DATA SCIENCES

DEEP LEARNING FOR BEGINNERS

Concepts and Algorithms

FRANÇOIS DUVAL

DEEP LEARNING FOR BEGINNERS

Practical Guide with Python and Tensorflow

François Duval

Data Sciences

If you find any damage, editing issues or any others issues in this book contains please immediately notify our customer service by email at:
customer_service@datasciences-book.com
You can also contact me directly if you have any questions or suggestions at:
Francois_Duval@datasciences-book.com

Our goal is to provide high-quality books for your technical learning in computer science subjects.

Thank you so much for buying this book.

François Duval

Preface

“People worry that computers will get too smart and take over the world, but the real problem is that they're too stupid and they've already taken over the world.”

— Pedro Domingos

Deep learning is a terminology used for a type of technological research and development that is responsible for building machines that are considered to be artificial intelligences. These are machines that are capable of completing complex functions such as speech recognition, image recognition, bioinformatics and more.

Understanding deep learning, including what it is and what makes it work, means that you will have a greater understanding of how computer scientists and researchers are developing the systems that may create a complete and pure artificial intelligence one day in the future. Until now, they have only created ones that can complete one or two functions. However, it is hoped that one day they will create something that closely resembles the human brain. Ideally, they want to create a “pure” artificial intelligence, which is one that would be created by a human but would learn and operate entirely free of human influence. In essence, it would be a “superhuman”.
“Deep Learning: Concepts and Algorithms” was written to help you understand what deep learning is and why it is such a fascinating and revolutionary concept. Although it is not a new concept, it is one that is becoming more and more popular as we speed towards the possibility of having a true artificial intelligence walking amongst us.

One thing you may be fascinated to learn as you read through this book is how much we are already using deep learning technologies. Even though the idea of a complete and pure artificial intelligence walking amongst us seems to be a reality of the distant future, the amount we use the technology already is rather fascinating.

Deep learning is an incredible concept that is here to stay. The revolutionary science that is taking place on a daily basis and is slowly being integrated into our society for regular consumer use is a fascinating one, and it is incredible to learn about. Understanding it may help you understand the

technology you are using right now, and the future of society. You may be surprised about what you learn! Please, take your time and enjoy reading this educational book about deep learning.

Book Objectives

If you are interested in learning more about deep learning, what is used to create them, the concept of deep learning, the techniques used by the machines, and the tools that are being designed by these scientists, then Deep Learning: Concepts and Algorithms is exactly what you need. This book will help you explore exactly what deep learning is and will also teach you about why it is so revolutionary and fascinating. By the end you will have a full understanding of what this technology is, how it works,

This book will help you:

- Have an appreciation for deep learning and an understanding of their fundamental principles.
- Have an elementary grasp of deep learning concepts and algorithms.
- Have achieve a technical background in deep learning and also artificial neural networks

Target Users

The book designed for a variety of target audiences. The most suitable users would include:

- Newbies in computer science techniques and deep learning
- Professionals in data science and social sciences
- Professors, lecturers or tutors who are looking to find better ways to explain the content to their students in the simplest and easiest way
- Students and academicians, especially those focusing on neural networks and deep learning

Is this book for me?

If you want to learn more about deep learning, this book is for you. Little maths knowledge is required. If you already have a basic notion in statistic and data science, you'll be OK. If not, online courses covers more than what's required. You can do one in a week, for free.

© Copyright 2017 by François Duval.
All rights reserved.
First Printing, 2017

Edited by Davies
Ebook Converted and cover by Pixel Studio
Publised by CreateSpace Publishing

ISBN-13: 978-1983843822
ISBN-10: 1983843822

The contents of this book may not be reproduced, duplicated or transmitted without the direct written permission of the author.

Under no circumstances will any legal responsibility or blame be held against the publisher for any reparation, damages, or monetary loss due to the information herein, either directly or indirectly.

Legal Notice:

You cannot amend, distribute, sell, use, quote or paraphrase any part or the content within this book without the consent of the author.

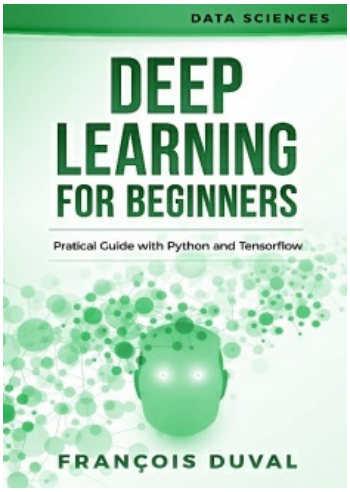
Disclaimer Notice:

Please note the information contained within this document is for educational and entertainment purposes only. No warranties of any kind are expressed or implied. Readers acknowledge that the author is not engaging in the rendering of legal, financial, medical or professional advice. Please consult a licensed professional before attempting any techniques outlined in this book.

By reading this document, the reader agrees that under no circumstances is the author responsible for any losses, direct or indirect, which are incurred as a result of the use of information contained within this document, including, but not limited to, errors, omissions, or inaccuracies.

To my wife Melanie and my two children Mariane and Thomas
You are my life and I love you so much!

In the same Series: Data Sciences



For more information click here

Data Sciences

Author Biography

François Duval is a seasoned data science expert and emerging author who has quickly earned the reputation as an innovative leader in the information technology and data analysis space. Throughout the course of the past decade, he has gained extensive firsthand experience in his field.
Currently, François is proudly serving as a scientific data consultant. He is also working on series of books in statistics and computer science.

Prior to his present venture, he was on the IBM team for two years and is the founder of Intelligence Data Consulting, which is a prominent company that provides data analysis consulting solutions in France and throughout Europe. He has also worked as an engineer across multiple industries, including but not limited to finance (KPMG within the Financial Advisory Department in London) as well as environmental and healthcare research (INERIS and Mines ParisTech in France).
No matter what venture he takes on, François is on a lifelong mission to impart data analysis newbies with the knowledge they need to better grasp various industry techniques. By effectively simplifying such methodologies, up-and-coming professionals are empowered to create positive changes in the world of computer science that will continue to have a ripple effect in the entire field as we know it.
Furthermore, François holds a PhD in Quantitative Analysis and Stochastic Calculus and an Undergraduate Degree in Statistics & Data Analysis.

When he isn't immersed in his multifaceted career, François Duval enjoys spending quality family time with his loved ones. Today, he happily resides in London, England and is the proud father of two beautiful children.

Table of Contents

Overview

1. Pre-requisite for Deep Learning

What is deep learning?
The Mechanics of Machine Learning
Different Machine Learning Approaches
- Supervised learning
- Unsupervised learning
- Reinforcement learning

Differences between Supervised and unsupervised algorithms
Classification and clustering
- Classification
- Clustering

Artificial Neural Networks and Deep Leaning
Fast graphics processing units (GPUs) for deep learning in neural networks
GPU vs. CPU
Training deep learning model
A three-way categorization
- Deep networks for unsupervised or generative learning
- Deep networks for supervised learning
- Hybrid deep networks

Recurring methods of deep learning
- Dynamic programming for supervised learning / reinforcement learning
- Unsupervised learning (UL) facilitating SL and RL
- Learning hierarchical representations through deep SL, UL, RL
- Occam's razor: compression and minimum description length (MDL)

2. Introduction to Artificial Neural Networks

An Overview of Neural Networks
- The human brain and the computer

What is a Neural Network?
Artificial Neural Network
- Input layer
- Hidden layer
- Output layer
- Artificial Neuron structure
- Supervised and unsupervised neural network

Applications of Neural Networks
- Clustering
- Classification/pattern recognition:
- Function approximation
- Prediction/dynamical systems

3. The Basics of Artificial Neural Networks

Neural Networks Viewed as Directed Graphs
The Perceptron
Types of Activation Function
- Threshold function
- Piecewise-function
- Sigmoid function

Adaline Learning Algorithm
The Widrow-Hoff Delta Rule
The Delta Rule and Learning Rate
Gradient Descent with Sigmoid Neurons

4. Deep Learning Evolution and Recurring Methods

History of Deep Learning
Deep Learning Timeline
Deep Learning and Artificial Neural Networks
Why is Deep Learning Called Deep?
Perceptron: An early deep learning algorithm
A three-way categorization
Recurring methods of deep learning
- Dynamic programming for supervised learning / reinforcement learning (SL/RL)
- Unsupervised learning (UL) facilitating SL and RL
- Learning hierarchical representations through deep SL, UL, RL
- Occam's razor: compression and minimum description length (MDL)
- Fast graphics processing units (GPUs) for deep learning in neural networks

5. Relationship between machine learning and deep learning

Comparison of deep learning and machine learning
- Data dependencies
- Hardware dependencies
- Feature engineering
- Problem solving approach
- Execution time
- Interpretability

Where deep learning and machine learning are applied
Future trends

6. Multilayer Perceptron (MLP)

Multilayer perceptron
- The MLP model
- Tricks for training MLPs

Logistic regression
Stochastic Gradient Descent (SGD)
Logistic regression to multilayer perceptron
Backpropagation

Gradient descent
Definition backward propagation
Applications of backpropagation
Long Short Term Memory networks for sentiments analysis
The LSTM model

7. Convolutional Neural Networks (CNN)

CNN Architecture
Layers used to build CNN
Introducing non-linearity (RELU)
Convolution layer
Without brain model
The brain-view
Local connectivity
Spatial arrangement
Real life example
Parameter sharing
Pooling layer
General pooling
Getting rid of pooling
Fully connected layer
Converting FC layers to CONV layers
CONV-Net Architectures
Layer pattern
Layer sizing Patterns
Other CONV-Net Architectures

8. Other Deep Learning Algorithms

Restricted Boltzmann machines
Boltzmann machines in depth:
RBMs with binary units
Update equations with binary units
Energy-based model (EBM)
EMBs with hidden units
Auto encoders
Stacked auto encoders
Deep belief networks
The RNN (recurrent neural network) and RBM (restricted Boltzmann machine)

9. Deep Learning Applications

Applying deep learning to real-world problems
The value of pre-training
Medical vision
Credit scoring
Solution to the high cost

Caveats of real-world label distributions
Understanding the black box model
Visual applications of deep learning
Adding sounds to silent movies:
Automatic machine translation
Automatic handwriting generation
Automatic game playing
Applications using existing APIs
Open-source applications

10. Glossary of Some Useful Terms in Deep Learning

Layer
Unit
Activation function
Feature engineering
Feature Learning
Gradient descent

11. Useful References

Thank you !

Overview

Deep neural networks have revolutionized the classification of images in the last decade. The increase in computing power and tools to train neural networks using several million free parameters is a state-of-the-art tool.

Deep learning permits computational models that consist of various processing layers to learn the portrayal of information with various levels of abstraction. These techniques have significantly enhanced the best in class in speech recognition, visual object acknowledgment, and question identification as well as numerous other areas, such as drug disclosure and genomics. Deep learning finds many-sided structures in substantial informational indexes by utilizing the backpropagation calculation to show how a machine should change its inner parameters, which are utilized to figure the portrayal in each layer from the portrayal in the past layer. Deep convolutional nets have achieved leaps forward in handling pictures, video, discourse, and sound, while repetitive nets have shed light on consecutive information, such as content and speech.

Encouraged by the brain, deep neural networks (DNN) are thought to learn abstract representations through their hierarchical architecture. However, at present, how this happens is not well understood.

Deep learning algorithms try to find patterns to make use of the unknown structure in the input distribution in order to determine high-quality representations, often at numerous levels, with higher-level learned features defined in terms of lower-level features. The purpose is to make these higher-level representations more conceptual, where by their individual features are more invariant compared to most of the variations that are naturally present in the training distribution, while collectively preserving as much as possible of the information contained in the input. In an ideal world, we would like these representations to disentangle the unknown factors of variation that underlie the training distribution. Such unsupervised learning of representations can be usefully broken down under the hypothesis that the input distribution $P(X)$ is structurally related to $P(Y|X)$.

The big payoff of deep learning is that it allows learning higher levels of abstraction. Chapter 1 gives a complete introduction to deep learning with a short historical background. Early deep learning models are also discussed here. The next few chapters will give the detailed structure of deep learning

algorithms with pictorial representations and their inner workings? As deep learning is used in almost every field, its application and some comparisons with other techniques are provided. As deep learning has become a crucial machine learning tool, real-world comparisons are explained. Mathematical and computational principles are covered, providing examples in order for the reader to acquire knowledge.

Applications of deep learning are used in everyday objects and the concept contains breakthrough ideas. Some of these applications blow your mind! While some of you might be aware of these applications, some of you might not be.

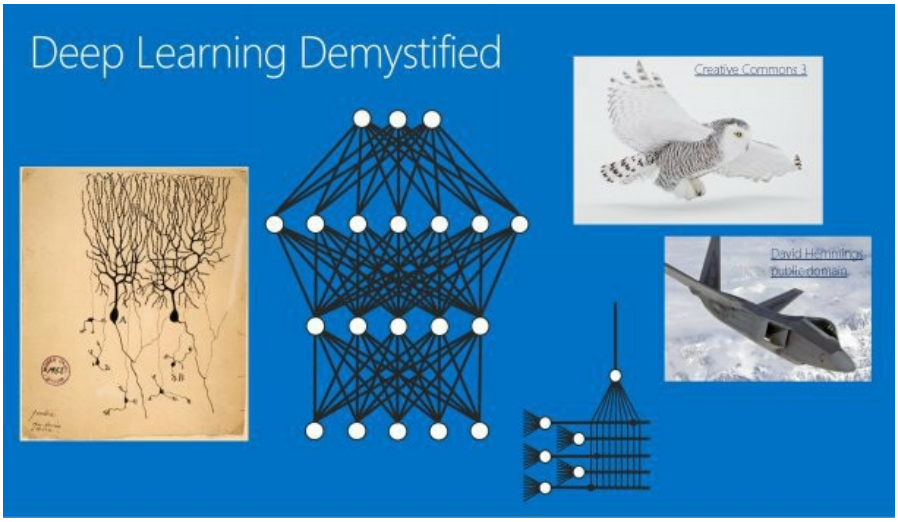
Appreciating how today's artificial intelligence is working may seem overwhelming, but it really boils down to two concepts, which you most likely have heard of before: "machine learning" and "deep learning". Neither are brand-new ideas, but the way they're used to describe intelligent machines seems to constantly evolve. Machine learning and deep learning tells us how Netflix knows what you might want to watch next, or how Facebook can recognize your friends' faces in a photo. Chapter 6 is about all such applications and how they work.

1. Pre-requisite for Deep Learning

Objective of this Chapter

At the end of this chapter, the reader should be able to understand:

- Link between Machine Learning and Deep Learning
- Machine learning algorithms



What is deep learning?

Deep learning is a subfield of machine learning that is concerned with calculations enlivened by the structure and capacity of the mind, known as artificial neural systems. There is a useful visual aid in figure 1.1, showing how artificial intelligence, machine learning, and deep learning relate to each other.

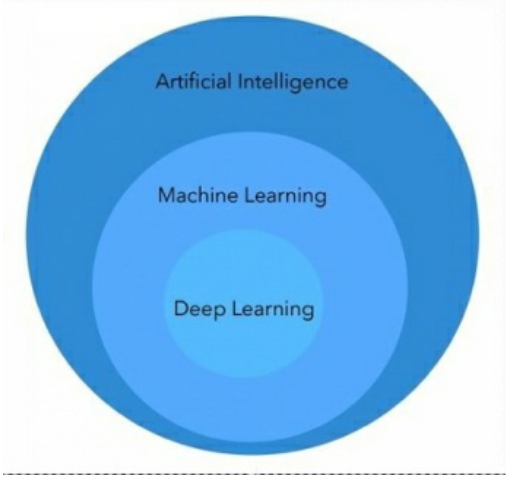


Figure 1.1: Three concentric circles

This method includes sustaining a model with huge volumes of information, yet it requires less element building than a linear regression model would.

"Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones."

"A class of machine learning techniques that exploit many layers of non-linear information processing for supervised or unsupervised feature extraction and transformation, and for pattern analysis and classification."

A deep neural network comprises of hierarchy of layers, also known as a hierarchal neural network, whereby each layer changes the input data into a more abstract representation. The output layer combines these features to make predictions. It improves calculation, to make it simple and to make it understand. It has allowed progressive advances in machine learning and AI (artificial intelligence) to be made. Deep learning models have the capacity to perform such tasks as extraction from raw information, otherwise called highlight learning (most commonly called feature learning), and shown in figure 1.2.

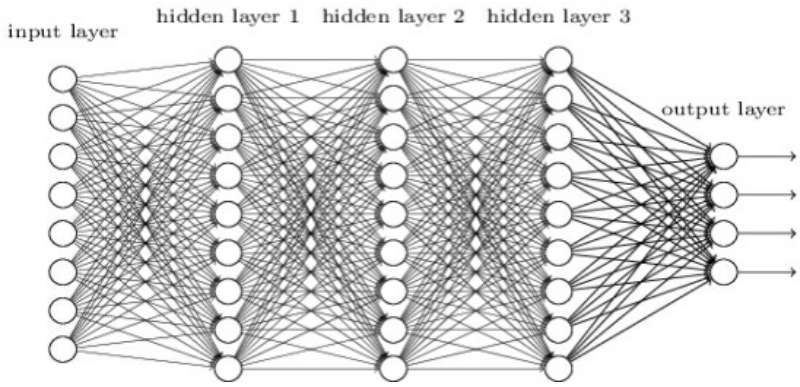


Figure 1.2: General architecture of deep neural network

How about we take a case of a creature recognizer, where our framework needs to perceive whether the given picture is of a cat or a dog.

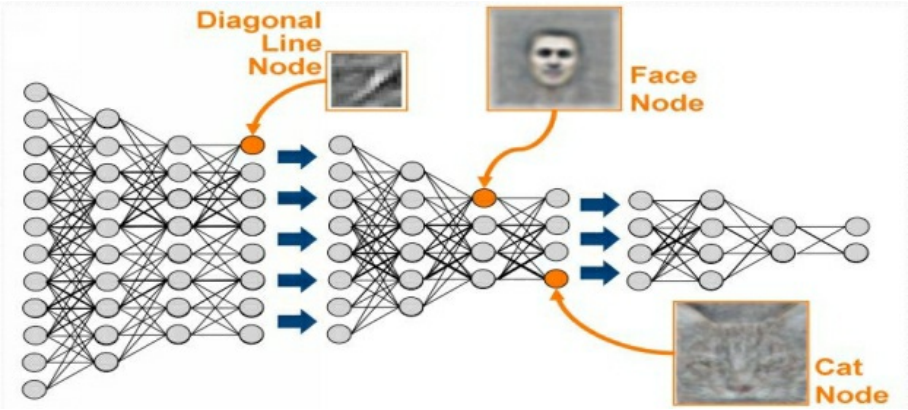


Figure 1.3: Face recognition

In the event that we settle this as a regular machine learning issue, we will characterize highlights; for example, whether the creature has hairs, whether it has ears and, if so, then at that point in the event whether they are pointed. To put it plainly, we will characterize the facial highlights and let the framework distinguish which highlights are more vital in categorizing specific creature.

Presently, deep learning takes this one step further. Deep adapting consequently discovers the highlights that are essential for grouping, whereby in machine learning we needed to physically give the highlights. Deep learning acts as follows:

- It initially distinguishes which edges are most applicable to determine “cat” or “dog”.
- At that point it expands on this progressively to discover what mix of

shapes and edges we can discover. For instance, whether hair or ears are available, and so forth.

- After a continuous hierarchal identification of complex ideas, it chooses which of these highlights are in charge of finding the appropriate response.

The Mechanics of Machine Learning

Machine learning strategies are being implemented in many fields and enterprises. We distinguish the procedure through which we can selected information that isn't promptly evident for information in order to decide, with the assistance of machine learning.

Deep learning is a subset of a broader field of manmade brainpower called machine learning, which is predicated on this thought of gaining from case. In machine learning, for example, when training a computer the enormous rundown principles take care of the issue, we give it a model with which it can weigh up cases as well as a little arrangement of directions to change the model when it makes an error. We expect that, after some time, an appropriate model would have the capacity to take care of the issue, to a certain degree, precisely.

We should be somewhat more exact about what this implies so that we can determine this thought scientifically. Let us depict our model to be a capacity h(x,θ). The information x is a case expressed in a vector shape. For instance, if x were a grayscale picture, the vector's parts would be pixel forces at each position, as shown in Figure 1.4.

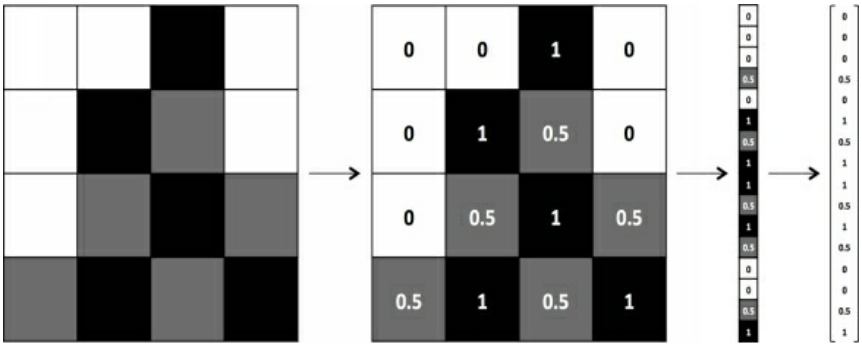


Figure 1.4: Vectorization process an image for a machine learning algorithm

The input θ is a vector of the parameters that our model uses. Our machine learning program tries to perfect the estimations of these parameters as it is appears for an ever-increasing number of cases. We will see this in real life in

more detail in a future area. To develop a deeper understanding for machine learning models, let us walk through a quick example. Let us say we are required to determine how to predict exam performance based on the number of hours of sleep we have had and the number of hours we have studied the previous day. We collect a lot of data, and for each data point x=[x1, x2] we record the number of hours of sleep we got (x1), the number of hours we spent studying (x2) and whether we performed above or below the class average. Our goal, then, might be to train a model h(x, θ), θ with a parameter vector

$$\theta = [\theta_0 \theta_1 \theta_2]^T$$

So that:

$$h(x,\theta) = \begin{cases} -1 \text{ if } x^T . \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} + \theta_0 < 0 \\ 1 \text{ if } x^T . \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} + \theta_0 \geq 0 \end{cases}$$

In other words, we guess that the blueprint for our model h(x,θ) is as described above (geometrically, this particular blueprint describes a linear classifier that divides the coordinate plane into two halves). Then, we want to train the parameter vector θ so that our model makes the right predictions (-1 if we perform below average, and 1 otherwise) given an input example x. This is model is called a linear perceptron. Let us assume our data is as shown in Figure 1.5.

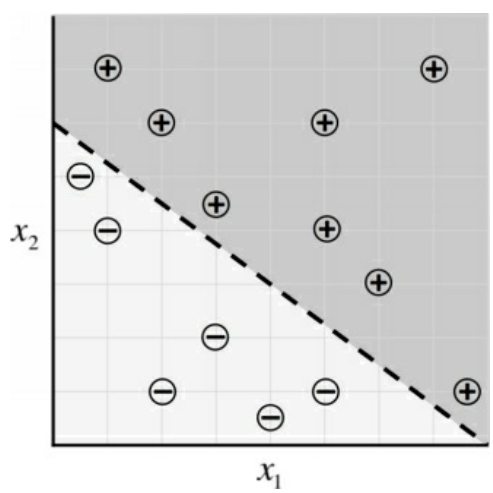


Figure 1.5: Sample data for our exam predictor algorithm and a

potential classifier

$$\theta = [-24\ 3\ 4]^T$$

Our machine learning model makes the correct prediction on every data point:

$$h(x,\theta) = \begin{cases} -1\ if\ 3x_1 + 4x_2 + -24 < 0 \\ 1\ if\ 3x_1 + 4x_2 + -24 \geq 0 \end{cases}$$

An ideal parameter vector θ positions the classifier with the goal that we make whatever number of correct forecasts as could reasonably be expected. Much of the time, there are numerous (or even boundlessly numerous) conceivable decisions for θ that are ideal. Fortunately for us, most of the time these alternatives are so close to one another that the difference in their performance is negligible. If this is not the case, we may want to collect more data to narrow down our choice of θ.The distributions of data shown in Figure 1.6 cannot be described well by a linear perceptron. As we move onto much more complex problems, such as object recognition and text analysis, our data not only becomes extremely high dimensional, but the relationships we want to capture also become highly nonlinear.

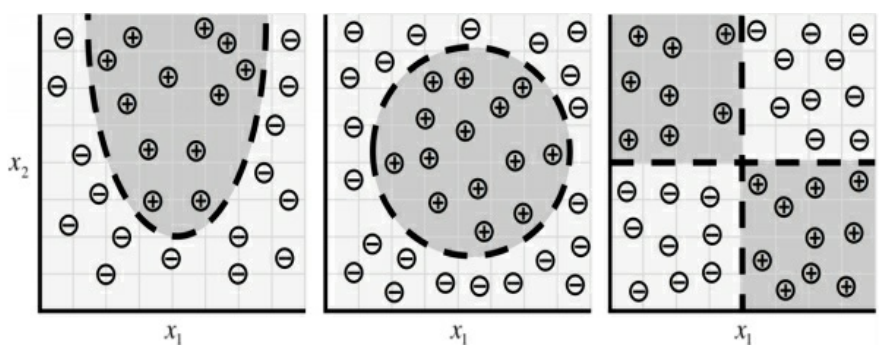


Figure 1.6: Complex models

Different Machine Learning Approaches

Machine learning strategies can for the most part be grouped into two classes, namely supervised and un-supervised classes, and more are often added that is, such as reinforcement learning.

Supervised learning

A supervised learning algorithm is a class of machine learning that uses a set

of labeled data to classify same un-labeled data. Data that has been classified is labeled data while un-labeled data is not yet labeled. Labels can be discrete or continuous. The majority of practical machine learning uses supervised learning. Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.

$$Y=f(x)$$

The aim is to approximate the mapping function so finely that when you have new input data (x) you can predict the output variables (Y) for that data.
To better comprehend, we should utilize another illustration. A robot is figuring out how to sort waste utilizing visual ID. All day it chooses recyclable things from rubbish as it passes along a transport line. It places materials such as glass, plastic and metal into 12 containers. Each thing is marked with an ID number on a sticker. Once per day, human specialists look at the containers and instruct the robot as to which things were incorrectly arranged. The robot utilizes this input to make strides.

