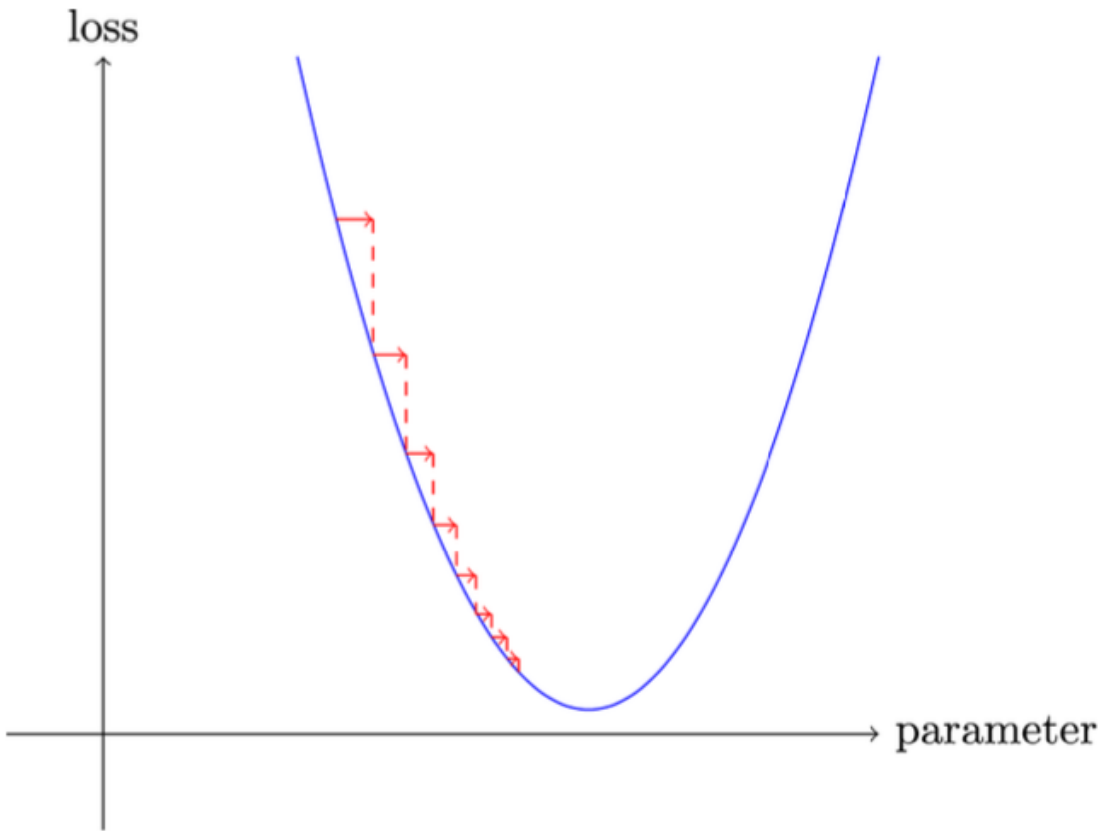
Chapter 4) LR & SGD

Stepping with a Learning Rate

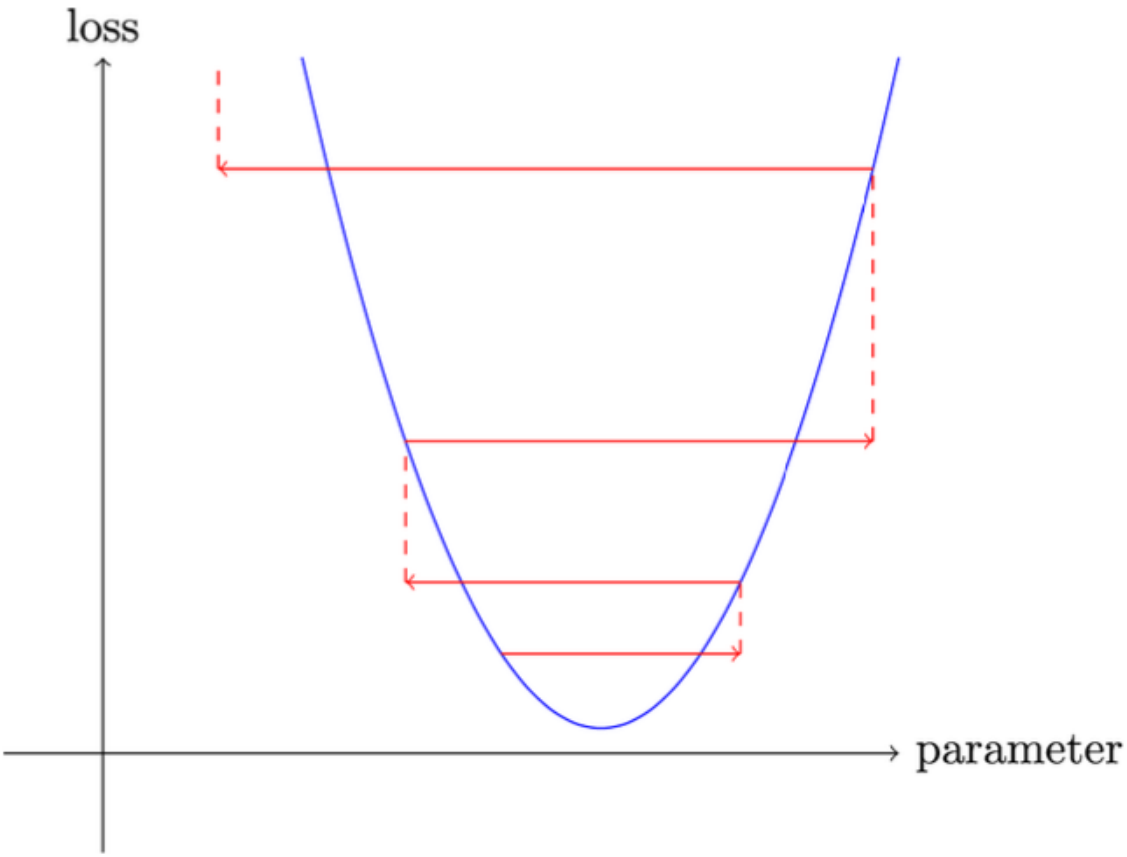
- 그래디언트 값을 기반으로 매개 변수를 변경하는 방법 결정하기
- 기울기 x 학습률(Learning Rate, 0.001 ⇒ 1e-3부터 0.1 사이 숫자) :
- 학습을 한 후 가장 좋은 결과를 내는 모델에 따라 학습률 결정.

```
w -= w.grad * lr
```

