kogpt2_train.py

```python
import pandas as pd
from tqdm import tqdm
import torch
from torch.utils.data import DataLoader
from transformers import PreTrainedTokenizerFast, GPT2LMHeadModel
from chatbotDataset import ChatbotDataset


class Transformers_kogpt():
    def __init__(self) -> None:
        self.Chatbot_Data = pd.read_csv('C:/MyProject/chatbot/data/ChatBotData.csv')
        self.USE_CUDA = torch.cuda.is_available()
        self.device = torch.device('cuda:0')
        self.Q_TKN = "<usr>"
        self.A_TKN = "<sys>"
        self.BOS = '</s>'
        self.EOS = '</s>'
        self.MASK = '<unused0>'
        self.SENT = '<unused1>'
        self.PAD = '<pad>'
        self.UNK = '<unk>'

    def hook(self):
        self.model_train()
        self.save_model()

    # 토크나이저 로드
    def load_tokenizer(self):
        tokenizer = PreTrainedTokenizerFast.from_pretrained("skt/kogpt2-base-v2",
                                                            bos_token=self.BOS, eos_token=self.EOS, unk_token=self.UNK,
                                                            pad_token=self.PAD, mask_token=self.MASK)
        return tokenizer

    # 프리트레인 모델 불러오기
    def load_model(self):
        model = GPT2LMHeadModel.from_pretrained('skt/kogpt2-base-v2').to(self.device)
        return model

    @staticmethod
    def collate_batch(batch):
        data = [item[0] for item in batch]
        mask = [item[1] for item in batch]
        label = [item[2] for item in batch]
        return torch.LongTensor(data), torch.LongTensor(mask), torch.LongTensor(label)

    def train_dataloader(self):
        train_set = ChatbotDataset(self.Chatbot_Data, max_len=40)
        #윈도우 환경에서 num_workers 는 무조건 0으로 지정, 리눅스에서는 2
        train_dataloader = DataLoader(train_set, batch_size=32, num_workers=0, shuffle=True, collate_fn=self.collate_batch)
        print("test data")
        for batch_idx, samples in enumerate(tqdm(train_dataloader)):
            token_ids, mask, label = samples
        self.load_tokenizer().tokenize("안녕하세요. 한국어 GPT-2 입니다. :)l^o")
        return train_dataloader

    def model_train(self):
        model = self.load_model()
        model = model.train()
        learning_rate = 5e-5
        criterion = torch.nn.CrossEntropyLoss(reduction="none")
        optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
        epoch = 10
        Sneg = -1e18
        print ("start")
        for epoch in range(epoch):
            log_interval = 200
            for batch_idx, samples in enumerate(tqdm(self.train_dataloader())):
                optimizer.zero_grad()
                token_ids, mask, label = samples
                token_ids = token_ids.long().to(self.device)
                mask = mask.long().to(self.device)
                label = label.long().to(self.device)
                out = model(token_ids)
                out = out.logits      #Returns a new tensor with the logit of the elements of input
                mask_3d = mask.unsqueeze(dim=2).repeat_interleave(repeats=out.shape[2], dim=2)
                mask_out = torch.where(mask_3d == 1, out, Sneg * torch.ones_like(out))
                loss = criterion(mask_out.transpose(2, 1), label)
                #loss.mean().backward()
                # 평균 loss 만들기 avg_loss[0] / avg_loss[1] <- loss 정규화
                avg_loss = loss.sum() / mask.sum()
                avg_loss.backward()
```

```
            # 학습 끝
            optimizer.step()
            if batch_idx % log_interval == 0:
                print("epoch {} batch_idx {} loss {}".format(epoch+1, batch_idx+1, avg_loss))
    print ("end")
    #torch.save(model, './save/mibot_v50.pt')
    return model


#모델 저장
def save_model(self):
    model = self.model_train()
    PATH = 'C:/MyProject/chatbot/save/chatbot_v10.pt'
    torch.save(model, PATH)
```