Chapter 3

Background

The background needed to understand the contribution of the thesis is described in this chapter. Section 3.1 briefly discusses information retrieval and social media since they are central to this thesis. The following two sections give an overview of machine learning and deep learning models employed in this thesis, such as Support vector machine, Multinomial Naive-Bayes (MNB), Stochastic Gradient Descent (SGD), Linear Regression (LR), Random Forest, Decision Tree, XG Boost, k-Nearest Neighbour, and Bidirectional LSTM, and Recurrent neural network in section 3.2 and section 3.3, respectively. The possible way of multi-task learning used to detect hate speech has been discussed in Section 3.4. After that, the input representation methods for deep learning models to a text have been explored in section 3.5.

3.1 Information Retrieval

The practice of archiving written information was traced around 3000 BC, When the Sumerians designated particular areas to keep clay tablets with cuneiform inscriptions. The Sumerians learned that proper organization and access to archives were vital to the efficient use of information. They developed unique classifications to identify each tablet

and its contents. Over the centuries the need to store and retrieve written information became increasingly important, especially with inventions like paper and the printing press. Soon after the invention of computers, people realized that they could be used to store and obtain large amounts of information mechanically [141].

In 1945, Vannevar Bush published an important article titled "As We May Think" which gave rise to the idea of automatic access to vast amounts of stored knowledge [142]. In the 1950s, the idea was embodied in more concrete details of how text collections could be searched automatically. Several works appeared in the mid-1950s that elaborate on the basic idea of searching text with a computer. One of the most effective methods was described by H.P. Luhn in 1957, in which he proposed using words as indexing units for documents and measuring word overlap as criteria for retrieval [143]. Most notable was the development of smart systems in the 1960s by Gerard Salton and his students, first at Harvard University and later at Cornell University [144]; and the Cranfield evaluation by Cyril Claverdon and his group at Cranfield's College of Aeronautics. The Cranfield tests developed an evaluation method for recovery systems that is still in use by IR systems today [145].

Various models for document retrieval were developed and progressed along all dimensions of the retrieval process. These new models/techniques proved to be effective experimentally on the small text collections (several thousand articles) available to researchers at that time. However, due to the lack of availability of large text collections, the question remained unanswered whether these models and techniques would be large enough to scale. This changed in 1992 with the founding of the Text Retrieval Conference, or TREC [146]. TREC has divided IR into related but important areas:

- retrieval of spoken information,
- non-English language retrieval,
- information filtering,

- user interaction with the retrieval system

In information retrieval scenarios where people care more about textual characteristics that facilitate text classification, methods of selection of different attributes like document frequency (DF), information gain (IG), mutual information (MI), 2-test (CHI-test), and term strength (TS) have different effects on text classification [147]. The generated embedding vector can be context-independent or context-dependent. We focus on English and non-English language retrieval on social media text. The evaluation indicators in this thesis are precision(P), recall(R), and F_1-score.

1. Precision: It is the ratio of true-positives (TP) to the sum of true-positives and false-positives (FP).

$$Precision(P) = \frac{TP}{TP+FP} \tag{3.1}$$

2. Recall: It is the ratio of true-positives (TP) to the sum of true-positives and false-negatives (FN).

$$Recall(R) = \frac{TP}{TP+FN} \tag{3.2}$$

3. F_1-score: It is the balanced harmonic mean of precision and recall and used to have a composite idea of precision and recall.

$$F_1 = \frac{2*R*P}{R+P} \tag{3.3}$$

4. $Macro\_F_1$: It is the average of per-class precision and recall scores over all classes. For each pair of classes, F_1 scores are computed and then arithmetic mean of these per-class F1-scores represent Macro-F_1.

5. $Weighted\_F_1$: It is the weighted version of the average F_1-scores where each class is weighted by the number of samples from that class.

6. Accuracy: It is the ratio of no. of correct predictions to the total number of original entities i.e.

$$Accuracy = \frac{\text{\# correct predictions}}{\text{Total \# test-instances}} \tag{3.4}$$

3.1.1 Hyperlink-Induced Topic Search (HITS) Algorithm

HITS; also known as hubs and authorities. Jon Kleinberg created the link analysis algorithm known as HITS, which rates Web pages. The idea behind Hubs and Authorities stemmed from a particular insight into the creation of web pages when the Internet was initially forming; that is, specific web pages, known as hubs, served as large directories that were not actually authoritative in the information that they held but were used as compilations of a broad catalog of information that led users directly to other authoritative pages. In other words, a good authority is a page that is linked by many different hubs, whereas a good hub is a page that points to many other pages [148].

