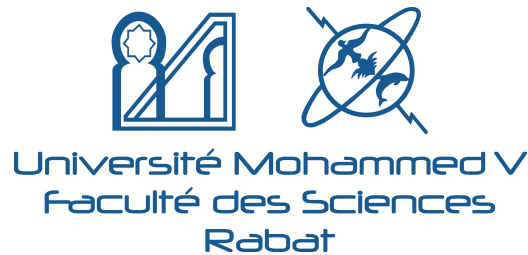


UNIVERSITÉ MOHAMMED V de Rabat
Faculté des Sciences



Département d'Informatique

Second year Master's IAO

Opinion mining using Deep Learning
Implicit Aspects Identification IAI

Prepared by:

ABDESSAMD CHALOUCHI

HAMZA HAMDAOUI

Supervised By:

Prof:Mohammed BENKHALIFA

Academic Year 2023-2024."

Abstract

English:

The research project titled "Data Mining: Opinion Mining using deep learning" focuses on the application of the Convolutional Neural Network (CNN) deep learning algorithm for implicit aspect sentiment analysis. We specifically delve into sentiment analysis regarding products and restaurants. The objective is to implement this algorithm for aspect-based sentiment analysis and evaluate its performance on two distinct datasets. The approach leverages deep neural networks to extract nuanced information related to opinions, providing an in-depth perspective on sentiments associated with various aspects in the domains of products and restaurants. The project aims to compare and assess the effectiveness of the CNN-based approach in sentiment analysis across different contexts.

French:

Le projet de recherche "Data Mining: Opinion Mining using deep learning" se concentre sur l'application de l'algorithme de deep learning CNN (Convolutional Neural Network) pour l'analyse des sentiments implicitement liés aux aspects. Nous nous penchons spécifiquement sur l'analyse des sentiments concernant les produits et les restaurants. L'objectif est de mettre en œuvre cet algorithme pour l'analyse des sentiments liés aux aspects et d'évaluer ses performances sur deux ensembles de données distincts. L'approche se base sur l'exploitation des réseaux neuronaux profonds pour extraire des informations subtiles liées aux opinions, offrant ainsi une perspective approfondie sur les sentiments associés aux différents aspects dans les domaines des produits et des restaurants.

Keywords: DM,ML,DL,ANN,CNN,RNN,LSTM

Contents

Abstract	i
Introduction	iv
1 Sentiment Analysis and Implicit Aspect Sentiment Analysis IASA	1
1.1 Introduction to Sentiment Analysis	1
1.1.1 What Is Sentiment Analysis?	1
1.1.2 Types Of Sentiment Analysis	1
1.1.3 How Does Sentiment Analysis Work?	2
1.2 Implicit Aspect Identification IAI	4
2 Deep Learning and Machine Learning Algorithms Overview	6
2.1 Introduction	6
2.2 Machine Learning	6
2.2.1 Supervised Learning	6
2.2.2 Unsupervised Learning	8
2.2.3 Reinforcement Learning	9
2.3 Deep Learning	9
2.3.1 Artificial Neural Networks ANN	9
2.3.2 Difference between Biological Neurons and Artificial Neurons . . .	10
2.3.3 ANN Architectures	11
2.3.4 Convolutional Neural Networks CNN	14
2.3.5 Recurrent Neural Networks RNN	15
2.3.6 Long Short-Term Memory LSTM	16
3 Convolutional Neural Networks for IAI	19
3.1 Introduction	19
3.2 Algorithm selection	19
3.3 CNN architectures for IAI	20
3.3.1 Data Preparation	20
3.3.2 Convolutional Layers	22
3.3.3 Pooling Layers	23
3.3.4 Fully-Connected Layers	23
3.3.5 Loss Functions	23
3.3.6 Training	24

4	Implementation of CNN for Implicit Aspect Sentiment Analysis	25
4.1	Introduction	25
4.2	Tools	25
4.2.1	Jupyter Notebook	25
4.2.2	Python	26
4.3	Libraries	26
4.3.1	Pandas	26
4.3.2	Mtplotlib	26
4.3.3	Numpy	27
4.3.4	keras	27
4.3.5	sklearn	27
4.3.6	tensorflow	27
4.4	Data Preparation	27
4.4.1	Reading the dataset	28
4.4.2	Tokenization of Text Data	28
4.4.3	GloVe Embeddings Integration	28
4.4.4	Preparing Data with One-Hot Encoding for Aspect-Based Sentiment Analysis Model Training	30
4.4.5	Data Splitting	31
4.5	Implementation of the CNN Algorithm	31
4.5.1	Optimizing CNN Hyperparameters for Implicit Aspect Analysis .	31
4.5.2	Model Construction with Discovered Hyperparameters	33
4.5.3	Training the Optimal Model	33
4.5.4	Evaluating the Best Model on the Test Set	35
4.5.5	Conclusion	35
5	Practical Demonstration and Results	36
5.1	Introduction	36
5.2	Validation Set Evaluation	36
5.2.1	Test Set Comparison	39
5.3	Discussion of Results	39
5.3.1	Model Performance	40
5.3.2	Generalization Capability	40
5.3.3	Domain-Specific Nuances	40
5.3.4	Overfitting and Underfitting	40
5.3.5	Conclusion	40
	Conclusion	41
	Bibliographie	42

Introduction

In recent years, the analysis of implicit aspects in textual data has gained significant attention in natural language processing and sentiment analysis. Implicit aspects refer to attributes or features that are not explicitly mentioned but can be inferred from the context of the text. Understanding implicit aspects is crucial for extracting deeper insights and sentiments from textual data, which can be valuable for various applications such as product reviews, social media sentiment analysis, and customer feedback analysis.

Convolutional Neural Networks (CNNs) have emerged as powerful tools for processing and analyzing textual data due to their ability to capture local patterns and hierarchical representations. By leveraging convolutional layers, max-pooling operations, and dense layers, CNNs can effectively learn features from textual input and make predictions about implicit aspects present in the text.

This report presents a comprehensive exploration of CNN implementation for implicit aspect analysis in textual data. We begin by detailing the data collection and preprocessing steps, followed by the construction and training of the CNN model. We discuss the choice of hyperparameters, model architecture, and training process, highlighting strategies for optimizing performance and ensuring robustness.

Subsequently, we present a practical demonstration and evaluation of the CNN model's performance on both validation and test datasets derived from diverse domains. Through validation set evaluation, we assess the model's accuracy and loss metrics, providing insights into its learning capabilities and generalization across different datasets.

Additionally, we conduct a comparative analysis of the model's performance on test datasets derived from Restaurant and Product domains, shedding light on its adaptability and effectiveness across varied domains.

Finally, we discuss the results obtained, identifying key findings, implications, and opportunities for future research. We highlight the significance of CNNs in implicit aspect analysis and emphasize the potential for further enhancements and applications in real-world scenarios.

Overall, this report aims to contribute to the understanding and advancement of CNN-based approaches for implicit aspect analysis in textual data, paving the way for innovative solutions in natural language processing and sentiment analysis.

Chapter 1

Sentiment Analysis and Implicit Aspect Sentiment Analysis IASA

1.1 Introduction to Sentiment Analysis

1.1.1 What Is Sentiment Analysis?

Sentiment analysis can be defined as analyzing the positive or negative sentiment of the customer in text. The contextual analysis of identifying information helps businesses understand their customers' social sentiment by monitoring online conversations.

As customers express their reviews and thoughts about the brand more openly than ever before, sentiment analysis has become a powerful tool to monitor and understand online conversations. Analyzing customer feedback and reviews automatically through survey responses or social media discussions allows you to learn what makes your customer happy or disappointed. Further, you can use this analysis to tailor your products and services to meet your customer's needs and make your brand successful.

Recent advancements in AI solutions have increased the efficiency of sentiment analysis algorithms. You can creatively use advanced artificial intelligence and machine learning tools for doing research and drawing out the analysis.

For example, sentiment analysis can help you automatically analyze 5000+ reviews about your brand by discovering whether your customers are happy or not satisfied with your pricing plans and customer service. Therefore, you can say that the application of sentiment is endless.

1.1.2 Types Of Sentiment Analysis

The sentiment analysis process mainly focuses on polarity, i.e., positive, negative, or neutral. Apart from polarity, it also considers the feelings and emotions (happy, sad, angry, etc.), intentions (interested or not interested), or urgency (urgent or not urgent) of the text.

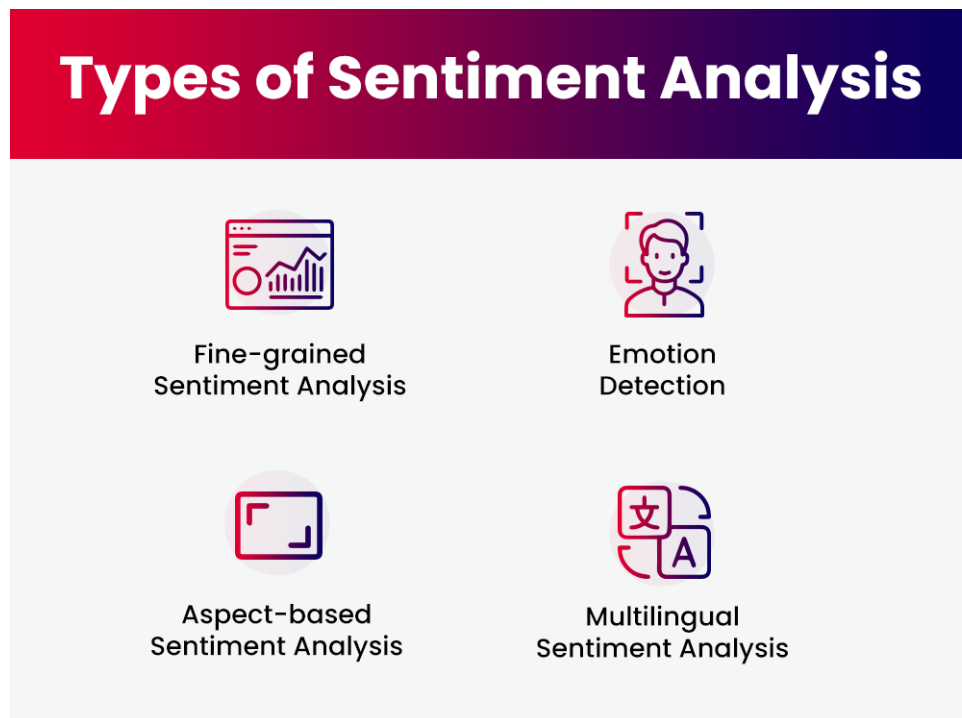


Figure 1.1.1: Types Of Sentiment Analysis

1.1.3 How Does Sentiment Analysis Work?

Sentiment analysis works with the help of natural language processing and machine learning algorithms by automatically identifying the customer's emotions behind the online conversations and feedback.

Depending on the amount of data and accuracy you need in your result, you can implement different sentiment analysis models and algorithms accordingly. Therefore, sentiment analysis algorithms comprise one of the three buckets below.

Rule-Based Approach

The rule-based system performs sentiment analysis based on manually crafted rules to identify polarity, subjectivity, or the subject of an opinion.

These rules contain different natural language processing techniques developed in computational linguistics, like stemming tokenization, parsing, lexicons (lists of words and expressions), or part of speech tagging.

For instance, you define two lists of polarized words, i.e., negative words (bad, worst, ugly, etc.) and positive words (good, best, beautiful, etc.). You have to count the number of positive and negative words in the text. If the number of positive words is greater than the number of negative words, the text will return a positive sentiment, and vice versa. If the number of negative and positive words is equal, then the text returns a neutral sentiment.

Since the rule-based system does not consider how words are combined in the sequence, this system is very naive. However, new rules can be added to support the new expression and vocabulary of the system by using more advanced processing techniques. But these

will also add complexity to the design and affect the previous results.