An AI that is figuring out how to distinguish pedestrians on a road is prepared with 2 million short recordings of road scenes from self-driving autos. A portion of the recordings contain no pedestrians at all while others have up to 25. An assortment of learning calculations is prepared based on the information, with each using the right answers. Each calculation builds up an assortment of models to recognize walkers in fast moving scenes. The calculations are then tried against another arrangement of information to assess exactness and accuracy.
After the algorithm has worked on labeled data, also known as training data, it stops improving its accuracy. It can be than used on new un-labeled data. Sometimes the output may not necessarily be discrete and we may not have a finite number of classes to classify our data. Figure 1.7 below, for example, can turn out to be very complex when we have hundreds or thousands of dimensions.

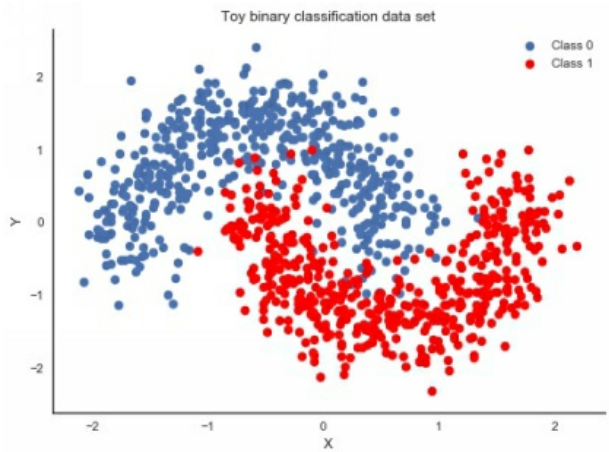
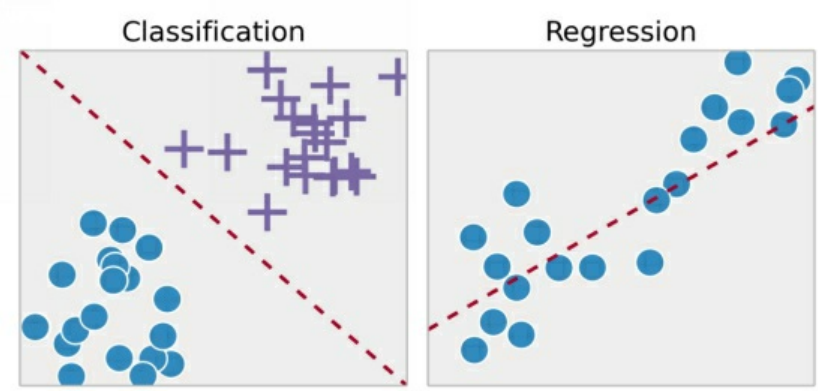


Figure 1.7: Classification as separating input

Supervised learning problems can be further grouped into regression and classification problems (Figure 1.8).

- *Classification of an arrangement issue is the point at which the yield variable is a class, for example, "red" or "blue" or "sickness" and "no infection".*
- *Regression of a relapse issue is the point at which the yield variable is a genuine estimation, for example, "dollars" or "weight".*



.

Figure 1.8: Difference between classification and regression

Some popular examples of supervised machine learning algorithms are:

- Linear regression for regression problems.
- Random forest for classification and regression problems.
- Support vector machines for classification problems.

Unsupervised learning

Unsupervised learning is where you just have input information (x) and there are no related yield factors; we allow the calculations to lead to the answer. An AI framework may amass unsorted data as per likenesses and contrasts despite the fact that no classifications have been given. The objective of unsupervised learning is to display the fundamental structure or dissemination in the information in order to take in more about the information. This is called unsupervised learning on the grounds that, unlike supervised learning, there is no right answer and no instructor. Calculations are left to their own particular devises to discover and introduce the strange structure in the information.

Unsupervised learning calculations can perform more troublesome preparing undertakings than supervised learning frameworks. Be that as it may, unsupervised learning can be more capricious than the substitute model. While an unsupervised learning AI framework may, for instance, make sense by itself how to sort cats from dogs, it may likewise add unanticipated and undesired classes to manage unusual breeds, making a mess rather than arranging. These strategies are frequently utilized for data mining. The spellbinding displaying assignment called design disclosure is utilized to find the relationships within the data. Unsupervised learning problems can be further grouped into clustering and association problems.

Clustering A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.

Association An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people who buy X also tend to buy Y.

Some popular examples of unsupervised learning algorithms are:

- K-means, for clustering problems.
- A priori algorithm for association rule learning problems.

Reinforcement learning

In reinforcement machine learning algorithms, the machine is trained to understand the activities going on in the world around, just like human beings, by applying a certain level of intelligence. The machine learns from the repercussions of its own actions without being taught a thing! In short, the purpose of reinforcement learning is to have a good policy, not a good decision. To understand this, let us take an example of your pet dog. Consider teaching the dog a new trick: you cannot tell it what to do, but you can reward if it does right things and penalize if it does wrong things. The reinforcement learning specialist (dog) needs to make sense of what it did that influenced it to getting the reward or penalty. A reinforcement learning operator (dog) learns by cooperating with its conditions and by watching the aftereffects of these communications.

5 Things you should know about reinforcement learning:

- There is no boss, just an idea of reward.
- That implies that the reinforcement learning agent is the one making the choices in view of some reward per activity.
- Feedback is postponed, not prompt. Unlike other machine learning methods, the criticism might be postponed in reinforcement learning; it can even occur toward the end, dissimilar to different systems.
- Time matters. The circulation is not any freer nor indistinguishably appropriated; it changes with time, as decided by the operator.
- The agent's activities influence the ensuing information it receives. As the specialist is really interfacing with the earth, the following activity is in fact subject to the present activity.

Differences between Supervised and unsupervised algorithms

Supervised learning (figure 1.9) is the most widely recognized type of machine learning. With supervised learning, an arrangement of cases, called the training set, is submitted as input to the framework amid the training phase. Each input is designated with a desirable output value; thus, the framework knows what the output is when input enters. For instance, consider an experimental observation that can be grouped into N diverse classes. Here, we have a training set (an arrangement of sets). The preparation is performed by the minimization of a specific cost work, which speaks to the problems from input xi and the desired output yi.

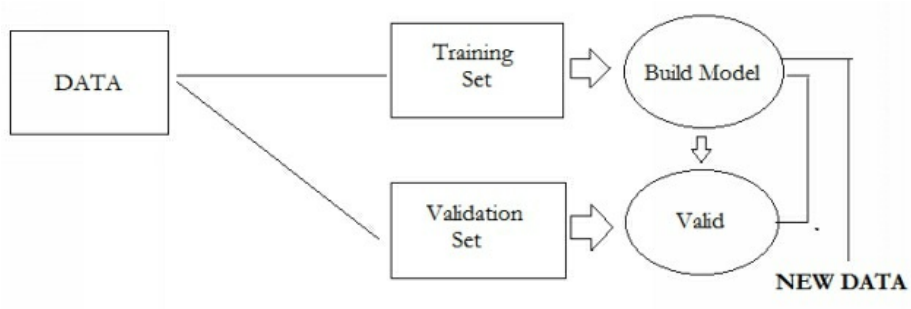


Figure 1.9: Supervised learning

An ordinary case of a supervised machine learning application is the mail spam identifier; the calculation is prepared using spam and not-spam messages and which each assemble a group. This preparation procedure occurs when you check an email as "spam" or "not spam" in your inbox.

In unsupervised learning (figure 1.10), on the other hand, the training illustrations given by the framework are not named with a class. Thus, the framework creates and arranges the information, looking for basic attributes among them and changing them in the light of internal knowledge.

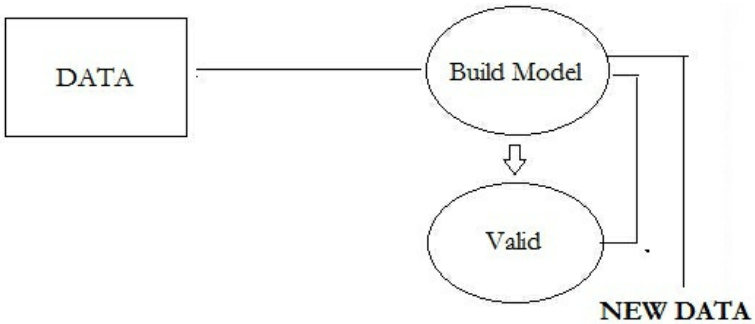


Figure 1.10: Unsupervised learning

Supervised Models	Unsupervised Models
• Neural networks • Multilayer perceptron • Decision trees	• Different types of clustering • Distances and normalization • K-means • Self-organizing maps

A case of unsupervised learning is clustering classification, which is a calculation that puts similar things in a group and different things in an alternate group, whereby the idea of similarity relies on a likeness measure.
Difference between supervised and unsupervised learning (figure 1.11):

Supervised learning	Unsupervised learning
• Training data includes input and desired output • Fast and accurate • Has to be able to generalize	• No correct results are provided • More complex • Cluster significance and labeling

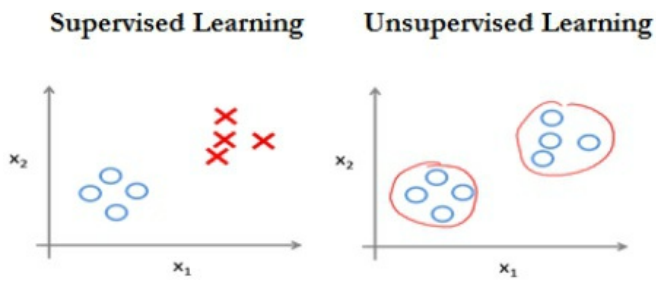


Figure 1.11: Difference between supervised and un-supervised learning

Classification and clustering

Clustering and classification seem to be similar processes. They are actually two types of learning methods, and they categorize objects into groups according to their features.

Classification

In classification on the basis of a training set of data, objects are differentiated. The aim is to predict the target class by analyzing the training set. To get better boundary conditions that can be used to determine each target class, a training dataset is used.

Classification algorithms

Any classification algorithm can be used; some are named below.

- Linear classifiers
    - Logistic regression
    - Naive Bayes classifier
    - Fisher’s linear discriminant

- Support vector machines
    - Least squares support vector machines
- Quadratic classifiers
- Kernel estimation
    - k-nearest neighbor
- Decision trees
    - Random forests
- Neural networks
- Learning vector quantization

Applications of classification algorithms

- Email spam classification
- Prediction of a bank customer's willingness to pay back a loan
- Tumor cell identification
- Sentiment analysis
- Drug classification
- Facial key points detection
- Pedestrian's detection in self-driving vehicles

Clustering

In clustering, objects with the same features are grouped together and those with dissimilar features are also grouped together. The target class is not predicted, as it is in classification. It is about grouping similar kinds of things by considering the most satisfied condition, whereby ***all the items in the same group should be similar and no two different group's items should be similar (shown in figure 1.11).***
Example: grouping similar language type document in one group.

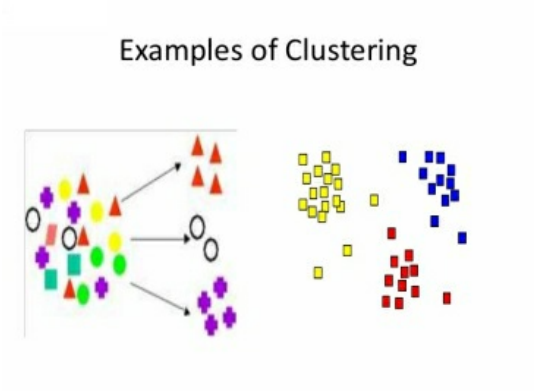


Figure 1.12: Clustering

Clustering algorithms

Clustering algorithms can be classified into two main categories linear clustering algorithms and Non-linear clustering algorithms.

- **Linear clustering algorithms**
  - k-means clustering algorithm
  - Fuzzy c-means clustering algorithm
  - Hierarchical clustering algorithm
  - Gaussian(EM) clustering algorithm
  - Quality threshold clustering algorithm
- **Non-linear clustering algorithms**
  - MST based clustering algorithm
  - Kernel k-means clustering algorithm
  - Density-based clustering algorithm

Applications of clustering algorithms

- Recommendation systems
- Anomaly detection
- Human genetic clustering
- Genome sequence analysis
- Analysis of antimicrobial activity
- Grouping of shopping items
- Search results grouping
- Slippy map optimization
- Crime analysis
- Climatology

Artificial Neural Networks and Deep Leaning

Deep learning is made possible through the introduction and use of artificial neural networks. These are networks that were developed to reflect the biological neural networks that work in animal brains, enabling them to function as independent creatures. These systems have the capability to learn on their own, which grants them the ability to progressively improve their knowledge and abilities. The idea behind these systems is that they can learn to do tasks on their own without requiring special programing or interference from humans to teach them how to do so, much like true animals do.

One great example of how an artificial neural network works is examining how they respond to pictures containing dogs in them. For example, if there was programming that had pictures labeled “dog” and “no dog”, the artificial neural network would begin to identify which element in the picture was actually the dog. As a result, it would know what dogs looked like. From thereon out, it would be able to identify dogs and it would have a sound understanding of what dogs were. This is much like if you were to teach a baby how to identify a dog. At first you must teach them how, but after that they can identify dogs in any number of situations. They can also then develop information about dogs, such as how they look or what defines a dog and what doesn’t. For example, they would be able to distinguish a dog from a cat based on appearance and, later, actions and sounds. Artificial neural networks have this same ability, making them capable of being their own

independent species.

Developing an artificial neural network requires a large series of connected units. These are called artificial neurons and they were created to resemble axons in the biological brain, which are a form of neuron. Each neuron has its own connection to other neurons, called a synapse. This synapse is used to transmit signals to other neurons in the artificial neural network. Much like an animal brain, the receiving neuron can then transmit that signal to other neurons in a sequence that allows the artificial neural network to operate just like an animal brain would. The result of the sequencing is that certain signals and messages are sent throughout the brain, which can lead to the body to which the brain is attached acting, reacting, and responding in a variety of unique ways.

In a true animal brain, neurons are organized in a series of layers. Each one of these layers is responsible for helping brain perform different tasks, which means each one essentially specializes in certain information and transformations. It can take a series of transmissions from one neuron to the next in order to get the signal to transmit from the first input layer to the last output layer. Originally, scientists responsible for working with and creating artificial neural networks were attempting to mimic the animal brain entirely, which is why it was built in such a way. Later, however, they decided to pay attention to creating specific abilities in each creation so that each one was specialized.

To date, scientists and researchers have used artificial neural networks to create systems that can accomplish a great variety of tasks. These tasks include anything from speech recognition and computer vision to computerized opponents in games and even medical diagnosis systems. Although artificial neural networks have become incredibly talented and have advanced far beyond what the original artificial neural networks ever were, they are still very simple in comparison to the human brain. At the time of this publication, they have been compared to worm brains. Despite only being considered as being at the same capacity as a worm’s brain, they have been made capable of completing many tasks that are far more advanced than anything a worm could do.

Fast graphics processing units (GPUs) for deep learning in neural networks

The CPU (central processing unit) has often been called the brains of the PC. But increasingly, that brain is being enhanced by another part of the PC – the GPU (graphics processing unit), which is its soul. All PCs have chips that render the display images to monitors. But not all these chips are created equal. Intel's integrated graphics controller provides basic graphics that can display only productivity applications like Microsoft PowerPoint, low-resolution video and basic games.

The GPU is in a class by itself it goes far beyond basic graphics controller functions, and is a programmable and powerful computational device in its own right. Deep Learning requires big systems to run execute and GPU must be a part of the solution.

GPUs are generally utilized for computer games, an enormous and focused market that has driven down equipment costs. GPUs exceed expectations at the quick matrix and vector multiplication required to persuade/ show or give virtual substances as well as for neural network training, where they can accelerate learning by a factor of 50.

GPU vs. CPU

GPUs have hundreds of simple cores, while a CPU has a few simple cores. GPUs have thousands of concurrent hardware threads and can maximize floating point throughput.

To comprehend the feature, we take an example that clarifies the distinction naturally. Assume you need to move goods from one place to the next. You can choose a Ferrari or a cargo truck.

A Ferrari would certainly be quicker and would enable you to move goods in the blink of an eye. Yet, the amount of merchandise you can convey is little, and the fuel cost would be high. A cargo truck would be slower and would take longer to move goods. However, the amount it can convey is greater in contrast to a Ferrari. Additionally, it is more fuel efficient, so costs are lower.

The current advanced deep neural networks use algorithms, big data, and the computational power of the GPU to change this dynamic. Deep learning is used by the research community and in the industry to help solve many big data problems, such as computer vision, speech recognition, and natural language processing. Practical examples include:

- Vehicle, pedestrian and landmark identification for driver assistance

- Image recognition
- Speech recognition and translation
- Natural language processing
- Life sciences

The NVIDIA Deep Learning SDK provides high-performance tools and libraries to power innovative GPU-accelerated machine learning applications in the cloud, data centers, workstations, and embedded platforms.

Training deep learning model

When you prepare a deep learning model, two fundamental operations are performed:

- **Forward pass**
- **Reverse pass**

In forward pass, the input goes through the neural system and, subsequent to handling the information, a yield is created. In the reverse pass, we refresh the weights of the neural system based on the errors were received in the forward pass (figure 1.13).

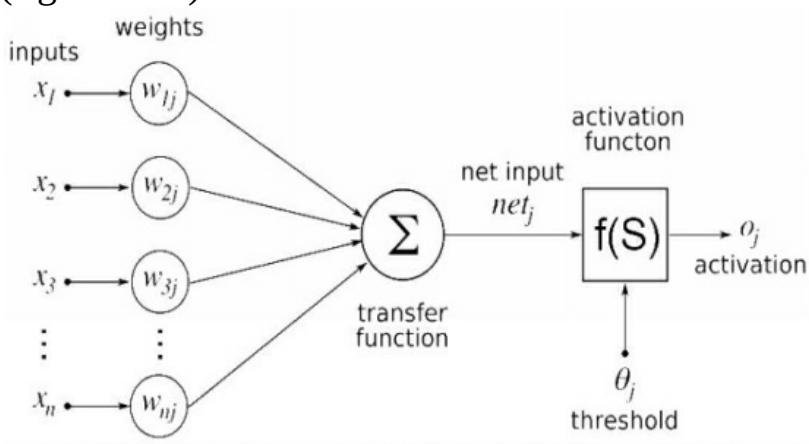


Figure 1.13: Deep learning models

Both of these operations are basically a matrix multiplication. A straightforward matrix multiplication can be expressed as in the picture underneath (figure 1.14).

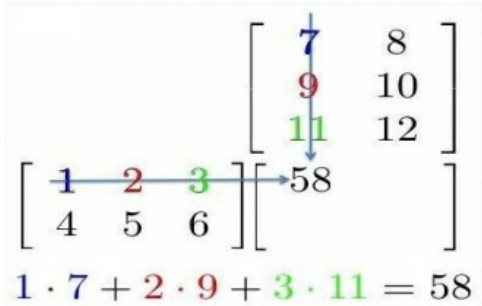


Figure 1.14: Matrix multiplication

Here, we can see that each operation in one line of the first array is increased with one segment of the second array. Thus, in a neural system, we can think of the first array as a contribution to the neural system, while the second array can be considered as weights of the system.

A three-way categorization

As portrayed before, deep learning alludes to a fairly wide class of machine learning strategies and models, with the trademark of utilizing many layers of non-linear data preparing that are variously leveled (hierarchical) in nature. Depending upon how the designs and strategies are to be utilized, e.g., combination/generation or recognition/ classification, one can extensively sort the vast majority of the work here into three noteworthy classes:

Deep networks for unsupervised or generative learning

These are proposed to catch highly arranged correlations of visible or obvious information for pattern analysis or synthesis purposes when no data about target class labels is available. Unsupervised component or representation learning in the literature alludes to this classification of deep systems. When utilized as part of the generative mode, it may likewise be planned to describe the joint statistical distribution of the visible information and the associated classes when accessible, and be dealt with as a feature of the visible data. In the last case, the utilization of Bayes, theorem can transform this sort of generative network into a discriminative one for learning.

Deep networks for supervised learning

These are proposed to specifically give discriminative power for pattern classification purposes, frequently by portraying the back distribution of classes adapted on the conditioned data. Target label data are constantly accessible in direct or backhanded structures for supervised learning. These

are also called discriminative deep systems.

Hybrid deep networks

The objective is discrimination, which is aided with the results of generative or also unsupervised deep networks. This can be improved through better optimization or/and regularization of the deep systems in classification (1.5.2). The objective can likewise be improved when discriminative criteria for supervised learning are utilized to evaluate the parameters in any of the deep generative or unsupervised deep arranges in the classification above.

Recurring methods of deep learning

Some recurring themes for deep learning are discussed below.

Dynamic programming for supervised learning / reinforcement learning

One recurring subject of deep learning is dynamic programming (DP), which can encourage task under specific assumptions. For instance, in supervised learning (SL) neural networks, backpropagation itself can be seen as a DP-inferred technique. In traditional reinforcement learning (RL), and in light of solid Markovian assumptions, DP-derived strategies can help to significantly diminish issue profundity. DP algorithms are likewise a basic necessity for frameworks that join the concepts of neural networks and graphical models, such as hidden Markov models (HMMs) and expectation maximization (EM).

Unsupervised learning (UL) facilitating SL and RL

Another recurring theme is the manner by which UL can encourage both SL and RL. UL is regularly used to encode raw incoming data, such as video or speech streams, into a shape that is more convenient for subsequent goal-directed learning. In particular, codes that describe the original data in a less redundant or more compact way can be fed into SL or RL machines, whose search spaces may thus become smaller (and CAPs shallower) than those necessary for dealing with the raw data. UL is closely connected to the topics of regularization and compression.

Learning hierarchical representations through deep SL, UL, RL

Numerous strategies for Good Old-Fashioned Artificial Intelligence (GOFAI) and, in addition, later ways to deal with AI and machine learning learn the

hierarchies of an ever-increasing number of dynamic information portrayals. For instance, certain techniques for syntactic pattern recognition, such grammar induction to find chains of formal standards to display observations. The mostly (un-) supervised Automated Mathematician/EURISKO ceaselessly learns ideas by joining already learnt ideas. Such hierarchical representation learning is also a repeating topic of deep learning neural networks for SL, UL-aided SL, and hierarchical RL. Often, abstract hierarchical portrayals are the normal results of information compression.

Occam's razor: compression and minimum description length (MDL)

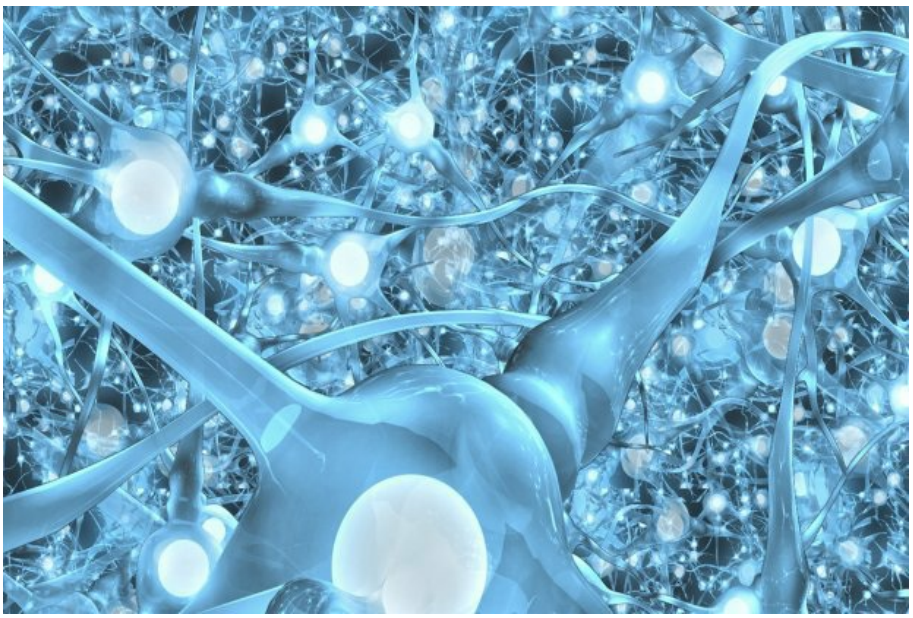
Occam's razor favors straightforward arrangements over complex ones. Given some programming language, the rule of Minimum Description Length (MDL) can be utilized to quantify the many-sided quality of an answer applicant by the length of the briefest program that processes it. A few strategies clearly consider program runtime; many consider just program with constant runtime, written in non-universal programming language. In the case of neural networks, the MDL standard recommends that low neural network weight difficulty compares to high neural network probability in the Bayesian view, and to high generalization performance, without over-fitting the training information. Numerous strategies have been proposed for regularizing neural networks, including scanning for solution computing and straightforward, low-intricacy SL NNs and RL NNs. This is firmly identified with certain UL techniques.

2. Introduction to Artificial Neural Networks

Objective of this Chapter

At the end of this chapter, the reader should have learnt:

- Artificial neural networks Architecture
- Artificial neural networks Applications



An Overview of Neural Networks

As we know, computers have processors and memory for computation. These perform complex computations in a short period of time, but they are not very adaptive. The adoption of neural networks introduced an alternative approach to problem solving than the use of conventional computers. Over the past decade, neural networks have received a great deal of attention. They are also widely used in pattern recognition because they have ability to respond to an unexpected input of patterns.

The human brain and the computer

Neural networks process data in the same way a human brain does. To understand the difference between a computer and a brain, we must know that a computer comprises 10^9 transistors with a switching time of 10^9 seconds. The brain contains 10^{11} neurons, but these only have a switching time of about 10^3 seconds. To help understand the human brain, the human nervous system can be viewed as a three-stage system, as shown in Figure 2.1. The major part is the brain (nerve), which receives continuous information, which it processes before making the appropriate decisions.