1. Let number of iterations be k
2. Each node is assigned a Hub score = 1 and an Authority score = 1
3. Repeat k times:
4. Hub update : Each node's Hub score = Σ (Authority score of each node it points to)
5. Authority update : Each node's Authority score = Σ (Hub score of each node pointing to it)

Normalize the scores by dividing each Hub score by square root of the sum of the squares of all Hub scores, and dividing each Authority score by square root of the sum of the squares of all Authority scores.

3.1.2 Latent Dirichlet Allocation (LDA)

LDA is a generative probabilistic model of a corpus. The fundamental concept is that the documents are depicted as random mixtures of latent topics, where a distribution of words characterizes a topic. One of the most widely used topic modeling techniques is LDA, which was first presented by Blei, Ng, and Jordan in 2003 [149]. Topics are represented in LDA by word probabilities. According to word probabilities from LDA, the words in each topic with the highest probabilities typically provide a good indication of the topic. A text corpus is denoted as given below (as per [150])

$$D = \{d_1, d_2, \ldots, d_m\} \tag{3.5}$$

where d_i is a document and m is the number of documents in the corpus. Each document consists of a sequence of words $\{w_{i1}, w_{i2}, \ldots, w_{in}\}$ where n is the number of words in document d_i. A dictionary $V = \{v_1, v_2, \ldots, v_N\}$. where N is the size of the dictionary. Moreover, z is a latent variable representing the latent topic associated with each observed word. $Z_m = \{z_{m,1}, z_{m,2}, \ldots, z_m, N_m\}$ The topic sequence associated with the word sequence W_m. The generative procedure of LDA can be formally defined as:

1. For all the topics

$$k \in 1, \ldots, K : \tag{3.6}$$

Choose a word distribution

$$\varphi_k \sim Dir(\varphi|\beta) \tag{3.7}$$

2. For all the documents dm where

$$m \in 1, ..., M : \tag{3.8}$$

Choose

$$N_m \sim Poisson(\xi) \tag{3.9}$$

Choose a topic distribution

$$\theta_m \sim Dir(\theta|\alpha) \tag{3.10}$$

For all the words w_m,n where

$$n \in 1,...,N_m : \tag{3.11}$$

Choose a topic index

$$z_{m,n} \sim Mult(z|\theta m) \tag{3.12}$$

Choose a word

$$w_{m,n} \sim Mult(w|\varphi_{z_{m,n}}) \tag{3.13}$$

$$\varphi_k = (\varphi_{k,1}, \varphi_{k2}, ..., \varphi_{k,V})^T \in R^V \tag{3.14}$$

where,

$$\varphi_{k,i} = p(w = v_i|z = k) \tag{3.15}$$

Thus, the parameters for the topic-word distribution can be represented as

$$\Phi = (\varphi_1, \varphi_2, ..., \varphi_K)^T \in R^{K*V} \tag{3.16}$$

where, K is the topic number. Moreover,

$$\theta_m = (\theta_{m,1}, \theta_{m2}, ..., \theta_{m,K})^T \in R^K \tag{3.17}$$

where,

$$\theta_{m,k} = p(z = k|d_m) \tag{3.18}$$

Then the parameters for document-topic distribution is

$$\Theta_m = (\theta_1, \theta_2, ..., \theta_m)^T \in R^{M*K} \tag{3.19}$$

3.2 Traditional Machine Learning

This section briefly describes the machine learning techniques which have become popular over time, such as Support vector machine, Multinomial Naive-Bayes (MNB), Stochastic Gradient Descent (SGD), Linear Regression (LR), Random Forest, Decision Tree, XG Boost, k-Nearest Neighbour.

3.2.1 Support vector machine

Support vector machines have a strong theoretical foundation and outstanding empirical breakthroughs first used by Vapnik (1982). Support vector machine is one of the most essential machine learning algorithms that has been implemented primarily on classification problems, for e.g. classifying the text and also in image processing for recognition. We consider SVMs in a binary classification setting. For a set of training vectors belonging to two separate classes [151],

$$\{(x_1, y_1), \ldots (x_m, y_m)\}, \{x \in R^n, y \in \{1, -1\}\} \tag{3.20}$$

Hyperplane can be found to separate these two classes:

$$\langle x + y \rangle + b = 0 \tag{3.21}$$

The above collection of vectors is said to be perfectly separated by the hyperplane if the closest vector to the hyperplane is maximally distant from it and the separation is error-free. A canonical hyperplane, where the parameters w, b are constrained by this equation,

$$min_i \, |\langle x + y \rangle + b| = 1 \tag{3.22}$$

The ideal separating hyperplane is deduced from the set of equations above to be:

$$w^* = \sum_{i=1}^{l} \alpha_i x_i y_i \tag{3.23}$$

$$b^* = -0.5 \langle w^*, x_r + x_s \rangle \tag{3.24}$$

where α_i is the Lagrange multiplier. x_r and x_s are support vectors from each class satisfying $\alpha_r > 0$, $y_r = -1$; $\alpha_s > 0$, $y_s = 1$.

