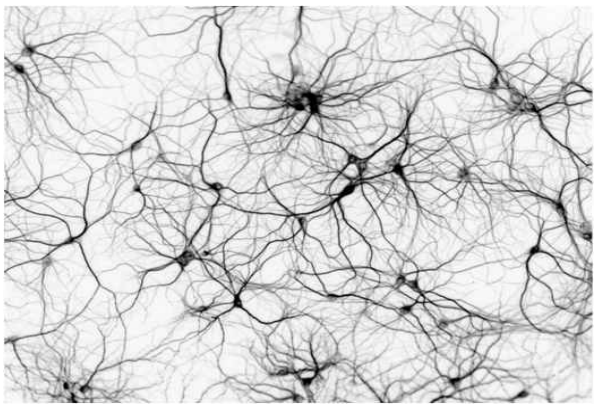
Chapter 11 신경망

[학습목표]
뇌의 정보처리 메커니즘을 모방하여 개발된 인공신경망은 다양한 비선형 판별함수를 표현하고 학습을 통해 최적의 파라미터를 찾는 훌륭한 패턴분류기이다. 특히 이 장에서 소개하는 다층 퍼셉트론은 6장에서 간략히 소개한 선형 판별함수를 나타내는 퍼셉트론의 확장으로 볼 수 있다. 여기서는 먼저 신경망의 기본적인 개념을 소개하고, 6장에서 살펴본 퍼셉트론의 한계점을 알아본 후, 이를 극복한 다층 퍼셉트론에 대하여 자세히 알아본다. 또한 군집화 문제에 적용할 수 있는 자기조직화 신경망에 대해서도 알아본다.

11.1 인공신경망

11.1.1 신경망이란?

신경망이란, 인간의 뇌의 구조와 뇌에서 수행되는 정보처리 방식을 모방함으로써 인간이 지능적으로 처리하는 복잡한 정보처리 능력을 기계를 통해 실현하고자 하는 연구이다. 따라서 신경망의 접근법을 이해하기 위해서는 먼저 인간의 뇌에서 일어나는 정보처리 방식에 대하여 이해할 필요가 있다.
인간의 뇌는 신경세포(neuron)라는 간단한 구조와 기능을 가진 세포들이 매우 복잡한 구조로 연결되어 지능적인(매우 복잡한 비선형적 정보 처리) 정보처리 기능을 수행한다. 즉, 인간의 뇌는 100억 개 이상의 신경세포와 60조 이상의 연결을 가진 매우 복잡한 구조를 가지고 있다.([그림 11-1] 참조). 또한 신경세포 간의 연결은 태어날 때부터 완성되어 있는 것이 아니라, 자라면서 자극을 받고 처리하는 과정에서 그 연결 정도가 스스로 조정된다. 이것이 생물체가 가지는 환경에 대한 적응성(adaptation)과 새로운 것에 대한 학습(learning) 능력의 기본이 된다.



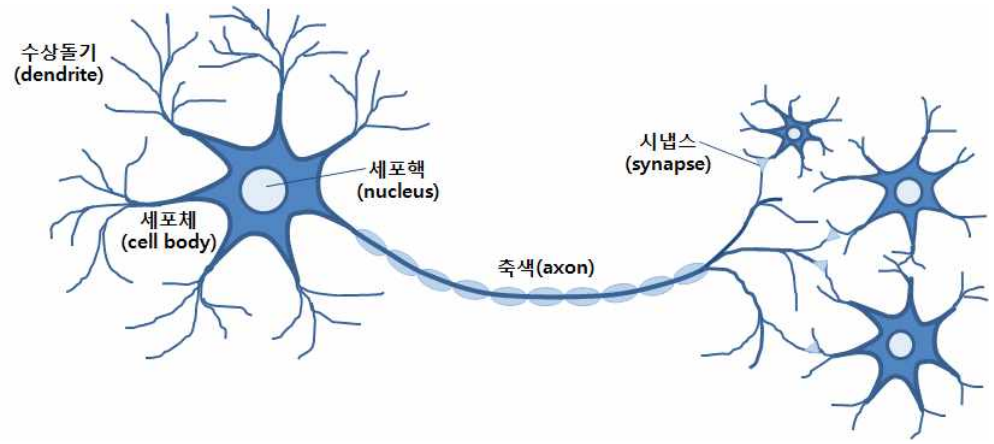
[그림 11-1] 신경세포들의 회로망 (저작권 고려하여 교환 필요)

인공신경망(Artificial Neural Networks)은 이러한 학습능력과 적응능력을 기계에 부여하기위해 인간의 뇌에서 일어나는 정보처리 방식을 모델링하는 방법이다. 이를 위해서는, 먼저 하나하나의 신경세포에 대한 모델링이 수행되어야 하고, 이어서 신경세포들이 연결되어 만드는 신경망(네트워크)의 구조에 대한 모델링이 필요하다. 마지막으로 가장 중요한 것은 신경세포들의 연결 강도를 스스로 조절할 수 있는 학습 메커니즘에 대한 모델링이 필요하다. 이 장에서는 이와 관련된 초기 연구에서부터 현재 가장 널리 사용되는 신경망인 다층 퍼셉트론과 자기조직화 신경망에 대하여 살펴볼 것이다.

11.1.2 생물학적 신경망

먼저 뇌의 가장 기본 구조인 생물학적인 신경세포의 구조와 세포간의 연결에 대하여 알아보

고, 이것을 인공적으로 모델링하는 방법에 대하여 살펴보겠다. [그림 11-2]에 전형적인 신경세포의 모형을 그림으로 나타내었다. 이 그림에서 보듯이 신경세포는 세포체, 수상돌기, 축색, 그리고 연결 부분에 해당하는 시냅스로 구성되어 있다. 신경세포의 각 구성 요소들을 기능의 관점에서 살펴보면, 수상돌기(dendrite)는 입력부분으로, 다른 여러 신경세포로부터 입력을 받아들이는 역할을 한다. 세포체(cell body)는 계산 부분으로, 다른 세포들로부터 입력된 정보를 정해진 방식에 따라 처리하는 부분이다. 축색(axon)은 출력부분으로, 처리된 정보를 다른 신경세포로 전달하기 위해 지나가는 경로이다. 마지막으로 신경세포들은 시냅스(synapse)를 통해 연결되어 있으며, 이 부분에서 실제 세포간의 연결 강도에 의존하여 정보 전달이 이루어진다.

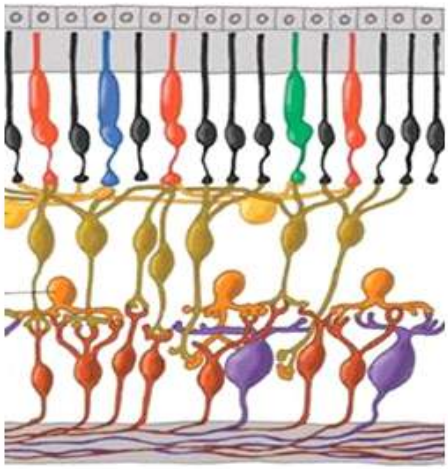


[그림 11-2] 신경세포

외부로부터 혹은 다른 신경세포들로부터 주어지는 자극(전기적인 신호)은 세포체로 흘러들어오고, 이들이 모두 합해져 일정한 임계치 이상이 되면 세포는 활성화되어 활동전위(스파이크)를 발생시킨다. 이는 축색을 통해 이동하여 시냅스를 통해 다른 신경세포로 전달된다. 하나의 신경세포에서 발생한 전기적 신호가 다른 신경세포로 전달됨에 있어서 가장 중요한 특징은 두 신경세포 간에는 시냅스를 통한 가중 연결(weighted connection)이 이루어진다는 점이다. 즉, 하나의 신경세포의 출력이 그대로 다음 신경세포로 전달되는 것이 아니라, 두 신경세포가 어떤 방식으로 어느 정도의 강도로 연결되어 있느냐에 따라 정보가 전달되는 양이 달라진다. 이 연결 강도를 가중치(weight)라고 하며, 특히 양의 가중치를 가지는 경우를 흥분성(excitatory) 연결, 음의 가중치를 가지는 경우를 억제성(inhibitory) 연결이라고 한다. 흥분성 연결은 정보를 받아들이는 신경세포의 활성화 정도를 증가시키는 역할을 하게 되고, 억제성 연결은 반대로 신경세포의 활성화 정도를 감소시키는 역할을 하게 된다. 이 가중치의 개념은 신경망 학습에 있어서 가장 핵심이 되는 것으로 다음 절에서 이를 조정하는 학습에 대하여 살펴볼 것이다.

지금까지는 두 세포간의 연결에 대하여 간단히 논의하였으나, 실제로 인간의 뇌는 많은 수의 신경세포들이 복잡한 구조를 가지고 연결되어 있어서 그 구조에 대해서도 알아볼 필요가 있다. 생물체의 신경 시스템에서 가장 대표적으로 관찰되는 연결 구조는 계층 연결(layered connection)이다. [그림 11-3]에는 인간의 망막에서 관찰되는 세포들 간의 계층적 구조를 보여주고 있다. 그림에서 가장 윗부분에 있는 세포에서 빛 자극을 받아 반응하면 그 정보가

아래쪽을 전달되어 가장 아래 부분의 신경절세포로 모아진 후 시신경섬유를 통해 뇌로 전달된다. 이와 같이 세포들의 기능에 따라 나누어진 층상 구조는 인공신경망을 개발함에 있어서 기본적인 모델이 되었다.



[그림 11-3] 망막의 신경세포들의 계층 구조(저작권 고려하여 교환 필요)

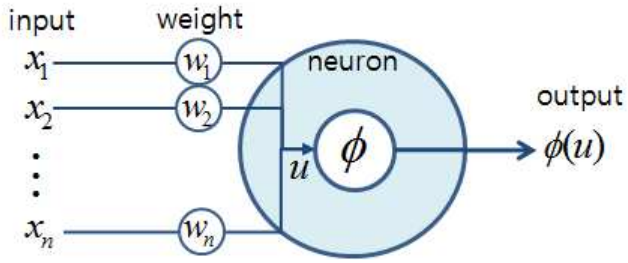
지금까지 살펴본 생물학적 신경세포를 기능적인 관점에서 정리하면, 다음과 같은 특성으로 요약할 수 있다.

- 신경세포는 정보를 받아들이는 입력 기능을 하는 수상돌기, 받아들인 정보를 합하여 연산 기능을 수행하는 세포체, 그리고 생성된 정보를 전달하는 출력 기능을 하는 축색으로 나눌 수 있다.
- 세포체 내부에서 일어나는 연산 기능이 하나의 세포를 특징짓는 것으로, 기본적으로는 들어온 입력의 합이 일정 수준에 달하면 활동전위(스파이크)를 발생시키는 특성을 가진다.
- 두 신경세포는 시냅스를 통하여 연결되고, 연결 강도에 따라 정보의 전달 방식과 크기가 달라진다.
- 신경세포들은 계층구조와 같은 체계적인 연결 구조를 가지고 있다.

이상과 같은 신경세포의 핵심 특성을 모방하면 인공 신경세포를 모델링 할 수 있다. 다음절에서 어떻게 이러한 특성들을 수학적으로 모델링하였는지 알아보겠다.

11.1.3 인공신경망의 구성요소

앞 절에서 살펴본 생물학적 신경망에 대한 고찰을 바탕으로 인공신경망을 모델링해 보겠다. 인공신경망은 크게 세 가지 요소, 즉 인공 신경세포(뉴런), 연결 구조, 학습 규칙에 의해 정의되므로, 각 요소별로 인공신경망이 어떻게 정의될 수 있는지 알아보겠다.

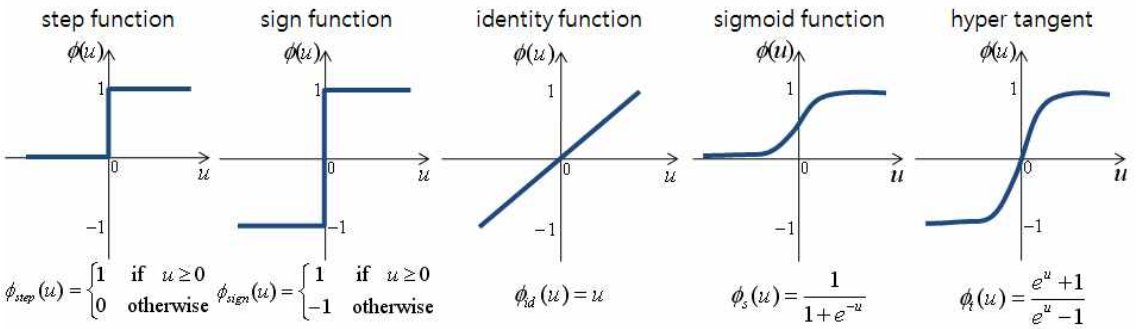


[그림 11-4] 인공 신경세포의 구조

먼저 생물학적 뉴런의 구조와 기능을 모방하여 인공 뉴런을 정의한다. [그림 11-4]에 간단한 인공 뉴런의 구조를 나타내었다. 이를 식으로 표현하면 다음과 같다.

$$u = \sum_{i=1}^{n} w_i x_i, \qquad \phi(u) = \begin{cases} 1 & \text{if } u \geq \theta \\ 0 & \text{otherwise} \end{cases} \qquad \text{[식 11-1]}$$

하나의 인공 뉴런은 n개의 입력($x_1, x_2, \ldots, x_n$)에 대해서 연결 강도($w_1, w_2, \ldots, w_n$)에 따른 가중치를 곱하여 받아들인 값을 모두 합하여 가중합($u = \boldsymbol{w}^T\boldsymbol{x} = \Sigma w_i x_i$)을 계산한다. 가중합은 다시 활성화 함수($\phi$)를 통하여 다음 뉴런으로 전달될 출력이 결정된다.
하나의 뉴런의 특성을 결정하는 것은 <활성화 함수(activation function)>인데, 가장 기본적으로는 [식 11-1]과 같이 정의된다. 이는 앞서 살펴본 생물학적 뉴런이 입력 자극이 어느 정도 수준(임계치 θ) 이상이 될 때만 전기적 방전(스파이크)을 일으켜 활성화된다는 사실을 간단히 수학적으로 표현한 것이다. 그러나 반드시 이 함수를 사용할 필요는 없으며, 활성화함수를 적절히 정의해 줌으로써 원하는 특성을 가진 인공신경망 모델을 개발할 수 있다.



