Lecture 1: Introduction to Natural Language Processing (NLP)

CS11-711 Advanced NLP
Instructor: Graham Neubig Spring 2024

This is the course note taken from the CMU course lecture (by Graham Neubig) for CMU CS 11-711, Advanced NLP (Spring 2024).

Note taker: Yangming Li

1 Introduction to NLP

1.1 What is NLP?

Natural Language Processing (NLP) is a subfield of Artificial Intelligence (AI) that focuses on the interaction between computers and human language. NLP enables machines to understand, interpret, generate, and manipulate human language.

1.2 NLP Applications

- **Human-Machine Communication:**
  - Chatbots and Virtual Assistants (e.g., Siri, Alexa, ChatGPT)
  - Question Answering Systems
  - Dialogue Systems
  - Code Generation (e.g., Copilot, Code Llama)
- **Human-Human Communication:**
  - Machine Translation (Google Translate, DeepL)
  - Spell Checking and Grammar Correction
  - Assisted Writing (e.g., Grammarly, AI-powered text suggestions)
- **Language Analysis & Understanding:**
  - Syntactic Analysis

- Text Classification
- Named Entity Recognition (NER) and Linking

1.3 Importance of NLP

NLP is integrated into applications such as Google Docs, email auto-correct, and search engines. ChatGPT and similar models can answer complex queries, though reliability may vary. Machine translation has improved significantly, but low-resource languages still face challenges.

2 Overview of NLP System Building

2.1 General Framework

An NLP system maps an input X to an output Y, where:

- X and/or Y involve human language.
- Example tasks:

Input (X)	**Output (Y)**	**NLP Task**
Text	Text in another language	Translation
Text	Continuing text	Language Modeling
Text	Classification label	Text Classification
Text	Linguistic structure	Syntactic Analysis
Image	Text description	Image Captioning

Table 1: Common NLP Tasks

3 Methods for Creating NLP Systems

- **Rule-based Systems:** Manually created rules.
- **Prompting:** Using a pre-trained language model without additional training.
- **Fine-Tuning:** Training a model on paired input-output examples.

4 Example Task: Sentiment Analysis

Given a movie review (X), determine its sentiment (Y):

- Positive (1)
- Neutral (0)
- Negative (-1)

Input Sentence	Expected Sentiment
'I hate this movie.'	Negative (-1)
'I love this movie.'	Positive (1)
'I saw this movie.'	Neutral (0)

Table 2: Sentiment Classification Examples

5 Rule-Based Approach

5.1 Steps

1. **Feature Extraction**: Extract sentiment-related words.
2. **Score Calculation**: Count occurrences of positive/negative words.
3. **Decision Function**: Assign sentiment based on score.

5.2 Example Code

Listing 1: Rule-Based Sentiment Classifier

```
def classify(x: str) -> int:
    good_words = ["love", "excellent", "great"]
    bad_words = ["hate", "awful", "terrible"]
    score = sum([1 for word in x.split() if word in good_words]) - \
            sum([1 for word in x.split() if word in bad_words])
    return 1 if score > 0 else (-1 if score < 0 else 0)
```

6 Machine Learning Approach

6.1 Bag of Words (BoW) Model

Each word is assigned a weight based on its contribution to sentiment.

Word	Feature Weight
Love	+2.4
Hate	-3.5

Table 3: Word Weights in Bag-of-Words Model

6.2 Training Algorithm: Perceptron

Listing 2: Perceptron Training Algorithm

```
feature_weights = {}
for x, y in data:
    features = extract_features(x)
    predicted_y = run_classifier(features)
    if predicted_y != y:
        for feature in features:
            feature_weights[feature] = feature_weights.get(feature, 0)
```

7 Roadmap for the Course

- **Language Modeling:** Sequence encoding, Transformer architecture.
- **Training Methods:** Fine-tuning, Reinforcement Learning.
- **Evaluation:** Debugging, Bias, Fairness.
- **Advanced Architectures:** Retrieval-Augmented Generation (RAG), Distillation.
- **Applications:** Code Generation, Knowledge-based QA.
- **Linguistics & Multilingual NLP:** Handling multiple languages efficiently.