Automatic Approach

Unlike rule-based systems, the automatic approach works on machine learning techniques, which rely on manually crafted rules. Here, the sentiment analysis system consists of a classification problem where the input will be the text to be analyzed. It will return a polarity if the text, for example, is positive, negative, or neutral.

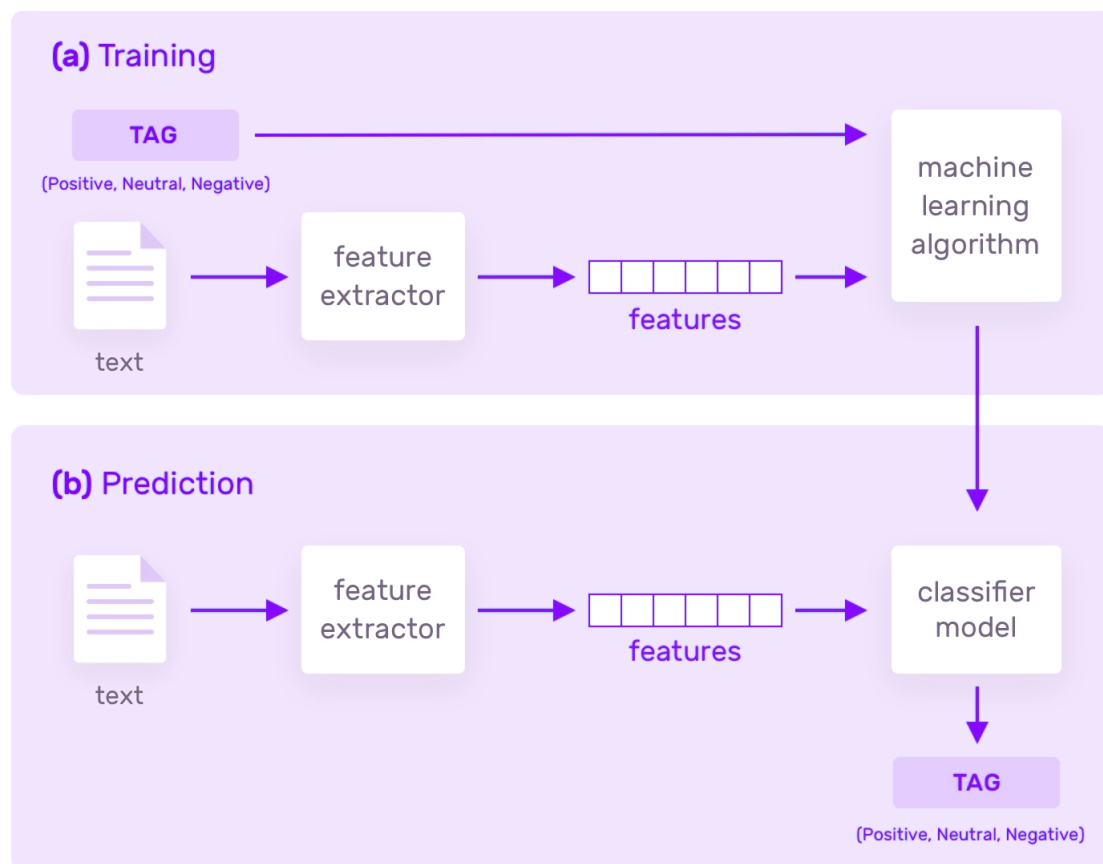


Figure 1.1.2: How Does Sentiment Analysis Work?

1. Training and Prediction

In the training process, your model links a particular input (i.e., text) to the corresponding output based on the test sample. The feature extractor will help transfer the input to the feature vector. These pairs of feature vectors and the tags provided are transferred to the machine learning algorithm to generate a model.

In the prediction process, the feature extractor transforms the unidentified text inputs into feature vectors. Further, these feature vectors generate predicted tags like positive, negative, and neutral.

2. Feature Extraction from Input Text

Machine learning text classifiers will transform the text extraction using the classical approach of bag-of-words or bag-of-n-grams with their frequency. A new feature extraction system is based on word embeddings known as word vectors.

This kind of representation helps to improve the performance of classifiers by making it possible for words with similar meanings to have similar presentations.

Classification Algorithms

Various classification algorithms involve statistical modeling, like naive bayes, support vector machines, deep learning, or logistic regression. Let us discuss them in detail below:

- **Naive Bayes:** It is a family of probabilistic algorithms that predict the category of a text by using the Bayes theorem.
- **Support Vector Machines:** It is a non-probabilistic model that uses a representation of the input text as a point in multi-dimensional space. Different text categories map to distinct regions within the space because the new texts are categorized based on their similarity with the existing text and the region they are mapping.
- **Deep Learning:** A family of algorithms that attempts to mimic the human brain with the help of artificial neural networks to process the data.
- **Linear Regression:** A family of algorithms in statistics that helps to predict some value (y) for a given set of features (x).

Hybrid Approaches

The hybrid model is a combination of elements of the rule-based approach and the automatic approach in one system. A massive advantage of this approach is that the results are often more accurate and precise than rule-based and automated approaches.

1.2 Implicit Aspect Identification IAI

Implicit Aspect Identification (IAI) is an advanced aspect of sentiment analysis that focuses on identifying sentiments related to specific aspects within text data. Traditional sentiment analysis may classify the overall sentiment of a document or sentence, but IAI goes a step further by pinpointing sentiments associated with implicit aspects or subtopics within the text.

For example, in a restaurant review, the overall sentiment might be positive, but IAI can help identify sentiments related to specific aspects such as food quality, service, ambiance, and pricing. This nuanced approach allows for a more granular understanding of sentiment, enabling businesses to address specific areas for improvement or capitalize on strengths.

Implicit aspects are elements that are not explicitly mentioned in the text but can be inferred from context or common knowledge. IAI employs advanced natural language processing techniques, machine learning algorithms, and contextual analysis to identify these implicit aspects and their associated sentiments accurately.

This method is particularly useful in domains where the overall sentiment might be a blend of positive and negative opinions on different aspects. By dissecting the sentiments associated with implicit aspects, organizations can gain deeper insights into customer feedback and make data-driven decisions to enhance their products or services.

Chapter 2

Deep Learning and Machine Learning Algorithms Overview

2.1 Introduction

The current discourse surrounding Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL) has led to significant confusion. AI encompasses computer systems capable of executing tasks that traditionally necessitate human intelligence, including visual perception, speech recognition, decision-making, and language translation. Deep learning is a subset of machine learning, which, in turn, falls under the broader umbrella of AI—an overarching term for intelligent computer programs. While all machine learning is considered AI, the reverse is not necessarily true. Machine learning represents a pivotal evolution in computer science, data analysis, software engineering, and AI, offering diverse areas for further exploration, such as algorithmic interpretability, robustness, privacy, fairness, causality inference, human-machine interaction, and security

2.2 Machine Learning

Machine learning refers to the study of computer systems that learn and adapt automatically from experience without being explicitly programmed. Machine learning describes the capacity of systems to learn from problem-specific training data to automate the process of analytical model building and solve associated tasks. In machine learning, tasks are generally classified into broad categories. These categories are based on how learning is received or how feedback on the learning is given to the system developed. Among the different types of Machine learning categories, a crucial distinction is drawn between supervised, unsupervised and Reinforcement learning:

2.2.1 Supervised Learning

Supervised learning, as we know is one of the most common types of ML learning methodology. The concept of this learning focuses on labeling of training data. Unlike unsupervised learning, the model first learns from the given training data. The training

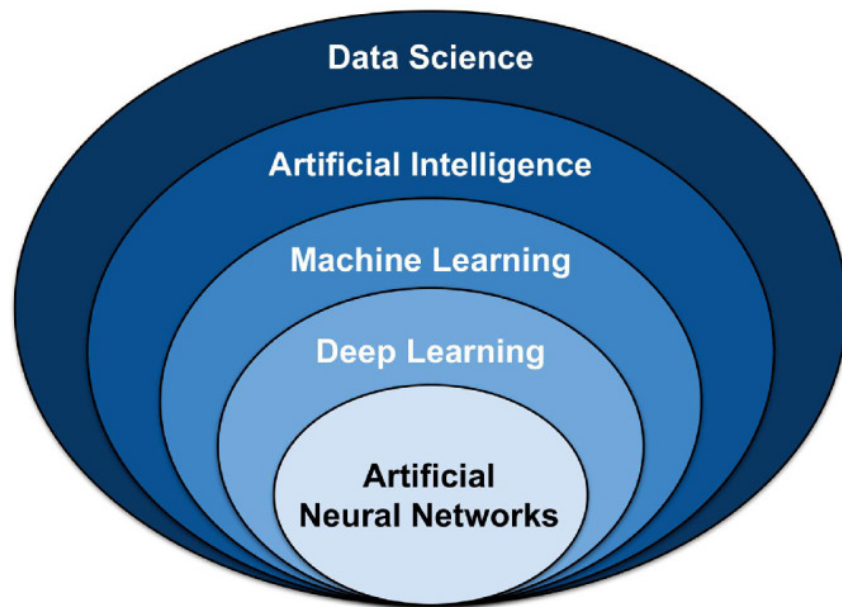


Figure 2.1.1: Relationship between DS,AI,ML,DL,ANN

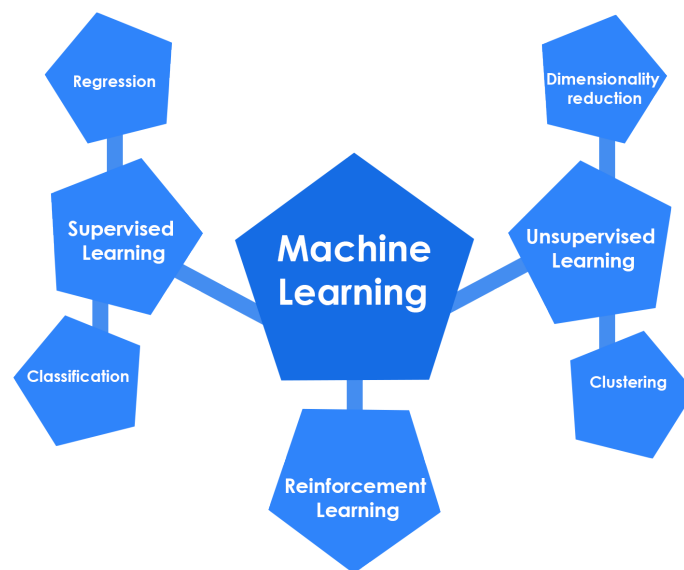


Figure 2.2.1: Overview of types of machine learning

data contains different patterns, which the model will learn. In other words, the outputs are already available. But, for a collection of data, various outputs are there. Supervising here means helping out the model to predict the right things. The data will contain inputs with corresponding outputs. This has hidden patterns in it. The algorithm will learn these patterns and will try to apply the same knowledge to unseen data. The aim is to predict future values. Mathematically, supervised learning can be shown as a linear

function, i.e., $y=f(x)$.