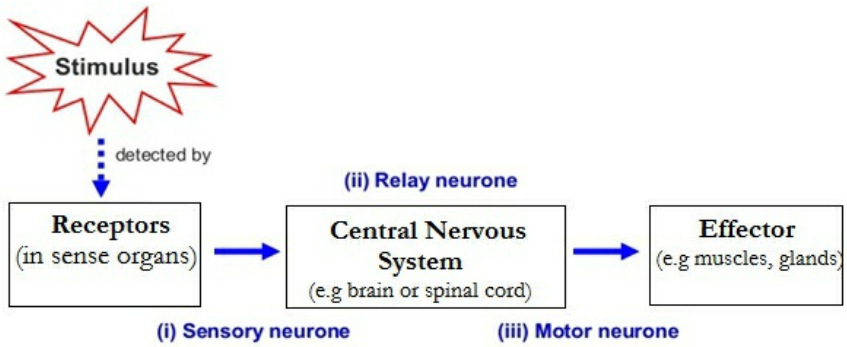


Figure 2.1: Block diagram representation of a nervous system

The arrows pointing from left to right indicate the forward transmission of information-bearing signals through the system, while and the arrows pointing right to left show the feedback. A comparison between a brain and a computer is given in table 2.1 below.

	Brain	Computer
No. of processing units	$\approx 10^{11}$	$\approx 10^{9}$
Type of processing units	Neurons	Transistors
Type of calculation	Massively parallel	Usual serial
Data storage	associative	address-based
Switching time	$\approx 10^{-3}$s	$\approx 10^{-9}$s
Possible switching operations	$\approx 10^{13}\frac{1}{s}$	$\approx 10^{18}\frac{1}{s}$
Actual switching operations	$\approx 10^{12}\frac{1}{s}$	$\approx 10^{10}\frac{1}{s}$

Comparison between brain and computer

First, we need to understand the structure of the brain in depth. The structural organization of the brain is shown in Figure 2.2 below.

- The synapses represent the most fundamental level, and depend on molecules and ions for their actions.
- A neural microcircuit alludes to a conglomeration of neurotransmitters sorted out into examples of a network to deliver a practical operation of intrigue.
- The neural microcircuits are grouped to form dendrite subunits within the dendrite trees of individual neurons.

- A whole neuron is about 100µm in size. It contains several dendrite subunits.
- The nearby circuits comprise neurons with comparable or diverse properties. Each circuit is around 1mm in size. The neural gatherings perform operations on attributes of a limited area in the cerebrum.
- The interregional circuits consist of pathways, segments and topographic maps, which include various districts situated in various parts of the cerebrum.
- The focal sensory system is the last level of unpredictability where the topographic maps intercede particular sorts of conduct.

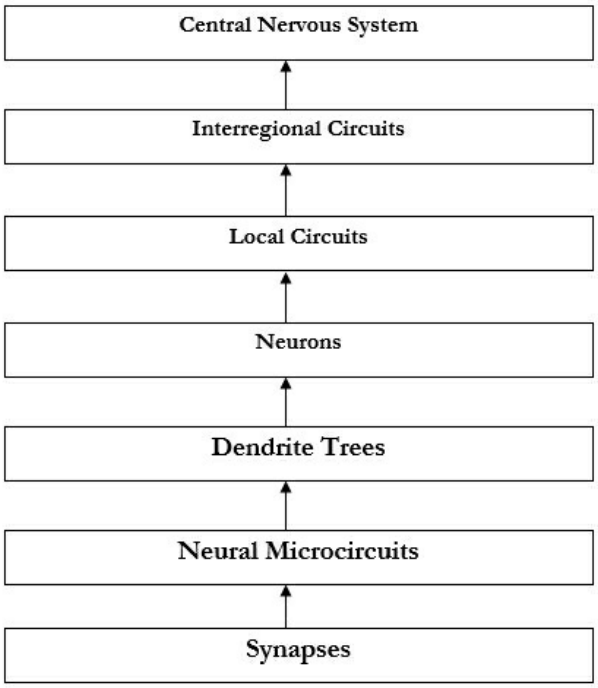


Figure 2.2: Structural organization of the brain

What is a Neural Network?

Artificial neural networks are also generally referred to as neural networks. This study has been motivated by the human brain, which is a complete and successful biological system working in different way from a traditional digital computer. The brain is a highly complex, non-linear, and parallel processor. It consists of numerous nerve cells of significant quality to work massively in parallel, and it also has ability to learn. There are nearly 100 billion neurons in the human brain (Figure 3 left). A typical neuron (Figure 2.3 right) consists of a cell body, dendrites, and an axon.

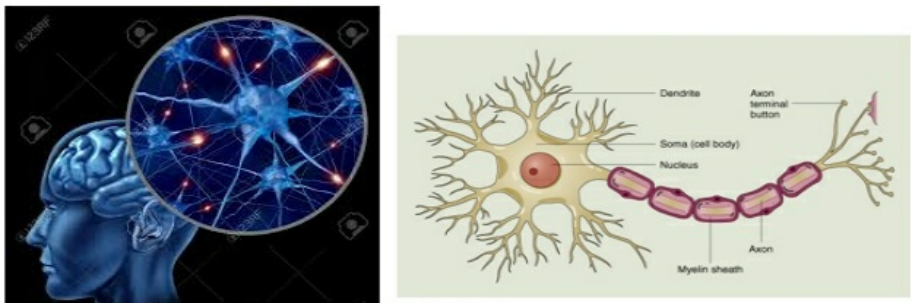


Figure 2.3: Human brain and Biological neuron

The building blocks of neural network are neurons. In technical system we

also refer to them as units or nodes. An artificial neural network is shown in Figure 2.4, and more precisely in Figure 2.5. Basically, each neuron:

- receives input from many other neurons,
- changes its internal state (activation) based on the current input,
- Sends one output signal to many other neurons, possibly including its input neurons (recurrent network).

Data is transmitted as a progression of electric motivations, supposed spikes. The recurrence and period of these spikes encode the data. In natural frameworks, one neuron can be associated with upwards of 10,000 other different neurons. More often than not, a neuron receives its data from different neurons.

Neural networks are able to learn by adapting their connectivity patterns so that the organism improves its behavior in terms of reaching certain (evolutionary) goals. The strength of a connection, whether it is excitatory or inhibitory, depends on the state of a receiving neuron’s synapses (as shown in the figure below). The neural network achieves learning by appropriately adapting the states of its synapses.

Artificial Neural Network

As we have seen earlier, artificial neural network (ANN) are commonly referred to as neural networks. Now, let us see some comprehensive definitions and pictorial representations (Figure 2.4).

- *A computing system is made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.*

- *An artificial neural network (ANN), frequently called a "neural network" (NN), is a numerical model or computational model in the manner of natural neural network; at the end of the day, it is an imitation of an organic neural framework.*

A typical neural network contains a large number of artificial neurons called units, arranged in a series of layers. The glossary offers more explanation about units and layers. Most neural networks are fully connected, which means that each hidden neuron is fully connected to every neuron in its previous layer (input) as well as to the next layer (output) layer.

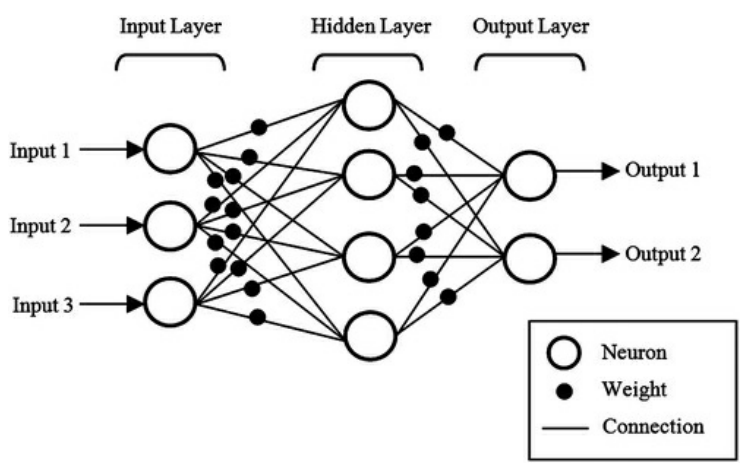


Figure 2.4: Artificial neural networks

Now we need to understand the main features of ANN. As shown in Figure 2.4, they have a three-layered architecture. The input layer is connected to hidden layer, which is connected to the output layer.

Input layer

The nodes of the input layer are passive in that they do not modify data. This

layer is responsible for receiving data from the external environment. They receive a single value.

Hidden layer

Each incentive from the information layer is copied and sent to all the shrouded hubs; this is called a completely interconnected structure. They perform calculations and exchange data from the information hubs to the yield hubs.

Output layer

This is responsible for computations and for transferring information from the network to the outside world.

Artificial Neuron structure

Neural networks or artificial neural networks are simulations on the computer that serve to perform certain tasks like clustering, classification, pattern recognition, etc. The figure below explains artificial neural networks in greater detail.

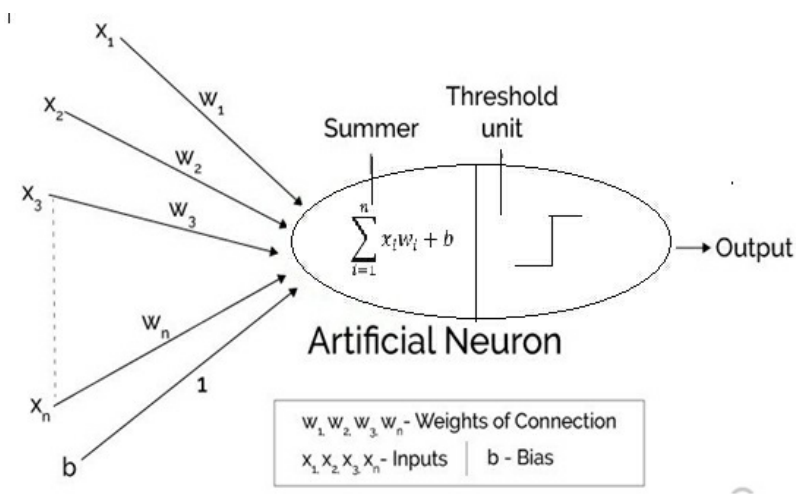


Figure 2.5 Artificial neural network

- The artificial neural network receives input from the external world in the form of a pattern and image in vector form. These inputs are mathematically designated by the notation x(n) for n number of inputs.
- Each input is multiplied by its weights equal to w(n). The weight represents the strength of the interconnection between neurons inside

the neural network. It is used by the neural network to solve a problem.

- The weighted inputs are all summed up inside the computing unit (artificial neuron).
- The sum corresponds to any numerical value ranging from 0 to infinity. In order to limit the response to arrive at a desired value, a threshold value is set up. For this, the sum is passed through an activation function, which transfers the function used to obtain the desired output.

Supervised and unsupervised neural network

The algorithm of a neural network can be either supervised or unsupervised.
A neural network is supervised if the desired output is already known. Let us consider the example of pattern association.
Suppose a neural net shall learn to associate the following pairs of patterns. The input patterns are decimal numbers, each represented in a sequence of bits. The target patterns are given in the form of binary values of the decimal numbers:

Input pattern	Target pattern
0001	001
0010	010
0100	011
1000	100

While learning, one of the input patterns is given to the input layer. This pattern is propagated through the networks (independent of its structure) to the output layer. The output layer generates an output pattern, which is then compared to the target pattern. Depending on the difference between the output and the target, an error value is computed.
This output error indicates the quality of this prediction, which can be controlled by the "imaginary supervisor". The greater the computed error value, the more the weight values will be changed.
Neural networks that learn unsupervised have no such target outputs. It is not possible to determine what the result of the learning process will look like.
During the learning process, the units (weight values) of such a neural network are "arranged" inside a certain range, depending on the given input

values. The goal is to group similar units close together in certain areas of the value range.

Applications of Neural Networks

Neural Network Applications can be grouped into the following categories.

Clustering

A clustering algorithm explores the similarity between patterns and places similar patterns in a cluster. The best known applications include data compression and data mining.

Classification/pattern recognition:

An example is distributing an informational design (like a manually written image) to one of many classes. This classification incorporates algorithmic usage, such as acquainted memory.

Function approximation

The task of function approximation is to find an estimate of the unknown function *f* subject to noise. Various engineering and scientific disciplines require function approximation.

Prediction/dynamical systems

The task is to forecast future values of time-sequenced data. Prediction significantly affects choice in emotionally supportive networks. The forecast, in contrast to the function approximation, considers the time factor. Here the framework is dynamic and may deliver distinctive outcomes for similar information in the light of the framework state (time).

The following is example of *facial recognition* and is just a mental image to help understand how massive level ANN works. At this stage you are just required to read the example and think of the NN architecture.

Let us assume that we want to build a network for a computer vision application. More specifically, our network is supposed to recognize faces and facial expressions. This is an example that has actually been implemented. All information, program coded, data, etc. can be found at *http://www.cs.cmu.edu/afs/cs.cmu.edu/user/mitchell/ftp/faces.html*

The goal is to classify camera images of faces of various people with various facial expressions and poses. The images of 20 different people were collected, with up to 32 images per person. The following variables were introduced.

- expression (happy, sad, angry, neutral)
- direction of looking (left, right, straight ahead, up)
- sunglasses (yes or no)

In total, 624 grayscale images were collected, each with a resolution of 30 by 32 pixels and intensity values between 0 and 255. The network presented here only has the task of determining the face pose (left, right, up, straight) shown in an input image.
It uses

- 960 input units (one for each pixel in the image),
- 3 hidden units
- 4 output neurons (one for each pose)

Each output unit receives an additional input, which is always 1.

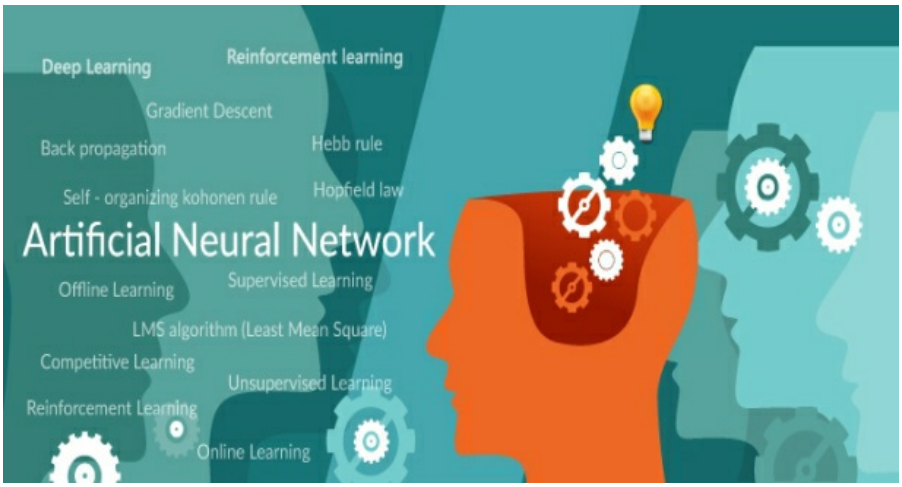
By varying the weight of this input, the backpropagation algorithm can adjust an offset for the net input signal.
Their values are indicated by brightness (ranging from black = -1 to white = 1). Each 30 by 32 matrix represents the weights of one of the three hidden layer units. Each row of four squares represents the weights of one output neuron (three weights for the signals from the hidden units and one for the constant signal 1). After training, the network is able to classify 90% of new (non-trained) face images correctly.

3. The Basics of Artificial Neural Networks

Objective of this Chapter

At the end of this chapter, the reader should have learnt:

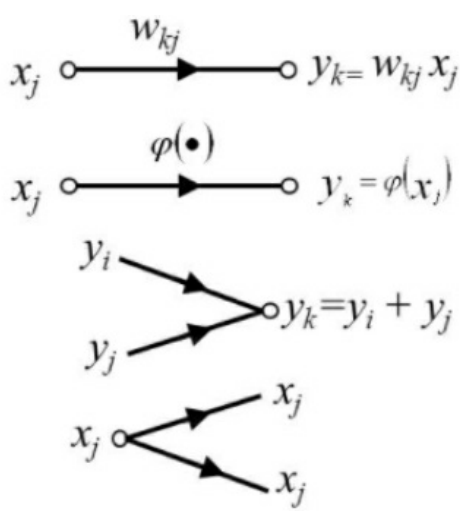
- Artificial neural networks Architecture
- Artificial neural networks Applications



Neural Networks Viewed as Directed Graphs

A signal-flow graph in a network of directed links that are interconnected at certain points, which are known as nodes. A node j has an associated node signal x_j. A typical directed link originates at node j and terminates at node k. It has an associated transmittance that specifies the manner in which the signal y_k at node k depends on the signal x_j at node j. This flow of signals must follow three basic rules.

- Signals flow only according to the direction specified by arrows. Two types of links may be distinguished, namely:
  - Synaptic links
  - Activation links
- A node signal equals the algebraic sum of all signals, shown in the following figures.
- The signal is transmitted to each outgoing link originating from that node. Transmission is independent of the transfer functions of the outgoing links.



A directed graph is called complete because it describes not only the signal flow from node to node but also the signal flow inside each node. A few details are reduced in comparison to Figure 6. Such a directed graph is called partially complete, as shown in Figure 3.1 below.

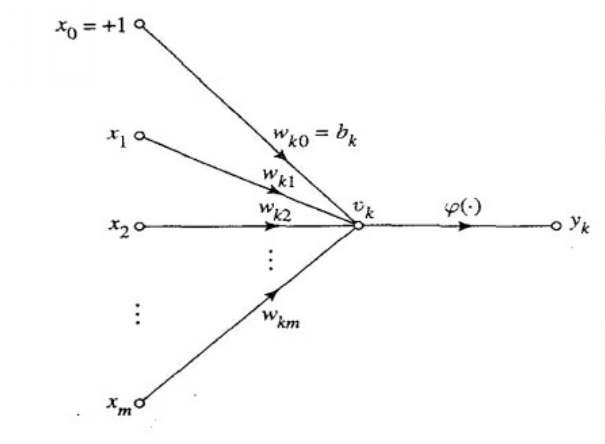


Figure 3.1: Partially complete graph

Thus, we have come to know three ways to represent neural networks.

- Block diagram
- Signal-flow graph
- Architectural graph (partial)

The Perceptron

These are the oldest and simplest artificial neural networks known were defined by Rosenblatt (1958) as a machine that learns using examples to assign input vectors (samples) to different classes using a linear function of the inputs. Minsky and Papert (1969) describe the perceptron as “a stochastic gradient descent algorithm that attempts to linearly separate a set of n dimensional training data”. We use the word perceptron in the former sense as a machine and refer explicitly to the “perceptron learning algorithm”.

The most basic form of an activation function is a simple binary function that has only two possible results (shown below).

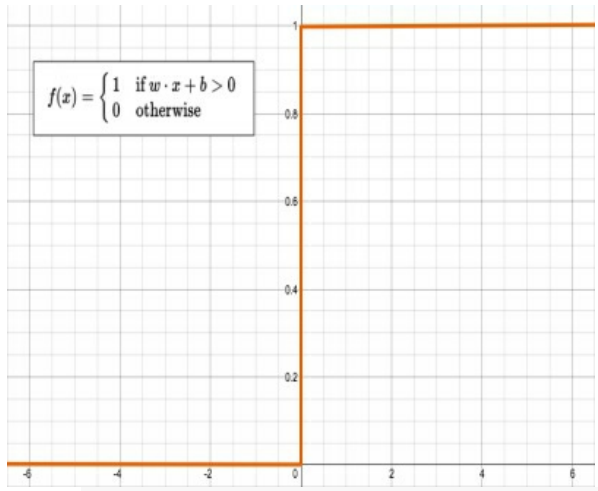


Figure 3.2: Activation function in the perceptron

This function returns 1 if the input is positive or zero, and 0 for any negative input. A neuron whose activation function is a function like this is called a *perceptron*. In its simplest form, a perceptron has single output whose values determines to which of two classes each input pattern belongs. The node applies a step function to the net weighted sum of its inputs. The input pattern is considered to belong to one class or the other, depending on whether the node output is 0 or 1.

Types of Activation Function

The neuron of a specific layer receives a similar kind of enactment work. In all cases, non-straight enactment capacities are utilized. The different sorts of enactment capacities are utilized as a part of a neural network. The three sorts of initiation capacities are examined below.

- Threshold function
- Piecewise-linear function
- Sigmoid function

Threshold function

The function is defined as

$$\gamma(v) = 1, if\ v \geq 0$$

$$\gamma(v) = 1, if\ v < 0$$

This form of a threshold function is also called as Heaviside function.

Correspondingly, the output of neuron k is expressed as

$$y_{k=}1, if\ v_k\ \geq 0$$

$$y_k = 0, if\ v_k < 0$$

Where

$$\mathrm{V}_k = \sum_{j=1}^{m} \left(W_{kj} X_j + \mathrm{bk}\right)$$

This model is additionally called the McCullouch-Pitts demonstrate. In this model, the yield of a neuron is 1 if the prompted neighborhood field of that neuron is nonnegative and 0 generally. This announcement depicts the all-or-none property of the model.

Piecewise-function

The activation function is defined as

$$\gamma = \begin{cases} 1, v \geq +\frac{1}{2} \\ v, +\frac{1}{2} > v > -\frac{1}{2} \\ 0, v \leq -\frac{1}{2} \end{cases}$$

In this equation, the amplification factor inside the linear region of operation is assumed to be a unit.

- A linear combiner arises if the linear region of operation is maintained without running into a situation.
- The piecewise-linear function reduces to a threshold function if the amplification factor of the linear region is made infinitely large.

Sigmoid function

This is the most common form of activation function used in the construction of artificial neural networks. It is defined as a strictly increasing function that exhibits a graceful balance between linear and nonlinear behavior. A logistic function is an example of a sigmoid function.

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

Simplified form (figure is shown below)

$$S(net) = \frac{1}{1+e^{(-net)}}$$

$$S'(x) = \frac{dS(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{1+e^{-x}-1}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} - \frac{1}{(1+e^{-x})^2}$$

$$= S(x)(1-S(x))$$

Adaline Learning Algorithm

The Adaline utilizes slope drop to decide the weight vector that prompts an insignificant mistake. The Error is characterized as the MSE between the neuron's net info net_j and its coveted yield dj (= class(ij)) over all preparation tests ij.

The idea is to pick tests in arbitrary request and perform a (moderate) angle plummet in their individual blunder capacities. This method permits incremental learning, i.e., a refining of the weights as additionally preparing tests are included.

$$E = (d_j - net_j)^2$$

$$\frac{\partial E}{\partial w_k} = 2(d_j - net_j)\frac{\partial}{\partial w_k}(-net_j)$$

$$= 2(d_j - net_j)\frac{\partial}{\partial w_k}\left(-\sum_{i=0}^{n} w_i i_{i,j}\right)$$

$$= -2(d_j - net_j)i_{k,j}$$

The Adaline uses a gradient descent to determine the weight vector that leads to minimal error. The gradient is then given by

$$\begin{bmatrix} \frac{\partial E}{\partial w_0} \\ \vdots \\ \frac{\partial E}{\partial w_n} \end{bmatrix} = -2(d_j - net_j)\mathbf{i}_j$$

For gradient descent, D **w** should be a negative multiple of the gradient:

$$\Delta \mathbf{w} = \eta(d_j - net_j)\mathbf{i}_j \text{ , with positive step-size parameter } \eta$$

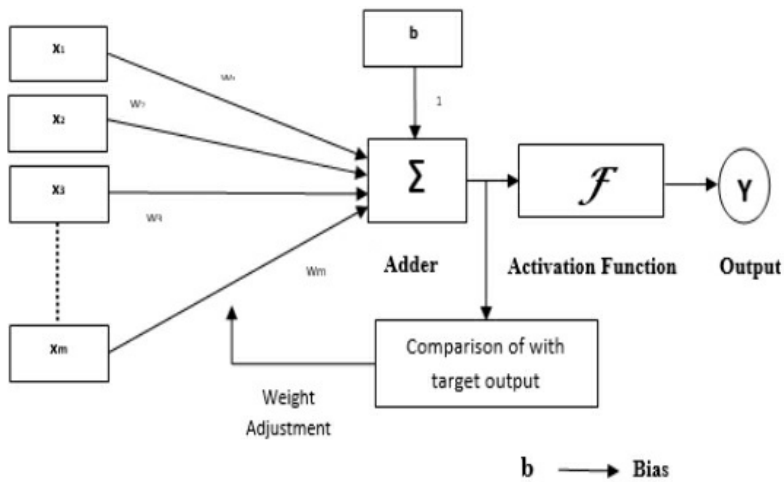


Figure 3.3: Adaline learning algorithm

The Widrow-Hoff Delta Rule

In the original learning rule

$$\Delta \mathbf{w} = \eta(d_j - net_j)\mathbf{i}_j$$

Longer input vectors result in greater weight changes, which can cause problems if there are extreme differences in vector length in the training set. Widrow and Hoff (1960) suggested the following modification to the learning rule:

$$\Delta \mathbf{w} = \eta(d_j - net_j)\frac{\mathbf{i}_j}{\| \mathbf{i}_j \|}$$

The Delta Rule and Learning Rate

To derive into the exact algorithm to train our neuron, we make a quick note on hyper parameters. In addition to the weight parameters defined in our neural network, learning algorithms also need a couple of additional parameters to carry out the training process. One of these so-called hyper parameters is the ***learning rate.***

In practice, at each step of moving perpendicular to the line, we need to determine how far we want to walk before recalculating our new direction. This distance needs to depend on the steepness of the surface. Why? The closer we are to the minimum, the shorter we want to step forward. We know we are close to the minimum, because the surface is a lot flatter, so we can use the steepness as an indicator of how close we are to the minimum. However, if our error surface is rather shallow, training can potentially take a large amount of time. As a result, we often multiply the gradient by a factor , the learning rate. The selection of the learning rate is a tough problem. As we have just discussed, if we pick a learning rate that's too small, we risk taking too long during the training process. However, if we pick a learning rate that's too big, we'll mostly likely start diverging away from the minimum.