Lecture 2: Word Representation and Text Classification

CS11-711 Advanced NLP
Instructor: Graham Neubig Spring 2024

---

8 Introduction

This lecture lays the foundation for more advanced NLP models by discussing methods for representing words and building text classifiers. We begin by reviewing the bag-of-words model, then address its limitations and introduce modern solutions such as subword models, continuous word embeddings, and neural network approaches.

9 Review of the Bag-of-Words Model

The *bag-of-words* (BoW) model represents each word in a vocabulary as a one-hot vector. For an input document, the model:

- Splits the document into words.
- Represents each word as a one-hot vector.
- Aggregates these one-hot vectors (typically by summing) to form a frequency vector.
- Multiplies the frequency vector by a weight vector (or weight matrix) to obtain a classification score.

However, BoW has several limitations:

- **Handling Conjugated/Compound Words:** Variants such as "company" versus "companies" are treated as different tokens.
- **Lack of Similarity:** The model does not capture the semantic similarity between related words.
- **Combination Features:** Important multiword expressions (e.g., "don't love" vs. "love") are not modeled.

- **Sentence Structure:** BoW ignores word order and syntactic structure.

10 Subword Models

10.1 Motivation

To address BoW limitations, modern NLP systems use *subword models*. By splitting less common words into multiple subword tokens, these models:

- Share parameters between similar word forms (e.g., "company" and "companies").
- Reduce the overall vocabulary size, which saves computational resources.

A key challenge is striking a balance between expressiveness (longer tokens) and frequency (shorter tokens occurring more often).

10.2 Byte Pair Encoding (BPE)

BPE is a greedy algorithm to construct a subword vocabulary:

1. **Initialization:** Begin with all characters (or bytes) and a special end-of-word symbol.
2. **Merging:** Compute frequencies of all adjacent token pairs in the corpus. Merge the most frequent pair (e.g., merging `e` and `s` to form `es`).
3. **Iteration:** Continue merging until reaching the desired vocabulary size (e.g., 60,000 tokens).

10.3 Unigram Models

Alternatively, a unigram language model can be used for subword segmentation:

- The model treats the probability of a token sequence as the product of individual token probabilities.
- It selects a vocabulary that maximizes the likelihood of the training corpus.
- Optimization is typically performed with the Expectation-Maximization (EM) algorithm combined with dynamic programming.

The `SentencePiece` library implements both the BPE and unigram approaches and supports additional techniques such as *subword regularization*.

11 Continuous Word Embeddings

11.1 Definition and Motivation

Unlike one-hot vectors, continuous word embeddings represent words as dense vectors in a fixed-dimensional space. The lookup operation is equivalent to multiplying by a one-hot vector:

$$\mathbf{e}_w = E \cdot \mathbf{1}_w,$$

where $E \in \mathbb{R}^{d \times V}$ is the embedding matrix, V is the vocabulary size, and d is the embedding dimension.

11.2 Continuous Bag-of-Words (CBOW)

In the CBOW model:

1. The embeddings for all context words are looked up.
2. The embeddings are summed (or averaged) to form a single dense vector.
3. This vector is passed through a transformation (often a linear layer) to compute classification scores.

For example, given context words $w_1, \ldots, w_n$, the target word is predicted as:

$$v_{w_t} = \frac{1}{n} \sum_{i=1}^{n} v_{w_i}.$$

11.3 Deep Continuous Bag-of-Words

A deeper model can learn *combination features* by passing the summed embeddings through additional layers:

$$h = \tanh(W_1 x + b_1), \tag{1}$$
$$y = W_2 h + b_2. \tag{2}$$

12 Training Word Embeddings

12.1 Gradient Descent

Training involves minimizing a loss function by updating parameters via gradient descent:

$$\theta \leftarrow \theta - \eta \nabla_\theta L,$$

where η is the learning rate.

12.2 Loss Functions

Two common loss functions in binary classification are:

- **Hinge Loss:**
$$L_{\text{hinge}}(s, y) = \max\{0, 1 - y \cdot s\},$$
- **Sigmoid + Negative Log-Likelihood (NLL):**
$$\sigma(s) = \frac{1}{1 + e^{-s}}, \quad L_{\text{NLL}}(s, y) = -\log \sigma(y \cdot s).$$

13 Neural Networks and Computation Graphs

Modern neural networks are implemented as computation graphs:

- **Nodes** represent variables or operations.
- **Edges** indicate the flow of data.

Forward propagation computes node values in topological order, while *backward propagation* computes gradients in reverse order.

