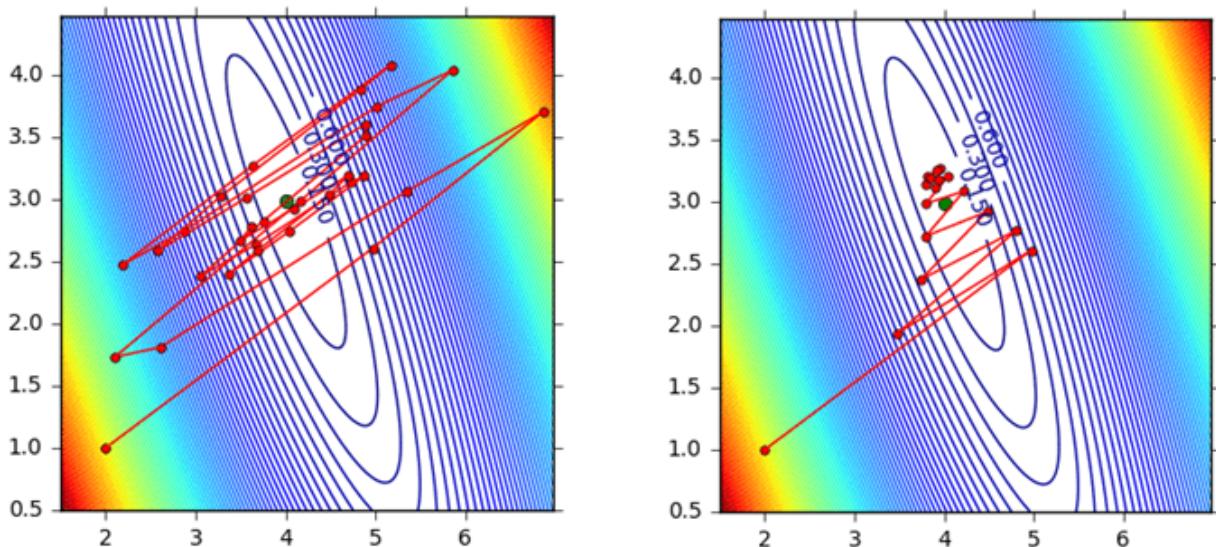
²⁰ Source: <http://sebastianruder.com/optimizing-gradient-descent/>

Hình 2.20 SGD có sử dụng Momentum²¹

Về cơ bản, khi sử dụng momentum, chúng ta đẩy một quả bóng xuống đồi. Bóng tích tụ momentum khi nó lăn xuống, trở nên nhanh hơn và nhanh hơn trên đường (cho đến khi đạt vận tốc cuối nếu có sức cản không khí, tức là $\gamma < 1$). Điều tương tự cũng xảy ra đối với cập nhật tham số: giai đoạn momentum tăng lên cho các chiều không gian mà gradient của nó có cùng hướng và giảm các cập nhật cho các chiều không gian có gradient của nó thay đổi hướng khác. Kết quả là chúng ta đạt được hội tụ nhanh hơn và giảm dao động.

2.4.3.2.2 Nesterov accelerated gradient

Tuy nhiên, một quả bóng lăn xuống một ngọn đồi, mù quáng theo độ dốc, là rất không đạt yêu cầu. Chúng ta muốn có một quả bóng thông minh hơn, một trái banh có xem xét về nơi nó sẽ đến để nó biết để làm chậm trước khi ngọn đồi dốc lên.

Hình 2.21 Minh họa thuật toán SGD với Momentum và NAG²²

Nesterov accelerated gradient (NAG) là một cách để cung cấp cho các giai đoạn momentum của chúng ta một loại dự đoán trước này. Chúng ta biết rằng chúng ta sẽ sử

²¹ Source: <http://sebastianruder.com/optimizing-gradient-descent/>

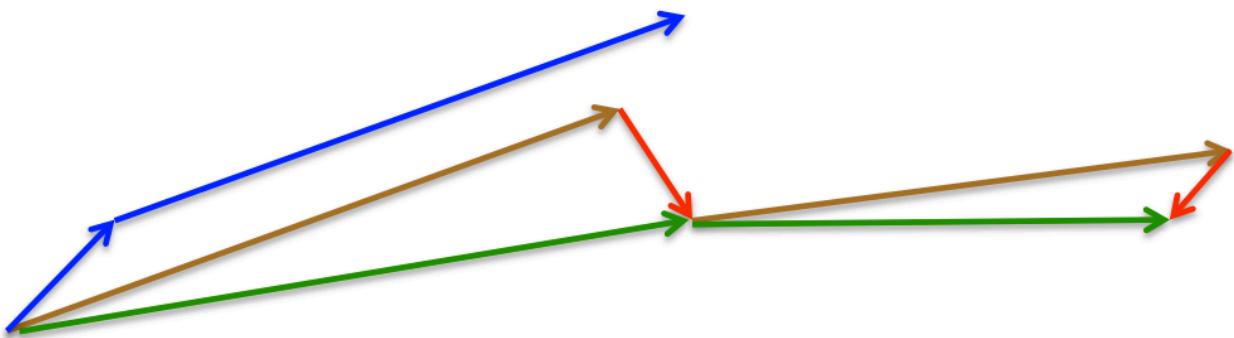
²² Source: <http://machinelearningcoban.com/2017/01/16/gradientdescent2/#-momentum>

dụng giai đoạn momentum γv_{t-1} để di chuyển các tham số θ . Tính toán $\theta - \gamma v_{t-1}$ làm cho chúng ta xấp xỉ vị trí tiếp theo của các tham số (gradient bị thiếu trong bản cập nhật đầy đủ). Vị trí tương lai gần đúng của các thông số:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta (\theta - \gamma v_{t-1})$$

$$\theta = \theta - v_t$$

Chúng ta thiết lập đà động từ γ với giá trị khoảng 0,9. Trong khi Momentum tính gradient hiện tại (vector nhỏ màu xanh ở Hình 2.21) và sau đó nhảy lớn theo hướng cập nhật tích lũy gradient (vector lớn màu xanh), NAG lần đầu tiên thực hiện bước nhảy lớn theo hướng gradient tích lũy trước đó (Màu nâu), đo gradient và sau đó chỉnh sửa (vector màu đỏ), kết quả là cập nhật NAG hoàn chỉnh (vector xanh). Bản cập nhật trước tiên này ngăn cản chúng ta đi quá nhanh và dẫn đến phản ứng nhanh hơn, làm tăng đáng kể hiệu suất của RNN trên một số nhiệm vụ.



Hình 2.22 Cập nhật Nesterov²³

Bây giờ chúng ta có thể điều chỉnh cập nhật với độ dốc của chức năng lỗi và tăng tốc độ SGD, chúng ta cũng muốn thích ứng cập nhật với từng tham số để thực hiện các cập nhật lớn hơn hoặc nhỏ hơn tùy thuộc vào tầm quan trọng của chúng.

Triệt tiêu learning rate

Trong huấn luyện deep network, thường rất hữu ích để triệt tiêu tỷ lệ học tập theo thời gian. Lời khuyên tốt cần ghi nhớ là với learning rate cao, hệ thống chứa quá nhiều động lực và vector tham số sẽ chạy vòng quanh, không thể lảng xuống sâu hơn, hẹp hơn của loss function. Biết thời điểm để phân rã learning rate cũng là một thủ thuật: phân rã chậm và chúng ta sẽ lảng phí tính toán này xung quanh với sự cải thiện rất ít trong một thời gian dài. Nhưng phân rã nó quá mạnh và hệ thống sẽ làm mát quá nhanh, không thể đạt được vị trí tốt nhất có thể. Có ba loại phổ biến để thực hiện phân rã learning rate:

²³ Source: [G. Hilton's lecture 6c](#)

Step decay (phân rã theo bước): Giảm thiểu learning rate theo một số yếu tố trong mỗi vài epochs. Các giá trị tiêu biểu có thể làm giảm learning rate xuống bằng nửa mỗi 5 epochs, hoặc bằng 0,1 mỗi 20 epochs. Những con số này phụ thuộc rất nhiều vào loại vấn đề và mô hình. Một heuristic bạn có thể thấy trong thực tế là để xem lỗi xác nhận trong khi đào tạo có learning rate cố định và giảm learning rate theo một hằng số (ví dụ: 0.5) bắt cứ khi nào lỗi xác nhận ngừng cải thiện.

Exponential decay (phân rã mũ): Có dạng toán học $\alpha = \alpha_0 e^{-kt}$, trong đó α_0, k là hyperparameters và t là số lặp (nhưng bạn cũng có thể sử dụng các đơn vị epochs).

1/t decay (phân rã 1/t): Có dạng toán $\alpha = \frac{\alpha_0}{1+kt}$ trong đó α_0, k là hyperparameters và t là số lặp.

2.4.3.2.3 Adagrad

Adagrad là một phương pháp tỷ lệ học tập thích nghi ban đầu được đề xuất bởi Duchi và các cộng sự.

Lưu ý rằng bộ nhớ cache biến có kích thước bằng với kích thước của gradient, và theo dõi tổng số các tham số của các bình phương gradient. Điều này sau đó được sử dụng để chuẩn hóa bước cập nhật thông số theo từng phần tử. Lưu ý rằng tham số nhận được gradient cao sẽ giảm learning rate, trong khi tham số nhận được cập nhật nhỏ hoặc không thường xuyên sẽ tăng learning rate. Toán tử căn bậc hai trở nên rất quan trọng mà nếu không có nó, thuật toán sẽ thực hiện tệ hơn nhiều. Thuật ngữ làm mịn eps (thường đặt ở một nơi nào đó trong khoảng từ 1e-4 đến 1e-8) tránh trường hợp chia cho 0. Nhược điểm của Adagrad là trong trường hợp Deep Learning, learning rate thường cải thiện quá mạnh và ngừng quá sớm.

```
# Assume the gradient dx and parameter vector x
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

Hình 2.23 Thuật toán Adagrad²⁴

2.4.3.2.3. RMSprop

RMSprop là một phương pháp hiệu quả. Cập nhật RMSProp điều chỉnh phương pháp Adagrad theo một cách rất đơn giản để giảm learning rate, đơn điệu. Cụ thể, nó sử dụng một trung bình của gradients bình phương, cho:

²⁴ Source: <http://cs231n.github.io/neural-networks-3/>

```
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

Hình 2.24 Thuật toán RMSprop²⁵

Ở đây, `decay_rate` là một hyperparameter và các giá trị điển hình là [0.9, 0.99, 0.999]. Lưu ý rằng bản cập nhật `x +=` giống với Adagrad, nhưng biến bộ nhớ `cache` "rò rỉ" (leaky). Do đó, RMSProp vẫn điều chỉnh learning rate của mỗi tham số dựa trên độ lớn của các gradient của nó, có hiệu quả như nhau, nhưng khác với Adagrad, các cập nhật không bị đơn điệu.

