Appendix B. 딥러닝 주요 개념

딥러닝의 기초 개념으로서 단일 신경망과 다층 신경망이 어떻게 다른지 살펴봅니다. 그리고 신경망의 주요 개념인 가중치, 활성화 함수, 손실 함수, 오차 역전파법, 경사 하강법, 배치 정규화 개념에 대해 살펴봅니다.

딥러닝 모델로는 합성곱 신경망(CNN), 순환 신경망(RNN, LSTM), 오토 인코더와 생성적 적대 신경망(GAN)을 소개합니다.

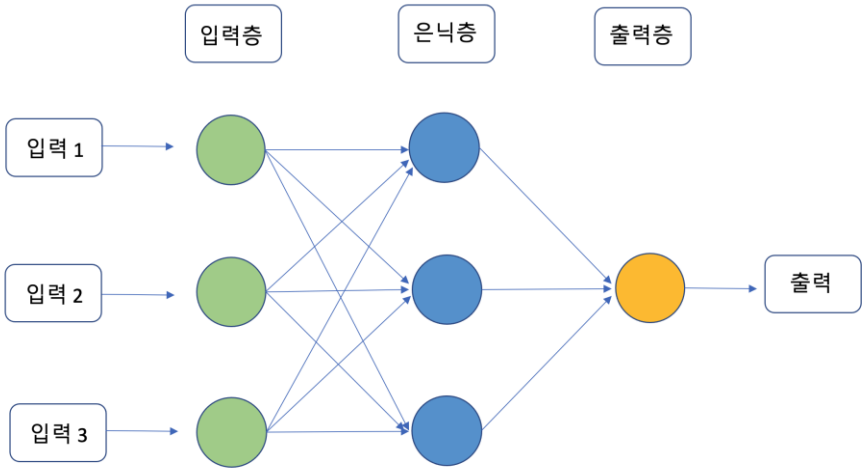
B.1 단일 신경망과 다층 신경망

인류는 산업 혁명기에 접어들면서 스스로의 힘으로 움직이는 기계를 꿈꾸었습니다. 증기 기관을 시작으로 내연 기관, 항공기 엔진, 로켓 엔진 등 이제 기계적인 동력을 얻는 것은 더 이상 꿈이 아닙니다. 19세기에 몇몇 선구자들이 계산하는 기계를 만들기 위해 시도하였고, 20세기에 2차 대전과 냉전을 거치면서 컴퓨터가 급격한 발전을 이루었습니다. 좋은 현상인지 아닌지는 모르겠지만, 생각하는 기계는 이제 손에 잡힐 듯한 거리에 있는 것 같습니다. 그렇다면 생각하는 기계는 어떤 원리로 만들어질까요? 아무리 생각해도 그 기계의 기본적인 논리 구조는 신경망 원리에 기초하여 만들어질 것 같습니다.

인간이 하늘을 나는 기계, 즉 비행기를 개발하기 위해 최초에 도입한 아이디어는 새의 날개를 모방한 각종 시제품을 만들어본 것입니다. 이 방법은 잘못된 시도였을까요? 동력원이 제대로 없었을 때는 이 방법이 잘못 디딘 첫걸음처럼 느껴졌습니다. 그러나 부단한 시도 끝에 날씬한 유선형 기체의 비행기를 갖게 된 지금, 되돌아서 생각해보면 비행기의 날개 단면은 새의 날개 단면 형상과 비슷하고 공기 역학적으로 비슷한 역할을 수행합니다. 일례로 B2 폭격기의 옆면 실루엣은 맹금류가 날 때의 옆면 실루엣과 비슷합니다. 기계가 생물을 닮아가는 수렴 진화의 한 현상입니다. 결과적으로 새를 모방하려던 시도는 성공적인 첫걸음이었습니다.

그럼 생각하는 기계를 제작하려면 최우선으로 모방해야 할 생명체는 무엇일까요? 바로 인간입니다. 그중에서도 두뇌입니다. 2차 대전을 전후한 시기에 생각하는 기계, 즉 본격적인 인공지능 연구가 시작되었습니다. 최초의 아이디어는 인간 두뇌의 신호 처리 방식을 모방하는 것이었습니다. 인공지능의 개발에 있어서 컴퓨팅 성능의 부족과 다층 신경망 학습 방법의 부재 등으로 수십 년의 정체기도 있었습니다. 지금은 이러한 난관이 어느 정도 해소되면서 특정 분야에서는 인간의 능력을 뛰어넘는 다층 신경망 기법이 각광받고 있습니다. 때문에 인간의 두뇌의 작동 원리를 모방하려던 1940년대부터 시작된 첫 시도 역시 올바른 방향으로 판단됩니다.

그렇다면 인간의 두뇌는 어떻게 작동할까요? 그 논리 구조를 기계에 실현시키기 위해 다음과 같은 **단일 신경망** 구조를 살펴보겠습니다. 이 그림은 인간의 두뇌가 신호를 처리하는 과정을 모방한 것입니다.



[그림 B-1] 단일 신경망 구조

위의 그림에서 보이는 단일 신경망 구조는 입력층(input layer), 출력층(output layer), 그리고 이들 사이에 있는 은닉층(hidden layer)으로 구성돼 있습니다. 위의 그림에서는 입력층에 외부 정보를 받아들이는 녹색 점(이하 노드)이 3개 있습니다. 모든 노드는 계산을 실행하는 능력이 있습니다. 은닉층이 하나 있는 경우를 **단일 신경망**이라 하고, 은닉층이 2개 이상 있는 모델을 **다층 신경망**이라 합니다. 그리고 다층 신경망을 **딥러닝 모델**이라고도 부릅니다.

우선 입력층에서 3개의 입력 정보를 받습니다. 그리고 화살표 방향으로 다음 층의 노드로 정보를 보냅니다. 녹색 노드의 입력층에서 받아들인 정보는 파란색 노드의 은닉층을 거쳐 주황색 노드인 출력층까지 보내집니다.

예를 들어 입력 1, 입력 2, 입력 3은 각기 나이, 평균 혈당치, 체질량 지수를 나타낸다고 가정하겠습니다. 이 정보가 녹색 입력층 노드부터 주황색 출력층 노드까지 화살표를 따라 흐르고, 각 노드에서는 계산이 일어납니다. 최종적으로 주황색 출력층 노드에서는 앞의 3가지 입력 정보를 활용해서 타깃 변숫값, 즉 뇌졸중이 발생했는지(1) 아닌지(0)를 판단합니다. 이것이 인간의 두뇌에서 일어나는 일이고, 머신러닝과 딥러닝 신경망에서 실제로 연산이 이뤄지는 방식입니다.

다만, 신경망이 작동하는 세부 프로세스를 알기 위해서는 뒤에서 설명하는 활성화 함수, 손실 함수, 오차 역전파법, 경사 하강법, 배치 정규화 등의 각종 개념들을 알아야 합니다.

B.2 가중치와 활성화 함수

[그림 B-1]을 더 자세히 살펴봅시다. 입력층의 녹색 노드에서 은닉층의 파란색 노드로 총 9개의 화살표가 나가고, 이를 통해 정보가 흐릅니다. 은닉층의 파란색 노드에서 출력층의 주황색 노드로는 총 3개의 화살표가 이어집니다.

수업 에피소드!
대학원 수업에서 신경망 모델을 배울 때 제 머릿속에 떠오른 결정적인 질문이 있었습니다. 뒤에서 추가로 설명하겠지만, 신경망을 구성하는 각 층의 노드는 바로 앞 층의 노드들로부터

정보(결괏값)를 선형 결합으로 받아들입니다. 수학에 능숙하지 못한 제가 이해하지 못했던 것은 다음과 같은 것이었습니다.
"선형 결합으로만 정보가 주욱 흘러가면 어떻게 그리고 어디서 '판단'이라는 과정이 일어나는 걸까?"
이 의문에 답을 하기 위해서는 **가중치**와 **활성화 함수**라는 개념을 알아야 합니다.

녹색 노드가 받아들이는 입력값을 위에서부터 순서대로 G1, G2, G3라고 명명하겠습니다. G는 Green의 약자입니다. 파란색 노드가 갖는 값은 Blue의 약자를 따서 위에서부터 순서대로 B1, B2, B3로 명명하겠습니다. 주황색 출력층이 갖는 값은 Orange의 두 번째 알파벳을 따서 R1이라고 하겠습니다. 은닉층의 첫 번째 그리고 두 번째 파란색 노드의 값은 앞의 노드에서 주어진 입력값들의 단순 합계라고 가정합니다.

$$B1 = G1 + G2 + G3$$

$$B2 = G1 + G2 + G3$$

$$B3 = G1 + G2 + G3$$

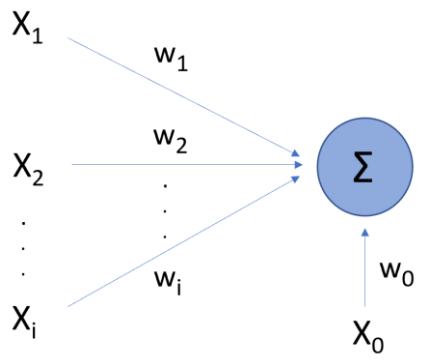
이제 출력층 노드의 값 R1은 다음과 같이 계산됩니다.

$$R1 = B1 + B2 = 3 * (G1 + G2 + G3)$$

'신경망 결괏값이 이렇게 간단하게 나오나?'하고 착각할 수 있습니다. 만약에 입력값 G1, G2, G3값이 각기 1, -2, 3으로 주어진다면 출력값 R1은 6(=3*(1-2+3))이 됩니다. 그런데 여기서 두 가지 문제가 발생합니다. 첫 번째는 우리 두뇌가 하나의 신경 세포에서 다음 신경 세포에 신호를 보낼 때 어떤 세기(배율)로 전달하는가의 문제입니다. 두 번째 문제는 중간에 도출된 선형결합값 6이 무슨 의미를 갖는지에 대한 해석이 필요합니다. 결론부터 말하자면 첫 번째 문제를 해결하기 위해서 **가중치** 개념이 나왔고, 두 번째 문제를 해결하기 위해서 **활성화 함수** 개념이 나왔습니다.

먼저 **가중치** 개념에 대해 설명하겠습니다. 인간은 생존에 필수적인 정보와 부차적인 정보를 구분합니다. 실수로 매우 뜨거운 물체에 손이 닿으면 '위험하다'는 정보가 순간적으로 두뇌로 전달되어 빨리 손을 뗄 수 있게 합니다. 이 경우 '위험하다'는 정보는 두뇌에서 신호 크기를 증폭 전달해서 가능한 최대한 많은 관련 뉴런으로 빠르게 전달될 것입니다.

반면에 좋아하는 취미에 몰두했을 때 주변의 배경 소음은 두뇌의 신호 전달 과정에서 중요하게 취급되지 않습니다. 이러한 신호 전달 크기 조절 역할을 신경망 모델에서 가중치가 맡고 있습니다. 설명의 편의를 위해 가중치가 주어진 신경망에서 입력층의 세 노드 그리고 은닉층의 첫 번째 노드만 부분적으로 그려보겠습니다.



