Chapter 4

Sequence labeling with Hugging Face Diffusers

Introduction

Sequence labeling is a core task in **natural language processing** (**NLP**) that involves assigning labels to individual tokens in a sequence, such as words in a sentence. Two significant sequence applications labeling include **Named Entity Recognition** (**NER**) and **Part-of-Speech** (**POS**) **tagging**. These tasks are crucial for extracting structured information, understanding sentence structure, and enhancing semantic understanding. This chapter focuses on implementing sequence labeling tasks using the Hugging Face Diffusion library.

Structure

This chapter covers the following topics:

* Overview of sequence labeling in NLP
* Named Entity Recognition
* Model training and evaluation

Objectives

By the end of this chapter, readers will understand the fundamentals of sequence labeling and its significance in NLP applications. Implement NER using pre-trained models and fine-tune them for domain-specific tasks. Perform POS tagging, labeling each token with its grammatical role in a sentence. Readers would also be able to train and evaluate sequence labeling models using relevant datasets and performance metrics, and apply them to specialized domains such as biomedical text analysis and social media text processing.

Overview of sequence labeling in NLP

Sequence labeling is the process of assigning categorical labels to elements in a sequence of tokens, typically words. This task plays a foundational role in NLP applications such as information extraction, syntactic analysis, and semantic understanding. These are everyday sequence labeling tasks:

* NER: Showing entities like persons, organizations, and locations in a text.
* POS **t**agging: Assigning grammatical labels (e.g., noun, verb, adjective) to each word in a sentence.

Both tasks are widely used in industries such as healthcare, where NER can be applied to extract key medical information from unstructured clinical notes, and in social media analytics, where POS tagging helps parse and understand informal text.

Fundamentals of sequence labeling

Sequence labeling models are designed to predict an output sequence that corresponds to the input sequence of words or characters. These models consider the contextual relationships between elements in a sequence, which is vital for understanding language patterns and structures. The most common approaches to sequence labeling include Conditional Random Fields (CRFs), Hidden Markov Models (HMMs), and, more recently, neural network-based models like Recurrent Neural Networks (RNNs) and transformers.

These methodologies represent the evolution of how sequence dependencies are mathematically modeled and computationally resolved in NLP. Each embodies a distinct balance between probabilistic structure and representational depth, defining the progression from traditional statistical frameworks to neural architectures. The following overview outlines the foundational models that have shaped sequence labeling research and practice:

* **Conditional Random Fields (CRFs):** CRFs are statistical modeling methods often used in sequence labeling that model the conditional probability of a label sequence given a corresponding sequence of input tokens. They are particularly effective in cases where the decision at one position depends on the decisions at earlier positions. [1]
* **Hidden Markov Models (HMMs):** HMMs are used for sequence labeling by modeling the sequence as a Markov process with hidden states. They are well-suited for tasks where a token's label depends only on its immediate predecessor, making them effective for simpler tagging tasks. [2]

As sequence labeling tasks became more complex, **Recurrent Neural Networks (RNNs)** emerged as a breakthrough, allowing models to capture long-range dependencies in text sequences. Unlike CRFs and HMMs, which rely on explicitly defined probabilistic assumptions, RNNs learn representations directly from data and maintain an internal state that encodes previous context. Variants such as **Long Short-Term Memory (LSTM)** and **Gated Recurrent Units (GRU)** further improved this architecture by mitigating the vanishing gradient problem, enabling models to preserve linguistic dependencies across longer spans of text. These neural methods set the stage for **transformer-based architectures**, which now dominate modern NLP by extending contextual modeling through self-attention mechanisms.

Together, these classical and neural models established the theoretical groundwork for modern sequence labeling. While CRFs and HMMs formalized the statistical relationships between tokens and their labels, RNNs and transformers expanded the representational capacity of models to capture semantic and syntactic dependencies at scale. This progression marks the transition from rule-based inference to representation learning—a shift that continues to define the evolution of NLP systems.

Significance of sequence labeling in NLP

Sequence labeling is crucial in NLP for structuring raw text data into formats that are easier to analyze and understand. This capability is essential for applications such as:

* **Information extraction**: Naming and classifying key pieces of information from text, such as dates, names, and locations, which is fundamental for data retrieval and organization tasks.
* **Sentiment analysis**: Tagging parts of texts to name sentiment-bearing phrases or to figure out the sentiment expressed toward specific entities.
* **Speech recognition**: Converting speech into text that can be further processed and labeled to enhance comprehensibility and interaction quality in human-computer interaction.

