교과목 내용

대략적인 강의 내용은 다음 세 가지 주제로 채워질 수 있겠는데요 교수님들께서 검토해 보시고 적당히 수정하셔서 진행하시면 될 듯합니다. 2) 데이터 설명 참고하셔서 기계학습에 적용할 수 있는 형태의 데이터는 뭐든 괜찮겠습니다. 1), 2) 강의해 주시면 제가 2) 다시 요약하고 3) 진행할 수 있을 듯합니다.

1) 생물화학 응용 분야 (개론)
- 단백질 구조와 활성
- 단백질 서열과 발현량
- 생물반응기

2) 데이터 설명
- 단백질 구조와 활성 - Y: 단백질 활성, X: 단백질 구조 (이미지, 좌표값)
- 단백질 서열과 발현량 - Y: 단백질 발현양, X: 단백질 서열
- 생물반응기 - Y:단백질 생산량, X: 반응기 운영 조건

3) 기계학습 실습

(1) 기계학습 이론 강의

최근 딥러닝 기반의 AI 기술은 급속도로 발전하여 이제는 중고등학교 학생들도 python이나 R 등의 프로그래밍 언어를 공부하면 쉽게 AI를 구현할 수 있게 되었다. 이제 관련 도메인 지식과 해당 분야의 빅데이터를 확보하는 것이 성공적인 AI 활용을 위해서 더욱 중요해지는 추세이다. 생물공학을 포함한 생물학 분야에서는 서열 데이터가 대표적인 빅데이터이며 본 강의에서는 이러한 서열데이터를 이용해서 딥러닝을 수행하기 위한 기본적인 기술을 소개하고자 한다. 프로그래밍 언어는 Python을 이용한다. Python은 전 세계적으로 가장 많이 사용되는 언어로서 만약 사용 경험이 없다 해도 다양한 문헌이나 인터넷 사이트를 통해 어렵지 않게 익힐 수 있다. 특히 자료구조인 List, Tuple, Dictionary 그리고 Numpy의 ndarray 개념을 명확히 이해하는 것은 서열 데이터를 다루는 데 필수이다.

컴퓨팅 기술이 발달하면서 과학자들은 특정 문제에 관한 판단을 수행하는 기계를 만들었다. 여기서 말하는 특정 문제는 크게 두 가지로 나눌 수 있다. 하나는 “분류” 문제와 다른 하나는 “회귀” 문제이다. 이러한 문제를 해결하기 위한 기계를 만들기 위해서는 이들을 충분히 학습할 수 있는 데이터가 필요하며 이러한 데이터를 이용해서 기계를 학습하는 것을 지도학습(supervised learning) 이라고 한다. 참고로 지도학습과 함께 비지도학습(unsupervised learning) 분야도 있지만 본 강의에서는 지도학습에 관련된 내용을 주로 다루도록 한다. AI와 기계학습 딥러닝의 관계는 다음과 같은 그림으로 나타낼 수 있다[1].

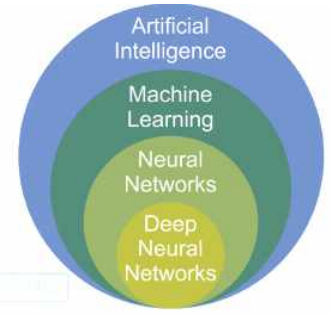

그림 1 AI, 기계학습, 딥러닝 관계 [1]

1) Harrison Kinsley & Daniel Kukiela, Neural Networks from Scratch in Python

따라서 딥러닝을 이해하기 위해서는 신경망을 먼저 이해할 필요가 있다. 신경망 개념은 1940년대에 나왔고 현재 딥러닝에서 핵심 알고리즘으로 사용되는 Backpropagation은 1960년대에 나온 이론이다. 당시에는 전혀 주목받지 못했으나 2010년대 하드웨어의 계산능력이 향상되고 이미지나 언어 번역 등의 다양한 문제를 해결할 수 있음을 보이면서 이제는 대부분 분야에서 빠지지 않고 도입할 핵심 기술이 되었다.

신경망에서의 인공뉴론은 뇌 안의 뉴론에 의해서 착안하였다고 할 수 있다. 작동되는 프로세스는 다를 수 있으나 입력 때문에 활성화되고 많은 연결로 이루어진 구성 등은 유사하다고 할 수 있다[2]. 신경망은 위와 같은 단일 뉴런들이 복잡하게 연결되어 있는 상태의 뉴런 집합을 말하며 이는 대략적으로 다음과 같은 형태를 갖는다. 인공신경망에서 단일 뉴런들은 퍼셉트론이라는 용어로 불리우며 신경망의 가장 간단한 형태이다.

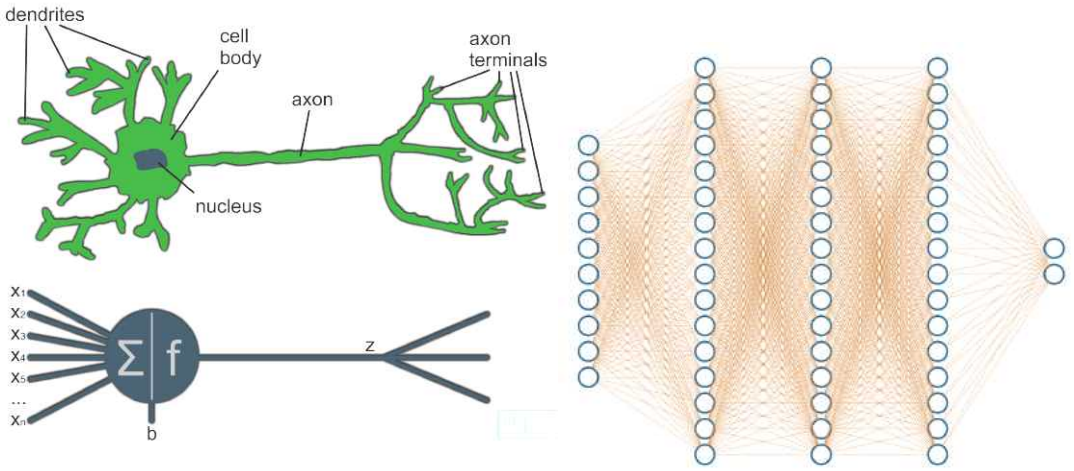

그림 2 (좌) 생물의 뉴런과 인공신경망의 뉴런과 (퍼셉트론) 비교. 좌측 X로 표시된 개체들은 입력이고 오른쪽 Z로 표시된 부분이 출력이라고 할 수 있다. (우) 단일 뉴런들이 복잡하게 연결된 신경망

위와 같은 신경망은 가장 왼쪽 입력 레이어에서 신호를 받아서 오른쪽 출력으로 결과 신호를 만들어낸다. 입력과 출력 중간에 위치한 레이어는 hidden layer로 불리며 각 레이어들은 퍼셉트론들이 나열된 형태로 구성된다. 하나의 퍼셉트론을 보면 다음과 같다.

