(인공지능과빅데이터) 연습문제_06

1. 심층 신경망(Deep Neural Network)에서 사용되는 잔차 연결(Residual Connection) 기법에 대한 설명 중 **가장 정확한** 것은 무엇인가?
    1. 잔차 연결은 신경망의 깊이가 깊어질수록 발생하는 기울기 소실 (Vanishing Gradient) 문제를 근본적으로 해결하기 위해 각 레이어의 활성화 함수를 비선형 함수 대신 선형 함수로 대체하는 방법이다. 레이어 출력을 입력에 더해주는 방식, 근본해결은 아님
    2. 잔차 연결은 입력 데이터를 건너뛰어(skip) 특정 레이어 블록의 출력을 입력과 직접 더해주는 방식으로, 이는 네트워크가 항등 함수(identity function)를 학습하는 것을 어렵게 만들어 최적화 난이도를 증가시킨다. 항등함수 학습을 쉽게 만든다. 최적화를 도움
    $H(x) = F(x) + x \rightarrow F(x) = 0$
    3. 잔차 연결은 네트워크의 초기 레이어에서 추출된 저수준(low-level) 특징들이 깊은 레이어까지 효과적으로 전달될 수 있도록 하여, 네트워크가 더욱 복잡하고 추상적인 특징을 학습하는 데 도움을 준다. ❤
    4. 잔차 연결을 사용하는 ResNet(Residual Network) 구조는 깊이가 매우 깊은 신경망에서도 성능 저하 없이 효과적인 학습이 가능하도록 하며, 이는 잔차 블록 내의 가중치 레이어 수를 무한대로 늘릴 수 있기 때문이다. 무한대는 아니다.
    5. 잔차 연결은 순환 신경망(Recurrent Neural Network, RNN)에서 시간 순서대로 발생하는 기울기 소실 문제를 해결하기 위해 처음 제안되었으며, 이후 컨볼루션 신경망(Convolutional Neural Network, CNN)에도 효과가 입증되어 널리 사용되고 있다. CNN 에서 신경망의 깊이와 관련하여 처음 연구되었다. 이후 RNN에도 변형 적용시키는 연구가 나온다
2. PyTorch를 사용하여 심층 신경망 구조에 잔차 연결(Residual Connection)을 적용하는 방식과 관련된 설명 중 **가장 정확한** 것은 무엇인가?
    1. PyTorch에서 잔차 연결을 구현하기 위해서는 `nn.ResidualBlock` 이라는 별도의 레이어 클래스를 사용해야 하며, 이를 통해 입력 텐서와 특정 레이어 블록의 출력을 자동으로 더해줄 수 있다. 제공하지 않음. 직접 잔차 블록 구조를 정의하고 순방향 연산에서 덧셈을 구현
    2. 잔차 연결을 적용할 때, 스킵 연결(skip connection)되는 입력 텐서와 레이어 블록의 출력 텐서의 차원(shape)이 반드시 동일해야 하며, 차원이 다를 경우 PyTorch는 자동으로 이를 맞춰주는 기능을 제공한다. 직접 차원을 맞춰주는 연산(예: 1x1 컨볼루션 레이어를 스킵 연결에 적용)을 사용자가 구현해야 합.