2.4.3.2.4 Adam

Adam là một cập nhật được đề xuất gần đây, khá giống với RMSProp với Momentum. Bản cập nhật (đơn giản) sẽ như sau:

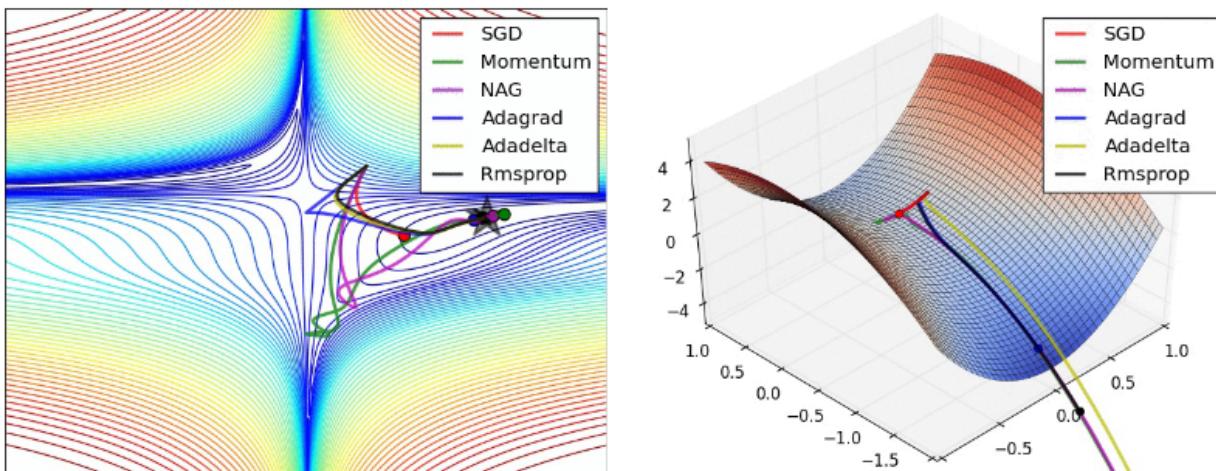
```
m = beta1*m + (1-beta1)*dx
v = beta2*v + (1-beta2)*(dx**2)
x += - learning_rate * m / (np.sqrt(v) + eps)
```

Hình 2.25 Thuật toán Adam(gần đúng)²⁶

Lưu ý rằng sự cập nhật trông chính xác như cập nhật RMSProp, ngoại trừ phiên bản "mịn" của gradient `m` được sử dụng thay vì vector thô `dx`. Các giá trị được đề nghị trong các tài liệu khoa học là `eps = 1e-8`, `beta1 = 0.9`, `beta2 = 0.999`. Trong thực tế Adam hiện đang được đề nghị như là thuật toán mặc định để sử dụng, và thường hoạt động tốt hơn một chút so với RMSProp. Tuy nhiên, nó thường cũng đáng để thử SGD + Nesterov Momentum như là một sự thay thế.^[6]

²⁵ Source: <http://cs231n.github.io/neural-networks-3/>

²⁶ Source: <http://cs231n.github.io/neural-networks-3/>



Hình 2.26 So sánh sự tối ưu SGD trên các thuật toán tối ưu²⁷

2.4.4 Regularization

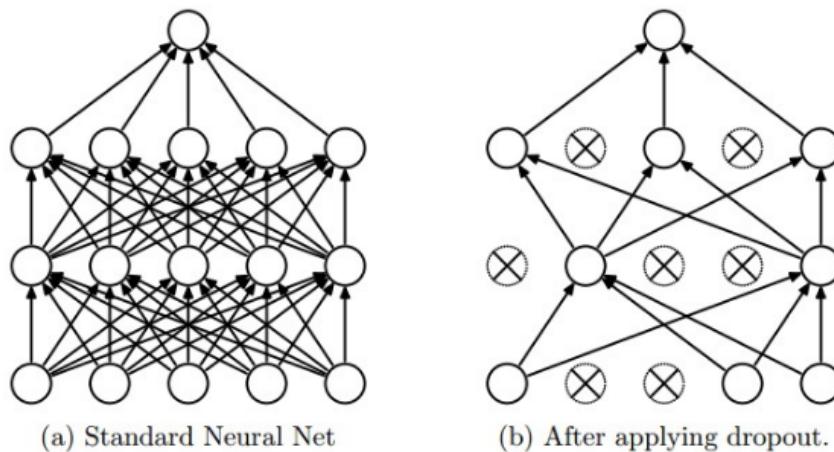
Trong mục này giới thiệu về Regularization, một số cách để tránh overfitting trong Neural Network.

L2 Regularization là dạng thông thường nhất của Regularization. Ta thêm vào hàm giả thiết một hàm của w để tránh hiện tượng overfitting. Ở đây, với mỗi tham số w của mạng, ta thêm vào một tham số $\frac{1}{2}\lambda w^2$, với λ là độ lớn Regularization. Ta thường thêm hệ số $\frac{1}{2}$ để đạo hàm theo w sẽ là kết quả là λw , tối giản hơn so với $2\lambda w$.

L1 Regularization một là một dạng phổ biến khác của Regularization. Tương tự với L2 Regularization nhưng với mỗi tham số w của mạng, thay vì thêm vào tham số $\frac{1}{2}\lambda w^2$ thì ta thêm vào giá trị λw . Chúng ta cũng có thể kết hợp giữa L1 Regularization và L2 Regularization.

Dropout là một phương pháp Regularization hiệu quả, đơn giản được giới thiệu gần đây. Trong quá trình huấn luyện, dropout được hiện thực bằng cách giữ các neuron hoạt động với một xác suất p .

²⁷ Source: <http://cs231n.github.io/neural-networks-3/>



Hình 2.27 Neural Network trước và sau khi áp dụng Dropout²⁸

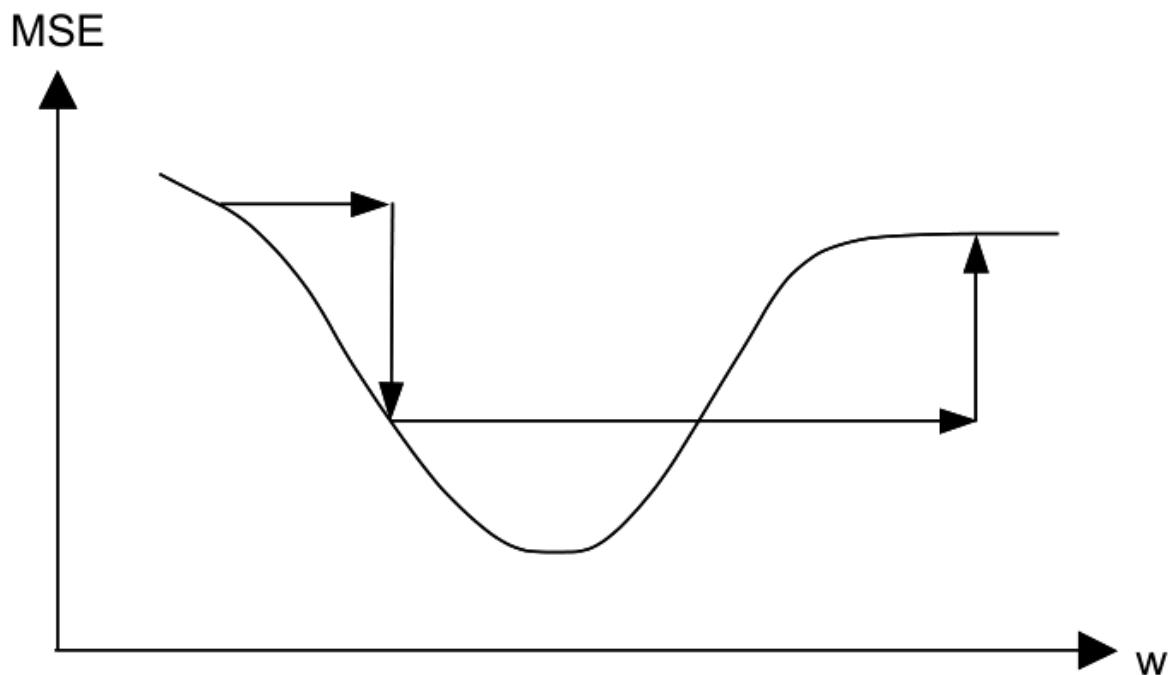
Trong thực tế, chúng ta thường sử dụng L2 Regularization để giảm thiểu overfitting. Chúng ta có thể kết hợp nó với phương pháp dropout. Giá trị $p = 0.5$ là giá trị mặc định được sử dụng nhiều với kết quả tốt, nhưng nó có thể thay đổi tùy theo tập validation.

2.4.5 Learning rate

Learning rate là giá trị định lượng Δw sẽ được cập nhật vào w lớn hay nhỏ, hay nói cách khác, learning rate sẽ ảnh hưởng trực tiếp đến sự thay đổi của w qua đó ảnh hưởng đến thời gian training và độ chính xác sau khi training. Learning rate là một con số không cố định và thay đổi phù hợp cho từng mô hình ANN và tập dữ liệu cụ thể. Learning rate quá lớn hay quá nhỏ sẽ gây ảnh hưởng đến độ chính xác và thời gian training.