3.2.2 Multinomial Naive-Bayes (MNB)

Text classification can be viewed from Bayesian learning, which assumes that a specific parametric model generates the word distribution in documents, and the parameters can be estimated from the training data [152].

$$P(c \mid d) = \frac{P(c) \Pi_{i=1}^{n} P(w_i \mid c)^{f_i}}{P(d)} \tag{3.25}$$

Where fi is the number of occurrences of a word wi in document d, $P(wi|c)$ is the conditional probability that a word wi may happen in a document d given the class value c, and n is the number of unique words in document d. $P(c)$ is the prior probability that a document with class label c may happen in the document collections.

The parameters in Equation 3.4 can be estimated by a generative parameter learning approach, called frequency estimate (FE), which is simply the relative frequency in data. FE estimates the conditional probability $P(wi|c)$ using the relative frequency of the word wi in documents belonging to class c.

$$\hat{P}(w_i|c) = \frac{N_{ic}}{N_c} = \frac{N_{ic}}{\sum_{j=1}^{|V|} N_{jc}} \tag{3.26}$$

Where N_{ic} is the number of occurrences of the word w_i in training documents T with the class label c, N_c is the total number of word frequencies in documents with class label c in T and can be estimated through N_{ic}.

3.2.3 Stochastic Gradient Descent (SGD)

It has often been proposed to reduce the empirical risk $E_n(f_w)$ by using gradient descent (GD). Each iteration updates the weight w based on the gradient of $E_n(f_w)$ [153].

$$w_{t+1} = w_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \bigtriangledown_w Q(z_t, w_t) \tag{3.27}$$

Where γ is an adequately chosen gain, under adequate regularity hypotheses, when the initial estimate w_0 is close sufficient to the optimum, and when the gain γ is sufficiently small, the algorithm achieves linear convergence [154].

The Stochastic gradient descent (SGD) algorithm is a rigorous simplification. Instead of computing the gradient of $E_n(f_w)$ precisely, each iteration estimates this gradient based on a randomly chosen example z_t:

$$w_{t+1} = w_t - y_t \bigtriangledown_w Q(z_t, w_t) \tag{3.28}$$

Stochastic process $w_t, t = 1, \ldots$. depends on randomly chosen examples at each iteration. It is expected that (3.28) behaves like its expectation (3.27) despite the noise introduced by this simplified process.

3.2.4 Linear Regression (LR)

Linear regression is a method for modelling the relationship between a scalar response and one or more explanatory variables using a linear approach (also known as dependent or independent variables). When there is only one explanatory variable, the process is known as “simple linear regression." When there are more than one, the process is known as “multiple linear regression."

Given a data set $\{y_i, x_{i1}, \ldots, x_{ip}\}_{i=1}^{n}$ of n statistical units, a linear regression model assumes that the relationship between the dependent variable y and the p-vector of regressors x is linear. This relationship is modeled through a disturbance term or error variable ε— an unobserved random variable that adds “noise" to the linear relationship between the dependent variable and regressors. Thus the model takes the form

$$y_i = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + x_i^T \beta + \varepsilon_i, i = 1, \ldots, n \tag{3.29}$$

where, T denotes the transpose, so that $x_i^T \beta$ is the inner product between vectors x_i and β. Often, these n equations are stacked together and written in matrix notation as [155].

$$y = X\beta + \varepsilon \tag{3.30}$$

3.2.5 Random Forest

Random Forest is a classifier that consists of multiple decision trees on different subsets of a given dataset and takes the average to improve the predictive accuracy of that dataset. A

random forest is a group of decision trees whose nodes are defined at the pre-processing step [156]. The training algorithm for random forests applies the general technique of bootstrap aggregation, or bagging, to tree learners. With a training set of $X = x_1, \ldots, x_n$ and responses of $Y = y_1, \ldots, y_n$, repeatedly bagging (B times) chooses a random sample with replacement of the training set and fits trees to these samples [157]:
For $b = 1, \ldots, B$:

- Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
- Train a classification or regression tree f_b on X_b, Y_b.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$$\hat{f} = \frac{1}{B} \sum_{b=1}^{B} f_b(x') \tag{3.31}$$

3.2.6 Decision Tree

The decision tree classifier builds a classification model by building a decision tree. Each node in the tree specifies a test on an attribute; every bough descending from that node coincides with one of the possible values of that attribute. Each leaf represents the class label associated with the sample. According to the result of tests along the path, the examples in the training set are classified by navigating from the root of the tree to a leaf [158].

