인공지능

MNIST성능향상

보고서



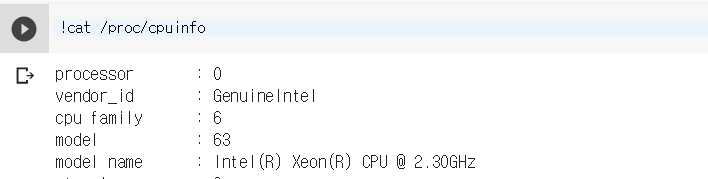
2014920009

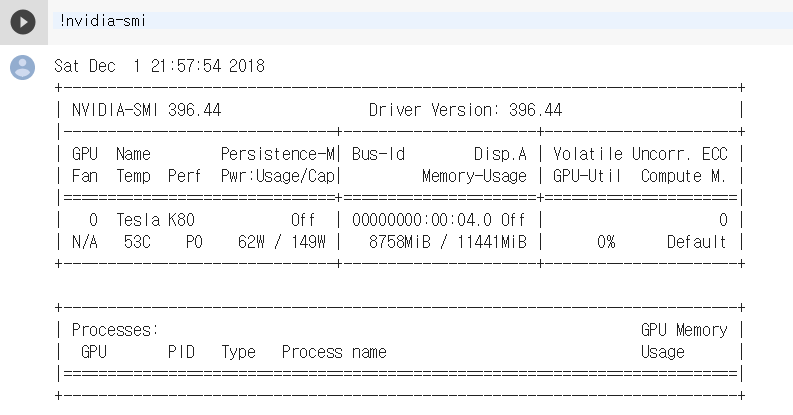
컴퓨터과학부

김현우

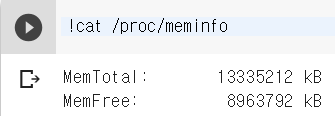
개요

1. 실행환경과 단층 MNIST 분류
   1. 실행환경
   2. SoftMax
2. 성능향상
   1. NeuralNetwork
      1. Hidden 1, node 1024
      2. Hidden 2, node 512\*2, initialization : randomnormal
      3. Hideen 2, node 512\*2, initialization : zeros
   2. Initialization
      1. Hidden 1, node 1024, initialization : Xavier Initialization
      2. Hidden 1, node 1024, initialization : He Initialization
   3. DeepNN
      1. Hidden 5, node 512\*5, initialization : Xavier Initialization
   4. Dropout
      1. Hidden 5, node 512\*5, initialization : Xavier Initialization, KeepProbability : 70%(0.7)
      2. Hidden 5, node 512\*5, initialization : Xavier Initialization, KeepProbability: 50%(0.5)
   5. CNN
      1. Single CNN
      2. Double Layer CNN
   6. CNN\_Data\_Augmentation\_Ensembling
      1. CNN\_Data\_Augmentation
      2. CNN\_Data\_Ensembling
3. 결론
4. 첨부파일
5. Reference
6. 실행환경과 단층 MNIST 분류
   1. 실행환경
      1. 저번과제에서 Colab은 파일저장에 불편함이 있다는 것을 알게 되었고, Jupyter나 vscode를 활용하는 방법을 고려하였다. 하지만 이번 과제는 99%이상의 정확도를 요구하는 만큼 많은 컴퓨팅파워가 필요하다고 판단했다.
      2. 내가 가지고 있는 개인컴퓨터로는 한번 테스트하는데에 많은 시간이 소요될 것을 알고 있었고, Colab의 실행환경을 확인 후, Colab으로 테스트하기로 했다.
         * Colab의 실행환경

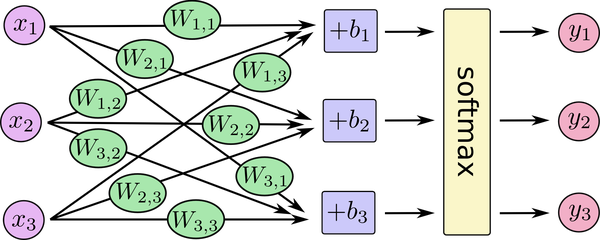
CPU : Intel Xeon 2.23GHz



GPU : NVIDIA Tesla K80



RAM : 13G

* 머신러닝에 적합
  1. Softmax
     + - Softmax를 이용한 MNIST 분류는 교재를 기반으로 작성하였다. 이를 기반으로 확장해 나갈 것이다.
       - Softmax를 이용한 단항분류기로 분류한 결과 정확도는 92.1%를 기록했다.
       - 목표는 99%↑이다.
       - Accuracy를 향상시키기 위해 시도해볼 수 있는 방법으로는 다음이 있다.

