Scheduler

작성일시	@2022년 4월 18일 오후 10:40
유형	개념💡
상태	
에픽	
스프린트	
우선순위	
작업	
시작일자	@2022년 4월 18일
제품 관리자	
엔지니어	

Learning Rate 임의 변경

- 학습에 사용되는 learning rate를 변경하기 위해서는 optimizer로 선언한 oprimizer 객체를 직접 접근하여 수정
- 1개의 optimizer를 사용한다면, `optimizer.param_groups[0]`을 통하여 현재 dictionary 형태의 optimizer 정보에 접근 가능
  - 그 중, `lr`이라는 key를 이용하여 learning rate의 value값을 접근할 수 있다.

```
optimizer.param_groups[0]['lr'] /= 2
```

공통 parameter

- **optimizer** : 학습 시에 사용하는 optimizer
- **last_epoch** : 마지막 epoch의 index
- **verbose** : 만약 `True`라면 learning rate 업데이트 시에 message를 stdout으로 프린트하다.

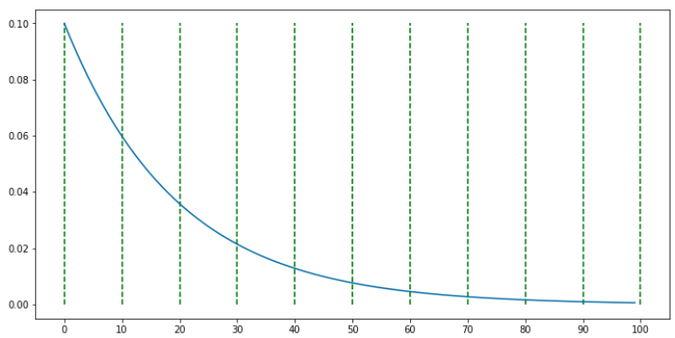
LambdaLR

scheduling 방법을 `lambda 함수` 또는 `함수`를 이용해 결정한다.

Parameter

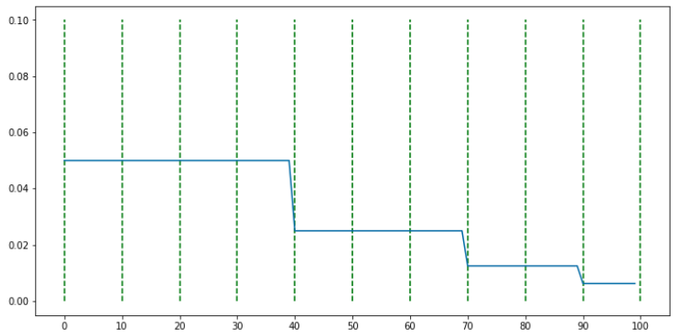
- **lr_lambda** : scheduling 함수

```
scheduler = LambdaLR(optimizer, lr_lambda = lambda epoch : 0.95 ** epoch)
```




```
def func(epoch) :
    if epoch < 40 :
        return 0.5
    elif epoch < 70 :
        return 0.5 ** 2
    elif epoch < 90 :
        return 0.5 ** 3
    else :
        return 0.5 ** 4

scheduler = LambdaLR(optimzier, lr_lambda = func)
```



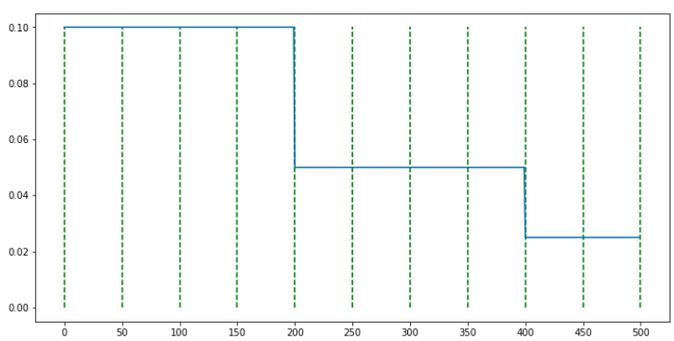
StepLR

일정한 step마다 learning rate에 gamma를 곱해준다.

parameter

- **step_size** : learning rate 갱신 주기
- **gamma** : step_size마다 learning rate에 gamma만큼 곱해준다.

```
scheduler = StepLR(optimizer, step_size = 200, gamma = 0.5)
```



200, 400 epoch에서 learning rate가 절반이 되는 모습을 확인할 수 있다.

