CS224n Review - 2023

Lecture 9~16

2024.08.09

유하영

Content

BERT
Fine tuning
QA task

Encoder - BERT

1. 인코더

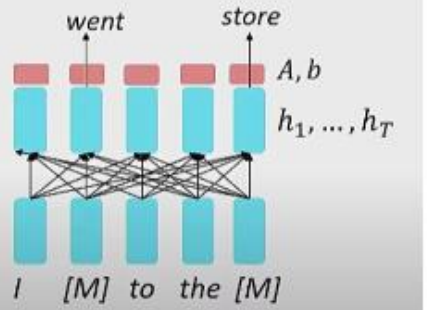
- 인코더 만으로 LM을 만들 수 없다. → 왜냐하면, 양방향의 맥락을 얻기 때문.

Pretraining encoders: what pretraining objective to use?

So far, we've looked at language model pretraining. But **encoders get bidirectional context,** so we can't do language modeling!

Idea: replace some fraction of words in the input with a special [MASK] token; predict these words.

$$h_1, \dots, h_T = \text{Encoder}(w_1, \dots, w_T)$$
$$y_i \sim Ah_i + b$$



Only add loss terms from words that are "masked out." If $\tilde{x}$ is the masked version of x, we're learning $p_\theta(x|\tilde{x})$. Called **Masked LM.**

- 위의 예시 → 중간중간에 [M]을 넣어 일부 단어를 가려놓음

  → 양방향으로 학습하며 전체적으로 예측하고, context 표현을 만들어냄

  → 빈 공간의 벡터 표현([M])은 전체 컨텍스트를 볼 수 있음

- 양방향 학습

  일반적인 언어 모델은 텍스트를 순차적으로 처리하여 다음 단어를 예측한다.

  하지만, 인코더는 양방향 맥락을 이해할 수 있어야 하므로, **순차적 모델링 방식을 사용할 수 없다.**

- 마스킹

  일부 단어를 mask 토큰으로 대체하고, 이 마스크된 위치의 원래 단어를 예측하도록 모델을 훈련시킴. → 모델은 좌우 맥락을 모두 고려하여 모델 성능을 높임.

- 최종 출력

  마스크된 단어들이 주어진 맥락에서 얼마나 적절한지 학습

→ **BERT**

15%의 토큰을 Masked한다!

그 중에서도 BERT는 새로운 idea를 제안

- 80% : Mask하고, 실제 단어 예측
- 10% : random token으로 대체하고, 실제 단어 예측
- 10% : 단어를 전혀 바꾸지 않고, 실 제 단어 예측

> 왜 이렇게..?
>
> - **mask 언어 모델링의 한계**
>
>   일부 단어 mask→ 모델은 마스크된 단어의 맥락을 이해하는 데 집중하게 됨 → no-masked 단어들에 대한 표현을 충분히 학습하지 못할 수 있음!..!
>
> - fine-tuning시 문제
>
>   사전 훈련 후, 실제 파인튜닝을 할때는 mask 토큰이 사용되지 않음.
>
>   따라서, pretrian 시 마스크에 의존하여 학습된 모델이, 실제 텍스트를 처리할 때 예상보다 성능이 떨어질 수 있음..

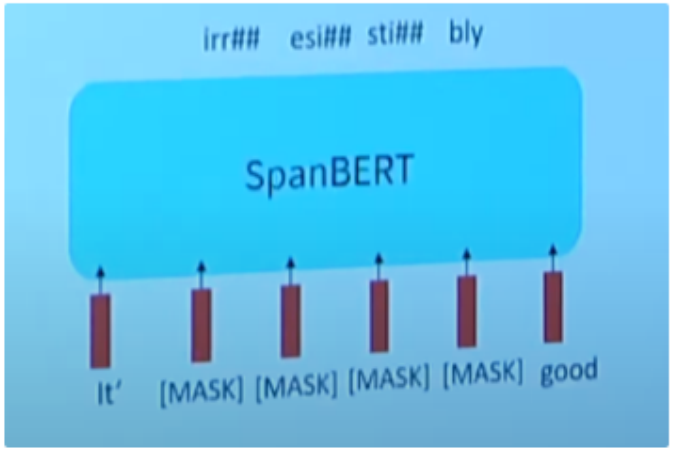
Extensions of BERT

> Extensions of BERT

>> spanBERT

연속된 단어의 시퀀스를 마스크

- 무작위 masked 대신 → 인접한 영역의 masked 사용 (어렵지만, 무작위보단 낫다)



>> RoBERTa is much BETTER!

(특징)

- 다이나믹 마스킹

  각 epoch마다 새로운 마스크 패턴을 생성

  / BERT는 사전 훈련을 시작하기 전에 전체 데이터셋에 대해 마스크를 미리 적용,

  같은 마스크 패턴을 여러 번 재사용

  → 과적합 가능성...

- NSP 태스크 사용 X

  연구에 따르면, NSP task는 성능향상에 크게 기여 X..!