14 Optimization Techniques

14.1 Adam Optimizer

Adam is widely used for training modern models such as Transformers:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{3}$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \tag{4}$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}, \tag{5}$$
$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t. \tag{6}$$

15 Visualization of Word Embeddings

Techniques such as PCA and t-SNE are used to reduce the dimensionality of word embeddings for visualization.

Lecture 3: Language Modeling

CS11-711 Advanced NLP
Instructor: Graham Neubig Spring 2024

---

16 Introduction to Language Modeling

Language modeling (LM) is the task of estimating the probability distribution over sequences of words. It is a fundamental building block in NLP and is used for:

- **Sentence Scoring:** Evaluating the fluency or likelihood of a sentence.
- **Text Generation:** Sampling from the probability distribution to generate coherent text.
- **Text Classification:** Scoring candidate texts conditioned on a label.
- **Grammar Correction:** Identifying low-probability words and replacing them.
- **Speech Recognition & Machine Translation:** Predicting sequences in spoken language or translating between languages.

This lecture discusses the probabilistic formulation, various modeling techniques (from count-based n-gram models to neural LMs), evaluation metrics, calibration, and efficiency concerns.

17 Generative vs. Discriminative Models

- **Discriminative models** compute $P(Y|X)$ and are typically used in tasks like sentiment analysis.
- **Generative models** compute $P(X)$ or the joint probability $P(X, Y)$ and are fundamental to language modeling.

Generative models can also be extended to multimodal tasks (e.g., combining text and images) but here we focus on text. In LM, we use the chain rule to decompose the probability of a sequence.

18 Probabilistic Language Models

A language model assigns a probability to a sequence $X = (x_1, x_2, \ldots, x_n)$ by decomposing it as:

$$P(X) = \prod_{i=1}^{n} P(x_i | x_1, \ldots, x_{i-1}).$$

This decomposition reduces the problem of modeling an exponential number of sequences (if attempted directly) into n predictions over a vocabulary of size V.

19 Applications of Language Models

Beyond text generation and scoring, language models are used for:

- **Multiple Choice Question Answering:** For example, scoring candidate answers to the question "Where is CMU located?" by computing the likelihood of each possibility.
- **Chain-of-Thought Generation:** Generating explanations that lead to an answer—although intermediate reasoning makes direct probability evaluation of the final answer challenging.
- **Paraphrasing and Grammar Correction:** Suggesting alternatives when low-probability words are detected.

20 Auto-regressive Language Models

Auto-regressive models generate text one token at a time:

$$P(X) = \prod_{i=1}^{n} P(x_i | x_1, \ldots, x_{i-1}).$$

They typically generate text left-to-right (as in English), but in languages like Arabic or Hebrew, a right-to-left generation may be more natural. This sequential generation is mathematically exact (via the chain rule) and avoids the combinatorial explosion of predicting an entire sequence at once.

21 Unigram Language Models

A unigram model assumes each word is independent of its context:

$$P(x_i|x_1, \ldots, x_{i-1}) \approx P(x_i).$$

Using maximum likelihood estimation:

$$P_{MLE}(x_i) = \frac{c(x_i)}{\sum_x c(x)},$$

where $c(x_i)$ is the count of x_i in the training data. However, unigram models:

- Ignore word dependencies.
- Suffer from the *unknown word problem*: if a word has never been seen, its probability is zero.
- Are often parameterized in log space to prevent numerical underflow when multiplying many small probabilities.

22 Higher-order n-gram Models

To capture context, we use n-gram models:

$$P(x_i|x_{i-n+1}, \ldots, x_{i-1}) = \frac{c(x_{i-n+1}, \ldots, x_i)}{c(x_{i-n+1}, \ldots, x_{i-1})}.$$

22.1 Smoothing and Interpolation

Because many n-grams may have zero counts in the training data:

- **Additive Smoothing:**

$$P(x_i|x_{i-n+1}, \ldots, x_{i-1}) = \frac{c(x_{i-n+1}, \ldots, x_i) + \alpha}{c(x_{i-n+1}, \ldots, x_{i-1}) + \alpha|V|}.$$

- **Kneser-Ney Smoothing:** Discounts counts by a factor d and interpolates with a lower-order model:

$$P(x_i|x_{i-n+1}, \ldots, x_{i-1}) = \frac{c(x_{i-n+1}, \ldots, x_i) - d}{c(x_{i-n+1}, \ldots, x_{i-1})} + \lambda P(x_i|x_{i-n+2}, \ldots, x_{i-1}).$$

The interpolation weight λ is chosen so that when the higher-order counts are low, the model relies more on lower-order statistics.