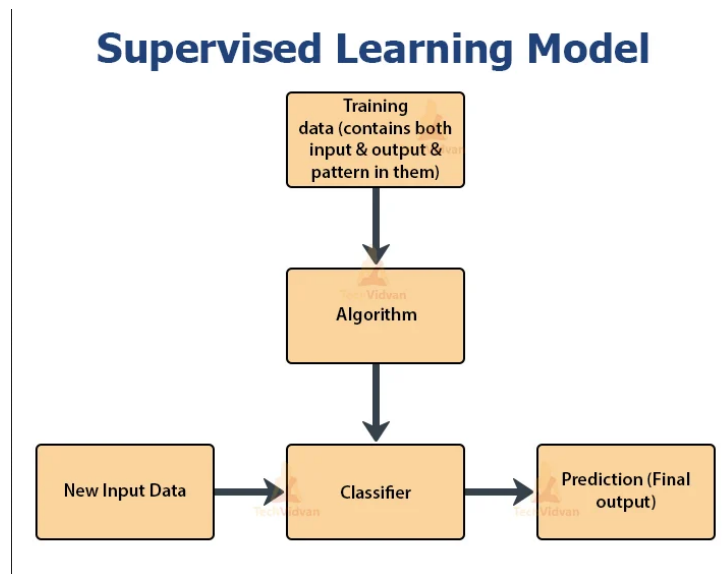


Figure 2.2.2: Supervised Learning

In supervised learning, we feed the algorithm's output into the system so that the machine knows the patterns before working on them. In other words, the algorithm gets trained on input data that has been labeled for a particular output. The model undergoes training until it can detect the underlying patterns and relationships between the input data and the output labels, enabling it to yield accurate labeling results when presented with never-before-seen data.

2.2.2 Unsupervised Learning

As the name suggests, unsupervised learning is a machine learning technique in which models are not supervised using a training dataset. Instead, the model itself finds hidden patterns and insights from the given data. It can be compared to learning, which takes place in the human brain while learning new things. It can be defined as:

[1] Unsupervised learning is a type of machine learning in which models are trained using unlabeled dataset and are allowed to act on that data without any supervision

Unsupervised learning is a type of machine learning in which models are trained using an unlabeled dataset and are allowed to act on that data without any supervision. Unsupervised learning cannot be directly applied to a regression or classification problem because, unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to find the underlying structure of a dataset, group that data according to similarities, and represent that dataset in a compressed format.

2.2.3 Reinforcement Learning

Unlike supervised and unsupervised learning, reinforcement learning has a feedback type of algorithm. In other words, for every result obtained, the algorithm gives feedback to the model under training. Reinforcement Learning is a type of learning methodology in ML along with supervised and unsupervised learning. But, when we compare these three, reinforcement learning is a bit different than the other two. Here, we take the concept of giving rewards for every positive result and make that the base of our algorithm. For an easier explanation, let's take the example of a human child. Kids often make mistakes. Adults try to make sure they learn from it and try not to repeat it again. In this case, we can take the concept of feedbacks. If the parents are strict, they will scold the children for any mistakes. This is a negative type of feedback. The child will remember it as if it does a certain wrong action, the parents will scold the kid. Then there is positive feedback, where the parent might praise them for doing something right. This type of learning is called enforced learning. Here, we enforce or try to force a correct action in a certain way. So, in short, reinforcement learning is the type of learning methodology where we give rewards of feedback to the algorithm to learn from and improve future results.

2.3 Deep Learning

Deep learning, characterized by the utilization of architectures with multiple hidden layers to learn hierarchical data representations, has gained popularity with the advancement of high-performance computing. In deep learning algorithms, data traverses through layers, progressively extracting features, and transmitting information for comprehensive representation. Unlike traditional machine learning, deep learning enables simultaneous learning and classification, eliminating sequential steps. This advantage makes it particularly effective for automating feature learning across diverse tasks. The era of deep learning has seen the development of various models, broadly categorized into discriminative (supervised) and generative (unsupervised) approaches, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), Generative Adversarial Networks (GANs), and Auto-Encoders (AEs). In contrast to machine learning algorithms that may require human correction, deep learning algorithms improve through repetition without human intervention, relying on large and diverse datasets for optimal outcomes. This evolution of machine learning involves artificial neural networks inspired by the structure of the human brain, with interconnected algorithms processing data in a non-linear fashion

2.3.1 Artificial Neural Networks ANN

Artificial Neural Network (ANN) learning exhibits robustness to errors in training data and has demonstrated success in tackling a diverse range of problems. These include learning real-valued, discrete-valued, and vector-valued functions, encompassing applications such as interpreting visual scenes, speech recognition, and formulating strategies for robot control.

The fascination with ANNs stems from their inspiration drawn from the intricacies of biological learning systems, particularly the highly complex networks of interconnected

neurons in the human brain. The human brain, composed of approximately $10^{11} - 10^{12}$ neurons, forms a densely woven web, with each neuron, on average, linked to $10^4 - 10^5$ other neurons. Remarkably, the human brain takes approximately 10^{-1} seconds, on average, to make complex decisions, highlighting the efficiency of highly parallel computation.

In the pursuit of mimicking this parallel processing and distributed representation, ANNs are constructed with densely interconnected simple units. Each unit takes multiple real-valued inputs and produces a single real-valued output.

2.3.2 Difference between Biological Neurons and Artificial Neurons

Biological neurons and artificial neurons, which are the fundamental components of biological brains and artificial neural networks (ANNs) respectively, have several differences. Here are some key distinctions:

Nature and Origin:

Biological Neurons: Found in the nervous system of living organisms, including humans. They are biological cells that process and transmit information through electrical and chemical signals.

Artificial Neurons: Created by humans for the purpose of building artificial neural networks. They are mathematical constructs designed to simulate the behavior of biological neurons in the context of machine learning and artificial intelligence.

Physical Structure:

Biological Neurons: Have a complex physical structure, including dendrites, a cell body (soma), an axon, and synapses. Information is transmitted through electrochemical signals.

Artificial Neurons: Typically simplified mathematical models that may not have a physical structure. They receive input signals, apply a mathematical function, and produce an output signal.

Communication:

Biological Neurons: Communicate through a combination of electrical impulses (action potentials) and chemical signals (neurotransmitters) across synapses.

Artificial Neurons: Communicate by passing numerical values through weighted connections. The output is often determined by applying an activation function to the weighted sum of inputs.

Learning Mechanism:

Biological Neurons: Learn through complex biological processes, including synaptic plasticity, which involves strengthening or weakening the connections between neurons based on experience.

Artificial Neurons: Learn by adjusting the weights of connections during a training phase. This process is often based on algorithms like backpropagation, which minimizes the difference between predicted and actual outputs.

Flexibility and Plasticity:

Biological Neurons: Exhibit high flexibility and plasticity, allowing the brain to adapt to new information, learn, and form memories.

Artificial Neurons: Lack the same level of flexibility and plasticity as biological neurons. Training artificial neural networks requires a large amount of labeled data and is generally more rigid than biological learning.

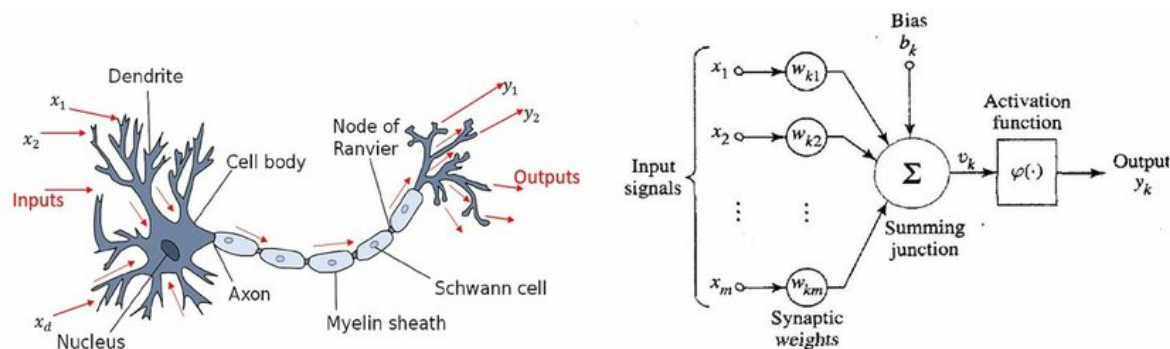


Figure 2.3.1: Biologic and Artificial Neural

Advantage of Using Artificial Neural Networks

Problem in ANNs can have instances that are represented by many attribute-value pairs. ANNs used for problems having the target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes. ANN learning methods are quite robust to noise in the training data. The training examples may contain errors, which do not affect the final output. It is used generally used where the fast evaluation of the learned target function may be required. ANNs can bear long training times depending on factors such as the number of weights in the network, the number of training examples considered, and the settings of various learning algorithm parameters.

2.3.3 ANN Architectures

Single-layer Neural Networks - Perceptrons

A single-layer neural network, also known as a perceptron, is the simplest form of a neural network. It consists of only one layer of artificial neurons (perceptrons) connected directly to the input. Perceptrons take multiple binary inputs, apply weights to them, sums the inputs and weights, and produces an output (binary). The perceptron output is determined by applying an activation function (usually a step function) to the weighted sum.

Mathematics Behind Perceptrons:

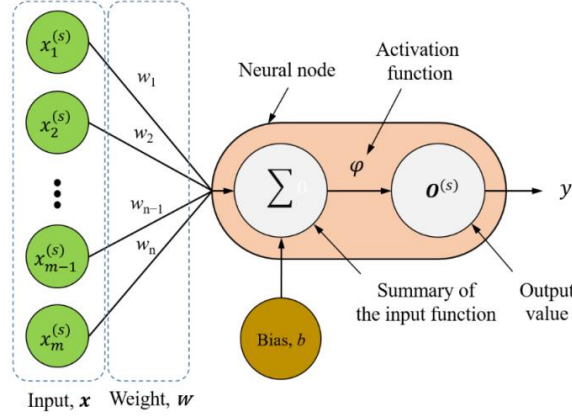


Figure 2.3.2: Single-neuron perceptron model

For a perceptron with n binary input features x_1, x_2, \dots, x_n , associated weights w_1, w_2, \dots, w_n , and a bias term b , the output y is computed as follows:

$$y = \text{step}\left(\sum_{i=1}^n (w_i \cdot x_i) + b\right)$$

Multi-layer Neural Networks MLP

A Multi-Layer Neural Network (MLP) is an extension of the single-layer perceptron, comprising an input layer, one or more hidden layers, and an output layer. Each layer contains artificial neurons, and connections between neurons have associated weights. MLPs can handle more complex relationships in data compared to single-layer networks.

Mathematics Behind MLP:

For a simple feedforward neural network with an input layer, one hidden layer, and an output layer, the computation of the output y involves the following steps:

1. **Input Layer:**

Input values x_1, x_2, \dots, x_n are fed into the network.

2. **Hidden Layer:**

Each neuron in the hidden layer computes a weighted sum of its inputs and passes the result through an activation function (commonly a sigmoid or hyperbolic tangent function).

$$a_j = \sigma\left(\sum_{i=1}^n (w_{ij}^{(1)} \cdot x_i) + b_j^{(1)}\right)$$

3. Output Layer:

The output layer neurons take the values from the hidden layer, compute a weighted sum, and apply another activation function.

$$y_k = \sigma\left(\sum_{j=1}^m (w_{kj}^{(2)} \cdot a_j) + b_k^{(2)}\right)$$

The training of MLP involves adjusting these weights and biases through a process called backpropagation and using optimization algorithms like gradient descent.

In summary, while perceptrons are limited to linearly separable problems, MLPs with multiple layers and non-linear activation functions can learn and approximate complex non-linear relationships in data.

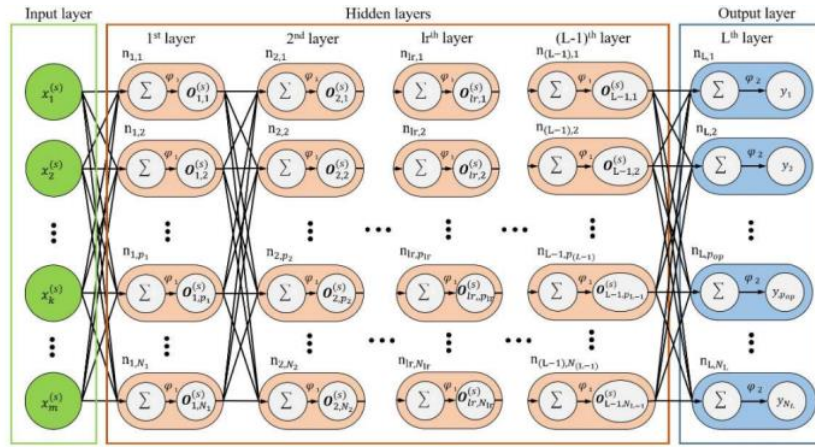


Figure 2.3.3: Structure of the multilayer perceptron

Activation Functions

Activation functions play a crucial role in artificial neural networks (ANNs) by introducing non-linearities into the network, enabling it to learn complex patterns and relationships in data. Here are some commonly used activation functions in ANNs:

Sigmoid Function (Logistic):

Formula:

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

Range: (0, 1)

Usage: Commonly used in the output layer for binary classification problems, where the goal is to produce probabilities.

