Assignment 6 : Pretrain and Transfer Learning (20 pts)

Before working on the assignment please read papers as following

- SUPERVISED CONTRASTIVE LEARNING FOR PRE-TRAINED LANGUAGE MODEL FINE-TUNING
  - link: https://openreview.net/pdf?id=cu7IUiO
- Few-Shot Intent Detection via Contrastive Pre-Training and Fine-Tuning
  - link: https://arxiv.org/abs/2109.06349

Question 1: Why do we need transfer learning ? (1.5pts)

- Because we can transfer the learned model to some particular tasks that we want, and we can reduce computational cost of learning.

Question 2: When transfer learning makes sense ? (1.5pts)

- We the model that we want to train is very huge, and the computational cost of it is very high. Therefore, it will be very save the cost, if we use transfer learning and train only from the particular tasks.

In [ ]:

```
import os
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from transformers import RobertaConfig, RobertaModel, RobertaTokenizer, RobertaForSequenceClassification
from transformers import AdamW
import random
from IPython.display import clear_output
from utils import create_supervised_pair, supervised_contrasive_loss, Similarity

#comment this if you are not using puffer
# os.environ['http_proxy'] = 'http://192.41.170.23:3128'
# os.environ['https_proxy'] = 'http://192.41.170.23:3128'
```

To download data from file directory both text samples and labels

In [ ]:

```
def load_examples(file_path, do_lower_case=True):
    examples = []

    with open('{}/seq.in'.format(file_path),'r',encoding="utf-8") as f_text, open('{}/label'.format(file_path),'r',encodin
        for text, label in zip(f_text, f_label):

            e = Inputexample(text.strip(),label=label.strip())
            examples.append(e)

    return examples
```

Each sample has a sentence and label format

In [ ]:

```
class Inputexample(object):
    def __init__(self,text_a,label = None):
        self.text = text_a
        self.label = label
```

Question3 : Write the code to be able to control batching data process for the sake of fine-tuning models with combining cross entropy and supervised contrastive loss in question 5, and only cross entropy in question 4. (7pts)

- assume : we have batch size = 4 but we have 64 classes, so sometime batching process will random sample in a batch, and then it has no any samples come from the same classes like below
  - samples_sentence = ['a','b','c','d'] : assume that one alphabet represent one sentence or one sample
  - labels = [0,1,2,3] ; Therefore, if a batch has unqiue classes equal to batch size, this batch will be skipped due to equation "1yi=yj" of supervised contrastive loss(equation in question 5) that's reason why we need to force like below buttlet.
  - you can see at least one pair that come from the same class.

    Therefore, we want dataloader to output like below

    - samples_sentence = ['a','b','c','f']
    - labels = [0,1,2,0] ; this batch will pass condition as "1yi=yj" because the label of y[0] = 0, y[3] = 0 in list of labels.

In [ ]:

```python
# create custom dataset class
# ===  =  Hint =  ===
# can train on two condition
# 1.) trainig training with supervise contrastive loss and cross entropy loss using in question 5.)
#    when self.repeated_label == True:
# 2.) train only cross entropy loss use in question 4.)
#    when self.repeated_label == False:
class CustomTextDataset(Dataset):
    def __init__(self,labels,text,batch_size,repeated_label:bool=False):
        self.labels = labels
        self.text = text
        self.batch_size = batch_size
        self.count = 0
        self.repeated_label = repeated_label

        # to use when training with supervise contrastive loss
        if self.repeated_label == True:
          # write the code here
          self.used_idx = []


    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):

        # write code here for 1)
        if self.repeated_label == True:
            if idx not in self.used_idx:
                self.used_idx.append(idx)

        label = self.labels[idx]

        data = self.text[idx]

        sample = {"Class": label,"Text": data}
        return sample
```

What is Few-shot Learning ?

- few-shot learning is the process of train model on small amount of data in each class to guide model on specific taks, opposed to standard fine-tuning method which requires a large amount of training data for the pretrained model to adapt to the desired task with accuracy.

