ISSN 2288-4866 (Print)
ISSN 2288-4882 (Online)
http://www.jiisonline.org
J Intell Inform Syst 2016 June: 22(2): 127~142 http://dx.doi.org/10.13088/jiis.2016.22.2.127

딥러닝의 모형과 응용사례

안성만
국민대학교 경영대학 경영학부
(sahn@kookmin.ac.kr)

딥러닝은 인공신경망(neural network)이라는 인공지능분야의 모형이 발전된 형태로서, 계층구조로 이루어진 인공신경망의 내부계층(hidden layer)이 여러 단계로 이루어진 구조이다. 딥러닝에서의 주요 모형은 합성곱신경망(convolutional neural network), 순환신경망(recurrent neural network), 그리고 심층신뢰신경망(deep belief network)의 세가지라고 할 수 있다. 그 중에서 현재 흥미로운 연구가 많이 발표되어서 관심이 집중되고 있는 모형은 지도학습(supervised learning)모형인 처음 두 개의 모형이다. 따라서 본 논문에서는 지도학습모형의 가중치를 최적화하는 기본적인 방법인 오류역전파 알고리즘을 살펴본 뒤에 합성곱신경망과 순환신경망의 구조와 응용사례 등을 살펴보고자 한다. 본문에서 다루지 않은 모형인 심층신뢰신경망은 아직까지는 합성곱신경망 이나 순환신경망보다는 상대적으로 주목을 덜 받고 있다. 그러나 심층신뢰신경망은 CNN이나 RNN과는 달리 비지도학습(unsupervised learning)모형이며, 사람이나 동물은 관찰을 통해서 스스로 학습한다는 점에서 궁극적으로는 비지도학습모형이 더 많이 연구되어야 할 주제가 될 것이다.

주제어 : 딥러닝, 합성곱신경망, 순환신경망, 오류역전파 알고리즘

논문접수일 : 2016년 5월 17일 논문수정일 : 2016년 5월 18일 게재확정일 : 2016년 5월 19일
원고유형 : 일반논문 교신저자 : 안성만

1. 서론

최근에 무인자동차와 구글의 알파고 등으로 인하여 인공지능(AI, artificial intelligence) 혹은 기계학습(machine learning)에 대한 세간의 관심이 높아졌다. 인공지능이란 용어는 오래 전부터 사용되어 왔었으나 학문적으로는 그 발전이 정체되었다가 최근10년정도 전부터 딥러닝(deep learning)이라는 분야가 등장하게 되고 관심을 받게 되면서 연구가 활발히 진행되어 왔다. 딥러닝에 대한 연구가 최근 몇 년 동안 많이 진척되어 많은 흥미로운 결과가 나와서 작년에는 네이처에 딥러닝분야의 세계적 권위자 세 분이 리뷰논문을 발표하기도 하였다(LeCun et al., 2015). 딥러닝은 인공신경망(neural network)이라는 인공지능분야의 모형이 발전된 형태로서, 계층구조로 이루어진 인공신경망의 내부계층(hidden layer)이 여러 단계로 이루어진 구조이다. 최근의 딥러닝 모형은 내부계층이 많아져서 노드(node)를 연결하는 가중치(weight, 연결강도를 의미함)의 수가 최대 수십억 개가 되기도 한다(LeCun et al., 2015). 과거에는 그런 경우에 최적화를 하기가 어려웠으나, 최근에는 GPU를 이용하거나 연산속도를 향상시키는 다양한 방법이 개발됨으로

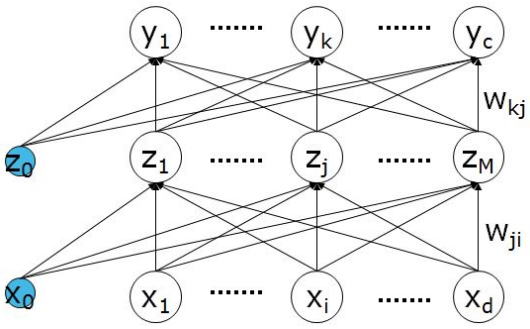
* 이 연구는 2016년 국민대 교내연구비 지원으로 수행되었음.

써 가능하게 되었다.

딥러닝에서는 여러 가지 모형이 있는데 그 중에서 관심을 모으고 있는 모형은 합성곱 신경망(convolutional neural network), 순환신경망(recurrent neural network), 심층신뢰신경망(deep belief network)의 세가지라고 할 수 있다. 그 중에서 현재 흥미로운 연구가 많이 발표되어서 관심이 집중되고 있는 모형은 처음 두 개의 모형이다. 따라서 본 논문에서는 합성곱 신경망과 순환신경망의 구조, 최적화 방법, 그리고 응용사례 등을 살펴보고자 한다.

본 논문의 서술 순서는 다음과 같다. 우선 인공신경망의 가중치를 최적화하는 기본적인 방법인 오류역전파 알고리즘을 다음절에서 살펴보고, 다음으로 합성곱 신경망의 특징과 응용사례, 그 다음절에서 순환신경망과 그 응용사례를 마지막으로 마무리를 하겠다.

2. 오류 역전파 알고리즘



〈Figure 1〉 Feed-forward neural network with one hidden layer

<Figure 1>에 있는 모형은 입력계층이 아래쪽에 위치해 있고 한 개의 내부계층이 그 위에 있고 출력계층이 가장 위에 위치해 있는 단순한 형태의 순방향신경망(feed-forward neural network)이다. Figure에서 x_i는 모형의 입력값이고 z_i는 내부계층에 있는 노드의 출력값이고, y_k는 출력계층의 출력값이다. x_0와 z_0는 각각 입력계층과 내부계층의 편향(bias)이며 값은 1로 간주한다. w_{kj}는 노드 J와 노드 k를 연결하는 가중치를 의미한다. 이런 형태의 신경망을 학습시킨다는 것은 입력계층에 패턴을 입력하였을 때 원하는 출력값이 나오도록 각 노드를 연결하는 가중치(그림에 있는 모든 w_{kj}와 w_{ji})를 설정하는 과정을 수행한다는 의미이다.