Sequence labeling is a cornerstone of NLP applications, including information extraction, sentiment analysis, and speech recognition. Structuring raw text into labeled formats allows systems to extract and classify meaningful data from unstructured inputs. To provide a clearer understanding of this process, the following figure illustrates the step-by-step flow of sequence labeling, from the first text input through to the generation of labeled output:

A diagram of a network

AI-generated content may be incorrect.

Figure 4.1 Evolution of Sequence Labeling Models in NLP.

Applications of sequence labeling

Sequence labeling plays a pivotal role in a wide range of applications, enhancing our interaction with and processing textual data. Here are key applications where sequence labeling techniques are particularly effective:

* **Information retrieval**: Sequence labeling is crucial in information retrieval systems where the goal is to extract structured information from unstructured text. For example, in legal and financial documents, entities such as case law references, statutes, monetary amounts, or party names can be named and used to index documents, making them easier to search and retrieve. [3]
* **Healthcare analytics**: In the healthcare sector, sequence labeling is used to extract medical entities from clinical notes. This includes naming symptoms, diagnoses, medications, and dosages, which can then be used for patient management systems, billing, and research purposes. By automating the extraction of these entities, healthcare providers can improve patient care efficiency and accuracy. [4]
* **Sentiment analysis**: In customer service and marketing, sequence labeling helps analyze customer feedback by naming and categorizing sentiments expressed in reviews and social media posts. This application is crucial for businesses to gauge public sentiment, understand customer needs, and tailor their services or products accordingly. [5]
* **Application example - Sentiment analysis**: Before we understand the technical setup of a sequence labeling model, let us consider a practical example of sentiment analysis applied to social media monitoring. Companies often use sentiment analysis to track how customers feel about their brand in real time by analyzing tweets, Facebook posts, and other social media content. By labeling word sequences in these posts as positive, negative, or neutral, companies can quickly respond to customer complaints, gauge overall sentiment toward product launches, and tailor marketing strategies to better align with public opinion.

This practical example sets the stage for the following Python code demonstration, where we will implement a basic CRF model for part-of-speech tagging, a foundational technique that can be adapted for more complex sequence labeling tasks, such as sentiment analysis.

By linking these applications with the upcoming code example, readers can see the direct relevance and impact of their learning, enhancing both understanding and engagement.

* **POS tagging**: POS tagging assigns grammatical labels to words in a sentence, finding their role as nouns, verbs, adjectives, and so forth. This task plays a critical role in syntactic and semantic analysis, making it fundamental for various NLP tasks, including machine translation, sentiment analysis, and information retrieval. With the Hugging Face Diffusion library, POS tagging becomes streamlined and accessible by leveraging pre-trained models to assign grammatical labels to text tokens.

To implement POS tagging, you first load a pre-trained model and apply it to a sample text. The following section discusses an example showing how to tokenize the text and use the Hugging Face Diffusion library to assign POS tags.

Application example of implementing a CRF for POS tagging

In the context of sequence labeling tasks, POS tagging is a crucial step for syntactic analysis in NLP applications. POS tagging assigns grammatical labels, such as nouns, verbs, and adjectives, to each word in a sentence. One practical approach to this task is to use a Conditional Random Field (CRF), a probabilistic model widely used for structured prediction. CRFs model the dependencies between adjacent labels, making them particularly effective for tasks where the context of neighboring words influences classification accuracy.

**Step 1 — Feature Extraction and Dataset Preparation**

The following snippet demonstrates how to prepare a minimal dataset and extract features for each token using Python’s sklearn-crfsuite library:

```python

import sklearn\_crfsuite

from sklearn\_crfsuite import metrics

# Example dataset (simplified)

sentences = [

[("I", "PRON"), ("saw", "VERB"), ("the", "DET"), ("cat", "NOUN")],

[("The", "DET"), ("cat", "NOUN"), ("sat", "VERB"), ("on", "ADP"), ("the", "DET"), ("mat", "NOUN")]

]

def word2features(sent, i):

"""Extract core word features for CRF training."""

word = sent[i][0]

feats = {'word.lower()': word.lower(), 'word[-3:]': word[-3:], 'word.isupper()': word.isupper()}

if i > 0: feats['-1:word.lower()'] = sent[i-1][0].lower()

if i < len(sent)-1: feats['+1:word.lower()'] = sent[i+1][0].lower()

return feats

