CNN을 이용한 이미지 분류

CIFAR-10 이미지 분류 딥러닝 모델 설계

요약

기계학습의 과정과 관련 개념 및 이론을 알아보고 다양한 딥러닝 알고리즘 중 CNN의 특징과 학습 과정을 살펴본다. 3차원 데이터인 이미지를 분류하는 모델을 코드로 구현하는 방법을 알아보고 이를 Colab 환경에서 라이브러리와 함수를 이용하여 Python으로 구현한다. 이미지 분류를 CNN 알고리즘을 이용하여 코드로 구현하는 방법을 배운다.

작성자

54기 박은서 dkan601@naver.com

54기 정희양 huiyang1022@naver.com

54기 최소윤 anna7a@naver.com

1. Introduction

1.1 이미지 분류 작업

1. Classification: 사물의 존재 여부 분류

2. Object Detection: 사물의 위치를 찾아냄과 동시에 여러 가지 사물을 인식한다. 다량의 물체에 바운딩 박스를 쳐가며 물체를 분간한다.

3. Instance Segmentation: 사물의 형태를 찾아내는 객체 분류(Instance Segmentation)는 픽셀마다 물체를 탐지하여 이를 마스킹해준다.

1.2 이미지 데이터 전처리

- 딥러닝 모델이 잘 학습하기 위해서는 준비된 데이터들을 모델에 맞게 틀을 맞추는 과정이 필요하다. 이를 '데이터 전처리 과정'이라고 부른다.

- 이미지 데이터의 경우 Scaling, Normalizing, Augmentation 등의 과정을 거친다.

- Normalization: 임의의 데이터를 0~1의 값으로 맞추는 작업

-Standardization: 정규분포 N(0,1)로 맞추는 작업

2. Deep Learning Process

2.1 퍼셉트론

인공지능을 구현하기 위해서 인간의 뇌가 패턴을 인식하는 방식을 모사한 알고리즘인 인공신경망(ANN, Artificial neural network)을 사용한다. 인공신경망에서 뉴런의 역할을 하는 것이 퍼셉트론(Perceptron)이다.

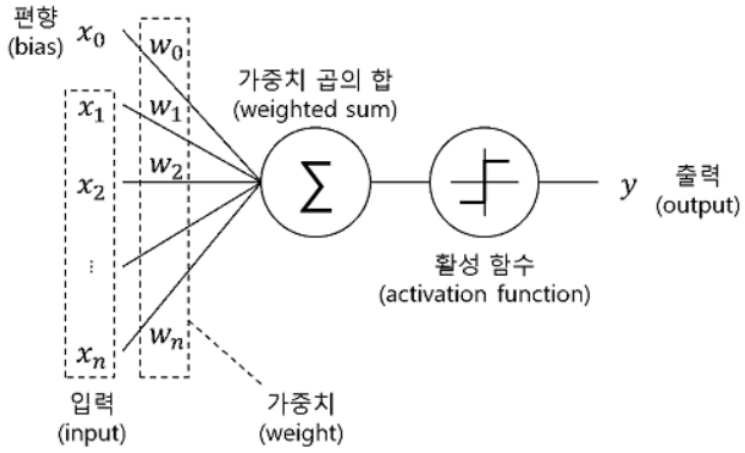


그림 1 퍼셉트론 구조 (quantylab's blog, Aug 21, 2020)

퍼셉트론은 다수의 신호를 입력 받아 하나의 신호를 출력한다. 이는 y=wx+b 와 같은 수식으로 나타낼 수 있으며 입력값에 가중치를 곱하여 결과값을 출력한다. 이는 일차 함수로 표현된다. 여기서 x는 입력값이고 w는 가중치, b는 편향, y는 출력값이다. 입력된 신호는 세기에 따라 가중치 w를 부여한다. 보내온 신호의 총합이 정해진 한계치인 임계값을 넘어설 때 1을 출력하고 그렇지 않을 때에는 0을 출력한다. 이러한 함수를 계단 함수라고 한다.

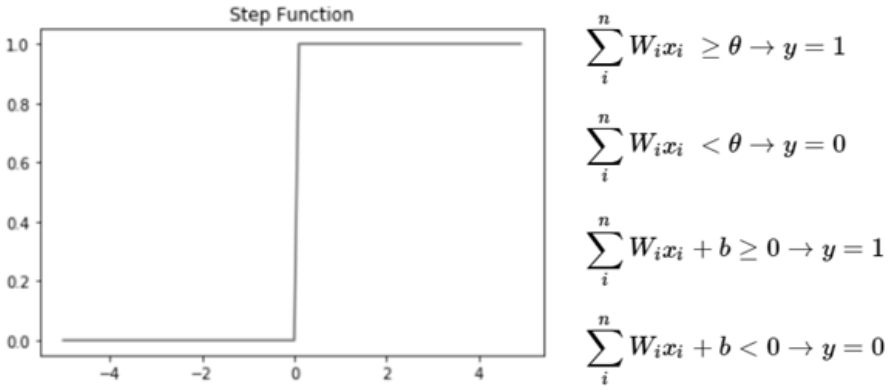


그림 2 Step 함수 그래프와 표현 식 (유원준's wikidocs, Aug 21, 2020)

위의 식과 같이 계단 함수에 사용된 임계치 값을 보통 θ(세타)로 표현하며 임계값을 좌변으로 이항하여 -θ를 b라고 치환하여 사용할 수 있다. 여기서 b는 bias의 약자로 편향이라고 부른다. 예를 들어 기계학습을 통해 고혈압에 걸린 환자를 예측한다고 한다면 x는 고혈압 발생에 영향을 주는 요소로 고혈압의 원인에 해당한다. w는 각각의 요소가 고혈압 발생에 얼마만큼의 영향을 미치는 지를 판단하기

위한 것으로 입력값에 대해 일정양의 중요도를 부여한다. b를 이용하여 정확도를 높이기 위해 미세한 조정을 할 수 있다.

$$y = w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n + b$$

그림 3 퍼셉트론 수식 (라인하트's brunch, Aug 21, 2020)

고혈압을 예측하는 기계를 만들 때 어려운 것은 고혈압의 원인 요소들에 대한 가중치를 산정하는 것이다. 각 요소의 가중치가 정확하지 않으면 정확한 예측이 어렵다. 그래서 y=wx+b라는 식과 빅데이터를 이용하여 각 원인 요소별의 정확한 가중치의 값이 나올 때까지 반복 학습을 시켜 여러 번의 갱신을 통해 정확한 가중치와 편향성을 찾는 것이 기계학습의 목표라고 볼 수 있다.

퍼셉트론은 단층 퍼셉트론과(Single-Layer Perceptron) 다층 퍼셉트론(Multi-Layer Perceptron, MLP)으로 나누어진다. 단층 퍼셉트론은 값을 보내는 층인 입력층과 값을 받아서 출력하는 층인 출력층으로 이루어져 있으며 총 2개의 층이 존재한다. 단층 퍼셉트론을 이용하여 AND, NAND, OR 게이트를 구현할 수 있다. AND 게이트는 입력이 두 개이고 출력이 하나일 때 입력값이 모두 1일 때만 출력값이 1이고 나머지 경우는 출력값이 0이다. NAND 게이트는 두 개의 입력값이 모두 1일 때 출력값이 0이고 나머지 경우는 출력값이 1이다. OR 게이트는 두 개의 입력값이 모두 0일 때 출력값이 0이고 나머지 경우는 출력값이 1이다.

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

그림 4 (왼쪽부터) AND 게이트, NAND 게이트, OR 게이트, XOR 게이트
(유원준's wikidocs, Aug 21, 2020)

단층 퍼셉트론은 AND, NAND, OR 논리회로를 계산할 수 있지만 XOR 게이트 연산을 할 수 없다는

한계점이 있다. XOR 게이트는 두 개의 입력값 중 하나의 입력 값이 1이면 출력값이 1이고 나머지 경우는 출력값이 0인 논리회로이다. 단층 퍼셉트론은 선형 분류이기 때문에 곡선으로 나눠야 하는 XOR 게이트 연산을 할 수 없고 비선형 분류를 이용해야 한다. 다층 퍼셉트론은 입력층과 출력층 사이에 은닉층(Hidden Layer)이 1개 이상인 퍼셉트론을 의미한다. XOR 게이트를 구현하기 위해서는 기존의 AND, NAND, OR게이트를 조합하여 층을 더 추가하여 만들어 낼 수 있다. 은닉층이 2개 이상인 신경망을 심층 신경망(Deep Neural Network, DNN)이라고 한다. 은닉층을 더 추가할수록 인공 신경망의 능력이 높아진다.

2.2 활성화 함수

활성화 함수(Activation function)는 입력 신호의 총합을 출력 신호로 변환하는 함수로 입력 받은 신호를 얼마나 출력할지 결정하고 네트워크에 층을 쌓아 비선형성을 표현할 수 있도록 한다. 활성화 함수는 선형 함수가 아닌 비선형 함수이다. 인공 신경망에서 활성화 함수는 반드시 비선형 함수여야 하며 선형 함수로 은닉층을 쌓을 수 없다. 활성화 함수의 종류로는 계단 함수(Step Function), 시그모이드 함수(Sigmoid Function), 하이퍼볼릭탄젠트 함수(Hyperbolic Tangent Function), 렐루 함수(ReLU Function), 리키 렐루 함수(Leaky ReLU Function), 소프트맥스 함수(Softmax Function) 등이 있다.