우선 <Figure 1>에 있는 모형에 패턴을 입력하였을 때 출력노드k의 출력값(y_k)을 수식으로 표현하면 다음과 같다.

$$y_k = \sigma\left(\sum_{j=0}^{M} w_{kj}\, h\left(\sum_{i=0}^{D} w_{ji} x_i\right)\right)$$

식에서 $\sigma(\cdot)$와 $h(\cdot)$는 각 노드에서의 활성화 함수 이며 s자모양의 함수를 사용하는데, 출력층의 경우에는 softmax 함수를 사용하기도 한다. 내부계층의 노드와 출력계층의 노드에 대한 입력값을 각각 a_j, a_k라고 하고, 입력부터 출력까지 수식을 나누어 쓰면 다음과 같다.

$$a_j = \sum_{i=0}^{D} w_{ji} x_i \quad (1)$$

$$z_j = h(a_j) \quad (2)$$

$$a_k = \sum_{j=0}^{M} w_{kj} z_j \quad (3)$$

$$y_k = \sigma(a_k) \quad (4)$$

이제 신경망을 학습시키기 위하여 오류함수를 설정한다. 오류함수는 비용함수라고도 하며 제곱합(sum-of-squares)의 형태와 교차 엔트로피(cross entropy) 등이 있는데, 여기서는 제곱합을 사용하기로 하고 다음과 같이 정의한다. 식에서 y_n과 t_n은 각각 신경망의 출력벡터와 목표벡터이며 N은 학습패턴의 수이다.

$$\mathrm{E}(\mathrm{W}) = \frac{1}{2}\sum_{n=1}^{N}\|\boldsymbol{y_n} - \boldsymbol{t_n}\|^2 \qquad (5)$$

위의 $\mathrm{E}(\mathrm{W})$는 w의 함수이며 곡면형태이므로 일차편미분벡터(gradient)가 0이 되는 지점에서 최소화된다. 그렇지만 $\mathrm{E}(\mathrm{W})$는 극도의 다차원공간에서 극도의 비선형이므로 일차편미분이 0이 되는 지점이 수없이 많이 존재하고 한번에 구할 수 있는 일반적인 방법조차 없다. 그 중의 일부는 안장점에 해당한다. 이러한 경우에 최적화하는 방법은 오류함수의 값을 감소시키는 w벡터를 반복적으로 개선해 나가는 방법으로서 다음과 같은 형태이다.

$$\mathrm{W}^{(\tau+1)} = \mathrm{W}^{(\tau)} + \Delta\mathrm{W}^{(\tau)}$$

식에서 윗첨자 τ는 반복단계를 나타내며 다음 단계로의 개선값인 $\Delta\mathrm{W}^{(\tau)}$는 알고리즘에 따라 다르다. 그 중에서 가장 단순한 형태의 방법이 일차편도함수를 이용하는 것으로서, gradient descent 혹은 steepest descent라고 불리며, 이 방법은 일차편도함수 벡터의 반대방향으로 개선하는 방식으로 다음의 식으로 표현된다.

$$\mathrm{W}^{(\tau+1)} = \mathrm{W}^{(\tau)} - \eta\nabla\mathrm{E}\left(\mathrm{W}^{(\tau)}\right)$$

식에서 η는 학습률(learning rate)로서 $\eta > 0$이다.

이제 일차편도함수를 효율적으로 구하는 방법인 오류역전파(error backpropagation) 알고리즘을 살펴보자. 이 알고리즘은 Rumelhart et al.(1986)에서 발표된 것으로 역사가 오래되었지만 오늘날의 신경망을 학습시키는 알고리즘의 기본이 되고 있다.

설명의 편의상 N개의 입력패턴 중에서 한 개의 패턴에 대한 오류함수를 고려하면 다음과 같다. t_k는 출력계층의 노드k의 목표값이다.

$$E_n = \frac{1}{2}\sum_{k=1}^{c}(y_k - t_k)^2 \qquad (6)$$

<Figure 1>에서 w_{kj}는 출력계층의 노드k와 내부계층(hidden layer)의 노드j를 연결하는 가중치이다. $\mathrm{E}(\mathrm{W})$의 w_{kj}에 대한 일차편도함수는 다음과 같다.

$$\frac{\partial E_n}{\partial w_{kj}} = \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial w_{kj}} \qquad (7)$$

위의 식은 편도함수를 연쇄법칙에 의하여 두 개의 편도함수의 곱으로 나타낸 것이다. a_k는 출력계층의 노드k의 입력값이므로 (4)와 (6)에 의해서 다음과 같다.

$$\frac{\partial E_n}{\partial a_k} = (y_k - t_k)\sigma'(a_k) \qquad (8)$$

그리고 (3)에 의해서

$$\frac{\partial a_k}{\partial w_{kj}} = z_j \qquad (9)$$

이제 δ_k를 다음과 같이 정의하자.

$$\delta_k \equiv \frac{\partial E_n}{\partial a_k} \quad (10)$$

그러면 (7)은 다음과 같다.

$$\frac{\partial E_n}{\partial w_{kj}} = \delta_k z_j \quad (11)$$

그러면 이제 내부계층의 노드*j*와 입력계층의 노드*i*를 연결하는 가중치인 w_{ji}에 대한 일차편도함수를 보자. (7)과 (11)을 참고하여 다음과 같이 표현할 수 있다.

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i \quad (12)$$

그런데 (12)에서 δ_j는 (10)에서 정의한 δ_k와 의미는 같지만 형태가 좀더 복잡하다. 그 이유는 다음과 같다. 신경망에서 a_k(출력노드 *k*의 입력값)이 y_k를 통하여 E_n에 미치는 영향은 비교적 단순하여서 (8)과 같이 표현 가능하였다. 그런데 a_j(내부노드 j의 입력값)가 E_n에 미치는 영향은 <Figure 1>에서 볼 수 있듯이 z_j를 거쳐서 모든 $y_k(\mathrm{k}=1,\cdots,\mathrm{c})$에 미치므로 그 경우를 다 합해야 한다. 따라서 δ_j는 연쇄법칙을 추가로 적용해서 다음과 같다.