source : https://huggingface.co/blog/few-shot-learning-gpt-neo-and-inference-api

Define Parameters

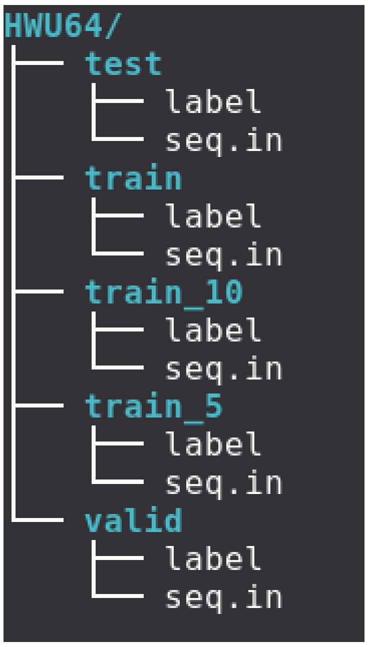
In [ ]:
```
N = 5
data = []
labels = []
train_samples = []
train_labels = []
embed_dim = 768
batch_size = 4
lr= 1e-5  # you can adjust
temp = 0.3  # you can adjust
lamda = 0.01  # you can adjust
skip_time = 0 # the number of time that yi not equal to yj in supervised contrastive loss equation
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

The Aim of these training is to fine tuning on few shot setting on text classification task

Path example of train, validation and test

In [ ]:
```
path_5shot = f'./HWU64/train_5/'
path_test = f'./HWU64/test/'
path_valid = f'./HWU64/valid/'
```

Dataset Structure



```
HWU64/
├── test
│   ├── label
│   └── seq.in
├── train
│   ├── label
│   └── seq.in
├── train_10
│   ├── label
│   └── seq.in
├── train_5
│   ├── label
│   └── seq.in
└── valid
    ├── label
    └── seq.in
```

In [ ]:
```
# !unzip HWU64.zip
```

```
In [ ]: # https://downgit.github.io/#/home?url=https:%2F%2Fgithub.com%2Fjianguoz%2FFew-Shot-Intent-Detection%2Ft

        # downloading training samples
        train_samples = load_examples(path_5shot)

        # write code here : for downloading validation samples
        valid_samples = load_examples(path_valid)

        #write code here : for downloading test samples
        test_samples = load_examples(path_test)

        # preprocess for training
        for i in range(len(train_samples)):
            data.append(train_samples[i].text)
            labels.append(train_samples[i].label)

        # write code here :preprocess validation samples
        valid_data = []
        valid_labels = []
        for i in range(len(valid_samples)):
            valid_data.append(valid_samples[i].text)
            valid_labels.append(valid_samples[i].label)

        # write code here : preprocess test samples
        test_data1 = []
        test_labels = []
        for i in range(len(valid_samples)):
            test_data1.append(test_samples[i].text)
            test_labels.append(test_samples[i].label)

        # dataloader for training
        train_data = CustomTextDataset(labels,data,batch_size=batch_size,repeated_label=True)
        train_loader = DataLoader(train_data,batch_size=batch_size,shuffle=True)

        # write code here : dataloader for validation
        validation_data = CustomTextDataset(valid_labels,valid_data,batch_size=batch_size,repeated_label=True)
        valid_loader = DataLoader(validation_data,batch_size=batch_size,shuffle=True)

        # write code here : dataloader for test
        test_data = CustomTextDataset(test_labels,test_data1,batch_size=batch_size,repeated_label=True)
        test_loader = DataLoader(test_data,batch_size=batch_size,shuffle=True)

        # got the number of unique classes from dataset
        num_class = len(np.unique(np.array(labels)))

        # get text label of uniqure classes
        unique_label = np.unique(np.array(labels))

        # map text label to index classes
        label_maps = {unique_label[i]: i for i in range(len(unique_label))}

        # Download tokenizer that use to tokenize sentence into words by using Pretrain from roberta-base
        tokenizer = RobertaTokenizer.from_pretrained('roberta-base')
```

Download Pretrain Model

```
In [ ]: # download config of Roberta config
        config = RobertaConfig.from_pretrained("roberta-base",output_hidden_states=True)

        #chnage modifying the number of classes
        config.num_labels = num_class
        # Download pretrain models weight
        model = RobertaForSequenceClassification.from_pretrained('roberta-base')
        # change from binary classification to muli-classification and loss automatically change to cross entropy loss
        model.num_labels = config.num_labels
        # change the output of last layer to num_class that we want to predict
```

```
model.classifier.out_proj = nn.Linear(in_features=embed_dim,out_features=num_class)
# move to model to device that we set
model = model.to(device)
```

```
Some weights of the model checkpoint at roberta-base were not used when initializing RobertaForSequenceClassification: ['lm_head.decoder.weight', 'roberta.pooler.dense.bias', 'roberta.pooler.dense.weight', 'lm_head.bias', 'lm_head.layer_norm.weight', 'lm_head.layer_norm.bias', 'lm_head.dense.bias', 'lm_head.dense.weight']
- This IS expected if you are initializing RobertaForSequenceClassification from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing RobertaForSequenceClassification from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
Some weights of RobertaForSequenceClassification were not initialized from the model checkpoint at roberta-base and are newly initialized: ['classifier.dense.weight', 'classifier.dense.bias', 'classifier.out_proj.weight', 'classifier.out_proj.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

In [ ]:

```
# Using adam optimizer
optimizer= AdamW(model.parameters(), lr=lr)
```

Question3: write function to freeze model (3pts)

