Chapter 2

Backgrounds and Literature survey

The background essential to understanding the contribution of the thesis is described in this chapter. Section 2.1 briefly discusses low resource languages since they are central to this thesis. The following two sections give an overview of machine learning and deep learning models employed in this thesis, such as Trigrams ‘n’ tags, Maximum entropy Markov model, Conditional random fields, Structured support vector machine and Convolutional neural network, Recurrent neural network and Attention mechanisms in section 2.2 and section 2.3, respectively. The possible way of transfer learning use to low resource languages has been discussed in Section 2.4. After that, the input representation methods for deep learning models to a text has been explored in section 2.5. Section 2.6 describes the importance of handcrafted features for deep learning models. A survey based on earlier sections form the content of section 2.7, focusing on low resource sequence labeling tasks.

2.1 Low Resource Languages

NLP is one of the subfields of artificial intelligence that deals with the challenges of teaching a computer how to interpret human languages. Although NLP has achieved significant success, it still struggles with many aspects, as there are many features of human languages like ambiguity, syntax and semantics, which can be highly complex to comprehend. Most of the success with any language in NLP comes from the availability of resources like annotated data, and NLP techniques require loads of it to yield significant results. Creating annotated data is expensive and time-consuming. It is almost impossible to annotate each and every aspect of any language in the world.

To make a rough estimate, there are more or less seven thousand languages on our planet and only a tiny fraction of them have language data available or even written records. NLP based applications have been primarily built for the most common, widely used, and well-annotated languages like English, Mandarin, Portuguese, etc. Languages with a rich history but poor annotations are falling out of use as we did not have the means to preserve them; it is one of the major challenges in NLP.

Based on the availability of resources, NLP scenarios can categorize as follows:

- Rich-Resource $\rightarrow$ Availability of a large amount of annotated data and parallel corpora
- Low-Resource $\rightarrow$ Lack of annotated data
- No-Resource $\rightarrow$ Unwritten languages

Characterizing languages into 'High' and 'Low' resources still requires much study because of the varying importance of aspects of a language's resources. Sometimes a 'High' resource language might be considered as a 'Low' resource depending on the task to be performed. Therefore a language can be regarded as low-resource for a given task if the algorithms at our disposal cannot use the available ample data to automate the task with sufficient performance. Processing of languages, as mentioned earlier, includes numerous syntactic, semantic and discourse-level tasks (e.g., Word segmentation, Word sense disambiguation, POS tagging, Chunking, Parsing, NER, Semantic parsing, Semantic role labeling, Anaphora resolution). This thesis focuses on POS tagging, Chunking and NER for low resource languages.

2.2 Traditional Learning Techniques in NLP

This section briefly describes the machine learning techniques which have become popular over time, such as Trigrams 'n' tags (which is an extended variation of the Hidden Markov model, using interpolated smoothing), Maximum entropy Markov model, Conditional random fields and Structured support vector machine, have been use for sequence tagging.

2.2.1 Trigrams 'n' Tags

Thorsten Brants's Trigrams 'n' Tags (TnT) uses second-order Markov models for sequence tagging. The Markov model has transition probability which is depending upon the tag, represented by the state. And the output probability depending on the recently assigned category. To be explicit, it is calculated by:

$$\underset{t_1 \ldots t_T}{argmax} \left[\prod_{i=1}^{T} P(t_i|t_{i-1}, t_{i-2}) P(w_i|t_i) \right] P(t_{T+1}|t_T) \tag{2.1}$$

Here, w_1 to w_T is a word sequence and t_1 to t_T are the elements of the tagset, of length T. It is, thus, an extended variation of the Hidden Markov Model (HMM).

2.2.2 Maximum Entropy Markov Model

The HMM is a probabilistic finite-state model and is based on state transition and emission probabilities. The traditional approach inappropriately uses a generative joint model to solve a conditional problem for a given set of observations since it sets the HMM parameters to maximize the likelihood of the observation sequence. Maximum entropy (MaxEnt) works by extracting features from the input, combining them linearly, then using this sum as an exponent. Maximum Entropy Markov Model (MEMM) or Conditional Markov Model (CMM) is a directive and discriminative graphical model used for sequence labeling tasks. MEMM combines the features of both the HMM and MaxEnt models. Unlike the

HMM, the current observation of MEMM [149] may depend on the previous state also. MEMM gives one probability estimate per hidden state, which is the probability of the next tag given the previous tag and the observation.

Given a finite state sequence $S_1,\ldots,S_r$, observation sequence $O_1,\ldots,O_r$, and conditional probability $P(S_1,\ldots,S_r \mid O_1,\ldots,O_r)$:

$$P(S_1,\ldots,S_r \mid O_1,\ldots,O_r) = \prod_{t=1}^{r} P(S_t \mid S_{t-1}, O_t) \tag{2.2}$$

The maximum entropy classifier works as:

$$p(s \mid s', o) = p_{s'}(s \mid o) = \frac{1}{Z(o,s')}.\exp\left(\sum_i \lambda_a f_a(o,s)\right) \tag{2.3}$$

Where $p(s \mid s', o)$ provides the probability of the transition from state s' to the state s on current observation (input) o. $p(s \mid s', o)$ is split into $|S|$ separately trained transition function $p_{s'}(s \mid o) = p(s \mid s', o).f_a(o,s)$ is a real valued feature function, λ_a is the learned parameter, and $Z(o,s')$ is the normalizing factor.

2.2.3 Conditional Random Fields

Conditional Random Fields (CRFs), first described by Lafferty et al., is a conditional probabilistic sequence model that resolves the label bias problem exhibited by MEMMs. In most problems, CRF provides a better tagging performance as compared to MEMMs [124, 204]. CRFs is undirected graphical models, which can be used to define the conditional probability distribution $p(T|W)$, over the label sequence, given the entire input sequence to be labeled:

$$p(T|W) = \frac{1}{Z(W)} \exp\left(\sum_{j=1}^{m}\sum_{i=1}^{n} \lambda_j f_j(t_{i-1}, t_i, i, W)\right) \tag{2.4}$$

Let $w_1, w_2, w_3 \ldots w_n \in W$ be an input sequence and, $t_1, t_2, t_3 \ldots t_n \in T$ a corresponding label sequence, where $Z(W)$ is a normalization factor over the entire state sequences. $f_j(y_{i-1}, y_i, X, i)$ is a feature function at positions i and $i-1$ in the label sequence; λ_j is a learned weight associated with each feature f_j.

2.2.4 Structured Support Vector Machine

First described by Vapnik, the Support Vector Machines (SVMs), is relatively new learning approach for solving binary classification problems. Altun has introduced the Hidden Markov SVM to tackle the dependencies among contextual labels [7]. Structural Support Vector Machines determines the highest score of the tag sequence for given input sequence through linear discrimination function unlike CRF which is getting the highest probability.

$$h_w(x,y) = w^T F(x,y) = \sum_i w_i^T F(x, y_i) = \sum_i w_i^T (f(x_i, y_i), f(x, y_{i-1}, y_i)) \tag{2.5}$$

This hyper-plan has been optimized by:

$$\underset{w,\xi}{min} \frac{1}{2} \|w\|^2 + \frac{C}{m} \sum_{x \in S} \xi_x \tag{2.6}$$

From the S training samples, m is the subset of identically distributed samples with a pair of word x and tag y. Here $f(x_i, y_i)$ and $f(x, y_i, y_{i-1})$ are the feature and a linear Markov Chain, respectively, and C is the trade-off between margin size and training error with the ξ as a slack variable.

However, all these machine learning based sequence labeling problems works the same way for low resource languages as high resource languages. Therefore, the performance of the sequence labeling model becomes restricted at one level.

2.3 Deep Learning Models in NLP

The journey of language processing in machines has gone through symbolic or rule-based to statistical to machine and then to deep learning approaches. Machine learning algorithms use handcrafted features to train earlier defined methods [84]. These methods limit their performance due to sparsity and many other challenges, such as domain expertise required for feature engineering. Over the past few decades, tremendous increase in data availability has caused the re-emergence of artificial neural networks, known as Deep Neural Networks (DNNs). DNNs are able to learn complex features automatically through their representation learning capability. The following section briefly introduces several DNN models that are exploited in this thesis.

