

source code는 <http://yann.lecun.com/exdb/mnist/>에서 train-images-idx3-ubyte.gz, train-labels-idx1-ubyte.gz, t10k-images-idx3-ubyte.gz, t10k-labels-idx1-ubyte.gz 파일을 다운받아서 아래와 같이 Colab 환경 기준으로 파일을 올린 후, file_path에 해당 경로를 수정한 후 사용할 수 있습니다.

```

import urllib.request
import os.path
import gzip
import pickle
import os
import numpy as np
import sys

file_path = ['./train-images-idx3-ubyte.gz', './train-labels-idx1-ubyte.gz', './t10k-images-idx3-ubyte.gz', './t10k-labels-idx1-ubyte.gz']

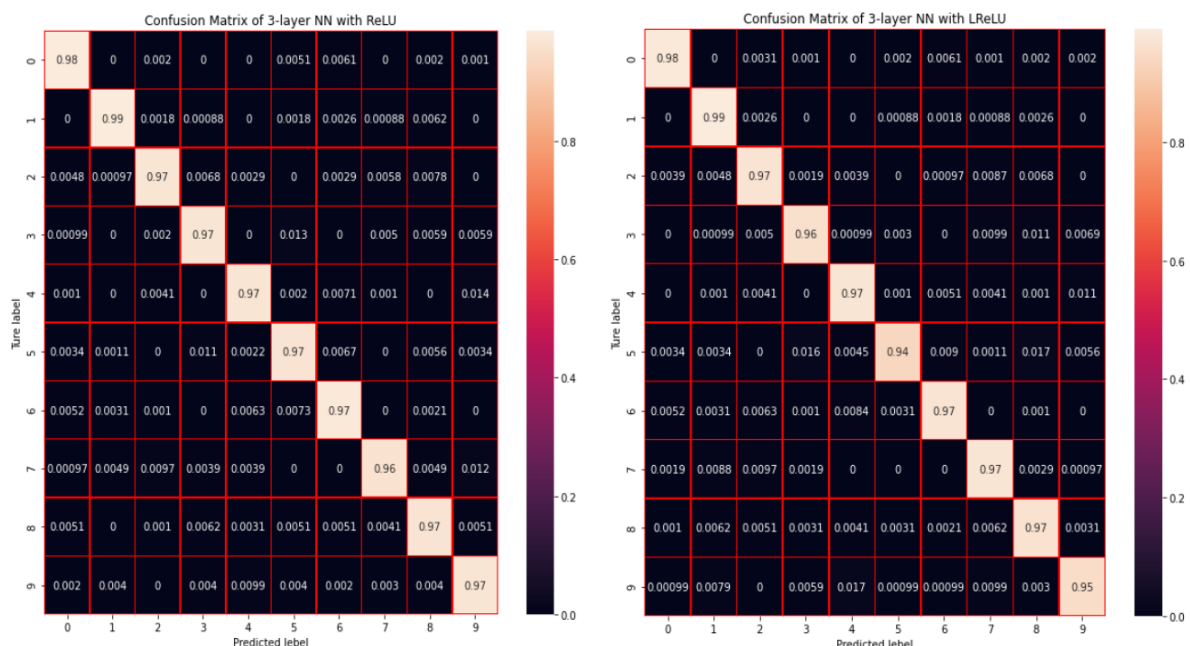
def file_load(path):
    if path.find('images') != -1:
        with gzip.open(path, 'rb') as f:
            data = np.frombuffer(f.read(), np.uint8, offset=16)
            data = data.reshape(-1, 784)
            return data
    else:
        with gzip.open(path, 'rb') as f:

```

(Main) 3-layer Neural Network for Classification without the deep learning framework (only python)

1. For all networks (3-layer NN with ReLU, 3-layer NN with LReLU), show the results and compare (ReLU vs LReLU)

(a) 10x10 Confusion Matrix (using test data)



Confusion matrix의 대각선 값은 각 Class별로 True label을 정확히 맞춘 predicted label들을 나타낸다. 전체적으로 0.01정도의 차이를 두고 ReLU를 활성화 함수로 적용한 경우가 LReLU를 활성화 함수로 적용한 경우보다 더 많이 정답을 맞춘 것을 알 수 있다.































