- 이미지 출처 : 밑바닥부터 시작하는 딥러닝

SGD

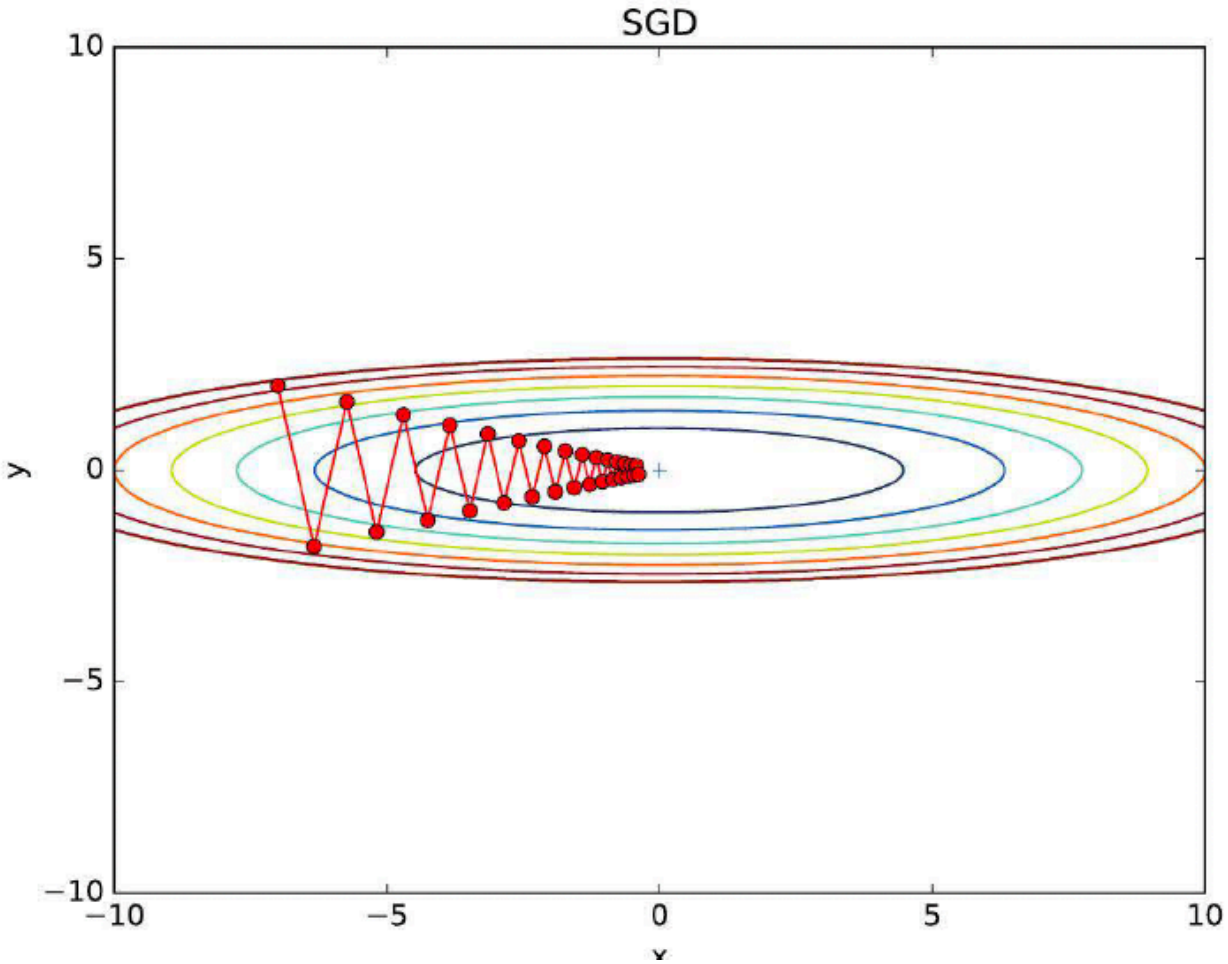
$$\theta_{t+1} \leftarrow \theta_t - \eta \nabla_\theta J(\theta)$$

- Stochastic Gradient Descent
- 가장 기본적인 옵티마이저로, 무작위로 선택된 데이터 샘플에 대한 손실 함수의 기울기를 계산하여 가중치를 업데이트
- 구현이 간단하고 빠르지만, 학습률(learning rate) 설정에 민감하고 지역 최솟값(local minima)에 빠지기 쉬움
- tensorflow에서 learning_rate의 기본값은 0.01
- momentum 매개변수에 0 이상의 값을 지정하면 모멘텀 최적화를 수행
- nesterov 매개변수를 True로 설정하면 네스테로프 모멘텀 최적화를 수행

```
optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)


# 문자열로 넘길수도 있음
#model.compile(loss='mean_squared_error', optimizer='sgd')
```

그림 6-3 SGD에 의한 최적화 갱신 경로 : 최솟값인 (0, 0)까지 지그재그로 이동하니 비효율적이다.