2.3.1 Convolutional Neural Network

Convolutional Neural Network (CNN) is a powerful deep learning neural network designed to process structured arrays of data such as images. They are widely used in Computer Vision for image classification, object detection, etc. In addition to Computer Vision, CNNs have also proved to be successful in the field of Natural Language Processing, such as text classification. CNNs use convolutional filters to extract high-level features from adjacent words or n-grams regardless of their position, while taking local ordering patterns into account [84]. The input will be in the form of a matrix representing words or character vectors, which is the key difference for text input. The filters used here will slide over full rows of the input matrix instead of sliding over some local patch.

CNNs have been used for the case of words as well as characters. One popular use of CNNs in NLP is in the task of sentence classification [269], where text is represented as an array of vectors (each word mapped to a specific vector in a vector space composed of the entire vocabulary) that can be processed with the help of a CNN. When working with sequential data, like text, one-dimensional convolutions have been employed, whereas the idea and the application remain the same to the image. we still want to pick up on patterns in the sequence that becomes more complex with each added convolutional layer.

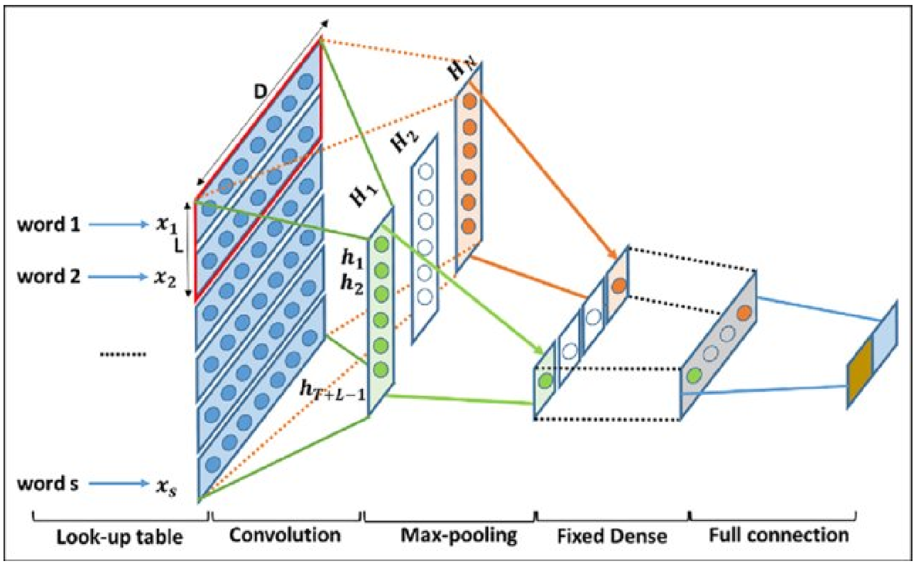

FIGURE 2.1: CNN over the text [171]

Usually, when a bag of words or word2vec (as discussed in the following section) is employed, then they only capture dependencies or structures at the word level and are unable to capture deep patterns/structures/dependencies. To alleviate this problem, CNN is applied at the character level. Moreover, to gain the advantage of both, the latter can be combined with pre-trained word vectors generated through a CNN.

The above Figure 2.1 represents the CNN architecture applied at the character level on the sentence classification task proposed by Zhang et al. [266]. Here, character vectors and padding vectors are represented in a matrix and fed in the input layer. This input matrix is fed to several convolutional, max-pooling and dense layers to obtain the final output. The convolutional layers have 256 filters and fully connected layers have 1024 outputs. The final output dense layer has a number of outputs corresponding to the number of classes in sentence classification. Here, the character level features of our text are utilized instead of simply using the word vectors.

2.3.2 Recurrent Neural Network

Traditional neural networks, including CNN do not have persistence. CNN can capture spatial information, but it cannot capture temporal or sequential information. When used for any task, a traditional neural networks cannot use their reasoning about previous events to make a decision about the later ones. The recurrent units in Recurrent neural networks (RNNs) [206, 72] address this issue. RNNs are network with loops in them, thus allowing information to persist. The fixed-size input vector (x_t) of a token of a sentence passed to the recurrent unit is shown in Figure 2.2. A loop allows information to be passed from one part of the recurrent unit to the next. RNNs can be considered multiple copies of the same recurrent unit, each passing information to their successor, as shown in Figure 2.3. Such a chain-like structure is used to solve various NLP tasks such as Part of Speech Tagging, Named Entity Recognition, Machine Translation, Language Identification and Text classification. The arrangement of this chain-like structure introduces extensions such as stacking two RNNs, where one network is unrolled in a forward direction whereas another network in a backward direction. Based on the assumption that the output at a specific time step solely depends upon the previous and future elements. It is known as Bidirectional recurrent neural networks [217].

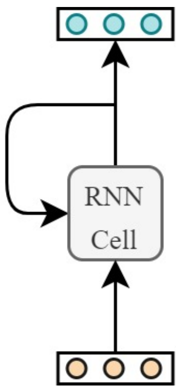

FIGURE 2.2: Folded RNN with feedback loop to a input sequence

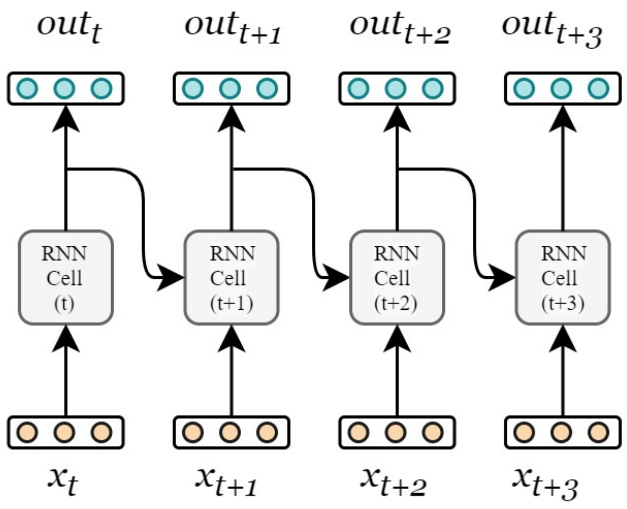

FIGURE 2.3: Unfolded RNN to a input sequence (x) along with its time steps (t)

Theoretically, RNNs are perfectly capable of handling "long-term dependencies" between different parts of sequential data. However, RNNs fail to capture the whole structure and link information when the difference between two parts of the sequential data is significant due to the vanishing gradient problem [97]. It refers to the gradient shrinkage during backpropagation through time. Since the gradient value is small, there is no learning

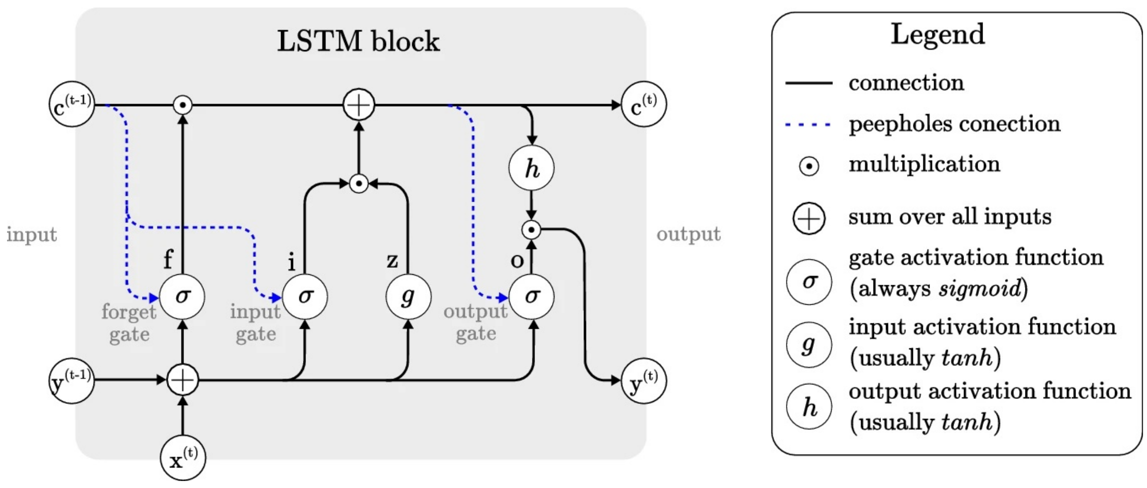

FIGURE 2.4: LSTM block at t^{th} time step [98]

process. Long Short Term Memory networks and Gated Recurrent Unit networks are a special kind of RNN, capable of learning long-term dependencies in sequential data.