Hyperbolic Tangent Function (tanh):

Formula:

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

Range: (-1, 1)

Usage: Similar to the sigmoid, but with an output range that is centered around zero. It is often used in hidden layers.

Rectified Linear Unit (ReLU):

Formula:

$$f(x) = \max(0, x)$$

Range: $[0, \infty)$

Usage: Commonly used in hidden layers due to its simplicity and effectiveness in overcoming the vanishing gradient problem. However, it can lead to the "dying ReLU" problem, where neurons become inactive and stop learning.

Choosing the appropriate activation function depends on the specific characteristics of the problem and the network architecture. Experimentation and tuning are often necessary to determine the most effective activation functions for a given task

2.3.4 Convolutional Neural Networks CNN

Convolutional Neural Networks (CNNs) stand as robust deep learning models widely employed across diverse tasks such as object detection, speech recognition, computer vision, image classification, and bioinformatics. Their versatility extends to success in time series prediction tasks as well. Operating as feedforward neural networks, CNNs utilize convolutional structures to automatically extract features from data. In contrast to conventional methods, CNNs eliminate the need for manual feature extraction by humans, showcasing an ability to learn and recognize features independently. Inspired by visual perception, CNNs consist of key components: the convolutional layer, pooling layer, and fully connected layer. Figure 5 illustrates a typical CNN architecture tailored for image classification and tasks

The Convolutional Layer, a pivotal element of CNNs,

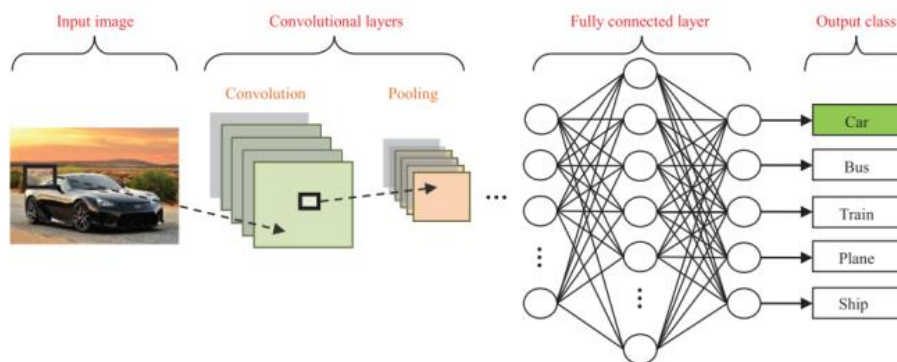


Figure 2.3.4: CNN pipeline for image classification

plays a crucial role in feature extraction through multiple convolutional layers. In image classification, lower layers capture fundamental features like texture, lines, and edges, while higher layers discern more abstract features. The convolutional layer comprises learnable convolution kernels, which are weight matrices typically of equal length, width,

and an odd number (e.g., 3x3, 5x5, or 7x7). These kernels are convolved with the input feature maps, sliding over the regions of the feature map and executing convolution operations. Figure 6 illustrates the schematic diagram of the convolution process. The

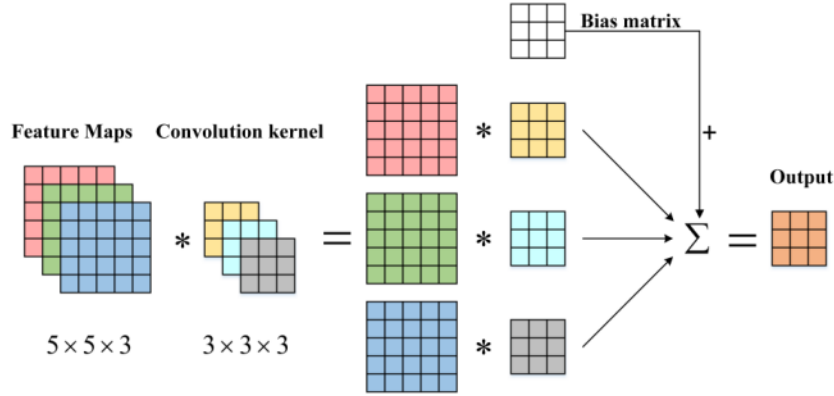


Figure 2.3.5: Schematic diagram of the convolution process

pooling layer, often succeeding the convolutional layer in a Convolutional Neural Network (CNN), plays a crucial role in down-sampling and dimensionality reduction, reducing the network's connections. Its main objectives include alleviating computational burdens, addressing overfitting, and enhancing the network's ability to recognize objects despite distortions or varied perspectives. Various pooling methods, such as Max Pooling and Average Pooling, contribute to creating more robust output feature maps. Figure 7 illustrates an example of Max Pooling, demonstrating a window sliding across input data processed by a pooling function.

The Fully Connected (FC) Layer, typically positioned at the end of a CNN architecture, follows principles similar to a conventional multi-layer perceptron neural network. Neurons in this layer are connected to all neurons in the preceding layer, receiving input from the last pooling or convolutional layer. The FC layer, operating as the classifier in the CNN, utilizes the flattened feature maps to make predictions

2.3.5 Recurrent Neural Networks RNN

Recurrent Neural Networks (RNNs) are deep learning models with internal memory, allowing them to capture sequential dependencies by considering the temporal order of inputs. In contrast to traditional neural networks treating inputs independently, RNNs use a loop to apply the same operation to each element in a series, with the current computation depending on both the current input and previous computations. RNNs prove valuable in natural language processing, video classification, and speech recognition, excelling at tasks like language modeling by understanding contextual information and capturing dependencies. However, a limitation of simple RNNs is their short-term memory, hindering information retention over long sequences. To address this, advanced RNN variants, including Long Short-Term Memory (LSTM), bidirectional LSTM, Gated Recurrent Unit (GRU), bidirectional GRU, Bayesian RNN, and others, have been developed. These variants overcome the short-term memory constraint and enhance the capability of RNNs in processing sequential data. Figure 8 depicts a simple

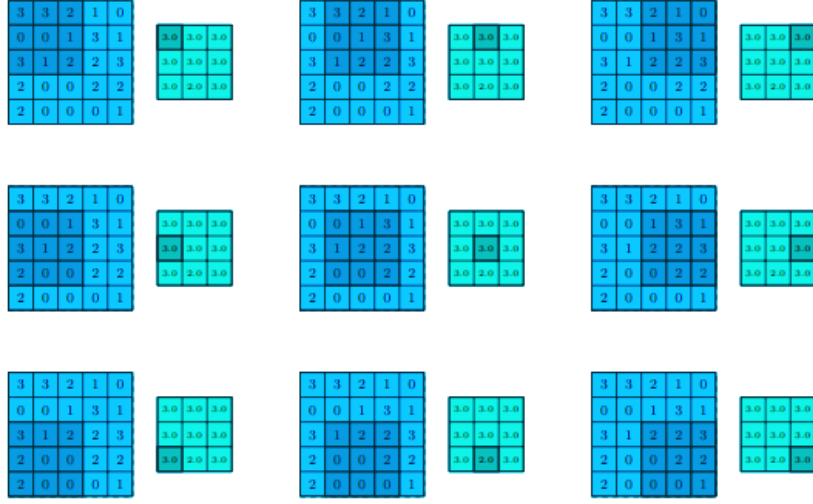


Figure 2.3.6: Computing the output values of a 3×3 max pooling operation on a 5×5 input

recurrent neural network, where the internal memory (t) is computed using Equation: $t = g(Wx_t + U t + b)$ (2) In this equation, $g()$ represents the activation function (typically the hyperbolic tangent), U and W are adjustable weight matrices for the hidden state (h), b is the bias term, and x denotes the input vector. RNNs have proven to be powerful

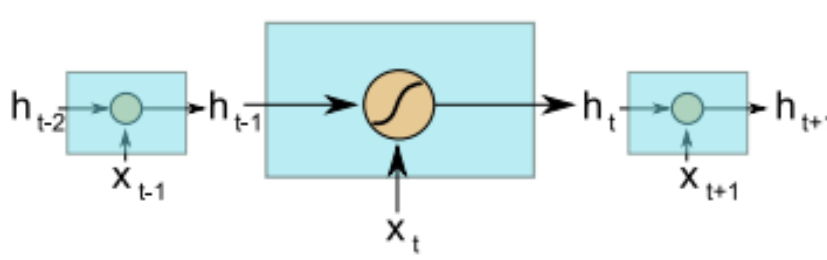


Figure 2.3.7: Simple RNN internal operation

models for processing sequential data, leveraging their ability to capture dependencies over time. The various types of RNN models, such as LSTM, bidirectional LSTM, GRU, and bidirectional GRU, have been developed to address specific challenges in different applications.

2.3.6 Long Short-Term Memory LSTM

Long Short-Term Memory (LSTM) stands as an advanced variant of Recurrent Neural Networks (RNN), addressing the challenge of capturing long-term dependencies. Introduced in 1997 and refined in 2013, LSTM has gained significant popularity for its effectiveness in retaining and utilizing information over extended sequences compared to standard RNNs. In the LSTM architecture, the current input at a specific time step and the output from the previous time step are fed into the LSTM unit, generating an output

for the subsequent time step. The final hidden layer of the last time step, often with all hidden layers, is commonly utilized for classification purposes. The LSTM network's overall architecture is depicted in Figure 2.3.8.

LSTM comprises three gates—input gate, forget gate, and output gate—each serving a distinct function in controlling information flow. The input gate determines how to update the internal state based on the current input and the previous internal state, while the forget gate decides how much of the previous internal state should be forgotten. Lastly, the output gate regulates the influence of the internal state on the system. The update mechanism within the inner structure of an LSTM is illustrated in Figure 2.3.9.

