[양식 4] 과제 요약보고서

[캡스톤디자인 과제 요약보고서]

과 제 명	LeNet-5의 구현 및 성능 향상: 활성 함수의 변경, 배치 정규화, 가중치 감쇠 기법을 중심으로

1. 과제 개요

가. 과제 선정 배경 및 필요성

Lenet-5는 1990년대에 Yann LeCun이 제안한 아키텍처로 합성신경망(CNN)의 초창기 모델이다. 이 모델은 발표된 지 오래 되었기 때문에 최근의 모델에 비해 망의 깊이가 얕고 단순한 구조를 가지고 있다. 하지만 합성신경망의 핵심인 컨볼루션 및 풀링 연산을 사용하고 분류를 위해 다층신경망(MLP)를 사용한다는 점에서 신경망의 기본적인 구조를 공부 할 수 있는 좋은 모델이다. 따라서 LeNet-5를 직접 구현함으로써 합성신경망에서 사용하는 여러 알고리즘을 깊이 있게 학습하고자 한다.

LeNet-5는 그래디언트 기반 학습 방법을 사용하기 때문에 그래디언트 소실/폭발 문제(vanishing and exploding gradient problem)를 갖는다. 현대의 신경망 연구에서는 이러한 문제를 해결하기 위해 여러 방법론을 제시했는데, (1) 활성 함수의 변경, (2) 배치 정규화가 이러한 방법론에 속한다. 본 연구에서는 이 두 가지 방법을 LeNet-5에 적용시키고, 그것이 가져오는 성능의 차이를 측정하고자 한다.

LeNet-5는 다른 신경망과 마찬가지로 과적합(overfitting)이 발생할 수 있다. 과적합은 모델의 일반화 능력을 떨어뜨리기 때문에 해결해야 할 중요한 문제이다. 이를 방지하기 위한 기법에는 가중치 감쇠(weight decay)가 있다. 본 연구에서는 이 방법을 LeNet-5에 적용시키고, 그것이 가져오는 성능의 차이를 측정하고자 한다.

나. 과제 주요내용

LeNet-5는 출력층을 제외한 모든 층에서 활성화 함수로 하이퍼볼릭탄젠트(tanh)를 이용한다. 하이퍼볼릭탄젠트의 도함수는 입력값이 0에서 멀어질수록 함숫값이 0으로 수렴한다는 특징이 있다. 이에 따라 그래디언트의 크기가 작아져서 신경망의 학습이 더뎌지는 문제가 발생한다. 현대의 신경망 모델은 이러한 문제를 해결하고자 렐루(ReLu)나 소프트맥스(softmax)를 활성화 함수로 사용한다. LeNet-5에 이 두 함수를 적용한 후 성능을 측정하여 기본 모델(baseline model)과 비교한다.

LeNet-5가 사용하는 그래디언트 기반 학습 방법은 학습률(learning rate)의 초깃값에 매우 민감하다. 예를 들어 학습률이 지나치게 크면 그래디언트가 폭발하여 학습이 제대로 되지 않는 문제가 발생한다. 또한 신경망의 구조적 특성상 각 층에 입력되는 훈련 집합의 분포가 달라지는 공변량 시프트(covariate shift) 현상이 발생한다. 이 역시 신경망의 학습을 저해하는 요인이다. 배치 정규화(batch normalization)는 공변량 시프트 문제를 해결하고 신경망이 학습하는 데 있어서 학습률의 영향을 덜 받도록 하는 기법이다. LeNet-5에 배치 정규화 기법을 적용한 후 성능을 측정하여 기본 모델과 비교한다.

신경망을 학습 할 때 모델의 복잡도에 비해 데이터가 부족하면, 작은 값이었던 가중치들이 점차 증가하면서 과적합이 발생 할 수 있다. 가중치 감쇠는 목적 함수에 규제 항을 두어 가중치의 영향력을 줄이는 식으로 과적합을 억제한다. 규제 항의 종류는 여러 가지이나 본 연구에서는 가장 널리 쓰이는 L2놈을 이용한다. LeNet-5의 목적함수에 규제 항(L2놈)을 추가한 후 성능을 측정하여 기본 모델과 비교한다.