[그림 B-2] 신경망 노드 작동 원리 1

X_1, X_2, X_3는 입력층의 세 노드가 갖는 값을 나타냅니다. w_1, w_2, w_3는 각 입력층에서 은닉층으로 가는 정보에 부가된 가중치를 의미합니다. B1은 위 그림에서 시그마 표시가 있는 파란색 노드입니다.

$$B1 = w_1*X_1 + w_2*X_2 + w_3*X_3$$

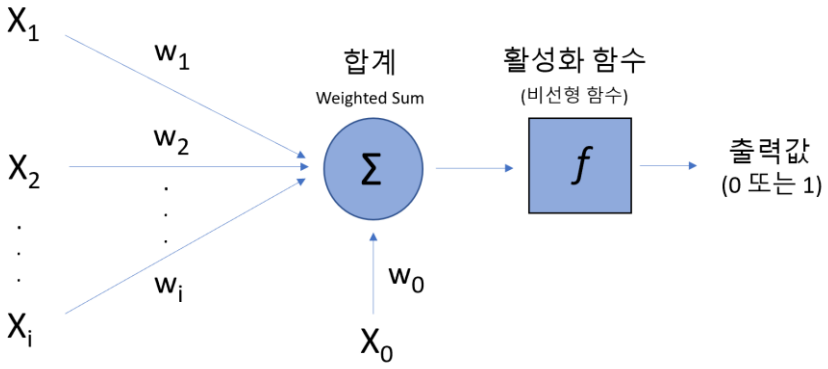
가중치 값이 w_1=1, w_2=0.5, w_3=0.3으로 주어진 경우 B1값을 계산해보겠습니다. X_1, X_2, X_3는 앞에서 설정한 대로 각기 1, -2, 3을 갖고 있습니다.

$$B1 = 1*1 + 0.5*(-2) + (0.3)*3$$
$$= 1 - 1 + 0.9$$
$$= 0.9$$

가중치를 줘서 B1을 계산했더니 값이 0.9가 나왔습니다. 가중치가 없을 때의 단순 합 2($B1=X_1+X_2+X_3$)의 결과와는 다릅니다. 가중치를 도입함으로써 신경망은 단순 합산 기계에서 벗어났습니다. 미약하지만 신경망이 '생각하는 기계'에 한 발 더 다가간 결과입니다.

가중치 1, 0.5, 0.3은 입력된 값을 다음 신경 노드로 100% 그대로 전달, 50% 축소 전달, 30%로 축소 전달합니다. 만약 가중치가 0이면 입력된 값은 다음 노드로 전달되지 않습니다. 즉, 이전 층의 신경 노드 값을 다음 층의 신경 노드로 전달할 때 가중치를 통해서 값을 증폭하거나 축소하여 정보를 선별적으로 전달합니다.

다음은 활성화 함수가 필요한 이유에 대해 설명하겠습니다. 활성화 함수는 하나의 신경망 노드를 중심으로 살펴보면 편리하므로 여기서도 은닉층의 첫 번째 파란색 노드 B1을 중심으로 앞의 입력층 3개의 노드값까지 포함한 그림으로 살펴보겠습니다.



[그림 B-3] 신경망 노드 작동 원리 2

위의 예에서 은닉층의 첫 번째 파란색 노드 B1의 값은 0.9로 나왔습니다. 이 결괏값을 다음 노드로 전달하기 전에 0.9라는 값에 대한 해석이 필요합니다. 이 과정이 없으면 가중치를 도입하더라도 신경망을 흐르는 값들은 앞 단계 입력값들의 선형 결합에 불과합니다. 약간 복잡해진 합계 계산기를 아직 벗어나지 못하고 있습니다. 이런 식으로는 출력층에서 특정 결괏값(예: 7)이 나와도 이게 무엇을 의미하는지 알 수 없습니다.

선형 결합을 벗어나기 위해 예를 들어 0보다 큰 입력값은 그대로 전달하고, 0 이하의 입력값은 0으로 변환하는 렐루(Relu) 함수를 도입하겠습니다. 이 함수를 f(x)로 표기합니다. 이 함수는 비선형 함수입니다. 은닉층의 첫 번째 노드인 B1이 비선형 함수 f(x)를 포함한다면 B1값은 다음과 같이 구해집니다.

$$B1 = f(\text{이전 층 노드 값들의 선형 결합 합계})$$
$$= f(w_1*X_1 + w_2*X_2 + w_3*X_3)$$
$$= f(0.9)$$

렐루 함수 f(x)의 입력값이 0.9이고, 이는 0보다 큰 입력값이므로 출력값은 입력값을 그대로 토해냅니다.

$$B1 = 0.9$$

이 경우의 해석은 다음과 같습니다.

신경 노드에 도입된 비선형 함수가 입력값인 선형 결합값을 그대로 출력하는 경우는 On 스위치 역할을 해서 해당 노드를 켜는 역할을 하고, 출력값은 다음 층의 신경 노드에 전달할 입력값으로 사용됩니다.
반면에 비선형 함숫값이 0이 나온 경우에는 Off 스위치처럼 해당 노드의 결과를 신경망에서 제거하는 역할을 합니다.

해석을 보면, 신경망에 쓰인 비선형 함수는 해당 신경 노드를 신경망 전체 작용에 계속 참여하도록 활성화하거나, 혹은 더 이상 참여하지 못하도록 스위치를 꺼버립니다. 이러한 비선형 함수를 **활성화 함수**라고 부릅니다. 즉, 활성화 함수는 신경망 전반에 걸쳐서 노드별로 일종의 '판단'을 내리고 있는 셈입니다.

위에서는 은닉층의 첫 번째 노드 값 B1을 계산했습니다. 유사한 과정을 거쳐 마지막에 위치한 출력층의 신경 노드 값 R1의 활성화 함수 결괏값으로 1이 나오면 Yes로 판별하고, 0이 나오면 No로 판별하면 됩니다. 만약 이진 분류 문제가 아니라 다중 분류 문제인 경우에는 출력층의 활성화 함수만 다중 레이블값을 출력하는 비선형 함수(예: Softmax 함수)로 바꿔주면 됩니다.

B.3 손실 함수와 오차 역전파법

가중치와 활성화 함수를 설명했지만, 신경망 모델이 어떻게 학습한다는 것인지는 아직 설명하지 않았습니다. 이번에는 손실 함수와 오차 역전파법을 통해서 신경망 모델의 학습 과정을 알아봅니다. 지금까지 설명한 것처럼 신경망 모델은 입력층에 데이터를 입력하면 하나 혹은 다수의 은닉층을 거쳐서 최종적으로 출력층까지 순차적으로 계산을 진행합니다. 결과적으로 분류 모델인 경우 출력층에서 분류 레이블(예: 0 혹은 1)을 출력하고, 회귀 모델인 경우 출력층에서 연속적인 값 형태를 지닌 숫자를 출력합니다. 그리고 출력 결과를 이용해서 성능 평가를 합니다.

머신러닝 모델의 평가 과정을 요약 설명하면 다음과 같습니다. 머신러닝 모델은 데이터를 학습 데이터세트와 테스트 데이터세트로 구분합니다. 입력 변수와 출력 변수가 함께 들어있는 학습 데이터세트에서 머신러닝 모델을 학습시킵니다. 그리고 테스트 데이터세트의 입력 변수를 학습이 완료된 머신러닝 모델에 투입하여 실행합니다.

머신러닝 모델은 결과적으로 테스트 데이터세트 입력 변수에 따른 출력 변숫값을 출력합니다. 이렇게 예측해낸 출력 변숫값과 머신러닝 모델에 투입하지 않았던 테스트 데이터세트의 원본 출력 변숫값을 비교합니다. 분류 평가 지표 중에서 예를 들어 **정확도**는 예측한 출력 변수 레이블과 실제 출력 변수 레이블이 일치하는 비율을 측정합니다. 참고로 분류 오류율(Misclassification rate)은 1에서 정확도를 뺀 값으로 구합니다.

$$\text{Misclassifcation rate} = 1 - \text{정확도}$$

회귀 평가 지표 중에서 예를 들어 **MSE**(Mean Squared Error)는 예측한 출력 변숫값과 실제 출력 변숫값 사이의 차이를 제곱하고, 이를 평균하여 구합니다. 분류 오류율과 MSE는 머신러닝 모델의 예측값이 실제값과 얼마나 차이가 나는지 측정한다는 공통점이 있습니다. 이 값들이 머신러닝 모델 성능의 손실(Loss) 혹은 비용(Cost)을 나타낸다고 해석할 수 있습니다. 그래서 평가 지표를 **손실 함수**(Loss function) 혹은 **비용 함수**(Cost function)라고 칭합니다.

손실 함수는 머신러닝 모델이 얼마나 부정확한 예측을 하는지 평가하는 함수이며, 이를 통해 머신러닝 모델 성능을 평가합니다. 부정확한 예측이 줄어들수록 손실 함숫값이 작아집니다. 따라서 손실 함숫값이 작아지도록 신경망 모델에 산재해 있는 가중치들을 조정해야 합니다. 이처럼 손실 함숫값이 작아지도록 가중치를 조정하는 과정을 **학습 과정**이라고 칭합니다.

참고로 텐서플로 케라스(tf.keras)에서 사용하는 신경망 모델의 손실 함수는 다음과 같습니다.

분류 혹은 회귀 유형	손실 함수
이진 분류	Binary_crossentropy
단일 레이블 다중 분류	Categorical_crossentropy
다중 레이블 다중 분류	Binary_crossentropy
임의의 값에 대한 회귀	MSE
0과 1 사이 값에 대한 회귀	MSE 또는 Binary_crossentropy

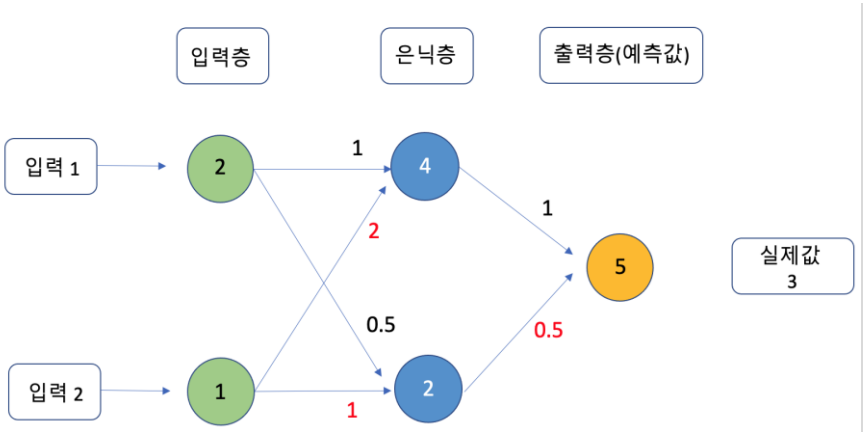
출처: 케라스 창시자에게 배우는 딥러닝, 프랑소와 숄레 저, 길벗, 2018, P 165

이로써 손실 함수가 왜 필요한지를 알게 되었습니다. 손실 함숫값을 줄이도록 신경망 모델의 가중치를 조정하는 것이 '학습 과정'입니다. 하지만 여기서 다음과 같은 의문이 꼬리를 물고 떠오릅니다.

그럼 구체적으로 신경망 가중치를 어떻게 조정하지?

이 의문에 대한 대답이 바로 오차 역전파법입니다. 머신러닝과 딥러닝의 역사에는 몇 차례 길고 긴 침체기가 있었습니다. 그 원인 중 하나는 다중 신경망 모델, 즉 딥러닝 모델에서 손실 함숫값이 최소가 되는 가중치값을 조정하는 방법을 몰라서였습니다. 오차 역전파법의 등장으로 이 문제를 해결하고, 딥러닝 모델을 학습시킬 수 있게 되었습니다. 오차 역전파법이 어떻게 이러한 과정을 수행하는지 구체적으로 살펴봅시다.

