|  |  |  |  |
| --- | --- | --- | --- |
| **2023-1 머신러닝 및 실습** | | | |
| 학번 | 2021136150 | 이름 | 양동재 |

이번 실습에서는 인공신경망에서 활성화 함수를 다양하게 사용해보고, 각각의 함수가 어떤 현상을 발생시키는지 분석하고자 합니다. 이를 통해 활성화 함수가 인공신경망 학습에 어떤 영향을 미치는지 이해할 수 있습니다.

인공신경망에서는 활성화 함수를 통해 비선형성을 추가하여 모델링합니다. 이를 통해 인공신경망은 다양한 데이터 패턴을 학습할 수 있습니다. 활성화 함수를 다양하게 사용하면, 인공신경망이 다양한 데이터 패턴을 학습할 수 있습니다.

또한, 데이터 특성 및 학습 환경에 따라 적절한 활성화 함수를 선택하는 것이 중요합니다. 예를 들어, 입력 데이터가 양수인 경우 ReLU 함수를 사용하는 것이 적절합니다. 반면, 입력 데이터가 음수인 경우 Leaky ReLU 함수를 사용하는 것이 더 좋은 결과를 얻을 수 있습니다. 또한, 다중 클래스 분류 문제에서는 Softmax 함수를 사용하는 것이 일반적입니다.

**따라서, 이번 실습에서는 다양한 데이터와 학습 환경에서 다양한 활성화 함수를 사용하여 모델의 학습 및 성능을 분석하고, 어떤 데이터 특성 및 학습 환경에서 어떤 활성화 함수를 선택하는 것이 적합한지에 대한 이해를 높이는 것이 목표입니다.**

**[데이터]**

Shape, arrow

Description automatically generatedMNIST 데이터셋은 70,000개의 손글씨 숫자 이미지 데이터셋으로, 각 이미지는 해당 숫자와 함께 라벨링되어 있으며, 이미지 분류 작업의 벤치마크 데이터셋으로 광범위하게 활용되고 있습니다. 데이터셋의 이미지는 28x28 크기의 그레이스케일로, 중앙에 위치하며, 이 데이터셋은 60,000개의 훈련 이미지와 10,000개의 테스트 이미지로 구성되어 있습니다. MNIST 데이터셋은 머신러닝 분야에서 많은 알고리즘, 특히 로지스틱 회귀, 신경망, 서포트 벡터 머신 등의 성능 평가에 사용됩니다. 이러한 알고리즘은 MNIST 데이터셋을 통해 이미지 분류와 같은 다양한 문제를 해결하는 데 활용됩니다.

**[문제1]**

1. **문제정의**

이번 실습 문제는 손으로 쓴 숫자 이미지를 인식하는 2-Layer Multiclass Classification 모델에 다양한 Activation Function을 적용하여, 각각의 차이점을 분석하고 이해하는 것을 목표로 합니다.

이를 위해 우선 Neural Network의 Forwarding, Backpropagation, Loss Function 등 기본 개념에 대한 이해가 필요합니다. 또한, Activation Function의 수학적 의미와 각 함수의 특성을 명확하게 이해해야 합니다.

실습에서는 다양한 Activation Function을 적용하여 모델의 학습 및 성능을 비교하고, 어떤 Activation Function이 어떤 데이터 특성에 더 적합한지 분석합니다. 이를 통해 데이터 특성 및 학습 환경에 따라 적절한 Activation Function을 선택하는 방법에 대해 학습하게 됩니다.

따라서, 이번 실습 문제는 Neural Network의 기본 개념과 Activation Function의 수학적 의미와 특성을 이해하고, 다양한 Activation Function을 적용하여 모델의 학습 및 성능을 분석하여 데이터 특성에 따라 적절한 Activation Function을 선택하는 방법을 익히는 것이 목표입니다.

1. **학습전략**

**A. Hypothesis**

1. 데이터 전처리: MNIST 데이터셋을 로드하고, 이미지 데이터를 벡터화하여 전처리합니다.

