인공지능 2&3차 과제 – 컴퓨터과학부 2022920024 박동찬

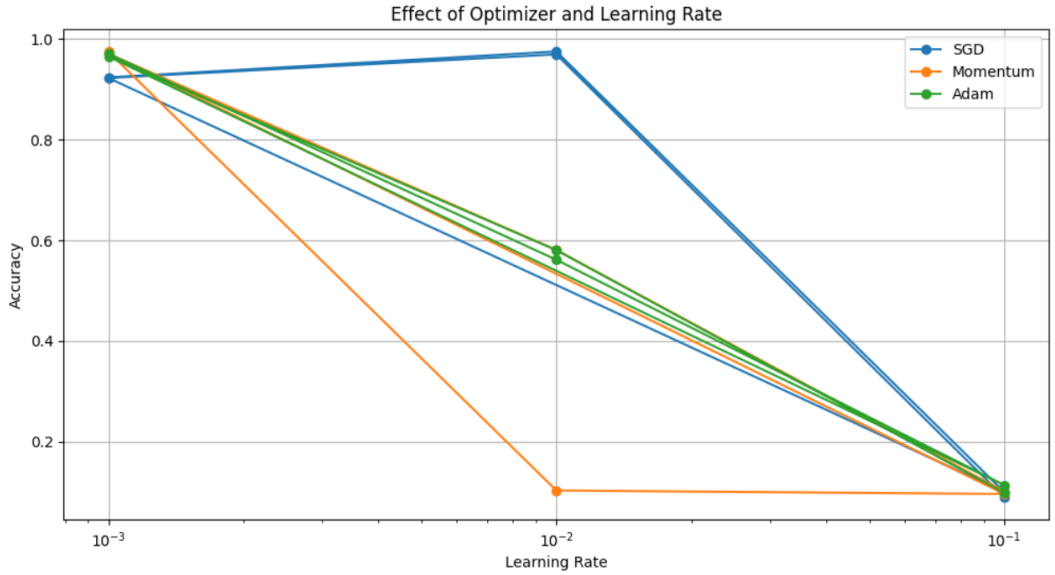
1. Fully-connected Neural Network
   A. 문제 분석 및 설계

      MNIST 데이터셋 내 이미지는 28*28 픽셀의 2D 이미지를 포함하고 있다. 이를 FCNN에 넣기 위해서 1차원 벡터 형태로 바꾸고 784개의 input node로 받아 처리하도록 한다. 또한 0부터 9까지 10개의 class로 분리해야 하므로 10개의 output node로 나오도록 만든다. Neural Network를 설계할 때, hidden layer 층 수, 각 hidden layer 층에 들어가는 node 수, learning rate, epoch, optimizer, loss function 등을 고려해야 한다. Loss function은 분류 문제에서 많이 사용하고 Softmax 출력에 효과적인 Cross-Entropy Loss를 사용하고 Activation function은 sigmoid와 softmax를 사용한다. Optimizer로는 SGD, Momentum, Adam을 사용하고 hidden layer는 [784, 128, 64, 10], [784, 256, 128, 10], Learning rate는 0.1, 0.01, 0.001을 사용해 이것들을 Grid Search 하여 가장 높은 Accuracy가 나오는 조합을 찾아낸다. MNIST는 소규모 데이터셋이고 깊지 않은 FCNN을 테스트하므로 빠르게 수렴하기 때문에 작은 epoch인 10을 택하여 실험해본다.

