Summer of Science 24
Mid-Term Report

NLP and LLMs

Mentor : Spandan Anaokar

Shorya Sethia

22B2725

Engineering Physics

Contents

Introduction

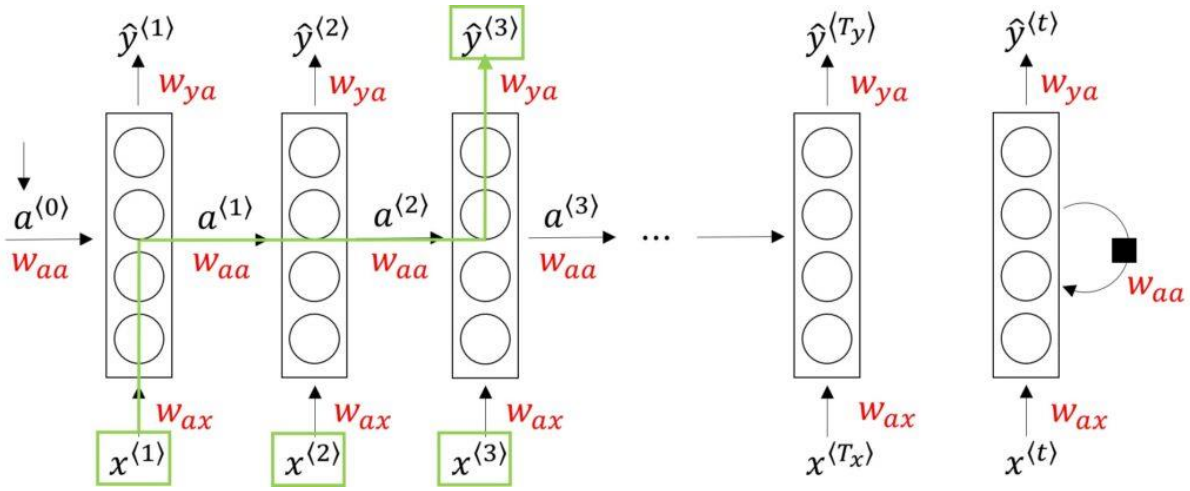
Natural Language Processing (NLP) is a field at the intersection of computer science, artificial intelligence, and language. It focuses on the interaction between computers and humans through natural language. To excel in NLP, a strong foundation in Machine Learning (ML) and Deep Learning (DL) is essential. This report details a structured learning path over last four weeks, covering ML basics, key NLP concepts, sequence models, and advanced architectures like Transformers.

Sequence Models

Recurrent Neural Networks (RNNs)

RNNs are designed for processing sequential data by maintaining a hidden state that captures information from previous time steps. Unlike traditional feedforward neural networks, RNNs can take into account the order of the inputs, making them suitable for tasks such as time series prediction, language modeling, and machine translation.

Architecture



W_{ij} means a weight matrix used to produce i like quantity and multiplied by j like quantity

The key component of an RNN is its hidden state, which is updated at each time step based on the current input and the previous hidden state. Mathematically, the RNN's operation at each time step t can be described as:

$$a_t = f(W_{aa}a_{t-1} + W_{ax}x_t + b_a)$$

Where:

- a_t is the hidden state at time step t
- a_t is the input at time step t
- W_{aa} is the weight matrix for the hidden state

- W_{ax} is the weight matrix for the input
- b_a is the bias term
- f is the activation function, as per paper $tanh$ but it can be $ReLU$, etc as well

The output y_t at each time step can be computed as:

$$y_t = g\ (W_{ya} a_t + b_y)$$

Where:

- g is the activation function for output layer, commonly softmax
- b_y is the bias term
- W_{ya} is the weight matrix for the output

Limitations

Vanishing Gradients: During backpropagation, gradients can become very small, making it difficult for the network to learn long-term dependencies. LeakyReLU is also somewhat not helpful.

Exploding Gradient: Conversely, gradients can become very large, leading to unstable training.