3.2.7 XG Boost

XGBoost was primarily designed for speed and performance by using gradient-boosted decision trees. It represents a method for machine boosting or applying boosting to machines, initially done by Tianqi Chen [159] and further adopted by many developers.

3.2.8 k-Nearest Neighbour (kNN)

K-nearest neighbours focus on keeping similar things. The model works on class labels and feature vectors in a data set. KNN stores all the cases and helps classify new cases with the help of similarity measures. K-nearest neighbours, text is represented using a spatial vector which is denoted by $S = S(t_1, w_1; t_2, w_2; \ldots t_n, w_n)$. For any text, similarity is found and calculated using the training text and the texts with highest similarity are selected [160].

$$sim(d_i, d_j) = \frac{\sum_{p=1}^{n} w_{ip} w_{jp}}{\sqrt{\sum_{p=1}^{n} w_{ip}^2} \sqrt{\sum_{p=1}^{n} w_{jp}^2}} \tag{3.32}$$

Where d_i and d_j are the feature vectors of the incoming and training text, respectively, n is the dimension of the feature vector. W_{ip} and W_{jp} are the p-th elements of the d_i and d_j vectors, respectively. The k-nearest neighbours of that incoming text are selected based on similarity or comparison of texts [161].

$$Q_{(d_i, C_m)} = \sum_{j=1}^{1} sim(d_i, d_j) \partial(d_i, C_m) \tag{3.33}$$

$$\partial(d_i, C_m) = \begin{cases} 1, & if, \quad d_i \in C_m \\ 0, & if, \quad d_i \in C_m \end{cases} \tag{3.34}$$

3.3 Deep Learning

Deep learning is a subfield of machine learning that deals with algorithms inspired by the structure and function of the brain called artificial neural networks [162]. Deep-learning architectures such as Deep Neural Networks, Recurrent Neural Networks, and Convolutional Neural Networks have been applied in areas including sentiment analysis, natural language processing, hate speech and offensive content identification, text analysis,

where they have produced comparable results in some areas that surpass human expert performance.

3.3.1 Convolutional Neural Network

Convolutional Neural Network (CNN) is a renowned deep learning architecture inspired by living creatures' natural visual perception mechanism. In 1959, Hubel and Wiesel [163] discovered that animals' visual cortex cells are responsible for detecting light in receptive fields. Inspired by this discovery, Kunihiko Fukushima proposed the Neocognitron in 1980 [164], which can be considered the predecessor of CNN. In 1990, LeCun et al. [165] published a seminal paper establishing the modern framework of CNN and later improved it [166].

Convolutional neural networks are similar to traditional ANNs in that they consist of neurons that self-adapt through learning. Each neuron will still receive input and operate (such as a scalar product followed by a non-linear function) - the basis of the numerous ANNs.

In existing studies using deep learning to classify texts, CNNs use convolutional filters that automatically learn the appropriate features for a given task. For example, using CNN for sentiment classification, convolutional filters can capture the syntactic and semantic features underlying emotional expressions, as shown in [167]. A combination of convolutional filters can achieve comparable performance even without any special hyperparameter adjustments by a single convolutional layer [168]. Furthermore, CNNs do not require expert knowledge about the linguistic structure of the target language [169]. CNNs have been successfully applied to various text analysis: semantic parsing [170], search by query [171], and sentence modeling [172].

