# 내 눈으로 보는 경사 하강법

권일재, 이현민

## 문제 제기

경사 하강법(Gradient Descent)은 뉴럴 네트워크의 대표적인 학습이다. 흔히 이에 대해 산에서 경사를 따라 굴러가는 공의 비유를 든다. 그런데 몇 만 차원을 훌쩍 넘는 뉴럴넷의 모수(parameter) 공간에서 공이 대체 어디서 어디로 굴러간단 말인가? 더 직관적으로 뉴럴넷의 모수를 눈으로 볼 수 있다면 뉴럴 네트워크에 대해 이해를 높일 수 있다고 생각했다.

## 가설

아래 요인들이 뉴럴 네트워크의 학습에 영향을 미치고 모수의 변화 경로를 다르게 만들 것이다.

* 1. 데이터의 종류: Train accuracy가 더 낮은 데이터셋, 즉 어려운 데이터일수록 뉴럴넷이 빠르게 학습하지 못하고 따라서 parameter 행렬의 특이값(singular vector)이 더 작을 것이다.
  2. Learning rate: Learning rate가 높을 수록 빠르게 학습하고
  3. Optimizer: Momentum이 있는 optimizer의 학습속도가 더 빠르고 parameter의 변화 궤적도 더 완만 할 것이다.

## 실험 방법

모수 공간을 시각화하기 위해서는 2차원 또는 3차원으로 줄일 필요가 있다. 따라서 대표적인 차원축소 방법인 주성분 분석(PCA)를 떠올렸고 구체적인 기법은 (Lorch E, 2016)을 참고했다.

1. 매 epoch 마다 각 layer의 weight 행벡터를 쌓아 행렬로 만든다.
2. Singular Value Decomposition을 이용해 모수 공간에서 singular value가 가장 높은 10개의 벡터를 뽑는다.
3. a에서 만든 행벡터들을 이 10개 벡터에 대한 좌표로 표현한 뒤 loss와 함께 도식화한다.

구체적인 환경 설정은 아래와 같다.

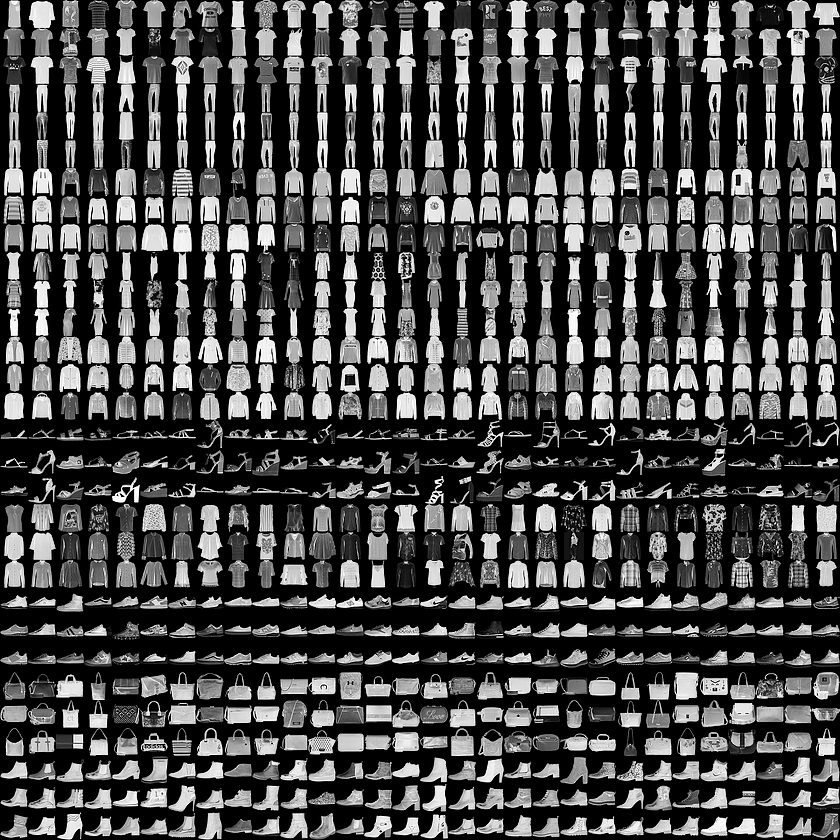
1. Hyperparameter 조정을 위한 실험이 아니므로 Validation set 없이 60000개의 Train set과 10000개의 Test set으로 나눴다.
2. Keras, tensorflow backend를 사용했고 SVD는 numpy.linalg package를 사용했다.
3. numpy와 tensorflow의 random 함수에 seed를 넣어 항상 initialization이 항상 같은 지점으로 되도록 했다.
4. Learning rate는 decay없이 일정하게 유지했다.
5. 실험에 사용된 코드는 <https://github.com/qpzm/MNIST-PCA> 에 올려두었다.

## 데이터

### MNIST

* 28x28 사이즈의 0~9 숫자에 대한 사람 손글씨 이미지이다.
* 각 픽셀은 0 이상 256 미만의 grayscale로 되어 있다.
* 60000개의 학습 데이터와 10,000개의 테스트 데이터로 이뤄져 있고 0~9까지 10개의 클래스로 분류되어 있다.