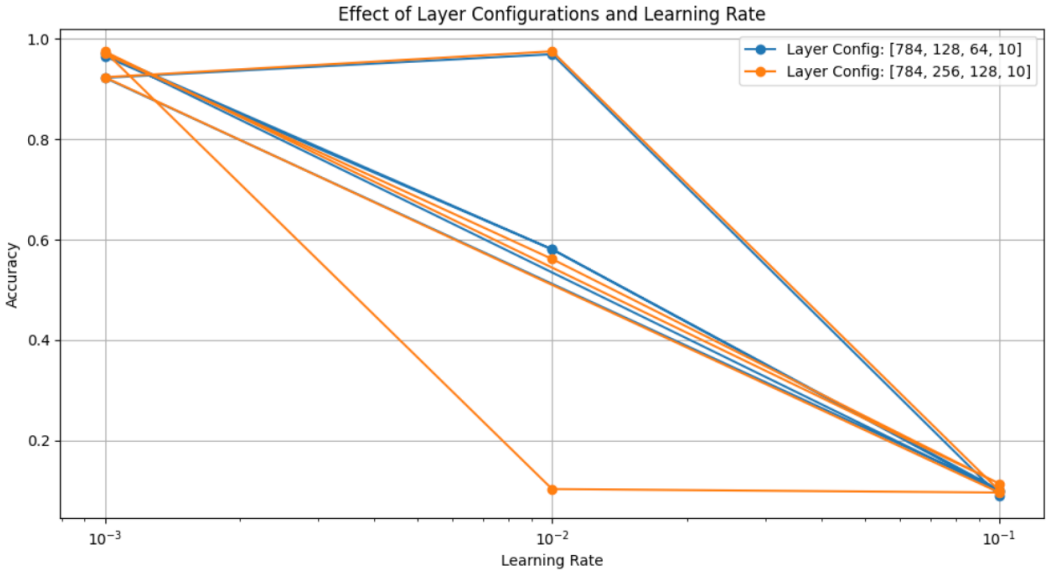
   B. 실험 결과

      i. Optimizer: [SGD, Momentum, Adam], Hidden Layer: [784, 128, 64, 10], [784, 256, 128, 10], Learning Rate: [0.1, 0.01, 0.001]을 Grid Search한 결과

         1. Optimizer & Learning Rate와 Accuracy의 관계



         2. Hidden Layer Config & Learning Rate와 Accuracy의 관계



C. 분석

MNIST 데이터셋을 FCNN으로 분류했을 때 Learning Rate가 Accuracy에 큰 영향을 끼친다는 결과를 볼 수 있다. Learning rate: 0.1로 설정한 경우, 대부분 Accuracy가 낮은데 이는 학습률이 너무 커서 발산했거나 최적점 근처에서 진동하는 경향을 보일 수 있기 때문이다. Learning rate: 0.01의 경우, 대부분의 조합에서 높은 정확도를 보였고 SGD나 Momentum Optimizer에서 각각 0.9748, 0.9744로 높은 성능을 보였다. Learning rate: 0.001의 경우, 대부분의 조합에서 높은 정확도를 보이고 안정적인 학습을 보였다. Learning rate: 0.01에서의 최적인 성능보다는 조금 낮은데 이는 학습 속도가 느려 최적점에 도달하기 전에 학습이 끝날 수 있고 Local Minima에 빠질 수 있다. Optimizer에 따른 Accuracy의 차이는 크지 않았고 SGD를 사용한 경우에서 최적인 성능이 도출되었다. Hidden Layer Config에 따른 Accuracy의 차이도 크지 않았으며 복잡한 layer 구성이 조금 더 높고 안정적인 성능을 보였다.

2. Convolutional Neural Network

A. 문제 분석 및 설계

28*28*1 -> Conv1 -> Max Pooling -> Conv2 -> Max Pooling -> Flatten -> FC1 -> FC2의 층을 가진 CNN을 구성한다. Conv 1의 경우, Filter 크기를 3*3*1이나 5*5*1으로 정하고 Stride는 1으로 정했다. Conv 2는 3*3*32나 5*5*32로 정하고 Stride는 1으로 정했다. 2번째 Max pooling 하면 5*5*64가 되는데 이를 Flatten 하여 FC에 적용한다. Activation function은 CNN에서 자주 사용되는 ReLU를 사용했으며 FC2에서는 Softmax를 사용한다. Optimizer로 [SGD, Momentum, Adam]을 사용하고 batch size는 [16, 32], learning rate는 [0.1, 0.01, 0.001]로 적용해보며 Accuracy와 어떤 관계가 있는지 분석한다. 현재 가지고 있는 컴퓨팅 자원이 좋지 않아 연산 속도가 매우 느려 dataset sampling을 통해 6000개로만 학습하였다. 또한 MNIST 데이터셋이 작아 빠르게 수렴하므로 epochs는 5로 설정했으며 컴퓨팅 자원이 좋지 않아 더 많은 epoch는 사용해보지 못했다.