Now we are finally ready to derive the delta rule for training our linear neuron. In order to calculate how to change each weight, we evaluate the gradient, which is essentially the partial derivative of the error function with respect to each of the weights. In other words, we want:

$$\Delta w_k = -\epsilon \frac{\partial E}{\partial w_k}$$

$$= -\epsilon \frac{\partial}{\partial w_k}\left(\frac{1}{2}\sum_i \left(t^{(i)} - y^{(i)}\right)^2\right)$$

$$\sum_i \epsilon \left(t^{(i)} - y^{(i)}\right) \frac{\partial y_i}{\partial w_k}$$

$$\sum_i \epsilon x_k^{(I)} \left(t^{(i)} - y^{(i)}\right)$$

Applying this method of changing the weights, for every iteration we are finally able to utilize the gradient descent.

Gradient Descent with Sigmoid Neurons

In this segment, we will deal with training neurons and neural networks that utilize nonlinearities. We use the sigmoid neuron as a model. For ease, we assume that the neurons do not use a bias term, although our analysis easily extends to this case. We merely need to assume that the bias is a weight on an incoming connection whose input value is always one.

Let's recall the mechanism by which logistic neurons compute their output value from their inputs:

$$z = \sum_k w_k \; x_k$$

$$y = \frac{1}{1 + e^{-z}}$$

The neuron computes the weighted sum of its inputs, the logit z. It then feeds this logit into the input function to compute y, its final output. Fortunately for us, these functions have very nice derivatives, which make learning simple. For learning, we want to compute the gradient of the error function with respect to the weights. To do so, we start by taking the derivative of the logit with respect to the inputs and the weights.

$$\frac{\partial z}{\partial w_k} = x_k$$

$$\frac{\partial z}{\partial x_k} = w_k$$

Also, and to a certain extent unpredictably, the derivative of the output with respect to the logit is quite simple if you cut across it in terms of the output.

$$\frac{dy}{dz} = \frac{e^{-z}}{(1 + e^{-z})^2}$$

$$= \frac{1}{1 + e^{-z}} \frac{e^{-z}}{1 + e^{-z}}$$

$$= \frac{1}{1 + e^{-z}} \left(1 - \frac{1}{1 + e^{-z}}\right)$$

$$= Y(1 - Y)$$

We apply the chain rule to receive the derivative of the output with respect to

each weight:

$$\frac{\partial y}{\partial w_k} = \frac{dy}{dz}\frac{\partial z}{\partial w_k} = x_k y(1-y)$$

Putting all of this together, we can now compute the derivative of the error function with respect to each weight:

$$\frac{\partial E}{\partial w_k} = \sum_i \frac{\partial E}{\partial y^{(i)}}\frac{\partial y^{(i)}}{\partial w_k} = -\sum_i x_k^{(i)} y^{(i)}\left(1-y^{(i)}\right)\left(t^{(i)}-y^{(i)}\right)$$

Thus, the final rule for modifying the weights becomes:

$$\Delta w_k = \sum_i \epsilon x_k^{(i)} y^{(i)}(1-y^{(i)})(t^{(i)}-y^{(i)})$$

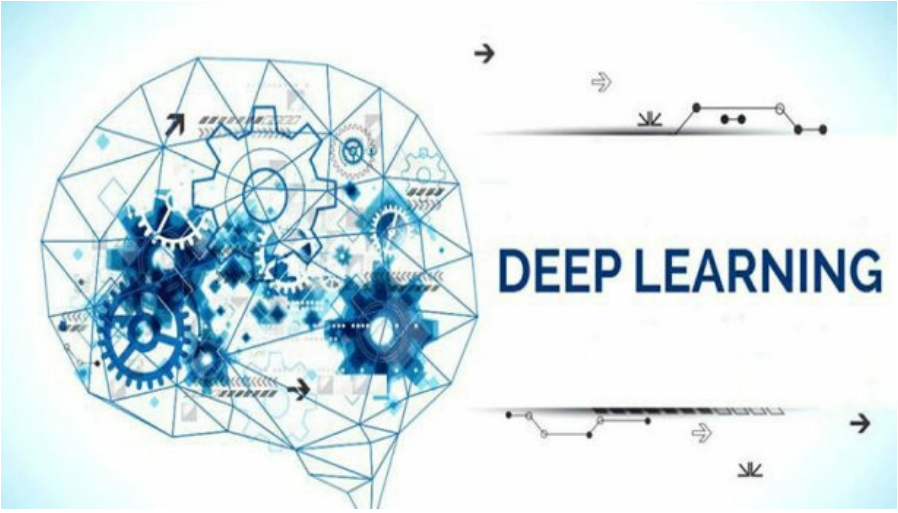
The new modification rule is just like the delta rule, except with extra multiplicative terms included to account for the logistic component of the sigmoid neuron.

4. Deep Learning Evolution and Recurring Methods

Objective of this Chapter

At the end of this chapter, the reader should have learnt:

- Definition of Deep Learning and its History
- A three-way categorization
- Recurring methods of deep learning



The idea of deep learning isn't new; it has been around for two or three years now. Yet, with all the hype these days, deep learning is getting more consideration. Deep learning permits computational models that consist of various preparing layers to learn portrayals of information with various levels of reflection. These techniques have drastically enhanced the cutting edge in speech and visual recognition, and object detection as well as numerous different fields, such as drug disclosure and genomics. Deep learning comprises finding the perplexing structure in vast informational collections by utilizing a back-propagation calculation to demonstrate how a machine should change its inside parameters, which are utilized to register the portrayal of each layer from the portrayal in the past layer. Deep convolution nets have realized leaps forward in preparing pictures, video, discourse, and sound, though intermittent nets have also shed light on successive information, for example, content and discourse.

Before you begin pondering deep learning, essentially, it's imperative to first completely comprehend the idea of a neural network. A neural network passes information through interconnected layers of hubs, arranging data and qualities of a layer before passing the outcomes on to different hubs in consequent layers. The contrast between a neural system and a profound learning system is dependent upon the quantity of layers: A fundamental neural system may have a few layers, while a deep learning system may have handfuls or even hundreds.

History of Deep Learning

Deep learning is a popular topic today, as organizations attempt to discover how to utilize progressed computational strategies to discover helpful data inside enormous swaths of information. While the field of artificial intelligence is decades old, leaps forward in the field of simulated neural systems are driving the sudden increase in deep learning.

Attempts to make computerized reasoning go back decades. In the wake of World War II, the English mathematician and code breaker Alan Turning penned a definition for genuine computerized reasoning. He named it the Turing Test, whereby a conversational machine would need to persuade a human that they were conversing with another human.

After 60 years, a computer finally breezed through the Turing Test in 2014, when automated bot created by the University of Reading named "Eugene" persuaded 33% of the judges gathered by the Royal Society in London that it

was genuine. It was the first occasion that the 30% limit had been surpassed.

From that point forward, the field of deep learning and AI has begun expanding as computers are moving closer nearer to conveying human-level capacities. Consumers interact with a variety of bots like Apple's Siri, Amazon's Alexa, and Microsoft's Cortana, which utilize regular language and machine in order to work out how to answer questions.

Presently, multiple organizations are hoping to use their huge informational indexes as a preparation ground to create better AI programs that can associate with the world in regular ways, while concentrating helpful data in ways that have not yet been determined. Scientists have discovered that amix of cutting-edge neural systems; prepared accessibility of immense masses of information, and, to great degree, effective circulated GPU-based frameworks can give us the plans to make shrewd, self-learning machines that can begin to equal people in their cognition.

A definitive wellspring of deep learning's energy is the manner in which it empowers computers to separate the contribution of essential bits and after that construct a deliberation layer above it that relates to our impression of reality.

Deep Learning Timeline

Below is a timeline that will help you see how deep learning has been established and the growth it has accumulated along the way. This timeline will give you insight to some of the biggest developments in deep learning technology and the people who played a key role in establishing these developments. This will help you understand how long deep learning has been around for and just how long it has come in a few short decades.

1965: Akexey Ivakhnenko and Lapa publish a working learning algorithm that uses supervised, deep, feedforward, multilayer perceptrons.

1971: A paper is published that describes the algorithm as an 8-layer deep network that is trained by the group method of data handling algorithm.

1980: A deep learning algorithm is published as the “Neocognitron” by Kunihiko Fukushima.

1986: Rina Dechter introduces the phrase “Deep Learning” and the concept becomes an official topic of interest for computer scientists.

1989: The standard back propagation algorithm is applied to a deep neural network by Yahn LeCun who uses it as a system to read and recognize handwritten ZIP codes on pieces of mail.

1990: Igor Aizenberg coins the phrase “Artificial Neural Networks”.

1991: The same piece of machinery used to read ZIP codes on mail was advanced and developed on so that it could begin recognizing 3-D objects. This was completed by matching 2-D images with handcrafted 3-D object models.

1992: Cresceptron was introduced by Weng. This was a publication that was used to perform 3-D recognition and could work even in a cluttered place where there wasn’t a single isolated object to recognize. Similar to Neocognitron that was introduced in 1980, Cresceptron was developed using multiple layers. Unlike Neocognitron, it did not require supervision or human interaction and manipulation to complete various functions. In other words, it could perform many functions and operations on its own.

1994: A multi-layer Boolean neural network was introduced by Andre C. P. L. F. de Carvalho, Fairhurst and Bisset. The neural network module was independently trained and created a completely weightless neural network that was composed of self-organizing feature extractions.

1995: A network comprised of six fully-connected layers was trained in 2 days by Brendan Frey. The network had several hundred hidden units within’ it that were each trained during this period. He proved that using the wake-sleep algorithm you could train the network to do several different functions. At the time, however, the speed was rather slow.

1997: Over the years the deep learning network has struggled with what was called the “vanishing gradient problem”. This prevented machine learning systems from retaining information in gradient-based learning methods. In 1997 Hochreiter and Schmidhuber published the Long short-term memory (LSTM) network which was capable of avoiding the vanishing gradient problem. It did so by maintaining memories of events that had happened several times over. This was important in the development of speech recognition.

2006: Hinton and Salakhutdinov successfully shared how many-layer

feedforward neural network systems were capable of being pre-trained one layer at a time. Each layer was treated as an unsupervised but restricted machine. Afterward, they used supervised backpropagation to fine-tune the system so that it worked the way they desired it to.

2009: Hinton and Li Deng worked together to apply deep learning algorithms to speech recognition. Together they co-organized a workshop that took place in late 2009 called "Workshop on Deep Learning for Speech Recognition". Here, they learned how they could deepen the generative models of speech and potentially use more capable hardware and large-scale data sets so that deep neural nets would become possible.

2010: Computer scientists and researchers extend deep learning to a larger vocabulary speech recognition.

2012: The "Merck Molecular Activity Challenge" was won by Dahl and his team when they used a multi-task deep neural network in order to predict the biomolecular target of a drug.

2014: The "Tox21 Data Challenge" was won by a team lead by Hochreiter who used deep learning as a system to detect off-target and toxic effects of environmental chemicals that can be found in nutrients, drugs, and various household products.

There have been many discoveries and advancements in deep learning over the years. The concept was originally introduced in 1965 and since then scientists and researchers have developed on it so much so that there is now a real and impending possibility that a true and complete artificial intelligence will soon exist. At this point, we are no longer concerned with "if" it will happen, rather we are wondering "when". It is clear that the research and science has shown that deep learning has the power to complete many incredible actions, each of which will contribute to an eventual artificial intelligence that will be able to completely function, learn, and advance on its own without requiring human input, interaction, or manipulation

Deep Learning and Artificial Neural Networks

Artificial neural networks (ANNs) are the basis for deep learning, but the development of deep learning networks goes far beyond artificial neural networks. Since the development of artificial neural networks, scientists and

researchers have been focusing on developing and using deep neural networks. These systems are extremely similar to artificial neural networks; however, they have some key differences. They are the future of deep learning, and as the deep learning system progresses it is expected that deep neural networks will eventually replace artificial neural networks entirely.

As you know, artificial neural networks are shallow networks that have been developed in an attempt to mimic the animal brain. However, they do have many disadvantages, part of which is that they are large and require many systems in order to work, and that they can only complete a small number of specialized tasks. As far as deep learning goes, the goal is to eventually create an entirely pure artificial intelligence that requires little to no input from a human in order to function effectively and learn and grow on its own. In order to attempt to draw us closer to that, the deep neural network was created.

The deep neural network is an advanced version of the artificial neural network. It features smaller units and less “stuff”, but has the ability to perform the same complex functions as artificial neural networks. The architects of these networks have focused on a few basic approaches as an opportunity to develop a newer and more useable program that allows the deep learning network to perform effectively and efficiently whilst taking up significantly less space.

To understand the benefits and purpose of this activity, one might consider the evolution of computers themselves. A few decades ago, computers were massive machines that took up entire rooms and operated extremely slowly. Now, however, they are small and speedy devices that can fit inside of our pocket. Just like computers have evolved in this way, scientists and researchers are seeking to get artificial neural networks to become significantly smaller so that they are more useable and can operate more efficiently. Naturally, this requires them to refine the techniques and develop new systems that will allow the device to complete the same functions with higher quality results while also taking up less space. This is why the deep neural network was created.

Why is Deep Learning Called Deep?

Recently, the term 'deep learning' has been frequently used. Deep learning, in such a vision, could change any industry. Thus, there are some unique

changes that will happen and that will enable computer vision to truly work. Deep learning exceeds expectations in areas where the data sources (and even the yield) are simple. This means that they are not a couple of bytes in a forbidden organization but rather are pictures, records, or documents.

Deep learning is called deep due to the structure of the ANNs (Artificial Neural Networks). Forty years ago, neural systems were just 2 layers deep as it was not computationally conceivable to construct bigger systems. At present, it is common to have neural systems with 10+ layers, and even 100+ layer ANNs are being attempted. You can in every practical sense stack layers of neurons on top of each other. The most minimal layer takes the raw information, like pictures, content, sound, and so forth, and afterward each neuron stores some data about the information they pass on. Every neuron in the layer sends information up to the following layers of neurons, which take in a more applied rendition of the information from underneath. So, the higher you go up, the more dynamic the highlights that you learn. Figure 4.1 below shows 5 layers, in which 3 are concealed layers.

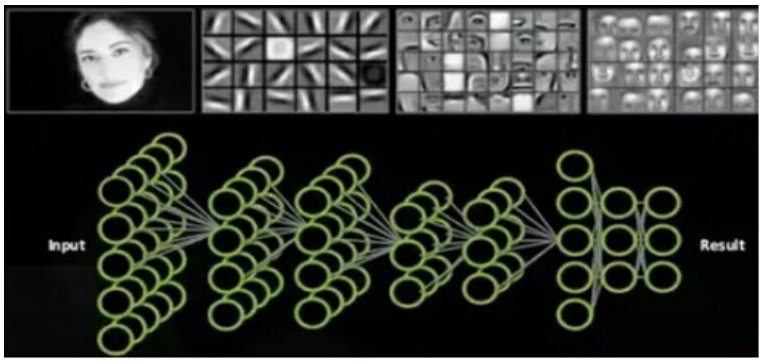


Figure 4.1: Deep learning

Perceptron: An early deep learning algorithm

One of the earliest supervised training algorithms was the perceptron, a fundamental neural system building block. Let us assume we have n focuses in the plane, named '0' and '1'. We're given another point and we need to figure its name (this is much the same as the "dog" and "not dog" situation above). How would we do it?

One approach may be to look at the nearest neighbor and restore that point's name. Yet, a marginally more sophisticated method would be to choose a line that best isolates the named information and utilize that as your classifier.

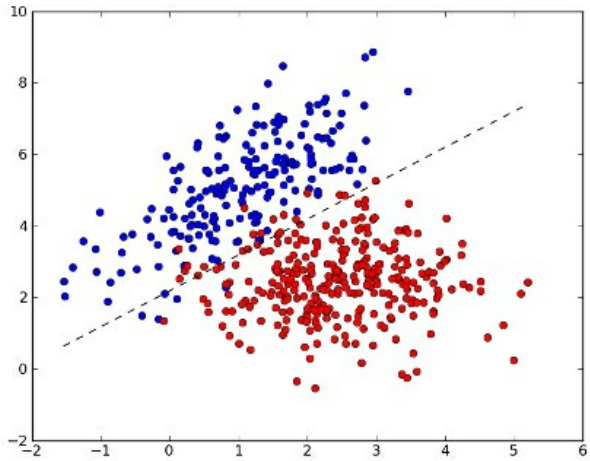


Figure 4.2: Linear classifier

For this situation, each piece of information would be called a vector x = (x_1, x_2) and our function would be something like "'0' if beneath the line, '1' if above".

To express this numerically, let our separator be characterized by a vector of weight w and a vertical balance b. At that point, our function would consolidate the sources of data (input) and weights with a weighted whole transfer function:

$$\mathbf{f(x)=x.w+b}$$

The consequence of this exchange function would then be encouraged into an actuation function to deliver a marking. In the case above, our activation work was an edge cutoff (e.g., 1 if more than some value):

$$h(x) = \begin{cases} 1 & if\ w.x + b > 0 \\ 0 & otherwise \end{cases}$$

Training a perceptron

The preparation of the perceptron consists of using different preparing tests and figuring the yield for each of them. After each example, the weights w are balanced in a path in order to limit the yield error, characterized as the distinction between the coveted (target) and the genuine yields. There are other error functions, similar to the mean square error, however the fundamental standard of preparing continues as before.

A three-way categorization

As portrayed before, deep learning alludes to a fairly wide class of machine learning strategies and models, with the trademark of utilizing many layers of non-linear data preparing that are variously leveled (hierarchical) in nature. Depending upon how the designs and strategies are to be utilized, e.g., combination/generation or recognition/ classification, one can extensively sort the vast majority of the work here into three noteworthy classes:

Deep networks for unsupervised or generative learning

These are proposed to catch highly arranged correlations of visible or obvious information for pattern analysis or synthesis purposes when no data about target class labels is available. Unsupervised component or representation learning in the literature alludes to this classification of deep systems. When utilized as part of the generative mode, it may likewise be planned to describe the joint statistical distribution of the visible information and the associated classes when accessible, and be dealt with as a feature of the visible data. In the last case, the utilization of Bayes, theorem can transform this sort of generative network into a discriminative one for learning.

Deep networks for supervised learning

These are proposed to specifically give discriminative power for pattern classification purposes, frequently by portraying the back distribution of classes adapted on the conditioned data. Target label data are constantly accessible in direct or backhanded structures for supervised learning. These are also called discriminative deep systems.

Hybrid deep networks

The objective is discrimination, which is aided with the results of generative or also unsupervised deep networks. This can be improved through better optimization or/and regularization of the deep systems in classification. The objective can likewise be improved when discriminative criteria for supervised learning are utilized to evaluate the parameters in any of the deep generative or unsupervised deep arranges in the classification above.

Recurring methods of deep learning

Dynamic programming for supervised learning / reinforcement

learning (SL/RL)

One recurring subject of deep learning is dynamic programming (DP), which can encourage task under specific assumptions. For instance, in SL neural networks, backpropagation itself can be seen as a DP-inferred technique. In traditional RL, and in light of solid Markovian assumptions, DP-derived strategies can help to significantly diminish issue profundity. DP algorithms are likewise a basic necessity for frameworks that join the concepts of neural networks and graphical models, such as hidden Markov models (HMMs) and expectation maximization (EM).

Unsupervised learning (UL) facilitating SL and RL

Another recurring theme is the manner by which UL can encourage both SL and RL. UL is regularly used to encode raw incoming data, such as video or speech streams, into a shape that is more convenient for subsequent goal-directed learning. In particular, codes that describe the original data in a less redundant or more compact way can be fed into SL or RL machines, whose search spaces may thus become smaller (and CAPs shallower) than those necessary for dealing with the raw data. UL is closely connected to the topics of regularization and compression.

Learning hierarchical representations through deep SL, UL, RL

Numerous strategies for Good Old-Fashioned Artificial Intelligence (GOFAI) and, in addition, later ways to deal with AI and machine learning learn the hierarchies of an ever-increasing number of dynamic information portrayals. For instance, certain techniques for syntactic pattern recognition, such grammar induction to find chains of formal standards to display observations. The mostly (un-) supervised Automated Mathematician/EURISKO ceaselessly learns ideas by joining already learnt ideas. Such hierarchical representation learning is also a repeating topic of deep learning neural networks for SL, UL-aided SL, and hierarchical RL. Often, abstract hierarchical portrayals are the normal results of information compression.

Occam’s razor: compression and minimum description length (MDL)

Occam's razor favors straightforward arrangements over complex ones. Given some programming language, the rule of Minimum Description

Length (MDL) can be utilized to quantify the many-sided quality of an answer applicant by the length of the briefest program that processes it. A few strategies clearly consider program runtime; many consider just program with constant runtime, written in non-universal programming language. In the case of neural networks, the MDL standard recommends that low neural network weight difficulty compares to high neural network probability in the Bayesian view, and to high generalization performance, without over-fitting the training information. Numerous strategies have been proposed for regularizing neural networks, including scanning for solution computing and straightforward, low-intricacy SL NNs and RL NNs. This is firmly identified with certain UL techniques.

Fast graphics processing units (GPUs) for deep learning in neural networks

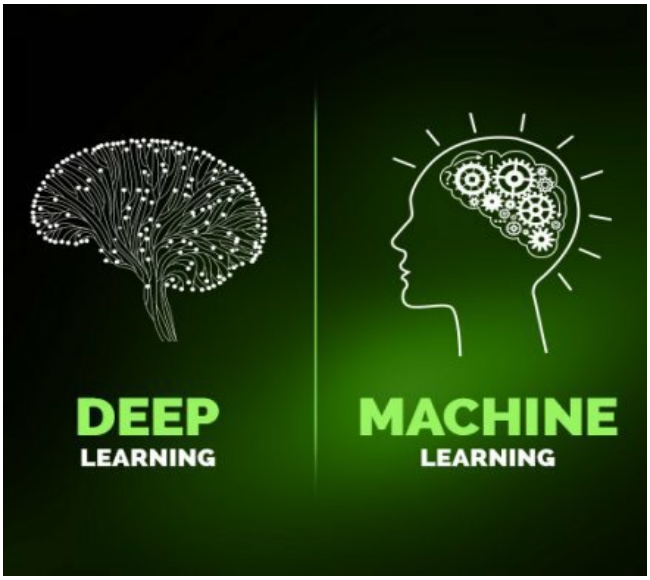
GPUs are generally utilized for computer games, an enormous and focused market that has driven down equipment costs. GPUs exceed expectations at the quick matrix and vector multiplication required to persuade/ show or give virtual substances as well as for neural network training, where they can accelerate learning by a factor of 50.

5. Relationship between machine learning and deep learning

Objective of this Chapter

At the end of this chapter, the reader should have learnt:

- Difference between machine learning and deep learning
- Where deep learning and machine learning are applied
- Future trends



Regular machine learning methods were constrained in their capacity to process normal information within their simple frame. For quite some time, developing a pattern recognition or machine-learning framework required cautious building and impressive space capacity to outline an element extractor that changed the raw information, (for example, the pixel esteems of a picture) into a reasonable inner portrayal or highlight vector from which the learning subsystem, regularly a classifier, could distinguish or characterize patterns in the information.

Machine learning likewise encourages us to discover answers to some issues in vision, speech recognition, and mechanical autonomy. Let us take the case of recognizing faces: This is an assignment we do easily; we consistently recognize relatives and companions by looking at their faces or from their photos, regardless of differences in posture, lighting, hairdo, etc. In any case, we do it unknowingly and cannot clarify how we do it. Since we are not ready to clarify our mastery, we can't compose the computer program. In the meantime, we realize that a picture of a face isn't only an irregular accumulation of pixels; a face has structure. It is symmetrical. There are the eyes, the nose, and the mouth, situated in specific places on the face. Every individual's face is an example comprising a specific blend of these. By examining a test confronted with pictures of a man, a learning program catches the example particular to that individual and afterward recognizes by checking for this example in a given picture. This is one case of pattern recognition.

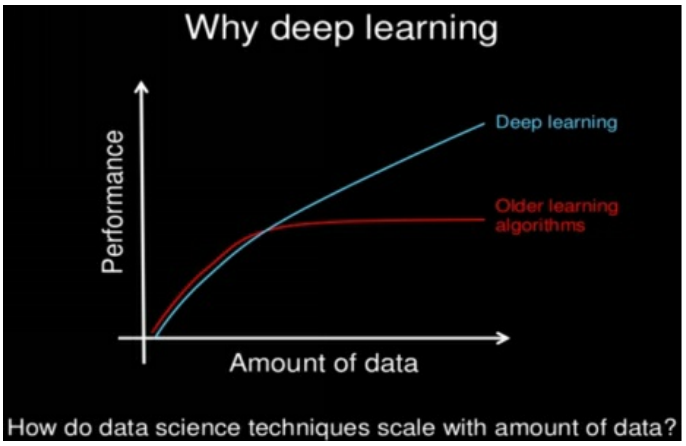
Machine learning is modifying computers to upgrade an execution basis utilizing illustration information or past experience. We have a model characterized with up to a few parameters, and learning is the execution of a computer program to streamline the parameters of the model utilizing the preparation information or past experience. The model might be prescient to make predictions for the future, or a graphic to pick up learning from information, or both.

Comparison of deep learning and machine learning

We will take a few vital points and contrast the two techniques of machine learning and deep learning.

Data dependencies

The most imperative contrast between deep learning and conventional machine learning is its performance as the size of data increases. When there is little data, deep learning calculations don't play out that well. This is because deep learning calculations require a lot of data to comprehend it well. Then again, conventional machine learning calculations with their carefully assembled rules win in this situation. The figure below demonstrates this reality.



Hardware dependencies

Deep learning algorithms rely a lot on high-end machines, as opposed to traditional machine learning algorithms, which can work on low-end machines. This is because the necessities of deep learning algorithms include GPUs, which are an important part of its working. Deep learning algorithms intrinsically do a great amount of matrix multiplication operations. These operations can be proficiently optimized using a GPU because a GPU is built for this purpose.

Feature engineering

Feature engineering is the process of putting domain information into the making of highlight extractors to lessen the complex nature of the

information and make patterns more noticeable for learning algorithms to work. This procedure is troublesome and costly both regarding time and expertise.

In machine learning, the greater part of the applied feature should be distinguished by a specialist and after that hand-coded according to the domain and data type.

For instance, features can be pixel values, shape, surfaces, position and orientation. The execution of the majority of machine learning algorithms relies upon how precisely the highlights are recognized and extracted.

