STEP18. 메모리 절약 모드

DeZero의 메모리 사용을 개선할 수 있는 구조 두 가지 도입

1. 역전파 시 사용하는 메모리 양을 줄이는 방법
   - 불필요한 미분 결과를 보관하지 않고 즉시 삭제함

2. ' 역전파가 필요 없는 경우용 모드 ' 를 제공
   - 이 모드에서는 불필요한 계산을 생략함

필요 없는 미분값 삭제

DeZero의 역전파 개선

- 현재의 DeZero에서는 `y.backward()` 를 실행하면 모든 변수가 미분값을 메모리에 유지함
- 머신러닝에서 역전파로 구하고 싶은 미분값은 ' 말단변수 ' 뿐일 때가 대부분임
  (중간 변수의 미분값은 필요하지 X)

1. 중간 변수에 대해서는 미분값을 제거하는 모드 추가

```
class Variable:

  ```

  """ retain_grad=False : 중간 변수의 미분값을모두 None으로 재설정 """
    def backward(self, retain_grad=False):
        # y.grad = np.array(1.0) 생략을 위한 if문
        if self.grad is None:
            self.grad = np.ones_like(self.data)

        ```

        """ 각 함수의 출력 변수의 미분값을 유지하지 않도록 설정 """
        if not retain_grad:
            for y in f.outputs:
                """ y는 약한 참조 ( weakref ) """
                y().grad = None
```

Function 클래스 복습

```
class Function:
    # *ㅁㅁㅁ : 임의 개수의 인수 ( 가변길이 ) 를 건내 함수를 호출할 수 있음
    def __call__(self, *inputs):
        # 리스트 xs를 생성할 때, 리스트 내포 사용
        # 리스트의 각 원소 x에 대해 각각 데이터 ( x.data ) 를 꺼냄
        xs = [x.data for x in inputs]

        # forward 메서드에서 구체적인 계산을 함
        ys = self.forward(*xs) # 리스트 언팩 ( 원소를 낱개로 풀어서 전달 )

        if not isinstance(ys, tuple): # 튜플이 아닌 경우 추가 지원
            ys = (ys, )

        # ys의 각 원소에 대해 Variable 인스턴스 생성, outputs 리스트에 저장
        outputs = [Variable(as_array(y)) for y in ys]

        self.generation = max([x.generation for x in inputs])

        # 각 output Variable 인스턴스의 creator를 현재 Function 객체로 설정
        for output in outputs:
            output.set_creator(self)

        """ 순전파 결과값 기억 """
        self.inputs = inputs

        self.outputs = [weakref.ref(output) for output in outputs]

        # 리스트의 원소가 하나라면 첫 번째 원소를 반환함
        return outputs if len(outputs) > 1 else outputs[0]
```

- 미분을 하려면 순전파를 수행 → 역전파를 수행
- 역전파 시에는 순전파의 계산 결과가 필요하기 때문에 순전파 때 결과값 기억해 둠
- 함수는 입력을 inputs라는 인스턴스 변수로 참조함
- inputs가 참조하는 변수의 참조 카운트가 1만큼 증가함
- __call 메서드에서 벗어난 뒤에도 inputs가 메모리에 생존함
- 인스턴스 변수 inputs는 역전파 계산 시 사용됨

💡 학습 (training)
   - 학습 시에는 미분값을 구해야 함

추론 (inference)
  - 추론 시에는 단순히 순전파만 하기 때문에 중간 계산 결과를 곧바로 버리면 메모리 사용량을 크게 줄 일 수 있음

Config 클래스를 활용한 모드 전환

순저파만 수행할 경우를 위한 모드 전환 구조

- 학습 시에는 역전파를 수행하지만, 추론 시에는 순전파만 수행함
- '역전파 활성 모드'와 '역전파 비활성 모드'를 전환하는 구조 필요

Config 클래스

```
""" 역전파가 가능한지 여부 """
class Config:
    """ True : 역전파 활성 모드 """
    enable_backprop = True
```

- 설정 데이터는 단 한 군데에서만 존재하는 것이 좋음
- 인스턴스화 하지 않고 클래스 상태로 이용
- 역전파가 가능한지 여부를 뜻하는 `enable_backprop` 속성을 가짐