Figure 3.1 depicts three filter region sizes: 2, 3, and 4, each with 2 filters. Filters perform convolutions on the sentence matrix and generate (variable-length) feature maps;

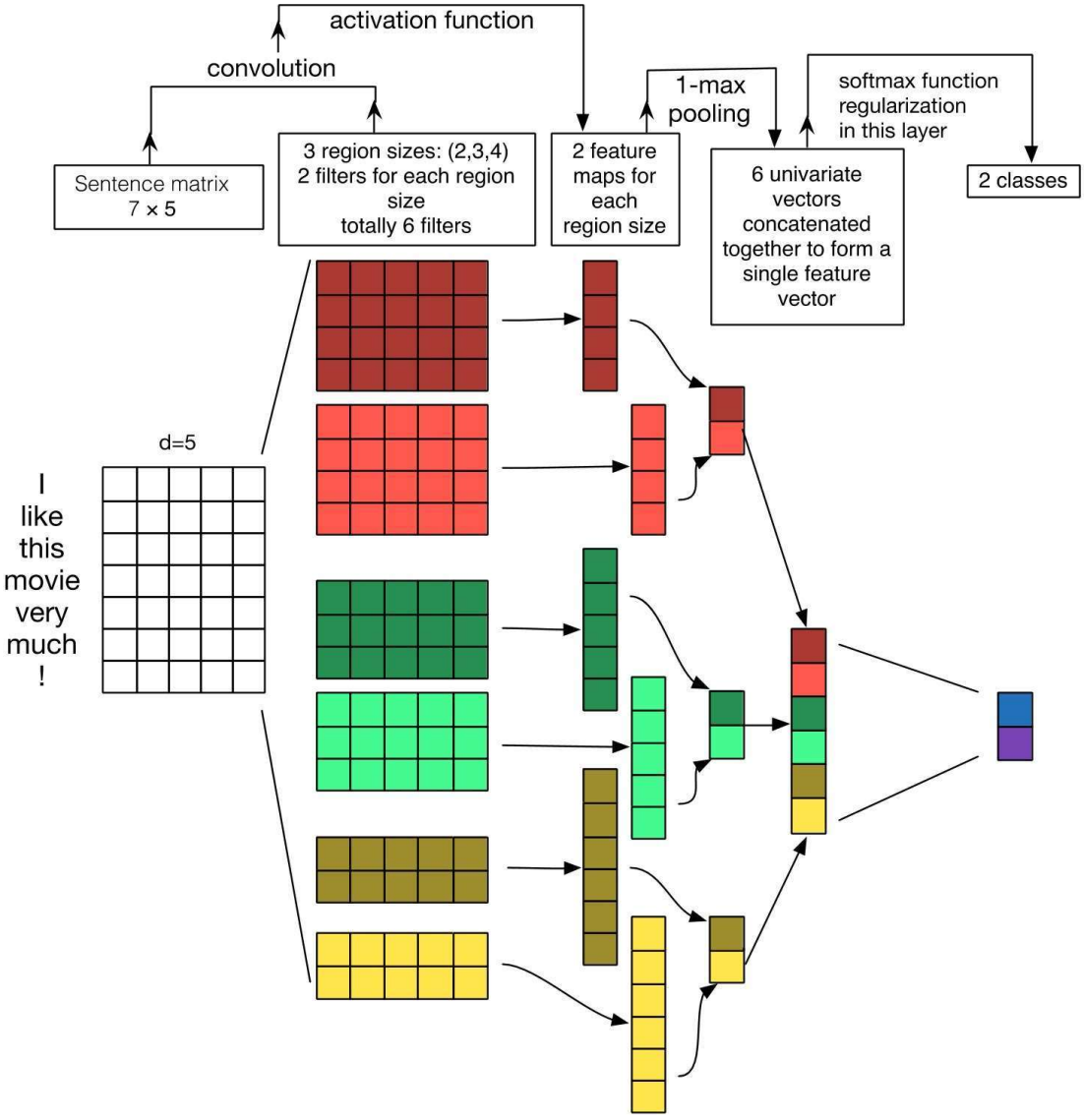

FIGURE 3.1 Illustration of a CNN architecture for sentence classification [1]

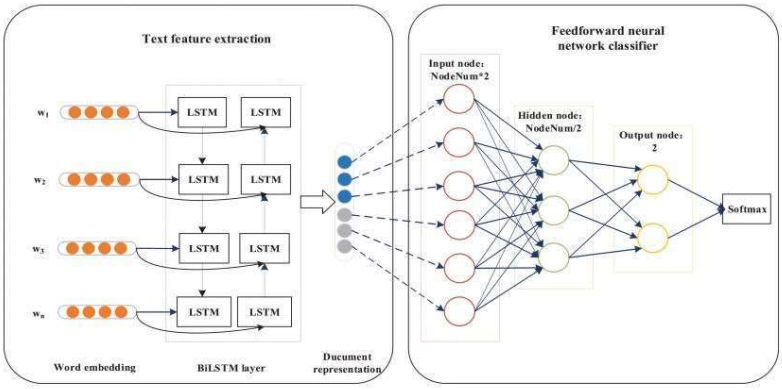

FIGURE 3.2 Sentiment Analysis based BiLSTM [2]

1-max pooling is performed over each map; the most significant number is recorded from each feature map. Thus a univariate feature vector is generated from all six maps, and these 6 features are concatenated to form a feature vector for the penultimate layer. The final softmax layer receives this feature vector as input and uses it to classify the sentence, assume binary classification, and depict two possible output states.