1. Node 수 증가
2. Layer 수 증가
3. Initialization방법 조정
4. Deep and Wide NN 구성
5. Dropout 사용
6. Single Layer CNN
7. Double Layer CNN
8. CNN + Data Augmentation
9. CNN + Data Augmentation + Ensemble
10. 성능향상

1024nodes

* 1. NeuralNetwork

Hidden

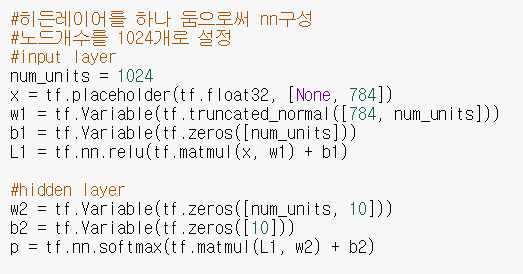
* + 1. Hidden 1, node 1024

Output

Input

...

…



<노드의 개수를 1024개로 설정하는 부분>

* 노드를 1024개로 설정하였다.
* 단항분류를 했을 때에 92.1%의 정확도를 보인 것에 비해 많이 향상된 것을 확인할 수 있다.
* Hidden 1, node 1024 사용 시 다음과 같이 97.2%의 Accuracy를 보임이 확인되었다.



* 목표는 99%↑이기에 다른 방법을 시도해볼 필요가 있다.

Hidden2

Output

...

…

* + 1. Hidden 2, node 512\*2, initialization : randomnormal

512nodes

512nodes

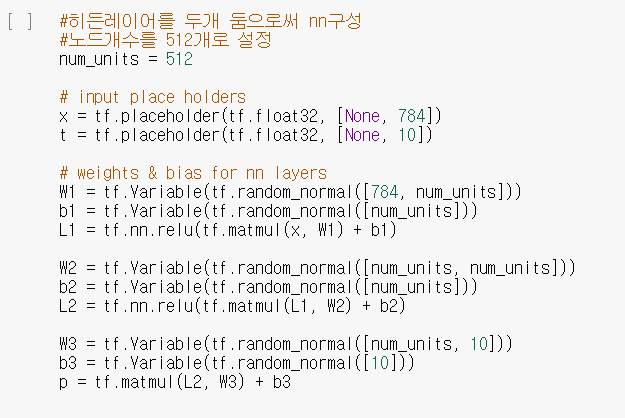
Hidden1

Input

...

Initialization : random\_normal

…



* 레이어 구성 부분

* + - * 위 그림과 같이 히든레이어를 두개 둠으로써 NN을 구성하였다 노드는 각각512개로 하였다.
      * initialize함수를 random\_normal로 하였다.
      * Accuracy는 95.9%로 크게 개선된 모습은 보이지않는다.
      * 역시 99%에 미치지 못하기 때문에 다른 방법을 시도해보았다.
    1. Hideen 2, node 512\*2, initialization : zeros

Initialization : zeros

Input

Hidden2

Output

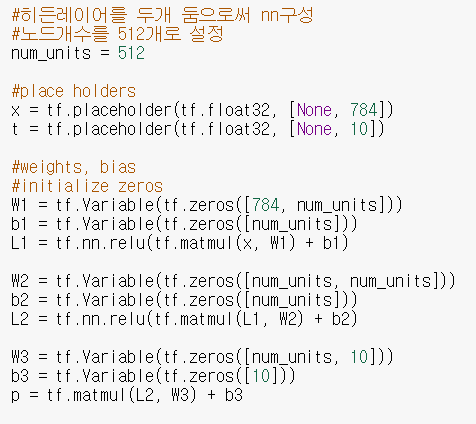
...

…

...

