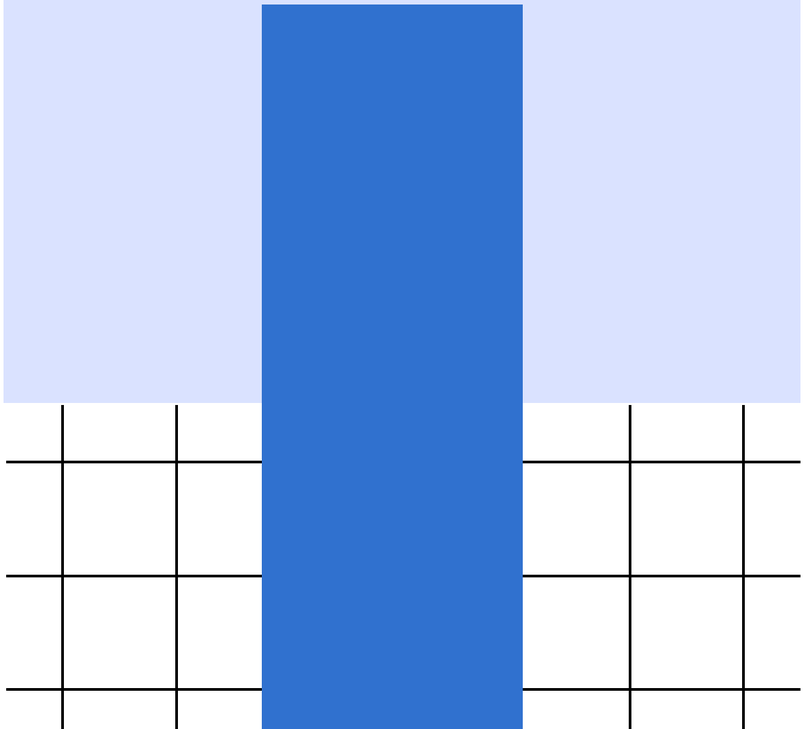
INTRO NATURAL LANGUAGE PROCESSING (NLP)

– Matee Vadrukchid –



Big Picture of NLP

Big picture of NLP

Preprocessing -> Embeddings -> Modeling -> Inference

Tokenization
- Normal
- Byte-Pair Encoding
- WordPiece
- Unigram

Numericalization

- Word2Vec
- GloVe
- FastText
- ELMo
- BERT
- SentenceBERT
- SimCSE

Supervised learning
- Machine learning
- Deep learning
  - RNN
  - LSTM
  - CNN
  - LSTM with attention
- Transformers

Self-supervised learning
- BERT - Masked LM
- GPT - LM
- BART,T5, Pegasus - LM

Transfer learning
- LLM+ Finetuning / Instruction tuning

Prompt-based learning
- LLM + prompt-engineering

Augmented-retrieval
- LLM + retrieval

Optimization Human Preference learning
- LLM + RLHF

Multimodal learning
- LLM + Visual encoder

Low Compute
- **Pruning**
- **Distillation**
- **Quantization**
- **Parameter Efficient Finetuning (PEFT)**
  - Adapter
  - Prompt-Tuning
  - Prefix-Tuning
  - LoRA

Probability-based
- Perplexity

Word overlap-based
- BLEU
- ROUGE

Model-based
- BERTScore
- BLEURT

LLM-based
- AlpacaEval
- AlpacaFarm

Human Evaluation

TASKS
- *Industry* - **Sequence Tagging** (Name Entity Recognition, Part of Speech), **Sentence classification** (sentiment analysis, topic classification), **Image to text (OCR)**, Closed-ended Question Answering, Code Generation, Machine translation **ChatBot** (Retrieval)
- *Research* - Summarization, Intent Detection, Open-ended Question Answering, Coreference Resolution, Music/Story/Poetry Generation (e.g., poetry), Natural language Inference, ASR, VQA

TOOLS
- **SpaCy** - basic stuffs - NER, dependency parsing, pos tagging
- **PyTorch** - designing models
- **Huggingface** - pretrained LLM and public datasets
- **Langchain** - toolkit to build context-aware, reasoning LLM applications
- **Deployment** - FastAPI

TOP VENUES
1. *ACL, NIPS, ICML, ICLR*
2. *EMNLP, NAACL, EACL*
3. *TACL, Computational Linguistics, Computer Speech and Language*

1. Introduction and Outline

1.1. What We're Building

- A **mini conversation generator** that responds to simple inputs.
- Example conversation lines:
    1. **User**: "Hello, how are you?"
       Bot: "I'm thank you and you?"
    2. **User**: "I'm good." **Bot**: "Great!"
- We'll store a small set of **20 or so** conversation lines (like QA pairs).
- We'll build a neural network model that, given an **input sentence**, generates the **output sentence** token by token, starting with a **special start token** and ending with a **special end token**.

