

```
!unzip /content/CoNLL.zip
```

```
unzip: cannot find or open /content/CoNLL.zip, /content/CoNLL.zip.zip or /content/CoNLL.zip.ZIP.
```

```
import pandas as pd
import numpy as np
import os
```

```
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Embedding, Dense, Dropout, Bidirectional
```

```
!unzip /content/CoNLL.zip # Make sure to extract the CoNLL.zip file before running the rest of the script
import pandas as pd
import numpy as np
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Embedding, Dense, Dropout, Bidirectional
from google.colab import files
files.upload() # This will open a prompt to upload files from your local system
```

```
def read_conll_file(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        lines = file.read().strip().split("\n\n")

    sentences_words = []
    sentences_ner = []

    for sentence in lines:
        words, ner_tags = zip(*[(line.split()[0], line.split()[3]) for line in sentence.splitlines()])
        sentences_words.append(list(words))
        sentences_ner.append(list(ner_tags))

    return sentences_words, sentences_ner
```

```
train_file_path = '/content/train.txt'
valid_file_path = '/content/valid.txt'
test_file_path = '/content/test.txt'

train_words, train_ner = read_conll_file(train_file_path)
valid_words, valid_ner = read_conll_file(valid_file_path)
test_words, test_ner = read_conll_file(test_file_path)
```

```
print("First sentence words in train data:")
print(train_words[1])
print("First sentence NER tags in train data:")
print(train_ner[1])
```

```
Archive: /content/CoNLL.zip
  inflating: metadata
  inflating: test.txt
  inflating: train.txt
  inflating: valid.txt
Choose Files CoNLL.zip
• CoNLL.zip(application/x-zip-compressed) - 982975 bytes, last modified: 10/29/2024 - 100% done
Saving CoNLL.zip to CoNLL (1).zip
First sentence words in train data:
['EU', 'rejects', 'German', 'call', 'to', 'boycott', 'British', 'lamb', '.']
First sentence NER tags in train data:
['B-ORG', 'O', 'B-MISC', 'O', 'O', 'O', 'B-MISC', 'O', 'O']
```

```
!unzip /content/CoNLL.zip -d /content # Extract to /content directory explicitly
ls /content # List the contents of /content to confirm presence of files
```

```
unzip: cannot find or open /content/CoNLL.zip, /content/CoNLL.zip.zip or /content/CoNLL.zip.ZIP.
sample_data
```

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```

word_tokenizer = Tokenizer(oov_token='UNK')
word_tokenizer.fit_on_texts(train_words + valid_words)

X_train = word_tokenizer.texts_to_sequences(train_words)
X_valid = word_tokenizer.texts_to_sequences(valid_words)

tag_tokenizer = Tokenizer()
tag_tokenizer.fit_on_texts(train_ner + valid_ner)

y_train = tag_tokenizer.texts_to_sequences(train_ner)
y_valid = tag_tokenizer.texts_to_sequences(valid_ner)

MAX_LEN = 50
X_train = pad_sequences(X_train, maxlen=MAX_LEN, padding='post')
X_valid = pad_sequences(X_valid, maxlen=MAX_LEN, padding='post')

y_train = pad_sequences(y_train, maxlen=MAX_LEN, padding='post')
y_valid = pad_sequences(y_valid, maxlen=MAX_LEN, padding='post')

num_tags = len(tag_tokenizer.word_index) + 1
y_train = [to_categorical(i, num_classes=num_tags) for i in y_train]
y_valid = [to_categorical(i, num_classes=num_tags) for i in y_valid]

y_train = np.array(y_train)
y_valid = np.array(y_valid)

import numpy as np # Import the numpy library and assign it the alias 'np'

