STEP26. 계산 그래프 시각화 (2)

- DeZero 계산 그래프를 DOT 언어로 변환
- DeZero에서 실행한 계산을 DOT 언어로 변환하는 기능 구현

시각화 코드 예

1. 계산 그래프의 노드를 DOT 언어로 변환

```
import numpy as np
from dezero import Variable
from dezero.utils import get_dot_graph

x0 = Variable(np.array(1.0))
x1 = Variable(np.array(1.0))
y = x0 + x1 # 어떤 계산

# 변수 이름 지정
x0.name = 'x0'
x1.name = 'x1'
y.name = 'y'

txt = get_dot_graph(y, verbose=False)
print(txt)

with open('sample.dot', 'w') as o:
    o.write(txt)
```

- dezero/utils.py 에 get_dot_graph 함수 구현
- 출력 변수 y를 기점으로 한 계산 과정을 DOT 언어로 전환한 문자열 반환
- 변수 노드에 레이블 달기 - Variable 인스턴스 속성에 name을 추가

계산 그래프에서 DOT 언어로 변환하기

2. _dot_var 함수

```
def __dot__var(v, verbose=False):
    # 노드의 속성을 정의하는 문자열 템플릿
    dot_var = '{} [label="{}", color=orange, style=filled]\n'

    # 변수의 이름과 데이터 타입을 가져옴
    name = '' if v.name is None else v.name  # 변수의 이름이 없으면 빈 문자열을 사용
    # verbose 모드가 활성화되어 있고 변수의 데이터가 있으면
    if verbose and v.data is not None:
        # 변수의 이름과 변수의 데이터 형태와 데이터 타입을 문자열에 추가
        if v.name is not None:
            name += ': '  # 변수의 이름이 있으면 콜론과 공백을 추가
        name += str(v.shape) + ' ' + str(v.dtype)  # 변수의 형태와 데이터 타입을 추가
    # 노드의 속성 문자열을 포맷하여 반환
    return dot_var.format(id(v), name)
```

- get_dot_graph 함수 전용으로 로컬에서만 사용함
- Variable 인스턴스를 건네면 인스턴스 내용을 DOT 언어로 작성된 문자열로 바꿔서 변환
- 변수 노드에 고유한 ID를 부여하기 위해 파이썬 내장 함수인 id를 사용
- id 함수에서 반환하는 객체 ID는 다른 객체와 중복되지 않아서 노드의 ID로 사용하기 적합
- format 메서드 문자열의 "{}" 부분을 인수로 건넨 객체로 차례로 바꿔줌

3. _dot_funce 함수

```
def get_dot_graph(output, verbose=True):
    txt = ''
    funcs = []
    seen_set = set()

    def add_func(f):
        if f not in seen_set:
            funcs.append(f)
            # funcs.sort(key=lambda x: x.generation)

    add_func(output, creator)
    txt += _dot_var(output, verbose)

    while
```

- get_dot_graph 함수 전용으로 로컬에서만 사용함
- DeZero 함수를 DOT 언어로 기술
- 함수의 입력 변수의 관계, 함수와 출력 변수의 관계도 DOT 언어로 기술함

- 함수와 입력 변수의 관계, 함수와 출력 변수의 관계도 DOT 언어로 기술함
- DeZero 함수는 Function 클래스를 상속하고, inputs와 outputs라는 인스턴스 변수를 가짐

4. get_dot_graph 함수

```
def get_dot_graph(output, verbose=True):
    txt = ''
    funcs = []
    seen_set = set()

    def add_func(f):
        if f not in seen_set:
            funcs.append(f)
            # funcs.sort(key=lambda x: x.generation)

    add_func(output, creator)
    txt += _dot_var(output, verbose)

    while funcs:
        func = funcs.pop()
        txt += _dot_func(func)
        for x in func.inputs:
            txt += _dot_var(x, verbose)

            if x.creator is not None:
                add_func(x.creator)

    return 'digraph g {\n' + txt + '}'
```