- 전체 데이터에 대해 역전파를 하는 것이 아니라 Batch 단위로 역전파을 하기 때문에 일직선이 아니라 지그재그로 이동

Momentum

그림 6-4 모멘텀의 이미지 : 공이 그릇의 곡면(기울기)을 따라 구르듯 움직인다.

$$v_t = \gamma v_{t-1} - \eta \nabla_\theta J(\theta)$$

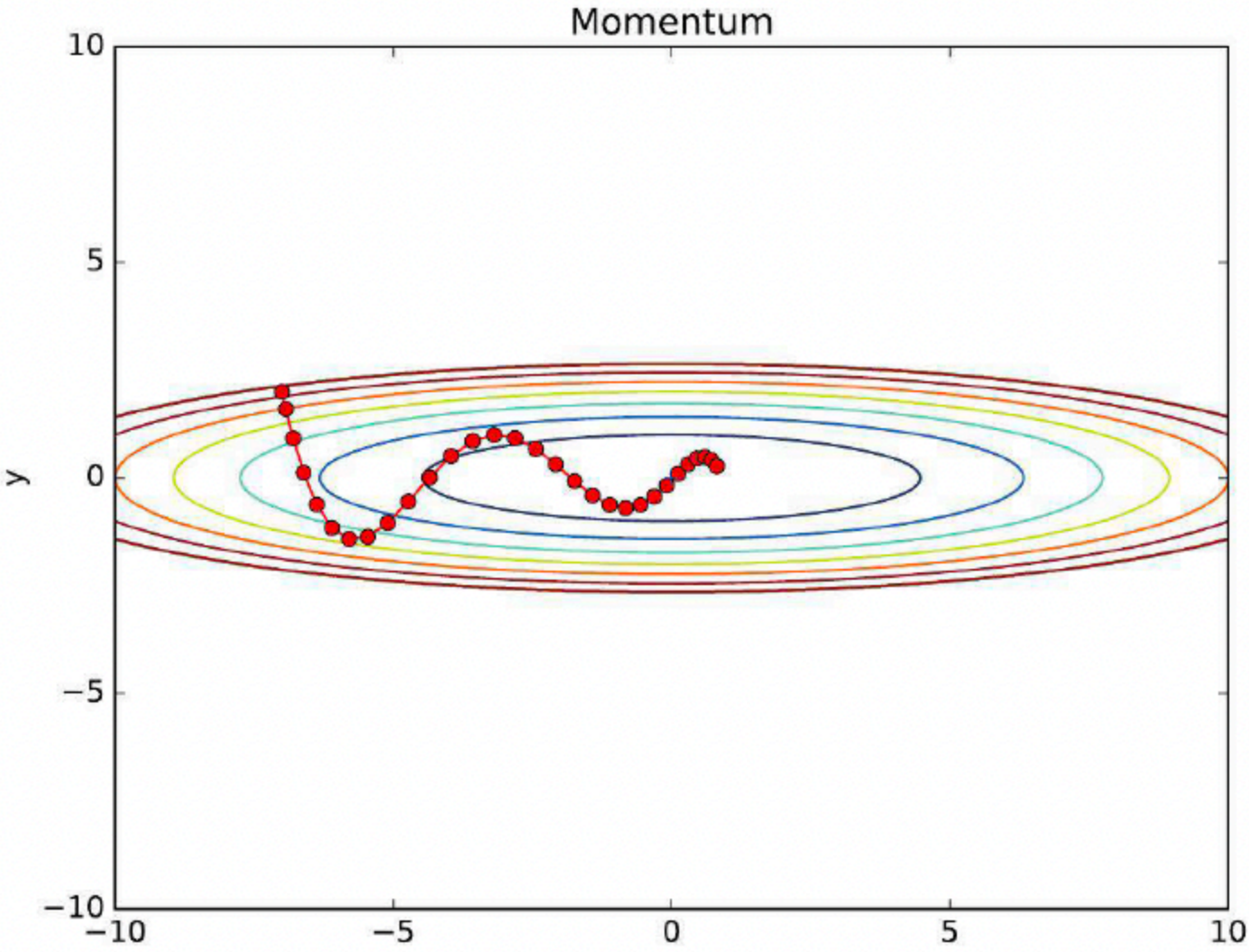
$$\theta_{t+1} \leftarrow \theta_t - v_t$$