Long Short-Term Memory Networks (LSTMs) LSTMs [97, 80] also have this chain-like structure, but the repeating module has a different structure. The LSTMs have a cell state, which is a horizontal line running through the top of the network, as shown in Figure 2.4. The information flows along this cell state. The LSTMs have the ability to remove or add information to the cell state, which prevents vanishing gradient problems. This process is carefully regulated by structures called gates, which are composed of a sigmoid layer and a pointwise multiplication operation. As shown in Figure 2.4, LSTMs have three gates. Forget Gate (f) decides what information we are going to throw away from the cell state. It looks at y_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state c_{t-1}. Here, 1 represents "completely keep this" and 0 represents "completely get rid of this." The Input gate decides what new information we are going to store in the cell state. This has two parts. First, a sigmoid layer (input gate layer) decides which values we will update. Next, a tanh layer creates a vector of new candidate values added to the cell state. Then these two are combined through pointwise multiplication to update the cell state c_{t-1}, into the new cell state c_t. The Output gate consists of a sigmoid layer that decides what parts of the cell state are going as an output. Then, we put the cell state through tanh (to push the values between -1 and 1) and then multiply it by the output of the sigmoid gate.

$$
\begin{aligned}
f_t &= \sigma(W_f.[y_{t-1}, x_t]) + b_f \\
i_t &= \sigma(W_i.[y_{t-1}, x_t] + b_i) \\
\tilde{c}_t &= \tanh(W_c.[y_{t-1}, x_t] + b_c) \\
c_t &= f_t * c_{t-1} + i_t * \tilde{c}_t \\
o_t &= \sigma(W_o.[y_{t-1}, x_t] + b_o) \\
y_t &= o_t * \tanh(c_t)
\end{aligned}
\tag{2.7}
$$

Gated Recurrent Unit Networks (GRUs) GRU [41] is a slightly altered version of the LSTM. The GRU operates using 2 gates, a reset gate and an update gate, as illustrated in Figure 2.5. The reset gate determines how to combine new input to the previous memory and the update gate determines how much of the previous state to keep. Update gate in GRU is what input gate and forget gate were in LSTM. GRU has two values at the output instead of 3 (output and hidden state), i.e., GRU merges the cell state and hidden state. The resulting model is more straightforward than standard LSTM models, computationally more efficient.

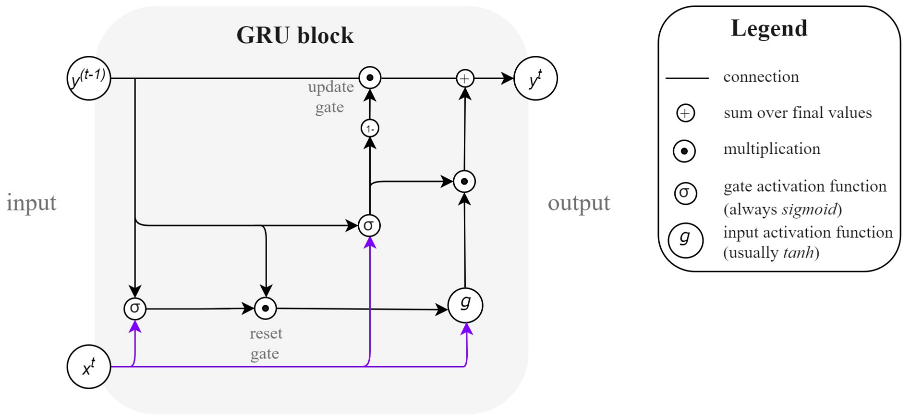

FIGURE 2.5: GRU block at t^{th} time step

$$
\begin{aligned}
z_t &= \sigma(W_z.[y_{t-1}, x_t]) \\
r_t &= \sigma(W_r.[y_{t-1}, x_t]) \\
\tilde{y}_t &= \tanh(W.[r_t * h_{t-1}, x_t]) \\
y_t &= (1 - z_t) * y_{t-1} + z_t * \tilde{y}_t
\end{aligned}
\tag{2.8}
$$

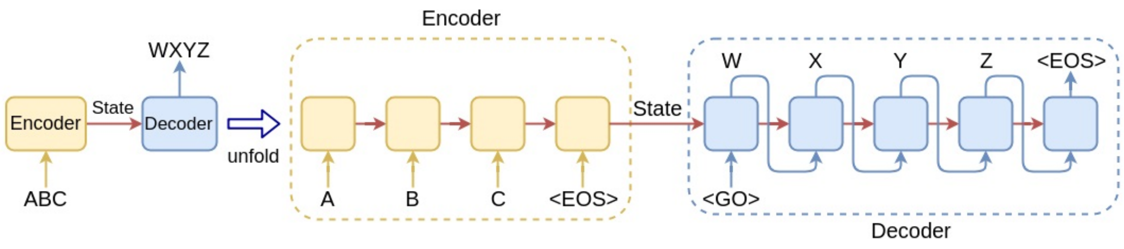

FIGURE 2.6: Seq2seq model with encoder and decoder to translate the input sequence ABC to $WXYZ$

2.3.3 Attention Mechanisms

CNN's are fit to learn the initial response from the temporal or spatial data but bear the capacity to learn the straight correlations. RNNs and their variations are specialized for linear modeling, consequently inexpert to represent non-continuous features. One of the applications of RNNs is the seq2seq model, which is employed on various NLP tasks such as language generation, text summarization and Machine translation systems. The seq2seq model encodes information of source sequence into a single vector and uses that encoded information into the generation of target sequence (usually, source and target sequence may vary in their lengths) for the machine translation system. As shown in Figure 2.6, the seq2seq model has two works:

- Encoding source information into a single vector known as context vector, with the assumption of composing a good summary of the whole source sequence. This process is done by the encoder in seq2seq.
- The generated context vector of the source sequence is used to produce the target sequence. The last hidden state of the encoder is considered as the initial decoder state in the standard seq2seq model. This process falls under the decoder.

Since seq2seq model is entirely based on the last state of the encoder and it is supposed to contain all information of the input sequence. The standard seq2seq model cannot memorize long sequences as it is hard to encode all information in a single context vector. Attention was invented to provide a mechanism to eliminate the drawback of the standard seq2seq encoder-decoder model. It overcomes the drawback of forgetting the long sequences. A vector of assigned weights is created corresponding to each output element, focusing on the important words for a specific prediction. It generates the weight for remaining words of that time step to identify how strongly words correlate to remaining words and compute the weighted sum of attention values.

Bahdanau et al. [14] proposed attention mechanism that improved the seq2seq model by aligning the decoder with the relevant input sentence. It first produces the encoder hidden states by using any RNNs or any of their variants to encode the input sequence, and then it calculates the alignment scores. It quantifies the amount of "attention" the decoder will place on each of the encoder outputs when producing the subsequent output, as shown in Figure 2.7.

$$alignmentscore = W_{combined}.tanh(W_{decoder}.H_{decoder} + W_{encoder}.H_{encoder}) \tag{2.9}$$

The SoftMax operation is employed to produce the attention weights over the alignment scores, which will cause the values in the vector to sum up to 1. The context vector is calculated on the output obtained from the above step by performing an element-wise multiplication of the attention weights with the encoder outputs. The decoder output is amplified if the score of a specific input element is closer to 1, whereas if the score is close to 0 then the influence is nullified. The finally decoded output is obtained by concatenating the context vector with the previous decoder's output.