MAX_LEN = 50
X_train = pad_sequences(X_train, maxlen=MAX_LEN, padding='post')
X_valid = pad_sequences(X_valid, maxlen=MAX_LEN, padding='post')

y_train = pad_sequences(y_train, maxlen=MAX_LEN, padding='post')
y_valid = pad_sequences(y_valid, maxlen=MAX_LEN, padding='post')

num_tags = len(tag_tokenizer.word_index) + 1
y_train = [to_categorical(i, num_classes=num_tags) for i in y_train]
y_valid = [to_categorical(i, num_classes=num_tags) for i in y_valid]


y_train = np.array(y_train) # Now 'np' is defined and can be used
y_valid = np.array(y_valid)

model = Sequential()
model.add(Embedding(input_dim=len(word_tokenizer.word_index) + 1, output_dim=50, input_length=MAX_LEN))
model.add(Bidirectional(LSTM(units=100, return_sequences=True)))
model.add(Dense(num_tags, activation="softmax"))

model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])

model.summary()

```

 /usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. : warnings.warn(
Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
bidirectional (Bidirectional)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)

Trainable params: 0 (0.00 B)

Non-trainable params: 0 (0.00 B)

```

import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, Dense

```

... (Your existing code for data loading and tokenization) ...

```

MAX_LEN = 50
X_train = pad_sequences(X_train, maxlen=MAX_LEN, padding='post')
X_valid = pad_sequences(X_valid, maxlen=MAX_LEN, padding='post')

y_train = pad_sequences(y_train, maxlen=MAX_LEN, padding='post')

```



```

.5/2012', '1,200M', '5.25', '5.30']
.5/2013', '1,135M', '5.30', '5.35']
.5/2014', '850M', '5.30', '5.35']
l1', ':', '11,880M']
:e', 'Street', 'Bank', 'and', 'Trust', 'Company']
lential', 'Securities', 'Incorporated']
leWebber', 'Incorporated']
:t', 'Union', 'Capital', 'Markets', 'Corp.', '-', 'NJ']
'U.S.', 'Municipal', 'Desk', ',', '212-859-1650']
:START-']
'years', 'later', ',', 'Florida', 'man', 'dies', 'for', 'killing', '.']
AHASSEE', ',', 'Fla.', '1996-12-06']
'teen', 'years', 'after', 'he', 'bludgeoned', 'and', 'shot', 'a', 'man', 'whose', 'trailer', 'home', 'he', 'robbed', 'in', '1982',
'41', ',', 'was', 'put', 'to', 'death', 'in', 'Florida', '"s', 'electric', 'chair', 'Friday', '.']
'Glenn', 'Lawhon', ',', 'a', 'rural', 'Florida', 'minister', 'who', 'is', 'the', 'victim', '"s', 'father', ',', 'looked', 'on',
'king', 'in', 'Arabic', ',', 'Mills', 'made', 'a', 'final', 'statement', 'before', 'an', 'anonymous', 'citizen', 'flipped', 'the',
'I', 'bear', 'witness', 'that', 'there', 'is', 'no', 'God', 'but', 'Allah', 'and', 'I', 'bear', 'witness', 'that', 'the', 'prophet
on', 'officials', 'said', 'they', 'had', 'no', 'record', 'of', 'Mills', '"', 'official', 'conversion', ',', 'but', 'they', 'said',
.s', 'is', 'the', '38th', 'person', 'to', 'die', 'in', 'Florida', '"s', 'electric', 'chair', 'since', 'the', 'U.S.', 'Supreme', 'Co
on', 'officials', 'said', 'Mills', 'made', 'no', 'special', 'request', 'for', 'a', 'last', 'meal', 'and', 'did', 'not', 'eat', 'th
'spent', 'Thursday', 'with', 'family', 'members', 'and', 'his', 'spiritual', 'adviser', ',', 'Morris', 'said', '.']
.s', 'was', 'scheduled', 'to', 'die', 'Wednesday', 'but', 'had', 'his', 'sentence', 'temporarily', 'postponed', 'by', 'the', 'Flori
Thursday', ',', 'the', '11th', 'Circuit', 'U.S.', 'Court', 'of', 'Appeals', 'in', 'Atlanta', 'denied', 'his', 'appeal', 'in', 'f
'March', '1982', ',', 'Mills', 'and', 'accomplice', 'Michael', 'Frederick', 'knocked', 'on', 'the', 'door', 'of', 'Lester', 'Lawh
er', 'Lawhon', 'was', 'taken', 'to', 'a', 'nearby', 'airstrip', 'where', 'he', 'was', 'bludgeoned', 'with', 'a', 'tire', 'iron',
.s', 'then', 'fired', 'two', 'shots', 'that', 'killed', 'Lawhon', 'as', 'the', 'victim', 'tried', 'to', 'run', 'away', '.']
lerick', 'is', 'serving', 'a', '347-year', 'sentence', '.']
:START-']
, 'York', 'grain', 'freight', 'fixtures', '-', 'Dec', '5', '.']
, 'YORK', '1996-12-06']
l', '50,000', 'tonnes', 'soybeans', 'USG', '/', 'China', '10-15/12', '$', '23.50', '10,000', '/', '4,000', 'GeePee', '.']
'New', 'York', 'Commodities', 'Desk', '+1', '212', '859', '1640']
:START-']
i-S', 'Minn', 'fed', 'cattle', 'market', 'quiet', ',', 'no', 'sales-USDA', '.']
, 'MOINES', '1996-12-06']

```