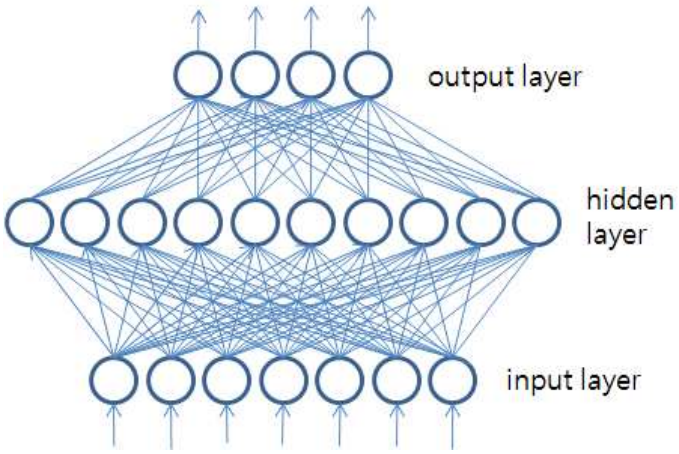
[그림 11-5] 다양한 활성화 함수의 정의와 형태

[그림 11-5]에는 자주 사용되는 활성화 함수들을 그래프와 함께 소개하였다. 계단 함수(step function)가 [식 11-1]에서 정의된 가장 기본적인 형태이며, 출력값을 0과 1이 아닌 -1과 1로 바꾼 부호 함수(sign function)도 그와 유사하다. 이외에 가중합을 그대로 출력으로 내보내는 항등 함수(identity function)도 사용된다. 가장 널리 사용되는 것은 [그림 11-5]의 왼쪽에서 네 번째와 다섯 번째의 시그모이드 함수(sigmoid function)와 하이퍼탄젠트 함수(hyper tangent function)로, 이 두 함수는 계단 함수나 부호 함수와는 달리 미분가능하다는 장점을

가지고 있으면서도 출력값이 0에서 1사이(혹은 -1에서 1사이)로 제한되는 특성을 가진다. 또한 함수의 곡선 형태가 파라미터의 값에 따라 계단 함수에서 선형 함수에 이르기까지 자유롭게 근사가 가능하도록 조정할 수 있는 장점도 가진다. 따라서 이 두 함수가 가장 널리 사용되고 있으며, 이에 대해서는 11.2절에서 보다 자세히 알아보겠다.

활성화 함수를 정의함으로써 하나의 신경세포에 대한 모델이 정립되면, 다음으로 신경망의 연결 구조에 대해 고려한다. 가장 대표적인 연결 구조는 [그림 11-3]과 유사하게 층상 구조를 가지는 경우이다. [그림 11-6]에서 보이는 바와 같이 뉴런들이 층별로 그룹을 이룬다. 같은 층 안에서는 뉴런간의 연결이 존재하지 않고, 이웃한 두 층 사이에서는 모든 뉴런들이 연결을 가진다. 첫 번째 층이 외부로부터 입력을 받아들이는 입력층(input layer)이고, 마지막 층이 외부에 출력을 내는 출력층(output layer)이 된다. 가운데에 있는 층은 외부와는 정보를 교환하지 않으며 다른 신경세포들과만 입출력을 주고받는 은닉층(hidden layer)이다. 앞서 6장에서 살펴본 퍼셉트론은 이와 같은 구조를 가지는 경우로, 입력층과 출력층으로만 구성되어 있는 단층 신경망(single-layer network)이다. 다음 절에서 살펴볼 신경망은 입력층과 출력층 사이에 1개 이상의 은닉층을 가지는 구조로, 이와 같은 신경망을 다층 신경망(multi-layer network)이라고 한다.

층상구조의 신경망에서 정보의 흐름은 일반적으로 입력층에서 출력층으로 한 방향으로만 흐른다. 즉, 입력층은 첫 번째 은닉층으로 입력값을 주기만 하고, 다시 그 은닉층은 계산된 출력값을 입력층으로 되돌리지는 않고 다음 층의 입력으로만 제공한다. 이와 같은 정보흐름을 가진 신경망을 전방향 신경망이라고 한다. 그림에서는 화살표로 정보흐름의 방향을 표시하였다. 전방향 다층 신경망이 가장 널리 사용되는 구조이기는 하지만, 다양한 변형들 역시 존재한다. 출력층의 신호를 다시 입력층으로 되돌리는 회귀 신경망(recurrent neural networks)이나 같은 층 내에서 상호 연결을 허용하는 신경망도 존재하며, 층상 구조를 이루지 않고 모든 뉴런들이 서로서로 연결되어 있는 구조를 가지는 신경망도 존재한다.



[그림 11-6] 층상 구조로 연결된 신경망 구조

신경망의 구조가 결정되면 학습 방법에 대하여 생각해 보아야 한다. 신경망에서의 학습이란 시냅스의 연결 강도(가중치)를 변화시키는 것으로 볼 수 있다. 가장 기본적인 방법은 헤브(Hebb)에 의해 개발된 헤브의 학습 규칙(Hebbian Learning Rule)으로, 연결된 두 신경세포가 동시에 활성화되면 가중치를 증가시키는 방향으로 학습한다. 이후 보다 발전된 형태로 6

장에서 살펴본 퍼셉트론 학습과 델타 학습, 그리고 이를 발전시킨 오류역전파 학습 알고리즘 등이 존재하며, 11.3절에서 살펴볼 자기조직화 학습법도 존재한다. 이에 대해서는 다음 절에서 자세히 알아보겠다.

11.2 신경망 분류기

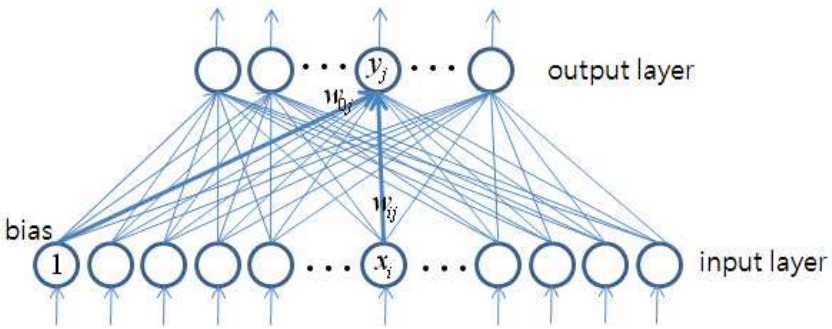
11.2.1 M-P 뉴런과 퍼셉트론

신경망 연구의 첫 시도는 1943년에 MaCulloch와 Pitts에 의해 제안된 M-P 뉴런에서 시작된다. M-P 뉴런은 신경세포를 모방하여 논리 함수를 구현하는 모델을 만들기 위해 제안되었다. M-P 뉴런의 구조는 11.1절에서 소개한 기본적인 인공신경망과 같으며, 활성화 함수로 계단 함수를 적용함으로써 이진 출력을 내도록 고안하였다. 입출력 계산식은 [식 11-1]과 거의 유사하나, [그림 11-5]에서 정의한 계단 함수를 사용하여 다시 표현하면 [식 11-2]와 같이 주어진다.

$$y_j = \phi_{step}\left(\sum_{i=1}^{n} w_{ij}\, x_i + w_{0j}\right) \qquad \text{[식 11-2]}$$

여기서 y_j는 j번째 출력 노드의 출력값을 나타내고, w_{0j}는 [식 11-1]의 θ와 동일한 임계치 역할을 하는 것으로, 바이어스(bias)라고 부른다. 즉, 입력의 가중합이 $-w_0$보다 크지 못하면 0의 출력을 낸다.

단일 신경세포에 대한 모델인 M-P 뉴런들을 여러 개 결합하여 네트워크 형태를 갖춘 신경망을 제안한 것이 1958년에 소개된 Rosenblatt의 퍼셉트론이다. 퍼셉트론에 관해서는 6장에서 수식으로 설명하였지만, 여기서는 신경세포들의 연결을 중심으로 간략히 다시 기술하겠다. 신경세포들의 연결을 통하여 패턴인식을 수행하는 최초의 기계인 퍼셉트론은 단층 전방향 신경망 (Single-layer Feed Forward Network)으로 [그림 11-6]과 같은 구조를 가지고 있다.

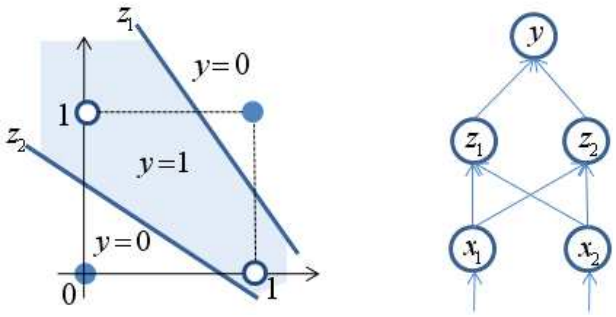


[그림 11-7] 퍼셉트론의 구조

퍼셉트론이 처음 발표되었을 당시에 주목을 받았던 가장 큰 이유는 원하는 패턴을 학습할 수 있는 학습능력(가중치 조절 규칙)을 가지고 있었기 때문이다. 퍼셉트론의 학습 규칙에 대하여서는 6장에서 살펴보았지만, 여기서는 신경망의 관점에서 간단히 다시 기술하겠다. 학습의 기본 규칙은, "만일 어떤 입력 뉴런의 활성이 출력 뉴런이 잘못된 결과를 내는데 공헌하였다면, 두 신경세포 간의 연결 가중치를 그것에 비례하여 조절해 주어야 한다."는 것이다. 이것을 수식으로 표현하면 다음과 같다.

$$w_{ij}^{(\tau+1)} = w_{ij}^{(\tau)} + \eta(t_j - y_j)x_i \qquad \text{[식 11-4]}$$

여기서 t_j는 j번째 출력 뉴런이 내야할 목표 출력값이며, η는 학습률이다. 결국 목표 출력값과 현재 출력값의 차이에 입력값을 곱하여 이에 비례하는 작은 값을 이용하여 가중치를 수정한다. 이렇게 함으로써 목적 패턴을 사용하여 신경망의 학습을 인위적으로 제어할 수 있다. 이는 6절에서 살펴본 것과 일치되는 내용이며, 목표 출력값을 사용하므로 퍼셉트론은 교사학습을 하는 신경망이다.



[그림 11-8] XOR 문제를 위한 결정경계와 신경망 구조

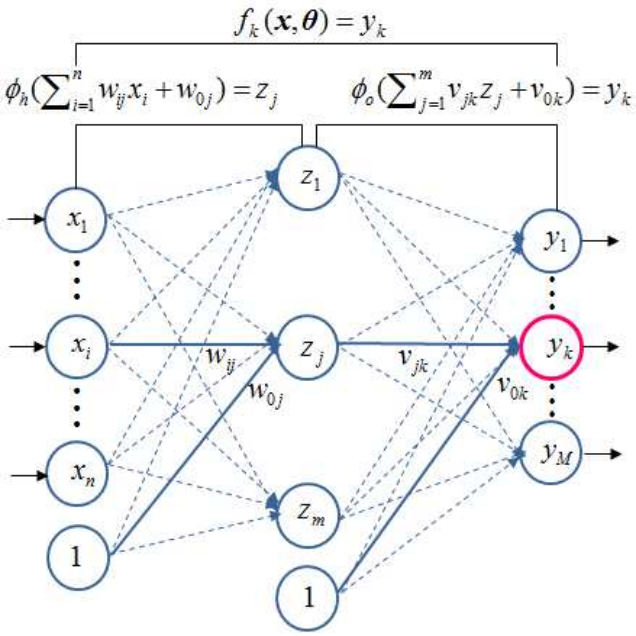
퍼셉트론을 이용한 패턴 분류에 대해서는 6장에서 이미 살펴본 바 있다. 퍼셉트론 분류기의 가장 큰 약점은 선형 판별함수를 가진다는 점이다. 저명한 인공지능 학자 Minsky와 Papert는 퍼셉트론이 가지는 이러한 한계에 대해 XOR 문제를 가지고 수학적으로 증명하였다. XOR 문제는 두 개의 입력 노드와 하나의 출력 노드를 가지는 퍼셉트론이 XOR 논리 함수를 표현하도록 학습하는 것이다. 그런데 2장에서 살펴본 바와 같이 이 퍼셉트론이 만드는 판별함수는 2차원 공간상의 하나의 직선으로 나타나므로, [그림 11-8]에서 보이는 것처럼 XOR와 같은 출력을 내도록 결정경계를 만드는 것은 불가능하다. 이 문제를 해결하기 위해서는 [그림 11-8]의 왼쪽과 같이 두 개의 직선을 결합하여 사용해야 하며, 이것은 [그림 11-8]의 오른쪽에 보이는 것과 같이 은닉층을 가지는 신경망 구조를 만들어야 한다. 이와 같이 비선형 결정경계를 만들기 위하여 은닉층을 추가한 신경망을 <다층 퍼셉트론(Multi-layer perceptron, MLP)>이라고 한다.

