20180786 통계학과 이효진

* 데이터 확인

fashion mnist

* <https://github.com/zalandoresearch/fashion-mnist/>
* utils/mnist\_reader.py를 참고
* 10가지 의류에 대한 이미지 데이터
* original mnist와 마찬가지로 28 x 28이 기본 단위
* Validation set

- sklearn (scikit learn) -> model\_selection -> train\_test\_split (-> split train & validation)

- train\_test\_split(Xtrain, Ytrain, test\_size=테스트비율, shuffle=셔플여부, ...)

X\_train2, X\_val, y\_train2, y\_val = train\_test\_split(X\_train, y\_train, test\_size=0.2)

print("train:", X\_train2.shape, y\_train2.shape,

"\n test:", X\_val.shape, y\_val.shape,

"\ntotal: ", X\_train2.shape[0] + X\_val.shape[0],

"\t /", y\_train2.shape[0] + y\_val.shape[0])

train: (48000, 784) (48000,)

test: (12000, 784) (12000,)

total: 60000 / 60000

* 토대 마련을 위한 layer 전개

가중치 초기값: 특정 표준편차 지정, Xavier 초깃값, he 초기값

행렬곱을 위한 Hidden Layer의 shape 확인.

Activation functions: Sigmoid, ReLU

* 대체로 연산식은 다 기억하는데, backward에서 이용하는 특정 값(특히 input) 저장에 대해 자꾸 잊어버림

Loss function: Gradient Descent, Cross Entropy

* 2차원 행렬에 대한 구성은 신경 쓰는데, 1차원일 때를 신경 쓰지 않음.
* Cross Entropy에서 one hot encoding을 하지 않았을 때의 직관 부족
* Weight decay

Batch: 샘플링 학습

* Numpy 패키지 함수 이름을 헷갈림.

Batch Normalization

* 다차원으로 확장할 때의 직관 부족

Dropout

* 학습시킬 때 mask의 구현은 확실히 기억하고 있음.
* 그러나 모형 test할 때 dropout되지 않은 비율을 곱해주는 걸 잊어버림.
* 적합 기본 토대 (MultiLayerNetExtend class 이용)

for i in range(100):

learning\_rate = 10 \*\* np.random.uniform(-3, 3)

weight\_decay = 10 \*\* np.random.uniform(-4, 4)

network = MultiLayerNetExtend(

input\_size=X\_train.shape[-1],

hidden\_size\_list=[100, 50, 100, 50, 100], output\_size=10,

use\_dropout=True, dropout\_rate=dropout\_ratio,

use\_batchnorm=True, weight\_decay\_lambda=weight\_decay)

optimizer = AdaGrad(lr=learning\_rate)

# 모형 적합

for j in range(1000):

batch\_mask = np.random.choice(X\_train2.shape[0], batch\_size)

X\_batch = X\_train2[batch\_mask]

y\_batch = y\_train2[batch\_mask]

grads = network.gradient(X\_batch, y\_batch)

optimizer.update(network.params, grads)

print(f"learning\_rate: {learning\_rate}, \tweight\_decay: {weight\_decay}, \tvalidation accuracy: {network.accuracy(X\_val, y\_val)}")

y\_pred = np.argmax(network.predict(X\_train), axis=1)

print(network.accuracy(X\_train, y\_train))

y\_pred = np.argmax(network.predict(X\_test), axis=1)

print(network.accuracy(X\_test, y\_test))

* Hyper parameter 조정

- Hidden Layer: hidden layer num, hidden node size

- Init Weight: He 초깃값

- Dropout: 0.2

- Batch Normalization: True

- Activation function: ReLU

- Gradient Descent: AdaGrad

- Regularization: weight decay

Gradient Descent 관련해서 확인하던 차에 의도치 않게 hyper-parameter에 대한 validation accuracy 상당히 높은 값을 발견

* learning\_rate = 10 \*\* np.random.uniform(-3, 3)
* weight\_decay = 10 \*\* np.random.uniform(-4, 4)
* hidden\_size\_list=[100, 50, 100, 50, 100]
* Gradient Descent: AdaGrad

learning\_rate: 0.0032656217201229008, weight\_decay: 0.005663252747611342,

validation accuracy: 0.8285

learning\_rate: 0.452835940270636, weight\_decay: 0.001314586450845189,

validation accuracy: 0.7043333333333334

일단 최적의 learning rate, lambda를 찾아보기로 함.