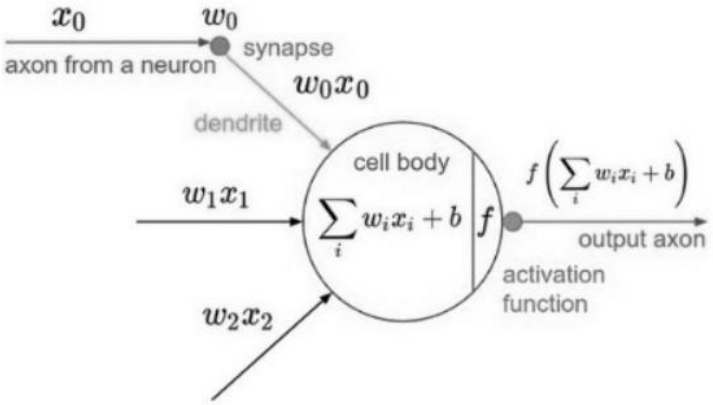
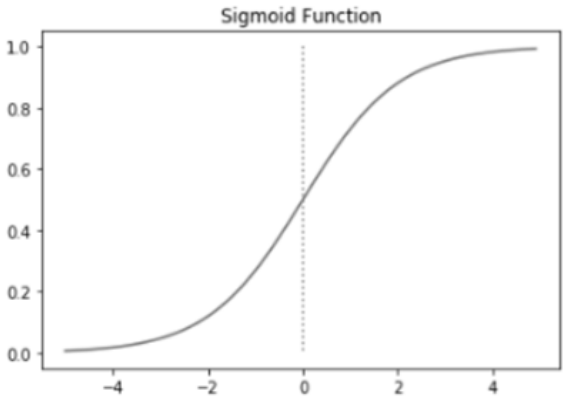


그림 5 활성화 함수 사용 단계 (CS231n's github, Aug 21, 2020)

활성화 함수 중 시그모이드 함수, 렐루 함수, 소프트맥스 함수에 대해 알아보려고 한다. 그전에 인공 신경망의 학습 과정에 대해서 짚고 가면, 인공 신경망은 입력값 x와 편향값을 받아온 후 입력층에

서 은닉층 방향으로 이동하면서 각 입력에 해당하는 가중치를 곱하고 그 값들을 합하고 활성화 함수를 지나 입력 신호의 총합을 출력 신호로 변환 시키면 이와 같은 순전파(Forward Propagation)를 통해 나온 예측값과 실제값의 오차를 손실 함수(Loss Function)를 통해 계산을 한 후 그 손실을 미분을 통해 기울기를 구하고 이를 통해 입력값과 출력값을 알고 있는 상태에서 신경망을 학습 시키는 방법인 역전파를 수행하며 오차를 줄여나가면서 인공 신경망을 학습시킨다.



$$f(x) = \frac{1}{1+e^{-x}}$$

그림 6 Sigmoid 함수 그래프와 표현 식 (유원준's wikidocs, Aug 21, 2020)

시그모이드 함수는 대표적인 Logistic 함수로 입력 값을 0~1사이의 값으로 변환하여 출력하며 데이터의 평균은 0.5가 된다. 시그모이드 함수의 기울기를 보면 입력값이 어느정도 작거나 크면 기울기가 아주 작아져 0과 가까워진다. 그로 인해 (Vanishing Gradient) 문제가 일어난다.

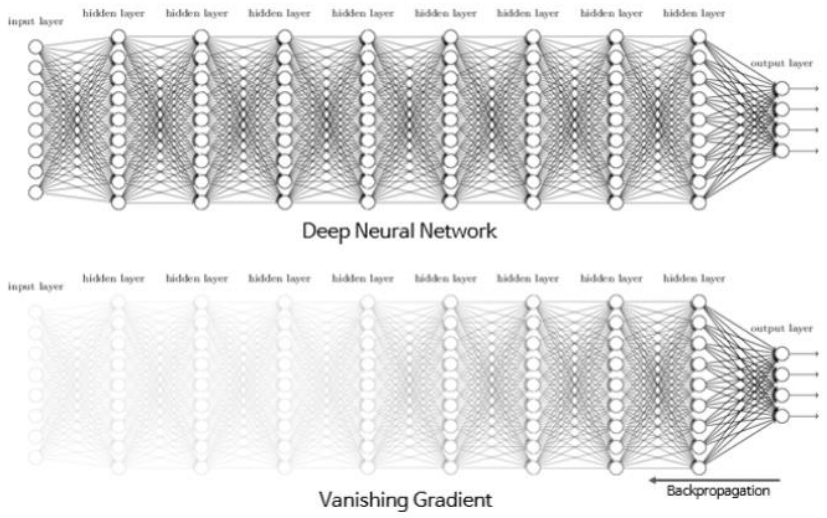
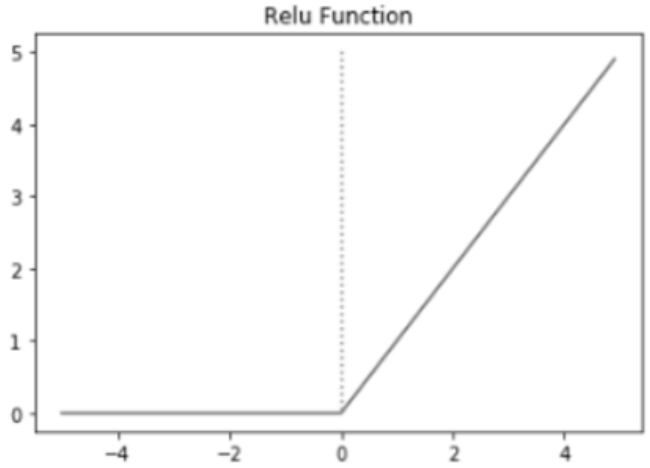


그림 7 Vanishing Gradient Problem (유원준's wikidocs, Aug 21, 2020)

이 현상은 기울기 소실 문제로, 시그모이드 함수를 사용하는 은닉층의 개수가 많아지면 0에 가까운

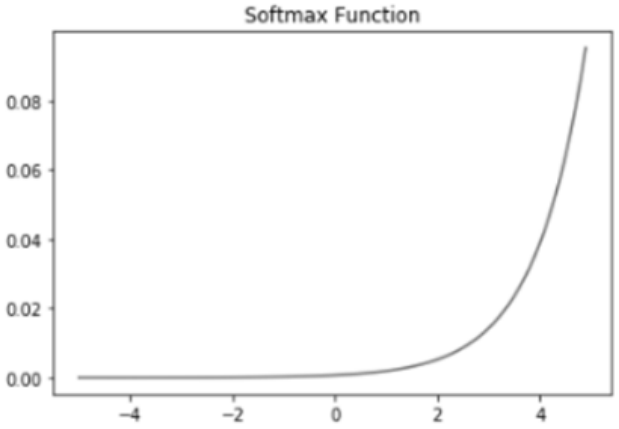
기울기가 계속 곱해지면서 역전파를 하면 할수록 기울기를 거의 전파 받을 수 없게 된다. 즉, 가중치가 업데이트가 일어나지 않아 학습이 잘 이루어지지 않는다. 대부분의 경우에 시그모이드 함수를 사용하지는 않지만 이진분류(Binary Classification)를 할 경우 출력층 노드가 1개이므로 이 노드에서 0~1 사이의 값을 가져야 마지막 과정을 거쳐 0 또는 1의 값을 출력값으로 받을 수 있기 때문에 이런 경우에는 시그모이드 함수를 사용한다.



$$f(x)=\begin{cases} x(x\geq 0) \\ 0(x<0) \end{cases}$$

그림 8 Relu 함수 그래프와 표현 식 (유원준's wikidocs, Aug 21, 2020)

렐루 함수는 성능이 좋아 대부분 은닉층에서 활성화 함수로 사용한다. 대부분의 입력값에 대해 기울기가 0이 아니기 때문에 빠르게 학습을 할 수 있다. 그래프를 보면 입력값이 0보다 작을 경우 0을 출력하고 양수를 입력하면 기울기가 1인 함수이기 때문에 입력값이 그대로 반환된다. 렐루 함수는 특정 양수 값에 수렴하지 않기 때문에 시그모이드 함수보다 잘 작동하며 속도도 더 빠르다. 하지만 입력값이 음수면 기울기가 0이기 때문에 뉴런을 다시 회생하는 것이 어려우며 이를 죽은 렐루 문제라고 부른다.



$$softmax(x) = \frac{x_i}{\sum_{j=0}^{k} e^{x_j}} \ (i = 0, 1, \dots k)$$

그림 9 Softmax 함수 그래프와 표현 식 (유원준's wikidocs, Aug 21, 2020)

소프트맥스 함수는 0~1사이의 숫자를 출력하는 함수로 출력하는 값은 확률이며 시그모이드 함수처럼 출력층의 뉴런에서 주로 사용된다. 이 함수는 세 가지 이상의 선택지 중 하나를 고르는 다중 클래스 분류 문제에 주로 사용된다.

2.3 손실 함수

손실 함수는 실제값과 예측값의 차이를 수치화하는 함수이다. 오차가 크면 손실 함수의 값이 커지고 오차가 작으면 손실 함수의 값이 작아진다. 회귀에서는 오차 제곱 평균(Mean Squared Error, MSE)을 사용하고 분류 문제에서는 교차 엔트로피 오차(Cross Entropy Error)를 주로 손실 함수로 사용한다. 손실 함수의 값을 최소화하여 더 정확한 가중치와 편향의 값을 찾는 것이다.

$$H(x) = wx + b$$

x	Y
1	1
2	2
3	3

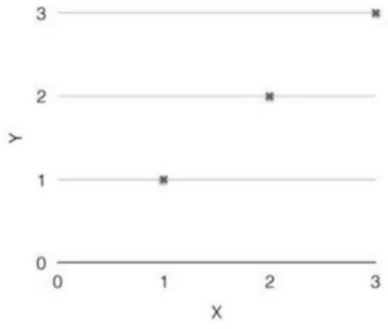


그림 10 오차 제곱 평균에 사용할 예시 데이터 (라인하트's brunch, Aug 21, 2020)

오차 제곱 평균의 예로 위의 데이터가 주어지고 랜덤하게 선택한 가중치 w를 0.3, b를 1.2, 일차 방정식은 y=0.3x + 1.2라고 가정해본다.