---

2) Harrison Kinsley & Daniel Kukiela, Neural Networks from Scratch in Python

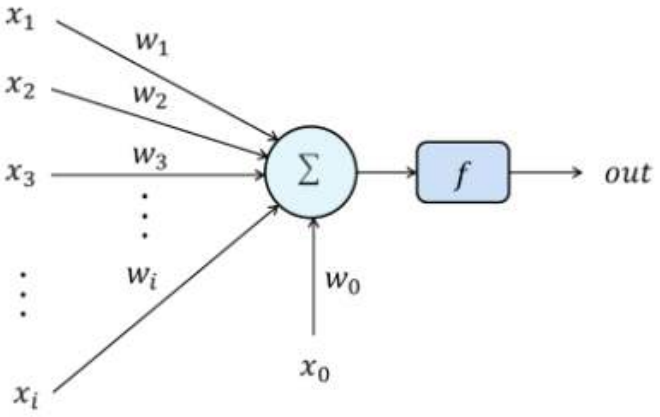

그림 3 퍼셉트론의 구조와 구성 요소

위 퍼셉트론에서 x는 입력 신호이고 w는 가중치 (weight)이다. 가중치는 모수 (parameter)라고 부르며 데이터로부터 신경망을 학습한다는 의미는 데이터로부터 이 모수를 추정한다는 의미와 같으며 통계에서는 모형을 적합 (fitting) 한다는 용어를 사용하기도 한다. w에는 weight (w_1, w_2, ..., w_i)와 bias (w_0)가 있다. weight들은 입력값을 출력으로 만들어주기 위한 함수의 크기나 부호를 정해주기 위한 모수로 볼 수 있으며 bias는 해당 출력값을 나타내는 함수의 위치를 정해주는 모수로 볼 수 있다. 퍼셉트론에서 이러한 모수와 입력값 출력값의 관계는 다음과 같은 식으로 나타낼 수 있다.

출력 = 합(입력 X 가중치) + Bias

해당 퍼셉트론에서 출력은 다음 연결된 퍼셉트론의 입력으로 들어가게 된다. 그러나 이런 식으로 계산이 이루어질 경우 레이어가 많아지고 연결된 퍼셉트론이 많아질수록 출력값이 너무 커지거나 0에 가까워지는 문제가 발생하게 된다. 생물학적인 관점에서 본다면 특정 임계점까지는 뉴런이 활성화되지 않고 임계값을 넘어가면 활성화되어 신호를 전달하는 상황이 더 자연스럽다고 볼 수 있다. 인공신경망에서도 이와같은 역할을 하는 함수가 있으며 이를 activation function이라고 한다. 즉, 출력이 특정 값 이하일 경우 실제 출력은 0이며 특정 값 이상 값이 계산되면 실제 출력값이 0이 되도록 만들어주는 함수이다.

실제 출력 = activation function (출력)

충분히 많은 양의 데이터가 준비되고 그에 맞는 신경망 모형을 가졌을 경우 다음으로 수행할 일은 적절한 모수를 추정하는 것이다. 모수는 관측된 데이터로부터 찾아내며 이렇게 모형의 훈련에 사용되는 관측 데이터를 훈련데이터 세트라고 하며 훈련된 모형의 성능을 시험하기 위해서 사용하는 데이터를 테스트 세트라고 한다. 훈련 데이터 세트에서는 다시 내부적으로 훈련데이터와 validation 데이터 세트가 구분되며 딥러닝 모형 훈련 과정에서 Training 데이터와 Validation 데이터가 무작위로 분할되어 모수를 학습하고 이 과정을 반복 수행하면서 얻어진 최종 훈련된 모형을 Test 데이터를 이용해서 평가한다.

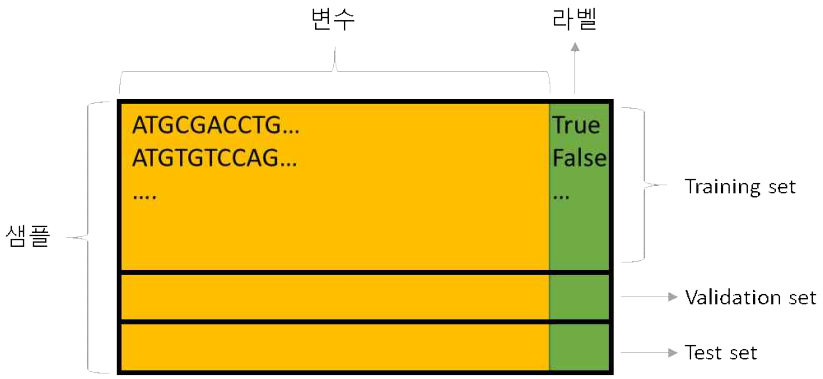

그림 4 신경망 모형 적합 및 검증을 위한 데이터 구분

일반적인 신경망의 경우 수만 개 이상의 모수가 있으며 주어진 데이터로부터 이를 찾아내는 과정은 일반적인 회귀분석에서 사용하는 least square와 같은 계산법은 사용할 수 없으며 시뮬레이션에 의한 수치계산법이 사용될 수밖에 없다. 인공신경망의 모형 적합을 이해하기 위해서는 우선 네트워크 에러와 Loss 에 대한 개념을 알아야 한다. 간단히 Loss는 계산된 값과 관측된 값의 차이라고 할 수 있다. 즉 특정 모수들을 가지고 최종 출력을 예측했을 경우 예측된 출력값과 관측된 값의 차이를 말한다.

$$L_i = -\sum_j y_{i,j} \log(\widehat{y_{i,j}})$$

위와 같은 식으로 나타낼 수 있으며 여기서 y_i,j 는 관측값이고 \hat{y}_{i,j} 는 계산된 예측값이라고 할 수 있다. 예를 들어 예측된 확률이 softmax_output = [0.7, 0.1, 0.2]라 하고 관측된 값을 [1, 0, 0] 이라 하면 위 식을 다음과 같이 다시 표현할 수 있다.

$$L_i = -\log(\widehat{y_{i,k}})$$

즉 여러 경우의 수가 있을 수 있으나 관측된 값은 하나이고 나머지 확률이 0이므로 관측된 값에 대해서만 예측값과의 차이를 계산한 것과 같게 된다. 이 경우는 0.7과 1의 차이를 말하며 이를 계산하면 -1 x log(0.7) = 0.3566 이 된다.

위와 같이 Loss를 정의한 후에는 어떻게 weight와 bias를 조절하면서 Loss를 줄이는 과정이 필요하다. 이 과정이 딥러닝의 가장 핵심인 알고리즘이다. 가장 쉬우면서도 직관적인 그러나 오래 걸리는 방법은 랜덤하게 weight와 bias를 할당한 후 가장 작은 Loss를 갖는 경우를 찾는 것이다. 그러나 무한에 가까운 조합을 모두 찾는 것은 불가능하며 좀 더 스마트한 방법으로 모수를 최적화할 필요가 있다. weight와 bias 모수들은 각각 크기가 다르긴 하지만 모두 loss의 형성에 어느 정도씩 기여하고 있다. 모수 최적화를 위해서는 이러한 모수들의 loss에 대한 기여도를 먼저 알아야 하고 어떻게 모수를 줄여나갈지를 알아야 한다. 딥러닝에서는 이러한 프로세스를 수행하기 위해서 편미분과 경사하강법 (gradient descent), 연쇄법칙 (chain rule), 그리고 backpropagation의 개념을 이용한다. y = f(x) 함수에서 x가 얼마나 y에 영향을 주는지 알기 위해서는 x의 증감분의 y의 증감을 나타내는 미분을 사용하며 함수로 볼 경우 접선의 기울기가 된다. 편미분은 함수에 관여하는 여러 모수가 (변수가) 있을 경우 각 모수마다 미분을 수행할 경우를 말하며 gradient는 특정 함수의 모든 가능한 편미분 식을 나열해 놓은 벡터라고 볼 수 있으며 함수 앞에 델로 (델타의 거꾸로 모양) 표시한다. backpropagation의 작동 방법은 각 모수들에 대해서 loss의 기여도를 편미분으로 구하는데 이를 계산하기 위해서 Chain rule을 이용하여 Loss에서부터 각 weight 및 bias에 대한 영향력을 계산한다.