Deep learning algorithms attempt to take in high-level features from the information. This is an exceptionally particular piece of deep learning and a noteworthy step forward in contrast to conventional machine learning. Accordingly, deep learning lessens the undertaking of growing a new feature extractor for each issue. Similarly, convolutional neural networks will endeavor to learn low-level features, for example, edges and lines in early layers then parts of faces of people and then a high-level representation of a face.

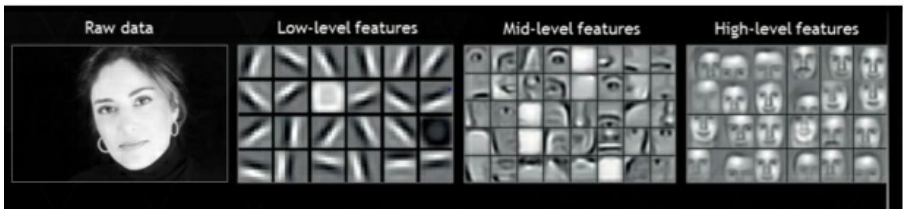


Figure 5.1: Pictorial representation

Problem solving approach

When you are sorting a problem by means of a conventional machine learning algorithm, it generally suggested to break the problem down into many parts, solve them independently, and merge them to get the result. Deep

learning in contrast advocates solving the problem end-to-end.

Let us take an illustration to examine this.

Let us say you have the task of detecting several objects. The task is to recognize what the object is and where it is present in the image (figure 5.2).

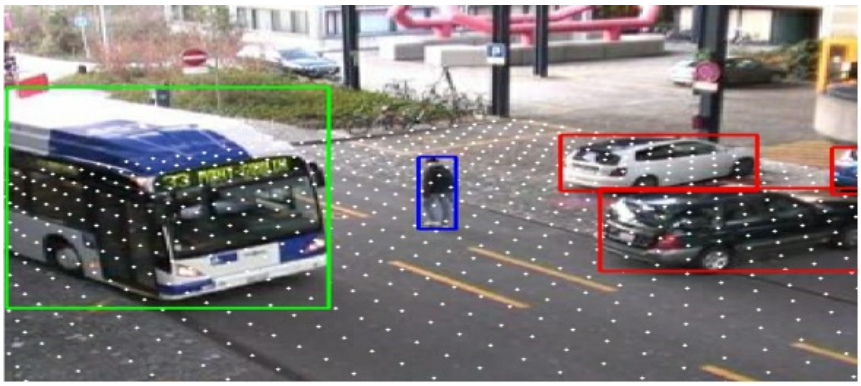


Figure 5.2: Object detection

In a typical machine learning approach, you would divide the problem into two steps, object detection and object recognition. Initially, you would use a bounding box detection algorithm, to skim through the image and find all the achievable objects. Then, for all the recognized objects, you would utilize an object recognition algorithm like SVM with HOG to recognize the relevant objects.

In contrast, in deep learning approach, you would do the process end-to-end. For example, in a YOLO net (which is a type of deep learning algorithm), you would pass in an image, and it would give out the location along with the name of object.

Execution time

Generally, a deep learning algorithm takes a long time to train. This is because there are so many parameters in a deep learning algorithm that training them takes more than usual time span. The state of the art deep learning algorithm RES-Net takes about 15 days to train completely from scratch. Machine learning, on the other hand, comparatively takes much less time to train, ranging from a few seconds to a few hours.

This in turn is completely reversed attesting time. At the test time, the deep learning algorithm takes a much reduced amount of time to run. However, if

you match it up to the k-nearest neighbors (a type of machine learning algorithm), the test time increases with the increasing size of the data. However, this is not applicable to all machine learning algorithms, as some of them have small testing times too.

Interpretability

Last but not least, we have interpretability as a feature of contrast between machine learning and deep learning. This is the main reason why deep learning is still being considered 10 times before its use in industry.

Let us take an example. Let us say we use deep learning to use automated marking on essays. The calculation it gives in scoring is quite tremendous and is close to the performance of human being. But there's is problem. It does not show why that score is given to any particular assignment. Indeed, mathematically you can find out which nodes of a deep neural network were activated, but we don't know what their neurons were thought to model and what these layers of neurons were doing as a group. So, we fail to interpret the outcomes.

On the other hand, machine learning algorithms like decision trees give us crisp rules as to why they chose what they chose, so it is particularly easy to interpret the reasoning behind them. Therefore, algorithms like decision trees and linear/logistic regressions are primarily used in the industry for interpretability.

Where deep learning and machine learning are applied

The areas in which machine learning has been used include:

- Computer vision; for applications like vehicle number plate identification and facial recognition.
- Information retrieval; for applications like search engines, both text search and image search.

- Marketing; for applications like automated email marketing and target identification
- Medical diagnosis; for applications like cancer identification and anomaly detection
- Natural language processing; for applications like sentiment analysis and photo tagging
- Online advertising, etc.

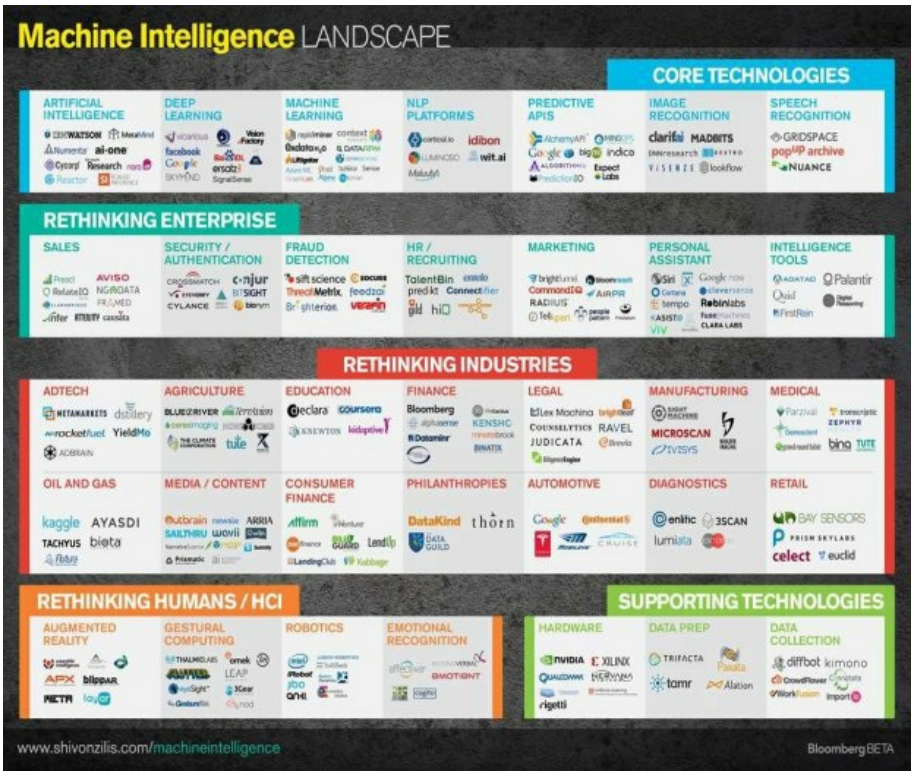


Figure 5.3: Machine learning applications

One major example of a corporation using machine learning/deep learning is Google (figure 5.3).

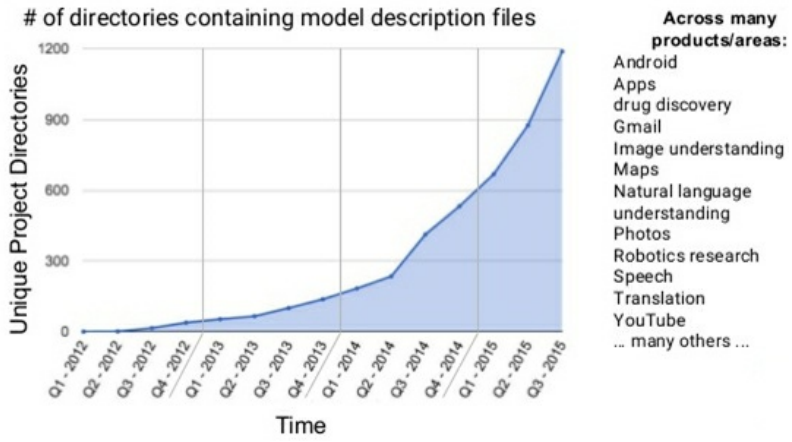


Figure 5.4: Growing use of deep learning at Google

In the above image, you can see how Google is using machine learning in its various products. Applications of machine learning/deep learning are never-ending; you just have to look at the right opportunity!

Future trends

The text above will have given you a general idea of machine learning and deep learning and the differences between them.

First of all, considering the ever-increasing trend of using data science and machine learning in the industry, it will become more important than ever before for each company that wants to survive to inculcate machine learning in its industry. Also, each and every individual will be expected to know the basic terminologies.

- Deep learning is shocking us each and every day, and will go on to do so in the near future. This is because deep learning is proving to be one of the most excellent techniques yet discovered using state-of-the-art performances.
- Research is nonstop in machine learning and deep learning. However, unlike in previous years, where research was limited to academia, research in machine learning and deep learning is exploding in both industry and academia. And with more funds available than ever before, it is more likely to play a key in human development overall.

6. Multilayer Perceptron (MLP)

Objective of this Chapter

This chapter will cover the essentials of neural nets utilizing a particular sort of system called a "multilayer perceptron", or MLP for short.

Multilayer perceptron

MLPs are normally utilized when you have what the vast majority would consider "standard data", e.g. data in a tabular format where rows are samples and column are variables. Another important thing to include here is that the rows and columns are both interchangeable, which means we can swap both the sequence of the rows or the order of the column and not change the “meaning” of the input information.

A MLP can be seen as a logistic regression classifier where the input is first changed utilizing a learnt non-linear transformation. This change projects the input data into a space, where it turns out to be linearly separable. This middle layer is alluded to as a hidden layer. A single hidden layer is sufficient to make MLPs.

The MLP model

An MLP with a single hidden layer is shown in figure 6.1.

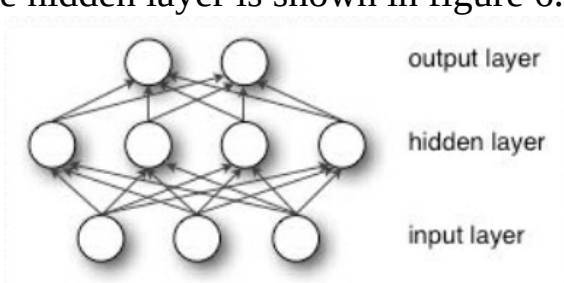


Figure 6.1: MLP model

The one-hidden-layer MLP has the function $f : R^D \rightarrow R^L$, where D is the size of input vector x and L is the size of the output vector f(x), such that, in matrix notation:

$$f(x) = G(b^{(2)} + W^{(2)}(s(b^{(1)} + W^{(1)}x))),$$

With bias vectors $b^{(1)}$, $b^{(2)}$; weight matrices $W^{(1)}$, $W^{(2)}$ and activation functions

G and s.

Tricks for training MLPs

There are a few hyper parameters, which are not (and, in general, cannot be) enhanced by gradient descent. Essentially, finding a hyper parameter of qualities for these hyper parameters isn't a feasible problem. To begin with, we can't just optimize each of them freely. Secondly, we can't promptly apply gradient strategies (partially because few parameters are discrete values and others are real values). Third, the optimization problem isn't convex and finding a local would include a significant amount of work.

Fortunately, in the last 25 years, scientists have developed different general guidelines for selecting hyper-parameters in a neural system. A decent outline of these traps can be found in Efficient BackProp by Yann LeCun, Leon Bottou, Genevieve Orr, and Klaus-Robert Mueller.

Logistic regression

Logistic regression can be seen as a straightforward type of neural system, so we will utilize it to develop a few instincts before moving on to more developed stuff.
Let us say that we've gathered data and might want to construct another classifier. To be specific, imagine that we've gathered data where each sample has two variables, $\{x1, x2\}$and there is an associated class label that lets us know whether it is a "blue dot" or a "red dot". When we plot our data, we see an image like the one in figure 6.2.

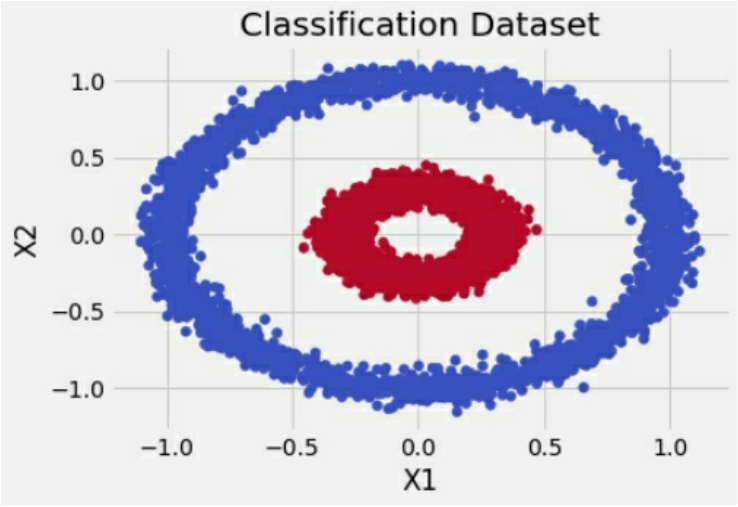


Figure 6.2: Classifier in case of logistic regression

Let us formalize our notation a bit and represent every observation as a vector of 2 variables $x_i=<x_{i1},x_{i2}>$ and transform our class labels of "blue" and "red" to a binary variable called $y_i \in \{0,1\}$, where blue=0 and red=1. Then, we need to build some function, let us call it $f(x_i)$ to demonstrate the relationship amongst x_i and y_i with the goal that when we get new data, we can accurately anticipate whether it's a blue dot or a red dot. The most straight-forward approach to develop $f(x_i)$ is to weight every variable and includes them in a linear model:

$$f(x_i) = w0 + xi1 \times w1 + xi2 \times w2$$

Where each {w1, w2} ϵ R is a scalar that weights the contribution of every variable and w0 and is the intercept term. So, $f(x_i)$ takes every variable, performs out a weighted-sum to make a single real valued number that we'd get a chance to contrast with our result variable, y_i. Since our result is binary, we can't straightforwardly compare it with $f(x_i)$, since our function can go up against any number on the real line. Rather, we'll change it to represent the probability $y_i=1$. To do this, we should present a function known as a logistic function, called $p(x_i)$:

$$p(x_i) := \frac{1}{1 + \exp(f(x_i))}$$

$p(x_i)$ represents the conditional probability that y_i is a 1, given the observed values in x_i, i.e. $Pr(y_i=1|xi)$. This possibly weird-looking function takes the number made by $f(x_i)$ and “squashes” it be between 0 and 1 and gives us the conditional probability in which we're interested. The graph of the sigmoid is shown in figure 6.3, with $f(x_i)$ on the x-axis and $p(x_i)$ on the y-axis:

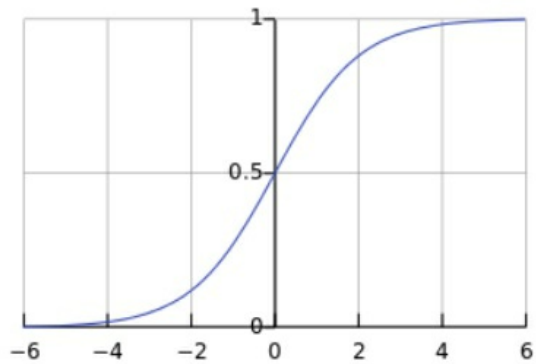


Figure 6.3: Sigmoid

Why would it be a good idea for us to lean toward this specific function over potential alternatives? All things considered, there are numerous approaches to force a number to fall between 0 and 1. For one, it has an extremely pleasant and simple to compute derivative and an assortment of decent statistical interpretations, yielding interpretable quantities known as odd-ratios; however, for the time being it allows us to simply go with the flow.

Thus, we've characterized a model $f(x_i)$, connected it to our target variable utilizing the logistic function, and now we simply have one more piece we have to define. We require an approach to "measure" how close or predicted the value, $p(x_i)$, is to the true value y_i. This is known as a loss function, shown as L(). The most widely recognized decision for a loss function in a classification task like this is binary cross entropy (BCE), demonstrated as follows:

$$L(y;w) := -y_i * log(p(x_i)) - (1 - y_i) * log(1 - p(x_i))$$

This function is low when $p(x_i)$ and y_i are similar and is high when they are different. For instance, if y_i=1 and our predicted probability is $p(x_i)$=0.9, then the loss function takes a value of 0.105. In the event that we have a predicted probability of 0.1, the loss function would have a value of 2.3. Once more, why would that be a sensible decision for a loss function? All things considered, there are a couple of reasons, including again a nice derivative, yet BCE additionally happens to be the negative log-likelihood (NLL) of a Bernoulli random variable. Since our task is anticipating a binary result, utilizing the NLL of a Bernoulli random variable appears to be a sensible activity.

To recap: We've found our model, $f(x_i)$ changed overtransformed it to be on the same scale as our response utilizing the logistic transformation, $p(x_i)$, and settled on a way to quantify how great our predictions are utilizing the double cross entropy loss function, $L(y, p(x_i))$.

All in all, how would we really learn values for {w0, w1, w2} that minimize $L(y, p(x_i))$? Enter the Stochastic Gradient Descent (SGD).

Stochastic Gradient Descent (SGD)

Envision for a minute that you have been left visually impaired and stranded on the side of a mountain. How might you get down? Below,you can feel that the incline of the steepest descent is roughly 35 ◦ to one side, so you turn and step toward th is path. You gather yourself, feel again, and make another step as per the nearby slop of the mountain. Once the ground feels level, you presume that you've achieved the base, or possibly a level.

This strategy is precisely how SGD discovers parameter values to minimizing the loss function. Rather than tactile input, SGD processes the derivative of every parameter with reference to the loss function and takes a little step the other way (e.g. negative gradient). So, we have to analytically compute the derivative for each weight in our model, with the goal that we know how to iteratively update it to enhance our expectations. To keep things basic, we're simply going to determine the update for w1, however it's precisely the same for alternate parameters. Remember, the chain rule discloses to us how to process a derivative for a composed set of functions, which is precisely what we have here. What we truly need is the partial derivative $\frac{\partial L}{\partial w_1}$; however, we need to utilize the chain rule to remove a portion of the layers of this functional onion first. The guide resembles:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial p_i}\frac{\partial p_i}{\partial w_1}$$

Computing the first term,

$$\frac{\partial L}{\partial p_i} = -\frac{y_i}{p_i} + \frac{1-y_i}{1-p_i} = \frac{p_i - y_i}{p_i(1-p_i)}$$

Computing second term,

$$\frac{\partial p_i}{\partial w1} = p_i \times (1 - p_i) \times x_1$$

Through the nice derivative property of the logistic function, to get full derivative we combine and get,

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial p_i}\frac{\partial p_i}{\partial w_1}$$

$$\left[\frac{p_i - y_i}{p_i(1 - p_i)}\right] \times [p_i \times (1 - p_i) \times x_1]$$

$$(p_i - y_i) \times x_{i1}$$

Through his sample $\{x_i, y_i\}$, we know how to change w_1 to make L go down. We first give the weight some random initial value, $w_1^{(0)}$, and then repeat the following updates until we reach convergence:

$$w_1^{(t+1)} = w_1^t - \eta \times (pi - yi) \times xi1$$

Where η is some positive number (usually <1) that controls our "step size" and is often referred to as the **learning rate**. Note this was only for one observation ii, but hopefully we have many more. The stochastic part of SGD derives from the fact that we randomly sample a number of observations BB from our full dataset, and compute the **average** gradient from this sample and use it to update our parameters. This is also called **mini-batching** or just **batching,** and it is useful when it could be difficult to fit the entire dataset into RAM, as is often the case with a GPU.I find it helpful to visualize the learning process.

Logistic regression to multilayer perceptron

At long last, a deep learning model! Since we've covered the greater part of the topic, the hop from logistic regression to a multilayer perceptron will be really simple. The fundamental distinction is that, as opposed to taking a single linear combination, we will take a few unique ones. Rather than changing the yield to a probability, as we did beforehand, we will utilize an alternate activation function to change it to a nonlinear way. We will treat the resulting nonlinear set of features as input variables to a traditional logistic regression classifier. We then do precisely what we did with logistic regression, specifically take the derivative of each parameter concerning the loss function, and limit it with SGD.

The general system resembles this:

- Pick a number h of the weighted sum of the input perform. Give the majority of the weights' distinctive initial random esteems.
- Take the yield of each of the h weighted sums and pipe it through the activation function. Each weighted sum + activation is alluded to as a **hidden unit**. The gathering of hidden units is known as a **hidden layer**.
- The subsequent set of nonlinear features are then fed as input variables into a standard logistic regression classifier.
- Train everything end-to-end with SGD.

The current most prevalent decision for the activation function is the "rectified linear unit" or RELU. It has the accompanying equation:

$$h(x) = max(0, x)$$

A graphical portrayal of a neural net with 2 hidden layers that each has 4 hidden units is demonstrated as follows (figure 6.4).

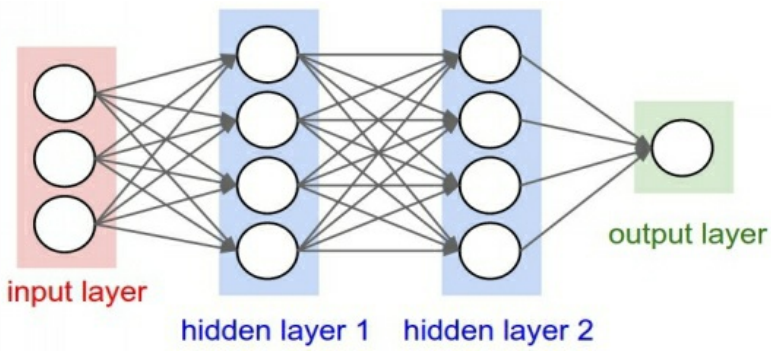


Figure 6.4: MLP illustration

We can continue adding hidden layers to make the model as deep as required. Note that very little changes when we make the model more complex. We can even add extra input, yields, or whatever else we'd like, and the training procedure will continue as before. That is a really noteworthy!

In graduate school, I discovered something many refer to as "The *Law of the Unconscious Statistician*". This law is a fun method for saying that everything the analyst does process desired by reconciliation. There is most like a comparative "Law of the Unconscious Machine Learner", which

expresses that everything the machine student does is take the subsidiary of the error work and limit. In a condition frame, it would most likely be:

$$argmin\ L(w, x, y)$$

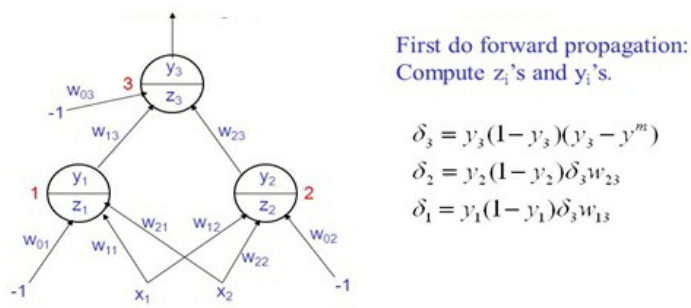
To recap: MLPs are simply logistic regressions where arrangements of nonlinear features are consequently gained from information. You need to pick a few parameters while developing a MLP:

- Number of hidden units in each layer
- Number of hidden layers in the system
- The nonlinear activation function
- Learning rate to use in SGD

Backpropagation

This was originally introduced in the 1970s but its worth wasn't fully known until 1986. It works faster than many early approaches and make it possible to solve many neural net problems which are insoluble.

Backpropagation, the short name for the "backward propagation of errors" (shown in figure 6.5) is an algorithm for the supervised learning of artificial neural networks for gradient descent and a common method for training neural networks. Known as an artificial neural network and an error function, the method calculates the gradient of the error function with respect to the neural network's weights. It is a generalization for perceptron's to multilayer feed-forward neural networks. It is named after the idea that the calculations of the gradient precede backward through the network. The gradient of the final layer of weights are calculated first and the gradient of the first layer of weights are calculated last.



$$w_{03} = w_{03} - r\delta_3(-1) \qquad w_{02} = w_{02} - r\delta_2(-1) \qquad w_{01} = w_{01} - r\delta_1(-1)$$
$$w_{13} = w_{13} - \eta\delta_3 y_1 \qquad w_{12} = w_{12} - \eta\delta_2 x_1 \qquad w_{11} = w_{11} - \eta\delta_1 x_1$$
$$w_{23} = w_{23} - \eta\delta_3 y_2 \qquad w_{22} = w_{22} - \eta\delta_2 x_2 \qquad w_{21} = w_{21} - \eta\delta_1 x_2$$

Figure 6.5 : Backpropagation example

Gradient descent

When training a neural network using the back-propagation algorithm, the gradient descent technique is used to determine the weight updates. Gradient descent is an iterative minimization method. The gradient of the error function always indicates the direction of the steepest ascent of the error function. Thus, we can start with a random weight vector.

Definition backward propagation

Backpropagation is analogous to calculating the delta rule for a multilayer feed-forward network. Thus, like the delta rule, backpropagation requires three things:

1. **A dataset** consisting of input-output pairs (x_i, y_i), where xi the input and y_i is the desired output of network depending on the input. This set of input-output pairs of size N is represented as {(x_1,y_1), …, (x_N, y_N)}.
2. **Feed forward neural networks** are collectively denoted by θ. The same layer nodes have no connection and the layers are fully connected.
3. **An error function, E(X, θ)** it defines error between the desired output y_i and the calculated output y_i for the input given.

A neural network with a gradient descent requires the calculation of the gradient of the error function **E(X,θ)** for training, with weights $\mathbf{w_{ij}^k}$ and biases $\mathbf{b_i^k}$. According to learning rate, α, iterations of gradient update weights and biases is given as:

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial E(X, \theta^t)}{\partial \theta}$$

At iteration **t,** $\boldsymbol{\theta^t}$ shows parameter of a neural network.
$\mathbf{w_{ij}^k}$: Weight for node **j** in layer $\mathbf{L_k}$ for incoming node **i**.
$\mathbf{b_i^k}$**:** Bias for node I in layer $\mathbf{L_k}$.