2. 모델 구현: 2-Layer Multiclass Classification 모델을 구현합니다. 모델은 입력층, 은닉층, 출력층으로 구성되며, 활성함수로는 ReLU, Sigmoid, TanH, Leaky ReLU를 사용합니다. 이때, 각 활성함수에 대한 forward, backward 함수를 구현합니다.

3. 모델 학습: 구현한 모델을 학습합니다. 이때, Cross-Entropy Loss Function을 사용합니다. 학습은Gradient Descent를 이용하며, Hyperparameter로는 Learning Rate, Epochs를 조정합니다.

4. 모델 평가: 학습된 모델의 성능을 평가합니다. 이때, Accuracy와 Loss 값을 기준으로 각 Activation Function에 대한 성능을 비교합니다.

5. 결과 분석: 각 Activation Function에 대한 성능을 분석하고, 데이터 특성에 따라 적합한 Activation Function을 선택하는 방법을 학습합니다.

위와 같은 학습 전략을 통해, ReLU, Sigmoid, TanH, Leaky ReLU Activation Function을 구현하고, 각 Activation Function에 대한 모델 학습 및 성능 평가를 수행합니다. 이를 통해 어떤 Activation Function이 어떤 데이터에 더 적합한지 분석하고, 적합한 Activation Function을 선택하는 방법을 익힐 수 있습니다.

**B. EDA**

본 데이터셋의 특징은 60,000장의 손글씨 이미지 데이터로 28\*28의 해상도이며 총 784개의 feature를 갖고있습니다.

1. **자신이 구현한 함수 및 클래스에 대한 설명 또는 유도 과정 작성**

Activation function은 neural network에서 linear function의 결과값을 non-linear한 형태로 변환하여 다음 layer의 입력값으로 사용하는 역할을 합니다. 이를 통해 neural network가 비선형 관계를 학습할 수 있게 되며, 이러한 nonlinearity가 neural network의 표현력을 향상시키는 역할을 합니다.

보통 activation function은 입력값을 정규화(normalize)하여 출력값을 0과 1 사이의 값으로 매핑시킵니다. 이렇게 0과 1 사이의 값을 갖게된 출력값은 다음 layer에서 입력값으로 사용됩니다. 따라서 activation function은 neural network에서 비선형 관계를 학습하는 데 매우 중요한 역할을 합니다.

주요한 activation function으로는 Sigmoid, ReLU(Rectified Linear Unit), TanH(Hyperbolic Tangent), Leaky ReLU 등이 있으며, 이들은 각각의 특징과 장단점이 있습니다. 이러한 activation function 중에서 어떤 것을 선택하느냐는 neural network의 특성과 데이터 특성, 그리고 학습 환경에 따라 달라질 수 있습니다.

따라서 activation function의 선택은 neural network의 성능에 매우 큰 영향을 미치므로, 이를 적절히 선택하는 것이 neural network의 성능을 개선하는 데 매우 중요합니다.

|  |  |
| --- | --- |
| Activation function | Formula |
| Relu |  |
| Sigmoid |  |
| TanH |  |
| Leaky Relu |  |

**4. 결과 비교 및 분석**

|  |  |
| --- | --- |
| Activation function | Performance(Acc) |
| ReLU | 0.786 |
| Sigmoid | 0.3685 |
| TanH | 0.544 |
| Leaky Relu | 0.7803 |

MINIST 데이터는 손으로 쓴 숫자 이미지를 포함하는 데이터셋으로, **이미지의 특성은 비교적 단순하고 분류하기 쉬운 특징을 가지고 있습니다.** 이러한 특성은 ReLU와 같은 activation function이 성능을 높이는 데 도움을 줄 수 있습니다.

ReLU: ReLU는 입력값이 0보다 작을 경우 0을 출력하고, 0보다 큰 경우에는 그 값을 그대로 출력하는 activation function입니다. MINIST 데이터의 이미지는 주로 검정색과 흰색의 contrast가 높기 때문에 ReLU와 같이 높은 contrast를 갖는 이미지에서 더 높은 성능을 보일 수 있습니다.

