VNU Journal of Science: Comp. Science & Com. Eng., Vol. 37, No. 1 (2021) 1–24

VNU Journal of Science:
Computer Science and Communication Engineering

Journal homepage: http://www.jcsce.vnu.edu.vn/index.php/jcsce

Original Article

Modern Approaches in Natural Language Processing

Quan Thanh Tho[1,2,*]

[1] *URA Research Group, Ho Chi Minh City City University of Technology, Ly Thuong Kiet Street, District 10, Ho Chi Minh City, Vietnam*

[2] *Vietnam National University Ho Chi Minh City, Ho Chi Minh City, Vietnam*

Received 30 May 2020
Revised 15 November 2020; Accepted 15 January 2021

Abstract:

Natural Language Processing (NLP) is one of the major branches in the emerging field of *Artificial Intelligence* (AI). Classical approaches in this area were mostly based on parsing and information extraction techniques, which suffered from great difficulty when dealing with very large textual datasets available in practical applications. This issue can potentially be addressed with the recent advancement of the *Deep Learning* (DL) techniques, which are naturally assuming very large datasets for training. In fact, NLP research has witnessed a remarkable achievement with the introduction of *Word Embedding* techniques, which allows a document to be represented meaningfully as a matrix, on which major DL models like CNN or RNN can be deployed effectively to accomplish common NLP tasks. Gradually, NLP scholars keep developing specific models for their areas, notably attention-enhanced BiLSTM, Transformer and BERT; which introduce a new wave of modern approaches which frequently report new breaking results and open much novel research directions. The aim of this paper is to give readers a roadmap of those modern approaches in NLP, including their ideas, theories and applications. This would hopefully offer a solid background for further research in this area.

Keywords: Natural Language Processing, Artificial Intelligence, Deep Learning, Word Embedding, CNN, RNN, LSTM, Sequence-to-sequence, Language Model, BERT

1. Introduction

Natural Language Processing (NLP) [1] is one of major branches of *Artificial Intelligence* (AI) [2], which focuses on applications based on human natural languages. The crucial challenge of this area lies on the ”process” to understand the meaning of human language. In this context, the term ”process” can be understood as a way to transform from a presentation into another one. In that case, written or spoken language forms will be processed to be transformed into a for-

* Corresponding author.
E-mail address: qttho@hcmut.edu.vn

https://doi.org/10.25073/2588-1086/vnucsce.253

malism understandable by computers. Typically, major tasks of NLP include the following.

- *Text Classification* [3]: It can be considered as one of the most popular tasks in this area. In this task, a document will be classified into a pre-defined category. For instance, Google's e-mail system can classify if an incoming e-mail is a spam or not.
- *Sentiment Analysis* [4]: It aims to analyze the sentiment of the author/writer of a textual document (normally *positive*, *negative* or *neutral*). This task can be considered as a specific kind of text classification applications where each sentiment value is a category. This task is very useful for e-commerce systems nowadays, where reviews of customers can be analyzed to help a brand to obtain the opinions/feedback of users over their products.
- *Entity Recognition* [5]: This aims to recognize entities mentioned in a document. In practical applications, those entities are *named* and usually belong into a certain class.
- *Topic Modeling* [6]: In this context, a topic is considered as a set of relevant keywords, which can provide hint to a certain concept. For example, the set of words *football*, *stadium*, *trekking*, *swimming* can imply the concept, or topic, of *sport*. The aim of this task is to detect possible topics mentioned in a document by means of finding sets of relevant keywords. In order to fulfill this, the Topic Modeling technique is needed to be trained with a very large corpus in order to evaluate the co-occurrence probability of words potentially belonging to a same topic.
- *Machine Translation* [7]: Sometimes referred to by the abbreviation *MT*, this task aims to automatically translate a document from one language into another.
- *Chatbot* [8]: Ideally, this is an interactive system which can communicate with users using natural languages without human interference. Nowadays, this kind of application has been increasingly attracted much attention from enterprises in various domains.
- *Automatic Summarization* [9]: This task aims to produce a shortened version of a document, which captures major important points of the original text using a coherent natural language. This task is highly practical today to help human to extract useful information from large textual datasets from various domains.

Typically, the classical approaches for NLP normally combine syntax analysis and information extraction where documents are represented as vectors, e.g TF-IDF of *bag-of-words* ones. With the rapid development of computational resources since the end the 20th century, *Deep Learning* (DL) technique has been deployed effectively to handle various problems of computer science, including NLP. Once applied to solve NLP tasks, the applications of Deep Learning have achieved some remarkable *milestones* as illustrated in Figure 1. Since the application of neural language model was reported in 2001, perhaps the first notable milestone of NLP in this direction is the introduction of *Word Embedding* in 2013. This technique allows encoding each word in a document into a numerical vector, which is interpreted from the contexts that this word is likely to appear in a training corpus. Thus, a document can be represented as a matrix or an ordered set of vectors, on which typical deep learning architectures such as *Convolutional Neural Network* (CNN) or *Long Short-Term Memory* (LSTM) can be effectively deployed. Especially, an architecture combined from two opposite (forward and backward directions) LSTM models, known as *Bidirectional LSTM*, or BiLSTM, had also become popular for NLP tasks due to its ca-

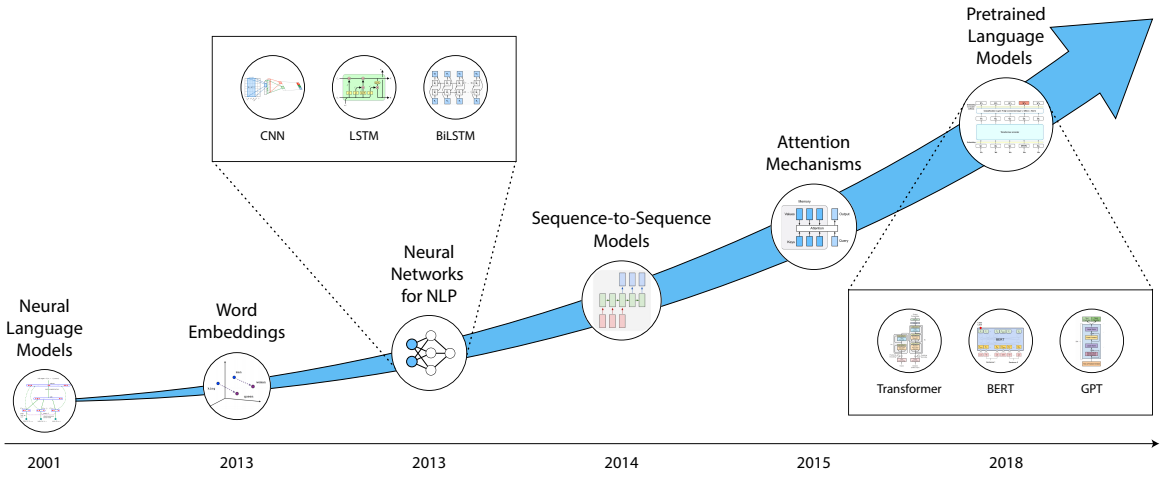

Figure 1. Milestones of DL applications in NLP.

pability of capturing information a documents in both opposite directions in a document.

After some initial success gained by those models on NLP tasks, the research community continued witnessing the births of some text-oriented advanced techniques including *sequence-to-sequence* (or *seq2seq*) architecture in 2014 and *attention mechanism* in 2015. In 2018, the use of pre-trained language models such as *Transformer*, *BERT* or *GPT* marked some significant improvements on performance of NLP tasks, which has still been adopted until today.

In this paper, we review the modern approached in NLP related to the above discussed research. Firstly, we present popular pipelines for processing data in NLP tasks, including both classical and DL-based ones. Next, we present the Word Embedding techniques, followed by remarkable achievements of DL-based ressearch for NLP. They include using *Convolutional Neural Network* (CNN) for feature extraction; the models based on *Recurrent Neural Network* (RNN) to handle sequence data; the sequence-to-sequence model and attention mechanism; the transfer learning technique withs pretrained language models. Finally, the paper concludes with some discussion about some most recent results and directions in this area.

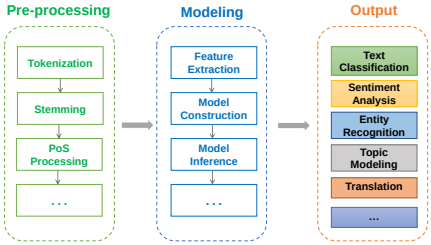

Figure 2. Classical NLP pipeline.

2. Processing pipelines of NLP applications

Like other data processing applications, NLP applications are usually developed on top of a *pipeline* to process data step-wise. In this section, we discuss two typical pipelines mostly used in classical and DL-based NLP applications.

2.1. The classical NLP Pipeline

In traditional approach, NLP consists of two main steps described in Figure 2. The first step is referred to as *Pre-processing*, which filters out redundant information and standardizes input text into a common format. Typical pre-processing actions includes *tokenization*, *stemming*, *stop word removal* and *Part-of-Speech (PoS) tagging*. The next step is called *Modeling*, which is in charge of introducing a model solving the intended tasks with three operations as follows.

- *Feature Extraction*: In machine learning, data is represented as features. Feature extraction process is of building feature set and filtering out important features that best represent input data. In terms of textual data, *bag-of-words* (BOW) [10] or *TF-IDF* vectors [11, 12] are common features.
- *Model Construction*: Model is the result of applying appropriate machine learning algorithms on input data. Traditional NLP approaches make use of supervised learning techniques including SVM [13, 14], NB Classifier [15] or MLP [16], and unsupervised methods such as clustering [6] or LDA [6, 17] to construct models.
- *Outcome Inference*: This is to apply the trained model on data to obtain the final results.