(b) Top 3 score images (all classess) using test data

Top 3 score image를 각 class마다 정답일 확률이 큰 순서대로 아래와 같이 나열하였다.

각 확률마다 소수점 이하를 보지않으면 큰 차이가 보이지 않아서 확률 그대로를 출력하였다.































- 3-layer NN with ReLU

Probability of images

  	[0.999999999793228, 0.999999999688967, 0.999999999375266]
  	[0.9999993340953384, 0.9999976868629328, 0.9999945528538616]
  	[0.9999999995770239, 0.9999999993344335, 0.9999999986869932]
  	[0.9999999998901861, 0.9999999988262991, 0.9999999982417558]
  	[0.9999999919520902, 0.9999999915728784, 0.9999999906887368]
  	[0.99999999998413, 0.999999999814875, 0.999999999439604]
  	[0.9999999994700468, 0.9999999993040323, 0.9999999992049771]
  	[0.999999962352084, 0.9999999592933827, 0.9999999289620664]
  	[0.9999999997283584, 0.9999999994681164, 0.9999999988034987]
  	[0.9999997213291859, 0.9999996849004467, 0.9999995632614473]

- 3-layer NN with LReLU

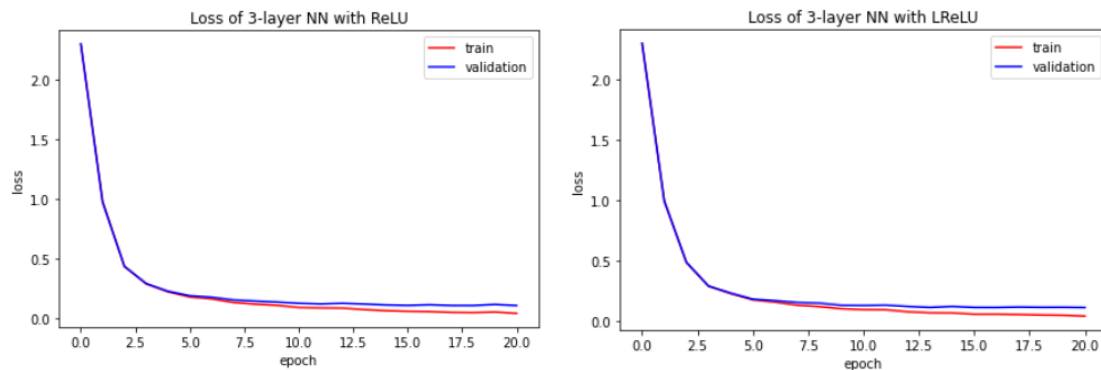
Probability of images

  	[0.999999999352549, 0.999999999310736, 0.9999999994313367]
  	[0.999999975688035, 0.9999999013906646, 0.9999997094630282]
  	[0.999999999944642, 0.999999999724873, 0.99999999963918]
  	[0.999999998761011, 0.999999998098423, 0.9999999996925939]
  	[0.9999999950834448, 0.9999999893469099, 0.9999999855851998]
  	[0.999999999777622, 0.999999999386853, 0.999999999110747]
  	[0.9999999997929647, 0.9999999995549571, 0.999999999302531]
  	[0.9999999538066989, 0.9999999292407841, 0.9999999156784407]
  	[0.999999999187785, 0.9999999997570783, 0.9999999994765947]
  	[0.9999998277746426, 0.999999729142457, 0.9999997125497861]

ReLU와 LReLU 둘 다 각 클래스별 높은 확률로 맞춘 image들의 확률들이 99%로 아주 높은 것을

확인할 수 있다. 둘의 차이는 0.0000001보다 작은 차이를 보이면서 Top 3의 클래스별 이미지들을 높은 확률로 각 클래스별 정답을 맞추고 있다. 하지만 Top 3별 이미지는 ReLU와 LReLU가 같은 경우도 있지만, 대부분 다른 이미지를 높은 확률의 정답으로 맞춘 것을 알 수 있다. 활성화함수가 달라짐에 따라 이미지의 정답학습이 다르게 학습됨을 알 수 있다. 그 외에 확률이 대부분 99%로 ReLU와 LReLU 둘 다 나타나기 때문에 다른 차이점은 위의 자료로 확인할 수는 없었다.