### Fashion MNIST

* 의류회사 Zalando의 열가지 상품군에 대해 60000개의 학습 데이터와 10,000개의 테스트 데이터로 이뤄져 있다.
* 사진이 좀 더 복잡하다는 점을 제외하면 MNIST와 전부 동일하다.
* 10개의 클래스는 아래와 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | T-shirt/top | 5 | Sandal |
| 1 | Trouser | 6 | Shirt |
| 2 | Pullover | 7 | Sneaker |
| 3 | Dress | 8 | Bag |
| 4 | Coat | 9 | Ankle boot |

## 실험1 Logistic Regression

Keras를 이용해 input과 가중치 parameter를 내적 => 10개의 클래스에 대한 내적 결과값에 softmax 함수 적용 => cross entropy loss를 계산했고 100 epoch 동안 학습시켰다. Batch size는 128로 했다.

### 정규화(Normalization)의 중요성

일단 정규화하지 않고 5 epoch 동안만 학습해보자. 아래와 같이 25%도 되지 않는 정확도를 보였다.

1. Normalization 이전 (int, 0~255)

|  |
| --- |
| 10000/10000 [==============================] - 1s 91us/step 0.2337 |

1. Normalization 이후 (float, 0~1)

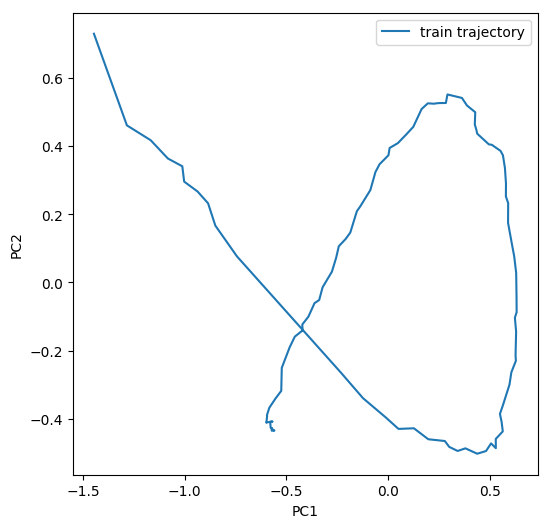
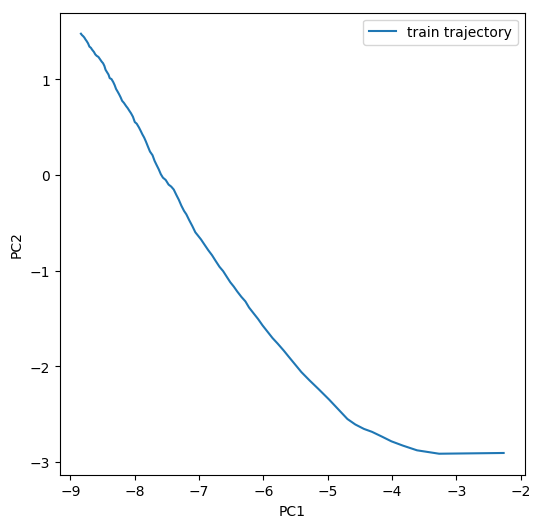
|  |
| --- |
| 10000/10000 [==============================] - 1s 83us/step 0.8872 |

왜 이런 큰 차이가 발생한 것일까? 우리는 두가지 정도의 이유를 생각해 보았다.

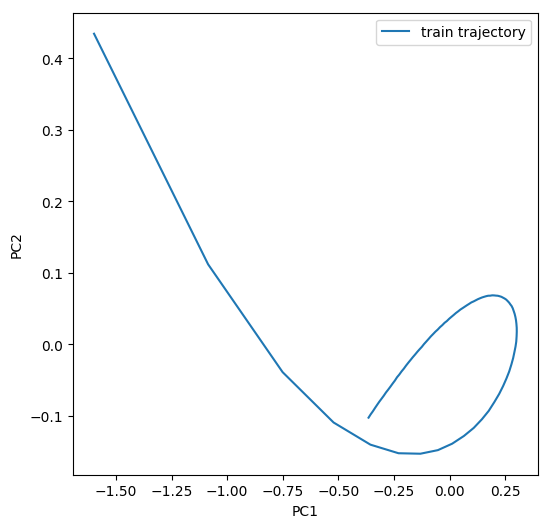
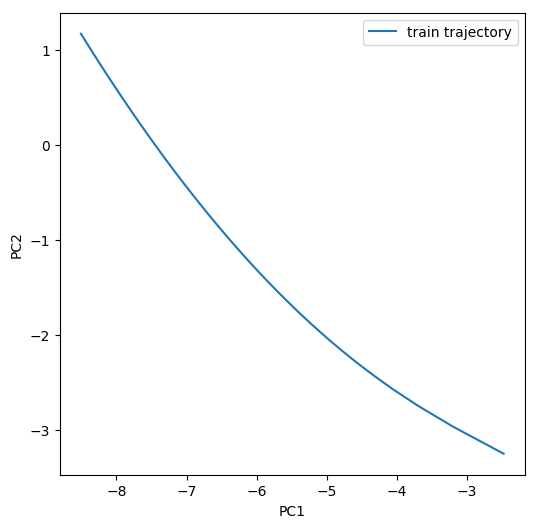
첫째로는 전처리 과정에서 normalization이나 re-scale을 하는 이유와 같다. 다른 요소들에 비해 큰 절대값을 갖는 값들이 존재할 경우 결과에 지나치게 많은 영향을 줄 수 있게 된다. 따라서 적정한 수준으로 값을 통일시켜주는 전처리 과정이 중요하다.