Applications of backpropagation

In **function approximation**, the network is trained to estimate the value of a certain function. Some noisy data is stored or fed, and system is trained to identify the required pattern. After training, the network productively estimates the value of the Gaussian function (figure 6.6).

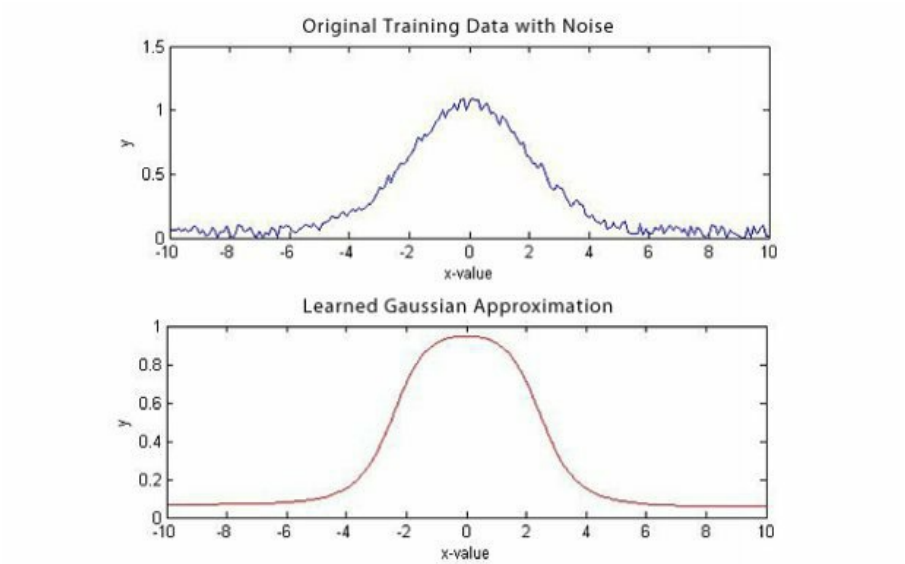


Figure 6.6: Function approximation

In ***time series prediction***, the target is to design a neural network to predict a value based on a given time-series data (i.e. stock market prediction based on given trends). To approach this problem, the inputs to the neural network have to be refectory in chunks, and the resulting output will be the next data item directly following that chunk (figure 6.7).

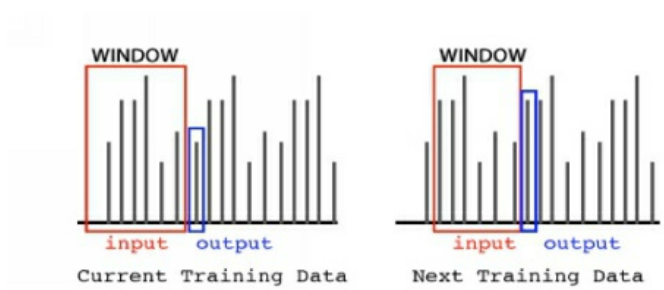


Figure 6.7: Series prediction

Long Short Term Memory networks for sentiments analysis

How a recurrent neural network (RNN) using **Long Short Term Memory** (LSTM) architecture can be implemented.

The LSTM model

In a conventional recurrent neural network, all throughout the gradient back-propagation period, the gradient signal can end up being multiplied a great number of times (as many as the number of time steps) by the weight matrix connected with the connections among the neurons of the recurrent hidden layer. This shows that the magnitude of weights in the transition matrix can have a powerful impact on the learning process.

If the weights in this matrix are little (or, more formally, if the leading eigenvalue of the weight matrix is smaller than 1.0), it can lead to a situation called *vanishing gradients*, where the gradient signal gets so little that learning either becomes very slow or stops working entirely. It can also make more complex the task of learning long-term dependencies in the data. On the other hand, if the weights in this matrix are big (or, again, more formally, if the leading eigenvalue of the weight matrix is larger than 1.0), it can lead to a situation where the gradient signal is so large that it can cause learning to diverge. This is frequently referred to as *exploding gradients*.

These problems are the major motivation behind the LSTM model, which introduces a new structure called a *memory cell* (see figure 6.8 below). A memory cell is made up of four major elements: an input gate, a neuron with a self-recurrent connection (a connection to itself), a forget gate, and an output gate. The self-recurrent link has a weight of 1.0 and gives surety that, barring any outside interference, the state of a memory cell can remain constant from one time step to another. The gates serve to modulate the interactions between the memory cell itself and its environment. The input gate can permit an incoming signal to alter the state of the memory cell or it can block it. On the other hand, the output gate can allow the state of the memory cell to have an effect on other neurons or it can prevent it. Finally, the forget gate can modulate the memory cell's self-recurrent connection and permit the cell to remember or forget its previous state, as needed.

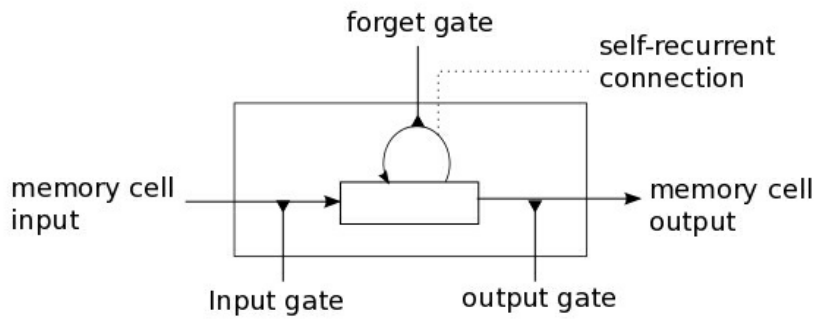


Figure 6.8: LSTM memory cell

The equations underneath explains how a layer of memory cells is updated at every time step t. In these equations:

- x_t is the input to the memory cell layer at time t
- $\mathbf{W_i}$, $\mathbf{W_f}$, $\mathbf{W_c}$, $\mathbf{W_o}$, $\mathbf{U_i}$, $\mathbf{U_f}$, $\mathbf{U_c}$, $\mathbf{U_o}$ and $\mathbf{V_o}$ are weight matrices
- $\mathbf{b_i}$, $\mathbf{b_f}$, $\mathbf{b_c}$ and $\mathbf{b_o}$ are bias vectors

Initially, we calculate the values for i_t, the input gate, and C_t, which is the candidate value for the states of the memory cells at time t:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$
$$\widetilde{C}_t = tanh(W_c x_t + U_c h_{t-1} + b_c)$$

Second, we compute the value for f_t, the activation of the memory cells' forget gates at time t:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

Knowing the value of the input gate activation i_t, the forget gate activation f_t and the candidate state value C_t, we can compute C_t, the memory cells' new state at time t:

$$C_t = i_t * \widetilde{C}_t + f_t * C_{t-1}$$

With the new state of the memory cells, we can compute the value of their output gates and, subsequently, their outputs:

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + V_o C_t + b_o)$$
$$h_t = o_t * tanh(C_t)$$

7. Convolutional Neural Networks (CNN)

Objective of this Chapter

At the end of this chapter, the reader should have learnt:

- Convolution layer
- Pooling layer
- Fully connected layer
- CONV-NET Architecture

Convolutional systems were propelled by natural or biological processes in which the network design between neurons is inspired by the association with the animal visual cortex. Individual cortical neurons react to stimuli just in a limited locale of the visual field known as the receptive field. The receptive fields of various neurons mostly cover the whole visual field.

Convolutional neural networks (CONV-Nets or CNNs) are classifications of neural networks that have demonstrated to be extremely powerful in certain fields, such as image recognition and classification. CNN have been effective in recognizing faces, objects and traffic signs separately from controlling vision in robots and self-driving autos.

CNN Architecture

A CNN comprises various convolutional and sub-sampling layers on the other hand followed by completely connected layers. The input to a convolutional layer is an m x m x r picture, where m is the height and width of the picture and r is the quantity of channels, e.g. a RGB picture has r=3. The convolutional layer will have k channels (or filters/kernels) of size n x n x q, where n is less than the dimension of the picture and q can either be the same as the number of channels r or less and may shift for every filter. The extent of the filters offers an ascent to the locally associated filters, which are each convolved with the picture to create k, including maps of size m−n+1. Each map is then sub-sampled normally with a mean or max pooling over a p x p contiguous region, where p is between 2 for small pictures (e.g. MNIST) and is typically not more than 5 for bigger sources of input. Either earlier or after the sub-sampling layer, an additive bias and sigmoidal nonlinearity is connected to each component map. The figure beneath represents a full layer in a CNN comprising convolutional and sub-sampling sub layers. Units of similar shading have tied weights. The architecture is shown in figures 7.1 and 7.2 below.

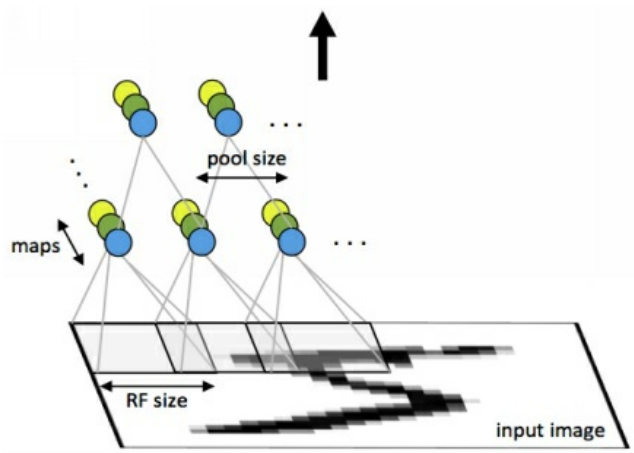


Figure 7.1: CNN architecture

After the convolutional layers, there might be any number of completely connected layers. The thickly connected layers are indistinguishable from the layers in a standard multilayer neural system.

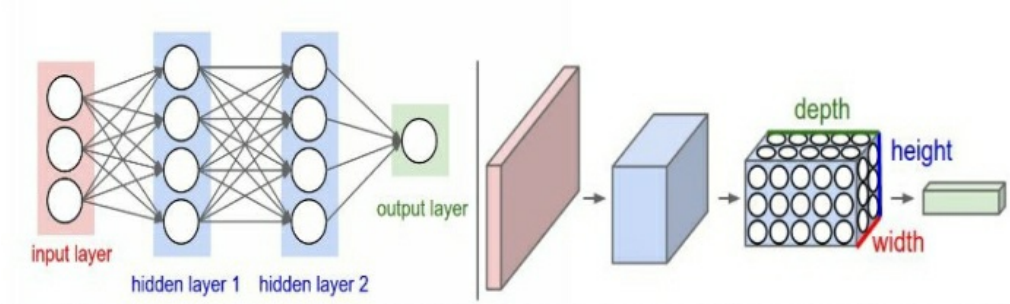


Figure 7.2: Left: A regular 3-layer neural network. Right: A CONV-Net arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Each layer of a CONV-Net transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input 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).

Layers used to build CNN

As we have shown above, a straightforward CNN is an arrangement of layers, and each layer of a CNN changes one volume of activation to another through a differentiable function. We use three fundamental kinds of layers to assemble CNN structures: Convolutional layer, pooling layer, and fully connected layer (precisely as found in standard neural networks). We will stack these layers to frame a full CNN architecture.

Case Architecture: Overview. However, a straightforward CONV-Net for CIFAR-10 classification could have the design [INPUT - CONV - ReLU - POOL - FC]. In more detail:

- INPUT [32x32x3] will hold the raw pixel values of the picture, for this situation a picture of width 32, height 32, and with three shading channels R, G, B.
- The CONV layer will figure the output of neurons that are associated with nearby regions in the input, each processing a dot product between their weights and a small region they are associated with in the input volume. This may bring about volume, for example, [32x32x12] on the off chance that we chose to utilize 12 filters.
- The ReLU layer will apply an element-wise activation work, for example, the max(0, x) thresholding at zero. This leaves the measure of the volume unaltered ([32x32x12]).
- The POOL layer will play out a down sampling operation along the

spatial measurements (width, stature), bringing about volume, for example, [16x16x12].

- The FC (i.e. fully connected) layer will process the class scores, bringing about volume of size [1x1x10], where each of the 10 numbers relate to a class score, for example, among the 10 classifications of CIFAR-10. As with standard neural networks, as the name infers, every neuron in this layer will be associated with every one of the numbers in the past volume.

Along these lines, CONV-Nets change the first picture layer by layer from the original pixel value to the last class scores. Note that a few layers contain parameters and others don't. Specifically, the CONV/FC layers perform changes that are function of the activation in the input volume, as well as of the parameters (the weights and biases of the neurons). Then again, the RELU/POOL layers will execute a fixed function. The parameters in the CONV/FC layers will be prepared with gradient descent so that the class scores that the CONV-Net processes are reliable with the labels in the training set for each picture. A visualization is given in figure 7.3 below:

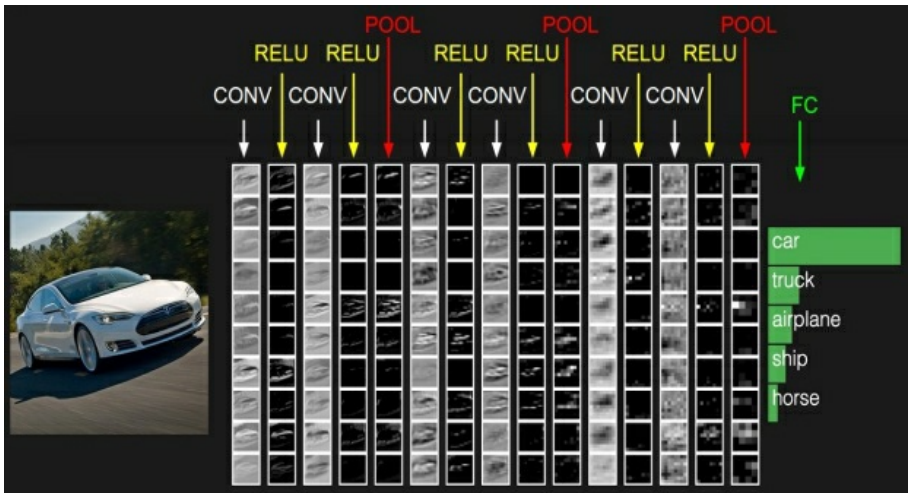


Figure 7.3: The activations of an example CONV-Net architecture. The first volume stores the raw image pixels (left) and the last volume stores the class scores (right). Each volume of activations along the processing path is shown as a column. Since it's difficult to visualize 3D volumes, we lay out each volume's slices in rows. The last layer volume holds the scores for each class, but here we only visualize the sorted top 5 scores, and print the labels of each one. The full [web-based demo](#) is shown in the header of our website. The architecture shown here is a tiny VGG Net, which we will discuss later.

Introducing non-linearity (RELU)

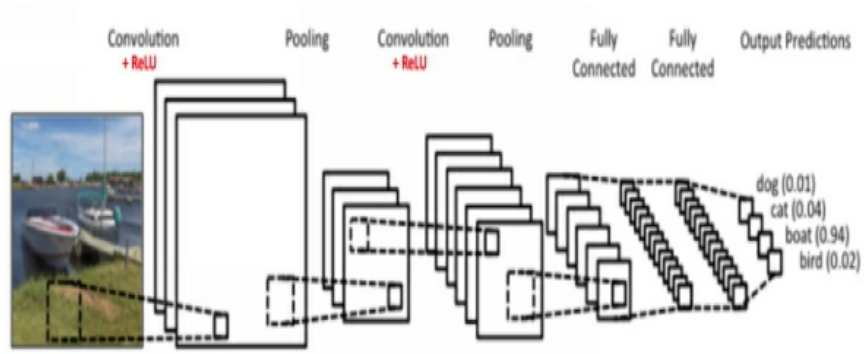


Figure 7.4: RELU illustation

An extra operation called RELU has been utilized after each convolution operation in Figure 7.4 above. RELU stands for Rectified Linear Unit and is a non-linear operation. Its output is given in figure 7.5:

Output = Max(zero, Input)

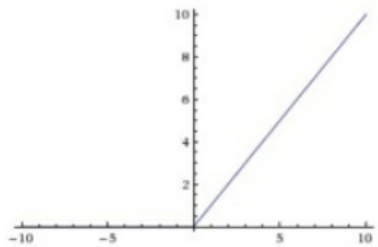


Figure 7.5: the RELU operation

RELU is an element-wise operation (connected per pixel) and replaces all negative pixel esteems in the element map by zero. The reason for RELU is to present non-linearity in our CONV-Net, since for the greater part of this present reality data we would need our CONV-Net to learn what would be non-linear (convolution is a straight/linear operation – component-wise matrix multiplication and addition, so we represent non-linearity by presenting a non-linear function like RELU).

The RELU operation can be seen unmistakably from Figure 7.6 below. It demonstrates the RELU operation connected to one of the component maps. The output feature map is also known as “rectified” feature map.

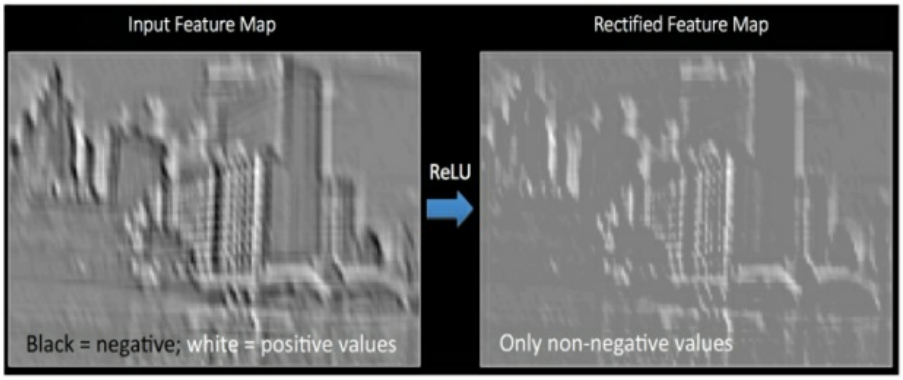


Figure 7.6: RELU feature

Other non-linear capacities, for example, tanh or sigmoid, can likewise be utilized rather than RELU; however, RELU has been found to perform better by and large.

Convolution layer

The CONV layer is the central building piece of a convolutional network and does a large portion of the truly difficult computational work.

Without brain model

Let us initially take this chance to talk about what the CONV layer processes without mind/neuron analogies. The CONV layer's parameters comprise a parameter of learnable filters. Each filter (channel) is spatially small (along width and height), however reaches out through the full depth of the input volume. For instance, a run of the typical filter on a first layer of a CONV-Net may have estimate 5x5x3 (i.e. 5 pixels width and height, and 3 since pictures have depth 3, the shading channels). Along with the forward pass, we slide (all the more absolutely, convolve) each filter over the width and height of the input volume and figure spot items between the passages of the channel and the contribution at any position. As we slide the channel over the width and height of the input volume, we will deliver a 2-dimensional actuation outline that gives the reactions of that channel at each spatial position. Naturally, the system will learn channels that initiate when they see some kind of visual element, for example, an edge of some introduction or a mark of some color on the principal (first) layer, or in the final whole honeycomb or wheel-like patterns on higher layers of the network. Presently, we will have a whole set of channels in each CONV layer (e.g. 12 channels/filters), and each of them will create a different 2-dimensional activation map (outline). We will stack these activation maps along the depth

measurement and deliver the output volume.

The brain-view

In case you're a devotee of brain/mind/neuron analogies, each section in the 3D output volume can likewise be translated as an output of a neuron that focuses on just a little region in the input and offers parameters with all neurons to one side and the right spatially (since these numbers all come about because of applying a similar channel). We now talk about the subtle elements of the neuron networks, their arrangement in space, and their parameter sharing plan.

Local connectivity

When managing high-dimensional sources of input, such as pictures, as we saw above it is unrealistic to connect neurons to all neurons in the past volume. Rather, we will associate every neuron to just a local region of the input volume. The spatial degree of this connectivity is a hyper parameter called the receptive field of the neuron (proportionately this is the filter size). The degree of the connectivity along the depth axis is constantly equivalent to the depth of the input volume. It is vital to put emphasis on this asymmetry again by the way we treat the spatial dimensions (width and height) and the depth dimension: The connections are nearby (local) in space (along width and height), however constantly full along the whole depth of the input volume.

Spatial arrangement

We have clarified the connectivity of every neuron in the ConvLayer to the input volume; however, we haven't yet examined what number of neurons there are in the output volume or how they are organized. Three hyper parameters control the span of the output volume: the depth, stride and zero-padding. We talk about these next.

1. In the first place, the depth of the output volume is a hyper parameter: It relates to the quantity of filters we might want to utilize, each learning to search for something else in the input. For instance, if the primary convolutional layer takes as input the crude/raw picture, at that point diverse neurons along the depth dimension may actuate in presence of different arranged edges, or

blobs of colors. We will allude to an arrangement of neurons that are on the whole taking a same region of the input as a depth column (a few people additionally incline toward the term fiber).

2. Second, we should determine the step with which we slide the filter. At the point when the stride is 1, then we move the filter one pixel at any given moment. At the point, when the stride is 2 (or remarkably at least 3, however this is generally uncommon) at that point the filters hop 2 pixels at any given moment as we slide them around. This will deliver littler output volumes spatially.

3. As we will soon observe, now and again it is helpful to pad the input volume with zeros around the border. The measure of this zero-padding is a hyper parameter. The good part of zero padding is that it will enable us to control the spatial size of the output volumes.

Real life example

The Krizhevsky et al. engineering that won the ImageNet challenge in 2012 acknowledged pictures of size [227x227x3]. On the principal convolutional layer, it utilized neurons with receptive field measure F=11, stride S=4 and no zero padding P=0. Since (227 - 11)/4 + 1 = 55, and since the ConvLayer had a depth of K=96, the ConvLayer output volume had the estimate [55x55x96]. Each of the 55*55*96 neurons in this volume was associated with a region of size [11x11x3] in the input volume. Additionally, every one of the 96 neurons in every depth column was associated with the same [11x11x3] area of the input, obviously with different weights. As a fun aside, on the off chance that you read the real paper, it asserts that the information pictures were 224x224, which is without a doubt off base because that (224 - 11)/4 + 1 is obviously not an integer. This has befuddled many individuals in the historical backdrop of CONV-Nets and little is thought about what happened.

Parameter sharing

A parameter sharing plan is utilized as a part of convolutional layers to control the quantity of the parameters. Utilizing the present real-world case given above, we see that there are 55*55*96 = 290,400 neurons in the primary ConvLayer, and each has 11*11*3 = 363 weights and 1 bias. Together, this indicates 290400*364 = 105,705,600 parameters on the principal layer of the CONV-Net alone. Obviously, this number is high.

Things being what they are, we can significantly lessen the quantity of parameters by making one sensible assumption: On the off chance that one component is valuable to compute at some spatial position (x,y), at that point it ought to likewise be helpful to compute at a different position (x2,y2). As such, indicating a single 2-dimensional slice of depth as a depth slice (e.g. a volume of size [55x55x96] has 96 depth slices, each of size [55x55]), we will constrain the neurons in every depth slice to utilize similar weights and bias. With this parameter sharing plan, the primary ConvLayer in our illustration would now have just 96 unique set of weights (one for every depth slice), for a sum of 96*11*11*3 = 34,848 unique weights, or 34,944 parameters (+96 biases). On the other hand, every one of the 55*55 neurons in every depth slice will now utilize similar parameters. By and by, along with backpropagation, each neuron in the volume will compute the gradients for its weights; however, these gradients will be included over every depth slice and just refresh a single set of weights per slice.

Notice that if all neurons in a single depth slice are utilizing a similar weight vector, at that point the forward motion of the ConvLayer can in every depth slice be figured as a convolution of the neuron's weights with the input volume (hence the name: convolutional layer). This is basically with the arrangements of weights are referred to as a channel/filter (or a kernel) that is convolved with the input.

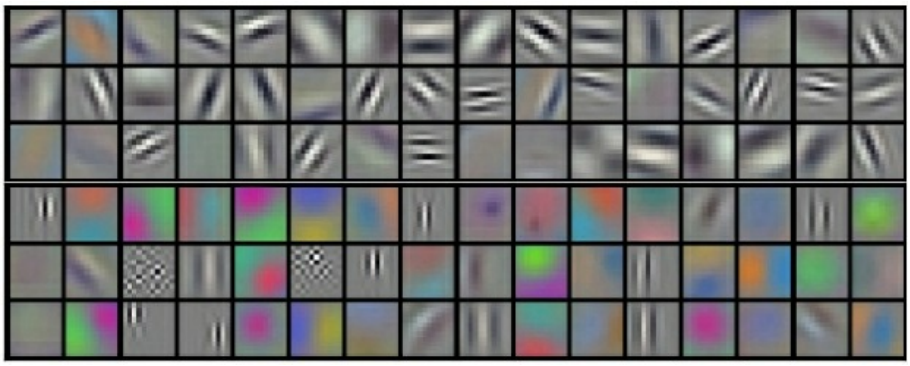


Figure 7.7 Example filters learned by Krizhevsky et al. Each of the 96 filters shown here is of size [11x11x3], and each one is shared by the 55*55 neurons in one depth slice. Notice that the parameter sharing assumption is relatively reasonable: If detecting a horizontal edge is important at some location in the image, it should intuitively be useful at some other location as well due to the translational-invariant structure of images. There is therefore no need to relearn to detect a horizontal edge at every one of the 55*55 distinct locations in the ConvLayer output

volume.

Pooling layer

It is normal to periodically embed a pooling layer in the middle of progressive Conv layers in CONV-Net architecture. Its function is to progressively decrease the spatial size of the representation to lessen the measure of parameters and calculation in the network, and thus to likewise control over-fitting. The pooling layer operates independently on each depth slice of the input and resizes it spatially, utilizing the MAX operation. The most widely recognized shape is a pooling layer with filters of size 2x2 connected with a stride of 2 down samples for each depth slice in the input by 2 along both width and height, discarding 75% of the activation. Each MAX operation would for this situation be taking a maximum of more than 4 numbers (minimal 2x2 regions in a depth slice). The depth dimension stays unaltered. For the most part, the pooling layer:

- Acknowledges a volume of size $W_1 * H_1 * D_1$
- Requires two hyper parameters:
  - their spatial degree F,
  - the stride S
- Produces a volume of size $W_2 * H_2 * D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 - D_1$
- Presents zero parameters since it computes a fixed function of input
- Note that it isn't normal to utilize zero-padding for pooling layers

It is important that there are generally just two normally observed varieties of the maximum pooling layer found: A pooling layer with F=3, S=2 (likewise called overlapping pooling), and all the more regularly F=2, S=2. Pooling sizes with bigger receptive fields are excessively dangerous.