딥러닝 수학 이론서 "이토록 쉬운 딥러닝을 위한 기초 수학 with 파이썬(마스이 도시카즈 저, 2019, 루비페이퍼, p.205~209)"에서는 오차 역전파법을 통한 가중치 수정 방법을 매우 쉽게 설명하고 있습니다. 설명의 편의를 위해 해당 책의 예시를 조금 개선하겠습니다. 여기서는 입력층 노드 2개, 은닉층 노드 2개, 출력층 노드 1개의 신경망을 예로 듭니다. 단, 계산의 편의를 위해 활성화 함수는 신경망에 도입하지 않았습니다.



[그림 B-4] 오차 역전파법에 의한 가중치 조정 1

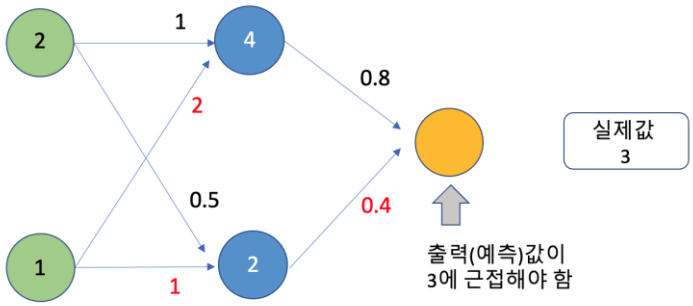
입력층 노드1의 값은 2, 노드 2의 값은 1로 각기 주어집니다. 위 신경망의 노드들을 잇는 화살표

에는 초기 가중치가 실려 있습니다. 이 상태는 신경망 모델이 최종적으로 예측값 5를 산출합니다. 실제 데이터에서 주어진 최종 출력값은 3이라고 가정하겠습니다. 이처럼 신경망 모델 중 다중 레이블 분류 모델, 혹은 회귀 모델은 최종 출력값으로 0과 1이 아닌 숫자값을 가질 수 있습니다. 여기서는 예측값(5)과 실제값(3)이 2만큼의 오차가 있습니다. 오차를 줄이려면 예측값이 3에 좀 더 가까운 값이 나와야 합니다.

참고로 오차 역전파법에서 오차를 줄인다는 의미는 오차에 기반하여 작성된 손실 함숫값을 최소화한다는 의미입니다. 이 예에서는 최대한 이해를 쉽게 돕기 위해 손실 함수를 도입하지 않고 직접적으로 '오차를 줄이는' 과정을 보여주고 있습니다만, 신경망에서는 실제로 모든 노드에 손실 함수를 적용한 상태에서의 오차를 줄입니다.

이를 위해서 다음과 같이 은닉층과 출력층 사이에 있는 가중치 2개의 수치를 조정합니다.

1단계: 출력층에 있는 예측값과 실제값의 오차를 줄이기 위해 은닉층과 출력층 사이에 있는 가중치를 조정합니다.



[그림 B-5] 오차 역전파법에 의한 가중치 조정 2

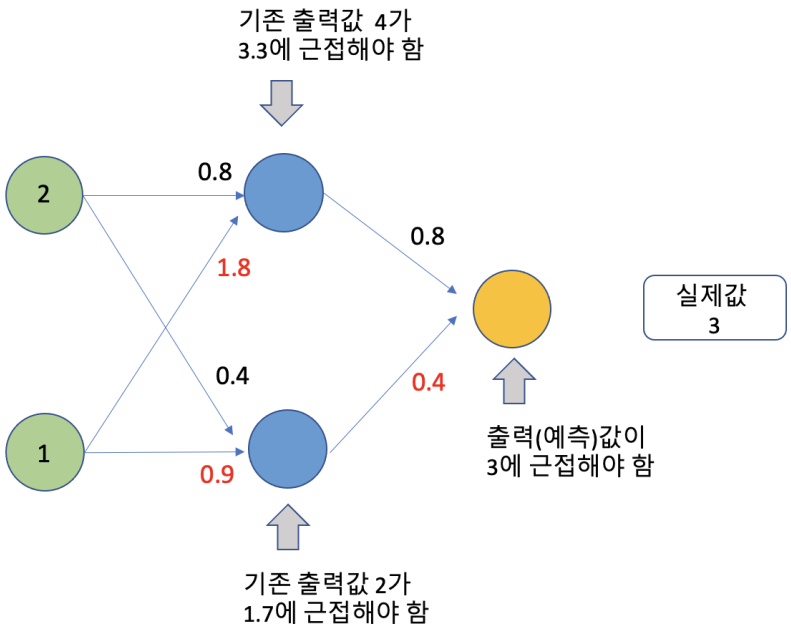
은닉층과 출력층 사이 가중치를 1에서 0.8로, 0.5에서 0.4로 내렸습니다. 이렇게 가중치를 조정하고 나서 은닉층의 출력값 4와 2를 활용해 출력층 노드의 변경될 예측값을 계산해봅니다. 0.8*4 + 0.4*2 = 0.32 + 0.8 = 4가 나옵니다. 조정된 예측값(4)과 실제값(3) 사이의 오차가 1로 줄어들었습니다.

가중치 조정 전보다는 줄어든 오차이지만, 여전히 오차가 있습니다. 따라서 오차를 더 줄이도록 다시 입력층과 은닉층 사이의 가중치를 조정합니다. 이처럼 오차가 신경망의 이전 층, 즉 역방향으로 전달되어 가중치를 연속적으로 조정하기 때문에 **오차 역전파법**이라는 이름이 붙었습니다.

2단계: 1단계 조치 후에도 아직 남은 오차가 있으면 다시 이전 층의 가중치를 조정합니다.

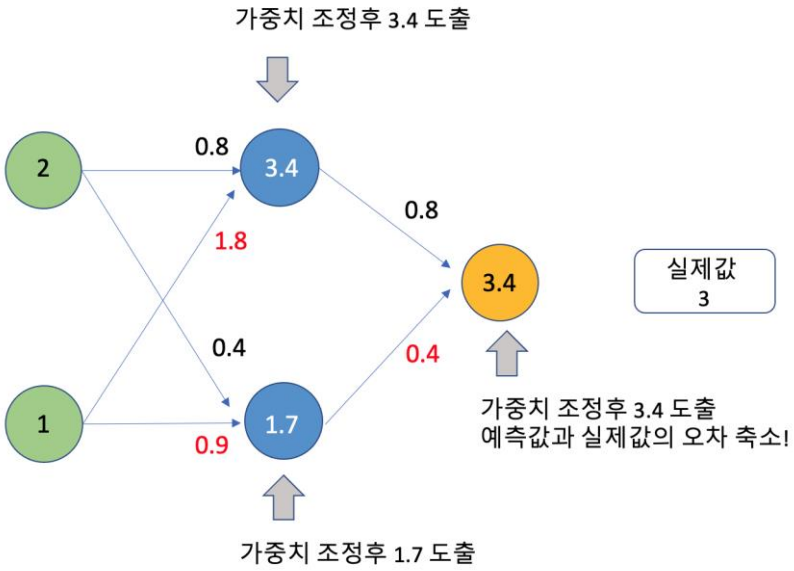
떠넘겨진 오차 1을 추가로 줄이기 위해서 이번에는 은닉층의 두 값, 4와 2가 어느 정도 감소해야 합니다. 예를 들어 히든 노드2의 출력값 2에서 0.3을 빼서 1.7을 만들고, 히든 노드1의 출력값 4

에서는 0.3의 대략 두 배를 조금 넘는 0.7을 빼서 3.3을 만드는 것을 목표로 삼겠습니다. 이를 위해 입력층과 은닉층 사이의 4개 화살표에 있는 가중치를 조금씩 변경해봅니다. 가중치 조정 결과는 다음과 같습니다.



[그림 B-6] 오차 역전파법에 의한 가중치 조정 3

1단계와 2단계 조치를 모두 취하고 나니 모형에 있던 모든 가중치들이 조금씩 수정되었습니다. 이제 수정된 가중치를 가지고 입력 노드의 원래 입력값 2와 1을 활용해 순서대로 신경망 모델을 돌려보겠습니다.



[그림 B-7] 오차 역전파법에 의한 가중치 조정 4

결과적으로 출력층에서 생성된 최종 예측값은 3.4가 나왔습니다. 오류 역전파법에 의한 가중치 조

정 전 결과인 5보다는 훨씬 더 실제값 3에 가까워졌습니다. 이 과정에서 신경망 전역에 산재한 초기 가중치들이 더 작은 오차를 산출하는 가중치로 바뀌었습니다.

3단계: 예측값이 실제값과 충분히 수렴할 때까지 1, 2단계를 반복합니다.

앞에서도 언급했지만, 이번 예에서는 계산의 편의를 위해 신경망 노드의 활성화 함수와 손실 함수를 일부러 포함하지 않은 채 오차 역전파법을 설명하였습니다. 그러나 활성화 함수와 손실 함수를 포함해도 오차 역전파법은 이와 비슷한 원리로 진행됩니다.

B.4 경사 하강법

앞에서 오차를 줄이는 과정을 살펴보았는데, 실제 신경망에서는 오차 자체를 줄이는 것이 아니라 오차에 기반한 손실 함수의 값을 줄입니다. 전형적인 신경망 모델은 입력층과 은닉층이 각각 품고 있는 신경 노드의 개수가 수십 개에서 수백 개를 훌쩍 넘습니다. 게다가 다층 신경망, 즉 딥러닝 모델은 은닉층의 개수도 여러 개를 갖습니다.

제가 수업에서 배운 상용 통계 패키지 SAS Enterprise Miner의 신경망 모델은 은닉층의 신경 노드 개수의 기본값(default)이 100개로 설정돼 있습니다. 만약 은닉층을 3개를 설정하면 은닉층 전체의 신경 노드 개수만 300개(=100*3)가 됩니다. 이 은닉층 노드 300개와 최종 출력층 노드 1개 모두에 손실 함수를 적용해서 손실 함숫값을 최소화하는 방법으로 가중치를 조정해야 합니다.

그런데 이게 끝이 아닙니다. 최근의 딥러닝 모델은 은닉층의 개수가 수십, 수백 개는 기본입니다. 은닉층이 200개, 출력층이 1개인 경우를 가정해보면 총 20,001개(200*100 + 1) 신경 노드의 손실 함수 각각의 최솟값을 구해야 합니다. 손실 함수는 이차함수 형태를 지니거나(MSE의 경우), 로그 함수와 확률 변수의 곱 형태(binary crossentropy 혹은 categorical crossentropy) 등으로 나타나기 때문에 하나하나의 최솟값을 구하는 과정에서 상당한 컴퓨팅 성능과 시간이 필요합니다. 이런 이유로 예전에는 딥러닝 모델을 실행하면 수일 혹은 수개월씩 걸리는 사례가 다반사였습니다.

특히 딥러닝 모델은 컴퓨터에 과부하를 많이 발생시키는 경향이 있어서 이를 방지하기 위해 하드웨어와 소프트웨어 면에서 많은 개선이 있었습니다. 멀티 CPU 및 GPU 사용, 클라우드 이용, 텐서플로 도입 등이 그것입니다. 그리고 수학적으로는 **경사 하강법**을 도입하여 수많은 손실 함수의 최솟값을 찾는 과정을 간편하게 만들었습니다.