11.2.2 다층 퍼셉트론

다층 퍼셉트론은 [그림 11-9]와 같이 1개 혹은 그 이상의 은닉층을 가지는 다층 전방향 신

경망의 구조를 가지고 있다. 그림에서는 n개의 입력 뉴런을 가지는 입력층, m개의 은닉 뉴런을 가지는 은닉층, 그리고 M개의 출력 뉴런을 가지는 출력층으로 구성된 다층 퍼셉트론을 나타내었다. 이와 같이 구성된 다층 퍼셉트론이 어떤 결정경계를 가지게 되는지 알아보기 위해, 그 입출력 매핑 함수를 수학적으로 표현해 보겠다.

이를 위해 먼저 [그림 11-9]에 표시된 기호들의 정의를 알아두어야 할 것이다. 먼저 입력 노드의 값을 $x_i(i=1...n)$, 은닉 노드의 출력값을 $z_j(j=1...m)$, 출력 노드의 출력값을 $y_k(k=1...M)$로 두고 입력 노드 x_i에서 은닉 노드 z_j로의 연결 가중치를 w_{ij}, 은닉 노드 z_j에서 출력 노드 y_k로의 연결 가중치를 v_{jk}라고 두었다. 또한 w_{0j}와 v_{0k}는 각각 은닉 노드와 출력 노드로의 바이어스 입력 가중치를 나타낸다. 다층 퍼셉트론의 입력 뉴런을 통해 주어지는 입력값은 모두 n개의 실수값이 되므로, 이를 n차원 벡터 $\boldsymbol{x}=[x_1, x_2, \ldots, x_i, \ldots, x_n]^T$ 로 나타낼 수 있다. 마찬가지로 출력값은 M차원의 벡터 $\boldsymbol{y}=[y_1, y_2, \ldots, y_k, \ldots, y_M]^T$ 로 나타내고, 모든 가중치를 묶어서 하나의 파라미터 $\boldsymbol{\theta}$로 나타내면 입력 $\boldsymbol{x}$가 주어졌을 때 k번째 출력 노드의 출력 y_k은 함수 $f_k(\boldsymbol{x},\boldsymbol{\theta})$ 로 나타낼 수 있다.



[그림 11-9] 다층 퍼셉트론의 구조와 함수식

계속해서 함수 $f_k(\boldsymbol{x},\boldsymbol{\theta})$의 구체적인 형태를 알아보기 위해서는 먼저 은닉 뉴런과 출력 뉴런의 활성화 함수에 대하여 알아보아야 할 것이다. 다층 퍼셉트론의 경우는 퍼셉트론과 달리 연속한 실수값을 다룰 수 있는 형태로 정의되었기 때문에 활성화 함수도 계단 함수와 같은 불연속 함수가 아닌 실수값을 가지는 연속 함수로 정의된다. 그 중 특히 널리 사용되는 것이 [그림 11-5]에서 소개한 시그모이드 함수와 하이퍼탄젠트 함수이다. 은닉 뉴런과 출력 뉴런에 대하여 각각 서로 다른 활성화 함수를 적용할 수도 있는데 은닉 뉴런에는 반드시 시그모이드나 하이퍼탄젠트와 같은 비선형 함수를 사용하는 반면, 출력 뉴런의 경우에는 선형 가중합을 그대로 출력값으로 주는 항등 함수를 사용하기도 한다. 은닉 뉴런의 활성화 함수

를 ϕ_h, 출력 뉴런의 활성화 함수를 ϕ_o라고 둔다. 이를 바탕으로 $f_k(\boldsymbol{x},\boldsymbol{\theta})$ 을 구체적으로 정의하면 다음 식과 같이 쓸 수 있다.

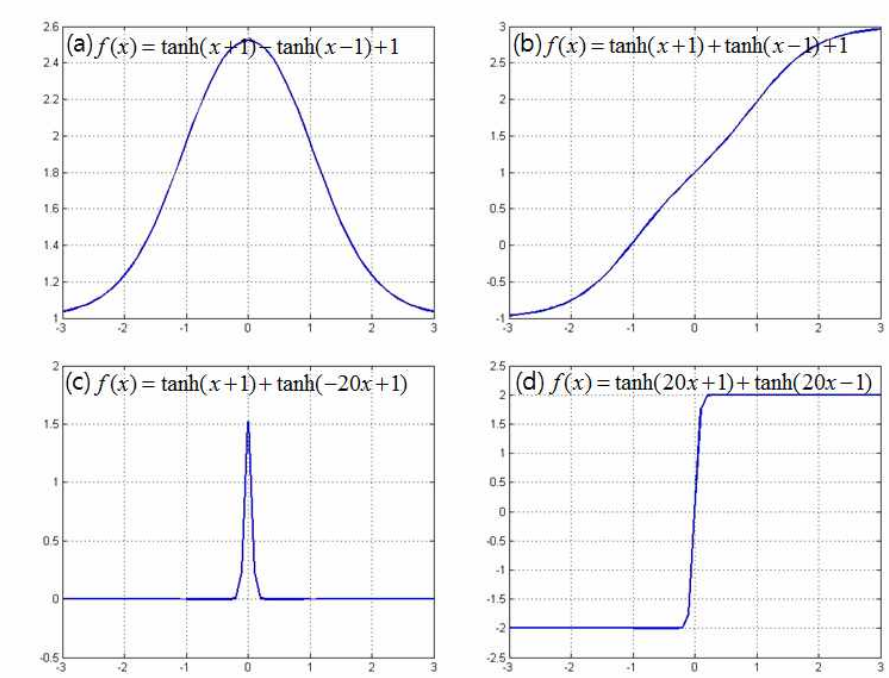
$$y_k = f_k(\boldsymbol{x},\boldsymbol{\theta}) = \phi_o\left(\sum_{j=1}^{m} v_{jk} z_j + v_{0k}\right) = \phi_o\left(\sum_{j=1}^{m} v_{jk}\phi_h\left(\sum_{i=1}^{n} w_{ij}x_i + w_{0j}\right) + v_{0k}\right) \qquad \text{[식 11-5]}$$

여기서 k번째 출력 노드로 들어가는 가중합을 u_k^o, j번째 은닉 노드로 들어가는 가중합을 u_j^h, 그 출력값을 z_j 라고 하면 [식 11-5]를 [식 11-6]과 같이 몇 개의 식으로 분리하여 쓸 수 있다. 이 식은 추후 다층 퍼셉트론의 학습 알고리즘을 유도할 때 사용될 것이며, 또한 11.4절의 프로그램을 이해하는데 있어서도 도움을 줄 것이다.

$$y_k = \phi_o(u_k^o), \quad u_k^o = \sum_{j=1}^{m} v_{jk} z_j + v_{0k}, \quad z_j = \phi_h(u_k^h), \quad u_j^h = \sum_{i=1}^{n} w_{ij}x_i + w_{0j} \qquad \text{[식 11-6]}$$

지금까지 수식으로 알아본 다층 퍼셉트론의 함수가 어떤 특성을 가지고 있는지 간단한 모델을 이용하여 알아보자. 하나의 입력 뉴런과 두 개의 은닉 뉴런, 그리고 하나의 출력 뉴런을 가진 다층 퍼셉트론을 생각한다. 은닉 뉴런의 활성화 함수를 하이퍼탄젠트 함수로 두고 출력 뉴런의 활성화 함수를 항등 함수로 두면 이 다층 퍼셉트론에 의해 정의되는 함수는 다음과 같이 쓸 수 있다.

$$\begin{aligned} y &= v_{11}z_1 + v_{21}z_2 + v_{01} \\ &= v_{11}\tanh(w_{11}x_1 + w_{01}) + v_{21}\tanh(w_{12}x_1 + w_{02}) + v_{01} \end{aligned} \qquad \text{[식 11-7]}$$



[그림 11-10] 두 은닉 뉴런의 가중합에 의해 만들어지는 다양한 함수 형태

[그림 11-10]에서는 가중치의 변화에 따른 y의 여러 형태를 나타내었다. 그림에서 알 수 있듯이 가중치에 따라 매우 다양한 형태의 함수가 다층 퍼셉트론에 의해 표현가능하다. 이론적으로는 하나의 은닉층을 가지는 다층 퍼셉트론으로 어떠한 연속 함수도 원하는 오차만큼 가깝게 근사할 수 있음이 증명되었다. 따라서 다층 퍼셉트론을 이용하면 복잡한 비선형 결정경계를 가진 분류 문제도 성공적으로 해결할 수 있을 것이다. 다음 절에서는 원하는 함수(결정경계)를 근사하기 위해 가중치 파라미터를 조정하는 학습 방법에 대하여 알아보겠다.

11.2.3 다층 퍼셉트론의 학습

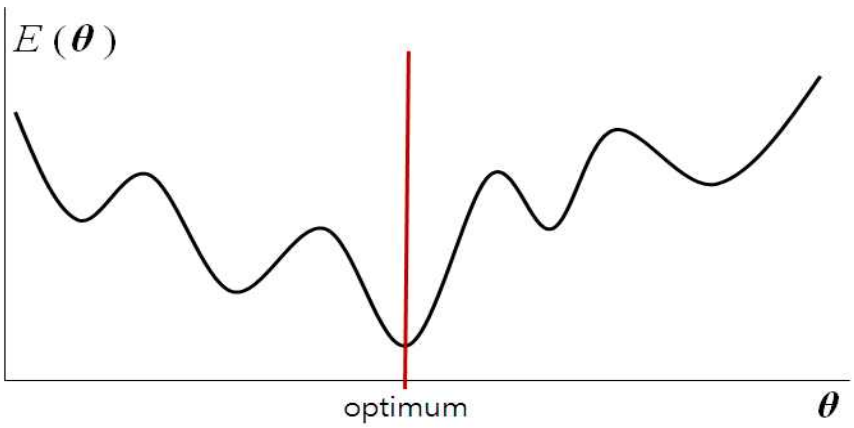
다층 퍼셉트론의 학습은 퍼셉트론과 마찬가지로 교사학습을 수행한다. 교사학습을 수행하기 위해서는 먼저 다층 퍼셉트론이 학습을 통해서 찾아야 하는 목표 출력값이 주어지고, 이 목표 출력값과 현재의 다층 퍼셉트론이 내는 출력값의 차이를 이용한 오차 함수가 정의되어야 한다. 따라서 학습 데이터는 입출력의 순서쌍 $\{(\boldsymbol{x}_i, \boldsymbol{t}_i)\}(i=1,...,N)$으로 주어진다. 학습 데이터 전체 집합 X에 대한 오차는 다음과 같이 평균제곱오차로 정의할 수 있다.

$$E(X,\boldsymbol{\theta}) = \frac{1}{2N}\sum_{i=1}^{N} \| \boldsymbol{t}_i - \boldsymbol{f}(\boldsymbol{x}_i,\boldsymbol{\theta}) \|^2 \qquad \text{[식 11-8]}$$

식의 앞부분에 있는 계수 1/2는 추후 학습식의 유도가 간단해지도록 추가한 값이다. 이 식에서는 출력 뉴런의 수가 복수개인 경우를 생각하므로 목표 출력값과 신경망 출력값이 모두

벡터로 주어짐에 유의하자. 이후에서는 계산을 간단히 하기 위하여 k번째 출력 뉴런 하나에 대하여 식을 유도할 것이다.

이 함수값은 데이터 집합 X와 파라미터 $\boldsymbol{\theta}$가 주어지면 하나의 값으로 정해진다. 그런데 데이터 집합은 외부에서 주어지는 값이며, 우리가 최적화해야하는 대상은 파라미터 $\boldsymbol{\theta}$이므로, 일반적으로는 $E(\boldsymbol{\theta})$와 같이 나타낸다. 이 함수가 파라미터 공간상에서 어떤 형태를 가지는지 생각해보자. 다층 퍼셉트론에 의해 정의되는 함수 $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta})$는 앞 절에서 살펴본 바와 같이 복잡한 비선형 함수이므로, 결과적으로 오차 함수 $E(\boldsymbol{\theta})$도 [그림 11-11]에 나타난 것과 같이 매우 복잡한 형태의 비선형 함수가 된다. 따라서 6장에서 알아본 퍼셉트론의 최소제곱법과 같이 간단한 수식 계산에 의해 바로 최적해를 찾는 것은 불가능하다. 이를 해결하기 위한 새로운 학습 알고리즘이 필요한데, 이렇게 비선형적인 함수의 최소값을 찾아가는 반복적 알고리즘으로 <기울기 강하 학습법(Gradient Descent Learning Method)>이 있다. 기울기 강하 학습법은 최적화 분야에서 오래전부터 사용되어 오던 방법이나, 이를 다층 퍼셉트론에 적용하여 실제 구현 가능한 알고리즘화한 것이 잘 알려진 <오류역전파 학습 알고리즘(error backpropagation learning algorithm)>이다. 따라서 이 책에서는 먼저 기울기 강하 학습법의 개념에서 접근하여 오류역전파 알고리즘을 유도해 보겠다.

