Inference and Encoders Guidebook

Prof. Dr. Ivan Habernal
Yassine Thlija

2025-12-02

1 Quick Overview

This document provides a formal and task-agnostic description of how inference is conducted using encoder-based Transformer models (e.g., BERT, RoBERTa, DistilBERT). It explains:

- The functional role of the tokenizer and the model
- The mathematical meaning of logits
- The conversion of logits into probabilities
- The derivation of predictions and confidence scores

The guide applies to:

- Sequence classification
- Token classification (e.g., Named Entity Recognition)
- Extractive question answering

---

2 Core Components of the Inference Pipeline

Inference with encoder-based Transformers relies on three fundamental components.

2.1 Tokenizer

The tokenizer maps raw text to numerical tensors:

- `input_ids`: Token indices
- `attention_mask`: Binary mask for padding
- `token_type_ids` (optional): Segment identifiers

Formally:

$$\text{Text} \xrightarrow{\text{Tokenizer}} \text{Model Inputs (Tensors)}$$

2.2 Encoder Model

The encoder model applies stacked self-attention layers to the tokenized input and produces unnormalized prediction scores (logits). The model itself does not produce probabilities or labels.

2.3 Post-Processing Layer

Post-processing converts logits into:

- Probabilities
- Predicted labels
- Confidence scores

This stage is user-defined during manual inference.

3 Model Output Structure

A forward pass is executed as:

```
outputs = model(**inputs)
```

The most relevant fields are:

- `outputs.logits`: Raw, unnormalized prediction scores
- `outputs.hidden_states` (optional): Layer-wise contextual representations

Output dimensionality depends on the task:

Task Type	**Logit Shape**
Sequence Classification	$(batch\_size,\ num\_classes)$
Question Answering	$(batch\_size,\ sequence\_length)$ for both start and end
Token Classification	$(batch\_size,\ sequence\_length,\ num\_labels)$

4 Interpretation of Logits

Logits are real-valued scores defined over $\mathbb{R}$. They:

- Are not probabilities
- Are not bounded
- Express only *relative* preference between classes

A higher logit indicates stronger model preference but carries no probabilistic meaning without normalization.

5 Conversion of Logits to Probabilities

5.1 Softmax Normalization (Single-Label Classification)

For mutually exclusive classes:

```
probs = softmax(logits)
```

Properties:

- Probability values in $[0, 1]$
- Sum of probabilities equals 1

5.2 Sigmoid Normalization (Multi-Label Classification)

For independent labels:

```
probs = sigmoid(logits)
```

Each output value represents an independent Bernoulli probability.

6 Deriving Predictions

6.1 Single-Label Prediction

```
pred_id = argmax(probs)
confidence = probs[pred_id]
```

- `pred_id`: Index of the predicted class
- `confidence`: Maximum class probability

6.2 Multi-Label Prediction

```
preds = probs > threshold
```

Multiple labels may be activated simultaneously.

7 Label Decoding

Human-readable labels are obtained using the model configuration:

```
label = model.config.id2label[pred_id]
```

8 Universal Inference Workflow

The complete inference process is invariant across tasks:

Text
↓
Tokenization
↓
Encoder Forward Pass
↓
Logits
↓
Probability Normalization
↓
Prediction Selection
↓
Label Decoding + Confidence Estimation

9 Pipeline-Based vs Manual Inference

9.1 Pipeline-Based Inference

The pipeline API automates:

- Tokenization
- Device placement
- Forward computation
- Normalization
- Label decoding

It can expose for example:

```
Label + Confidence
```

9.2 Manual Inference

Manual inference requires explicit handling of:

- Tensors
- Device placement
- Logit normalization
- Thresholding
- Label mapping

It is required for:

- Scientific evaluation
- Debugging
- Custom model behavior
- Performance-critical deployment

10 Importing Models and Using Them

10.1 Essential Imports:

```
from transformers import (
    AutoTokenizer,           # for text tokenization
    AutoModelForSequenceClassification,  # for classification tasks
    AutoModelForTokenClassification,     # for NER, POS tagging
    AutoModelForQuestionAnswering,       # for QA tasks
    AutoModel,               # for base models without heads
    pipeline                 # for easy-to-use inference pipelines
)
```

10.2 Loading Models and Tokenizers

```
# example model from huggingface
model_name = "nlptown/bert-base-multilingual-uncased-sentiment"

# load tokenizer (handles text preprocessing)
tokenizer = AutoTokenizer.from_pretrained(model_name)

# load model (architecture depends on task)
model = AutoModelForSequenceClassification.from_pretrained(model_name)

# move to GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```