Chapter 9

Natural Language Processing with TensorFlow: Sentiment Analysis

Natural language processing plays a central role in extracting meaning from text, whether for tasks such as machine translation, entity recognition, or sentiment analysis. The process begins by downloading a dataset of Amazon video game reviews and preparing it for modeling. Preprocessing involves cleaning the text—removing noise, applying lemmatization to reduce words to their base forms, and filtering stop words while retaining critical tokens like not, which can flip sentiment meaning. To prepare inputs for deep learning, text must be tokenized into word IDs and standardized through padding or bucketing so that variable-length reviews can be batched efficiently (figure 1, shows sequences before and after padding).

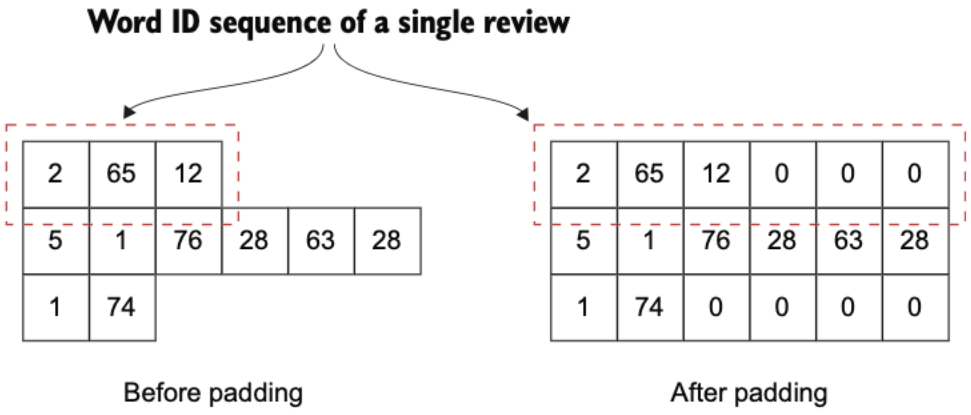


Figure 1 Text sequences before and after padding

Reproduced code for preprocessing function:

```
def clean_text(doc):

    doc = doc.lower()

    doc = doc.replace("n\'t ", ' not ')

    doc = re.sub(r"(?:\'ll |\'re |\'d |\'ve )", " ", doc)

    doc = re.sub(r"\d+","", doc)


    tokens = [

        w for w in word_tokenize(doc)
        if w not in EN_STOPWORDS and w not in string.punctuation
    ]


    pos_tags = nltk.pos_tag(tokens)
    clean_text = [
        lemmatizer.lemmatize(w, pos=p[0].lower())
        if p[0] == 'N' or p[0] == 'V' else w
        for (w, p) in pos_tags
    ]
    return clean_text
```

Once text is numerical, the challenge becomes modeling sequential dependencies. Sentiment is not determined by isolated words but by patterns across the sequence. To address this, the chapter introduces recurrent deep learning models, particularly long short-term memory networks (LSTMs). Unlike simple RNNs, LSTMs maintain both short-term and long-term memory using gated mechanisms. At each timestep, the model updates its cell state and hidden state through input, forget, and output gates, controlling what information is remembered, discarded, or emitted (figure 2 illustrates this flow; figure 3, contrasts output options when toggling return_state and return_sequences). This allows the model to capture dependencies across long spans of text.

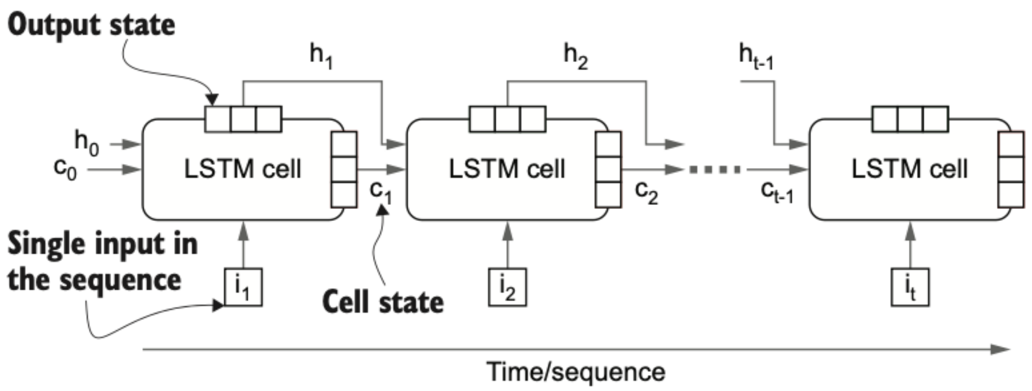


Figure 2 High-level longitudinal view of an LSTM cell. At a given time step t, the LSTM cell takes in two previous states (ht-1 and ct-1), along with the input, and produces two states (ht and ct)