여기서도 논의를 간단하게 하기 위해서 손실 함수가 MSE일 때의 경사 하강법 작동 원리를 설명하겠습니다. 손실 함수가 다른 형태의 함수일 때도 경사 하강법의 작동 원리는 동일합니다. 평균 제곱오차(MSE)의 수식은 다음과 같습니다.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$$

n = 데이터 수

Y_i = 실제 데이터 관측값

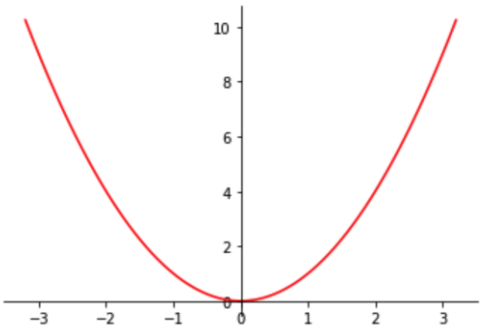
$\hat{Y}$ = 해당 신경 노드에 의해 출력된 예측값으로서,
가중치와 입력 변수의 선형 결합에 비선형 활성화 함수를 취한 값

수업 에피소드!
MSBA 수업은 다양한 전공의 학부생을 받아들이고, 또 미국 직장인들이 필요에 따라서 수시로 파트타임으로 등록해서 듣기 때문에 수학과 담을 쌓고 있는 학생들이 의외로 많습니다. 때문에 수학을 조금만 깊게 들어가면 반 전체가 멍해질 때가 종종 있었습니다. 저도 그중 하나였습니다. 그런데 총 3번의 연구 조교 머신러닝 프로젝트에 추가로 3번의 학생팀 머신러닝 프로젝트를 수행하면서 머신러닝, 딥러닝 모델에 쓰인 수학을 증명하라고 요구받은 적은 한 번도 없었습니다. 그래서 여러분도 수학에 대한 부담 없이 머신러닝, 딥러닝을 공부해도 된다고 자신 있게 말할 수 있습니다.
머신러닝, 딥러닝을 뒷받침하는 수학은 머신러닝과 딥러닝 모델을 자유자재로 돌릴 수 있게 되고, 입력 데이터에 대한 데이터 처리 및 탐색적 자료 분석을 제대로 할 수 있게 된 다음에, 추가로 찾아봐도 늦지 않습니다. 오히려 기초적인 통계 개념을 수학 이론보다 먼저 익히는 것이 좋습니다.

신경망 모델의 첫 번째 은닉층 노드에서의 예측값 $\hat{Y}$을 얻으려면 $f(w_1*X_1 + w_2*X_2 + w_3*X_3)$를 계산해야 합니다. 여기서 f는 활성화 함수이고 w 시리즈는 가중치이며 X_1, X_2, X_3는 입력층으로부터 넘겨받은 입력값입니다. 손실 함수(여기서는 MSE) 수식에서 $\hat{Y}$을 $f(w_1*X_1 + w_2*X_2 + w_3*X_3)$로 대체하면 손실 함숫값을 구할 수 있습니다.

이를 그래프로 설명하기 위해서 입력층의 입력값이 하나만 있고, 가중치도 하나만 있는 극단적으로 단순한 모형을 생각해보겠습니다. 그러면 예측값 $\hat{Y}$은 $f(w_1*X_1)$만 계산하면 됩니다. 이를 손실 함수(MSE)에 대입하면 X_1 입력 변수의 이차함수가 됩니다. 그런데 우리의 관심사는 가중치 w_1입니다. 입력 변수 X_1의 값은 이미 결정돼 있으며, 모델에서 변경할 수 없습니다. 모델에서 조정할 수 있는 것은 가중치뿐입니다. 때문에 손실 함수는 가중치 w_1의 이차함수로도 볼 수 있습니다.

여기서 여러분이 중학교 때 배운 이차함수 지식을 잠시만 빌려오면, 손실 함수에 제곱항이 있으므로 제곱을 풀면 w_1^2항 앞에 놓일 계수 부호는 플러스(+)가 됩니다. 이 경우 손실 함수는 다음과 같은 통상적인 이차함수의 궤적을 가집니다. 아래 그래프는 간단한 이차함수 형태인 손실 함수 w_1^2의 함수를 그린 것입니다. 가로축은 가중치 w_1을 의미하고, 세로축은 손실 함수의 값을 나타냅니다.

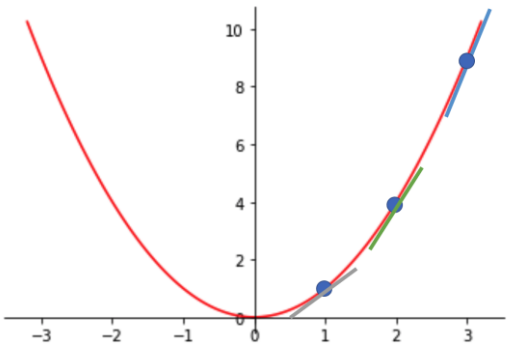


[그림 B-8] 손실 함수 그래프

위에서는 그래프를 간편하게 그렸지만, 비활성화 함수 f의 형태에 따라 실제 손실 함수를 전개하면 이차항인 w_1^2항만 있는 것이 아니라 일차항인 w_1항, 그리고 상수항까지 생깁니다. 따라서 일반적으로는 w_1값이 0인 지점에서 손실 함수가 최솟값을 갖는다고 보장할 수 없습니다. 다만 최솟값을 갖는 곳에서 손실 함수의 기울기가 0이 되는 것은 확실합니다.

신경망 모델에서는 최초에 랜덤하게 가중치가 부여되고, 이 상태에서 입력 변수 데이터를 모델에 입력하여 순차적 계산을 통해 각 신경 노드의 손실 함수를 계산합니다. 위의 단순화한 손실 함수 그래프는 예시로서 은닉층의 첫 번째 신경 노드의 손실 함수 그래프를 나타내고 있습니다만, 실은 모든 신경 노드에서의 손실 함수도 유사한 모양을 갖습니다.

신경망 모델에서 최초에 랜덤하게 가중치가 부여된다는 것은, 예를 들어 w_1이 최초에 랜덤하게 3의 값이 주어지면 손실 함숫값이 9가 된다는 것입니다. 이를 줄이기 위해서 컴퓨터가 일일이 모든 가중치의 가능한 경우의 수를 다 넣어보고, 손실 함수가 최소로 되는 가중치 w_1을 찾아야 합니다. 여기서 수학의 미분을 적용하면 좀 더 빠르게 손실 함수가 최솟값을 갖는 가중치 w_1을 찾을 수 있습니다. 즉, 손실 함수의 기울기를 이용하는 것입니다.



[그림 B-9] 손실 함수 기울기 변화

가중치 w_1 = 3인 상태에서 손실 함수는 양의 기울기를 갖습니다. 이때 가중치 w_1을 줄이면 손실 함숫값이 떨어지게 됩니다. w_1 = 2로 가중치를 줄이면 여전히 손실 함수는 양의 기울기를 가지면서 기울기가 약간 완만해집니다. w_1 = 1로 가중치를 줄이면 더욱 완만해진 양의 기울기를 갖습니다. w_1=0으로 놓으면 기울기가 0이 되면서 손실 함수가 최솟값이 됩니다. 만약 w_1=-1값을 놓으면 기울기가 음이 되면서 가중치 w_1을 조금 더 증가시켜야 손실 함숫값을 줄일 수 있게 됩니다.

이렇듯 손실 함수의 기울기를 구해서 기울기가 플러스 값이면 가중치를 줄이고, 기울기가 마이너스 값이면 가중치를 늘립니다. 그러다가 기울기가 0인 지점에서 손실 함수는 최솟값을 갖게 되고, 최종 가중치 값을 해당 지점에서의 가중치 값으로 확정 짓습니다. 손실 함수의 곡선을 따라 기울기가 내려가는 기법이기 때문에 **경사 하강법**이라는 이름이 붙었습니다.

지금까지 설명한 신경망 모델을 정리하면 다음과 같습니다.

1. 입력층, 은닉층, 출력층으로 구성된 단일 신경망 혹은 다중 신경망 모델을 구성합니다. 신경망은 1개의 입력층, N개의 은닉층, 1개의 출력층으로 구성됩니다.

2. 신경망 모형에 있는 모든 **가중치**는 초기에 랜덤하게 주어집니다.
3. 각 신경 노드는 이전 층의 신경망 노드로부터 출발한 입력값(X)을 가지고, 이를 가중치(W)와 선형 결합한 결괏값을 받습니다.
4. 이 선형 결합값에 비선형 함수인 **활성화 함수**를 적용하여 출력값을 산출합니다. 출력된 값은 다음 층의 신경 노드에게 입력값으로 사용됩니다.
5. 다음 층에서도 3과 4의 과정을 반복해서 최종적으로 출력층에서 최종 예측값을 출력합니다.
6. 최종 출력층 신경 노드에서 산출한 예측값과 최종 출력 관측값의 차이(오차)에 기반한 **손실 함수** 값을 계산합니다. 이 손실 함수를 최소화하는 방향으로 이전 층(**N번째 은닉층**)과 출력층 사이에 있는 가중치를 조정합니다. 이때 **경사 하강법**을 사용해 연산 시간을 줄이면서 가중치를 조정합니다. **오차 역전법**의 시작 단계입니다. 손실 함수를 최소화하는 값이 출력층의 값을 조정하고 가중치도 조정되었으니, 이를 거꾸로 연산하면 바로 이전 층(**N번째 은닉층**) 신경 노드의 바람직한 출력값이 계산됩니다. 앞에서부터 순차적으로 계산된 이전 층(**N번째 은닉층**)의 출력값과 역방향으로 계산된 바람직한 출력값을 비교하면 오차가 발생합니다.
7. 6단계 과정에서 마지막에 계산한 오차는 N번째 은닉층에서의 오차입니다. 이를 가지고 N번째 은닉층과 N-1번째 은닉층에서 6단계 절차를 반복합니다. 그러면 이 두 은닉층 사이의 모든 가중치가 조정되며, N-1번째 은닉층의 입력 노드들에서 앞에서부터 순차적으로 계산된 원래 출력값과 역방향으로 계산된 바람직한 출력값 사이의 오차가 발생합니다.
8. 이런 과정을 거쳐 최종적으로 첫 번째 입력층과 첫 번째 은닉층 사이의 가중치까지 조정합니다. 오차 역전파법에 의해서 신경망에 있는 모든 가중치의 조정이 이루어집니다. 이를 **학습이 일어났다**고 말합니다.

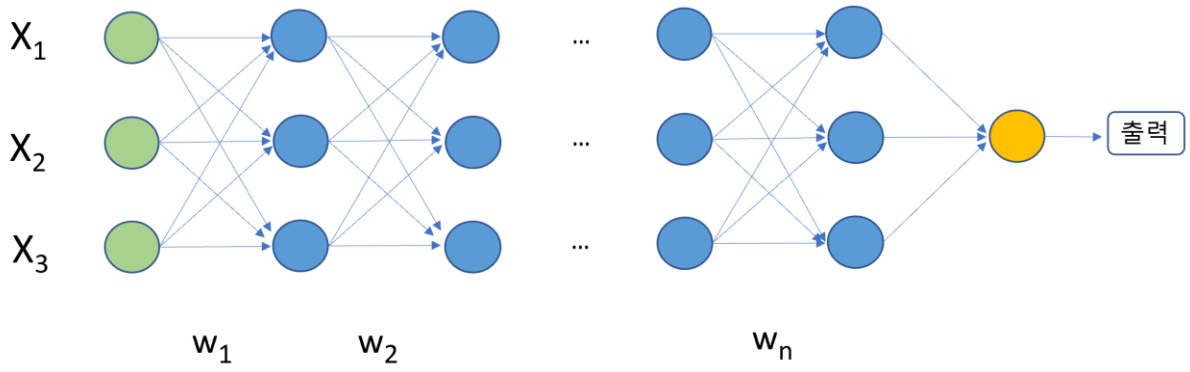
B.5 배치 정규화

주성분 분석은 입력 데이터의 분산을 가장 잘 설명하는 방향의 주성분을 하나씩 추출합니다. 그런데 입력 변수 스케일이 다르면 큰 스케일을 가진 변수가 작은 스케일의 변수를 압도해서 왜곡이 생길 수 있습니다. 때문에 주성분 분석을 하기 전에 **데이터 스케일**(측정 단위)에 대한 **표준화**를 먼저 실행했었습니다. 또한, 군집 분석에서도 데이터 스케일 표준화를 실행한 바 있습니다. 군집 분석은 데이터 간의 '거리'에 기반하기 때문입니다.