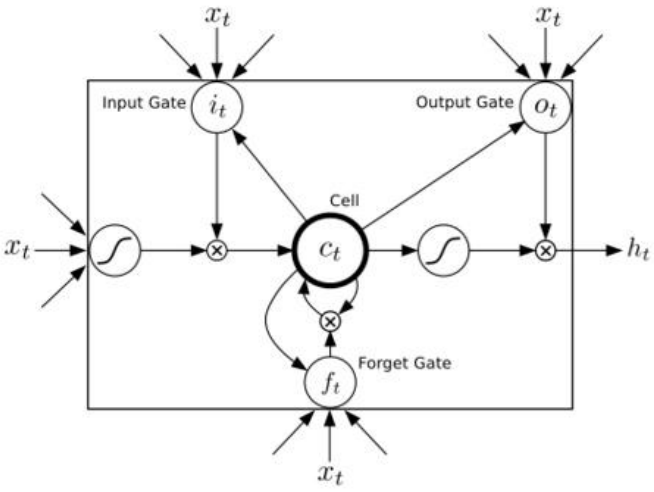
Limited Long-Term Memory: RNNs struggle to retain information from far-back time steps due to their inherently sequential nature.

Long Short-Term Memory Networks (LSTMs)

Introduced to address the vanishing gradient problem and improve the ability of RNNs to capture long-term dependencies. LSTMs achieve this by introducing a more complex cell state and a gating mechanism.

Architecture

An LSTM cell contains three gates: input, forget, and output gates, which control the flow of information into and out of the cell.



Input Gate: Determines how much of the new information will be added to the cell state.

$$i_t = \sigma\,(W_i\,[h_{t-1}, x_t] + b_i)$$

Forget Gate: Decides what portion of the previous cell state will be carried forward.

$$f_t = \sigma\,(W_f\,[h_{t-1}, x_t] + b_f)$$

Output Gate: Controls the output and what part of the cell state will be used to compute the hidden state.

$$o_t = \sigma\,(W_o\,[h_{t-1}, x_t] + b_o)$$

The cell state c_t is updated as:

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(W_c\,[h_{t-1}, x_t] + b_c)$$

$$h_t = o_t \odot tanh(c_t)$$

Where:

- σ is sigmoid function
- $\odot$ denotes element wise multiplication

Advantages

Long-Term Memory: Can retain information over long time sequences, making them suitable for tasks requiring long-term dependencies

Controlled Memory: The gating mechanism allows LSTMs to control the flow of information, mitigating the vanishing gradient problem

Bidirectional LSTMs (Bi-LSTMs)

Bi-LSTMs extend the LSTM architecture by processing the input sequence in both forward and backward directions. This allows the network to have access to both past and future context for each time step.

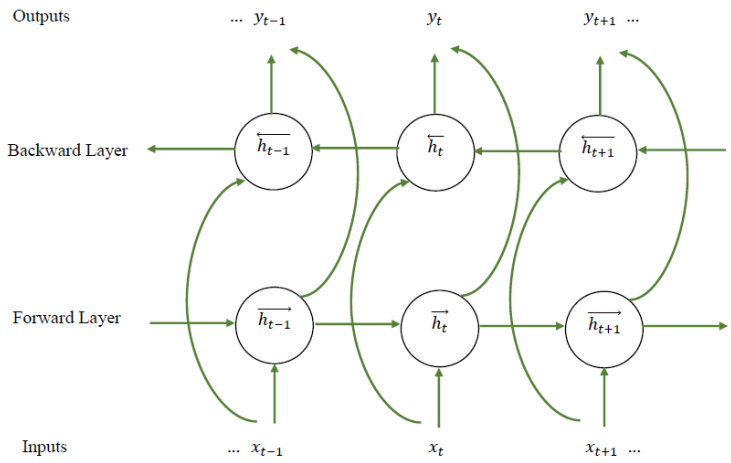
Architecture

A Bi-LSTM consists of two LSTM layers:

Forward LSTM: Processes the sequence from $t = 1$ to $t = T$

Backward LSTM: Processes the sequence from $t = T$ to $t = 1$

The hidden states h from both directions are concatenated at each time step



Advantages

Contextual Understanding: Bi-LSTMs can utilize information from both past and future contexts, leading to better performance on tasks like named entity recognition and speech recognition.