* learning\_rate = 10 \*\* np.random.uniform(-4, 1)
* weight\_decay = 10 \*\* np.random.uniform(-4, 0)

learning\_rate: 0.027672604546278616, weight\_decay: 0.19742363724128992,

validation accuracy: 0.71275

learning\_rate: 0.031228595772647624, weight\_decay: 0.004082432849215927,

validation accuracy: 0.8446666666666667

learning\_rate: 0.08034450055388272, weight\_decay: 0.0009445248622614722,

validation accuracy: 0.8149166666666666

learning\_rate: 0.031996324549654794, weight\_decay: 0.00016308424687448787,

validation accuracy: 0.8595833333333334

learning\_rate: 0.0014686869839965423, weight\_decay: 0.005932047090716765,

validation accuracy: 0.767

* learning\_rate = 10 \*\* np.random.uniform(-3, -1)
* weight\_decay = 10 \*\* np.random.uniform(-4, -1)

learning\_rate: 0.05266372254753123, weight\_decay: 0.07494896971957984, validation accuracy: 0.5326666666666666

learning\_rate: 0.0832670688617266, weight\_decay: 0.05060831392636402, validation accuracy: 0.5613333333333334

둘 다 값이 클 때를 제외하면 괜찮은 평균 0.8 즈음의 정확도를 보임

* test accuracy 확인

y\_pred = np.argmax(network.predict(X\_test), axis=1)

img\_show(X\_test, y\_pred)

print(network.accuracy(X\_test, y\_test))

test accuracy: 0.86

앞서 구한 학습률과 람다 값으로 다른 모형 구성

* hidden\_size\_list=[100, 100, 100, 100, 100, 100, 100]
* use\_batchnorm=False

learning\_rate: 0.027805690338969544, weight\_decay: 0.000973469397161784,

validation accuracy: 0.7895833333333333

learning\_rate: 0.07716945951634449, weight\_decay: 0.0013097390008018112,

validation accuracy: 0.6528333333333334

learning\_rate: 0.07137679308054343, weight\_decay: 0.0017223832846123743,

validation accuracy: 0.71975

learning\_rate: 0.0020878638462957025, weight\_decay: 0.00026975304145417653,

validation accuracy: 0.703

Train accuracy: 0.71535 / Test accuracy: 0.7014

일부러 은닉층과 은닉층의 노드를 늘리면서 정확도를 높이고자 했는데, 이전 모형과 정확도에 확실히 차이가 있는 것으로 보인다. batch normalization이 정확도를 높이는데 기여하는 것으로 보인다.

* dropout을 이용하지 않음.

learning\_rate: 0.007530003900423157, weight\_decay: 0.0022057686145189608,

validation accuracy: 0.87525

learning\_rate: 0.0037494390280371644, weight\_decay: 0.0002896994253184336,

validation accuracy: 0.86625

learning\_rate: 0.003484488909184922, weight\_decay: 0.0030176308125896955,

validation accuracy: 0.861

learning\_rate: 0.0023282558885536833, weight\_decay: 0.05037134834449611,

validation accuracy: 0.855

Train accuracy: 0.85681666 / Test accuracy: 0.839

dropout도 이용하지 않았을 때보다 체감상 학습이 조금 느려진다. 정확도 측면에서 과대적합은 일어나지 않은 것으로 보인다.

* Gradient Descent: Adam

바꾼 것이라곤 optimizer로 AdaGrad를 이용하던 걸 Adam으로 바꾼 것밖에 없는데, 학습률이나 람다 값을 AdaGrad를 중심으로 설정한 값이라 그런지 validation accuracy가 상당히 떨어진다. 혹시나 해서 학습 횟수도 늘려봤는데 여전히 accuracy가 낮다.

* Activation function: Sigmoid
* Init Weight: Xavier 초깃값

learning\_rate: 0.010435555614413346, weight\_decay: 0.0016788230303147903,

validation accuracy: 0.8045833333333333

learning\_rate: 0.0013239241176477784, weight\_decay: 0.08158533834432241,

validation accuracy: 0.60175

learning\_rate: 0.005526934025531127, weight\_decay: 0.016823037262989535,

validation accuracy: 0.7209166666666667

learning\_rate: 0.06697402064937605, weight\_decay: 0.0806458064359064,

validation accuracy: 0.23758333333333334

수치상으로 그렇게 크지 않은 값임에도 여전히 학습률과 람다 둘 다 크면 안 된다.

Train accuracy: 0.676533 / Test accuracy: 0.6721

가중치가 0에 가까운 값이 기울기 소실로 사라지기도 하지만, sigmoid의 특성상 완벽히 0이 되지 않을 수 있어서 그런지 정확도가 상대적으로 낮은 편이다.

여러 번 시도해본 결과, 제일 처음에 AdaGrad를 이용하고, ReLU를 이용한 모형의 test 정확도가 가장 높다.

이미 0.8의 정확도를 보이긴 하지만, 이미지 데이터이기 때문에 CNN을 이용하면 좀 더 높은 정확도를 기대할 수 있을 것. (Trainer, Convolution class 이용)

Adam은 beta2에 의해 overflow가 일어난다. 이를 제어하기보다 AdaGrad를 이용.

Trainer class를 이용해서 학습 시 loss가 계속 출력된다. 이때 loss는 한 epoch마다 그 범위나 분포가 크게 달라지지 않는데, epoch마다 accuracy는 조금씩이라도 증가했다.

이미 epoch 4에서 train accuracy = 0.908, test accuracy = 0.88를 달성했고, epoch 11에서 test accuracy가 0.9를 넘어섰으며 이후에도 비슷한 값을 유지한다.

Loss는 전체적으로 [0.1, 0.4]에서만 관측되고, 결과적으로 이미지 데이터는 CNN 모형이 더 학습이 잘 된다.