Sigmoid: Sigmoid는 입력값이 어느 정도 큰 경우에는 saturation되어 gradient vanishing 문제가 발생할 가능성이 높습니다. 따라서 MINIST 데이터와 같이 비교적 단순한 이미지에서는 다른 activation function들에 비해 성능이 떨어질 수 있습니다.

TanH: TanH는 Sigmoid와 비슷한 형태의 함수이지만, 출력값의 범위가 [-1, 1]로 -1과 1 사이의 값을 갖습니다. 이러한 함수는 입력값이 크거나 작은 경우에도 gradient가 0이 되지 않기 때문에, Sigmoid보다는 성능이 더 우수할 수 있습니다.

Leaky ReLU: Leaky ReLU는 ReLU와 비슷한 형태를 갖지만, 입력값이 0보다 작을 경우 일정한 기울기를 갖는 함수입니다. 이러한 함수는 입력값이 0보다 작은 경우에도 gradient가 0이 되지 않기 때문에, ReLU와 비교해서 성능이 더 우수할 수 있습니다.

**따라서 MINIST 데이터와 같이 비교적 단순한 이미지를 다룰 때에는 ReLU와 Leaky ReLU와 같은 activation function이 더 우수한 성능을 보일 수 있습니다.**

**Saturation : activation function의 입력값이 특정 범위를 벗어나면 출력값이 일정 수준 이상으로 고정되어버리는 현상 – sigmoid는 x가 극한값으로 갈수록 출력값이 아주 미미하게 변경됨**

**Gradient Vanishing : backpropagation에서 발생할 수 있는 문제로 activation function이 saturation state일 때, gradient score가 아주 작아져 the weight couldn’t updated.**

**[문제2]**

1. **문제정의**

이번 실습 문제는 인공지능 모델 학습 시에 batchsize를 적용해보는 것입니다. 우선적으로, batchsize란 모델이 학습 시에 한 번에 몇 개의 데이터를 처리하는지를 의미합니다. 이러한 batchsize를 적용하는 이유는 일괄적인 데이터 처리로 인해 모델이 데이터의 특징을 파악하고 학습하는 데 있어서 일정한 효과를 보일 수 있기 때문입니다.

이번 실습에서는 batchsize를 적용하여 stochastic gradient descent, mini batch gradient descent, 그리고 batch gradient descent 방식의 학습 모델을 구현합니다. Stochastic gradient descent는 전체 데이터 셋에서 무작위로 하나의 데이터 샘플만을 선택하여 학습을 진행하는 방식이며, mini batch gradient descent는 데이터 셋을 여러 개의 작은 batch로 나누어 각각의 batch를 사용하여 학습을 진행하는 방식입니다.

마지막으로, batch gradient descent는 전체 데이터 셋을 한 번에 사용하여 학습을 진행하는 방식입니다. 이 방식은 데이터 셋이 매우 큰 경우 모델 학습에 매우 많은 시간이 소요될 수 있지만, 전체 데이터를 사용하기 때문에 정확한 결과를 얻을 수 있습니다. 따라서, 이번 실습에서는 batchsize를 적용한 각각의 학습 방식의 장단점과 성능 차이를 비교해보고, 가장 적합한 방식을 선택하여 모델을 학습시켜보는 것이 목표입니다.

1. **학습전략**

**A. Hypothesis**

먼저, batch size는 학습에 사용되는 데이터를 나누는 단위로, 한 번의 학습에서 사용되는 데이터의 개수를 의미합니다.

Batch Gradient Descent는 전체 데이터셋을 한 번에 모두 사용하여 가중치를 업데이트하는 방식입니다. 이 방식은 모든 데이터를 고려하기 때문에 전체적인 특성을 파악할 수 있어서 정확도가 높지만, 계산량이 많기 때문에 학습 시간이 오래 걸리는 단점이 있습니다.

Stochastic Gradient Descent는 한 번의 학습에서 하나의 데이터만 사용하여 가중치를 업데이트하는 방식입니다. 이 방식은 계산량이 적어 빠른 속도로 학습이 가능하지만, 데이터의 노이즈에 민감하여 학습이 불안정할 수 있습니다.