B. 실험 결과

i. Optimizer: SGD, batch size: [16, 32], learning rate: 0.01, epochs: 5, weight initialize: random

Batch size를 바꾸며 실험했는데 Loss 값이 거의 변하지 않고 Accuracy가 약 0.11정도 나왔다. 이는 거의 랜덤으로 추측한 수준이며 학습이 잘 되고 있지 않음을 뜻한다.

ii. Optimizer: SGD, batch size: 16, learning rate: 0.01, epochs: 5, weight initialize: HE

가중치 초기화 HE를 적용했는데 Accuracy가 약 0.100이다. 이는 거의 랜덤으로 추측한 수준이며 학습이 잘 되고 있지 않음을 뜻하고 가중치 초기화와는 별로 관계가 없음을 뜻한다.

iii. Optimizer: SGD, batch size: [16, 32], learning rate: 0.1, epochs: 5, weight initialize: HE

두 경우 모두 Loss가 2.55로 시작했는데 이는 Cross-Entropy에서 거의 균일한 확률 분포에 가까울 때 나타나는 값이다. 즉, 모델이 랜덤하게 예측하고 있으며 loss가 변화하지 않고 Accuracy가 약 0.1이 도출되었다. 학습이 잘 되고 있지 않다.

iv. Optimizer: ADAM, batch size: 32, learning rate: 0.1, epochs: 5, weight initialize: HE

초기 epoch부터 Loss가 매우 크고 loss가 거의 변하지 않는다. Accuracy가 약 0.1이 도출되며 랜덤 추측한 수준이다. 즉, 학습이 되지 않았다.

v. Optimizer: SGD, batch size: 32, learning rate: 0.1, epochs: 5, weight initialize: HE

초기 epoch부터 Loss가 매우 크고 loss가 거의 변하지 않는다. Accuracy가 약 0.1이 도출되며 랜덤 추측한 수준이다. 즉, 학습이 되지 않았다.

vi. Optimizer: ADAM, batch size: 32, learning rate: 0.001, epochs: 5, weight initialize: HE

초기 epoch부터 Loss가 매우 크고 loss가 거의 변하지 않는다. Accuracy가 약 0.1이 도출되

며 랜덤 추측한 수준이다. 즉, 학습이 되지 않았다.

vii. Optimizer: SGD, batch size: 32, learning rate: [0.1, 0.01, 0.001], epoch: 5, weight initialize: HE, Filter Size: 5*5

Learning Rate가 0.1, 0.01인 경우는 Loss가 크고 Accuracy가 약 0.1이 도출되며 랜덤 추측한 수준이다. Learning Rate가 0.001인 경우는 Epoch 1부터 Loss가 안정적으로 감소하며 Accuracy가 약 0.80이 도출되었다.

viii. Optimizer: Adam, batch size: 32, learning rate: 0.001, epoch: 5, weight initialize: HE, Filter Size: 5*5

Loss가 약 0.8부터 0.15까지 안정적으로 감소하며 Accuracy가 0.9189으로 도출되었다.

C. 분석

i. Filter size

Convolution layer의 filter 크기를 3*3으로 정하고 여러 Optimizer, batch size, learning rate, initialize를 테스트 해보았지만 모두 Accruacy가 약 0.1으로 도출되며 학습이 되지 않은 경향을 보였다. Filter 크기를 5*5로 적용하고 낮은 learning rate를 테스트하니 높은 Accuracy가 도출되고 안정적인 Loss 감소를 보였다. 3*3 크기의 filter는 세부적인 특성들을 잘 파악할 수 있고 5*5 크기의 filter는 큰 구조를 학습하는데 도움이 된다. MNIST 데이터셋과 같은 간단한 숫자 이미지에서는 세부적인 특징보다 숫자의 구조적인 큰 특징을 파악하는 것이 더 중요하므로 큰 사이즈의 filter를 적용시키는 것이 더 유리하다. 또한 더 큰 크기의 filter는 더 많은 파라미터로 학습할 수 있어 표현력과 복잡한 패턴에 유리해진다. 컴퓨팅 자원의 한계로 인해 층이 얕은 Network를 구성했는데 이때 5*5 필터 같이 큰 사이즈의 필터가 더 유리하다.

ii. Learning Rate

Learning Rate가 0.1이나 0.01 같이 큰 경우는 잘 학습되지 않았고 0.001의 경우에는 안정적이고 잘 학습되었다. 큰 Learning Rate는 Loss가 감소하지 않고 유지되었는데 기울기 방향으로 너무 크게 update 되고 최적점 근처에서 발산하거나 진동한다고 생각할 수 있다. 또한 CNN에서는 비선형적이고 복잡한 구조를 가지고 있어서 큰 learning rate는 Overshooting이 일어날 수 있다. 작은 learning rate를 적용하여 최적점 중심으로 안정적으로 수렴할 수 있게 하고 단순한 데이터셋에서 더 효과적으로 학습을 진행할 수 있게 한다.