    3. PyTorch에서 잔차 연결은 순차적인 레이어 구성 방식인 `nn.Sequential` 내에서만 적용 가능하며, 함수형 API(`torch.nn.functional`)를 사용하는 경우에는 직접 덧셈 연산을 구현해야 한다. 두가지 형식에서 직접 구현해야 함이 다르지 않다.
    4. 잔차 연결을 구현할 때, 스킵 연결되는 입력 텐서에 활성화 함수를 적용한 후 레이어 블록의 출력과 더해야 기울기 소실 문제 완화에 더 효과적이다. 스킵 연결은 원본 정보를 그대로 전달하는 역할. 입력 텐서에는 별도의 활성화 함수를 적용하지 않고, 레이어 블록의 출력에 활성화 함수를 적용
    5. PyTorch에서 잔차 연결은 일반적으로 사용자 정의 모듈(class 상속 `nn.Module`) 내에서 구현되며, 순방향(forward) 함수에서 입력 텐서를 저장하고 특정 레이어 블록을 통과시킨 후 저장된 입력 텐서와 결과를 직접 더하는 방식으로 구현된다. ❤
3. 인공지능 모델 학습 시 과적합(Overfitting)을 방지하기 위해 사용되는 조기 학습 종료(Early Stopping) 기법에 대한 설명 중 **가장 정확한** 것은 무엇인가?
    1. 조기 학습 종료는 학습 데이터셋에 대한 손실(loss)이 더 이상 감소하지 않거나 증가하기 시작할 때 학습을 강제로 멈추는 기법이다. 학습 데이터가 아니다. 검증 데이터의 손실을 지표로 사용할 수 있다.
    2. 조기 학습 종료는 검증 데이터셋(validation dataset)에 대한 성능 지표(예: 정확도, 손실)가 개선되지 않거나 악화되기 시작할 때 학습을 멈추는 기법이다. ❤
    3. 조기 학습 종료는 학습률(learning rate)을 점진적으로 감소시키는 학습 스케줄링(learning rate scheduling) 기법의 일종으로, 학습 후반부의 과도한 파라미터 업데이트를 방지한다. 학습 스케줄링과는 다른 기법이다.
    4. 조기 학습 종료를 적용하기 위해서는 학습 데이터셋과 검증 데이터셋 외에 별도의 테스트 데이터셋을 준비하여 학습 중 모델의 최종 성능을 주기적으로 평가해야 한다. 테스트 데이터셋은 사용하면 안된다.
    5. 조기 학습 종료는 모델의 학습 시간을 단축시키는 데에만 목적이 있으며, 과적합 방지 효과는 부수적인 결과이다. 주요 목적이 과적합 방지이다. 학습 시간 단축은 부수적인 효과이다.
4. PyTorch 환경에서 조기 학습 종료(Early Stopping) 기법을 적용하는 방식에 대한 설명 중 **가장 정확한** 것은 무엇가?
    1. PyTorch는 `torch.optim.EarlyStopping` 클래스를 내장하여 제공하며, 학습 루프 내에서 해당 객체를 사용하여 검증 손실을 자동으로 추적하고 학습 중단 여부를 결정할 수 있다. 조기 학습 종료를 위한 내장된 클래스 없음
    2. 조기 학습 종료를 PyTorch에서 구현할 때, 학습 루프 내에서 매 에폭(epoch)마다 검증 데이터셋으로 모델을 평가하고, 검증 손실이 이전 최고 기록보다 나빠지면 즉시 학습을 중단해야 한다. `` `patience`라는 파라미터를 사용하여 일정 에폭 동안 성능 향상이 없을 때 종료` ``
    3. PyTorch에서 조기 학습 종료를 효과적으로 적용하기 위해서는 학습률 스케줄러(learning rate scheduler)와 반드시 함께 사용해야 하며,

이를 통해 학습 후반부의 성능 향상을 유도하고 조기 종료 시점을 늦출 수 있다. 필수적인 것은 아님. 각각 과적합 방지 및 최적화라는 다른 목적

4. PyTorch에서 조기 학습 종료를 구현할 때, 학습 루프 외부에서 별도의 스레드 또는 프로세스를 생성하여 검증 손실을 모니터링하고, 성능 저하가 감지되면 주 학습 루프에 종료 신호를 보내는 방식으로 비동기적으로 처리하는 것이 효율적이다. 학습 루프 내에서 동기적으로 검증 성능을 평가하고 종료 여부를 결정
5. PyTorch에서 조기 학습 종료는 일반적으로 사용자 정의 클래스로 구현되며, 해당 클래스는 검증 손실 또는 다른 성능 지표를 추적하고, 지정된 patience(개선이 없는 에폭 수) 동안 성능 향상이 없으면 학습을 중단하는 로직을 포함한다. ❤

5. 인공지능 모델 학습 시 학습률(Learning Rate)을 동적으로 조정하는 학습률 스케줄러(Learning Rate Scheduler)에 대한 설명 중 **가장 부적절한** 것은 무엇인가?
    1. 학습률 스케줄러는 학습 과정 동안 고정된 학습률을 사용하는 대신, 특정 조건이나 스케줄에 따라 학습률의 값을 변경하여 모델의 수렴 속도와 성능 향상을 도울 수 있다.
    2. 대표적인 학습률 스케줄러로는 StepLR, MultiStepLR, ExponentialLR 등이 있으며, 이들은 주로 학습 에폭(epoch) 수에 따라 학습률을 감소시키는 방식을 사용한다.
    3. 학습률 스케줄러를 효과적으로 사용하기 위해서는 초기 학습률을 매우 작은 값으로 설정하고, 학습이 진행됨에 따라 점진적으로 증가시키는 전략이 일반적이다. ❤
    4. 학습률 스케줄러는 모델이 학습 초기에 빠르게 최적 지점 근처로 이동하도록 돕고, 후반부에는 세밀한 조정을 통해 지역 최적해(local optima)를 탈출하거나 더 나은 해를 찾도록 유도할 수 있다.
    5. 일부 고급 학습률 스케줄러는 검증 데이터셋의 성능 변화를 모니터링하여 학습률을 자동으로 조정하기도 하며, 이는 조기 학습 종료(Early Stopping) 기법과 함께 사용하여 시너지 효과를 낼 수 있다.
6. PyTorch 환경에서 학습률 스케줄러(Learning Rate Scheduler)를 사용하는 방식에 대한 설명 중 **가장 부적절한** 것은 무엇인가?
    1. PyTorch에서 학습률 스케줄러는 `torch.optim.lr_scheduler` 모듈에 다양한 클래스 형태로 제공되며, 이를 사용하여 옵티마이저(optimizer)의 학습률을 관리할 수 있다.
    2. 학습률 스케줄러 객체를 생성할 때, 반드시 최적화 대상인 옵티마이저 객체를 인자로 전달해야 하며, 일부 스케줄러는 추가적인 파라미터(예: 감소 비율, 스텝 크기)를 요구하기도 한다.
    3. PyTorch의 학습 루프 내에서 학습률을 업데이트하기 위해서는 일반적으로 각 배치(batch)마다 스케줄러 객체의 `step()` 메서드를 호출해야 한다.
    4. 검증 데이터셋의 성능을 기반으로 학습률을 조정하는 `ReduceLROnPlateau` 스케줄러를 사용할 경우, 매 에폭(epoch)마다 검