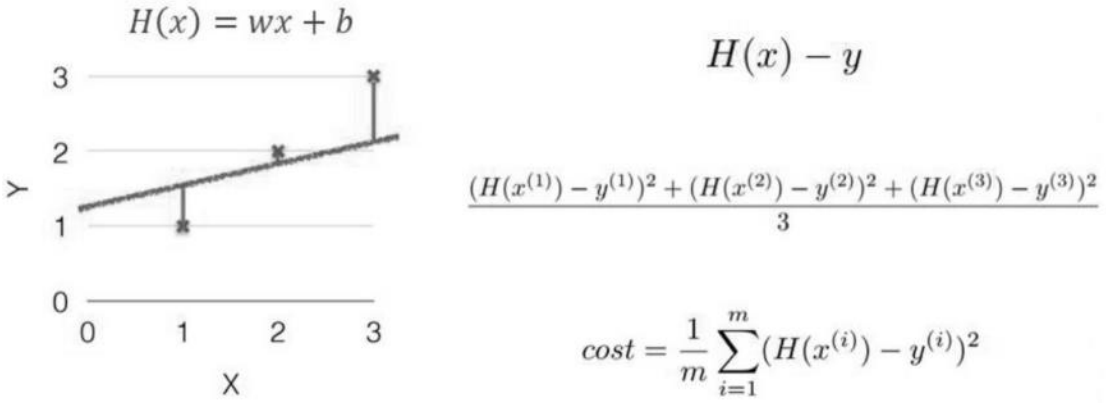


그림 11 오차 제곱 평균 계산 (라인하트's brunch, Aug 21, 2020)

실제값 y와 기계가 예측한 H(x)의 오차를 계산하기 위해 'H(x)-y'식을 사용한다. 이러한 오차를 손실이라고 하며 오차 표현뿐만 아니라 오차를 줄이도록 최적화를 해야 한다. 위와 같은 오차를 계산하기 위한 식에서 음수가 나오면 실제 오차를 정확히 측정할 수 없기 때문에 제곱을 한 후 그 값을 모두 더한 후 데이터의 개수로 나누어 오차를 계산한다.

교차 엔트로피 오차는 분류 문제에서 원-핫 인코딩(One-Hot Encoding)을 했을 경우에만 사용할 수 있는 오차 계산법이다. 여기서 원-핫 인코딩이란 단어 집합의 크기를 벡터 차원으로 하고 표현하고 싶은 단어의 인덱스에 1값을 부여하고, 다른 인덱스에는 0을 부여하는 단어의 벡터 표현 방식이다. 로그의 밑이 e인 자연로그를 예측값에 씌워 실제 값과 곱한 후 전체값을 합한 후 음수로 변환한다.

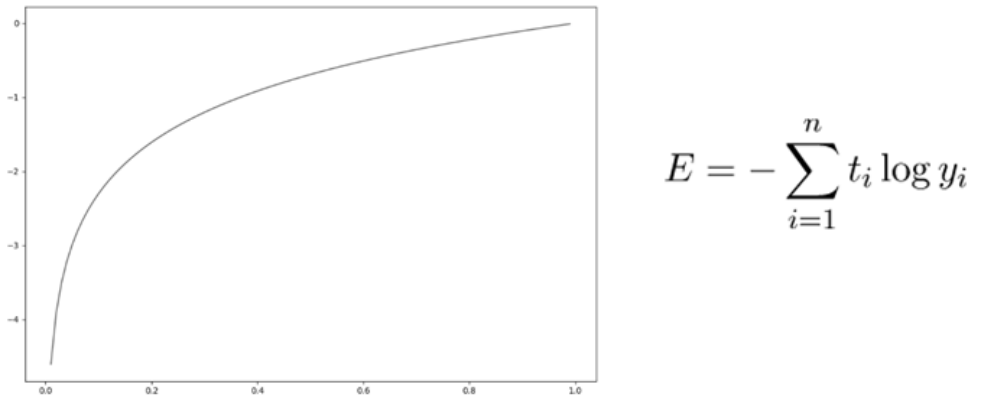


그림 12 교차 엔트로피 오차 그래프와 표현 식 (김형준's GIS DEVELOPER, Aug 21, 2020)

위의 식에서 실제값 t가 원-핫 인코딩된 벡터이고 t에 모델의 출력값이며 예측값인 y에 자연로그를 취한 것이 곱해지는 형태이다. 즉 교차 엔트로피 오차는 정답일 때의 모델 값에 자연로그를 계산하는 식이 된다. 위 그래프에서 가로축은 정답일 확률이고 세로축은 손실값에 -1을 곱한 값이다. 가로축 값이 1이면 정답일 확률이 100%이라는 것이고 그 때의 손실값은 0이 된다. 정답의 확률이 낮아질수록 손실값은 무한대로 커진다.

2.4 최적화 알고리즘-(경사 하강법)

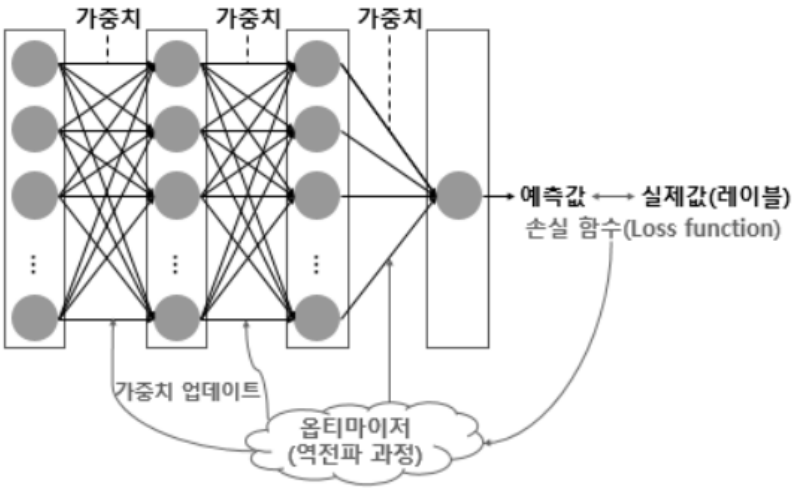


그림 13 최적화 알고리즘 사용 단계 (유원준's wikidocs, Aug 21, 2020)

손실 함수의 값을 줄여 오차를 작게 만들어가면서 학습하는 방법은 어떤 최적화 알고리즘(Optimizer)을 사용하느냐에 따라 다르다. 최적화 알고리즘은 배치 경사 하강법(Batch Gradient Descent), 확률적 경사 하강법(Stochastic Gradient Descent, SGD), 미니 배치 경사 하강법(Mini-Batch Gradient Descent), 모멘텀(Momentum), 아담(Adam) 등이 있다. 그 중 배치 경사 하강법, 확률적 경사 하강법, 아담에 대해 알아보려 한다.

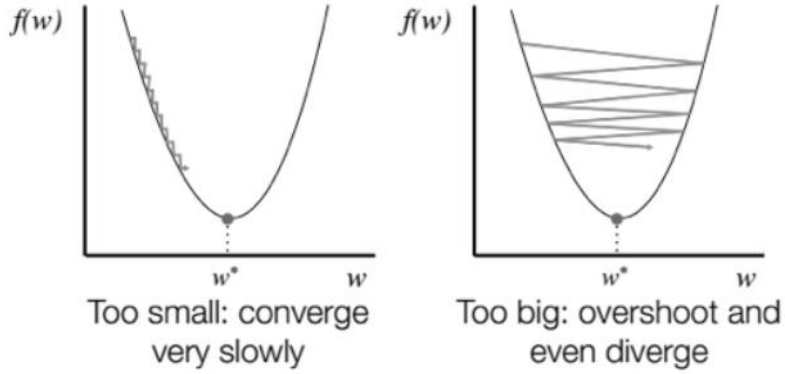


그림 14 학습률이 작은 경사 하강법 적용(왼쪽), 학습률이 큰 경사 하강법 적용(오른쪽) (JAEHYEONG's DS, Aug 21, 2020)

여기서 경사하강법이란 비용함수를 최소화하기 위해서 파라미터를 반복하여 조정해나가는 것을 말한다. 경사 하강법에서 파라미터로서의 학습률이 너무 작거나 클 경우에 학습시간 문제와 정확한 전역 최솟값을 찾기 어렵다.

배치 경사 하강법은 가장 기본적인 경사 하강법으로 오차를 구할 때 전체 데이터를 고려한다. 전체 데이터 셋에 대한 에러를 구한 뒤 기울기를 한번만 계산하여 모델의 파라미터를 업데이트 하는 방법이다. 전체 데이터 셋에 대한 손실을 계산하고 그래디언트(Gradient)의 반대방향으로 매개 변수를 개선하여 얼마나 큰 업데이트를 수행 할지를 학습 속도를 이용하여 진행한다.

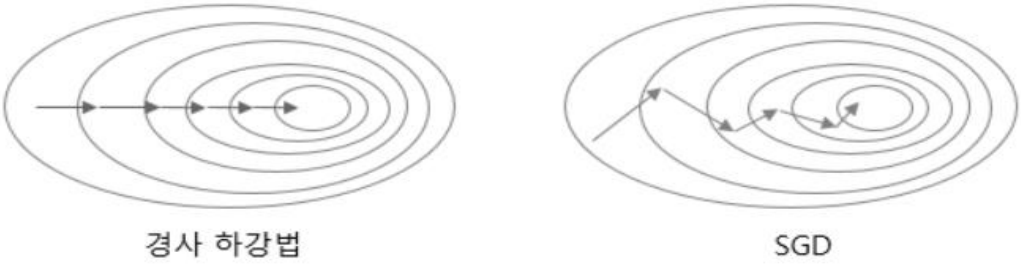


그림 15 경사 하강법 도식화(왼쪽), SGD 도식화(오른쪽) (유원준's wikidocs, Aug 21, 2020)

$$w \leftarrow w - \eta \frac{\partial E}{\partial w}$$
$$b \leftarrow b - \eta \frac{\partial E}{\partial b}$$

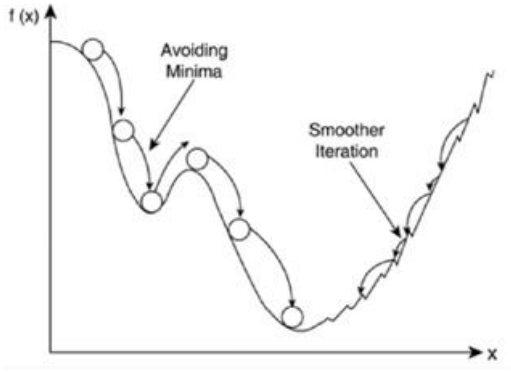
그림 16 기존 가중치 또는 편향에서 오차 기울기에 상수를 곱한 값을 빼서 갱신하는 수식

(Hyeonnii's tistory, Aug 21, 2020)

확률적 경사 하강법은 매개변수 값을 수정할 때마다 전체 데이터가 아니라 샘플을 무작위로 선택하여 하나의 데이터에 대해서만 계산하는 방법이다. 배치 경사 하강법보다 더 적은 데이터를 이용하므로 더 빠르게 계산할 수 있지만 매개변수의 변경폭이 불안정하고 정확도가 낮을 수 있다는 단점이 있다.

