Week 3

Comprehensive Guide to Sentiment Analysis and Inference

Contents

1 Introduction

The field of Natural Language Processing (NLP) has undergone a paradigm shift, moving from rule-based systems and statistical methods relying on hand-crafted features to end-to-end deep learning models. A central theme in this evolution is the representation of text. We have moved from sparse, high-dimensional representations (like One-Hot Encoding and TF-IDF), which suffer from the "curse of dimensionality," to dense, continuous vector spaces (Word Embeddings) and finally to contextualized representations learned via pretraining.

This report explores the application of these deep representations to two fundamental tasks: **Sentiment Analysis** (classifying the emotional tone of a single text) and **Natural Language Inference** (determining the logical relationship between two texts). We specifically examine the efficacy of Convolutional Neural Networks (CNNs) for capturing local n-gram features and Attention mechanisms for modeling alignment between sequences, concluding with the state-of-the-art approach of fine-tuning Transformers like BERT.

The hierarchy of modern NLP can be visualized as a three-stage process:

1. **Pretraining:** Learning universal language representations (Word2Vec, GloVe, BERT) from massive unlabeled corpora.

2. **Architecture Design:** Selecting the neural scaffold (CNN, RNN, Transformer) appropriate for the structural properties of the input.

3. **Application Fine-tuning:** Adapting the pretrained features and architecture to a specific downstream task like sentiment classification or entailment checking.

2 Sentiment Analysis and the Dataset

Sentiment analysis, often referred to as opinion mining, is the computational study of people's opinions, sentiments, emotions, and attitudes expressed in written language. It is one of the most active research areas in NLP and is also widely studied in data mining, Web mining, and text mining.

2.1 Theoretical Framework: Text Classification

Mathematically, sentiment analysis is a mapping function $f : X \to Y$, where X is the space of possible text sequences (variable length) and Y is a finite set of classes (e.g., $Y = \{\text{positive}, \text{negative}\}$).

The fundamental challenge lies in the variability of X. Unlike fixed-size vectors in traditional statistics, a sentence can be of arbitrary length L. The model must therefore learn a **composition function** to aggregate L token vectors into a single fixed-size vector representation that captures the semantic orientation of the text. This aggregation must be invariant to the length of the input while being sensitive to the specific ordering of words that negates or amplifies sentiment (e.g., "not bad" vs "bad").

2.2 The IMDb Dataset

For our analysis, we utilize the Stanford Large Movie Review Dataset (IMDb).

- **Volume:** The dataset comprises 50,000 highly polar movie reviews.
- **Partitioning:** It is split evenly into 25,000 training and 25,000 testing examples.
- **Class Balance:** The dataset is perfectly balanced, containing an equal number of positive and negative reviews. This negates the need for techniques like oversampling or class-weighted loss functions typically required for imbalanced data.

2.3 Preprocessing Pipeline

Deep learning models cannot process raw text strings directly. We must convert them into numerical tensors through a standardized pipeline:

1. **Tokenization:** Breaking the text into atomic units (tokens). While simple whitespace splitting works for English, modern approaches often use Subword Tokenization (like Byte-Pair Encoding) to handle rare words and morphological variations.
2. **Vocabulary Construction:** We build a vocabulary $\mathcal{V}$ by filtering rare words (e.g., appearing < 5 times) to reduce the dimensionality of the embedding matrix. This reduces the parameter space in the embedding layer, preventing overfitting on noise.
3. **Sequence Standardization (Padding/Truncation):** To enable batch processing on GPUs, all sequences in a mini-batch must have the same length T.

$$x'_t = \begin{cases} x_t & \text{if } t \leq T \\ \text{PAD} & \text{if } t > \text{original length} \end{cases}$$

   For the IMDb dataset, we standardize reviews to 500 tokens. Reviews longer than 500 tokens are truncated, potentially losing information at the end of the text, while shorter reviews are padded with a special `<pad>` token (usually index 0).

Historical Context: The Pre-Deep Learning Era

Before deep learning dominance, sentiment analysis relied heavily on lexicons (lists of "good" and "bad" words) and classic machine learning algorithms like Naive Bayes and Support Vector Machines (SVMs). A landmark paper by Pang, Lee, and Vaithyanathan (2002) demonstrated that standard machine learning techniques on "Bag-of-Words" features outperformed human-produced baselines, setting the stage for statistical NLP. However, these methods failed to capture context; "not good" was often treated simply as the presence of "not" and "good".