더 큰 범주에서 이 두 문제는 지도학습 분류 문제는 판단을 딥러닝은 DNA 서열데이터 분석을 위한 딥러닝 기술은 크게 두 종류로 나눌 수 있다. 하나는 Convolutional neural network (CNN)이며 [1] 다른 하나는 Recurrent neural network (RNN)이다 [2]. 이들은 이미지 분석과 텍스트 분석에 각각 널리 사용되는 알고리즘들로서 이미지와 텍스트는 빅데이터와 Goldstandard 데이터를 가장 많이 확보한 분야들이며 딥러닝이 가장 활발히 발전하는 분야라고 볼 수 있다. 생물학적 서열데이터를 분석하는데 CNN과 RNN 알고리즘을 그대로 활용할 수 있다는 것은 데이터 생산이 느리고 Goldstandard 데이

터가 없는 생물학 분야의 연구자들에게는 무척 다행스러운 일이 아닐 수 없다.

(2) python/Keras 환경 설정

대용량의 데이터를 다루기 위해서는 python과 같은 프로그래밍 언어의 사용이 필수적으로 요구된다. 특히 많은 계산을 필요로 하는 딥러닝의 경우 CPU나 GPU 같은 하드웨어 의존성이 높은 관계로 오픈소스 기반의 Linux와 라이브러리 패키지들을 선호한다. 생물정보 분석 관련 패키지만 하더라도 약 7000개가 넘는 것으로 알려져 있으며 이들을 설치하거나 관리하기 위해서 다양한 패키지 관리 프로그램이 있으나 본 글에서는 Anaconda 라는 패키지 관리 프로그램을 사용한다 (https://www.anaconda.com/). 아나콘다는 특히 운영체제나 프로그래밍 언어의 종류 그리고 다른 패키지관리 프로그램들에 제약을 받지 않고 함께 사용될 수 있어서 데이터 과학 연구자들 사이에서 널리 사용되고 있다. 본 기고를 작성하는 2020.5월을 기준으로 다음과 같은 순서로 Python (3.7.7) 과 Anaconda를 (Anaconda3-2020.02) 설치한다 (Anaconda 설치는 컴퓨터 환경에 따라서 시간이 오래 걸릴 수 있음).

```
1. Python 3.7 다운로드 및 기본 설정으로 Python 설치
2. Anaconda 다운로드 및 기본 설정으로 Anaconda 설치
```

위 순서로 아나콘다를 설치하면 윈도우 시작 메뉴에 Anaconda3가 생성되고 Anaconda Navigator 아이콘을 클릭해서 실행할 수 있으며 Navigator 화면 왼쪽의Environments 탭을 통해 필요한 패키지들을 설치/삭제 할 수 있다. 본 글에서 사용한 biopython 패키지 설치를 위해 “Search package“ 버튼에 biopython을 입력하고 리스트 아이템 체크박스에 체크한 후 우측 하단의 apply 버튼을 누르면 설치가 시작된다. 만약 원하는 패키지 검색이 안 될 경우 “Update index“ 버튼을 누르면 보일 수 있다. 참고로 윈도우에서 Navigator를 이용한 패키지 관리시 컴퓨터 리소스를 과도하게 사용해서 느려지는 현상이 가끔 나타나는 이유로 가능하면 명령창 환경을 이용하여 패키지를 관리하는 것이 좋다. 본 글에서 소개하는 실습을 위해 biopython 외에 tensorflow와 keras를 설치할 필요가 있으며 명령창에서 conda 또는 pip를 이용해서 이들 패키지를 설치하는 법은 인터넷 자료 등을 참고하기 바란다. 아래 에디터 사용법을 소개한 글에서도 명령창 사용법을 참고할 수 있다.

코드를 입력하고 컴파일 할 수 있는 에디터로서 Jupyter Lab을 사용할 것이고 이 소프트웨어는 아나콘다를 설치하면 기본으로 설치된다 (Jupyter Lab은 Jupyter Notebook의 차세대 버전으로서 아나콘다 설치시에 Notebook과 Lab 둘 다 설치됨). Jupyter Lab은 웹브라우저에서 실행되는 대화형 에디터로서 데이터 분석과 딥러닝 연구에 적합한 편의성과 효율성으로 최근 사용자가 크게 증가하고 있다. 웹브라우저는 Chrome이나 Firefox를 사용하길 권장하며 브라우저를 사용하는 특성상 일반적인 아이콘을 클릭해서 프로그램을 실행하는 방법과는 다르다. 본 글에서는 필자가 주로 이용하는 방법을 소개하며 인터넷 자료를 참고해서 다른 방법으로 실행할 수도 있다. 방법은 윈도우를 기준으로 “제어판“ > “시스템 및 보안“ > “시스템“ > “고급 시스템 설정“으로 이동하여 아래 그림과 같이 윈도우 PATH에 anaconda3 경로를 추가한다 (추가할 경로는 anaconda3 설치디렉토리의 Scripts 디렉토리이며 필자의 경우 로그인 사용자가 user 라고 가정할 경우 다음과 같은 경로임 C:\Users\user\anaconda3\Scripts\).

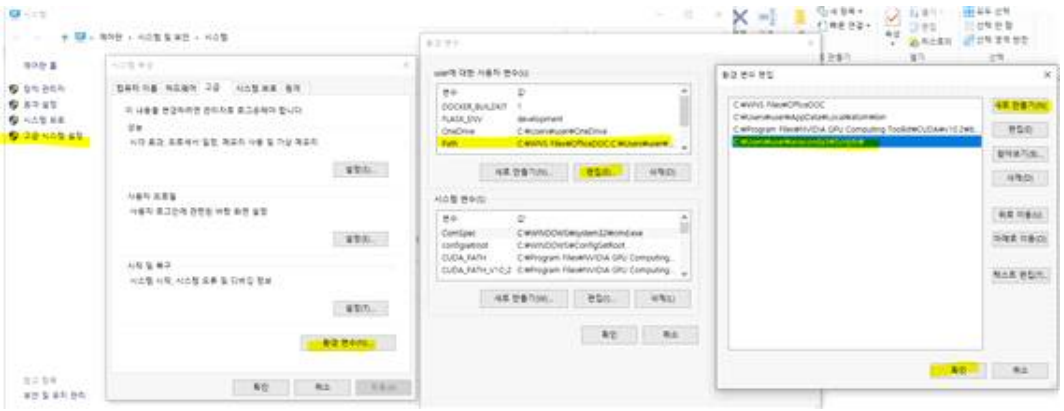

그림 5 윈도우 환경 설정 (Anaconda3 경로 PATH 추가). 왼쪽에서 오른쪽 방향으로 Highlight 된 항목부터 차례대로 실행

이 후 파일 탐색기를 열고 분석을 원하는 데이터가 있는 디렉토리로 (예를 들어 C:\mydocs\2020\dev\covid19) 이동한 후 탐색기 주소창에 “cmd“ 입력하고 엔터를 누르면 해당 디렉토리를 홈으로 하는 command 명령창이 나온다. 이 후 “activate“ 명령어와 “jupyter lab“ 명령어를 실행한다 (참고로 “activate“ 명령을 실행하면 (base) 라는 anaconda 환경에 진입했다는 표시가 프롬프트 앞에 붙음, 그림 참고).