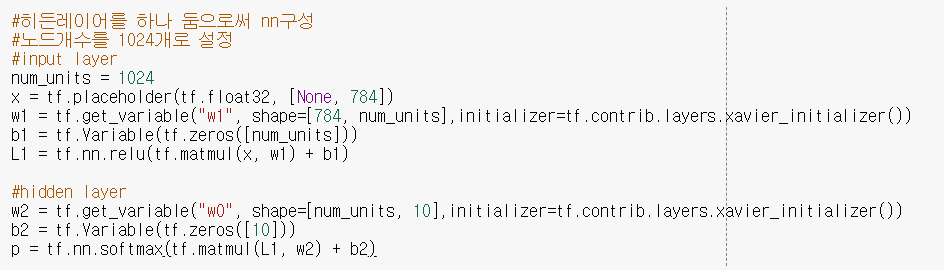
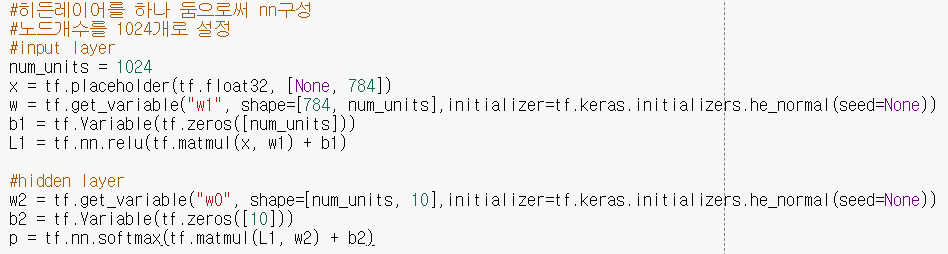
…

Hidden1



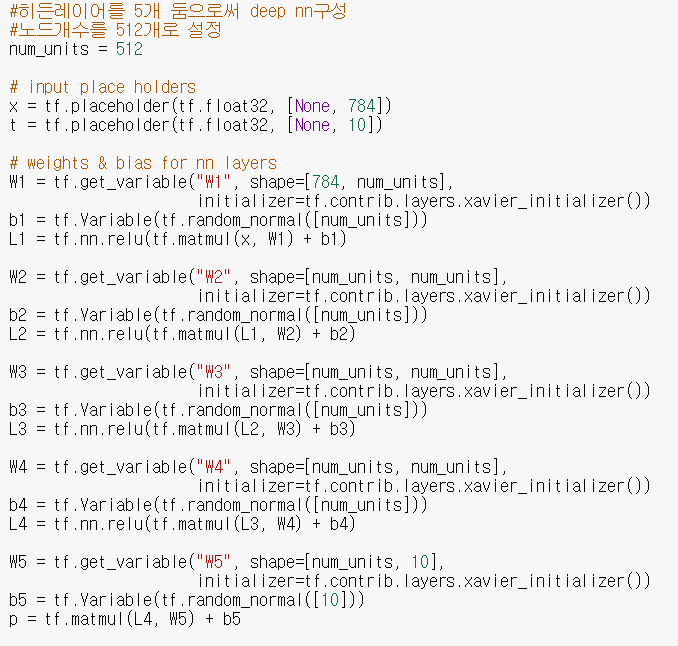
* 그림에서 확인할 수 있듯이 앞의 방법과 같지만 initialize함수를 zeros로 하였다.
* 아래와 같이 Accuracy는 11.3%로 초기화함수에 따라 많은 차이가 난다는 것을 알게 되었다.