2.2. The Deep Learning-based NLP Pipeline

With the rising of Deep Learning, traditional NLP approaches are now gradually replaced by deep learning models. Though many highly complicated deep learning models are built for different NLP tasks, they basically follow the same pipeline described in Figure 3, which is carried out by the following operations.

- Similar to the classical pipeline, the raw textual data are firtsly pre-processed with the common modules of *tokenization*, *stemming*, *stop word removal*, etc.

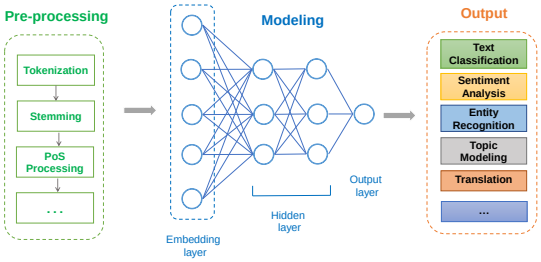

Figure 3. DL-based NLP pipeline.

- Data representation is essential in deep learning architectures. Different from traditional approaches which make use of document-level information, modern models use embedding layer to transform words or even characters into numerical vectors that contain higher amount of meaning information.
- The core of NLP system is a deep learning model. The model architecture may vary on the task it is built on. Some common deep architectures will be described in the next sections.

3. Word representation models

In the book entitled *Introduction to Information Retrieval* [18] published in 2008, the authors explained about the techniques to exploit information from various data sources, especially in the textual form. There are two well-known classical techniques for this goal discussed in this book, including *Count-Vectorization* or *Bag-of-words* (BOW) [10], and the weights of *Term Frequency - Inverse Document Frequency* (*TF-IDF*). These two methods, in particular *TF-IDF* weights, are proven effective in determination of important keywords in documents. However,

they cannot reflect the information of word positions in the documents and hardly convey the semantics of the words as well as the whole documents.

Word Embedding is a technique later emerging as a more effective method for word representation. This term was firstly coined by Yoshua Bengio in the famous paper entitly "Neural Probabilistic Language Models" [19] published in 2003. It recommended that a word vector can "embed" the semantics of words in a numerical vectors to let them be processed effectively by machines. Inspired by those idea, a number of word embedding techniques were then reported as *Word2vec* [20], GloVe [21], *fastText* [22], ELMO [23] and BERT [24]. In general, a word embedding technique should fulfill the following conditions.

- There is only one unique representation per word, i.e., two different words should be represented by two different vectors.
- Two similar words, in terms of their semantic meaning, should be represented by two similar vectors, in terms of their distance in the embedding space.

3.1. Word representation by one-hot vector

One-hot vector perhaps is one of the earlies technique to encode *categorical* data items as numerical vectors. Basically, a one-hot vector is a binary vector whose dimensional values are only of 0 or 1. Particularly, there is only one dimension has the value of 1 in a one-hot vector, whereas the rest are of 0. Based on that technique, one can represent every word in a text document as a one-hot vector. For instance, with a sentence of *"I am a member of URA research group"*, the word *am* can be represented as a one-hot vector as illustrated in Figure 4, whose number of dimension is exactly the vocabulary size of the corresponding language.

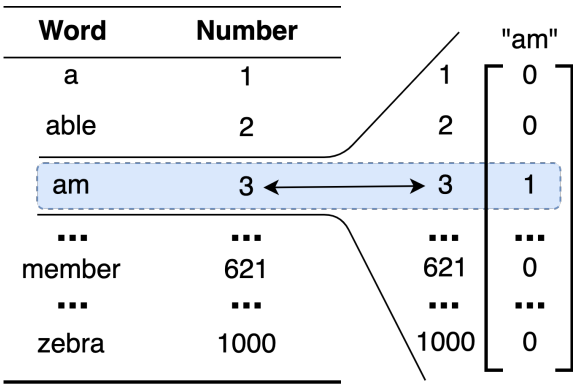

Figure 4. Representation of the word "am" as one-hot vector.

Thus, one-hot encoding can be employed to convert any word in a dictionary into a form of numerical vector. Therefore, this technique was used in many NLP and other machine learning models, especially in the case that the vocabulary size is relatively small. Obviously, this technique fulfills the requirement that different words should be represented as different vectors. However this technique still suffers from the following drawbacks.

- When working with a large vocabulary size, this technique tends to generate very sparse high-dimensional vectors, most of whose dimensional values are of 0.
- More importantly, it can be undeniably observed that the Euclidean distance between any pair of one-hot vectors is constantly of $\sqrt{2}$. This prevent one-hot vectors from representing the difference in semantics of the words.

3.2. The Word2vec technique

Word2vec is one of the popular techniques in the Natural Language Processing field. It was publicly announced in 2013 by a group of researchers led by Tomas Mikolov and has been registered for protection of the mode of the invention [20]. Based on *AutoEncode*r [25], *Word2vec*

has solved the problems related to context meanings of the *AutoEncoder* model by transforming each word of the corpus into a vector based on their contextual information in that corpus. Therefore, the model is able to learn so as to produce a similar vector for those words that share the same contextual information. The contextual information of a focus word is a window containing words on the left side and right side of the focus word, which are called *context words*. The size of the window, denoted by k, represents k words on the left side and k words on the right side of the focus word.

Two main training techniques of *Word2vec* are *CBOW* and *Skip-gram*.

CBOW (Continuous Bag of Words)

The idea behind the CBOW model is that it predicts the focus word vector based on the context words collected from a k-size window in which k is finite. Given a focus word w_t at the position t inside the sequence, the context words contains $(w_t - m, \ldots, w_t - 1, w_t + 1, \ldots, w_t + m)$ surround the focus word w_t in the window size of m.

The CBOW model can be generalized as illustrated by Figure 5. In Figure 5, C is the number of context words while V is the size of vocabulary and the hyperparameter N is the number of nodes in the hidden layers where N is relatively small compared to V. Every node of the hidden layer is fully connected to the nodes of the previous layer.

When the CBOW model is trained with a large corpus, we will gain a good representative matrix W with size $V \times N$ when the training process converges. From then, with an arbitrary one-hot vector T with size $1 \times V$, we can get an embedding vector E with size $1 \times N$ based on the product operation of $E = T \times W$.

For example, with this one-hot vector for the

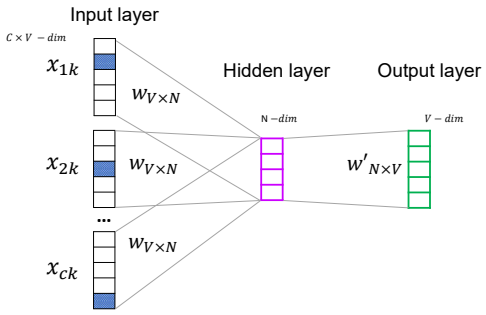

Figure 5. A general CBOW model.

word "am": $x_{am} = \begin{bmatrix} 0 \\ 0 \\ \mathbf{1} \\ \vdots \\ 0 \end{bmatrix}_{V \times 1}$

and matrix $W_{V \times N}$ shown below:

$$W^T_{V \times N} = \begin{bmatrix} 2.4 & 1.6 & 1.8 & \ldots & 3.2 \\ 2.6 & 1.4 & 2.9 & \ldots & 6.1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1.8 & 2.7 & 1.9 & \ldots & 1.2 \end{bmatrix}_{N \times V}$$

We can perform matrix multiplication $v^T_{am} = W^T \times x_{am}$ and receive the resulting matrix as below:

$$v^T_{am} = \begin{bmatrix} 2.4 & 1.6 & \mathbf{1.8} & \ldots & 3.2 \\ 2.6 & 1.4 & \mathbf{2.9} & \ldots & 6.1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1.8 & 2.7 & \mathbf{1.9} & \ldots & 1.2 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ \mathbf{1} \\ \vdots \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{1.8} \\ \mathbf{2.9} \\ \vdots \\ \mathbf{1.9} \end{bmatrix}$$

As we can see, vector v^T has the dimension

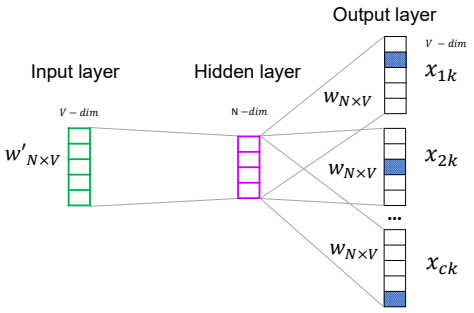

Figure 6. A general Skip-gram model.

N which is relatively small compared to V. This vector is no longer considered as a sparse vector since all of its values are now different from 0. Moreover, the training procedure of the CBOW model also helps create embedding vectors with closer distances for the words that frequently appear together in the context words. After the CBOW training converges, the matrix $W_{V\times N}$ will be used as an embedding layer for other deep learning models specialized in NLP tasks as illustrated in Figure 3.

The skip-gram model

The skip-gram model is demonstrated in Figure 6. This model is similar to CBOW; however, the focus word is now used as the input for the model and the context word as the expected output.

4. Convolutional Neural Network for NLP

4.1. Convolutional Neural Network

Convolutional Neural Network, known as *CNN* or *ConvNet*, is one of foundation techniques in the area of *Computer Vision*, or *Deep Learning* in general. This kind of architecture is typically used for image classification, such as the problem of facial recognition. This architecture is proposed by the famous scientist Yann LeCun, winner of the Turing award in 2018, for the problem of handwritten zip code recognition [26]

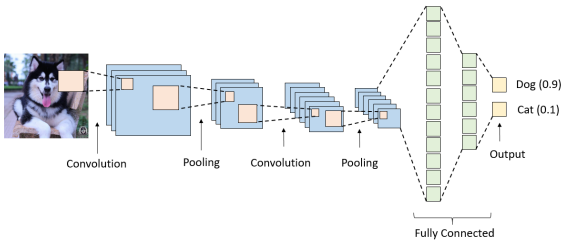

Figure 7. An illustrative CNN architecture.