```

**Step 2 — Model Training and Evaluation**

Once the features are extracted, the CRF model can be trained and evaluated as shown below:

```python

X = [[word2features(s, i) for i in range(len(s))] for s in sentences]

y = [[label for \_, label in s] for s in sentences]

crf = sklearn\_crfsuite.CRF(algorithm='lbfgs', c1=0.1, c2=0.1, max\_iterations=100)

crf.fit(X, y)

print("Accuracy:", metrics.flat\_accuracy\_score(y, crf.predict(X)))

```

This example begins by preparing a small set of token–tag pairs and defining a word2features function that extracts informative linguistic cues such as suffixes, capitalization, and contextual neighbors. These features enable the CRF model to learn sequential dependencies between words and their corresponding grammatical roles. After training, the model’s predictions are compared against the ground truth to calculate accuracy on unseen data.

By leveraging these structured dependencies, CRF models effectively capture grammatical relationships in text, making them a reliable baseline for modern NLP systems.  
*Full implementation available at:* ***GitHub Repository – Chapter 4 | POS Tagging CRF Example***

Next, the CRF model is trained using the sklearn-crfsuite library. The training process involves splitting the dataset into training and testing sets and applying the CRF algorithm. The model is optimized using the **Broyden–Fletcher–Goldfarb–Shanno** (**LBFGS**) algorithm, with both L1 (c1) and L2 (c2) regularization to prevent overfitting. After the model is trained, it makes predictions on the test set, which are compared with the actual labels to compute the model's accuracy. The accuracy score is calculated using metrics.flat\_accuracy\_score, which evaluates how well the model performs on unseen data.

This example highlights the practical application of a CRF model for sequence-labeling tasks such as POS tagging. By considering the features of words and their neighboring tokens, CRF models can effectively assign each word in a sentence its correct grammatical role, highlighting the utility of advanced NLP techniques in structured prediction tasks.

Named Entity Recognition

NER involves identifying and classifying named entities, including persons, organizations, and locations, within a text. It plays a crucial role in tasks like information extraction, knowledge graph creation, and question answering. With the Hugging Face Diffusion library, implementing NER is accessible using pre-trained models such as the **Bidirectional Encoder Representations from Transformers** (**BERT**).

To demonstrate, we can load a pre-trained model fine-tuned on NER tasks and further fine-tune it for domain-specific use cases, such as finding biomedical entities. In the following code example, we will use a pre-trained BERT model fine-tuned on NER for general purposes:

```python

from transformers import AutoModelForTokenClassification, AutoTokenizer, pipeline # Load a pre-trained model

tokenizer = AutoTokenizer.from\_pretrained("bert-base-cased")

model = AutoModelForTokenClassification.from\_pretrained("dbmdz/bert-large-cased-finetuned-conll03-english") # Use Hugging Face pipeline for Named Entity Recognition

ner\_pipeline = pipeline("ner", model=model, tokenizer=tokenizer)

# Test the NER pipeline text = "Hugging Face is a company based in New York." print(ner\_pipeline(text))

```

This snippet demonstrates how to load a pre-trained BERT model and use it for NER. The pipeline function from the Hugging Face library simplifies the process by encapsulating the model and tokenizer, allowing easy text input for entity recognition. In the example, the text "Hugging Face is a company based in New York" is processed, and the model identifies relevant named entities, such as "Hugging Face" (an organization) and "New York" (a location). The pre-trained model can be further fine-tuned for domain-specific tasks, offering flexibility and accuracy for more specialized applications.

This streamlined implementation highlights the ease with which NER can be customized using the Hugging Face Diffusion library, displaying its potential for both general and domain-specific applications.

Mechanisms of NER

NER systems typically work by combining grammatical rules with statistical techniques, including **machine learning** (**ML**) algorithms trained on large, annotated corpora. Advanced NER systems use deep learning models, particularly those based on the transformer architecture, to enhance the accuracy and efficiency of entity recognition.

* **Rule-Based systems**: Early NER systems relied heavily on handcrafted rules. These systems perform well when the language structure is consistent, but they lack flexibility and scalability.
* **Statistical models**: With the advent of statistical NLP, models like CRF and HMM have been used. These models learn from data and can generalize across different contexts. [1]
* **Deep learning approaches**: Recent advancements involve using deep learning models such as BERT and other transformer-based models, which consider the context of each word in a sentence to improve the accuracy of entity recognition. [6]

The mechanisms behind NER range from early rule-based systems to sophisticated deep learning methods, such as transformer models like BERT. These approaches enable the extraction of key information —including people, organizations, and locations — from raw text, making NER a flexible tool across various fields such as healthcare, journalism, and finance. To better illustrate this, the following figure shows an example of how NER converts a sample text into labeled entities, clearly displaying how each part of the input is categorized based on the identified entities.