Mini-Batch Gradient Descent는 Batch Gradient Descent와 Stochastic Gradient Descent의 중간 방식으로, 일정한 개수의 데이터를 묶어서 가중치를 업데이트하는 방식입니다. 이 방식은 Batch Gradient Descent보다 빠르면서도 Stochastic Gradient Descent보다 안정적인 학습이 가능합니다.

따라서, 이번 실습에서는 위 세 가지 방식을 각각 구현하여 비교해보고, 학습 데이터의 개수와 모델 복잡도에 따라 가장 적합한 방식을 선택하여 최적화된 학습 모델을 구현하는 것이 목표입니다.

**B. EDA**

본 데이터셋의 특징은 60,000장의 손글씨 이미지 데이터로 28\*28의 해상도이며 총 784개의 feature를 갖고있습니다.

1. **자신이 구현한 함수 및 클래스에 대한 설명 또는 유도 과정 작성**

**Stochastics Gradient Descent**

|  |
| --- |
| for each epoch:  shuffle(data)  for each mini-batch in data:  compute gradients of loss function w.r.t. parameters using the mini-batch  update parameters using the gradients and learning rate |

Stochastic gradient descent(SGD)의 의사코드는 매번 데이터 한 개씩 가져와 학습을 수행하는 방법입니다.

**Mini-batch gradient descent**

|  |
| --- |
| for each epoch:  shuffle(data)  for each mini-batch in data:  compute gradients of loss function w.r.t. parameters using the mini-batch  update parameters using the gradients, learning rate, and mini-batch size |

Mini batch gradient descent는 데이터를 작은 묶음으로 나누어 학습을 수행하는 방법입니다.

**Batch Gradient Descent**

|  |
| --- |
| for each epoch:  compute gradients of loss function w.r.t. parameters using all data  update parameters using the gradients and learning rate |

Batch gradient descent는 전체 데이터셋을 한 번에 학습하여 파라미터를 업데이트하는 방법입니다.

**4. 결과 비교 및 분석**

위의 결과에서, 배치 사이즈가 큰 경우(batch\_size 2500, batch gradient descent) 정확도가 높게 나왔고, 배치 사이즈가 작은 경우(stochastic gradient descent) 정확도가 낮게 나왔습니다. 이는 배치 사이즈가 작을수록 노이즈와 다양성이 높아져서 최적의 모델 파라미터에 수렴하는 시간이 더 오래 걸리고, 수렴에 대한 불안정성도 증가하기 때문입니다.

따라서, 배치 사이즈가 작으면 일반적으로 더 높은 학습 속도와 더 빠른 수렴을 가져올 수 있지만, 이를 위해서는 더 많은 epoch(학습 단계)가 필요합니다. 그리고 배치 사이즈가 큰 경우에는 더 적은 epoch를 사용할 수 있고, 수렴에 대한 안정성도 향상됩니다.

또한, batch gradient descent 방식에서는 한 번에 모든 데이터를 처리하기 때문에 메모리 사용량이 매우 크고, 수렴에 대한 불안정성도 존재할 수 있습니다. 따라서, 대규모 데이터를 처리하는 경우에는 mini-batch gradient descent 방식이 가장 효과적이며, 적절한 배치 사이즈를 선택하는 것이 중요합니다.

따라서 결과를 정리해보면 아래와 같습니다.

**Stochastic gradient descent: 매 epoch마다 한 데이터 포인트를 사용해 학습하므로 속도가 빠르지만 수렴이 불안정하고 노이즈에 민감함. 따라서 테스트 정확도가 낮음.**

**Mini-batch gradient descent (batch size = 2500): 데이터를 작은 묶음으로 나눠 학습하므로 노이즈에 대한 영향을 줄일 수 있고, Stochastic GD보다는 안정적인 수렴 가능. 따라서 테스트 정확도가 상승함.**