Basically, a CNN architecture consists of the following layers: *convolutional*, *pooling* and *fully connected*. In certain variants, convolutional and pooling layers can be repeated many times before feeding to the final fully connected layer, as illustrated in Figure 7.

In the typical Feedforward Neural Network, the layers directly connected via a weight vector W, known as fully connected layer or *affine layer*. In constrast, in the convolutional layer, the layers are connected via the convolution mechanism. That is, each neural of the next layer connects to a local region of the previous layer via a *convolution window*, or *filter*. During the training process, the CNN network will automatically learn the appropriate weights of the filters.

After the convolutional layer is the *pooling layer*, or *subsampling layer*. The objective of this layer is to standardize the output vectors of the filters, to make them have the same dimension space using a *pooling window*. In addition, by only selecting a value in each pooling window as the next output, the number of parameters to be trained would be significantly reduced, as compared to a corresponding MLP network with the same number of neural nodes.

Figure 8 illustrates the computational process in the convolutional and pooling layers of a CNN network. The output results from the each filter will be eventually concatenated, producing the fi-

nal *feature map*, which can in turn be used as input for the next similar process, or fed to the a fully connected layer to get the final prediction of the model.

4.2. CNN for NLP tasks

The CNN network is initially intended to work with images represented as matrices. Thus, in order to be applied with NLP tasks, the original textual documents needed to be represented as matrices as well. With the Word Embedding technique discussed previously, each word in a document can be encoded as vector of the same dimension. Then, a document can be formed as a matrix, on which the CNN can be applied as depicted in Figure 9. In this example, the problem of Sentiment Analysis is picked as a case study, adapted from the work published in [27], which is considered as a pioneered work that applied CNN for NLP. Interested readers can further refer to this paper for the way to select suitable *hyperparameters* when implementing the model.

For more details, we can consider the architecture in Figure 9 as 5 parts, namely #sentence, #filters, #featuremaps, #1-max and #concat-1-max, whose information is given as follows.

#sentence

This is the input of the system, in this case is a sentence of 7 words: *"The film contains a few funny bits"*. Assuming that the embedding dimension is 5, the sentence is then represented as a 7×5 matrix.

#filters

Initially, CNN is used for Computer Vision applications, whose input is normally a 2D or 3D images/objects. Thus, their filters are also organized as 2D matrices or 3D cubes to reflect this dimensional characteristic. However, for textual data, the input is naturally organized as a 1D series of word. In that case, the filters adopted by a NLP-intended CNN would also be adapted accordingly. To be more precise, the number of columns in the filter will be the same of the size of the embedding word vector, meanwhile its rows are used to reflect the concept of *n*-gram of text data. In this example, the row for each filter will be either in (2,3,4), corresponding to the concepts of *bigram*, *trigram* and *4-gram* of classical NLP. The are two filters for each kind, eventually making 6 filters for whole system.

#featuremap

In this context, the feature maps are the matrices generated once we perform the convolutional operation between the *#filters* on the *#sentence*. Since the columns of the filters are the same with the dimension of the embedding space, each feature map is then a column matrix whose size is varied depending on the filter size.

#1-max

This is in fact corresponding to the pooling layer in the standard CNN architecture, where the maximal value in each column vector of the feature map will be selected for further processing. As a result, we have a set of 6 maximal numbers from 6 column vectors.

#concat-1-max

In this step, the results from *#1-max* are concatenated together, making another column vector of 6 element, which is then connected to a *softmax* layer using the fully connected mechanism. The result of this *softmax* layer is treated as a binary classification of sentiment opinion, e.g *positive* or *negative*.

In addition, CNN is also considered for information modeling at character level. For instance, Santos [28] và Labeau [29] had successfully leveraged this work for the problems of NER and POS-tagger, respectively.

5. Recurrent Neural Network for NLP

5.1. Sequence data

Natural language data is an ordered sequence of data. Unlike the traditional data type where the patterns are unordered and independent of each

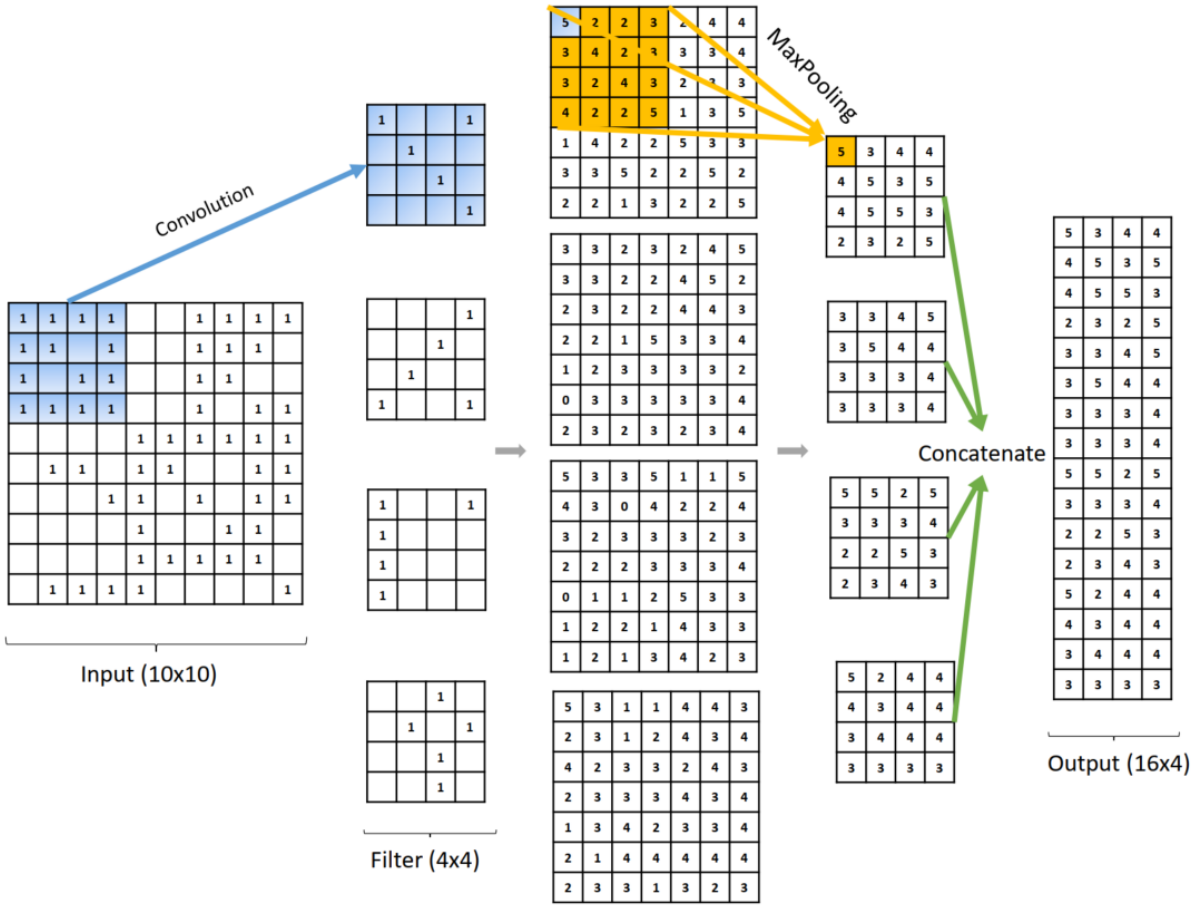

Figure 8. The generation of a feature map in a CNN network via convolutional and pooling layers.

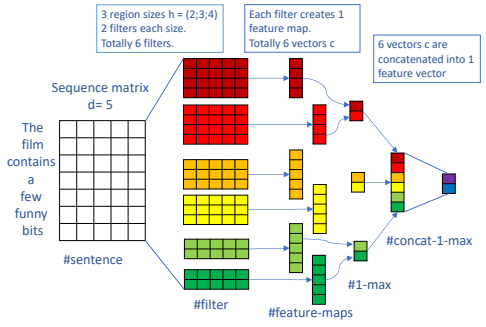

Figure 9. Using CNN to perform the task of Sentiment Analysis.

other, with natural language data, the order of occurrence of the data patterns also carries certain information. In text data, the order in which words appear in a text plays a very important role in conveying the meaning of a sentence or paragraph.

Feed-forward networks with fixed input data cannot solve the problem with arbitrary length sequence data. However, sequence data can be transformed (split) for using on input-only architectures by taking a fixed amount of k samples of previous consecutive data as input. Models that approach this direction are classified as *auto regressive models*. In general, a generic k-degree auto regressive model can be understood

as a mapping f:

$$\hat{y}_i = f(x_i, y_{i-1}, y_{i-2}, ..., y_{i-k}) \quad (1)$$

where x_i is the existing observation, y_i is the sequence value at time i, y_i is the predicted value at time i, $\hat{y}_i$ is the sequence value at time i, k is the model degree. With the above approach, we temporarily solve the problem of vary data length. However, there are still many problems that make MLP networks unsuitable for sequence data as follows.

- The MLP model is not yet able to remember the state from previous data to support decision making. In fact, it still depends entirely on the input. In case it is necessary to remember data at a length greater than k-degree of the model, the feed-forward network will not be able to make an accurate prediction. Indeed, the way in which the above regressive model derives its data from earlier samples in the sequence is entirely up to the assumptions of the network designer.
- The number of parameters is very large: when using a fully connected layer, the number of parameters of a layer will be equal to the product of the input dimension and the output dimension. The large number of parameters also makes the model take up a lot of resources both in terms of memory requirement and training time.
- The model is not capable of sharing weights between steps (fully connected layers) and uses a large number of parameters specific to each input on every layer.

