모델 설계하기

1. 모델의 정의

(실습코드는 github 참고)
딥러닝의 모델을 설정하고 구동하는 부분은 모두 **model**이라는 함수를 선언하며 시작이 됨.

model = Sequential() 딥러닝의 구조를 짜고 층을 설정하는 부분
model.compile() 정해진 모델을 컴퓨터가 알아들을 수 있게끔 컴파일 하는 부분
model.fit() 모델을 실제로 수행하는 부분

2. 입력층, 은닉층, 출력층

```
model = Sequential()
model.add(Dense(30, input_dim=17, activation='relu'))
model.add(Dense(1,activation='sigmoid'))
```

케라스에서 Sequential() 함수를 통해 층들이 쉽게 구현됨. Sequential() 함수를 model로 선언해 놓고, model.add()라는 라인을 추가하면 새로운 층이 만들어짐.

model.add()로 시작되는 라인 2개 -> 두 개의 층을 가진 모델 생성
맨 마지막 층 - 출력층(결과를 출력) 나머지 - 은닉층

각각의 층은 **Dense**라는 함수를 통해 구체적으로 구조가 결정됨

```
model.add(Dense(30, input_dim=17, activation='relu'))
```

30 - 30개의 노드 생성
input_dim - 입력 데이터에서 몇 개의 값을 가져올지 정하기
activation - 사용할 활성화 함수 정하기
(출력층에서는 활성화 함수로 sigmoid 함수 사용)

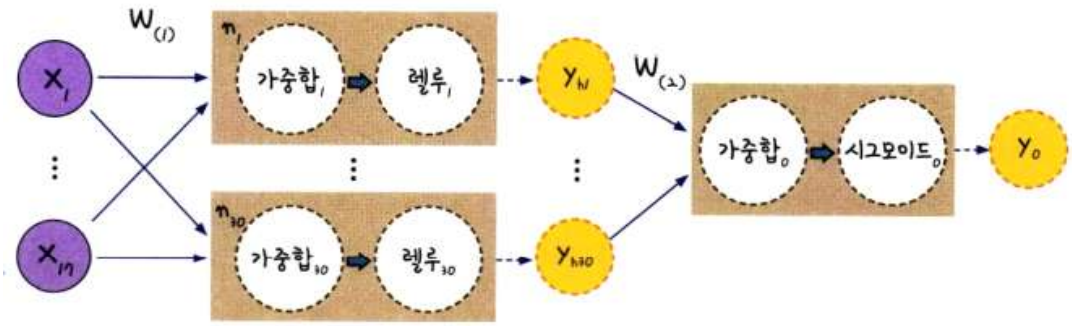
3. 모델 컴파일

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

loss='mean_squared_error' - 오차 함수로 평균 제곱 오차 함수를 사용
optimizer - 최적화 알고리즘 설정 (adam, sgd, rmsprop, adagrad 등이 있음)
metrics - 훈련을 모니터링 하기 위해 사용
(분류에서는 accuracy, 회귀에서는 mse, rmse, r2, mae, mspe, mape, msle 등이 있다.)

평균 제곱 오차 계열의 함수는 수렴하기까지 속도가 많이 걸린다는 단점이 있음. **교차 엔트로**

피 계열의 함수는 출력 값에 로그를 취해서, 오차가 커지면 수렴 속도가 빨라지고 오차가 작아지면 수렴 속도가 감소하게끔 만든 것이다.



(W는 각 층별 가중치(w)들의 집합)

metrics() 함수 - 모델이 컴파일될 때 모델 수행 결과를 나타내게끔 설정하는 부분
정확도를 측정하기 위해 사용되는 테스트 샘플을 학습 과정에서 제외시킴으로써 과적합 문제(overfitting)를 방지함.

4. 교차 엔트로피

교차 엔트로피 - 주로 분류 문제에서 많이 사용되는데, 특별히 예측 값이 참과 거짓 둘 중 하나인 형식일 때는 binary_crossentropy(이항 교차 엔트로피)를 쓴다.