3.3.2 BiLSTM

In the traditional recurrent neural network model and the LSTM model, information can only be propagated forward, with the result that the time state depends only on the information before time t. In order to include context information at every moment, BiLSTM, which combines a bidirectional recurrent neural network (BiRNN) model and LSTM units, is used to capture context information. The BiLSTM model treats all inputs equally. For sentiment analysis, the sentiment polarity of the text largely depends on the words with sentiment information.

The weighted word vectors are used as the inputs of the BiLSTM model, and the outputs of the BiLSTM model are used to represent the texts. Then, the text vectors are

input into the feedforward neural network classifier. The sentiment tendency of the text is obtained. In order to prevent the over-fitting phenomenon in the training process, the dropout mechanism was introduced, and the dropout discarding rate was set to 0.5.

The schematic diagram of the sentiment classification is shown in the FIGURE. The left subgraph is the process of text feature extraction. The right subgraph is obtaining the sentiment polarity of the text. NodeNum mentions the number of nodes of the LSTM hidden layer.

3.4 Transfer Learning

The areas of machine learning and data mining have been widely and successfully used in many applications where patterns of past information can be extracted to predict future outcomes [173]. Traditional ML is characterized by training data and test data having the same input feature space and uniform data distribution. When there is a difference in the data distribution between the training and test data, the predictive learner results can be degraded [174]. In some scenarios, it may be difficult and costly to obtain training data that matches the feature space and obtain the predicted data distribution characteristics of the test data. Transfer learning improves a learner from one domain to another by transferring information from related domains.

A domain D is represented by two parts, a marginal probability distribution $P(X)$ and a feature space X, where $X = x_1, ..., x_n X$. If the learning task is text classification, and every term is accepted as a binary segment, then the space of all term vectors is X, x_i is the i-th term vector corresponding to some text, and X is a particular learning sample. If two domains are different, they may have different feature spaces or marginal probability distributions.

The different settings of transfer learning are described by Pan et al. [174]:

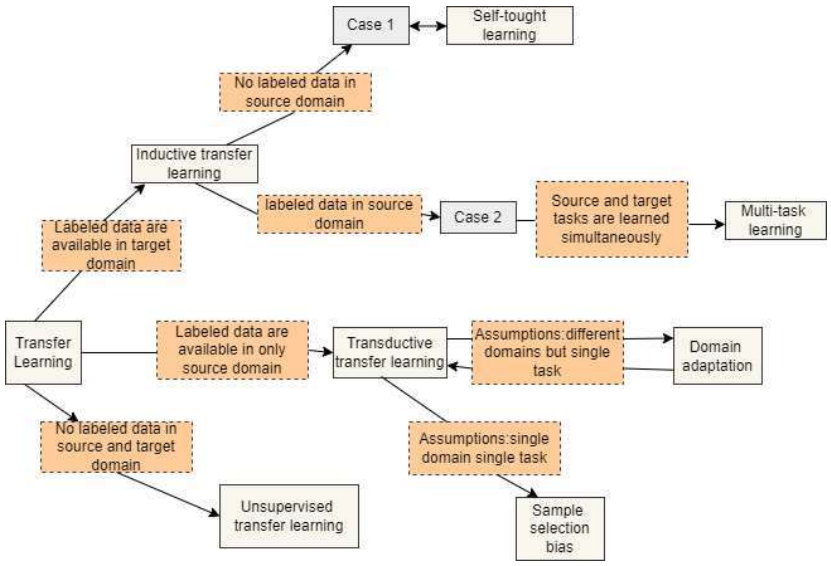

FIGURE 3.3 Overview of various transfer learning settings [3]

- **Case 1:** A source domain D_S and learning task T_S, a target domain D_T and learning task T_T , transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)$ in D_T using the knowledge in D_S and T_S, where $D_S \neq D_T$, or $T_S \neq T_T$. Case 1 matches when the two sets of documents are described in different languages. It is cross lingual transfer learning.