iii. Optimizer

batch size: 32, learning rate: 0.001, epoch: 5, weight initialize: HE, Filter Size: 5*5 조건에서 Optimizer를 SGD와 Adam을 적용하여 비교해보았다. 모두 높은 Accuracy를 보였지만 Adam Optimizer를 사용한 경우가 더 좋은 성능을 보였다. 단순한 SGD보다 Adaptive Learning Rate와 Momentum을 결합한 Adam이 학습 초기에 더 빠르게 Loss를 감소시킨다. 또한 learning rate를 조절하여 큰 Gradient에서 빠르게 학습하거나 작은 Gradient에서 속도를 줄여 overfitting이나 발산하는 것 등을 방지할 수 있다. CNN에서는 filter와 층별 weight가 서로 연관되어 있긴 때문에 불안정한 update가 전체 학습에 영향을 미칠 수 있는데 Adam은 이런 불안정함을 감소시킬 수 있다. 또한 CNN에서 층이 깊어질수록 Gradient Vanishing 문제가 발생할 수 있는데 Adam은 기울기 크기에 맞춰 조절하므로 작은 기울기에서도 효과적인 학습이 가능하다. 따라서 복잡한 CNN 구조에 Adam Optimizer가 더 효과적이다.

3. Recurrent Neural Network

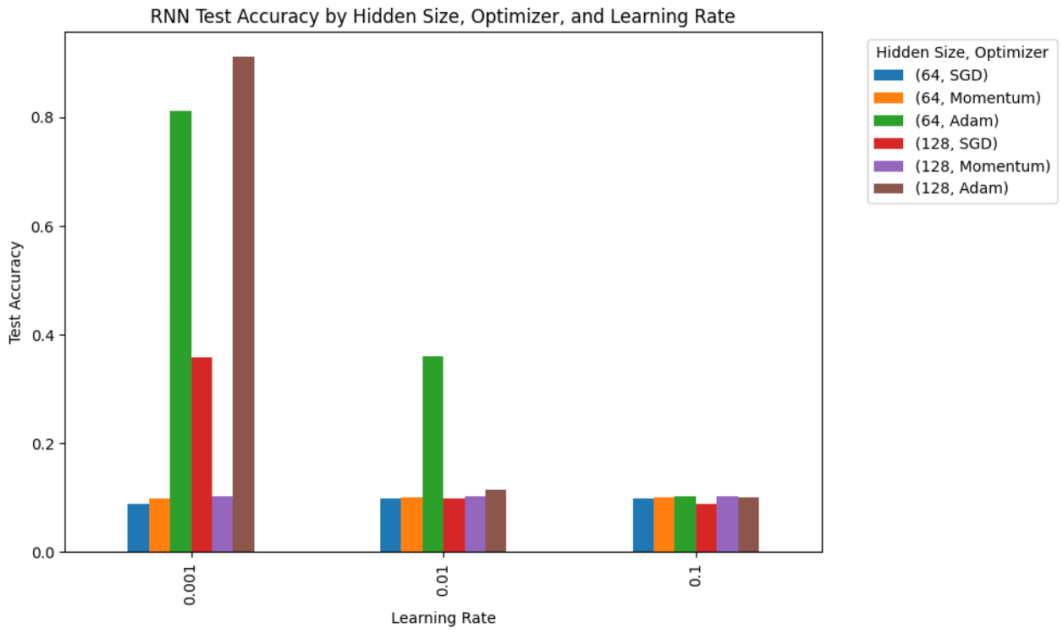
A. 문제 분석 및 설계

RNN에 시간 단위로 28개의 입력값을 전달하고 10개의 class로 분류하도록 만든다. 은닉 상태는 $h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$를 적용하고 출력 상태는 $y_t = softmax(W_{wy}h_t + b_y)$를 적용한다. Loss function은 Cross-Entropy를 적용하며 복잡한 LSTM이나 GRU 같은 고급 구조는 사용하지 않았다. Optimizer로 [SGD, Momentum, Adam]을 사용하고 batch size는 [16, 32], learning rate는 [0.1, 0.01, 0.001]로 적용해보며 Accuracy와 어떤 관계가 있는지 분석한다. MNIST 데이터셋이 작아 빠르게 수렴하므로 epochs는 5로 설정했으며 컴퓨팅 자원이 좋지 않아 더 많은 epoch는 사용해보지 못했다.

