

TensorFlow in Action (Part 2)

Chapter 9: Natural Language Processing with TensorFlow – Sentiment Analysis

Introduction

Chapter 9 marks a substantial shift towards Natural Language Processing (NLP)—one of the most fascinating domains in deep learning. The transition from images to text presents fundamentally different challenges. While images are fixed-dimensional grids of pixels with relatively consistent spatial relationships, text is sequential data with variable length, complex grammatical structure, and rich, subtle semantic meaning. Sentiment analysis—classifying text according to emotional tone (positive, negative, neutral)—was chosen as the perfect entry point for NLP with TensorFlow because it is accessible enough for beginners yet demonstrates fundamental NLP techniques applicable to broader NLP tasks.

Text Preprocessing: Preparing Raw Data for Learning

Before neural networks can learn from text, the raw text must be converted into a computable format. This preprocessing involves several critical steps that directly impact model performance.

Tokenization is the first step—splitting text sequences into individual tokens (words or subwords). Word-level tokenization is the most straightforward—splitting on whitespace and punctuation. However, the word-level approach has limitations: handling complex punctuation, unknown words, and diluting word statistics for rare words. Character-level tokenization—treating each character as a token—avoids the unknown word problem but dramatically increases sequence length and modeling difficulty. Subword tokenization (such as Byte Pair Encoding or WordPiece)—common in modern NLP—offers middle ground, breaking words into frequent subwords and preserving word identity for common words.

Vocabulary building follows tokenization—creating a mapping from tokens to integer indices. Tokens are ranked by frequency, and the top N tokens (typically 20,000–100,000) are retained. Out-of-vocabulary (OOV) handling is an important consideration—words not in the vocabulary must be handled, either with special UNK tokens or with character-level processing.

Text cleaning includes lowercase conversion (to reduce vocabulary size and improve generalization), removing or normalizing punctuation, removing URLs and mentions (for social media text), and removing HTML tags (for web-scraped data). Stemming and lemmatization—reducing words to their root forms (running, runs, ran → run)—can improve generalization but risk losing semantic information. Modern neural networks often learn effective representations without explicit stemming, so preprocessing must be balanced—removing obvious noise without overprocessing.

Padding and truncation are practical considerations for batching variable-length sequences. Short sequences are padded with special PAD tokens to the maximum length in the batch or dataset. Long sequences are truncated to model input length. Masking mechanisms can inform the model to ignore padding tokens during training and inference.

Word Embeddings: Semantic Representation of Words

A fundamental representation for NLP is word embeddings—dense vectors that represent words in a continuous space where semantic similarity corresponds to geometric proximity. Unlike one-hot encoding (sparse vectors with single 1s and rest of the 0s), embeddings are learned representations that capture semantic relationships.

Word2Vec is a foundational embedding approach, with two variants: Skip-gram and CBOW (Continuous Bag of Words). Skip-gram predicts surrounding words (context) given a center word, encouraging words in similar contexts to have similar embeddings. CBOW is the opposite mechanism, predicting the center word from context words. The training procedure is elegantly simple—binary classification tasks (identifying the correct context word from negative samples) are used to update embedding weights.

GloVe (Global Vectors for Word Representation) takes a different approach, factoring the co-occurrence matrix of word-context pairs. The result is embeddings that capture both local context (from Word2Vec-like objectives) and global corpus statistics. GloVe embeddings are particularly effective for capturing analogical relationships—a famous example is $\text{king} - \text{man} + \text{woman} \approx \text{queen}$, demonstrating that algebraic operations in the embedding space correspond to semantic relationships.

FastText is an improvement over Word2Vec that handles out-of-vocabulary words by representing words as sums of subword embeddings. If a word is not in the vocabulary, FastText can still generate an embedding based on known subword components. This is particularly valuable for languages with rich morphology or datasets with typos.

TensorFlow provides `tf.keras.layers.Embedding`—a learnable embedding layer that is initialized randomly and trained alongside the downstream model. This allows the model to learn custom embeddings optimized for specific tasks. Alternatively, pre-trained embeddings (GloVe vectors) can be used, loaded and used, potentially with fine-tuning or frozen weights.

Recurrent Neural Networks for Sequences

Recurrent Neural Networks (RNNs) are a fundamental architecture for processing sequences. Basic RNNs process sequential input one element at a time, maintaining a hidden state that carries information from previous elements. The hidden state is updated based on the current input and the previous hidden state: $h_t = \text{activation}(W_h * h_{t-1} + W_x * x_t + b)$.