A computer screen shot of a document

AI-generated content may be incorrect.

Figure 4.2 Named Entity Recognition: From Raw Text to Labeled Entities

Applications of NER

NER has proven invaluable across various industries, streamlining processes and improving the extraction of critical information from unstructured text. The following list outlines some key examples of how NER is applied in real-world scenarios:

* **Healthcare**: In healthcare, NER systems extract medical information from patient records, such as disease names, medication types, and dosages, which are crucial for patient care management and clinical decision-making. [4]
* **Media and journalism**: In the media industry, NER helps in automatically categorizing news articles and extracting relevant entities such as people, organizations, and locations, thus aiding in content management and recommendation systems.
* **Finance**: In the financial sector, NER is used to check and analyze financial documents, extracting critical data points like company names, stock symbols, and economic indicators, which are essential for automated trading systems and financial analysis.

Application example of implementing NER with Hugging Face transformers

This example shows how to implement NER using the Hugging Face Transformers library, which uses a pre-trained BERT model fine-tuned for NER. BERT models are particularly effective for tasks such as NER due to their ability to comprehend context and the relationships between words within a sentence.

The following code begins by loading a pre-trained BERT model that has been fine-tuned on the CoNLL-2003 dataset, designed explicitly for NER tasks. Alongside the model, a tokenizer is also loaded, which manages converting the raw text into a format suitable for the model to process. Once the model and tokenizer are in place, the next step is to set up the NER pipeline. The pipeline abstracts the complex operations required for named entity recognition, enabling a streamlined approach to processing text and detecting entities.

The provided example sentence, Bill Gates and Paul Allen founded Microsoft on April 4, 1975, is passed through the NER pipeline. The model processes this text and outputs the recognized entities along with their respective categories, such as persons or organizations. The results are then printed, showing how the model correctly identifies and classifies “Microsoft”, "Bill Gates", and "Paul Allen" as entities, providing insight into the model's ability to parse and analyze structured information within unstructured text.

```python

from transformers import AutoModelForTokenClassification, AutoTokenizer, pipeline

# Load pre-trained model and tokenizer

model = AutoModelForTokenClassification.from\_pretrained("dbmdz/bert-large-cased-finetuned-conll03-english")

tokenizer = AutoTokenizer.from\_pretrained("dbmdz/bert-large-cased-finetuned-conll03-english")

# Setup NER pipeline

ner\_pipeline = pipeline("ner", model=model, tokenizer=tokenizer)

# Example text

text = "Microsoft was founded by Bill Gates and Paul Allen on April 4, 1975."

# Perform NER

results = ner\_pipeline(text)

# Display results

print("Detected Entities:")

for entity in results:

print(f"Entity: {entity['word']}, Type: {entity['entity']}")

```

Implementing this NER task with Hugging Face begins by loading the pre-trained BERT model and tokenizer. This model, fine-tuned on the CoNLL-2003 dataset, has already been trained to find entities such as people, organizations, locations, and dates. The tokenizer converts the text into a tokenized format that the model can process for NER.

Next, the NER pipeline is set up. This pipeline simplifies the overall process by encapsulating the tokenization, model prediction, and result generation in a single step. By passing the input text into this pipeline, the model automatically finds named entities and categorizes them. In the example, "Microsoft" is identified as an organization, while "Bill Gates" and "Paul Allen" are classified as persons. The detected entities and their categories are printed for review, proving the model's ability to interpret the text effectively.

This implementation demonstrates how pre-trained models, such as BERT, can be used to perform NER tasks efficiently, making them valuable tools for information extraction, question answering, and document analysis. By using the Hugging Face Diffusion library, complex NLP tasks such as NER become more accessible, enabling practitioners to apply innovative AI techniques in real-world scenarios.

POS tagging

POS tagging is a fundamental task in NLP that assigns a part-of-speech tag (e.g., noun, verb, adjective) to each word in a text. This process is crucial for comprehending the syntactic structure and semantics of language, which supports a range of NLP tasks, including parsing, sentiment analysis, and machine translation.

Importance in syntax and semantics

POS tagging is essential for parsing sentences correctly, as it helps disambiguate word meanings and grammatical functions, improving the accuracy of syntactic analysis. For example, distinguishing between *"record"* (a noun) and *"record"* (a verb) based on context is crucial for correctly interpreting and processing sentence structure.

