**Recent Trends in Deep Learning Based Natural Language Processing**

**Abstract**

Deep learning techniques utilize multiple processing layers to comprehend hierarchical representations of data. In the context of natural language processing (NLP), a variety of model designs and methods have blossomed in recent times. In this paper, we evaluate important deep learning related models and methods that have been assigned for several NLP assignments and deliver the path of their evolution. We also summarize, compare, and contrast the various models and put forward a detailed understanding of the past, present, and future of deep learning in NLP.

Index Terms

Natural Language Processing, Deep Learning, Word2Vec, Attention, Recurrent Neural Networks, Convolutional Neural Net-

works, LSTM, Sentiment Analysis, Question Answering, Dialogue Systems, Parsing, Named-Entity Recognition, POS Tagging,

Semantic Role Labeling

I. INTRODUCTION

Natural language processing (NLP) is a theory-motivated range of computational techniques for the automatic analysis and

representation of human language. NLP research has evolved from the era of punch cards and batch processing, in which the

analysis of a sentence could take up to 7 minutes, to the era of Google and the likes of it, in which millions of webpages can

be processed in less than a second [1]. NLP enables computers to perform a wide range of natural language related tasks at

all levels, ranging from parsing and part-of-speech (POS) tagging, to machine translation and dialogue systems.

Deep learning architectures and algorithms have already made impressive advances in fields such as computer vision and

pattern recognition. Following this trend, recent NLP research is now increasingly focusing on the use of new deep learning

methods (see Figure 1). For decades, machine learning approaches targeting NLP problems have been based on shallow models

(e.g., SVM and logistic regression) trained on very high dimensional and sparse features. In the last few years, neural networks

based on dense vector representations have been producing superior results on various NLP tasks. This trend is sparked by

the success of word embeddings [2, 3] and deep learning methods [4]. Deep learning enables multi-level automatic feature

representation learning. In contrast, traditional machine learning based NLP systems liaise heavily on hand-crafted features.

Such hand-crafted features are time-consuming and often incomplete.

Collobert et al. [5] demonstrated that a simple deep learning framework outperforms most state-of-the-art approaches in

several NLP tasks such as named-entity recognition (NER), semantic role labeling (SRL), and POS tagging. Since then,

numerous complex deep learning based algorithms have been proposed to solve difficult NLP tasks. We review major deep

learning related models and methods applied to natural language tasks such as convolutional neural networks (CNNs), recurrent

neural networks (RNNs), and recursive neural networks. We also discuss memory-augmenting strategies, attention mechanisms

and how unsupervised models, reinforcement learning methods and recently, deep generative models have been employed for

language-related tasks.

To the best of our knowledge, this work is the first of its type to comprehensively cover the most popular deep learning

methods in NLP research today 1

. The work by Goldberg [6] only presented the basic principles for applying neural networks

to NLP in a tutorial manner. We believe this paper will give readers a more comprehensive idea of current practices in this

domain.

The structure of the paper is as follows: Section II introduces the concept of distributed representation, the basis of

sophisticated deep learning models; next, Sections III, IV, and V discuss popular models such as convolutional, recurrent,

and recursive neural networks, as well as their use in various NLP tasks; following, Section VI lists recent applications of

reinforcement learning in NLP and new developments in unsupervised sentence representation learning; later, Section VII

≡ means authors contributed equally

∗ Corresponding author (e-mail: cambria@ntu.edu.sg)

1We intend to update this article with time as and when significant advances are proposed and used by the community

[8:04 AM, 4/15/2022] Sha\_lini: illustrates the recent trend of coupling deep learning models with memory modules; finally, Section VIII summarizes the

performance of a series of deep learning methods on standard datasets about major NLP topics.

II. DISTRIBUTED REPRESENTATION

Statistical NLP has emerged as the primary option for modeling complex natural language tasks. However, in its beginning,

it often used to suffer from the notorious curse of dimensionality while learning joint probability functions of language models.

This led to the motivation of learning distributed representations of words existing in low-dimensional space [7].

A. Word Embeddings

Distributional vectors or word embeddings (Fig. 2) essentially follow the distributional hypothesis, according to which words

with similar meanings tend to occur in similar context. Thus, these vectors try to capture the characteristics of the neighbors of a

word. The main advantage of distributional vectors is that they capture similarity between words. Measuring similarity between

vectors is possible, using measures such as cosine similarity. Word embeddings are often used as the first data processing layer

in a deep learning model. Typically, word embeddings are pre-trained by optimizing an auxiliary objective in a large unlabeled

corpus, such as predicting a word based on its context [8, 3], where the learned word vectors can capture general syntactical

and semantic information. Thus, these embeddings have proven to be efficient in capturing context similarity, analogies and

due to its smaller dimensionality, are fast and efficient in processing core NLP tasks.