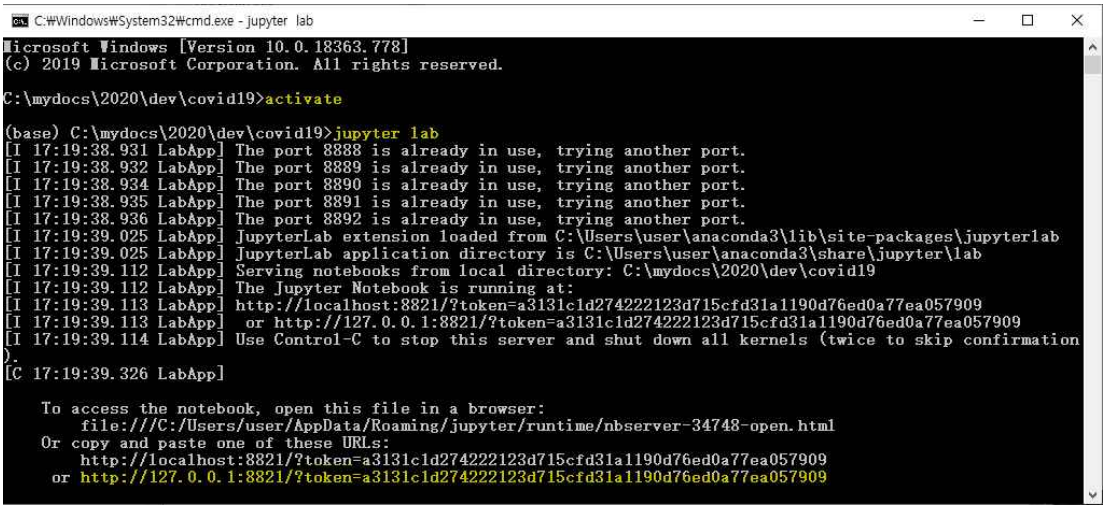

그림 6 명령창을 이용한 anaconda 환경 진입과 Jupyter lab 실행

“jupyter lab“ 명령어를 실행하면 위 그림과 같이 http://127.0.0.1:8892/?token=99... 로 시작하는 메시지가 출력되며 이 부분을 복사하여 크롬 브라우저의 주소창에 붙여 넣으면 그림 3과 같이 Jupyter lab 에디터가 실행된다 (컴퓨터 설정에 따라 jupyter lab을 실행하면 바로 브라우저가 실행되는 경우도 있음). 에디터 화면에서 왼쪽 판넬은 파일 탐색창 등이 보여지고 오른쪽 화면 Launcher의 Notebook 중 Python3를 클릭하면 파이썬 코드를 작성할 수 있는 에디터가 실행 된다. Jupyter lab을 사용하는 방법에 대한 자료는 인터넷에서 쉽게 찾을 수 있으며 자주 사용하는 몇 가지 단축키 사용법도 같이 익혀두도록 한다.

(3) python 사용법

딥러닝을 위해서는 라벨링 데이터가 필수이다. 예를 들어 동물을 인식하기 위한 딥러닝 모형을 학습한다고 가정할 때 고양이 사진은 “고양이“라는, 강아지 사진은 “강아지“라는 라벨이 붙은 훈련데이터를 모형에 제공해야 한다. 서열 분석의 경우에는 DNA 서열과 함께 해당 서열의 표현형이 라벨이 될 수 있다 (Genotype-phenotype 짝 데이터). 예를 들어 특정 전사인자가 결합하는 DNA 서열을 예측하는 딥러닝 모형을 학습하고자 할 경우 전사 시작 위치로부터 약 500bp길이의 DNA 서열 데이터와 해당 전사인자가 위 DNA에 실제로 붙는지를 나타내는 True 또는 False 라벨이 붙은 데이터가 필요하다. 일반적으로 통계적 분석을 위한 데이터는 샘플의 개수와 (행) 변수의 개수로 (열) 구분되어 2차원 배열 형태로 표현된다. 딥러닝에서도 같은 방식으로 데

이터를 표현하며 필요한 샘플의 수는 학습할 모형의 복잡도에 따라서 달라질 수 있지만 최소 수천 개 이상이 필요하며 수 만개 이상의 가능한 많은 데이터를 사용하기를 권장하고 있다. 예를 들어 ImageNet[3] 데이터베이스의 경우 약 140만개 이상의 라벨링된 이미지를 보유하고 있으며 계속해서 증가하고 있다.

python은 전세계적으로 가장 많이 사용되는 프로그래밍 언어로서 기계학습이나딥러닝 관련된 다양한 소프트웨어들이 python 라이브러리 형태로 제공되고 있다. 다음은 몇 가지 기본적인 python 사용법을 소개한다. 대부분의 언어는 ‘변수’, ‘자료구조’, ‘함수’, ‘제어문’, ‘조건문’ 등의 내용만 알면 익히기 쉽다. 파이썬에서 사용하는 변수는 값을 저장하는 메모리의 주소를 가리키는 바인더이다. 논리연산자로 True, False와 and, or가 있으며 조건문은 ‘if’, ‘elif’, ‘else’ 등을 사용하며 반복문은 ‘for’, ‘while’ 등의 키워드를 사용한다. python을 사용한 딥러닝에서 특별히 중요한 부분은 자료형을 이해하는 것이다. 먼저 리스트는 여러개의 데이터를 순서대로 저장하고 관리할 때 사용한다.

```
expression = [“geneA”, 1]
expression = [1, 2, 3]
expression = [] [4]
```

리스트의 인덱싱은 0부터 시작하며 값 자체를 가리키고 리스트를 슬라이싱(:을 사용한 경우) 할 경우는 값 사이 경계선을 나타낸다. 리스트에 데이터를 삽입하거나 삭제할 경우에는 .append()와 .pop() 함수를 사용할 수 있다.

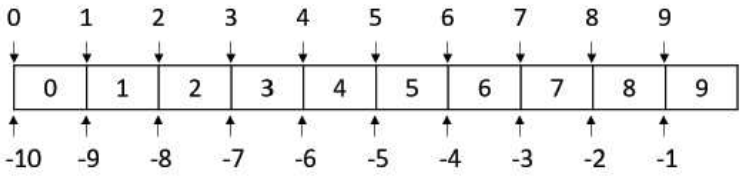

그림 7 0부터 9까지 값을 가진 리스트와 슬라이싱

리스트와 비슷한 기능을 가진 튜플의 경우 리스트의 “[”, “]” 대신

3) http://www.image-net.org/

“(” , “)” 를 사용하는 것이 다르며 원소를 변경할 수 없다는 특징이 있다. 리스트보다 빠른 속도를 가지며 리스트와 동일한 인덱싱 방법을 사용한다. 리스트나 튜플을 반복문과 함께 사용하여 다음과 같이 활용할 수 있다.

```
geneids = [x for x in range(10)] #리스트 컴프리헨션

for geneid in geneids:
    print(“geneid: %s” %geneid)
f
```

딕셔너리는 key와 value를 쌍으로 저장하는 데이터 구조이며 “{” 와 “}” 를 사용한다. 키 값으로만 인덱싱을 할 수 있으며 데이터 추가는 키 값으로 추가하며 삭제는 del 함수를 이용한다.

```
geneExpr = {}
geneExpr[‘A’] = 0.5
geneExpr[‘B’] = 1.2
del geneExpr[‘B’]

geneExpr.keys()
geneExpr.values()

for geneid, expval in geneExpr.items():
    print(“%s expression value is %s” %(geneid, expval))
```

위 python에서 제공하는 기본 자료구조 외에 Numpy와 Pandas 모듈에서 제공하는 자료구조가 있다. Numpy의 자료구조인 ndarray는 행렬이나 다차원 배열 처를 위해 사용될 수 있으며 같은 타입의 데이터만 허용한다. 또한 리스트 구조에 비해서 20배 이상 빠른속도를 가지며 tensor 자료구조와 같다고 볼 수 있다.

```
import numpy as np

arr = [1, 2, 3]
print(arr)
print(type(arr))

a = np.array([1, 2, 3])
```

```
print(a.shape)
(3,)
```

array의 shape은 다음과 같이 나타낼 수 있다[4].