위의 두 분석 방법에서 쓰인 데이터 스케일 표준화는 앞으로 설명할 **배치 정규화**의 한 부분을 담당합니다. 신경망, 특히 다중 신경망을 쓰는 딥러닝 모델에서는 매 신경층의 출력값 분포가 이전 신경망에서 들어오는 입력 데이터 분포를 가급적 유지하는 것이 바람직합니다. 매 신경층의 출력값은 다음 층의 입력값이 됩니다. 때문에 매 신경층의 출력값에 대한 데이터 표준화를 수행하는데, 이를 **정규화**(Regularization) 과정이라고 합니다. 아울러 전형적인 신경망 모델에서는 데이터를

하나하나 입력하여 학습하지 않고, **배치(batch)**라고 부르는 데이터 뭉치(collected set)를 사용해서 학습합니다. 그래서 신경망에서 사용하는 정규화를 **배치 정규화(batch regularization)**라고 부릅니다.

신경망에 배치 정규화를 하면 좋은 이유를 알아보겠습니다. 전형적인 딥러닝 모델의 신경망은 아래와 같이 나타낼 수 있습니다. 이 모델의 맨 앞 입력층에 들어오는 원래 입력 데이터 X_1, X_2, X_3가 데이터 스케일 표준화, 즉 정규화된 데이터라고 가정하겠습니다. 입력 변수 X_1, X_2, X_3는 간편하게 입력 변수 매트릭스 X로 표기하겠습니다.



[그림 B-10] 다층 신경망 구조

입력층을 지나서 첫 번째 은닉층에 도달하면 가중치 행렬 w_1을 거쳐 선형 결합 w_1*X가 된다고 앞에서 설명한 바 있습니다. 여기에 비선형 함수인 활성화 함수 f를 적용하면 첫 번째 은닉층 신경 노드의 최종 출력값은 $f(w_1*X)$가 됩니다. 선형 결합된 값을 입력하여 비선형 함수 연산한 결과의 분포는 원래 입력 데이터 분포와 달라집니다.

그런데 두 번째 은닉층에서도 유사한 계산 과정을 거치면 첫 번째 은닉층의 출력값 분포에서 조금 더 멀어집니다. 이런 과정을 n개의 은닉층까지 거치면 맨 앞의 입력층에서 투입된 입력 데이터의 분포와 마지막 은닉층 결괏값의 분포는 매우 달라지게 됩니다.

머신러닝, 딥러닝은 학습 데이터세트를 통해 학습하고 테스트 데이터세트를 통해 실전 성능을 평가합니다. 여기에는 동일한 학습 데이터세트를 써서 학습한다는 전제 조건이 있습니다. 그런데 딥러닝 모델에서 별도의 추가 조치 없이 그냥 모델을 돌리면 위에서 설명한 사유로 인해 은닉층마다 사용하는 입력 데이터의 분포가 조금씩 달라지게 됩니다.

이 말은 은닉층마다 사실상 다른 학습 데이터세트를 쓴다는 의미입니다. 이래서는 테스트 데이터세트를 통한 일반화 성능 평가에서 좋은 결과를 낼 수 없습니다. 그래서 은닉층마다 동일한 학습 데이터세트를 써서 학습하는 것을 보장하기 위해 딥러닝 모델의 매 신경층 출력값의 표준화 조치, 즉 배치 정규화 조치를 도입합니다.

그렇다면 매 신경층에서 데이터 정규화는 구체적으로 어떤 방식으로 이루어질까요? 특정 신경층의 배치 정규화 이전의 결괏값 매트릭스를 X라고 단순하게 표기하겠습니다. X에 담긴 데이터 분포를 원래 입력 변수 분포와 유사하게 되돌리는 조치는 다음과 같습니다. 우선 X에 대해 표준화(Standardization) 조치를 취한 값 Z를 구합니다.

$$Z = (X - U) / S$$

여기서 U는 데이터의 평균, S는 데이터의 표준편차를 의미합니다. 위 수식은 앞에서 소개한 데이터 스케일 표준화의 수식과 동일한데, 여기서는 수식 표기를 간략화하기 위해 Z, X, U, S는 모두 매트릭스 형태로 표기했습니다. 이렇게 표준화한 Z에 추가로 다음과 같은 변형을 가해줍니다.

$$\Gamma Z + B$$

여기서 Γ(감마, Gamma)는 표준화된 Z의 스케일을 재조정해서 해당 신경층 변환 이전의 입력값 스케일로 되돌리는 역할을 합니다. 그리고 절편 B는 이동된(Shift) 거리를 보정합니다. Γ와 B 역시 모두 매트릭스 형태입니다. 이러한 조치를 통해 매 신경망에서 산출되는 값의 분포는 이전 신경망에서 들어오는 입력 데이터의 분포와 유사해집니다.

결국 딥러닝 모델의 전 신경층에서 동일하거나 유사한 분포의 학습 데이터를 사용하게 되어 모델의 일반화 성능이 높아집니다. 이는 테스트 데이터세트를 통한 성능 평가가 향상된다는 의미입니다. 즉, 모든 신경층의 결괏값에 대해서 배치 정규화가 필요합니다.

B.6 합성곱 신경망

수업 에피소드!
제가 수업에서 배운 머신러닝 모델은 수십 가지가 넘습니다. 그런데 합성곱 신경망만은 커리큘럼에 들어가있지 않았습니다. 왜일까요? 졸업 후, 합성곱 신경망에 대해 좀 더 깊이 있게 공부를 하고 나서 그 이유를 알게 됐습니다. 그것은 제가 다닌 MSBA 학과가 비즈니스 분석(Business Analytics)을 주로 하는 학과였기 때문이었습니다.
고객 분석, 매출 분석, 텍스트 분석, 주가 예측 등은 비즈니스 분석의 주요 사례들입니다. 그런데 이미지(사진) 분석은 아직까지는 비즈니스 분석에서 취급하는 주제가 아닙니다. 그럼에도 불구하고 합성곱 신경망은 딥러닝 연구에 있어서 획기적인 장을 열었고 현재는 안면 인식, 자율 주행 등의 분야에서 활발하게 쓰이고 있습니다. 비즈니스 분석에서도 이미지 분석 기술을 활용할 날이 얼마 남지 않았습니다.
일례로 미국의 대표적 온라인 부동산 중개 사이트 Zillow.com에서는 Zestimate라는 집값 예측치를 제공하는데, 각각의 집이 가진 정성적인 측면을 반영하지 못한다는 비난을 받아왔습니다. 그래서 2019년부터 집 내부의 사진들을 합성곱 신경망으로 분석해서 일조량, 내부 인테리어 품질 같은 정성적인 부분을 반영하고 있습니다.

생각하는 기계를 만들겠다는 인류의 염원이 인공지능에 대한 연구를 이끌었으며, 그 일환으로 머신러닝과 딥러닝 기법이 개발되었습니다. 때문에 신경망 모델은 인간 두뇌 신경 세포의 연결망을 모방했다고 말씀드린 바 있습니다. 합성곱 신경망(Convolutional Neural Network, CNN)은 그중에서도 고양이가 이미지를 단계별로 인식하는 방식을 밝혀내서 시각의 작동 원리를 모방하여 이를 인공 신경망에 구현한 것입니다.

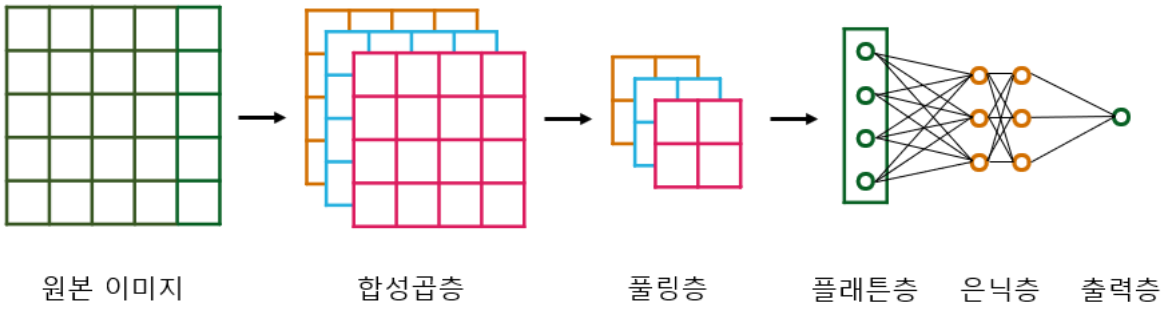
위의 수업 에피소드에서 밝혔듯이 이미지 분석은 아직 비즈니스 분석에 활발하게 도입되지는 않았지만, 딥러닝의 중흥을 이끈 매우 중요한 신경망 모델입니다.

개략적인 작동 원리는 다음과 같습니다. 고양이 혹은 인간은 어떻게 물체 이미지를 인식하여 식별하는가를 살펴보겠습니다. 고양이를 대상으로 한 실험 결과에서 가장 중요한 발견은 각 신경망이 전체 이미지 중에서 특정 위치를 커버하며, 그중에서도 특정 모양에만 반응한다는 것이었습니다.

고양이가 다른 물체를 바라볼 때 하나의 신경 세포가 눈에 들어온 모든 전경을 담당하지 않고, 시각 이미지의 특정한 부분을 담당합니다. 예를 들어 어떤 신경 세포는 수직선 모양의 입력에만 반응하고, 어떤 신경 세포는 수평선 모양의 입력에만 반응하는 식입니다. 따라서 시각 피질을 연구하는 과정에서 자연스럽게 인공 신경망을 층층이 겹쳐서 쌓아야 하는 필요성을 발견했습니다. 시각 인지 과정을 모방하다 보니 자연스럽게 망 구조를 딥러닝 구조로 만들어야 했던 것입니다.

딥러닝 모델에서 여러 신경망을 순서대로 쌓은 후 각각의 신경망에 각기 다른 모양의 시각 인지 기능을 준다는 아이디어가 합성곱 신경망의 핵심 아이디어입니다. 이런 정보 처리 과정을 수차례 거쳐서 마지막 정보 처리 단계에서 해당 이미지에 대한 최종 판단을 내립니다. 글자라면 어떤 글자인지, 동물이라면 고양이인지 개인지, 도로 위 물체라면 사람인지 쓰레기봉투인지를 판단합니다. 이렇듯 서체 인식, 안면 인식, 자율 주행 기술 등이 합성곱 이미지 신경망에 기초합니다.

이제 합성곱 신경망의 주요 구성 요소에 대해 간략하게 알아보겠습니다. 다음은 합성곱 신경망의 주요 구성 요소를 그림으로 나타낸 것입니다.