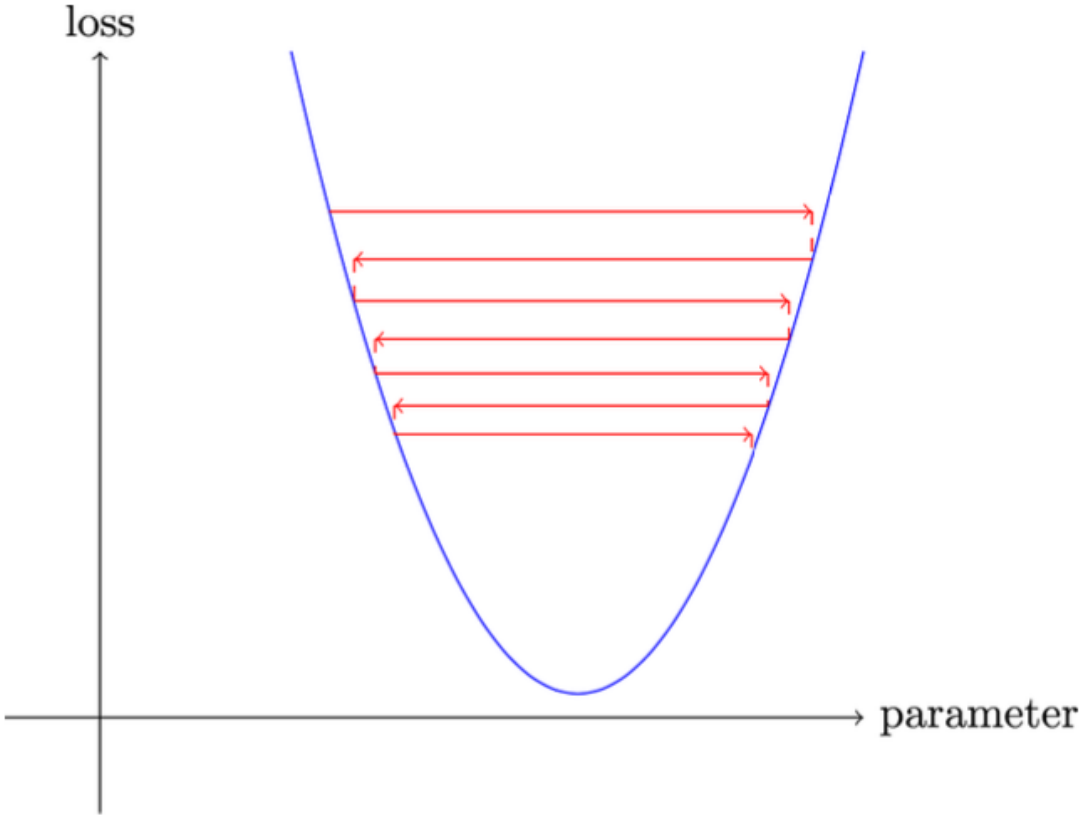
- 파라미터를 stepping한다고 표현합니다.
- LR가 너무 작을 때



- LR가 너무 클 때



- LR가 너무 클 때(바운스)



SGD 예제 : 처음부터 끝까지

- 손실을 최소화하기 위해 그라디언트를 사용하는 방법을 알아보았고, SGD 예제를 톡해 최소값을 찾아 모델을 피팅시키는 방법을 알아보자!

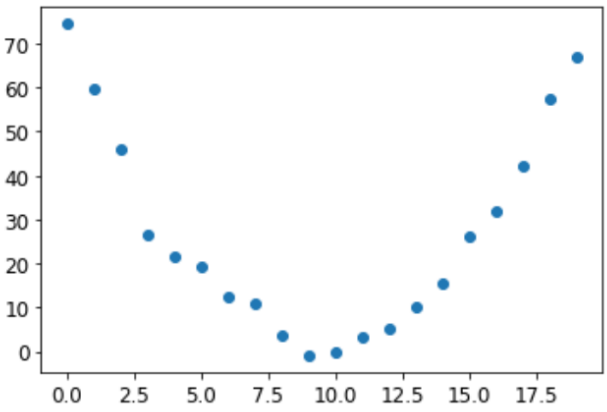
간단한 합성 예제 모델

- 롤러코스터에서 꼭대기에 다다를 때는 느려지고, 내려갈 때는 빨라지는 것을 떠올려보자.
- 20초에 한번씩 점을 찍는다면 아래와 같이 나올 것!

```
time = torch.arange(0,20).float(); time
```

```
tensor([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11., 12., 13.,
 > 14., 15., 16., 17., 18., 19.])
```

```
speed = torch.randn(20)*3 + 0.75*(time-9.5)**2 + 1
plt.scatter(time,speed);
```



- 수치를 손으로 측정하는 건 정확하지 않아서 노이즈를 추가
- SGD를 사용해서 우리가 관찰한 함수를 a * (time ** 2) + (b * time) + c 형식의 2차원 함수로 추측
- 시간 t와 매개 변수(2차 정의 값, 즉 a, b, c)를 분리

```
def f(t, params):
    a,b,c = params
    return a*(t**2) + (b*t) + c
```

- 가장 좋은 2차 함수를 찾기 위해 가장 좋은 a, b, c 찾기
- 이 방법을 신경망과 같이 매개 변수가 더 많은 다른 복잡한 함수에도 동일한 접근 방식을 적용 가능

```
def mse(preds, targets): return ((preds-targets)**2).mean()
```