General pooling

In addition to max pooling, the pooling units can likewise perform different functions, such as average pooling or even L2-norm pooling. Average pooling was often utilized verifiably, however it has as of late lost support in contrast to the maximum pooling operation, which has been appeared to work better in practice.

Getting rid of pooling

Many individuals despise (or at least dislike) the pooling operation and imagine that we can get away without it, such as for simplicity. The All Convolutional Net proposes to discard the pooling layer for an architecture that only comprises repeated CONV layers. To decrease the measure of the representation, they suggest utilizing bigger strides in CONV layer now and again. Discarding pooling layers has additionally been observed to be imperative in training great generative models, such as variation auto encoders (VAEs) or generative adversarial networks (GANs). It appears to be likely that future models will include not many pooling layers or even none at all.

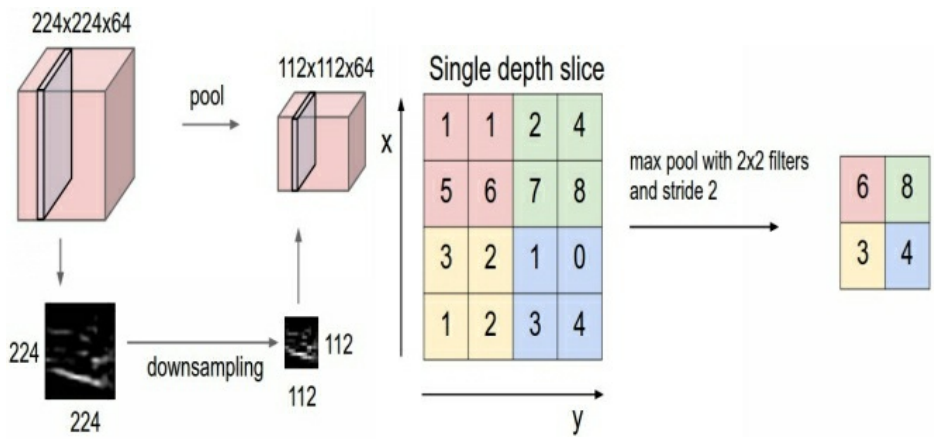


Figure 7.8: Pooling layer down samples the volume spatially, independently in each depth slice of the input volume. Left: In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. Right: The most common down sampling operation is max, giving rise to max pooling, and here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

Fully connected layer

Neurons in a completely associated layer have full associations with all activations in the previous layer, as found in regular neural networks. Their actuations can subsequently be computed with a matrix multiplication taken over by a bias offset.

Converting FC layers to CONV layers

It is significant that the main contrast between FC and CONV layers is that the neurons in the CONV layer are associated with just a local region in the input, and that a large number of the neurons in a CONV volume share parameters. Be that as it may, the neurons in the two layers still figure a dot product, so their functional form is identical. Consequently, for reasons unknown it's conceivable to covert between FC and CONV layers:

- For any CONV layer there is a FC layer that actualizes the same forward function. The weight matrix would be a huge matrix that is for the most part zero. With the exception of a specific block (because of local connectivity) where the weights in a considerable lot of the blocks are equivalent (because of parameter sharing).
- Then again, any FC layer can be changed to a CONV layer. For instance, a FC layer with K=4096 that is taking at some input volume of size 7×7×512 can be equivalently expressed as a CONV layer with F=7,P=0,S=1,K=4096. At the end of the day, we are setting the filter size to be precisely the size of the input volume, and subsequently the output will basically be 1×1×4096 since just a single depth column "fits" over the input volume, giving identical outcome as the initial FC layer.

FC->CONV conversion; of these two conversions, the capacity to change over a FC layer to a CONV layer is especially valuable, practically speaking. Consider a CONV-Net design that takes a 224x224x3 picture and after that uses a progression of CONV layers and POOL layers to lessen the picture to an activation volume of the size 7x7x512 (in an *ALEX-Net* architecture. This is finished by utilizing5 pooling layers that down sample the input spatially by a factor of two each time, making the last spatial size 224/2/2/2/2/2 = 7. From that point, an ALEX-Net utilizes two FC layers of the size 4096, and lastly the last FC layers with 1000 neurons that process the class scores. We

can transform each of these three FC layers to CONV layers, as depicted previously:

- Replace the main FC layer that looks at a [7x7x512] volume with a CONV layer that utilizes channel size F=7, giving output volume [1x1x4096].
- Supplant the second FC layer with a CONV layer that utilizes channel size F=1, giving the outcome volume [1x1x4096]
- Supplant the last FC layer likewise, with F=1F=1, giving the last yield [1x1x1000]

Each of these conversions could generally include manipulating (e.g. reshaping) the weight matrix W in each FC layer into CONV layer channels. For reasons unknown, this change enables us to "slide" the first CONV-Net productively crosswise over numerous spatial positions in a bigger picture, in a single forward pass.

For instance, if a 224x224 picture gives a volume of size [7x7x512] – i.e. a decrease by 32, at that point sending a picture of the size 384x384 through the transformed architecture would give the equal volume in size [12x12x512], since 384/32 = 12. Finishing the following 3 CONV layers, which we simply transform from FC layers, would now give the last volume of the size [6x6x1000], since (12 - 7)/1 + 1 = 6. Note that rather than a single vector of class scores of size [1x1x1000], we're presently getting a whole 6x6 array of class scores over the 384x384 picture.

Assessing the first CONV-Net (with FC layers) freely crosswise over 224x224 products of the 384x384 picture in steps of 32 pixels gives an identical outcome to sending the transformed CONV-Net one time.

Normally, sending the converted CONV-Net a single time is substantially more proficient than iterating the first CONV-Net over each one of those 36 locations, since the 36 assessments share a calculation. This trap is regularly utilized as a part of training to show signs of improvement execution, whereby for instance, it is normal to resize a picture to make it greater, utilize a transformed CONV-Net to evaluate the class scores at numerous spatial positions, and then average the class scores.

Finally, imagine a scenario in which we need to effectively apply the first (original) CONV-Net over the picture, however at steps smaller than 32

pixels. We could accomplish this with multiple forward passes. For instance, take note of the fact that, on the off chance that we needed to utilize a stepof 16 pixels, we could do so by joining the volumes receivedby forwarding the transformed CONV-Net twice: First over the original picture and second over the picture yet with the picture moved spatially by 16 pixels along both width and height.

CONV-Net Architectures

We have seen that convolutional networks commonly comprise just three layers types: CONV, POOL (we expect Max pool unless expressed generally) and FC (short for fully connected). We will likewise expressly compose the ReLU actuation function as a layer, which applies element-wise non-linearity. In this segment we examine how these are generally stacked together to frame whole CONV-Nets.

Layer pattern

The best-known type of CONV-Net architecture stacks a couple of CONV-ReLU layers, followed with POOL layers, and repeats this pattern until the picture has been merged spatially to a small size. Sooner or later, it is common to transition to completely connected layers. The last completely connected layer holds the output, for example, the class scores. As such, the most widely recognized CONV-Net design takes after this pattern:

INPUT → [[CONV → RELU]*N →POOL]*M → [FC → RELU]*K → FC

Where the * demonstrates repetition, and the POOL? Demonstrates the optional pooling layer. In addition, N >= 0 (and more often than not N <= 3), M >= 0, K >= 0 (usually K < 3). For instance, here are some normal CONV-Net models you may see that take after this example:

- INPUT - > FC, implements a linear classifier. Here N = M = K = 0.
- INPUT - > CONV - > RELU - > FC
- INPUT - > [CONV - > RELU - > POOL]*2 - > FC - > RELU - > FC. Here we see that there is a single CONV layer between each POOL layer.

INPUT - > [CONV - > RELU - > CONV - > RELU - > POOL]*3 - > [FC - > RELU]*2 - > FC Here we see two CONV layers stacked before each POOL layer. This is by and large a smart idea for bigger and deeper systems, in light

of the fact that different stacked CONV layers can grow more complex features of the input volume before the destructive pooling operation.

Move toward a pile of small filter CONV to one substantial receptive field CONV layer. Assume that you stack three 3x3 CONV layers over each other (with non-linearity's in the middle, obviously). During this course of action, every neuron on the first CONV layer has a 3x3 perspective of the input volume. A neuron on the second CONV layer has a 3x3 view of the first CONV layer, and henceforth by expansion a 5x5 view of the input volume. Similarly, a neuron on the third CONV layer has a 3x3 perspective of the second CONV layer, and thus a 7x7 view of the input volume. Assume that rather than these three layers of 3x3 CONV, we just need to utilize a single CONV layer with 7x7 receptive fields. These neurons would have a receptive field size of the input volume that is identical in spatial extent (7x7), yet with a few disadvantages. To begin with, the neurons would be computing a linear function over the input, while the three piles of CONV layers contain non-linearities that make their features more expensive. Second, if we assume that every one of the volumes have C channels, at that point it can be seen that the single 7x7 CONV layer would contain $C\times(7\times7\times C)=49C^2$ parameters, while the three 3x3 CONV layers would just contain $3\times(C\times(3\times3\times C))=27C^2$ parameters. Naturally, stacking CONV layers with small filters instead of having one CONV layer with enormous filters enables us to express more intense features of the input, and with less parameters. As a practical disadvantage, we may require more memory to hold all the middle CONV layer outcomes if we intend to do backpropagation.

Recent departures: It should be noted that the conventional paradigm of a linear list of layers has as of late been challenged by Google's Inception models and, furthermore, by current (state of art) residual networks from Microsoft Research Asia. Both of these highlight more complex and are diversely available structures.

In practice, utilize whatever works best on ImageNet. In case you're feeling somewhat exhausted of pondering the architectural decision, you'll be satisfied to realize that in at least 90% of cases you should not need to stress over these. Instead of moving your own particular architecture for an issue, you should take a best available architecture.

Layer sizing Patterns

Up until this point we've overlooked instances of common hyper parameters utilized as a part of each of the layers in a CONV-Net. We will first express the general guidelines for measuring the models and after that follow the rules with a dialog of the notation:

The info layer (which contains the picture) ought to be divisible by a twofold of times. Common numbers include 32 (e.g. CIFAR-10), 64, 96 (e.g. STL-10), or 224 (e.g. common ImageNet CONV-Nets), 384, and 512.

The CONV-Layers ought to utilize small channels (e.g. 3x3 or at most 5x5), utilizing a stride of S=1, and crucially, padding the input volume with zeros in such way that the CONV-Layer does not change the spatial dimension of the input. That is, F=3, when utilizing P=1, will hold the first size of the input. Whenever F=5, then P=2. For a general F, it can be seen that P= (F −1)/2 preserves the input size. In the event that you should utilize greater channel sizes, (for example, 7x7 or somewhere in the vicinity), it is just normal to see this on the main CONV-Layer that is looking at the input image.

The pool layers are responsible for down sampling the spatial dimensions of the input. The most widely recognized setting is to utilize max-pooling with 2x2 receptive fields (i.e. F=2), and with a stride of 2 (i.e. S=2). Note that this discard exactly 75% of the actuations in the input volume (because of down sampling by 2 in both width and height). Another somewhat less basic setting is to utilize 3x3 receptive fields with a stride of 2, regardless of what this makes. It is extremely remarkable to see receptive field sizes for max pooling that are bigger than 3, where by the pooling is then excessively lost and aggressive. This ordinarily prompts a worse execution.

Reducing sizing headaches: The scheme displayed above is satisfying in light of the fact that all the CONV layers preserve the spatial size of their input, while the POOL layers alone are accountable for down- sampling the volumes spatially. In an alternative scheme, where we utilize strides more noteworthy than 1 or don't zero-pad the input to CONV layers, we would need to deliberately track the input volumes all through the CNN design and ensure that all steps and channels "work out", and that the CONV-Net architecture is pleasantly and symmetrically wired.

Why utilize a stride of 1 in CONV? Smaller strides work better practically speaking. Also, the specified stride of 1 enables us to leave all spatial down-sampling to the POOL layers, with the CONV layers just changing the input volume depth-wise.

Why utilize padding? Notwithstanding the previously mentioned advantage of keeping the spatial sizes consistent after CONV, doing so really enhances execution. On the off chance that the CONV layers were not zero-padding the sources of input and just performing valid convolutions, the extent of the volumes would decrease by a small amount after each CONV, and the data at the fringes would be "washed away" too rapidly.
Compromising based on memory constraints: Now and again (particularly later in the CONV-Net designs), the measure of memory can develop rapidly using the general guidelines introduced previously. For instance, sifting a 224x224x3 picture with three 3x3 CONV layers with 64 channels each and padding 1 would make three initiation volumes of size [224x224x64]. This adds up to an aggregate of around 10 million initiations, or 72MB of memory (per picture, for the two actuations and gradient). Since GPUs are frequently bottlenecked by memory, it might be important to compromise. In general, individuals like to make compromises at just the main CONV layer of the network. For instance, one trade-off may be to utilize a first CONV layer with a channel size of 7x7 and a stride of 2 (as found in a ZF net). As another case, an ALEX-Net utilizes channel sizes of 11x11 and stride of 4.

Other CONV-Net Architectures

- **ALEX-Net (2012) –** In 2012, Alex Krizhevsky (and others) released ALEX-Net, which was a deeper and much wider version of the LE-Net and won the difficult ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012by a large margin. It was a significant breakthrough with respect to the previous approaches and the current widespread application of CNNs can be attributed to this work.

- **ZF Net (2013) –** The ILSVRC 2013 winner was a convolutional network from Matthew Zeiler and Rob Fergus. It became known as the ZF-Net (short for Zeiler & Fergus Net). It improved on ALEX-Net by tweaking the architecture hyper parameters.

8. Other Deep Learning Algorithms

Restricted Boltzmann machines

The next logical step is to look at a Restricted Boltzmann machines (RBM), *a* generative stochastic neural network that can learn a probability distribution over its set of inputs (figure 8.1).

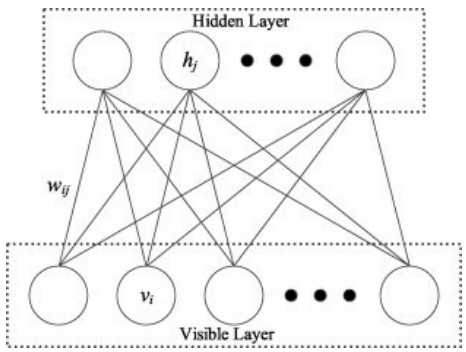


Figure 8.1: RBM

RBMs are self-possessed of a hidden, visible, and bias layer. Unlike like the feed-forward networks, the connections between the visible and hidden layers are undirected (the values can be propagated in both the visible-to-hidden and hidden-to-visible directions) and fully connected (each unit from a given layer is connected to each unit in the next—if we allow any unit in any layer to connect to any other layer, then we'd have a Boltzmann [rather than a *restricted Boltzmann*] machine).

A RBM is an uncommon kind of Markov random field that has one layer of (regularly Bernoulli) stochastic hidden units and one layer of (ordinarily Bernoulli or Gaussian) stochastic visible or detectable units. RBMs can be referred to as bipartite diagrams, where every single obvious unit is associated with every hidden unit, and there are no visible or hidden associations.

Boltzmann machines in depth:

Boltzmann Machines (BMs) are of a particular type of log-linear Markov Random Field (MRF), i.e., for which the energy function is linear in its free of charge parameters. To construct them to be as powerful as necessary and to be shown as complex distributions (i.e., go from the limited parametric

setting to a non-parametric one), we consider that several variables are by no means observed (they are called hidden). By having more hidden variables (also called hidden units), we can add to the modeling capacity of the Boltzmann machine (BM). Restricted Boltzmann machines further restrict BMs to those without visible-visible and hidden-hidden connections. A graphical depiction of an RBM is shown below (figure 8.2).

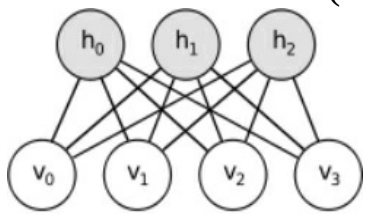


Figure 8.2: BM's

The energy function E(v, h) of an RBM is defined as:

Eq (1)

$$E(v,h) = -b'v - c'h - h'Wv$$

Where, W shows the weights connecting hidden and visible units and b, c are the offsets of the visible and hidden layers in that order.

This translates directly to the following free energy formula:

$$\mathcal{F}(v) = -b'v - \sum_i \log \sum_{h_i} e^{h_i(c_i + W_i v)}.$$

The specific structure of RBM's visible and hidden units are conditionally independent given one-another. Using this property, we can write down:

$$p(h|v) = \prod_i p(h_i|v)$$

$$p(v|h) = \prod_j p(v_j|h).$$

RBMs with binary units

In the formal studied case of using binary units (where v_j and $h_i \in \{0, 1\}$), we obtain from eq(1) and 2, a probabilistic version of the usual neuron activation function:

Eq(2)

$$P(h_i = 1|v) = sigm(c_i + W_i v)$$

Eq(3)

$$P(v_j = 1|h) = sigm(b_j + W_j'h)$$ 3

The free energy of an RBM with binary units further simplifies to:

Eq(4)

$$\mathcal{F}(v) = -b'v - \sum_i \log(1 + e^{(c_i + W_i v)}).$$

Update equations with binary units

According to Eq(4), we obtain the following log-likelihood gradients for an RBM with binary units:

$$-\frac{\partial \log p(v)}{\partial W_{ij}} = E_v[p(h_i|v) \cdot v_j] - v_j^{(i)} \cdot sigm(W_i \cdot v^{(i)} + c_i)$$
$$-\frac{\partial \log p(v)}{\partial c_i} = E_v[p(h_i|v)] - sigm(W_i \cdot v^{(i)})$$
$$-\frac{\partial \log p(v)}{\partial b_j} = E_v[p(v_j|h)] - v_j^{(i)}$$

Energy-based model (EBM)

Energy-based models connect a scalar energy to every configuration of the variables of interest. Learning corresponds to modifying the energy function so that its shape has the properties that we desire. For instance, we would like plausible or desirable configurations to have low energy. Energy-based probabilistic models define a probability distribution all the way through an energy function; as follows:

Eq(1)

$$p(x) = \frac{e^{-E(x)}}{Z}.$$

The normalizing feature Z is called the **partition function** in an analogy with physical systems.

$$Z = \sum_x e^{-E(x)}$$

An energy-based model can be trained by performing (stochastic) gradient descent on the observed negative log-likelihood of the training data. With respect to the logistic regression, we will initially define the log-likelihood and afterwards the loss function as being the negative log-likelihood.

$$\mathcal{L}(\theta, \mathcal{D}) = \frac{1}{N} \sum_{x^{(i)} \in \mathcal{D}} \log\ p(x^{(i)})$$

$$\ell(\theta, \mathcal{D}) = -\mathcal{L}(\theta, \mathcal{D})$$

By means of the stochastic gradient $-\frac{\partial \log p(x^{(i)})}{\partial \theta}$, where θ are the parameters of the model.

EMBs with hidden units

In many scenarios of interest, we do not examine the example x entirely, or we would like to introduce some non-observed variables to increase the expressive power of the model. So, we think about an observed part (still denoted x here) and a **hidden** part h. We can then write down:

Eq(2)

$$P(x) = \sum_h P(x, h) = \sum_h \frac{e^{-E(x,h)}}{Z}.$$

In these cases, to map this formulation to one similar to Eq. (1) above, we set up the notation (inspired by physics) of **free energy**, defined as follows:

Eq(3)

$$\mathcal{F}(x) = -\log \sum_h e^{-E(x,h)}$$

That will allow us to write,

$$P(x) = \frac{e^{-\mathcal{F}(x)}}{Z} \text{ with } Z = \sum_x e^{-\mathcal{F}(x)}.$$

The data negative log-likelihood gradient then has a particularly interesting form.

Eq(4)

$$-\frac{\partial \log p(x)}{\partial \theta} = \frac{\partial \mathcal{F}(x)}{\partial \theta} - \sum_{\tilde{x}} p(\tilde{x})\ \frac{\partial \mathcal{F}(\tilde{x})}{\partial \theta}.$$

Observe that the above gradient consists of two terms, which are called the **positive** and **negative phases**. The conditions of positive and negative do not refer to the sign of each term in the equation, but, to a certain extent, reflect their impact on the probability density defined by the model. The initial term increases the probability of training data (by reducing the

corresponding free energy), while the next term decreases the probability of samples generated by the model.

It is more often than not difficult to verify this gradient analytically, as it involves the computation of $E_P[\frac{\partial \mathcal{F}(x)}{\partial \theta}]$. This is nothing less than an expectation regarding all likely configurations of the input x (under the distribution P formed by the model).

The initial step in making this computation tractable is to estimate the expectation using a fixed number of model samples. Samples used to estimate the negative phase gradient are referred to as **negative particles**, which are denoted as N. The gradient can then be written as:

Eq(5)

$$-\frac{\partial \log p(x)}{\partial \theta} \approx \frac{\partial \mathcal{F}(x)}{\partial \theta} - \frac{1}{|\mathcal{N}|} \sum_{\tilde{x} \in \mathcal{N}} \frac{\partial \mathcal{F}(\tilde{x})}{\partial \theta}.$$

Where we would preferably like elements x of N to be sampled according to P (i.e. we are doing MonteCarlo). With the formula mentioned above, we almost have a practical, stochastic algorithm for learning an EBM. The only missing ingredient is how to extract these negative particles N. As the statistical literature abounds with sampling methods, Markov Chain Monte Carlo methods are especially well suited to models such as the restricted Boltzmann machines (RBM), a specific type of EBM.

Auto encoders

An auto encoder takes an input **x** ϵ $[0, 1]^d$ and primarily maps it (with an *encoder)* to a hidden representation **y** ϵ $[0, 1]^{d'}$ all the way through a deterministic mapping, e.g.:

$$\mathbf{y} = s(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Here, s is a non-linearity such as the sigmoid. The latent symbol **y**, or **code,** is then mapped back (with a *decoder)* into a **reconstruction z** of the same shape as **x**. The mapping occurs through a similar transformation, e.g.:

$$\mathbf{z} = s(\mathbf{W}'\mathbf{y} + \mathbf{b}')$$

(Here, the major symbol does not point out the matrix transposition.) **z** should be seen as a prediction of $\mathbf{x}$, given the code $\mathbf{y}$. Optionally, the weight matrix **W'** of the turnaround (reverse) mapping may be forced to be the transpose of the forward mapping: $\mathbf{W'=W^T}$. This is referred to as *tied*

weights. The parameters of this form (namely **W', b, b'** and, if one doesn't use tied weights, also **W'**) are optimized so that the average reconstruction error is minimized.

The rebuilding error can be measured in many ways, depending on the appropriate distributional assumptions on the input given the code. The traditional *squared error* $\mathbf{L(X, Z) = \|X - Z\|^2}$ can be used. If the input is interpreted as either bit vectors or vectors of bit probabilities, *cross-entropy* of the reconstruction can be utilized:

$$L_H(\mathbf{x}, \mathbf{z}) = -\sum_{k=1}^{d} [\mathbf{x}_k \log \mathbf{z}_k + (1 - \mathbf{x}_k) \log(1 - \mathbf{z}_k)]$$

The expectation is that the code y is a *distributed* representation that captures the coordinates all along the main factors of variation in the data. This is similar to the way the projection on principal components would capture the main factors of variation in the data. Indeed, if there is one linear hidden layer (the *code)* and the mean squared error criterion is used to train the network, then the k hidden units learn to project the input in the span of the first k principal components of the data. If the hidden layer is non-linear, the auto-encoder behaves differently from PCA, with the ability to capture multi-modal aspects of the input distribution. The disappearance from PCA becomes even more important when we consider *stacking multiple encoders* (and their corresponding decoders) when building a deep auto-encoder.

Because y is seen as a lossy compression of x, it cannot be a good quality (small-loss) compression for all x. Optimization makes it a good compression for training examples, and hopefully for other inputs as well, but not for arbitrary inputs. That is the sense in which an auto-encoder generalizes: It gives a low reconstruction error on test examples from the same distribution as the training examples, but generally high reconstruction error on samples randomly chosen from the input space.

Stacked auto encoders

This network consists of multiple stacked auto encoders, as the name shows (figure 8.2).
The hidden layer of auto encoder *t* acts as an input layer to auto encoder *t* +

1. The input layer of the first auto encoder is the input layer for the complete network. The greedy layer-wise training procedure mechanism is like this:

- Train the initial auto encoder (*t=1*, or the red connections in the figure above, but with an additional output layer) individually using the backpropagation method with all accessible training data.

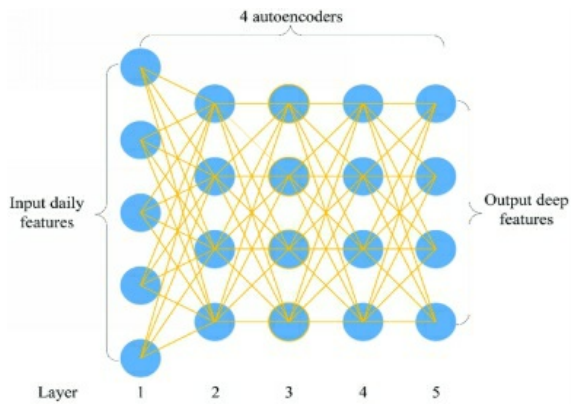


Figure 8.2 stacked auto encoders

- Train the next auto encoder *t=2* . In view of the fact that the input layer for *t=2* is the hidden layer of *t=1*, we are no longer interested in the output layer of *t=1* and we eliminate it from the network. Training begins by clamping an input sample to the input layer of *t=1*, which is propagated forward to the output layer of *t=2*. Next, the weights (input-hidden and hidden-output) of *t=2* are updated using backpropagation. *t=2* uses all the training samples, similar to *t=1*.
- Repeat the earlier procedure for all the layers (i.e., remove the output layer of the previous auto encoder, replace it with yet another auto encoder, and train with backpropagation).
- All above steps are called *pre-training* and leave the weights correctly initialized. On the other hand, there's no mapping between the input data and the output labels. For example, if the network is trained to identify imagery of handwritten digits, it's still not possible to map the units from the last feature detector (i.e., the hidden layer of the last auto encoder) to the digit type of the image. In that case, the most common solution is to add one or more fully connected layer to the last layer. The whole network can now be viewed as a multilayer perceptron and

is trained using backpropagation (this step is also called *fine-tuning*).