[그림 B-11] 합성곱 신경망 구조

분석 대상 이미지는 설명을 단순화하기 위해서 5*5픽셀의 흑백 이미지를 예로 삼겠습니다. 참고로 실제 분석에서는 256*256픽셀 혹은 HD, 4K 등의 초고해상도 픽셀의 컬러 이미지를 분석합니다. 컬러 이미지는 RGB(빨강, 초록, 파랑) 3원색을 각각 커널로 사용하면 됩니다. 위 그림에서 합성곱 결과가 3개의 매트릭스 형태로 도식화된 것은 그런 이유 때문입니다.

각 픽셀에는 아래와 같이 명암 정보가 담겨 있다고 설정하겠습니다. 숫자가 클수록 어둡고, 숫자가 작을수록 밝은 명암입니다.

0	1	0	0	1
1	0	1	0	2
0	1	0	0	0
0	0	2	0	1
1	0	1	0	0

[그림 B-12] 숫자로 표현된 원본 이미지

자연의 실제 이미지가 위의 예처럼 5*5픽셀로만 보이면 고등 생물의 시각 피질은 거의 발달하지 않았을 것입니다. 처리해야 할 정보량이 몇 개 없기 때문입니다. 그러나 자연의 이미지가 4K 이미지로 우리에게 주어진다면 3840*2160픽셀이 주어집니다. 우리의 뇌는 매 순간 이러한 고화질의 시각 이미지를 처리하고 있습니다. TV나 휴대폰 디스플레이를 기준으로 보면, 우리의 시신경이 연속적인 동작의 이미지 묶음, 즉 동영상을 인식하기 위해서는 초당 수십 번의 이미지 분석을 해야 합니다.

이처럼 엄청난 양의 데이터를 매우 짧은 시간 내에 처리해야 하므로 우리의 시신경은 입력받은 데이터 중에 처리해야 할 데이터는 줄이고, 중요한 정보는 보전하는 방식으로 진화해 왔습니다. 인공 신경망에서 이 일을 해내는 것이 **합성곱층**(convolution layer)이며, **커널**(kernel) 혹은 **필터**(filter)라고 합니다. 이 예에서는 2*2 필터를 다음과 같이 설정합니다.

1	0
0	1

[그림 B-13] 2*2 필터

위 필터는 대각선 방향의 입력값만 살리는 기능을 합니다. 필터의 가중치를 바꾸면 가로선 혹은 세로선 등으로 다른 방향의 이미지 정보를 받아들일 수 있습니다. 원본 입력 데이터의 처음 2*2 픽셀에 필터를 적용해보겠습니다.

0*1	1*0	0	0	1
1*0	0*1	1	0	2
0	1	0	0	0
0	0	2	0	1
1	0	1	0	0

0 = 0*1 + 1*0 + 1* 0 + 0*1

[그림 B-14] 처음 2*2픽셀에 필터를 적용한 결과

원본 입력 데이터에서 두 번째 2*2픽셀에도 동일하게 필터를 적용합니다.

0	1*1	0*0	0	1
1	0*0	1*1	0	2
0	1	0	0	0
0	0	2	0	1
1	0	1	0	0

→ | 0 | 2 | = 1*1 + 1*0 + 0*0 + 1* 1

[그림 B-15] 두 번째 2*2픽셀에 필터를 적용한 결과

이 과정을 반복하여 원본 입력 이미지의 마지막 2*2픽셀까지 필터를 적용하면 최종적으로 4*4 행렬이 도출됩니다.

0	1	0	0	1
1	0	1	0	2
0	1	0	0	0
0	0	2	0*1	1*0
1	0	1	0*0	0*1

→

0	2	0	2
2	0	1	0
0	3	0	1
0	1	2	0

= 0*1 + 1*0 + 0*0 + 0*1

[그림 B-16] 마지막 2*2픽셀에 필터를 적용한 결과

5*5 원본 입력 데이터 이미지를 4*4 출력 데이터 행렬로 바꾸었습니다. 대각선 모양의 음영을 찾아내는 필터에 의해 원본 이미지에서 대각선 모양의 이미지 정보를 담은 출력층을 만들었습니다. 위의 작업 결과가 하나의 합성곱층을 통해 이루어집니다.

이러한 합성곱층을 여러 개 쌓으면 처음에는 직선이나 곡선 등의 저차원 이미지 특성을 뽑아내지만, 뒤로 갈수록 저차원 이미지 특성을 통해 전체 이미지를 해석하고 판별하는 데 도움이 되는 고차원 이미지 특성을 뽑습니다.

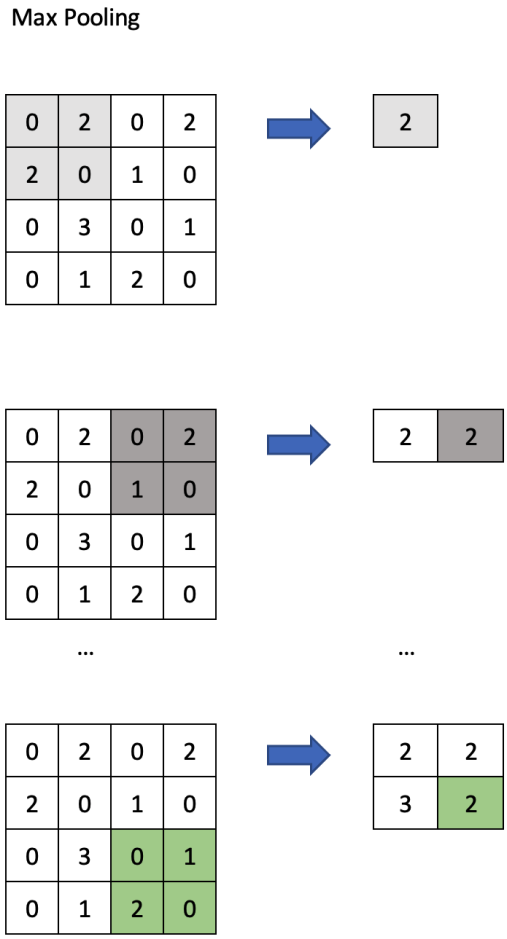
즉, 합성곱층의 핵심 작용은 입력 이미지로부터 저차원적인 특성부터 고차원적인 특성을 순차적으로 뽑아내서 처리해야 할 정보량을 줄이는 것입니다.

[그림 B-17] 풀링층(Pooling Layer)

합성곱층에서는 필터를 통해 이미지의 특성을 뽑아냈습니다. 이렇게 뽑아낸 특성은 위의 예에서처럼 4*4 행렬을 이룹니다. 이를 더 작은 저차원의 행렬로 차원 축소를 행하는 것을 풀링(Pooling)

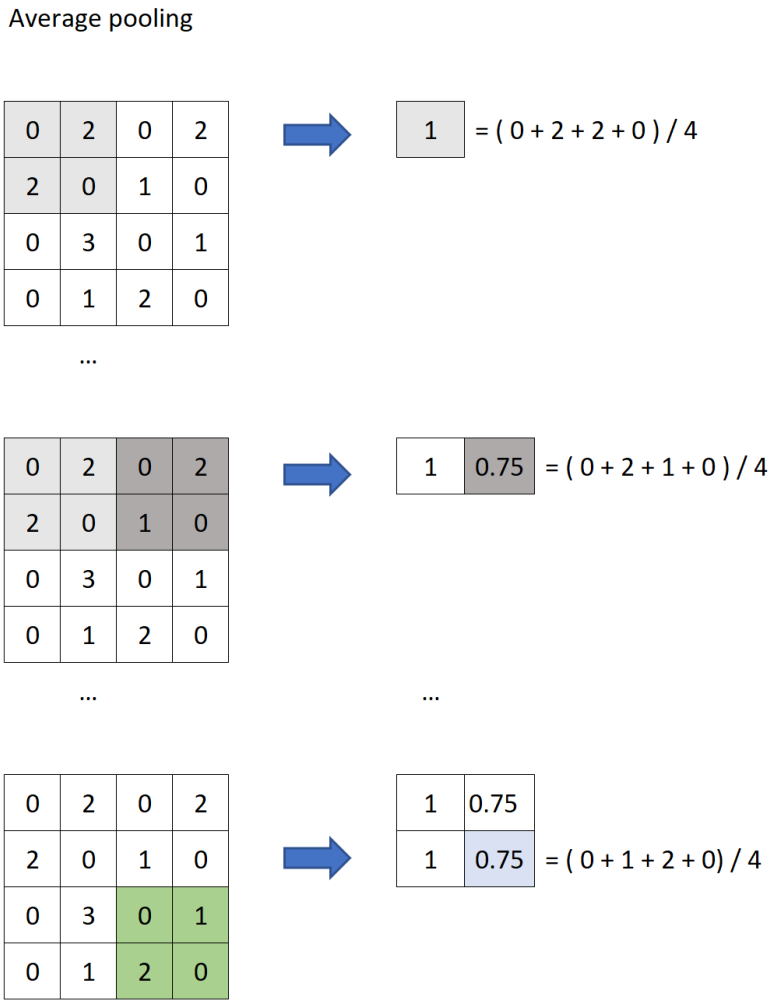
이라고 합니다.

차원 축소는 컴퓨터의 연산 능력을 아끼는 장점이 있습니다. 위의 합성곱층 결과물에 2*2 풀링을 시도해보겠습니다. 풀링에는 최댓값 풀링(Max Pooling)과 평균 풀링(Average Pooling) 방법이 있습니다. 우선 최댓값 풀링 계산을 진행합니다.



[그림 B-18] 최댓값 풀링 적용 결과

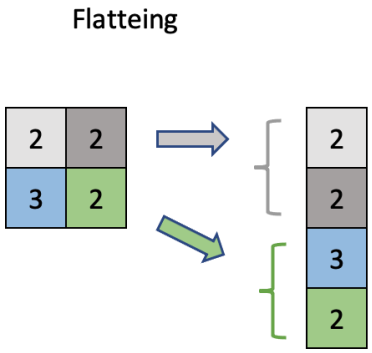
최댓값 풀링 결과, 합성곱의 결과인 4*4차원보다 더 작은 2*2차원의 결괏값을 얻었습니다. 평균 풀링 계산 과정은 다음과 같습니다. 결과는 역시 2*2차원이나, 최댓값 풀링 결과와 값이 다릅니다.