Techniques and models for effective POS tagging

To implement POS tagging effectively, various techniques and models have emerged, each offering unique advantages depending on the complexity of the text and the tagging goals. From early rule-based methods to sophisticated deep learning approaches, the advancements in POS tagging have greatly enhanced the accuracy of syntactic and semantic analysis. The following is a sample of the most used techniques and models for POS tagging, reflecting the progression from traditional methodologies to innovative neural networks:

* **Rule-based techniques**: Early POS taggers used handwritten rules to decide tags based on word suffixes and the context within a sentence.
* **Statistical models**: These include HMMs and **Maximum Entropy Markov Models** (**MEMMs**), which calculate the probability of a tag sequence given a sequence of words. [7]
* **Deep learning approaches**: Recent approaches use neural networks, especially RNNs and **Long Short-Term Memory networks** (**LSTMs**), to consider the context more effectively. These models often outperform traditional methods, especially when combined with techniques like CRFs for sequence prediction. [8]

Applications of POS tagging

POS tagging plays an integral role in various applications, enhancing the functionality and precision of language-based tools. For instance, in text-editing and grammar-checking tools like Grammarly, POS tagging identifies the parts of speech in a sentence, enabling the system to provide grammatical suggestions. Similarly, content filtering systems use POS tagging to categorize content based on specific nouns, verbs, and adjectives, making it indispensable for recommendation systems. In voice recognition systems, POS tagging improves the accuracy of speech-to-text conversion, helping the model better understand and process speech subtleties.

POS tagging is essential for understanding the syntactic and semantic aspects of language and is key to various NLP applications. Various techniques, including rule-based systems, statistical models, and deep learning, enhance the accuracy and reliability of this process. To clarify how POS tagging works, the accompanying figure illustrates the full workflow from input text to the assignment of part-of-speech tags, offering a straightforward view of how these tags are applied in practical scenarios.

A diagram of a sequence of steps

AI-generated content may be incorrect.

Figure 4.3 RNN for sequence labeling

POS tagging with Hugging Face transformers example

In this example, we demonstrate how to perform POS tagging using the Hugging Face Transformers library with a pre-trained BERT model fine-tuned for this task. First, load the tokenizer and model trained for token classification. Then, set up the POS tagging pipeline, which simplifies the process by combining tokenization, model inference, and result output into a single step.

Once the pipeline is set up, we provide an example sentence, "The quick brown fox jumps over the lazy dog." The pipeline processes this input and outputs the POS tags for each word, naming them as nouns, verbs, adjectives, and other parts of speech. This approach highlights the utility of transformer-based models for simplifying and improving the efficiency of everyday NLP tasks, such as POS tagging, which are foundational for grammar checking, machine translation, and syntactic analysis.

The following code snippet illustrates how this process works in practice:

```python

from transformers import AutoModelForTokenClassification, AutoTokenizer, pipeline

# Load tokenizer and model

model = AutoModelForTokenClassification.from\_pretrained("bert-base-cased-finetuned-pos")

tokenizer = AutoTokenizer.from\_pretrained("bert-base-cased-finetuned-pos")

# Setup POS tagging pipeline

pos\_pipeline = pipeline("token-classification", model=model, tokenizer=tokenizer)

# Example sentence

sentence = "The quick brown fox jumps over the lazy dog."

# Perform POS tagging

pos\_results = pos\_pipeline(sentence)

# Display POS tags

print("POS Tags:")

for token in pos\_results:

print(f"Word: {token['word']}, POS Tag: {token['entity']}")

```

In this example, the model setup involves loading a pre-trained BERT model that has been fine-tuned explicitly for POS tagging. The POS tagging pipeline simplifies the token classification task by analyzing each word in a sentence and tagging it with its corresponding part of speech. As the model processes the input sentence, it outputs POS tags, demonstrating its ability to identify and label each word's part of speech accurately.

This section offers a comprehensive understanding of POS tagging, detailing its technical mechanisms and practical applications. It provides a hands-on coding example that demonstrates how to implement this task using advanced models, such as BERT. By using the Hugging Face transformers library, we can streamline complex NLP tasks, making them more accessible and efficient in real-world applications.

Model training and evaluation

Training and evaluating sequence labeling models effectively are crucial for achieving high performance in tasks such as NER and POS tagging. This section outlines the best practices for training these models and the methodologies used to evaluate their performance.

Best practices for training sequence labeling models

Training sequence labeling models requires a careful approach to ensure that the model not only learns to predict the correct labels but also generalizes well to new, unseen data.