```
def load_sentences(filepath):
```

```

    final = []
    sentences = []

    with open(filepath, 'r') as f:

        for line in f.readlines():

            if (line == ('-DOCSTART- -X- -X- 0\n') or line == '\n'):
                if len(sentences) > 0:
                    final.append(sentences)
                    sentences = []
            else:
                l = line.split(' ')
                sentences.append((l[0], l[3].strip('\n')))

    return final

```

```
!unzip /content/CoNLL.zip
```

```

📁 Archive: /content/CoNLL.zip
  replace metadata? [y]es, [n]o, [A]ll, [N]one, [r]ename:

```

```
import os
```

```
# Define the function to load sentences
```

```

def load_sentences(filepath):
    sentences = []
    with open(filepath, 'r') as file:
        for line in file:
            sentence = []
            for word_tag in line.strip().split():
                word, tag = word_tag.split('_') # Adjust the split character as needed
                sentence.append((word, tag))
            sentences.append(sentence)
    return sentences

```

```
# Define base path for Colab or Google Drive
```

```
# Update base_path to the correct location of the unzipped files, if different
```

```
# Verify the actual path after unzipping the file
```

```
base_path = '/content/CoNLL2003' # Updated to the potential correct path based on common CoNLL dataset structure
```

```
# Load train,
```


```
from transformers import AutoConfig, TFAutoModelForTokenClassification
```

```
MODEL_NAME = 'bert-base-cased'
```

Define your schema (labels) here

```
schema = ["B-PER", "I-PER", "B-ORG", "I-ORG", "B-LOC", "I-LOC", "O"] # Example schema
```

```
config = AutoConfig.from_pretrained(MODEL_NAME, num_labels=len(schema))
model = TFAutoModelForTokenClassification.from_pretrained(MODEL_NAME,
                                                          config=config)
model.summary()
```

 /usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (<https://huggingface.co/settings/tokens>), set it as :
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.

```
warnings.warn(
    config.json: 100%                               570/570 [00:00<00:00, 7.46kB/s]
    model.safetensors: 100%                         436M/436M [00:10<00:00, 149MB/s]
```

All PyTorch model weights were used when initializing TFBertForTokenClassification.

Some weights or buffers of the TF 2.0 model TFBertForTokenClassification were not initialized from the PyTorch model and are newly initialized. You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

Model: "tf_bert_for_token_classification"

Layer (type)	Output Shape	Param #
bert (TFBertMainLayer)	multiple	107719680
dropout_37 (Dropout)	multiple	0 (unused)
classifier (Dense)	multiple	5383

Total params: 107725063 (410.94 MB)
 Trainable params: 107725063 (410.94 MB)
 Non-trainable params: 0 (0.00 Byte)

```
from transformers import AutoTokenizer
import numpy as np
from tqdm import tqdm