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\left(\frac{\partial a_k}{\partial z_j}\frac{\partial z_j}{\partial a_j}\right) \quad (12)$$

좀 복잡해 졌지만 (1), (2), (10)을 이용해서 (12)를 다음과 같이 표현할 수 있다.

$$\delta_j = \mathrm{h}'(a_j)\sum_k w_{kj}\delta_k \quad (13)$$

여기서 δ_j가 어떻게 계산되는지 이해해 보자. δ_j는 내부계층의 노드*j*로의 입력값에 대한 E_n의 일차편도함수로서, (13)에서 볼 수 있듯이 두 가지 항의 곱으로 이루어져 있다. 첫 번째 항은 노드*j*의 활성화 함수를 a_j로 미분한 것이고, 두 번째 항은 상위계층의 일차편도함수(δ_k)를 w_{kj}로 가중합한 것이다.

오류역전파 알고리즘은 이러한 방식으로 δ_j를 역방향으로 계속 전달함으로써 E_n의 일차편도함수를 계산하는 방식이다. <Figure 1>에서는 내부계층이 하나만 존재하지만 일반적으로 딥러닝 모형에서는 두 개 이상의 내부계층이 존재한다. <Figure 1>에서 내부계층과 입력계층 사이에 또 다른 내부계층이 존재하는 경우에는 그 계층의 노드(*m*이라고 하자)와 기존의 내부계층의 노드*i*를 연결하는 가중치(w_{kj})에 대한 E_n의 일차편도함수는 (12)에 의하여 $\delta_i z_m$이 되며, δ_i는 (13)에 의하여 다음과 같다.

$$\delta_i = \mathrm{h}'(a_i)\sum_j w_{ji}\delta_j \quad (14)$$

이러한 방식으로 입력계층과 연결되는 가중치에 대한 일차편도함수까지를 계산하면 된다. 이제 오류 역전파 알고리즘을 요약하면 다음과 같다.

1. 입력패턴 x_n을 신경망의 입력계층에 입력하여 (1)과 (2)에 따라 순방향으로 값을 전

달한다. 그 과정에서 각 노드의 출력값인 z_j를 저장해 놓는다.

2. 출력계층의 노드에 대한 δ_k를 (10)을 이용하여 계산한다.
3. δ를 역방향으로 전달하는 과정을 통하여 내부계층의 노드에 대한 δ_j를 계산한다.
4. (12)를 이용하여 일차편도함수를 계산한다.

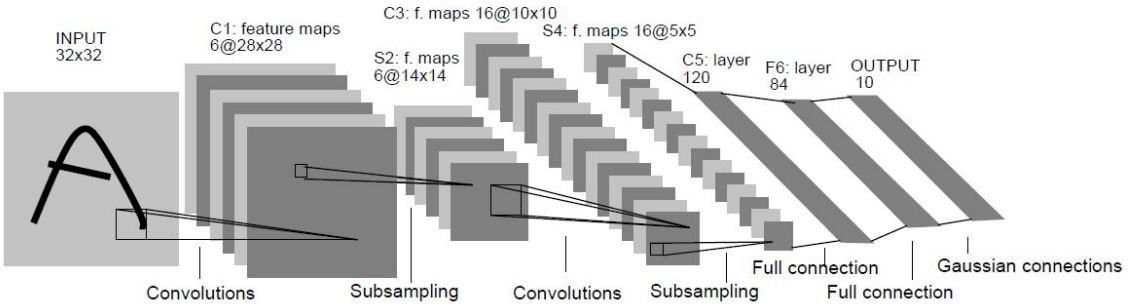
오류역전파 알고리즘의 가장 큰 장점은 계산속도가 빠르다는 것이다. (1)과 (3)을 보면 각 가중치에 대하여 한번의 곱셈이 일어나므로 순방향으로 값을 전달할 때 총 W번의 곱셈을 하고 오류를 역방향으로 전달할 때 W번의 곱셈을 하므로 총 $2W$번의 곱셈을 하게 되어서 O(W)의 연산비용이 발생한다. 한편 일차편도함수를 계산하는 다른 방식으로서 유한차분(finite difference)을 이용할 수 있는데, 다음과 같이 정의된다.

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{E_n(w_{ji}+\epsilon) - E_n(w_{ji}-\epsilon)}{2\epsilon} \qquad (15)$$

(15)를 통하여 일차편도함수를 계산하려면 현재의 w_{ji}에서 매우 작은 ϵ값을 사용해야 한다. 이런 방식으로 계산하게 되면 각 w_{ji}에 대한 일차편도함수의 계산에 O(W)의 연산비용이 발생하므로 모든 w_{ji}에 대하여 일차편도함수를 계산하려면 O(W^2)의 연산비용이 발생한다. 따라서 두 방식은 연산비용 측면에서 보면 그 차이가 실로 어마어마하다. 그래서 반복계산을 통한 신경망의 학습을 위해서는 오류역전파 알고리즘의 이용이 필수적이다. 단 (15)는 오류역전파 알고리즘을 통한 일차편도함수의 계산이 올바른지를 비교하기 위한 목적으로 사용되기도 한다.

3. 합성곱 신경망

합성곱 신경망(CNN, convolutional neural network)은 생물의 시각처리과정을 모방하여 패턴의 크기나 위치가 바뀌어도 인식을 할 수 있다는 장점이 있는 모형으로서 그 뿌리는 Neocognitron(Fukushima, 1980)이라는 것이다. 이 모형이 발전되어서 LeCun et al.(1998)을 통하여 현재 사용되고 있는 합성곱 신경망이 널리 쓰이는 계기가 되었다. LeCun et al.(1998)에 의하면 합성곱 신경망은 세가지 특징을 가지고 있는데, 그것은 local receptive field, shared weight, 그리고 sub-sampling이다.