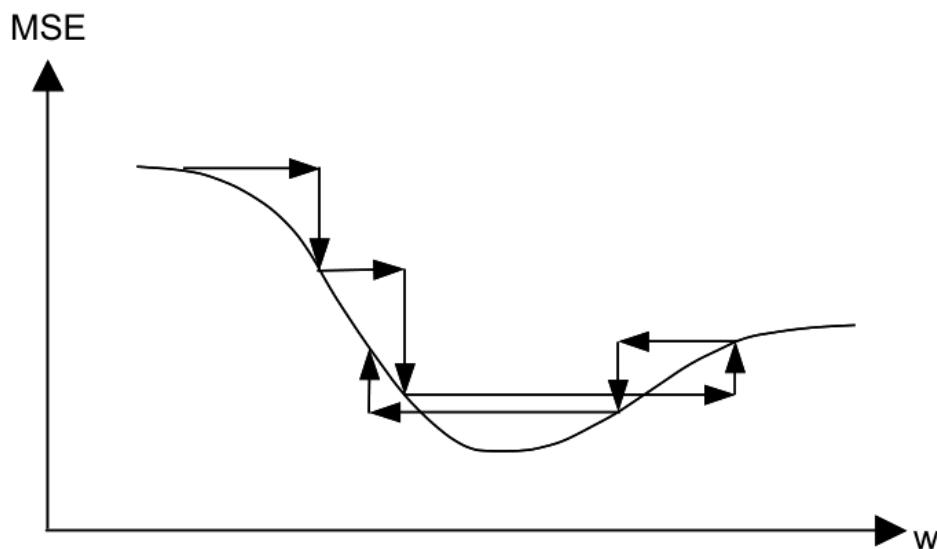
Với learning rate quá lớn, sẽ dẫn tới không thể hội tụ cho gradient descent, thậm chí dẫn tới phân kỳ (hình 2.28)

²⁸ Source: <http://cs231n.github.io/neural-networks-2/>



Hình 2.28 Learning rate quá cao²⁹

Với learning rate cao, vector tham số sẽ chạy vòng quanh, không thể lăng xuống sâu hơn (Hình 2.29).

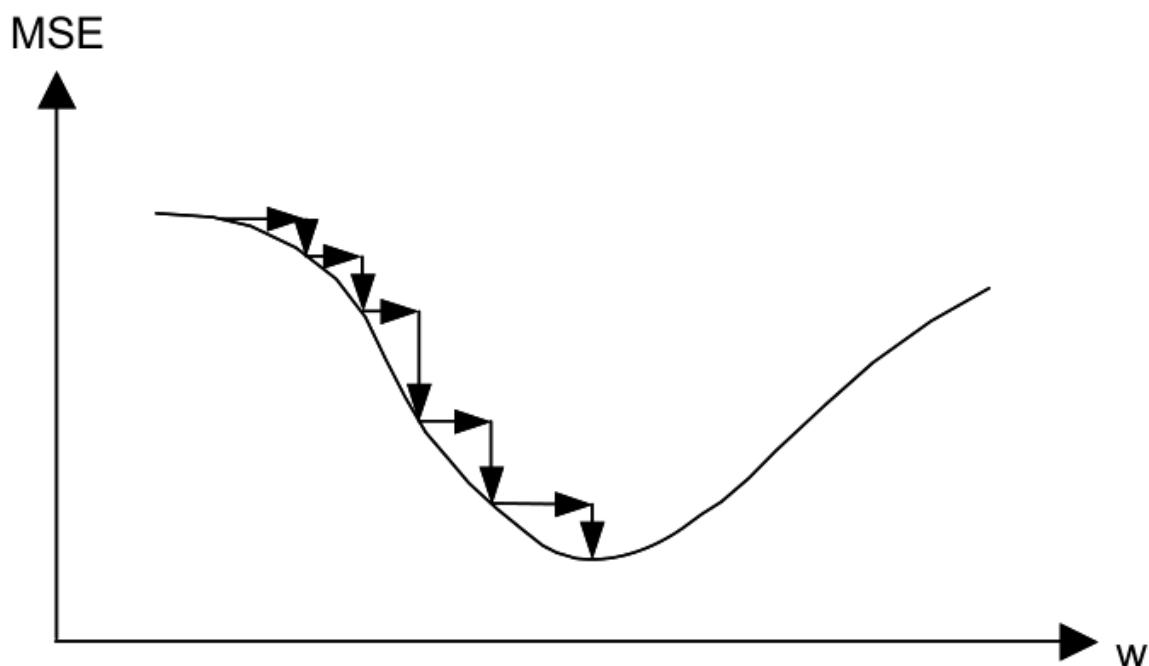


Hình 2.29 Learning rate cao³⁰

²⁹ ²⁹ Source: Sandhya Samarasinghe. Neural networks for applied sciences and engineering: From fundamentals to complex pattern recognition. Auerbach Publications, 2006.

³⁰ Source: Sandhya Samarasinghe. Neural networks for applied sciences and engineering: From fundamentals to complex pattern recognition. Auerbach Publi-

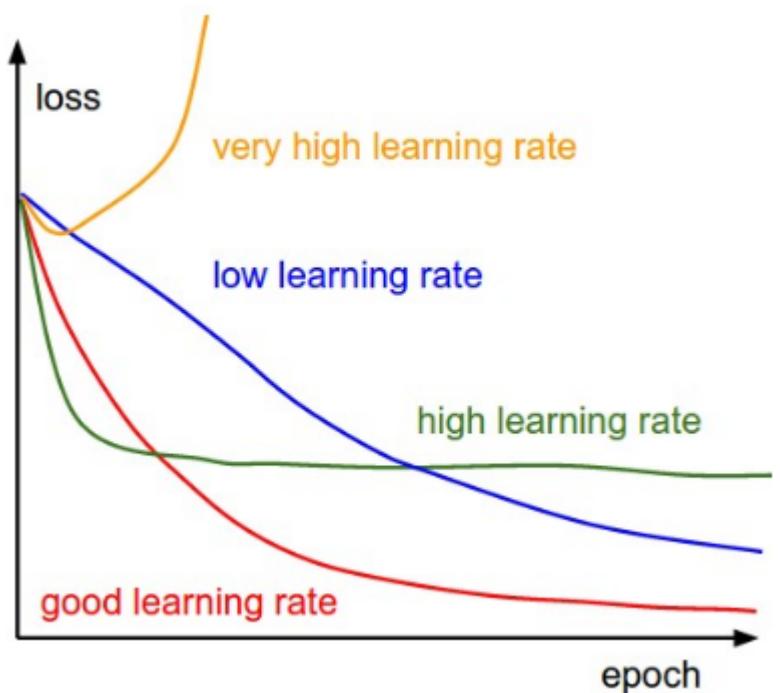
Với learning rate nhỏ, tốc độ hội tụ chậm hơn dẫn đến tốn thời gian tính toán (hình 2.30)



Hình 2.30 Learning rate quá nhỏ³¹

cations, 2006.

³¹ Source: Sandhya Samarasinghe. Neural networks for applied sciences and engineering: From fundamentals to complex pattern recognition. Auerbach Publications, 2006.



Hình 2.31 Đánh giá learning rate qua đồ thị hàm loss³²

2.5 Convolution Neural Network

2.5.1 Kiến trúc tổng quan

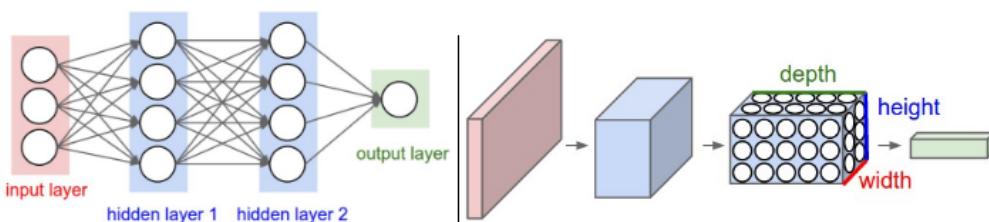
Như đã giới thiệu, Neural Network nhận các input (single vector), biến đổi nó qua các hidden layer. Mỗi hidden layer là tập hợp các neuron, mà mỗi neuron thì được kết nối đầy đủ với các neuron ở lớp trước. Các neuron trong một layer thì hoàn toàn độc lập và không có bất kỳ sự kết nối nào với nhau. Lớp cuối cùng gọi là output layer biểu diễn “class score”.

Nhược điểm của kiến trúc Neural Network thông thường là không làm việc tốt với hình ảnh lớn, đầy đủ. Ví dụ trong tập CIFAR-10, là tập mà các hình ảnh chỉ có kích thước là $32 \times 32 \times 3$ (chiều dài 32 pixel, chiều cao 32 pixel và có 3 kênh màu), vì vậy một neuron được kết nối đầy đủ trong lớp đầu tiên của Neural Network có thể có $32 \times 32 \times 3 = 3072$ tham số. Số lượng tham số này có vẻ như có thể quản lí được, nhưng rõ ràng cấu trúc kết nối đầy đủ này không mở rộng được với các hình ảnh lớn hơn. Ví dụ với hình ảnh có kích thước lớn hơn như $200 \times 200 \times 3$, số lượng tham số mà một neuron được kết nối đầy đủ có trong lớp đầu tiên là $200 \times 200 \times 3 = 120000$. Trong một lớp sẽ lại sẽ có nhiều neuron như vậy, nên số lượng tham số sẽ tăng lên rất lớn. Rõ ràng, sự kết nối đầy đủ này là lãng phí và số lượng lớn tham số sẽ dẫn đến overfitting.