In [ ]:

```
def freeze_layers(model,freeze_layers_count:int):

    """
    model : model object that we create
    freeze_layers_count : the number of layers to freeze
    """
    for param in model.parameters():
        param.requires_grad = True
    # write the code here
    if freeze_layers_count == -1:
        for param in model.parameters():
            param.requires_grad = False
    else:
        i = 0
        for name, layer in model.named_modules():
            i += 1
            for name, param in layer.named_parameters():
                param.requires_grad = False
            if i == freeze_layers_count:
                break

    return model
```

Question4: Training on text classification task on CrossEntropy loss (3.5 pts)

- Using API of hugging face of RobertaForSequenceClassification
  - source : https://huggingface.co/transformers/v3.0.2/model_doc/roberta.html#robertaforsequenceclassification
- report performance of models (test acc) with differrent experiement of unfreezing of bottom layers and compare the result of each
  - 4.1. freeze weight from pretrain model all layer except classifier

    ```
    RobertaClassificationHead(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
      (out_proj): Linear(in_features=768, out_features=64, bias=True)
    )
    ```
  -

```
- 4.2. freeze all from top embeddings to encoder layers (9)
  - embeddings
    - ![image.png](attachment:2a69ebe1-6893-45ca-b7ea-38f9715b8c9a.png)
  - layer 9
    - ![image.png](attachment:ef8674dc-7743-40d4-bde4-21a3819e62fb.png)
- 4.3 add code to collect loss and accuracy of training history  of (4.1 and
4.2)
- 4.4 add the code in below in training loop collect validation loss and
accuracy history of (4.1 and 4.2)
```

- hint: for this training on Cross entropy loss no need to control the outcome of class in each batch using code below to train model base on how many layers that you freeze
  - to see whole architecture look like you can use mode.eval()

In [ ]:

```python
# this code training models on Cross entropy loss

train_losses = []
train_acc = []
valid_losses = []
valid_acc = []

for epoch in range(30):  # loop over the dataset multiple times

    running_loss = 0.0
    corr = 0
    total = 0

    for (idx, batch) in enumerate(train_loader):
        sentence = batch["Text"]
        inputs = tokenizer(sentence,padding=True,truncation=True,return_tensors="pt")

        # move parameter to device
        inputs = {k:v.to(device) for k,v in inputs.items()}

        # map string labels to class idex
        labels = [label_maps[stringtoId] for stringtoId in (batch['Class'])]

        #print("show out: ",np.unique(labels, return_counts=True))
        # convert list to tensor
        labels = torch.tensor(labels).unsqueeze(0)
        labels = labels.to(device)


        #(batch_size, seq_len)
        #print(inputs["input_ids"].shape)

         # zero the parameter gradients
        optimizer.zero_grad()

        outputs = model(**inputs,labels=labels)
        # you can check
        loss, logits = outputs[:2]
        _, pred = torch.max(logits.data, 1)

        loss.backward()
        optimizer.step()

        corr += (pred == labels.reshape(-1)).sum().item()
        total += labels.reshape(-1).size(0)
        running_loss += loss.item()

        # write code here
        # to save model eg. model.pth look at pytorch document how to save model
```

```
            print(f'Train: [{epoch + 1}, {idx}] loss: {loss.item()}')
            print(running_loss)
            clear_output(wait=True)

        train_losses.append(running_loss/total)
        train_acc.append(corr*100/total)

        running_loss = 0.0
        corr = 0
        total = 0

        for (idx, batch) in enumerate(valid_loader):
            sentence = batch["Text"]
            inputs = tokenizer(sentence,padding=True,truncation=True,return_tensors="pt")

            # move parameter to device
            inputs = {k:v.to(device) for k,v in inputs.items()}

            # map string labels to class idex
            labels = [label_maps[stringtoId] for stringtoId in (batch['Class'])]

            #print("show out: ",np.unique(labels, return_counts=True))
            # convert list to tensor
            labels = torch.tensor(labels).unsqueeze(0)
            labels = labels.to(device)


            #(batch_size, seq_len)
            #print(inputs["input_ids"].shape)

             # zero the parameter gradients
            optimizer.zero_grad()

            outputs = model(**inputs,labels=labels)
            # you can check
            loss, logits = outputs[:2]
            _, pred = torch.max(logits.data, 1)

            loss.backward()
            optimizer.step()

            corr += (pred == labels.reshape(-1)).sum().item()
            total += labels.reshape(-1).size(0)
            running_loss += loss.item()

            # write code here
            # to save model eg. model.pth look at pytorch document how to save model

            print(f'Train: [{epoch + 1}, {idx}] loss: {loss.item()}')
            print(running_loss)
            clear_output(wait=True)

        valid_losses.append(running_loss/total)
        valid_acc.append(corr*100/total)
```

```
Train: [9, 63] loss: 0.04172031208872795
2.784421928226948
```

