513317822513213522513318122513417 1

May 31, 2024

We need to install libraries below.

[50]:
```
!pip install transformers datasets evaluate seqeval tqdm
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-
packages (4.41.1)
Requirement already satisfied: datasets in /usr/local/lib/python3.10/dist-
packages (2.19.1)
Requirement already satisfied: evaluate in /usr/local/lib/python3.10/dist-
packages (0.4.2)
Requirement already satisfied: seqeval in /usr/local/lib/python3.10/dist-
packages (1.2.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(4.66.4)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from transformers) (3.14.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.0 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.23.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
packages (from transformers) (1.25.2)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from transformers) (24.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-
packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers) (2024.5.15)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from transformers) (2.31.0)
Requirement already satisfied: tokenizers<0.20,>=0.19 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.19.1)
Requirement already satisfied: safetensors>=0.4.1 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.4.3)
Requirement already satisfied: pyarrow>=12.0.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (14.0.2)
Requirement already satisfied: pyarrow-hotfix in /usr/local/lib/python3.10/dist-
packages (from datasets) (0.6)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (0.3.8)
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(from datasets) (2.0.3)
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packages
(from datasets) (3.4.1)
Requirement already satisfied: multiprocess in /usr/local/lib/python3.10/dist-
packages (from datasets) (0.70.16)
Requirement already satisfied: fsspec[http]<=2024.3.1,>=2023.1.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (2023.6.0)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-
packages (from datasets) (3.9.5)
Requirement already satisfied: scikit-learn>=0.21.3 in
/usr/local/lib/python3.10/dist-packages (from seqeval) (1.2.2)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-
packages (from aiohttp->datasets) (23.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (6.0.5)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-
packages (from aiohttp->datasets) (1.9.4)
Requirement already satisfied: async-timeout<5.0,>=4.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (4.0.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.23.0->transformers) (4.11.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->transformers) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers) (2024.2.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=0.21.3->seqeval) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=0.21.3->seqeval) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.21.3->seqeval)
(3.5.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->datasets) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->datasets) (2024.1)
```

```
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.2->pandas->datasets) (1.16.0)
```

0. CONFIGURATION

[88]:
```python
class DotDict(dict):
    """dot.notation access to dictionary attributes"""
    __getattr__ = dict.get
    __setattr__ = dict.__setitem__
    __delattr__ = dict.__delitem__


config = {
################################################## EDIT␣
 ↪##################################################
    'seed' : 2024,
    'batch_size' : 32,
    'lr' : 0.0001,
    'weight_decay' : 0.01,
    'hidden_size' : 512,
    'num_heads' : 8,
    'num_encoder_layers' : 6,
    'hidden_dropout_prob': 0.1,
    'use_lstm': True,
################################################## EDIT␣
 ↪##################################################
    'num_epochs' : 5, # NEVER TOUCH
    'vocab_size' : 30522, # NEVER TOUCH
    'pad_token_id' : 0, # NEVER TOUCH
    'num_labels' : 9, # NEVER TOUCH
}

config = DotDict(config)
```

[52]:
```python
import torch
import random
import warnings
warnings.filterwarnings('ignore')

import numpy as np

def set_seed(config):
    random.seed(config.seed)
    np.random.seed(config.seed)
    torch.manual_seed(config.seed)
    torch.cuda.manual_seed(config.seed)
    torch.cuda.manual_seed_all(config.seed)
    torch.backends.cudnn.benchmark = True
```

```
    torch.backends.cudnn.deterministic = True

set_seed(config)
```

1. DATASET

[53]:
```
from datasets import load_dataset

dataset = load_dataset("conll2003")
# dataset
```

The letter that prefixes each ner_tag indicates the token position of the entity:

B- indicates the beginning of an entity.

I- indicates a token is contained inside the same entity (for example, the State token is a part of an entity like Empire State Building).

0 indicates the token doesn't correspond to any entity.

