**Operating System   
과제 보고서**

|  |  |
| --- | --- |
| 과제 명 | *Machine Problem 1*  *(Introduction to Programming in C on Unix)* |



|  |  |
| --- | --- |
| **이 름** | 이 삭 |
| **학 번** | 12124913 |

**1. Data.**

**a. Make data. ( training\_data.csv, test\_data.csv, eval\_data.csv(test.csv) )**

엑셀에서 INT(RAND()\*20)을 이용해 무작위값으로 데이터를 생성하였습니다.

예시) INT(RAND()\*20+1) → 1~20 : Label 0 데이터

INT(RAND()\*20+1) +20 → 21~40 : Label 1 데이터

각 Label 당 15개씩 생성하여 총 training 데이터 수는 75개이며, 동일하게 모델의 정확도를 테스트하기 위한 데이터(총 10개)를 만들었습니다. 최종적인 정확도 테스트는 제공된 데이터인 eval\_data.csv(기존 test.csv)로 진행하였습니다.

**b. Normalization.**

학습시킬 때 더 빠르고 정확한 결과를 내기 위해 모든 feature 데이터에 normalization을 적용하였고, 각 라벨 당 feature 데이터가 같아지는 것을 방지하기 위해 기존 normalization 식에 y\_label 값을 더해주었습니다.



따라서 feature data는 0~1 (label:0) / 1~2 (label:0) / 2~3 (label:0) / 3~4 (label:0) 과 같이 됩니다.

**2. Model**

모델을 만들기 위해 tf.contrib.learn.DNNClassifier를 사용하였습니다.

**a. Parameters.**

-**Shape of layers** : [10, 20, 10]

: 총 레이어 수는 3개, 각 히든 레이어의 노드 수는 10, 20, 10입니다. 깊게 쌓을수록 vanishing gradient problem이 있을 수 있으므로 위와 같이 설정하였습니다.

-**Optimizer** : AdagradOptimizer

: Optimizer는 여러 개를 바꿔가며 사용해본 결과 Adagrad가 좋은 효과를 나타내었습니다.

-**Learning rate** : 0.1

: Learning rate 또한 0.1부터 0.01 까지 바꿔본 결과 0.1일 때 좋은 성능을 보였습니다.

-**iteration** : 1000

-**batch\_size** : 8

: iteration을 1000으로 맞추기 위해 batch\_size를 8로 적용하였고, 좋은 결과를 나타내었습니다.

**3. Result**

제공된 데이터(eval\_data)로 최종 테스트 해본 결과 1.0의 정확도를 나타내었습니다.