  For example, in the text classification, between a source text set and a target text set, either the term features are different between the two sets (e.g., they use different languages), or their marginal distributions are different.

- **Case 2:** A source domain D_S and a learning task T_S, a target domain D_T and a learning task T_T, inductive transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)$ in D_T using the knowledge in D_S and T_S, where $T_S \neq T_T$. Case 2 deals with when the source domain document and the target domain document focus on different topics. The inductive transfer learning settings could be further categorized into two cases:

There is a lot of labeled data available in the source domain. The inductive transfer learning setting is similar to the multitask learning setting. However, the inductive transfer learning setting aims to achieve high performance in the target task by simply transferring knowledge from the source task, whereas multitask learning attempts to simultaneously learn the target and the source task.

No labeled data is available in the source domain. In this case, the setting of inductive transfer learning is similar to the self-taught learning setting.

- **Case 3:**A source domain D_S and a corresponding learning task T_S, a target domain D_T and a corresponding learning task T_T, transductive transfer learning aims to improve the learning of the target predictive function $f_T(\cdot)$ in D_T using the knowledge in D_S and T_S, where $D_S \neq DT$ and $T_S \neq T_T$. Where they required that the source and target tasks be the same, although the domains may be different. It is domain adaptation.

  Feature spaces are different between the source and target domains, $X_s \neq X_t$.

  The feature spaces between domains are the same, $X_S = X_T$, but the marginal probability distributions of the input data are different, $P(X_S) \neq P(X_T)$.

- **Case 4:**A source domain D_S with a learning task T_S, a target domain DT and a corresponding learning task T_T, unsupervised transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)^7$ in D_T using the knowledge in D_S and T_S, where $T_S \neg T_T$ and Y_S and Y_T are not observable.

3.5 Distributional Vector Representation

The use of transfer learning is defined as a method of encoding input where the embedding vector is generated over a large corporation and using that generated embedding for various tasks. The goal of word embedding is to map words in unlabeled text data to a

constant-valued low-dimensional space in order to capture intrinsic semantic and syntactic information. Embedding vector generation is the first step in converting a word into a vector. The embedding vector can be real-valued such as a distribution or discrete-valued. An embedding module transforms a word into a fixed-dimensional real-valued vector. The easiest way to represent a word in a computer-readable way is a one-hot vector, which takes the dimension of the lexical size and assigns 1 to the corresponding position of the word and 0 to the others. One-hot vectors contain hardly any semantic information about the words other than merely distinguishing them from each other.

3.5.1 Context Independent Embedding

Over time, several methods have been proposed to achieve embeddings built on top of the word [175–178]. All those embeddings are calculation-based and prediction-based. Some of them are prevalent in NLP because of achieving remarkable results on various NLP tasks, including sequence labeling.

One of the most popular is Google's Word2Vec [179] which is a statistical method with two different learning models which are the Continuous Skipgram Model and the Continuous Bag-of-Words Model, which can be used to generate real-valued dense vectors.

3.5.2 Context Dependent Embedding

Recent developments in context-dependent embeddings (Peters et al., 2018; Devlin et al., 2019) have shown that such representations are capable of achieving state-of-the-art performance in many complex NLP tasks. There are many applications for word embeddings, particularly in NLP tasks where word embeddings are fed directly as input data or as text data features. Surrounding words are also called context words because they appear in the context of the central word. Embeddings are vector space representations of each word, sentence, paragraph, image, etc. Word embeddings represent each word

of a vocabulary in vector space. They are contextual if that representation also captures the context in which used, and hence it can offer more semantic meaning. We can train a word vector to get a representation of words, such as word2vec, GloVe, and FastText embeddings. There are two problems for word representations, for word2vec, GloVe, and FastText embeddings. First, there is always the exact representation for a word type regardless of the context in which a word taken occurs. Secondly, we have a representation for a word, but words have different aspects, including semantics, syntactic behavior, and register/meaning. For example, in some cases, 'arrive,' and 'arrival' have almost the same semantics, but they are different parts of speech: 'arrive' is a verb and 'arrival' is a noun, so they can appear entirely in different places.

ELMo: Embeddings from Language Models

Embeddings from Language Models is a new type of deep contextual word representation that models both the complex features of word usage (e.g., syntax and semantics) and how these differ across linguistic contexts (i.e., to model polysemy) [180]. The traditional word type embeddings representation differs from ELMo. In ELMo, each token is assigned a representation that is a function of the entire input sentence. The ELMo representations are deep because they are a function of all the inner layers of the biLM. More specifically, a linear combination of vectors stacked over each input word for each final task improves performance when only the top LSTM layer is used.