³² Source: <http://cs231n.github.io/neural-networks-3/>

Không giống như mạng Neural thông thường, các layer của Convolutional Neural Networks có các neuron được sắp xếp 3 chiều: width, height, depth. Ví dụ với tập input là tập CIFAR-10, các khối sẽ có chiều là $32 \times 32 \times 3$. Các neuron trong một lớp sẽ chỉ được liên kết với một vùng nhỏ của các layer trước nó, thay vì kết nối với tất cả các neuron của layer trước như trong mạng neural thông thường. Hơn nữa, output của tập CIFAR-10 sẽ có chiều là $1 \times 1 \times 10$, bởi vì khi kết thúc kiến trúc Convolutional Neural Network, chúng ta sẽ giảm hình ảnh đầy đủ thành một vector của class score.

Một CNN được cấu tạo nên bởi các Layer. Mỗi layer biến đổi khối 3D input thành một khối 3D output với vài hàm khác nhau có hoặc không có tham số.



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Hình 2.32 Convolutional Neural Network³³

2.5.2 Các lớp cấu tạo nên Convolutional Neural Network

Có 3 kiểu layer chính dùng để xây dựng nên kiến trúc của CNN: Lớp chập (Convolutional Layer), lớp gộp (Pooling Layer) và lớp được kết nối đầy đủ (Fully-connected Layer -tương tự như mạng neural thông thường).

Chúng ta sẽ đi vào chi tiết của các layer trong phần sau, nhưng trước tiên chúng ta sẽ đi vào một ví dụ cụ thể, kiến trúc CNN tổng quan trong bài toán phân loại cho tập ảnh CIFAR-10, một CNN đơn giản sẽ có kiến trúc [INPUT – CONV – RELU – POOL – FC]. Cụ thể:

- **INPUT LAYER.** Đầu tiên là lớp Input. INPUT[$32 \times 32 \times 3$] sẽ giữ giá trị pixel thô của hình ảnh, trong trường hợp này là hình ảnh có chiều rộng 32, chiều cao 32 và ba kênh màu R, G, B.
- **CONV LAYER.** Lớp Convolutional tính toán output của các neuron được liên kết với một số vùng cục bộ của khối input. Kết quả có thể là [$32 \times 32 \times 12$] nếu ta sử dụng 12 filters.

³³ Source: <http://cs231n.github.io/convolutional-networks/>.

- **ReLU LAYER.** áp dụng hàm kích hoạt (ReLU) theo từng phần tử. Kích thước của khối không đổi. [32x32x12].
- **POOL LAYER.** thực hiện phép tính theo chiều rộng, chiều cao. Kết quả sẽ biến đổi khối trở thành [16x16x12].
- **FC LAYER.** Tương tự với kiến trúc mạng neuron thông thường, mỗi neuron trong lớp này sẽ được kết nối đầy đủ với tất cả các neuron ở lớp trước. Fully-connected layer sẽ tính toán class score, kết quả là một khối với kích thước [1x1x10], với mỗi trong 10 số tương ứng với 1 class score, là 10 loại trong tập CIFAR-10.

Qua ví dụ cụ thể trên, chúng ta thấy Convolutional Neural Network đã biến đổi hình ảnh ban đầu qua từng layer, từ các giá trị gốc là giá trị pixel tới giá trị class score cuối cùng. Mỗi layer có thể có hoặc không có tham số. Thông thường, CONV LAYER và FC LAYER thực hiện việc chuyển đổi với chức năng không chỉ là kích hoạt (activation) khối input, mà còn cả với các tham số (như weight hay bias của neuron). ReLU LAYER và POOL LAYER sẽ hiện thực một hàm cố định (ví dụ với ReLU layer là hàm kích hoạt $\max(0, x)$, với POOL LAYER là toán tử downsampling). Các tham số trong CONV/FC LAYER sẽ được huấn luyện cùng với phương pháp Gradient Descent để mà class score của Convolutional Neural Network tính ra nhất quán với các nhãn của mỗi hình trong tập huấn luyện.

Tiếp theo, chúng ta sẽ đi vào tìm hiểu chi tiết các Layer trong Convolutional Neural Network.

CONVOLUTIONAL LAYER (CONV Layer). Là khối cốt lõi của một Convolutional Neural Network, chịu phần lớn gánh nặng tính toán của mô hình. Nhiệm vụ chủ yếu của CONV Layer là thu thập, tìm kiếm các đặc điểm đặt trung của dữ liệu đầu vào. Trước tiên ta thảo luận về CONV Layer thực hiện các tính toán gì dưới góc nhìn trực giác, không thông qua sự tương tự với não bộ/ nơ ron. Các tham số của CONV Layer gồm tập các cấu trúc được gọi là filter. Filter có chiều dài, chiều rộng được xác định trước và chiều sâu bằng chiều sâu dữ liệu đầu vào. Ví dụ, một filter đặc thù tại lớp đầu tiên của CNN có thể có kích thước 5x5x3 (Ví dụ với 5 pixel chiều dài, rộng và bởi vì chiều sau hình ảnh là 3 color channel). Suốt giai đoạn sau, ta dùng phép trượt (slide hay convolve) mỗi filter qua chiều dài và chiều rộng của input volume và tích tích vô hướng của filter và input tại các vị trí tương ứng. Kết quả sau khi tính toán ta sẽ được một ma trận 2 chiều. Chúng ta sẽ kết hợp ma trận này với chiều sâu để sinh ra output volume.

1 <small>$\times 1$</small>	1 <small>$\times 0$</small>	1 <small>$\times 1$</small>	0	0
0 <small>$\times 0$</small>	1 <small>$\times 1$</small>	1 <small>$\times 0$</small>	1	0
0 <small>$\times 1$</small>	0 <small>$\times 0$</small>	1 <small>$\times 1$</small>	1	1
0	0	1	1	0
0	1	1	0	0

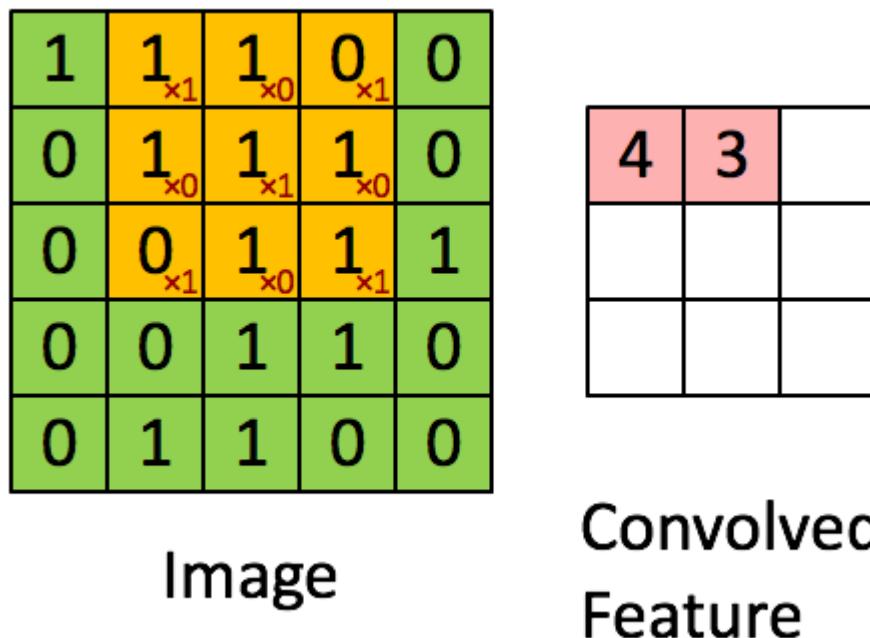
Image

4		

Convolved Feature

Hình 2.33 Filter và Convolved Feature trong Convolutional Layer³⁴

³⁴ Source: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution



Hình 2.34 Filter và Convolved Feature trong Convolutional Layer sau khi thực hiện phép trượt với $Stride = 1^{35}$

Tính liên kết cục bộ của Convolutional Layer: Khi làm việc với input nhiều chiều như hình ảnh, sẽ là không thực tế khi liên kết các neuron tới tất cả các neuron của volume trước. Thay vào đó, chúng ta sẽ chỉ liên kết mỗi neuron với chỉ một vùng cục bộ của volume trước. Độ rộng không gian của sự kết nối này là một hyperparameter gọi là receptive field (vùng tiếp thu) của neuron (tương đương với kích thước filter). Sự liên kết này chỉ cục bộ với chiều dài và chiều rộng, nhưng liên kết đầy đủ với chiều sâu của input volume.

Ví dụ 1. Giả sử input volume có size [32x32x3] (một ảnh RGB trong tập CIFAR-10). Nếu reception field (vùng tiếp thu - hay kích thước filter là) 5x5, thì mỗi neuron ở Conv Layer, sẽ có weight với một vùng [5x5x3] của input volume, tổng cộng là $5*5*3 = 75$ weights. (+1 tham số bias).

Ví dụ 2. Giả sử input volume có size [16x16x20]. Sử dụng filter có kích thước 3x3, mỗi neuron trong Conv Layer sẽ có tổng cộng $3*3*20 = 180$ liên kết tới input volume.

Qua 2 ví dụ ở trên, chúng ta đã giải thích được sự liên kết của mỗi neuron của CONV Layer với tập input. Tiếp theo chúng ta sẽ nói về số neuron của tập output và chúng được sắp xếp ra sao. Có 3 hyperparameter điều khiển kích thước của output volume: depth, stride và zero-padding.

³⁵ Source: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

- Hyperparameter đầu tiên là **Depth**. Nó tương ứng với số filter chúng ta muốn sử dụng.
- Hyperparameter thứ hai là **Stride**. Stride là độ lớn của mỗi bước dịch chuyển của filter. Khi stride bằng 1 thì chúng ta sẽ di chuyển filter 1 pixel cho mỗi bước dịch chuyển. Khi stride bằng 2 thì filter sẽ dịch chuyển 2 pixel mỗi lần, khi đó thì output volume sinh ra sẽ có kích thước nhỏ hơn.
- Hyperparameter thứ ba là **Zero-padding**. Đây là số giá trị 0 được thêm ngoài đường biên của ảnh để làm tăng kích thước của ảnh. Thông thường ta sử dụng hyperparameter này để điều chỉnh kích thước input volume sao cho width và height của input và output là như nhau.