〈Figure 2〉 Convolutional neural network (LeCun et al., 1998)

local receptive field는 <Figure 1>에서와 같은 기존의 모형이 하위계층의 각 노드가 모든 상위계층의 노드와 연결되어 있는 것과 달리, 상위계층의 일부의 노드에만 가중치로 연결되는 것을 의미한다. <Figure 2>에서 가장 왼쪽에 있는 것이 입력패턴이며, 입력패턴의 오른쪽 하단에 있는 작은 사각형이 local receptive field가 되며 이것이 우측에 있는 다음계층(C1)의 한 개의 노드와 가중치(필터라고도 부른다. 필터의 크기가 5 X 5라면 편향을 더하여 총 26개의 가중치가 있다)로 연결되어 있는 것을 볼 수 있다. 이러한 필터를 입력패턴의 모든 영역(왼쪽상단에서 아래쪽 하단으로)에 적용하여(가중치를 해당 영역에 곱하여 합하는 연산을 함) C1에 있는 패널의 노드로 전달하는 것을 합성곱(convolution)이라고 한다. 한편, shared weight는 가중치를 공유한다는 뜻이다. <Figure 2>의 C1에는 여러 개의 패널이 있는 것을 볼 수 있는데, 한 개의 패널에 있는 노드와 입력패턴(혹은 하위계층)에 적용되는 필터는 동일하다는 의미이다. <Figure 2>의 C1에는 6개의 패널이 있으므로 이 경우에는 전부 6개의 서로 다른 필터가 존재한다. 한 개의 필터는 한 개의 특징(혹은 작은 패턴)을 인식할 수 있으므로 C1에 있는 한 개의 패널은 입력패턴의 특정 위치에 존재하는 해당 특징을 발견할 수 있게 된다. sub-sampling은 <Figure 2>의 왼쪽에서 두 번째 계층(C1)과 세 번째 계층(S2)에서 일어나는 과정으로서 C1에 있는 패널을 압축하는 것이며 pooling이라고도 부른다. 예를 들면 C1에 있는 2 X 2영역을 S2의 하나의 노드로 만드는 것으로서, 축소하는 방법은 다양하다. LeCun et al.(1998)에서는 2 X 2영역의 값을 평균하여 가중치를 곱한 뒤 *S*자 모양의 활성화 함수를 적용하는 방법 average pooling을 사용하였으나 최근에는 2 X 2 영역의 네 값 중에서 가장 큰 값을 사용하는 max-pooling 방법을 많이 사용한다. sub-sampling은 입력패턴의 크기를 줄여주는 기능과 발견한 특징에 대한 위치의 변동에 덜 민감하게 하는 기능을 한다.

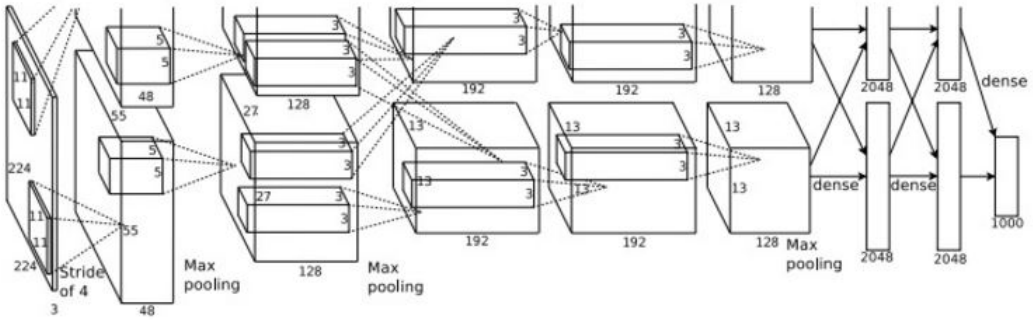
이와 같은 convolution과 sub-sampling과정은 응용사례에 따라서 여러 번 반복되며 <Figure 2>에서는 2번이 반복되는 것을 볼 수 있다. sub-sampling이 종료된 후에는 <Figure 1>과 같은 형태의 신경망이 연결된다. <Figure 2>에서는 우측에 두 개의 내부계층(C5와 F6)과 출력계층이 있는 것을 볼 수 있다. 출력계층에는 10개의 노드가 있는데 이는 숫자(0~9)를 인식하는 모형이기 때문이다.

합성곱 신경망의 가중치는 어떤 방법으로 학습시키는가? 합성곱 신경망은 내부계층이 매우 많으므로 따라서 노드의 수가 많아지고 노드를 연결하는 가중치도 많아지게 된다. 일반적으로 모형에서 가중치의 수가 많아지면 앞에서 설명한 오류 역전파 알고리즘을 적용할 때 일차편도함수를 반복적으로 계산하는 과정에서 그 크기가 매우 커지거나 반대로 0이 되는 문제가 발생하게 된다. 하지만 합성곱 신경망의 경우에는 shared weight라는 특징이 있어서 가중치의 수가 그런 문제가 발생할만큼 많지 않아서 오류 역전파 알고리즘을 무리 없이 적용할 수 있다. 다만 shared weight란 특징과 sub-sampling이란 과정이 일반적인 신경망 모형과 다르기 때문에 알고리즘을 조금 수정할 필요가 있다. 우선 shared weight에 대해서는 가중치가 서로 다른 경우와 동일한 방법으로 일차편도함수를 계산한 후에 동일한 가중치에 대하여는 일차편도함수의 값을 모두 더해주는 과정을 적용하면 된다. sub-sampling을 통하여 패턴이 압축되는 경우에

는 오류를 역전파 할 때 반대방향의 연산을 해주어야 하므로 up-sampling을 통하여 확장해주는 과정이 필요하다. 그런데 sub-sampling의 방식이 응용사례에 따라 다르므로 경우에 따라 다르게 적용할 필요가 있다. 예를 들어 LeCun et al. (1998)에서 적용된 sub-sampling에서는 가중치와 활성화 함수를 사용하므로 일반적인 오류역전파 알고리즘을 적용하여 가중치의 일차편도함수를 계산하고 그때 계산한 δ를 up-sampling을 통하여 아래로 전달하면 된다. 가중치와 활성화 함수가 없이 max-pooling을 했다면 상위노드에서 계산한 δ를 max-pooling단계에서 가장 큰 값이었던 노드로 전달하여 하위노드의 δ를 계산한다.