**Batch gradient descent: 전체 데이터셋을 한 번에 학습하므로 가장 안정적으로 수렴하며, 테스트 정확도도 가장 높음. 하지만, 학습 속도가 가장 느림.**

**[Extra 문제1]**

1. **문제정의**

이번 실습문제에서는 MINIST 데이터셋을 이용하여 Multi-class SVM 모델과 cross entropy 기반 모델을 구현해보는 것입니다. Multi-class SVM은 서포트 벡터 머신을 활용하여 다중 클래스 분류 문제를 해결하는 방법입니다. 반면, cross entropy 기반 모델은 softmax 함수를 이용하여 다중 클래스 분류 문제를 해결하는 방법입니다. 각 모델의 성능을 비교하고, 어떤 모델이 MINIST 데이터셋에 적합한지 파악해보는 것이 이번 실습의 목표입니다.

이번 실습문제를 해결하기 위해서는 먼저 SVM과 cross entropy, 그리고 softmax 함수의 개념을 이해해야 합니다. 또한 MINIST 데이터셋의 구성과 분류 문제의 특징을 파악하고, 모델의 학습과 평가를 위한 성능 지표에 대한 이해가 필요합니다. 따라서, SVM과 cross entropy, softmax 함수 등의 개념과 MINIST 데이터셋의 특징, 그리고 성능 지표에 대한 이해가 선행되어야 합니다.

1. **학습전략**

**A. Hypothesis**

이번 문제를 풀기 위한 학습 전략은 크게 세 가지로 나눌 수 있습니다.

첫째로, SVM과 cross entropy, 그리고 softmax 함수 등의 개념에 대한 이해가 필요합니다. SVM은 서포트 벡터 머신으로, 분류 경계선과 가장 가까이 있는 데이터 포인트들을 찾아내는 알고리즘입니다. Cross entropy는 다중 클래스 분류에서 사용되는 손실 함수로, 실제값과 예측값 사이의 차이를 계산하여 모델을 학습시킵니다. Softmax 함수는 출력값을 확률값으로 변환하여 다중 클래스 분류 문제를 해결하는 함수입니다. 이러한 개념들을 충분히 이해하고 있는 상태에서 모델을 구현하고 학습시켜야 합니다.

둘째로, MINIST 데이터셋의 구성과 분류 문제의 특징을 파악해야 합니다. MINIST 데이터셋은 손글씨 숫자 이미지 데이터셋으로, 각 이미지에는 0부터 9까지의 숫자 중 하나가 라벨링되어 있습니다. 이러한 데이터셋에서 모델을 학습시키기 위해서는 데이터 전처리가 필요합니다. 예를 들어, 이미지 데이터를 벡터 형태로 변환하고, 정규화를 수행해야 합니다. 또한, 데이터셋을 학습용과 검증용으로 나누어 모델의 성능을 평가할 수 있도록 해야 합니다.

셋째로, 모델의 학습과 평가를 위한 성능 지표에 대한 이해가 필요합니다. 분류 문제에서 일반적으로 사용되는 성능 지표로는 정확도, 정밀도, 재현율, F1-score 등이 있습니다. 모델의 학습이 끝난 후, 검증용 데이터셋을 사용하여 모델의 성능을 평가하고, 이러한 성능 지표들을 계산하여 최종 모델의 성능을 결정해야 합니다. 하지만 본 과제를 수행하기 위해서는 미리 가정한 accuracy로 model evaluation 을 수행할 예정입니다.

따라서, SVM과 cross entropy, softmax 함수 등의 개념을 충분히 이해하고, MINIST 데이터셋의 구성과 분류 문제의 특징을 파악하며, 모델의 학습과 평가를 위한 성능 지표에 대한 이해가 필요합니다. 이러한 전략을 토대로, 모델을 구현하고 성능을 비교하여, MINIST 데이터셋에 적합한 모델을 결정할 수 있습니다.

**B. EDA**

본 데이터셋의 특징은 60,000장의 손글씨 이미지 데이터로 28\*28의 해상도이며 총 784개의 feature를 갖고있습니다.

1. **자신이 구현한 함수 및 클래스에 대한 설명 또는 유도 과정 작성**

Multi-class SVM은 neural network에서도 loss function으로 활용될 수 있습니다. 이때, SVM을 loss function으로 활용하는 방법은 다음과 같습니다.

