파이썬 딥러닝을 활용해 함수의 값, 미분계수, 정적분 값 예측하기

20105 설규원

1) 연구 목적

우리가 임의의 함수를 정하고,
딥러닝 인공지능에게 주어진 몇 개의 x좌표에 대한 함숫값 데이터만을 훈련 시켰을 때,
인공지능은 함수를 예측해낼 수 있을까?
함수를 예측해냈다면, 그것에서 얻어낸 미분계수나 정적분 값 또한 같게 나올까?

2) 연구 동기 및 용어의 정의

지난 주제탐구 시간에 '회귀 알고리즘과 최소제곱법' 에 대해서 탐구했었다.
이론만 탐구했었기 때문에, 직접 함수를 예측하는 회귀 모델을 파이썬 프로그래밍 언어로 만들어보며 딥러닝에 대해 탐구하고 싶어 주제를 정하였다.
*회귀: 연속된 데이터를 기반으로 다른 데이터를 예측하는 인공지능.
*최소제곱법: 오차의 제곱의 부분합이 최소가 되는 식을 찾아 그것을 모델로 쓰는 것.
*경사하강법: 최소제곱법에서, 오차의 제곱의 부분합의 최솟값을 찾기 위해 미분계수의 절댓값이 작아지는 방향으로 이동하며 최솟값을 찾는 수학적 기법.
*인공신경망: 인간 뇌의 신경망 구조를 모방하여 만든 머신러닝 기법. 입력층(신호를 받음), 은닉층(신호 간의 복잡한 연산), 출력층(결과를 출력함)으로 이루어져 있다.
*딥러닝: 은닉층이 아주 많고 복잡한 인공신경망을 만드는 머신러닝 기법.

3) 연구내용

아래 코드들은 파이썬 프로그래밍 언어로 직접 프로그래밍한 것이다.
따라서 내 개인 깃허브 레포지토리에 업로드되어 있다 (https://github.com/seolcu/polynomial_regression)
① 필요한 라이브러리 불러오기

```
"""
2021 2학년 2학기
20105 설규원 수학2 주제탐구
주제: 딥러닝을 활용해 함수의 값, 미분계수, 정적분 값 예측하기!
사용 라이브러리: Tensorflow(딥러닝), numpy(어레이 데이터), matplotlib(데이터 시각화)
"""

import matplotlib.pyplot as plt
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import optimizers
```

연구에 필요한 라이브러리들을 import 해준다.
우선 tensorflow와 tensorflow.keras는 딥러닝 신경망 모델을 만드는 데 사용되는 라이브러리다.
그리고 numpy는 데이터 어레이를 만들 때 사용된다.
마지막으로 matplotlib는 데이터 시각화 라이브러리로 그래프를 그릴 때 사용된다.

② 함수식 정의하기(인공지능에게 전달되지 않음)

```python
def f(x):
    # 함수식 설정하기 (인공지능에게는 일부 함숫값만이 주어짐)
    return 8 * (x ** 3) + 6 * (x ** 2) + 4 * (x ** 1) + 10

def der_f(x):
    # f(x)를 미분한 식 (다르면 안됨!)
    return 24 * (x ** 2) + 12 * (x ** 1) + 4

def int_f(x):
    # f(x)를 적분한 식 (정적분에 사용되므로 적분상수 C 무시하기!)
    return 2 * (x ** 4) + 2 * (x ** 3) + 2 * (x ** 2) + 10 * (x ** 1)
```

예측할 함수를 정의해준다.

그리고 이 함수는 인공지능에게 주어지지 않는다.

f(x) = 8x^3 + 6x*2 + 4x + 10 으로 3차식이다.

이때 나중에 예측한 값이랑 비교하기 위해 f(x)를 미분한 함수인 der_f(x) 또한 정의해야 하기에

def_f(x) = 24x^2 + 12x + 4 가 된다.

또한 나중에 인공지능이 정적분까지 예측하기 때문에, f(x)를 부정적분한 int_f(x) 또한 정의해야 한다.

어차피 정적분에 사용할 것이므로 적분상수 C는 무시하고 나머지만 적분하면 된다.

int_f(x) = 2x^4 + 2x^3 + 2x^2 + 10x 가 된다.

③ 학습 데이터 생성

```python
# 빈 데이터 어레이 생성
x_data = []
y_data = []
for i in range(-100, 101, 2):
    # 반복문으로 -100부터 100까지 짝수인 x값에 대한 데이터 생성
    x_data.append(i)
    y_data.append(f(i))
x_data = np.array(x_data)
y_data = np.array(y_data)
```

인공지능에게 주어질 학습 데이터를 만든다.