[54]:
```
label_list = dataset['train'].features['ner_tags'].feature.names
config.num_labels = len(label_list)
# label_list
```

[55]:
```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained('distilbert/distilbert-base-uncased')
config.vocab_size = len(tokenizer.get_vocab())
config.pad_token_id = tokenizer.get_vocab()['[PAD]']
```

[56]:
```
example = dataset["test"][0]
# example
```

[57]:
```
tokenized_input = tokenizer(example["tokens"], is_split_into_words=True)
# tokenized_input
```

[58]:
```
tokens = tokenizer.convert_ids_to_tokens(tokenized_input["input_ids"])
# tokens
```

[87]:
```
def tokenize_and_align_labels(examples):
    tokenized_inputs = tokenizer(examples["tokens"], truncation=True,␣
 ↪is_split_into_words=True)
    labels = []

    for i, label in enumerate(examples[f"ner_tags"]):
        # Map tokens to their respective word.
        # Set the special tokens to -100.
        # Only label the first token of a given word.
```

```
        ################################################ EDIT␣
↪################################################

        word_ids = tokenized_inputs.word_ids(batch_index=i)
        label_ids = []
        previous_word_idx = None
        for word_idx in word_ids:
            if word_idx is None:
                label_ids.append(-100)
            elif word_idx != previous_word_idx:
                label_ids.append(label[word_idx])
            else:
                label_ids.append(-100)
            previous_word_idx = word_idx
        labels.append(label_ids)

        ################################################ EDIT␣
↪################################################

    tokenized_inputs["labels"] = labels
    return tokenized_inputs
```

[76]:
```
tokenized_dataset = dataset.map(tokenize_and_align_labels, batched=True,␣
↪remove_columns=dataset["train"].column_names)
```

```
Map:   0%|          | 0/14041 [00:00<?, ? examples/s]
Map:   0%|          | 0/3250 [00:00<?, ? examples/s]
Map:   0%|          | 0/3453 [00:00<?, ? examples/s]
```

[77]:
```
from transformers import DataCollatorForTokenClassification

data_collator = DataCollatorForTokenClassification(tokenizer=tokenizer)
```

[78]:
```
from torch.utils.data import DataLoader

train_dataloader = DataLoader(
    tokenized_dataset["train"],
    shuffle=True,
    collate_fn=data_collator,
    batch_size=config.batch_size,
)

valid_dataloader = DataLoader(
    tokenized_dataset["validation"],
    shuffle=True,
    collate_fn=data_collator,
```

```
    batch_size=config.batch_size,
)

test_dataloader = DataLoader(
    tokenized_dataset["test"],
    shuffle=False,
    collate_fn=data_collator,
    batch_size=config.batch_size,
)
```

2. METRIC

[79]:
```
import evaluate

seqeval = evaluate.load("seqeval")
```

[80]:
```
def compute_metrics(all_predictions, all_labels):
    predictions, labels = None, None

    # For evaluation, postprocess for shape.
    ################################################ EDIT␣
 ↪################################################

    predictions, labels = [], []

    for batch_preds, batch_labels in zip(all_predictions, all_labels):
        for preds, labs in zip(batch_preds, batch_labels):
            predictions.append(preds)
            labels.append(labs)

    ################################################ EDIT␣
 ↪################################################

    true_predictions = [
        [label_list[p] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]

    true_labels = [
        [label_list[l] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]

    results = seqeval.compute(predictions=true_predictions,␣
 ↪references=true_labels)
    return {
        "precision": results["overall_precision"],
```

```
        "recall": results["overall_recall"],
        "f1": results["overall_f1"],
        "accuracy": results["overall_accuracy"],
    }
```