둘째로는 옷의 카테고리를 정하는 문제는 유독 옷의 색이 아닌 옷의 모양이 중요하기 때문이다. 예를 들어 3번 카테고리인 드레스의 경우 민소매와 긴 기장이 특징적이다. 그 드레스가 밝은 회색 드레스인지 진한 검은색 드레스인지는 옷의 카테고리를 정할때 전혀 중요하지 않은 요인인 것이다. 따라서 정확도가 높은 학습을 위해선 흰색 배경과 옷을 구분하는 것, 즉 테두리를 정확하게 학습하는 것이 중요하다고 생각했다.

100 epoch동안 학습한 결과인 그림1, 2를 비교하면 아래의 Normalization 후 상위 singular vector 방향(PC1,2,3,4)에서 parameter의 진동이 덜한 학습 궤적을 확인할 수 있다.



*그림**1. MNIST, SGD, lr=0.01, no normalization PC1&2(좌) PC3&4(우)*

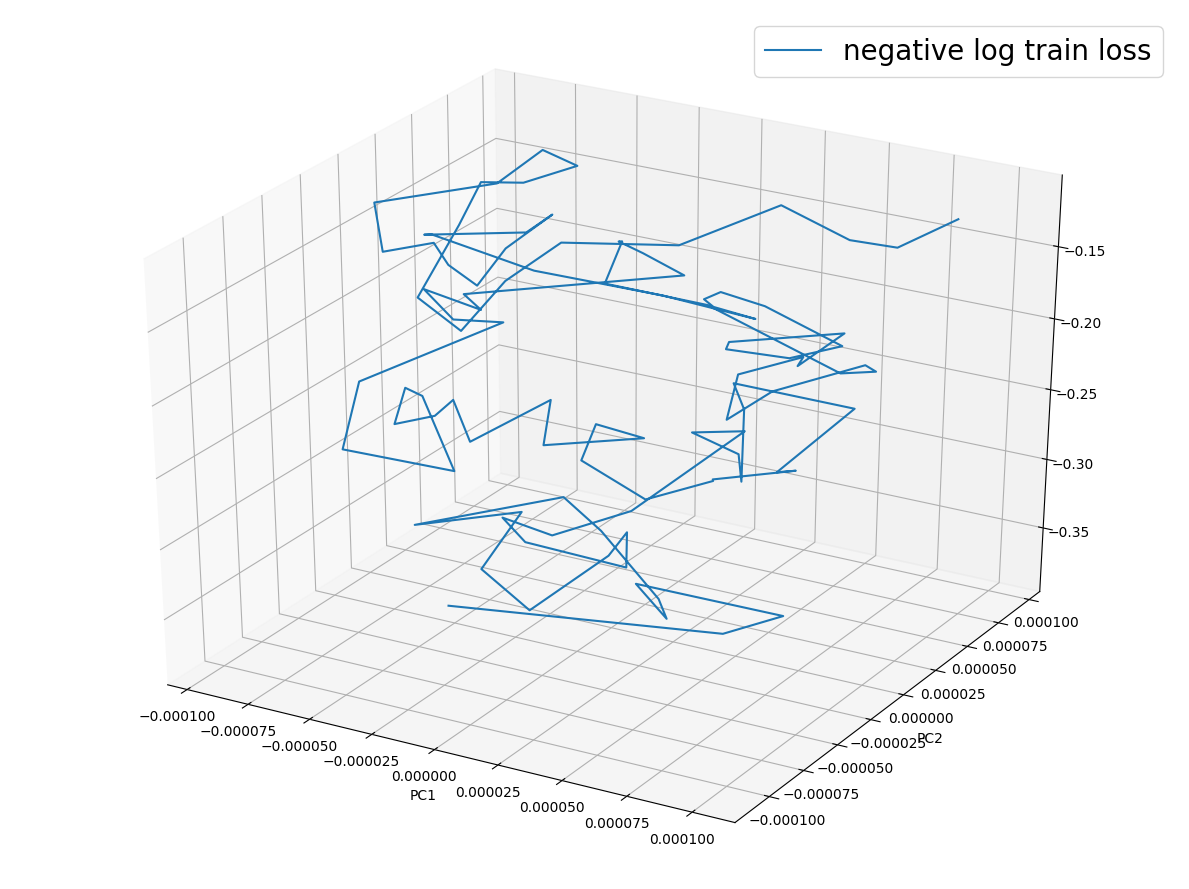
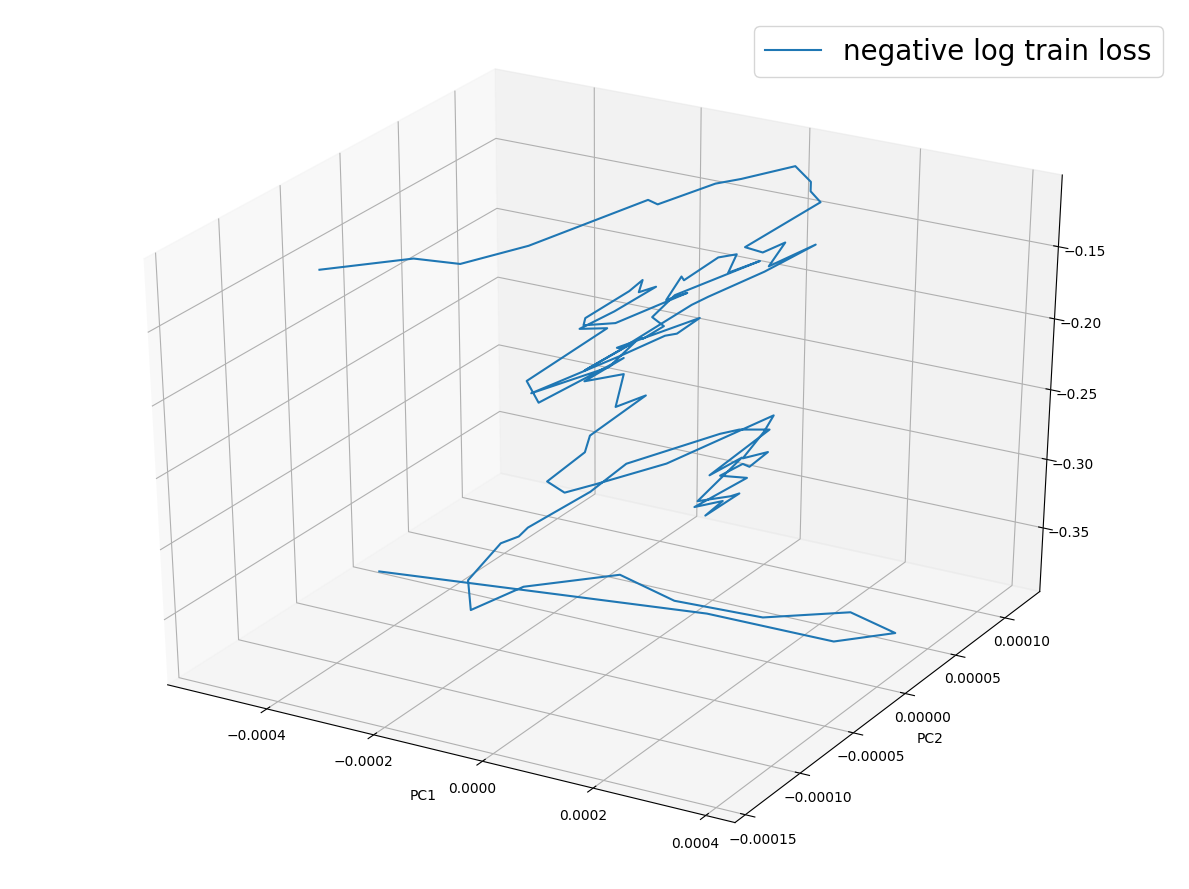