Things to Ponder

- What hyperparameters in the preprocessing stage (e.g., vocabulary size, sequence length) can be modified to accelerate training?
- How would the model handle a review that is entirely composed of "unknown" tokens (words not in the vocabulary)?

3 Sentiment Analysis: Using Convolutional Neural Networks

While Convolutional Neural Networks (CNNs) gained fame in Computer Vision for their ability to capture spatial hierarchies in images, their application to NLP is grounded in the detection of **local semantic features**. In vision, a filter might detect an edge; in text, a filter detects a specific n-gram pattern (e.g., "not good", "very happy").

3.1 Theoretical Mechanism: 1D Convolutions

In NLP, we treat a sentence as a 1D signal with d channels, where d is the embedding dimension of the words. Unlike image processing where kernels move over height and width, in text processing, the kernel width is usually equal to the embedding dimension d. Therefore, the kernel only slides along the *temporal* dimension (the sequence of words).

Mathematical Derivation: 1D Cross-Correlation

Let $\mathbf{X} \in \mathbb{R}^{L \times d}$ be the input sentence matrix, where L is the sequence length and d is the embedding dimension. Let $\mathbf{K} \in \mathbb{R}^{w \times d}$ be a kernel (filter) of window size w. The convolution operation slides $\mathbf{K}$ over $\mathbf{X}$. The output scalar y_i at position i is the Frobenius inner product of the kernel and the window of text:

$$y_i = \sum_{j=0}^{w-1} \sum_{k=1}^{d} X_{i+j,k} \cdot K_{j,k} + b$$

where b is a bias term. This operation implies that the kernel looks at w consecutive words simultaneously, effectively capturing w-gram context.

3.2 Multi-Channel Input Architectures

A significant innovation in the textCNN architecture is the use of multiple "channels" for text. In images, channels typically refer to RGB colors. In textCNN, we can treat different embedding versions as different channels.

Commonly, two channels are used:

1. **Static Channel:** Pretrained embeddings (like GloVe or Word2Vec) that are kept frozen (not updated) during training. This ensures the model retains the general semantic relationships learned from the massive pretraining corpus.
2. **Dynamic (Non-Static) Channel:** A copy of the embeddings that is fine-tuned via backpropagation during training. This allows the model to learn domain-specific nuances (e.g., "unpredictable" might be negative in general contexts, but positive in a movie thriller context).

3.3 Max-Over-Time Pooling

A convolutional layer produces a feature map $\mathbf{y} = [y_1, y_2, \ldots, y_{L-w+1}]$. The length of this map depends on the input length L. To feed this into a fixed-size fully connected layer, we need to collapse this dimension.

We use **Max-Over-Time Pooling**, defined as:

$$\hat{y} = \max_i(y_i)$$

The theoretical justification is that we are interested in whether a specific feature (e.g., a strong negation) occurred *anywhere* in the sentence. The exact position matters less than its presence. This makes the model robust to shifts (translation invariance) and varying sentence lengths.

3.4 The textCNN Architecture

Proposed by Kim (2014), the textCNN architecture aggregates features from kernels of different sizes.

1. **Input Layer:** Sentence matrix with static and dynamic embedding channels.
2. **Convolutional Layers:** The model applies multiple filters with distinct window sizes (e.g., 3, 4, and 5). This parallels extracting trigrams, 4-grams, and 5-grams.
3. **Feature Aggregation:** The max-pooled outputs from all filters are concatenated into a single wide vector, forming a comprehensive representation of the salient n-grams in the text.
4. **Regularization:** Dropout is applied to the penultimate layer to prevent co-adaptation of features.

Historical Context: From TDNN to CNN

The concept of applying convolutions to time-series data predates modern NLP. In the late 1980s and early 1990s, Time-Delay Neural Networks (TDNNs) were developed for speech recognition. TextCNNs can be viewed as a modern, non-linear variant of TDNNs applied to discrete symbol sequences, leveraging the power of dense word embeddings.

Things to Ponder

- Can you further improve classification accuracy by using different kernel sizes (e.g., 2, 7) or adding positional encodings?
- Why does max-pooling work better for sentiment analysis than average-pooling? (Hint: Consider the sparsity of sentiment-bearing words in a long review).