(c) Training Loss graph using train, validation data



Loss graph 상으로는 두 모델의 차이보다는 공통된 모습을 더 확인할 수 있다. 똑같이 batch size=100으로 20 epoch을 돌렸는데 validation loss는 둘 다 비슷한 시기에 일정하게 유지되는 것을 확인할 수 있다.

시그모이드나 쌍곡탄젠트 함수와 달리 ReLU 함수는 직선으로 이루어져 미분하면 계단함수 꼴이 된다. 즉, 도함수의 input x 가 엄청 큰 값을 가져도 무조건 1의 값을 반환하므로 경사가 소실될 위험이 없다. 그렇기에 시그모이드나 쌍곡탄젠트 함수보다 학습 속도가 빠르다고 할 수 있다. 이와 비슷하게 LReLU의 함수 역시 비슷한 특성을 가져 학습 속도가 빠르다고 할 수 있다. 이러한 점은 loss graph에도 나타나는데 1번의 epoch을 지나고 나면 train set에 대해서 정확도가 둘 다 50%이상은 나오기 때문에 빠른 학습률을 가진다고 할 수 있다.

하지만 LReLU와 달리 ReLU의 도함수는 x 가 0보다 작으면 무조건 0을 반환하므로 학습시 활성화되지 않는 뉴런이 생길 위험이 있다. LReLU는 이러한 ReLU의 단점을 보완하여 x 가 0보다 작아도 작은 경사값(alpha)을 가질 수 있도록 한다. 본 코드에서는 alpha를 0.1로 설정하였다.

학습 결과는 MNIST 데이터에서는 어느 활성화 함수가 더 좋다고 확실히 말할 수 있을 만큼의 큰 성능 차이가 보이지 않았다. 동일한 data set크기에 epoch=20으로 학습시켰을 때, 마지막 epoch에서 ReLU의 경우 "train accuracy: 0.9858958333333333 valid accuracy: 0.96725", LReLU의 경우 "train accuracy: 0.9865833333333334 valid accuracy: 0.9698333333333333"이 결과로 나왔다. Test dataset에 대한 accuracy는 ReLU는 "0.9701", LReLU는 "0.9682"으로 큰 차이를 보이지 않고 소수점 차이로 ReLU의 정확도가 조금 더 높게 나왔다. (a)의 confusion matrix를 참고하더라도 test dataset에서 각 클래스별 정확도는 ReLU가 조금 더 높게 나오는 것을 확인할 수 있다. 하지만 실험을 거듭할수록 소수점 차이가 반복되면서 LReLU가 학습이 더 잘되면서 test dataset에 대해

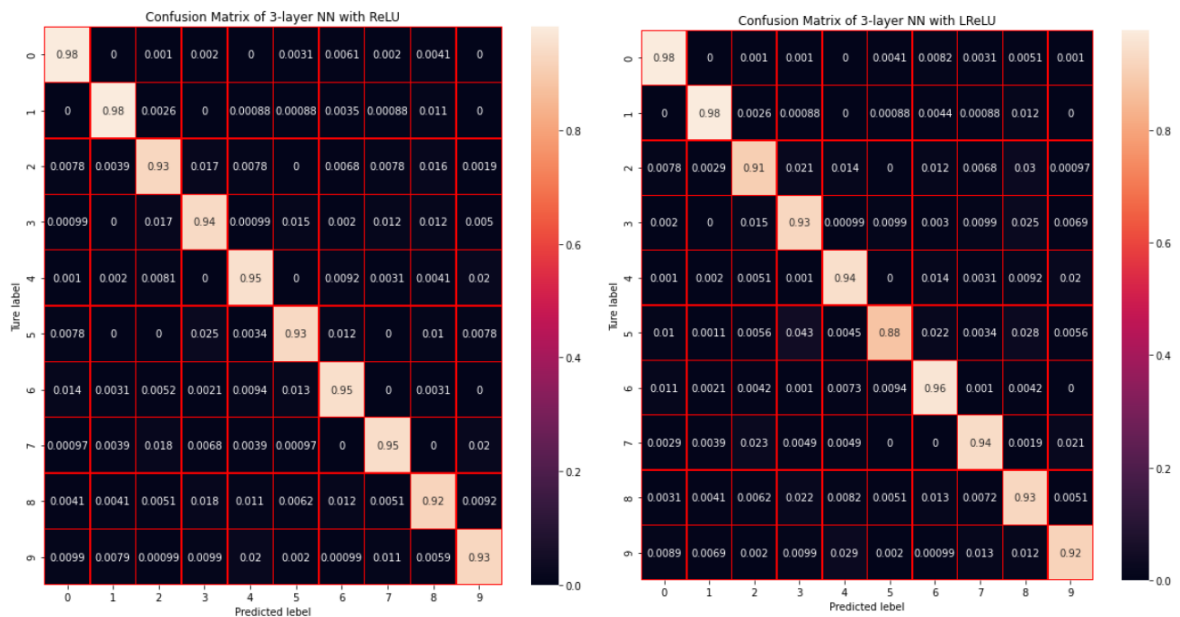
0.97의 확률이 나올때가 있기 때문에 정확히 ReLU가 LReLU보다 더 학습이 잘 되는 활성화 함수라고 말할 수 없다.