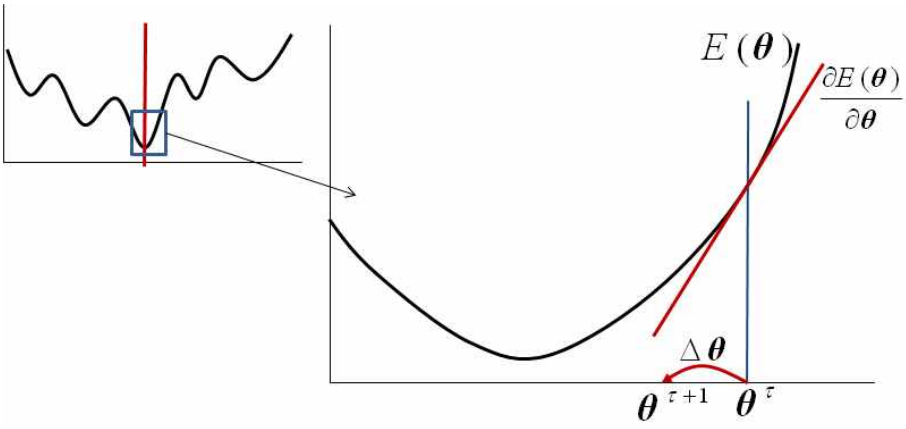


[그림 11-11] 다층 퍼셉트론이 만드는 오차 함수의 일반적 형태

[그림 11-12]에 기울기 강하 학습법의 기본적인 개념을 나타내었다. 기울기 강하 학습법은 10장에서 살펴본 EM 알고리즘과 같은 반복적 알고리즘에 해당한다. 즉, 한 번에 전체 탐색 공간에 대한 최적해를 얻는 것이 아니라, 어떤 시점 τ에서의 파라미터 $\boldsymbol{\theta}^{(\tau)}$에서 그 주변의 정보를 바탕으로 오차값을 감소시킬 수 있는 방향 $\Delta\boldsymbol{\theta}$를 찾아 $\boldsymbol{\theta}^{(\tau+1)}$을 얻는 과정을 반복하여 조금씩 해를 찾아 가는 방법이다. 전체 탐색 공간에 대한 오차 함수가 [그림 11-11]과 같이 복잡한 비선형 함수라 하더라도, 현재 파라미터 $\boldsymbol{\theta}^{(\tau)}$의 주변에서는 [그림 11-12]와 같이 간단한 구조를 가진다고 가정할 수 있다. 이러한 가정 하에서 오차값을 감소시키는 방향은 오차 함수를 파라미터로 편미분해서 얻을 감소기울기에 비례 파라진다고 얻을 수 있다. 즉, 현 시점에서의 기울기를 살펴보아 내리막이 되는 방향(오차값이 줄어드는 방향)다고 파라미터를 수정한다. 있다. 이 정보는 어디까지나 현재 파라미터의 주변에서만 유용한 값이므로, 한 번에 큰 폭으로 수정하는 것은 불가능하며, 조금씩 수정해 나가야 할 것이다. 이러한 개념을 바탕으로 한 기울기 강하 학습 알고리즘의 일반식은 다음과 같이 쓸 수 있다.

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} + \Delta\boldsymbol{\theta}^{(\tau)} = \boldsymbol{\theta}^{(\tau)} - \eta\frac{\partial E(\boldsymbol{\theta}^{(\tau)})}{\partial \boldsymbol{\theta}} \qquad \text{[식 11-9]}$$

이 때 상수값 η는 학습의 속도를 조정하는 작은 실수 값으로, <학습률(learning rate)>이라고 부른다. η값이 크면 학습 속도가 빨라지나 학습이 불안정해지고, 반대로 그 값이 작으면 안정적으로 학습 가능하나 속도가 느려지는 문제가 있다. 따라서 문제에 적합한 값을 정하여 학습 속도를 적절히 조정해 주어야 한다.



[그림 11-12] 기울기 강하 학습법

이제 기울기 강하 학습법을 다층 퍼셉트론에 적용하여 구체적인 학습 알고리즘을 찾아보자. 다층 퍼셉트론의 학습에서는 일반적으로 학습 데이터 전체를 한꺼번에 사용하는 대신 한 번에 하나의 데이터만을 사용하여 학습하는 온라인 학습을 수행하므로, 여기서도 [식 11-8]을 사용하는 대신 하나의 데이터에 대한 오차 함수 $E(x,\boldsymbol{\theta})$를 사용한다. 특히, 각 출력 뉴런들을 중심으로 계산이 진행되므로, 여기서는 논의를 간단히 하기 위하여 k번째 출력 노드 y_k를 중심으로 현재 들어온 입력 x에 대한 오차 함수를 다음과 같이 정의한다.

$$E(\boldsymbol{x},\boldsymbol{\theta}) = \frac{1}{2}(t_k - y_k)^2 = \frac{1}{2}(t_k - f_k(\boldsymbol{x},\boldsymbol{\theta}))^2 \qquad \text{[식 11-10]}$$

먼저 주어진 입력에 대한 신경망의 출력값 y_k가 [식 11-5]로부터 정해진다고 하자. 우리가 학습을 통해 수정해주어야 하는 파라미터는 v_{jk} $(j=0\ldots m, k=1\ldots M)$와 w_{ij} $(i=0\ldots n, j=1\ldots m)$이다. 파라미터의 수정식은 오차 함수 [식 11-10]을 각 파라미터로 편미분함으로써 얻을 수 있으므로, 계산의 핵심은 각 파라미터에 대한 오차 함수의 편미분값이 된다. 먼저 은닉층에서 출력층으로의 파라미터 v_{jk}에 대한 오차 함수의 편미분값을 계산하면 다음 식과 같이 간단히 얻을 수 있다.

$$\frac{\partial E}{\partial v_{jk}} = \frac{\partial E}{\partial u_k^o}\frac{\partial u_k^o}{\partial v_{jk}} = \delta_k z_j, \qquad \delta_k = -\phi_o'(u_k^o)(t_k - y_k) \qquad \text{[식 11-11]}$$

이 식에서 z_j는 j번째 은닉 뉴런의 출력값으로 [식 11-6]에 정의되어 있고(단, z_0는 바이어스 입력값으로 항상 1이다.), δ_k값은 k번째 출력 뉴런의 출력값과 목표 출력값과의 차이에 비례하는 값으로 이 δ_k는 추후 입력 뉴런에서 은닉 뉴런으로의 파라미터에 대한 편미분 계산에도 다시 사용될 것이다. 비례상수에 해당하는 $\phi_o'(u_k^o)$은 출력 뉴런의 활성화 함수의 미분값으로 활성화 함수의 형태에 따라 달라진다. 일반적으로 많이 쓰이는 시그모이드 함수의 경우 $\phi_o'(u_k^o)=(1-z_k)z_k$, 하이퍼탄젠트 함수의 경우에는 $\phi_o'(u_k^o)=(1-z_k)(1+z_k)$ 로 간단히 계산된다.

이어서 입력 뉴런에서 은닉 뉴런으로의 파라미터 w_{ij}에 대한 편미분도 계산해 보자. w_{ij}는 직접적으로 오차 함수 계산에 적용되는 것이 아니라 출력 노드를 한 번 더 거쳐서 계산이 이루어지므로 바로 편미분하는 것이 어렵다. 따라서 다음 식과 같이 체인 규칙으로 풀어서 $\partial u_j^h/\partial w_{ij}$와 $\partial E/\partial u_j^h$로 나누어 접근해 보겠다.

$$\frac{\partial E}{\partial w_{ij}}=\frac{\partial E}{\partial u_j^h}\frac{\partial u_j^h}{\partial w_{ij}}=\delta_j x_i \qquad \text{[식 11-12]}$$

이 식에서 $\partial u_j^h/\partial w_{ij}$는 쉽게 z_j로 얻어지나, δ_j로 나타낸 $\partial E/\partial u_j^h$의 계산이 조금 복잡하다. $\partial E/\partial u_j^h$는 j번째 은닉 뉴런으로의 가중합에 대해 오차 함수를 미분한 값인데, 이 역시 바로 계산은 불가능하고, 이를 체인 규칙으로 풀어서 표현하면 다음과 같다.

$$\delta_j=\frac{\partial E}{\partial u_j^h}=\sum_{k=1}^{M}\frac{\partial E}{\partial u_k^o}\frac{\partial u_k^o}{\partial u_j^h} \qquad \text{[식 11-13]}$$

이렇게 풀어서 표현하면 $\partial E/\partial u_k^o$는 [식 11-11]에서 계산된 δ_k에 해당되므로 쉽게 얻을 수 있다. 이어서 $\partial u_k^o/\partial u_j^h$를 계산해야 하는데, [식 11-14]의 관계식을 이용하면 [식 11-15]를 얻을 수 있다.

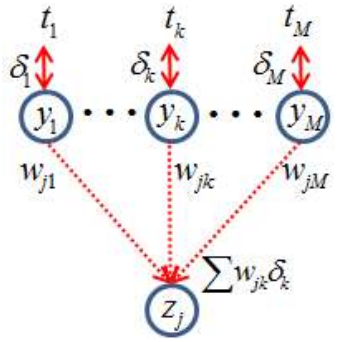
$$u_k^o=\sum_{j=1}^{m}v_{jk}z_j=\sum_{j=1}^{m}v_{jk}\phi_h(u_j^h) \qquad \text{[식 11-14]}$$

$$\frac{\partial u_k^o}{\partial u_j^h}=\phi_h'(u_j^h)v_{jk} \qquad \text{[식 11-15]}$$

이상에서 계산된 내용을 모두 정리하면 다음 식을 얻게 된다.

$$\delta_j=\frac{\partial E}{\partial u_j^h}=\sum_{k=1}^{M}\frac{\partial E}{\partial u_k^o}\frac{\partial u_k^o}{\partial u_j^h}=\phi_h'(u_j^h)\sum_{k=1}^{M}v_{jk}\delta_k \qquad \text{[식 11-16]}$$

여기서 이 식의 직관적 의미를 한 번 생각해 보자. [그림 11-13]에 이 식에서 이루어지는 연산을 개념적으로 나타내었다. j번째 은닉 뉴런으로의 가중합에 대해 오차 함수를 미분한 값 $\partial E/\partial u_j^h$는 결국 j번째 은닉 뉴런이 출력값의 오차에 어느 정도 영향을 미치고 있는지를 의미한다. 그런데 u_j^h는 출력 뉴런들의 가중합 $u_k^o(k=1\cdots M)$를 계산하는데 모두 쓰이게 되므로 각각의 출력 뉴런이 오차에 미치는 영향, 즉 δ_k에 가중치 w_{jk}를 곱한 값을 모두 합함으로써 얻을 수 있다. 결국 출력 뉴런의 오차가 은닉 뉴런에 거꾸로 전파되어오는 형태를 띠게 된다. 이는 프로그램으로 구현함에 있어서도 출력층에서 계산된 오차를 은닉층에서도 다시 재사용함으로써 중복 계산을 피할 수 있다는 장점도 갖게 되어 알고리즘의 핵심적인 특징이 된다. 이상에서 알아본 다층 퍼셉트론의 기울기 강하 학습법을 오류역전파 학습 알고리즘이라고 부르는 이유도 이에 근거한 것이다.



[그림 11-13] 오류역전파 계산의 개념도

지금까지 계산된 [식 11-11]과 [식 11-12, 11-16]을 이용하면 최종적으로 기울기 강하 학습법에 의한 가중치 수정식 [식 11-9]를 얻을 수 있다. 이상에서 살펴본 내용을 바탕으로 다층 퍼셉트론을 위한 오류역전파 학습 알고리즘을 정리하면 다음과 같다.

[기울기 강하 학습법에 의한 다층 퍼셉트론의 학습]

① 임의의 값으로 가중치 파라미터를 초기화하고, 학습률(η)과 원하는 오차 함수의 최소값을 설정한다.

② 모든 학습 데이터에 대해, 한 번에 하나의 데이터를 추출하여 다음 단계에 따라 가중치를 수정하는 과정을 반복한다.

[전방향 계산] 현재의 가중치 w_{ij}, v_{jk}를 이용하여 출력값을 계산한다.

②-1. 하나의 입력 패턴 $\boldsymbol{x}$에 대하여 각 은닉 뉴런의 출력값 $z_j\,(j=1...m)$를 계산한다.

$$u_j^h = \sum_{i=1}^{n} w_{ij}x_i + w_{0j}, \quad z_j = \phi_h(u_k^h)$$

②-2. 이어서 출력 뉴런의 출력값 $y_k(k=1...M)$를 계산한다.

$$u_k^o = \sum_{j=1}^{m} v_{jk}z_j + v_{0k}, \quad y_k = \phi_o(u_k^o)$$

[역방향 계산] 현재의 가중치 w_{ij}, v_{jk}를 이용하여 오류를 계산한다.

②-3. 각 출력 노드의 출력값 y_k와 목표 출력값 t_k를 비교하여 다음 식을 이용하여 출력 뉴런으로의 가중치 수정항을 계산한다.($k=1...M,\ j=0...m,\ \ z_0=1$)

$$\frac{\partial E}{\partial v_{jk}} = \frac{\partial E}{\partial u_k^o}\frac{\partial u_k^o}{\partial v_{jk}} = \delta_k z_j, \qquad \delta_k = -\phi_o{}'(u_k^o)(t_k - y_k)$$

②-4. 계산된 δ_k값을 이용하여 각 은닉 뉴런으로의 가중치 수정항을 계산한다. $(j=1...m, i=0...m, x_0=1)$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial u_j^h}\frac{\partial u_j^h}{\partial w_{ij}} = \delta_j x_i$$

$$\delta_j = \frac{\partial E}{\partial u_j^h} = \sum_{k=1}^{M}\frac{\partial E}{\partial u_k^o}\frac{\partial u_k^o}{\partial u_j^h} = \phi_h{}'(u_j^h)\sum_{k=1}^{M} v_{jk}\delta_k$$

②-5. 계산된 수정항과 학습률을 이용하여 가중치 파라미터를 수정한다.

$$w_{ij}^{(\tau+1)} = w_{ij}^{(\tau)} - \eta\frac{\partial E}{\partial w_{ij}}, \quad v_{jk}^{(\tau+1)} = v_{jk}^{(\tau)} - \eta\frac{\partial E}{\partial v_{jk}},$$

③ 전체 학습 데이터에 대하여 ② 과정이 완료된 후, 학습 데이터 집합 X 전체에 대한 제곱평균오차를 다음과 같이 계산한다.

$$E(X,\boldsymbol{\theta}) = \frac{1}{2N}\sum_{i=1}^{N} \| \boldsymbol{t}_i - \boldsymbol{f}(\boldsymbol{x}_i,\boldsymbol{\theta}) \|^2$$

④ ③에서 계산된 오차값이 원하는 오차보다 적으면 학습을 마치고, 그렇지 않으면 ②~③ 과정을 반복한다.