- Variable 클래스의 backward 메서드와 거의 같음
- backward 메서드는 미분값을 전파 -> 미분 대신 DOT 언어로 기술한 문자열 txt에 추가
- 역전파는 노드를 따라가는 순서가 중요하여 함수에 generation 정수값 부여
- 노드를 추적하는 순서는 필요없어 generation 값으로 정렬하는 코드는 주석 처리

이미지 변환까지 한번에

dot 명령 실행까지 한 번에 해주는 함수

- get_dot_graph 함수는 계산 그래프를 DOT 언어로 변환
- DOT 언어를 이미지로 변환하려면 dot 명령을 수동으로 실행
- dot 명령 실행까지 한 번에 해주는 함수를 제공

코드 설명

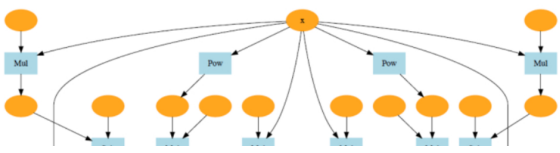
- 계산 그래프를 DOT 언어(텍스트)로 변환하고 파일에 저장
  (대상 디렉토리는 ~/.dezero 이고 파일 이름은 tmp_graph.dot, '~'은 홈 디렉터리 뜻)
- 저장한 파일 이름을 지정하여 dot 명령을 호출
- To_file에 저장할 이미지 파일의 이름을 지정
- 파이썬에서 외부 프로그램을 호출하기 위해 subprocess.run 함수를 사용
- from dezero.utils import plot_dot_graph 로 임포트하여 사용

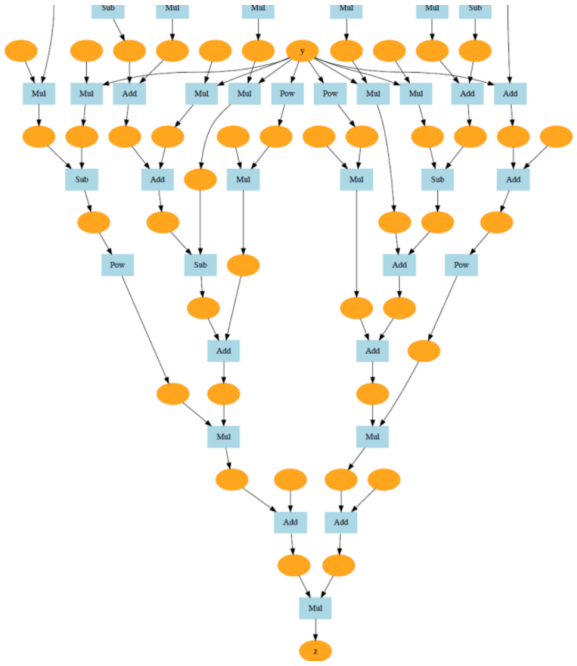
동작 확인

5. Goldstein-Price 함수 시각화

```
import numpy as np
from dezero import Variable
from dezero.utils import plot_dot_graph

def goldstein(x, y):
    z = (1 + (x + y + 1)**2 * (19 - 14*x + 3*x**2 - 14*y + 6*x*y + 3*y**2)) * \
        (30 + (2*x - 3*y)**2 * (18 - 32*x + 12*x**2 + 48*y - 36*x*y + 27*y**2))
    return z

x = Variable(np.array(1.0))
y = Variable(np.array(1.0))
z = goldstein(x, y)
z.backward()
# print(x.grad, y.grad)

x.name = 'x'
y.name = 'y'
z.name = 'z'
plot_dot_graph(z, verbose=False, to_file='goldstein.png')
```

- 이 코드를 실행하면 goldstein.png 파일 생성
- 변수 x 와 y 에서 시작하여 최종적으로 변수 z가 출력됨




Sub
Mul
Mul
Mul
Mul
Sub
y
Mul
Mul
Add
Mul
Mul
Pow
Pow
Mul
Mul
Add
Add
Sub
Add
Mul
Mul
Sub
Add
Pow
Sub
Add
Pow
Add
Add
Mul
Mul
Add
Add
Mul
z