ReLU 함수는 $x < 0$ 일 때 경사가 사라져버려 학습 과정이 불안해질 수 있는 문제가 있지만 LReLU는 $x < 0$ 일 때도 학습이 진행되기 때문에 ReLU 함수보다 효과적인 활성화 함수라고 말할 수 있을 것이다. 하지만 실제로는 train data에 따라서 효과의 여부가 다르기 때문에 둘 중 어떤 활성화 함수가 항상 더 좋다고 말할 수 없으며, 항상 train data set의 특성에 맞춰서 학습할 수 있는 활성화 함수를 고르는 것이 최선임을 확인할 수 있다.

(Extra credit) 3-layer Neural Network for Classification using a deep learning framework (e.g. pytorch, tensorflow)

1. For all networks (3-layer NN with ReLU, 3-layer NN with LReLU), show the results and compare (ReLU vs LReLU)

(a) 10x10 Confusion Matrix (using test data)



Test dataset에 대해서 ReLU를 사용한 모델이 각 클래스별 정답을 맞출 확률이 높은 것을 볼 수 있다. Class 가 5일 때를 제외하고는 소수점 차이밖에 보이지 않으므로 ReLU와 LReLU의 성능이 비슷하다고 볼 수 있다.































(b) Top 3 score images (all classess) using test data

Top 3 score image를 각 class마다 정답일 확률이 큰 순서대로 아래와 같이 나열하였다.

각 확률마다 소수점 이하를 보지않으면 큰 차이가 보이지 않아서 확률 그대로를 출력하였다.































- 3-layer NN with ReLU

Probability of images

			[0.999966, 0.9999689, 0.99993765]
			[0.99896157, 0.9989851, 0.9989593]
			[0.9999801, 0.9999863, 0.9999825]
			[0.99991596, 0.99990904, 0.9998512]
			[0.9999014, 0.9999031, 0.99996173]
			[0.99988747, 0.9999113, 0.9998996]
			[0.9999201, 0.999881, 0.9998727]
			[0.9998228, 0.9997341, 0.9997366]
			[0.99950635, 0.99957675, 0.9995041]
			[0.99934095, 0.99935836, 0.9992387]

