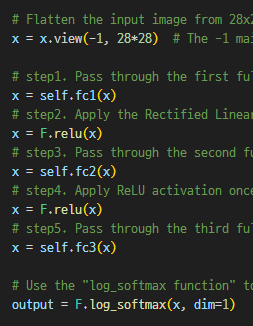
**Advanced Coding Practice HW Problem #5**

**20191599 송경호**

**A) Default**

다양한 분석을 위해 우선 문제에서 제시된 조건인 **① The MLP has 2 hidden layers ② Activation Function : ReLU**를 만족하도록 기본 구현을 완료하고 실행한다.



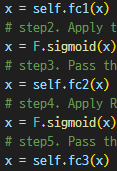
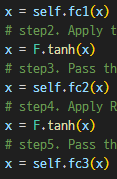
layer를 통과 시키고 activation function으로 Relu를 적용시키는 작업을 반복하여 구현했다. 추가적으로 매 학습의 시간이 과하게 걸려 다수의 데이터를 확보하기 어려운 점을 해소하기 위해 에포크를 3으로 설정했다. 이외의 조건들은 모두 유지한 상태로 *Average loss*와 *Accuracy* 그리고 *Total training and evalution time*을 구했다. 후에 진행되는 분석은 위 세 가지 항목을 기준으로 기본 구현과 비교한다.



**B)** analysis based on **“Activation function”**

**B-1) Sigmoid & Tanh**

Activation function으로 아래와 같이 Tanh와 Sigmoid를 사용하여 세 비교항목에 대한 결과를 구하고 default 구현과 비교한다.

** **

다음은 그에 대한 결과이다. (위) sigmoid (아래) tanh





|  |
| --- |
| ① Average loss : ReLU < Tanh < Sigmoid  ② Accuracy : ReLU > Tanh > Sigmoid  ③ Running time : ReLu < Tanh < Sigmoid |

Average loss와 Running time은 작을수록, Accuracy는 클수록 성능이 좋은 것이므로 결과적으로 ReLU, Tanh, Sigmoid 순으로 좋은 성능을 보이는 것을 알 수 있다.

결과 분석을 위해 Sigmoid와 Tanh function의 형태 그리고 두 함수를 도함수의 형태 살펴본다.

라인, 그래프이(가) 표시된 사진

자동 생성된 설명라인, 그래프, 도표, 번호이(가) 표시된 사진

자동 생성된 설명

Sigmoid 함수를 활성화 함수로 사용할 경우 크게 두가지 문제가 발생한다. 먼저 Sigmoid 함수는 y축을 대칭으로 입력 값이 작아질수록 0에, 커질수록 1에 수렴하는 형태이기 때문에 함수 값의 중심이 0이 아니다. 이때, 함수 값의 중심이 0이 아닐 경우 가중치가 지그재그의 형태로 업데이트가 되어 학습 시간이 길어진다는 문제점이 발생한다. 또한 Sigmoid 함수의 미분된 형태를 보면 알 수 있듯이 미분 계수의 최댓값이 굉장히 작을 뿐더라, 입력값이 커질수록 미분 계수가 0으로 수렴하는 Vanishing Gradient Problem이 발생한다.

Tanh function은 Sigmoid function의 변형함으로써 Sigmoid 함수와 달리 함수값의 중심을 0에 맞춘다. 또한 미분 계수의 최댓값 역시 크기 때문에 따라서 Sigmoid 함수에 비해 최적화 성능이 좋다. 그러나 미분된 형태를 보면 알 수 있듯이 여전히 vanishing gradient problem이 발생한다.

이러한 원인으로 인해 두 함수를 활성화 함수로 사용했을 때 ReLU에 비해 성능이 떨어질 수 있으며, Tanh function은 Sigmoid function을 보완한 형태인 만큼 두 함수 사이에도 성능의 차이가 발생함을 알 수 있다,

텍스트, 라인, 폰트, 그래프이(가) 표시된 사진

자동 생성된 설명

추가적으로 ReLU함수는 데이터가 주로 양수 값인 경우 더 좋은 성능을 보이는데, MNIST 데이터 셋은 픽셀값이므로 0에서 255의 값 중 하나를 데이터로 가진다. 다시 말해 모든 데이터 셋이 양수로 이루어져 있어 ReLU의 성능이 더욱 좋게 발휘되는 데이터셋이다.

이러한 다양한 이유에 있어 ReLU의 성능이 tanh와 sigmoid 함수에 비해 좋게 발휘된 것으로 해석된다.

**B-2) LeakyReLU with c = 0.01**

Activation function으로 아래와 같이 leaky\_relu를 사용한다.

**텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

다음은 그에 대한 결과값으로 ReLU의 결과값과 거의 차이가 없는 것을 확인할 수 있다.

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

이 역시 LeakyReLU 함수의 그래프를 통해 결과를 해석한다.

텍스트, 라인, 스크린샷, 도표이(가) 표시된 사진

자동 생성된 설명

LeakyReLU는 a = max(0.01z, z)라는 함수를 가진다. 따라서 ReLU와 달리 입력 값이 음수일 때 단순히 결과값이 0이 아닌 0.01z이 되며 그래프에서도 음수 입력에 대해 0에서 그래프가 간소하게 벗어나는 것을 확인 할 수 있다.. 그러나 앞서 말한 것처럼 MNIST 데이터셋은 오직 양수의 데이터만을 가지고 있으므로 leakyReLU는 ReLU와 동일한 역할을 하게된다. 따라서 결과값에 큰 차이가 발생하지 않는다.