이러한 합성곱신경망은 문자인식과 다양한 형태의 영상인식 등에 성공적으로 적용되어 왔다. 특히 음성인식, 문서읽기, 그리고 필기체문자인식 등의 분야에서는 1990년대 초반부터 적용되어 왔으며, LeCun et al.(1998)의 모형은 MNIST (H1)에 대하여 매우 낮은 오류율을 보인바 있다. 그러나 합성곱신경망이 본격적으로 학계의 주요 관심사가 된 것은 2012년에 발표된 영상인식모형(Krizhevsky et al., 2012)이 놀라운 성과를 거두면서부터이다.

이 모형은 백만 개 정도의 컬러영상을 천 개의 클래스로 분류하는 일을 하는 것인데, 전체적인 모형은 <Figure 3>과 같다. <Figure 3>에서 보듯이 이 모형은 두 개의 GPU를 통하여 구현되었기 때문에 입력계층과 출력계층을 제외한 내부계층은 두 부분으로 구분되어 구성되어 있다. 컬러영상을 인식해야 하므로 입력패널은 3개이고, <Figure 3>에서 입력계층 오른쪽에 있는 첫 번째 내부계층은 convolution과 max-pooling을 수행하는 것으로 총 96개의 패널로 이루어져 있다. 그 오른쪽의 두 번째 내부계층은 마찬가지로 convolution과 max-pooling을 수행하며 총 256개의 패널로 구성된다. 그 다음에 있는 두 개의 내부계층은 max-pooling이 없는 convolution을 하고 다섯 번째 내부계층은 convolution과 max-pooling을 수행하는 256개의 패널로 구성된다. 여섯 번째와 일곱 번째 내부계층은 하위계층의 노드와 전부 연결되어 있는 구조이고 마지막 출력계층은 1000개의 노드로 구성되어 있다. 이 모형은 대략 6천만개의 가중치가 있으며 학습시간으로 GTX 580 3GB GPU를 사용하여 6~7일 정도가 소요되었다. 이 모형은 활성화 함수로 ReLU (max(0,x))를 사용하여 학습시간을 단축시키고, dropout이라는 방식으로 과적합을 방지하고, 또



〈Figure 3〉 CNN for image classification (Krizhevsky et al., 2012)

학습영상을 변형하여 입력패턴의 수를 늘리는 방법으로 모형의 인식률을 높였다. 그 결과 당시의 다른 모형의 인식 오류율의 절반에 가까운 오류율을 달성하는 성과를 얻었다. 그러한 이유로 현재의 많은 합성곱신경망 모형은 10~20개의 ReLU 계층과 수십억 개의 노드간 연결이 있고 수억 개의 가중치를 학습시킨다. 물론 학습알고리즘의 발전과 GPU기술의 발전으로 학습속도는 수시간단위로 단축이 되었다.

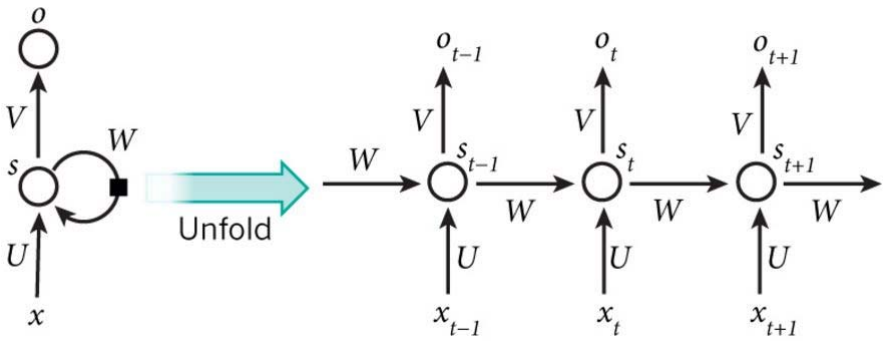
4. 순환신경망

순환신경망(RNN, recurrent neural network) 모형은 음성인식이나 언어인식 등과 같은 순차적인 정보를 처리하는 데 적합하다. RNN은 순차적 데이터를 구성하는 원소를 한번에 하나씩 입력받아서 처리하며, 처리된 정보를 내부노드에 저장하는데, 입력된 시간에 따라 그에 대응하는 내부노드를 생성하여 그 시점까지 처리한 정보를 저장한 뒤에 필요하면 출력노드를 통하여 출력을 한다.

RNN은 <Figure 4>의 왼쪽에 있는 형태의 모형으로서 t시간의 입력데이터(x_t)가 주어지면 가중치(U)를 곱하여 내부노드로 입력되고 내부노드는 이전에 저장된 정보를 추가하여 상태값을 생성하게 되는데, t시간의 내부노드의 상태값(s_t)을 식으로 표현하면 다음과 같다.

$$s_t = f(Ux_t + Ws_{s-1}) \tag{16}$$

f는 활성화 함수로서 S자함수 혹은 ReLU를 사용한다. t시간에서 모형의 출력은 내부노드의 상태값에 V를 곱하여 생성하며 경우에 따라서 softmax함수를 적용하기도 한다. <Figure 4>의 왼쪽에 있는 모형은 일견 간단해 보이지만 내부노드에서 순환고리로 이전 정보를 재입력하는 부분이 순방향신경망과 다른 점이며 이것이 모형의 작동을 복잡하게 만드는 요인이다. 모형의 이해를 좀 쉽게 하기 위하여 RNN을 시간의 진행단계에 따라 펼쳐서 그리기도 하는데, <Figure 4>의 오른쪽에 있는 모형이 바로 그것이다. 이와 같이 모형을 펼쳐서 그리면 내부계층이 세 개(s_{s-1}, s_t, s_{t+1})인 순방향신경망이라고 할 수 있



〈Figure 4〉 Recurrent neural network

다. 다른 점이라고 하면 입력데이터가 서로 다른 내부노드로 입력된다는 것과 내부계층을 연결하는 가중치(*W*)가 모두 동일하다는 것이다. <Figure 4>의 오른쪽에 있는 펼쳐진 구조에서 내부노드의 수는 세 개로 그려져 있지만, 그 수는 입력되는 데이터의 수에 따라 달라진다. 예를 들어 입력되는 문장의 단어수가 10개라고 하고 한 단어를 한번에 입력하는 모형이라면 내부노드는 10개가 만들어 질 것이다.