Developing a robust sequence labeling model involves more than selecting an algorithm; it requires a disciplined methodological pipeline that integrates data curation, feature engineering, and iterative optimization. Each stage, from preparing the dataset to fine-tuning hyperparameters, affects not only the model’s immediate performance but also its long-term adaptability across domains and languages. The following best practices outline key considerations that collectively define a high-performing, generalizable sequence labeling system:

* **Data preparation**: Quality and quantity of training data significantly affect model performance. It is crucial to use a well-annotated, diverse dataset that stands for the variability of language in real-world scenarios.
* **Feature selection**: Choosing the correct set of features is vital. For traditional models like CRFs, manually crafted features such as word suffixes, prefixes, and POS tags are useful. For neural models, embeddings that capture semantic meaning, such as word2vec or GloVe, enhance the model's understanding.
* **Regularization and dropout**: To prevent overfitting, especially in deep learning models, techniques like L2 regularization and dropout are employed during training.
* **Transfer learning**: Using pre-trained models and fine-tuning them on specific tasks can drastically improve performance due to the pre-learned contextual representations in the model. [6]

Together, these practices form the structural backbone of effective sequence labeling pipelines. When combined, they ensure that a model not only captures linguistic patterns during training but also maintains resilience when exposed to new, unseen text. However, even a meticulously trained model must be rigorously assessed to verify its reliability in real-world applications. The following section introduces the core evaluation methodologies, such as accuracy, precision, recall, and F1-score, that quantify model performance and guide iterative refinement across diverse NLP tasks, such as NER and POS tagging.

NER and POS tagging systems performance evaluation

When training sequence labeling models, it is essential to follow best practices that ensure robust performance and adaptability across various real-world datasets. These practices not only help the model learn correct label predictions but also improve its ability to generalize effectively to unseen data.

Evaluating sequence labeling models is an indispensable step in verifying their accuracy, consistency, and domain robustness. A well-trained model must not only perform optimally on its training data but also generalize to new contexts and linguistic variations. Evaluation metrics provide the quantitative foundation for this verification process, offering clear and comparable benchmarks that reveal both the strengths and limitations of a model’s predictions. In natural language processing, such metrics also guide iterative improvement, supporting the refinement of model architectures, data pipelines, and hyperparameter configurations. The following evaluation methodologies are central to assessing the effectiveness of NER and POS tagging systems:

* **Accuracy**: The most straightforward metric, accuracy measures the proportion of correctly predicted labels over all predictions.
* **Precision, recall, and F1-Score**: Precision measures the accuracy of optimistic predictions, recall measures the coverage of actual positive cases, and the F1-score provides a balance between precision and recall.
* **Confusion matrix**: Provides a detailed breakdown of predictions versus actual labels, helping name specific areas where the model is underperforming.

By systematically applying these metrics, practitioners can gain a multidimensional view of a model’s behavior, capturing not only how often it is correct, but also where and why it errs. Such analysis informs targeted retraining, dataset rebalancing, or architectural adjustments that incrementally enhance system reliability. With a validated understanding of performance, sequence labeling models can then be confidently deployed across practical domains, from biomedical text mining to social media analytics. The following section illustrates how these principles are embodied in real-world applications and code-based training scenarios.

The following picture shows a workflow for fine-tuning a pre-trained BERT model for Named Entity Recognition (NER) using the Hugging Face Transformers library. The diagram illustrates each step of the process, from dataset loading and token-label alignment to model training and evaluation. This visualization illustrates how tokenization, contextual embeddings, and supervised learning combine to create a model that can identify entities, people, organizations, and locations within text sequences.

A diagram of a workflow

AI-generated content may be incorrect.

Figure 4.4 Training and Evaluating a BERT-Based NER Model

Examples of application

NER and POS tagging are powerful tools that support a wide range of real-world applications, from biomedical research to understanding the language of social media. These applications demonstrate the versatility and importance of NLP techniques in extracting meaningful information, enabling advanced insights across various domains.