5.2. Recurrent Neural Network

In 1986, Rumelhart and his colleagues published a research paper entitled ”Learning internal representations by error propagation” [30]. In the study, a classic network architecture for processing sequence data is presented, called a *recurrent neural network*. Basically, a recurrent neural network is a neural network architecture in which the state of the model at the previous step is used as the input of the current step. In other words, a recurrent neural network model behaves in a way that maps the input data and the previous state of the model to output data, as shown in Figure 10.

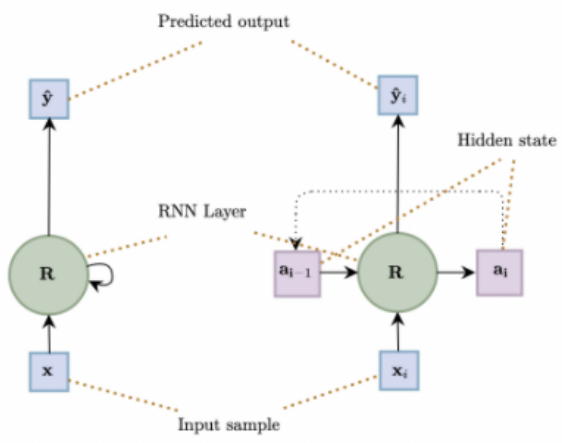

Figure 10. The conceptualization of an RNN

With a traditional feed-forward neural network, data are only passed in one direction, computation and output results are encapsulated in one phase and based entirely on input data (i.e. no loops exist). We need to clarify that the loop here does not create an infinitely repetitive computation cycle, but only uses data from the pre-computation phase as input to the current phase. More specifically, we can ”unfold” the loop to provide a more specific computational architecture when passing sequence data to the network. With a recurrent neural network, the information from the previous transmission is saved (in the hidden state) for the current computation, creating a loop on the computation graph. In other words, the hidden state acts as a memory of the recalling neuron layer. This is especially useful when dealing with time-series data, where the same calculation logic (same calculation and set

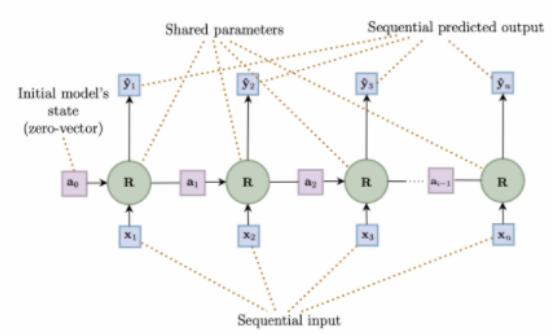

Figure 11. Sequence data processing with RNN

of weights) can be applied to each data point in the series, preserving information from previous points. while preserving sequence information. Figure 11 illustrates how RNNs process sequence data.

Mathematically, all calculations in an RNN are presented in (2a) - (2c).

$$\begin{cases} a_0 = 0 & \text{(2a)} \\ a_i = f(\mathbf{W}_{ax} \cdot x_i + \mathbf{W}_{aa} \cdot a_{i-1} + \mathbf{b}_a) & \text{(2b)} \\ \hat{y}_i = g(\mathbf{W}_{ya} \cdot a_i + \mathbf{b}_y) & \text{(2c)} \end{cases}$$

where

- a_i is the hidden state of the model and $a_0 = 0$ is assumed the initial state.
- $\hat{y}_i$ is the output of corresponding i-th step.
- x_i is the input of corresponding i-th step.
- f and g is the appropriate activation functions. Usually, we choose $f(x) = tanh(x)$ or $f(x) = ReLU(x)$ and $g(x) = sigmoid(x)$ or $g(x) = softmax(x)$ according to the design method and problem requirements.
- $\mathbf{W}$ and $\mathbf{b}$ are model weights
  - $\mathbf{W}_{ax}$ is the matrix of linear mapping which transforms x to a part in state a.
  - $\mathbf{W}_{aa}$ is the matrix of linear mapping which transforms old state a to a part in new state a.
  - $\mathbf{b}_a$ is the bias of transformation from old state a to a part in new state a.
  - $\mathbf{W}_{ya}$ is the matrix of linear mapping which transforms current state a to the output $\hat{y}$.
  - $\mathbf{b}_y$ is the bias of transformation from current state a to the output $\hat{y}$.

5.3. Long Short-Term Memory Network

Even though the recurrent neural network is theoretically a simple and powerful model, it is difficult to learn properly due to a limit to capture long-term dependencies, caused by these two well-known issues in training a model: *vanishing* and *exploding gradient* [31]. The vanishing gradient will become worse when a *sigmoid* activation function is used, whereas a *Rectified Linear Unit* (ReLU) can easily lead to an exploding gradient. Fortunately, a formal thorough mathematical explanation of the vanishing and exploding gradient problems was represented by Bengio *et al.* [32], analyzing conditions under which these problems may appear.

To deal with the long-term dependency problem, a developed version of RNNs was introduced by Hochreiter and Schmidhuber in 1997, called *Long Short-Term Memory* (LSTM) [33]. LSTMs has overcome the limitations of RNNs and delivered more accurate performance by using a hidden layer as a memory cell instead of a recurrent cell (see Figure 12). In the standard LSTM model, processing information is more complicated when modules containing computational blocks are repeated over many time steps to selectively interact with each other in order to determine what information will be added or removed. This process is controlled by three *gates* namely *input gate*, *output gate*, and *forget gate*. Controlling the flow of information inside

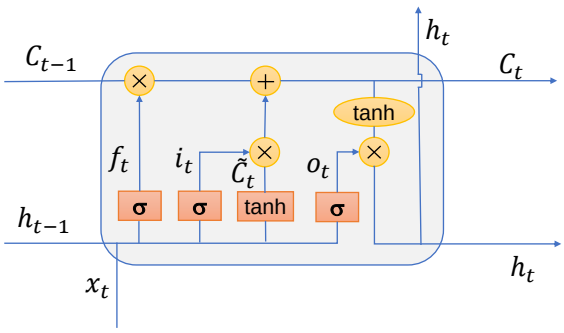

Figure 12. The structure of an LSTM cell

an LSTM model can be described as the following (3a)-(3e) mathematical equations.

$$\begin{cases} i = \sigma(\mathbf{W}_i x_t + \mathbf{W}_{hi} h_{t-1} + \mathbf{b}_i) & \text{(3a)} \\ f = \sigma(\mathbf{W}_f x_t + \mathbf{W}_{hf} h_{t-1} + \mathbf{b}_f) & \text{(3b)} \\ o = \sigma(\mathbf{W}_o x_t + \mathbf{W}_{ho} h_{t-1} + \mathbf{b}_o) & \text{(3c)} \\ (C)_t = (C)_{t-1} \otimes f + tanh(\mathbf{W}_C x_t + \mathbf{W}_{hC} h_{t-1} + \mathbf{b}_C) & \text{(3d)} \\ h_t = o \otimes tanh((C)_{t-1}) & \text{(3e)} \end{cases}$$

In (3a)-(3e), i, f, o, (C), h denote input gate, forget gate, output gate, internal state, and hidden layer, respectively. x_i is the input of corresponding i-th step. Next, $\mathbf{W}_i$, $\mathbf{W}_f$, $\mathbf{W}_o$, $\mathbf{W}_C$, and $\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_o$, and $\mathbf{b}_C$ represent the weights and bias of three gates and a memory cell, in the order given. Concretely, the activation function *sigmoid* (σ) helps an LSTM model to control the flow of information because the range of this activation function varies from 0 to 1 so that if the value is 0 then all of the information will be cut off, otherwise the entire flow of information will pass through. Similarly, the output gate will allow information to be revealed appropriately due to the *sigmoid* activation function then the weights will be updated by the element-wise multiplication of output gate and internal state activated by nonlinearity *tanh* function. With the pivotal component is the memory cell accommodating three gates: input, forget, and output gate, LSTMs has overcome limitations of RNN, enhancing the ability to remember values over an arbitrary time interval by regulating the flow of information inside the memory cell. Therefore, LSTMs possess a capacity to work tremendously well on learning features from sequential data such as documents, connected handwriting, speech processing, or anomaly detection [34].

Later, a simpler version of LSTM, known as GRU [35], was also introduced. GRU uses no hidden memory and fewer gates than LSTMs does. As a result, GRU has a faster training process, consumes less memory while maintains high performance and can compete with LSTMs in some cases.

5.4. Bi-LSTM and NLP applications

RNN and its upgraded versions like LSTM and GRU can handle sequence data well in NLP problems. To enrich information encoded in each step, NLP applications often use a *bidirectional* mechanism, known as *Bidirectional LSTM* (BiLSTM) or *Bidirectional GRU* (BiGRU), in which data is processed across two opposing LSTM or GRU networks. The result at each step of these two opposite networks will be concatenated to the final representation of each step. In addition, because CNN is very powerful in feature extraction, RNN-based architectures often combine with CNN to extract features from the original text before racing to be processed in RNN layers. Figure 13 illustrates architecture of using CNN-BiLSTM for named entity recognition [36]. In addition, the same architecture is also used for sentiment analysis [37], PoS tagging [38], text classification [27] problems.