3. MODEL

[81]:
```
from torch import nn

from typing import Optional


def _expand_mask(mask, tgt_len = None):
    """
        Inputs
            mask.shape = (B, S_L)
        Outputs
            output.shape = (B, 1, T_L, S_L)
    """
    batch_size, src_len = mask.size()
    tgt_len = tgt_len if tgt_len is not None else src_len

    expanded_mask = mask[:, None, None, :].expand(batch_size, 1, tgt_len,␣
 ↪src_len).to(torch.float)

    inverted_mask = 1.0 - expanded_mask

    return inverted_mask.masked_fill(inverted_mask.bool(), torch.finfo(torch.
 ↪float).min)


class PositionalEncoding(nn.Module):
    def __init__(self, hidden_size):
        super().__init__()
        self.encoding = torch.zeros(1024, hidden_size)
        self.encoding.requires_grad = False

        pos = torch.arange(0, 1024)
        pos = pos.float().unsqueeze(dim = 1)

        _2i = torch.arange(0, hidden_size, step = 2).float()

        ################################################ EDIT␣
 ↪################################################

        #       positional encoding
        self.encoding[:, 0::2] = torch.sin(pos / (10000 ** (_2i / hidden_size)))
```

```
        self.encoding[:, 1::2] = torch.cos(pos / (10000 ** (_2i / hidden_size)))

        ################################################ EDIT␣
↪################################################

    def forward(self, x):
        batch_size, seq_len = x.size()
        device = x.device

        return self.encoding[:seq_len, :].unsqueeze(0).to(device)


class MultiHeadAttention(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.hidden_size = config.hidden_size
        self.num_heads = config.num_heads
        self.d_head = self.hidden_size // self.num_heads
        self.scaling = self.d_head ** -0.5

        self.q_proj = nn.Linear(self.hidden_size, self.hidden_size)
        self.k_proj = nn.Linear(self.hidden_size, self.hidden_size)
        self.v_proj = nn.Linear(self.hidden_size, self.hidden_size)
        self.out_proj = nn.Linear(self.hidden_size, self.hidden_size)
        self.dropout = nn.Dropout(0.1)

    def _shape(self, tensor, seq_len, batch_size):
        return tensor.view(batch_size, seq_len, self.num_heads, self.d_head).
↪transpose(1, 2).contiguous()

    def forward(self, query_states, key_value_states, attention_mask):
        attn_output = None

        ################################################ EDIT␣
↪################################################

        # Query, key, value    projection
        query_states = self.q_proj(query_states) * self.scaling
        key_states = self.k_proj(key_value_states)
        value_states = self.v_proj(key_value_states)

        # Multi-head
        batch_size, seq_len, _ = query_states.size()
        query_states = self._shape(query_states, seq_len, batch_size)
        key_states = self._shape(key_states, seq_len, batch_size)
        value_states = self._shape(value_states, seq_len, batch_size)
```

```
        # Attention weight
        attn_weights = torch.matmul(query_states, key_states.transpose(-1, -2))
        expanded_attention_mask = _expand_mask(attention_mask, seq_len)  #
 ↪
        attn_weights = attn_weights + expanded_attention_mask
        attn_weights = torch.nn.functional.softmax(attn_weights, dim=-1)
        attn_weights = self.dropout(attn_weights)

        # Attention
        attn_output = torch.matmul(attn_weights, value_states)

        # Multi-head
        attn_output = attn_output.transpose(1, 2).contiguous().view(batch_size,
 ↪seq_len, self.hidden_size)

        # Output projection
        attn_output = self.out_proj(attn_output)

        ################################################## EDIT
 ↪##################################################

        return attn_output


class EncoderLayer(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.hidden_size = config.hidden_size

        self.self_attn = MultiHeadAttention(config)
        self.self_attn_layer_norm = nn.LayerNorm(self.hidden_size)

        self.activation_fn = nn.ReLU()

        self.fc1 = nn.Linear(self.hidden_size, 4 * self.hidden_size)
        self.fc2 = nn.Linear(4 * self.hidden_size, self.hidden_size)
        self.final_layer_norm = nn.LayerNorm(self.hidden_size)

        self.dropout = nn.Dropout(0.1)

    def forward(self, hidden_states, enc_self_mask):
        residual = hidden_states
        hidden_states = self.self_attn(
            query_states = hidden_states,
            key_value_states = hidden_states,
            attention_mask = enc_self_mask
        )
```

```
        hidden_states = self.dropout(hidden_states)
        hidden_states = residual + hidden_states
        hidden_states = self.self_attn_layer_norm(hidden_states)

        residual = hidden_states