*그림**2. MNIST, SGD, lr=0.01, normalization PC1&2(좌) PC3&4(우)*

### B. Learning rate

### 

*그**림3. Fashion-MNIST, SGD, lr=1e-2, normalization PC7&8(좌) PC9&10(우)*



*그**림4. Fashion-MNIST, SGD, lr=1e-4, normalization PC7&8(좌) PC9&10(우)*

위의 그래프에서 z축은 negative log train loss을 나타낸다. 그림3을 보면 높은 Learning rate에서는 손실 함수값이 빠르게 하강하지만 일정 epoch에 도달하면 손실 함수값은 더 이상 낮아지지 않고 parameter 벡터만 진동한다. 반면에 그림4에서 낮은 learning rate에서는 손실 함수값이 천천히 하강하지만 parameter 벡터가 진동하는 시점(epoch)은 더 늦게 온다.

실제로 learning rate가 낮은 아래의 그래프에서는 아직 negative log train loss가 음수의 영역에 있고, 축의 scale도 위에 비해 작은 것을 확인할 수 있다. 다만 상위 특이값 벡터(PC1&2, PC3&4)에서는 스케일을 제외하고 그래프의 모양은 거의 유사했다.

### C. Optimizer

그래프의 모양보다는 scale의 차이가 컸다. 수치적으로 특이값을 비교함으로써 해당 특이벡터 방향에 투영한 parameter 벡터들의 분산을 비교할 수 있다. SGD와 Adam은 momentum의 여부에 차이가 있고 직관적으로 momentum이 있는 최적화 기법이 더 빠르게 즉 큰 분산을 가지고 있으리라 짐작할 수 있다. 실제로 상위 10개 특이값을 MNIST, lr=0.001, normalization 설정에서 비교하면 아래와 같다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Top10 Singular Values under MNIST, lr=0.001, normalization | | | | | | | | | |
| Adam | 346.281 | 55.338 | 16.674 | 6.470 | 3.477 | 2.000 | 1.095 | 0.688 | 0.575 | 0.482 |
| SGD | 50.40 | 8.73 | 0.94 | 0.18 | 7.17E-02 | 1.95E-02 | 5.36E-03 | 2.82E-03 | 2.47E-03 | 2.02E-03 |

### D. Dataset

MNIST 와 Fashion MNIST를 그래프의 모양으로는 구분하기는 힘들었다. 상위 10개 특이값의 분포를 비교하면 차이가 있지만 유의미한 차이의 수준을 판별하는 기준을 세우지는 못했다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Top10 Singular Values under Adam, lr=0.001, normalization | | | | | | | | | |
| Fashion | 346.281 | 55.338 | 16.674 | 6.470 | 3.477 | 2.000 | 1.095 | 0.688 | 0.575 | 0.482 |
| MNIST | 255.410 | 48.299 | 14.851 | 6.129 | 3.360 | 1.962 | 1.046 | 0.651 | 0.496 | 0.427 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Top10 Singular Values under SGD, lr=0.001, normalization | | | | | | | | | |
| Fashion | 4.9E+01 | 7.1E+00 | 1.2E+00 | 2.5E-01 | 1.1E-01 | 4.4E-02 | 1.5E-02 | 6.6E-03 | 5.6E-03 | 5.0E-03 |
| MNIST | 5.04E+01 | 8.73E+00 | 9.43E-01 | 1.85E-01 | 7.17E-02 | 1.95E-02 | 5.36E-03 | 2.82E-03 | 2.47E-03 | 2.02E-03 |

### E. 설정별 결과 비교

* Normalization을 한 경우, momentum이 있는 Adam optimizer > SGD 보다 높은 성능을 보였다.
* Learning rate는 Adam optimizer의 경우 decay가 없고 100 epoch의 제한을 두어서인지 0.001이 가장 좋은 결과를 보였다. 반면에 SGD의 경우 momentum이 없어서 인지 lr 0.01이 가장 좋은 성능을 보였다. 정확한 이유는 위의 의심에 기초한 추가 실험을 통해 밝혀야겠다.
* Fashion MNIST에서 MNIST보다 낮은 성능을 보였다.