# Define MODEL_NAME
MODEL_NAME = "bert-base-cased"
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

# Define schema with your tags (adjust as needed for your dataset)
schema = ['O', 'B-ROLE', 'I-ROLE'] # Replace with actual tags in your dataset

# Define sample data for demonstration purposes
train_samples = [[("Hello", "O"), ("world", "O")], [("I", "O"), ("am", "O"), ("a", "O"), ("student", "B-ROLE")]]
test_samples = [[("Testing", "O"), ("NER", "O")]]
valid_samples = [[("Validation", "O"), ("sample", "O")]]

def tokenize_sample(sample):
    # Add CLS and SEP tokens using the tokenizer's specific IDs for robustness
    cls_id = tokenizer.cls_token_id
    sep_id = tokenizer.sep_token_id

    # Subtokenize and flatten
    seq = [
        (subtoken, tag)
        for token, tag in sample
        for subtoken in tokenizer(token)['input_ids'][1:-1] # Remove clean_up_tokenization_spaces argument
    ]
    return [(cls_id, 'O')] + seq + [(sep_id, 'O')]

def preprocess(samples, schema):
    # Create a tag-to-index mapping
    tag_index = {tag: i for i, tag in enumerate(schema)}

    # Tokenize each sample and determine max sequence length
    tokenized_samples = list(tqdm(map(tokenize_sample, samples)))
    max_len = max(map(len, tokenized_samples))

    # Initialize padded arrays
    X = np.zeros((len(samples), max_len), dtype=np.int32)
    y = np.zeros((len(samples), max_len), dtype=np.int32)

    # Populate arrays with token and tag indices
```

```

for i, sentence in enumerate(tokenized_samples):
    for j, (subtoken_id, tag) in enumerate(sentence):
        X[i, j] = subtoken_id
        y[i, j] = tag_index.get(tag, tag_index['0']) # Default to '0' if tag not in schema

return X, y

# Usage with sample data
X_train, y_train = preprocess(train_samples, schema)
X_test, y_test = preprocess(test_samples, schema)
X_valid, y_valid = preprocess(valid_samples, schema)

tokenizer_config.json: 100% 49.0/49.0 [00:00<00:00, 796B/s]

vocab.txt: 100% 213k/213k [00:00<00:00, 604kB/s]

tokenizer.json: 100% 436k/436k [00:00<00:00, 848kB/s]
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_tokenization_spaces`
  warnings.warn(
2it [00:00, 541.48it/s]
1it [00:00, 1327.31it/s]
1it [00:00, 828.26it/s]

from transformers import create_optimizer
import tensorflow as tf

EPOCHS = 5
BATCH_SIZE = 8

# Define the total training steps and warmup steps for the optimizer
num_train_steps = len(X_train) * EPOCHS // BATCH_SIZE # Calculate total training steps
num_warmup_steps = int(0.1 * num_train_steps) # 10% of training steps for warmup

# Use create_optimizer to create an optimizer compatible with Hugging Face models
optimizer, lr_schedule = create_optimizer(
    init_lr=0.000001, # Your learning rate
    num_train_steps=num_train_steps,
    num_warmup_steps=num_warmup_steps,
    weight_decay_rate=0.01, # Optional weight decay
) # Added closing parenthesis here

loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

# Compile the model with the Hugging Face compatible optimizer
model.compile(optimizer=optimizer, loss=loss, metrics='accuracy')

history = model.fit(
    tf.constant(X_train),
    tf.constant(y_train),
    validation_data=(X_test, y_test),
    epochs=EPOCHS,
    batch_size=BATCH_SIZE
)