Over the years, the models that create such embeddings have been shallow neural networks and there has not been need

for deep networks to create good embeddings. However, deep learning based NLP models invariably represent their words,

phrases and even sentences using these embeddings. This is in fact a major difference between traditional word count based

models and deep learning based models. Word embeddings have been responsible for state-of-the-art results in a wide range

of NLP tasks [9, 10, 11, 12].

For example, Glorot et al. [13] used embeddings along with stacked denoising autoencoders for domain adaptation in senti-

ment classification and Hermann and Blunsom [14] presented combinatory categorial autoencoders to learn the compositionality

of sentence. Their wide usage across the recent literature shows their effectiveness and importance in any deep learning model

performing a NLP task.

Distributed representations (embeddings) are mainly learned through context. During 1990s, several research develop-

ments [15] marked the foundations of research in distributional semantics. A more detailed summary of these early trends is

King

(-) Man

(+) Woman

Queen

Fig. 2: Distributional vectors represented by a D-dimensional vector where D << V, where V is size of Vocabulary. Figure

[8:04 AM, 4/15/2022] Sha\_lini: provided in [16, 17]. Later developments were adaptations of these early works, which led to creation of topic models like

latent Dirichlet allocation [18] and language models [7]. These works laid out the foundations of representation learning in

natural language.

In 2003, Bengio et al. [7] proposed a neural language model which learned distributed representations for words (Fig. 3).

Authors argued that these word representations, once compiled into sentence representations using joint probability of word

sequences, achieved an exponential number of semantically neighboring sentences. This, in turn, helped in generalization

since unseen sentences could now gather higher confidence if word sequences with similar words (in respect to nearby word

representation) were already seen.

Collobert and Weston [19] were the first work to show the utility of pre-trained word embeddings. They proposed a neural

network architecture that forms the foundation to many current approaches. The work also establishes word embeddings as

a useful tool for NLP tasks. However, the immense popularization of word embeddings was arguably due to Mikolov et al.

[3] who proposed the continuous bag-of-words (CBOW) and skip-gram models to efficiently construct high-quality distributed

vector representations. Propelling their popularity was the unexpected side effect of the vectors exhibiting compositionality, i.e.,

adding two word vectors results in a vector that is a semantic composite of the individual words, e.g., ‘man’ + ‘royal’ = ‘king’.

The theoretical justification for this behavior was recently given by Gittens et al. [20], which stated that compositionality is

seen only when certain assumptions are held, e.g., the assumption that words need to be uniformly distributed in the embedding

space.

Glove by Pennington et al. [21] is another famous word embedding method which is essentially a “count-based” model.

Here, the word co-occurrence count matrix is pre-processed by normalizing the counts and log-smoothing operation. This

matrix is then factorized to get lower dimensional representations which is done by minimizing a “reconstruction loss”.

Below, we provide a brief description of the word2vec method proposed by Mikolov et al. [3].

B. Word2vec

Word embeddings were revolutionized by Mikolov et al. [8, 3] who proposed the CBOW and skip-gram models. CBOW

computes the conditional probability of a target word given the context words surrounding it across a window of size k. On

the other hand, the skip-gram model does the exact opposite of the CBOW model, by predicting the surrounding context words

given the central target word. The context words are assumed to be located symmetrically to the target words within a distance

equal to the window size in both directions. In unsupervised settings, the word embedding dimension is determined by the

accuracy of prediction. As the embedding dimension increases, the accuracy of prediction also increases until it converges at

some point, which is considered the optimal embedding dimension as it is the shortest without compromising accuracy.

Let us consider a simplified version of the CBOW model where only one word is considered in the context. This essentially

replicates a bigram language model.

As shown in Fig. 4, the CBOW model is a simple fully connected neural network with one hidden layer. The input layer,

which takes the one-hot vector of context word has V neurons while the hidden layer has N neurons. The output layer is softmax

probability over all words in the vocabulary. The layers are connected by weight matrix W ∈ RV ×N and W

0

∈ RH×V

,

[8:04 AM, 4/15/2022] Sha\_lini: respectively. Each word from the vocabulary is finally represented as two learned vectors vc and vw, corresponding to context

and target word representations, respectively. Thus, k

th word in the vocabulary will have

vc = W(k,.) and vw = W

0

(.,k)

(1)

Overall, for any word wi with given context word c as input,

p

wi

c

= yi =

e

ui

PV

i=1 e

ui

where, ui = v

T

wi

.vc (2)

The parameters θ = {vw, vc}w,c ∈ Vocab are learned by defining the objective function as the log-likelihood and finding its

gradient as

l(θ) = X

w∈Vocab

log

p

w

c

(3)

∂l(θ)

∂vw

= vc

1 − p

w

c

(4)

In the general CBOW model, all the one-hot vectors of context words are taken as input simultaneously, i.e,

h = WT(x1 + x2 + ... + xc) (5)

One limitation of individual word embeddings is their inability to represent phrases [3], where the combination of two or

more words – e.g., idioms like “hot potato” or named entities such as “Boston Globe’ – does not represent the combination

of meanings of individual words. One solution to this problem, as explored by Mikolov et al. [3], is to identify such phrases

based on word co-occurrence and train embeddings for them separately. Later methods have explored directly learning n-gram

embeddings from unlabeled data [23].