4 Natural Language Inference and the Dataset

Natural Language Inference (NLI), also known as Recognizing Textual Entailment (RTE), is a higher-order understanding task. While sentiment analysis looks at a single text, NLI requires understanding the **logical relationship** between two distinct sequences: a *Premise* (P) and a *Hypothesis* (H).

4.1 Logical Relations

The task is to classify the pair (P, H) into one of three classes:

- **Entailment ($P \implies H$):** If P is true, then H must also be true.
- *Example:* P: "A soccer player is running." $\rightarrow$ H: "A person is moving."
- **Contradiction ($P \implies \neg H$):** If P is true, then H must be false.
- *Example:* P: "A man is inspecting the uniform of a woman." $\rightarrow$ H: "The man is sleeping."
- **Neutral:** The truth of H cannot be determined from P.
- *Example:* P: "The musicians are performing." $\rightarrow$ H: "The musicians are famous."

4.2 The SNLI Dataset

The Stanford Natural Language Inference (SNLI) corpus was a watershed moment for NLI. Prior datasets (like the RTE challenges) were small, preventing the application of hungry deep learning models. SNLI provided over 500,000 human-annotated pairs.

- **Crowdsourcing methodology:** Annotators were given a premise (from image captions) and asked to write three new sentences: one that is definitely true (entailment), one that is definitely false (contradiction), and one that might be true (neutral).
- **Data Scale:** 550k training pairs, 10k validation, 10k test.

Insight: The Need for Interaction

Simple classifiers that process P and H separately (e.g., extracting features from P and H independently and concatenating them) perform poorly. Successful NLI models must capture the **interaction** between words in P and words in H (e.g., aligning "soccer player" with "person"). This interaction is often modeled via Attention.

5 Natural Language Inference: Using Attention

Recurrent Neural Networks (RNNs) process sequences sequentially, which is computationally expensive ($O(L)$) and difficult to parallelize. Parikh et al. (2016) proposed a **Decomposable Attention Model** that solves NLI using only simple Matrix-Matrix multiplications, reducing complexity to parallelizable operations.

5.1 Theoretical Architecture

The model operates on the principle that NLI can be decomposed into sub-problems: aligning words, comparing aligned phrases, and aggregating the results.

5.1.1 Step 1: Soft Alignment (Attend)

The first step is to determine which words in the hypothesis correspond to which words in the premise. We construct an alignment matrix $\mathbf{E} \in \mathbb{R}^{L_P \times L_H}$ where e_{ij} represents the similarity between premise word i and hypothesis word j.

$$e_{ij} = F(\mathbf{a}_i)^\top F(\mathbf{b}_j)$$

where F is a feed-forward network.

We then compute "soft" aligned vectors. For every word in the premise $\mathbf{a}_i$, we compute a weighted sum of the hypothesis $\boldsymbol{\beta}_i$:

$$\boldsymbol{\beta}_i = \sum_{j=1}^{L_H} \frac{\exp(e_{ij})}{\sum_k \exp(e_{ik})} \mathbf{b}_j$$

This vector $\boldsymbol{\beta}_i$ represents "what the hypothesis looks like from the perspective of premise word i". If $\mathbf{a}_i$ is "dog", $\boldsymbol{\beta}_i$ will be a weighted sum of hypothesis words, dominated by semantically related terms like "canine" or "animal".

5.1.2 Step 2: Local Comparison (Compare)

We now compare the original word $\mathbf{a}_i$ with its aligned counterpart $\boldsymbol{\beta}_i$. This comparison is done by a non-linear function G (another MLP):

$$\mathbf{v}_{A,i} = G([\mathbf{a}_i, \boldsymbol{\beta}_i])$$

This step is crucial. It allows the model to locally detect contradictions or entailments. For example, if $\mathbf{a}_i$ is "sleeping" and $\boldsymbol{\beta}_i$ (the aligned hypothesis part) aligns to "running", the network G can output a feature vector indicating a distinct conflict. This step effectively acts as a local NLI classifier for every word pair.

5.1.3 Step 3: Global Aggregation (Aggregate)

Finally, we sum the comparison vectors to get a global view of the relationship:

$$\mathbf{v}_{final} = \sum_i \mathbf{v}_{A,i} + \sum_j \mathbf{v}_{B,j}$$

This aggregated vector is fed into a final classifier. The summation operation is order-invariant, meaning the model essentially treats the sentence as a "Bag of Aligned Features", discarding strict sequential dependencies in favor of efficient parallel computation.