모멘텀, AdaGrad, Adam

SGD의 단점은 비등방성(방향에 따라 성질이 달라지는 함수)에서는 탐색 경로가 비효율적이라는 것이다. SGD의 단점을 개선하기 위해서 모멘텀, AdaGrad, Adam이라는 방법이 도입되었다.



$$v \leftarrow av - \eta \frac{\partial L}{\partial W}$$

$$W \leftarrow W + v$$

그림 17 Avoiding Local Minima (사이토 고키의 밑바닥부터 시작하는 딥러닝: 파이썬으로 익히는 딥러닝 이론과 구현, Aug 21, 2020)

모멘텀은 '운동량'을 뜻하는 단어로, 물리와 관계가 있다. 기존 SGD에서와는 달리 속도라는 변수가 나온다. 즉, 모멘텀은 기울기 방향으로 힘을 받아 물체가 가속된다는 물리 법칙을 나타낸다. 우측의 식이 모멘텀을 나타낸 식이다. 여기서 v가 바로 속도이다. av항은 물체가 아무런 힘을 받지 않을 때 서서히 하강시키는 역할을 한다.

신경망 학습에 있어서 학습률의 값은 매우 중요하다. 값이 너무 작으면 학습 시간이 길어지고, 값이 너무 크면 발산하기 때문이다. 이 학습률을 정하는 효과적 기술로 '학습률 감소'가 있다. 학습률 감소는 처음에는 큰 학습률로 학습하다 나중에는 점차 작게 학습하는 것이다. 이를 더욱 발전시킨 것이 AdaGrad이다.

$$h \leftarrow h + \frac{\partial L}{\partial W} \times \frac{\partial L}{\partial W}$$

$$W \leftarrow W - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$$

그림 18 AdaGrad 업데이트 수식 (사이토 고키의 밑바닥부터 시작하는 딥러닝: 파이썬으로 익히는 딥러닝 이론과 구현, Aug 21, 2020)

AdaGrad의 갱신 방법은 수식으로 다음과 같다. 여기서 새로운 변수 h가 등장한다. 이 가중치 제곱을 더한 h를 통해 가중치를 업데이트하는 것이다. (여기서 x는 행렬 각각의 원소곱이다) Adam은 위의 두 기법을 합쳐놓은 것으로 자세한 설명은 생략한다.

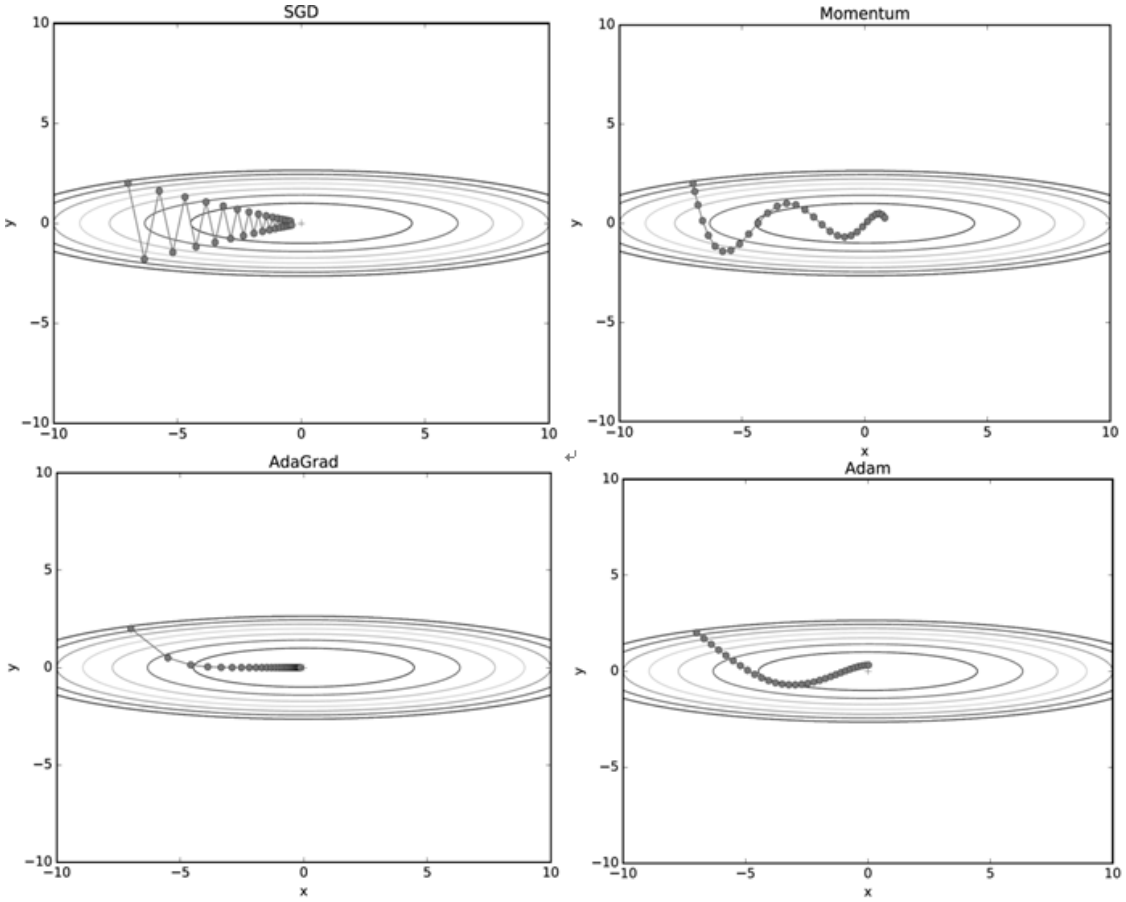


그림 19 (왼쪽 위에서부터) SGD, Momentum, AdaGrad, Adam 도식화 (사이토 고키의 밑바닥부터 시작하는 딥러닝: 파이썬으로 익히는 딥러닝 이론과 구현, Aug 21, 2020)

2.5 하이퍼파라미터(hyperparamter)

신경망의 가중치나 편향은 경사하강법에 의해 자동적으로 획득되지만, 학습률은 그렇지가 않다. 학습률과 같이 인간이 직접 할당해야 하는 변수를 하이퍼파라미터라고 한다. 하이퍼파라미터는 여러 후보의 값 중에서 시험을 통해 가장 잘 학습하는 값을 찾는 과정을 거쳐야 한다.

2.6 수치 미분

컴퓨터에서는 극한을 계산하지 못한다. 즉, 컴퓨터는 미분을 해석적으로 계산할 수 없다. 이를 해결하기 위해 수치미분을 사용한다. 수치미분은 도함수에 정의에 입각하여 입력값이 미세하게 변할 때 함수값은 얼마나 변하는지를 알 수 있다. 이때 도함수의 정의에 의해 분모의 변화량을 충분히 작은 상수로 설정할 수 있기 때문에 컴퓨터에서 근사한 미분값을 구할 수 있다.

```
def numerical_diff(f, x):
    h = 1e-4
    return (f(x+h) - f(x-h)) / (2*h)
```

그림 20 수치 미분 계산 (사이토 고키의 밑바닥부터 시작하는 딥러닝: 파이썬으로 익히는 딥러닝 이론과 구현, Aug 21, 2020)

수치 미분은 다변수함수에서도 적용된다. 이때 다변수함수에서는 편미분이라는 개념이 도입되므로 미분할 변수에 대해서만 미분하고, 그 이외의 나머지 변수들은 상수로 취급한다. 물론 모든 변수들이 이러한 편미분 과정을 가진다.

$$y = f(x_1, x_2) = 3x_1^4 + 2x_1^2x_2^2 + 7x_2^4$$

$$\frac{\partial y}{\partial x_1} = 12x_1^3 + 4x_1x_2^2$$

$$\frac{\partial y}{\partial x_2} = 4x_1^2x_2 + 28x_2^3$$

그림 21 다변수 함수에서의 수치 미분 (사이토 고키의 밑바닥부터 시작하는 딥러닝: 파이썬으로 익히는 딥러닝 이론과 구현, Aug 21, 2020)

2.7 오차 역전파

수치미분의 구현은 쉽지만 계산이 오래걸린다는 단점이 있다. 함수의 변수의 개수가 많아지거나 층과 노드의 개수가 많아질수록, 처리하는 시간은 더 많이 걸릴 것이다. 이를 해결하기 위해 도입된 개념이 오차 역전파이다. 오차역전파에 대해 살펴보도록 하자.

다음은 오차 역전파에 대한 간략한 그림이다. 순방향과 달리 반대 방향으로 국소적 미분을 하고 있다는 것을 알 수 있다. 이때 이 국소적 미분을 인간이 해석적으로 계산할 수 있으며, 이 미분에 이전 값을 곱해서 구한다. 여기서 알 수 있는 점은 연쇄법칙으로 여러 층의 국소적 미분을 하나로 정리할 수 있다는 것이다. 다음 그림을 통해 이해할 수 있다.

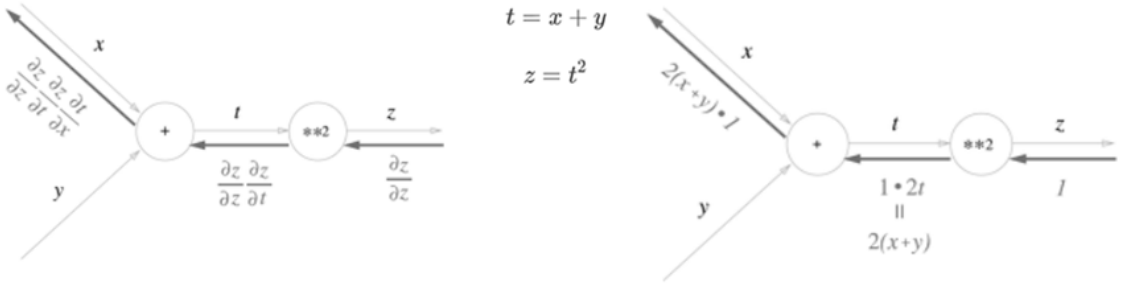


그림 22 미분을 이용한 오차 역전파 (EXCELSIOR-JH's tistory, Aug 21, 2020)

예시로 덧셈 노드의 역전파와 곱셈 노드의 역전파를 다음 그림에 나타냈다.

