Practice 10

1. Transformers for text classification

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/
https://jalammar.github.io/illustrated-transformer/
https://jalammar.github.io/illustrated-gpt2/

In this section, we will try to train the transformer for the classification problem. One of the recent trends in deep learning is Transfer Learning. We train models to solve simple problems on a huge amount of data, and then use these pre-trained models, for solution of more specific problems.

- Transformer (Google Brain/Research)
- BERT (Google Research)
- GPT-2 (OpenAI)
- XLNet (Google Brain)
- CTRL (SalesForce)
- Megatron (NVidia)
- Turing-NLG (Microsoft)

Let's try to use it on real task.

Splitting data to training and testing sets

```
train_data, test_data = np.split(df.sample(frac=1), [int(.8*len(df))])

train_data = train_data.reset_index(drop=True)
test_data = test_data.reset_index(drop=True)

print("Size of training set: {}".format(len(train_data)))
print("Size of testing set: {}".format(len(test_data)))
```

Installing packages

```
!conda install -y pytorch torchvision cudatoolkit=10.1 -c pytorch
!pip install transformers
```

Importing libraries and define the device to make calculation on GPU

```
from transformers import BertTokenizer, BertForSequenceClassification
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
```

```
device = torch.device('cuda') if torch.cuda.is_available() else
torch.device('cpu')

tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = BertForSequenceClassification.from_pretrained("bert-base-
uncased")
model.config.num_labels = 1
```

Freeze the pre trained parameters and add three new layers at the end of the network

```
for param in model.parameters():
    param.requires_grad = False

model.classifier = nn.Sequential(
    nn.Linear(768, 256),
    nn.ReLU(),
    nn.Linear(256, 64),
    nn.ReLU(),
    nn.Linear(64, 2),
    nn.Softmax(dim=1)
)

model = model.to(device)
```

Choosing criterion and optimizer

```
criterion = nn.MSELoss().to(device)
optimizer = optim.SGD(model.classifier.parameters(), lr=0.01)
```

Making specific text preprocessing

```
def preprocess_text(text):
    parts = []

    text_len = len(text.split(' '))
    delta = 300
    max_parts = 5
    nb_cuts = int(text_len / delta)
    nb_cuts = min(nb_cuts, max_parts)


    for i in range(nb_cuts + 1):
        text_part = ' '.join(text.split(' ')[i * delta: (i + 1) *
```

```
delta])
        parts.append(tokenizer.encode(text_part, return_tensors="pt",
max_length=500).to(device))

    return parts
```

Creating a training script and train model

```
print_every = 300

total_loss = 0
all_losses = []

CUDA_LAUNCH_BLOCKING=1

model.train()

for idx, row in train_data.iterrows():
    text_parts = preprocess_text(str(row['clean_text']))
    label = torch.tensor([row['target']]).long().to(device)

    optimizer.zero_grad()

    overall_output = torch.zeros((1, 2)).float().to(device)
    for part in text_parts:
        if len(part) > 0:
            try:
                input = part.reshape(-1)[:512].reshape(1, -1)
                # print(input.shape)
                overall_output += model(input,
labels=label)[1].float().to(device)
            except Exception as e:
                print(str(e))

#     overall_output /= len(text_parts)
    overall_output = F.softmax(overall_output[0], dim=-1)

    if label == 0:
        label = torch.tensor([1.0, 0.0]).float().to(device)

    elif label == 1:
        label = torch.tensor([0.0, 1.0]).float().to(device)

    # print(overall_output, label)
```

```
    loss = criterion(overall_output, label)
    total_loss += loss.item()

    loss.backward()
    optimizer.step()

    if idx % print_every == 0 and idx > 0:
        average_loss = total_loss / print_every
        print("{}/{}. Average loss: {}".format(idx, len(train_data),
average_loss))
        all_losses.append(average_loss)
        total_loss = 0
```

Plotting loss graph

```
import matplotlib.pyplot as plt

%matplotlib inline
torch.save(model.state_dict(), "model_after_train.pt")

plt.plot(all_losses)
```

Evaluate model on test data

```
total = len(test_data)
number_right = 0
model.eval()
with torch.no_grad():
    for idx, row in test_data.iterrows():
        text_parts = preprocess_text(str(row['clean_text']))
        label = torch.tensor([row['target']]).float().to(device)

        overall_output = torch.zeros((1,2)).to(device)
        try:
            for part in text_parts:
                if len(part) > 0:
                    overall_output += model(part.reshape(1, -1))[0]
        except RuntimeError:
            print("GPU out of memory, skipping this entry.")
            continue

        overall_output = F.softmax(overall_output[0], dim=-1)
```

```
        result = overall_output.max(0)[1].float().item()
        if result == label.item():
            number_right += 1

        if idx % print_every == 0 and idx > 0:
            print("{}/{}. Current accuracy: {}".format(idx, total,
number_right / idx))

print("Accuracy on test data: {}".format(number_right / total))
```

Final task:

Use fake news dataset or choose your own. Do EDA, cleaning and preprocessing, train 1 simple model, one NN model and one transformer. Compare the results.