어떤 x좌표에 대한 함숫값이 신경망에게 주어지는 것이다.

이때 x_data와 y_data를 나누는 포맷을 사용했다.

반복문을 통해 x값이 -100 ~ 100까지의 짝수값만이 전달되기 때문에

x_data = [-100, -98, -96, ... , 96, 98, 100]

y_data = [-7970390, -7501106, -7050614, ... , 7105930, 7558750, 8030410]

으로 나오게 된다.

데이터 생성이 끝난 후 사용되는 np.array 구문은, 기존 파이썬 어레이를 더 효율적인 numpy 어레이로 변환하는 함수이다. 이로써 데이터 정제가 끝났다.

④ 신경망 구조 설계하기

```
39  # 모델 생성, 은닉층 설정
40  model = keras.Sequential()
41  model.add(layers.Dense(30, input_dim=1, activation="relu"))
42  model.add(layers.Dense(30, activation="relu"))
43  model.add(layers.Dense(30, activation="relu"))
44  model.add(layers.Dense(1))
```

keras에서 Sequential 메서드를 불러와 model에 저장시켜준다.
이때 model은 신경망의 틀이 된다.
그리고 model에 여러개의 층을 추가시켜준다.
우선 짚고 넘어갈 개념은 입력층, 은닉층, 출력층이다.
입력층이란 정보를 받아들이는 뉴런의 층이고,
은닉층이란 정보를 처리하는 뉴런의 층이고,
출력층이란 정보를 출력하는 뉴런의 층이다.
이 모델을 보자.
모델을 생성할 때, 입력층은 데이터에 따라 자동으로 하나가 생성되게 된다.
우리의 데이터는 정의역의 변수가 1개이므로, 입력층의 뉴런도 1개가 자동 생성된다.
그러면 우리는 은닉층을 만들어야 한다.
은닉층의 뉴런이 많을수록 학습하는데 많은 데이터가 필요하고, 적을수록 모델이 단순하게 나타난다.
여러 차례 시도해보았고, 나는 한 층마다 30개의 뉴런씩, 3층을 만들었을 때 적절히 모델이 생성되었다.
그리고 이제 출력층을 만들어야 하는데,
함수 f(x)는 치역의 변수가 1개이므로 출력층의 뉴런도 1개가 되어야 한다.
따라서 제일 끝에 뉴런이 1개인 층을 하나 추가해주었다.

⑤ 신경망 컴파일 후 학습시키기

```
47  adam = optimizers.Adam(learning_rate=0.01)
48  model.compile(loss="mse", optimizer=adam, metrics=["accuracy"])
49
50  # 모델 학습시키고 history 에 저장
51  history = model.fit(x_data, y_data, epochs=500, batch_size=1, shuffle=False, verbose=1)
```

우선 optimizer가 필요하다.
optimizer란 신경망의 정확도를 올리기 위한 수학적 알고리즘인데,
저번에 탐구했던 경사하강법 또한 optimizer에 속한다.
이번엔 adam이라는 optimizer를 사용했다.
이후 compile 메서드로 모델을 구조대로 컴파일한다. 모양을 잡는다고 생각하면 좋다.
그리고 fit 메서드로 앞서 만든 학습 데이터들을 학습시킨다.
이때 epoch는 학습을 몇 번 반복할지 설정하는 것이다. 500번 학습시키기로 했다.
그리고 batch_size는 데이터를 한번에 몇 개씩 학습할지 정하는 것이다.
데이터의 양이 많지 않기에, 하나하나씩 학습하도록 batch_size를 1로 주었다.
그리고 학습 과정에 대한 기록은 변수 history에 저장했다.

⑥ 함수 예측 해보기

```
# 빈 테스트 데이터 어레이 생성
predict_x_data = []
predict_y_data = []
for i in range(-99, 101, 2):
    # 반복문으로 -99부터 99까지 홀수인 x값에 대한 데이터 예측
    predict_x_data.append(i)
    # float: 실수화
    predict_y_data.append(float(model.predict([i])))
predict_x_data = np.array(predict_x_data)
predict_y_data = np.array(predict_y_data)
```

-99 ~ 99까지의 홀수 x값에 대한 함숫값을 예측해보았다.

이때 값이 float형태가 아닌 다른 값으로 나타나 결과값에 float를 적용해야 했다.

학습 데이터를 만들 때와 동일하게 numpy 배열로 만들어주었다.

⑦ 예측한 함수의 도함수 정의

```
# 예측한 모델의 도함수
def predict_der_f(x):
    # 0에 가까운 값
    delta_x = 1e-4
    # 도함수의 정의 사용 ( lim:h→0 에서 (f(x+h)-f(x))/h )
    return float(model.predict([x + delta_x]) - model.predict([x]))/delta_x
```

도함수를 구하기 위해서는 극한이 필요한데, 파이썬에서 극한을 구현하기 위해서는 추가적인 라이브러리들이 더 필요하여 복잡해진다.

