Natural Language Processing Fundamentals: Revision Notes

IIT Ropar - Minor in AI

28th April, 2025

1 Introduction to Natural Language Processing

Natural Language Processing (NLP) bridges the gap between human language and computer understanding. It involves developing algorithms and models that enable computers to process, analyze, and generate human language. The field combines linguistics, computer science, and artificial intelligence to create systems that can understand, interpret, and generate meaningful text.

1.1 Core Components of NLP

- **Text Processing**: Tokenization, stemming, lemmatization
- **Syntactic Analysis**: Part-of-speech tagging, parsing
- **Semantic Analysis**: Word sense disambiguation, semantic role labeling
- **Sequence Processing**: Handling text as ordered sequences for prediction and generation

2 Sequence Processing in NLP

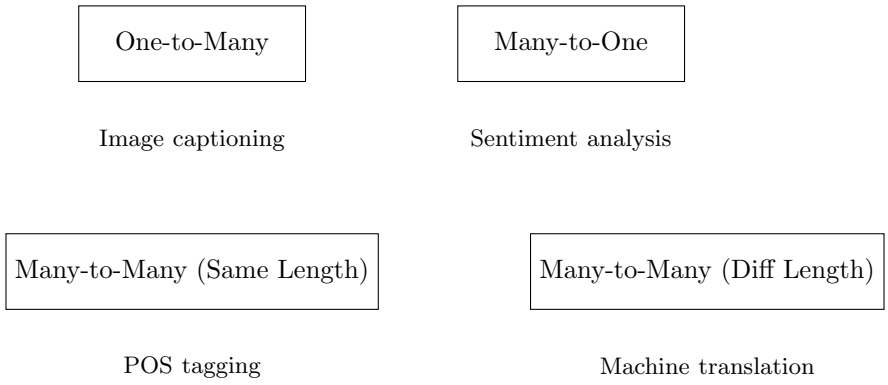
2.1 Importance of Sequences

Text is inherently sequential, with meaning derived not just from individual words but from their order and relationships. Traditional machine learning approaches that treat inputs as independent features fail to capture these sequential dependencies.

Key Insight

In NLP, the order of words matters significantly. The sentences "Dog bites man" and "Man bites dog" contain identical words but convey entirely different meanings due to sequence.

2.2 Sequence-Related Problem Types



- **One-to-Many**: Single input generates a sequence output
- **Many-to-One**: Sequence input produces a single output
- **Many-to-Many (Same Length)**: Input and output sequences have identical length
- **Many-to-Many (Different Length)**: Input and output sequences can vary in length

3 Word Embeddings

Word embeddings represent words as dense vectors in a continuous vector space, where semantic relationships between words are captured by their relative positions.

3.1 Key Embedding Techniques

- **Word2Vec**: Creates word vectors using either:
  - *Continuous Bag of Words (CBOW)*: Predicts a target word from surrounding context words
  - *Skip-gram*: Predicts surrounding context words from a target word
- **GloVe (Global Vectors)**: Combines global matrix factorization and local context window methods
- **FastText**: Extends Word2Vec by treating each word as composed of character n-grams

Example: Word Similarity in Embeddings

In a well-trained embedding space:

$$\vec{king} - \vec{man} + \vec{woman} \approx \vec{queen}$$

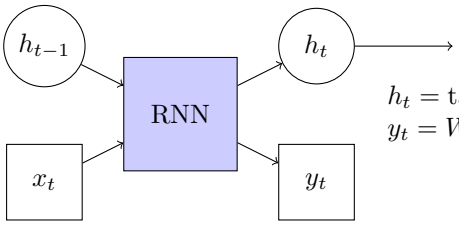
This demonstrates how embeddings capture semantic relationships and analogies between words.

4 Recurrent Neural Networks (RNNs)

For More details and images feel free to refer the Stanford RNN cheat-sheet.

RNNs are designed specifically for processing sequential data by maintaining a hidden state that captures information from previous time steps.