#### MNIST 결과

|  |  |  |
| --- | --- | --- |
| **Optimizer-lr-normalization** | **Train Accuracy** | **Test Accuracy** |
| Adam-0.001-norm | 0.937 | 0.928 |
| Adam-0.0001-norm | 0.923 | 0.924 |
| Adam-0.01-norm | 0.937 | 0.922 |
| SGD-0.01-norm | 0.906 | 0.912 |
| SGD-0.001-norm | 0.862 | 0.871 |
| SGD-0.001-norm | 0.850 | 0.835 |
| Adam-0.0001-no\_norm | 0.848 | 0.833 |
| SGD-0.01-no\_norm | 0.760 | 0.756 |
| SGD-0.0001-no\_norm | 0.720 | 0.720 |
| Adam-0.001-no\_norm | 0.580 | 0.575 |
| Adam-0.01-no\_norm | 0.477 | 0.475 |

#### 

#### Fashion MNIST 결과

|  |  |  |
| --- | --- | --- |
| **Data-Optimizer-lr-normalization** | **Train Accuracy** | **Test Accuracy** |
| Adam-0.001-norm | 0.874 | 0.846 |
| Adam-0.0001-norm | 0.857 | 0.839 |
| Adam-0.01-norm | 0.874 | 0.833 |
| SGD-0.01-norm | 0.841 | 0.829 |
| SGD-0.001-norm | 0.779 | 0.767 |
| SGD-0.0001-norm | 0.646 | 0.643 |
| Adam-0.001-no\_norm | 0.566 | 0.559 |
| Adam-0.0001-no\_norm | 0.548 | 0.531 |
| SGD-0.0001-no\_norm | 0.537 | 0.528 |
| SGD-0.001-no\_norm | 0.292 | 0.289 |
| Adam-0.01-no\_norm | 0.289 | 0.288 |
| SGD-0.01-no\_norm | 0.200 | 0.200 |

## 실험2 CNN

### A. ConvNet

컨볼루션 신경망은 ANN 중에서도 이미지의 특성에 맞는 신경망이다. 기존의 인공 신경망처럼 에러를 back-propagation 함으로써 weight를 조정해 나가는 것은 같으나 이미지의 규격이 커질수록 지수승으로 증가하는 parameter를 효과적으로 줄일 수 있기 위해 몇 가지 차별점이 있다.

먼저는 Convolutional Layer의 특징이다. Conv의 큰 특징은 어떤 층의 한 노드가 굳이 이전 층의 모든 node 들과 연결되어 있지 않다는 점이다(Locally Connected Neural Net). 이번 모델에서 사용한 것처럼 3사이즈의 Kernel을 사용할 경우 해당 위치의 깊이 평면들끼리만 weight와 bias를 공유한다. 다시 말해 왼쪽 끝에 있는 점이 오른쪽 아래에 있는 점에 영향을 줄 확률이 매우 낮은 이미지의 특징을 바탕으로 forward pass 과정이 수월해진다.

또한 Pooling Layer를 추가함으로 overfitting의 문제를 방지해준다. Pooling Layer는 앞선 Conv 층과 달리 activation layer가 아니라 단순히 가로와 세로의 규격을 줄여 다음 층에 넘겨주는 역할을 한다. 이번 모형에서 사용한 pooling method는 가장 대중적으로 사용되는 Max-Pooling이다. Max Polling은 주어진 풀의 사이즈 중에서 최대값을 선택해서 넘겨주게 된다. 결과적으로는 3/4의 액티베이션 값들을 버리게 됨으로 계산을 단순하게 만들어준다.

이렇게 Pooling된 출력값은 Fully-Connected layer로 이어진다. 이 층의 경우 앞선 모든 노드들과 완전 연결되어 1x1x{클라스의 수}로 출력되게 된다.

### B. Model

이번 실험에서는 2개의 Hidden layer를 사용했으며 각각의 layer는 Convolutional Layer, Max-Pooling Layer, Dropout으로 이뤄져 있다.

#### 1) Conv2D

Fashion MNIST는 28X28X1의 입력 크기를 갖고 있다. 이 모델에서는 ( 3, 3 )의 커널 사이즈를 사용했다. Activation Function은 RELU를 사용했다. Output filter는 32로 출력되게 모델을 만들었다.

이번에 사용한 CNN의 모델 구조는 다음과 같다.

#### 2) Pooling Layer

Pooling 은 가장 많이 쓰는 MaxPooling을 사용했다. 풀의 사이즈는 (2, 2)로 설정했고 stride 역시 2로 유지했다.

#### 3) Dropout

Overfitting 문제를 피하기 위해 dropout을 실시했다. 계수는 0.2로 실행했다.