BERT 사용할거면, 그냥 RoBERTa를 써라

Fine tuning

Full Finetuning vs. Parameter-Efficient Finetuning

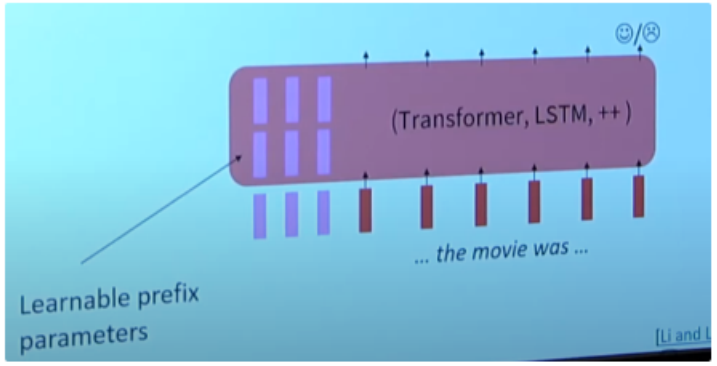
> Full Finetuning → Adapt all parameters
>
> Lightweight Finetuning → 최소한의 변화를 통해 tuning한다.

1) Prompt Tuning

- 매개변수 조정 X (gradient계산X,최적화 저장X→cheaper!)
- 일련의 시퀀스 시작에 **가짜 단어(pseudoword 벡터)를 만들어서 훈련**
  - 입력 시퀀스의 시작부분에 삽입. → 첫번째 정보로 모델이 해석하게 함

    (맥락 제공, 이러한 작업에 대한 응답을 생성하도록 유도)
  - 다른 부분은 고정하고, 이 prompt vector 자체만 학습

(가짜단어의 위치? 앞이든 끝이든 상관 없음, but 디코더는 앞에 넣어야함(전체 시퀀스를 처리하기 전에 보지 못하기 때문))



2) prefix tuning

각 디코더 레이어에 소량의 조정 가능한 파라미터를 도입 → 입력 시퀀스에 'prefix'라고 하는 가짜 토큰 시퀀스를 추가

💡 두 tuning 방법의 차이?

- **프롬프트 튜닝**

  : 가짜 단어(프롬프트)를 사용하여 모델 입력에 직접적인 작업 지시를 제공
- **프리픽스 튜닝**

  : 가짜 토큰 시퀀스(프리픽스)를 추가하여 모델의 디코더 레이어에 더 미세한 조정 하도록 함

QA task

모델 1) LSTM-based with attention

모델 2) BERT-like model for reading comprehension

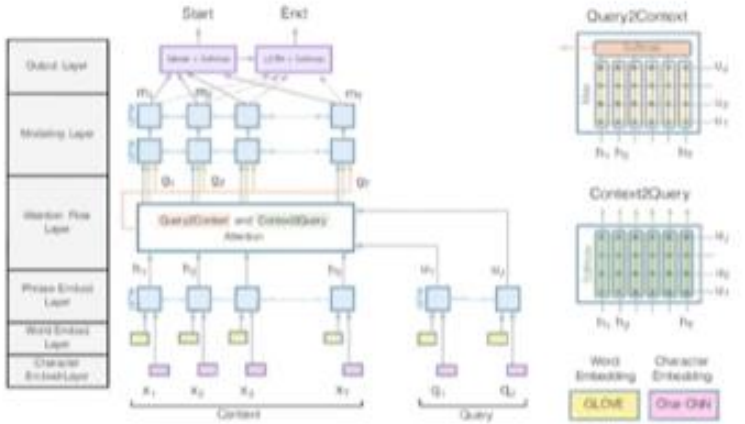


Image credit: (Seo et al, 2017)