RNN에서는 교차엔트로피 오류함수를 많이 사용하고 있는데, *t*시간에서의 목표값을 t_t라고 하고 모형의 출력을 o_t라고 하면 *t*시간에서의 오류값은 다음과 같으며, 모형 전체의 오류는 $\sum_t E_t$이다

$$E_t(t_t, o_t) = -t_t \log o_t \tag{17}$$

RNN에서 가중치를 학습하는 방법은 <Figure 4>의 펼쳐진 모형에 대하여 오류역전파 알고리즘을 적용하는 것이다. 다만 중복되는 가중치에 대하여 편도함수를 합하는 과정이 필요한데, 이 방법을 BPTT(backpropagation through time)라고 부르기도 한다. 이제 내부계층을 연결하는 가중치인 W에 대한 일차편도함수를 계산하는 과정을 보기로 하자. 모형 전체의 *W*에 대한 일차편도함수는 다음과 같다.

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W} \tag{18}$$

<Figure 4>의 펼쳐진 모형에 있는 *t*시간의 출력(o_t)에 대한 편도함수를 이용하여 연쇄법칙을 적용하면 s_t로 표현된 내부노드로 연결되는 *W*에 대한 E_t의 일차편도함수는 다음과 같다.

$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial o_t}\frac{\partial o_t}{\partial s_t}\frac{\partial s_t}{\partial W} \tag{19}$$

한편 *W*는 시간을 거슬러 첫 단계의 내부노드까지 가야 하므로 s_1에 연결된 *W*에 대한 E_t의 일차편도함수는 다음과 같다.

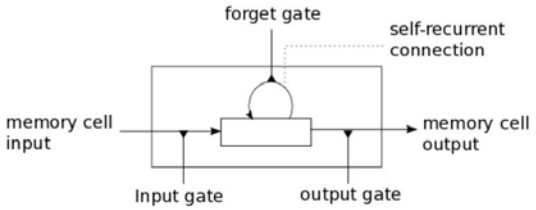
$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial o_t}\frac{\partial o_t}{\partial s_t}\frac{\partial s_t}{\partial s_{t-1}}\cdots\frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W} \tag{20}$$

(19)와 (20)을 중간단계의 *W*에 대한 편도함수를 계산하여 모두 더하고 (18)을 이용하면 다음과 같다.

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T}\sum_{k=1}^{t}\frac{\partial E_t}{\partial o_t}\frac{\partial o_t}{\partial s_t}\left(\prod_{j=k}^{t-1}\frac{\partial s_{j+1}}{\partial s_j}\right)\frac{\partial s_k}{\partial W} \tag{21}$$

비슷한 방법으로 *U*와 *V*에 대한 편도함수도 계산할 수 있다. 그런데 (21)을 이용하여 편도함수를 계산하는 과정에서 여러 번 편도함수를 곱하는 연산이 일어나게 되는데, 입력되는 문장의 수가 늘어나게 되어서 내부노드의 수가 많아지게 되면 곱셈의 수도 늘어나게 된다. 이렇게 많은 값을 곱하게 되면 경우에 따라서 편도함수의 값이 0에 가까워 지거나(vanishing gradient) 아니면 매우 큰 값이 될(exploding gradient) 수가 있다. 편도함수의 값이 매우 커지는 경우는 값을 제한함으로써 비교적 쉽게 해결할 수 있지만 0에 가까워지는 경우는 발견하기가 쉽지 않고 그렇게 되면 가중치의 학습속도가 매우 느려지거나 멈추는 상황이 발생한다. 활성화 함수로 *S*자 함수 이외에 ReLU를 사용하여 편도함수가 작은 값이

되지 않도록 할 수도 있지만, 그것을 근본적으로 해결하기 위한 모형이 1997년에 제안 되었는데 그것이 LSTM(Long short-term memory)이다 (Hochreiter & Schmidhuber, 1997). LSTM은 <Figure 4>에 있는 RNN의 내부노드를 메모리셀(memory cell)이라고 부르는 복잡한 구조로 바꾸었는데, 다양한 형태의 그림으로 표현되지만 그 중에서 비교적 단순하게 표현한 것이 <Figure 5>이다.



〈Figure 5〉 Memory cell of LSTM (2)

LSTM의 메모리셀은 네 개의 구성요소가 있는데, 그것은 순환 입력값을 가지는 내부기억노드와 세 개의 개폐장치(input gate, forget gate, output gate)이다. 각 개폐장치는 로지스틱 함수와 같은 0에서 1 사이의 값으로서, 그것을 통과하는 값을 조절하는 기능을 한다. 그 중에서 forget gate은 Hochreiter & Schmidhuber(1997)에서는 없었지만 Gers et al.(2000)에서 추가된 것으로 현재의 모든 모형에서 사용되고 있다.

t시간에서 LSTM의 메모리셀의 상태값을 수정하는 공식은 다음과 같다. 우선 i_t를 input gate의 값이라 하고 $\tilde{C}_t$를 내부기억노드의 입력값이라고 하면 그 값은 다음과 같이 계산한다.

$$i_t = \sigma(U_i x_t + W_i s_{t-1} + b_i) \quad (22)$$

$$\tilde{C}_t = tanh(U_c x_t + W_c s_{t-1} + b_c) \quad (23)$$

이제 f_t를 forget gate의 값이라고 하면 내부기억노드의 새로운 상태값은 다음과 같이 계산한다.

$$f_t = \sigma(U_f x_t + W_f s_{t-1} + b_f) \quad (24)$$

$$C_t = i_t * \tilde{C}_t + f_t * C_{t-1} \quad (25)$$

새로운 상태값을 사용하여 output gate의 값 (o_t)과 메모리셀의 출력값을 다음과 같이 계산한다.

$$o_t = \sigma(U_o x_t + W_o s_{t-1} + b_0) \quad (26)$$

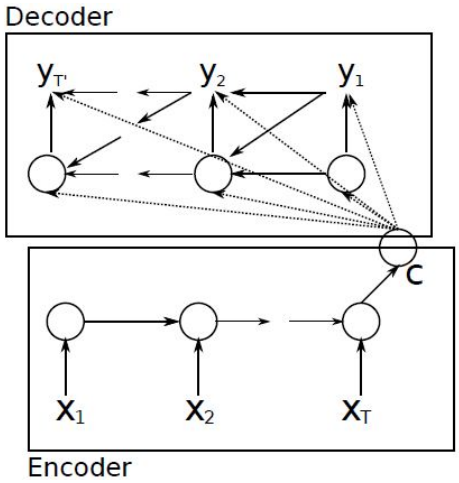
$$s_t = o_t * tanh(C_t) \quad (27)$$

위의 식에서 *W*, *U*, *V*는 해당 gate에서의 가중치이며 *b*는 편향이다.