표 10-1
대표적인 오차 함수

* 실제 값을 yt, 예측 값을 yo라고 가정할 때

평균 제곱 계열	mean_squared_error	평균 제곱 오차 계산: mean(square(yt - yo))
	mean_absolute_error	평균 절대 오차(실제 값과 예측 값 차이의 절댓값 평균) 계산: mean(abs(yt - yo))
	mean_absolute_percentage_error	평균 절대 백분율 오차(절댓값 오차를 절댓값으로 나눈 후 평균) 계산: mean(abs(yt - yo)/abs(yt)) (단, 분모 ≠ 0)
	mean_squared_logarithmic_error	평균 제곱 로그 오차(실제 값과 예측 값에 로그를 적용한 값의 차이를 제곱한 값의 평균) 계산: mean(square((log(yo) + 1) - (log(yt) + 1)))
교차 엔트로피 계열	categorical_crossentropy	범주형 교차 엔트로피(일반적인 분류)
	binary_crossentropy	이항 교차 엔트로피(두 개의 클래스 중에서 예측할 때)

5. 모델 실행하기

model.fit(X, Y, epochs=100, batch_size=10)

model.fit() - 컴파일 단계에서 정해진 환경을 주어진 데이터를 불러 실행시킬 때 사용되는 함수
X - 학습 데이터
Y - 레이블 데이터
epochs = 100 - 전체 데이터셋을 100번 반복학습시키기
batch_size = 10 - 10 개의 샘플로 가중치를 갱신하기

batch_size가 너무 크면 학습 속도가 느려지고, 너무 작으면 각 실행 값의 편차가 생겨서 전체 결괏값이 불안정해질 수 있다.

데이터 다루기

1. 딥러닝과 데이터

'빅데이터'는 분명히 머신러닝과 딥러닝으로 하여금 사람에 버금가는 판단과 지능을 가질 수 있게끔 했다. 하지만 데이터의 양보다 훨씬 중요한 것은, '필요한' 데이터가 얼마나 많은가이다.

머신러닝 프로젝트의 성공과 실패는 얼마나 좋은 데이터를 가지고 시작하느냐에 영향을 많이 받는다. 여기서 좋은 데이터란 내가 알아내고자 하는 정보를 잘 담고 있는 데이터를 말한다. 한쪽으로 치우치지 안혹, 불필요한 정보를 가지고 있지 않으며, 왜곡되지 않은 데이터여야 한다.
-> 데이터를 정제할 필요가 있음!

2. 피마 인디언 데이터 분석하기

피마 인디언은 1950년대까지만 해도 비만인 사람이 단 한 명도 없는 민족이었으나 지금은 미국의 기름진 패스트푸드 문화를 만나면서 전체 부족의 60%가 당뇨, 80%가 비만으로 고통받고 있다.

	정보 1	정보 2	정보 3	…	정보 8	당뇨병 여부
1번째 인디언	6	148	72	…	50	1
2번째 인디언	1	85	66	…	31	0
3번째 인디언	8	183	64	…	32	1
…	…	…	…	…	…	…
768번째 인디언	1	93	70	…	23	0

모델의 정확도를 향상시키기 위해서는 데이터의 추가 및 재가공이 필요할 수도 있음. 이러한 이유로 딥러닝의 구동에 앞서 데이터의 내용과 구조를 잘 파악하는 것이 중요!

3. pandas를 활용한 데이터 조사

(실습코드는 github 참고)

데이터의 크기가 커지고 정보량이 많아지면 데이터를 불러오고 내용을 파악할 수 있는 효과적인 방법이 필요하다. 이때 가장 유용한 방법이 데이터를 시각화해서 눈으로 직접 확인해 보는 것이다.

csv (comma separated values file) - 콤마(,)로 구분된 데이터들의 모음
헤더(header) - 데이터를 설명하는 한 줄

names 함수- 속성별 키워드 지정
head() 함수 - 상위 5개의 행 출력
describe() 함수 - 정보별 특징 출력

4. 데이터 가공하기

```
print(df[['pregnant','class']].groupby(['pregnant'],as_index=False).mean().sort_values(by='pregnant',ascending=True))
```

groupby() 함수를 사용해 pregnant 정보를 기준으로 하는 새 그룹 생성
as_index=False는 pregnant 정보 옆에 새로운 인덱스(index)를 만들어 줌
mean() 함수를 사용해 평균을 구하고 **sort_values() 함수**를 써서 pregnant 컬럼을 오름차순(ascending)으로 정리하게끔 설정