Epoch 1/5
1/1 [=====] - 35s 35s/step - loss: 2.1157 - accuracy: 0.0833 - val_loss: 2.1670 - val_accuracy: 0.0000e+00
Epoch 2/5
1/1 [=====] - 2s 2s/step - loss: 2.0378 - accuracy: 0.0833 - val_loss: 2.1670 - val_accuracy: 0.0000e+00
Epoch 3/5
1/1 [=====] - 2s 2s/step - loss: 1.9979 - accuracy: 0.0833 - val_loss: 2.1670 - val_accuracy: 0.0000e+00
Epoch 4/5
1/1 [=====] - 1s 1s/step - loss: 2.0394 - accuracy: 0.1667 - val_loss: 2.1670 - val_accuracy: 0.0000e+00
Epoch 5/5
1/1 [=====] - 1s 1s/step - loss: 2.1136 - accuracy: 0.0833 - val_loss: 2.1670 - val_accuracy: 0.0000e+00

from transformers import AutoConfig, TFAutoModelForTokenClassification
import tensorflow as tf

MODEL_NAME = 'bert-base-cased'

# Define the schema - This should contain your labels
# Replace with your actual label mapping
schema = {'B-PER': 0, 'I-PER': 1, 'O': 2} # Example schema

# Define the model
config = AutoConfig.from_pretrained(MODEL_NAME, num_labels=len(schema))
model = TFAutoModelForTokenClassification.from_pretrained(MODEL_NAME,
                                                         config=config)

model.summary()