(22)에서 (27)까지의 식을 보면 LSTM 모형은 매우 복잡하며, 따라서 모형의 가중치의 편도함수의 계산도 매우 복잡하다. 모형의 편도함수는 Hochreiter & Schmidhuber(1997) 와 Graves(2012)에서 확인가능하며 소프트웨어로 가중치의 학습을 구현하는 경우에는 Python의 Theano library(H2)와 같은 자동미분 기능을 활용한다.

RNN은 일반적으로 입출력되는 데이터의 형태가 순차적인 모형이다. 그러나 경우에 따라서는 순차적이지 않은 데이터가 입출력될 수도 있다. 입출력되는 데이터의 형태에 따라서 다양한 응용이 가능한데, 입출력 데이터가 순차적인 경우는 기계번역 혹은 동영상에 주석 달기에 적용되는 경우이고, 출력이 비순차적인 경우는 감성분석에, 그리고 입력이 비순차적인 경우는 사진에 주석 달기 등에 적용되는 경우이다. 이와 같

이 다양한 분야에 적용될 수 있는 RNN의 응용사례 중에서 Cho et al.(2014)에서 제안된 기계번역모형을 간략히 소개하기로 한다.



〈Figure 6〉 RNN for machine translation

기계번역은 특정 언어로 된 문장을 입력하면 목표 언어로 번역이 되어서 출력되는 모형을 의미하며, Cho et al.(2014)에서는 영어를 불어로 번역하는 사례에 적용되었다. <Figure 6>에서 보듯이 이 모형은 encoder와 decoder로 불리는 두 개의 RNN을 연결하여 구성되었다. encoder에서는 입력문장을 구성하는 단어를 순차적으로 입력받아서 내부노드의 상태값으로 축적하여 최종 상태를 C로 요약하여 decoder로 전달한다. Decoder는 encoder를 아래위로 뒤집은 형태로서 목표언어로 번역된 문장을 한 단어씩 출력하게 된다. 이 모형에서 내부노드는 LSTM을 조금 단순화한 GRU라는 모형을 사용하였는데 이 모형은 LSTM과 성능 면에서 비슷한 것으로 평가 받고 있다. 자세한 내용은 Cho et al.(2014)를 참조하기 바란다.

기계번역 이외에 흥미로운 응용사례로서 단어나 글자를 입력하면 RNN이 자동으로 그 다음에 나올 단어나 글자를 예측하여 출력해주는 모형이 있다. 이 모형을 응용하여 Linux 코딩이나 셰익스피어의 문장을 흉내 내도록 할 수도 있다. 관심 있는 독자는 해당 블로그(H3)를 참조하기 바란다.

5. 요약

지금까지 딥러닝의 기본 알고리즘과 주요 모형을 살펴보고 몇 가지 응용사례도 살펴보았다. 그러나 본문에서 살펴본 사례는 다양한 응용사례의 극히 일부분에 지나지 않는다. 본문의 내용 중에 오류역전파 알고리즘에 대한 내용은 Bishop(2006)을 참조하였으며, 최신의 응용사례나 Python 코드에 관해서는 여러 블로그나 웹사이트에 있는 내용을 참조하였다. 딥러닝의 전반적인 내용은 (H2)를 참조하였고, CNN에 관련해서는 (H4)를, RNN에 관련해서는 (H3) 혹은 (H5)를, 그리고 LSTM에 관한 자세한 설명은 (H6)을 참조하였다. 신경망의 가중치를 학습하는 방법으로 오류역전파 알고리즘과 BPTT 등을 설명하였으나, 그것만 가지고는 응용사례별로 가중치를 최적화 하기에는 부족하다. 모형의 과적합(overfitting)을 방지하기 위하여 사례별로 다양한 regularization방법과 탐색속도를 높이기 위한 다양한 방법, 그리고 오류율을 낮추기 위한 다양한 방법을 많은 시행착오를 겪은 뒤에 찾아내기도 한다. 그런 점에서 최적화 과정은 높은 인내심이 요구되는 작업이기도 하다.

서론에서 언급하였지만 본문에서 다루지 않은 모형인 심층신뢰신경망(Deep belief network)은

아직까지는 CNN이나 RNN처럼 흥미로운 응용사례가 없어서 상대적으로 주목을 덜 받고 있다. 그러나 심층신뢰신경망은 CNN이나 RNN과는 달리 비지도학습(unsupervised learning)모형이며, 사람이나 동물은 관찰을 통해서 스스로 학습한다는 점에서 궁극적으로는 비지도학습모형이 더 많이 연구되어야 할 주제가 될 것이다(LeCun et al., 2015).

본문에 언급되지 않은 사례로서 최근의 딥러닝 연구의 발전을 보여주는 것으로서는 사진이나 영상의 주석을 다는 모형일 것이다. Vinyals et al.(2014)와 Cho et al.(2015)에 있는 모형은 CNN과 RNN을 결합한 것으로서 사진이나 영상을 CNN이 입력 받아서 요약을 한 뒤에 그 결과를 RNN으로 입력하여 의미 있는 문장을 출력하게 한다. 모형이 동작하는 원리를 떠나서 그 결과만을 보면 인공지능이 마치 사물을 이해할 수 있는 경지에 이른 것처럼 보이기도 한다. 이와 같은 응용사례에 비추어 본다면, 비록 기계학습이나 그것이 발전한 인공지능이라는 것은 하나의 소프트웨어에 지나지 않지만 그 기능은 사람만이 할 수 있다고 여겨졌던 영역의 일부를 모방할 수 있는 경지에 도달하고 있으며 그 영역을 가까운 미래에 더욱 확장할 것으로 기대된다.