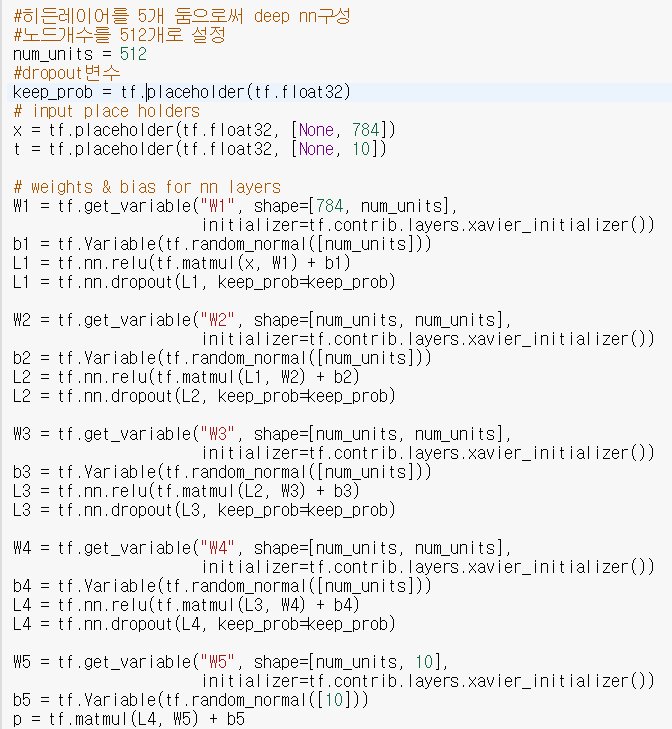
* 1. Initialization
     + - A-1, A-2를 테스트하며 initialization의 중요성을 깨달았고, 여러 초기화 방법들을 찾아보았다.
       - Initialization 방법으로는 주로 He initialization과 Xavier initialization이 사용 된다고 한다.
     1. Hidden 1, node 1024, initialization : Xavier Initialization 참고1)
        + Relu등장 후에 Glorot이 2010년에 제안한 방법으로 vanishing gradient 문제를 해결하기 위해 만들었다고 한다.
        + Input과 output neuron의 수를 기반으로 초기화를 한다는 특징이 있다.
        + Initialization weight가 0일경우 각 레이어를 통과하면서 signal이 0이 되어버리고 너무크다면 signal도 너무 커져버린다는 점을 해결한다.
        + 구체적인 방법은 다음과 같다.
          - 입력값과 출력값 사이의 난수를 선택해서 입력값의 제곱근으로 나눈다.
          - Random(in, out)/sqrt(in)
        + Sigmoid, tanh같은 비선형 함수에서 효과적인 결과를 보여주지만, Relu에서 사용 시 출력 값이 0으로 수렴하게 되는 현상이 발생한다.
        + Xavier\_initializer()를 사용하는 부분
        + 초기화 방법만 바꿨을 뿐인데 95% -> 98%의 성능개선이 이루어 진 것을 확인할 수 있다.
     2. Hidden 1, node 1024, initialization : He Initialization 참고2)
        + Glorot과 유사하지만 Neuron의 ousize를 고려하지 않는다.
        + Relu가 0이하의 신호를 제거하기 때문에 분산을 두배로 주어 분산을 유지한다는 목적이 있다.
        + 자세한 방법은 다음과 같다.
          - 입력값을 반으로 나눈 제곱근을 사용한다. 분모가 작아지기 때문에 Xavier보다 넓은 범위의 난수가 생성된다.
          - Random(in,out)/sqrt(in/2)
        + initializers.he\_normal(seed=None)을 사용하여 초기화하는 부분



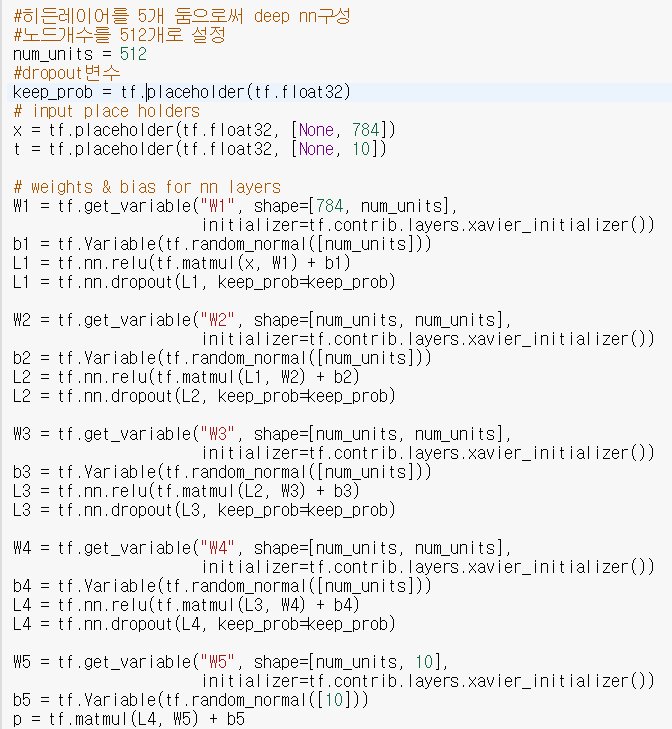
* + - * Accuracy가 98%로 역시 개선되긴 하였으나, 99%에 도달하지 못했으므로 다른 방법을 시도한다.
      * He initializtion과 Xavier initialization 둘 다 높은 정확도 개선률을 보이지만, 데이터양이 적은 탓인지 두 초기화방법의 정확도에는 큰 차이가 없었다.
      * 앞으로 시도할 방법에서는 Xavier initialization을 사용할 것이다.
  1. DeepNN
     1. Hidden 5, node 512\*5, initialization : Xavier Initialization