Quantitative evaluation alone does not fully capture a model’s value. The accurate measure of a sequence labeling system lies in its applicability, how effectively it translates theoretical performance into practical outcomes. By deploying these models across diverse contexts, researchers and practitioners can observe how well they adapt to noisy, domain-specific, or multilingual data. The following examples illustrate how Named Entity Recognition (NER) and Part-of-Speech (POS) tagging operate within real-world workflows, highlighting their contributions to fields such as biomedical text analysis, social media monitoring, and intelligent information retrieval. The following are specific examples that show the practical use of NER and POS tagging, as well as an illustrative case of training and evaluating an NER model:

* **Application**: **NER on biomedical text**: In the biomedical field, NER systems are vital for extracting critical medical entities such as drug names, symptoms, and diseases from clinical texts. These extractions play a crucial role in enhancing patient care, advancing medical research, and supporting the discovery of new drugs. For example, an NER system could parse electronic health records to find mentions of specific medications or conditions, enabling data-driven decision-making in healthcare settings.
* **Application**: **POS Tagging on social media text**: Social media text is often informal, having slang, abbreviations, and emoticons. Applying POS tagging in this domain provides structural insights into the language, enabling tasks such as sentiment analysis, content filtering, and linguistic research. By assigning parts of speech to informal language elements, researchers and practitioners can gain a deeper understanding of communication trends, helping to refine algorithms for tasks such as automated moderation or opinion mining.
* **Application**: **Training and evaluating an NER model**: This example illustrates the process of training and evaluating an NER model using the Hugging Face transformers library. In this scenario, the goal is to adapt a pre-trained BERT model for token classification to identify entities, such as names or locations, in text data. The process begins by loading a labeled dataset and preparing the data using the BERT tokenizer. Labels are aligned with the tokenized inputs to ensure consistency during training. The model is then fine-tuned with a specified number of epochs and batch size. After training, the model is evaluated using metrics such as precision, recall, and F1 Score to assess its effectiveness in accurately naming entities.

This example demonstrates how to train and evaluate a Named Entity Recognition (NER) model using the **Hugging Face Transformers** library.

The goal is to adapt a pre-trained BERT model for token-level classification to identify entities, such as names or locations, in text data.

The workflow proceeds through three logical phases: data preparation, model configuration, and training and evaluation.

**Step 1 — Data Loading and Tokenization**

We begin by loading a labeled dataset and preparing it for token-level classification:

```python

from transformers import BertTokenizer, BertForTokenClassification

from datasets import load\_dataset

tokenizer = BertTokenizer.from\_pretrained("bert-base-cased")

model = BertForTokenClassification.from\_pretrained("bert-base-cased", num\_labels=9)

dataset = load\_dataset("conll2003")

def tokenize\_and\_align\_labels(examples):

tokens = tokenizer(

examples["tokens"],

truncation=True,

padding="max\_length",

is\_split\_into\_words=True

)

labels = []

for i, lbl in enumerate(examples["ner\_tags"]):

word\_ids = tokens.word\_ids(batch\_index=i)

labels.append([lbl[w] if w is not None else -100 for w in word\_ids])

tokens["labels"] = labels

return tokens

dataset = dataset.map(tokenize\_and\_align\_labels, batched=True)```

This preprocessing step ensures that every token aligns correctly with its label, preserving sequence integrity for supervised training.

**Step 2 — Training Configuration**

Next, we define the core hyperparameters and initialize the trainer.

```python

from transformers import Trainer, TrainingArguments

args = TrainingArguments(

output\_dir="./results",

num\_train\_epochs=3,

per\_device\_train\_batch\_size=16,

learning\_rate=2e-5

)

trainer = Trainer(

model=model,

args=args,

train\_dataset=dataset["train"],

eval\_dataset=dataset["validation"]

)

```

**Step 3 — Model Training and Evaluation**

Finally, we train and evaluate the fine-tuned model.