Another limitation comes from learning embeddings based only on a small window of surrounding words, sometimes

words such as good and bad share almost the same embedding [24], which is problematic if used in tasks such as sentiment

analysis [25]. At times these embeddings cluster semantically-similar words which have opposing sentiment polarities. This

leads the downstream model used for the sentiment analysis task to be unable to identify this contrasting polarities leading to

poor performance. Tang et al. [26] addressed this problem by proposing sentiment specific word embedding (SSWE). Authors

incorporated the supervised sentiment polarity of text in their loss functions while learning the embeddings.

A general caveat for word embeddings is that they are highly dependent on the applications in which it is used. Labutov

and Lipson [27] proposed task specific embeddings which retrain the word embeddings to align them in the current task space.

This is very important as training embeddings from scratch requires large amount of time and resource. Mikolov et al. [8] tried

to address this issue by proposing negative sampling which does frequency-based sampling of negative terms while training

the word2vec model.

Traditional word embedding algorithms assign a distinct vector to each word. This makes them unable to account for

polysemy. In a recent work, Upadhyay et al. [28] provided an innovative way to address this deficit. The authors leveraged

multilingual parallel data to learn multi-sense word embeddings. For example, the English word bank, when translated to

French provides two different words: banc and banque representing financial and geographical meanings, respectively. Such

multilingual distributional information helped them in accounting for polysemy.

Table I provides a directory of existing frameworks that are frequently used for creating embeddings which are further

[8:04 AM, 4/15/2022] Sha\_lini: C. Character Embeddings

Word embeddings are able to capture syntactic and semantic information, yet for tasks such as POS-tagging and NER, intra-

word morphological and shape information can also be very useful. Generally speaking, building natural language understanding

systems at the character level has attracted certain research attention [29, 30, 31, 32]. Better results on morphologically rich

languages are reported in certain NLP tasks. Santos and Guimaraes [31] applied character-level representations, along with

word embeddings for NER, achieving state-of-the-art results in Portuguese and Spanish corpora. Kim et al. [29] showed positive

results on building a neural language model using only character embeddings. Ma et al. [33] exploited several embeddings,

including character trigrams, to incorporate prototypical and hierarchical information for learning pre-trained label embeddings

in the context of NER.

A common phenomenon for languages with large vocabularies is the unknown word issue, also known as out-of-vocabulary

(OOV) words. Character embeddings naturally deal with it since each word is considered as no more than a composition

of individual letters. In languages where text is not composed of separated words but individual characters and the semantic

meaning of words map to its compositional characters (such as Chinese), building systems at the character level is a natural

choice to avoid word segmentation [34]. Thus, works employing deep learning applications on such languages tend to prefer

character embeddings over word vectors [35]. For example, Peng et al. [36] proved that radical-level processing could greatly

improve sentiment classification performance. In particular, the authors proposed two types of Chinese radical-based hierarchical

embeddings, which incorporate not only semantics at radical and character level, but also sentiment information. Bojanowski

et al. [37] also tried to improve the representation of words by using character-level information in morphologically-rich

languages. They approached the skip-gram method by representing words as bag-of-character n-grams. Their work thus had

the effectiveness of the skip-gram model along with addressing some persistent issues of word embeddings. The method was

also fast, which allowed training models on large corpora quickly. Popularly known as FastText, such a method stands out over

previous methods in terms of speed, scalability, and effectiveness.

Apart from character embeddings, different approaches have been proposed for OOV handling. Herbelot and Baroni [38]

provided on-the-fly OOV handling by initializing the unknown words as the sum of the context words and refining these

words with a high learning rate. However, their approach is yet to be tested on typical NLP tasks. Pinter et al. [39] provided

an interesting approach of training a character-based model to recreate pre-trained embeddings. This allowed them to learn a

compositional mapping form character to word embedding, thus tackling the OOV problem.

Despite the ever growing popularity of distributional vectors, recent discussions on their relevance in the long run have

cropped up. For example, Lucy and Gauthier [40] has recently tried to evaluate how well the word vectors capture the

necessary facets of conceptual meaning. The authors have discovered severe limitations in perceptual understanding of the

concepts behind the words, which cannot be inferred from distributional semantics alone. A possible direction for mitigating

these deficiencies will be grounded learning, which has been gaining popularity in this research domain.

D. Contextualized Word Embeddings

The quality of word representations is generally gauged by its ability to encode syntactical information and handle polysemic

behavior (or word senses). These properties result in improved semantic word representations. Recent approaches in this area

encode such information into its embeddings by leveraging the context. These methods provide deeper networks that calculate

word representations as a function of its context.