[그림 B-19] 평균 풀링 적용 결과

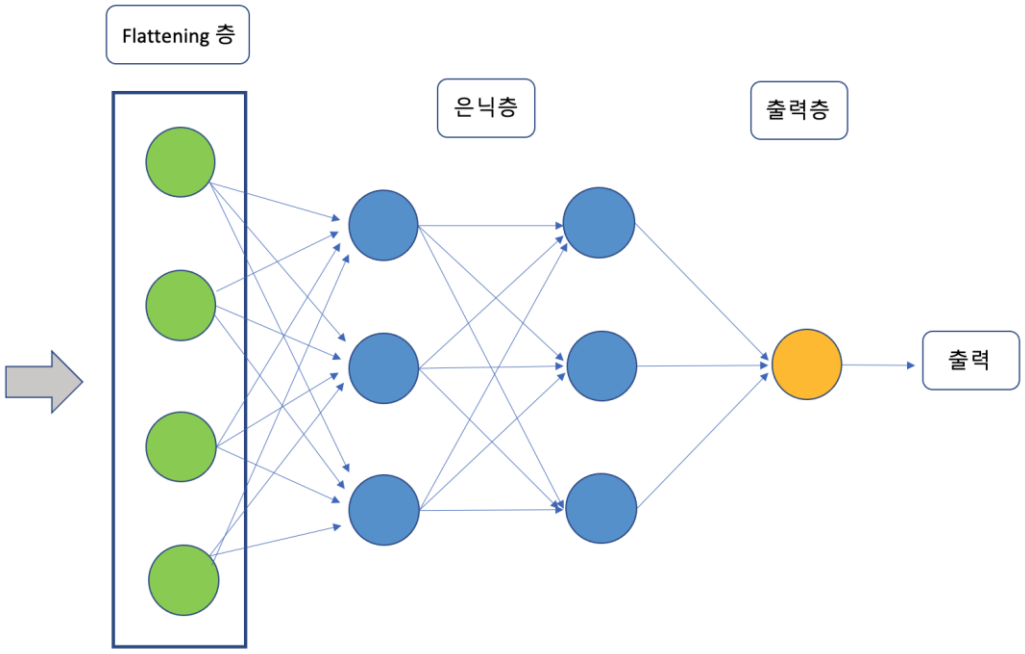
이상으로 풀링층에 의한 차원 축소 과정을 살펴보았습니다. 풀링층은 합성곱층 바로 다음에 오는 것이 좋습니다. 그리고 처리해야 할 이미지가 크면 이런 합성곱층과 풀링층의 쌍을 여러 번 쌓을 수 있습니다. 즉, 첫 번째 합성곱과 풀링층 쌍, 두 번째 합성곱과 풀링층 쌍, ..., n번째 합성곱과 풀링층 쌍, 이런 식으로 신경망을 순차적으로 구성할 수 있습니다.

원래 5*5차원이었던 입력 데이터를 합성곱층에 의해 4*4차원으로 줄였고, 이를 다시 풀링층에 의해서 2*2차원으로 줄였습니다. 이를 일반적인 다층 신경망에 입력하기 위해서는 다차원 행렬 형식의 데이터를 하나의 열만 가진 배열, 즉 열벡터로 바꿔줘야 합니다. 이러한 기능을 하는 층을 플래튼(Flattening)층이라고 합니다. 최댓값 풀링 결과 행렬을 플래튼층을 통해 열벡터로 바꿔보겠습니다.



[그림 B-20] 플래튼층 적용 결과

플래튼층에 의해 열벡터로 변한 이미지 데이터를 일반적인 다층 신경망에 연결하면 학습을 진행하고, 최종적으로 출력층에서 이미지 분류가 이루어집니다. 주어진 숫자 이미지를 0에서 9까지 중에 이미지가 어디에 속하는지 구분하고, 개와 고양이 사진을 보고 개인지 고양이인지를 구분합니다.



[그림 B-21] 다층 신경망에서 플래튼층부터 최종 출력층까지의 구조

이상으로 합성곱 신경망의 원리에 대해 살펴봤습니다. 구체적인 데이터를 기반으로 한 합성곱 신경망 실습은 책의 5장을 참고하기 바랍니다.

B.7 순환 신경망

신경망 분석에서 시간 혹은 사건 순서에 대한 고려가 필요한 분야가 있습니다. 예를 들어 주식 시장에서는 내일의 주가를 예측하는 데 어제의 주가가 일주일 전의 주가보다 더 중요합니다. 구글 번역이나 애플의 시리 등은 문장에서 단어가 배열된 순서가 중요합니다. 한 문장에 쓰인 단어

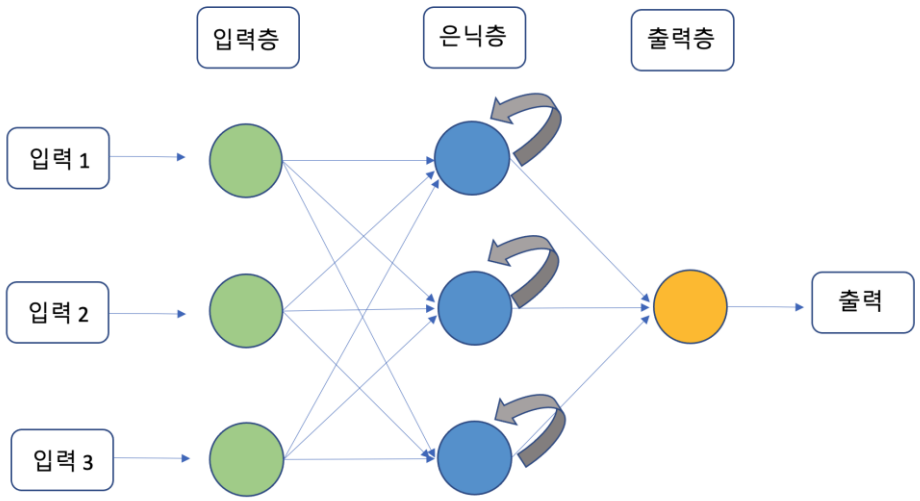
들을 무작위로 입력하는 것보다 순서대로 입력해야 본래 뜻이 이해되고, 번역도 쉬워집니다.

이처럼 시간 순서 혹은 사건 순서를 신경망 분석에 도입하면 매우 유용합니다. 시간 요소를 도입해서 최근의 정보에 더욱 무게를 실어주는 것은 신경망에 '내부 기억(internal memory)'을 도입한다는 것과 같은 의미입니다.

B.7.1 RNN

위에서 언급한 사유로 순환 신경망(Recurrent Neural Network, RNN)이 고안되었습니다. 순환 신경망은 순서가 있는 데이터(sequential data) 혹은 시계열 데이터(time-series data)를 다룹니다. 참고로 지금까지 설명해왔던 통상적인 신경망은 입력층을 거친 데이터가 은닉층, 출력층을 순서대로 통과합니다. 이를 피드 포워드(Feed Forward) 신경망이라 칭합니다. 이러한 피드 포워드 신경망은 현재 시점의 입력만 고려하기 때문에 과거의 정보를 학습에 포함시키기 어렵습니다.

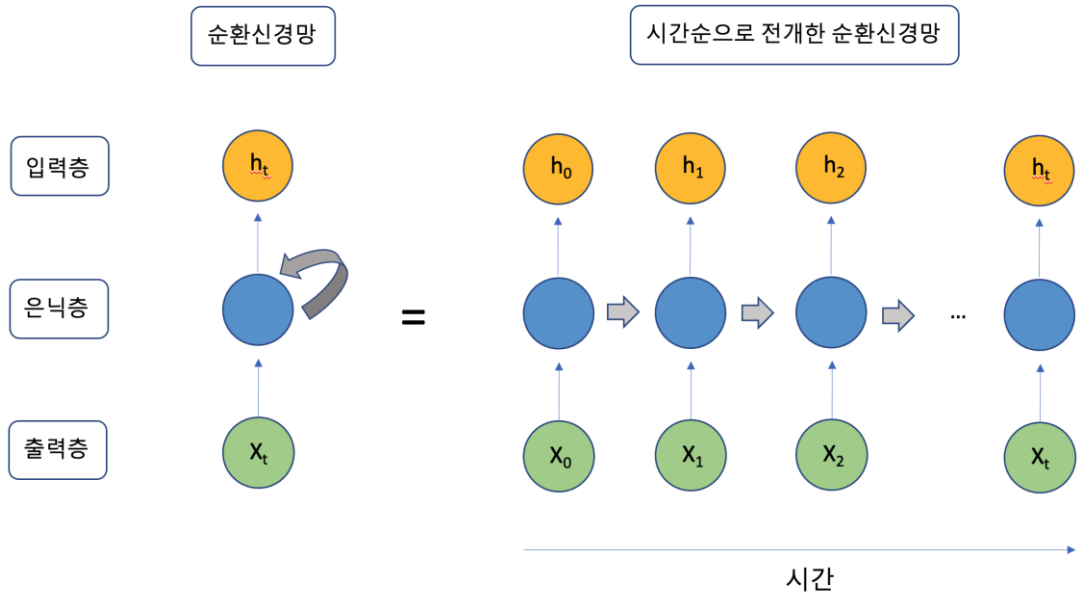
반면에 시간 요소를 고려한 순환 신경망은 과거의 입력으로부터 학습한 정보에 더해서, 현재 시점의 입력까지 고려하여 추가로 학습합니다. 아래 그림을 보면 피드 포워드 신경망과 순환 신경망의 구조의 차이를 알 수 있습니다. 순환 신경망은 현재 시점의 입력뿐만 아니라 과거 정보에 대한 입력도 받습니다.



[그림 B-22] 순환 신경망 구조

순환 신경망은 최근 순서(sequence)의 작업 결과일수록 바로 다음 순서에 나오는 결과물에 더 영향을 미칩니다. '주민'과 '등록'이라는 단어를 순차적으로 들으면 그다음은 '번호'라는 단어가 떠오를 것입니다. 인간이 이렇게 입력 단어의 순서에 민감하게 반응하는 것을 순환 신경망이 모방하였습니다.

그럼 순환 신경망의 주요 구성 요소에 대해 살펴보겠습니다. 아래 그림에서 왼쪽은 시간 요인에 따라 전개하지 않은 순환 신경망이고, 오른쪽은 같은 순환 신경망을 시간 요인을 X축에 따라 전개한 것입니다.



[그림 B-23] 시간순으로 전개한 순환 신경망

이러한 신경망 구조 때문에 순환 신경망은 신경층을 통틀어서 가중치를 현재 시점의 입력뿐만 아니라 과거의 입력에도 적용합니다. 순환 신경망도 오차 역전파법을 사용합니다. 다만 시간에 걸쳐 펼쳐진 순환 신경망(unrolled RNN)에서 오차 역전파법을 사용하기 때문에 이를 **BPTT(Backpropagation Through Time)**이라고 부릅니다.

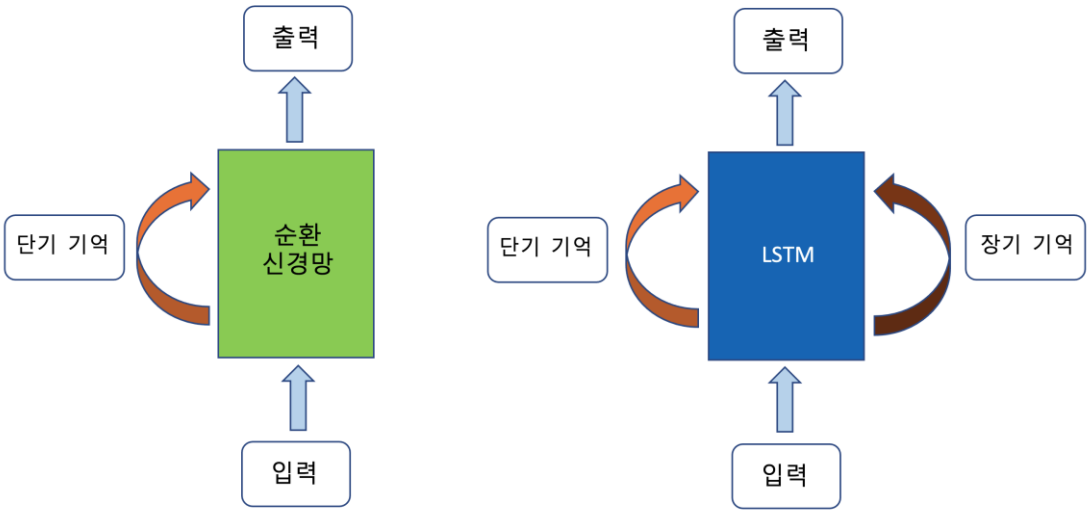
BPTT에서는 펼쳐져 있는 순환 신경망의 맨 마지막 시각(time step) t기부터 시작해서 맨 앞의 시각 0기까지 오차를 역전파합니다. 이를 통해 각 시각의 오차와 손실 함수를 계산하고, 경사 하강법을 통해 각 시각의 가중치를 조정합니다.