이번 실험에 사용한 모델의 요약은 다음과 같다.

|  |
| --- |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Layer (type) Output Shape Param # ================================================================= conv2d\_1 (Conv2D) (None, 26, 26, 32) 320 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ max\_pooling2d\_1 (MaxPooling2 (None, 13, 13, 32) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dropout\_1 (Dropout) (None, 13, 13, 32) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ conv2d\_2 (Conv2D) (None, 11, 11, 64) 18496 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ max\_pooling2d\_2 (MaxPooling2 (None, 5, 5, 64) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dropout\_2 (Dropout) (None, 5, 5, 64) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ flatten\_1 (Flatten) (None, 1600) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dense\_1 (Dense) (None, 250) 400250 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dropout\_3 (Dropout) (None, 250) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dense\_2 (Dense) (None, 10) 2510 ================================================================= Total params: 421,576 Trainable params: 421,576 Non-trainable params: 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

결과는 다음과 같았다.

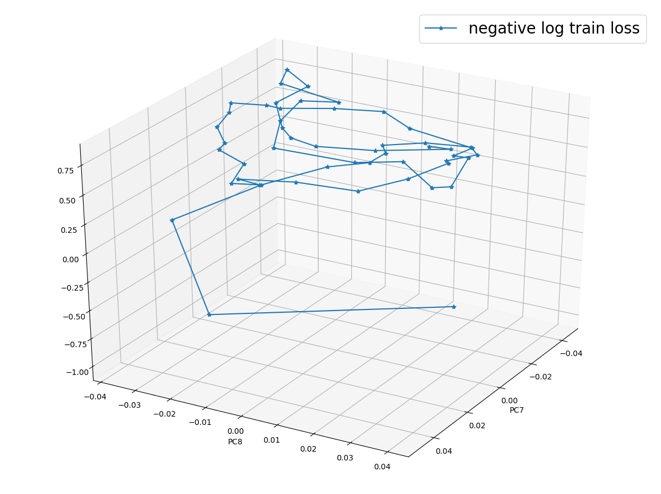
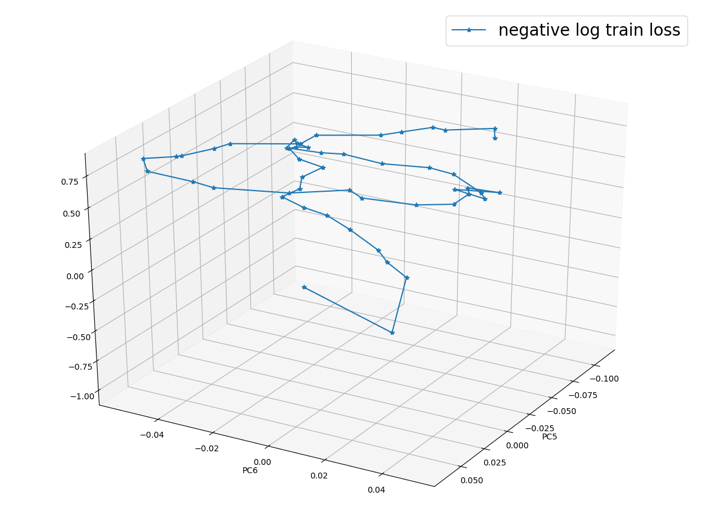
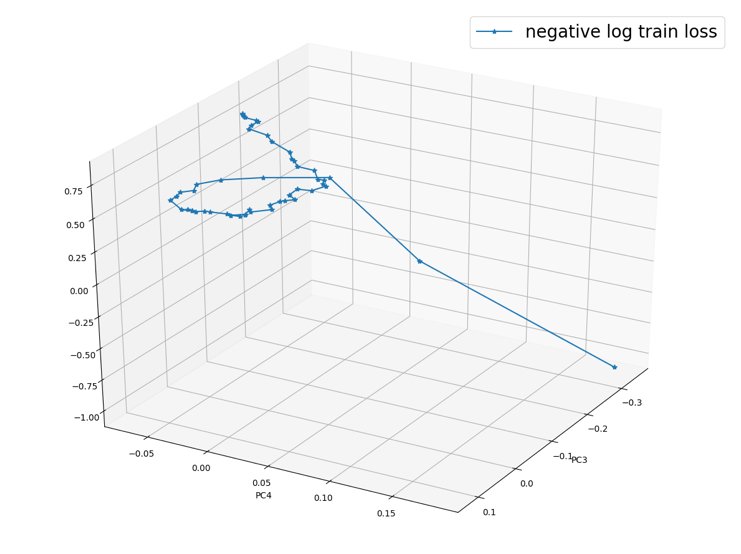
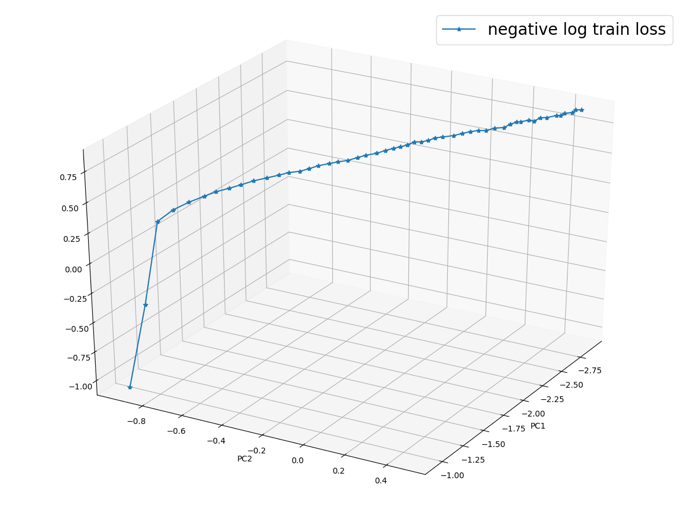
|  |
| --- |
| Training: 0.17190531 loss / 0.93478666 acc Validation: 0.23376373 loss / 0.92156000 acc |

Fashion MNIST 데이터에 대해 앞선 Logistic Regression에 비해 크게 향상된 결과를 보였다. Hyperparameter인 드랍 아웃 비율, stride 등을 바꿔가며 실험을 해본 결과 대부분의 결과가 91%를 넘는 정확도를 보여줬다.