23 Evaluation of Language Models

Common evaluation metrics include:

- **Log-Likelihood:** For a test corpus X_{test},

$$LL(X_{\text{test}}) = \sum_{X \in X_{\text{test}}} \log P(X).$$

- **Perplexity:** Related to the cross-entropy $H(X_{\text{test}})$,

$$PPL(X_{\text{test}}) = 2^{H(X_{\text{test}})} \quad \text{or} \quad PPL(X_{\text{test}}) = e^{-WLL(X_{\text{test}})},$$

  where WLL is the per-word log-likelihood.

Perplexity intuitively represents the average number of guesses the model would need to predict the next word. (It is essential to use a consistent tokenization scheme when comparing perplexities.)

24 Calibration of Language Models

Calibration measures whether a model's predicted probabilities match the true likelihood of correctness. For example, if a model assigns a probability of 0.7 to a prediction, then—ideally—70% of such predictions should be correct. A reliability diagram plots the model's expected confidence against its actual accuracy. Overconfident models (often due to overfitting) may have poor calibration even if their overall accuracy is high.

25 Neural Language Models

Neural language models overcome many limitations of n-gram models by learning continuous representations:

- They use word embeddings to share statistical strength among similar words.
- Models such as feed-forward networks, Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Transformer architectures (e.g., BERT, GPT) are common.
- Techniques such as *embedding tying* (sharing parameters between input and output embeddings) help reduce the number of parameters and improve training.

Although neural LMs generally require more data to capture low-frequency events, they excel at modeling long-range dependencies and complex contexts.

26 Efficiency Considerations

Efficiency in language modeling is critical for both training and inference. Some key techniques include:

- **Mini-batching:** Grouping multiple sequences together (often based on the total number of tokens) to take advantage of parallel computation.
- **GPU Acceleration:** Leveraging GPUs for large matrix-matrix multiplications, which are significantly faster than repeated matrix-vector operations.
- **Reducing CPU-GPU Data Movement:** Minimizing memory transfers between host and device.
- **Quantization and Knowledge Distillation:** Reducing model size and improving inference speed.

As discussed in class, while CPUs are efficient for small, frequent operations (like starting a new task), GPUs are far superior when executing large, parallel computations once initialized.

27 Conclusion

In this lecture, we covered:

- The probabilistic formulation of language modeling and the importance of decomposing $P(X)$ using the chain rule.
- The differences between generative and discriminative models.
- Count-based models: unigram and higher-order n-gram models with smoothing/interpolation techniques.
- Evaluation metrics such as log-likelihood, perplexity, and calibration.
- Neural language models that employ embeddings and deep architectures.
- Efficiency considerations crucial for practical training and deployment.

In upcoming lectures, we will explore advanced neural architectures (e.g., Transformers) for language modeling, methods for handling long contexts, and further applications in generation and reasoning.

Further Reading:

- Sennrich, R., Haddow, B., & Birch, A. (2016). *Neural Machine Translation of Rare Words with Subword Units.*
- Kudo, T., & Richardson, J. (2018). *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing.*
- Chen, T., et al. (2020). *Evaluating Large Language Models Trained on Code.*
- Guo, C., Pleiss, G., Sun, Y., & Weinberger, K. Q. (2017). *On Calibration of Modern Neural Networks.*

Lecture 4: Sequence Modeling

CS11-711 Advanced NLP
Instructor: Graham Neubig Spring 2024

---

28 Introduction to Sequence Modeling

Sequence modeling is a fundamental aspect of Natural Language Processing (NLP) because language and many other forms of data are inherently sequential. In this lecture, we explore why sequence modeling is critical, the types of sequential prediction problems encountered in NLP, and the primary architectures used to model sequences. We also discuss recent advances and practical strategies to improve efficiency.

28.1 Motivation

Natural language is structured hierarchically:

- **Characters** form words.
- **Words** form sentences.
- **Sentences** form paragraphs and documents.