Chúng ta có thể tính toán kích thước của output volume như là một hàm của input volume size (W), kích thước vùng tiếp thu (receptive field) của neuron (F), số stride được áp dụng (S) và số lượng zero-padding được dùng ở biên (P). Chúng ta có thể tính toán kích thước neuron output bằng công thức

$$\frac{W - F + 2P}{S} + 1$$

Ví dụ với input 7x7, filter 3x3, áp dụng stride bằng 1, và zero-padding bằng 0 ta sẽ có output là 5x5. Với stride bằng 2 output là 3x3.

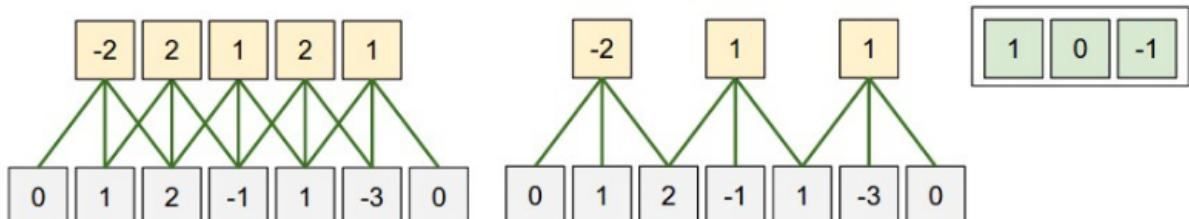


Illustration of spatial arrangement. In this example there is only one spatial dimension (x-axis), one neuron with a receptive field size of $F = 3$, the input size is $W = 7$, and there is zero padding of $P = 1$. **Left:** The neuron strided across the input in stride of $S = 1$, giving output of size $(7 - 3 + 2)/1+1 = 5$. **Right:** The neuron uses stride of $S = 2$, giving output of size $(7 - 3 + 2)/2+1 = 3$. Notice that stride $S = 3$ could not be used since it wouldn't fit neatly across the volume. In terms of the equation, this can be determined since $(7 - 3 + 2) = 4$ is not divisible by 3.

The neuron weights are in this example [1,0,-1] (shown on very right), and its bias is zero. These weights are shared across all yellow neurons (see parameter sharing below).

Hình 2.35 Minh họa tính toán kích thước output volume³⁶

Công dụng của zero-padding được giải thích như sau. Trong ví dụ trên, ta thấy rằng kích thước input và output đều bằng 5 khi ta chọn $F = 3$ và zero-padding bằng 1. Nếu không sử dụng zero-padding thì output volume sẽ chỉ có kích thước là 3. Thông thường, ta đặt

³⁶ Source: <http://cs231n.github.io/convolutional-networks/>

zero padding bằng $P = (F - 1)/2$ khi $S = 1$ để đảm bảo rằng input volume và output volume có kích thước như nhau.

Có sự ràng buộc khi sử dụng Stride. Chú ý rằng các hyperparameter quyết định kích thước output có ràng buộc lẫn nhau. Ví dụ, khi input size $W = 10$, không sử dụng zero-padding ($P = 0$), filter size $F = 3$, ta không thể chọn $S = 2$ bởi vì khi đó theo công thức kích thước neuron output ta sẽ có đáp số là 4.5, không phải là số nguyên. Suy ra không phải bộ hyperparameter nào cũng thích hợp để chọn.

Tổng kết. Trong Conv Layer, với input volume có kích thước $W_1 \times H_1 \times D_1$, yêu cầu 4 hyperparameter là số filter K và kích thước của nó là F , Stride S và zero-padding P .

Khi đó output volume sẽ có kích thước $W_2 \times H_2 \times D_2$ với

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

$$D_2 = K$$

Với parameter sharing, sẽ có $F \times F \times D_1$ weight trên mỗi filter, và tổng cộng $F \times F \times D_1 \times K$ weights và K bias

Hyperparameter thường được sử dụng là $F = 3$, $S = 1$, $P = 1$

POOLING LAYER. Thông thường, Pooling Layer sẽ được đặt một cách cố định ngay sau Conv Layer trong kiến trúc Convolutional Neural Network. Chức năng của nó là giảm chiều không gian biểu diễn dẫn đến sự giảm số parameter và sự tính toán, đồng thời cũng kiểm soát overfitting. Pooling Layer thực hiện độc lập trong mỗi depth slice của input và resize lại chiều không gian của nó, sử dụng phép toán max. Dạng phổ biến nhất là pooling layer với filter 2x2. Một cách tổng quan, pooling layer:

Nhận vào input volume có kích thước $W_1 \times H_1 \times D_1$

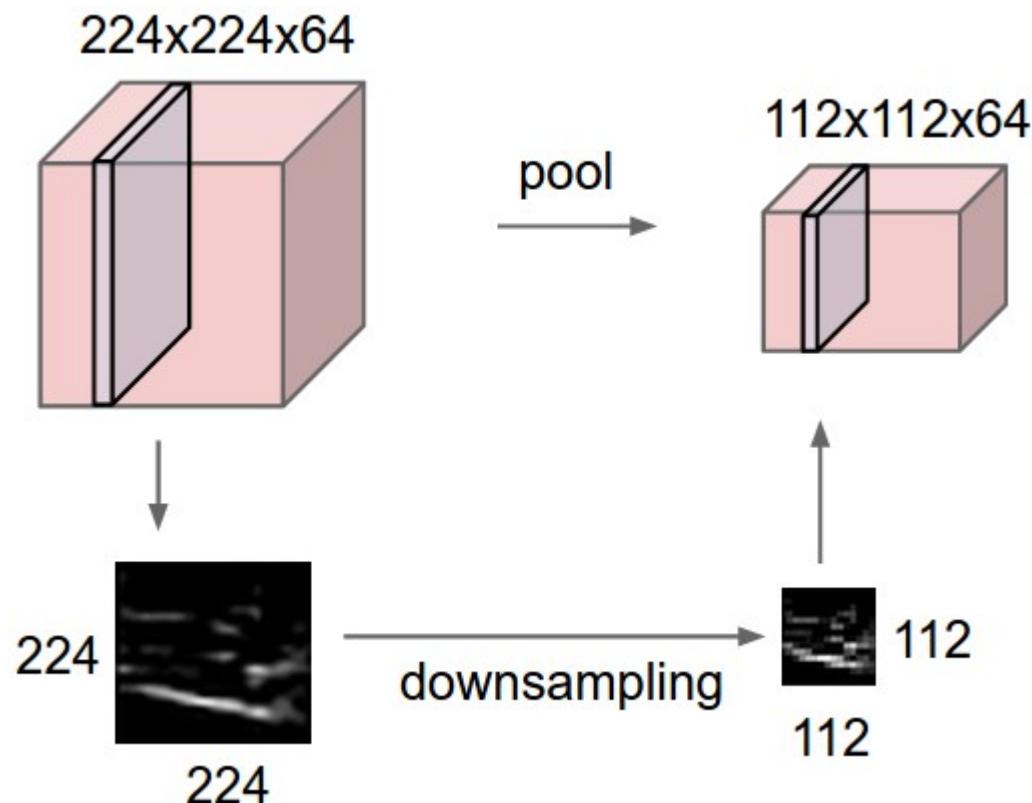
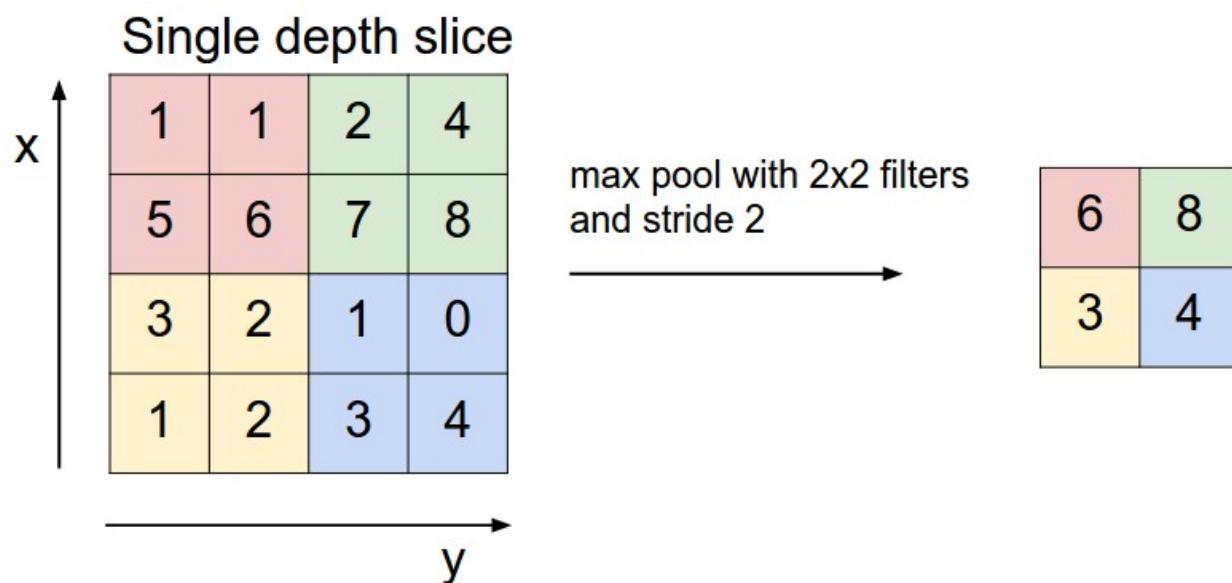
Cần các hyperparameter: chiều không gian F và stride S