참고문헌(References)

Bishop, C., *"Pattern Recognition and machine Learning,"* Springer, 2006.

Cho, K., A. Courville and Y. Bengio, "Describing Multimedia Content using Attention-based Encoder–Decoder Networks," arXiv preprint arXiv:1507.01053, 2015.

Fukushima, K., "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol.36, no.4(1980), 193~202.

Gers, F. A., J. Schmidhuber and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural computation*, Vol.12, No.10(2000), 2451~2471.

Hochreiter, S. and J. Schmidhuber, "Long short-term memory," *Neural Compuation*, Vol.9(1997), 1735~1780.

Krizhevsky, A., I. Sutskever and G. Hinton, "ImageNet classification with deep convolutional neural networks," *Proc. Advances in Neural Information Processing Systems 25*, 2012, 1090~1098.

LeCun, Y., Y. Bengio, and G. Hinton, "Deep Learning," *Nature* 521, 2015, 436~444.

LeCun, Y., L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE* 86, 1998, 2278~2324.

Rumelhart, D. E., G. E. Hinton and R. J. Williams, "Learning representations by back-propagating errors," *Nature* 323, 1986, 533~536.

Salakhutdinov, R. and Hinton, G., "Deep Boltzmann machines," *Proc. International Conference on Artificial Intelligence and Statistics*, 2009, 448~455.

Vinyals, O., A. Toshev, S. Bengio and D. Erhan, "Show and tell: a neural image caption generator," *Proc. International Conference on Machine Learning*, Available at http://arxiv.org/abs/1502.03044 (Downloaded 2014).

(H1) http://yann.lecun.com/exdb/mnist/

(H2) http://deeplearning.net/tutorial/contents.html

(H3) http://karpathy.github.io/2015/05/21/rnn-effectiveness/

(H4) http://neuralnetworksanddeeplearning.com/

(H5) http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

(H6) http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Abstract

Deep Learning Architectures and Applications

Ahn, SungMahn*

Deep learning model is a kind of neural networks that allows multiple hidden layers. There are various deep learning architectures such as convolutional neural networks, deep belief networks and recurrent neural networks. Those have been applied to fields like computer vision, automatic speech recognition, natural language processing, audio recognition and bioinformatics where they have been shown to produce state-of-the-art results on various tasks. Among those architectures, convolutional neural networks and recurrent neural networks are classified as the supervised learning model. And in recent years, those supervised learning models have gained more popularity than unsupervised learning models such as deep belief networks, because supervised learning models have shown fashionable applications in such fields mentioned above.

Deep learning models can be trained with backpropagation algorithm. Backpropagation is an abbreviation for "backward propagation of errors" and a common method of training artificial neural networks used in conjunction with an optimization method such as gradient descent. The method calculates the gradient of an error function with respect to all the weights in the network. The gradient is fed to the optimization method which in turn uses it to update the weights, in an attempt to minimize the error function.

Convolutional neural networks use a special architecture which is particularly well-adapted to classify images. Using this architecture makes convolutional networks fast to train. This, in turn, helps us train deep, muti-layer networks, which are very good at classifying images. These days, deep convolutional networks are used in most neural networks for image recognition. Convolutional neural networks use three basic ideas: *local receptive fields*, *shared weights*, and *pooling*. By local receptive fields, we mean that each neuron in the first(or any) hidden layer will be connected to a small region of the input(or previous layer's) neurons. Shared weights mean that we're going to use the *same* weights and bias for each of the local receptive field. This means that all the neurons in the hidden layer detect exactly the same feature, just at different locations in the input image. In addition to the convolutional layers just described, convolutional

* College of Business Administration, Kookmin University
77 JeongNeung-Ro, SeongBuk-Gu, Seoul, 02707, Korea
sahn@kookmin.ac.kr

neural networks also contain *pooling layers*. Pooling layers are usually used immediately after convolutional layers. What the pooling layers do is to simplify the information in the output from the convolutional layer. Recent convolutional network architectures have 10 to 20 hidden layers and billions of connections between units. Training deep learning networks has taken weeks several years ago, but thanks to progress in GPU and algorithm enhancement, training time has reduced to several hours.

Neural networks with time-varying behavior are known as *recurrent neural networks* or *RNNs*. A recurrent neural network is a class of artificial neural network where connections between units form a directed cycle. This creates an internal state of the network which allows it to exhibit dynamic temporal behavior. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. Early RNN models turned out to be very difficult to train, harder even than deep feedforward networks. The reason is the unstable gradient problem such as vanishing gradient and exploding gradient. The gradient can get smaller and smaller as it is propagated back through layers. This makes learning in early layers extremely slow. The problem actually gets worse in RNNs, since gradients aren't just propagated backward through layers, they're propagated backward through time. If the network runs for a long time, that can make the gradient extremely unstable and hard to learn from. It has been possible to incorporate an idea known as long short-term memory units (LSTMs) into RNNs. LSTMs make it much easier to get good results when training RNNs, and many recent papers make use of LSTMs or related ideas.

Key Words : Deep learning, Convolutional neural networks, Recurrent neural networks, Error backpropagation algorithm

Received : May 17, 2016 Revised : May 18, 2016 Accepted : May 19, 2016
Publication Type : Regular Paper Corresponding Author : Ahn, SungMahn

저 자 소 개

안 성 만

현재 국민대학교 경영대학 경영학부교수로 재직하고 있다. George Mason University에서 박사학위를 취득하였다. Johns Hopkins University의 응용수학과에서 박사후과정을 보냈고 ㈜쌍용정보통신에서 근무했으며, 주요 관심분야는 통계적 방법론, 모형선택, 데이터 마이닝, 뉴럴네트워크 등이다.