Such data exhibit long-distance dependencies—for instance, agreement in gender (*"he" vs. "she"*), selectional preferences, and pronoun resolution (as seen in the Winograd Schema Challenge). Capturing these dependencies is essential for tasks like machine translation, text generation, and sentiment analysis.

29 Types of Sequential Prediction Problems

There are two major prediction paradigms for sequences: unconditioned and conditioned predictions.

29.1 Prediction Types

Sequence prediction problems in NLP include:

- **Binary/Multi-class Classification:** e.g., sentiment analysis.
- **Structured Prediction:** where each token is assigned a label (e.g., part-of-speech tagging, named entity recognition).

29.2 Unconditioned vs. Conditioned Prediction

- **Unconditioned Prediction (Language Modeling):** The model predicts the probability of a sequence X by decomposing it as:

$$P(X) = \prod_{i=1}^{|X|} P(x_i \mid x_1, \ldots, x_{i-1}).$$

- **Conditioned Prediction (Sequence-to-Sequence):** The model predicts an output sequence Y given an input X:

$$P(Y \mid X) = \prod_{i=1}^{|Y|} P(y_i \mid X, y_1, \ldots, y_{i-1}).$$

30 Sequence Models

There are three major families of models used for processing sequential data: Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and Attention Mechanisms. We now explore each in detail.

30.1 Recurrent Neural Networks (RNNs)

RNNs process sequences by maintaining a hidden state that is updated at every time step. A simple RNN is defined by:

$$h_t = f(W\, h_{t-1} + W_x\, x_t + b),$$

where:

- h_t is the hidden state at time t,
- x_t is the input at time t,
- f is a nonlinear activation function (e.g., tanh or ReLU).

30.1.1 Challenges and LSTMs

Vanilla RNNs suffer from the *vanishing gradient problem*—gradients shrink exponentially over long sequences. Long Short-Term Memory (LSTM) networks introduce gating mechanisms to mitigate this:

$$\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \\
\tilde{c}_t &= \tanh(W_c \cdot [h_{t-1}, x_t] + b_c), \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \\
h_t &= o_t \odot \tanh(c_t),
\end{aligned}$$

where c_t is the cell state and $\odot$ denotes element-wise multiplication.

30.2 Convolutional Neural Networks (CNNs)

CNNs apply filters over local windows of the input sequence. For example, with a window size of 3:

$$h_t = f\Big(W \cdot [x_{t-1};\, x_t;\, x_{t+1}] + b\Big),$$

where $[x_{t-1};\, x_t;\, x_{t+1}]$ is the concatenation of neighboring input vectors. CNNs are effective at capturing local patterns and are inherently parallelizable, though they typically require stacking many layers to capture long-range dependencies.

30.3 Attention Mechanisms

Attention mechanisms allow models to weigh contributions from different parts of the sequence. Given a query vector q and key vectors k_i, the attention weights are computed as:

$$\alpha_i = \text{softmax}\left(\frac{q^\top k_i}{\sqrt{d}}\right),$$

where d is the dimensionality of the key vectors. The output is the weighted sum of corresponding value vectors:

$$c = \sum_i \alpha_i v_i.$$

Attention is central to Transformer architectures, which have revolutionized NLP by enabling highly parallel processing and effective modeling of long-range dependencies.

30.3.1 Recent Advances in Attention

Recent developments include:

- **Multi-Head Attention:** Splitting queries, keys, and values into multiple subspaces to capture diverse relations.
- **Sparse and Local Attention:** Reducing the $O(n^2)$ computational cost by limiting the attention to nearby tokens or employing learned sparsity patterns.
- **Relative Positional Encodings:** Allowing models to capture relative rather than absolute positions to improve generalization.

31 Applications of Sequence Models

Sequence models are applied in a variety of NLP tasks:

- **Sequence Encoding:** Converting variable-length sequences into fixed-size vectors for tasks such as retrieval or classification.
- **Sequence Labeling:** Assigning labels to each token (e.g., part-of-speech tagging, named entity recognition).
- **Language Modeling and Text Generation:** Predicting the next token in a sequence for auto-completion, translation, or creative writing.
- **Machine Translation:** Using encoder-decoder architectures with attention to convert text from one language to another.