Sinh ra output volume có kích thước $W_2 \times H_2 \times D_2$ với

$$W_2 = \frac{W_1 - F}{S} + 1$$

$$H_2 = \frac{H_1 - F}{S} + 1$$

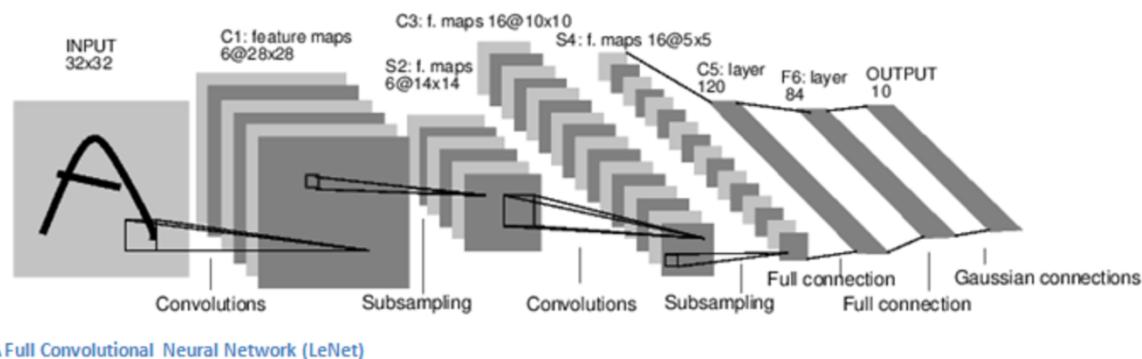
$$D_2 = D_1$$

Hình 2.36 Toán tử downsampling³⁷Hình 2.37 Minh họa Pooling Layer³⁸³⁷ Source: <http://cs231n.github.io/convolutional-networks/>³⁸ Source: <http://cs231n.github.io/convolutional-networks/>

Trong Pooling Layer, thông thường chúng ta không dùng zero-padding. Các hyperparameter thường được sử dụng trong pooling layer sẽ có $F = 3$, $S = 2$ hoặc $F = 2$, $S = 2$

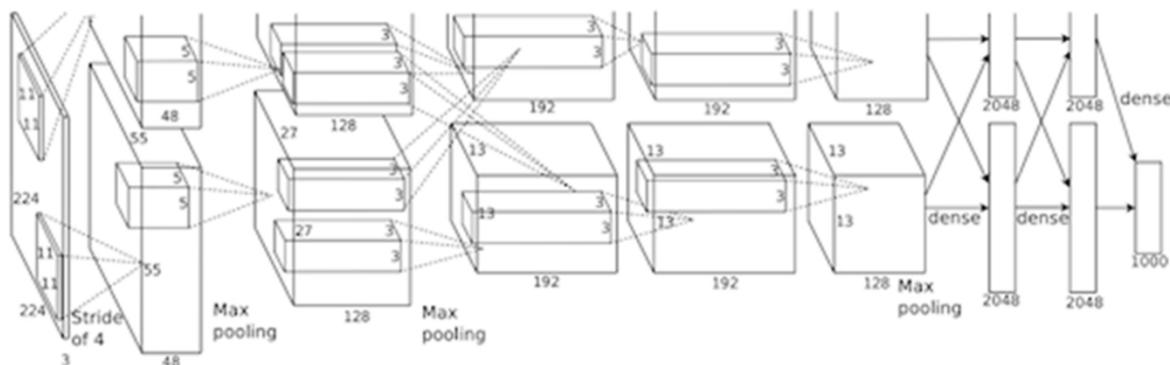
FC LAYER và OUTPUT Layer. Fully Connected Layer (FC) có cấu tạo và hoạt động hoàn toàn giống trong mô hình Neural Network thông thường. FC Layer có chức năng dựa vào những đặc điểm, đặc trưng của ảnh có được khi dữ liệu đi qua Convolutional Layer và Pooling Layer, rồi tính toán truyền qua Output Layer. Output Layer là lớp cuối cùng trong mô hình, tương tự như mô hình Neural Network thông thường. Chức năng của Output Layer là nhận dữ liệu xử lý từ FC Layer, tính toán và trả ra kết quả cuối cùng.

2.5.3 Một số mô hình kiến trúc Convolutional Neural Network



A Full Convolutional Neural Network (LeNet)

Hình 2.38 Kiến trúc LeNet³⁹

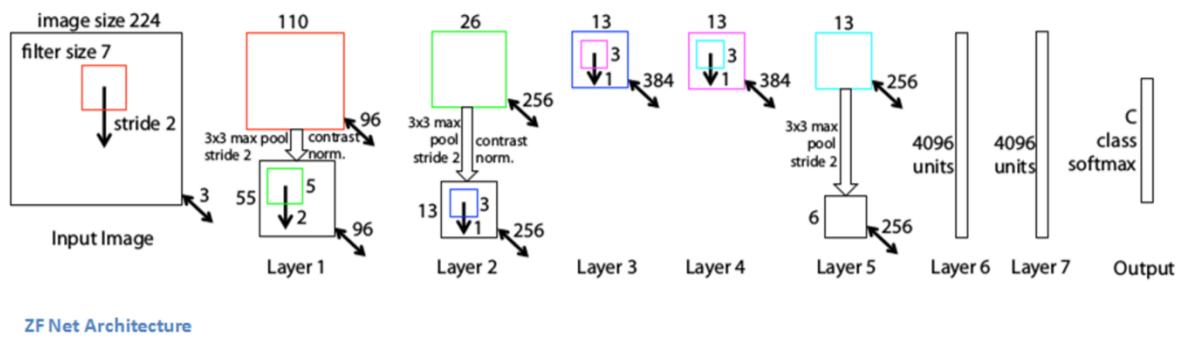


AlexNet architecture (May look weird because there are two different “streams”. This is because the training process was so computationally expensive that they had to split the training onto 2 GPUs)

Hình 2.39 Kiến trúc AlexNet⁴⁰

³⁹ Source: <https://adshpande3.github.io/adshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html>

⁴⁰ Source: <https://adshpande3.github.io/adshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html>



ZF Net Architecture

Hình 2.40 Kiến trúc ZF Net⁴¹

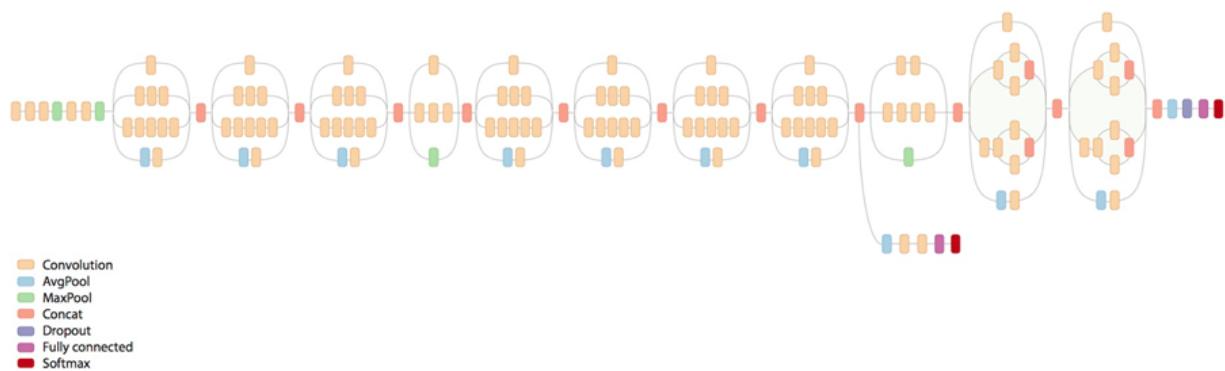
⁴¹ Source: <https://adshpande3.github.io/adshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html>

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

The 6 different architectures of VGG Net. Configuration D produced the best results

Hình 2.41 6 kiến trúc khác nhau của VGG Net⁴²

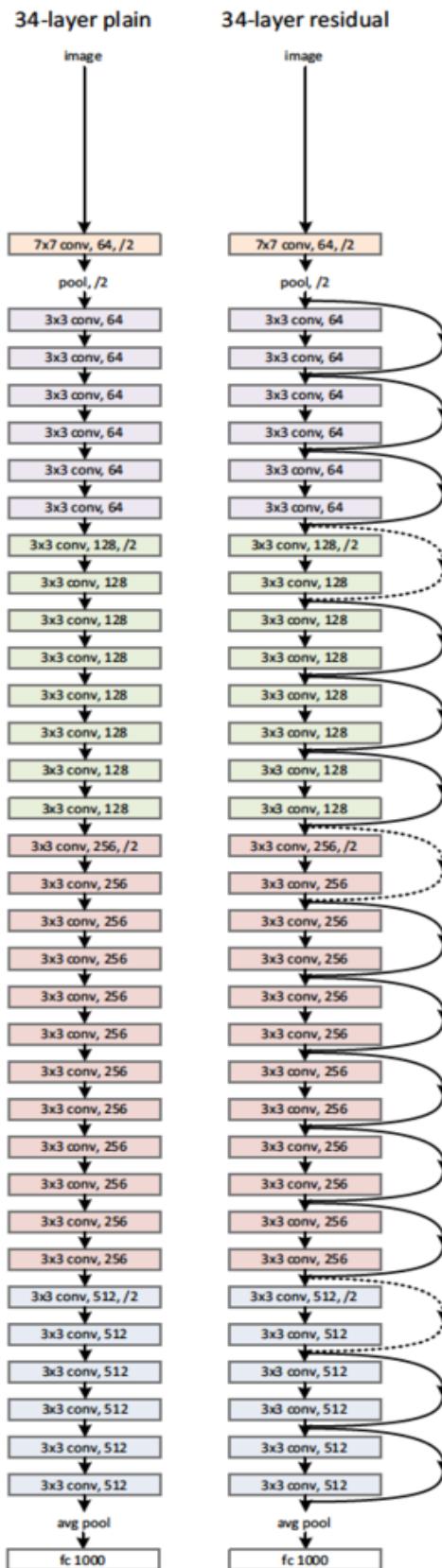
⁴² Source: <https://adshpande3.github.io/adshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html>



Another view of GoogLeNet's architecture.