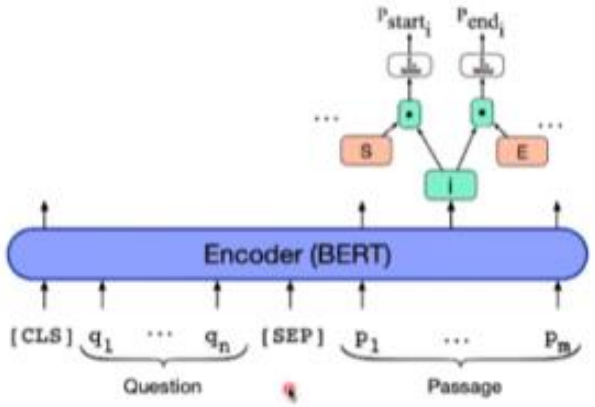


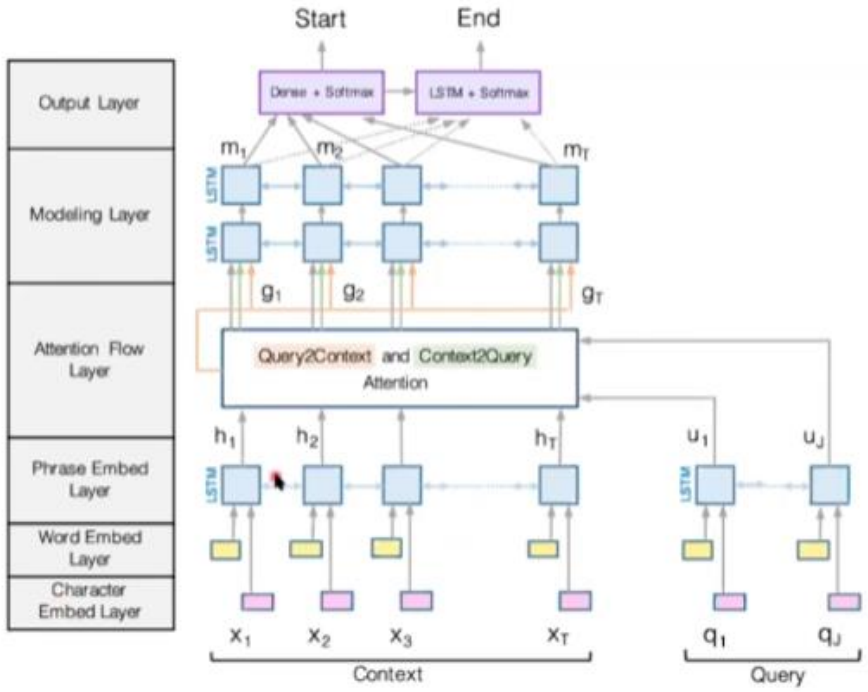
Image credit: J & M, edition 3

QA task

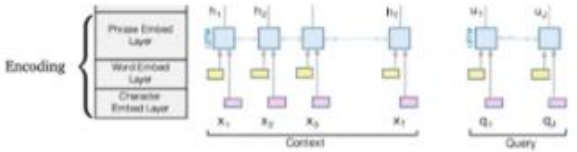
1) LSTM-based with attention

BiDAF

: the Bidirectional Attention Flow Model



1) Encoding Function



- Use a concatenation of word embedding (GloVe) and character embedding (CNNs over character embeddings) for each word in context and query.

$$e(c_i) = f([\text{GloVe}(c_i); \text{charEmb}(c_i)]) \qquad e(q_i) = f([\text{GloVe}(q_i); \text{charEmb}(q_i)])$$

f: high-way networks omitted here

- Then, use two **bidirectional** LSTMs separately to produce contextual embeddings for both context and query.

$$\overrightarrow{\mathbf{c}}_i = \text{LSTM}(\overrightarrow{\mathbf{c}}_{i-1}, e(c_i)) \in \mathbb{R}^H \qquad \overrightarrow{\mathbf{q}}_i = \text{LSTM}(\overrightarrow{\mathbf{q}}_{i-1}, e(q_i)) \in \mathbb{R}^H$$

$$\overleftarrow{\mathbf{c}}_i = \text{LSTM}(\overleftarrow{\mathbf{c}}_{i+1}, e(c_i)) \in \mathbb{R}^H \qquad \overleftarrow{\mathbf{q}}_i = \text{LSTM}(\overleftarrow{\mathbf{q}}_{i+1}, e(q_i)) \in \mathbb{R}^H$$

$$\mathbf{c}_i = [\overrightarrow{\mathbf{c}}_i; \overleftarrow{\mathbf{c}}_i] \in \mathbb{R}^{2H} \qquad \mathbf{q}_i = [\overrightarrow{\mathbf{q}}_i; \overleftarrow{\mathbf{q}}_i] \in \mathbb{R}^{2H}$$

Stanfo

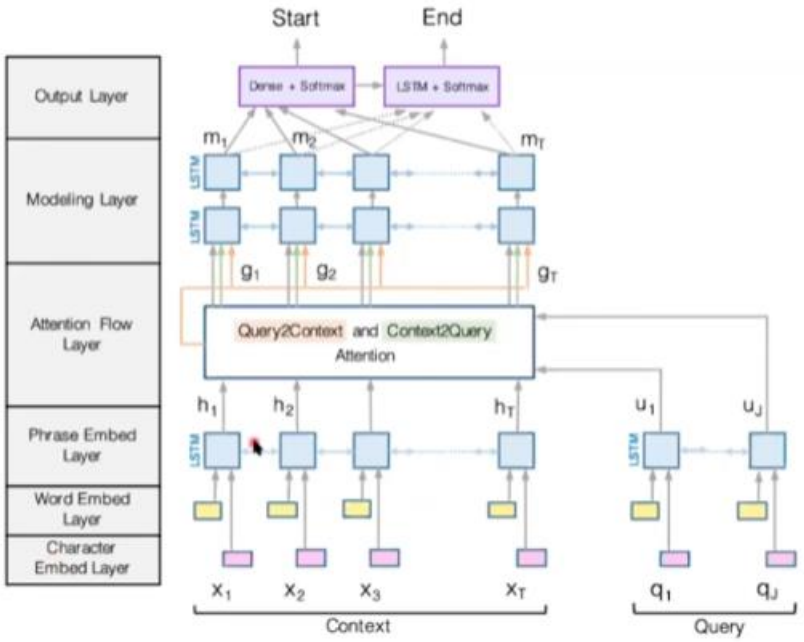
- **단어 임베딩과 문자 임베딩의 결합:**
  GloVe를 사용하여 사전 훈련된 단어 임베딩과, 문자 단위 임베딩(character embeddings)을 CNN을 통해 처리한 결과를 concat하여 표현
- **문맥과 질문의 양방향 LSTM 처리:**
  생성된 임베딩은 문맥과 질문 각각에 대해 독립적인 양방향 LSTM(BiLSTM) 네트워크를 통과
- seq2seq에서는 디코더가 자동 회귀 모델(Autoregressive Model)이기 때문에, 실제로 두 시퀀스에 대한 양방향 lstm을 수행할 수 없다.
  인코더에서만 Bi LSTM 사용!