Hard attention is a modification of soft attention [14] for selecting the specific patch to focus on, one at a time. Thus, less calculation and inference time is involved compared to soft attention. However, it needs more complex computations to reduce variance. This technique is also non-differentiable. **Local attention** mechanisms guide the construction of various other attention mechanisms such as global attention. Hard attention, along with its differentiability using adjacent words, forms local attention. In the other hand, global attention uses the current timestamp's output of the encoder and decoder. The combination of global and local attention is proposed as the **Global-local mutual attention** model [139]. Apart from the seq2seq model, another state-of-the-art of encoder-decoder model is Transformer which uses a different attention mechanism called self-attention.

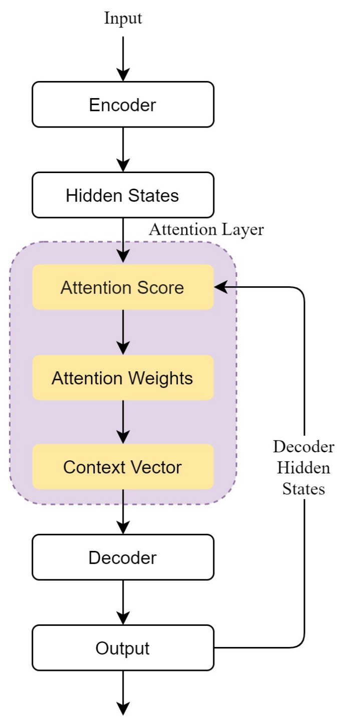

Figure 2.7: Bahdanau's attention model to a encoder-decoder based model

The Transformer model has an encoder and decoder stack. Like encoder stack, the decoder stack has many decoders, where each stack comprises self-attention and feed-forward neural network, shown in Figure 2.8. Self-attention captures pair-wise relations among input tokens in a sequence. Due to such property, it is also known as the intra-attention model. The self-attention network gets contextual features of every word by attending to all words in the same sentence. Thus, it can extract global information from a sentence. Self-attention network first of all computes attention weights between every adjacent word and then uses a weighted sum operation to obtain contextual information. The feed-forward neural network predicts the score for the word. It makes sense to provide appropriate attention to words that have already been produced during decoding to produce a resulting sentence. The encoder and decoder of the transformer also comprise other techniques such as positional encodings, layer normalization [12], residual connections [94] and dropouts.

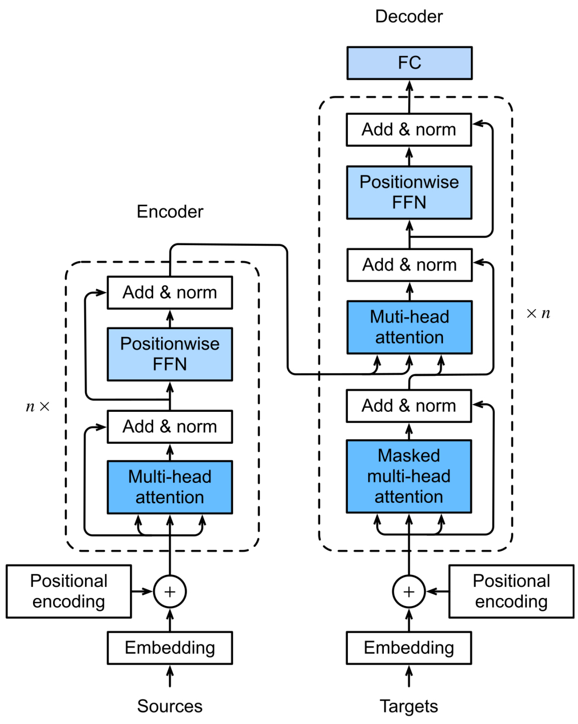

FIGURE 2.8: Vaswani et al. [242] transformer architecture which comprise the encoder-decoder

2.4 Transfer Learning

Many NLP tasks have improved performance after using transfer learning which helps to learn a new task or the same task but in a different language or domain more efficiently. Let us suppose a deep learning-based POS tagger has trained on the high resource language. Transfer learning allows us to train a new POS tagger for a similar language utilizing the existing knowledge of the previous POS tagger. Such knowledge transfer enhances the model generalization by leveraging data from another language during learning. It yields a significant improvement in not only POS tagging but also in various NLP tasks [38, 63, 99, 186]. NLP tasks share common knowledge about language at distinct levels, orthography, word order, phonological, semantic, lexical and structural similarity [205, 67] to accomplish such knowledge transfer. However, transfer learning yields a few questions such as -

- What if the common knowledge is entirely based on the language or domain instead of the task.

- What if the common knowledge is entirely based on the task instead of the language.
- What if the common knowledge is shared in a sequential or parallel style during model training.

All these questions were addressed in the transfer learning taxonomy for the NLP field, proposed by Pan and Yang [181] and Ruder et al. [205], shown in Figure 2.9. Transfer learning is defined according to them as:

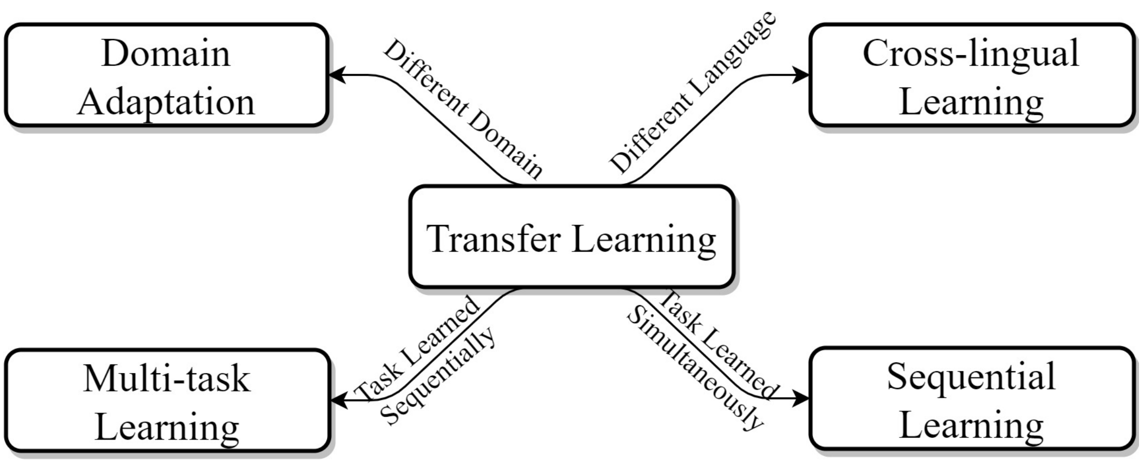

FIGURE 2.9: Transfer learning taxonomy for NLP

"Given a settings $S = \{D, T\}$ where, D is a dataset that contains a feature space $X = \{x_1, \ldots, x_n\}$ with a marginal probability distribution $P(X)$ over the feature space. On the other hand, a task $T = \{Y, P(Y), P(Y|X)\}$ consists of a label space Y, a prior distribution $P(Y)$, and a conditional probability distribution $P(Y|X)$, which is usually learned using the training data consisting pairs of $x_i \in X$ and $y_i \in Y$. Having a source setting S_s including D_s and a corresponding source task T_s, as well as a target setting S_t with D_t and target task T_t, the aim of transfer learning is to perform the target task in order to learn the target $P_t(Y_t|X_t)$ in D_t using the information provided by the elements in the source setting where $D_s \neq D_t$ or $T_s \neq T_t$. Typically, it is assumed that the target setting is either in low-resource or zero-resource mode."

According to the taxonomy, transfer learning has following scenarios:

1. Case I: When a dataset of source and target are contrasted due to variation in feature space, $X_s \neq X_t$, but the task of both datasets are identical. This scenario addresses question 1 for the language perspective, which is considered as a cross-lingual transfer.

2. Case II: When both the tasks are identical, but the marginal probability distribution of both datasets are contrasted to each other, $P_s(X_s) \neq P_t(X_t)$, then it comes under the domain adaptation. For example, the vocabulary distribution of the news genre will be different from the healthcare genre. It also supports the answer to question 1 in terms of the domain.

3. Case III: Regardless of whether both datasets are contrastive or identical, both tasks are contrastive, $T_s \neq T_t$. In this case, if these tasks are learned simultaneously, it is considered multi-task learning, which addresses question 2 and parallel style learning from question 3.