4.1 Basic RNN Architecture



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$
$$y_t = W_{hy}h_t + b_y$$

4.2 Parameter Calculation in RNN Models

For an RNN with input dimension d_{in}, hidden dimension d_h, and output dimension d_{out}:

$$\begin{aligned}\text{Number of parameters} &= \underbrace{d_h \times d_h + d_{in} \times d_h + d_h}_{\text{Hidden state computation}} + \underbrace{d_h \times d_{out} + d_{out}}_{\text{Output computation}} \\ &= d_h(d_h + d_{in} + d_{out} + 1) + d_{out}\end{aligned}$$

Example: Calculating RNN Parameters

For an RNN with:

- Input dimension: 300 (word embedding size)
- Hidden dimension: 128 neurons
- Output dimension: 10000 (vocabulary size)

Total parameters $= 128 \times 128 + 300 \times 128 + 128 + 128 \times 10000 + 10000$
$= 16384 + 38400 + 128 + 1280000 + 10000$
$= 1,344,912$ parameters

4.3 Variants of RNNs

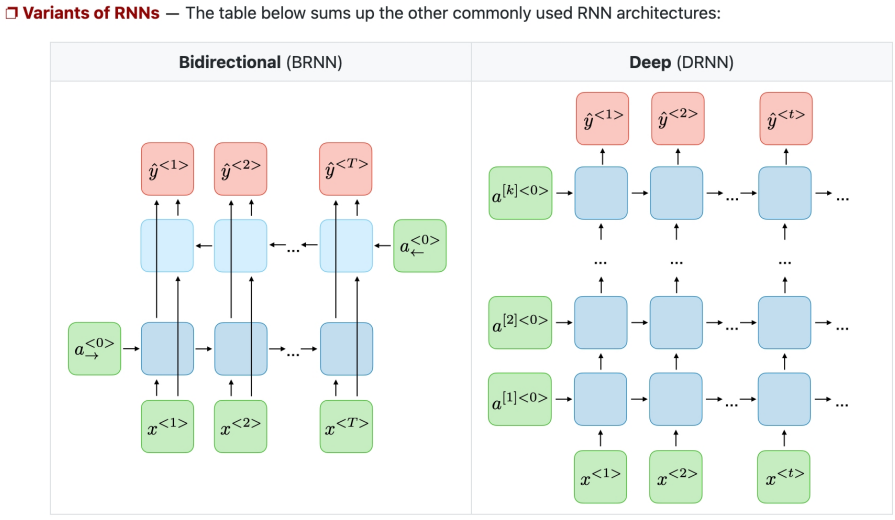


Figure 1: RNN Variants from the Stanford RNN cheat-sheet.

- **LSTM (Long Short-Term Memory)**: Addresses the vanishing gradi-

ent problem using a more complex cell structure with gates

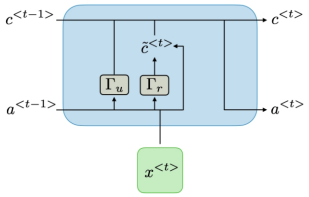
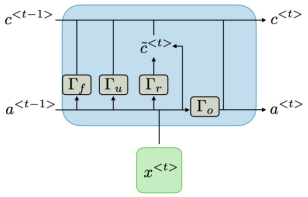
$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
\tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
h_t &= o_t * \tanh(C_t)
\end{aligned}
$$

- **GRU (Gated Recurrent Unit)**: Simplified version of LSTM with fewer parameters

$$
\begin{aligned}
z_t &= \sigma(W_z \cdot [h_{t-1}, x_t]) \\
r_t &= \sigma(W_r \cdot [h_{t-1}, x_t]) \\
\tilde{h}_t &= \tanh(W \cdot [r_t * h_{t-1}, x_t]) \\
h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t
\end{aligned}
$$

Type of gate	Role	Used in
Update gate Γ_u	How much past should matter now?	GRU, LSTM
Relevance gate Γ_r	Drop previous information?	GRU, LSTM
Forget gate Γ_f	Erase a cell or not?	LSTM
Output gate Γ_o	How much to reveal of a cell?	LSTM

❒ **GRU/LSTM** — Gated Recurrent Unit (GRU) and Long Short-Term Memory units (LSTM) deal with the vanishing gradient problem encountered by traditional RNNs, with LSTM being a generalization of GRU. Below is a table summing up the characterizing equations of each architecture:

Characterization	**Gated Recurrent Unit** (GRU)	**Long Short-Term Memory** (LSTM)
$\tilde{c}^{<t>}$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$
$c^{<t>}$	$\Gamma_u \star \tilde{c}^{<t>} + (1 - \Gamma_u) \star c^{<t-1>}$	$\Gamma_u \star \tilde{c}^{<t>} + \Gamma_f \star c^{<t-1>}$
$a^{<t>}$	$c^{<t>}$	$\Gamma_o \star c^{<t>}$
Dependencies		

Remark: the sign ⋆ denotes the element-wise multiplication between two vectors.