Stacked auto encoders, then, are all about providing an effective pre-training technique for initializing the weights of a network, leaving you with a complex, multi-layer perceptron that's ready to train (or *fine-tune)*

Deep belief networks

As with auto encoders, we can also stack Boltzmann machines to generate a class known as deep belief networks (DBNs) as shown in figure 8.3.
In this scenario, the hidden layer of an RBM, *t*, acts as a visible layer for RBM *t+1*. The input layer of the initial RBM is the input layer for the whole network, and the greedy layer-wise pre-training works like this:

1. Train the initial RBM *t=1* using contrastive divergence with all the training samples.
2. Train the next RBM *t=2*. In view of the fact that the visible layer for *t=2* is the hidden layer of *t=1*, training begins by clamping the input sample to the visible layer of *t=1*, which is propagated frontward to the hidden layer of *t=1*. This data then serves to initiate contrastive divergence training for *t=2*.
3. Repeat the previous procedure for all the layers.
4. Alike the stacked auto encoders, after pre-training the network can become extensive by connecting one or more fully connected layers to the final RBM hidden layer. This forms a multilayer perceptron which can then be *fine-tuned* using backpropagation.

This procedure is akin to that of stacked auto encoders, but with the auto encoders replaced by RBMs and backpropagation replaced with the contrastive divergence algorithm.

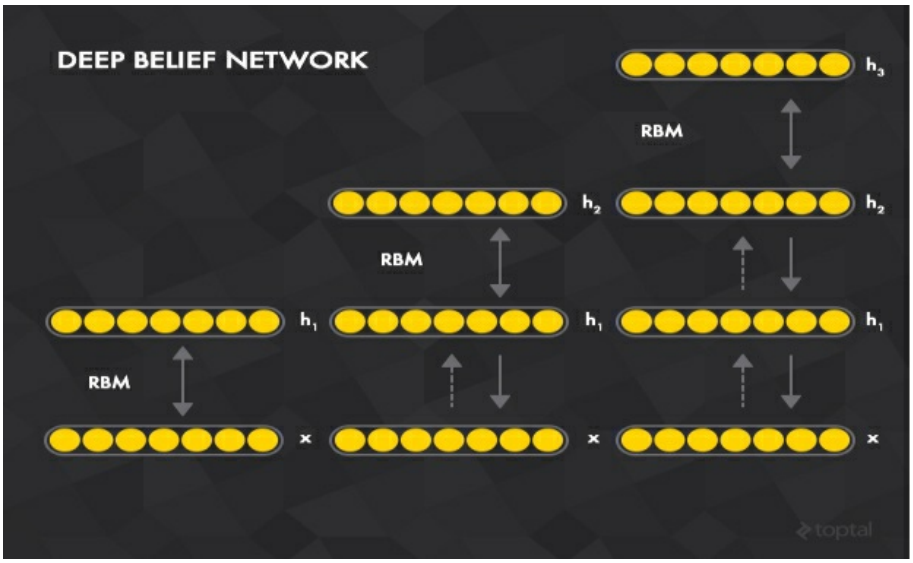


Figure 8.3 Deep belief networks

The RNN (recurrent neural network) and RBM (restricted Boltzmann machine)

The RNN-RBM is an energy-based model for the thickness assessment of temporal sequences, in which the attribute vector $v^{(t)}$ at time step t may be high-dimensional. It allows an explanation of multimodal conditional distributions of $v^{(t)}|A^{(t)}$, where

$$\mathcal{A}^{(t)} \equiv \{v_\tau | \tau < t\}$$

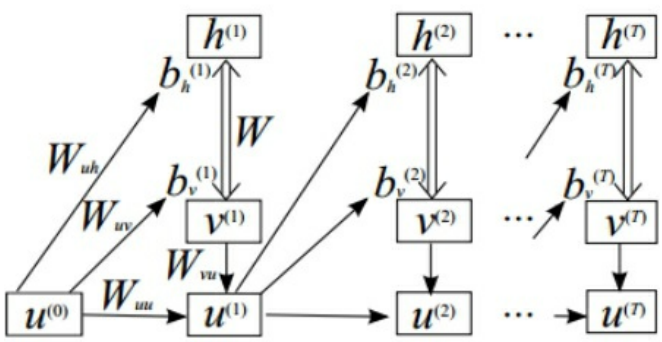
denotes the *sequence history* at time t, via a series of conditional RBMs (one at each time step) whose parameters $\mathbf{b_v^{(t)}}$, $\mathbf{b_h^{(t)}}$ depend on the output of a deterministic RNN with hidden units $\mathbf{u^{(t)}}$:

$$b_v^{(t)} = b_v + W_{uv} u^{(t-1)}$$
$$b_h^{(t)} = b_h + W_{uh} u^{(t-1)}$$

And the single-layer RNN recurrence relation is defined by:

$$u^{(t)} = \tanh(b_u + W_{uu} u^{(t-1)} + W_{vu} v^{(t)})$$

The resultant model is unrolled in time in the subsequent figure:



The entire probability distribution is known by the sum over the T time steps in a given series:

$$P(\{v^{(t)}\}) = \sum_{t=1}^{T} P(v^{(t)}|\mathcal{A}^{(t)})$$

Where, the right-hand side multiplicand is the marginalized probability of the tth RBM.

9. Deep Learning Applications

The recent ascent of artificial intelligence is grounded in the accomplishment of deep learning. Three noteworthy drivers have caused the leap forward of (deep) neural systems: The accessibility of immense measures of training data, effective computational foundations, and advances in the scholarly community. Accordingly, deep learning frameworks have begun to overcome traditional techniques, as well as human benchmarks in different undertakings like image characterization or face recognition. This makes the potential for some, troublesome new organizations to utilize deep learning to figure out how to take care of certain issues.

Applying deep learning to real-world problems

It is simpler than ever before to train a neural network. However, it is hardly ever the case that you can just take code from a tutorial and directly make it work for your purpose.

The value of pre-training

In the intellectual universe of machine learning, there is little focus on acquiring datasets. Rather, it is the opposite: in order to contrast deep learning techniques with other methodologies and guarantee that one method outperforms others, the standard strategy is to measure the performance on a standard dataset with a similar evaluation procedure. However, in real-world situations, it is less about showing that your new algorithm squeezing out an additional 1% in performance compared to another method. Rather it is about building a robust system which solves the required task with enough accuracy. As for all machine learning systems, this requires labeled training from which the algorithm can learn.

For many real-world problems it is unfortunately somewhat costly to get well-labeled training data. To elaborate on this issue, let us consider two cases.

Medical vision

If we are required to manufacture a system that detects lymph nodes in a human body in computed tomography (CT) images, we require annotated images where the lymph nodes are labeled. This is a somewhat time consuming task, as the images are 3D and the recognition of very small structures is required. Assuming that a radiologist earns 100$/h and can cautiously annotate 4 images per hour, this shows that we incur costs of 25 $ per image or 250k $ for 10,000 labeled images. Considering that we require more than a few physicians to label the same image to ensure close to a 100% diagnosis correctness, acquiring a dataset for the given medical task would easily exceed those 250k $ and that is obviously costly.

Credit scoring

If we would like to build a system that makes credit decisions, we want to know who is likely to default so that we can train a machine learning system to know this beforehand. Unluckily, you only know for sure if somebody

defaults when it occurs. Thus, a naive strategy would be to give loans of, say, 10k $ to everyone. However, this means that every person who defaults will cost us 10k $. This puts a very expensive price tag on each labeled data point. We can definitely lower the costs, but the key point is their cost is high.

Solution to the high cost

Pre-training is the solution to lowering the cost. The crucial idea of pre-training is that we initially train a neural network (or another machine learning algorithm) on a low-priced and large dataset in an associated domain or on noisy data in a similar domain. Even though this will not directly solve the original problem, it will give the neural network a rough idea of what your prediction problem looks like. Now, in a next step, the parameters of the neural network are additionally optimized on a much smaller and more expensive dataset of the problem you are actually trying to solve. This two-step procedure is depicted in figure 5.1 below.

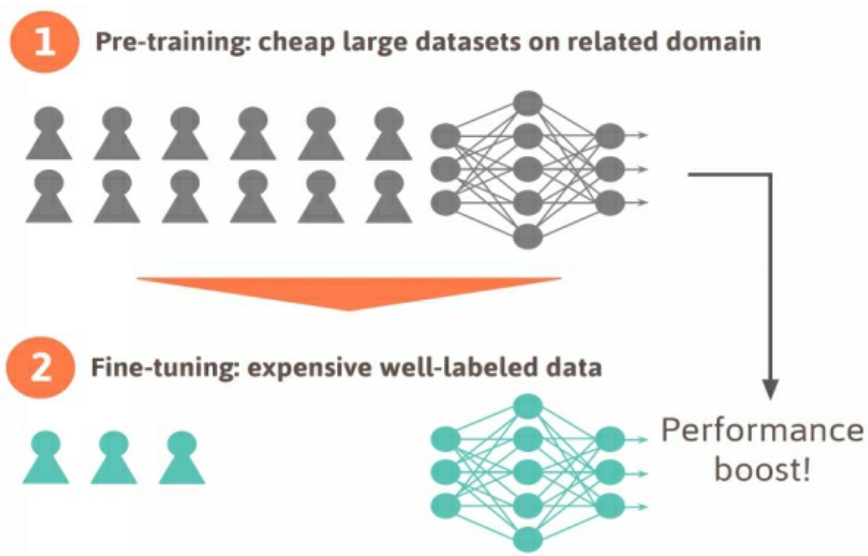


Figure 9.1: when training data is tough to get: First pre-train the neural network on a large and low-priced dataset in a related domain; secondly, fine-tune it on a pricey well-labeled dataset. This will result in a performance boost compared to just training on the small dataset.

When fine-tuning, the quantity of classes might alter: People often pre-train a neural network on a dataset like ImageNet with 1000 classes and then fine-tune it to their explicit problem, which, as is to be expected, has a different number of classes. This tells us about the last layer needs to be re-initialized.

The learning rate is then normally set a bit higher on the last layer as it needs to be learned from scratch, whereas the previous layers are trained with a lower learning rate. For some datasets like ImageNet, the features (the last fully connected layer) learned are so generic that they can be taken off-the-shelf and directly be used for several other computer vision problems.

Caveats of real-world label distributions

Once we have gathered data for both pre- and fine-tuning, we can move on and begin training our neural networks. Here comes one more huge difference among academia and the real world. In academia, the datasets are generally balanced. That means for supervised classification problems there are usually equally many samples per class. Underneath (figures 5.2 and 5.3) you find two examples: MNIST is a well-known dataset of handwritten digits containing approximately equally many samples of each digit. Food 101is one more example of an academic dataset that contains exactly 1000 images for each of the 101 food categories.

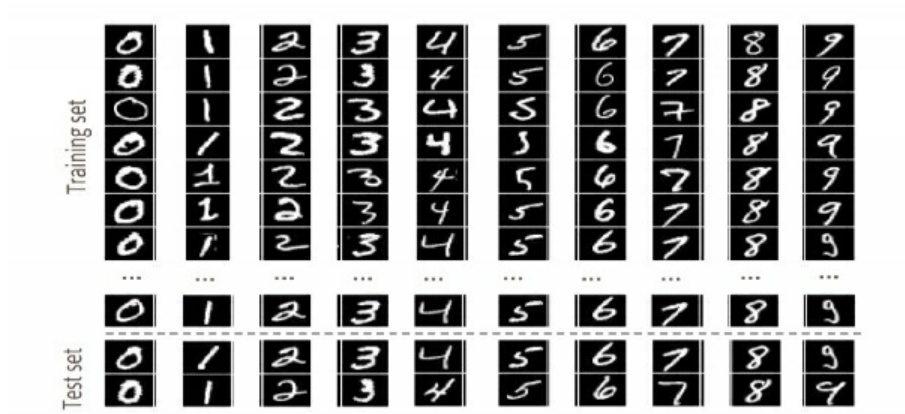


Figure 9.2 Balanced dataset; MNIST

Figure 9.3 Balanced dataset;Food101

Understanding the black box model

The primary goal in academia is to attain or outperform state-of-the-art performance, no matter what the model is like. When functioning on real-world applications it is often not enough to just design a model that performs well.

As an alternative, it is important to be able to:

- Know why and how a model can make incorrect predictions,
- Give some intuition why our model can perform better than any previous solution,
- Make sure that the model cannot be tricked.

Visual applications of deep learning

Some visual applications of deep learning are listed below:

- Adding sounds to silent movies.
- Automatic machine translation.
- Object classification in photographs.
- Automatic handwriting generation.
- Character text generation.

- Image caption generation.
- Automatic game playing.

Adding sounds to silent movies:

In this undertaking, the framework must synthesize sounds that coordinate with a silent video. The framework is trained utilizing 1000 examples of video with the sound of a drumstick striking distinct surfaces and creating different sounds. A deep learning model associates the video frames with a database of pre-recorded sounds in order to choose a sound that best matches what is going on in the scene. The framework was then assessed utilizing a Turing Test-like setup where humans had to figure out which video had the real or the fake (synthesized) sounds.

Automatic machine translation

This is where given words, expressions or sentences in one language, and translation of it into another language. Automatic machine translation has been around for quite a while, yet deep learning is accomplishing top outcomes in two particular territories:

- Automatic translation of text.
- Automatic translation of images.

Text translation can be performed with no preprocessing of the arrangement, enabling the algorithms to learn the dependencies between words and their mapping to a new language. Stacked networks of large LSTM recurrent neural networks are used to perform this translation.

As you would expect, convolutional neural networks are used to recognize images that have letters and where the letters are in the scene. Once identified, they can be turned into text, translated and the image recreated with the translated text. This is often called instant visual translation.

Object classification and detection in photographs

This task involves the classification of objects inside a photograph as one of a set of earlier known objects.

State-of-the-art outcomes have been attained on a benchmark; for instance, in this problem using very large convolutional neural networks. A breakthrough regarding this problem by Alex Krizhevsky et al. resulted in the ImageNet classification problem called ALEX-Net.

Figure 9.4 Object classification taken from ImageNet classification with deep convolutional neural networks

A more complicated variation of this task is called object detection and includes specifically identifying one or more objects within the scene of a photograph and drawing a box around them.

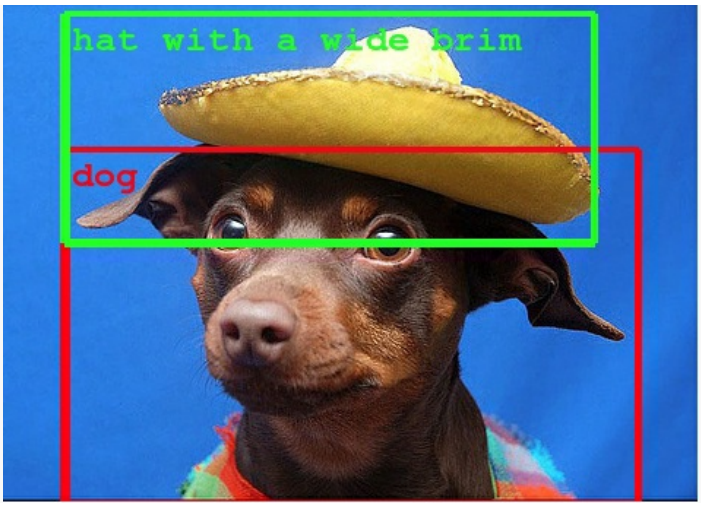


Figure 9.5 Object detection within photographs

Automatic handwriting generation

This is a something where, given a corpus of handwriting examples, new handwriting is produced for a specified word or phrase. The handwriting is given as a series of coordinate's utilizing a pen with which the handwriting samples were formed. By knowing this corpus, the relationship between the pen movement and the letters is learned and new examples can be created ad hoc.

What is fascinating is that different styles can be learned and then mimicked.

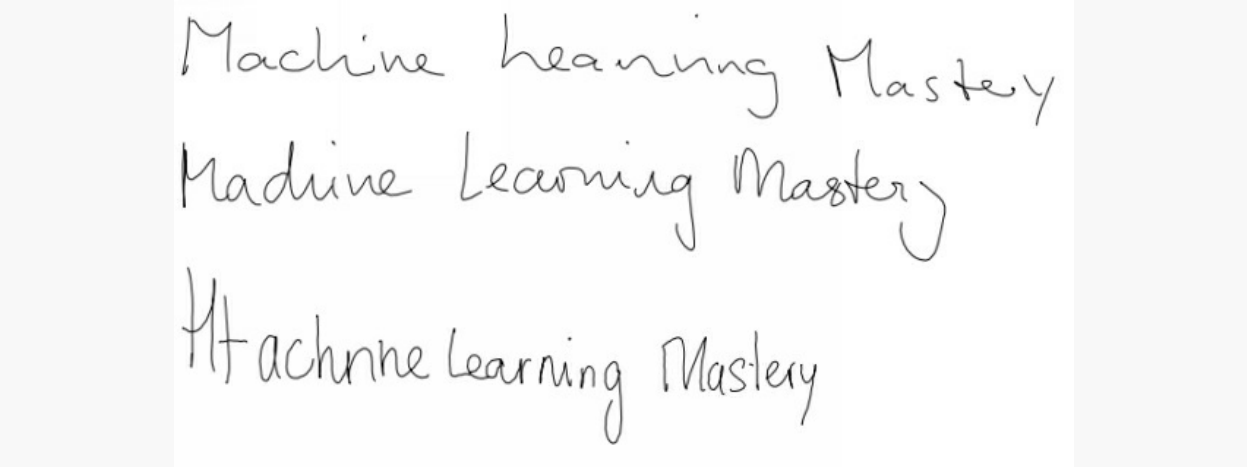


Figure 9.6 Handwriting

Automatic game playing

This is a job in which a model learns how to play a computer game based only on the pixels on the screen. This is the most complicated task in the field of deep reinforcement models and is the breakthrough that Deep Mind (now part of Google) is renowned for achieving.

Automatic image caption generation

Automatic image captioning is the task where certain images are given and the system must generate a caption that describes the contents of the image.

In 2014, there were a number of deep learning algorithms acquiring very exciting results on this problem, leveraging the work from top models for object classification and object detection in photographs.

Once objects in photographs are spotted and labels are generated for those objects, the next step is to turn those labels into a coherent sentence description. This is one of those outcomes that knocked my socks off and still does; it is very notable indeed!

Generally, the system involves the use of very large convolutional neural networks for the object detection in the photographs and then a recurrent neural network like an LSTM to turn the labels into a coherent sentence.

Automatic text generation

This is an appealing thing, where a corpus of text is well read and new text is generated from this model, wordbyword or character by character. The representation is capable of learning how to spell, punctuate, form sentences, and also confine it's style to that of the text in the corpus.

Huge recurrent neural networks are being utilized to gain knowledge on the relationship among items in the sequences of input strings and then generate text. More recently, LSTM recurrent neural networks are demonstrating great success regarding this problem using a character-based model, generating one character at time.

Andrej Karpathy provides many examples in his popular blog post on the topic, including:

- *Paul Graham essays*
- *Shakespeare*
- *Wikipedia articles (including the markup)*
- *Algebraic geometry (with LaTeX markup)*
- *Linux source code*
- *Baby names*

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

Figure 9.7 Automated text generation text

Applications using existing APIs

APIs are not merely software running on the other side of the Internet in a remote computer which one can get in the neighborhood. For instance, you may connect Bluetooth speakers to your laptop even if your machine has natural speakers. So, we are able to access the speaker remotely at the same time as we are sitting at our laptop.

Advantages of deep learning APIs

- A standard deep learning application requires weighty computation power in terms of GPUs and data storage/processing. For this reason, you have to set a workstation of your own (or use any of the cloud services) and utilize any system locally to access the workstation and run your applications.
- The local system is not weighed down by the computations.
- The latest functionalities can be integrated without difficulty.

Disadvantages of deep learning APIs

- Constructing an API may not be of use in terms of cost. An API needs both time and resources to build up and maintain, which tends to be tedious.
- You are restricted by the Internet connection. If your connection fails at any time, the whole system breaks down.
- Your application is open to the elements, security-wise, as anyone can access it without difficulty. Here, you have to add extra security layers

like username and password to access the service and limit how many times you can access it over a certain time period.

Open-source applications

A most excellent and helpful thing in the research community right now is its open-source frame of mind. Researchers are all set to share their findings, no matter what they achieved, so that deep learning research can grow; as a result, it's growing continuously.

Advantages of open source applications

- As the applications are open sourced, we have access to the each and every element of the application and can easily customize them according to one's requirements.
- A lot of developers from diverse organizations and experiences act as a team in the application. This makes the application better than the original version. Also, as many people can use it, it can be considered as being constantly tested and hence more ready for use.

Disadvantages of open source applications

- There may be less "ownership" of an open-source product, as there is no association backing them. So, if they break no one can be held responsible!
- There are clear licensing issues included and many companies would not prefer to release their projects so "openly".

10. Glossary of Some Useful Terms in Deep Learning

Layer

A layer is the highest-level building block in deep learning. A layer is a container that usually receives weighted input, transforms it with a set of mostly non-linear functions, and then passes on these values as output to the next layer. A layer is usually uniform, that is, it only contains one type of activation, pooling, convolution etc., so that it can be easily compared to other parts of the network. The first and last layers in a network are called the input and output layers, respectively, and all layers in-between are called hidden layers.

Unit

A unit refers to the activation function in a layer by which the inputs are transformed via a nonlinear activation function. Usually, a unit has several incoming connections and several outgoing connections. Pooling, convolution, and other input transforming functions are usually not referred to as units.

Activation function

A neural networks calculates a weighted sum of its input, adds a bias, and then decides whether it should be “considered” or not. The value of the weighted sum + bias can range between any intervals.

So, how do we decide whether the neuron should be considered or not? The activation functions fulfil this task; they check the value produced by a neuron and decide whether outside connections should consider this neuron as “fired” or not.

The difference between units and activation functions is that units can be more complex, that is, a unit can have multiple activation functions or a slightly more complex structure.

Feature engineering

Here, we give a practical explanation of feature engineering. It is the art of extracting useful patterns from data that will make it easier for machine

learning models to distinguish between classes. The features can be pixel values, shape, textures, position and orientation.

In a machine learning algorithm, the performance is based on the accuracy of these features as well as their identification and extraction. However, deep learning algorithms try to learn high-level features from data. In fact, deep learning reduces the task of developing a new feature extractor for each problem.

Let's consider the example of face recognition. When we as humans try to recognize faces, we concentrate on various features such as eyebrows, lips, the shape of the eyes, the pattern of the hair, and so on.

We recognize these features by combining low level features such as different types of curves around the eyes, eyebrows, lips, and so on.

A deep learning convolutional neural network is similar. It first identifies low level features, such as edges, and then learns to recognize curves and combines these curves to learn patterns of lips, eyes, and eyebrows. These different levels of features come from different layers of the network.

Feature Learning

Feature learning is a set of techniques that learn a feature, namely a transformation of raw data input into a representation that can be effectively exploited in machine learning tasks. This obviates manual feature engineering, which is otherwise necessary, and allows a machine to both learn a specific task (using the features) and learn the features themselves.

Feature learning is motivated by the fact that machine learning tasks such as classification often require input that is both mathematically and computationally convenient to process. However, real-world data, such as images, video, and sensor measurements, is usually complex, redundant, and highly variable. Thus, it is necessary to discover useful features or representations from raw data. Traditional hand-crafted features often require expensive human labor and often rely on expert knowledge. Also, they normally do not generalize well. This motivates the design of efficient feature learning techniques to automate and generalize this.

Feature learning can be divided into two categories: supervised and unsupervised feature learning, analogous to these categories in machine learning generally.

- In supervised feature learning, features are learned with labeled input data. Examples include supervised neural networks, multilayer perceptrons, and (supervised) dictionary learning.

- In unsupervised feature learning, features are learned with unlabeled input data. Examples include dictionary learning, independent component analysis, autoencoders, and various forms of clustering.

Gradient descent

Gradient descent is an algorithm used to find the local minimum of a function. By initially guessing the solution and using the function gradient at that point, we guide the solution in the negative direction of the gradient and repeat this technique until the algorithm eventually converges at the point where the gradient is zero – the local minimum. We essentially descend to the error surface until we arrive at a valley.

11. Useful References

1. https://www.mathworks.com/videos/series/introduction-to-deep-learning.html
2. https://www.datascience.com/blog/machine-learning-and-deep-learning-what-is-the-difference
3. http://www.deeplearningbook.org/contents/intro.html
4. https://machinelearningmastery.com/what-is-deep-learning/
5. https://www.analyticsvidhya.com/blog/2017/05/gpus-necessary-for-deep-learning/
6. https://en.wikipedia.org/wiki/Convolutional_neural_network
7. http://neuralnetworksanddeeplearning.com/chap6.html
8. https://www.tensorflow.org/tutorials/deep_cnn
9. http://deeplearning.net/tutorial/lenet.html
10. http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwo
11. http://deeplearning.net/tutorial/mlp.html
12. https://medium.com/merantix/applying-deep-learning-to-real-world-problems-ba2d86ac5837
13. https://machinelearningmastery.com/inspirational-applications-deep-learning/
14. http://www.expertsystem.com/machine-learning-definition/
15. https://www.toptal.com/machine-learning/machine-learning-theory-an-introductory-primer
16. https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/
17. https://www.datascience.com/blog/introduction-to-machine-learning-algorithms

Thank you !

Thank you for buying this book! It is intended to help you understanding Deep Learning. If you enjoyed this book and felt that it added value to your life, I ask that you please take the time to review it.

Your honest feedback would be greatly appreciated. It really does make a difference.