5. matplotlib를 이용해 그래프로 표현하기

matplotlib - 파이썬에서 그래프를 그릴 때 가장 많이 사용되는 라이브러리
seaborn - 정교한 그래프를 그리게끔 도와주는 라이브러리

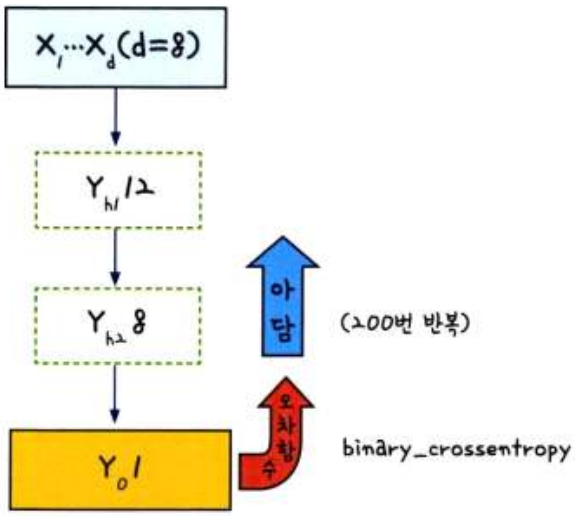
plt.figure() - 그래프의 크기 조절
heatmap() 함수 - 두 항목씩 짝을 지은 뒤 각각 어떤 패턴으로 변화하는지를 관찰하는 함수
(전혀 다른 패턴으로 변화하고 있으면 0을, 서로 비슷한 패턴을 변할수록 1에 가까운 값을 출력)
(vmax(색상의 밝기 조절), cmap(미리 정해진 matplotlib 색상의 설정값 불러오기) annot=True (각 셀에 숫자 표시))

숫자가 높을수록 그래프 셀이 밝은 색상으로 채워져 있음
plasma 항목이 class 항목과 가장 상관관계가 높다는 것을 알 수 있음
결과에 미치는 영향이 큰 항목을 발견하는 것이 '데이터 전처리 과정'의 한 예이다.

6. 피마 인디언의 당뇨병 예측 실행

random() 함수 - 임의의 숫자를 만들어 내는 것처럼 보여도 이는 컴퓨터 안에 미리 내장된 수많은 '랜덤 테이블' 중 하나를 불러내 그 표의 순서대로 숫자를 보여 주는 것이다. seed 값이 같으면 똑같은 랜덤 값을 출력함

딥러닝을 구현할 때는 일정한 결괏값을 얻기 위해 넘파이 seed 값과 텐서플로 seed 값을 모두 설정해야 한다.



둘 중 하나를 결정하는 이항 분류(binary classification) 문제이므로 오차 함수는 binary_crossentropy를 사용하고, 최적화 함수로 adam을 사용.
전체 샘플이 200번 반복해서 입력될 때까지 실험을 반복하고 한 번에 입력되는 입력 값을 10개로 하고 종합하도록 함.

다중 분류 문제 해결하기

1. 다중 분류 문제

아이리스(붓꽃)는 꽃잎의 모양과 길이에 따라 여러 가지 품종으로 나뉜다. 딥러닝을 사용하여 아이리스 품종을 예측해보자

	정보 1	정보 2	정보 3	정보 4	품종
1번째 아이리스	5.1	3.5	4.0	0.2	Iris-setosa
2번째 아이리스	4.9	3.0	1.4	0.2	Iris-setosa
3번째 아이리스	4.7	3.2	1.3	0.3	Iris-setosa
...	...	...	...	...	...
150번째 아이리스	5.9	3.0	5.1	1.8	Iris-virginica

클래스가 2개가 아니라 3개이므로 이 분류 문제는 **다중 분류(multi classification)**라고 한다. 여러 개의 답 중 하나를 고르는 분류 문제를 **다중 분류**라고 하고 이항 분류와는 접근 방식이 조금 다르다.