### C. 각 layer별 parameter 궤적

PCA를 이용한 논문 (Lorch E, 2016) 에서는 모든 parameter를 하나의 벡터로 만들었는데 각 layer 별로 학습에 다른 특징을 나타내는 지 확인해보기 위해 각각의 layer 별로 도식화 했다. 하위 특이 벡터로 갈 수록 손실 함수의 증가에 거의 기여하지 못하면서 진동하는 모습을 보인다. Layer 종류, 깊이별로 어떻게 달라지는 지는 더 알아볼 과제이다.

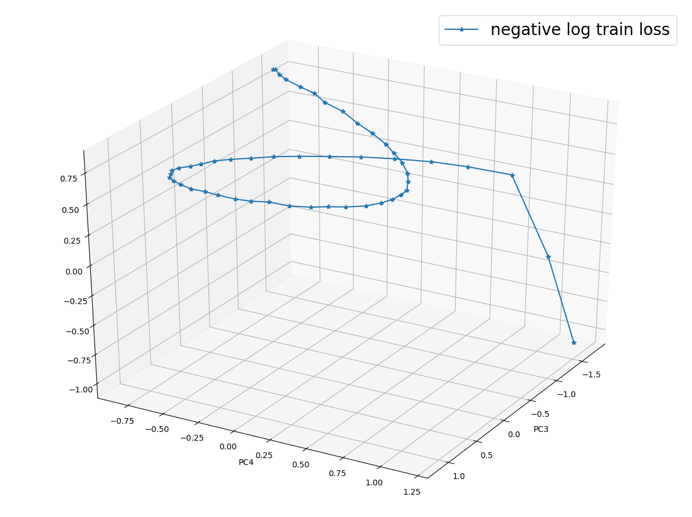
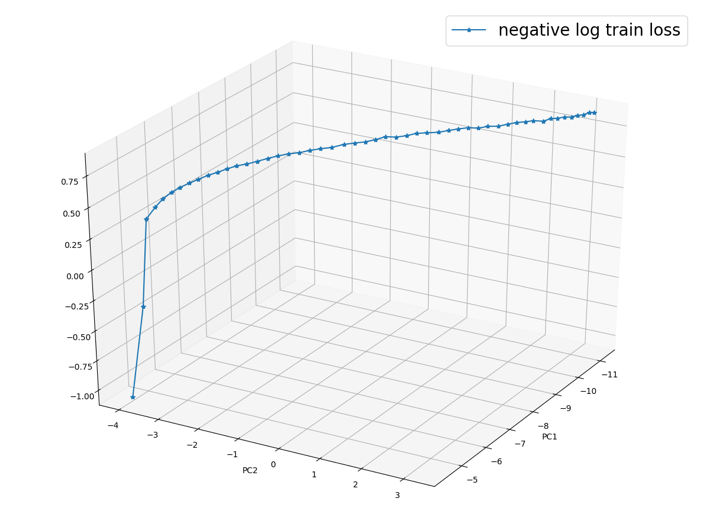
#### 1) Conv1 layer

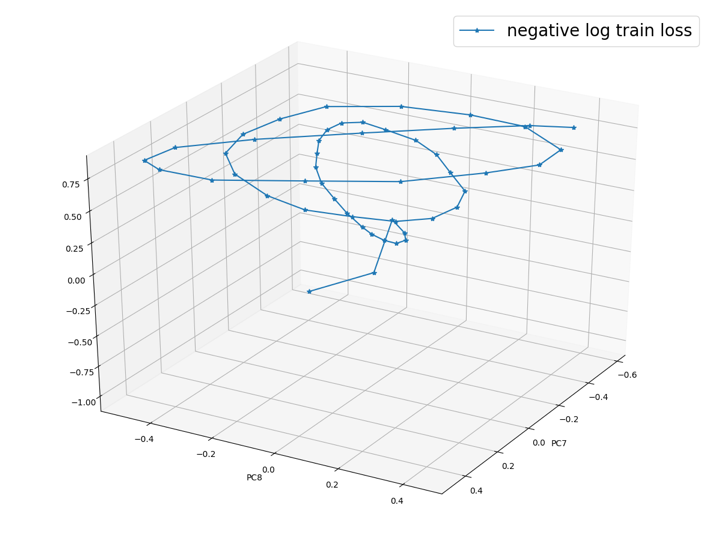
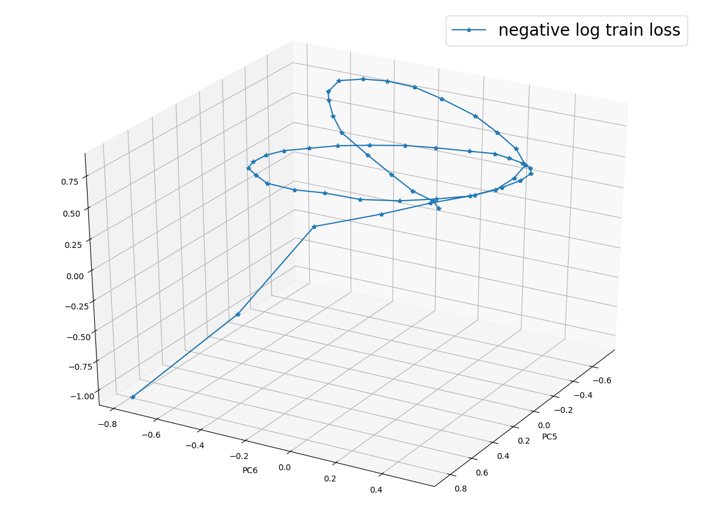