4. Case IV: Like Case III for datasets and tasks, it is considered sequential learning if learning is performed in a sequential style. Based on the dataset similarity (cross-lingual transfer) and dis-similarity (domain adaptation), this scenario also considers Case I and Case II.

There is another view of transfer learning for the NLP problems mainly based on resource availability. If the availability of the target dataset has few labeled data, it comes under the umbrella of few-shot learning. Even a minimal resource allows model generalization due to the way of dealing with parameters. When there is no labeled target dataset, all the problems are solved through zero-shot learning. Such a learning technique looks at auxiliary or unlabeled data. Apart from these auxiliary data, the model performance is also enhanced by model regularization and optimization.

2.5 Distributional Vector Representation

The use of transfer learning is defined as a way of input encoding where the embedding vector has been generated on a large corpora (it could be monolingual or multilingual) and using that generated embedding for different tasks (language). The embedding vector generation is the first step to convert a word into a vector. The embedding vector could be a real-valued such as distributional or discrete-valued, e.g., one-hot vector. The input is in the form of text and one-hot vector representation faces the curse of dimensionality because the vector size is dependent on the number of unique tokens. An embedding module transforms a word into a fixed (low) dimensional real-valued vector. Bengio et al. [19] proposed the first neural network architecture, which used a feed-forward neural network for getting the vector for a word. The generated embedding vector could be context-independent or context-dependent.

2.5.1 Context Independent Embedding

Over time, several methods [207, 72, 218, 151, 154] have been proposed for getting the embeddings, that are built on the top of the word. All those embeddings are count-based and prediction-based. A few of them are prevalent in NLP due to attaining notable results on the various NLP tasks, including sequence labeling.

One of the most popular is Google's Word2Vec [152] that is a statistical method with two different learning models that are Continuous SkipGram model and the Continuous Bag-of-Words model, used to generate real-valued dense vector. **Continuous SkipGram model** learns by predicting the surrounding words of a given current word (w_t), as depicted in Figure 2.10. There are two possible techniques for these model training, hierarchical softmax and negative sampling. The researchers prefer the SkipGram model with negative sampling training due to its robustness and efficiency. The negative log-likelihood of the training data is minimized under the assumptions of skip-gram as an objective.

$$\mathcal{L} = -\frac{1}{|C|} \sum_{t=1}^{|C|} \sum_{-C \leq j \leq C, j \neq 0} \log \; P(w_{t+j}|w_t) \tag{2.10}$$

Softmax is used to compute $P(w_{t+j}|w_t)$ by:

$$P(w_{t+j}|w_t) = \frac{\exp\left(\tilde{x}_{t+j}{}^{\top} x_t\right)}{\sum_{i=1}^{|V|} \exp\left(\tilde{x}_i{}^{\top} x_t\right)} \tag{2.11}$$

Here, w_i is the word for which the word vector (x_i) and contextual word vector ($\tilde{x}_i$) are generated. The model architecture is a shallow but wider network obsoletes a hidden layer. The hidden state of the word vector x_i to the current word w_i is passed into the softmax. Here, each word has its representations ($\tilde{x}_i$), based on the context of the current word. It is hard to compute the partition function of the softmax, and negative sampling approximates the softmax in a computationally efficient way. Gutmann and Hyvärinen [92] have used negative sampling for noise contrastive estimation simplification. Here, the model training with negative sampling distinguishes negative samples drawn from a noise distribution, P_n from an actual target word w_t is defined as follows:

$$P(w_{t+j}|w_t) = \log \sigma(\tilde{x}_{t+j}^{\top} x_t) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n} \log \sigma(-\tilde{x}_i^{\top} x_t) \tag{2.12}$$

Where k and σ are the number of negative samples and sigmoid function, respectively, the P_n is the unigram distribution which has $3/4^{th}$ power.

Continuous Bag-of-Words model learns the word vector by predicting the current word (w_t) based on its C context words, as depicted in Figure 2.11. It can be assumed as an inverse of the SkipGram model. The objective of CBOW is defined as:

$$\mathcal{L} = -\frac{1}{|C|} \sum_{t=1}^{|C|} \log P(w_t|w_{t-C}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c}) \tag{2.13}$$

$$P(w_t|w_{t-C}, \ldots, w_{t+C}) = \frac{\exp\left(\tilde{x}_t{}^{\top} x_s\right)}{\sum_{i=1}^{|V|} \exp\left(\tilde{x}_i{}^{\top} x_s\right)} \tag{2.14}$$

The sum of word vectors, $w_{t-C}, \ldots, w_{t+C}$, i.e. $\sum_{-C \leq j \leq C, j \neq 0} x_{t+j}$ is represented by x_s. Due to the computational efficiency, it is also trained through negative sampling.

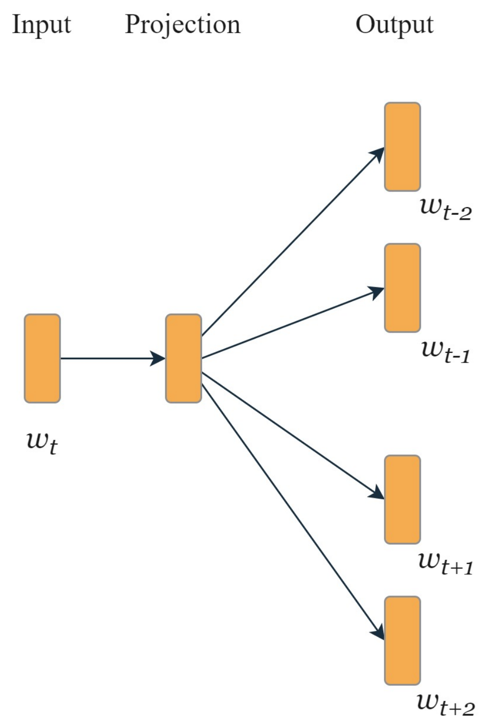

FIGURE 2.10: The continuous skip-gram model

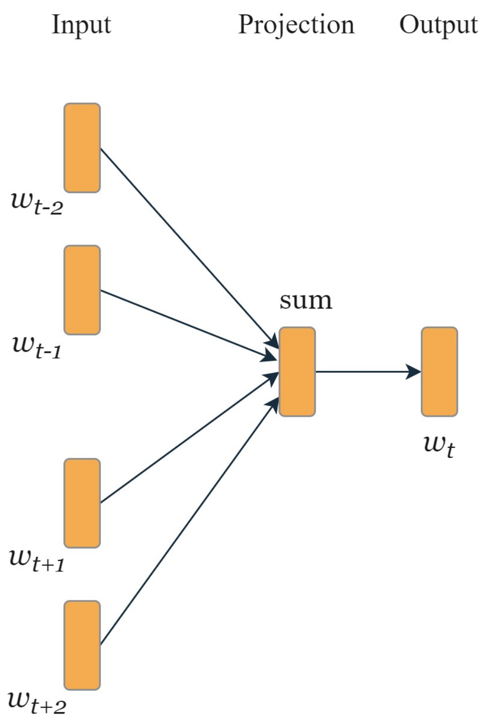

FIGURE 2.11: The continuous bag-of-words model

Global matrix factorization method and fixed local context window method are two main ways for learning word vectors. The frequent words influence the vectors' similarity, which is the major shortcoming of the global matrix factorization methods, whereas the Word2Vec Skip-Gram model strengthens local contextual information. **Global Vectors (GloVe)** [185] unites the advantages of the local contextual features and global matrix factorization. Initially, it constructs C, a co-occurrence matrix of specified window size. Then, it reduces the difference between the dot product of the word w_i vector's, logarithm of C and context word c_t.

$$\mathcal{L} = \sum_{i,j=1}^{|V|} f(C_{ij})(x_i^\top \tilde{x}_j + b_i + \tilde{b}_j - \log(C_{ij}))^2 \tag{2.15}$$

Here, w_i and w_j are the word and its context word along with their biases b_i and $\tilde{b}_j$, respectively. The rare and frequent co-occurrences regulated by the weighted function $f(.)$.