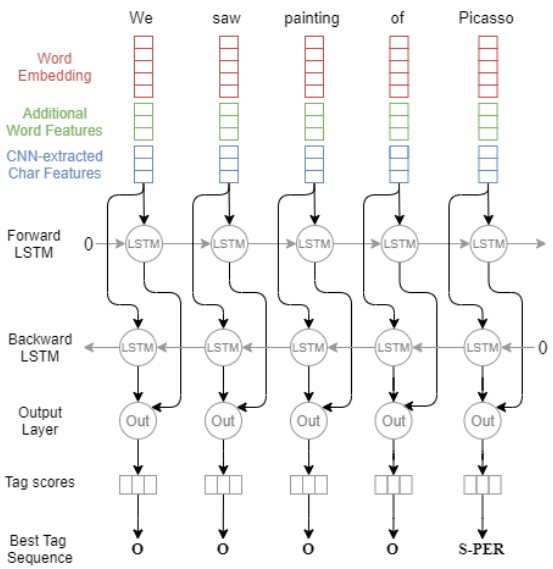

Figure 13. A CNN-BiLSTM architecture for NER task.

6. Sequence-to-sequence model and attention mechanism

6.1. Sequence-to-sequence architecture

Sequence-to-sequence (seq2seq) [39] is a deep-learning architecture initially devised for machine translation. Its intuition comes from an observation such that in order to translate a text, human would first read some parts of the text and then start to do the translation. NLP researchers also employ that idea into designing a structure dubbed as *sequence-to-sequence*.

Basically, sequence-to-sequence model takes the design principle of comprising 2 components, an *encoder* and a *decoder*, as illustrated in Figure 14. For a vanilla sequence-to-sequence model, both the encoder and decoder are recurrent neural networks, but in order to capture long-term dependencies, we could choose LSTM or GRU instead. In general, the encoder has a function to digest and understand the input text as well as encode the information into a fixed-shape hidden state. Output sequences are generated by using the method of token-by-token, which means

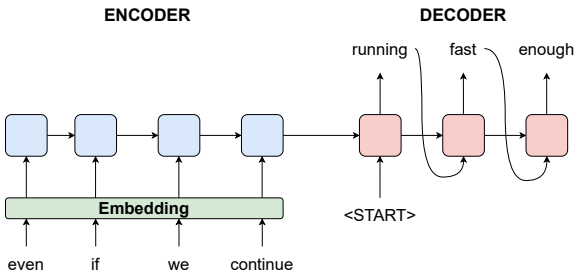

Figure 14. An example of a sequence-to-sequence model.

the previously created token will be the next time step input token. The decoder combines two source inputs encoder hidden state and current input token to consider the probability of generating the next word. The process of generating output will stop if the sequences touch the maximum length of the result or the current token is a special "end" token.

Encoder

For an input sequence $x_1, \ldots, x_T$, the encoder will read and print out a fixed-shaped vector $\mathbf{c}$ taking the context into account. We assume that the input sequence is $\mathbf{x}_1 \cdots \mathbf{x}_T$ known that $\mathbf{x}_t$ is the t^{th} token and. $\mathbf{h}_t$ is the hidden state of time step t. Then, let us express f as the computation of an RNN unit as follows.

$$\mathrm{h}_t = f(\mathrm{x}_t, \mathrm{h}_{t-1}) \tag{4}$$

Eventually, after passing through all the tokens in the sequence, the encoder transforms the whole sequence into a context vector.

$$\mathrm{c} = q(\mathrm{h}_1, \ldots, \mathrm{h}_T) \tag{5}$$

Usually, q function takes the final hidden state of encoder into context variable c. In order to make the encoder has better performance, instead of using unidirectional RNNs which just focus on previous hidden states, we can use bidirectional networks to consider both sides - the previous and after ones.

Decoder

As mentioned in the last passage, an encoder converts the input sequence into a context vector c. Hereafter, it is time for the decoder to evaluate the conditional probability $P(y_{t'} \mid y_1, \ldots, y_{t'-1}, \mathrm{c})$ given the context vector c and past decoded tokens $y_1, \ldots, y_{t'-1}$.

At any time step t′, decoder takes the output token $\mathrm{y}_{t'-1}$, context variable c and previous decoder hidden state $\mathrm{s}_{t'-1}$ then transforms them into current decoder hidden state as below

$$\mathrm{s}_{t'} = g(\mathrm{y}_{t'-1}, \mathrm{c}, \mathrm{s}_{t'-1}) \tag{6}$$

Note that g is a transform function. After gaining the current decoder hidden state, sequence-to-sequence will put effort into estimating the conditional probability by applying a *softmax* operation on output layer, which is a fully-connected layer to transform final hidden state. Then, the final probability is calculated by multiple probability of each word along time steps.

$$\begin{aligned} & p(y_1, \ldots, y_m \mid x_1, \ldots, x_n) \\ &= \prod_{t=1}^{m} p(y_t \mid \mathrm{c}, y_1, \ldots, y_{t-1}) \end{aligned} \tag{7}$$

6.2. Attention mechanism

Sequence-to-sequence model causes the RNN problems such as vanishing gradient excessively severe. Although the encoder and decoder can take LSTM or GRU forms, the performance still deteriorates for input with a length approximating 100 tokens, which is sorely popular in tasks such as machine translation or text summarization.

Besides, in prominent tasks such as machine translation, there are a number of alignments between the output and input. That is, some words in the output could be linked to some from the input sequence. For instance, translating from "And the programme has been implemented" (English) to "Le programme a ét é mis en application" (French), phrases such as "programme" and "mis en application" shall be mapped to "programme" and "implemented" in the respective order.

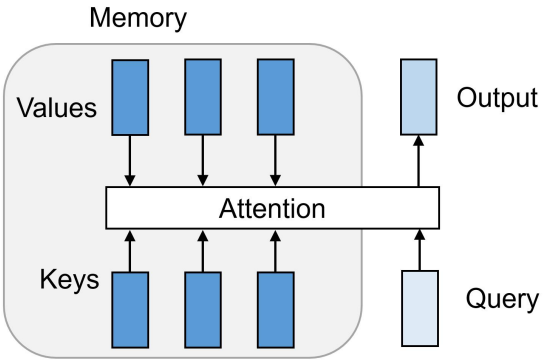

Figure 15. A simple attention model.

Bahdanau et al., [40] proposed an *attention* mechanism to implement that concept. The input of the attention scheme is a query vector q. For a query, attention returns an output related to the memory consisting of key and value pairs $(k_1, v_1), ..., (k_n, v_n)$.

We will start to derive the output of the attention. For each key $k_1, ..., k_n$, we performs the *dot product* with the query to learn the corresponding value. This operation demands that the query and the key have the same dimension d. For query and key $\mathrm{q}, \mathrm{k} \in \mathbb{R}^d$,

$$\alpha(\mathrm{q}, \mathrm{k}) = \langle \mathrm{q}, \mathrm{k} \rangle / \sqrt{d} \tag{8}$$

To compute operations effectively, we often employ matrix computation. Assuming that input attention has n queries and m key-value pairs. Length of queries and key is d, while length of values is equal v. So, we have matrices called query $\mathrm{Q} \in \mathbb{R}^{\mathrm{nxd}}$, key $\mathrm{K} \in \mathbb{R}^{\mathrm{mxd}}$ and value $\mathrm{V} \in \mathbb{R}^{\mathrm{mxv}}$. Then, the equation becomes

$$\alpha(\mathrm{Q}, \mathrm{K}) = \frac{\mathrm{QK}^\top}{\sqrt{d}} \cdot \mathrm{V} \tag{9}$$

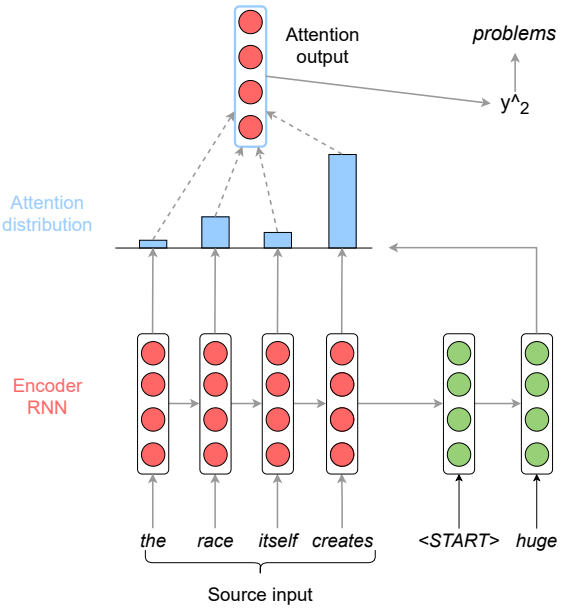

Figure 16. Sequence-to-sequence model with attention.

Additive Attention In general, we may utilize additive attention as the attention scoring function when queries and keys are vectors of various lengths. For query and key $\mathrm{q} \in \mathbb{R}^{\mathrm{q}}, \mathrm{k} \in \mathbb{R}^{\mathrm{k}}$,

$$\alpha(\mathrm{k}, \mathrm{q}) = \mathrm{v}^{\top} \tanh(\mathrm{W}_k \mathrm{k} + \mathrm{W}_q \mathrm{q}) \qquad (10)$$

where $\mathrm{W}_k \in \mathbb{R}^{\mathrm{hxk}}$, $\mathrm{W}_q \in \mathbb{R}^{\mathrm{hxq}}$ are two weight parameters corresponding to key k, query q respectively and $\mathrm{w}_v \in \mathbb{R}^{\mathrm{h}}$.

6.3. Enhancement of seq-to-seq model by attention

In Figure 16 is an example of using sequence-to-sequence with attention mechanism for machine translation. When endeavoring to generate the next token, the hidden representation will attend to the representations of input tokens. As it can be seen from Figure 16, with a view to obtain the attention scores, the system will compute the dot products between the representation of "he" with those of the source input. Subsequently, the computed scores will play a role as the weights to coalesce the hidden representations of input tokens to form the attention output. Eventually, this output will be employed to determine the logits to generate the output token.