Access to future context often results in more accurate predictions.

Gated Recurrent Units (GRUs)

GRUs are a simplified variant of LSTMs that combine the input and forget gates into a single update gate. This results in a more compact architecture with fewer parameters.

Architecture

A GRU cell contains two gates: reset and update gates.

Update Gate: Determines how much of the previous hidden state should be retained

Reset Gate: Controls the degree to which the previous hidden state is ignored

Advantages

Simplified Architecture: Fewer parameters make GRUs computationally efficient and easier to train

Performance: Comparable performance to LSTMs on various tasks with reduced complexity

Why We Switched to LSTMs ?

The primary reason for switching from standard RNNs to LSTMs is the ability of LSTMs to capture long-range dependencies effectively. RNNs suffer from the vanishing gradient problem, where gradients diminish exponentially as they are propagated back through time steps. This makes it difficult for RNNs to learn from data points that are far apart in the sequence.

LSTMs address this issue through their gating mechanisms, which control the flow of information and gradients. By maintaining a cell state that can carry information over long periods, LSTMs are better suited for tasks where context from earlier in the sequence is crucial.

Hence, the switch to LSTMs (and subsequently to GRUs and Bi-LSTMs) has been driven by the need to improve the capability of neural networks to handle long-range dependencies in sequential data, which is a common requirement in many NLP tasks.

Key NLP Concepts

NLP involves several stages, each focusing on different aspects of language to transform it into a form that a machine learning model can understand.

Morphology

It is the study of the structure and formation of words. It deals with how words are built from the smallest units of meaning. It helps in parsing and understanding the grammatical structure of words.

Inflection: Modifying a word to express different grammatical categories (e.g., tense, number, gender).

> Examples: "walk" → "walks", "walking", "walked".

Derivation: Creating new words by adding prefixes or suffixes, often changing the word class (e.g., noun to adjective).

> Examples: "happy" → "happiness", "unhappy".

Compounding: Combining two or more free morphemes to create a new word.

> Examples: "notebook", "sunflower".

Understanding these processes helps in tasks like stemming and lemmatization, where the goal is to reduce words to their base or root forms.

Tokenization

Tokenization is the process of splitting text into smaller units called tokens. These tokens can be words, subwords, or characters, depending on the level of granularity required for the NLP task.

Challenges in Tokenization

Handling Punctuation: Deciding whether to keep or discard punctuation marks.

Dealing with Contractions: Splitting contractions correctly (e.g., "don't" → "do", "n't").

Multi-word Expressions: Identifying and treating multi-word expressions as single tokens (e.g., "New York").

Stemming and Lemmatization

Stemming is the process of reducing words to their base or root form, typically by removing suffixes. The goal is to group different forms of a word so they can be analyzed as a single item.

Common Stemming Algorithms

Porter Stemmer: Uses a series of rules to iteratively trim suffixes.

Snowball Stemmer: An improved version of the Porter Stemmer, offering more extensive rules.

Lancaster Stemmer: A more aggressive stemmer compared to Porter and Snowball.

Lemmatization also reduces words to their base or root form, but unlike stemming, it considers the context and returns the lemma, which is a valid word in the language. Lemmatization uses vocabulary and morphological analysis.

Example "running" → "run" (verb) "better" → "good" (adjective)

Lemmatization is generally preferred over stemming in applications requiring high accuracy and context understanding.

Part-of-Speech (POS) Tagging

POS tagging is the process of assigning a part of speech to each word in a sentence. Parts of speech include nouns, verbs, adjectives, adverbs, etc. POS tagging helps in understanding the syntactic structure of the text.

POS Tagging Algorithms

Rule-Based Tagging: Uses a set of hand-written rules.

Statistical Tagging: Uses machine learning models trained on labeled data.

Neural Network Tagging: Uses deep learning models like Bi-LSTMs to capture context from both directions.