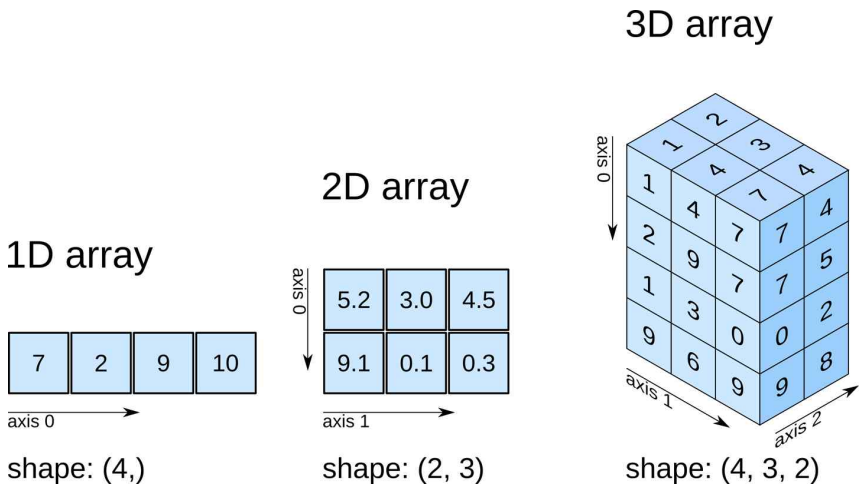

그림 8 Numpy 모듈의 1d, 2d, 3d array

Pandas는 Series와 DataFrame 두 개의 자료형을 가지며 각각 1차원, 2차원 데이터를 다루는 자료구조이다. 리스트와 딕셔너리의 조합형으로 볼 수 있으며 숫자형, 문자형, 범주형 등의 다양한 데이터를 값으로 가질 수 있다. DataFrame의 생성은 “{”, “}”를 사용하며 DataFrame은 Series의 집합으로 볼 수 있음.

```
from pandas import Series, DataFrame

genes = Series([0.1, 0.2, 1.4, 0.6, 1.1])
print(genes)

0    0.1
1    0.2
2    1.4
3    0.6
4    1.1
dtype: float64

genes = Series([0.1, 0.2, 1.4, 0.6, 1.1], index=['A', 'B', 'C', 'D', 'E'])
print(genes)
```

4) https://www.oreilly.com/library/view/elegant-scipy/9781491922927/ch01.html

```
A    0.1
B    0.2
C    1.4
D    0.6
E    1.1
dtype: float64
```

다차원 numpy 자료구조를 Tensor라고 볼 수 있으며 텐서는 수치형 (float32, uint8, float64) 데이터를 주로 다룬다. 0D, 1D, 2D는 각각 스칼라, 벡터, 행렬로 볼 수 있으며 랭크 (ndim)와 크기 (shape), 타입 (dtype) 속성이 있다.

```
x = np.array(
    [[[1, 2, 3],
      [2, 3, 4]],
     [[5, 6, 7],
      [8, 9, 10]],
     [[11, 12, 13],
      [14, 15, 16]]]
)
print(x.ndim)
print(x.shape)
print(x.dtype)

3
(3, 2, 3)
int64
```

(4) 서열 데이터 전처리

DNA 서열 분석을 위해서는 biopython 패키지를 사용하나 딥러닝에서는 서열을 문자열로 다루기 때문에 필수 요구 조건은 아니라고 할 수 있다. 서열을 이용한 기계학습을 수행하기 위해서는 먼저 염기 서열 (A, T, G, C) 문자들을 컴퓨터가 계산하기 쉽도록 변환하는 과정이 필요하다. 크게 두 가지 접근법이 있으며 하나는 One-hot encode 방법과 다른 하나는 Text 데이터와 같이 취급하여 Bag of words를 만들어서 분석하는 방법이다. 이들은 앞에서 언급한 CNN과 RNN 알고리즘이 각각 연계되어 이미지와 자연어처리 분석에 사용된다. 본 글에서는 One-hot encoding을 적용하는 CNN 방법을 소개한다. One-hot encoding은 딥러닝에서 가장 널리 사용되는 방법이며 4 종류의 염

기를 갖는 DNA의 경우 “A“는 [1,0,0,0], “T“는 [0,0,0,1], “G“는 [0,0,1,0], 그리고 “C“는 [0,1,0,0]으로 인코딩 할 수 있다. 따라서 “ATGCTA“는 [[0,0,0,1], [0,0,0,1], [0,0,1,0], [0,1,0,0], [0,0,0,1], [1,0,0,0]]의 6x4 행렬로 표현 된다. scikit-learn 패키지는 이러한 인코딩을 쉽게 수행할 수 있는 함수 OneHotEncoder를 제공하고 있으나 아래와 같이 직접 서열에 대한 인코딩을 수행 할 수도 있다. 결과에서 보는 바와 같이 2차원의 6x4 배열이 생성되었음을 확인할 수 있으며 이러한 인코딩을 적용한 서열 데이터는 CNN 모형의 첫 레이어에 입력되는 데이터로 사용된다.

```
import numpy as np

my_string="ATACAA"
my_array=np.array(list(my_string))
onehot_encode = np.zeros((len(my_array),4), dtype=int)
base_dict  = {'A':0, 'C':1, 'G':2, 'T':3}
for i in range(len(my_array)):
    onehot_encode[i, base_dict[my_array[i]]]=1
print(onehot_encode)
[[1 0 0 0]
 [0 0 0 1]
 [1 0 0 0]
 [0 1 0 0]
 [1 0 0 0]
 [1 0 0 0]]
```

CNN 모형을 이해하기 위해서는 먼저 PFM (Position Frequency Matrix)와 PWM (Position Weight Matrix) 개념의 이해가 필요하다. PWM 방식의 계산을 통해 특정 전사인자의 결합 서열을 찾는 연구는 생물정보학 분야에서 오랫동안 연구되어온 분야 중 하나이다[5]. 설명을 위해 alignment가 수행된 몇 개의 서열들을 가정하면 PFM은 이 서열들의 특정 위치에 A, T, G, C 각 염기들의 빈도수를 나타내며 PWM은 각 염기의 비율을 나타낸다. 다음은 세 개의 서열에서 PFM과 PWM을 구하는 코드로서 PWM 값이 계산되는 과정을 익혀두자.

```
from Bio import motifs
```

5) C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton, “Detecting subtle sequence signals: A gibbs sampling strategy for multiple alignment,” Science (80-.)., vol. 262, no. 5131, pp. 208.214, 1993

```
from Bio.Seq import Seq
instances = [Seq("TACAA"), Seq("TACGA"), Seq("TACAA")]
m = motifs.create(instances)
pfm = m.counts
pwm = m.counts.normalize(pseudocounts=0.5)
print(pwm)
       0     1     2     3     4
A:   0.10  0.70  0.10  0.50  0.70
C:   0.10  0.10  0.70  0.10  0.10
G:   0.10  0.10  0.10  0.30  0.10
T:   0.70  0.10  0.10  0.10  0.10
```

pseudocounts는 계산시 NULL이나 0으로 나누어지는 경우를 방지하기 위한 조치로 각 PFM의 각 원소에 0.5를 더하여 PWM을 구한다. 이렇게 얻어진 특정 서열 모티프의 PWM은 새로운 서열이 One-hot encoding 방식으로 주어져 있을 경우 서열 처음 위치부터 마지막까지 Sliding window 방식으로 해당 모티프가 있는 위치를 탐색할 수 있다. 앞서 One-hot encoding을 설명하기 위한 “ATACAA“ 서열에서 위 PWM 모티프가 존재하는지 탐색하는 과정을 살펴보자. “ATACAA“는 길이 5인 슬라이딩 윈도우를 사용하면 “ATACA“와 “TACAA“ 두 개의 서열로 나눌 수 있다. 이 두 서열을 One-hot encoding으로 전환 후 위 PWM과 원소들끼리 곱하면 One-hot encoding에서 0이 아닌 위치와 동일 위치의 PWM 값들만 남게 된다. 여기서 0이 아닌 값들을 모두 곱한 후 log를 취해 주면 해당 서열이 모티프와 얼마나 비슷한지를 나타내는 스칼라 값이 구해진다. 이론적으로 이 값이 0이면 동일한 서열이다. 아래 코드의 s2 값이 0에 더 가까우므로 두 번째 서열 TACAA이 첫 번재 서열보다 주어진 모티프와 유사하다는 올바른 결론을 보여주고 있다.

```
pwm_arr = np.array(list(pwm.values())).transpose()
s1 = np.multiply(onehot_encode[0:5,], pwm_arr)
s2 = np.multiply(onehot_encode[1:6,], pwm_arr)
print(np.log(np.prod(np.sum(s1, axis=1))))  #s1 score
print(np.log(np.prod(np.sum(s2, axis=1))))  #s2 score
-9.567015315914915
-2.119846956314875
```