7. Pre-trained Language Models

7.1. Language models

In NLP, language models are models that can assign probabilities for a sequence of words or a sentence. In addition, language models can infer *likelihood* for a particular word (or sequence of words) from a preset input sequence. In mathematics methodology, the probability assignment for a word sequence can be modeled by chain rule in Equation 11:

$$P(w_1, w_2, ..., w_n) = \prod_{i=1}^{n} P(w_i | w_{1:i-1}) \qquad (11)$$

where w_i denoth the i-th word in a sentence.

7.2. Language model construction with RNN

In classical approaches, probability assignment is usually based on n-gram Markov model [41]. However, with the help of Deep Learning, recent approaches choose RNN architectures to train language models because of their outperformanced leaning capability on large dataset. Moreover, language models trained from RNN can be efficiently used as pre-trained language model for other NLP tasks as described below.

RNN works by using probabilities of previous words to generate probability for the next word. The generated word is then used as input of RNN to produce the next word until the end-of-sequence word is produced as depicted in Figure 17. Finally the probability of the whole sequence is computed as in Equation 11.

Let "Today I go to school. Tomorrow I go to school too." be an input sequence for RNN. It is

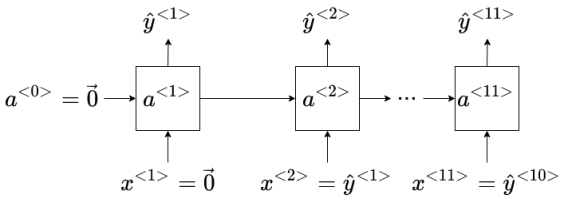

Figure 17. Language model with RNN

supposed that the corpus only contains words in this input sequence, then each word in corpus is a 1×7 vector in one-hot presentation. For example $x^{<0>}$ is a zero vector of size corpus length, which is 1x7 in this case. The first hidden state $a^{<0>}$ is a zero vector of any particular size based on our decision, which is 1×5 in this example. W_{aa} is a 5×5 matrix and W_{ax} is a 7×5 matrix so $a \times W_{aa} + x \times W_{ax}$ is a 1×5 matrix. The 5×7 matrix W_{ay} is then used to resize this hidden state to desired output size 1×7. The result is fed into softmax activation function to produce output vector $\hat{y}$. This output acts as a probability distribution for each word in corpus. Finally, the appropriate loss criteria based on y and $\hat{y}$ is used to update model's weights. The whole procedure is illustrated in Figure 18.

7.3. Pre-trained language models

Since 2018, many state-of-the-art NLP researches are based on pre-trained language models such as ELMo [23], OpenAI GPT [42] or ULMFit [43]. There are two existing strategies for applying pre-trained language representations to downstream tasks as follows.

- *Feature-based*: This strategy includes pre-trained representations as additional features (e.g., ELMo)
- *Fine-tunning*: This strategy introduces task-specific parameters and fine-tune the pre-trained parameters (e.g., OpenAI GPT, ULMFit)

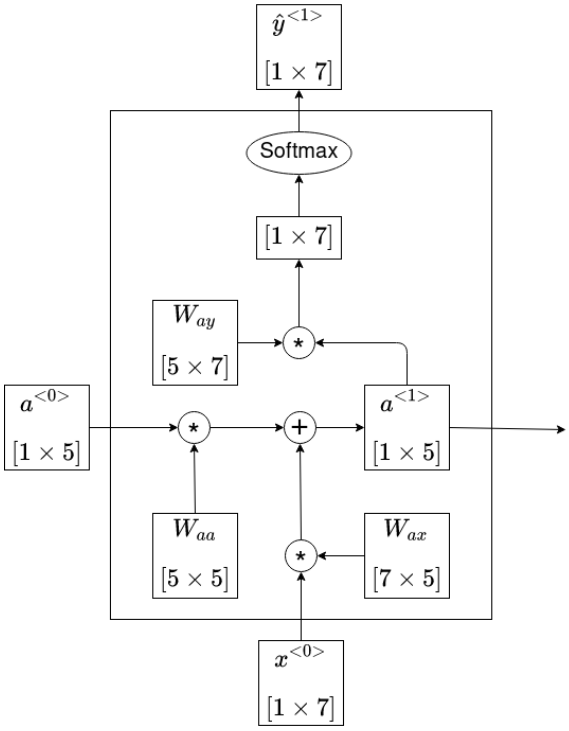

Figure 18. Detailed language model training procedure at one timestep with RNN.

Figure 19 illustrates a representative situation using pre-trained language models for downstream task using UMLFit. This is a language model with many LSTM layers stacked on to each other. This model is first trained with a very large dataset then fine-tuned on domain dataset. Finally, with a specific target application dataset, the model with pre-trained weights is used as an embedding layer to represent input values as vector. These representation vectors is then used as input for other downstream task such as classification by SVM [44].

Limitation of these language models is unidirectional manner because they restrict the power of the pre-trained representations. For example, OpenAI GPT uses left-to-right architecture whereas ELMo concatenates forward and backward language models. In 2018, BERT [24] was introduced, which based on Transformer archi-

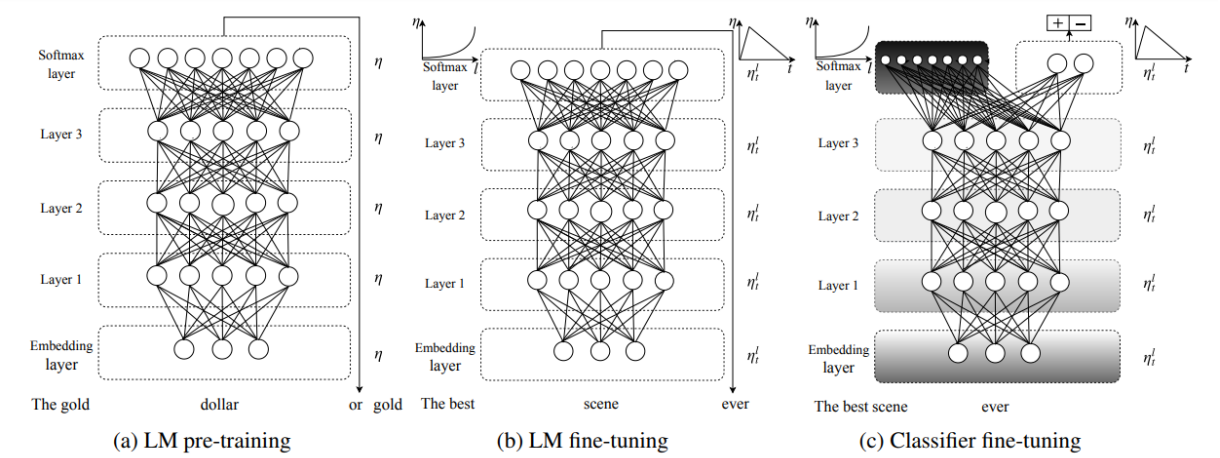

Figure 19. Using pre-trained language model of UMLFit [43].

tecture [45] became a more comprehensive solution.

7.4. Transformer model

RNN units with their counterparts such as LSTM and GRU have established state-of-the-art results in multiple sequence modeling tasks. Combined with attention, the performance of problems such as machine translation and text summarization has been upgraded to a new level. Nonetheless, there is still one obstacle with those architectures, which is embedded in the sequential nature of the input and output. The incapability to parallelize the computation has caused a huge overhead in computation time. Transformer's authors proposed an architecture dubbed as Transformer removing the recurrence or convolutional layers. Instead, they suggested that solely relying on attention is the key to achieve parallelism. Self-attention is a core mechanism that connects distinct places of a single sequence to compute a representation of it. Almost NLP tasks such as reading comprehension or learning task-independent sentence representations have all been effectively utilized with self-attention.

Seq2Seq paradigm in Transformer

Transformer also features the *seq2seq* architecture including encoder and decoder combining with stacked self-attention and fully-connected layers. However, instead of including a single layer in every part, (Vaswani et al., 2017) made a decision to construct two stacks of encoder and decoder.

Encoder

In particular, an encoder is composed of 6 identical layers stacked upon one another in which a layer is made up of 2 sub-layers. The specified order is self-attention computation, subsequently followed by a position-wise fully-connected feed-forward network. At the end of two layers, the authors attached a residual connection integrated into a layer normalization. All sub-layers in the model, as well as the embedding layers, provide outputs of size d_{model} to support these residual connections.

Decoder

Analogous to encoder, the decoder comprises a stack of 6 layers. Besides two sub-layers, there is also a third sub-layer to carry out the attention between the latest output and the encoder ones. Furthermore, at first sub-layer, instead of fully attending to the whole sequence in the input, for the time step t', the decoder only performs attention from the current token to tokens $1, \ldots, t-1$.

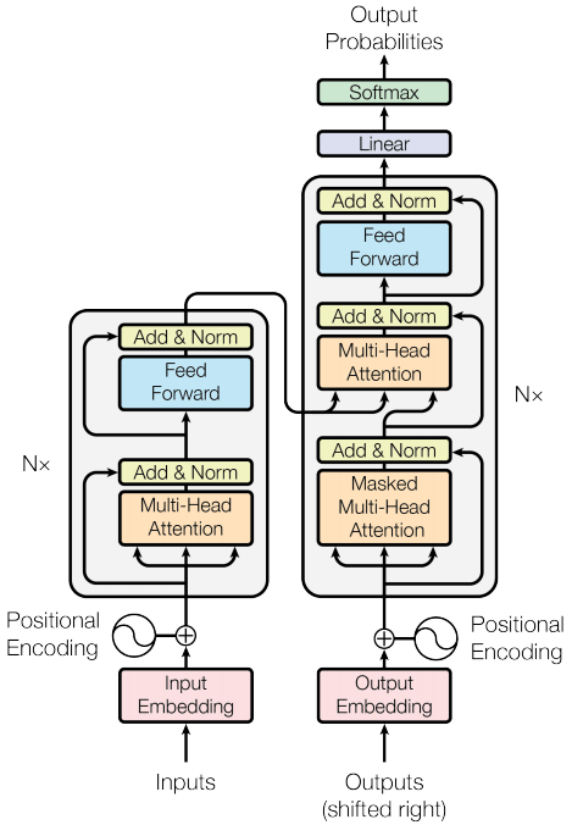

Figure 20. Transformer architecture [45]

.