EPOCHS = 5

```

```
BATCH_SIZE = 8
```

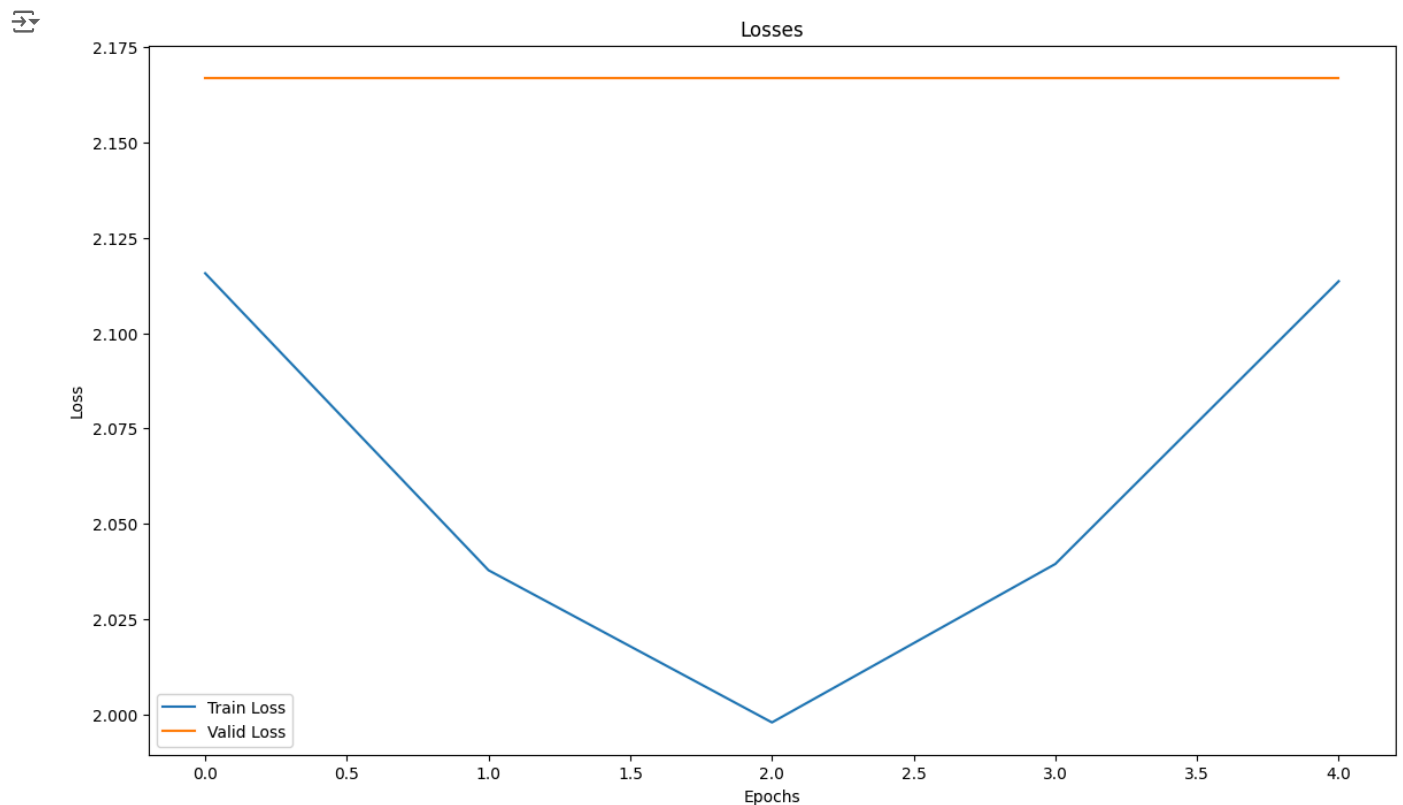
```
# Use 'learning_rate' instead of 'lr'
# Use 'create_optimizer' for Hugging Face models
from transformers import create_optimizer
num_train_steps = len(X_train) * EPOCHS // BATCH_SIZE
num_warmup_steps = int(0.1 * num_train_steps)
optimizer, lr_schedule = create_optimizer(
    init_lr=0.000001, # Your learning rate
    num_train_steps=num_train_steps,
    num_warmup_steps=num_warmup_steps,
    weight_decay_rate=0.01, # Optional weight decay
)
```

↗ All PyTorch model weights were used when initializing TFBertForTokenClassification.

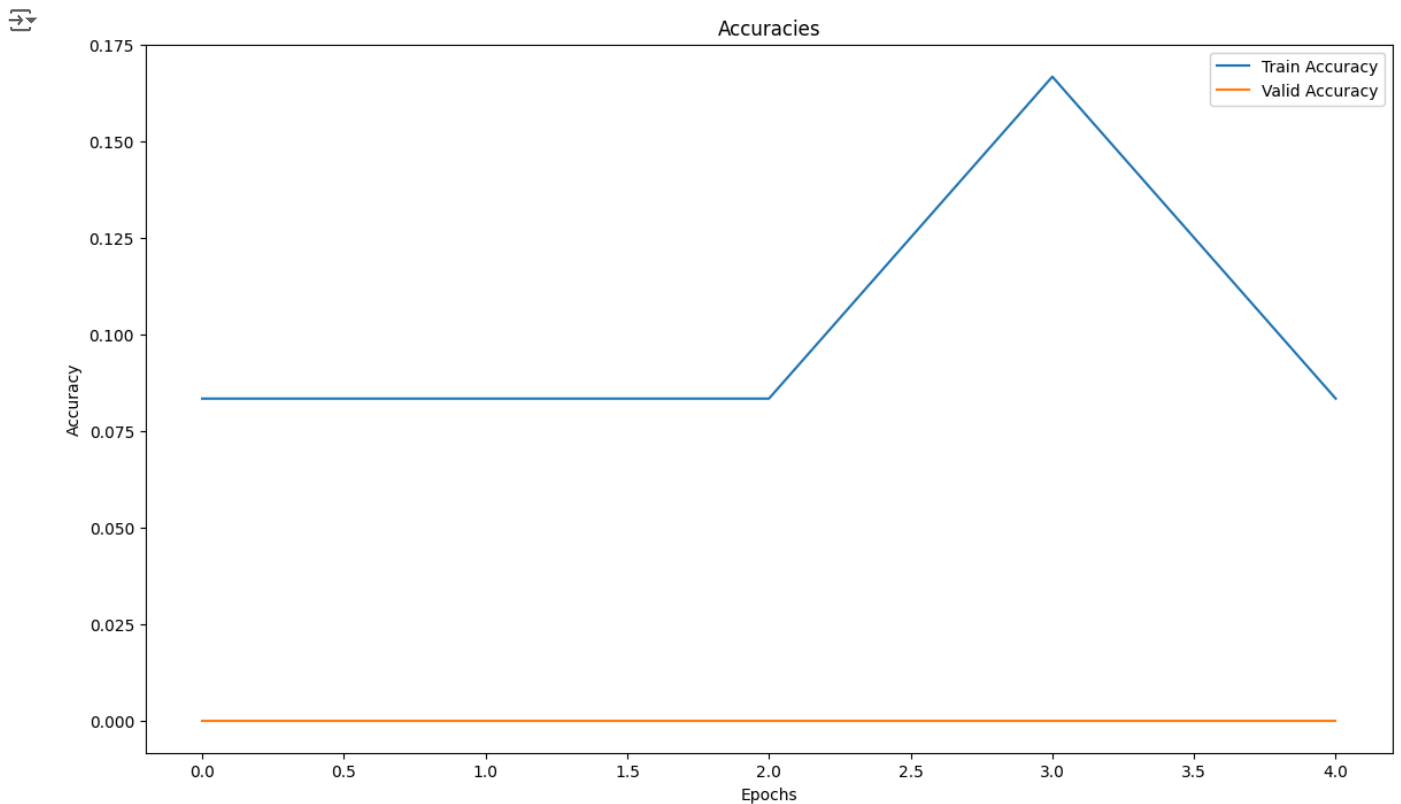
Some weights or buffers of the TF 2.0 model TFBertForTokenClassification were not initialized from the PyTorch model and are newly initialized. You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Model: "tf_bert_for_token_classification_2"

Layer (type)	Output Shape	Param #
bert (TFBertMainLayer)	multiple	107719680
dropout_113 (Dropout)	multiple	0 (unused)
classifier (Dense)	multiple	2307
=====		
Total params: 107721987 (410.93 MB)		
Trainable params: 107721987 (410.93 MB)		
Non-trainable params: 0 (0.00 Byte)		