        hidden_states = self.activation_fn(self.fc1(hidden_states))
        hidden_states = self.dropout(hidden_states)
        hidden_states = self.fc2(hidden_states)
        hidden_states = self.dropout(hidden_states)
        hidden_states = residual + hidden_states
        hidden_states = self.final_layer_norm(hidden_states)

        return hidden_states


class Encoder(nn.Module):
    def __init__(self, config, embed_tokens, embed_positions):
        super().__init__()
        self.hidden_size = config.hidden_size

        self.embed_tokens = embed_tokens
        self.embed_positions = embed_positions

        self.layers = nn.ModuleList([EncoderLayer(config) for _ in range(config.
 ↪num_encoder_layers)])
        self.embedding_layer_norm = nn.LayerNorm(self.hidden_size)

    def forward(self, enc_ids, enc_mask):
        enc_hidden_states = None

        ################################################ EDIT␣
 ↪################################################

        #   ID         positional encoding
        enc_hidden_states = self.embed_tokens(enc_ids) + self.
 ↪embed_positions(enc_ids)
        enc_hidden_states = self.embedding_layer_norm(enc_hidden_states)

        # Encoder   layer
        for layer in self.layers:
            enc_hidden_states = layer(enc_hidden_states, enc_mask)

        ################################################ EDIT␣
 ↪################################################
        return enc_hidden_states
```

```
class TransformerEncoder(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.embed_tokens = nn.Embedding(config.vocab_size, config.hidden_size,␣
 ↪config.pad_token_id)
        self.embed_positions = PositionalEncoding(config.hidden_size)
        self.encoder = Encoder(config, self.embed_tokens, self.embed_positions)

    def forward(self, enc_ids, enc_mask = None):
        enc_hidden_states = self.encoder(enc_ids, enc_mask)

        return enc_hidden_states


class LSTMCell(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.input_size = config.hidden_size
        self.hidden_size = config.hidden_size

        self.linear_ih = nn.Linear(self.input_size, 4 * self.hidden_size)
        self.linear_hh = nn.Linear(self.hidden_size, 4 * self.hidden_size)

    def forward(self, input, hidden):
        hy, cy = None, None

        ################################################ EDIT␣
 ↪################################################

        hx, cx = hidden

        # input  hidden state    linear transformation
        gates = self.linear_ih(input) + self.linear_hh(hx)

        #   gate    input
        ingate, forgetgate, cellgate, outgate = gates.chunk(4, 1)

        #
        ingate = torch.sigmoid(ingate)
        forgetgate = torch.sigmoid(forgetgate)
        cellgate = torch.tanh(cellgate)
        outgate = torch.sigmoid(outgate)


        #    cell state, hidden state
        cy = (forgetgate * cx) + (ingate * cellgate)
        hy = outgate * torch.tanh(cy)
```

```
        ############################################## EDIT␣
↪##################################################

        return hy, cy


class ModelForNER(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.transformer_encoder = TransformerEncoder(config)
        self.lstm = LSTMCell(config)

        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier = nn.Linear(config.hidden_size, config.num_labels)
        self.num_labels = config.num_labels
        self.use_lstm = config.use_lstm

    def forward(self,
                input_ids: Optional[torch.Tensor] = None,
                attention_mask: Optional[torch.Tensor] = None,
                ):

        logits = None

        ############################################## EDIT␣
↪##################################################

        if self.use_lstm:
            # LSTM     hidden states
            enc_hidden_states = self.transformer_encoder(input_ids,␣
↪attention_mask)
            batch_size = enc_hidden_states.size(0)
            hx, cx = torch.zeros(batch_size, self.hidden_size,␣
↪device=enc_hidden_states.device), torch.zeros(batch_size, self.hidden_size,␣
↪device=enc_hidden_states.device)
            outputs = []
            for t in range(enc_hidden_states.size(1)):
                hx, cx = self.lstm(enc_hidden_states[:, t, :], (hx, cx))
                outputs.append(hx)
            # LSTM    classifier
            logits = self.classifier(torch.stack(outputs, dim=1))
        else:
            # Transformer Encoder     classifier
            enc_hidden_states = self.transformer_encoder(input_ids,␣
↪attention_mask)
```

```
            logits = self.classifier(enc_hidden_states)

        ############################################## EDIT␣
↪#################################################

        return logits
```

