In [1]:

```
import os
from pathlib import Path

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt

import torch
from torch.utils.data import DataLoader
from torch.nn.utils.rnn import pad_sequence
from torch import nn

from tqdm.auto import tqdm,trange

from transformers import AdamW


BATCH_SIZE = 64
```

In [2]:

```
!rm -rf PoS-Tagging
!git clone https://github.com/Janluke0/PoS-Tagging/
os.chdir('PoS-Tagging')
out_dir = Path('/kaggle/working/')
out_dir.mkdir(exist_ok=True)
```

```
Cloning into 'PoS-Tagging'...
remote: Enumerating objects: 86, done.
remote: Counting objects: 100% (86/86), done.
remote: Compressing objects: 100% (55/55), done.
remote: Total 86 (delta 30), reused 80 (delta 24), pack-reused 0
Unpacking objects: 100% (86/86), 15.25 MiB | 9.59 MiB/s, done.
```

In [3]:

```
from model.transformers.italian import ItBERTCasedPos,ItBERTUncasedPos
from model import train_model
from dataset import TWITADS
```

Common

this part should be adde to the repo, it's time to go lighting

In [4]:

```
def train_model(model, dl_train, dl_test, cuda=False, lr=0.001, epochs=10, show_plots=Fa
lse, save_dir=None):
    loss_function = nn.NLLLoss()
    optimizer = AdamW(model.parameters(), lr=lr,weight_decay=0.01)

    if cuda:
        model = model.cuda()
    if save_dir is not None:
        save_dir.mkdir(exist_ok=True)

    losses = []
    accuracies = []
    best_acc = 0
    best_loss = float('inf')
    pbar = trange(epochs)
    for epoch in pbar:
        model.train()
        for sample in tqdm(iter(dl_train), desc=f"Training {epoch}° epoch", leave=False)
:
```

```
                x,m,y = sample['input_ids'],sample['attention_mask'], sample['labels']
                if cuda:
                    x, m, y = x.cuda(), m.cuda(), y.cuda()
                optimizer.zero_grad()

                tag_scores =  model(input_ids=x,attention_mask=m)
                loss = loss_function(tag_scores.transpose(1, 2),y)

                loss.backward()
                optimizer.step()

            acc = []
            los = []
            ## evaluation
            model.eval()
            with torch.no_grad():
                for sample in tqdm(iter(dl_test), desc=f"Eval {epoch}° epoch", leave=False):
                    x,m,y = sample['input_ids'],sample['attention_mask'], sample['labels']
                    if cuda:
                        x, m, y = x.cuda(), m.cuda(), y.cuda()

                    tag_scores =  model(input_ids=x,attention_mask=m)
                    if hasattr(tag_scores,'logits'):
                        tag_scores = tag_scores.logits

                    loss = loss_function(tag_scores.transpose(1, 2),y)
                    los.append(loss.cpu().item())

                    acc.append(((tag_scores.argmax(2))==y)[m==1].float())

            acc = torch.cat(acc).mean().item()
            los = np.array(los).mean()

            losses.append(los)
            accuracies.append(acc)
            #show epoch results
            pbar.set_description(f"Loss:{los}\tAccurancy:{acc}")
            if show_plots:
                plt.subplot(121)
                plt.title("Test loss")
                plt.plot(losses)

                plt.subplot(122)
                plt.title("Test accuracy")
                plt.plot(accuracies)
            if save_dir is not None and acc >= best_acc:
                torch.save(model.state_dict(),save_dir/f"model_best_acc.pth")
            if save_dir is not None and loss <= best_loss:
                torch.save(model.state_dict(),save_dir/f"model_best_loss.pth")

            best_acc = max(acc,best_acc)
            best_loss = max(los,best_loss)

        return losses,accuracies

def show_pred(model, ds, i):
    REVTAG = {v:k for k,v in ds._TAGS.items()}
    model.cpu()
    sample = ds.collate([ds[i]])
    x,m,y = sample['input_ids'],sample['attention_mask'], sample['labels']
    with torch.no_grad():
        pred = model(input_ids=x,attention_mask=m)
    tkns = ds.tokenizer.convert_ids_to_tokens(x[0,1:-1])
    return list(zip(tkns,[REVTAG[v.item()] for v in pred[0].argmax(1)[1:-1]],[REVTAG[v.i
tem()] for v in y[0][1:-1]]))
```