앞서 설명한 PWM을 이용한 모티프 탐색 과정을 실제 딥러닝 과정에서 구현할 필요는 없다. One-hot encoding 방법으로 서열을 변환하여 입력 데이터로 사용할 경우 앞에서 언급한 바와 같이 이미지 분석에서 사용한 딥러닝 알고

리즘과 동일한 알고리즘을 그대로 가져다 사용할 수 있다. 본 글에서는 분석에 필요한 데이터를 외부에서 읽어오는 대신 모의 서열 데이터를 생성하여 분석을 수행하도록 한다. 전사인자가 결합하는 특정 DNA 모티프를 찾는 딥러닝 모형을 개발하는 것을 목표로 하며 바인딩 모티프, 즉, PWM 생성에 사용된 모티프는 “CCGGAA“로 미리 설정해 둔다. 전체 서열의 길이는 20bp로 모티프 양쪽 7bp를 랜덤 서열로 할당했으며 위 모티프를 갖는 positive 서열과 20bp 전부가 랜덤 서열인 negative 서열 각각 1000개씩을 생성했다.

```
seq_length = 20
num_sample = 1000

motif_pwm = np.array([[10.41, 22.86, 1.92, 1.55, 98.60, 86.66],
                [68.20, 65.25, 0.50, 0.35, 0.25, 2.57],
                [17.27, 8.30, 94.77, 97.32, 0.87, 0.00],
                [4.13, 3.59, 2.81, 0.78, 0.28, 10.77]])

pwm = np.hstack([np.ones((4,7)), motif_pwm, np.ones((4,7))])
pos = np.array([np.random.choice(['A', 'C', 'G', 'T'],
                         num_sample,
                         p=pwm[:,i]/sum(pwm[:,i]))
                         for i in range(seq_length)]).transpose()
neg = np.array([np.random.choice(['A', 'C', 'G', 'T'],
                         num_sample,
                         p=np.array([1,1,1,1])/4)
                         for i in range(seq_length)]).transpose()
print(pos.shape)
tmp_out = [''.join(x) for x in pos[1:5,]]
print(tmp_out)

['TCGAGTCGAGGATTATTGTC',
 'TGTGGGTAAGGAAAGCGGGT',
 'AGGATTCGAGGAACTAACAA',
 'TGTGGCGCAGGAAAGCGAAA']
```

위 출력에서 보듯 positive (pos) 서열의 8번째 위치부터 CCGGAA 패턴의 서열이 많이 발생하고 있는 것을 확인할 수 있으며 negative (neg) 서열을 출력해 본다면 전체 서열이 랜덤하게 분포되어 있는 것도 확인할 수 있을 것이다. 전사인자가 결합하는 유무를 나타내는 각 서열의 라벨링 데이터는 다음 전처리 과정에서 One-hot encoding 형식으로 바로 생성하도록 한다.

다음으로 생성된 모의 서열 데이터를 CNN 모형에 적용하기 위한 One-hot

encoding 변환과 training 및 test 데이터셋을 분할하는 과정을 수행한다. One-hot encoding 변환은 전반부에 설명한 방법과 동일하며 다만 positive와 negative 서열들 각각에 대한 코딩과 이들을 vstack 함수를 이용해서 하나의 매트리스로 병합하는 과정이 필요하다. 서열들의 라벨링 데이터는 결합 모티프가 존재하는 positive 서열의 경우 [0,1], negative 서열의 경우는 [1,0]으로 One-hot encoding 형식으로 생성한다.

```
base_dict  = {'A':0, 'C':1, 'G':2, 'T':3}
onehot_encode_pos = np.zeros((num_sample, seq_length, 4))
onehot_encode_pos_label = np.zeros((num_sample, 2), dtype=int)
onehot_encode_pos_label[:,0] = 1
onehot_encode_neg = np.zeros((num_sample, seq_length, 4))
onehot_encode_neg_label = np.zeros((num_sample, 2), dtype=int)
onehot_encode_neg_label[:,1] = 1
for i in range(num_sample):
    for j in range(seq_length):
        onehot_encode_pos[i,j,base_dict[pos[i,j]]] = 1
        onehot_encode_neg[i,j,base_dict[neg[i,j]]] = 1
X = np.vstack((onehot_encode_pos, onehot_encode_neg))
y = np.vstack((onehot_encode_pos_label, onehot_encode_neg_label))
print(X.shape, y.shape)
(2000, 20, 4) (2000, 2)
```

위 출력에서 확인한 것처럼 인코딩된 데이터 X는 positive와 negative가 병합된 2000개의 샘플이고 각 샘플은 20bp 길이의 서열이며 4개 배열로 one-hot encoding 형식으로 저장되어 있다. 또한 라벨링 데이터 y도 2000개 서열 샘플에 대해서 모두 one-hot encoding 형식의 배열 데이터가 저장되어 있다. 이들 데이터를 training 과 test 세트로 분할하는 과정은 scikit-learn 패키지의 train_test_split 함수를 사용하면 간편하게 수행할 수 있다.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
                                X,
                                y,
                                test_size=0.2,
                                random_state=125)
print(X_train.shape, y_train.shape)
(1600, 20, 4) (1600, 2)
```

여기서는 test 데이터 세트의 크기를 20%로 설정하여 training 세트는 1600개의 샘플 크기를 가지게 된다. 실제 코드를 실행해보고 각 데이터들의 실제 값들과 차원 등이 올바로 만들어졌는지 확인해 보도록 하자.

(5) 기계학습 적용 모형 학습

이제 실제로 CNN 모형을 만들어 보겠다. Keras는 딥러닝을 쉽게 수행할 수 있는 프레임워크로 CNN 모형을 위해 Conv1D, Conv2D 등의 함수를 제공한다. 아래 코드를 참고하자.

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Flatten
from keras.layers import Conv1D, MaxPooling1D

parameters = {
    'num_filters': 10,
    'kernal_size': 5,
    'pooling_size': 4,
    'num_classes': 2
}

def create_model(params):
    model = Sequential()
    model.add(
        Conv1D(
            filters = params['num_filters'],
            kernel_size = params['kernal_size'],
            activation = 'relu',
            padding = 'same'
        )
    )

    model.add(
        MaxPooling1D(
            pool_size = params['pooling_size']
        )
    )

    model.add(Flatten())
    model.add(
        Dense(
            units = params['num_classes'],
```

```
            activation = 'sigmoid'
        )
    )

    model.compile(
        loss = 'categorical_crossentropy',
        optimizer = 'adam',
        metrics = ['accuracy']
    )

    return model
```