```
import weakref

# Variable 인스턴스를 변수로 다룰 수 있는 함수를 Function클래스로 구현
class Function:
    # *ㅁㅁㅁ : 임의 개수의 인수 ( 가변길이 ) 를 건내 함수를 호출할 수 있음
    def __call__(self, *inputs):
        # 리스트 xs를 생성할 때, 리스트 내포 사용
        # 리스트의 각 원소 x에 대해 각각 데이터 ( x.data ) 를 꺼냄
        xs = [x.data for x in inputs]

        # forward 메서드에서 구체적인 계산을 함
        ys = self.forward(*xs) # 리스트 언팩 ( 원소를 낱개로 풀어서 전달 )

        if not isinstance(ys, tuple): # 튜플이 아닌 경우 추가 지원
            ys = (ys, )

        # ys의 각 원소에 대해 Variable 인스턴스 생성, outputs 리스트에 저장
        outputs = [Variable(as_array(y)) for y in ys]

        """ enable_backprop = True 일 때만 역전파 코드 실행 """
        if Config.enable_backprop:
            """ 역전파 시 노드에 따라 순서를 정하는데 사용 """
            self.generation = max([x.generation for x in inputs]) # 세대 설정

            # 각 output Variable 인스턴스의 creator를 현재 Function 객체로 설정
            for output in outputs:
                """ 계산들의 연결을 만듬 """
                output.set_creator(self) # 연결 설정

            self.inputs = inputs # 순전파 결과값 기억
            self.outputs = [weakref.ref(output) for output in outputs]

        # 리스트의 원소가 하나라면 첫 번째 원소를 반환함
        return outputs if len(outputs) > 1 else outputs[0]
```

모드 전환

square 함수를 세 번 적용 수행

```
Config.enable_backprop = True
x = Variable(np.ones((100, 100, 100)))
y = square(square(square(x)))
y.backward()

Config.enable_backprop = False
x = Variable(np.ones((100, 100, 100)))
y = square(square(square(x)))
```

- Config.enable_backprop = True면 중간 계산 결과가 계속 유지되어 메모리 차지함
- Config.enable_backprop = False면 중간 계산 결과는 곧바로 삭제됨

with 문을 활용한 모드 전환

```
f = open('sample.txt', 'w')
f.write('hello world!')
f.close()

# close를 깜빡할 수도 있음. 이런 실수를 막기위해 with를 씀
with open('sample.txt', 'w') as f:
    f.write('hello world!')
```

```
f.write('hello world!')
```

- with 블록에 들어갈 때의 처리 (전처리) 와 with 블록을 빠져나올 때의 처리 (후처리) 를 자동으로 수행
- with 문의 원리를 이용하여 '역전파 비활성 모드'로 전환하려 함
  (with 블록 안에서만 '역전파 비활성 모드', with 블록을 벗어나면 일반모드)

```
with using_config('enable_backprop', False):
  x = Variable(np.array(2.0))
  y = square(x)
```

- contextlib 모듈을 사용하여 구현
- `@contextlib.contextmanager` 데코레이더를 달면 문맥을 판단하는 함수가 만들어짐

using_config 함수 구현

```
import contextlib

@contextlib.contextmanager
def using_config(name, value):
    # 설정값을 변경하기 전에 이전 설정값을 임시로 저장
    old_value = getattr(Config, name)
    # 지정된 속성에 새로운 값 설정
    setattr(Config, name, value)
    try:
        # 설정값을 변경한 후에 코드 블록을 실행
        yield
    finally:
        # 코드 블록 실행 후에 이전 설정값을 복원
        setattr(Config, name, old_value)
```

- name은 타입이 str이며, 사용할 Config 속성의 이름 (클래스 속성) 을 가르킴
- name을 getattr 함수에서 Config 클래스에서 꺼내고, setattr함수로 새로운 값을 설정함
- with 블록에 들어갈 때 name으로 지정한 Config 클래스 속성이 value로 설정
- with 블록을 빠져나오면서 원래 값 (old_value) 로 복원

역전파에 적용

```
def no_grad():
    return using_config('enable_backprop', False)
```

```
""" with using_config('enable_backprop', False)적기 귀찮으니 no_grad사용

  with using_config('enable_backprop', False):
  x = Variable(np.array(2.0))
  y = square(x) """

with no_grad():
    x = Variable(np.array(2.0))
    y = square(x)
```

- 역전파가 필요 없는 경우에는 with 블록에서 순전파만 실행
- 편의를 위해 no_grad 함수 준비
- 기울기 계산이 필요 없을 때는 no_grad 함수를 호출하면 됨