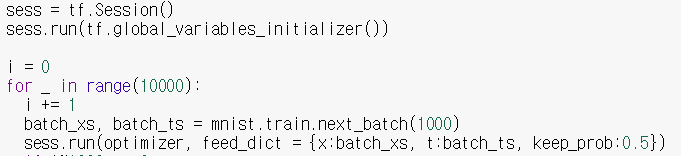
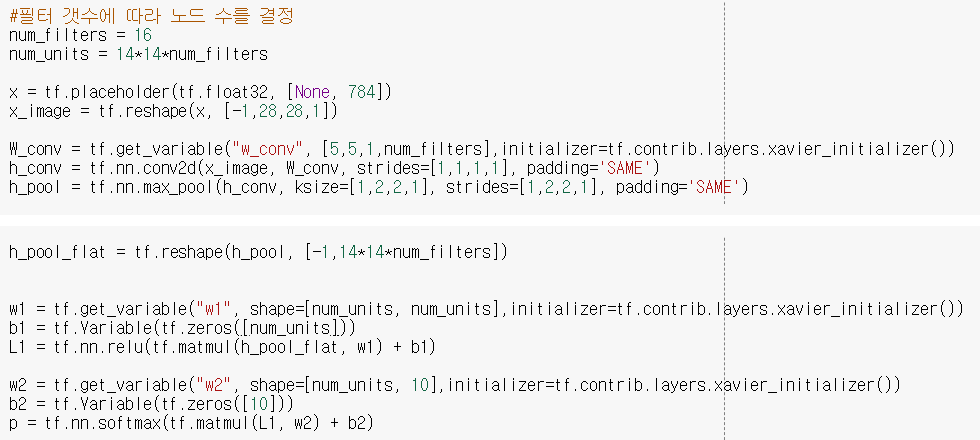


* + - * 5개의 레이어와 각512개의 노드로 DeepNN을 구성하는 부분
      * Accuracy가 98.3%로 약간 개선된 것을 확인할 수 있다.
  1. Dropout
     1. Hidden 5, node 512\*5, initialization : Xavier Initialization, KeepProbability : 70%(0.7)
        + Dropout이란 데이터를 KeepProbability에 따라 포함시키거나 제외시켜서 학습하는 방법이다.
        + KeepProbability를 70%로 설정후 테스트해보았다.



* + - * Keep\_prob 을 선언



* + - * Keep\_prob을 0.7로 설정
      * Next\_batch 또한 1000으로 조정하였다.
      * 테스트 결과 Accuracy가 97.9%로 그다지 개선되지 않았음을 알 수 있다.
    1. Hidden 5, node 512\*5, initialization : Xavier Initialization, KeepProbability: 50%(0.5)
       - Keep\_prob를 0.5로 설정
       - 테스트 결과 Accuracy가 97.4%로 개선률이 낮다는 것을 알 수 있다.
       - 데이터 수가 부족한 탓인지 Drop\_out은 기존 방식에 비해 개선률이 낮음을 알 수 있었다.
  1. CNN
     1. Single CNN
        + CNN을 구성하는 부분
        + Conv-> pooling->flatting->Layer1(ReLU)->Layer2(Softmax)
        + Accuracy는 98.5%로 상당히 개선된 모습을 확인할 수 있었다.
     2. Double Layer CNN
        + CNN을 구성하는 부분
        + Dropout도 사용하였으며, 전체적인 이미지는 다음과 같다.
        + Keep\_prob = 0.5로 설정
        + Conv1->Pooling->Conv2->Pooling->