In [5]:

```
def collate_fn(batch):
    input_ids, token_type_ids, attention_mask, labels = [[] for _ in range(4)]
    for sample in batch:
        input_ids.append(sample['input_ids'])
        token_type_ids.append(sample['token_type_ids'])
        attention_mask.append(sample['attention_mask'])
        labels.append(sample['labels'])
    d = {
        'input_ids':pad_sequence(input_ids,batch_first=True),
        'token_type_ids': pad_sequence(token_type_ids,batch_first=True),
        'labels':pad_sequence(labels, padding_value=-100,batch_first=True),
    }
    d['attention_mask'] = (d['labels']!=-100).float()*torch.ones(d['labels'].shape)
    return d
```

In [6]:

```
def tokenize_and_align_labels(tokenizer, tokens, tags):
    tokens = list(tokens)
    tokenized_inputs = tokenizer(tokens, truncation=True, is_split_into_words=True)

    word_ids = tokenized_inputs.word_ids(batch_index=0)  # Map tokens to their respectiv
e word.
    previous_word_idx = None
    label_ids = []
    for word_idx in word_ids:                             # Set the special tokens to -10
0
        if word_idx is None:
            label_ids.append(-100)
        elif word_idx != previous_word_idx:               # Only label the first token of
a given word.
            label_ids.append(tags[word_idx])
            previous_word_idx = word_idx
        else:
            label_ids.append(-100)

    tokenized_inputs["labels"] = label_ids

    return {k:torch.tensor(v) for k,v in tokenized_inputs.items()}
```

Cased model

In [7]:

```
cased_model = ItBERTCasedPos(23)
```

```
Some weights of the model checkpoint at dbmdz/bert-base-italian-cased were not used when
initializing BertForTokenClassification: ['cls.predictions.bias', 'cls.predictions.transf
orm.dense.bias', 'cls.seq_relationship.weight', 'cls.predictions.transform.dense.weight',
'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.LayerNorm.bias',
'cls.seq_relationship.bias', 'cls.predictions.decoder.weight']
- This IS expected if you are initializing BertForTokenClassification from the checkpoint
of a model trained on another task or with another architecture (e.g. initializing a Bert
ForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForTokenClassification from the checkp
oint of a model that you expect to be exactly identical (initializing a BertForSequenceCl
assification model from a BertForSequenceClassification model).
Some weights of BertForTokenClassification were not initialized from the model checkpoint
at dbmdz/bert-base-italian-cased and are newly initialized: ['classifier.weight', 'classi
fier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predi
ctions and inference.
```

In [8]:

```
tokenizer = ItBERTCasedPos.tokenizer()
ds_train = TWITADS('resampled_train',
                   lambda w:[w],
                   transform=lambda a,b: tokenize_and_align_labels(tokenizer,a,b)
)
ds_val = TWITADS('resampled_validation',
                   lambda w:[w],
                   transform=lambda a,b: tokenize_and_align_labels(tokenizer,a,b)
)
```

In [9]:

```
dl_train = DataLoader(ds_train, shuffle=True, batch_size=BATCH_SIZE, collate_fn=collate_f
n)
dl_val   = DataLoader(ds_val, shuffle=True, batch_size=BATCH_SIZE, collate_fn=collate_fn)
```

In [10]:

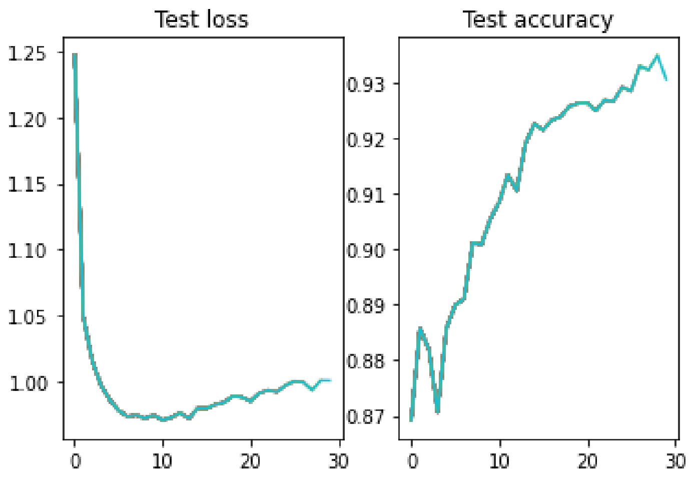
```
?train_model
```

In [11]:

```
torch.manual_seed(42)
train_model(cased_model,dl_train,dl_val,cuda=torch.cuda.is_available(), lr=2e-5, epochs=
30, show_plots=True, save_dir=out_dir/"cased_model")
```

Out[11]:

```
([1.246960747241974,
  1.0487639367580415,
  1.0154193758964538,
  0.9969477176666259,
  0.9859574675559998,
  0.9781641244888306,
  0.9733147263526917,
  0.9744363844394683,
  0.9719797492027282,
  0.9744706094264984,
  0.9706034243106842,
  0.9726258814334869,
  0.976215136051177 9,
  0.9717868208885193,
  0.9796198666095733,
  0.9794528067111969,
  0.9824764490127563,
  0.9841284096240998,
  0.9887665688991547,
  0.9880777478218079,
  0.9851040422916413,
  0.9910544395446778,
  0.9932881772518158,
  0.9919820308685303,
  0.997039407491684,
  1.000197023153305,
  0.9995040416717529,
  0.9934559106826782,
  1.0007389307022094,
  1.0006577789783477],
 [0.8692137002944946,
  0.8857396841049194,
  0.8820080161094666,
  0.8707241415977478,
  0.8857396841049194,
  0.8899156451225281,
  0.8910706639289856,
  0.9011995196342468,
  0.9008440971374512,
  0.9054642915725708,
  0.9084851741790771,
  0.9134607315063477,
  0.9106175303459167,
  0.9190582036972046,
  0.922612190246582,
  0.9215459823608398,
  0.9233229756355286,
  0.9239449501037598,
  0.9258107542991638,
  0.9263438582420349,
  0.926432728767395,
  0.925011157989502,
  0.926876962184906,
  0.9266992807388306,
  0.9292759299278259,
  0.9285650849342346,
  0.9330075979232788,
  0.9323856234550476,
  0.934962272644043,
  0.930697500705719])
```



Uncased model

In [12]:

```
uncased_model = ItBERTUncasedPos(23)
```

```
Some weights of the model checkpoint at dbmdz/bert-base-italian-uncased were not used whe
n initializing BertForTokenClassification: ['cls.predictions.bias', 'cls.predictions.tran
sform.dense.bias', 'cls.seq_relationship.weight', 'cls.predictions.transform.dense.weight
', 'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.LayerNorm.bia
s', 'cls.seq_relationship.bias', 'cls.predictions.decoder.weight']
- This IS expected if you are initializing BertForTokenClassification from the checkpoint
of a model trained on another task or with another architecture (e.g. initializing a Bert
ForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForTokenClassification from the checkp
oint of a model that you expect to be exactly identical (initializing a BertForSequenceCl
assification model from a BertForSequenceClassification model).
Some weights of BertForTokenClassification were not initialized from the model checkpoint
at dbmdz/bert-base-italian-uncased and are newly initialized: ['classifier.weight', 'clas
sifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predi
ctions and inference.
```