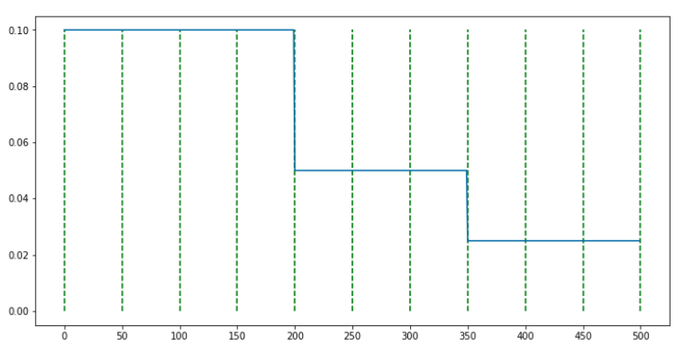
MultiStepLR

step size를 여러 기준으로 적용할 수 있는 `StepLR의 확장 버전`

parameter

- **milestones** : 리스트 형태의 step 기준
- **gamma** : step_size마다 learning rate에 gamma만큼 곱해준다.

```
scheduler = MultiStepLR(optimizer, milestones = [200, 350], gamma = 0.5)
```



200, 350 epoch에서 learning rate가 줄어드는 모습을 확인할 수 있다.

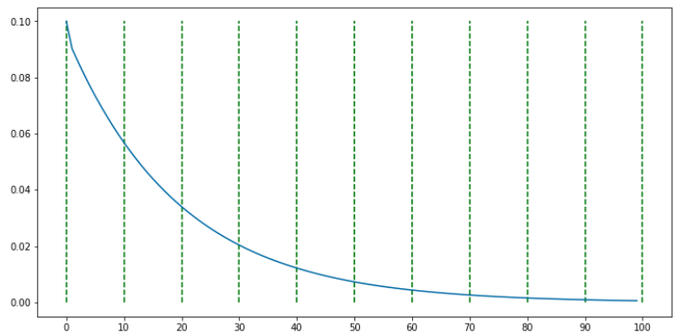
ExponentialLR

지수적으로 learning rate가 감소하는 방법

parameter

- **gamma** : 감소율

```
scheduler = ExponentialLR(optimzier, gamma = 0.95)
```



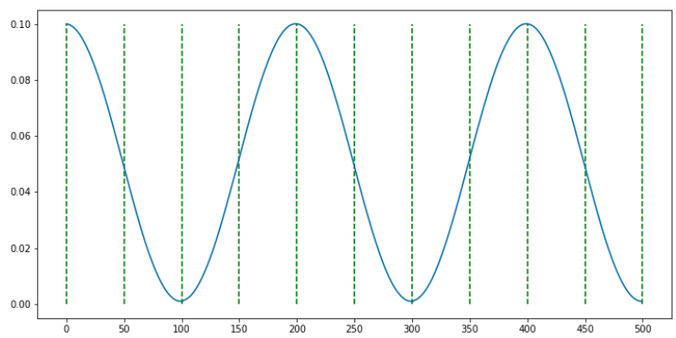
CosineAnnealingLR

cosine 그래프를 그리면서 learning rate가 진동한다. 이 방법을 통해 learning rate가 단순히 감소하기보다는 진동하면서 최적점을 찾아간다.

parameter

- **T_max** : cosine의 반주기
- **eta_min** : learning rate의 최소값

```
scheduler = CosineAnnealingLR(optimizer, T_max = 100, eta_min = 0.001)
```



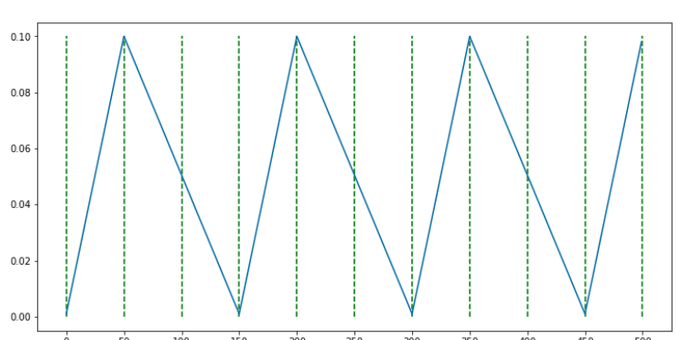
CyclicLR

CosineAnnealingLR은 단순한 cosine 곡선인 반면에 CyclicLR은 3가지 모드를 지원하면서 변화된 형태로 주기적인 learning rate 증감을 지원한다.