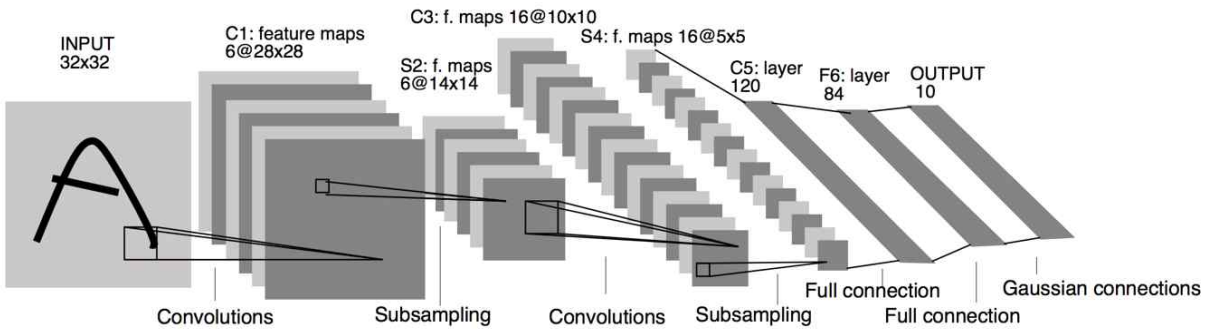
다. 최종결과물의 목표

1) C++로 LeNet-5를 구현하며 stl(standard template library)을 제외한 다른 라이브러리는 사용하지 않는다. 작성한 모든 파일(.h, .cpp 등)은 깃허브에 공개한다.
2) 구현한 LeNet-5의 분류 정확도(accuracy)가 99% 이상이 되도록 한다. 정확도 측정 시 데이터셋은 MNIST를 사용한다.
3) 활성 함수의 변경, 배치 정규화, 그리고 가중치 감쇠는 일반적으로 신경망의 성능을 향상시킨다고 알려져 있다. 따라서 이러한 기법을 적용한 LeNet-5의 정확도는 적용하지 않은 것의 정확도 이상이 되도록 한다.
4) LeNet-5와 성능 향상 기법을 적용한 LeNet-5의 성능 비교를 정리하여 보고서로 작성한다.

2. 과제 수행방법

1) 논문 리뷰

모델을 구현하기에 앞서 LeCun et al. (1998)의 논문을 읽고 LeNet-5의 구조를 파악했다. 주요한 내용을 설명하자면 아래와 같다.



LeNet-5는 2개의 컨볼루션층과 풀링층, 그리고 3개의 완전연결층으로 이루어져 있다. 각 층에서 생성되는 출력 이미지의 크기는 아래의 표와 같다.

Input	Convolution	Pooling	Convolution	Pooling	Dense	Dense	Dense
$28\times28\times6$	$28\times28\times6$	$14\times14\times6$	$10\times10\times6$	$5\times5\times6$	120	84	10

위에 제시한 모델의 구조에서 C3층은 일반적인 컨볼루션층과 달리 이미지 여러 장을 특정하여 컨볼루션 연산을 한 후 하나의 출력 이미지를 생성한다. 아래의 행렬을 예로 들어 설명하자면 다음과 같다. 이 행렬의 행 인덱스(row index)는 이전 층(S2)의 출력 이미지를 의미하고, 열 인덱스(col index)는 현재 층의 출력 이미지를 의미한다. 또한 행렬 내의 x 표시는 행 인덱스에 해당하는 이전 층의 출력 이미지와 열 인덱스에 해당하는 현재 층의 필터를 컨볼루션한다는 것이다. 조금 더 구체적으로 첫 번째 열은 행이 1, 2, 3일 때 각각 x를 갖는다. 이는 현재 층의 첫 번째 출력 이미지가 이전 층의 첫 번째, 두 번째, 세 번째 출력 이미지를 모두 컨볼루션하여 생성된다는 의미이다.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	x				x	x	x			x	x	x	x	x	x	
2	x	x				x	x	x			x	x	x	x	x	x
3	x	x	x				x	x	x			x		x		x
4		x	x	x			x	x	x	x			x		x	
5			x	x	x			x	x	x	x		x		x	x
6				x	x	x			x	x	x	x		x		x

2) 모델 구현

C++를 이용하여 모델을 구현하였다. 또한 standard template library 외의 다른 라이브러리는 사용하지 않았다. 신경망의 구조는 LeCun et al. (1998)이 제시한 것을 그대로 적용하되, 손실 함수는 평균오차제곱과 크로스엔트로피를 사용하였다. LeNet-5를 완성한 후에는 활성화 함수(relu, softmax), 가중치 감쇠 항(L2 norm), 그리고 배치 정규화를 구현하였다.