Flatting->Layer1(ReLU)->Dropout(P=0.5)->Layer2(Softmax)



* + - * Accuracy가 드디어 99.2%로 99%이상을 달성하였다.
      * 최대한 높은 Accuracy를 얻기 위해 DataAugmentation과 Ensembling을 추가하여 테스트해보겠다.
  1. CNN\_Data\_Augmentation\_Ensembling
     1. CNN\_Data\_Augmentation
        + Data Augmentation이란 한정된 Data를 조정을 통해 늘리는 것이다.
        + 이를 위해 keras.preprocessing.image 로부터 ImageGenerator을 import하여 사용하였다.
        + ImageGenerator변수는 다음과 같이 설정하였다.
        + datagen = ImageDataGenerator(

rotation\_range = 5, #회전각도

zoom\_range = 0.2, #확대/축소범위

width\_shift\_range=0.2, #수평이동범위

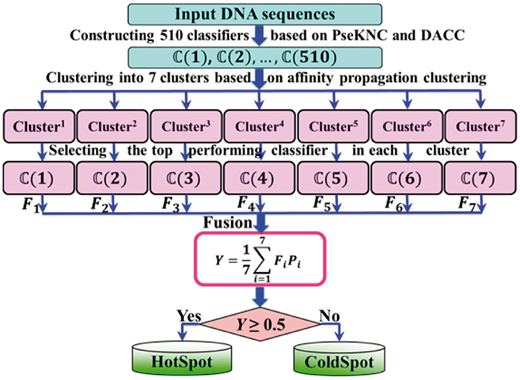
height\_shift\_range=0.2) #높낮이 이동범위

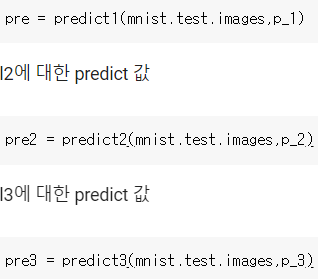
* + - * 데이터를 mnist에서 55000개 불러온 후 55000\*1= 55000개를 확장하였다.
      * 총데이터 110000개사용
      * 구조 : Double CNN에서 데이터셋을 확장시킴

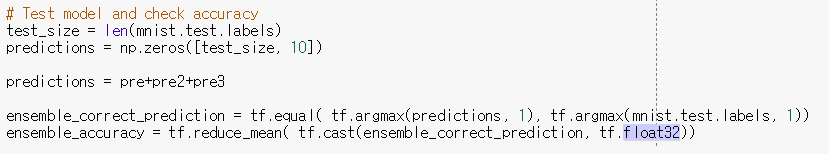


* + - * 확장시킨 데이터 중 100개의 데이터
      * Data Augmentation활용 시 확장은 정상적으로 이루어 졌으나, 정확도는 어떤이유인지 모르겠지만 감소하였다.

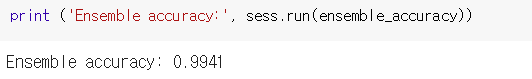


* + - * 이를 해결하기 위해 Image generator의 값을 조정해보고, 총 데이터 셋을 조정해보기도 하고, minibatch를 구현하여 보기도 하였다. 하지만 모두 accuracy가 감소하는 결과를 얻게 되었다.
      * 데이터셋을 더 확장해보려했으나, 메모리 문제로 더 확장하지는 못하였다.
    1. CNN\_Data\_Ensembling
       - Augmentation과 Ensembling을 함께 시도하려 하였으나, Augmentation의 결과가 좋지않아 Ensembling만 시도해 보기로 하였다.
       - Ensembling이란 데이터 모델을 여러 개 만들고 그 결과들을 합치는 간단한 방법이다.
       - 예를 들면, 여러 전문가들이 어떤 연구에 대해 각자 조사한 후 그 결과를 마지막에 합친다고 할 수 있을 것이다.
       - Ensemble의 도식화 ->
       - 클래스를 이용하여 작성하려 했으나, 잘되지 않아서 model을 각각 코딩하였다.
       - Double Layer CNN모델 세개를 사용하였다.
       - Voting 과 같이 각각의 모델이 동일한 투표권을 가지고 있는 것 같이 의견을 합칠 수 있다는 특징이 있다.