1. Introduction and Outline

1.2. Lesson Plan

1. **Theory**
    - **Word → Vector** (Vocabulary, tokens, embeddings)
    - **Sequence-to-sequence** modeling
    - **Transformer** basics
2. **Dataset & Vocabulary**
3. **Tokenization & Special Tokens** (`<sos>` , `<eos>` , `<pad>`)
4. **Simple Transformer Model**
5. **Training**
6. **Inference** (Generating new responses)
7. **Assignments** for deeper practice

2. Theory

2.1. Word → Vector (Embeddings)

- **Why do we need word vectors?**
  - Neural networks operate on numbers, not text.
  - We convert words to dense vectors (embeddings) so that semantically similar words have similar vector representations.
- **Vocabulary**
  - We create a set (or list) of known words from our conversation data (e.g., “hello”, “how”, “are”, “you”, “i’m”, etc.).
  - Each unique word is given a **token ID** (e.g., word `"hello"` → token ID `4`).

2. Theory

- **Adding Special Tokens**
  - `<sos>` : start of sentence.
  - `<eos>` : end of sentence.
  - `<pad>` : for padding to a fixed length.
  - `<unk>` : unknown words not in our vocab.

2. Theory

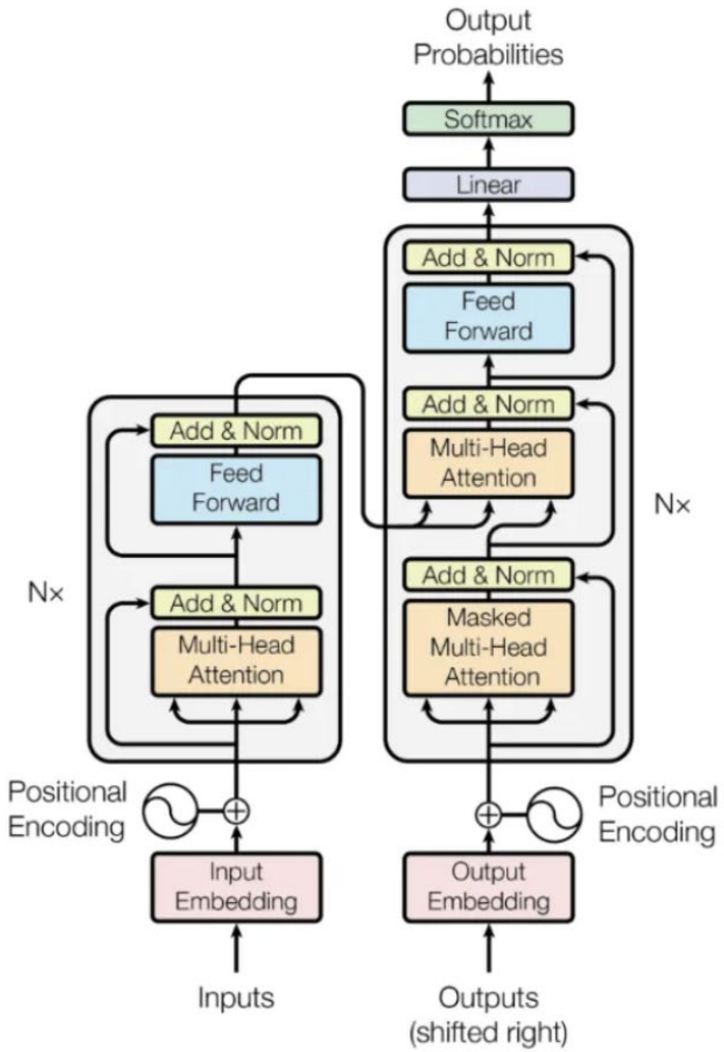
2.2. Sequence-to-Sequence (Seq2Seq)

- **Concept**: We feed an **input sentence** to an **encoder** which processes it into a hidden representation. A **decoder** then processes that representation (and previous tokens it generated) to **predict the next token**.
- **Key**: The decoder continues generating tokens until it outputs `<eos>`.

2. Theory

2.3. Transformer Basics

- Uses **attention** to see different parts of the sentence simultaneously.
- Works well for translations, conversation generation, etc.
- **Encoder**: reads input sequence.
- **Decoder**: generates output sequence, one token at a time, while looking at the encoder's hidden representation.
- We'll implement a **simplified** version for demonstration.



Link:https://medium.com/@sayedebad.777/building -a-transformer -from -scratch-a-step-by-step-guide-a3df0aeb7c9a