parameter

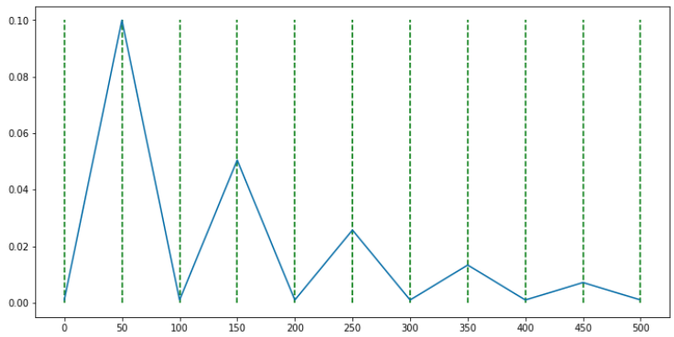
- **base_lr** : learning rate의 가장 작은 점인 lower bound
- **max_lr** : learning rate의 가장 큰 점인 upper bound
- **step_size_up** : base_lr → max_lr의 주기
- **step_size_down** : max_lr → base_lr의 주기
- **mode** : 증감 형태 `triangular`, `triangular2`, `exp_range`

```
scheduler = CyclicLR(optimizer, base_lr = 0.001, max_lr = 0.1, step_size_up = 50, step_size_down = 100, mode = 'triangular')
```



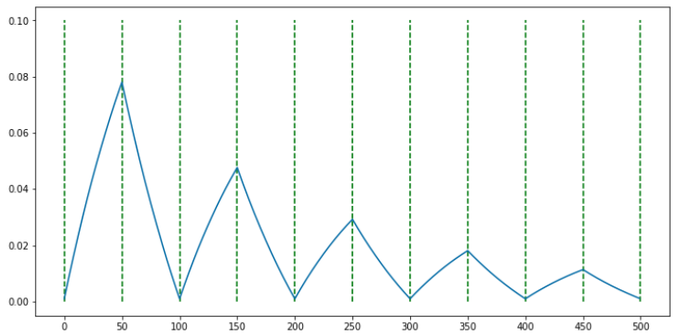
`base_lr = 0.001` 부터 시작하여 `step_size_up = 50 epoch` 동안 증가하여 `max_lr = 0.1`에 도달하게 된다. 그 이후, `step_size_down = 100 epoch` 동안 감소하여 `base_lr = 0.001` 에 도달하게 된다.

```
scheduler = CyclicLR(optimizer, base_lr=0.001, max_lr=0.1, step_size_up=50, step_size_down=None, mode='triangular2')
```

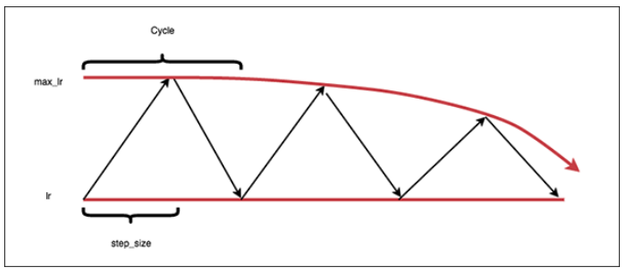


`max_lr`의 값이 반씩 줄어든다. 또한 `step_size_down = None` 은 step_size_up과 주기가 같다는 것을 의미한다.

```
scheduler = CyclicLR(optimizer, base_lr=0.001, max_lr=0.1, step_size_up=50, step_size_down=None, mode='exp_range', gamma=0.995)
```



triangle2와 유사하다. 하지만, `max_lr`의 선형 증감이 아닌 지수적 증감 방식을 사용한다. 따라서 지수식의 밑에 해당하는 `gamma` 값을 따로 사용한다.

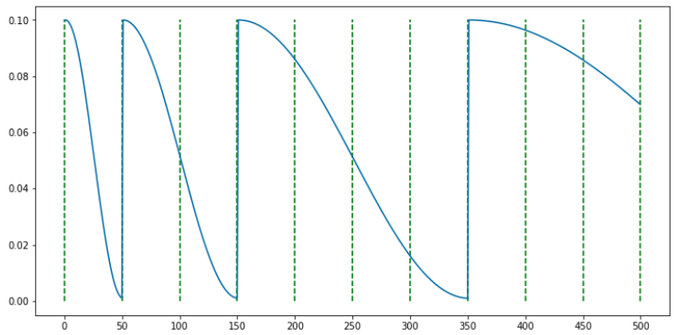


CosineAnnealingWarmRestart

parameter

- **T_0** : 최초의 주기값
- **T_multi** : 주기가 반복되면서 최초 주기값에 비해 얼만큼 주기를 늘려나갈지를 결정하는 값
- **eta_min** : learning rate의 최솟값

```
scheduler = CosineAnnealingWarmRestarts(optimizer, T_0=50, T_mult=2, eta_min=0.001)
```



주기가 2배씩 증가하는 모습을 확인할 수 있다.

출처 : Pytorch Learning Rate Scheduler (러닝 레이트 스케쥴러) 정리

출처 : Pytorch