Traditional word embedding methods such as Word2Vec and Glove consider all the sentences where a word is present in

order to create a global vector representation of that word. However, a word can have completely different senses or meanings

in the contexts. For example, lets consider these two sentences - 1) “The bank will not be accepting cash on Saturdays” 2)

“The river overflowed the bank.”. The word senses of bank are different in these two sentences depending on its context.

Reasonably, one might want two different vector representations of the word bank based on its two different word senses.

The new class of models adopt this reasoning by diverging from the concept of global word representations and proposing

contextual word embeddings instead.

Embedding from Language Model (ELMo) [41] is one such method that provides deep contextual embeddings. ELMo

produces word embeddings for each context where the word is used, thus allowing different representations for varying senses

[8:04 AM, 4/15/2022] Sha\_lini: of the same word. Specifically, for N different sentences where a word w is present, ELMo generates N different representations

of w i.e., w1, w2, ˙,wN .

The mechanism of ELMo is based on the representation obtained from a bidirectional language model. A bidirectional

language model (biLM) constitutes of two language models (LM) 1) forward LM and 2) backward LM. A forward LM takes

input representation x

LM

k

for each of the k

th token and passes it through L layers of forward LSTM to get representations

−→h

LM

k,j where j = 1, . . . , L. Each of these representations, being hidden representations of recurrent neural networks, is context

dependent. A forward LM can be seen as a method to model the joint probability of a sequence of tokens: p (t1, t2, . . . , tN ) =

QN

k=1 p (tk|t1, t2, . . . , tk−1). At a timestep k−1 the forward LM predicts the next token tk given the previous observed tokens

t1, t2, ..., tk. This is typically achieved by placing a softmax layer on top of the final LSTM in a forward LM. On the other

hand, a backward LM models the same joint probability of the sequence by predicting the previous token given the future

tokens: p (t1, t2, . . . , tN ) = QN

k=1 p (tk|tk+1, tk+2, . . . , tN ). In other words, a backward LM is similar to forward LM which

processes a sequence with the order being reversed. The training of the biLM model involves modeling the log-likelihood of

both the sentence orientations. Finally, hidden representations from both LMs are concetenated to compose the final token

vectors [42].

For each tokem, ELMo extracts the intermediate layer representations from the biLM and performs a linear combination

based on the given downstream task. A L-layer biLM contains 2L + 1 set of representations as shown below -

Rk =

n

x

LM

k

,

−→h

LM

k,j ,

←−

h

LM

k,j |j = 1, . . . , Lo

=

h

LM

k,j |j = 0, . . . , L

(6)

Here, h

LM

k,0

is the token representation at the lowest level. One can use either character or word embeddings to initialize

h

LM

k,0

. For other values of j,

h

LM

k,j =

h−→h

LM

k,j ,

←−

h

LM

k,j i

∀j = 1, . . . , L. (7)

ELMo flattens all layers in R in a single vector such that -

ELMotask

k = E

[8:04 AM, 4/15/2022] Sha\_lini: Fig. 5: CNN framework used to perform word wise class prediction (Figure source: Collobert and Weston [19])

The use of CNNs for sentence modeling traces back to Collobert and Weston [19]. This work used multi-task learning to

output multiple predictions for NLP tasks such as POS tags, chunks, named-entity tags, semantic roles, semantically-similar

words and a language model. A look-up table was used to transform each word into a vector of user-defined dimensions.

Thus, an input sequence {s1, s2, ...sn} of n words was transformed into a series of vectors {ws1

, ws2

, ...wsn

} by applying

the look-up table to each of its words (Fig. 5).

This can be thought of as a primitive word embedding method whose weights were learned in the training of the network.

In [5], Collobert extended his work to propose a general CNN-based framework to solve a plethora of NLP tasks. Both these

works triggered a huge popularization of CNNs amongst NLP researchers. Given that CNNs had already shown their mettle

for computer vision tasks, it was easier for people to believe in their performance.

CNNs have the ability to extract salient n-gram features from the input sentence to create an informative latent semantic

representation of the sentence for downstream tasks. This application was pioneered by Collobert et al. [5], Kalchbrenner et al.

[49], Kim [50], which led to a huge proliferation of CNN-based networks in the succeeding literature. Below, we describe the

working of a simple CNN-based sentence modeling network:

A. Basic CNN

1) Sentence Modeling: For each sentence, let wi ∈ Rd

represent the word embedding for the i

th word in the sentence,

where d is the dimension of the word embedding. Given that a sentence has n words, the sentence can now be represented as

an embedding matrix W ∈ Rn×d

. Fig. 6 depicts such a sentence as an input to the CNN framework.

Let wi:i+j refer to the concatenation of vectors wi

, wi+1, ...wj

. Convolution is performed on this input embedding layer.

