


Keras

 Star 55,959

[About Keras](#)

[Getting started](#)

[Developer guides](#)

[Keras API reference](#)

Code examples

Computer Vision

Natural Language Processing

Structured Data

Timeseries

Audio Data

Generative Deep Learning

Reinforcement Learning

Graph Data

Quick Keras Recipes

[Why choose Keras?](#)

[Community & governance](#)

[Contributing to Keras](#)

[KerasTuner](#)

[KerasCV](#)

[KerasNLP](#)

» [Code examples](#) / [Natural Language Processing](#) / Named Entity Recognition using Transformers

Named Entity Recognition using Transformers

Author: [Varun Singh](#)
Date created: Jun 23, 2021
Last modified: Jun 24, 2021
Description: NER using the Transformers and data from CoNLL 2003 shared task.

 [View in Colab](#) ·  [GitHub source](#)

Introduction

Named Entity Recognition (NER) is the process of identifying named entities in text. Example of named entities are: "Person", "Location", "Organization", "Dates" etc. NER is essentially a token classification task where every token is classified into one or more predetermined categories.

In this exercise, we will train a simple Transformer based model to perform NER. We will be using the data from CoNLL 2003 shared task. For more information about the dataset, please visit [the dataset website](#). However, since obtaining this data requires an additional step of getting a free license, we will be using HuggingFace's datasets library which contains a processed version of this dataset.

Install the open source datasets library from HuggingFace

We also download the script used to evaluate NER models.

```
!pip3 install datasets
!wget https://raw.githubusercontent.com/sighsmlr/conlleva1/master/conlleva1.py
```

```
import os
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from datasets import load_dataset
from collections import Counter
from conlleva1 import evaluate
```

We will be using the transformer implementation from this fantastic [example](#).

Let's start by defining a `TransformerBlock` layer:

```
class TransformerBlock(layers.Layer):
    def __init__(self, embed_dim, num_heads, ff_dim, rate=0.1):
        super(TransformerBlock, self).__init__()
        self.att = keras.layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim
        )
        self.ffn = keras.Sequential(
            [
                keras.layers.Dense(ff_dim, activation="relu"),
                keras.layers.Dense(embed_dim),
            ]
        )
        self.layernorm1 = keras.layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = keras.layers.LayerNormalization(epsilon=1e-6)
        self.dropout1 = keras.layers.Dropout(rate)
        self.dropout2 = keras.layers.Dropout(rate)

    def call(self, inputs, training=False):
        attn_output = self.att(inputs, inputs)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(inputs + attn_output)
        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output, training=training)
        return self.layernorm2(out1 + ffn_output)
```

Next, let's define a **TokenAndPositionEmbedding** layer:

```
class TokenAndPositionEmbedding(layers.Layer):
    def __init__(self, maxlen, vocab_size, embed_dim):
        super(TokenAndPositionEmbedding, self).__init__()
        self.token_emb = keras.layers.Embedding(
            input_dim=vocab_size, output_dim=embed_dim
        )
        self.pos_emb = keras.layers.Embedding(input_dim=maxlen, output_dim=embed_dim)

    def call(self, inputs):
        maxlen = tf.shape(inputs)[-1]
        positions = tf.range(start=0, limit=maxlen, delta=1)
        position_embeddings = self.pos_emb(positions)
        token_embeddings = self.token_emb(inputs)
        return token_embeddings + position_embeddings
```

Build the NER model class as a **keras.Model** subclass

```
class NERModel(keras.Model):
    def __init__(
        self, num_tags, vocab_size, maxlen=128, embed_dim=32, num_heads=2, ff_dim=32
    ):
        super(NERModel, self).__init__()
        self.embedding_layer = TokenAndPositionEmbedding(maxlen, vocab_size, embed_dim)
        self.transformer_block = TransformerBlock(embed_dim, num_heads, ff_dim)
        self.dropout1 = layers.Dropout(0.1)
        self.ff = layers.Dense(ff_dim, activation="relu")
        self.dropout2 = layers.Dropout(0.1)
        self.ff_final = layers.Dense(num_tags, activation="softmax")

    def call(self, inputs, training=False):
        x = self.embedding_layer(inputs)
        x = self.transformer_block(x)
        x = self.dropout1(x, training=training)
        x = self.ff(x)
        x = self.dropout2(x, training=training)
        x = self.ff_final(x)
        return x
```

Load the CoNLL 2003 dataset from the datasets library and process it

```
conll_data = load_dataset("conll2003")
```