32 Efficiency Considerations for Sequence Models

Modern sequence modeling must balance representational power with computational efficiency. Key strategies include:

32.1 Mini-Batching

- **Padding and Masking:** When processing multiple sequences, inputs are padded to a common length and a mask is applied to ignore padded tokens during loss computation.

- **Dynamic Batching:** Grouping sequences of similar lengths together to minimize wasted computation.

32.2 GPU Acceleration

- Leveraging GPUs enables large-scale matrix-matrix multiplications that are significantly faster than repeated matrix-vector operations.
- Many modern frameworks (e.g., PyTorch, TensorFlow) optimize these operations to maximize parallel throughput.

32.3 Reducing CPU-GPU Data Movement

- Minimizing memory transfers between host and device is critical; data should be preprocessed and kept on the GPU during training whenever possible.

32.4 Strided Architectures and Sparse Attention

- **Strided Architectures:** Techniques like pyramidal RNNs downsample the sequence length between layers, reducing computational cost.
- **Sparse Attention:** Instead of computing full attention over all tokens, some models restrict the attention to a local window or learn sparse patterns to reduce complexity.

32.5 Truncated Backpropagation Through Time (BPTT)

- For very long sequences, gradients are backpropagated only for a fixed number of time steps. The hidden state is passed forward to maintain context.
- This approximation reduces memory usage and computational cost while still allowing the network to learn long-term dependencies.

32.6 Computational Complexity

Different architectures have distinct time complexities:

- **RNNs:** $O(n)$ per sequence (due to sequential dependency).
- **CNNs:** $O(nW)$, where W is the filter width.

- **Attention Mechanisms:** $O(n^2)$ owing to pairwise token interactions, though parallelism on GPUs can offset this cost.

33 Practical Considerations and Recent Trends

33.1 Pre-trained Models and Fine-Tuning

- Pre-trained language models (e.g., BERT, GPT) leverage large-scale data and are fine-tuned for downstream tasks.
- Transfer learning has become a standard approach in NLP, reducing the need for training large models from scratch.

33.2 Hybrid Architectures

- Many state-of-the-art systems combine multiple sequence modeling techniques (e.g., using CNNs for local feature extraction followed by attention layers for global context).
- Hybrid models often achieve a better balance between efficiency and performance.

33.3 Model Compression Techniques

- **Quantization:** Reducing the precision of model weights to decrease memory usage and speed up inference.
- **Knowledge Distillation:** Training a smaller "student" model to mimic a larger "teacher" model.

34 Conclusion

In this lecture, we covered:

- The motivation behind sequence modeling and its importance in handling hierarchical and long-range dependencies in language.
- The two paradigms of sequence prediction: unconditioned (language modeling) and conditioned (sequence-to-sequence).
- Detailed overviews of three major sequence model families: RNNs (and LSTMs), CNNs, and Attention Mechanisms.

- Recent advances such as multi-head attention, sparse attention, and relative positional encodings.
- Practical efficiency considerations including mini-batching, GPU acceleration, reducing data movement, and model compression techniques.

In upcoming lectures, we will dive deeper into advanced architectures—such as Transformers—and explore methods for handling very long contexts and further applications in generation and reasoning.

Further Reading:

- Bahdanau, D., Cho, K., & Bengio, Y. (2015). *Neural Machine Translation by Jointly Learning to Align and Translate.*
- Vaswani, A., et al. (2017). *Attention is All You Need.*
- Hochreiter, S., & Schmidhuber, J. (1997). *Long Short-Term Memory.*
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.*
- Sanh, V., Wolf, T., & Ruder, S. (2019). *A Hierarchical Multi-task Approach for Learning Embeddings from Semantic Tasks.*

Lecture 5: Detailed Notes on Transformers

CS11-711 Advanced NLP
Instructor: Graham Neubig Spring 2024

---

35 Introduction to Transformers

Transformers were introduced in the paper "Attention is All You Need" (https://arxiv.org/abs/1706.03762). They are sequence-to-sequence models based entirely on attention mechanisms, achieving state-of-the-art results in machine translation and many other NLP tasks. Transformers are efficient because they rely solely on matrix multiplications rather than recurrent operations. As highlighted in the transcript, the breakthrough of Transformers was not only in their performance but also in their speed—thanks to the removal of sequential recurrence, allowing for extensive parallelization during training.

36 Review: Attention Mechanism

Attention mechanisms allow models to focus on relevant parts of a sequence while processing inputs.