It involves a filter k ∈ Rhd which is applied to a window of h words to produce a new feature. For example, a feature ci

is

generated using the window of words wi:i+h−1 by

ci = f(wi:i+h−1.k

T + b) (9)

Here, b ∈ R is the bias term and f is a non-linear activation function, for example the hyperbolic tangent. The filter k is

applied to all possible windows using the same weights to create the feature map.

c = [c1, c2, ..., cn−h+1] (10)

In a CNN, a number of convolutional filters, also called kernels (typically hundreds), of different widths slide over the

entire word embedding matrix. Each kernel extracts a specific pattern of n-gram. A convolution layer is usually followed by

a max-pooling strategy, cˆ = max{c}, which subsamples the input typically by applying a max operation on each filter. This

strategy has two primary reasons.

[8:04 AM, 4/15/2022] Sha\_lini: Firstly, max pooling provides a fixed-length output which is generally required for classification. Thus, regardless the size of

the filters, max pooling always maps the input to a fixed dimension of outputs. Secondly, it reduces the output’s dimensionality

while keeping the most salient n-gram features across the whole sentence. This is done in a translation invariant manner where

each filter is now able to extract a particular feature (e.g., negations) from anywhere in the sentence and add it to the final

sentence representation.

The word embeddings can be initialized randomly or pre-trained on a large unlabeled corpora (as in Section II). The

latter option is sometimes found beneficial to performance, especially when the amount of labeled data is limited [50]. This

combination of convolution layer followed by max pooling is often stacked to create deep CNN networks. These sequential

convolutions help in improved mining of the sentence to grasp a truly abstract representations comprising rich semantic

information. The kernels through deeper convolutions cover a larger part of the sentence until finally covering it fully and

creating a global summarization of the sentence features.

2) Window Approach: The above-mentioned architecture allows for modeling of complete sentences into sentence repre-

sentations. However, many NLP tasks, such as NER, POS tagging, and SRL, require word-based predictions. To adapt CNNs

for such tasks, a window approach is used, which assumes that the tag of a word primarily depends on its neighboring words.

For each word, thus, a fixed-size window surrounding itself is assumed and the sub-sentence ranging within the window is

considered. A standalone CNN is applied to this sub-sentence as explained earlier and predictions are attributed to the word

in the center of the window. Following this approach, Poria et al. [52] employed a multi-level deep CNN to tag each word in

a sentence as a possible aspect or non-aspect. Coupled with a set of linguistic patterns, their ensemble classifier managed to

perform well in aspect detection.

The ultimate goal of word-level classification is generally to assign a sequence of labels to the entire sentence. In such cases,

structured prediction techniques such as conditional random field (CRF) are sometimes employed to better capture dependencies

between adjacent class labels and finally generate cohesive label sequence giving maximum score to the whole sentence [53].

To get a larger contextual range, the classic window approach is often coupled with a time-delay neural network (TDNN) [54].

Here, convolutions are performed across all windows throughout the sequence. These convolutions are generally constrained

by defining a kernel having a certain width. Thus, while the classic window approach only considers the words in the window

around the word to be labeled, TDNN considers all windows of words in the sentence at the same time. At times, TDNN

layers are also stacked like CNN architectures to extract local features in lower layers and global features in higher layers [5].

B. Applications

In this section, we present some of the crucial works that employed CNNs on NLP tasks to set state-of-the-art benchmarks

in their respective times.

Kim [50] explored using the above architecture for a variety of sentence classification tasks, including sentiment, subjectivity

and question type classification, showing competitive results. This work was quickly adapted by researchers given its simple

yet effective network. After training for a specific task, the randomly initialized convolutional kernels became specific n-gram

feature detectors that were useful for that target task (Fig. 7). This simple network, however, had many shortcomings with the

[8:04 AM, 4/15/2022] Sha\_lini: Fig. 7: Top 7-grams by four learned 7-gram kernels; each kernel is sensitive to a specific kind of 7-gram (Figure

Source: Kalchbrenner et al. [49])

This issue was partly handled by Kalchbrenner et al. [49], who published a prominent paper where they proposed a dynamic

convolutional neural network (DCNN) for semantic modeling of sentences. They proposed dynamic k-max pooling strategy

which, given a sequence p selects the k most active features. The selection preserved the order of the features but was insensitive

to their specific positions (Fig. 8). Built on the concept of TDNN, they added this dynamic k-max pooling strategy to create

a sentence model. This combination allowed filters with small width to span across a long range within the input sentence,

thus accumulating crucial information across the sentence. In the induced subgraph (Fig. 8), higher order features had highly

variable ranges that could be either short and focused or global and long as the input sentence. They applied their model on

multiple tasks, including sentiment prediction and question type classification, achieving significant results. Overall, this work

commented on the range of individual kernels while trying to model contextual semantics and proposed a way to extend their

reach.

Tasks involving sentiment analysis also require effective extraction of aspects along with their sentiment polarities [55].

Ruder et al. [56] applied a CNN where in the input they concatenated an aspect vector with the word embeddings to get

competitive results. CNN modeling approach varies amongst different length of texts. Such differences were seen in many

works like Johnson and Zhang [23], where performance on longer text worked well as opposed to shorter texts. Wang et al.