Figure 2: LSTM vs GRU from the Stanford RNN cheat-sheet.

5 Auto-regressive Models

Auto-regressive models predict the next element in a sequence based on the preceding elements, making them fundamental for text generation tasks.

$$P(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} P(x_t | x_1, \ldots, x_{t-1})$$

5.1 Language Modeling

Language modeling involves assigning probabilities to sequences of words, allowing systems to predict likely continuations of text.

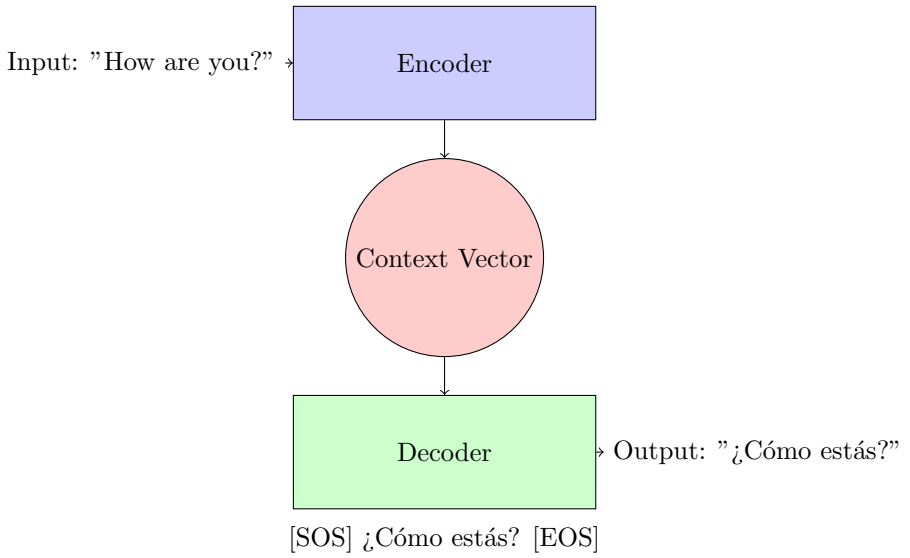
Example: Next Word Prediction

Given the sentence fragment: "The chef cooked the..."
An auto-regressive language model might predict:

- "meal" (probability: 0.35)
- "food" (probability: 0.25)
- "dinner" (probability: 0.15)
- "pasta" (probability: 0.10)
- "..." (other possibilities)

6 Sequence-to-Sequence Models

Sequence-to-sequence (Seq2Seq) models handle the translation of one sequence to another, often with different lengths. They typically consist of an encoder and a decoder.



6.1 Encoder-Decoder Framework

- **Encoder**: Processes the input sequence and compresses it into a context vector
- **Decoder**: Generates the output sequence based on the context vector
- **Special Tokens**:
  - **SOS (Start of Sentence)**: Signals the beginning of generation
  - **EOS (End of Sentence)**: Signals the end of generation

6.2 Attention Mechanism

Attention allows the decoder to focus on different parts of the input sequence when generating each output token, addressing the information bottleneck in the basic encoder-decoder model.

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

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

$$\text{where } e_{ij} = \text{score}(s_{i-1}, h_j)$$

Example: Attention in Translation

When translating "The cat sat on the mat" to Spanish:

- When generating "gato" (cat), attention focuses heavily on "cat" in the input
- When generating "sentó" (sat), attention shifts to "sat"
- This dynamic focus improves translation quality, especially for longer sentences

7 Evaluating Machine Translation: BLEU Score

The Bilingual Evaluation Understudy (BLEU) score quantifies the quality of machine translation by comparing it to reference human translations.

7.1 BLEU Score Calculation

1. **Precision Calculation**: Compare n-grams in the candidate translation to reference translations

$$P_n = \frac{\sum_{C \in \{\text{Candidates}\}} \sum_{n\text{-gram} \in C} \text{Count}_{\text{clip}}(n\text{-gram})}{\sum_{C' \in \{\text{Candidates}\}} \sum_{n\text{-gram}' \in C'} \text{Count}(n\text{-gram}')}$$

2. **Brevity Penalty**: Penalizes translations that are too short

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases}$$

   where c is candidate length and r is reference length.