2. 상관도 그래프

(실습코드는 github 참고)

pairplot() 함수를 써서 데이터 전체를 한번에 보는 그래프를 출력(각 column별 데이터에 대한 상관관계나 분류적 특성을 확인시켜 줌)
(hue 인수에 카테고리 변수 이름을 지정하여 카테고리 값에 따라 색상을 다르게 할 수 있다.)

3. 원-핫 인코딩

데이터 안에 Iris-setosa, Iris-virginica 등 데이터 안에 **문자열이 포함**되어 있음. 이럴 때는 numpy보다는 pandas로 데이터를 불러와 X와 Y 값을 구분하는 것이 좋음.

LabelEncoder() 함수 (sklearn 라이브러리에 있음) - 문자열을 숫자로 바꿔 주려면 클래스 이름을 숫자 형태로 바꿔 주어야 함.
a r r a y (['I r i s -s e to s a ' , ' I r i s -v e r s ic o lo r' , 'I r i s -v i r g in ic a 1]) -> a r r a y ([1,2,3])

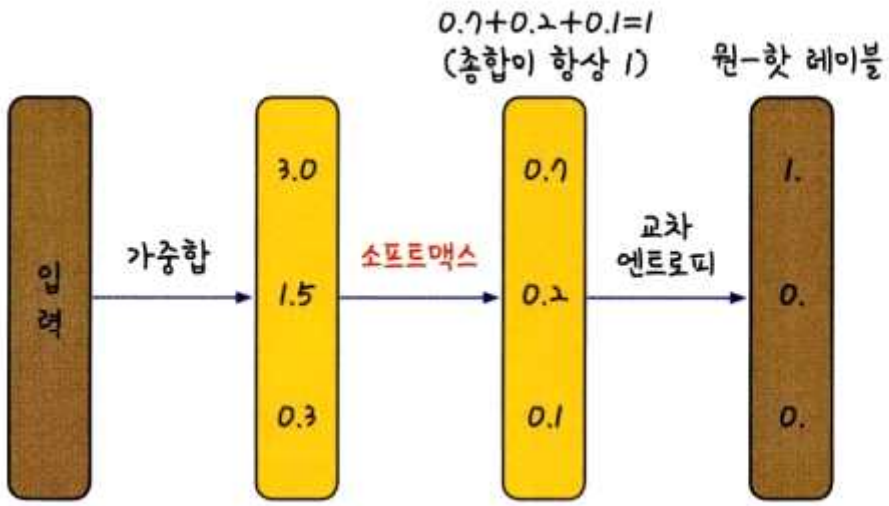
활성화 함수를 적용하려면 Y 값이 숫자 0과 1로 이루어져 있어야 함!!
-> **tf.keras.utils.categorical() 함수**를 적용
array([1,2,3]) -> array([[1.,0.,0.], [0., 1., 0.], [0.,0., 1.]])

원-핫 인코딩(one-hot-encoding) - 여러 개의 Y 값을 0과 1로만 이루어진 형태로 바꿔 주는 기법

4. 소프트맥스

최종 출력 값이 3개 중 하나여야 하므로 출력층에 해당하는 Dense의 노트 수를 3으로 설정
활성화 함수로 **소프트맥스(softmax)**를 사용

소프트맥스 - 총합이 1인 형태로 바꿔서 계산해 주는 함수 (합계가 1인 형태로 변환하면 큰 값이 두드러지게 나타나고 작은 값은 더 작아짐)
소프트맥스를 거친 값이 교차 엔트로피를 지나 [1., 0., 0.]으로 변하게 되면 우리가 원하는 원-핫 인코딩 값, 즉 하나만 1이고 나머지는 모두 0인 형태로 전환시킬 수 있다.



5. 아이리스 품종 예측 실행

다중 분류에 적절한 오차 함수인 categorical_crossentropy를 사용, 최적화 함수로 adam을 사용
전체 샘플이 50회 반복될 때까지 실험을 진행하되 한 번에 입력되는 값은 1개로 하고 종합하도록 함.

참고자료
https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=win0k&logNo=221603387293

https://sevillabk.github.io/Dense/

https://wooono.tistory.com/100

https://sevillabk.github.io/1-batch-epoch/

https://tykimos.github.io/2017/01/27/Keras_Talk/