- 4.5 write code to plot both loss and accuracy for training and validation repectively.
- 4.6 write test function to get test accuracy for (4.1,4.2)

In [ ]:

```
# write code here for 4.5 and 4.6
import matplotlib.pyplot as plt
for i in range(len(train_losses)):
    train_losses[i] = train_losses[i].detach().cpu().numpy()
    valid_losses[i] = valid_losses[i].detach().cpu().numpy()
f, axs = plt.subplots(1,2,figsize=(15,4))
```

```
axs[0].plot(train_losses,label = 'train')
axs[0].plot(valid_losses,label = 'valid')
axs[0].suptitle("Loss")
axs[0].legend()
axs[1].plot(train_acc,label = 'train')
axs[1].plot(valid_acc,label = 'valid')
axs[1].suptitle("Accuracy")
axs[1].legend()
```

Question 5: Training on text classification task on combine two losses Cross Entropy and Supervised Contrastive. (3.5 pts)

- Cross Entropy loss

$$\mathcal{L}_{\text{CE}} = -\frac{1}{m}\sum_{i=1}^{m} yi \cdot log(\hat{yi})$$

- Supervised Contrastive learning loss

$$\mathcal{L}_{\text{S\\_cl}} = -\frac{1}{T}\sum_{i=1}^{N}\sum_{j=1}^{N} \mathbf{1}_{\boldsymbol{yi=yj}}\ \ log\frac{e^{sim(hi,hj)/\tau}}{\sum_{n=1}^{N} e^{sim(hi,hn)/\tau}}$$

  - detail
    - ui ~ sentence i
    - hi ~ BERT(ui) in our case using Roberta as a encoder
    - hi : (batch_size,sequence_len,embed_size)
    - hi is the output of model which is last hidden layers before classifier head in the model architecture
    - 1yi=yj ~ we select only the sample that come from the same class to compute in each i and j
    - T ~ the number of pairs that come from the same classes
    - τ ~ temperature parameter
    - Sim(x1,x2) : cosine similarity [-1, 1]
    - λ' is just weighted of cross entropy loss
    - Sim function is the cosine similarity
    - N ~ the number of samples in a batch

$$sim(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

- Loss total

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{s\\_cl}} + \lambda' \mathcal{L}_{CE}$$

- you can get cross entropy loss like below
  - outputs = model(input_ids, labels=labels)
  - loss, logits = outputs[:2]
  - loss : this is cross entropy loss
- hint : for this question you will utilize the function CustomTextDataset to force dataloader to have at least one pair that come from the same class
  - eg. batch_size = 4
  - the labels in a batch should be like [0, 21, 43, 0]

1. training this model in the code below on loss_total by do experiment the same as question 4.1, 4.2, 4.3, 4.4, 4.5, 4.6

In [ ]:

```
for epoch in range(30):  # loop over the dataset multiple times

    running_loss = 0.0

    for (idx, batch) in enumerate(train_loader):
        sentence = batch["Text"]
        inputs = tokenizer(sentence,padding=True,truncation=True,return_tensors="pt")
        inputs = {k:v.to(device) for k,v in inputs.items()}

        # map string labels to class idex
        labels = [label_maps[stringtoId] for stringtoId in (batch['Class'])]


        # convert list to tensor
        labels = torch.tensor(labels).unsqueeze(0)
        labels = labels.to(device)


        #(batch_size, seq_len)
        #print(inputs["input_ids"].shape)

         # zero the parameter gradients
        optimizer.zero_grad()

        outputs = model(**inputs,labels=labels,output_hidden_states=True)

        hidden_states = outputs.hidden_states

        last_hidden_states = hidden_states[12]

        # https://stackoverflow.com/questions/63040954/how-to-extract-and-use-bert-encodings-of-sentences-for-text-
        # (batch_size,seq_len,embed_dim)
        h = last_hidden_states[:,0,:]

        # create pair samples
        T, h_i, h_j, idx_yij = create_supervised_pair(h,batch['Class'],debug=False)

        if h_i is None:
            print("skip this batch")
            skip_time +=1
            continue

        # supervised contrastive loss
        loss_s_cl = supervised_contrasive_loss(h_i, h_j, h, T,temp=temp,idx_yij=idx_yij,debug=False)

        # cross entropy loss
        loss_classify, logits = outputs[:2]

        # loss total
        loss = loss_s_cl + (lamda * loss_classify)

        loss.backward()
        optimizer.step()


        print(f'[{epoch + 1}, {idx}] loss_total: {loss.item()}, loss_s_cl:{loss_s_cl.item()}, loss_classify:{lamda * loss_cla

        clear_output(wait=True)
```