- 3-layer NN with LReLU

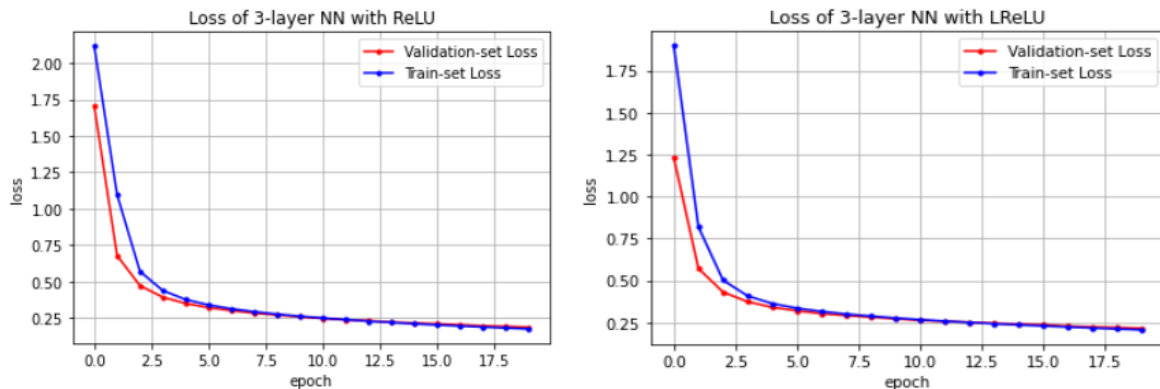
Probability of images

			[0.99998236, 0.9999764, 0.9999672]
			[0.9994999, 0.99931324, 0.9994381]
			[0.99993443, 0.9999434, 0.99992967]
			[0.9997744, 0.99981993, 0.9997576]
			[0.9999647, 0.9999478, 0.9999578]
			[0.9996166, 0.99962544, 0.999608]
			[0.99986696, 0.9998722, 0.9998783]
			[0.9998388, 0.9998084, 0.9997688]
			[0.99978, 0.99961436, 0.99968123]
			[0.99897754, 0.9988147, 0.9989435]

직접짠 코드와 유사한 결과로, ReLU와 LReLU 둘 다 각 클래스별 높은 확률로 맞춘 image들의 확률들이 99%로 아주 높은 것을 확인할 수 있다. 둘의 차이는 0.001보다 작은 차이를 보이면서 Top 3의 클래스별 이미지들을 둘 다 높은 확률로 각 클래스별 정답을 맞추고 있다. 마찬가지로 대부분 다른 이미지를 높은 확률의 정답으로 맞춘 것을 통해 활성화함수가 달라짐에 따라 이미지의 정답학습이 다르게 학습됨을 알 수 있다. ReLU와 LReLU 둘 다 확률이 대부분 99%로 나타나기

때문에 다른 차이점은 위의 자료로 확인할 수는 없었다.

(c) Training Loss graph using train, validation data



LReLU의 alpha 값을 0.3으로 설정하고 진행하였다.

똑같이 batch size=100으로 20 epoch을 돌렸는데 validation loss는 둘 다 비슷한 시기에 일정하게 유지되는 것을 확인할 수 있다. 차이점은 LReLU의 경우 처음 시작부터 train과 validation set 둘 다 loss값이 작게 시작함을 알 수 있다. 하지만 2 epoch 이후로는 ReLU와 LReLU 둘 다 loss값의 경향이 비슷한걸로 보아 학습률은 비슷하다고 할 수 있다.

마찬가지로, 학습 결과는 MNIST 데이터에서는 어느 활성화 함수가 더 좋다고 확실히 말할 수 있을 만큼의 큰 성능 차이가 보이지 않았다. 동일한 data set크기에 epoch=20으로 학습시켰을 때, 마지막 epoch에서 ReLU의 경우 "train_loss: 0.1708 - train_accuracy: 0.9505 - val_loss: 0.1832 - val_accuracy: 0.9462", LReLU의 경우 "loss: 0.2049 - accuracy: 0.9411 - val_loss: 0.2159 - val_accuracy: 0.9404"이 결과로 나왔다.

Test dataset에 대한 accuracy는 ReLU는 "0.9483", LReLU는 "0.9394"으로 큰 차이를 보이지 않고 소수점 차이로 ReLU의 정확도가 조금은 더 높게 나왔다. (a)의 confusion matrix를 참고하더라도 test dataset에서 각 클래스별 정확도는 ReLU가 조금 더 높게 나오는 것을 확인할 수 있다. 하지만 마찬가지로 실험을 거듭할수록 소수점 차이가 반복되면서 LReLU가 학습이 더 잘되면서 test dataset에 대해 0.95의 확률이 나올 때가 있기 때문에 정확히 ReLU가 LReLU보다 더 학습이 잘 되는 활성화 함수라고 말할 수 없다.