3) 성능 측정

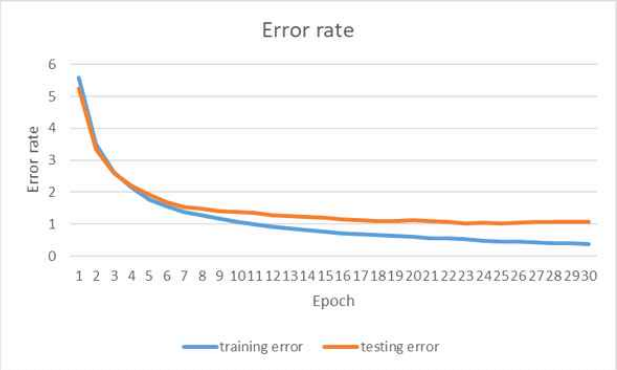
성능 측정은 총 8번 시행되었고, 하이퍼볼릭탄젠트와 평균오차제곱을 적용한 모델(tanh + mse)을 기준(baseline)으로 설정하여 다른 모델과 비교하였다. 성능을 측정한 여덟 가지 모델은 아래와 같다.

- Tanh + MSE
- Tanh + MSE + L2 regularization(lambda: 0.0005)
- Tanh + MSE + L2 regularization(lambda: 0.001)
- Tanh + MSE + Batch normalization
- ReLu + Softmax + Cross entropy
- ReLu + Softmax + Cross entropy + L2 regularization(lambda: 0.001)
- ReLu + Softmax + Cross entropy + L2 regularization(lambda: 0.01)
- ReLu + Softmax + Cross entropy + Batch normalization

3. 수행결과

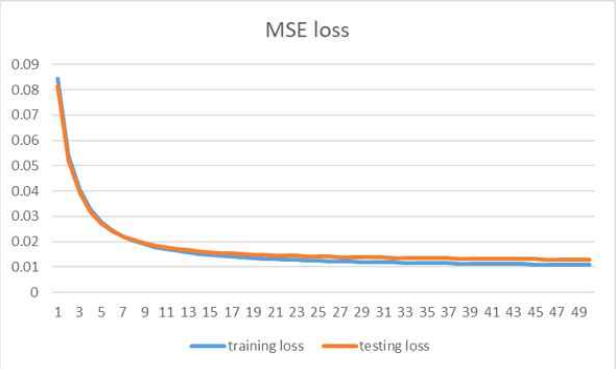
가. 과제수행 결과

1) Tanh + MSE



Epoch	Batch	Learning rate	Lambda	Training error	Testing error
30	20	0.5	0	0.37%	1.03%

2) Tanh + MSE + L2 regularization

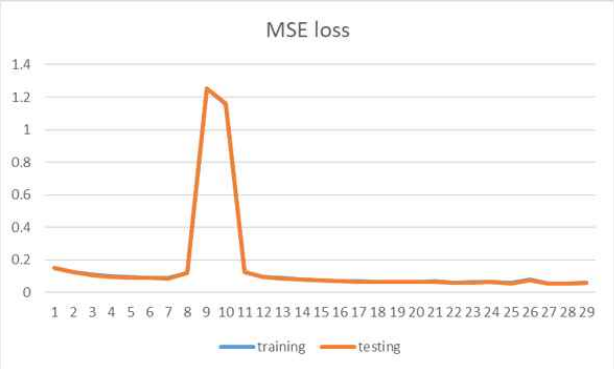


Epoch	Batch	Learning rate	Lambda	Training error	Testing error
30	20	0.5	0.0005	0.72%	1.1%

3) Tanh + MSE + L2 regularization

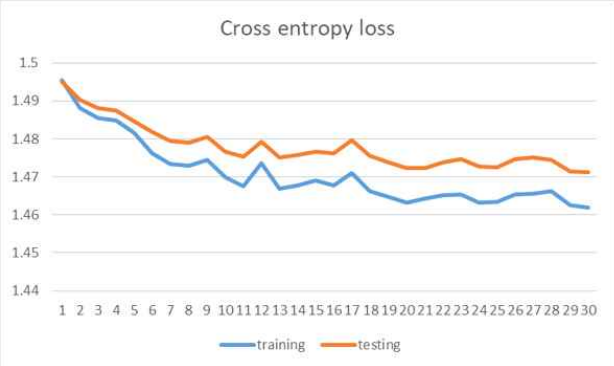
Epoch	Batch	Learning rate	Lambda	Training error	Testing error
30	20	0.5	0.001	1.09%	1.31%