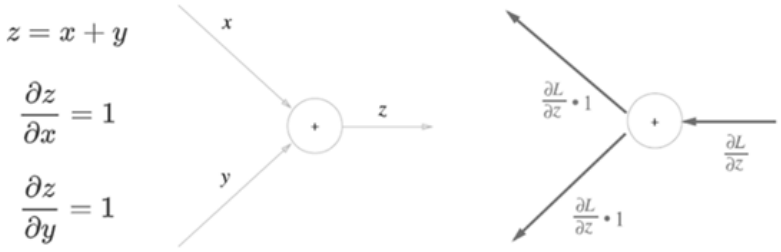


그림 23 덧셈 노드 역전파 (EXCELSIOR-JH's tistory, Aug 21, 2020)

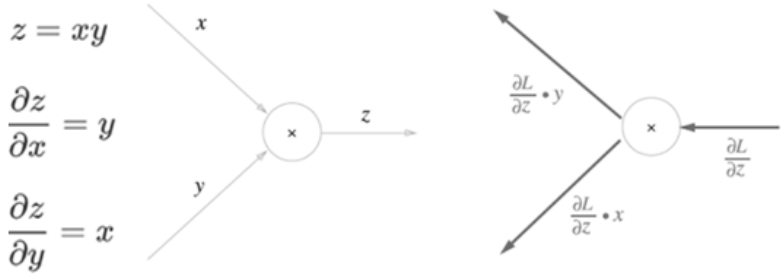


그림 24 곱셈 노드 역전파 (EXCELSIOR-JH's tistory, Aug 21, 2020)

이와 같이 역전파의 의의는 미분을 인간이 직접 계산하고, 그 미분한 값을 컴퓨터에 입력할 수 있다는 것이다. 이를 통해 컴퓨터는 더 빠른 처리 속도를 가질 수 있다. 다만, 미분을 인간이 한다는 점에서 인간의 실수가 일어날 수도 있다는 점을 유의해야 한다.

역전파는 단순히 덧셈, 곱셈뿐만 아니라 다양한 함수에 대해서도 구할 수 있다. 또한, 행렬에 대한 역전파도 구할 수 있으므로 이를 잘 활용하면 더 효율이 좋은 딥러닝 네트워크를 구축할 수 있다. 추가적으로, MSE나 CEE와 같은 손실함수가 그러한 식으로 만들어진 이유는 오차역전파이다. 역전파를 구하게 되면 각각의 손실함수에 대한 미분을 더 간단한 수식으로 나타낼 수가 있다.

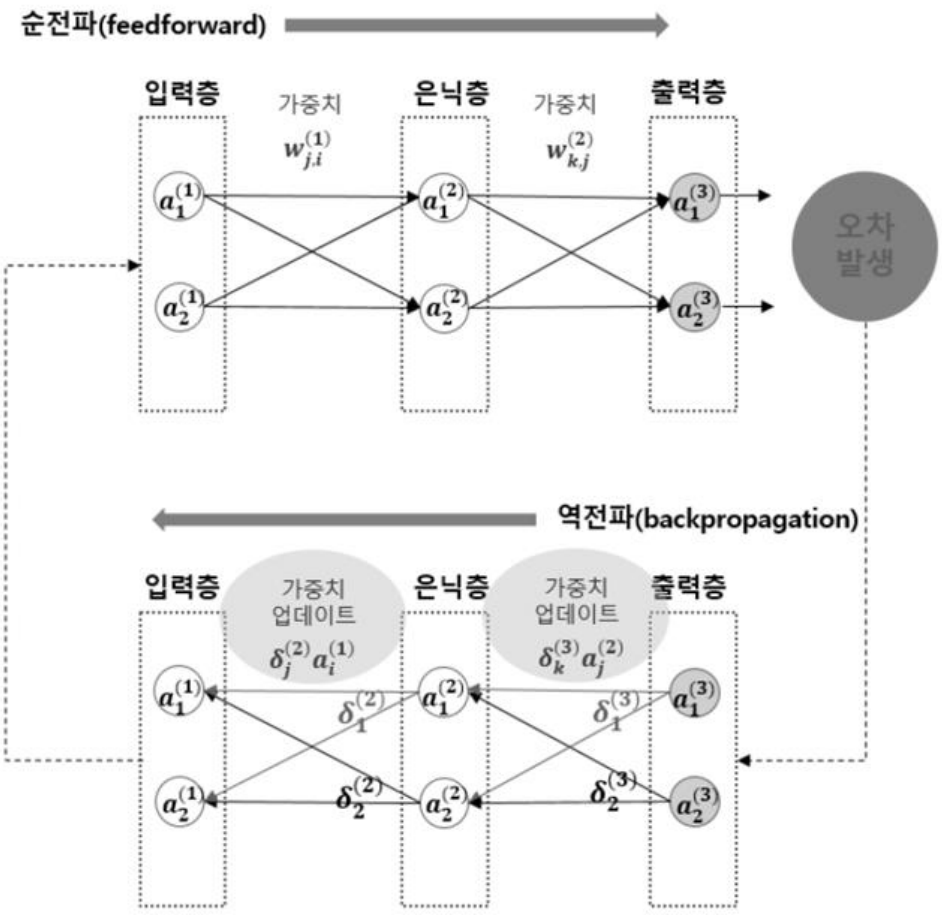


그림 25 순전파와 역전파 과정 (옥수별's blog, Aug 21, 2020)

2.8 행렬(Matrix)

딥러닝 계산 작업은 기본 행렬 연산인 행렬 덧셈, 뺄셈, 전치 등으로 이루어진다.

행렬 덧셈과 뺄셈

$$A=\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix},\ B=\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$\Rightarrow A \pm B=\begin{pmatrix} a_{11} \pm b_{11} & a_{12} \pm b_{12} \\ a_{21} \pm b_{21} & a_{22} \pm b_{22} \end{pmatrix}$$

그림 26 행렬의 덧셈과 뺄셈 연산 (jenemia's tistory, Aug 21, 2020)

행렬의 스칼라 곱

$$\mathrm{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

$$k\mathrm{A} = \begin{pmatrix} ka_{11} & ka_{12} \\ ka_{21} & ka_{22} \end{pmatrix}$$

그림 27 행렬의 스칼라 곱 연산 (jenemia's tistory, Aug 21, 2020)

행렬의 곱

행렬 곱 AB에 대하여 행렬 A의 열의 수와 행렬 B의 행의 수가 같아야 하며 행렬 곱은 교환법칙이 성립되지 않는다.

$$A=\begin{pmatrix} a & b \\ c & d \end{pmatrix},\ B=\begin{pmatrix} e & f \\ g & h \end{pmatrix}\text{의 곱}$$

$$AB=\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} e & f \\ g & h \end{pmatrix}=\begin{pmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \times \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} = \begin{pmatrix} 5 & 11 \\ 11 & 25 \\ 17 & 39 \end{pmatrix}$$

그림 28 행렬 곱 연산 (jenemia's tistory, Aug 21, 2020)

전치 행렬

전치 행렬은 각 원소의 행과 열을 바꾼 행렬을 말한다.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

그림 29 전치 행렬 연산 (jenemia's tistory, Aug 21, 2020)

3. CNN

3.1 CNN(Convolutional Neural Network)

CNN이란 말 그대로 'convolution'이라는 작업이 들어가는 Neural Network이다. CNN은 이미지를 인식하기 위해 패턴을 찾는 데 특히 유용하다. 데이터에서 직접 학습하고 패턴을 사용해 이미지를 분류한다. 즉, 특징을 수동으로 추출할 필요가 없는 것이다. 이러한 장점 덕분에 자율주행자동차, 얼굴 인식과 같은 객체 인식 등에 많이 사용된다.

3.2 합성곱 층 (Convolutional Layer)

DNN의 가장 기본적인 구조는 완전연결 계층(Fully Connected Layer, FC layer)으로 전 layer와 후 layer가 모두 연결된 구조이다.

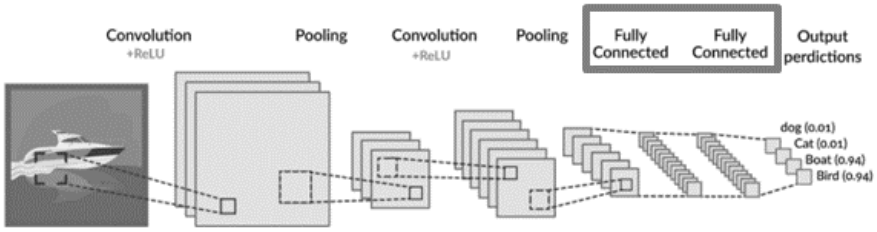


그림 30 Convolutional layer (ebb and flow's tistory, Aug 19, 2020)

위의 그림 CNN 과정에서 네모로 표시된 부분이 Fully Connected 과정이며 이것은 CNN을 이용해 추출한 feature를 최종적으로 어떤 이미지인지 분류하기 위해서 연결되어 있는 layer이다. CNN은 위와 같이 마지막 부분에 FC layer를 이용하는 경우가 많다.

CNN에서의 합성곱 층이 아닌 완전연결 계층(FC layer)만을 이용해 MNIST(Mixed National Institute of Standards and Technology) 데이터셋(필기 숫자의 분류를 위한 학습 데이터 집합)을 분류하는 모델을 만든다고 가정하면, 각각 높이, 폭, 채널로 이루어진 3차원의 데이터(28, 28, 1)를 입력층으로 넣기 위해 평평한 데이터인 1차원으로 만드는 차원 축소 과정을 거쳐야 한다.

