DB 응용 1 차 과제

DL4J 수행

과목명 : 데이터베이스 응용

학번 : 12131579

이름 : 이진아

연락처 : 010-5282-1903

e-mail : wlsdk7700@naver.com

1. 개요

- **목적**

Deep Learning 을 이해하고 DL4J 를 통해 직접 구현해 보면서 원리를 파악한다.

- **요구사항**

1) Hidden layer 의 수 변화 시켜 실행(1 개, 2 개, 3 개)
2) Hidden layer 의 input, output class 수를 변화 시켜 실행(5 개, 10 개, 20 개)
3) Train data 의 input 수를 다르게 실행(200 개, 2000 개, 20000 개)
4) Train data 의 input 들 중 output 의 값이 0 과 4 인 비중을 높게 하여 실행
5) Tanh -> ReLu 로 바꾸어 실행

- **개발 환경**

1) OS : Windows 10 / 64bit
2) DL4J
3) Intellij
4) 언어 : Java 사용

2. 실행결과

1) Hidden layer 수에 따른 결과(Train data : 200 개, Tanh 사용)

- Hidden layer 의 input, output class : 5 개

========== Scores ==========		========== Scores ==========		========== Scores ==========	
Accuracy:	0.6	Accuracy:	0.6	Accuracy:	0.6
Precision:	0.6667	Precision:	0.6667	Precision:	0.6667
Recall:	0.6	Recall:	0.6	Recall:	0.6
F1 Score:	0.6316	F1 Score:	0.6316	F1 Score:	0.6316
==========		==========		==========	

1 개 2 개 3 개

➔ 5 개인 경우 layer 의 수 증가에 따른 accuracy 의 차이를 보이지 않음

Layer 1 개

```
o.d.o.l.ScoreIterationListener - Score at iteration 0 is 1.7210732774293698
o.d.o.l.ScoreIterationListener - Score at iteration 100 is 0.610956823085106
o.d.o.l.ScoreIterationListener - Score at iteration 200 is 0.4087442797855661
o.d.o.l.ScoreIterationListener - Score at iteration 300 is 0.3089547008680562
o.d.o.l.ScoreIterationListener - Score at iteration 400 is 0.260792394118619
o.d.o.l.ScoreIterationListener - Score at iteration 500 is 0.23346363731725484
o.d.o.l.ScoreIterationListener - Score at iteration 600 is 0.21571025977438013
o.d.o.l.ScoreIterationListener - Score at iteration 700 is 0.2031069040842352
o.d.o.l.ScoreIterationListener - Score at iteration 800 is 0.19361921524829093
o.d.o.l.ScoreIterationListener - Score at iteration 900 is 0.1861832322425438
```

Layer 2 개

```
o.d.o.l.ScoreIterationListener - Score at iteration 0 is 1.6227118721684834
o.d.o.l.ScoreIterationListener - Score at iteration 100 is 0.5641799363528208
o.d.o.l.ScoreIterationListener - Score at iteration 200 is 0.3712784730927723
o.d.o.l.ScoreIterationListener - Score at iteration 300 is 0.28955080631303576
o.d.o.l.ScoreIterationListener - Score at iteration 400 is 0.24683910201476444
o.d.o.l.ScoreIterationListener - Score at iteration 500 is 0.22038227246345335
o.d.o.l.ScoreIterationListener - Score at iteration 600 is 0.20224228513380496
o.d.o.l.ScoreIterationListener - Score at iteration 700 is 0.18909911973771443
o.d.o.l.ScoreIterationListener - Score at iteration 800 is 0.17926350787700263
o.d.o.l.ScoreIterationListener - Score at iteration 900 is 0.17174325635329368
```

Layer 3 개

```
o.d.o.l.ScoreIterationListener - Score at iteration 0 is 1.763463566039655
o.d.o.l.ScoreIterationListener - Score at iteration 100 is 0.6356563663447
o.d.o.l.ScoreIterationListener - Score at iteration 200 is 0.3409188134045126
o.d.o.l.ScoreIterationListener - Score at iteration 300 is 0.24791216157291548
o.d.o.l.ScoreIterationListener - Score at iteration 400 is 0.20896751030066366
o.d.o.l.ScoreIterationListener - Score at iteration 500 is 0.19207595856906068
o.d.o.l.ScoreIterationListener - Score at iteration 600 is 0.17833450783265153
o.d.o.l.ScoreIterationListener - Score at iteration 700 is 0.16944086277651288
o.d.o.l.ScoreIterationListener - Score at iteration 800 is 0.16140755235855042
o.d.o.l.ScoreIterationListener - Score at iteration 900 is 0.15421342815127
```

➔ 그러나 layer 의 수에 따라 반복 시 오차가 다르게 줄어듦. Layer 가 3 개인 경우 마지막 반복 후 오차가 가장 작게 나타남.

- Hidden layer 의 input, output class : 10 개

========== Scores ==========		========== Scores ==========		========== Scores ==========	
Accuracy:	0.6	Accuracy:	0.8	Accuracy:	0.6
Precision:	0.6667	Precision:	0.875	Precision:	0.6667
Recall:	0.6	Recall:	0.8	Recall:	0.6
F1 Score:	0.6316	F1 Score:	0.8358	F1 Score:	0.6316
==============================		==============================		==============================	
1 개		2 개		3 개	

- Hidden layer 의 input, output class : 20 개

========== Scores ==========		========== Scores ==========		========== Scores ==========	
Accuracy:	0.8	Accuracy:	0.99	Accuracy:	0.8
Precision:	0.875	Precision:	0.9905	Precision:	0.875
Recall:	0.8	Recall:	0.99	Recall:	0.8
F1 Score:	0.8358	F1 Score:	0.9902	F1 Score:	0.8358
				------------------------------	
1 개		2 개		3 개	

➔ 10, 20 개인 경우 layer 수가 2 개일 때 accuracy 가 가장 높게 나타남. 오히려 layer 수가 가장 많은 3 개인 경우 accuracy 가 떨어지는 현상 발생

⇨ 결과 분석 : layer 수가 2 개인 경우 가장 높은 accuracy 가 발생함.