4) Tanh + MSE + Batch normalization



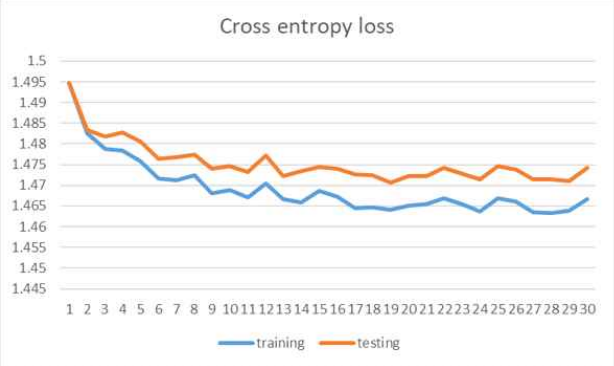
Epoch	Batch	Learning rate	Lambda	Training error	Testing error
30	20	0.1	0	5.35%	5.04%

5) ReLu + Softmax + Cross entropy



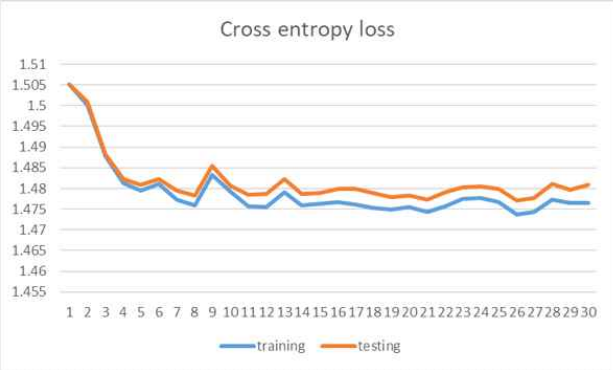
Epoch	Batch	Learning rate	Lambda	Training error	Testing error
30	20	0.1	0	0.04%	1%

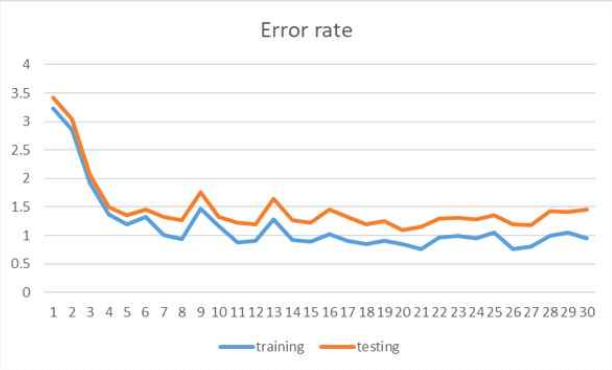
6) ReLu + Softmax + Cross entropy + L2 regularization



Epoch	Batch	Learning rate	Lambda	Training error	Testing error
30	20	0.1	0.001	0.44%	1.16%

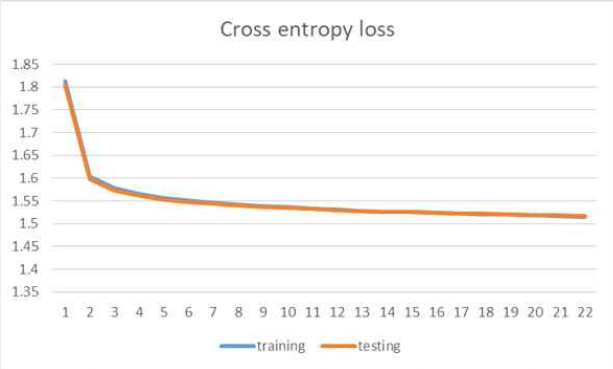
7) ReLu + Softmax + Cross entropy + L2 regularization





Epoch	Batch	Learning rate	Lambda	Training error	Testing error
30	20	0.1	0.01	0.94%	1.46%

8) ReLu + Softmax + Cross entropy + Batch normalization



Epoch	Batch	Learning rate	Lambda	Training error	Testing error
30	20	0.01	0	3.9%	4.1%

나. 최종결과물 주요특징 및 설명

라이브러리가 지원하는 layer, initialization, activation, loss는 아래와 같다.