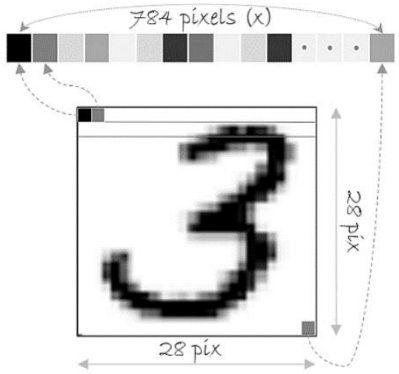


그림 31 28 * 28 pixel (Wikidocs, Aug 19, 2020)

28 x 28 x 1 픽셀의 3차원 데이터를 1차원으로 만들면 784개의 픽셀로 이루어진 데이터가 만들어진다. 하지만 차원 축소로 인해 데이터의 형상이 무시된다는 문제점이 생긴다. 3차원 데이터로 존재했을 때에는 공간적 구조를 가져 3차원 공간 정보를 가지지만 1차원일 때에는 그러한 정보가 사라지게 된다. 이와 같은 문제점을 해결하기 위해 CNN에서 완전연결 계층 대신 합성곱 층을 사용한다. 합성곱 층은 입력 데이터의 형상을 유지하기 때문에 3차원의 이미지를 입력층으로 입력 받고 3차원 데이터로 출력하여 다음 layer로 전달한다. 따라서 CNN은 이미지 데이터와 같이 형상을 가지고 있는 데이터를 다른 신경망보다 더 정확하게 학습한다.

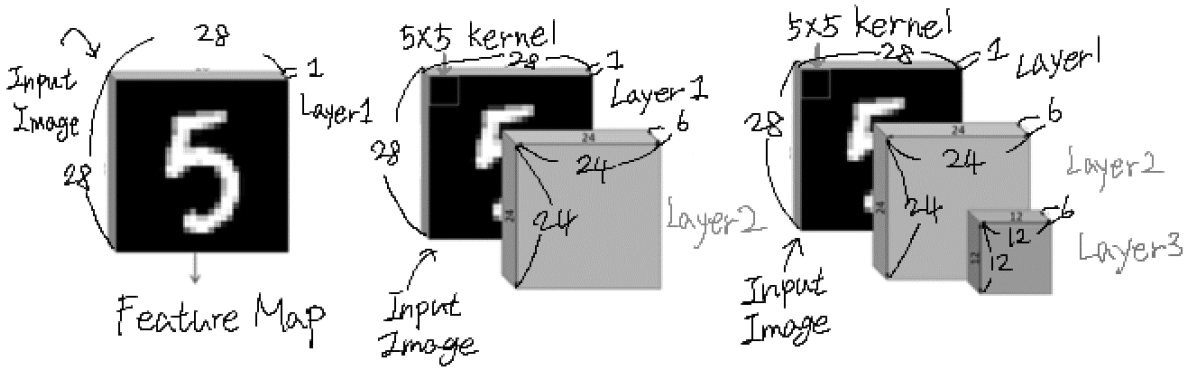


그림 12 Convolutional layer (Excelsior-JH's tistory, Aug 19, 2020)

합성곱 층에서는 완전연결 신경망과 같이 각 유닛(뉴런)이 앞 계층의 모든 유닛과 연결되어 있지 않고 각각의 유닛은 이전 계층에서 근접해 있는 몇 개의 유닛들에만 연결이 된다. 즉 합성곱 층은 입력 이미지의 모든 픽셀과 연결되는 것이 아니라 수용영역(receptive field) 이라고도 불리는 필터(Filter) 안에 있는 픽셀에만 연결되어 있다. 또한 모든 유닛은 이전 계층에 동일한 방법으로 연결되어 있어 같은 값의 가중치와 구조를 공유한다. 그리고 그 연결 사이에 합성곱 연산이 있고 그 신경망을 합성곱 신경망이라고 한다.

합성곱 층 연산

합성곱 연산은 필터를 일정한 간격으로 이동해가며 계산한다.

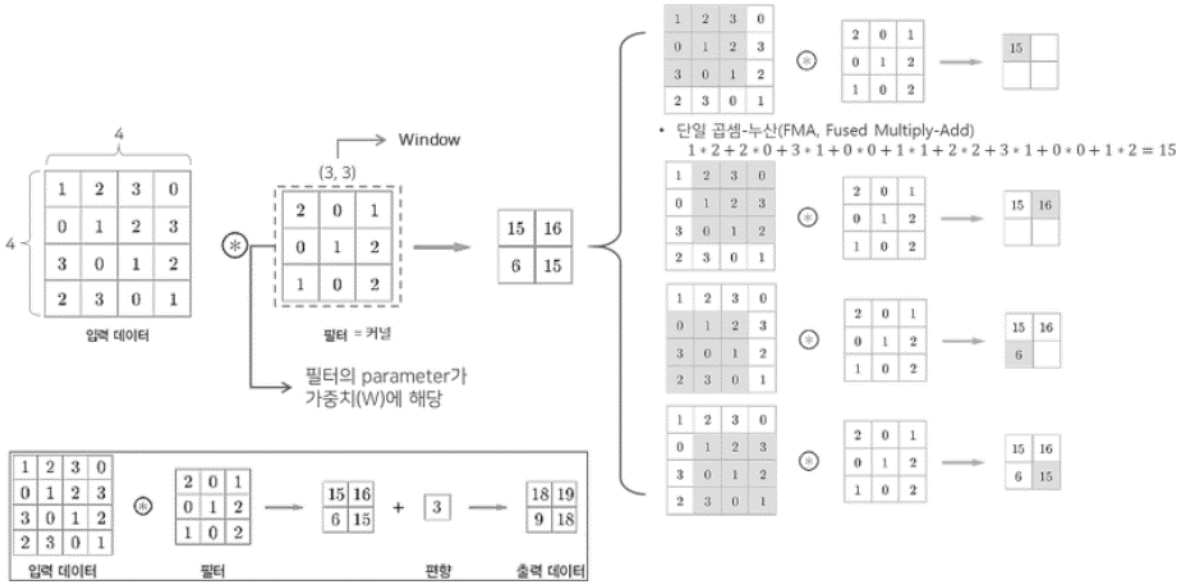


그림 33 Fused Multiply-Add (FMA) (Excelsior-JH's tistory, Aug 19, 2020)

위의 그림처럼 합성곱 연산은 입력 데이터와 필터간에 서로 대응하는 원소끼리 곱한 후 총합을 구하는 것이고 이를 Fused Multiply-Add(FMA) 라고 한다. 편향(bias)은 필터를 적용한 후에 더해주어 계산한다.

3.3 필터

수용영역을 합성곱 층에서 필터(Filter) 또는 커널(Kernal) 또는 윈도우(window)라고 하며 필터는 그 특징이 데이터에 존재하는지 검출해주는 함수이다.

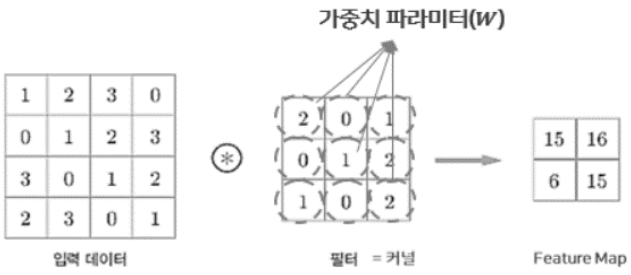


그림 34 CNN Filter (Excelsior-JH's tistory, Aug 19, 2020)

필터는 합성곱 층에서 가중치 파라미터인 w에 해당한다. 학습 단계에서 적절한 필터를 찾도록 학습되며 합성곱 층에서 입력 데이터에 필터를 적용하여 필터와 유사한 이미지의 영역을 강조하는 특성맵(Feature Map)을 출력하여 다음 층으로 전달한다.

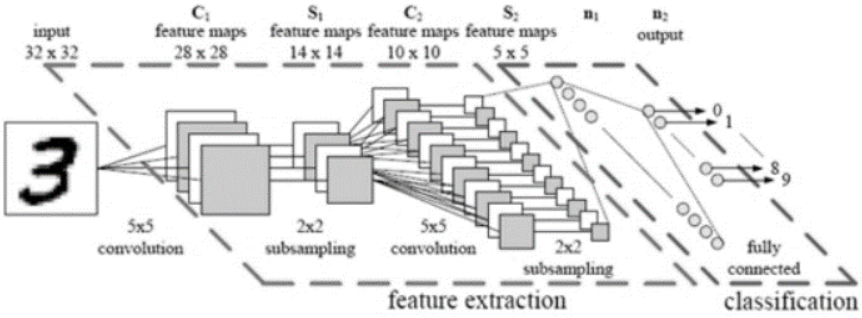


그림 35 Convolutional layer (안재홍, CNN 기반, Multi-class 이미지 분류 기술, Aug 19, 2020)

정리하면 입력 데이터를 필터로 합성곱 연산을 하여 데이터의 특징(feature)을 추출하고 이러한 합성곱 layer를 여러 계층으로 연결한다. 합성곱 layer을 지날 때마다 저수준의 특징들이 점차 고수준의 특징들로 만들어진다. 그런 후에 마지막에는 완전연결 계층으로 최종 결과를 학습한다.

3.4 Channel

대부분이 흔히 알고 있는 색의 3원색인 RGB는 Red, Green, Blue이다. 이를 이미지의 채널에도 적용할 수 있는데, 일반적으로 생각하는 컬러 이미지는 Red채널, Green채널, Blue채널로 이루어져 있다. 즉, 컬러 사진인 경우, 각 픽셀을 RGB 3개의 실수로 표현한 3차원 데이터이고 3개의 채널로 구성되며, 흑백 사진의 경우에는 2차원 데이터로 1개의 채널로 구성된다. 보통 연산량(오차)을 줄이기 위해 전처리에서 이미지를 흑백으로 만들어 처리한다. 만약 높이가 30픽셀이고 폭이 20픽셀인 컬러 사진 데이터의 shape은 (30, 20, 3)으로 표현하고, 흑백인 경우엔 (30, 20, 1)로 표현한다.

3.5 Pooling

이미지의 크기를 계속 유지하며 **FC(Fully Connected) layer**로 가게 된다면 연산량은 기하급수적으로 늘어날 것이다. 적당히 크기도 줄이고, 특징을 강조할 수 있어야 하는데 그 역할을 하는 것이 **Pooling**이다. Pooling은 쉽게 convolution layer를 거쳐서 나온 activation maps가 있을 때, 이를 이루

는 convolution layer를 resizing하여 새로운 layer를 얻어 특정 데이터를 강조하는 용도로 사용된다. 또는, matrix연산을 사용하지 않고 각 픽셀에서 하나의 값을 뽑아내는 과정으로도 이해할 수 있다.