2) Hidden layer 의 input, output class 차이에 따른 결과(Train data : 200 개, Tanh 사용)

- Hidden layer 수 : 1 개, 3 개

Scores		Scores		Scores	
Accuracy:	0.6	Accuracy:	0.6	Accuracy:	0.8
Precision:	0.6667	Precision:	0.6667	Precision:	0.875
Recall:	0.6	Recall:	0.6	Recall:	0.8
F1 Score:	0.6316	F1 Score:	0.6316	F1 Score:	0.8358
5 개		10 개		20 개	

➔ Hidden layer 수가 1, 3 개 경우 둘다 5 개와 10 개는 같은 accuracy 가 발생하였으며 20 개인 경우 가장 높은 accuracy 가 나타남

- Hidden layer 수 : 2 개

Scores		Scores		Scores	
Accuracy:	0.6	Accuracy:	0.8	Accuracy:	0.99
Precision:	0.6667	Precision:	0.875	Precision:	0.9905
Recall:	0.6	Recall:	0.8	Recall:	0.99
F1 Score:	0.6316	F1 Score:	0.8358	F1 Score:	0.9902
5 개		10 개		20 개	

➔ Hidden layer 수가 2 개인 경우 10 개와 20 개는 hidden layer 수가 1,3 인 경우보다 증가했으나 5 개인 경우는 동일하였음

Hidden layer 수가 2 개인 경우도 20 개인 경우 accuracy 가 가장 높게 나타남

⇨ 결과 분석 : hidden layer 의 input, output class 의 개수를 20 개로 한 경우 가장 높은 accuracy 가 발생함.

3) Train data 의 개수를 다르게 실행한 결과(0,4 의 결과 비중이 적은 경우)

- Hidden layer 수 = 1 개, hidden layer 의 input, output classes 수 = 5 개

Scores		Scores		Scores	
Accuracy:	0.6	Accuracy:	0.6	Accuracy:	0.6
Precision:	0.6667	Precision:	0.6667	Precision:	0.6667
Recall:	0.6	Recall:	0.6	Recall:	0.6
F1 Score:	0.6316	F1 Score:	0.6316	F1 Score:	0.6316
200 개		2000 개		20000 개	

➔ 0 과 4 의 비중이 적을 경우 train data 수를 늘려도 accuracy 가 증가하지 않음

4) Train data 의 개수를 다르게 실행한 결과(0,4 의 결과 비중을 크게 수행)

- Hidden layer 수 = 1 개, hidden layer 의 input, output classes 수 = 5 개

Scores	
Accuracy:	1
Precision:	1
Recall:	1
F1 Score:	1

➔ 0,4 의 비중을 크게 둘 시 모든 경우 accuracy 가 1 이 발생함.

5) ReLu 시행(train data = 200 개)

- Hidden layer 수 = 1 개, hidden layer 의 input, output class 수를 다르게함

Scores		Scores		Scores	
Accuracy:	1	Accuracy:	0.92	Accuracy:	0.97
Precision:	1	Precision:	0.9429	Precision:	0.9739
Recall:	1	Recall:	0.92	Recall:	0.97
F1 Score:	1	F1 Score:	0.9313	F1 Score:	0.972
5 개		10 개		20 개	

➔ 5 개, 10 개, 20 개 모두 tanh 로 시행 했을 때 보다 accuracy 가 증가함.

⇨ 결과 분석 : Tanh 로 시행 했을 시 20 개인 경우가 가장 높게 accuracy 가 발생했으나 ReLu 로 시행 했을 시 5 개인 경우가 가장 높은 accuracy 가 나타남

3. 결과 정리

1) Hidden layer 의 수 변화 시켜 실행(1 개, 2 개, 3 개)
   ➔ Hidden layer 의 수가 2 개인 경우 가장 높은 accuracy 발생
2) Hidden layer 의 input, output class 수를 변화 시켜 실행(5 개, 10 개, 20 개)
   ➔ Input, output class 수가 20 개인 경우 가장 높은 accuracy 발생
3) Train data 의 input 수를 다르게 실행(200 개, 2000 개, 20000 개)
4) Train data 의 input 들 중 output 의 값이 0 과 4 인 비중을 높게 하여 실행
   ➔ Train data 의 input 수와 accuracy 는 관계가 없었으며 0 과 4 의 비중과 관련이 있었음. 0 과 4 의 비중을 높이면 train data 의 수와 관련 없이 accuracy 가 1 이 발생
5) Tanh -> ReLu 로 바꾸어 실행
   ➔ ReLu 로 실행시 Tanh 의 결과보다 높은 accuracy 가 발생하였으며 input, output classes 가 5 인 경우 가장 높은 accuracy 가 발생함.

- **가장 높은 accuracy 가 발생한 경우**
  - Train data 의 0 과 4 의 비중을 높여 실행한 경우
  - ReLu 방식을 사용하고 hidden layer 의 input, output class 수가 5 인 경우
  - Train data 의 0 과 4 비중이 낮을 시 tanh 에서는 hidden layer 2 개, hidden layer 의 input, output class 수를 20 개로 둔 경우 가장 높게 나타남