HW9 Neural Network

짱짱조

[1] Neural Network, SOM의 개념

Neural Network는 생물의 뇌에서 사건을 예측하고 작동하는 방식을 모방한 알고리즘 기법이다. 생물에서 시냅스의 결합으로 네트워크를 형성한 뉴런으로 작동하는 것처럼, Neural Network에서는 인공 뉴런(노드)가 네트워크를 형성하여 학습을 통해 문제해결 능력을 가지게 된다.

Perceptron은 초기 형태의 인공신경망으로 시냅스와 유사하게 동작하여 다수의 입력으로부터 하나의 결과를 내보내는 알고리즘이다. Perceptron은 Single-Layer Perceptron과 Multi-Layer Perceptron으로 나눠진다. Single-Layer Perceptron은 값을 보내는 단계와 값을 받아서 출력하는 두 단계로 이루어진(input layer와 output layer) 간단한 형태의 모델이다. **Multi-Layer Perceptron(MLP)**은 input layer와 output layer 사이에 1개 이상의 hidden layer를 추가한 모델이다. MLP는 input들 사이의 복잡한 상호작용을 설명할 수 있고, non-linear한 문제도 설명할 수 있어 자주 쓰인다.

Radial Basis Function(RBF)는 MLP와 유사하나 hidden layer를 하나만 가지고 있다. 유클리드 거리 측정 데이터를 기반으로 동작하며 linear한 output layer를 가지고 있어 weight 계산에 용이하고 쉽게 update할 수 있어서 MLP보다 학습이 빠르다는 장점이 있다.

Self-Organizing Map(SOM)은 사람이 눈으로 볼 수 있는 저차원 격자에 고차원 데이터의 각 개체들이 대응하도록 neural network와 유사한 방식의 학습을 통해 군집을 도출해내는 기법이다. SOM은 node와 grid로 구성되며, 각 node에는 input data와 동일한 차원의 weight vector가 포함되어 있다. 모든 weight vector의 데이터 공간 상에서 유클리드 거리를 계산해 가장 최적의 노드인 BMU(Best Machine Unit)을 찾아 네트워크가 수렴할 때까지 샘플링을 사용해 여러 차례 반복적으로 조정한다. (Instance-Based Learning)

[2] 데이터 전처리

2-1. 데이터 정보

본 실습에서는 Kaggle "Oranges vs. Grapefruit"의 Citrus 데이터를 활용하여 실습을 진행했다. (출처: https://www.kaggle.com/joshmcadams/oranges-vs-grapefruit)

```
1 dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   name      10000 non-null  object
 1   diameter  10000 non-null  float64
 2   weight    10000 non-null  float64
 3   red       10000 non-null  int64
 4   green     10000 non-null  int64
 5   blue      10000 non-null  int64
dtypes: float64(2), int64(3), object(1)
memory usage: 468.9+ KB
```

```
data.describe()
```

	diameter	weight	red	green	blue
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	9.975685	175.050792	153.847800	76.010600	11.363200
std	1.947844	29.212119	10.432954	11.708433	9.061275
min	2.960000	86.760000	115.000000	31.000000	2.000000
25%	8.460000	152.220000	147.000000	68.000000	2.000000
50%	9.980000	174.985000	154.000000	76.000000	10.000000
75%	11.480000	197.722500	161.000000	84.000000	17.000000
max	16.450000	261.510000	192.000000	116.000000	56.000000

Sample Size는 10,000이고 사용할 변수에 대한 기초 통계량을 확인하였다. 변수에 대한 설명은 다음과 같다.

변수	설명	Type
Name	Label for the column. This should be either 'orange' or 'grapefruit'. **(Target변수, Classification)**	Object
Diameter	Diameter of the citrus fruit in centimeters.	Float
Weight	Diameter of the citrus fruit in grams.	Float
Red, Green, Blue	Average red/green/blue reading from an RGB scan. Values should be from 0 to 255.	Int