We will export this data to a tab-separated file format which will be easy to read as a [tf.data.Dataset](#) object.

```
def export_to_file(export_file_path, data):
    with open(export_file_path, "w") as f:
        for record in data:
            ner_tags = record["ner_tags"]
            tokens = record["tokens"]
            if len(tokens) > 0:
                f.write(
                    str(len(tokens))
                    + "\t"
                    + "\t".join(tokens)
                    + "\t"
                    + "\t".join(map(str, ner_tags))
                    + "\n"
                )

os.mkdir("data")
export_to_file("./data/conll_train.txt", conll_data["train"])
export_to_file("./data/conll_val.txt", conll_data["validation"])
```

Make the NER label lookup table

NER labels are usually provided in IOB, IOB2 or IOBES formats. Checkout this link for more information: [Wikipedia](#)

Note that we start our label numbering from 1 since 0 will be reserved for padding. We have a total of 10 labels: 9 from the NER dataset and one for padding.

```
def make_tag_lookup_table():
    iob_labels = ["B", "I"]
    ner_labels = ["PER", "ORG", "LOC", "MISC"]
    all_labels = [(label1, label2) for label2 in ner_labels for label1 in iob_labels]
    all_labels = ["-".join([a, b]) for a, b in all_labels]
    all_labels = ["[PAD]", "O"] + all_labels
    return dict(zip(range(0, len(all_labels) + 1), all_labels))

mapping = make_tag_lookup_table()
print(mapping)
```

```
{0: '[PAD]', 1: 'O', 2: 'B-PER', 3: 'I-PER', 4: 'B-ORG', 5: 'I-ORG', 6: 'B-LOC', 7: 'I-LOC',
8: 'B-MISC', 9: 'I-MISC'}
```

Get a list of all tokens in the training dataset. This will be used to create the vocabulary.

```

all_tokens = sum(conll_data["train"]["tokens"], [])
all_tokens_array = np.array(list(map(str.lower, all_tokens)))

counter = Counter(all_tokens_array)
print(len(counter))

num_tags = len(mapping)
vocab_size = 20000

# We only take (vocab_size - 2) most commons words from the training data since
# the `StringLookup` class uses 2 additional tokens - one denoting an unknown
# token and another one denoting a masking token
vocabulary = [token for token, count in counter.most_common(vocab_size - 2)]

# The StringLook class will convert tokens to token IDs
lookup_layer = keras.layers.StringLookup(
    vocabulary=vocabulary
)

```

```
21009
```

Create 2 new `Dataset` objects from the training and validation data

```

train_data = tf.data.TextLineDataset("./data/conll_train.txt")
val_data = tf.data.TextLineDataset("./data/conll_val.txt")

```

Print out one line to make sure it looks good. The first record in the line is the number of tokens. After that we will have all the tokens followed by all the ner tags.

```
print(list(train_data.take(1).as_numpy_iterator()))
```

```
[b'9\tEU\trejects\tGerman\tcall\tto\tboycott\tBritish\tlamb\t.\t3\t0\t7\t0\t0\t0\t7\t0\t0']
```

We will be using the following map function to transform the data in the dataset:

```

def map_record_to_training_data(record):
    record = tf.strings.split(record, sep="\t")
    length = tf.strings.to_number(record[0], out_type=tf.int32)
    tokens = record[1 : length + 1]
    tags = record[length + 1 :]
    tags = tf.strings.to_number(tags, out_type=tf.int64)
    tags += 1
    return tokens, tags

def lowercase_and_convert_to_ids(tokens):
    tokens = tf.strings.lower(tokens)
    return lookup_layer(tokens)

# We use `padded_batch` here because each record in the dataset has a
# different length.
batch_size = 32
train_dataset = (
    train_data.map(map_record_to_training_data)
    .map(lambda x, y: (lowercase_and_convert_to_ids(x), y))
    .padded_batch(batch_size)
)
val_dataset = (
    val_data.map(map_record_to_training_data)
    .map(lambda x, y: (lowercase_and_convert_to_ids(x), y))
    .padded_batch(batch_size)
)

ner_model = NERModel(num_tags, vocab_size, embed_dim=32, num_heads=4, ff_dim=64)