따라서 0에 아주 가까운 1e-4값을 x의 변화량(h)로 두었다.

그리고 도함수의 정의를 활용하여 식을 만들어냈다.

⑧ 도함수 예측 해보기

```
# 도함수 비교를 위한 빈 배열 생성
der_x_data = []
der_y_data = []
der_predict_y_data = []
for i in range(-100, 101, 1):
    der_x_data.append(i)
    der_y_data.append(der_f(i))
    der_predict_y_data.append(float(predict_der_f(i)))
der_x_data = np.array(der_x_data)
der_y_data = np.array(der_y_data)
der_predict_y_data = np.array(der_predict_y_data)
```

6번 단계와 같은 방식으로, -100 ~ 100까지의 x값에 대한 미분계수를 예측해준다.

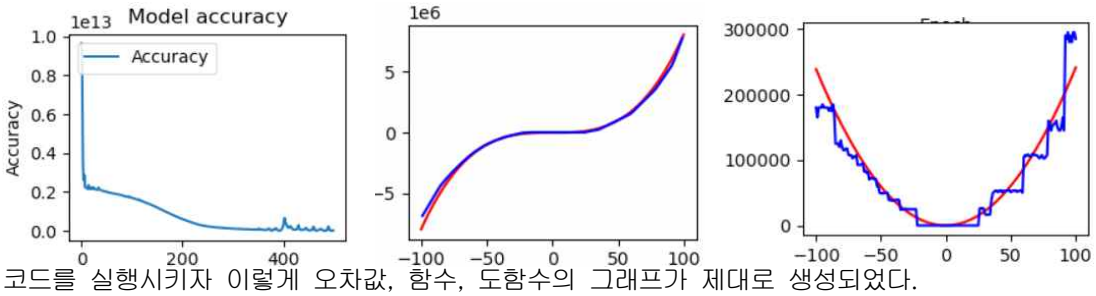
그리고 이것을 각각의 numpy 배열에 담았다.

⑨ 시각화하기

```
87    # epoch에 대한 오차값 그래프
88    plt.subplot(221)
89    plt.plot(history.history["loss"])
90    plt.title("Model accuracy")
91    plt.xlabel("Epoch")
92    plt.ylabel("Accuracy")
93    plt.legend(["Accuracy", "Loss"], loc="upper left")
94
95    # 함수 그래프
96    plt.subplot(222)
97    plt.plot(x_data, y_data, color="red")
98    plt.plot(predict_x_data, predict_y_data, color="blue")
99
100   # 도함수 그래프
101   plt.subplot(223)
102   plt.plot(der_x_data, der_y_data, color="red")
103   plt.plot(der_x_data, der_predict_y_data, color="blue")
104   plt.savefig("results.png")
```

matplotlib를 활용해 그래프를 그린다.
첫 번째는 epoch(학습을 한바퀴 도는 것) 횟수에 따른 오차값을 나타내는 그래프를 그린다.
두 번째는 실제 함수를 빨간색으로, 예측한 함수를 파란색으로 나타내는 그래프를 그린다.
세 번째는 실제 도함수를 빨간색으로, 예측한 도함수를 파란색으로 나타내는 그래프를 그린다.

⑩ 시각화 결과 보기



코드를 실행시키자 이렇게 오차값, 함수, 도함수의 그래프가 제대로 생성되었다.
오차값도 거의 0에 수렴하며, 함수 그래프의 경우 거의 완전 일치하는 것을 볼 수 있다.
그러나 도함수의 경우, 예측한 도함수의 모습이 부드럽지 않고 끊겨있는 것을 볼 수 있는데,
이는 도함수의 정의에서 실제로 극한을 사용하지 않았기 때문이라고 생각된다.
실제로 극한을 사용한다면 도함수까지 완벽하게 구해졌을 것이다.

4) 결론 및 느낀 점

파이썬 딥러닝을 이용해 성공적으로 함수와 도함수를 예측하는 인공지능 신경망 모델을 만들었다.
아쉽게도 예측한 함수를 적분하는 방법을 찾지 못해, 정적분값은 비교하지 못했다.
다음 탐구를 진행할 때는 구분구적법에 대해 탐구하여 딥러닝을 통해 정적분의 값 또한 예측해 보고 싶다.

5) 참고문헌

구글 텐서플로우 라이브러리 공식문서 (https://www.tensorflow.org/guide?hl=ko)