QA task

1) LSTM-based with attention

BiDAF

: the Bidirectional Attention Flow Model



🚩 쿼리- 컨텍스트 attention, 컨텍스트-쿼리 attention이 대칭이 아닌이유?

각 어텐션 유형이 서로 다른 목적을 가지고 설계되었기 때문

- 컨텍스트-쿼리 - **문맥 내의 각 단어**가 질문 내의 어떤 단어와 가장 관련이 깊은지를 식별
- 쿼리- 컨텍스트 - **질문에 대한 가장 중요한 대답**을 문맥에서 찾아내는 데 초점

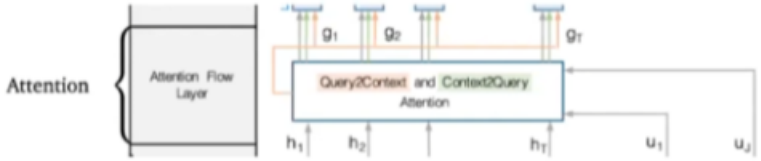
→ 서로 다른 역할을 수행하기에, 다른 부분에 집중하므로, 비대칭적인 어텐션이 이뤄짐

2) Attention

idea) Query 단어 중 가장 관련성이 높은 몇가지 문맥단어를 선택하는 것!

양방향 Attention : Query→Context / Context→ Query

어떤 질문 단어가 문맥에서 가장 연관이 있는지? / 어떤 문맥단어가 질문과 연관이 있는지?



- First, compute a similarity score for every pair of $(\mathbf{c}_i, \mathbf{q}_j)$:

$$S_{i,j} = \mathbf{w}_{\text{sim}}^{\intercal}[\mathbf{c}_i; \mathbf{q}_j; \mathbf{c}_i \odot \mathbf{q}_j] \in \mathbb{R} \qquad \mathbf{w}_{\text{sim}} \in \mathbb{R}^{6H}$$

- Context-to-query attention (which question words are more relevant to c_i):

$$\alpha_{i,j} = \text{softmax}_j(S_{i,j}) \in \mathbb{R} \qquad \mathbf{a}_i = \sum_{j=1}^{M} \alpha_{i,j}\mathbf{q}_j \in \mathbb{R}^{2H}$$

- Query-to-attention attention (which context words are relevant to some question words):

$$\beta_i = \text{softmax}_i(\max_{j=1}^{M}(S_{i,j})) \in \mathbb{R}^N \qquad \mathbf{b}_i = \sum_{i=1}^{N} \beta_i \mathbf{c}_i \in \mathbb{R}^{2H}$$

컨텍스트 단어의 중요성을 측정하기 위해 softmax 안에서 max를 사용

The final output is

$$\mathbf{g}_i = [\mathbf{c}_i; \mathbf{a}_i; \mathbf{c}_i \odot \mathbf{a}_i; \mathbf{c}_i \odot \mathbf{b}_i] \in \mathbb{R}^{8H}$$

c_i : 컨텍스트 벡터

a_i는 모든 단어가 열거

b는 모든 문맥단어를 집계.