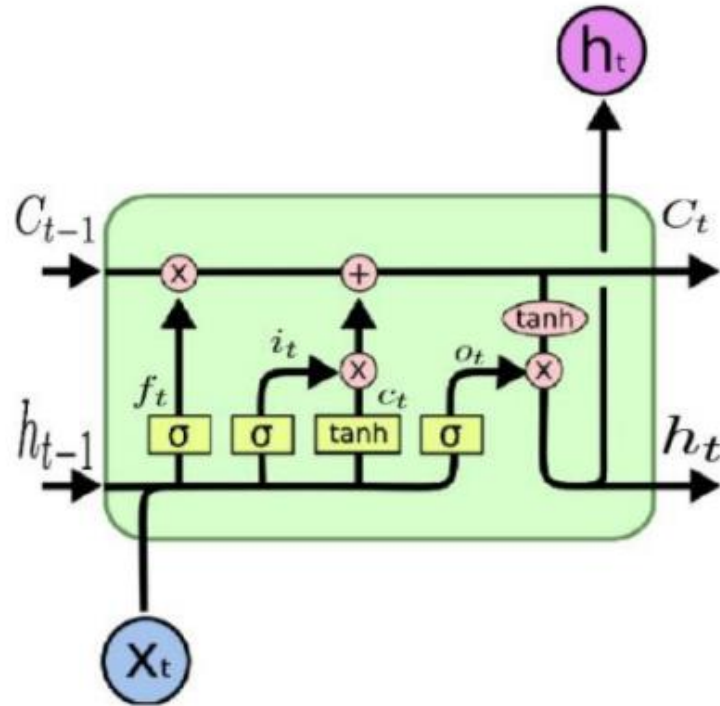


Figure 2.3.8: The high- level architecture of LSTM model

While standard LSTM has demonstrated promising performance in various tasks, it may struggle to comprehend input structures that are more complex than a sequential format. To address this limitation, a tree-structured LSTM network, known as S-LSTM consists of memory blocks comprising an input gate, two forget gates, a cell gate, and an output gate. While S-LSTM exhibits superior performance in challenging sequential modeling problems, it comes with higher computational complexity compared to standard LSTM

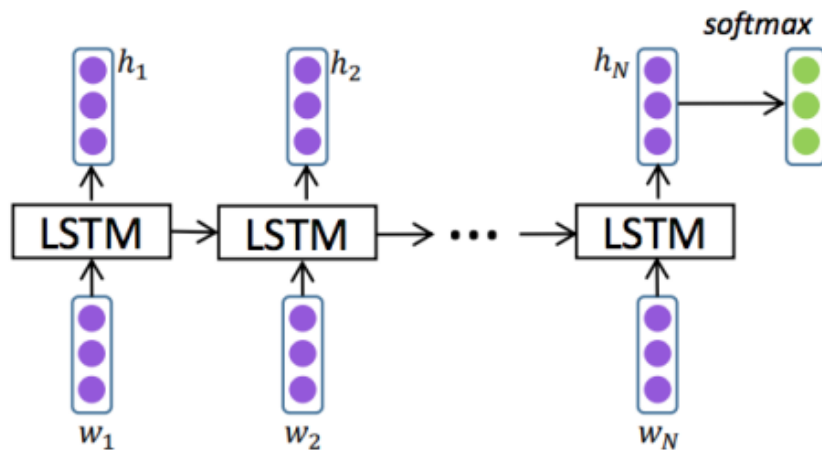


Figure 2.3.9: The inner architecture of a standard LSTM module

Chapter 3

Convolutional Neural Networks for IAI

3.1 Introduction

Implicit Aspect Identification (IAI) is a task related to Natural Language Processing (NLP) and sentiment analysis, where the goal is to automatically identify aspects or features of a given entity (such as a product or service) that are not explicitly mentioned in the text. Convolutional Neural Networks (CNNs) can be employed for IAI, because CNNs have emerged as a powerful tool for various natural language processing (NLP) tasks, including Implicit Aspect Identification (IAI). Their ability to learn complex non-linear relationships from text data makes them well-suited for capturing the implicit nature of aspects.

3.2 Algorithm selection

All three, CNNs, RNNs, and LSTMs, are powerful tools for Natural Language Processing (NLP) tasks like IAI. However, they have distinct advantages and disadvantages, making them suitable for different scenarios. Implicit Aspect Identification (IAI) involves uncovering aspects in text that are not explicitly mentioned but can be inferred from context and related words. CNNs stand out as a compelling choice for IAI due to several key advantages:

Capturing Local Context:

CNNs excel at capturing local context around words, a critical factor in identifying implicit aspects. Leveraging convolutional layers that slide over n-grams enables the extraction of features that reveal how neighboring words influence meaning, particularly valuable for understanding subtle cues where aspects aren't explicitly mentioned.

Learning Non-linear Relationships:

Unlike traditional methods reliant on explicit feature engineering, CNNs can automatically learn complex non-linear relationships between words and aspects. This ability allows them to identify nuanced patterns and associations that might be overlooked by simpler models, contributing to more effective identification of implicit aspects.

Efficient Feature Representation:

Word embeddings play a crucial role in IAI by representing semantic similarities and relationships between words. CNNs effectively utilize these embeddings, enabling the network to comprehend context and identify implicit aspects by leveraging semantic connections between nearby words.

Handling Varying Text Lengths:

In IAI tasks, text data can vary widely in length, from short reviews to longer sentences. CNNs' fixed-size window operation makes them flexible in handling diverse text lengths without requiring additional preprocessing, thus avoiding the introduction of artificial information that can impact performance.

Robustness to Noise and Errors:

Real-world text data often contains noise and errors. CNNs' pooling layers play a crucial role in summarizing information and mitigating the impact of individual errors, resulting in more stable and reliable performance in IAI tasks compared to models sensitive to noise propagation.

Potential for Multi-Lingual Applications:

CNNs can be adapted to work with different languages by using appropriate word embeddings and training data. This adaptability positions them as a promising choice for developing multilingual IAI systems, adding a layer of versatility to their utility.

In summary, CNNs provide a powerful and versatile approach for IAI, showcasing strengths in capturing context, learning non-linear relationships, handling different text lengths, and being robust to noise. Their ability to automatically learn from data makes them valuable tools for extracting insights from unstructured text data and understanding implicit aspects within.

3.3 CNN architectures for IAI

3.3.1 Data Preparation

Data preparation for Convolutional Neural Networks (CNNs) involves several key steps to ensure that the input data is in a suitable format for training and evaluation. Here are the main steps involved in data preparation for CNN algorithms:

Text Cleaning:

Remove any unnecessary characters, symbols, or special characters that do not contribute to the meaning of the text. Convert the text to lowercase to ensure uniformity and avoid treating the same word differently due to case differences.

Tokenization:

Split the text into individual words or tokens. Tokenization is the process of breaking down a text into its basic units (words or subwords). Libraries like NLTK or SpaCy can be used for tokenization.

Padding:

CNNs require input data with fixed dimensions. Since text sequences can vary in length, pad or truncate the sequences to ensure they all have the same length. Padding adds zeros to shorter sequences, while truncation removes excess tokens from longer sequences.

Word Embeddings:

Convert words into dense vectors using pre-trained word embeddings (e.g., Word2Vec, GloVe, FastText). Embeddings capture semantic relationships between words and provide a numerical representation for each word in the vocabulary.

Create Input Sequences:

Organize the text data into input sequences suitable for CNNs. This involves representing each document or sentence as a sequence of word embeddings. Each word in the sequence is represented by its corresponding word embedding.

Labeling:

Assign labels or categories to the text data. In the context of text classification, each text instance should be associated with a specific category or class.

Splitting Data:

Divide the dataset into training, validation, and test sets. The training set is used to train the model, the validation set helps tune hyperparameters, and the test set evaluates the model's performance on unseen data.

Data Batching:

Divide the training data into batches. Batching is essential for efficient training, as it allows the model to update its weights more frequently.

Data Augmentation (Optional):

For text data, data augmentation techniques are less common compared to image data. However, you can consider techniques like synonym replacement or slight paraphrasing to artificially increase the size of your training dataset.

One-Hot Encoding (Optional):

Convert categorical labels into one-hot encoded vectors, especially if you're using categorical cross-entropy as the loss function.

Text data is preprocessed by tokenizing words, performing stop word removal, and possibly applying stemming or lemmatization. Aspect categories and their relevant keywords are defined. Word embeddings are utilized to represent the words in the text. These embeddings capture semantic similarities between words, crucial for identifying implicit aspects.

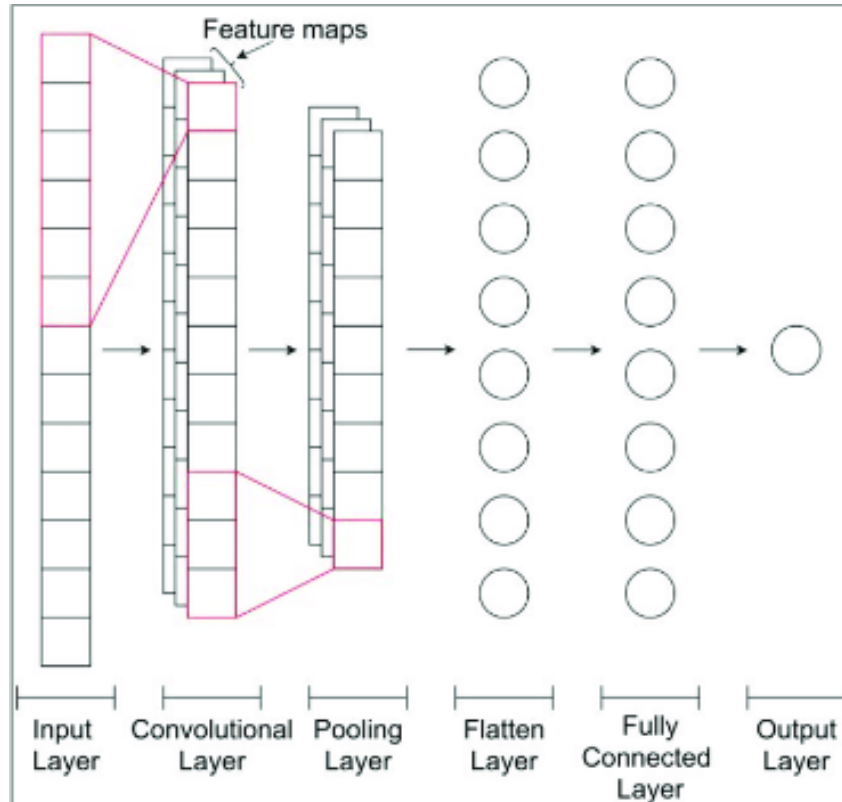


Figure 3.3.1: One-dimensional convolutional neural network (1D CNN) architecture.

3.3.2 Convolutional Layers

The preprocessed text is fed into a series of convolutional layers. Each layer convolves the input with a filter, capturing local patterns in the text. The filter size and number of filters determine the patterns the CNN learns. For IAI, filters are typically small (e.g., 3-5 words) to capture local context around potential aspect words. Each convolutional layer is followed by a non-linear activation function, such as ReLU, aiding the network in learning non-linear relationships. 1D Convolutional Neural Networks (1D CNNs) are a type of neural network architecture commonly used for processing sequential data, such as text, time series, or audio. They are an adaptation of traditional 2D CNNs, which are widely used in computer vision tasks, to handle one-dimensional sequences.

Here's how a 1D CNN works in the context of text processing:

Input Representation: In text processing, the input data is typically represented as a sequence of words, where each word is embedded into a high-dimensional vector space. These word embeddings capture semantic information about the words' meanings and relationships.

Convolutional Layers: The core of a 1D CNN is the convolutional layer. This layer applies a set of filters or kernels over the input sequence. Each filter slides along the input sequence and performs a convolution operation, which is essentially a weighted sum of the input values within a window.

Feature Extraction: As the filters slide over the input sequence, they extract local patterns or features. These features can capture various aspects of the text, such as word

combinations, syntactic structures, or semantic information.

3.3.3 Pooling Layers

Pooling layers downsample the output of the convolutional layers by summarizing information from a region of the input. This reduces dimensionality and improves the network's generalizability. Max pooling is commonly used in CNNs for IAI, selecting the maximum value from a region of the input.

3.3.4 Fully-Connected Layers

The output of the pooling layers is flattened and fed into fully-connected layers. These layers learn complex relationships between extracted features and predict the aspect category for each word in the text. The final layer uses a Softmax activation function to output a probability distribution over all aspect categories.

3.3.5 Loss Functions

The previous section has presented various layer-types of CNN architecture. In addition, the final classification is achieved from the output layer, which represents the last layer of the CNN architecture. Some loss functions are utilized in the output layer to calculate the predicted error created across the training samples in the CNN model. This error reveals the difference between the actual output and the predicted one. Next, it will be optimized through the CNN learning process. However, two parameters are used by the loss function to calculate the error. The CNN estimated output (referred to as the prediction) is the first parameter. The actual output (referred to as the label) is the second parameter. Several types of loss function are employed in various problem types. The following concisely explains some of the loss function types.