[57] proposed the usage of CNN for modeling representations of short texts, which suffer from the lack of available context

and, thus, require extra efforts to create meaningful representations. The authors proposed semantic clustering which introduced

multi-scale semantic units to be used as external knowledge for the short texts. CNN was used to combine these units and

form the overall representation. In fact, this requirement of high context information can be thought of as a caveat for CNN-

based models. NLP tasks involving microtexts using CNN-based methods often require the need of additional information and

external knowledge to perform as per expectations. This fact was also observed in [58], where authors performed sarcasm

detection in Twitter texts using a CNN network. Auxiliary support, in the form of pre-trained networks trained on emotion,

sentiment and personality datasets was used to achieve state-of-the-art performance.

CNNs have also been extensively used in other tasks. For example, Denil et al. [59] applied DCNN to map meanings of

words that constitute a sentence to that of documents for summarization. The DCNN learned convolution filters at both the

sentence and document level, hierarchically learning to capture and compose low-level lexical features into high-level semantic

concepts. The focal point of this work was the introduction of a novel visualization technique of the learned representations,

which provided insights not only in the learning process but also for automatic summarization of texts.

CNN models are also suitable for certain NLP tasks that require semantic matching beyond classification [60]. A similar

model to the above CNN architecture (Fig. 6) was explored in [61] for information retrieval. The CNN was used for projecting

queries and documents to a fixed-dimension semantic space, where cosine similarity between the query and documents was

used for ranking documents regarding a specific query. The model attempted to extract rich contextual structures in a query

or a document by considering a temporal context window in a word sequence. This captured the contextual features at the

word n-gram level. The salient word n-grams is then discovered by the convolution and max-pooling layers which are then

aggregated to form the overall sentence vector.

In the domain of QA, Yih et al. [62] proposed to measure the semantic similarity between a question and entries in a

knowledge base (KB) to determine what supporting fact in the KB to look for when answering a question. To create semantic

representations, a CNN similar to the one in Fig. 6 was used. Unlike the classification setting, the supervision signal came

from positive or negative text pairs (e.g., query-document), instead of class labels. Subsequently, Dong et al. [63] introduced

a multi-column CNN (MCCNN) to analyze and understand questions from multiple aspects and create their representations.

MCCNN used multiple column networks to extract information from aspects comprising answer types and context from the

input questions. By representing entities and relations in the KB with low-dimensional vectors, they used question-answer

pairs to train the CNN model so as to rank candidate answers. Severyn and Moschitti [64] also used CNN network to model

optimal representations of question and answer sentences. They proposed additional features in the embeddings in the form

of relational information given by matching words between the question and answer pair. These parameters were tuned by the

[8:04 AM, 4/15/2022] Sha\_lini: CNNs are wired in a way to capture the most important information in a sentence. Traditional max-pooling strategies

perform this in a translation invariant form. However, this often misses valuable information present in multiple facts within

the sentence. To overcome this loss of information for multiple-event modeling, Chen et al. [65] proposed a modified pooling

strategy: dynamic multi-pooling CNN (DMCNN). This strategy used a novel dynamic multi-pooling layer that, as the name

suggests, incorporates event triggers and arguments to reserve more crucial information from the pooling layer.

CNNs inherently provide certain required features like local connectivity, weight sharing, and pooling. This puts forward

some degree of invariance which is highly desired in many tasks. Speech recognition also requires such invariance and, thus,

Abdel-Hamid et al. [66] used a hybrid CNN-HMM model which provided invariance to frequency shifts along the frequency

axis. This variability is often found in speech signals due to speaker differences. They also performed limited weight sharing

which led to a smaller number of pooling parameters, resulting in lower computational complexity. Palaz et al. [67] performed

extensive analysis of CNN-based speech recognition systems when given raw speech as input. They showed the ability of

CNNs to directly model the relationship between raw input and phones, creating a robust automatic speech recognition system.

Tasks like machine translation require perseverance of sequential information and long-term dependency. Thus, structurally

they are not well suited for CNN networks, which lack these features. Nevertheless, Tu et al. [68] addressed this task by

considering both the semantic similarity of the translation pair and their respective contexts. Although this method did not

address the sequence perseverance problem, it allowed them to get competitive results amongst other benchmarks.

Overall, CNNs are extremely effective in mining semantic clues in contextual windows. However, they are very data heavy

models. They include a large number of trainable parameters which require huge training data. This poses a problem when

scarcity of data arises. Another persistent issue with CNNs is their inability to model long-distance contextual information and

preserving sequential order in their representations [49, 68]. Other networks like recursive models (explained below) reveal

themselves as better suited for such learning.