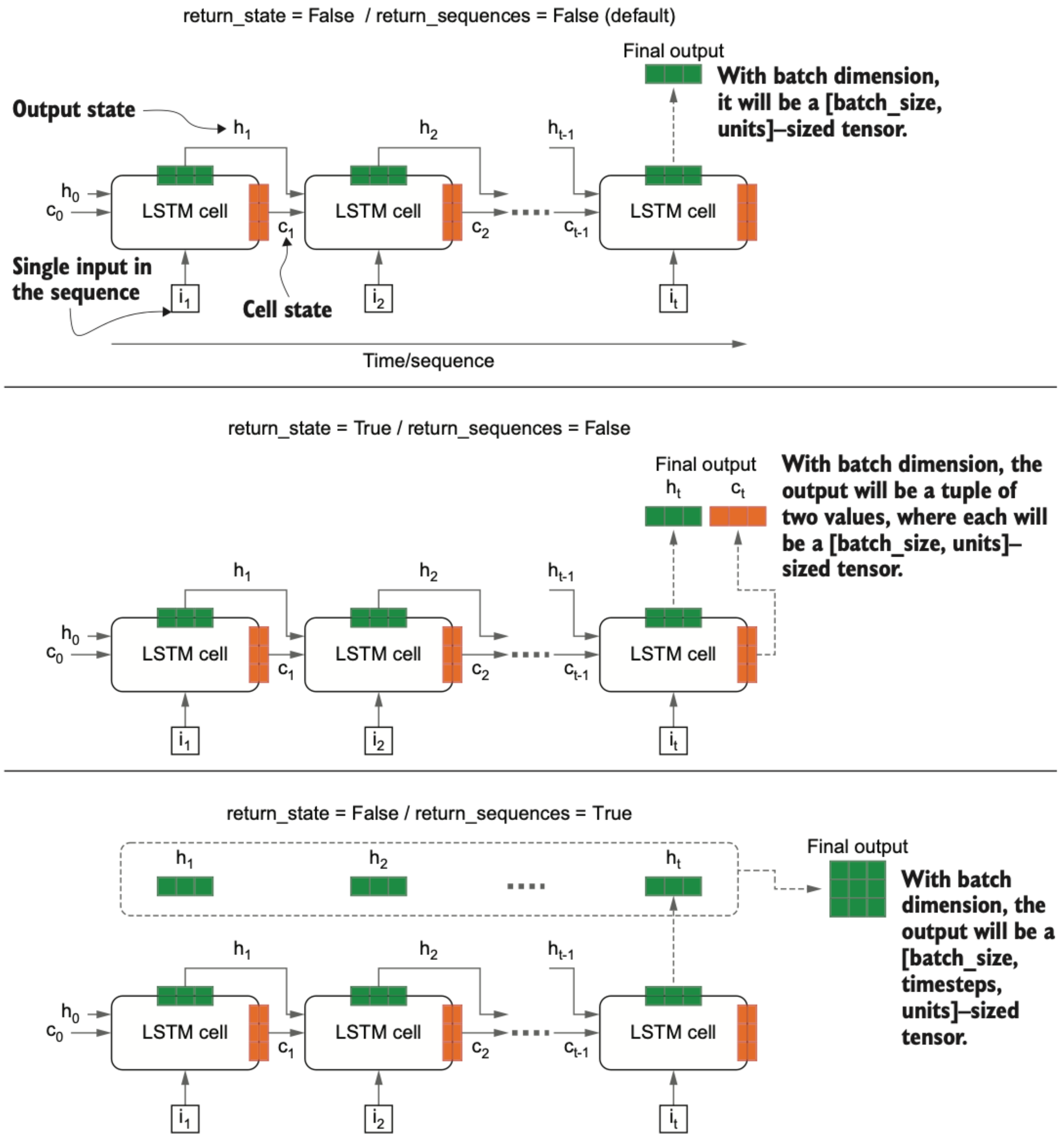
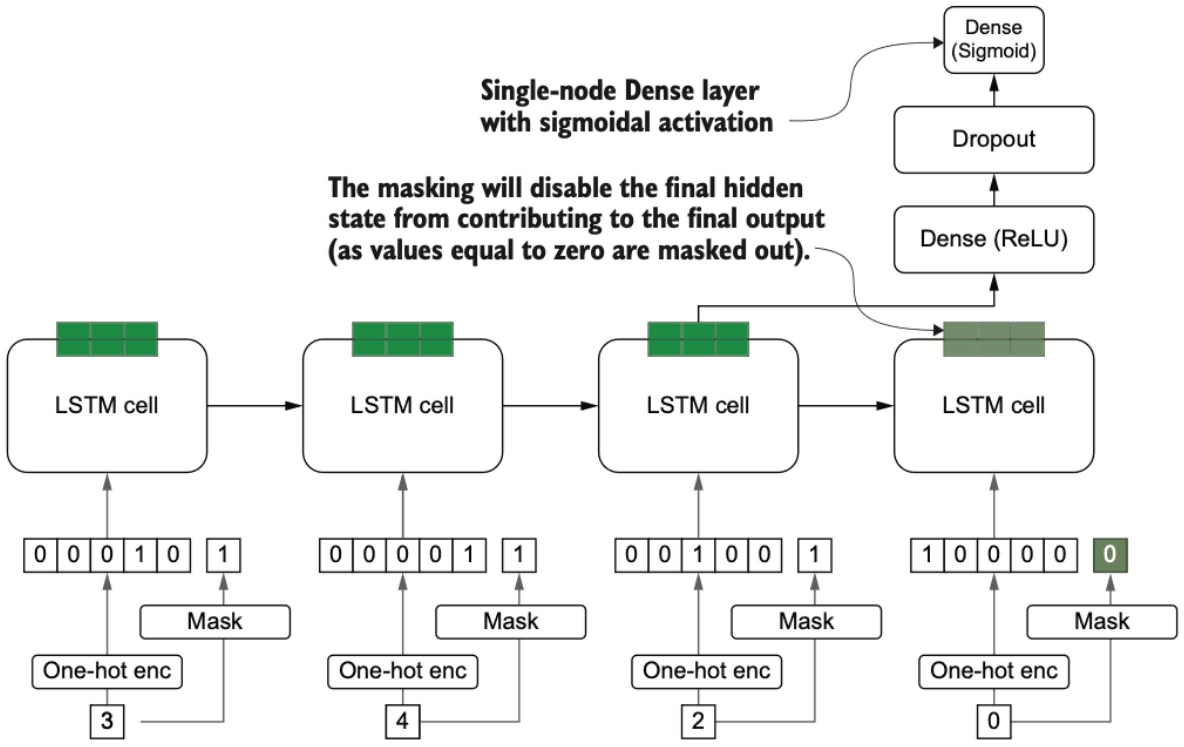