1단계) 파라미터 초기화하기

- 매개 변수를 임의의 값으로 초기화하고 requires_grad_를 사용하여 경사를 찾겠다고 PyTorch에 알리기

```
params = torch.randn(3).requires_grad_()
```

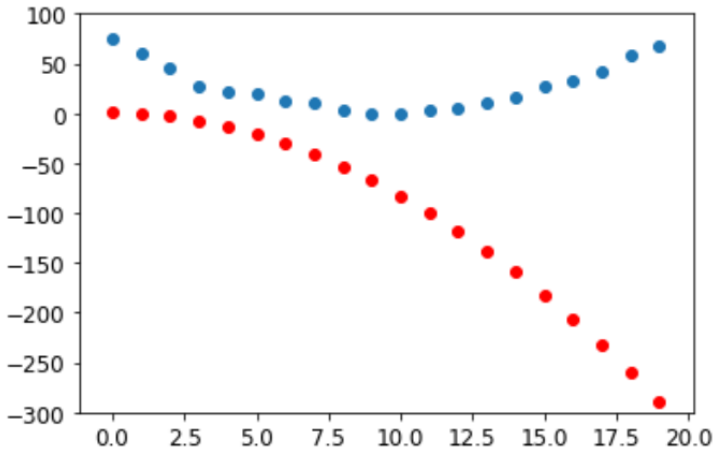
2단계) 예측 계산하기

```
preds = f(time, params)
```

- 예측이 목표에 얼마나 가까운 지 확인하는 함수 작성

```
def show_preds(preds, ax=None):
    if ax is None: ax=plt.subplots()[1]
    ax.scatter(time, speed)
    ax.scatter(time, to_np(preds), color='red')
    ax.set_ylim(-300,100)

show_preds(preds)
```



- 가까워지지 않음. 임의의 매개 변수는 속도가 음수이므로 반대로 가도록 변경

3단계) loss 계산하기

```
loss = mse(preds, speed)
loss
```

```
tensor(25823.8086, grad_fn=<MeanBackward0>)
```

- 이 수치를 개선하는 것이 목표, 이제 경사를 알아보자!

4단계) 경사 계산하기

```
loss.backward()
params.grad
```

```
tensor([-53195.8594,  -3419.7146,   -253.8908])
```

```
params.grad * 1e-5
```

```
tensor([-0.5320, -0.0342, -0.0025])
```

- 경사를 사용하여 매개 변수 개선 가능. LR(학습률) 선택하기
- 다음 장에서 LR 선택하는 법 알아볼 것. 지금은 0.00001

```
params
```

```
tensor([-0.7658, -0.7506,  1.3525], requires_grad=True)
```

5단계) 가중치 계산하기

- 계산한 경사를 바탕으로 매개변수를 업데이트합니다.

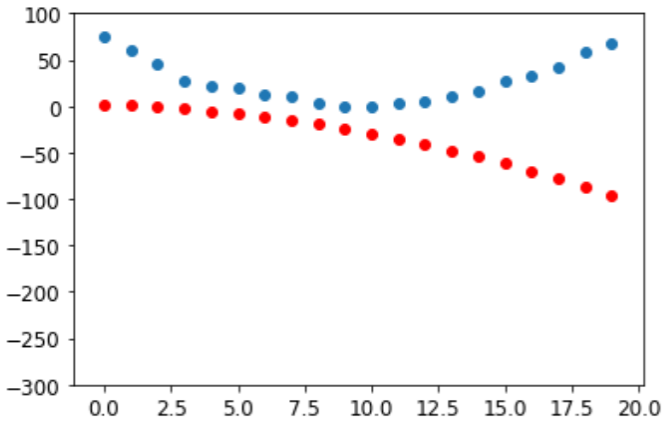
```
lr = 1e-5
params.data -= lr * params.grad.data
params.grad = None
```