In [13]:

```
tokenizer = ItBERTUncasedPos.tokenizer()
ds_train = TWITADS('resampled_train',
                 lambda w:[w],
                 transform=lambda a,b: tokenize_and_align_labels(tokenizer,a,b)
)
ds_val = TWITADS('resampled_validation',
                 lambda w:[w],
                 transform=lambda a,b: tokenize_and_align_labels(tokenizer,a,b)
)
```

In [14]:

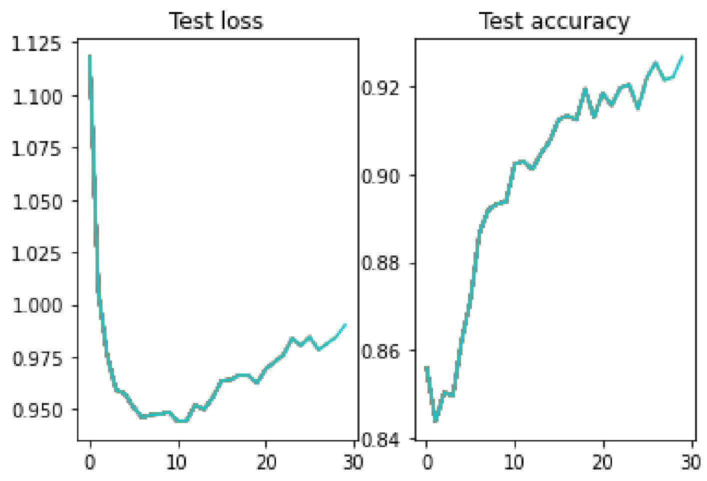
```
dl_train = DataLoader(ds_train, shuffle=True, batch_size=BATCH_SIZE, collate_fn=collate_f
n)
dl_val   = DataLoader(ds_val, shuffle=True, batch_size=BATCH_SIZE, collate_fn=collate_fn)
```

In [15]:

```
torch.manual_seed(42)
train_model(uncased_model,dl_train,dl_val,cuda=torch.cuda.is_available(), lr=2e-5, epochs
=30, show_plots=True,save_dir=out_dir/"uncased_model")
```

Out[15]:

```
([1.1181639432907104,
  1.006150245666504,
  0.9759851455688476,
  0.959184730052948,
  0.9574857890605927,
  0.9509188532829285,
  0.9460362434387207,
  0.947427386045456,
  0.9476013720035553,
  0.948854410648346,
  0.9443119168281555,
  0.9445123970508575,
  0.9520338535308838,
  0.9499574899673462,
  0.9556913673877716,
  0.963675731420517,
  0.9640557587146759,
  0.966399508714676,
  0.9662407100200653,
  0.9625177025794983,
  0.9691667556762695,
  0.9725211501121521,
  0.9758700668811798,
  0.9838366091251374,
  0.9804231405258179,
  0.9843933701515197,
  0.978424996137619,
  0.9814141809940338,
  0.9845369637012482,
  0.9901404619216919],
 [0.8558478355407715,
  0.8437610864639282,
  0.8503376841545105,
  0.8495378494262695,
  0.8624244332313538,
  0.8715783357620239,
  0.8865979313850403,
  0.8919302821159363,
  0.8932633996009827,
  0.8937077522277832,
  0.9023284316062927,
  0.9029505848884583,
  0.9011731147766113,
  0.9048168659210205,
  0.9076608419418335,
  0.912548840045929,
  0.9133487343788147,
  0.9124599695205688,
  0.9194809794425964,
  0.9129931926727295,
  0.9185033440589905,
  0.9156594276428223,
  0.9197475910186768
  0.9197475910186768,
  0.9204585552215576,
  0.9149484038352966,
  0.9219694137573242,
  0.9254354238510132,
  0.9215250611305237,
  0.9221471548080444,
  0.9267685413360596])
```



In [16]:

```
#remove repo from saved output
!rm -rf /kagle/working/PoS-Tagging
```

```
huggingface/tokenizers: The current process just got forked, after parallelism has alread
y been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
 - Avoid using `tokenizers` before the fork if possible
 - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
```