Named Entity Recognition (NER)

NER is the task of identifying and classifying named entities in text into predefined categories such as persons, organizations, locations, dates, etc.

NER Techniques

Rule-Based Approaches: Use hand-crafted rules and dictionaries.

Deep Learning Approaches: Use models like BERT, which leverage large pre-trained language models.

Machine Learning Approaches: Train models on annotated data.

CRFs: Commonly used for sequence labeling tasks like NER.

BiLSTM-CRF: Combines BiLSTM for feature extraction and CRF for sequence labeling.

Parsing

Parsing involves analyzing the grammatical structure of a sentence to produce a parse tree. The parse tree represents the syntactic structure according to a formal grammar.

All of these stages help in extracting meaningful features from text, which can be used for various NLP applications such as machine translation, sentiment analysis, and information extraction (IE).

Attention Mechanism in NLP

The attention mechanism is a powerful concept in neural networks, particularly in the context of NLP. It was introduced to address some limitations of sequence models like RNNs and LSTMs, especially when dealing with long sequences.

In traditional sequence models, such as RNNs and LSTMs, each output word in a sequence is typically produced based on a fixed-length context vector (often the hidden state of the network). This context vector is intended to summarize all the information about the input sequence. However, this approach can be problematic for long sequences, as it forces the network to compress all the necessary information into a single fixed-size vector, which can lead to the loss of important details.

The attention mechanism addresses this issue by allowing the model to dynamically focus on different parts of the input sequence when generating each part of the output sequence. This approach enables the model to make more informed and accurate predictions, especially for long and complex sequences.

How Attention Works

The attention mechanism involves computing a set of weights that represent the relevance of different parts of the input sequence for each element of the output sequence. These weights are then used to create a weighted sum of the input representations, which serves as a dynamic context vector.

Components

Query (Q): The element for which we want to calculate the attention. In sequence-to-sequence models, the query is usually the hidden state of the decoder at a particular time step.

Key (K): The elements that the query will attend to. In sequence-to-sequence models, the keys are typically the hidden states of the encoder.

Value (V): The elements that will be combined according to the attention weights. In sequence-to-sequence models, the values are also the hidden states of the encoder.

Attention Score Calculation

The attention mechanism computes a score that quantifies how well each key matches the query.

- Dot-Product Attention:

$$score(Q,K) = Q \cdot K$$

- Scaled Dot-Product Attention:

$$score(Q,K) = Q \cdot K / sqrt(d_k)$$

d_k is dimensionality of key vectors

- Additive (Bahdanau) Attention: The score is computed using a feedforward network with a single hidden layer.

$$score(Q,K) = V_a^T \, tanh(W_a \, [Q;K])$$

where W_a and V_a are learnable parameter matrices, and $[Q;K]$ denotes the concatenation of the query and key.

Attention Weights

The attention scores are converted to attention weights using the softmax.

$$\alpha_{ij} = exp(score(Q_i,K_j)) \, / \, [\sum_k exp(score(Q_i,K_k))]$$

where α_{ij} is the attention weight corresponding to the $i\text{-}th$ query and $j\text{-}th$ key.

Context Vector

The context vector is a weighted sum of the value vectors, where the weights are the attention weights computed in the previous step.

$$c_i = \sum_j \alpha_{ij} \, V_j$$

This context vector dynamically captures the relevant parts of the input sequence for each query.

Benefits of Attention

Improved Performance: significantly improves the performance of NLP models on tasks that require understanding and generating long sequences.

Interpretability: attention weights provide insights into which parts of the input the model is focusing on, making the model's decisions more interpretable.

Flexibility: Attention can be easily integrated into various neural network architectures and can be used with different types of data, such as text, images, and audio.

Updated Plan of Action

- **Week 5:** Attention is all you need, Transformer architecture
- **Week 6:** Study some pre-trained language models like BERT, GPT, and their variants.
- **Week 7:** Advanced LLM Techniques : Explore techniques for model optimization and efficient training (e.g., pruning, quantization).
- **Week 8:** Choose a project integrating multiple aspects of NLP and LLMs

  End Term Report