36.1 Cross-Attention

Cross-attention allows elements of one sequence to attend to another sequence. Given a query vector q and key vectors k_i, the attention weight is computed as:

$$a_i = \text{softmax}(q^T k_i) \tag{7}$$

The value vectors are then weighted and summed to obtain the attended representation:

$$v_{att} = \sum_i \alpha_i v_i \tag{8}$$

Cross-attention is the foundation of sequence-to-sequence models, where queries come from the decoder, and keys/values come from the encoder (https://arxiv.org/abs/1409.0473).

36.2 Self-Attention

Self-attention allows each element in a sequence to attend to all other elements in the same sequence, providing contextual representations (https://arxiv.org/abs/1601.067 https://arxiv.org/abs/1706.03762). The attention mechanism is formulated similarly to cross-attention, except that keys, queries, and values all come from the same sequence.

Transcript Insights: The transcript highlights how self-attention enables context-sensitive encodings. Each token can incorporate global dependencies efficiently, unlike recurrent architectures that struggle with long-range dependencies.

37 Transformer Architecture

Transformers consist of stacked layers composed of self-attention and feed-forward networks. Two major types of architectures exist:

- **Encoder-Decoder Transformers** (e.g., T5, BART) – Used for structured input-output tasks such as machine translation.
- **Decoder-Only Transformers** (e.g., GPT, LLaMa) – Used for autoregressive generation, including chatbots.

37.1 Key Components of Transformers

The essential building blocks are:

- Positional Encoding
- Multi-Head Attention
- Masked Attention (in decoder models)
- Residual Connections combined with Layer Normalization
- Feed-Forward Layers

38 Multi-Head Attention

Multi-head attention enables multiple, parallel attention computations over different subspaces of the input:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W_O \tag{9}$$

where each attention head is computed as:

$$\text{head}_i = \text{Attention}(QW_Q^i, KW_K^i, VW_V^i) \tag{10}$$

Transcript Insights: Multi-head attention allows different heads to focus on different linguistic properties. Some heads capture syntactic dependencies, while others capture semantic relationships. Efficient implementations compute all heads simultaneously via large matrix multiplications rather than separate iterative processing.

39 Positional Encodings

Since Transformers lack recurrence, positional encodings are essential to inject information about the order of tokens. Various schemes exist:

- **Sinusoidal Encoding** (https://arxiv.org/abs/1706.03762)
- **Learned Encoding** (https://arxiv.org/abs/1803.02155)
- **Relative Position Encoding** (https://arxiv.org/abs/1803.02155)
- **Rotary Positional Encoding (RoPE)** (https://arxiv.org/abs/2104.09864)

Transcript Insights: Without positional encodings, models would fail to distinguish repeated words. RoPE uses trigonometric transformations to encode relative positions, making it particularly effective for generalization.

40 Normalization and Residual Connections

Layer normalization stabilizes training and improves gradient flow:

$$\text{LayerNorm}(x) = \gamma \cdot \frac{x - \mu(x)}{\sigma(x)} + \beta \tag{11}$$

An alternative, **RMSNorm** (https://arxiv.org/abs/1910.07467), removes mean subtraction:

$$\text{RMSNorm}(x) = \frac{x}{\text{RMS}(x)} \cdot g, \quad \text{RMS}(x) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} \tag{12}$$

41 Feed-Forward Layers and Activation Functions

The feed-forward network (FFN) is applied independently to each token:

$$\text{FFN}(x) = f(xW_1 + b_1)W_2 + b_2 \tag{13}$$

Recent architectures like LLaMa prefer SiLU over ReLU:

$$\text{SiLU}(x) = x \cdot \sigma(x) \tag{14}$$

42 Optimization Strategies for Transformers

42.1 Optimizers

- **SGD** – Basic optimization method.
- **Adam** – Momentum-based optimization.
- **AdamW** (https://arxiv.org/abs/1711.05101) – Applies weight decay for better regularization.

42.2 Low Precision Training

To optimize memory and computation:

- **FP16** – Reduces floating-point precision.
- **BFloat16** – Offers better stability than FP16.

42.3 Checkpointing and Restarts

Large-scale training requires:

- Monitoring gradient norms to detect divergence.
- Rolling back to previous checkpoints upon sudden loss spikes.
- Restarting from different shuffles of data to recover from failures.