Historical Context: Attention is All You Need

The Decomposable Attention model was a precursor to the Transformer revolution. Published in 2016, it demonstrated that attention mechanisms alone—without the recurrence of RNNs or the convolution of CNNs—could achieve state-of-the-art results. This paved the way for the "Attention Is All You Need" paper (Vaswani et al., 2017) one year later.

Things to Ponder

- What are the major drawbacks of this model? (Hint: Does it consider the order of words? Is "dog bites man" different from "man bites dog" in this architecture?)
- How would you modify this to output a continuous similarity score (0 to 1) instead of a classification?

6 Fine-Tuning BERT for NLI

The current dominant paradigm in NLP is **Transfer Learning**. Instead of training models from scratch (initializing weights randomly), we start with a model pre-trained on a massive corpus (like Wikipedia) to learn the structure of language, and then "fine-tune" it for a specific task.

6.1 BERT: Bidirectional Encoder Representations from Transformers

BERT differs from previous models because it is **deeply bidirectional**. It uses the Transformer Encoder architecture to read the entire sequence at once, utilizing Self-Attention to allow every token to attend to every other token.

6.1.1 Pretraining Objectives

BERT is pretrained on two unsupervised tasks:

1. **Masked Language Modeling (MLM):** 15% of input tokens are masked at random, and the model must predict the original token based on the bidirectional context. This forces the model to learn deep contextual dependencies.

2. **Next Sentence Prediction (NSP):** The model receives pairs of sentences and predicts if the second follows the first. This objective is specifically designed to help the model understand relationships between sequences, making BERT ideal for NLI.

6.1.2 Input Representation

BERT's input representation is a composite of three embeddings. For NLI, where we have a Premise and Hypothesis, the input is constructed as:

`[CLS]` Premise `[SEP]` Hypothesis `[SEP]`

The embedding for a given token is the sum of:

- **Token Embedding:** The WordPiece embedding for the word itself.
- **Segment Embedding:** A learned vector indicating whether the token belongs to Sentence A (Premise) or Sentence B (Hypothesis). This is critical for NLI so the model can distinguish the two parts of the logical argument.
- **Position Embedding:** A learned vector indicating the token's position in the sequence (0, 1, 2...), as the Transformer architecture itself is permutation invariant.

6.2 Fine-Tuning Strategy for NLI

NLI fits perfectly into BERT's pretraining structure because BERT was trained to handle sentence pairs (via the NSP task).

To fine-tune for SNLI: 1. **Forward Pass:** Run the sequence through the 12 (or 24) layers of the Transformer. 2. **Extraction:** Extract the vector $\mathbf{h}_{CLS}$ corresponding to the `[CLS]` token from the final layer. The `[CLS]` token is designed to aggregate information from the entire sequence into a single classification vector. 3. **Classification:** Pass $\mathbf{h}_{CLS}$ through a simple linear layer $\mathbf{W} \in \mathbb{R}^{K \times H}$ (where $K = 3$ for NLI labels) followed by Softmax:

$$P(y|x) = \text{softmax}(\mathbf{W}\mathbf{h}_{CLS})$$

6.3 Training Dynamics

During fine-tuning, we update **all** parameters of the model, not just the final linear layer. However, because the pretrained weights are already good feature extractors, the model converges very quickly (typically 2-4 epochs).

Crucially, we typically use a triangular learning rate schedule with a warm-up period and a very small peak learning rate (e.g., 2×10^{-5}). This cautious approach is necessary to avoid "catastrophic forgetting," where the model overwrites the general language knowledge learned during pretraining with the specific patterns of the NLI dataset.

Historical Context: The ImageNet Moment

The release of BERT (and predecessors like ELMo and ULMFiT) is often referred to as "NLP's ImageNet Moment." Just as Computer Vision made a leap when researchers stopped training from pixels and started fine-tuning AlexNet/ResNet, NLP made a massive leap when it moved from initializing with static Word2Vec vectors to fine-tuning entire Transformer networks.

Things to Ponder

- How can we leverage BERT to train a machine translation system?
- If we have a very long premise and hypothesis that exceed BERT's 512-token limit, how should we truncate them? (Ratio of lengths vs. hard cut-off).