1) Layer

- Convolutional
- Pooling
- Dense(fully connected)
- Batch normalization

2) Initialization

- Normal distribution
- Uniform distribution

3) Activation

- Hyperbolic tangent
- ReLu
- Softmax

4) Loss

- MSE
- Cross entropy

라이브러리를 이용하여 LeNet-5를 생성하는 방법은 아래와 같다.

```
int main()
{
    int n_img_train = 60000, n_img_test = 10000, img_size = 784;
    vector<Matrix> train_X, test_X;
    Vector train_Y, test_Y;

    ReadMNIST("train-images.idx3-ubyte", n_img_train, img_size, train_X);
    ReadMNISTLabel("train-labels.idx1-ubyte", n_img_train, train_Y);
    ReadMNIST("test-images.idx3-ubyte", n_img_test, img_size, test_X);
    ReadMNISTLabel("test-labels.idx1-ubyte", n_img_test, test_Y);

    vector<vector<int>> indices(6, vector<int>(16));
    indices[0] = { 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1 };
    indices[1] = { 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1 };
    indices[2] = { 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1 };
    indices[3] = { 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1 };
    indices[4] = { 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1 };
    indices[5] = { 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1 };

    SimpleNN model;
    model.add(new Conv2D({ 28, 28, 1 }, { 5, 5, 6 }, 2, Init::UNIFORM));
    model.add(new BatchNorm({ 28, 28, 6 }));
    model.add(new Activation(6, Activate::RELU));
    model.add(new Pool2D({ 28, 28, 6 }, { 2, 2, 6 }, 2, Pool::AVG));
    model.add(new Conv2D({ 14, 14, 6 }, { 5, 5, 16 }, 0, Init::UNIFORM, indices));
    model.add(new BatchNorm({ 10, 10, 16 }));
    model.add(new Activation(16, Activate::RELU));
    model.add(new Pool2D({ 10, 10, 16 }, { 2, 2, 16 }, 2, Pool::AVG));
    model.add(new Dense(400, 120, Init::UNIFORM));
    model.add(new BatchNorm({ 120, 120, 1 }));
    model.add(new Activation(1, Activate::RELU));
    model.add(new Dense(120, 84, Init::UNIFORM));
    model.add(new BatchNorm({ 84, 84, 1 }));
    model.add(new Activation(1, Activate::RELU));
    model.add(new Dense(84, 10, Init::UNIFORM));
    model.add(new BatchNorm({ 10, 10, 1 }));
    model.add(new Activation(1, Activate::SOFTMAX));
    model.add(new Output(10, Loss::CROSS_ENTROPY));

    int n_epoch = 30, batch = 20;
    double l_rate = 0.1, lambda = 0.0;

    model.fit(train_X, train_Y, l_rate, n_epoch, batch, lambda, test_X, test_Y);

    return 0;
}
```

4. 기대효과 및 활용방안

1) 성능 향상 기법을 LeNet-5에 적용했을 때 어떠한 성능 차이가 있는지 확인할 수 있다.
(여기서 성능 향상 기법은 (1) 활성 함수의 변경, (2) 배치 정규화, (3) 가중치 감쇠를 말한다)
2) 합성신경망을 깊이 있게 학습하고자 할 때 참고자료로 활용할 수 있다.

5. 결론 및 제언

1) 구현한 모델의 에러율(error rate)이 약 1%로 논문에서 제시한 0.95%에 근접하였다.
2) 활성화 함수를 ReLu, Softmax로 손실 함수를 Cross entropy로 변경했을 때 학습이 빨라지는 것을 확인하였다.
3) 가중치 감쇠, 배치 정규화를 적용했을 때 과적합이 억제되는 것을 확인하였다.
4) 한편 배치 정규화를 적용했을 때 모델 성능이 더 향상될 것이라 예상했으나, 측정 결과 오히려 성능이 떨어진 것을 확인하였다.

※ 본 양식은 요약보고서이며, 최종결과물을 필히 추가 제출하여야 함.

팀 학생대표 성명 : 남승태 (인)