QA task

2) BERT for reading comprehension

- **Start position layer**: 질문의 답변이 시작하는 위치를 예측하는 레이어.
- **End position layer**: 질문의 답변이 끝나는 위치를 예측하는 레이어.
- 이 레이어들은 각각의 토큰에 대해 확률을 출력하며, 가장 높은 확률을 가진 토큰이 답변의 시작점과 끝점으로 선택

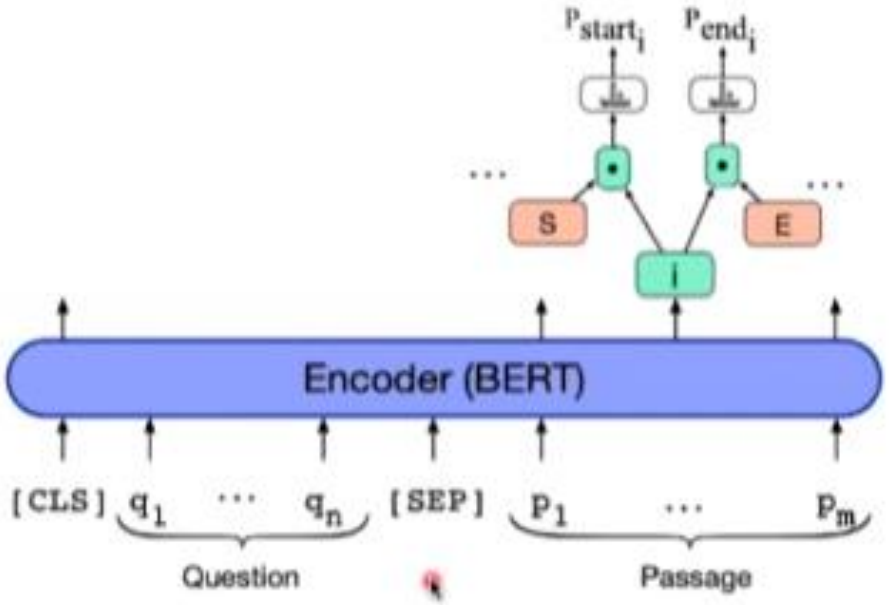


Image credit: J & M, edition 3

1-2주차 내용
발전시키기

목표

3-4주차: 기본 구현 (2) + 분야별 기초 이론

- 코딩 목표: 1주차때의 내용을 발전시켜보기
  이론 목표: Vision 인턴이라면 cs231n 뒷 내용 숙지, NLP인턴이라면 cs224n 숙지
- 내용:
  - 각 코드들 모듈화 진행 (예; model, dataset, train, valid(test), main, util)
  - argparser 추가
  - 가중치 값 저장 및 로드
  - loss, accuracy 등에 대한 log 출력 및 시각화
  - data augmentation 진행 (선택)
  - Gradient clipping, scheduler 등 다양한 기법들 활용하여 성능 향상 (목표 정확도: 85%)
  - 분야별 기초 이론 공부를 위해 vision 인턴은 cs231n 뒷부분 까지 전부 수강,
    NLP인턴은 cs224n 수강

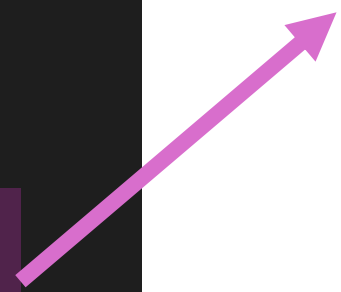
코드 모듈화 – dataset.py

```python
# 데이터 변환 정의
transform_cifar10 = transforms.Compose([
    transforms.Resize((224, 224)),  # ResNet 모델에 맞추기 위해 이미지 크기를 224x224로 조정
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
])

# CIFAR-10 데이터셋 로드 및 분할
cifar10_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_cifar10)
cifar10_test_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform_ci

# 데이터셋을 학습 세트와 검증 세트로 분할 (80% 학습, 20% 검증)
train_size = int(0.8 * len(cifar10_dataset))
val_size = len(cifar10_dataset) - train_size
cifar10_train_dataset, cifar10_val_dataset = random_split(cifar10_dataset, [train_size, val_size])

# 데이터로더 정의
cifar10_train_loader = DataLoader(cifar10_train_dataset, batch_size=64, shuffle=True)
cifar10_val_loader = DataLoader(cifar10_val_dataset, batch_size=64, shuffle=False)
cifar10_test_loader = DataLoader(cifar10_test_dataset, batch_size=64, shuffle=False)

# ResNet 모델 불러오기 및 수정
model = models.resnet18(pretrained=False)
num_features = model.fc.in_features
model.fc = nn.Linear(num_features, 10)  # CIFAR-10 클래스 수에 맞게 수정

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

# 손실 함수 및 최적화 함수 정의
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

```python
# dataset.py
import torchvision.transforms as transforms
from torchvision import datasets
from torch.utils.data import DataLoader, random_split

def get_dataloaders(batch_size=128, val_split=0.2, num_workers=2):
    transform_train = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(10),
        transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2),
        transforms.ToTensor(),
        transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
    ])

    transform_test = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
    ])

    cifar10_train_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_train)
    cifar10_test_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform_test)

    train_size = int((1 - val_split) * len(cifar10_train_dataset))
    val_size = len(cifar10_train_dataset) - train_size
    cifar10_train_dataset, cifar10_val_dataset = random_split(cifar10_train_dataset, [train_size, val_size])

    train_loader = DataLoader(cifar10_train_dataset, batch_size=batch_size, shuffle=True, num_workers=num_workers)
    val_loader = DataLoader(cifar10_val_dataset, batch_size=batch_size, shuffle=False, num_workers=num_workers)
    test_loader = DataLoader(cifar10_test_dataset, batch_size=batch_size, shuffle=False, num_workers=num_workers)

    return train_loader, val_loader, test_loader
```

코드 모듈화 – model.py

```python
# 데이터 변환 정의
transform_cifar10 = transforms.Compose([
    transforms.Resize((224, 224)),  # ResNet 모델에 맞추기 위해 이미지 크기를 224x224로 조정
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
])