return_state = False / return_sequences = False (default)
Final output
With batch dimension, it will be a [batch_size, units]–sized tensor.
Output state
h1
h2
ht-1
h0
c0
LSTM cell
c1
c2
ct-1
Single input in the sequence
Cell state
i1
i2
it
Time/sequence
return_state = True / return_sequences = False
Final output
ht
ct
With batch dimension, the output will be a tuple of two values, where each will be a [batch_size, units]–sized tensor.
return_state = False / return_sequences = True
Final output
With batch dimension, the output will be a [batch_size, timesteps, units]–sized tensor.

Figure 3 The changes to the output of the LSTM layer resulted in changes to the return_state and return_sequences arguments

To build the sentiment analyzer, a sequential architecture is defined (figure 4). It starts with a masking layer to ignore padded zeros, followed by a custom one-hot encoder converting word IDs into sparse binary vectors. These pass through an LSTM layer, which compresses sequential information into a single representation. The output then flows into a dense hidden layer of 512 ReLU units, followed by dropout regularization, and finally into a sigmoid-activated single-node output that predicts sentiment polarity. This design is trained with binary cross-entropy loss and optimized with Adam. The initial model reaches around 80% validation accuracy, showing that it can generalize well across unseen reviews.



Dense (Sigmoid)
Single-node Dense layer with sigmoidal activation
Dropout
The masking will disable the final hidden state from contributing to the final output (as values equal to zero are masked out).
Dense (ReLU)
LSTM cell
LSTM cell
LSTM cell
LSTM cell
0 0 0 1 0
1
0 0 0 0 1
1
0 0 1 0 0
1
1 0 0 0 0
0
Mask
Mask
Mask
Mask
One-hot enc
One-hot enc
One-hot enc
One-hot enc
3
4
2
0

Figure 4 The high-level model architecture of the sentiment analyzer

Despite this success, one-hot encoding remains inefficient. It creates extremely sparse vectors, ignores semantic similarity, and scales poorly with large vocabularies. To overcome these limitations, we need to do word embeddings. Word vectors represent words as dense, low-dimensional vectors where semantic meaning is preserved: cat and dog are closer than cat and volcano. The narrative introduces key embedding algorithms such as Skip-gram, Continuous Bag-of-Words, GloVe, and ELMo, which capture context in different ways. For example, GloVe leverages global co-occurrence statistics, while ELMo uses bidirectional LSTMs to learn context-dependent embeddings.

The final sentiment analyzer replaces the one-hot encoder with an embedding layer that directly looks up word vectors and jointly trains them with the LSTM classifier. The architecture simplifies masking since embeddings can automatically handle zero-padding with the mask_zero=True option. Training this enhanced model demonstrates improved generalization: after several epochs, validation accuracy rises slightly higher than the earlier one-hot version, reaching beyond 82% and sustaining strong test performance (~81% accuracy). This proves that embeddings not only reduce computational cost but also capture richer semantics.

Evaluation does not stop at accuracy alone. The chapter emphasizes sanity-checking predictions by inspecting top positive and negative outputs to ensure the model's reasoning aligns with human intuition. This step guards against silent errors or misleadingly high numerical metrics. By the end, the sentiment analyzer demonstrates a robust NLP pipeline: from raw text through preprocessing and batching, into sequential modeling with LSTMs, enhanced by embeddings, and validated against real-world data.