11.2.4 다층 퍼셉트론 학습의 고려사항

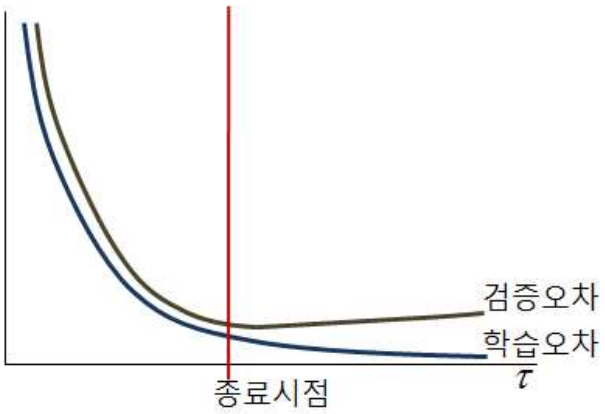
앞 절에서 살펴본 오류역전파 알고리즘은 다층 퍼셉트론 학습을 위한 가장 기본적인 알고리즘이다. 이를 이용하여 실제 응용문제에서 다층 퍼셉트론의 학습을 수행할 때에는 몇 가지 고려해야 할 점들이 있다. 이 절에서는 이에 대해서 간략히 살펴보겠다.

지역 극소의 문제: 기울기 강하 학습법은 기울기를 따라서 학습을 수행하므로, 한 번에 움직이는 수정폭의 크기를 충분히 작게 설정만 해 주면 기울기를 따라 오차값이 점점 감소하는 방향으로 움직이다가 기울기가 0이 되는 점에서 멈추게 되어 극소값을 찾는 것이 보장된다. 극소값에 도달하여 더 이상 움직이지 않는다는 것이 보장되므로, 안정적으로 해에 도달하는 것을 보장한다. 그렇지만, 이렇게 찾아지는 해는 어디까지나 지역 극소에 해당한다. 기울기 강하 학습법에 대해서 설명하는 단계에서 어느 정도 알아차린 독자들이 있겠지만, [그림 11-12]를 다시 한 번 주의 깊게 살펴보자. 만약 파라미터가 움직이기 시작하는 점이 최적해(전역 극소)에서 멀리 떨어진 지점이라고 한다면, 학습 도중에 처음 만나게 되는 지역 극소에 도달 한 후에 더 이상 움직이지 않게 될 것이다. 이러한 문제를 <지역 극소의 문제(local minima problem)>이라고 한다. 이를 해결하기 위해서 수정폭을 결정하는 학습률을 적응적으로 조정하거나, 학습의 초기 단계에서는 지역 극소로부터 빠져나올 수 있는 여지를 만들어 주는 시뮬레이티드 어닐링 기법 등이 사용된다.
지역 극소는 사실상 실제 문제에서 빈번하게 발생되기는 하나, 만약 찾아진 지역 극소가 우리가 원하는 정도의 오차값을 줄 수 있다면 크게 문제가 되지는 않을 것이다. 이러한 관점에서 현실적인 해결책으로는 탐색의 시작점을 결정하는 초기치를 변화시키면서 여러 번 학습을 시도하여 원하는 정도의 오차를 얻어내는 방법이나, 충분히 많은 수의 은닉 노드를 사용함으로써 원하는 정도의 오차를 얻도록 시도해 볼 수 있을 것이다.

수렴 속도의 문제: 오류역전파 알고리즘의 또 한 가지 문제점은 원하는 해에 수렴하기까지 필요한 학습 시간이 길다는 점이다. 이는 반복 알고리즘이 가지는 일반적인 특성이라고도 볼 수 있으나, 특히 오류역전파 알고리즘과 같이 기울기를 따라 강하하는 방법에서는, 오차함수의 기울기가 완만한 지점에서는 급격히 학습 속도가 느려지는 현상이 나타난다. 많은 경우에 기울기가 완만한 지역에서의 학습이 전체적인 학습 시간의 대부분을 차지하게 되어 결과적으로 학습이 매우 느려지게 되는데, 이를 <플라토 문제(plateau problem)>이라고 부른다. 이를 해결하기 위하여 바로 직전 단계의 수정에 사용된 수정항을 추가적으로 더해주어 관성의 역할을 하도록 하는 모멘텀 방법을 비롯하여, 오차 함수의 2차미분을 사용하는 뉴턴 방법, 그리고 기울기를 새롭게 정의하여 사용하는 자연기울기 방법 등 다양한 가속화 방법이 제안되었다.

학습 종료점의 문제: 앞서 제시한 학습 알고리즘에서, 학습의 시작 전에 원하는 학습 오차를 설정하여 학습의 종료점을 결정하도록 하였다. 그러나 많은 경우에 원하는 학습 오차를 시작 전에 알기는 힘들며, 1장에서 설명한 바와 같이 학습 오차가 적다고 해서 항상 좋은 결과를 주는 것은 아니므로, 과다적합을 피할 수 있는 적절한 학습 종료 시점을 결정할 필요가 있다. 이를 위해서 사용할 수 있는 방법 중의 하나는 학습 데이터 집합 외에 검증 집합

을 따로 두어, 학습이 진행되는 과정에서 검증 집합에 대한 검증 오차도 함께 계산해 보는 것이다. 이렇게 하면 일반적으로 [그림 11-14]와 같은 형태의 오차 곡선을 얻게 된다. 즉, 학습 데이터에 대한 학습 오차는 학습이 진행됨에 따라 계속 줄어드는 반면, 검증 데이터에 대한 오차는 어느 시점에서 다시 증가하게 되는데, 이 시점이 학습 데이터에 대한 과다적합이 발생하게 되는 지점이 되므로 이 지점에서 학습을 완료하는 것이 가장 바람직하다.



[그림 11-14] 학습곡선과 학습의 종료시점

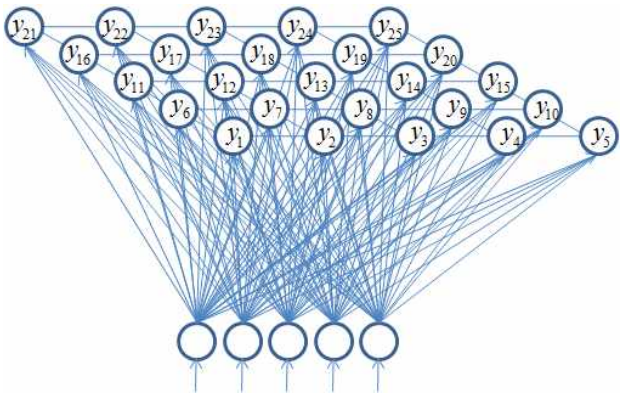
은닉 뉴런의 수: 다층 퍼셉트론의 학습하기에 앞서 우리는 그 구조를 결정해 주어야 한다. 즉, 입력 뉴런과 출력 뉴런의 수와 함께 은닉층의 수와 은닉 뉴런의 수를 결정해 주어야 한다. 입력 뉴런과 출력 뉴런은 주어진 데이터에 의해 결정되는 값이므로, 문제에 맞게 설정해 주면 될 것이다. 예를 들어, 70×50 크기의 영상으로 이루어진 숫자 패턴 0부터 9까지를 인식하는 문제를 신경망으로 학습한다면 입력층의 뉴런 수는 영상의 한 픽셀 당 하나의 입력 뉴런이 필요하므로 모두 3500개의 뉴런이 필요하다. 출력 뉴런의 수는 각 숫자 패턴당 하나를 할당하여 모두 10개의 뉴런이 필요할 것이다. 은닉층의 수는 일반적으로 한 개를 사용한다. 이는 한 개의 은닉층만을 사용하더라도 은닉 뉴런의 수만 충분히 주어지면 원하는 형태의 함수를 모두 근사해 낼 수 있음이 증명되어 있기 때문이다. 이에 비하여 은닉 뉴런의 수는 실제 문제에서 학습의 속도와 찾아지는 해의 성능을 좌우하게 되므로 문제에 맞게 적절히 정해 주어야 한다. 그런데 이는 다분히 문제에 의존적이 값이므로 어떻게 결정해 주어야 할지에 대한 정확한 해답은 주어지지 않는다. 다만, 입력 데이터의 차원과 데이터의 수에 비례하여 그 수를 조정해야 할 것이다. 11.4절에서 은닉 노드 수에 따른 변화를 살펴보겠다.

11.3 군집화를 위한 신경망

11.3.1 자기조직화 신경망

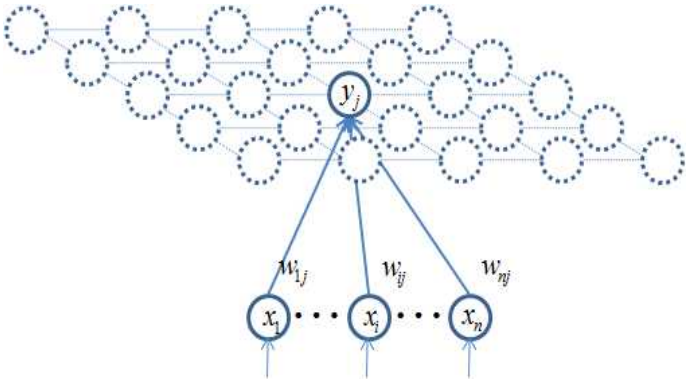
앞 절에서 살펴본 다층 퍼셉트론은 대표적인 분류기 중의 하나로, 교사학습에 의해서 학습을 수행한다. 따라서 분류 문제에는 적합하나 군집화에는 적용하기 힘들다. 또한 인간의 뇌

를 모방하여 만든 모델이라는 관점에서 볼 때에도, 신경세포의 학습이 항상 정확한 출력값을 가지고 이루어진다고 보는 것은 부자연스럽다. 이와 별도로 목표 출력값이 따로 주어지지 않더라도 스스로 가중치를 조절하는 비교사학습 신경망에 대한 연구는 수행되었다. <자기조직화 신경망(Self organizing map, SOM)>은 가중치를 조정하여 비슷한 입력값에 대하여 비슷한 출력을 낼 수 있도록 출력층을 스스로 조직화하는 능력을 가진 신경망으로, 군집화나 데이터 시각화의 문제에 널리 사용되고 있다. 이 절에서는 자기조직화 신경망의 정보처리 방식에 대하여 알아보고, 다음 절에서 학습 알고리즘을 살펴보겠다.



[그림 11-15] 자기조직화 신경망의 구조

앞 절에서 우리는 신경망의 기본 요소로서 신경세포(활성화 함수), 신경망의 구조, 그리고 학습 방식에 대해 배웠다. 하나의 신경망 모델에 대해 알기 위해서는 이 세 가지 요소를 중심으로 파악해 보면 될 것이다. 여기서는 먼저 자기조직화 신경망(SOM)의 구조에 대해서 알아보겠다. [그림 11-15]에 SOM의 구조를 나타내었다. SOM 역시 층상 구조를 이루고 있으며, 입력층과 출력층만을 가지는 단층 신경망이다. 그런데 SOM이 가지는 독특한 특성은, 출력층이 일렬로 나열되는 것이 아니라 2차원의 격자 구조로 배치된다는 점이다. [그림 11-15]에서는 25개의 출력 뉴런이 5행 5열의 배열 구조로 배치되어 있다. 이러한 배치 구조는 SOM의 가장 큰 특징 중 하나로, SOM이 학습을 통해 찾은 입력에 대한 군집 특성을 나타내는데 유용하게 사용된다. 이에 대해서는 다음 절에서 자세히 알아보겠다. 격자로 배열된 출력층에서 하나의 노드는 좌/우/상/하에 이웃한 노드를 가지게 되고, [그림 11-15]에서는 이를 직선으로 연결하여 표시해 두었다. 이 이웃 관계는 추후 학습에서 중요한 역할을 하게 될 것이다.



[그림 11-16] 입력 뉴런에서 출력 뉴런으로의 가중치

다음으로, 각각의 신경세포가 어떻게 정보를 처리하는지에 대하여 알아보겠다. 다른 층상 구조의 신경망들과 마찬가지로 입력층은 단지 정보를 전달하는 역할만 하며, 실질적인 계산은 출력층에서 이루어진다. 출력층의 하나의 뉴런은 입력층의 모든 뉴런과 가중치를 가지고 연결되어 있다([그림 11-16 참조]). 각 출력 뉴런의 출력값을 계산하는데 있어서는 이 가중치가 중요한 역할을 한다. j번째 출력 뉴런이 i번째 입력 뉴런과 연결된 가중치를 w_{ij}라고 할 때, n개의 모든 입력 뉴런들로부터 j번째 출력 뉴런으로의 연결 가중치를 하나로 묶어서 n 차원 벡터 $\boldsymbol{w_j} = [w_{1j}, w_{2j}, \ldots, w_{nj}]^T$ 로 나타낼 수 있다. 이 가중치 $\boldsymbol{w_j}$가 j번째 출력 뉴런의 특성을 나타내는 중요한 값이 된다. 이를 이용하여 입력 뉴런들로부터 입력값 $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]^T$ 를 받아 각 출력 뉴런의 출력값 y_j $(j = 1 \ldots M)$를 계산하는 과정은 다음과 같은 단계로 정리할 수 있다.