Pooling의 처리방법으로는 최댓값을 뽑아내는 max pooling, 평균값을 뽑아내는 mean pooing, 최솟값을 뽑아내는 Min pooling이 있는데 CNN에서는 주로 **Max Pooling**을 사용한다. 일반적으로 pooling의 크기와 stride의 크기를 같게 설정하여 모든 원소가 한 번씩은 처리되도록 한다.

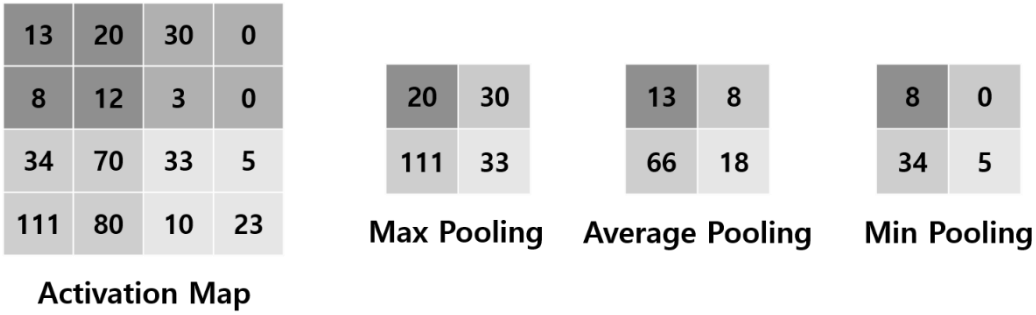


그림 36 Pooling (GRU's github, Aug 19, 2020)

3.6 완전 연결형 레이어 (Fully Connected layer)

Fully Connected layer (FC layer) 는 완전 연결형 레이어로 합성곱 층의 마지막에 존재한다. FC layer는 앞의 Convolution, Relu, Pooling 의 과정을 반복하면서 나온, 가장 마지막 activation map 들의 pooling 까지 끝났을 때, 각각의 클래스마다 점수를 도출해 내기 위해 그 이미지를 입력으로 받는다. 마지막 Convolution layer 의 출력은 3 차원 volume 으로 이루어진다. 이를 전부 펴서 1 차원 벡터로 만들어 FC layer 의 입력으로 사용한다. 전부 다 하나로 통합시키고, 최종적인 추론을 하는 것이다.

FC layer 는 일반적인 input layer, hidden layer, output layer 를 가지는 부분이다. 이 부분은 실제로 분류를 해주기 위해 존재한다. CNN 은 보통 이미지 분류와 같은 영역에서 자주 사용되는데, 이때 주어진 이미지를 분류해야 하므로 FC layer 를 이용해 실제 분류를 해내는 것이다. 즉, FC layer 의 앞에 있는 Conv layer 는 주어진 이미지 데이터에 대해 N 개의 필터를 이용해 올바른 특징을 뽑아내는 역할을 하고, FC layer 는 그 특징들을 기반으로 실제 분류를 실행한다.

3.7 CNN의 Parameter

- Convolution filter의 개수

각 layer에서의 연산시간을 비교적 일정하게 유지하며 시스템의 균형을 맞추는 것이 좋다. 보통 Pooling layer를 거치면 출력이 1/4로 줄어들기 때문에 convolution layer의 결과인 feature map의 개수를 4배 정도 증가시키면 된다.

- Filter의 Size

작은 filter를 여러 개 중첩하면 원하는 특징을 더 부각하며 연산량을 줄일 수 있다.

- Padding 여부와 Padding Size

Padding을 사용하게 되면 입력 이미지의 크기를 줄이지 않을 수 있다. 아래에 관련 정보를 추가했다.

- Stride의 간격

stride는 filter의 이동 간격을 조절하는 매개변수이다. 이 값이 커지면 결과 데이터의 크기가 작아지게 된다.

- Pooling의 크기

데이터의 크기를 줄이며 특징을 강조해준다.

(Convolution layer 다음에 Pooling layer가 온다면, Feature map의 행과 열 크기는 Pooling 크기의 배수여야 한다. 이 조건을 만족하도록 CNN의 Parameter를 조절해야 한다.)

Padding

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

Image
(+zero Padding)

그림 37 Padding (GRU's github, Aug 19, 2020)

Padding은 Convolution을 수행하기 전, 입력 데이터(이미지) 주변을 특정 픽셀 값(보통은 0)으로 채워 늘리는 것이다. Padding을 사용하게 되면 입력 이미지의 크기를 줄이지 않을 수 있다. Convolution에서는 filter와 stride의 작용으로 convolution layer는 입력 데이터(이미지)보다 작아지게 된다. 이렇게 입력 데이터보다 출력 데이터의 값이 작아지는 것을 방지하는 방법이 바로 Padding이다. Padding을 하면 convolution을 해도 크기가 작아지지 않는다.

3.8 Convolution layer 출력 데이터 크기

입력 데이터 높이: H　　입력 데이터 폭: W

필터 높이: FH　　필터 폭: FW

Stride 크기: S　　Padding 크기: P

$$\text{Output Height} = OH = \frac{(H + 2P - FH)}{S} + 1 \quad \text{OutputWeight} = OW = \frac{(W + 2P - FW)}{S} + 1$$

그림 38 출력 데이터 크기 계산 (taewan kim's blog, Aug 19, 2020)

위 식의 결과는 자연수가 되어야 한다. 또한, Convolution layer 다음에 Pooling layer가 온다면,

Feature map의 행과 열 크기는 Pooling 크기의 배수여야 한다. 예를 들어, 만일 Pooling의 사이즈가 (3, 3)이라면 위 식의 결과는 자연수이고 3의 배수여야 한다. 이를 만족하도록 Parameter (filter의 크기, stride의 간격, pooling의 크기 및 패딩 크기)를 조절하여야 한다.

4. CNN Simulation

이 장에서는 CNN을 구현하기 위한 코드를 적어놓았다. 작성한 코드는 b스카이비전의 코드를 참고했다. 코드의 기본적인 정보는 다음과 같다.

1. 데이터셋

CIFAR-10 데이터셋은 10종류의 이미지가 담겨져 있다. 50000개의 훈련 이미지와 10000개의 테스트 이미지가 있다. 사진은 RGB이며, 32 x 32의 크기를 가지고 있다. 정답은 인덱스로 되어 있으므로 결과를 출력할 때는 이미지의 이름을 보기 위하여 리스트에 따로 적어두는 작업을 한다.

2. CNN 신경망

가로, 세로, RGB의 정보가 담긴 이미지는 3차원이다. 합성곱은 RGB의 경우만 고려하면 되므로 각각의 필터 3개를 만든다. 신경망은 Convolution 과정과 Max Pooling 과정을 차례로 반복하며 데이터의 feature를 추출하였고 여기에서 데이터 출력을 위해 relu 라는 활성화 함수를 이용했다. 마지막 과정으로 완전연결 계층을 이용해 최종 이미지를 분류했고 여기서 softmax 활성화 함수를 이용했다.

구조도: Conv - Pooling - Conv - Pooling - Affine(Full Conected) - Softmax

3. 학습 순서도

데이터셋을 불러옴 -> 데이터 전처리 -> 신경망에 훈련용 데이터를 넣어 훈련 -> 테스트용 데이터를 넣고 정확도 및 범용도를 검증

4. 테스트 이미지 출력

마지막 부분에서는 임의의 테스트 데이터를 넣어보고 CNN 신경망이 판단한 결과를 시각화하기 위

한 함수를 만들었다. 예측값, 오답이 나온 이미지 등 여러 정보를 확인할 수 있다.

코드는 핵심적인 내용만 넣었다. 이외의 코드는 깃허브(Github)에 올려놓았다.

"https://github.com/dIronmanb/AI.Challenge/blob/master/CIFAR-10(completed).ipynb"

```
#0. 데이터 로드
cifar10 = datasets.cifar10 # cifar10 에 데이터셋 cifar10 대입
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data() # cifar-10 데이터셋 다운로드
```

```
#1. 데이터 구조 파악

# 학습할 이미지 데이터셋과 라벨 형태 출력
print("Train samples:", train_images.shape, train_labels.shape)
print("Test samples:", test_images.shape, test_labels.shape)
```

```
Train samples: (50000, 32, 32, 3) (50000, 1)
Test samples: (10000, 32, 32, 3) (10000, 1)
```

```
#2. 데이터 전처리

# numpy 라이브러리 reshape 함수를 이용하여 적절한 형태의 배열로 변환
train_images = train_images.reshape((50000, 32, 32, 3))
test_images = test_images.reshape((10000, 32, 32, 3))

# 픽셀값을 0과 1사이의 값으로 정규화 (RGB는 각각 0부터 255까지의 숫자를 가질 수 있음)
train_images = train_images/255.0
test_images = test_images/255.0
```

```
#3. CNN  Layer 구성

# 선형으로 계층을 쌓는 모델 만들기
model = models.Sequential()

# Convolution layer 추가
# 인수: 합성곱 필터수: 32, 필터 모양: 3 x 3, 행의 수(32) x 열의 수(32) x 채널 수(RGB이므로 3)
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))

# Max Pooling layer 추가 (2 x 2의 형태의 풀 사이즈로 max polling 하기)
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# 2차원 데이터를 1차원으로 바꿔주는 계층 추가
model.add(layers.Flatten())

# dense:이전층과 다음층을 완전연결 - 다음 뉴런을 64개로 설정, 활성화함수로 relu 사용
model.add(layers.Dense(64, activation='relu'))
# 출력 뉴런: 10개, 활성화함수를 softmax
model.add(layers.Dense(10, activation='softmax'))

# 모델 컴파일
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
#optimizer(모델을 업데이트 하는 방법)는 adam을 이용
#loss function(훈련하는 동안 모델의 오차 측정)은 sparse_categorical_crossentropy를 이용
#metrics(훈련 단계와 테스트 단계를 모니터링 함)는 올바르게 분류된 이미지의 비율인 정확도를 사용


# 모델 훈련 단계로 모델이 이미지와 라벨을 매핑하는 방법을 학습, epochs(전체 데이터 학습 반복 횟수)는 10번으로 설정
model.fit(train_images, train_labels, epochs=10)
#테스트 데이터로 평가
test_loss, test_acc = model.evaluate(test_images, test_labels)
# 정확도 출력
print('Test accuracy:', test_acc)
```