- 위의 수식에서 γ가 Momentum의 파라미터
- SGD의 단점을 보완하기 위해 운동량(momentum) 개념을 도입
- 이전 업데이트 방향을 고려하여 가중치를 업데이트
- 걸어가는 보폭을 크게 하는 개념이라고 이해할수 있음

- Local Minimum을 지나칠 수도 있다는 장점이 있음
- SGD보다 빠르게 수렴하고 지역 최솟값을 벗어나는 데 도움이 되지만, 학습률 설정이 여전히 중요함

```
optimizer = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
```

그림 6-5 모멘텀에 의한 최적화 갱신 경로



Adagrad

- 각 매개변수에 대해 적응적으로 학습률을 조절
- 자주 업데이트되는 매개변수의 학습률은 낮추고, 드물게 업데이트되는 매개변수의 학습률은 높임
- 가보지 않은 곳은 많이 움직이고, 가본 곳은 조금씩 움직이자라는 개념

$$G_t = G_{t-1} - (\nabla_\theta J(\theta_t))^2$$

$$\theta_{t+1} \leftarrow \theta_t - \frac{\eta}{\sqrt{G+\epsilon}} \nabla_\theta J(\theta_t)$$

- Gt: t 번째 시점까지 각 매개변수의 기울기 제곱합을 누적하여 저장하는 변수 - ∇θJ(θt): t 번째 시점에서의 손실 함수 J 를 매개변수 θ 에 대해 편

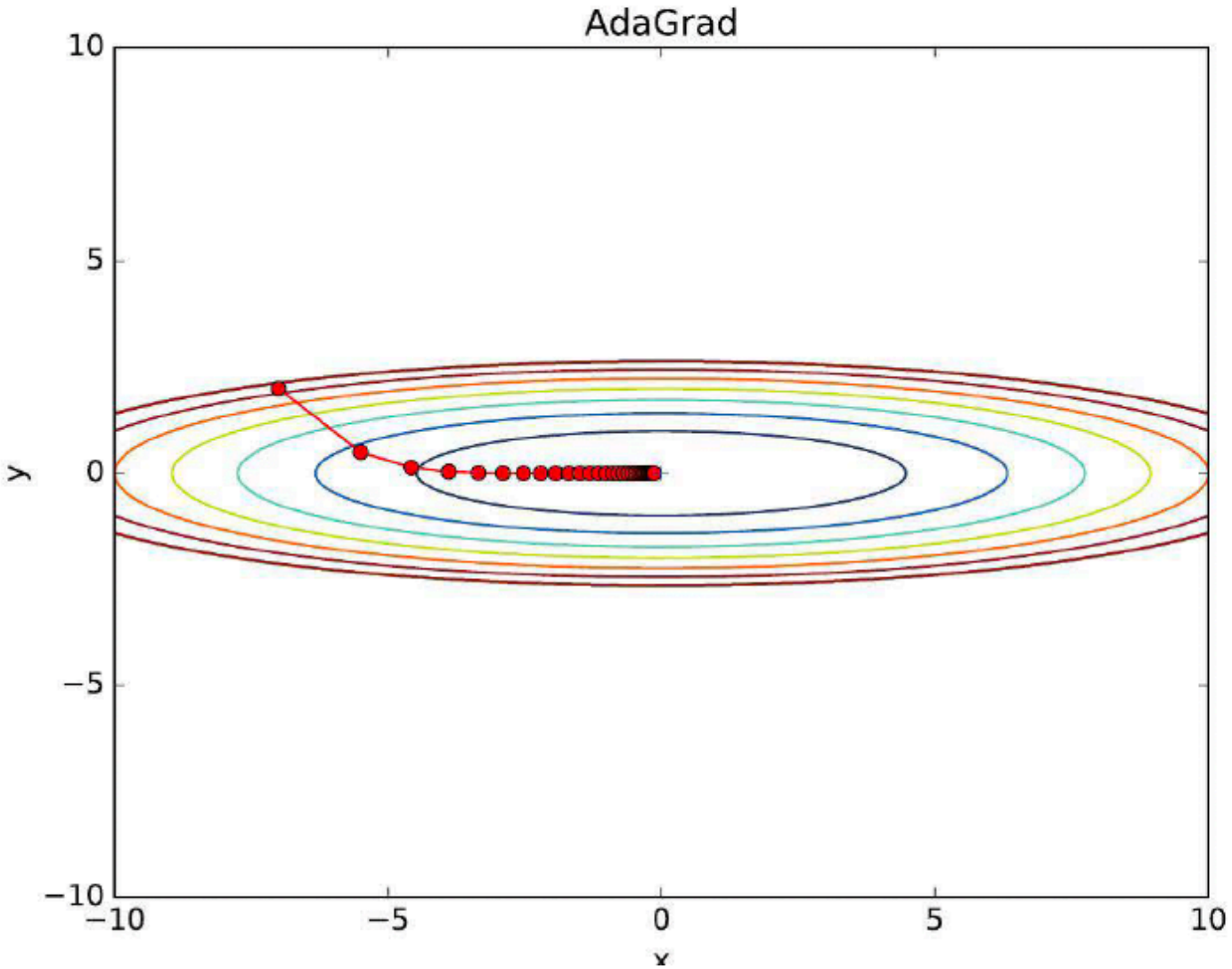
미분한 값, 즉 기울기를 나타냄 - η: 학습률(learning rate)을 의미 - ϵ: 0으로 나누는 것을 방지하기 위해 추가하는 작은 상수입니다. - 매개변수를 업데이트할 때, 누적된 기울기 제곱합의 제곱근으로 학습률을 나누어 조절 - 즉, 기울기가 큰 매개변수는 학습률을 작게, 기울기가 작은 매개변수는 학습률을 크게 적용