```python

trainer.train()

```

After training, evaluation metrics such as **precision**, **recall**, and **F1-score** quantify the model’s performance in recognizing and classifying named entities. This workflow illustrates the end-to-end process of adapting a pre-trained BERT model for NER tasks, demonstrating how data preparation, token alignment, and fine-tuning converge into a practical and reproducible pipeline.

*Full implementation available at:* ***GitHub Repository – Chapter 4 | NER Fine-Tuning Example***

In the complete code, we first prepare and tokenize the dataset using a BERT tokenizer, ensuring the labels are correctly aligned with the tokenized inputs through the function tokenize\_and\_align\_labels. This preprocessing step is essential for training the model with the correct label mappings. Next, we set up a pre-trained BERT model that has been fine-tuned for token classification tasks, including configuring the model to manage the specific number of labels in the dataset. The training process begins by specifying key parameters, such as the number of epochs and the batch size, after which the model is evaluated using metrics like precision, recall, and F1-score to assess its performance in recognizing and classifying named entities. This streamlined workflow outlines the process of building and optimizing an NER model using Hugging Face Transformers, offering both a theoretical and practical understanding of how these systems operate in real-world applications. It not only provides an in-depth knowledge of training and evaluating sequence labeling models but also illustrates their practical applications in real-world scenarios, supported by a coding example.

Conclusion

In this chapter, we thoroughly explored the fundamentals of sequence labeling in natural language processing (NLP), focusing on core tasks such as Named Entity Recognition (NER) and Part-of-Speech (POS) tagging. We began by examining the role of sequence labeling in structuring and analyzing raw text data, followed by an in-depth discussion of its practical applications across various domains, including healthcare, media, and finance. Throughout the chapter, we demonstrated how advanced architectures, particularly transformer-based models, have enhanced the precision, contextual awareness, and scalability of these tasks. Practical coding examples using the Hugging Face Diffusers library illustrated real-world implementation, solidifying an operational understanding of how sequence labeling models can be designed, trained, and evaluated.

Our goal was to equip readers with both conceptual clarity and practical skills, enabling them to develop models that seamlessly transition from theory to production-ready applications. With the methods, examples, and evaluation frameworks discussed, this chapter establishes a solid foundation for addressing the complexities of modern NLP pipelines.

As we move forward to **Chapter 5: Advanced Fine-Tuning and Domain Adaptation**, we will extend these principles by exploring how pre-trained models can be optimized for specialized datasets and niche applications. The next chapter will examine transfer-learning strategies, task-specific adaptation techniques, and performance-stabilization practices that elevate sequence labeling and other NLP systems from generalized capability to domain-expert precision.

**Here, I added an alternative to maintain consistency across my work. And it is just select and delete id you do not see fit.**

**Key Takeaways**

* **Sequence labeling** forms the foundation of structured language understanding, enabling critical NLP tasks such as NER and POS tagging.
* **Classical models** (HMMs and CRFs) introduced statistical rigor in sequence prediction, while **neural architectures** (RNNs, LSTMs, and transformers) expanded contextual depth and scalability.
* **Best-practice workflows**—including data curation, feature selection, regularization, and transfer learning—ensure model generalization across real-world text.
* **Evaluation metrics** such as accuracy, precision, recall, and F1-score provide transparent criteria for validating model performance.
* **Applications** in healthcare, media, and social analytics demonstrate the operational relevance of sequence labeling beyond theoretical boundaries.
* **Hands-on examples** using Hugging Face Diffusers show how to implement, train, and optimize modern sequence labeling pipelines.

**Skills Acquired**

* Implementing NER and POS tagging using transformer-based architectures.
* Fine-tuning pre-trained models for domain-specific text.
* Designing and evaluating NLP workflows with Hugging Face Diffusers.
* Applying quantitative evaluation and interpretability methods for production-grade models.

**Practical Tools Introduced**

* **Libraries:** Hugging Face Transformers, Diffusers, sklearn-crfsuite, and supporting NLP packages.
* **Techniques:** CRF modeling, transformer fine-tuning, token classification pipelines, and evaluation with precision/recall/F1 metrics.
* **Datasets:** CoNLL-2003 and domain-specific corpora for sequence labeling experiments.

References

1. M. A. Lafferty and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in Proceedings of the Eighteenth International Conference on Machine Learning, 2001.
2. L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," in Proceedings of the IEEE, 1989.
   1. McCallum, "Information extraction: Distilling structured data from unstructured text," Queue, vol. 3, p. 48–57, 2005.
3. Jagannatha and H. Yu, "Structured prediction models for RNN-based sequence labeling in clinical text," in Proceedings of the Conference on Empirical Methods in Natural Language Processing, 2016.
4. K. C. G. Mikolov and J. Dean, "Efficient estimation of word representations in vector space," 2013.
5. J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," NAACL-HLT, 2018.
6. D. Manning, "Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics?," in Computational Linguistics and Intelligent Text Processing, 2011.
7. Z. Huang, W. Xu and K. Yu, "Bidirectional LSTM-CRF Models for Sequence Tagging," preprint, 2015.
8. J. Pennington, R. Socher and C. D. Manning, "Glove: Global vectors for word representation," in Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), 2014.
9. B. Pang and L. Lee, "Opinion mining and sentiment analysis," Foundations and Trends® in Information Retrieval, vol. 2, p. 1–135, 2008.
10. D. Manning, "The Stanford CoreNLP Natural Language Processing Toolkit," Association for Computational Linguistics (ACL) System Demonstrations, p. 55–60, 2014.
11. G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami and C. Dyer, "Neural architectures for named entity recognition," preprint, 2016.
12. M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," preprint, 2017.