Several methods are also proposed over time, e.g., SENNA [46]. Some of them are extensions of these techniques at the document and paragraph [128] levels. Bojanowski et al. [28] consider the word as a set of n-gram characters to skipGram model. Such kind of modelling is useful for morphologically rich languages. The Continuous Bag-of-Word model is augmented with compositional n-gram characteristics by Zhao et al. [270], Pagliardini et al. [179].

2.5.2 Context Dependent Embedding

The earlier defined embedding methods always generate a single vector corresponding to a word. However, the ambiguous words change their meaning according to their contexts in a sentence such as "My friend is considering to take a loan from a bank." and "Rainfall caused Rhine river to overflow it's bank." Both the sentences have the same embedding vectors for the *bank*, although the semantics is entirely different here. In the first sentence, the *bank* refers to a financial service provider, whereas in the second sentence, it represents the slope beside a body of water. Such kind of issues are accommodated by generating word vector embedding considering the sentence-level semantics.

Recently, many methods have been proposed [3, 186, 63] which are prediction-based and consider contextual information. These methods train language models on an ample corpus. In these methods, the internal representation of states is considered as word embedding.

Embeddings From Language Model (ELMo) is an NLP framework introduced by Peters et al. [186]. Unlike traditional word embeddings, ELMo is a deep contextualized word representation that models both complex aspects of word use (such as syntax and semantics) as well as how these characteristics vary across linguistic contexts (i.e., to model polysemy). It represents embeddings for a word using the complete sentence containing that word. Therefore, it can capture the context of the word used in the sentence and generate different embeddings for the same word based on its context in sentences. ELMo model is built on the top of two-layer bi-directional language models (biLMs) designed

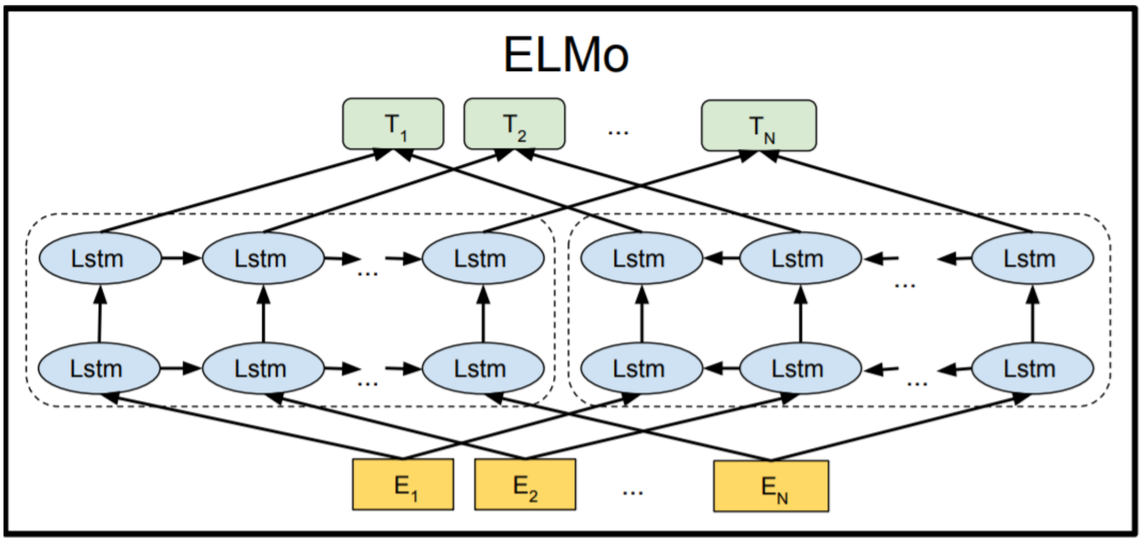

FIGURE 2.12: ELMo model architecture for pre-training representation

using LSTM, with character convolution, as shown in Figure 2.12. Each layer of biLMs has a forward and backward language model.

The forward language model computes the probability to a token t_k over the N tokens, based on previous tokens $t_1, t_2, \ldots, t_{k-1}$. Like the forward language model, the backward language model computes the probability of token t_k based on the based on forwarding tokens $t_{k+1}, t_{k+2}, \ldots, t_N$.

$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k | t_1, t_2, \ldots, t_{k-1}) \tag{2.16}$$

$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k | t_{k+1}, t_{k+2}, \ldots, t_N) \tag{2.17}$$

A biLM combines forward and backward LM and aims to minimize the negative log-likelihood of the biLM as given below:

$$\mathcal{L} = -\sum_{k=1}^{N} (\log p(t_k | t_1, \ldots, t_{k-1}; \theta_x, \overrightarrow{\theta}_{LSTM}, \theta_s) + \log p(t_k | t_{k+1}, \ldots, t_N; \theta_x, \overleftarrow{\theta}_{LSTM}, \theta_s)) \tag{2.18}$$

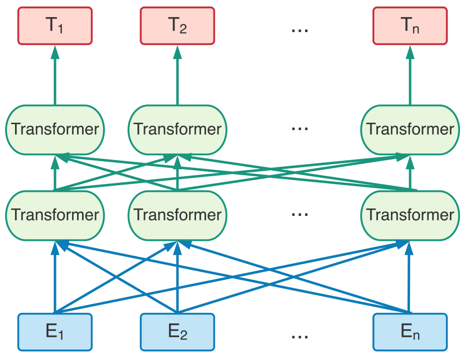

FIGURE 2.13: BERT model architecture for pre-training representation

ELMo needs to be pre-trained on vast corpora to better learn sentences with a vast number of different vocabularies, which yield the same neighbourhood of the word in the vector space representing similar meaning. As a result, it provides a satisfactory improvement on different NLP downstream tasks such as question answering, textual entailment and sentiment analysis.

Devlin et al. [63] introduced the **Bidirectional Encoder Representations from Transformers (BERT)** model, which is a multi-layer bidirectional Transformer encoder based on the architecture of Vaswani et al. [242], as shown in Figure 2.13. Unlike ELMo, BERT is designed to pre-train deep bidirectional representations using the Transformer that holds words on both right and left context in all layers of unlabeled text. BERT is pre-trained using two unsupervised tasks: Masked Language Model and Next Sentence Prediction to predict masked tokens and capture sentence relationships, respectively. To achieve these two tasks through the cross-entropy loss, BERT considers token embeddings with specific words along with tokens such as the first token of the first sentence identifier (CLS), sentence separator (SEP) and an unknown word (MASK), sentence embeddings where the number of sentences generate embedding and positional embeddings as the Transformer does, as input. The output from the encoder layers is used to fine-tune it for the NLP downstream tasks. There have been several advances in the context dependent embedding generation by reducing learning parameters of existing models, using different learning tasks to produce state-of-the-art results [269, 266, 135, 210, 258].

The context-dependent and independent embeddings are used as pre-trained embedding for the low resource languages. Based on the task, these pre-trained embedding used either word and sentence vector or finetune the existing learned word vector.

2.6 Hand Crafted Features in Neural Network

Deep learning models are based on automatic feature learning based on the training instances, making them superior to the machine learning algorithm. The handcrafted features allow a model to deal with unknown or new inputs. However, before fully end-to-end deep learning models were proposed, feature engineering was typically utilized in neural models where handcrafted features, such as word spelling, gazetteer, and morphological features, can significantly benefit NLP problems. These features are represented as discrete vectors integrated into the input or embedding module. Later on, much work has been compiled on integrating features into a neural network in computer vision but lacks in the NLP domain. Limited work has been done in this area that has shown that these features can boost the performance of deep learning models for NLP tasks. For example, Collobert et al. [46] used word affix, gazetteer and capitalization features, and cascading features that include tags from their related tasks, semantic role labeling, named entity recognition, part of speech tagging and chunking.

2.7 Literature Survey on Sequence Labeling

This thesis evaluates three essential sequence labeling NLP tasks required when working with any new languages. These tasks, POS tagging, Chunking and NER are considered building blocks for NLP problems. POS tagging assigns a sequence of grammatical categories (POS tags) to the given word sequence (sentence), while Chunking links POS tagged words into groups of words or chunks, which can be roughly defined as minimal phrases or minimal constituents. NER is another building block for NLP problems, marking proper nouns and other named entities such as Location, Person, Organization, etc. There are more than 7000 natural languages that are still widespread in the world. It is important to remember that most of these languages are low resource languages or resource-scarce languages Christianson et al. [44]. So in the next section, we cover the current work on Indian languages since most Indian languages are low resource languages. The subsequent section summarizes deep learning-based solutions that include handcrafted features as additional input.