1. 모든 출력 뉴런에 대하여 주어진 입력값 $\boldsymbol{x}$와 가중치 $\boldsymbol{w_j}$와의 거리를 계산한다.

$$d(\boldsymbol{x}, \boldsymbol{w_j}) = \| \boldsymbol{x} - \boldsymbol{w_j} \|^2 = \sum_{i=1}^{n} (x_i - w_{ij})^2 \qquad \text{[식 11-17]}$$

2. 모든 출력 뉴런들 중 거리값이 가장 작은 출력 뉴런을 찾아 승자(winner)로 둔다.

$$\omega = \mathrm{argmin}_j \{ d(\boldsymbol{x}, \boldsymbol{w_j}) \} \qquad \text{[식 11-18]}$$

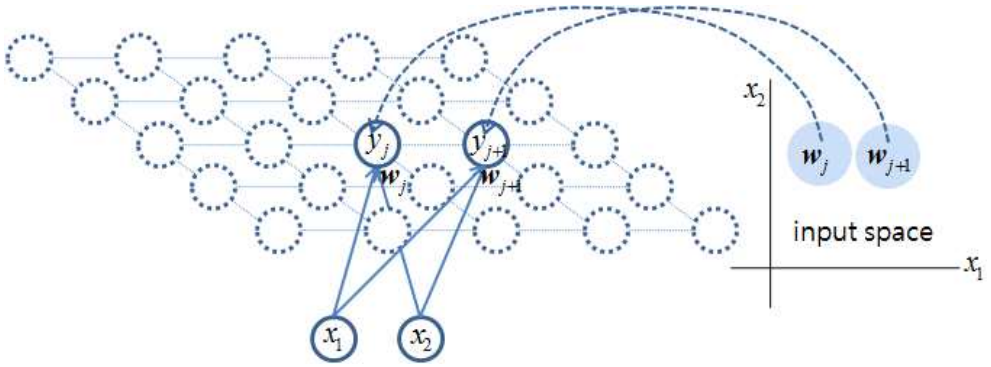
3. 승자가 된 뉴런만 출력값 1을 가지고, 나머지 뉴런들은 0의 출력값을 가진다.

$$y_j = \begin{cases} 1 & \text{if } \omega = j \\ 0 & \text{otherwise} \end{cases} \qquad \text{[식 11-19]}$$

이 처리 과정을 통하여 결국 현재 주어진 입력 패턴과 가장 유사한 가중치를 가지는 출력 뉴런이 승자가 되고, 승자만 1의 출력값을 낼 수 있게 된다. 이러한 계산 방식을 <승자전취(winner-take-all)>라고 한다. 승자는 가중치가 입력 뉴런과 얼마나 유사한지에 의해 결정되므로, 결국 출력 뉴런이 가지는 가중치 $\boldsymbol{w_j}$는 각 뉴런이 기억하고 있는 입력 패턴이라고 볼 수 있다. 군집화의 관점에서 보면, 각 출력 뉴런이 하나의 군집을 나타내고, 가중치 $\boldsymbol{w_j}$는 그 군집의 대표 벡터에 해당한다. 결국 SOM에서의 정보처리 과정은 주어진 입력과 가장 비슷한 대표 벡터를 가진 군집을 찾는 것으로 해석할 수 있다. 출력층의 노드들이 좋은 군집화를 이루기 위해서는 가중치 $\boldsymbol{w_j}$를 적절히 조정해 주는 학습법이 필요하다.

11.3.2 자기조직화 신경망의 학습

앞 절에서 살펴본 바에 따르면 SOM에서의 학습의 목적은 입력 데이터 집합을 분석하여 M개(출력 뉴런의 개수)의 군집으로 나누고, 각 군집의 대표 벡터는 해당 출력 뉴런의 가중치에 저장되는 것으로 볼 수 있다. 그런데 여기에 덧붙여서 SOM은 한 가지 중요한 특징을 더 가지고 있는데, "이웃한 출력 노드는 비슷한 특성을 가진 군집을 대표해야 한다."는 것이다. 이에 대한 개념을 [그림 11-17]에 나타내었다. 입력 공간상에서 이웃한 영역에 있는 군집들은 출력 노드들로 나타나는 이차원의 격자 공간상에서도 이웃한 영역에 대응되도록 한다. 이렇게 함으로써 입력 공간상에서의 거리 관계가 출력 뉴런들이 구성하는 2차원 격자공간상에서도 어느 정도 유지되어, 결국 출력 노드는 일종의 입력 데이터에 대한 2차원의 특징지도를 형성하게 된다. 이러한 의미에서 SOM을 <자기조직화 특징지도 (self organizing feature map, SOFM)>이라고도 부른다.



[그림 11-17] SOM에서의 군집화의 특성

이제 이러한 특징지도를 찾기 위하여 가중치를 조절하는 방법에 대하여 알아보겠다. 하나의 입력 패턴 $\boldsymbol{x}$가 주어지면, 앞 절에서 살펴본 바와 같이 그와 가장 비슷한 가중치를 가지는 출력 뉴런이 승자로 선택된다. 그러면 이 뉴런의 가중치는 현재의 입력 패턴과 보다 유사해질 수 있도록 수정되어야 할 것이다. 따라서 다음과 같이 가중치 수정식을 정의할 수 있다.

$$\boldsymbol{w}_j^{(\tau+1)} = \boldsymbol{w}_j^{(\tau)} + \eta(\boldsymbol{x} - \boldsymbol{w}_j^{(\tau)}) \qquad \text{[식 11-20]}$$

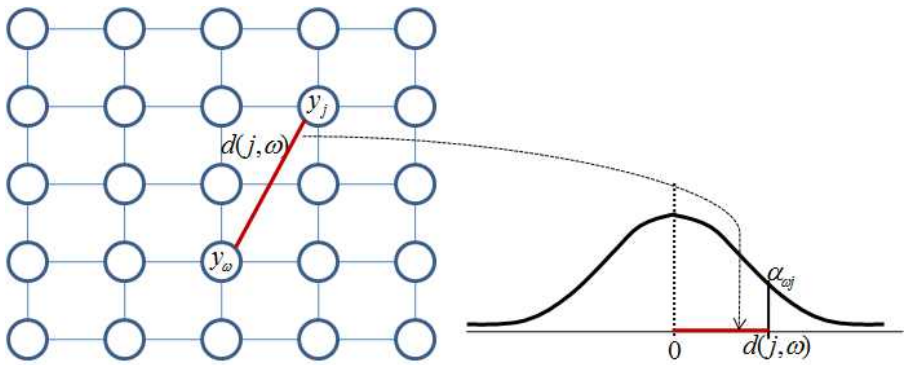
이 식에서 $\boldsymbol{w}_j^{(\tau)}$는 j번째 출력 뉴런의 현재의 가중치 파라미터이고, $\boldsymbol{w}_j^{(\tau+1)}$는 수정된 후 얻어지는 가중치 파라미터이다. 학습식은 결국 입력 패턴과 현재 가중치의 차이에 작은 상수값(η, 학습률)을 곱하여 더해줌으로써 입력 패턴에 조금씩 가까워지도록 조정하는 역할을 한다.

승자 뉴런에 대해서는 이렇게 학습하여 가중치를 수정할 수 있으나, 나머지 다른 뉴런들에 대해서는 어떻게 할지 결정해야 할 것이다. 기본적인 군집화의 경우를 생각하면, 현재 데이터를 대표하는 군집에 대해서만 학습을 수행하면 되므로 다른 뉴런들의 가중치는 수정하지 않고 그대로 두는 것이 바람직 할 것이다. 그런데, SOM의 경우는 이웃에 위치한 출력 뉴런

들이 비슷한 군집을 나타내도록 가중치가 조정되어야 하므로, 현재의 승자 뉴런이 학습을 수행할 때, 그 이웃한 뉴런들도 어느 정도 수행되어야 할 것이다. 따라서 이웃한 뉴런들에 대해서는 다음과 같은 가중치 수정식을 생각해 볼 수 있다.

$$w_j^{(\tau+1)} = w_j^{(\tau)} + \eta\alpha(x - w_j^{(\tau)}) \qquad \text{[식 11-20]}$$

이 식에서 α는 승자 뉴런과의 이웃 관계에 따라 수정의 폭을 결정하는 값이 된다. 만약 j번째 뉴런이 승자 뉴런이라면 α=1로 둠으로써 [식 11-19]와 동일한 수정식을 얻을 수 있다. 만약 j번째 뉴런이 승자 뉴런의 이웃이 아니라면, α=0으로 둠으로써 학습이 일어나지 않도록 한다. j번째 뉴런이 승자 뉴런의 이웃인 경우에는 0에서 1사이의 값으로 두어 승자 노드보다는 작은 폭의 수정이 일어날 수 있도록 하면 될 것이다. 그렇다면 이웃 뉴런을 어떻게 정의할 지를 생각해 보아야 하겠다. 이웃이란 앞 절에서 설명한 것처럼, 2차원으로 배열된 출력층의 격자 구조상에서 가까이 존재한 뉴런들을 의미한다. 따라서 가장 기본적으로는 현재 승자 뉴런의 상/하/좌/우에 있는 뉴런들을 이웃으로 볼 수 있을 것이다. 그런데 이것을 조금 더 일반화하여 각 뉴런들을 이웃인 경우와 그렇지 않은 경우로 분류하는 대신, 각 뉴런과 승자 노드간의 거리를 계산하여 이 거리값이 가까울수록 이웃 그룹에 속할 소속도를 높여주는 방식을 생각해 볼 수 있다. 이렇게 하면 이웃 그룹에 속할 소속도가 곧 α값이 되어 자연스럽게 가중치 수정식이 완성될 수 있다.



[그림 11-18] 이웃 뉴런의 소속도

그렇다면 구체적으로 소속도를 계산하는 함수를 정의해 보자. 우선 뉴런들 사이의 거리가 계산되어야 할 것인데, 이는 2차원으로 배열된 출력층에서의 거리이므로, j번째 출력 뉴런의 2차원 배열에서의 위치를 (r_j행,c_j열)로 나타내면, 승자 뉴런 ω까지의 거리는 다음과 같이 나타낼 수 있다.

$$d(j,\omega) = (r_j - r_\omega)^2 + (c_j - c_\omega)^2 \qquad \text{[식 11-21]}$$

이 거리에 의한 소속도는 가우시안 함수를 사용하여 다음과 같이 정의 할 수 있다.

$$\alpha_j = \frac{}{\sqrt{2\pi}\sigma} exp\left\{-\frac{1}{2}\frac{(d(j,\omega))^2}{\sigma^2}\right\} \qquad \text{[식 11-22]}$$

이 식을 이용하면 j가 승자 뉴런인 경우는 α_j=1이 되고, 승자 뉴런과의 거리가 멀어질수록 그 값이 0에 가까워진다. 여기서 σ는 소속도 함수의 퍼진 정도를 나타내는 값으로 그 값이 크면 넓은 범위까지 이웃으로 간주하는 것이 되며, 그 값이 작으면 이웃의 범위를 좁히는 결과가 된다. σ의 값은 학습의 초기 단계에서는 크게 주어 이웃의 범위를 넓히고, 학습이 진행됨에 따라 점점 줄여서 학습의 안정도를 높이는 방식을 취한다. 따라서 σ는 학습횟수 τ에 대한 함수로 다음 식을 이용하여 조정할 수 있다.

$$\sigma(\tau) = \sigma_0 \exp\left(-\frac{\tau}{T}\right) \qquad \text{[식 11-23]}$$

여기서 σ_0는 학습 시작 당시의 초기값이며, T는 설계자가 결정하는 상수값이다. 이를 종합하면 j번째 출력 뉴런에 대한 소속도 α_j는 승자 뉴런 ω와 학습횟수 τ에 의존하는 함수가 되어 $\alpha_{\omega j}^{(\tau)}$로 나타낼 수 있다. 지금까지 설명한 내용을 바탕으로 SOM의 학습 알고리즘을 정리하면 다음과 같다.

[자기조직화 학습]

① 임의의 값으로 가중치 파라미터를 초기화하고, 학습률(η)과 소속도의 범위를 결정하는 파라미터(σ)의 초기치를 설정한다.

② 모든 학습 데이터에 대해, 한 번에 하나의 데이터를 추출하여 다음 단계에 따라 가중치를 수정하는 과정을 반복한다.

[승자 뉴런의 계산] 현재의 가중치 $\boldsymbol{w}_j(j=1...M)$를 이용하여 출력값을 계산한다.

②-1. 하나의 입력 패턴 $\boldsymbol{x}$에 대하여 각 가중치 $\boldsymbol{w}_j$와의 거리를 계산한다.

$$d(\boldsymbol{x},\boldsymbol{w}_j) = \| \boldsymbol{x}-\boldsymbol{w}_j \|^2 = \sum_{i=1}^{n}(x_i - w_{ij})^2$$

②-2. 거리가 가장 가까운 승자 뉴런을 찾는다.

$$\omega = \mathrm{argmin}_j\{d(\boldsymbol{x},\boldsymbol{w}_j)\}$$

[가중치 수정] 출력 뉴런들의 가중치 $\boldsymbol{w}_j(j=1...M)$를 수정한다.

②-3. 각 출력 뉴런들의 소속도를 계산한다.

$$\alpha_{\omega j}^{(\tau)} = \exp\left\{-\frac{1}{2}\frac{((r_j-r_\omega)^2+(c_j-c_\omega)^2)}{\sigma(\tau)^2}\right\}$$

②-4. 각 출력 뉴런들의 가중치를 계산한다.

$$\boldsymbol{w}_j^{(\tau+1)} = \boldsymbol{w}_j^{(\tau)} + \eta\alpha_{\omega j}^{(\tau)}(\boldsymbol{x}-\boldsymbol{w}_j^{(\tau)})$$

②-5. 파라미터 σ를 감소시킨다.