Hình 2.42 Kiến trúc GoogleLeNet⁴³

⁴³ Source: <https://adshpande3.github.io/adshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html>



Hình 2.43 Kiến trúc Microsoft ResNet

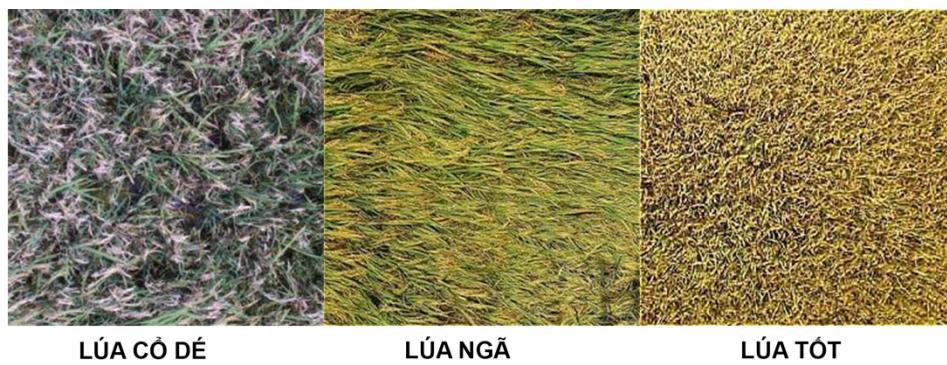
Chương 3. ỨNG DỤNG CNN TRONG PHÂN LOẠI ẢNH NÔNG NGHIỆP

3.1 Giới thiệu bài toán

Trong nhiều năm trở lại đây, vấn đề gia tăng năng suất cây trồng đã trở thành vấn đề được quan tâm lớn tại một số nước nông nghiệp, trong đó có Việt Nam. Nhu cầu về lương thực ngày một tăng nhưng diện tích đất nông nghiệp lại không tăng, vì vậy để giải quyết vấn đề gia tăng năng suất cây trồng, nông nghiệp chính xác (precision farming) là một giải pháp cơ bản. Nông nghiệp chính xác là một kiểu sản xuất nhằm đáp ứng, xác định chính xác nhu cầu, tình trạng của cây trồng, tránh sự lãng phí sử dụng phân bón và công chăm sóc, từ đó gia tăng năng suất và lợi nhuận cho nông dân. Một trong những yếu tố quan trọng trong nông nghiệp chính xác là việc xác định đúng bệnh hại trong cây trồng. Ở Việt Nam, đa số việc dự đoán bệnh hại phụ thuộc vào kinh nghiệm của nông dân. Áp dụng công nghệ kỹ thuật cao trong nông nghiệp, chúng tôi sử dụng mô hình Deep Learning vào trong bài toán phân loại ảnh nông nghiệp nhằm giúp nông dân có kết quả dự đoán chính xác và khách quan hơn về tình trạng cây trồng, qua đó giảm bớt chi phí, gia tăng năng suất.

3.2 Giới thiệu tập dữ liệu

Tập dữ liệu ảnh nông nghiệp sử dụng trong việc training được cung cấp bởi Ths. Nguyễn Cao Trí, bao gồm 7169 ảnh Lúa Cỏ Dé, 20513 ảnh Lúa Ngã và 17793 ảnh Lúa Tốt.



Hình 3.1 Ba loại ảnh được sử dụng trong bài toán phân loại ảnh nông nghiệp.

Để chuẩn bị cho việc training cho mô hình Convolutional Neural Network, chúng tôi chia tập dữ liệu đầu vào thành tập training data và tập test data:

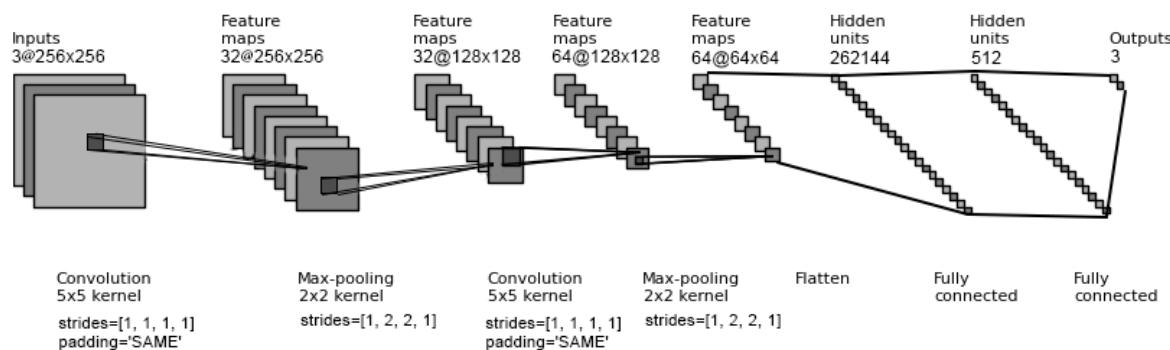
- Tập training data gồm 18000 ảnh, trong đó mỗi loại lúa gồm 6000 ảnh để đảm bảo cân bằng lớp.
- Tập test data gồm 3000 ảnh, trong đó mỗi loại lúa có 1000 ảnh.

Tất cả dữ liệu được gắn label, sau đó được trộn lại ngẫu nhiên và được lưu tại file cropfile.hdf5 theo dạng directory.

3.3 Kết quả

Chúng tôi tiến hành thử nghiệm tập dữ liệu trên mô hình Convolutional Neural Network với nhiều tham số khác nhau. Chúng tôi sử dụng thư viện mã nguồn mở Tensorflow để cấu hình các mô hình và tiến hành thực nghiệm trên máy tính cá nhân có CPU 2.7 GHz Intel Core i5, 8GB RAM, GPU Intel Iris Graphics 6100 1536 MB.

Dữ liệu được lấy ra và áp dụng kiến trúc tương tự LeNet, sử dụng framework TensorFlow:



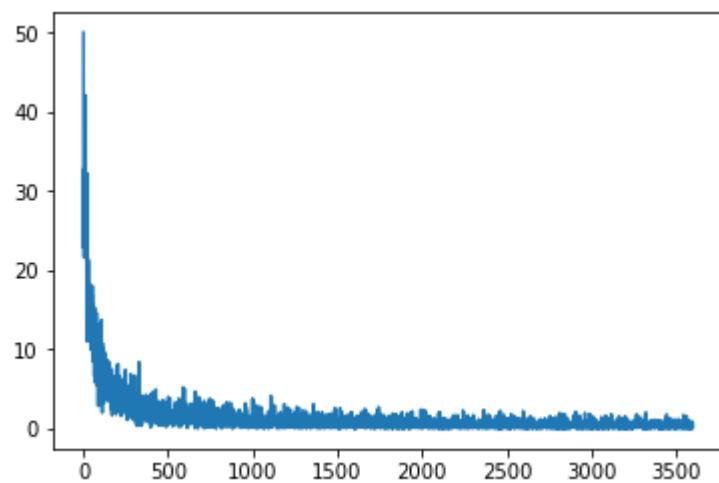
Hình 3.2 Kiến trúc sử dụng trong mô hình nhóm sử dụng

Trong đó áp dụng drop-out 50% trong fully connected layer đầu tiên ($252,144 \Rightarrow 512$).

3.3.1 Training với SGD

Tên	Giá trị	Ghi chú
Data	$[-1, 1]$	
Learning rate	1×10^{-4}	
Momentum	0	
L2 Regularization	0	
Dropout	0.5	Ở FC Layer thứ nhất
Chế độ training	Mini-batch gradient descent	
Mini-batch size	50	
Epochs	10	
Activation Function	ReLU	
Thuật toán tối ưu gradient descent	SGD	
Loss function	Cross Entropy	
% Training	6/7	
% Test	1/7	
Accuracy	0.966666638851	

Bảng 3-1 Training với SGD

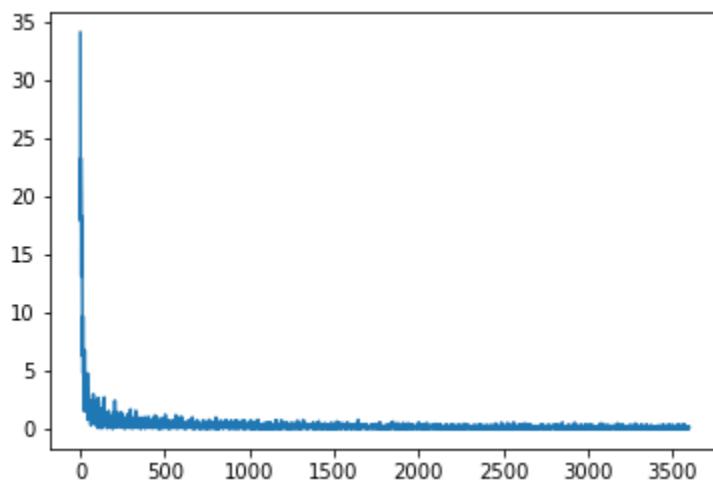


Hình 3.3 Đồ thị hàm loss khi training với SGD

3.3.2 Training với Nesterov Accelerated Gradient

Tên	Giá trị	Ghi chú
Data	$[-1, 1]$	
Learning rate	1×10^{-4}	
Momentum	0.9	
L2 Regularization	0	
Dropout	0.5	Ở FC Layer thứ nhất
Chế độ training	Mini-batch gradient descent	
Mini-batch size	50	
Epochs	10	
Activation Function	ReLU	
Thuật toán tối ưu gradient descent	NAG	
Loss function	Cross Entropy	
% Training	6/7	
% Test	1/7	
Accuracy	0.975000023842	