3. **Final BLEU Score**:

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log P_n\right)$$

   where w_n are weights typically set to $w_n = 1/N$

Example: BLEU Score Calculation

Reference: "The cat is sitting on the mat."
Candidate: "The cat sits on the mat."
Unigram Precision:

- Matched unigrams: "The", "cat", "on", "the", "mat" (5 out of 6)
- $P_1 = 5/6 \approx 0.833$

Bigram Precision:

- Matched bigrams: "The cat", "on the", "the mat" (3 out of 5)
- $P_2 = 3/5 = 0.6$

Brevity Penalty:

- Reference length: 7 words
- Candidate length: 6 words
- $BP = e^{(1-7/6)} \approx 0.846$

BLEU-2 Score (using only uni-grams and bi-grams):

- $BLEU = 0.846 \times \exp((0.5 \times \ln(0.833) + 0.5 \times \ln(0.6)))$
- $BLEU = 0.846 \times \exp(0.5 \times (-0.183) + 0.5 \times (-0.511))$
- $BLEU = 0.846 \times \exp(-0.347) \approx 0.846 \times 0.707 \approx 0.598$

So the BLEU-2 score is approximately 0.598 or 59.8%.

8 Modern NLP Architectures

8.1 Transformer Architecture

Transformers have largely replaced RNNs in state-of-the-art NLP systems, using multi-head self-attention mechanisms. The Transformer – a model that uses attention to boost the speed with which these models can be trained. The Transformer outperforms the Google Neural Machine Translation model in specific tasks. The biggest benefit, however, comes from how The Transformer lends itself to parallelization.

The encoding component is a stack of encoders (the paper stacks six of them on top of each other – there's nothing magical about the number six, one can definitely experiment with other arrangements). The decoding component is a stack of decoders of the same number.

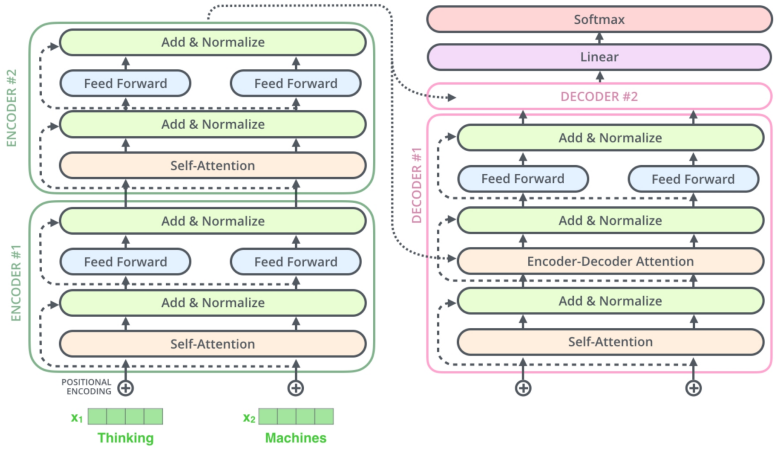


Figure 3: Transformer Architecture

8.2 Pre-trained Language Models

Modern NLP systems often use transfer learning with pre-trained language models that are fine-tuned for specific tasks.

- **BERT (Bidirectional Encoder Representations from Transformers)**
- **GPT (Generative Pre-trained Transformer)**
- **T5 (Text-to-Text Transfer Transformer)**
- **RoBERTa (Robustly Optimized BERT Pretraining Approach)**

Example: Fine-tuning for Sentiment Analysis

A pre-trained BERT model with 110 million parameters can be fine-tuned on just 25,000 movie reviews to achieve 95% accuracy on sentiment classification, demonstrating the power of transfer learning in NLP.

9 Practical Applications of NLP

- **Machine Translation**: Google Translate, DeepL
- **Sentiment Analysis**: Social media monitoring, customer feedback analysis
- **Named Entity Recognition**: Extracting people, organizations, locations from text

- **Question Answering**: Virtual assistants, customer support bots
- **Text Summarization**: News article condensation, report summarization
- **Speech Recognition**: Voice assistants, transcription services
- **Text Generation**: Creative writing, content creation, code generation

10 Key Takeaways

Summary Points

- **Sequence Processing**: The foundation of NLP, handling text as ordered sequences
- **Word Embeddings**: Dense vector representations capturing semantic relationships
- **RNN Architecture**: Specialized neural networks designed for sequential data
- **Seq2Seq Models**: Encoder-decoder frameworks for translation and generation tasks
- **Attention Mechanisms**: Methods to focus on relevant input parts during decoding
- **BLEU Score**: Standard metric for evaluating machine translation quality
- **Modern Architectures**: Transformers and pre-trained models dominate current NLP