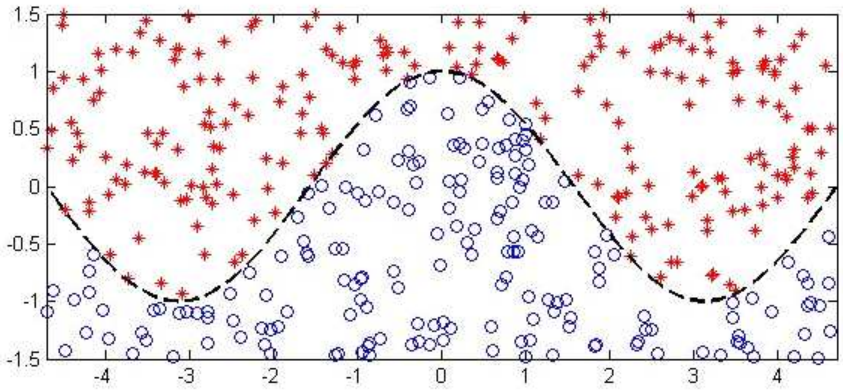
$$\sigma(\tau) = \sigma_0\exp\left(-\frac{\tau}{T}\right)$$

③ 학습 데이터에 대한 출력값의 변화가 충분히 작아질 때까지 ②과정을 반복한다.

11.4 매트랩을 이용한 구현

11.4.1 MLP를 이용한 패턴 분류

11.2절에서 배운 다층 퍼셉트론의 학습 알고리즘을 이용하여 다층 퍼셉트론으로 2차원 데이터를 분류하는 문제를 매트랩을 이용하여 구현해 보겠다. [그림 11-19]에 학습에 사용된 데이터를 나타내었다. 데이터는 2차원 실수 공간 $\{(x_1,x_2)\,|\,x_1\in[-1.5\pi,1.5\pi], x_2\in[-1.5,1.5]\}$으로부터 균일분포(uniform distribution)에 따라 400개를 추출하여, 그 중 코사인곡선($\cos(x_1)$)보다 위에 있는 점을 클래스 C_1으로, 아래에 있는 점을 클래스 C_2로 할당하였다. 따라서 두 클래스에 대한 최적의 결정경계는 그림 [11-19]에 나타난 것과 같은 비선형 코사인 곡선이 된다. 테스트 데이터 집합도 같은 방법으로 400개의 데이터를 추출하여 만들었다. 다층 퍼셉트론을 학습하기 위해서는 목표 출력값도 데이터로 주어져야 한다. 이 문제에서는 두 개의 클래스를 분류하는 문제이므로 출력 뉴런을 두 개로 설정하고, C_1에 속하는 데이터에 대해서는 목표 출력을 [1,-1]로 두고, C_2에 속하는 데이터에 대해서는 목표 출력을 [-1,1]로 두었다.



[그림 11-19] 학습데이터와 결정경계

패턴 분류를 위한 다층 퍼셉트론 학습 알고리즘이 [프로그램 11-1]에 주어졌다. 데이터를 읽어온 후, 그에 맞게 신경망의 구조를 설정한다. 이 경우에는 2차원 입력이 주어지므로 입력 뉴런의 개수는 2이며, 목표 출력도 2차원 벡터로 주어지므로 출력 뉴런의 개수도 2이다. 은닉 뉴런의 개수는 여러 가지 경우를 수행해 보아 가장 적절한 것을 선택할 예정이나 첫 수행에서는 5로 두었다. 신경망의 뉴런 수가 정해지면, 그에 따라 개수가 정해지는 가중치 파라미터를 초기화해야 한다. 가중치 파라미터는 은닉 뉴런의 가중치(w_{ij})와 바이어스 입력에 대한 가중치(w_{0j}), 그리고 출력 뉴런에 대한 가중치(v_{jk})와 바이어스 입력에 대한 가중치 (v_{0k})가 있으며, 이들은 모두 작은 실수값으로 랜덤하게 설정하였다. 이 외에 학습률(η)와 최대 반복횟수도 적당한 값으로 설정해 두었다. 또한 은닉 뉴런과 출력 뉴런의 활성화 함수는 -1에서 1사이의 값을 주는 하이퍼탄젠트 함수를 사용하였다.

학습은 하나의 데이터가 주어질 때마다 가중치를 수정하는 온라인 방식으로 수행되도록 구성하였는데, 학습 데이터 집합에 속한 모든 데이터에 대해 한 번의 학습이 수행된 후에는, 학습 데이터 집합과 테스트 데이터 집합에 대하여 각각 오차를 계산하여 그 변화를 저장할

수 있도록 하였으며, 학습은 주어진 원하는 학습오차가 얻어질 때까지 수행하도록 하였다. 오차 계산을 위해서는 데이터 집합과 가중치 파라미터를 입력받아 평균제곱오차와 분류율을 계산하여 반환하는 함수 **MLPtest**를 따로 구현하여 [프로그램 11-1-2]에 나타내다. 이 외에 신경망 학습계산 도중에 필요한 하이퍼탄젠트 함수의 1차 도함수도 따로 매트랩 함수 **d_tanh**로 작성하여 사용하였는데, 이는 [프로그램 11-1-1]에 나타내었다.

프로그램 11-1 MLP 분류기 학습 프로그램

2차원 데이터 분류를 위한 MLP 분류기를 학습하는 프로그램

```
001 load data12_20                          %데이터 불러오기
002 N=size(X,1);                            %데이터의 수
003 INP=2; HID=5; OUT=2;                    %뉴런의 수 설정
004 w=rand(INP,HID)*0.4-0.2;                %입력->은닉 뉴런 가중치 초기화
005 w0=rand(1,HID)*0.4-0.2;
006 v=rand(HID,OUT)*0.4-0.2;                %은닉->출력 뉴런 가중치 초기화
007 v0=rand(1,OUT)*0.4-0.2;
008 eta=0.001;                              %학습률 설정
009 Mstep=5000; Elimit=0.05;                %반복횟수 설정
010
011 for j = 2:Mstep                          %학습 반복 시작
012   for i=1:N                             %각 데이터에 대한 반복 시작
013     x=X(i,:); t=T(i,:);                 %입력과 목표 출력 데이터 선택
014     uh=x*w+w0;                          %은닉 뉴런의 가중합 계산
015     z=tanh(uh);                         %은닉 뉴런의 출력 계산
016     uo=z*v+v0;                          %출력 뉴런의 가중합 계산
017     y=tanh(uo);                         %출력 뉴런의 출력 계산
018     e=y-t;                              %신경망 출력과 목표 출력의 차 계산
019     E(i,1)=e*e';                        %제곱오차 계산
020     delta_v=d_tanh(y).*e;               %학습을 위한 델타값 계산 (출력뉴런)
021     delta_w=d_tanh(z).*(delta_v*v');    %학습을 위한 델타값 계산 (은닉뉴런)
022     v=v-eta*(z'*delta_v);               %출력 뉴런의 가중치 수정
023     v0=v0-eta*(1*delta_v);              %출력 뉴런의 바이어스 가중치 수정
024     w=w-eta*(x'*delta_w);               %은닉 뉴런의 가중치 수정
025     w0=w0-eta*(1*delta_w);              %은닉 뉴런의 바이어스 가중치 수정
026   end
027   [serr, cerr]=MLPtest(X,T,w,w0,v,v0);      %학습 오차 계산 함수 호출
028   fprintf(1,'%d %7.3f %7.3f₩n',j,serr,cerr);  %오차 변화 출력
029   Serr(j-1,:)=serr;  Cerr(j-1,:)=cerr;      %오차 변화 저장
030   if (serr<Elimit) break; end           %원하는 오차를 얻으면 학습 종료
031 end
032
033 save MLPweight INP HID OUT v v0 w w0     %가중치와 신경망 구조 저장
```

프로그램 11-1-1 하이퍼탄젠트의 1차 도함수 d_tanh

입력 벡터를 받아서 하이퍼탄젠트 함수의 1차 도함수의 값을 반환

```
001 function [out]=d_tanh(a)
002 out=(1-a).*(1+a);
```

프로그램 11-1-2 학습된 MLP 분류기를 이용한 오차계산 함수 MLPtest

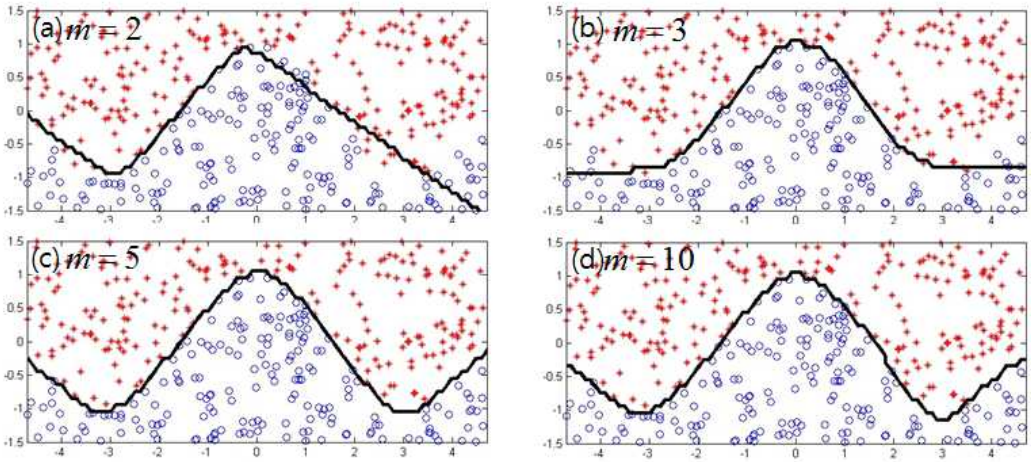
데이터 집합과 가중치를 받아 평균제곱오차와 분류오차를 계산

```
001 function [SEtst, CEtst] = MLPtest(Xtst, Ttst, w,w0, v, v0)
002 N=size(Xtst,1);                          %데이터의 수
003 for i=1:N                                %각 데이터에 대한 인식 시작
004     x=Xtst(i,:);   t=Ttst(i,:);          %입/출력 데이터 설정
005     uh=x*w+w0;                           %은닉 뉴런의 가중합 계산
006     z=tanh(uh);                          %은닉 뉴런의 출력 계산
007     uo=z*v+v0;                           %출력 뉴런의 가중합 계산
008     y=tanh(uo);                          %출력 뉴런의 출력 계산
009     e=y-t;                               %목표 출력과의 차 계산
010     E(i,1)=e*e';                         %제곱오차 계산
011     if y(1)>y(2) Ytst(i,:) = [1, -1];    %최종 인식 결과 판단
012     else Ytst(i,:) = [-1 1]; end
013 end
014 SEtst=sum(E.^2)/N;                       %평균제곱오차 계산
015 diffTY=sum(abs(Ttst-Ytst))/2;
016 CEtst=diffTY(1)/N;                       %분류오차 계산
```

[그림 11-20]에는 은닉 뉴런의 수(m)가 각각 2, 3, 5, 10인 경우에 각각 학습을 수행시킨 결과로부터 얻어진 결정경계를 보여주고 있다. 은닉 뉴런의 수가 2인 경우에는 바른 결정경계를 찾지 못하였으며, 3인 경우는 유사한 결정경계를 찾았으나 좌/우 끝부분의 곡선 경계를 찾는데 실패하였다. 반면 은닉 뉴런이 5나 10인 경우에는 [그림 11-19]에서 보여준 결정경계와 유사한 경계를 찾았다. 테스트 데이터에 대해 분류를 수행해 본 결과를 [표 11-1]에 나타내었다. 결정경계로부터 예측할 수 있는 바와 같이 은닉 뉴런이 5나 10인 경우에는 거의 완벽하게 분류를 수행하였다.

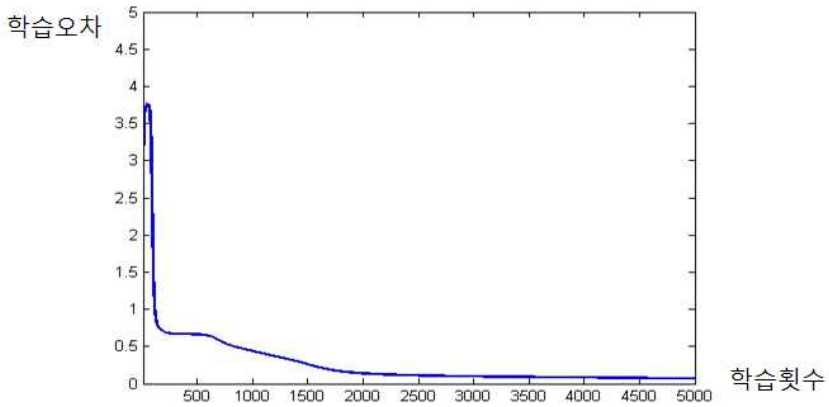
[표 11-1] 은닉 뉴런 수에 따른 분류오차

은닉 뉴런의 수	2	3	5	10
분류오차	6%	5%	1.5%	1%



[그림 11-20] 학습을 통해 얻어진 결정경계 (은닉 뉴런의 수에 따른 변화)

학습 데이터를 이용하여 학습을 수행하는 동안 신경망이 내는 출력값에 대한 평균제곱오차의 변화를 [그림 11-21]에 나타내었다. 그림에서 세로축은 목적 출력과 실제 출력과의 제곱오차의 전체 데이터에 대한 평균을 나타내며, 가로축은 역전파 데이터 집합 전체에 대한 학습을 한 번 수행한 것을 학습횟수 1로 하여 나타내었다. 이 경우는 은닉 뉴런의 수가 5인 경우로, 3000번 정도의 수행으로 학습이 거의 완료됨을 알 수 있다.