# CIFAR-10 데이터셋 로드 및 분할
cifar10_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_cifar10)
cifar10_test_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform_cifar10)

# 데이터셋을 학습 세트와 검증 세트로 분할 (80% 학습, 20% 검증)
train_size = int(0.8 * len(cifar10_dataset))
val_size = len(cifar10_dataset) - train_size
cifar10_train_dataset, cifar10_val_dataset = random_split(cifar10_dataset, [train_size, val_size])

# 데이터로더 정의
cifar10_train_loader = DataLoader(cifar10_train_dataset, batch_size=64, shuffle=True)
cifar10_val_loader = DataLoader(cifar10_val_dataset, batch_size=64, shuffle=False)
cifar10_test_loader = DataLoader(cifar10_test_dataset, batch_size=64, shuffle=False)

# ResNet 모델 불러오기 및 수정
model = models.resnet18(pretrained=False)
num_features = model.fc.in_features
model.fc = nn.Linear(num_features, 10)  # CIFAR-10 클래스 수에 맞게 수정

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

# 손실 함수 및 최적화 함수 정의
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```



```python
import torch.nn as nn
from torchvision import models

class ResNet_2(nn.Module):
    def __init__(self, dropout_rate=0.5):
        super(ResNet_2, self).__init__()
        self.model = models.resnet34(weights=None)
        num_features = self.model.fc.in_features
        self.model.fc = nn.Sequential(
            nn.Dropout(dropout_rate),
            nn.Linear(num_features, 10)
        )
    def forward(self, x):
        return self.model(x)
```

코드 모듈화 – train.py

```python
# 모델 학습
def train_model(model, train_loader, val_loader, criterion, optimizer, num_epochs=10):
    train_losses=[]
    val_losses=[]
    train_accuracies=[]
    val_accuracies=[]

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        correct = 0
        total=0
        for i, (inputs, labels) in enumerate(tqdm(train_loader, desc=f"Epoch {epoch
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)#예측값
            loss=criterion(outputs, labels)#loss계산
            loss.backward()#역전파
            optimizer.step()#가중치 업데이트

            running_loss += loss.item() * inputs.size(0) # 배치 loss 저장
            _, predicted = torch.max(outputs, 1)#예측된 클래스
            total+=labels.size(0)#총 레이블 수 업데이트
            correct+=(predicted ==labels).sum().item()#잘 예측된 레이블 업데이트


        epoch_loss =running_loss/len(train_loader.dataset)# loss 계산
        train_losses.append(epoch_loss)
        train_accuracy =100 *correct/total #acc 계산
        train_accuracies.append(train_accuracy)
        print(f"Epoch {epoch+1}/{num_epochs}, Training Loss: {epoch_loss:.4f}, Tra
```

```python
def train_model(model, train_loader, val_loader, criterion, optimizer, scheduler, device, num_epochs=10):
    train_losses = []
    val_losses = []
    train_accuracies = []
    val_accuracies = []

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0

        for i, (inputs, labels) in enumerate(tqdm(train_loader, desc=f"Epoch {epoch + 1}/{num_epochs}")):
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * inputs.size(0)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        epoch_loss = running_loss / len(train_loader.dataset)
        train_losses.append(epoch_loss)
        train_accuracy = 100 * correct / total
        train_accuracies.append(train_accuracy)
```

코드 모듈화 – train(val).py

```python
    # 검증 손실 및 정확도 계산
    model.eval()
    val_running_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_running_loss += loss.item() * inputs.size(0)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    val_loss = val_running_loss / len(val_loader.dataset)
    val_losses.append(val_loss)
    val_accuracy = 100 * correct / total
    val_accuracies.append(val_accuracy)
    print(f"Epoch {epoch+1}/{num_epochs}, Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.2f}%"

return train_losses, val_losses, train_accuracies, val_accuracies
```

```python
    model.eval()
    val_running_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_running_loss += loss.item() * inputs.size(0)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    val_loss = val_running_loss / len(val_loader.dataset)
    val_losses.append(val_loss)
    val_accuracy = 100 * correct / total
    val_accuracies.append(val_accuracy)
    print(
        f"Epoch {epoch + 1}/{num_epochs}, Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.2f}%")

return train_losses, val_losses, train_accuracies, val_accuracies
```

```python
# 모델 평가
def evaluate_model(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    test_losses = []
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            test_losses.append(loss.item())
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    print(f"Test Accuracy: {accuracy:.2f}%")
    return accuracy, test_losses

# 모델 학습 및 평가
num_epochs = 5
train_losses, val_losses, train_accuracies, val_accuracies = train_model(model, cifar10_train_loader, cifar10_val_loader, criterion, optimizer, num_epochs)
accuracy, test_losses = evaluate_model(model, cifar10_test_loader)
```

```python
def evaluate_model(model, test_loader, criterion, device):
    model.eval()
    correct = 0
    total = 0
    test_losses = []
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            test_losses.append(loss.item())
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    print(f"Test Accuracy: {accuracy:.2f}%")
    return accuracy, test_losses
```