1-Cross-Entropy Cross-Entropy or Softmax Loss Function: This function is commonly employed for measuring the CNN model performance. It is also referred to as the log loss function. Its output is the probability p in $(0,1)$. In addition, it is usually employed as a substitution of the square error loss function in multi-class classification problems. In the output layer, it employs the softmax activation to generate the output within a probability distribution. The mathematical representation of the output class probability is:

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^N e^{a_k}} \quad (3.3.1)$$

Here, e^{a_i} represents the non-normalized output from the preceding layer, while N represents the number of neurons in the output layer. Finally, the mathematical representation of the cross-entropy loss function is:

$$H(p, y) = - \sum_i y_i \log(p_i) \text{ where } i \in [1, N].$$

2-Euclidean Loss Function: This function is widely used in regression problems. In addition, it is also the so-called mean square error. The mathematical expression of the

estimated Euclidean loss is:

$$H(p, y) = \frac{1}{2N} \sum_{i=1}^N (p_i - y_i)^2$$

3-Hinge Loss Function: This function is commonly employed in problems related to binary classification. This problem relates to maximum-margin-based classification; this is mostly important for SVMs, which use the hinge loss function, wherein the optimizer attempts to maximize the margin around dual objective classes. Its mathematical formula is:

$$H(p, y) = \sum_{i=1}^N \max(0, m - (2y_i - 1)p_i)$$

The margin m is commonly set to 1. Moreover, the predicted output is denoted as p_i , while the desired output is denoted as y_i .

3.3.6 Training

The CNN is trained using supervised learning, provided with labeled data containing text and corresponding aspect categories for each word. The network iteratively updates parameters to minimize the loss function, measuring the difference between predicted and true aspect labels.

In summary, CNNs provide a powerful and versatile approach for IAI, showcasing strengths in capturing context, learning non-linear relationships, handling different text lengths, and being robust to noise. Their ability to automatically learn from data makes them valuable tools for extracting insights from unstructured text data and understanding implicit aspects within.

Chapter 4

Implementation of CNN for Implicit Aspect Sentiment Analysis

4.1 Introduction

In the realm of sentiment analysis and opinion mining, delving into the nuances of Implicit Aspect Analysis (IAI) has become imperative for understanding the layers of sentiment concealed within textual data. Implicit aspects encapsulate the subtle and nuanced expressions that often elude explicit articulation but play a pivotal role in shaping opinions and sentiments. Harnessing the power of deep learning, this chapter unfolds the implementation of a Convolutional Neural Network (CNN) tailored for Implicit Aspect Analysis.

4.2 Tools

4.2.1 Jupyter Notebook

Jupyter Notebook is an open-source web-based application that enables the creation and sharing of documents combining live code, narrative text, equations, and visualizations. Created by Project Jupyter, the application supports the use of over 40 programming languages, including Python, R and Scala. Widely used in data science, machine learning, and scientific computing.



Figure 4.2.1: Jupyter Notebook

4.2.2 Python

Python is an open source, interpreted, cross-platform and object-oriented language used in different situations such as software development, data analysis or infrastructure management. Therefore, it does not just focus on a single area. Regardless of the platform used, Python offers great flexibility and compatibility for tasks to be performed. Thanks to its numerous libraries such as Pandas, Numpy, Scipy, Matplotlib, Scikit-Learn and TensorFlow.

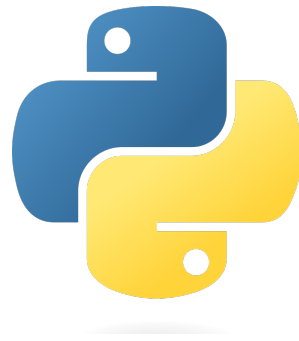


Figure 4.2.2: Python

4.3 Libraries

4.3.1 Pandas

Pandas is a Python library used for working with data sets. It provides a variety of tools for manipulation, analyzing data and visualizing data.



Figure 4.3.1: Pandas

4.3.2 Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations. It is a popular choice for data visualization in Python.

4.3.3 Numpy

NumPy (Numerical Python) is a software library used for scientific programming in Python, this free and open source library provides multiple functions, including the direct creation of multidimensional, these arrays are identified by a certain number of indices.



Figure 4.3.2: Pandas

4.3.4 keras

Keras is an open-source library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

4.3.5 sklearn

scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. Scikit-learn is a NumFOCUS fiscally sponsored project.

4.3.6 tensorflow

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

4.4 Data Preparation

At the foundation of our implementation lies the meticulous curation and preparation of datasets. In this section, we elucidate the journey from acquiring diverse datasets containing implicit aspects to the intricate process of cleaning, tokenization, and embedding. By ensuring a balanced and representative dataset, we lay the groundwork for training a CNN model capable of discerning implicit nuances across various domains.

4.4.1 Reading the dataset

By Using the pandas library, a widely-used tool for data manipulation and analysis in Python, the script sets the stage for efficient data handling. The core functionality of reading a dataset from a CSV file is achieved through the "pd.read_csv()" function, which loads the data into a pandas DataFrame. This tabular data structure allows for easy manipulation and analysis of the dataset. The "head()" method is then employed to provide a concise preview of the dataset by displaying the first few rows. This initial exploration step is crucial for understanding the structure and content of the data, enabling further preprocessing, analysis, and modeling tasks.

```
data = pd.read_csv('./RESTAURANT.data', sep=',')
data.head()
```

✓ 0.3s

	word	AspectCategory
0	setting	ambience
1	check	service
2	waiting	service
3	Indian	food
4	eating	food

Figure 4.4.1: Reading the dataset into a DataFrame

4.4.2 Tokenization of Text Data

The initial step involves downloading NLTK(Natural Language Toolkit is a renowned Python library for natural language processing tasks) resources essential for tokenization, facilitating the subsequent text processing tasks. Tokenization, a fundamental preprocessing step in natural language processing, is then applied to the text data contained in the 'word' column of the DataFrame data. Through the application of the "word_tokenize" function from NLTK, each textual entry is segmented into individual tokens. These tokenized representations are then stored in a new column labeled 'tokens' within the DataFrame, paving the way for further analysis and modeling tasks. This process of tokenization is pivotal for extracting meaningful insights from textual data, laying the groundwork for subsequent text analysis and machine learning endeavors.

4.4.3 GloVe Embeddings Integration

In this code snippet, we integrate pre-trained GloVe (Global Vectors for Word Representation) embeddings into our natural language processing pipeline. Initially, the GloVe embeddings are loaded from a specified file path using the Gensim library's "load_word2vec_format()" function, indicating that the embeddings are stored in plain text format. Subsequently, an embedding function, "get_embedding()", is defined to

```

nltk.download('punkt')      "punkt": Unknown word.
data['tokens'] = data['word'].apply(word_tokenize)
data.head()
✓ 0.9s

```

[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\abdes\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!

	word	AspectCategory	tokens
0	setting	ambience	[setting]
1	check	service	[check]
2	waiting	service	[waiting]
3	Indian	food	[Indian]
4	eating	food	[eating]

Figure 4.4.2: Tokenization of Text Data

retrieve the embedding vector for a given token from the loaded GloVe model. This function handles cases where the token is not found in the model by returning a vector of zeros. The tokenized text data, stored in a column named 'tokens' within the DataFrame data, undergoes transformation as each token is mapped to its corresponding embedding vector using a lambda function applied via the apply() method. The resulting embeddings are then stored in a new column labeled 'embeddings'. This integration of GloVe embeddings enriches our dataset with dense vector representations, empowering downstream natural language processing tasks with semantic understanding and context-awareness.

```

# Load GloVe embeddings
glove_model = KeyedVectors.load_word2vec_format('./glove.6B.50d.txt', binary=False)

# embedding function
def get_embedding(token):
    try:
        return glove_model[token]
    except KeyError:
        return [0.0] * 50

data['embeddings'] = data['tokens'].apply(lambda tokens: [get_embedding(token) for token in tokens])
data.head()
✓ 11.8s

```

	word	AspectCategory	tokens	embeddings
0	setting	ambience	[setting]	[[0.071259, 0.089283, -0.39261, -0.12705, 0.34...
1	check	service	[check]	[[0.16788, 0.10003, 0.79083, -0.32853, 0.12775...
2	waiting	service	[waiting]	[[0.78954, 0.10954, 0.49796, -0.85129, 0.7169...
3	Indian	food	[Indian]	[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...
4	eating	food	[eating]	[[0.52979, -0.35266, -1.3524, -0.59844, 0.8774...

Figure 4.4.3: GloVe Embeddings Integration

The following code snippet undertakes the crucial task of padding embeddings to enforce uniformity in length, a prerequisite for many machine learning models. Initially, a maximum length of 50 is specified for the embeddings. Subsequently, the "pad_sequences()" function from the Keras library is employed to pad each sequence of embeddings in the 'embeddings' column of the DataFrame data. This process ensures that all sequences are extended or truncated to match the specified maximum length, with zeros appended at the end to achieve post-padding. The resulting padded embeddings are then stored in a new column labeled "padded_embeddings".

```
# Pad embeddings
maxlen = 50
data['padded_embeddings'] = data['embeddings'].apply(lambda x: pad_sequences(x, maxlen=maxlen, dtype='float32', padding='post')[0])

# data.shape
```

✓ 0.0s

Figure 4.4.4: Pad Embeddings

4.4.4 Preparing Data with One-Hot Encoding for Aspect-Based Sentiment Analysis Model Training

the input features (X) are constructed by converting the padded embeddings from the 'padded_embeddings' column of the DataFrame into a NumPy array. These embeddings serve as the basis for the model to learn from the textual data. Next, the target labels (y) are extracted from the 'AspectCategory' column of the DataFrame, representing the aspect categories associated with each data instance. Concurrently, a list of unique aspect categories is generated and stored in the variable 'aspect_labels' for reference. Subsequently, the target labels undergo one-hot encoding using the OneHotEncoder class from scikit-learn. This transformation ensures that the categorical aspect labels are represented in a format suitable for machine learning algorithms, enabling effective training and evaluation. By one-hot encoding the target labels, each aspect category is represented as a binary vector, where a '1' indicates the presence of the category and '0' otherwise. This comprehensive data preparation process sets the stage for subsequent model training and evaluation, facilitating the development of accurate and robust models for aspect-based sentiment analysis.

```
# Prepare X and Y
X = np.array(data['padded_embeddings'].tolist())
y = data['AspectCategory'].values
aspect_labels = data['AspectCategory'].unique().tolist()

# One-hot encode Y
encoder = OneHotEncoder(sparse=False)
y = encoder.fit_transform(y.reshape(-1, 1))
```

✓ 0.0s

Figure 4.4.5: One-hot Encoding

4.4.5 Data Splitting

This code snippet delineates the crucial step of partitioning the dataset into training, validation, and test sets to facilitate robust model training, validation, and evaluation processes. Initially, the dataset is split into training and temporary sets, with 80% of the data allocated for training and the remainder held temporarily. Subsequently, the temporary set is further divided into validation and test sets, each comprising 50% of the remaining data. This stratified splitting ensures that the distribution of aspect categories remains consistent across all subsets, thereby preventing biases in model training and evaluation. By specifying a random seed of 42, the splitting process becomes reproducible, enabling consistent results across multiple executions. Through this systematic data partitioning, the machine learning model can be trained on the training set, fine-tuned and validated on the validation set, and ultimately evaluated for generalization performance on the test set. This rigorous approach to data splitting forms a foundational step in the development of robust aspect-based sentiment analysis models.

```
# Split data
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

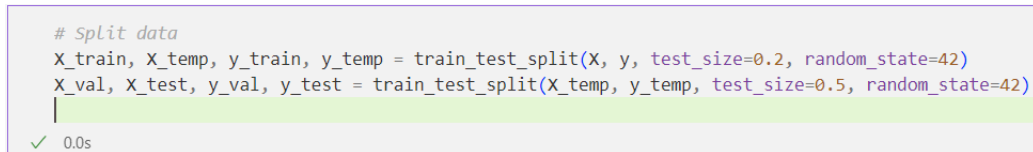


Figure 4.4.6: One-hot Encoding

4.5 Implementation of the CNN Algorithm

This section is dedicated to detailing the operationalization of the Convolutional Neural Network (CNN) architecture tailored for Implicit Aspect Analysis (IAI). Herein, we delve into the intricacies of configuring the neural network, outlining the specific layers and mechanisms employed to capture implicit aspects within textual data.

4.5.1 Optimizing CNN Hyperparameters for Implicit Aspect Analysis

To ensure the Convolutional Neural Network (CNN) achieves optimal performance in discerning implicit aspects, a meticulous search for the most effective hyperparameters is undertaken. In this process, various combinations of hyperparameters are systematically explored to identify the configuration that maximizes the model's accuracy on the validation set. By iterating over different values for filters, kernel sizes, and layer sizes, the CNN architecture is fine-tuned to enhance its sensitivity to implicit nuances within textual data. Through rigorous experimentation and evaluation, the hyperparameters yielding the highest validation accuracy are determined, providing crucial insights into the optimal configuration for implicit aspect analysis using CNNs.

We starting by defining a lists containing different values for the number of filters, kernel sizes, and layer sizes, respectively. These values represent the hyperparameters that will be explored during the grid search.