*그림5. Conv1 layer Fashion-MNIST, Adam batch size=128, lr=1e-3, normalization*

*PC1&2(좌상) PC3&4(우상) PC7&8(좌하) PC9&10(우하)*

#### 2) Conv2 layer

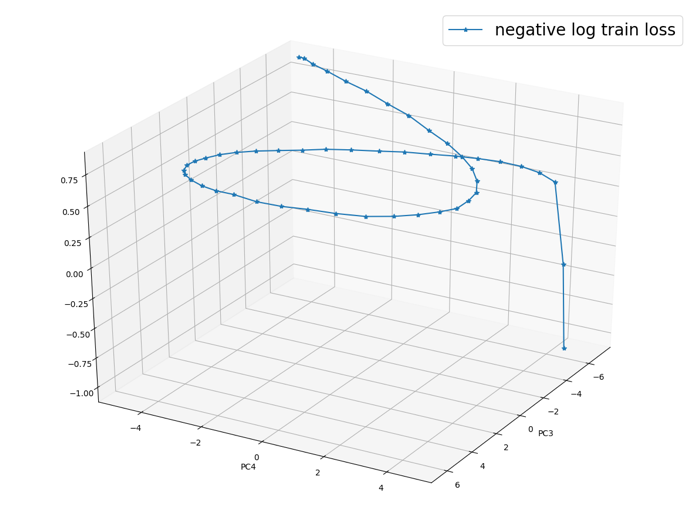
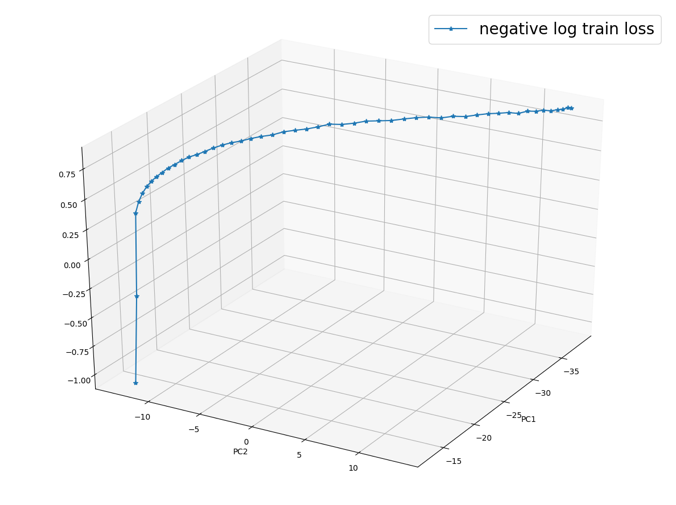




*그림6. Conv2 layer Fashion-MNIST, Adam batch size=128, lr=1e-3, normalization*

*PC1&2(좌상) PC3&4(우상) PC7&8(좌하) PC9&10(우하)*

#### 3) Dense1 layer (Fully Connected)



#### ../Development/NaverDSC18/MNIST-PCA/CNN/FASHION_plots/512-weights-Adam-0.001-maxmin/layer2/3d-2.png../Development/NaverDSC18/MNIST-PCA/CNN/FASHION_plots/512-weights-Adam-0.001-maxmin/layer2/3d-3.png

*그림7. Dense1 layer Fashion-MNIST, Adam batch size=128, lr=1e-3, normalization*

*PC1&2(좌상) PC3&4(우상) PC7&8(좌하) PC9&10(우하)*

## 결론

이번 실험에서는 MNIST와 Fashion MNIST 데이터를 활용해 Logistic Regression과 Convolutional Neural Net을 학습시켜보았다. 이 과정에서 경사 하강법에 따른 weight parameter의 경로를 PCA를 이용해시각화하고 특이값을 비교함으로써  normalization, learning rate와 optimizer가 뉴럴 네트워크의 학습에 미치는 영향을 확인했다. 다만 데이터의 학습 난이도를 특이값과 연결 짓는 것은 좀 더 다양한 데이터와 환경설정 하에서 실험해 볼 필요성이 있었다. Layer 종류, 깊이, batch size에 따른 학습 궤적의 변화도 더 알아볼 주제이다.

CNN 모델의 테스트 정확도는 92%에 그쳐 더 나은 모델 혹은 학습 방법이 필요하다고 생각했다. Hyperparameter의 값들에 따라 정확도가 조금씩 변해 최적의 hyperparameter를 찾는 것이 까다로움을 확인했다. 추가적인 실험을 통해 Validation set을 나누어 최적의 hyperparameter를 경험적으로 찾아보는 것이 중요해 보인다.

## 참고 문헌

Lorch E. Visualizing deep network training trajectories with pca. In The 33rd International Conference on Machine Learning JMLR volume 2016 (Vol. 48).

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

<https://github.com/zalandoresearch/fashion-mnist> (Fashion MNIST github)