코드 모듈화 – main.py

```python
# 데이터 변환 정의
transform_cifar10 = transforms.Compose([
    transforms.Resize((224, 224)),  # ResNet 모델에 맞추기 위해 이미지 크기를 224x224로 조정
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
])

# CIFAR-10 데이터셋 로드 및 분할
cifar10_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_cifar10)
cifar10_test_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform_cifar1

# 데이터셋을 학습 세트와 검증 세트로 분할 (80% 학습, 20% 검증)
train_size = int(0.8 * len(cifar10_dataset))
val_size = len(cifar10_dataset) - train_size
cifar10_train_dataset, cifar10_val_dataset = random_split(cifar10_dataset, [train_size, val_size])

# 데이터로더 정의
cifar10_train_loader = DataLoader(cifar10_train_dataset, batch_size=64, shuffle=True)
cifar10_val_loader = DataLoader(cifar10_val_dataset, batch_size=64, shuffle=False)
cifar10_test_loader = DataLoader(cifar10_test_dataset, batch_size=64, shuffle=False)

# ResNet 모델 불러오기 및 수정
model = models.resnet18(pretrained=False)
num_features = model.fc.in_features
model.fc = nn.Linear(num_features, 10)  # CIFAR-10 클래스 수에 맞게 수정

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

# 손실 함수 및 최적화 함수 정의
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

```python
num_epochs = 20
train_losses, val_losses, train_accuracies, val_accuracies = train_model(model, cifar10_train_loader, cifar10_val_loader, criterion, optimizer, n
accuracy, test_losses = evaluate_model(model, cifar10_test_loader)
```

```python
# main.py
import torch
import torch.nn as nn
import torch.optim as optim
from model import ResNet_2
from dataset import get_dataloaders
from train import train_model, evaluate_model

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

#데이터 불러오기
train_loader,val_loader,test_loader = get_dataloaders(batch_size=128)

#모델정의
model = ResNet_2(dropout_rate=0.5)
model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(),lr=0.1,momentum=0.9)
scheduler = optim.lr_scheduler.StepLR(optimizer,step_size=5,gamma=0.1)


num_epochs = 20
train_losses, val_losses, train_accuracies, val_accuracies = train_model(
    model, train_loader, val_loader, criterion, optimizer, scheduler, device, num_epochs=num_epochs
)

# 모델 평가
test_accuracy, test_losses =evaluate_model(model,test_loader,criterion,device)
```

다양한 기법 적용

로그 출력 및 시각화

- 학습 중 손실 값(loss)과 정확도(accuracy)를 로그로 출력하여 학습 과정을 모니터링

`LoggingCallback` 대신에 `tqdm` 을 사용해 진행상황도 시각적으로 사용할수있도록 함.

```
Epoch 1/20: 100%|██████████| 313/313 [02:44<00:00,  1.90it/s]
Epoch 1/20, Training Loss: 2.0406, Training Accuracy: 26.16%, Learning Rate: 0.1

Epoch 1/20, Validation Loss: 1.7156, Validation Accuracy: 37.37%
Epoch 2/20: 100%|██████████| 313/313 [02:44<00:00,  1.90it/s]
Epoch 2/20, Training Loss: 1.4994, Training Accuracy: 44.39%, Learning Rate: 0.1

Epoch 2/20, Validation Loss: 2.0270, Validation Accuracy: 36.10%
Epoch 3/20: 100%|██████████| 313/313 [02:46<00:00,  1.88it/s]
Epoch 3/20, Training Loss: 1.1987, Training Accuracy: 56.79%, Learning Rate: 0.1

Epoch 3/20, Validation Loss: 4.9415, Validation Accuracy: 33.92%
Epoch 4/20: 100%|██████████| 313/313 [02:43<00:00,  1.91it/s]
Epoch 4/20, Training Loss: 0.9970, Training Accuracy: 64.62%, Learning Rate: 0.1

Epoch 4/20, Validation Loss: 1.8310, Validation Accuracy: 45.13%
Epoch 5/20: 100%|██████████| 313/313 [02:44<00:00,  1.90it/s]
Epoch 5/20, Training Loss: 0.8400, Training Accuracy: 70.38%, Learning Rate: 0.010000000000000002