- 희소(sparse)한 데이터에 적합하지만, 학습률이 너무 빨리 감소하여 **학습이 멈출 수 있음**
- tensorflow에서는 learning_rate의 기본값을 0.001로 설정되어 있음
- 그레디언트 제곱을 누적하여 학습률을 나누고, initial_accumulator_value 매개변수로 누적 초깃값을 지정할 수 있음(기본값 0.1)

```
optimizer = tf.keras.optimizers.Adagrad(learning_rate=0.01)

# adagrad
```

그림 6-6 AdaGrad에 의한 최적화 갱신 경로



RMSprop

- Adagrad의 단점을 개선하기 위해 지수 가중 이동 평균(exponential moving average)을 사용하여 학습률을 조절

- Adagrad보다 안정적인 학습률 조절이 가능하지만, 학습률 설정이 여전히 중요

$$G = \gamma G + (1 - \gamma)(\nabla_\theta J(\theta_t))^2$$

$$\theta_{t+1} \leftarrow \theta_t - \frac{\eta}{\sqrt{G + \epsilon}} \nabla_\theta J(\theta_t)$$

- tensorflow에서 learning_rate의 기본값은 0.001
- Adagrad처럼 그레디언트 제곱으로 학습률을 나누지만 최근의 그레디언트를 사용하기 위해 지수 감소를 사용
  - rho 매개변수에서 감소 비율을 지정하며 기본값은 0.9

```
optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.001)
```

Adadelta

- Adagrad와 유사하지만, 학습률을 수동으로 설정할 필요가 없음
- 희소한 데이터에 적합하지만, 학습률이 너무 빨리 감소하여 학습이 멈출 수 있음

```
optimizer = tf.keras.optimizers.Adadelta()

# adadelta
```

Nadam

- Adam에 Nesterov Accelerated Gradient (NAG)를 적용한 옵티마이저
- Adam보다 약간 더 빠른 수렴 속도를 보일 수 있습니다.

```
optimizer = tf.keras.optimizers.Nadam(learning_rate=0.001)

# nadam
```

Adam

- Momentum과 RMSprop의 장점을 결합한 옵티마이저
- 모멘텀을 사용하여 이전 업데이트 방향을 고려하고, RMSprop처럼 적응적으로 학습률을 조절합니다.
- 다양한 문제에서 좋은 성능을 보이며, 하이퍼파라미터 튜닝에 대한 민감도가 낮음
- tensorlfow에서 기본값은 0.1
- 모멘텀 최적화에 있는 그레이디언트의 지수 감소 평균을 조절하기 위해 beta_1 매개변수가 있으며 기본값은 0.9
- RMSprop에 있는 그레이디언트 제곱의 지수 감소 평균을 조절하기 위해 beta_2 매개변수가 있으면 기본값은 0.999

```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

# 문자열로 넘길수도 있음
model.compile(loss='mean_squared_error', optimizer='adam')
```

그림 6-7 Adam에 의한 최적화 갱신 경로