Basic RNNs suffer from a fundamental problem: the vanishing gradient problem. During backpropagation through time (BPTT), gradients multiply by weight matrices repeatedly, and if weights < 1 , gradients vanish exponentially, preventing updates to early network layers. This makes basic RNNs ineffective for capturing long-range dependencies.

Long Short-Term Memory (LSTM) networks solve this problem by introducing cell states—separate memory tracks that flow through the network with minimal modification, allowing gradients to flow efficiently. LSTM augments the hidden state with a cell state (C_t) updated by carefully controlled gates: an input gate (controlling where new information enters), a forget gate (controlling where old information is discarded), and an output gate (controlling where the cell state influences the output). Gates are sigmoid-parameterized mechanisms that learn to open or close, allowing dynamic routing of information.

A Gated Recurrent Unit (GRU) is a simplification of LSTM with fewer parameters but comparable performance. The GRU combines input and forget gates into a single update gate, reducing the number of parameters while maintaining expressiveness.

Bidirectional RNNs and Contextual Understanding

Basic RNNs process sequences left-to-right—the hidden state at position t depends only on previous inputs. For tasks like sentiment analysis, future context is often informative—if a positive review mentions "not good but amazing cinematography," understanding the complete phrase requires knowing both the context before and after each word.

Bidirectional RNNs (BiRNNs) process sequences in two directions simultaneously. Forward RNN processes left-to-right, while backward RNN processes right-to-left. The hidden states of both are concatenated, providing an enriched context that includes both historical and future information. BiLSTMs and BiGRUs are particularly effective for sequence labeling and classification tasks.

Attention Mechanisms: Focusing on Relevant Information

In longer sequences, not all positions are equally important for the final decision. Attention mechanisms allow models to dynamically weight the importance of different positions. In self-attention, the network learns to attend to which positions when processing each position.

Attention computation involves queries (representations of the current position), keys (representations of all positions), and values (actual content). Attention weights are computed as softmax normalized similarities between queries and keys, then used to weight the sum of the values.

In the context of sentiment analysis, attention mechanisms can reveal the most influential words for the final sentiment decision. High attention weights on words like "amazing," "terrible," and

"excellent" indicate that the model focuses on evaluative language. This not only improves performance but also provides interpretability.

Training Sentiment Analysis Models

Common datasets for sentiment analysis tasks are IMDb movie reviews (50,000 reviews with binary positive/negative labels), Stanford Sentiment Treebank (fine-grained 5-way classification), and Twitter sentiment (challenging due to informal language and limited context).

Model architecture for sentiment analysis usually follows the pattern: embedding layer → BiLSTM/BiGRU layers → attention or global pooling → dense classification layers. Embedding layer maps token indices into dense vectors. RNN layers process sequences, outputting sequences of hidden states. Pooling layer aggregates sequence representations—global average pooling (averaging hidden states) or attention pooling (weighted combination) produces fixed-size representations of variable-length sequences. Classification layers (fully connected with dropout) output logits for classes.

Training uses binary cross-entropy loss for binary classification or categorical cross-entropy for multi-class. Optimization using adaptive optimizers like Adam. Batch size typically 32-128. Learning rate scheduling helpful—starting with a moderate rate and decreasing based on validation performance.

Evaluation metrics include accuracy (proportion correct classifications), precision and recall per class, and F1-score. ROC-AUC is particularly useful for imbalanced datasets. Confusion matrix reveals specific error patterns—misclassifications as false positives versus false negatives.

Handling Class Imbalance

Real-world sentiment datasets are often imbalanced—if positive reviews are more common, the model can achieve high accuracy by predicting positive for everything. Stratified sampling ensures balanced representation in training batches. Class weighting assigns higher loss contributions to minority classes. Techniques such as oversampling the minority class or undersampling the majority class can be used, with care to avoid overfitting on oversampled duplicates.

Real-World Applications and Impact

Sentiment analysis is widely used in practice. In social media monitoring, brands track mention sentiment to understand customer perception and identify potential crises. Customer feedback analysis helps companies understand satisfaction levels and identify improvement areas. Market research uses sentiment analysis to analyze consumer opinions about products or events. Content moderation platforms use sentiment combined with toxicity detection to identify harmful content. Movie rating prediction uses review text for predictions. Financial sentiment analysis analyzes news and social media to predict market movements.

Conclusion and Significance

Chapter 9 introduces fundamental concepts in NLP that are applicable far beyond sentiment analysis. Text preprocessing, embeddings, RNNs, and attention mechanisms are the building blocks for nearly all modern NLP applications. Skills for building effective sentiment analysis systems transfer to other text classification tasks. Understanding preprocessing importance, embedding representations, and sequence modeling paradigms provides a solid foundation for deeper NLP exploration.