먼저, SVM은 경계면과 각 데이터 샘플과의 거리를 최대화 하는 방식으로 분류를 수행합니다. 이를 neural network에서도 활용하기 위해서는, 각 클래스를 분류하는 결정 경계를 나타내는 가중치 행렬과 바이어스 벡터를 학습하는 것이 필요합니다. 이 가중치와 바이어스를 이용하여 입력 데이터를 클래스에 대응하는 점수(score)로 변환하고, 이 점수를 이용하여 각 클래스에 대한 손실(loss)를 계산합니다.

여기서 손실 함수는 다음과 같이 정의됩니다.

여기서 는 입력 데이터 i가 j번째 클래스에 속할 확률을 나타내는 점수이며, $y\_i$는 $i$번째 데이터 샘플이 실제로 속한 클래스입니다. 따라서, 는 실제 클래스에 대한 점수이며,는 margin을 나타내는 부분입니다. 이 margin 값은 SVM에서 사용되는 마진과 동일한 개념으로, 클래스 간의 간격을 의미합니다. 마지막으로, $L$은 모든 데이터 샘플에 대한 손실을 계산한 값입니다.

이러한 Multi-class SVM을 loss function으로 활용하여 neural network를 학습시킬 때에는, 손실 함수를 최소화하는 방향으로 가중치와 바이어스를 업데이트하면 됩니다. 이를 위해, 역전파(backpropagation) 알고리즘을 사용하여 각 가중치와 바이어스의 기울기(gradient)를 계산하고, 경사 하강법(gradient descent) 알고리즘을 이용하여 가중치와 바이어스를 업데이트합니다.

따라서, Multi-class SVM을 neural network에서 loss function으로 활용하는 방법은 다음과 같이 요약할 수 있습니다.

입력 데이터를 클래스에 대응하는 점수(score)로 변환하는 가중치와 바이어스를 학습합니다.

입력 데이터의 클래스를 결정하는 결정 경계를 학습하기 위해, Multi-class SVM의 손실 함수를 loss function으로 활용합니다.

손실 함수를 최소화하는 방향으로 가중치와 바이어스를 업데이트하여 neural network를 학습시킵니다.

**4. 결과 비교 및 분석**

위 코드에서 발생하는 문제점은 크게 두 가지로 볼 수 있습니다.

1. Softmax\_with\_MSVM 클래스의 구현 문제

2. 학습 파라미터 설정 문제

Softmax\_with\_MSVM 클래스에서는 MSVM(Maximum Margin Support Vector Machine) 알고리즘을 이용하여 Softmax를 구현합니다. 그러나 현재 구현된 코드에서는 Softmax와 MSVM를 혼동하여 사용하고 있습니다.

**Softmax\_with\_MSVM 클래스에서는 Softmax 출력 값을 self.softmax\_out에 저장하고, 이 값을 이용하여 MSVM을 계산합니다. 그러나 Softmax 출력 값이 아닌 x 값(선형 변환값)을 이용하여 MSVM을 계산하는 문제가 있습니다. 이로 인해 Softmax와 MSVM 간의 연관성이 사라지게 되며, Softmax가 올바르게 작동하지 않아 학습이 이루어지지 않고, exp overflow가 발생하는 문제가 발생합니다.**

**따라서 Softmax와 MSVM를 각각 정확하게 구현하여 클래스를 수정해야 합니다.**

학습 파라미터 설정 문제

또 다른 문제는 학습 파라미터의 설정입니다. 현재 코드에서는 학습률(learning rate), epoch 수, 배치 크기(batch size), 정규화(regularization) 하이퍼파라미터를 지정할 수 있습니다.

그러나 현재 학습 파라미터 설정에서는 적절한 값을 찾지 못하면서 발생하는 문제가 있습니다. 예를 들어, 학습률이 너무 크거나 작을 경우에는 학습이 수렴하지 않을 수 있으며, epoch 수가 적거나 배치 크기가 작으면 학습이 제대로 이루어지지 않을 수 있습니다.