2-2. 데이터 전처리

학습을 진행하기 위해 features와 target을 분리하고 Train용 데이터와 Test용 데이터를 구분하기 위해 8:2로 나눴다. Train용 데이터에 대하여 Z-score standardization을 진행하였으며 Test용 데이터는 Train용 데이터의 기준으로 진행하였다. Target변수는 Categorical Data이었기에 One-Hot Encoding을 진행하였다.

```
[ ]  1 features = dataset.iloc[:,1:]
     2 target = pd.DataFrame([1 if each=="orange" else 0 for each in dataset["name"]],columns=["features"])
```

```
[ ]  1 from sklearn.model_selection import train_test_split
     2 train_features, test_features , train_labels, test_labels = train_test_split(
     3     features, target, test_size = 0.2, random_state = 2021)
```

```
[ ]  1 print(train_features.shape)
     2 print(train_target.shape)
     3 print(test_features.shape)
     4 print(test_target.shape)
```

```
(8000, 5)
(8000, 1)
(2000, 5)
(2000, 1)
```

```
[ ]  1 # Z-score Standardization
     2
     3 mean = train_features.mean(axis=0)
     4 train_features -= mean
     5 std = train_features.std(axis=0)
     6 train_features /= std
```

```
[ ]  1 # test data (Features)도 train의 기준으로 표준화
     2 test_features -= mean
     3 test_features /= std
```

```
[ ]  1 # One-hot Encoding
     2 from tensorflow.keras.utils import to_categorical
     3 train_labels_one_hot = to_categorical(train_labels)
     4 test_labels_one_hot = to_categorical(test_labels)
```

[3] Training 결과 해석

Hidden Layer의 수, Hidden Node의 수, Learning Rate를 조절해가며 다양한 모델에 대해 학습을 진행하였다. Early Stopping 방법을 사용해 각 모델마다 적절한 epoch 값을 결정했고 최적의 성능을 비교하였다. **각 모델의 대한 자세한 학습 결과와 시각화 결과는 첨부된 "HW9_Results.pdf"에 정리되어 있고 본 보고서에서는 학습 결과를 정리하고 해석하는 것을 중점적으로 다루고자 한다.**

3-0. Models

Training의 목적은 "Diameter, Weight, Red, Blue, Green" 5개의 features로 해당 과일이 "Orange인지 Grapefruit"인지 Classification하는 것이므로 모든 model의 Input Layer의 Node의 개수는 5, Output Layer의 Node의 개수는 2개(원 핫 인코딩을 진행했음)로 모두 동일하다. 각 Hidden Layer의 Activation 함수는 ReLU를 적용하였고, Output Layer의 경우 이진분류이므로 sigmoid를 적용했다. Optimizer의 경우 RMSProp을 적용했다.

본 보고서와 "HW9_Results.docx"에서 모델 네이밍을 다음과 같은 규칙으로 진행하였다.

Ex) "h4_16_8_8_8.h5"는 Hidden Layer의 수가 **4**개, 각 Hidden Layer의 node의 개수가 **16, 8, 8, 8**개

3-1. 모델 별 최적의 Epochs 확인

Neural Network에서 너무 많은 Epoch는 Overfitting을 일으킬 수 있다. 반대로 너무 적은 Epoch는 Underfitting을 일으킬 수 있다. **Early Stopping**을 활용하여 Epoch를 많이 돌린 후 특정 시점에서 멈추는 방법을 적용한다면 이러한 문제를 해결할 수 있다. 매 Epoch마다 Validation Loss를 측정하여 Training Loss와 Validation Loss가 둘 다 감소하다 Validation Loss가 증가하는 시점에서 모델의 훈련 종료를 제어하여 Overfitting을 방지할 수 있다. 본 실습에서는 Patience(성능이 증가하지 않는 epoch를 몇 번이나 허용할 것인가?)를 10으로 잡고 학습을 진행시켜 최적의 Epoch를 측정하였다.

Model	Batch Size	Learning Rate	Epochs
h4_16_8_8_8.h5	1024	0.01	105
h4_4_4_4_4.h5	1024	0.01	94
h4_8_8_8_8.h5	1024	0.01	124
h3_16_8_8.h5	1024	0.01	115
h3_8_8_8.h5	1024	0.01	127
h3_4_4_4.h5	1024	0.01	97
h2_16_8.h5	1024	0.01	113
h2_8_8.h5	1024	0.01	124
h2_4_4.h5	1024	0.01	131
h3_16_8_4_2.h5	1024	0.01	108
h3_16_8_4_2.h5	1024	0.1	63
h3_16_8_4_2.h5	1024	1	11
h3_16_8_4_2.h5 (Optimizer: rmsprop)	1024	0.01	108
h3_16_8_4_2.h5 (Optimizer: sgd)	1024	0.01	300 이상
h3_16_8_4_2.h5 (Optimizer: adam)	1024	0.01	95
h3_16_8_4_2.h5 (Optimizer: adamdelta)	1024	0.01	300 이상

3-2 모델 별 Weight 해석

Neural Network에서 Weight의 역할은 직전 layer에서 전달받은 데이터를 얼마나 중요하게 고려할 것인지 결정하는 것이다. 각 Hidden Unit에서 작은 Weight를 가진 feature는 상대적으로 덜 중요하다고 판단할 수 있다.