Thanks to this masking method and the output tokens are offset by one place, the predicted token for position i can only be based on the known outputs at locations less than i.

Self-attention

There is a key factor to differentiate between the "original" attention and self-attention. Whereas attention evaluates the alignment of a token to its previous ones in a recursive manner, self-attention allows one to fully attend to all of the tokens in the sequence, making parallelism possible. In particular, for all keys, queries and values are packed into matrices K, Q and V. The attention scoring function used is scaled dot-product attention. After calculating scoring, we divide by $\sqrt{d}$ and then apply softmax function to show attention distributions on determined values.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \quad (12)$$

The scaling factor $\frac{1}{\sqrt{d_k}}$ is to reduce the effect of the dimension to the final output. The reason why attention function is chosen with scaled dot-product but not addictive attention is that the former is much faster and has space-efficient.

Multi-Head Attention

One interesting result of Transformer [45] is that they found it is advantageous to conduct attention multiple times with respect to a wide variety of linear projections. Multi-head attention allows the model to simultaneously attend to input from several representation aspects at various locations. Whilst using a single attention head, model will learn average of all these aspects, which not detailed. The detailed equation is as follows.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, \ldots, head_h)W^O$$
$$\text{where } head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Note that $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ are parameter matrices for projecting queries, keys and values. In particular, $d_k = d_v = d_{model}/h$ where h is the number of heads.

Positional Encoding

In sequential data, positional information bears an important effect to the semantics of the sequence. Since Transformer is not established on recurrence nature, it must encode another method to take into account that kind of information.

There are several learnt and fixed positional encoding types to choose from. Specifically, Transformer authors took advantage of sinusoidal signal, in which they transported the positions into sine and consine functions. The positional encodings and embeddings layers have the same dimension, thus they could be added together. According to the experiment of authors, the first method

using learned positional embedding and the second which use sinusoidal signal gain the nearly identical results.

$$\mathrm{PE}_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$\mathrm{PE}_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

where pos stands for position and i is the dimension. That means each dimension in the encoded vector would represent a sinusoidal wave.

7.5. Transformer-based Language Models

Transformer model has resulted in the development of a large number of pre-trained language models. Some outstanding models can be mentioned as GPT-2 [46], BERT [24] and Transformer XL [47]. GPT-2 employs layered Transformer decoder layers. In the meanwhile, BERT employs Transformer encoder layers, and Transformer XL provides a recurrent decoder architecture based on the Transformer decoder.

These days, perhaps the most commonly utilized Transformer-based architecture in NLP tasks is BERT. The version landscape of BERT is depicted in the Figure 21.

BERT's overall architecture is shown in Figure 22, which comprises a stack of Transformer encoder's layers. Furthermore, BERT uses two training techniques during pre-training steps, whose details are as follows.

Masked LM (MLM)

The concept is straightforward: Masking 15% of the words in the input with a [MASK] token at random, then running the full sequence through the BERT attention-based encoder and forecasting just the masked words given on the context provided by the other non-masked words in the sequence. This basic masking strategy, however, has a flaw: the model only tries to predict the correct tokens when the [MASK] token is present in the input, while we want the model to try to predict the proper tokens regardless of the masked token's presentation in the input. To address this issue, 15 percent of the tokens chosen for masking were:

- 80% of the tokens are actually replaced with the token [MASK].
- 10% of the time tokens are replaced with a random token.
- 10% of the time tokens are left unchanged.

During training, the BERT loss function only considers masked token predictions and ignores non-masked token predictions. As a result, the model converges far more slowly than models that are left-to-right or right-to-left.

Next Sentence Prediction (NSP)

The BERT training procedure also uses next sentence prediction to understand the relationship between two sentences. For jobs like question answering, a pre-trained model with this level of knowledge is useful. During training, the model is given pairs of sentences as input and is taught to predict if the second sentence is the same as the next sentence in the original text. BERT uses a specific [SEP] token to separate sentences. The model is fed two input sentences at a time during training, as follows:

- 50% of the time the second sentence comes after the first one.
- 50% of the time it is a random sentence from the full corpus.

BERT is then required to predict whether the second sentence is random or not, with the assumption that the random sentence will be disconnected from the first sentence. To determine if the second phrase is connected to the first, the entire input sequence is passed through a Transformer-based model, the final hidden state of the [CLS] token is transformed into a 2×1 shaped vector by using a simple classification layer, and the probability of IsNextSentence-Label is calculated by softmax - an activation function.

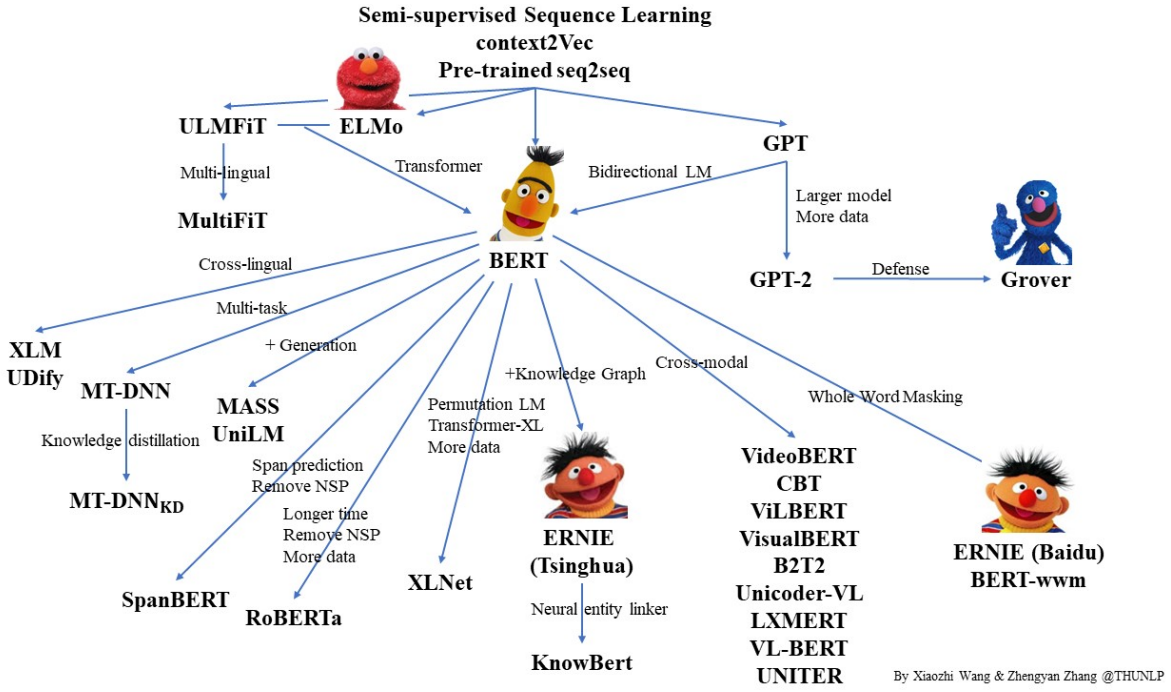

Figure 21. Variants of BERT [48].

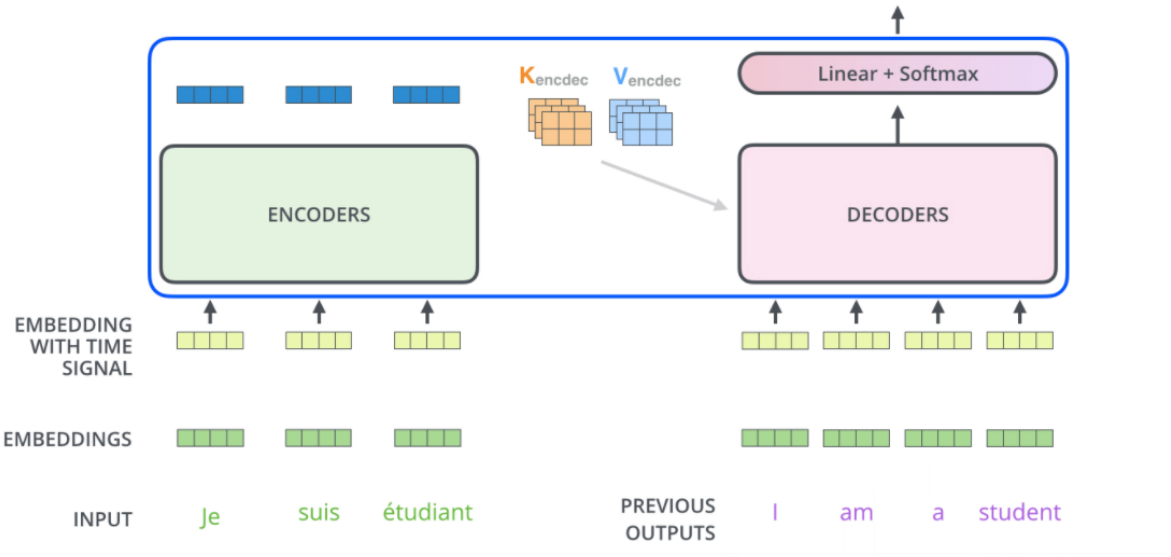

Figure 22. BERT general architecture [49].