따라서 학습 파라미터의 설정을 적절하게 조정해야 합니다. 학습 파라미터를 조정하는 방법으로는 grid search, random search, Bayesian optimization 등이 있으며, 이를 이용하여 최적의 학습 파라미터를 찾아야 합니다.

현재 코드 수정중에 있습니다.

**[Extra 문제1]**

1. **문제정의**

이번 실습에서는 multi-layer neural network를 구현하고, 2-layer와 비교 분석하는 것이 목표입니다. 이를 위해, 3-layer 이상의 neural network를 구현할 것이며, hidden layer의 dimension은 자유롭게 설정할 수 있습니다. 학습 방법으로는 Batch gradient descent를 사용할 예정입니다. 이를 통해, multi-layer neural network가 어떤 경우에 2-layer보다 더 좋은 성능을 보이는지를 분석하고, 딥러닝에서 hidden layer의 역할과 중요성에 대해 이해할 수 있도록 하는 것 입니다.

1. **학습전략**

**A. Hypothesis**

데이터셋 준비: 학습에 사용할 데이터셋을 로드하고, 전처리 작업을 수행합니다. 이번 실습에서는 MNIST 데이터셋을 사용할 것이며, 데이터를 정규화하고 train, validation, test 셋으로 분리합니다.

모델 구성: 3-layer 이상의 neural network를 구현하고, hidden layer의 dimension은 자유롭게 설정합니다. 이번 실습에서는 activation function으로 ReLU를 사용하고, output layer의 activation function으로 softmax를 사용합니다.

모델 학습: Batch gradient descent 방식을 사용하여 모델을 학습합니다. 학습률과 epoch 수, batch size 등을 조정하여 모델의 성능을 향상시키도록 합니다. 이번 실습에서는 cross-entropy loss를 사용하고, L2 regularization을 적용합니다.

모델 평가: 학습된 모델을 validation set 또는 test set으로 평가합니다. 이때, 모델의 성능을 측정하는 지표로 accuracy, precision, recall, F1-score 등을 사용할 수 있습니다. 2-layer와 3-layer 이상의 neural network를 비교하여 모델의 성능을 분석합니다.

모델 개선: 모델의 성능을 향상시키기 위해 다양한 방법을 시도합니다. 이번 실습에서는 hyperparameter tuning, dropout, batch normalization 등의 방법을 적용하여 모델의 성능을 향상시킬 수 있습니다.

**B. EDA**

본 데이터셋의 특징은 60,000장의 손글씨 이미지 데이터로 28\*28의 해상도이며 총 784개의 feature를 갖고있습니다.

1. **자신이 구현한 함수 및 클래스에 대한 설명 또는 유도 과정 작성**

Multilayer Neural Network에서 분류 문제를 해결하기 위해 사용되는 Loss Function 중 가장 널리 사용되는 것은 Cross-Entropy Loss Function입니다. 이 함수는 분류 문제에서 사용되는 Softmax 함수와 함께 사용되며, 각 클래스에 대한 확률 값을 계산하고, 이 확률 값과 실제 레이블 값 사이의 차이를 계산하여 Loss 값을 구합니다.

Cross-Entropy Loss Function은 다음과 같이 정의됩니다.

여기서 y\_ij는 i번째 데이터 샘플이 j번째 클래스에 속하는지 여부를 나타내는 이진 레이블(binary label)입니다. 즉, y\_ij=1인 경우 i번째 데이터 샘플이 j번째 클래스에 속하는 것이며, y\_ij=0인 경우에는 속하지 않는 것입니다. 또한, hat{y}\_{ij}는 i번째 데이터 샘플이 j번째 클래스에 속할 확률을 나타내며, Softmax 함수를 통해 계산됩니다.

Softmax 함수는 다음과 같이 정의됩니다.

여기서는 i번째 데이터 샘플이 j번째 클래스에 대한 점수입니다. 이 점수는 입력 데이터와 가중치의 곱에 바이어스를 더한 값으로 계산됩니다.