```

# Hyperparameter
filters_values = [64, 128, 256]
kernel_sizes_values = [3, 5, 7]
layer_sizes_values = [32, 64, 128]

best_accuracy = 0
best_hyperparameters = {}

```

Figure 4.5.1: Hyperparameters

This following nested loop iterates over all combinations of hyperparameters defined in the lists "filters_values", "kernel_sizes_values", and "layer_sizes_values". For each combination, a CNN model is constructed and trained, and its performance is evaluated on the validation set. Finally, the best hyperparameters yielding the highest validation accuracy are selected and printed out.

```

for filters in filters_values:
    for kernel_size in kernel_sizes_values:
        for layer_size in layer_sizes_values:
            # Build model with current hyperparameters
            model = Sequential()
            model.add(Embedding(input_dim=vocabulary_size, output_dim=50, input_length=maxlen))
            model.add(Conv1D(filters=filters, kernel_size=kernel_size, activation='relu')) "relu": Unknown word.
            model.add(MaxPooling1D(pool_size=2))
            model.add(Conv1D(filters=filters, kernel_size=kernel_size, activation='relu')) "relu": Unknown word.
            model.add(MaxPooling1D(pool_size=2))
            model.add(Flatten())
            model.add(Dense(layer_size, activation='relu')) "relu": Unknown word.
            model.add(Dropout(0.5))
            model.add(Dense(y.shape[1], activation='softmax')) "softmax": Unknown word.

            model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']) "crossentropy": U

            # Train model
            history = model.fit(X_train, y_train, epochs=20, batch_size=64, validation_data=(X_val, y_val), verbose=0)

            # Evaluate model on validation set
            _, val_accuracy = model.evaluate(X_val, y_val, verbose=0)

            # Update best hyperparameters if the current configuration is better
            if val_accuracy > best_accuracy:
                best_accuracy = val_accuracy
                best_hyperparameters = {'filters': filters, 'kernel_size': kernel_size, 'layer_size': layer_size}

print("Best Hyperparameters:", best_hyperparameters)
print("Best Validation Accuracy:", best_accuracy)

```

✓ 1m 1.1s

Best Hyperparameters: {'filters': 128, 'kernel_size': 5, 'layer_size': 64}
 Best Validation Accuracy: 0.8333333134651184

Figure 4.5.2: Best hyperparameters

Let's explain the code in more detail:

Model Construction:

Within the loop, a new Sequential model is instantiated to explore each combination of hyperparameters systematically. Leveraging Keras' Sequential API, the model architecture unfolds sequentially. It commences with the incorporation of an embedding layer, facilitating the transformation of word indices into dense vector representations. Subsequently, two Conv1D layers are added in succession, each accompanied by a MaxPooling1D layer. These layers collaboratively contribute to feature extraction and dimensionality reduction, fostering the model's ability to capture nuanced patterns in the input data. A Flatten layer intervenes to reshape the feature maps into a one-dimensional array, preparing them for processing through densely connected layers. Two Dense layers, activated by Rectified Linear Units (ReLU), follow suit, imparting nonlinear transformations to the data. A Dropout layer is strategically inserted to mitigate overfitting by randomly deactivating a fraction of neurons during training. Finally, a Dense layer employing softmax activation crowns the architecture, enabling multi-class classification by generating probabilities for each aspect category.

Model Compilation, Training, and Hyperparameter Selection:

The compilation and training phase, intricately entwined with hyperparameter selection, orchestrate a meticulous journey towards optimal model performance. As the model embarks on its training expedition, configured with categorical cross-entropy loss, the Adam optimizer, and accuracy as the guiding metric, it navigates through epochs, guided by a fixed batch size. Throughout this iterative process, the model's validation accuracy serves as a beacon, meticulously scrutinized to discern the zenith of performance. Should a validation accuracy surpass the previously recorded best, a pivotal milestone is achieved, heralding the mantle of best accuracy, alongside the corresponding hyperparameters, meticulously enshrined for future refinement. This seamless integration of compilation, training, and hyperparameter selection illuminates the architectural nuances essential for the zenith of efficacy in implicit aspect analysis.

4.5.2 Model Construction with Discovered Hyperparameters

With the meticulous determination of hyperparameters through rigorous evaluation, the construction of the optimal model commences. Sequentially built, the model begins with an embedding layer, translating word indices into dense vector representations. Two Conv1D layers, each with the identified optimal number of filters and kernel size, are added, followed by MaxPooling1D layers for feature extraction. The Flatten layer reshapes the feature maps for dense layer processing. A Dense layer, activated by ReLU, imparts nonlinear transformations, while Dropout reduces overfitting. Finally, a Dense layer with softmax activation enables multi-class classification. Compiled with categorical cross-entropy loss and Adam optimizer, the model awaits training and validation.

4.5.3 Training the Optimal Model

The culmination of model construction leads to the pivotal phase of training, where the best model is honed to achieve peak performance. With "X_train" and "y_train"

```

# Build the best model using the discovered hyperparameters
best_model = Sequential()
best_model.add(Embedding(input_dim=vocabulary_size, output_dim=50, input_length=maxlen))
best_model.add(Conv1D(filters=best_hyperparameters['filters'], kernel_size=best_hyperparameters['kernel_size'], activation='relu'))
best_model.add(MaxPooling1D(pool_size=2))
best_model.add(Conv1D(filters=best_hyperparameters['filters'], kernel_size=best_hyperparameters['kernel_size'], activation='relu'))
best_model.add(MaxPooling1D(pool_size=2))
best_model.add(Flatten())
best_model.add(Dense(best_hyperparameters['layer_size'], activation='relu')) "relu": Unknown word.
best_model.add(Dropout(0.5))
best_model.add(Dense(y.shape[1], activation='softmax')) "softmax": Unknown word.
best_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']) "crossentropy": Unknown word.
✓ 0.1s

```

Figure 4.5.3: Model Construction with Discovered Hyperparameters

as guiding forces, the model undergoes training across 20 epochs, with a fixed batch size of 64. Validation during training is ensured by incorporating the validation data ("X_val", "y_val"). This iterative process facilitates the fine-tuning of model parameters, optimizing its ability to generalize to unseen data. Through meticulous weight adjustments guided by observed performance metrics, the model evolves to maximize its efficacy in implicit aspect analysis.

```

# Train the best model
best_history = best_model.fit(X_train, y_train, epochs=20, batch_size=64, validation_data=(X_val, y_val))
✓ 2.1s

```

Figure 4.5.4: Training the Optimal Model

4.5.4 Evaluating the Best Model on the Test Set

After training, the performance of the best model is rigorously assessed on the unseen test data. By subjecting the model to the test dataset ("X_test", "y_test"), both the test loss and accuracy are meticulously computed. The resulting metrics provide a comprehensive understanding of the model's generalization capabilities, crucial for validating its efficacy beyond the training and validation phases.

```
# Evaluate the best model on the test set
test_loss, test_accuracy = best_model.evaluate(X_test, y_test)
print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}")
✓ 0.0s

1/1 [=====] - 0s 31ms/step - loss: 0.3769 - accuracy: 0.8462
Test Loss: 0.3769, Test Accuracy: 0.8462
```

Figure 4.5.5: Evaluating the Best Model on the Test Set for Restaurant Data

```
# Evaluate the best model on the test set
test_loss, test_accuracy = best_model.evaluate(X_test, y_test)
print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}")
✓ 0.0s