A forward language model calculates the probability to a sequence of N tokens, $(t_1, t_2, ..., t_N)$ by modeling the probability of a token t_k given the history $(t_1, ..., t_{k-1})$:

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_1, t_2, ..., t_{k-1}) \tag{3.35}$$

A backward LM is similar to a forward LM, except that it runs in reverse, in sequence, predicting the previous token given the future context:

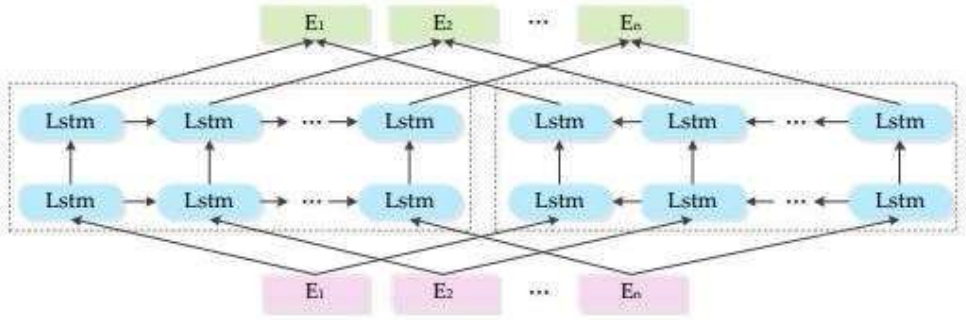

FIGURE 3.4 The architecture of embeddings from language models (ELMo)

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_{k+1}, t_{k+2}, ..., t_N) \tag{3.36}$$

A biLM combines both a forward and backward LM, and jointly maximizes the log-likelihood of the forward and backward directions:

$$\sum_{K=1}^{N} (\log p(t_k | t_1, ...t_{k-1}; \theta x, \vec{\theta} LSTM, \theta s) + \log p(t_k | t_{k+1}, ...t_N; \theta x, \theta LSTM, \theta s)) \tag{3.37}$$

Parameters for both the token representation (θx) and the softmax layer (θs) in the forward and backward direction, while maintaining separate parameters for the LSTM in each direction. The structure of the language model is shown in Figure 3.4

FastText

A word embedding tool called fastText was developed by Facebook [181]. Words are treated as the smallest unit whose vector representation is determined by *n*-grams of characters. It is easy to find the vector representation of rare words. This concept can convert non-dictionary words into vectors [182]. A 300-pixel dimension is used for each word representation to represent the vocabulary. We used padding to make the tweets equal since they are not all the same length. To make each tweet as long as possible, we have padded it. For every tweet, we used post padding to make it 30 characters long. To make

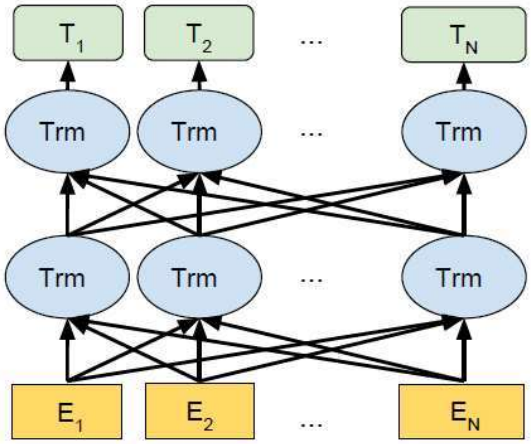

FIGURE 3.5 BERT uses many layers of bidirectional transformers0

the tweets equal to the average length, tweets with a length greater than the average have been truncated and those with smaller lengths have been appended with zeros.

BERT:

BERT, which stands for Bidirectional Encoder Representations from Transformers, is based on Transformers, a deep learning model in which every output element is connected to every input element, and the weightings between them are dynamically calculated based upon their connection. Language models can only read text input sequentially – either left-to-right or right-to-left – but cannot do both simultaneously. BERT is programmed to detect in both directions simultaneously because the transformer is known as bi-directionality. Masked Language Modeling and Next Sentence Prediction are two NLP tasks that BERT has been pre-trained on it. Masked Language Model (MLM) training is to hide a word in a sentence, and then the program has to guess which word is hidden (masked) based on the context of the hidden word. The next sentence prediction training program predicts whether two given sentences have a logical, sequential relationship or whether their relationship is just random.

A visual representation of BERT is given in Figure 3.5.