Epoch 5/20, Validation Loss: 0.8992, Validation Accuracy: 68.23%
```

가중치 값 저장 및 로드

- 모델 학습 후 가중치를 저장, 저장된 가중치를 불러오는 기능 구현

```
PATH = "/kaggle/working/resnet_3.pth"

#모델 가중치 저장
torch.save(model.state_dict(),PATH)

# 저장된 가중치 불러오기
model.load_state_dict(torch.load(PATH))
```

```
<All keys matched successfully>
```

Output (421.6MiB / 19.5GiB)

- /kaggle/working
  - data
  - resnet_3.pth

다양한 기법 적용

스케줄러(scheduler) 등 성능 향상 기법 적용

- lr을 epoch에 따라 동적으로 조절하는 기법
- 처음에는 높은 학습률로 빠르게 수렴시키고, 이후에는 학습률을 점진적으로 낮추어 더 안정적으로 최적화 함.

  `torch.optim.lr_scheduler`
  학습 초기에는 높은 학습률로 빠르게 최적화, 이후에는 학습률을 낮추어 더 안정적인 수렴

- **초기 학습률**: 0.1
- **Epoch 1-5**: 0.1 * 0.1 = 0.01
- **Epoch 6-10**: 0.01 * 0.1 = 0.001
- **Epoch 11-15**: 0.001 * 0.1 = 0.0001
- **Epoch 16-20**: 0.0001 * 0.1 = 0.00001

```
#**스케줄러 사용**
scheduler = optim.lr_scheduler.StepLR(optimizer,step_size=5, gamma=0.1)
```

data augmentation

```
transform_train = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2),
    transforms.RandomAffine(degrees=0,translate=(0.1, 0.1)),#랜덤이동
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
])
```

성능 개선

ResNet18 - 2

ResNet18-1
+ epoch = 20,
Flip, Rotation, ColorJitter

Test Accuracy: 81.78%

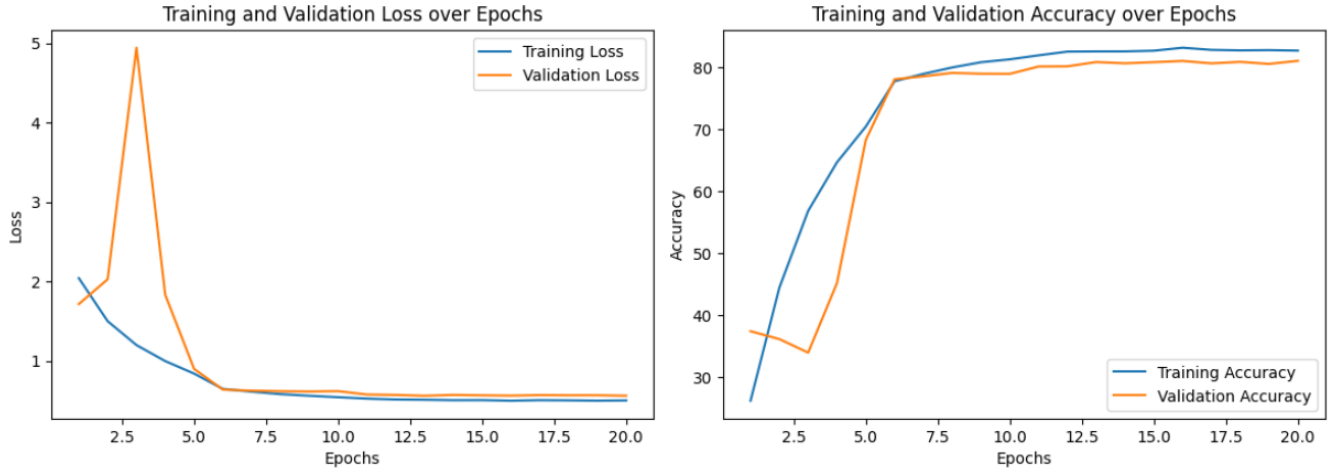
- 모델 아키텍처 변경
  resnet 18→34로 변경
- 학습률 스케줄링

```
scheduler = optim.lr_scheduler.StepLR(optimizer,step_size=5, gamma=0.1)
```

- drop out 사용

```
"""model = models.resnet34(pretrained=False)
num_features = model.fc.in_features
model.fc = nn.Linear(num_features, 10)  # CIFAR-10 클래스 수에 맞게 수정"""

#드롭아웃 추가
class ResNet_2(nn.Module):
    def __init__(self, dropout_rate=0.5):
        super(ResNet_2,self).__init__()
        self.model = models.resnet34(weights=False)
        num_features = self.model.fc.in_features
        self.model.fc = nn.Sequential(
            nn.Dropout(dropout_rate),
            nn.Linear(num_features,10)
        )
    def forward(self,x):
        return self.model(x)

model = ResNet_2(dropout_rate=0.5)
model = model.to(device)
```

Test Accuracy: 82.58%




```
Test Accuracy: 82.58%
```