IV. RECURRENT NEURAL NETWORKS

RNNs [69] use the idea of processing sequential information. The term “recurrent” applies as they perform the same task

over each instance of the sequence such that the output is dependent on the previous computations and results. Generally, a

fixed-size vector is produced to represent a sequence by feeding tokens one by one to a recurrent unit. In a way, RNNs have

“memory” over previous computations and use this information in current processing. This template is naturally suited for

many NLP tasks such as language modeling [2, 70, 71], machine translation [72, 73, 74], speech recognition [75, 76, 77, 78],

image captioning [79]. This made RNNs increasingly popular for NLP applications in recent years.

A. Need for Recurrent Networks

In this section, we analyze the fundamental properties that favored the popularization of RNNs in a multitude of NLP tasks.

Given that an RNN performs sequential processing by modeling units in sequence, it has the ability to capture the inherent

sequential nature present in language, where units are characters, words or even sentences. Words in a language develop their

semantical meaning based on the previous words in the sentence. A simple example stating this would be the difference in

meaning between “dog” and “hot dog”. RNNs are tailor-made for modeling such context dependencies in language and similar

sequence modeling tasks, which resulted to be a strong motivation for researchers to use RNNs over CNNs in these areas.

[8:04 AM, 4/15/2022] Sha\_lini: Another factor aiding RNN’s suitability for sequence modeling tasks lies in its ability to model variable length of text,

including very long sentences, paragraphs and even documents [80]. Unlike CNNs, RNNs have flexible computational steps

that provide better modeling capability and create the possibility to capture unbounded context. This ability to handle input of

arbitrary length became one of the selling points of major works using RNNs [81].

Many NLP tasks require semantic modeling over the whole sentence. This involves creating a gist of the sentence in

a fixed dimensional hyperspace. RNN’s ability to summarize sentences led to their increased usage for tasks like machine

translation [82] where the whole sentence is summarized to a fixed vector and then mapped back to the variable-length target

sequence.

RNN also provides the network support to perform time distributed joint processing. Most of the sequence labeling

tasks like POS tagging [32] come under this domain. More specific use cases include applications such as multi-label text

categorization [83], multimodal sentiment analysis [84, 85, 86], and subjectivity detection [87].

The above points enlist some of the focal reasons that motivated researchers to opt for RNNs. However, it would be gravely

wrong to make conclusions on the superiority of RNNs over other deep networks. Recently, several works provided contrasting

evidence on the superiority of CNNs over RNNs. Even in RNN-suited tasks like language modeling, CNNs achieved competitive

performance over RNNs [88]. Both CNNs and RNNs have different objectives when modeling a sentence. While RNNs try

to create a composition of an arbitrarily long sentence along with unbounded context, CNNs try to extract the most important

n-grams. Although they prove an effective way to capture n-gram features, which is approximately sufficient in certain sentence

classification tasks, their sensitivity to word order is restricted locally and long-term dependencies are typically ignored.

Yin et al. [89] provided interesting insights on the comparative performance between RNNs and CNNs. After testing on

multiple NLP tasks that included sentiment classification, QA, and POS tagging, they concluded that there is no clear winner:

the performance of each network depends on the global semantics required by the task itself.

Below, we discuss some of the RNN models extensively used in the literature.

B. RNN models

1) Simple RNN: In the context of NLP, RNNs are primarily based on Elman network [69] and they are originally three-

layer networks. Fig. 9 illustrates a more general RNN which is unfolded across time to accommodate a whole sequence. In

the figure, xt is taken as the input to the network at time step t and st represents the hidden state at the same time step.

Calculation of st is based as per the equation:

st = f(Uxt + Wst−1) (11)

Thus, st is calculated based on the current input and the previous time step’s hidden state. The function f is taken to be a

non-linear transformation such as tanh, ReLU and U, V, W account for weights that are shared across time. In the context of

NLP, xt typically comprises of one-hot encodings or embeddings. At times, they can also be abstract representations of textual

content. ot illustrates the output of the network which is also often subjected to non-linearity, especially when the network

contains further layers downstream.

The hidden state of the RNN is typically considered to be its most crucial element. As stated before, it can be considered

as the network’s memory element that accumulates information from other time steps. In practice, however, these simple RNN

networks suffer from the infamous vanishing gradient problem, which makes it really hard to learn and tune the parameters

of the earlier layers in the network.

This limitation was overcome by various networks such as long short-term memory (LSTM), gated recurrent units (GRUs),

and residual networks (ResNets), where the first two are the most used RNN variants in NLP applications.

[8:04 AM, 4/15/2022] Sha\_lini: 2) Long Short-Term Memory: LSTM [91, 92] (Fig. 10) has additional “forget” gates over the simple RNN. Its unique

mechanism enables it to overcome both the vanishing and exploding gradient problem.