```

We will be using a custom loss function that will ignore the loss from padded tokens.

```
class CustomNonPaddingTokenLoss(keras.losses.Loss):
    def __init__(self, name="custom_ner_loss"):
        super().__init__(name=name)

    def call(self, y_true, y_pred):
        loss_fn = keras.losses.SparseCategoricalCrossentropy(
            from_logits=True, reduction=keras.losses.Reduction.NONE
        )
        loss = loss_fn(y_true, y_pred)
        mask = tf.cast((y_true > 0), dtype=tf.float32)
        loss = loss * mask
        return tf.reduce_sum(loss) / tf.reduce_sum(mask)

loss = CustomNonPaddingTokenLoss()
```

Compile and fit the model

```
ner_model.compile(optimizer="adam", loss=loss)
ner_model.fit(train_dataset, epochs=10)

def tokenize_and_convert_to_ids(text):
    tokens = text.split()
    return lowercase_and_convert_to_ids(tokens)

# Sample inference using the trained model
sample_input = tokenize_and_convert_to_ids(
    "eu rejects german call to boycott british lamb"
)
sample_input = tf.reshape(sample_input, shape=[1, -1])
print(sample_input)

output = ner_model.predict(sample_input)
prediction = np.argmax(output, axis=-1)[0]
prediction = [mapping[i] for i in prediction]

# eu -> B-ORG, german -> B-MISC, british -> B-MISC
print(prediction)
```

```
Epoch 1/10
439/439 [=====] - 13s 26ms/step - loss: 0.9300
Epoch 2/10
439/439 [=====] - 11s 24ms/step - loss: 0.2997
Epoch 3/10
439/439 [=====] - 11s 24ms/step - loss: 0.1544
Epoch 4/10
439/439 [=====] - 11s 25ms/step - loss: 0.1129
Epoch 5/10
439/439 [=====] - 11s 25ms/step - loss: 0.0875
Epoch 6/10
439/439 [=====] - 11s 25ms/step - loss: 0.0696
Epoch 7/10
439/439 [=====] - 11s 25ms/step - loss: 0.0597
Epoch 8/10
439/439 [=====] - 11s 25ms/step - loss: 0.0509
Epoch 9/10
439/439 [=====] - 11s 25ms/step - loss: 0.0461
Epoch 10/10
439/439 [=====] - 11s 25ms/step - loss: 0.0408
tf.Tensor([[ 989 10951   205   629    7 3939   216 5774]], shape=(1, 8), dtype=int64)
['B-ORG', 'O', 'B-MISC', 'O', 'O', 'O', 'B-MISC', 'O']
```

Metrics calculation

Here is a function to calculate the metrics. The function calculates F1 score for the overall NER dataset as well as individual scores for each NER tag.

```
def calculate_metrics(dataset):
    all_true_tag_ids, all_predicted_tag_ids = [], []

    for x, y in dataset:
        output = ner_model.predict(x)
        predictions = np.argmax(output, axis=-1)
        predictions = np.reshape(predictions, [-1])

        true_tag_ids = np.reshape(y, [-1])

        mask = (true_tag_ids > 0) & (predictions > 0)
        true_tag_ids = true_tag_ids[mask]
        predicted_tag_ids = predictions[mask]

        all_true_tag_ids.append(true_tag_ids)
        all_predicted_tag_ids.append(predicted_tag_ids)

    all_true_tag_ids = np.concatenate(all_true_tag_ids)
    all_predicted_tag_ids = np.concatenate(all_predicted_tag_ids)

    predicted_tags = [mapping[tag] for tag in all_predicted_tag_ids]
    real_tags = [mapping[tag] for tag in all_true_tag_ids]

    evaluate(real_tags, predicted_tags)

calculate_metrics(val_dataset)
```

```
processed 51362 tokens with 5942 phrases; found: 5504 phrases; correct: 3855.
accuracy:  63.28%; (non-0)
accuracy:  93.22%; precision:  70.04%; recall:  64.88%; FB1:  67.36
          LOC: precision:  85.67%; recall:  78.12%; FB1:  81.72  1675
          MISC: precision:  73.15%; recall:  65.29%; FB1:  69.00  823
          ORG: precision:  56.05%; recall:  63.53%; FB1:  59.56  1520
          PER: precision:  65.01%; recall:  52.44%; FB1:  58.05  1486
```

Conclusions

In this exercise, we created a simple transformer based named entity recognition model. We trained it on the CoNLL 2003 shared task data and got an overall F1 score of around 70%. State of the art NER models fine-tuned on pretrained models such as BERT or ELECTRA can easily get much higher F1 score -between 90-95% on this dataset owing to the inherent knowledge of words as part of the pretraining process and the usage of subword tokenization.

You can use the trained model hosted on [Hugging Face Hub](#) and try the demo on [Hugging Face Spaces](#).