B. 실험 결과

i. Hidden sizes: [64, 128], learning rate: [0.1, 0.01, 0.001], optimizer: [SGD, Momentum, Adam]으로 Grid Search한 결과

	hidden_size	learning_rate	optimizer	test_accuracy	final_loss
0	64	0.100	SGD	0.0982	15.7642
1	64	0.100	Momentum	0.1009	16.2050
2	64	0.100	Adam	0.1028	2.8630
3	64	0.010	SGD	0.0974	4.1622
4	64	0.010	Momentum	0.1009	13.7404
5	64	0.010	Adam	0.3602	1.8060
6	64	0.001	SGD	0.0892	2.3178
7	64	0.001	Momentum	0.0974	2.4670
8	64	0.001	Adam	0.8123	0.6148
9	128	0.100	SGD	0.0892	16.2079
10	128	0.100	Momentum	0.1028	16.4497
11	128	0.100	Adam	0.1010	3.5843
12	128	0.010	SGD	0.0982	10.0848
13	128	0.010	Momentum	0.1032	15.4953
14	128	0.010	Adam	0.1135	2.4401
15	128	0.001	SGD	0.3585	2.1590
16	128	0.001	Momentum	0.1026	2.6558
17	128	0.001	Adam	0.9115	0.4374



C. 분석

i. Learning Rate

위 표를 보면 Learning Rate가 0.1 경우는 Accuracy가 약 0.1 정도로 매우 작았다. 이는 학습이 잘 이루어 지지 않음을 알 수 있다. 0.01인 경우도 0.1로 설정했을 때 보다는 약간의 개선이 보이지만 마찬가지로 Accuracy가 매우 낮고 학습이 잘 이루어지지 않았다. Learning Rate가 0.001인 경우는 Adam Optimizer를 사용하면 Loss를 안정적으로 감소시키며 매우 높은 Accuracy를 도출했다. 작은 Learning Rate를 가지면 Loss를 통해 더 안정적으로 최적점에 도달할 수 있다. 또한 Adam을 이용한다면 세밀하게 Learning Rate를 조정할 수 있으므로 좋은 성능을 가진다.

ii. Optimizer

SGD를 사용한 경우, Grid Search한 모든 결과에서 낮은 Accruacy를 보였다. RNN 구조에서 나타날 수 있는 Gradient Vanishing/Exploding 문제가 단순한 SGD와 결합되며 이를 완화하지 못하기 때문이다. Local Minima에 벗어나기도 어려워 안좋은 성능을 보인다. Momentum을 사용한 경우, SGD에 비해 weight update 속도를 개선하지만 마찬가지로 RNN 구조의 Gradient Vanishing/Exploding 문제를 해결하지 못한다. Momentum Optimizer는 이전 기울기 값에 일정 비율을 곱해 현재 기울기를 더해주는 방식으로 이루어지는데 Gradient Vanishing 같은 문제가 발생하면 기울기 자체가 0에 수렴하는 매우 작은 값이 되므로 Momentum의 효과가 미미해진다. 따라서 Momentum을 이용해도 SGD와 거의 비슷하게 안좋은 성능을 보인다. Adam을 사용한 경우, 초반에는 빠르게 학습하고 후반에 작은 update로 안정적인 수렴을 보여준다. Loss가 빠르게 감소하며 가장 좋은 성능을 보였다. 또한 Adam은 RNN의 Gradient Vanishing 상황에서 기울기의 크기가 매우 작아지더라도 그만큼 적절히 learning rate를 증가시켜 일정 수준으로 유지할 수 있다. 그리고 초기 단계에서 빠르게 수렴하므로 초기에 소실되는 성능 저하를 막을 수 있다는 점도 존재한다. 이러한 특성으로 RNN 구조적 문제를 일부 해결할 수 있어 좋은 성능이 나타난다.

iii. Hidden Size

Hidden Size가 64인 경우보다 128인 경우에서 더 좋은 성능이 도출되었다. Hidden Size가 커질수록 더 많은 정보와 복잡한 패턴 등을 학습할 수 있어 성능이 향상된다. 또한 RNN은 시간 단계에 따른 정보를 처리하므로 hidden size가 커질수록 더 복잡한 순차적 패턴을 학습할 수 있다.