2.7.1 Sequence Labeling Work on Indian Languages

There has been a substantial amount of work related to general sequence labeling for Indian languages: Hindi [160, 221, 224, 55, 262, 119], Oriya [262], Marathi [262], Punjabi [262, 119], Bengali [262, 119], Kannada [10], Malayalam [121, 119], Tamil [64] and Telugu [104, 119] etc. We mention some of the results below, not in a chronological order. A Malayalam POS tagger was developed based on HMM [144] and trained with using approximately 1,400 tokens. The authors claimed that the increasing the number of tokens, the accuracy of the system could increase. The tagger gave a performance measuring 90% accuracy.

Similarly, Antony et al. attempted on Malayalam using a machine learning approach based on SVM [9]. The performance measurement of the POS tagger was evaluated in terms of accuracy, using SVMTeval. Initially, the smaller the size of the lexicon, the lower the accuracy of the tagger. As the size of the lexicon increased to 1,80,000 words, it reached an accuracy of 94%. Based on the two approaches, Ekbal et al. used stochastic schemes for the Bengali tagger with HMM and Maximum Entropy. This Experiment performed on varying sizes of training data (40K, 20K, and 10K terms) to understand the relative performance of the models. The best performance was achieved by morphological restriction and suffix information on the possible grammatical categories of a word. Using any of the models, the Morphological analyzer enhances the performance of the POS tagger significantly [71].

In the next attempt, Ekbal et al. used the SVM approach for POS tagging of the Bengali language. A reliable test set of 20K word forms was used to get the evaluation result of 86.84% by introducing the unknown word handler system [70]. The main idea of POS tagging in Bengali was based on using the different possible combinations of available word and tag context, including affixes, for all the words as the features. The POS tagger was trained and tested on 72K and 20K word forms, respectively. With the lexicon, NER, and unknown word features, the POS tagger improved the accuracy significantly and achieved an accuracy of 90.3% with the CRF model [71].

Singh et al. proposed the POS tagger for the Hindi Language. They claimed that the use of affix information without contextual information showed that Verb Group (VG) accurately identified the auxiliaries and the main verb. The average accuracy of this learning-based (LB) tagger is 93.45% on the 3:1 ratio of corpora [224] using 4-fold cross-validation. Mishra et al. introduced a Hybrid approach on which HMM leverages tag sequence probability and hand-crafted rules applied to increase the accuracy of tag words. The system achieved 96.01% of average precision and 89.32% of average accuracy on 13,000 words [159]. The Maximum entropy (ME) based tagger used four main features: Context-based, Word-based, Dictionary-based, Corpus-based features, and obtain 88.4% accuracy after 10-fold cross-validation on Hindi POS tagging [54].

The first POS tagger for Sindhi (Persio-Arabic) applied a rule-based approach to a corpus of the Sindhi dictionary with 96.28% accuracy of tagging and tokenization algorithms [141]. Another rule-based Sindhi tagger proposed by Mahar et al. adopted the WordNet to alleviate the ambiguous situations in the undigitized text. This tagger was trained on the corpus of a comprehensive Sindhi dictionary and tested on lexicon, which contained 26,366 tagged words with 97.14% accuracy [142]. The POS tagging tasks based on CRF used Sindhi-Devanagari script on the tag-set of the Bureau of Indian Standards (BIS), and the tagger achieved an accuracy of up to 92% [165].

The divide and conquer approach was applied in the identification of local word groups or phrases of Oriya text. This strategy breaks the task of Chunking into different sub-tasks according to sensible features of per phrase and identifies different phrases in parallel and achieves an accuracy of 91.3% [15]. Yajnik et al. introduced POS tagging for Nepali text based on HMM and the Viterbi algorithm. The Viterbi algorithm achieved an accuracy of 95.43% [254].

The development for the Sambalpuri POS tagger was based on using SVM and CRF techniques on the 121K corpus procured from the web. The collected corpus was converted into the Unicode encoding and annotated with BIS tagset. The SVM tagger provides a better accuracy 83% then CRF 71.56% on this dataset [18].

For the Gujarati Chunker, initially an HMM-based model was developed and hand-crafted rule were applied during post-processing on an annotated corpus of 0.1M sentences. The model uses a train-test split with ratio 80-20 for the training set, and the testing set achieves the accuracy of 95.31% [112].

HMM was also used for Assamese tagger development on an annotated corpus of 256,690 words with 38 tag labels and it obtained an accuracy of 89.21% [53]. As part of Kashmiri tree-bank development, the CRF POS tagger, a Chunker has been developed and it obtains 83.45% accuracy with morph segmentation and 81.34% with automatic POS tagging [22].

A recent attempt at annotating a Khasi language corpus using the BIS tagset was evaluated on 86,087 words by using the plain HMM and TnT that achieved the accuracies of 88.64% and 95.68%, respectively [238]. Manipuri sequential labelling was evaluated on Rule-based tagger, HMM, CRF and SVM [175, 228, 227].

A collection of a corpora of 1M words for the Bhojpuri and Magahi was created, out of which 89997 and 63000 words were annotated at the POS level. For the Bhojpuri, 86.7% and 88.5% performance was achieved by applying CRF and SVM, respectively, and Magahi was evaluated on Memory based model, TnT, SVM, MxPost with 86.09%, 86.22%, 41.46%, 89.61% accuracies, respectively [176, 225, 120]. Maithili corpus from web resources and Wikipedia dump was used towards developing a POS, using CRF on 52,190 words with an accuracy of 82.67% [192]. Apart from this, HMM model was also employed on 10K words with 95.67% [203].

One of the earliest works on NER for Indian languages was reported by Cucerzan et al. [51], mainly for Hindi. The authors used a bootstrapping algorithm and an iterative learning algorithm to classify the names (both first and last name) and places on the 18806 tokens and achieved 41.70% and 79.04% F_1-score and accuracy, respectively. The IJCNLP 2008 workshop on NER for South and South East Asian Languages[1] was the first shared task for NER for Indian (or South Asian) languages and it also reported perhaps the

[1]http://ltrc.iiit.ac.in/ner-ssea-08/

first dataset (in the public domain) for NER for Indian languages. It included five Indian languages: Hindi, Urdu, Bengali, Oriya and Telugu [222]. Of these, the Bengali dataset was developed by Jadavpur University and IIIT, Hyderabad, and Urdu by CRULP, Lahore and IIIT, Allahabad and the rest was developed by workshop organizing institute (IIIT, Hyderabad). These datasets were annotated with 12 NE tags. Ekbal et al. [69] achieved 91.8% as the best F_1-score from Support Vector Machines (SVM) on the annotated Bengali news corpus of 467858 tokens by 16 entities.

Saha et al. [209] worked on Hindi NER by using MEM, which assigns an outcome for each token based on its history and features. They used about 243K words for training purposes, which was taken from the Dainik Jagran[2] (a popular Hindi newspaper), out of which about 16482 belonged to 4 named entities. Their MEM based NER system was able to achieve a F_1-score of 81.52%, using a hybrid set of Gazetteer, patterns, and lexical and contextual features.

Sudha et al. [163] worked on NER using HMM for Hindi, Urdu and Punjabi. They used different number of tags for different corpora belonging to different domains. For example, they used Person, Location, River and Country tags in the tourism corpus; and Person, Time, Month, Dry-fruits and Food items tags in the story corpus.

The Punjabi language is mainly written in two scripts: Gurmukhi (of Brahmi origin, like Devanagari) and Shahmukhi (a variant of the Persio-Arabic script). Most of the earlier works [110, 111, 163] on NER for this language were on data in Gurmukhi script and used statistical algorithms. Recent work on NER for Punjabi language using Shahamukhi script was explored by using 318275 tokens, out of which 16300 are entities such as Person, Location and Organization. The authors obtained 85.2% as best F_1-score after applying the RNN over other classical and neural network techniques [2].