- loss 값이 향상되었는지 확인해볼까요?

```
preds = f(time,params)
mse(preds, speed)
```

```
tensor(5435.5366, grad_fn=<MeanBackward0>)
```

```
show_preds(preds)
```



- 1 step을 반영할 함수를 만듭니다.

```
def apply_step(params, prn=True):
    preds = f(time, params)
    loss = mse(preds, speed)
    loss.backward()
    params.data -= lr * params.grad.data
    params.grad = None
    if prn: print(loss.item())
    return preds
```

6단계) 절차 반복하기

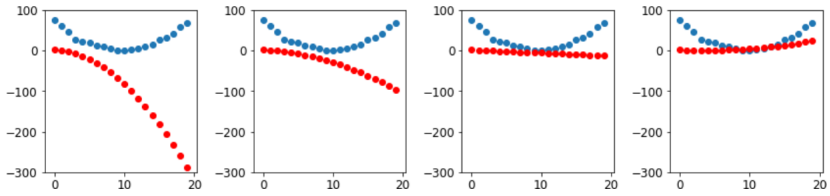
```
for i in range(10): apply_step(params)
```

```
5435.53662109375
1577.4495849609375
847.3780517578125
709.22265625
683.0757446289062
678.12451171875
677.1839599609375
677.0025024414062
676.96435546875
676.9537353515625
```

- 바라는대로 loss가 줄어들고 있음

- 그런데 완전히 다른 2차 함수가 만들어지고 있음
- loss(손실) 함수 대신 모든 단계의 함수를 시각화하면 프로세스를 시각적으로 볼 수 있음

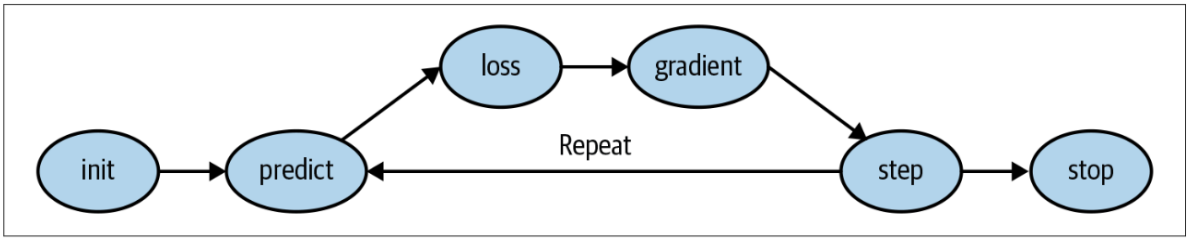
```
_,axs = plt.subplots(1,4,figsize=(12,3))
for ax in axs: show_preds(apply_step(params, False), ax)
plt.tight_layout()
```



7단계) 그만하기

- 10번 반복 후에 중단하기로 결정
- 학습과 검증 지표에 따라 중단할 지점을 결정

경사 하강법 요약



- 모델의 가중치는 처음부터 학습했다면 무작위, 전이 학습으로 학습되었다면 모델에서 올 수 있음
- 처음부터 학습했으면 출력값은 우리가 원하는 것과 거리가 멀고, 사전 훈련된 모델은 목표 작업에 적합하지 않을 수 있음.
- 따라서 더 나은 가중치를 학습해야 함.

- loss fuction(손실 함수)를 활용하여 모델의 출력을 타겟값과 비교

- loss function(손실 함수)를 사용하여 해당 대상을 비교하고 얻은 수치의 예측이 얼마나 잘못되었는지 알아내고, 가중치를 조금씩 변경하여 개선
- loss 값을 낮추기 위해 가중치를 변경하는 방법을 찾으려면 미적분을 사용하여 기울기를 계산
- 우리는 기울기의 크기(즉, 경사의 가파른 정도)를 사용하여 얼마나 큰 step을 가야 하는지 알 수 있음. step 크기를 결정하기 위해 학습률이라고 하는 우리가 선택한 숫자로 기울기를 곱함. 그런 다음 가장 낮은 지점 (주차장)에 도달 할 때까지 반복하고, 중단