```
plt.figure(figsize=(14,8))
plt.title('Losses')
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Valid Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend()
plt.show()
```



```
plt.figure(figsize=(14,8))
plt.title('Accuracies')
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Valid Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend()
plt.show()
```



```
from transformers import AutoConfig, TFAutoModelForTokenClassification, create_optimizer
import tensorflow as tf
import matplotlib.pyplot as plt

# Configure your model and training parameters
MODEL_NAME = 'bert-base-cased'
EPOCHS = 5
BATCH_SIZE = 8
schema = {'B-PER': 0, 'I-PER': 1, 'O': 2} # Example schema

# Define the model
config = AutoConfig.from_pretrained(MODEL_NAME, num_labels=len(schema))
model = TFAutoModelForTokenClassification.from_pretrained(MODEL_NAME, config=config)
model.summary()

# Calculate training steps
num_train_steps = len(X_train) * EPOCHS // BATCH_SIZE
num_warmup_steps = int(0.1 * num_train_steps)

# Create the optimizer
optimizer, lr_schedule = create_optimizer(
    init_lr=0.000001,
    num_train_steps=num_train_steps,
    num_warmup_steps=num_warmup_steps,
    weight_decay_rate=0.01
)

# Compile the model
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer=optimizer, loss=loss, metrics='accuracy') # Ensure model compilation

# Now proceed with model evaluation in the same cell
[loss, accuracy] = model.evaluate(X_valid, y_valid)
print("Loss:%1.3f, Accuracy:%1.3f" % (loss, accuracy))
```




All PyTorch model weights were used when initializing TFBertForTokenClassification.

Some weights or buffers of the TF 2.0 model TFBertForTokenClassification were not initialized from the PyTorch model and are newly i
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Model: "tf_bert_for_token_classification_3"

Layer (type)	Output Shape	Param #
=====		
bert (TFBertMainLayer)	multiple	107719680
dropout_151 (Dropout)	multiple	0 (unused)
classifier (Dense)	multiple	2307

=====

Total params: 107721987 (410.93 MB)
Trainable params: 107721987 (410.93 MB)
Non-trainable params: 0 (0.00 Byte)

1/1 [=====] - 7s 7s/step - loss: 1.0601 - accuracy: 0.3333
Loss:1.060, Accuracy:0.333