* + - * 각 모델에 대한 predict 값을 얻는 부분
      * Ensemble model의 값을 합하여 전체모델의 예측값을 구하는 부분
      * 모델1의 정확도



* + - * 모델2의 정확도
      * 모델3의 정확도
      * Ensemble 결과
      * 놀랍게도 각각의 predict값을 합친 결과 99.41%라는 결과를 얻을 수 있었다. 모델을 더 늘려보면 정확도를 더 올릴 수 있을 것 같지만, 메모리 부족으로 더 늘릴 수 없었다.
      * Ensemble을 통하여 가장 높은 정확도인 99.41%를 얻었으며, 모델 테스트를 끝냈다.

1. 결론

- 머신러닝에서 데이터를 학습시키는 방법에는 무수히 많은 방법이 있다. 그 안에서도 무엇이 어떤 상황에 적절한가 하는 정답은 없다. 예를 들면, 초기화 방법에 대해서는 계속해서 새로운 방법들이 나오는 중이고 he initialization같은 경우도 세상에 나온지 5년도 채 되지 않았다. 그럼에도 가장 좋은 초기화 방법이 무엇인가? 라고 물으면 답은 ‘모른다’라는 것이 정설인듯 하다.

- 이번 과제에서는 정확도를 99.41%정도까지 올렸지만, 조사한 결과 99.8%이상으로는 불가능한 영역이며, 현재 가장 높은 방법은 99.7%정도였다.(Kaggle.com출처) 이 방법은 CNN, DataAugmentation을 사용하였으며, 총 5개의 레이어를 사용하였다고 한다.

- 마지막 ensemble모델에서 파이썬 문법에 익숙치 않아 클래스로 만들지 못한 것이 아쉬웠다. 또한 Data augmentation 모델에서 메모리 문제인지 자꾸 세션이 죽어서 많이 augmentation하지 못한 부분이 아쉬웠다.

4. 첨부파일

* MNIST\_SoftMax.ipynb
  + SoftMax를 사용하여 다항분류
* MNIST\_NN\_hidden1\_node1024.ipynb
  + Hidden layer 1개와 node 1024사용, 초기화 방법 : truncated\_normal
* MNIST\_NN\_hidden2\_node512\_randomnormal.ipynb
  + Hidden layer 2개와 node 512개씩 사용, 초기화 방법 : random normal
* MNIST\_NN\_hidden2\_node512\_zeros.ipynb
  + Hidden layer 2개와 node 512개씩 사용, 초기화 방법 : zeros
* MNIST\_Xavier\_Initialization.ipynb
  + Hidden layer 1개와 node 1024개를 사용, 초기화 방법 : Xavier\_initialization
* MNIST\_He\_Initialization.ipynb
  + Hidden layer 1개와 node 1024개를 사용, 초기화 방법 : He\_initialization
* MNIST\_Deep\_NN\_hidden5\_node512\_Xavierinitialization.ipynb
  + Hidden layer 5개와 node 512개씩 사용, 초기화 방법 : Xavier initialization
* MNIST\_Dropout\_Keep\_Prob\_0.7.ipynb
  + Hidden layer 5개와 node 512개씩 사용, 초기화 방법 : Xavier initialization
  + Keep\_prob : 0.7
* MNIST\_Dropout\_Keep\_Prob\_0.5.ipynb
  + Hidden layer 5개와 node 512개씩 사용, 초기화 방법 : Xavier initialization
  + Keep\_prob : 0.5
* MNIST\_Single\_CNN.ipynb
  + Input->Conv->Pooling->FC->Softmax->Evaluator의 레이어 구성을 가짐
* MNIST\_Double\_CNN.ipynb
  + Input->Conv->Pooling->Conv->Pooling->FC->Dropout->Softmax->Evaluator의 레이어 구성을 가짐
* MNIST\_CNN\_DataAugmentation.ipynb
  + Double\_CNN구조에서 데이터를 ImageDataGenerator를 사용하여 Augment함
* MNIST\_CNN\_Ensemble.ipynb
  + Model3개를 이용해 ensemble함

1. References
2. Xavier Glorot, and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks, 2010 – Xavier initialization
3. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, In Microsoft Research, 2015 – He initialization(Kaiming He)
4. https://towardsdatascience.com/data-augmentation-experimentation-3e274504f04b