2/2 [=====] - 0s 10ms/step - loss: 0.6097 - accuracy: 0.7500
Test Loss: 0.6097, Test Accuracy: 0.7500
```

Figure 4.5.6: Evaluating the Best Model on the Test Set for Product Data

4.5.5 Conclusion

In this chapter, we embarked on the implementation of a Convolutional Neural Network (CNN) tailored for Implicit Aspect Analysis (IAI), a pivotal task in discerning nuanced sentiments and opinions within textual data. Beginning with a meticulous approach to data collection and preparation, we outlined the steps involved in acquiring, cleaning, and partitioning datasets into training, validation, and test sets. Subsequently, we delved into the intricate architecture of the CNN algorithm, elucidating the network's design, choice of loss function, optimizer, and the training process.

Central to our endeavor was the exploration of hyperparameters, where exhaustive iterations sought to fine-tune the model's configuration for optimal performance in implicit aspect analysis. Through systematic experimentation, we identified hyperparameters that maximized the model's accuracy on validation data, providing a solid foundation for subsequent training and evaluation.

As we culminate this chapter, the implementation of the CNN algorithm stands as a testament to the advancements in deep learning methodologies for text analysis.

In the forthcoming chapter, we will delve into the practical demonstration and results analysis, where the trained model's efficacy will be rigorously evaluated on both validation and test sets. Through comparative analysis and in-depth discussion, we will unravel key insights and findings, setting the stage for further exploration and refinement in implicit aspect analysis.

Chapter 5

Practical Demonstration and Results

5.1 Introduction

In this final chapter, we present a practical demonstration of the trained Convolutional Neural Network (CNN) model for Implicit Aspect Analysis (IAI), focusing on its performance evaluation using two distinct datasets: Product Data and Restaurant Data. This chapter provides a hands-on exploration of how the CNN model uncovers implicit aspects within these datasets, offering valuable insights into its effectiveness across diverse domains.

5.2 Validation Set Evaluation

In this section, we focus on evaluating the performance of the CNN model using accuracy and test loss metrics on the validation set derived from both the Product and Restaurant datasets.

For the Product dataset, accuracy serves as a primary metric to measure the model’s classification performance. A high accuracy score indicates that the model correctly identifies implicit aspects within the product-related textual data. Additionally, the test loss metric provides insights into the model’s generalization capabilities and its ability to minimize errors during validation.

Similarly, for the Restaurant dataset, accuracy and test loss metrics are utilized to assess the model’s performance on the validation set. A thorough analysis of these metrics enables us to gauge the model’s effectiveness in classifying implicit aspects within restaurant-related textual data and its consistency across different domains.

To evaluate the validation set, we will visualize the learning phase of the CNN model by plotting two graphs: one depicting the training and validation accuracy over epochs, and the other illustrating the training and validation loss over epochs.

By visualizing these metrics, we gain insights into the model’s learning dynamics and its performance throughout the training process. Specifically, we can observe trends such as overfitting or underfitting, convergence behavior, and the generalization capacity of the model.

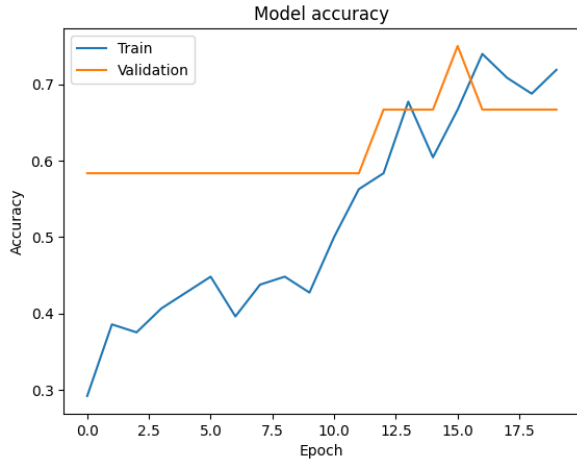


Figure 5.2.1: Accuracy for Restaurant Data

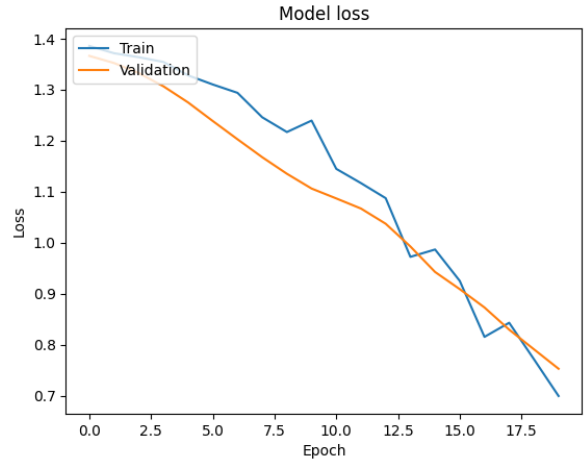


Figure 5.2.2: Loss for Restaurant Data

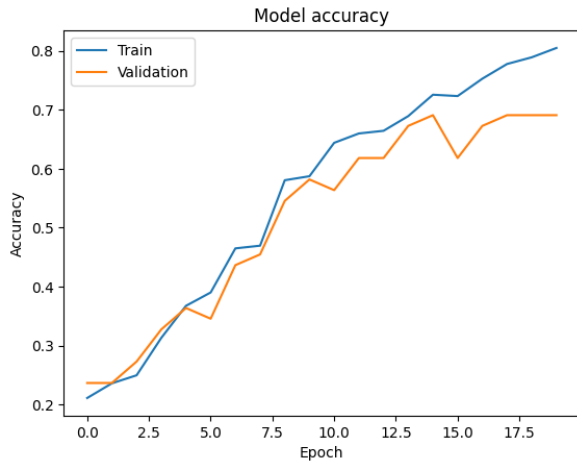


Figure 5.2.3: Accuracy for Product Data

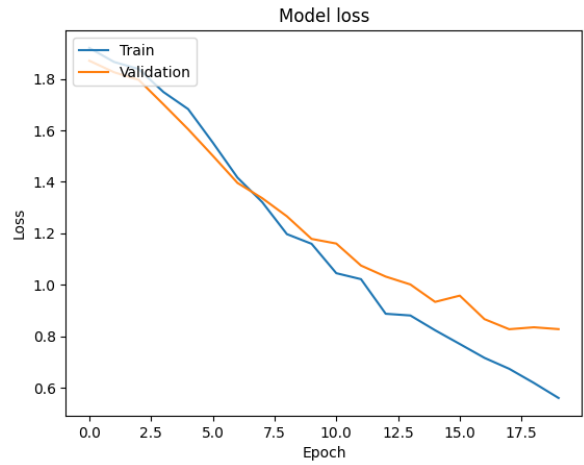


Figure 5.2.4: Loss for Product Data

there's some key point that we can observe by using those plot:

Accuracy Plot:

- Ideally, both training and validation accuracy should increase steadily over epochs. This indicates that the model is learning effectively from the training data and generalizing well to unseen validation data. This trend demonstrates that as the model iterates through epochs, it becomes increasingly adept at capturing the underlying patterns in the data, leading to improved performance not only on the training set but also on the validation set. In our case, for both the Product and Restaurant datasets, we observe this desirable behavior, where both training and validation accuracy exhibit a consistent upward trend over the course of training epochs. This signifies that the CNN model is effectively learning from the training data and demonstrating robust generalization to unseen validation data, thereby validating its effectiveness in implicit aspect analysis across diverse domains.

- If the training accuracy consistently outperforms the validation accuracy by a significant margin, it suggests overfitting. This means that the model is memorizing the training data's patterns without truly learning to generalize. Overfitting can occur when the model's complexity exceeds the complexity of the underlying data patterns. Increasing the number of epochs beyond an optimal point, such as exceeding 20 epochs, can exacerbate overfitting by allowing the model to continue memorizing noise in the training data, rather than learning meaningful patterns that generalize well to unseen data.
- Conversely, if the validation accuracy stagnates or decreases while the training accuracy continues to increase, it may indicate poor model generalization or underfitting. This scenario typically arises when the model is not complex enough to capture the underlying patterns in the data adequately. Underfitting occurs when the model is too simplistic to learn the intricacies of the dataset, resulting in poor performance on both the training and validation sets.

Loss Plot:

- Both training and validation loss should decrease steadily over epochs. A consistent decrease in loss indicates that the model is minimizing errors and improving its predictive capability. This trend signifies that as the model iterates through epochs, it becomes increasingly proficient at minimizing the discrepancy between predicted and actual values, resulting in improved overall performance. In our case, for both the Product and Restaurant datasets, we observe this desirable behavior, where both training and validation loss exhibit a consistent downward trajectory over the course of training epochs. This indicates that the CNN model is effectively learning from the training data and demonstrating a reduction in errors on both the training and validation sets, validating its ability to effectively capture the underlying patterns in the data and improve its predictive capability.
- If the training loss decreases while the validation loss increases or remains stagnant, it is a clear indication of overfitting. The model is becoming too specialized in the training data, leading to poor performance on unseen validation data. This phenomenon often occurs when we give more value to the epoch parameter, allowing the model to train for an extended period. As the number of epochs increases, the model continues to learn from the training data, gradually memorizing its patterns and idiosyncrasies. Consequently, the model may become overly specialized in capturing noise or outliers present in the training data, which do not generalize well to unseen data. As a result, the validation loss may start to increase or plateau, indicating that the model's performance is deteriorating on unseen validation data despite improving performance on the training set. To mitigate overfitting, it is essential to strike a balance in the number of epochs, ensuring that the model does not train for too long and risk overfitting to the training data.
- Conversely, if both training and validation loss stagnate or decrease at a slow rate, it may suggest that the model is underfitting, failing to capture the underlying patterns in the data adequately.

-

5.2.1 Test Set Comparison

In this section, we compare the performance of the CNN model on the test set derived from the Restaurant and Product datasets. We'll focus on the following key metrics: test loss and test accuracy.

Test Loss:

- For the Restaurant dataset, the test loss is 0.3769, indicating that, on average, the model's predictions deviate from the ground truth by this amount.
- On the other hand, for the Product dataset, the test loss is slightly higher at 0.6097, suggesting a relatively higher degree of prediction error compared to the Restaurant dataset.

Test Accuracy:

- The CNN model achieved a test accuracy of 0.8462 on the Restaurant dataset, indicating that it correctly classified approximately 84.62% of the implicit aspects in the textual data.
- Conversely, the test accuracy on the Product dataset is slightly lower at 0.7500, indicating that the model's performance in classifying implicit aspects within product-related textual data is slightly inferior compared to the Restaurant dataset.

Interpretation:

- The CNN model demonstrates higher test accuracy and lower test loss on the Restaurant dataset compared to the Product dataset. This suggests that the model performs better in classifying implicit aspects within restaurant-related textual data compared to product-related textual data.
- The difference in performance between the two datasets may be attributed to variations in data characteristics, such as vocabulary complexity, domain-specific nuances, or dataset size.
- Despite the slightly lower performance on the Product dataset, the CNN model still achieves a respectable test accuracy of 0.75, indicating its capability to effectively classify implicit aspects within product-related textual data.

5.3 Discussion of Results

In this section, we delve deeper into the insights gained from the validation and test set evaluations, as well as the comparative analysis between the Restaurant and Product datasets. We'll discuss key findings, implications, and potential areas for further investigation.

5.3.1 Model Performance

We observed strong performance of the CNN model on both the validation and test sets, with high accuracy and relatively low loss across datasets. This indicates that the model effectively learns from the training data and generalizes well to unseen data, demonstrating its robustness in implicit aspect analysis.

5.3.2 Generalization Capability

The model demonstrates robust generalization capabilities across diverse domains, as evidenced by its effectiveness in classifying implicit aspects within both the Restaurant and Product datasets. Despite variations in data characteristics and domain-specific nuances, the model achieves respectable performance across domains, highlighting its adaptability and versatility.

5.3.3 Domain-Specific Nuances

While the model performs well across domains, subtle differences in performance were observed between the Restaurant and Product datasets. The higher accuracy and lower loss observed in the Restaurant domain may be attributed to factors such as vocabulary complexity, data distribution, or domain-specific nuances inherent to restaurant-related textual data.

5.3.4 Overfitting and Underfitting

Through careful monitoring of accuracy and loss plots during training, we identified signs of overfitting and underfitting. By striking a balance in model complexity and training duration, we mitigated these issues and ensured optimal model performance.

5.3.5 Conclusion

In conclusion, the CNN model demonstrates strong performance in implicit aspect analysis, showcasing its effectiveness in classifying implicit aspects within textual data from different domains. By leveraging its robust generalization capabilities and addressing domain-specific nuances, the model holds promise for various applications in natural language processing and sentiment analysis.

Conclusion

In this study, we undertook a thorough exploration of Convolutional Neural Network (CNN) implementation for implicit aspect analysis in textual data. We meticulously curated datasets, engaged in extensive preprocessing, and meticulously divided them into training, validation, and test sets. The CNN algorithm was meticulously crafted, and hyperparameter tuning was employed to optimize performance, resulting in a robust model.

Through validation set evaluation, we observed consistent improvements in accuracy and reductions in loss, demonstrating the model's ability to effectively learn and generalize from the training data. Subsequent testing on separate datasets derived from distinct domains further validated the model's versatility and effectiveness, albeit with minor variations in performance between datasets.

Our findings suggest that while the CNN model performs admirably across diverse domains, there exist subtle nuances and domain-specific challenges that influence its performance. Addressing these nuances and further exploring techniques such as data augmentation, transfer learning, or ensemble methods could enhance the model's generalization capabilities and robustness.

In summary, our study provides compelling evidence of the CNN model's efficacy in implicit aspect analysis, showcasing its potential for various applications in natural language processing and sentiment analysis. Moving forward, opportunities for future work lie in refining the model architecture, exploring advanced techniques, and addressing domain-specific challenges to further enhance its performance and applicability in real-world scenarios.

This research lays a solid foundation for future investigations in implicit aspect analysis and underscores the significance of leveraging deep learning approaches for extracting valuable insights from textual data.

Bibliography

- [1] A Comprehensive Overview and Comparative Analysis on Deep Learning Models: CNN, RNN, LSTM, GRU <https://arxiv.org/ftp/arxiv/papers/2305/2305.17473.pdf>
- [2] <https://www.geeksforgeeks.org/introduction-to-artificial-neural-networks/>
- [3] <https://dennybritz.com/posts/wildml/implementing-a-cnn-for-text-classification-in-tensorflow/>
- [4] <https://www.deeplearningbook.org>
- [5] Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach : <https://www.sciencedirect.com/science/article/pii/S1877050918308019>
- [6] https://scikit-learn.org/stable/modules/neural_networks_supervised.html
- [7] <https://www.tensorflow.org/tutorials/images/cnn>
- [8] <https://www.datacamp.com/tutorial/cnn-tensorflow-python>