Both Masked LM and Next Sentence Prediction are used to train the model. This is to minimize the combined loss function of the two strategies. The following are some applications

of BERT's usage in common NLP tasks:

- *Text Summarization*: the most commonly used model is BERTSUM [50] to produce summary by highlighting or identifying key sentences.
- *Text Classification*: one of the typical Text Classification is Sentiment Analysis - labeling a sentence with positive, negative or neutral emotions.
- *Question Answering*: the input is a question related to a given text and BERT is asked to mark the answer by identify two vectors marking the beginning and the end.
- *Named Entity Recognition*: input is a passage and the model is requested to find the different types of entities (*Person*, *Organization*, *Date*, etc.) that appear in the source.

8. Conclusion

In the era of the Fourth Industrial Revolution, digital data have been incrementally generated in all domains in an automatic manner, calling for intelligent systems to effectively process them for making useful information and knowledge for human beings. As natural languages are still the main channel for communication in human societies, Natural Language Processing has emerged as one of the important approaches to response such a call. Especially, with the advancement of Deep Learning techniques, NLP researchers find for themselves powerful tools to handle enormous textual computer-generated data, which bring NLP tasks to new breaking results.

In this paper, we have introduced an overall roadmap about the modern approaches in NLP. As the scientists in this area never cease to explore novel research directions for practical application, currently there are several ongoing interested works are under investigation for further processing and extentions, remarkably as follows.

- Using *Pointer Generator Network* (PGN) for summarizaion [51]. Perhaps PGN is one of the most recent techniques for improving *seq2seq* model to handle the *out-of-vocabulary* problem. The combination of Transformer/BERT and PGN promises much room for exploration in this direction.
- Using *memory network* as a the next generation of deep neural networks [52]. It is another direction to manipulate the hidden memory of recurrent networks more effectively. One can consider it as counterpart of *push-down automata* in the ANN world.
- Developing DL-based models such as *Variational AutoEncoder* (VAE) for topic modeling [53]. Up to now, LDA-based methods have still been considered the most appropriate approach for the problem of topic modeling. However, traditional techniques of this approach suffers from the scalability problem when dealing with very large training corpus. Making use of ANN-based architecture to simulate LDA process is considered as a good choice to overcome such problem.
- Developing multi-modal architecture for NLP such as *Visual Question Answering* systems [54]. This direction envisions an interesting combination of multimedia data processing, such as image and text. The attention/self-attention techniques are then evolved as *cross-attention* or *co-attention* once handling such a task, producing much inspiring results at the moment.

Acknowledgments

Foremost, the author would like to express his great appreciation to his students, who worked tirelessly to contribute in material preparation for this paper. The names of contributors are alphabetically listed as Nguyen Minh Dang, Nguyen Quang Duc, Tran Duy Khanh, Le Minh Khoi, Bui Ngo Hoang Long, Bui Le Ngoc Min, Pham Quoc Nguyen, Nguyen Thanh Thong, Nguyen Vo Thuy Trang, Lu Ngoc Thien Truc and Bui Cong Tuan.

References

[1] G. Guida and G. Mauri, "Evaluation of natural language processing systems: Issues and approaches," *Proceedings of the Institute of Electrical and Electronics Engineers (IEEE)*, vol. 74, pp. 1026–1035, 1986.

[2] S. J. Russell and P. Norvig, *Artificial intelligence - a modern approach, 2nd Edition*. Prentice Hall, 2003.

[3] D. Soergel, "Organizing information - principles of data base and retrieval systems." Academic Press, 1985.

[4] P. J. Stone, D. C. Dunphy, and M. S. Smith, "The general inquirer: A computer approach to content analysis," *American Educational Research Journal*, vol. 4, p. 397, 1967.

[5] E. Marsh and D. Perzanowski, "Muc-7 evaluation of ie technology: Overview of results," in *Seventh Message Understanding Conference (MUC-7): Proceedings of a Conference Held in Fairfax, Virginia, April 29 - May 1, 1998*, 1998.

[6] D. M. Blei, "Probabilistic topic models," *Communications of the Association for Computing Machinery (ACM)*, vol. 55, no. 4, pp. 77–84, 2012.

[7] M. W. Madsen, "The limits of machine translation," 2009.

[8] L. Bradeško and D. Mladenić, "A survey of chabot systems through a loebner prize competition," *Proceedings of Slovenian language technologies society eighth conference of language technologies*, pp. 34–37, 2012.

[9] I. Mani and M. T. Maybury, "Advances in automatic text summarization," *Computational Linguistics*, vol. 26, pp. 280–281, 1999.

[10] Z. S. Harris, "Distributional structure," *WORD*, vol. 10, no. 2-3, pp. 146–162, 1954.

[11] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information Processing and Management*, vol. 24, no. 5, pp. 513–523, 1988.

[12] J. Leskovec, A. Rajaraman, and J. D. Ullman, "Mining of massive datasets, 2nd ed." Stanford University, 2014.

[13] M. A. Hearst, "Trends & controversies: Support vector machines," *IEEE Intelligent Systems*, vol. 13, pp. 18–28, 1998.

[14] M. A. Kumar and M. Gopal, "An investigation on linear svm and its variants for text categorization," *2010 Second International Conference on Machine Learning and Computing*, pp. 27–31, 2010.

[15] H. Zhang and D. Li, "Naïve bayes text classifier," *2007 IEEE International Conference on Granular Computing (GRC 2007)*, pp. 708–708, 2007.

[16] M. S. Akhtar, A. Kumar, D. Ghosal, A. Ekbal, and P. Bhattacharyya, "A multilayer perceptron based ensemble technique for fine-grained financial sentiment analysis," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2017, pp. 540–546.

[17] D. M. Blei, A. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of Machine Learning Research (JMLR)*, vol. 3, pp. 993–1022, 2003.

[18] H. Schütze, C. D. Manning, and P. Raghavan, "Introduction to information retrieval," in *Cambridge University Press Cambridge*, vol. 39, 2008.

[19] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *the Journal of Machine Learning Research (JMLR)*, vol. 3, pp. 1137–1155, 2003.

[20] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*, pp. 1–12, 2013.

[21] J. Pennington, R. Socher, and C. Manning, "Glove: Global Vectors for Word Representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2014, pp. 1532–1543.

[22] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext.zip: Compressing text classification models," *arXiv preprint arXiv:1612.03651*, 2016.

[23] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018.

[24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *North American Chapter of the Association for Computational Linguis-*

tics (NAACL), 2019, pp. 4171–4186.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[26] Y. A. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, pp. 541–551, 1989.

[27] Y. Kim, "Convolutional Neural Networks for Sentence Classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1746–1751.

[28] C. N. dos Santos and V. Guimarães, "Boosting named entity recognition with neural character embeddings," in *Proceedings of the Fifth Named Entity Workshop*. Beijing, China: Association for Computational Linguistics, jul 2015, pp. 25–33.

[29] M. Labeau, K. Löser, and A. Allauzen, "Non-lexical neural architecture for fine-grained pos tagging," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sep. 2015, pp. 232–237.

[30] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[31] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[32] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013, pp. 8624–8628.

[33] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[34] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning Precise Timing with LSTM Recurrent Networks," *Journal of Machine Learning Research (JMLR)*, vol. 3, no. Aug, pp. 115–143, 2002.

[35] K. Cho, B. van Merrienboer, Çaglar Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734.

[36] J. P. Chiu and E. Nichols, "Named Entity Recognition with Bidirectional LSTM-CNNs," *Transactions of the Association for Computational Linguistics*, vol. 4, pp. 357–370, 2016.

[37] X. Ouyang, P. Zhou, C. H. Li, and L. Liu, "Sentiment Analysis Using Convolutional Neural Network," in *Proceedings of the 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (ICCIT)*, 2015, pp. 2359–2364.

[38] M. K. Balwant, "Bidirectional LSTM based on POS tags and CNN Architecture for Fake News Detection," in *Proceedings of the 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2019, pp. 1–6.

[39] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 3104–3112.

[40] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *The Computing Research Repository (CoRR) in arXiv*, 2015. [Online]. Available: https://arxiv.org/abs/1409.0473

[41] P. A. Gagniuc, *Markov Chains: From Theory to Implementation and Experimentation*. John Wiley & Sons, 2017.

[42] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," vol. 1, no. 8, 2019, p. 9.

[43] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 328–339.

[44] L. Wang, *Support vector machines : theory and applications*. Springer Science & Business Media, 2005, vol. 177.

[45] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[46] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[47] Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 2978–2988.

[48] X. Wang and Z. Zhang. Must-read papers on pre-

trained language models (plms). [Online]. Available: https://github.com/thunlp/PLMpapers

[49] J. Alammar. The illustrated transformer. [Online]. Available: https://jalammar.github.io/illustrated-transformer/

[50] Y. Liu, "Fine-tune BERT for extractive summarization," *The Computing Research Repository (CoRR) in arXiv (CoRR)*, vol. abs/1903.10318, 2019. [Online]. Available: http://arxiv.org/abs/1903.10318

[51] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 1073–1083.

[52] S. Sukhbaatar, A. D. Szlam, J. Weston, and R. Fergus, "End-to-end memory networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, p. 2440–2448.

[53] T. Trinh, T. T. Quan, and T. Mai, "Nested variational autoencoder for topic modeling on microtexts with word vectors," *Expert Systems*, vol. 38, no. 2, p. e12639, 2021.

[54] Y. Srivastava, V. Murali, S. R. Dubey, and S. Mukherjee, "Visual question answering using deep learning: A survey and performance analysis," in *International Conference on Computer Vision and Image Processing*. Springer, 2020, pp. 75–86.