우선 Conv1D로 하나의 레이어를 추가하고 레이어의 Unit은 10개로서 10개의 서로 다른 모티프 ('num_filters')를 이용한다. 하나의 모티프를 기준으로 모티프의 길이로 볼 수 있는 Filter 크기를 ('kernal_size') 5로 설정하였다. 즉, 5x4의 sliding window가 (PWM) 입력 데이터인 20x4의 one-hot encoding 서열에 대해서 1bp씩 (strides 옵션, Conv1D에 default 1로 설정되어 있어서 생략, API reference 참고) 움직여가며 score를 계산하는 것과 같은 개념이다. padding을 "same"으로 하고 Maxpooling 크기를 4로 (`'pooling_size'`) 설정하여 20개의 score를 4개씩 나누어 Max 값을 저장하므로 최종 5개의 score 배열이 출력되며 이런 모티프가 10개 이므로 Max pooling을 수행한 후 출력 shape는 (5, 10) 이다. Flatten 함수는 이를 50 개 원소를 갖는 1차원 배열로 변환하고 다음 레이어에서 sigmod activation 함수를 이용하여 두 개의 클래스에 (결합 또는 비결합) 대한 결과를 얻는다. 그러나 위 코드는 (`create_model`) 실제 데이터를 이용해서 모형을 적합하는 단계가 아닌 모형의 환경을 생성하는 단계로 Convolution 구조와 함께 `model.compile` 함수에서 모형을 적합하기 위한 최적화 방법(`'adam'`)과 loss 함수 (`'categorical_crossentropy'`) (0 또는 1의 분류 문제이므로)를 사용한다는 환경을 설정해준 것이다. 모형을 적합하는 코드는 다음과 같다.

```
model = create_model(parameters)
history = model.fit(
    X_train,
    y_train,
    batch_size=100,
    epochs=20,
    validation_split=0.1
)
```

만들어진 CNN 모형을 실제 데이터를 이용해 학습하는 과정이 필요하다. 이를 데이터를 모형에 적합 (fit) 한다고 말하며 학습은 모형의 파라메터를 추정하는 과정이다. 위 코드와 같이 create_model로 지정한 환경대로 모형을 생성해 주었고 fit 함수를 통해서 데이터를 전달해 주고 적합하기 위한 몇몇 옵션들을 지정해 주었다. batch_size는 데이터를 메모리로 옮길 때 얼마나 많은 데이터를 이동할지를 결정해 주는 옵션이며 epochs은 반복의 횟수이다. validation_split 옵션은 훈련데이터를 이용해서 모형을 적합할 때 일부 데이터를 (10%) 검증용으로 사용한다고 지정한 옵션이다. 이를 수행하면 대략적으로 다음과 같이 loss와 accuracy 값이 출력되는 것을 볼 수 있는데 이 중 val_loss와 val_accuracy가 validation 데이터를 사용할 경우 손실과 정확도에 대한 정보를 나타낸다.

```
Epoch 1/20
15/15 [==============================] - 1s 18ms/step - loss: 0.8414
- accuracy: 0.5025 - val_loss: 0.7043 - val_accuracy: 0.5312
Epoch 2/20
15/15 [==============================] - 0s 7ms/step - loss: 0.6982 -
accuracy: 0.5227 - val_loss: 0.6676 - val_accuracy: 0.5813
Epoch 3/20
15/15 [==============================] - 0s 8ms/step - loss: 0.6379 -
accuracy: 0.6682 - val_loss: 0.6435 - val_accuracy: 0.6125
Epoch 4/20
15/15 [==============================] - 0s 6ms/step - loss: 0.5998 -

....

Epoch 18/20
15/15 [==============================] - 0s 6ms/step - loss: 0.2247 -
accuracy: 0.9480 - val_loss: 0.2314 - val_accuracy: 0.9375
Epoch 19/20
15/15 [==============================] - 0s 6ms/step - loss: 0.2087 -
accuracy: 0.9451 - val_loss: 0.2200 - val_accuracy: 0.9438
Epoch 20/20
15/15 [==============================] - 0s 6ms/step - loss: 0.2083 -
accuracy: 0.9397 - val_loss: 0.2084 - val_accuracy: 0.9438
```

위 모형의 summary 함수를 통한 각 레이어들에서 출력되는 shape를 앞서 소개한 코드의 설명과 비교할 수 있다.

```
model.summary()

Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_8 (Conv1D)            (None, 16, 10)            210
_________________________________________________________________
max_pooling1d_8 (MaxPooling1 (None, 4, 10)             0
_________________________________________________________________
flatten_8 (Flatten)          (None, 40)                0
_________________________________________________________________
dense_8 (Dense)              (None, 2)                 82
=================================================================
Total params: 292
Trainable params: 292
Non-trainable params: 0
```

위 코드를 실행하고 모형에 대한 정확도와 (accuracy) 손실 (loss) 정도를 그래프로 그려보면 다음과 같다. epochs=50을 수행했으며 정확도는 맞게 예측한 개수를 전체 예측 수로 나눈 비율이고 손실은 예측값과 실제값의 차이를 cross-entropy 형식으로 계산한 값을 의미한다.

```
import matplotlib.pyplot as plt

plt.figure()
plt.plot(history.history['val_loss'])
plt.plot(history.history['val_accuracy'])
plt.title('valid accuracy and valid loss')
plt.ylabel('val accuracy and val loss')
plt.xlabel('epoch')
plt.legend(['valid loss', 'valid accuracy'])
plt.show()
```

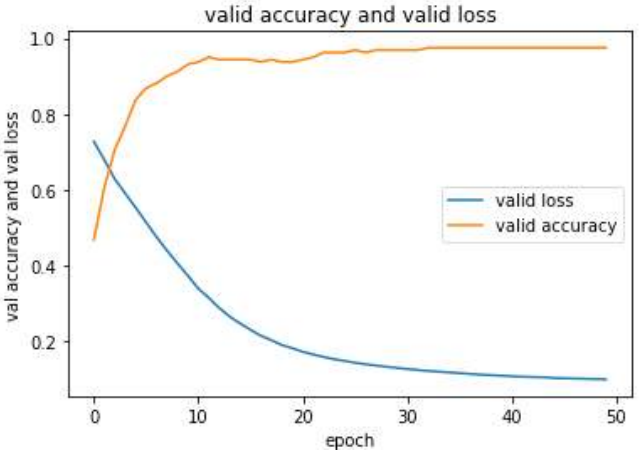

그림 9 만들어진 모형을 관측된 데이터와 적합하는 과정을 50번 수행한 후 계산된 손실과 정확도 변화

위 그림과 같이 반복적인 적합이 진행되면서 손실은 줄어들과 정확도는 높아지는 것을 알 수 있다. 이는 모형이 반복적으로 적합되면서 신경망의 모수들이 (back-propagation과 forward-propagation이 반복되면서) 점점 최적화가 되어가는 것이라고 볼 수 있으며 약 20회의 반복이 지난 후에는 손실과 정확도의 정도가 크게 달라지지 않는 것을 볼 수 있다.

이렇게 훈련된 CNN 모형을 이용해서 앞서 분류해 둔 test 데이터 세트를 이용하여 모형의 예측 성능을 평가해 본다. model.evaluate 함수는 테스트 데이터셋의 loss 값과 accuracy 값을 계산해 준다.

```
from sklearn.metrics import confusion_matrix

print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
--------------------------
X_test shape: (400, 20, 4)
y_test shape: (400, 2)
--------------------------

score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', round(score[0], 3))
print('Test accuracy:', round(score[1], 3))
----------------------
Test loss: 0.095
Test accuracy: 0.973
----------------------
```

그러나 테스트 데이터를 이용한 손실과 정확도의 계산보다는 일반적으로 False positive (효과가 좋지 않지만 좋다고 판단) 비율이나 (의료 분야에서는) False negative (암환자를 정상이라고 판단) 비율 등에 대한 관심이 더 높다. 이들은 각각 제1종 오류와 제2종 오류로 알려져 있으며 이들에 대한 정량 수치를 비교하여 조건에 맞는 딥러닝 모형을 평가하고 선별하여 사용할 수 있다.