Bảng 3-2 Training với NAG

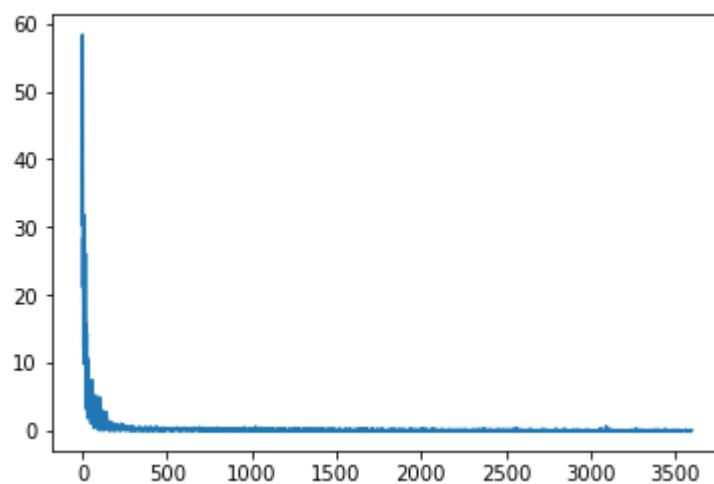


Hình 3.4 Đồ thị hàm loss khi training với NAG

3.3.3 Training với Adam

Tên	Giá trị	Ghi chú
Data	$[-1, 1]$	
Learning rate	1×10^{-4}	
Beta1	0.9	
Beta2	0.999	
Dropout	0.5	Ở FC Layer thứ nhất
Chế độ training	Mini-batch gradient descent	
Mini-batch size	50	
Epochs	10	
Activation Function	ReLU	
Thuật toán tối ưu gradient descent	Adam	
Loss function	Cross Entropy	
% Training	6/7	
% Test	1/7	
Accuracy	0.975333333015	

Bảng 3-3 Training với Adam

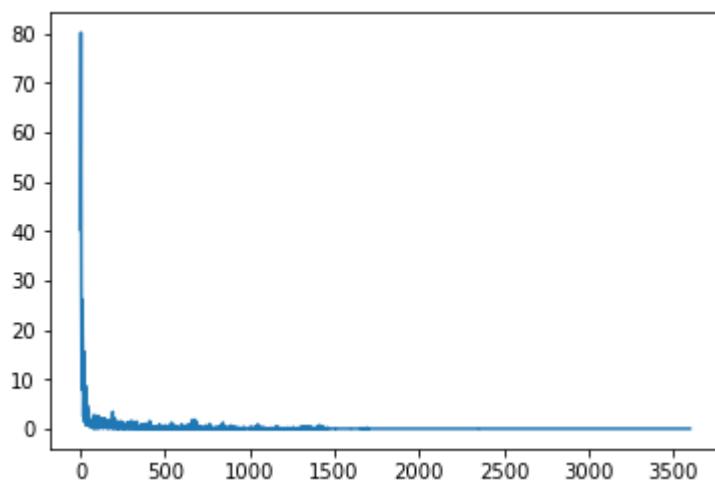


Hình 3.5 Đồ thị hàm loss khi training với Adam

3.3.4 Training với Adam không có Dropout

Tên	Giá trị	Ghi chú
Data	$[-1, 1]$	
Learning rate	1×10^{-4}	
Beta1	0.9	
Beta2	0.999	
Dropout	1	Ở FC Layer thứ nhất – Không có dropout.
Chế độ training	Mini-batch gradient descent	
Mini-batch size	50	
Epochs	10	
Activation Function	ReLU	
Thuật toán tối ưu gradient descent	Adam	
Loss function	Cross Entropy	
% Training	6/7	
% Test	1/7	
Accuracy	0.984666705132	

Bảng 3-4 Training với Adam không có Dropout

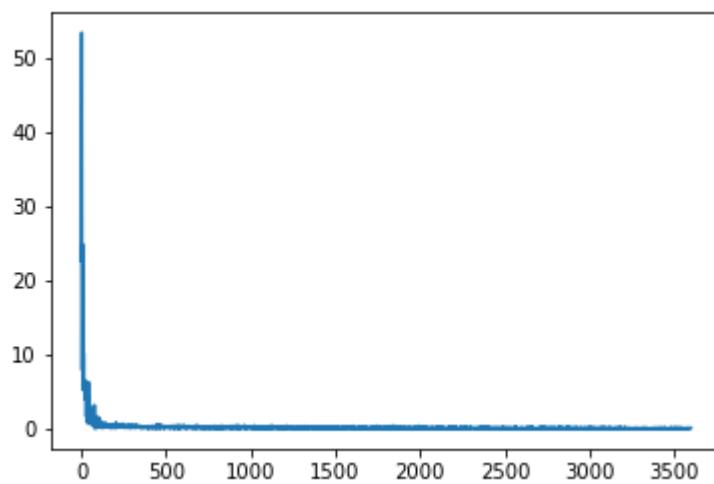


Hình 3.6 Đồ thị hàm loss khi training với Adam không có Dropout

3.3.5 Training với chỉ 1 Convolutional Layer

Tên	Giá trị	Ghi chú
Data	[-1, 1]	
Learning rate	1×10^{-4}	
Beta1	0.9	
Beta2	0.999	
Dropout	0.5	Ở FC Layer thứ nhất
Chế độ training	Mini-batch gradient descent	
Mini-batch size	50	
Epochs	10	
Activation Function	ReLU	
Thuật toán tối ưu gradient descent	Adam	
Loss function	Cross Entropy	
% Training	6/7	
% Test	1/7	
Accuracy	0.967000007629	

Bảng 3-5 Training với chỉ 1 Convolutional Layer



Hình 3.7 Đồ thị hàm loss khi training với chỉ 1 Convolutional Layer

CHƯƠNG 4. TỔNG KẾT

4.1 Tổng kết

4.1.1 Những việc đã làm được

Tìm hiểu sơ bộ về Machine Learning, Supervised Learning, Unsupervised Learning.

Tìm hiểu cấu tạo, kiến trúc và cách thức hoạt động của hypothesis function và loss function các bài toán Supervised Learning như Linear Regression, Logistic Regression, Neural Network. Trong đó tìm hiểu các khái niệm liên quan có ảnh hưởng đến kết quả bài toán như:

- Feature normalization trong việc preprocess data.
- Polynomial regression trong việc tăng các tính năng để tăng độ chính xác trong quá trình training.
- Bias variance tradeoff tránh dẫn đến overfitting và underfitting.
- Regularization sử dụng trong tradeoff để giảm overfitting.
- F1 score trong đánh giá một mô hình mà dữ liệu đầu vào ‘skewed classes’ hơn là dùng accuracy để đánh giá.
- So sánh các activation function, để tìm một activation function thích hợp.
- Gradient descent và các thuật toán tối ưu gradient descent.
- Sự ảnh hưởng của learning đến kết quả bài toán thông qua xem xét đồ thị hàm loss để điều chỉnh learning rate đến một giá trị thích hợp.

Tìm hiểu Convolutional Neural Network, các khái niệm liên quan như convolutional layer, convolutional kernel, pooling layer, padding, strides và một số các mô hình kiến trúc nổi tiếng đã được sử dụng trong thời gian gần đây.

Tìm hiểu thư viện toán học của python là numpy và các thư viện liên quan để thực hiện demo feedforward và backpropagation các bài toán đơn giản Linear Regression, Logistic Regression, Neural Network.

Thông qua đó tìm hiểu thư viện tensorflow được xây dựng với google để áp dụng chạy thử bài toán Convolutional Neural Network với tập dữ liệu được thầy cung cấp.

4.1.2 Những việc chưa làm được

- Chưa mô phỏng thể hiện hình ảnh ở từng layer.
- Chưa xuất kết quả ra một cách real-time.
- Chưa áp dụng training trên máy chủ chạy GPU.
- Chưa áp dụng được phương pháp map reduce để huấn luyện song song trên nhiều core trong một máy hoặc chạy song song trên nhiều máy.
- Chưa kết hợp áp dụng với hệ thống lưu trữ và xử lý dữ liệu lớn như Hadoop và Spark

TÀI LIỆU THAM KHẢO

- [1] SAS. *Machine Learning – What it is and why it matters.*
https://www.sas.com/en_us/insights/analytics/machine-learning.html
- [2] Andrew Ng. *Machine Learning Course (Week 1, 2, 3, 4, 5, 6, 10, 11).*
<https://www.coursera.org/learn/machine-learning/>
- [3] Standford University. *Optimization: Stochastic Gradient Descent.*
<http://cs231n.github.io/optimization-1/>
- [4] Standford University. *Neural Networks Part 1: Setting up the Architecture.*
<http://cs231n.github.io/neural-networks-1/>
- [5] Standford University. *Neural Networks Part 2: Setting up the Data and the Loss.*
<http://cs231n.github.io/neural-networks-2/>
- [6] Standford University. *Neural Networks Part 3: Learning and Evaluation.*
<http://cs231n.github.io/neural-networks-3/>
- [7] Standford Univesity. *Convolutional Neural Networks: Architectures, Convolution / Pooling Layers.* <http://cs231n.github.io/convolutional-networks/>
- [8] Sebastian Ruder. An overview of gradient descent optimization algorithms.
<http://sebastianruder.com/optimizing-gradient-descent/>
- [9] Adit Deshpande. *The 9 Deep Learning Papers You Need To Know About (Understanding CNNs Part 3).* <https://adethpande3.github.io/adethpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html>
- [10] Branden Archer. *Lagrange Interpolating Polynomial.*
<http://mathworld.wolfram.com/LagrangeInterpolatingPolynomial.html>
- [11] Standford University. *CS 20SI: Tensorflow for Deep Learning Research.*
<http://web.stanford.edu/class/cs20si/>
- [12] Google. *Deep Learning Course.* <https://www.udacity.com/course/deep-learning--ud730>

PHÂN CÔNG CÔNG VIỆC

Vũ Ngọc Linh	Tìm kiếm các site, source cung cấp kiến thức về đề tài, và chốt lại cái site, source (tài liệu kham khảo) được sử dụng để tìm hiểu. Tìm hiểu kiến thức thông qua các site, source đã chốt dùng để thực hiện demo Hỗ trợ viết báo cáo phần gradient descent và ứng dụng để demo
Nguyễn Đức Hoàn	Tìm hiểu kiến thức thông qua các site, source đã chốt và tiến hành trích xuất những ý chính được sử dụng để demo Dịch và viết báo cáo