STEP33. 뉴턴 방법으로 푸는 최적화 (자동 계산)

2차 미분 자동 계산 수행 (새로운 DeZero 사용)
- 간단한 수식의 2차 미분 계산 수행
- 뉴턴 방법을 사용하여 최적화 수행

2차 미분 계산하기

1. $y = x^4 + 2x^2$ 수식 2차 미분 계산

```
import numpy as np
from dezero import Variable

# y = x⁴ + 2x²
def f(x):
    y = x ** 4 - 2 * x ** 2
    return y

x = Variable(np.array(2.0))
y = f(x)
y.backward(create_graph=True)
print(x.grad)

gx = x.grad
gx.backward()
print(x.grad) # 44가 나와야함
```

```
import numpy as np
from dezero import Variable

# y = x⁴ + 2x²
def f(x):
    y = x ** 4 - 2 * x ** 2
    return y

x = Variable(np.array(2.0))
y = f(x)
y.backward(create_graph=True)
print(x.grad)

gx = x.grad
x.cleargrad()
gx.backward()
print(x.grad)
```

- y = backward(create_graph=True) 의해 첫번째 역전파 진행하고, 계산 그래프 생성
- gx = x.grad 로 y의 x에 대한 미분값을 꺼냄
- gx.backward 꺼낸 미분값인 gx에 한 번더 역전파 진행. 이 두번째 미분이 바로 2차 미분임
  - 확인 결과 1차 미분값 24, 2차 미분값 68. (2차 미분값은 44가 되어야 함)
- 잘못된 결과는 1차 미분결과(24)에 2차 미분결과(44)가 더해진 값임

뉴턴 방법을 활용한 최적화

뉴턴 방법을 활용한 최적화

- 뉴턴 방법을 활용한 최적화 수식

$$x \leftarrow x - \frac{f'(x)}{f''(x)}$$

```
# y = x⁴ + 2x²
def f(x):
    y = x ** 4 - 2 * x ** 2
    return y

x = Variable(np.array(2.0))
iters = 10

for i in range(iters):
    print(i, x)
    y = f(x)
    x.cleargrad()
    y.backward(create_graph=True)

    gx = x.grad
    x.cleargrad()
    gx.backward()
    gx2 = x.grad

    x.data -= gx.data / gx2.data
```

```
0 variable(2.0)
1 variable(1.4545454545454546)
2 variable(1.1510467893775467)
3 variable(1.0253259289766978)
4 variable(1.0009084519430513)
5 variable(1.0000012353089454)
6 variable(1.000000000002289)
7 variable(1.0)
8 variable(1.0)
9 variable(1.0)
```

- f(x)의 1차 미분과 2차 미분을 사용하여 x를 갱신함
- backward 메서드를 두 번 실행하여 자동으로 계산하게 수정함
- 7회만에 최솟값 1에 도달함