그림 39 CIFAR-10 이미지 기반의 딥러닝 모델 (Aug 28, 2020)

5. Results

CNN 모델은 약 70%의 정확도를 보였다. 각 이미지에 대한 예측값을 그래프로 나타낼 때, 오답인 경우는 해당 이미지의 정답보다 다른 답이 더 높은 것을 볼 수 있다. Softmax로 마지막 계층을 구현했으므로 각 이미지가 얼마나 정답에 근사한지를 확률로 볼 수 있다. 정답 이미지에서 정답을 가질 확률은 낮게는 50%, 높게는 99%까지의 결과를 보였다. 오답인 경우, 이들도 대체로 정답일 확률이 두 번째로 크고 이외의 다른 답들이 도리 확률은 적었다. (코드를 실행하면 이해가 될 것이다.) 즉, 모델이 정답를 고를 수는 있지만, 오답을 고를 확률이 더 높았기 때문임을 짐작할 수 있다.

```
Epoch 1/10
1563/1563 [==============================] - 71s 45ms/step - loss: 1.5413 - accuracy: 0.4332
Epoch 2/10
1563/1563 [==============================] - 69s 44ms/step - loss: 1.1638 - accuracy: 0.5864
Epoch 3/10
1563/1563 [==============================] - 69s 44ms/step - loss: 1.0135 - accuracy: 0.6423
Epoch 4/10
1563/1563 [==============================] - 70s 44ms/step - loss: 0.9150 - accuracy: 0.6789
Epoch 5/10
1563/1563 [==============================] - 71s 46ms/step - loss: 0.8479 - accuracy: 0.7028
Epoch 6/10
1563/1563 [==============================] - 70s 45ms/step - loss: 0.7926 - accuracy: 0.7229
Epoch 7/10
1563/1563 [==============================] - 70s 45ms/step - loss: 0.7487 - accuracy: 0.7382
Epoch 8/10
1563/1563 [==============================] - 70s 45ms/step - loss: 0.7091 - accuracy: 0.7515
Epoch 9/10
1563/1563 [==============================] - 70s 45ms/step - loss: 0.6740 - accuracy: 0.7652
Epoch 10/10
1563/1563 [==============================] - 73s 46ms/step - loss: 0.6372 - accuracy: 0.7763
313/313 [==============================] - 4s 14ms/step - loss: 0.8743 - accuracy: 0.7122
Test accuracy: 0.7121999859809875
```

그림 40 딥러닝 모델 트레이닝 및 정확도 (Aug 28, 2020)

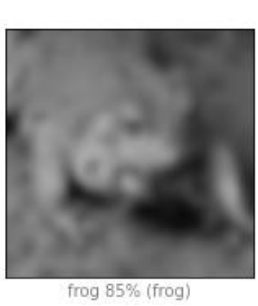


그림 41 예측값이 정답인 경우 (Aug 28, 2020)

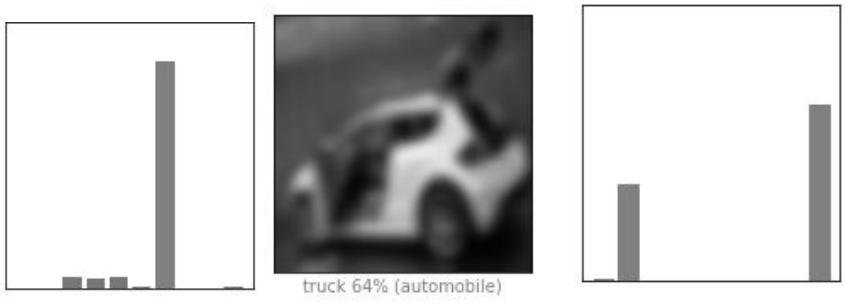


그림 42 예측값이 오답인 경우 (Aug 28, 2020)

6. Conclusion

이미지 분류를 위해 퍼셉트론에 CNN을 사용하는 이유는 합성곱의 이점 때문이다. 합성곱은 2차원 행렬로 이루어진 이미지에서, 각 픽셀에 인접한 데이터값들의 비슷한 관계을 알아낼 수 있다. Pooling 역시 합성곱과 비슷한 성질을 가지고 있으므로 CNN구현에 필수적인 요소가 된다.

같은 코드를 실행하면 정확도가 약간씩 차이가 나지만, 대부분 70% ~ 80% 정도였다. 정확도를 더 늘리기 위해서는 학습률이나 레이어 층의 개수, 해당 레이어의 노드 개수 등의 하이퍼 파라미터를 조

정해야 할 필요가 있었다. 이외에도 드롭아웃(Dropout) 가중치 초기값 등 여러가지 해결법이 존재한다. 인공지능 구현에 있어서 하드웨어적인 부분도 필수적이다. 작성한 코드는 10분 내지 20분이면 모델을 구축할 수 있지만, 층이 더 깊어지고 노드 수가 많아질수록 시간은 기하급수적으로 늘어난다. 따라서 GPU를 활용한 컴퓨팅 기술이 필요하다. 향후 하이퍼 파라미터를 조정하는 강화학습과 CNN이외에도 RNN, LSTM등의 딥러닝을 공부할 것이다. 이미지 인식, 이미지 분할 등과 같이 더 심화된 컴퓨터 비전도 설계할 것이다.

'이번 학술지 작성을 통해 CNN에 관한 전반적인 지식을 새롭게 얻고, 복습할 수 있었으며, 여러 프로그램을 보고 직접 만들며 인공지능 분야에 한 발짝 다가갈 수 있었습니다. 처음엔 어려워 보여서 조금 걱정이 됐지만, 기초부터 차근차근 공부하며 이해하니 재미를 붙일 수 있었습니다. 비슷한 생각을 갖고 계신 분들이 이 학술지를 보고 조금이나마 이미지분류에 관심을 갖고 공부하셨으면 좋겠다는 생각을 갖고, 최대한 이해하기 쉽도록 작성했습니다. 이미지 분류, CNN을 처음 공부하시는 분들께 이 글이 도움이 된다면 좋겠습니다.'

7. Reference

논문	➢ 안재홍,"CNN 기반, Multi-class 이미지 분류 기술." 석사학위, 한양대학교 대학원: 전자공학과, 2018. 8
웹페이지	➢ "CNN, Convolutional Neural Network 요약" (Aug 2, 2020). TAEWAN.KIM 블로그. last modified Jan 4, 2018 http://taewan.kim/post/cnn/ ➢ "33. TensorFlow 2.0 Release 및 변경사항 정리 - Tf.Session & Tf.Placeholder 삭제, @Tf.Function, Tf_upgrade_v2" (Aug 18, 2020) 솔라리스의 인공지능 연구실. last modified Oct 3, 2019 http://solarisailab.com/archives/2640 ➢ "CIFAR-10 이미지 분류를 위한 CNN 을 구성해보자!(Keras)" (Aug 7, 2020) 호롤리한 하루. last modified Aug 20, 2019 https://gruuuuu.github.io/machine-learning/cifar10-cnn/# ➢ "배치(batch)와 에포크(epoch)란?" (Aug 2, 2020). b 스카이비전. last modified Jun 11, 2020 https://bskyvision.com/803 ➢ "데이터셋 이야기" (Aug 2, 2020). tykimos.github.io . last modified Mar 25, 2017 https://tykimos.github.io/2017/03/25/Dataset_and_Fit_Talk/

웹페이지	➢ "CNN(Convolutional Neural Network)이란?". (Aug 2, 2020). Medium.com. last modified Aug 1, 2018 https://medium.com/@hobinjeong/cnn-convolutional-neural-network-9f600dd3b395 ➢ "CNN 에서 pooling 이란?" (Aug 2, 2020). medium.com. last modified Aug 1, 2018https://medium.com/@hobinjeong/cnn%EC%97%90%EC%84%9C-pooling%EC%9D%B4%EB%9E%80-c4e01aa83c83 ➢ "[딥러닝 강의][1] 데이터 기반 이미지 분류(Data-driven Image Classification)". (Aug 2, 2020) Deep Play. last modified Jan 23, 2020 https://3months.tistory.com/512 ➢ "무이메이커스_딥러닝을 활용한 인공지능(AI) 이미지 분류(Image Classification) 프로젝트". (Aug 2, 2020). honeycomb-makers. last modified Jun 17, 2019 https://honeycomb-makers.tistory.com/3 ➢ "15.텐서플로우(TensorFlow)를 이용해서 CIFAR-10 이미지 분류(Image Classification)를 위한 Convolutional Neural Networks(CNNs) 구현해보기(간결한 코드) - MNIST 다음 단계 예제". (Aug 18, 2020) 솔라리스의 인공지능 연구실. last modified Nov 28, 2017 http://solarisailab.com/archives/2325 ➢ "[Anaconda_python]CIFAR-10 데이터셋으로 이미지 분류기 만들기(컨볼루션 신경망)" (Aug 18, 2020) b 스카이비전. last modified Feb 6,2020 https://bskyvision.com/700 ➢ "[Deep Learning] Convolutional Neural Network". (Aug 21, 2020) 거북이. last modified Jun 10, 2017 https://operatingsystems.tistory.com/entry/Data-Mining-Convolutional-Neural-Network ➢ "[CS231n] 5. Convolutional Neural Networks". (Aug 21, 2020). 참신러닝(Fresh - Learning) last modified Arp 17, 2020 https://leechamin.tistory.com/94
도서	➢ 사이토 고키. 밑바닥부터 시작하는 딥러닝: 파이썬으로 익히는 딥러닝 이론과 구현. 서울: 한빛미디어. 2017. 01. 02 ➢ 조태호. 모두의 딥러닝: 원리를 쉽게 이해하고 나만의 딥러닝 모델을 만들 수 있다!. 서울: 길벗. 2017. 12. 27 ➢ 오렐리앙 제롱. 핸즈온 머신러닝: 사이킷런, 케라스, 텐서플로 2 를 활용한 머신러닝, 딥러닝 완벽 실무. 서울: 한빛미디어. 2020. 05. 04 ➢ 다니엘 첸. Do it! 데이터 분석을 위한 판다스 입문. 서울: 이지스퍼블리싱. 2018. 10. 18