There is very little work on the Sambalpuri language, an Indo-Aryan language spoken in parts of the Indian state of Odisha. Behera et al. (2017) worked on Sambalpuri and Odia NER using SVM [17]. They took 112K words for Sambalpuri and 250K words for Odia.

[2]`https://www.jagran.com/`

They made 7 labels for annotating named entities. The F_1-score measure obtained for Sambalpuri was 96.72% and 98.10% for Odia.

Lalitha Devi et al. (in 2008) worked on 94K words of Tourism domain for Tamil NER by using CRF. They used a total of 106 tags divided into three categories of ENAMEX[3], TIMEX and NUMEX. They obtained 80.44% F_1-score [194]. Malarkodi C. S. et al. (2012) [49] also worked on Tamil NER using CRF model. They observed challenges in NER which occur due to several factors and are also applicable to other Indian languages as follows: agglutination, ambiguity, nested entities, spelling variations, name variations and the lack of capitalization.

Rao et al.(2015) [196] conducted a shared task as part of the FIRE 2015 conference on Entity Extraction From Social Media using Named Entity Recognizer for Indian languages. They collected their corpus using the Twitter API in the period of May-June 2015 for training data and August-September for testing data of Tamil, Malayalam, Hindi and English languages. They used 22 tags for different kinds of names. Different participating teams used various machine learning methods (CRF, SVM, HMM, Naive Bayes, Decision Tree) with diverse sets of features. The baseline F_1-score 47.10 for Hindi, 19.05 for Tamil 31.24 for Malayalam and 40.56 for English was reported.

Ali et al.(2020) [6] has reported very recent work on the NER corpus for Sindhi. In which they annotated over 1.35 million tokens with eleven entity classes using corpus derived from Awami Awaz and Kavish newspapers and reported a best benchmark F1-score of 89.16% by using CNN-LSTM-CRF model on it.

Kumar et al. proposed a deep learning approach for Malayalam Twitter POS tagging and perform a comparison among the deep learning sequential models in order to find the suitable method for tagging at both the word-level and character-level granularities. For performing experiments, a sequential model algorithm, such as a RNN and its variants: LSTM, GRU and Bi-LSTM were used. The experiment was conducted at word-level and character-level for four different number of hidden states such as 4, 16, 32, 64. It was found that the highest F1-measure of GRU based sequential model at word-level and Bi-LSTM

[3]`http://cs.nyu.edu/cs/faculty/grishman/NEtask20.book_6.html#HEADING17`

based deep learning sequential model at character-level gave results of 92.54% and and 87.39%, respectively [122].

Kann et al. accomplished this work with the help of three auxiliary tasks a) lemmatization, b) character-based random string autoencoding, and c) character-based word autoencoding. POS tagging for low-resource language takes advantage of the raw text autoencoding, as it uses the lemma information. In low-resource scenarios, the neural POS tagger closes the gap to the-state-of-art POS tagger (single task) by 43%, even exceeding it on languages with templatic morphology (e.g., Arabic, Hebrew, and Turkish) to a significant degree [108].

Plank et al. explained a novel multi-task Bi-LSTM model with auxiliary loss and they evaluated tokens and sub-levels for neural network-based POS tagging on 22 Indian languages. The auxiliary loss can used to improve the accuracy of rare words [189]. Mishra et al. used feature transfer from a rich-resource language to poor-resource languages. Across eight different Indian Languages, accuracies achieved were encouraging, without any knowledge of the target language and human annotation. This approach can help us in creating annotated corpora for resource-poor Indian languages [161].

Prabha et al. proposed a deep learning-based POS tagger for Nepali text, which was built using variations of RNN, LSTM, GRU and their bidirectional variants with more than 99% accuracy [191]. The first reported attempt towards the development of deep learning-based Maithili POS tagger was trained on the CBOW, word embedding trained with the help of available web resources and Wikipedia dump as corpus and got 82.67% on CRF classifier, which increased to 85.88% when using the embeddings [192].

2.7.2 Sequence Labeling Models with Feature Engineering

Deep learning models are more robust for feature engineering than machine learning while having large annotated datasets. In the past few years, deep learning techniques ruled for NLP tasks, especially bottleneck applications due to wide support of real-vector representation of a word through pre-trained word embeddings, Word2Vec [152, 155], Glove [185], Polyglot [4], Senna [46], FastText [28] or contextual enabled word embeddings, ELMo [186], BERT [63], GPT [195], etc. However, the contextual information representation is based on the RNN and its variants, CNN and a combination of these. Similarly, CRF, Softmax and pointer network are used for the label disambiguation. Some work focuses on extending the contextual representation [200, 246, 134], incorporating handcrafted features [268, 40, 81] into the deep learning model in a more aggressive manner and obtaining further decent improvements for sequence labeling. Based on these feature representations, contextual encoding and learning algorithms, along with feature inclusion, deep learning-based solutions have been presented here.

Akbik et al. [3] have used a pretrained language model along with Glove embedding as external features to the model. One of the earliest attempts for feature integration was made by Chiu et al. [40], where capitalization and lexicon are used for NER tool development. Later on, this works extended by integrating spelling features and gazetteer as external features along with multi-order bidirectional LSTM contextual encoder for performing segmenting and sequence labeling, proposed by Zhang et al. [**?**]. Huang et al. [102] have adopted the spelling features, including the word affixes, capitalization, length of characters, contextual words based on n-gram (unigram, bigram and trigram), and gazetteer. Similarly, Chiu and Nichols [40] also used character-type, capitalization and lexicon features. Dozat et al. [66] have used attention mechanism with LSTM at embedding module along with pretrained word embedding, Word2vec and FastText for POS tagging and dependency parsing. Xin et al. [252] tried to capture the best representation of the character to word by proposing the IntNet model, a funnel-shaped wide CNN model without downsampling that captured the internal structure of words by composing their characters.

Wu et al. [250] proposed a hybrid neural model which combines loss components of auto-encoder to utilize handcrafted features, POS tags and word information such as shapes and gazetteers. They got significant improvements compared to existing competitive models on the NER task. The auto-encoder auxiliary component takes handcrafted features as input and reconstructs them into output, which helps the model preserve important information stored in these features. However, designing such features for low-resource languages is challenging because gazetteers in these languages are absent. To address this problem, Rijhwani et al. [201] proposed a 'soft gazetteers' method that incorporates information from English knowledge bases through cross-lingual entity linking and creating continuous-valued gazetteer features.

Ghaddar et al. [81] proposed a novel lexical representation (called Lexical Similarity vector) for NER, indicating that robust lexical features are quite useful and can greatly benefit deep neural network architectures. The authors first embed words and named entity types into a joint low-dimensional vector space, then a feature vector for each word is computed offline, where each dimension encodes the similarity of the word embedding with the embedding of an entity type. The lexical similarity vectors are finally incorporated into the embedding module of their neural NER model.

The majority of earlier proposed work on neural POS tagging assumed that handcrafted features were antiquated for deep learning-based models and uniquely depended on end-to-end training. However, Faruqui et al. [75] earlier work combines semantic, symbolic features with word embedding. Similarly, Sagot et al. [208] use morphological lexicons as additional input, collected from Apertium and Alexina lexicons (language-specific) as n-hot features in the Plank et al. [189] model, which uses Bi-LSTM for input encoder and CRF for label decoder. The impact on the deep learning model by using handcrafted features provides a significant gain in performance. Recently, the research communities have been trying to make a robust model by improving self feature learning through the attention mechanism on the assumption of learning of the contextual features for the POS tagging. The attention mechanism captures the non-continuous relationship among the words of a sentence. Mundotiya et al. [166] have proposed self-attention and monotonic chunk-wise attention-based model and experimented on the Hindi dataset to handle non-continuous relationships within a sentence and a separate window by using a respective attention mechanism. Similarly, Wei et al. [247] proposed a new model based on the attention mechanism. In this paper, standard additive self-attention and position-aware self-attention mechanisms are exploited to implicitly encode positional information at discrete and variable-length of an input sequence, respectively, to provide complementary context information based on Bi-LSTM, a global sequence encoder. Shao et al. [220] used self-attention at the word and sentence level to obtain the contextual information on the same global sequence encoder, which was further used for disambiguating the labels of words by the semi-CRF approach.