한편, 경사 하강법에 기반한 오차 역전파 학습을 하는 신경망 모델들은 공통적으로 기울기 소실 혹은 폭증이라는 문제에 직면합니다. 이는 경사 하강법이 기본적으로 손실 함수의 기울기에 의존하기 때문입니다. 만약 기울기가 너무 작아지면 대단히 미세한 양만큼만 가중치를 조정할 수 있어서 학습이 정체에 빠집니다. 반면에 기울기가 너무 커지면 가중치를 조금만 조정해도 손실 함숫값이 크게 변동하기 때문에 최적 가중치를 찾는 것이 힘들어지고 모델이 불안정해집니다.

이러한 기울기 소실 혹은 폭증이라는 문제는 순환 신경망 모델에도 문제가 될 수 있습니다. 이를 개선한 것이 LSTM 순환 신경망입니다.

B.7.2 LSTM

통상의 순환 신경망은 단기 메모리(working memory), 즉 가장 최근 시점의 메모리만 갖습니다. **LSTM(Long Short-Term Memory) 순환 신경망**은 기존의 순환 신경망에 장기 메모리를 추가한 것입니다. 때문에 가장 최근의 메모리뿐 아니라 장기 시차가 있는 메모리도 기억할 수 있습니다.



[그림 B-24] 순환 신경망과 LSTM 비교

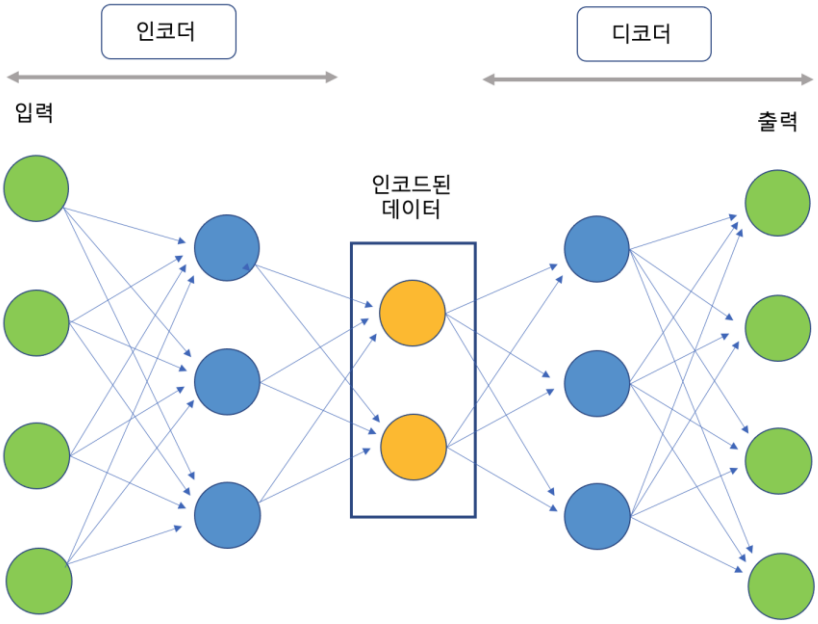
LSTM은 망각 게이트, 입력 게이트, 출력 게이트라는 3개의 게이트를 가지고 있고, 순환 신경망에서 장기 메모리와 단기 메모리 흐름 두 가지를 가지고 있습니다. LSTM을 간소화하려는 시도로는 **GRU(Gated Recurrent Units)** 순환망이 있습니다. 즉, GRU는 LSTM의 슬림화 버전이며 LSTM보다 조금 더 빠르게 실행됩니다. 구체적인 데이터를 기반으로 한 순환 신경망 실습은 6장을 참고하기 바랍니다.

B.8 오토인코더와 생성적 적대 신경망

B.8.1 오토인코더

> **일상생활 속의 딥러닝!**
> 딥러닝을 배우다 보면 예전에는 무심코 지나쳤던 일상 기술이 '아, 그게 딥러닝 기술이었어?'라고 깨닫게 되는 경우가 몇 번 있습니다. 예를 들어 음성 혹은 영상 압축 및 기술은 오토 인코더의 원리와 유사합니다.

오토인코더는 일반적인 피드 포워드 다층 신경망 중에서 입력 데이터 차원과 출력 데이터 차원이 같은 특별한 종류의 신경망입니다. 입력 변수의 레이블을 이용하지 않는다는 점에서 비지도 학습이기도 합니다. 컴퓨터로 한 번쯤 영상 변환을 해봤다면 인코딩(Encoding) 혹은 디코딩(Decoding)이라는 용어를 들어 봤을 것입니다. 오토인코더의 신경망 구성은 다음과 같습니다.



[그림 B-25] 오토인코더의 구조

오토인코더는 원본 입력 데이터를 압축하는 인코더와 이렇게 압축된 데이터를 원래의 입력 차원으로 복원하는 디코더로 구성돼 있습니다. 인코더를 거치면 압축된 데이터가 만들어지는데, 실생활에서의 예로는 음성 압축 파일 혹은 영상 압축 파일을 들 수 있습니다. 이렇게 압축된 파일을 휴대폰이나 컴퓨터 등 다른 전자기기에 보내서 재생하면 해당 프로그램 혹은 앱에서 디코더를 통해 음성 혹은 영상을 복원하여 출력합니다.

입력 데이터 차원과 같은 출력 데이터를 만들어낸 것이 무슨 의미가 있을까요? 그 의미는 두 가지가 있습니다. 첫째, 입력 데이터를 압축된 형태로 만들면 입력 데이터의 주요 특성을 잡아낼 수 있습니다. 둘째, 주요 특성을 중심으로 새로운 데이터를 만들어내는 과정에서 원본 입력 데이터의 중요하지 않은 특성은 무시할 수 있습니다. 즉, 노이즈를 제거합니다.

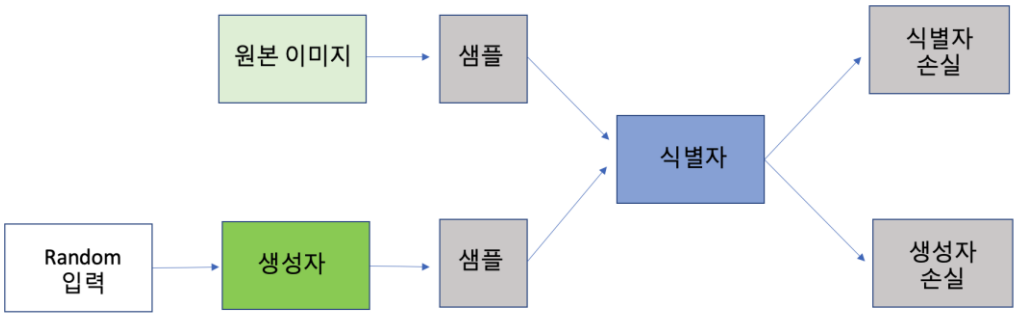
위의 두 장점을 담고 있는 오토인코더를 사용한 예로는 차원 축소, 특성 추출(feature extraction), 노이즈 제거 등이 대표적인 예입니다. 이는 오토인코더가 머신러닝 혹은 딥러닝 데이터 전처리 과정에서 사용될 수도 있음을 의미합니다.

반면에 단점도 있습니다. 오토인코더의 출력 결과는 입력 데이터의 주요 특성을 중심으로 생성되기에 입력 결과의 다운그레이드된 출력 결과만 나옵니다. 예를 들어 원본 사진을 입력하면 약간 흐릿한 출력 이미지를 산출합니다. 음악을 넣으면 출력 결과는 원래에 비해 음질이 떨어집니다. 때문에 실제로 음악이나 영상을 변환할 때는 오토인코더를 사용하지 않고, 별도의 전용 알고리즘을 활용합니다.

B.8.2 생성적 적대 신경망

오토인코더는 원본 입력 데이터와 비교해서 출력 데이터의 재현 정확성이 떨어진다고 언급하였습

니다. 그런데 사람이 원본 입력 데이터와 구별하기 힘들 정도로 출력 데이터의 재현 정확성을 올릴 수 있을까요? 생성적 적대 신경망(GAN, Generative Adversarial Networks)을 사용하면 가능합니다. GAN도 오토인코더처럼 입력 데이터의 레이블을 활용하지 않는 비지도 학습입니다. GAN은 다음 그림처럼 생성자(generator)와 식별자(discriminator)로 구성됩니다.



[그림 B-26] 상대적 적대 신경망 구조

생성자와 식별자 모두 신경망으로 구성됩니다. 오토인코더와 다르게 GAN은 원본 데이터(위 그림에서는 원본 이미지)로부터 직접 학습하지 않고, 랜덤 노이즈를 입력하여 데이터를 생성합니다. 식별자는 원본 데이터와 생성된 데이터를 비교합니다. 그 후 노이즈를 추가하여 다시 생성자에 입력합니다. 그다음은 위의 과정을 계속 반복합니다. 회로를 한 바퀴 돌 때마다 식별자 손실을 계산하는데, 오차 역전파법에 의해 식별자 손실값을 최소화하는 방향으로 식별자 가중치가 조정됩니다.

다음은 생성자 가중치를 조정할 차례입니다만, 생성자 손실을 직접적으로 계산할 방법이 없습니다. 식별자는 원본 데이터와 생성 데이터라는 비교 대상이 있어서 손실을 계산할 수 있는 반면에 생성자는 그러한 비교 대상이 없기 때문입니다. 따라서 생성자 손실이 식별자 손실에 이미 포함돼 있다는 가정하에, 식별자 손실을 최소화하는 방향으로 오차 역전파법에 의해 생성자의 가중치를 조정합니다.

이런 과정을 통해 식별자를 먼저 학습시키고, 그 이후 생성자를 학습시킵니다. 식별자가 원본 데이터와 생성 데이터를 구분해낼 수 없는 지경에 도달하면 학습을 종료합니다. 이 과정을 통해서 인간도 원본 데이터와 생성 데이터를 구분할 수 없을 정도의 **'원본보다 더 원본 같은'** 생성 데이터를 얻게 됩니다.

'원본보다 더 원본 같은'이라는 표현을 어디서 들어본 것 같습니다. 이 표현은 1982년 작 영화 블레이드 러너에서는 인공지능에 기반한 기계 인간 안드로이드를 만드는 타이렐 회사의 슬로건으로 등장합니다. 이 회사의 슬로건은 '인간보다 더 인간 같은(more human than human'입니다. GAN의 등장을 예지한 놀라운 영화라고 할 수 있습니다.
이 영화에서는 실제 인간(원본)과 안드로이드(원본을 복사한 생성물)를 구분하는 것이 주 업무인 '블레이드 러너'라는 직업군이 등장합니다. 타이렐 회사가 GAN 모델의 생성자이고, 블레이드 러너가 식별자인 셈입니다. 이 영화에서 생성된 안드로이드들을 원본인 인간과 구별할 수 없다면

GAN 모델의 학습이 성공한 셈입니다. 이 영화에서 숨겨진 GAN 모델은 학습에 성공했을까요? 영화를 직접 보고 판단해보기 바랍니다.

지금까지 딥러닝의 기초 개념들을 살펴봤습니다. 다음 부록에서는 책에 수록된 프로젝트의 심화 학습 과정으로 고급 프로젝트를 수행합니다.