3-2-1. h2_4_4.h5

 # Input Layer(5) -> Hidden Layer 1(4)의 Weight

Hidden Layer1 에서 첫 번째 Unit 의 경우 weight(1)를, 두 번째 Unit 의 경우 diameter(0)를, 세 번째 Unit 의 경우 weight(1)를, 네 번째 Unit 의 경우 diameter(0), weight(1), green(4)을 중요하게 받아들인다고 볼 수 있다.

	0	1	2	3
0	-2.311687	1.026105	-1.992417	0.738000
1	1.360328	-2.114043	1.599887	0.670607
2	-0.014536	0.003111	0.003726	-0.161186
3	-0.002758	0.006744	0.002749	-0.440520
4	0.003648	-0.009521	-0.000626	0.378742

 # Hidden Layer 1(4) -> Hidden Layer 2(4)의 Weight

	0	1	2	3
0	-0.844050	1.668787	-0.675653	1.717823
1	2.010182	-1.263327	1.846895	-1.787518
2	-2.431802	3.676776	-1.519956	3.446792
3	0.832944	0.581164	1.026404	-0.001465

 # Hidden Layer 2(4) -> Output Layer(2)의 Weight

	0	1
0	1.660329	-1.389659
1	-1.662582	2.505645
2	1.400207	-1.535715
3	-4.009952	3.929888

3-2-2. h3_4_4_4.h5

 # Input Layer(5) -> Hidden Layer 1(4)의 Weight

Hidden Layer1 에서 첫 번째 Unit 의 경우 weight(1)를, 두 번째 Unit 의 경우 diameter(0)를, 세 번째 Unit 의 경우 weight(1)를, 네 번째 Unit 의 경우 weight(1), green(4)을 중요하게 받아들인다고 볼 수 있다.

	0	1	2	3
0	-1.343502	1.122160	-1.354311	-0.979315
1	1.253220	-0.618739	1.319098	1.267841
2	-0.029152	-0.241149	0.009239	-0.188058
3	-0.030575	-0.362372	0.035973	-0.317954
4	0.053128	0.369994	-0.045570	0.338054

 # Hidden Layer 1(4) -> Hidden Layer 2(4)의 Weight

	0	1	2	3
0	-0.140581	1.607372	0.491627	1.496662
1	-0.513783	-0.862095	-0.007702	-0.670205
2	-0.418062	1.507962	0.372017	1.067612
3	0.280351	0.657821	1.375035	0.674426

 # Hidden Layer 2(4) -> Hidden Layer 3(4)의 Weight

	0	1	2	3
0	0.358330	-0.426199	0.470362	0.152566
1	1.579030	-1.356597	-0.797640	1.608131
2	0.147369	1.216140	-0.430764	0.359823
3	2.118449	-1.119185	-0.527615	1.059473

 # Hidden Layer 3(4) -> Output Layer(2)의 Weight

	0	1
0	-1.208644	1.664445
1	0.348237	-1.588245
2	0.117734	0.466276
3	-1.244312	1.907261

3-3 성능 비교

다음은 Batch size가 1024, Learning Rate가 0.1로 동일한 모델에 대하여 Test_Accuracy와 R-Squared를 나타낸 표이다. 모델 'h4_16_8_8_8.h5'가 Test Accuracy 0.9815, R-Squared 0.937로 가장 좋은 성능을 보였다. Hidden Layer의 개수가 동일 할 때, Layer의 node 개수에 따른 성능 변화는 특별한 경향성을 보이지 않았다. Hidden Layer의 개수가 다른 모델들 간의 성능 역시 특별한 경향성을 보이지 않았다. 평균적으로 가장 높은 성능을 보인 경우는 Hidden Layer의 개수가 2개인 경우였다.

Model	Test_Accuracy	R-Squared
h4_16_8_8_8.h5	0.9815000295639038	0.937
h4_4_4_4_4.h5	0.9775000214576721	0.917
h4_8_8_8_8.h5	0.9775000214576721	0.925
h3_16_8_8.h5	0.9775000214576721	0.926
h3_8_8_8.h5	0.9785000085830688	0.935
h3_4_4_4.h5	0.9714999794960022	0.903
h2_16_8.h5	0.9810000061988831	0.934
h2_8_8.h5	0.9760000109672546	0.923
h2_4_4.h5	0.9785000085830688	0.936