[82]:
```
model = ModelForNER(config=config)
```

4. Optimizer & Scheduler

[83]:
```
from torch.optim import AdamW
from torch.nn import CrossEntropyLoss

optimizer = AdamW(model.parameters(), lr=config.lr,
                  weight_decay=config.weight_decay)
criterion = CrossEntropyLoss()
```

[84]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)
```

5. TRAINING

[89]:
```
from tqdm.auto import tqdm

for epoch in range(config.num_epochs):
    print(f'########## EPOCH: {epoch} ##########')
    model.train()

    # TRAINING LOOP
    for i, data in tqdm(enumerate(train_dataloader),
                        total=len(train_dataloader),
                        desc='TRAINING'):

        input_ids = data['input_ids'].to(device)
        attention_mask = data['attention_mask'].to(device)
        labels = data['labels'].to(device)

        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask)
        loss = criterion(outputs.view(-1, outputs.shape[-1]), labels.view(-1))

        loss.backward()
        optimizer.step()

    # VALIDATION LOOP
    model.eval()
```

```
    valid_loss = 0.0
    all_predictions, all_labels = [], []

    with torch.no_grad():
        for i, data in tqdm(enumerate(valid_dataloader),␣
 ↪total=len(valid_dataloader), desc='VALIDATION'):
            input_ids = data['input_ids'].to(device)
            attention_mask = data['attention_mask'].to(device)
            labels = data['labels'].to(device)

            outputs = model(input_ids, attention_mask)
            loss = criterion(outputs.view(-1, outputs.shape[-1]), labels.
 ↪view(-1))
            valid_loss += loss.item()

            all_predictions.append(np.argmax(outputs.cpu().numpy(), axis=2))
            all_labels.append(labels.cpu().numpy())

    print(f'Validation Loss: {valid_loss / len(valid_dataloader)}')

    metrics = compute_metrics(all_predictions, all_labels)
    print(f'Validation metrics: {metrics}')

print('Finished Training')
```

```
########## EPOCH: 0 ##########

TRAINING:   0%|          | 0/439 [00:00<?, ?it/s]

VALIDATION:   0%|          | 0/102 [00:00<?, ?it/s]

Validation Loss: 0.2937811746755067
Validation metrics: {'precision': 0.6755396851160526, 'recall':
0.700437563110064, 'f1': 0.6877633644550938, 'accuracy': 0.9390989447451423}
########## EPOCH: 1 ##########

TRAINING:   0%|          | 0/439 [00:00<?, ?it/s]

VALIDATION:   0%|          | 0/102 [00:00<?, ?it/s]

Validation Loss: 0.33233677318283156
Validation metrics: {'precision': 0.6692090863870005, 'recall':
0.6792325816223493, 'f1': 0.674183579721039, 'accuracy': 0.9380670534636502}
########## EPOCH: 2 ##########

TRAINING:   0%|          | 0/439 [00:00<?, ?it/s]

VALIDATION:   0%|          | 0/102 [00:00<?, ?it/s]

Validation Loss: 0.38266465155517354
Validation metrics: {'precision': 0.6796283551273228, 'recall':
```

```
0.6647593402894648, 'f1': 0.6721116215756339, 'accuracy': 0.9369962228885168}
########## EPOCH: 3 ##########

TRAINING:   0%|          | 0/439 [00:00<?, ?it/s]

VALIDATION:   0%|          | 0/102 [00:00<?, ?it/s]

Validation Loss: 0.34022429936072407
Validation metrics: {'precision': 0.656288532675709, 'recall':
0.7165937394816561, 'f1': 0.6851166532582461, 'accuracy': 0.9378723569954441}
########## EPOCH: 4 ##########

TRAINING:   0%|          | 0/439 [00:00<?, ?it/s]

VALIDATION:   0%|          | 0/102 [00:00<?, ?it/s]

Validation Loss: 0.38159268743851604
Validation metrics: {'precision': 0.6946910356832028, 'recall':
0.6716593739481656, 'f1': 0.6829810901001112, 'accuracy': 0.9398193216775048}
Finished Training
```

[91]:
```python
# TEST
model.eval()
all_predictions, all_labels = [], []

with torch.no_grad():
    for i, data in tqdm(enumerate(test_dataloader), total=len(test_dataloader),␣
 ↪desc='TEST'):
        input_ids = data['input_ids'].to(device)
        attention_mask = data['attention_mask'].to(device)
        labels = data['labels'].to(device)

        outputs = model(input_ids, attention_mask)
        loss = criterion(outputs.view(-1, outputs.shape[-1]), labels.view(-1))

        all_predictions.append(np.argmax(outputs.cpu().numpy(), axis=2))
        all_labels.append(labels.cpu().numpy())

metrics = compute_metrics(all_predictions, all_labels)
print(f'Test metrics: {metrics}')
```

```
TEST:   0%|          | 0/108 [00:00<?, ?it/s]

Test metrics: {'precision': 0.610944527736132, 'recall': 0.5771954674220963,
'f1': 0.5935906773488709, 'accuracy': 0.920727899213955}
```