증 성능 지표를 스케줄러의 `step()` 메서드에 인자로 전달해야 학습률이 적절하게 조정된다.

5. 학습률 스케줄러는 옵티마이저의 학습률을 직접 변경하므로, 학습 루프 내에서 `optimizer.param_groups`의 `lr` 값을 직접 수정하는 방식과 혼용하여 사용하는 것이 권장된다. ❤ 값을 직접 수정하는 것은 스케줄러의 작동을 방해하고 스케줄러의 작용예측성을 떨어 뜨려 권장하지 않는다.

7. TensorFlow 환경에서 조기 학습 종료(Early Stopping) 기법을 적용하는 가장 일반적이고 정확한 방법은 무엇인가?
    1. 학습 루프 내에서 매 에폭마다 검증 손실을 직접 비교하여 손실이 증가하는 시점에서 `break` 문을 사용하여 학습을 수동으로 중단시키는 것이다. TF는 전용 콜백 기능이 존재한다.
    2. TensorFlow의 `tf.keras.callbacks.TerminateOnNaN` 콜백 함수를 사용하여 학습 데이터의 손실이 NaN으로 발산하는 경우에만 학습을 자동으로 중단시키는 것이다. 손실 발산 시 중단하는 것이 주 목적
    3. TensorFlow의 `tf.train.CheckpointManager`를 사용하여 주기적으로 모델의 가중치를 저장하고, 검증 성능이 더 이상 향상되지 않으면 가장 성능이 좋았던 체크포인트를 복원하여 학습을 종료하는 것이다. 모델 복원용이지 중단목적이 아니다.
    4. TensorFlow의 `tf.keras.callbacks.EarlyStopping` 콜백 함수를 사용하여 학습 중 검증 세트의 특정 지표(예: 손실, 정확도)를 모니터링하고, 지정된 `patience` 기간 동안 개선이 없으면 학습을 자동으로 중단시키는 것이다. ❤ TF 표준 권장 방법
    5. 학습률 스케줄러(`tf.keras.optimizers.schedules`)의 한 종류인 `tf.keras.callbacks.EarlyStopping` 스케줄러를 사용하여 검증 손실이 감소하지 않으면 학습률을 점진적으로 0으로 만들어 학습을 종료시키는 것이다. 학습률 스케줄러와는 다른 것이다.

8. TensorFlow 환경에서 학습률 스케줄러(Learning Rate Scheduler)를 구현하는 가장 일반적이고 정확한 방법은 무엇인가?
    1. 학습 루프 내에서 `optimizer.learning_rate.assign(new_learning_rate)`와 같은 코드를 사용하여 매 스텝 또는 에폭마다 학습률을 직접 수동으로 변경한다. 직접 할당하는 것은 좋은 방법이 아니다.
    2. TensorFlow의 `tf.keras.optimizers.Optimizer` 클래스의 `learning_rate` 속성에 함수를 할당하여 학습 스텝마다 해당 함수가 호출되어 학습률을 동적으로 계산하도록 한다. 이것도 권장하는 방법이 아니다.
    3. TensorFlow의 `tf.train.exponential_decay`와 같은 함수를 사용하여 학습 에폭 수에 따라 지수적으로 감소하는 학습률을 정의하고, 이를 `tf.keras.optimizers.Adam`과 같은 옵티마이저에 직접 할당한다. 가능은 하나 스케줄러 콜백함수와 함께 사용하는 것인 학습방법 통합 측면에서 권장된다.

4. TensorFlow의 `tf.keras.callbacks.LearningRateScheduler` 콜백 함수를 사용하여 학습률을 변경하는 사용자 정의 함수를 정의하고, `model.fit()` 메서드의 `callbacks` 인자를 통해 적용한다. ❤ 권장되는 방법
5. TensorFlow에서는 학습률 스케줄링을 위해 별도의 모듈이나 콜백 함수를 제공하지 않으므로, 사용자가 직접 학습 루프 내에서 조건문을 사용하여 학습률을 조정하는 방식을 사용해야 한다. 제공함.