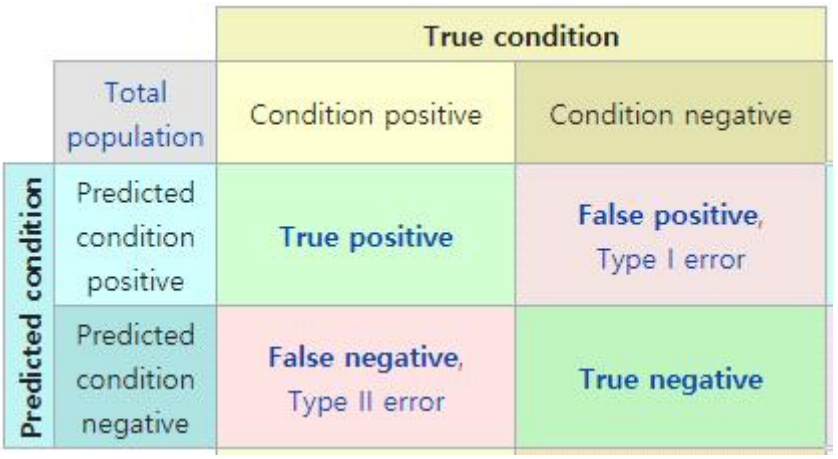

그림 10 예측값과 실제값의 분류에 따른 confusion matrix (Wiki)

```
predictions = np.argmax(model.predict(X_test), axis=-1)
cm = confusion_matrix(y_test[:,1], predictions)
print(cm)

[[188   6]
 [  5 201]]
```

위와 같이 model.predict 함수와 np.argmax 함수를 이용해서 분류를 수행할 수 있으며 confusion_matrix 함수로 분류표를 만들 수 있다. 위 결과에서는 전체 테스트 결과 중 실제 모티프가 있는 서열을 있다고 예측한 개수가 188개 (true positive), 모티프가 없는 서열에 대해서 없다고 예측한 경우가 195개 (true negative) 이다. 한편 모티프가 없음에도 있다고 분류한 1종 오류의 개수는 6개, 모티프가 있음에도 없다고 분류한 개수는 5개로서 적합한 모형이 비교적 높은 정확도로 작동하고 있음을 알 수 있다. 이러한 분류표를 이용해서 모형의 예측력을 평가하는 지표를 계산할 수 있으며 이 외에도 잘 알려진 평가 지표로 Accuracy, Recall, Precision, F1 등이 있다. 각 항목에 대

한 의미와 계산법은 인터넷을 통해 어렵지 않게 찾아서 참고할 수 있다.

(6) 실습 데이터 적용 및 고찰

위 강의 내용은 실제 서열 데이터를 사용하기보다는 모의의 DNA 데이터를 생성하여 딥러닝 개념의 이해와 실제 구현을 수행하는 것에 중점을 두고 있다. 모의 DNA 데이터의 경우 특정 모티프가 있고 없음을 기준으로 두 종류에 대한 분류를 (classification) 수행했으나 본 수업의 실제 데이터의 경우 회귀 (regression) 문제에 관한 데이터를 사용할 것이다. 실험으로 사용되는 서열의 경우 5’ UTR 위치의 일정 길이를 갖는 DNA 서열이 (X) 될 수 있으며 이에 따른 하위 GFP 단백질의 발현양을 (y) 형광측정기를 이용하여 정량화 할 수 있다. 아래 테이블은 이러한 데이터의 예제를 보여준다.

	UTR 서열 (X)	발현값 (Y)
샘플1	TTGTAGTTGATTGTAGTTGA,	30
샘플2	TAGTAATGAGTTTAGCCTGA,	12
...	...	...
샘플10000	ATAGGTATGGCTGCTTATAA,	10

표 1 모의데이터 5’UTR 서열 (X), 각 서열에 해당하는 단백질 정량 발현값 (Y)

문제는 충분히 많은 양의 이러한 데이터를 수집할 수 없다는 것이다. 딥러닝을 위해서는 약 수천에서 수만 개 이상의 샘플이 필요하다. 그러나 위와 같은 UTR 서열을 갖는 실제 클론을 하나하나 제작하는 실험은 많은 시간과 노동력이 필요한 과정이다. 이에 대한 대안으로 위와 같은 서열을 올리고머 칩을 활용해서 합성한 후 라이브러리로 클로닝하여 플레이트 콜로니 형태로 형광을 관찰할 경우 10,000개 이상의 클론에 대한 형광을 관측할 수 있다. 그러나 이 경우도 콜로니에서 관측한 형광값에 해당하는 콜로니의 UTR 서열을 해독 과정이 필요하다. 이는 콜로니 하나하나를 키워서 DNA 정재 후 sanger 시퀀싱을 해야하는 과정 때문에 플레이트에서 다량의 세포를 키우고 형광을 관찰하는 장점을 살릴 수 없다. 최근 소개되는 Oxford Nanopore Technology를 이용한 서열 해독 기술은 저렴한 가격으로 소규모 실험실에서도 충분히 많은 양의 서열 데이터를 해독할 수 있게 되었다. 위에서 소개한 올리고칩 기술과 ONT 기술을 활용하여 대량의 딥러닝용 빅데이터를 수집할

수 있다.

그림 11 Oxford Nanopore Technology의
시퀀싱 플랫폼 MinION Mk1C

위와 같이 직접 데이터를 생산하기 어려울경우 시뮬레이션을 통한 데이터 생산도 가능하다. 본 강의 자료에서 언급하지 않은 중요한 딥러닝의 요소 중 하나는 hypter parameter에 대한 개념과 선택 방법, 그리고 모형 적합에 있어서 overfit과 underfit의 개념 등이 있다.

본 강의에서는 생물학 서열을 데이터로 활용하여 딥러닝 특히 CNN 알고리즘을 구현하는 방법을 알아 보았다. python 기반의 keras를 사용하였으나 tensorflow나 pytorch와 같은 다양한 딥러닝 프레임워크를 사용할 수 있고 python 외에도 R이나 C++ 등의 언어를 사용하여 AI를 구현할 수 있다. 이렇게 쉽게 접할 수 있는 딥러닝 기술을 활용하기 위해서 가장 중요한 점은 라벨링된 서열 데이터를 대량으로 수집하고 자유자제로 다룰 수 있는 능력을 기르는 것이다. 대량의 서열 데이터를 다루는 관점에서는 python에서는 biopython, numpy, 그리고 pandas에 익숙해져야 하고 R의 경우는 tidyverse, ggplot2 등의 패키지와 함께 bioconductor 저장고를 활용하는데 익숙해져야 하겠다. 데이터가 커질수록 계산량은 많아지게 되고 자연스럽게 GPU를 활용하여 계산을 수행할 수 있다. 딥러닝 개발 환경이 점점 복잡해지게 된다면 다른 전문가들이 기존에 구축해 놓은 딥러닝 개발 환경을 그대로 가져와 사용하기 위한 Docker (https://www.docker.com/)도 활용할 수 있을 것이다.

딥러닝을 위한 코딩뿐만 아니라 일반적인 프로그래밍을 수행할 때 가장 쉽게 실력을 쌓는 방법 중 하나는 다른 사람이 만들어 놓은 코드를 보고 따라하는 것이다. 최근에는 참고할 수 있는 무료 동영상 강좌를 어렵지 않게 찾을 수 있게 되어서 이들을 참고하는 것도 하나의 방법이지만 필자가 추천하는 방법 중 하나는 github나 (https://github.com/) 딥러닝의 경우 kaggle

(https://www.kaggle.com/) 사이트에서 필요한 코드를 직접 검색해서 활용하는 방법도 고려해 볼 수 있다. 물론 사용하는 패키지나 함수의 reference를 꼼꼼히 읽어 보는 것도 필요하다.