Unlike the vanilla RNN, LSTM allows the error to back-propagate through unlimited number of time steps. Consisting of

three gates: input, forget and output gates, it calculates the hidden state by taking a combination of these three gates as per

the equations below:

x =

ht−1

xt

(12)

ft = σ(Wf .x + bf ) (13)

it = σ(Wi

.x + bi) (14)

ot = σ(Wo.x + bo) (15)

ct = ft ct−1 + it tanh(Wc.X + bc) (16)

ht = ot tanh(ct) (17)

3) Gated Recurrent Units: Another gated RNN variant called GRU [82] (Fig. 10) of lesser complexity was invented with

empirically similar performances to LSTM in most tasks. GRU comprises of two gates, reset gate and update gate, and handles

the flow of information like an LSTM sans a memory unit. Thus, it exposes the whole hidden content without any control.

Being less complex, GRU can be a more efficient RNN than LSTM. The working of GRU is as follows:

z = σ(Uz.xt + Wz.ht−1) (18)

r = σ(Ur.xt + Wr.ht−1) (19)

st = tanh(Uz.xt + Ws.(ht−1 r)) (20)

ht = (1 − z) st + z ht−1 (21)

Researchers often face the dilemma of choosing the appropriate gated RNN. This also extends to developers working in

NLP. Throughout the history, most of the choices over the RNN variant tended to be heuristic. Chung et al. [81] did a critical

comparative evaluation of the three RNN variants mentioned above, although not on NLP tasks. They evaluated their work on

tasks relating to polyphonic music modeling and speech signal modeling. Their evaluation clearly demonstrated the superiority

of the gated units (LSTM and GRU) over the traditional simple RNN (in their case, using tanh activation) (Fig. 11). However,

they could not make any concrete conclusion about which of the two gating units was better. This fact has been noted in other

works too and, thus, people often leverage on other factors like computing power while choosing between the two.

C. Applications

1) RNN for word-level classification: RNNs have had a huge presence in the field of word-level classification. Many of

their applications stand as state of the art in their respective tasks. Lample et al. [93] proposed to use bidirectional LSTM

[8:04 AM, 4/15/2022] Sha\_lini: Fig. 11: Learning curves for training and validation sets of different types of units with respect to (top) the number of iterations

and (bottom) the wall clock time. y-axis corresponds to the negative log likelihood of the model shown in log-scale (Figure

source: Chung et al. [81])

for NER. The network captured arbitrarily long context information around the target word (curbing the limitation of a fixed

window size) resulting in two fixed-size vector, on top of which another fully-connected layer was built. They used a CRF

layer at last for the final entity tagging.

RNNs have also shown considerable improvement in language modeling over traditional methods based on count statistics.

Pioneering work in this field was done by Graves [94], who introduced the effectiveness of RNNs in modeling complex

sequences with long range context structures. He also proposed deep RNNs where multiple layers of hidden states were used

to enhance the modeling. This work established the usage of RNNs on tasks beyond the context of NLP. Later, Sundermeyer

et al. [95] compared the gain obtained by replacing a feed-forward neural network with an RNN when conditioning the

prediction of a word on the words ahead. In their work, they proposed a typical hierarchy in neural network architectures

where feed-forward neural networks gave considerable improvement over traditional count-based language models, which in

turn were superseded by RNNs and later by LSTMs. An important point that they mentioned was the applicability of their

conclusions to a variety of other tasks such as statistical machine translation [96].

2) RNN for sentence-level classification: Wang et al. [25] proposed encoding entire tweets with LSTM, whose hidden

state is used for predicting sentiment polarity. This simple strategy proved competitive to the more complex DCNN structure

by Kalchbrenner et al. [49] designed to endow CNN models with ability to capture long-term dependencies. In a special case

studying negation phrase, the authors also showed that the dynamics of LSTM gates can capture the reversal effect of the word

not.

Similar to CNN, the hidden state of an RNN can also be used for semantic matching between texts. In dialogue systems,

Lowe et al. [97] proposed to match a message with candidate responses with Dual-LSTM, which encodes both as fixed-size

vectors and then measure their inner product as the basis to rank candidate responses.

3) RNN for generating language: A challenging task in NLP is generating natural language, which is another natural

application of RNNs. Conditioned on textual or visual data, deep LSTMs have been shown to generate reasonable task-specific

text in tasks such as machine translation, image captioning, etc. In such cases, the RNN is termed a decoder.

In [74], the authors proposed a general deep LSTM encoder-decoder framework that maps a sequence to another sequence.

One LSTM is used to encode the “source” sequence as a fixed-size vector, which can be text in the original language (machine

translation), the question to be answered (QA) or the message to be replied to (dialogue systems). The vector is used as the

initial state of another LSTM, named the decoder. During inference, the decoder generates tokens one by one, while updating

its hidden state with the last generated token. Beam search is often used to approximate the optimal sequence.

Sutskever et al. [74] experimented with 4-layer LSTM on a machine translation task in an end-to-end fashion, showing

competitive results. In [99], the same encoder-decoder framework is employed to model human conversations. When trained

on more than 100 million message-response pairs, the LSTM decoder is able to generate very interesting responses in the open