따라서, Multilayer Neural Network에서 Cross-Entropy Loss Function을 사용하여 분류 문제를 해결하기 위해서는 다음과 같은 과정을 거칩니다.

**4. 결과 비교 및 분석**

위 코드는 Multi-layer Perceptron을 사용하여 MNIST 데이터셋에 대해 학습하는 코드입니다. 코드의 핵심적인 부분은 forward pass와 backward pass를 구현한 부분입니다.

먼저 forward pass에서는 입력 이미지를 5개의 fully connected layer와 activation function을 통과시켜 최종 출력값을 구합니다. 이때 Softmax\_with\_Mul\_CrossEntropy 클래스를 사용하여 Softmax 함수와 Multi-Class Cross Entropy Loss 함수를 적용합니다.

이후 backward pass에서는 Multi-Class Cross Entropy Loss 함수에 대한 미분값을 구한 뒤, Softmax 함수의 역전파를 통해 마지막 출력층의 미분값을 구합니다. 그리고 5개의 fully connected layer와 activation function을 역순으로 통과시키면서 각각의 미분값을 구하고 이를 이용하여 가중치와 편향을 업데이트합니다.

위 코드를 실행하면 주어진 하이퍼파라미터에 따라서 MNIST 데이터셋에 대해 학습이 진행됩니다. 에폭이 진행됨에 따라서 손실함수 값이 감소하는 것을 볼 수 있습니다. 학습이 끝난 후에는 학습된 가중치와 편향을 저장한 모델을 반환합니다.

그러나 위 코드에는 몇 가지 문제점이 있습니다. 학습이 진행되면서 loss 값이 급격하게 증가하는 현상이 발생할 수 있습니다. 이는 Softmax 함수에서 overflow 혹은 underflow가 발생하기 때문입니다. 이를 해결하기 위해서는 Softmax 함수의 계산식에서 최대값을 빼주는 방식으로 구현해야 합니다.

또한, 이 코드는 SGD(Stochastic Gradient Descent) 방식을 사용하고 있으며, 이는 학습 속도가 느리다는 단점이 있습니다. 따라서 다양한 Optimizer 기법을 적용하여 학습 효율을 높일 수 있습니다.

**5. 고찰 및 느낀점**

* Exponential function에 대한 이해도로 overflow가 발생하는 원인을 명확히 규명행야 한다. 현재로써 model의 overflow를 확인할 수 있는 방법은 아래의 두가지 경우로 scope를 줄일 수 있다.
  + Function의 입력값이 너무 크거나 작아서 saturation 현상이 발생하는가 ? 단, 위 가정에 대 해서는 activation function 특성상 ReLU의 경우 saturation현상이 발생하지 않음으로 확인하는 정도로만 넘어가면 될 듯 하다.
  + 두번째 가정이 중요하다. 현재 Numpy 최신 버젼상 float8 즉 특정 소수점 이하로 내려가면 해당 데이터 타입에 대해서 지원하지 않는다. 과거에는 float 128까지 지원하였으나 현재 최신 numpy버젼인 1.24.5는 공식문서상 float8까지밖에 지원하지 않는다.
    - 위 경우를 고려했을때 back propagation을 수행할 때, 얼마나 작은 소수점의 숫자들이 연산되는지를 확인해볼 필요가 있을 것 같다.
* 컴퓨터는 특정 반복되는 연산을 방지하기 위해 다양한 솔루션을 제공한다. 특히 saturation과 같은 현상을 방지하기 위해서는 특정 계산값이 enduring할 때, 그 값을 nan value로 바꿔버린다. 해당 부분에 대해서 명확히 체크하고 코드를 수정할 필요가 있다.
* Extra quiz2의 경우 코드는 정확히 동작하나, hyper parameter tuning을 통해 local minimum에서 global minimum으로 넘어갈 수 있는 경우를 고려해볼 필요가 있다.
* Multi layer가 되거나 chain rule을 적용할게 많아질수록 고려해야할 사항이 많아지며 작은 계산의 오차가 전체 성능에 치명적인 영향을 준다. 따라서 사전에 잘 확인할 필요가 있다.