**C)** analysis based on **“Learning rate”**

python mnist.py 명령어와 함께 learning rate를 입력하고 해당 learning rate에 변화를 주며 세 가지 비교 항목에 대해 관찰한다. 다음은 learning rate에 따른 비교 항목들의 결과값을 정리한 표이다.

|  |  |  |  |
| --- | --- | --- | --- |
| **Learning Rate** | **Average Loss** | **Accuracy(%)** | **Running Time (sec)** |
| 0.001 | 1.2212 | 76 | 71 |
| 0.005 | 0.3339 | 90 | 64 |
| 0.01 | 0.262 | 92 | 75 |
| 0.03 | 0.1616 | 95 | 70 |
| 0.07 | 0.1077 | 96 | 71 |
| 0.1 | 0.1003 | 97 | 71 |
| 0.2 | 0.0913 | 97 | 73 |
| 0.4 | 0.1153 | 97 | 74 |
| 0.6 | 0.1585 | 96 | 76 |
| 0.7 | 0.7355 | 72 | 80 |
| 0.8 | 0.5588 | 79 | 74 |

Learning rate는 모델에서 가중치를 업데이트할 때, 업데이트를 얼만큼 크게 할지에 대해 결정하는 변수이다, 따라서learning rate를 적절히 조절하는 것은 모델의 성능에 직접적인 영향을 미친다. 결과를 정리하는 표를 보면 알 수 있듯이 Learning Rate를 0.001에서 0.03까지 증가하면 성능이 향상되는 것을 알 수 있다. 이후 0.03에서 약 0.6의 learning rate까지 비슷한 성능을 보이다가 그 이상의 값을 갖게 되면 다시 급격히 성능이 떨어지는 것을 알 수 있다.

지나치게 낮은 학습률은 가중치 업데이트의 측면에서는 안정적일 수 있지만 적합한 모델에 수렴하는 시간이 오래 걸리게 된다. 때문에 결과적으로 평균 손실이 크게 줄지 않게 된다. 또한 모델에 대한 학습이 느리게 발생하기 때문에 정확도 역시 낮아진다.

반대로 지나치게 높은 학습률은 모델의 슈렴이 아닌 오버슈팅을 초래할 수 있다. 결과적으로 학습이 불안정해지며 평균 손실이 급격하게 증가하고 오버슈팅으로 인해 특정 데이터에 대한 정확도만 높아져 일반화 능력이 떨어지게 된다.

결과적으로, 해당 모델은 약 0.1 ~ 0.6 사이의 학습률을 가지는 것이 가장 적합하다는 것을 알 수 있다.

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

**텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명**

**D)** analysis based on **“Depth of hidden layers”**

****

****

위와 같이 새로운 hidden layer를 추가하고 해당 layer를 한번 더 거치도록 구현한다. 구현 결과는 다음과 같다.

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

hidden layer의 depth를 증가시키게 되면 모델의 표현력이 향상된다. 결과적으로 더욱 복잡한 패턴에 대한 학습력이 올라가게 되며 정확도 역시 증가한다. 그러나 학습률과 마찬가지로 hidden layer가 너무 깊은 depth를 갖게 되는 경우 hidden 는 overfitting을 발생하여 loss가 증가하고 정확도가 떨어질 가능 성이 있다.

hidden layer를 추가했음에도 accuracy가 낮아졌기 때문에 확실한 확인을 위해 하나의 hidden layer을 더 추가하여 결과를 확인했고 아래와 같이 성능이 더욱 악화된 것을 확인할 수 있었다.

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

또한 반대로 hidden layer를 한번 줄이는 구현을 수행했는데 오히려 성능이 좋아지는 것을 확인할 수 있었다. 결론적으로 해당 모델은 한 개의 hidden layer 만을 가지고 있을 때가 최적이라는 결론을 냈다.

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

hidden layer를 늘렸을 때 모델의 복잡성을 충분히 이해하기 위해서는 충분한 데이터와 더 많은 에포크가 요구될 수 있다. 때문에 성능의 측정의 시간을 줄이기 위하여 에포크를 3으로 줄였기 때문에 hidden layer를 줄였을 때 오히려 성능이 향상되고 hidden layer를 늘렸을 때 성능이 감소했을 가능성 역시 존재한다.

**E)** analysis based on **“Hidden layer width”**

아래와 같이 Hidden layer의 너비를 조절하여 결과를 확인한다.

**텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명**

기존에 비해 average loss는 줄어들었지만 동시에 실행 시간이 증가한 것을 알 수 있다. hidden layer의 너비를 늘리는 것은 깊이를 늘리는 것과 마찬가지로 모델에 대한 심화된 파악을 가능하게 해주며 데이터의 보다 복잡한 패턴을 표현 가능하다.

그러나 동시에 overfitting의 가능성의 증가하고 무엇보다도 기본적인 연산량이 증가하기 때문에 학습 시간이 증가한다.

**F) 결과 분석**

모든 모델에서 절대적인 성능을 나타내는 값이란 없다. 활성화 함수의 경우 데이터에 특성에 따라 다른 성능이 발생할 수 있으며 leargning rate, hidden layer와 depth와 width 모두 적절한 값이 존재하기 때문에 이들을 잘 조절하며 학습하는 것이 중요하다. 또한 모델의 복잡도에 대한 분석이 심화될수록 모델에 대한 이해가 증가하여 정확도가 증가할 수 있지만 동시에 학습 시간이 오래 걸린다는 trade off가 발생한다. 따라서 이러한 여러 가지 사항을 고려해야 한다.