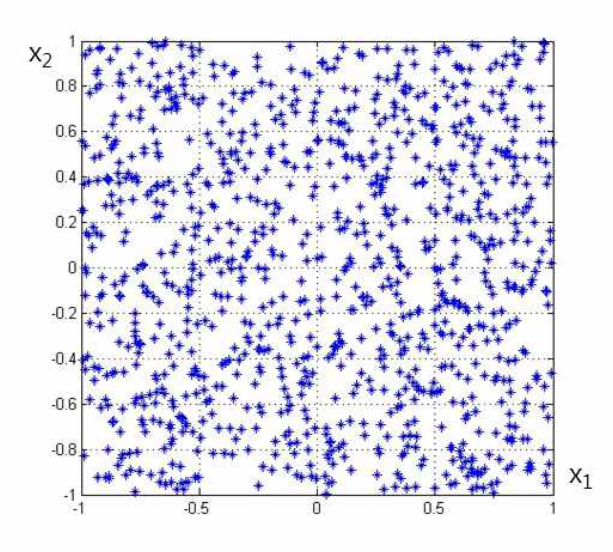
[그림 11-21] 학습횟수에 따른 학습 오차의 변화

11.4.2 SOM을 이용한 군집화와 특징지도

이어서 11.3절에서 소개한 자지조직화 신경망을 이용하여 2차원 데이터에 대한 특징지도를 만드는 실험을 수행한다. SOM의 학습 알고리즘을 [프로그램 11-2]에 매트랩 코드로 구현하였다. 입력으로는 2차원의 실수 공간 $\{(x_1,x_2)\,|\,x_1\in[-1,1], x_2\in[-1,1]\}$ 내에서 균등분포를 따르도록 1000개의 데이터를 추출하여 사용하였다. 학습에 사용된 1000개의 데이터를 [그림 11-22]에 나타내었다. 그림에서 보이듯이 주어진 공간 안에 존재하는 점들에 대하여 학습을

수행하면, 출력층의 각 뉴런들이 [그림 11-22]에 나타난 전체 입력 공간 중 특정 부분을 담당하도록 가중치가 학습되고, 또한 이와 함께 이웃한 출력 노드들이 인접한 이웃 공간을 대표하도록 학습이 이루어지는 특징도 나타나게 된다. 이에 대해서는 실험 결과를 이용하여 뒤에서 다시 자세히 설명하겠다.

[프로그램 11-2]의 첫 부분에서 데이터를 생성한 후에 신경망의 구조를 결정한다. 입력 데이터가 2차원이므로 입력 뉴런의 수는 2가 되며, 출력층은 10행 10열의 격자 구조를 가지도록 출력 뉴런을 배치하여 모두 100개의 출력 뉴런이 존재한다.(적절한 출력 뉴런의 수는 문제에 따라 달라지므로, 몇 가지 경우에 대해 수행해 보면서 최적의 값을 찾아주어야 할 것이다.) 신경망 구조가 결정되면 출력 뉴런의 가중치를 작은 랜덤 값으로 초기화하고, 학습률(η)와 이웃의 소속도 함수를 위한 파라미터(σ), 그리고 최대 반복횟수를 설정한다.



[그림 11-22] 2차원 입력 데이터

가중치 초기화를 끝낸 후 수행되는 함수 **drawfeaturemap**은 가중치 벡터에 의해 결정되는 출력층의 특징지도를 그리기 위한 함수로, [프로그램11-2-1]에 그 코드를 나타내었다. 특징지도란, 각 출력 뉴런들이 가지는 가중치의 값을 그 뉴런이 대표하는 값으로 보고, 출력 층의 2차원 격자 구조상에 각 뉴런들이 가지는 가중치를 나타내어 표시한 것이다. [그림 11-23a]에 초기화된 가중치를 가진 SOM에 대한 특징지도를 나타내었다. 출력층은 10행 10열의 격자 구조를 가지고 있으므로, 그림에서도 각 뉴런을 대표하는 정사각형들이 10행 10열로 배치되어 있다. 각각의 정사각형 안에 나타나 있는 선분은, 각 출력 뉴런의 가중치를 2차원 공간상의 벡터로 보고, 원점을 중심으로 하는 공간상에 벡터로 표현한 것이다. 초기화 단계에서는 작은 실수값으로 랜덤하게 할당하였으므로 크기가 작고 그 방향도 무작위인 벡터들이 각 뉴런들의 대표값으로 나타나 있다. [프로그램 11-2-1]의 함수는 가중치와 출력층의 구조(행과 열일 수)를 받아 이러한 특징지도를 그려주는 함수이다.

프로그램 11-2 SOM 학습 프로그램

2차원 데이터에 대한 특징지도를 학습하는 SOM 프로그램

```
001 Dim=2; N=1000;                                  %입력 차원과 학습 데이터 수 설정
002 X=2*rand(1000,Dim)-1;                           %입력 데이터 생성
003 INP=size(X,2);                                  %입력 뉴런수 설정
004 R=10; C=10; OUT= R*C;                           %출력 뉴런수, 행/렬 크기 설정
005 w=rand(INP,OUT)*0.6-0.3;                        %출력 뉴런 가중치 설정
006 W(1,:,:)=w;
007 eta=0.5; sig=1;                                 %학습률, 이웃 소속도 파라미터 설정
008 Mstep=6;                                        %반복횟수 설정
009 drawfeaturemap(1,w, R, C);                      %가중치 초기치에 대한 특징지도 그리기
010
011 for j = 2:Mstep                                  %반복 학습 시작
012   for i=1:N                                     %각 데이터에 대한 학습 시작
013     x=X(i,:);                                   %입력 데이터 선택
014     for k=1:OUT                                 %출력 뉴런 가중치와의 거리 계산
015         d(k)=(x-w(:,k)')*(x-w(:,k)')';
016     end
017     [winv, wini]=min(d);                        %승자 뉴런 선정
018     for k=1:OUT
019         alpha(k)=som_alpha(wini,k,sig,R,C);     %소속도 계산
020         dw(:,k)=(x-w(:,k)')*alpha(k);           %가중치 수정항 계산
021     end
022     w=w+eta*dw;                                 %가중치 수정
023     sum_dw(i)=trace(dw'*dw);                    %가중치 변화량 계산
024     if ((mod(i,10)==0) & (j==2) & (i<=100))
025         drawfeaturemap(i,w, R, C);              %가중치 특징지도 그리기
026     end
027   end
028   sig=sig*0.9;                                  %이웃 소속도 파라미터 감소
029   W(j,:,:)=w;
030   if (mean(sum_dw)) < 0.001                     %가중치 변화가 없을시 학습 완료
031     break;
032   end
033   drawfeaturemap(j,w, R, C);                    %가중치 특징지도 그리기
034 end
```

프로그램 11-2-1 출력 뉴런의 특징지도 그리는 함수 drawfeaturemap

출력 뉴런의 가중치와 행/열의 크기를 입력받아 특징지도를 그리는 함수

```
001 function drawfeaturemap(fid,w,R,C)
002 figure(fid);
003 for i=1:R
004     for j=1:C
005         k=(i-1)*C+j;
006         hold on
007         h=subplot('position',[(j*0.1-0.098) 1-(i*0.1)+0.002 0.095, 0.095]);
008         set(gca, 'XTickLabel',{},'YTickLabel',{});
009         box on; axis([-1 1 -1 1]); hold on;
010         line([0 w(1,k)],[0, w(2,k)],'LineWidth', 3); grid on
011     end
012 end
```

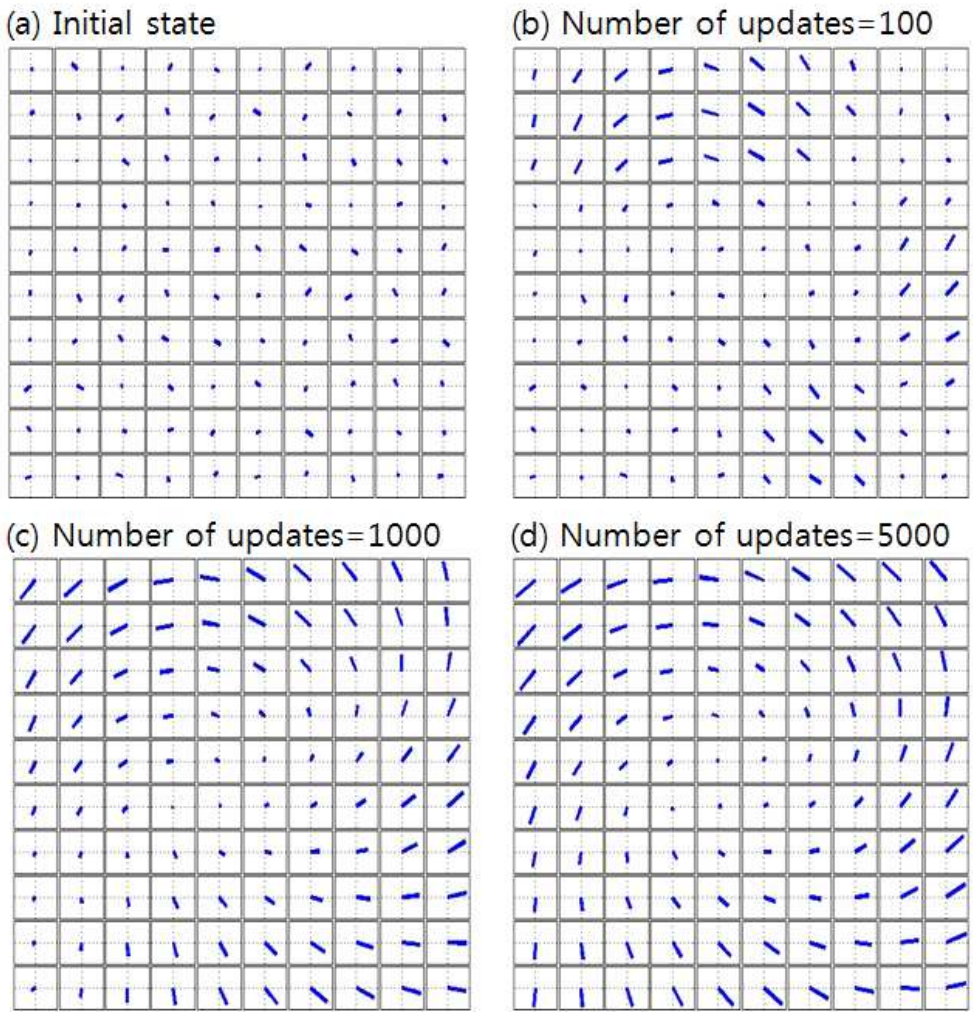
계속해서 [프로그램 11-2]을 살펴보면, 초기화가 수행된 후에 반복 학습을 시작하여, 각각의 데이터에 대하여 모든 출력 뉴런의 가중치와의 거리를 계산하여 승자 뉴런을 결정하고, 각 뉴런들은 이 승자 뉴런의 이웃에 속하는 정도에 따라 비례하여 가중치를 수정하게 된다. 이 때 사용되는 또 다른 함수 **som_alpha**는 승자 뉴런과 현재의 뉴런의 인덱스를 입력으로 받아 그 거리에 따라 이웃의 소속도를 결정하는 함수로, [프로그램 11-2-2]에 구체적인 코드를 나타내었다. 함수의 입력으로 주어지는 것은 승자 뉴런의 인덱스 및 소속도를 알고 싶은 뉴런의 인덱스와 함께 출력층의 구조를 나타내는 행과 열의 값, 그리고 소속도 함수를 시간에 따라 조정하기 위한 파라미터 σ가 있다. 먼저 두 뉴런의 인덱스를 이용하여 각각이 출력층의 격자 구조에서 몇 행 몇 열에 해당하는지를 계산해 내고, 이 값을 2차원 공간상의 좌표로 생각하여 거리를 계산한다. 계산된 거리값을 가우시안 함수에 적용함으로써 0에서 1사이 값으로 정해지는 이웃의 소속도값이 결정된다. 이 소속도 값이 [프로그램 11-2]에 전달되어 승자 뉴런에 가까울수록 학습이 많이 이루어지도록 가중치 수정항이 계산되고 이 값에 학습률 η가 곱하여져 최종적으로 가중치가 수정된다.

프로그램 11-2-1 이웃 뉴런의 소속도 결정 함수 som_alpha

승자 뉴런과 이웃 뉴런, 출력층의 행과 열의 수, 그리고 파라미터 σ를 입력받아 소속도 출력

```
001 function [out]=som_alpha(i,j,sig,NumR,NumC)
002 x(1)=ceil(i/NumC);                    %승자 뉴런의 위치(행)
003 x(2)=mod(i-1,NumC)+1;                 %승자 뉴런의 위치(열)
004 y(1)=ceil(j/NumC);                    %이웃 뉴런의 위치(행)
005 y(2)=mod(j-1,NumC)+1;                 %이웃 뉴런의 위치(열)
006 r=(x-y)*(x-y)';                       %거리 계산
007 out=exp(-r/(2*sig^2))*(1/(sqrt(2*pi)*sig));      %소속도 계산
```

전체 학습 데이터에 대하여 한 번씩 학습이 수행되고 나면 파라미터 σ의 값을 약간 감소시켜서 다시 학습을 반복하도록 한다. 그리고 학습이 수행되는 중간에 **drawfeaturemap**을 호출하여 특징지도의 변화를 살펴볼 수 있도록 하였다. 학습은 가중치 수정폭이 매우 적어서 더 이상 학습이 이루어 지지 않는다고 판단되는 경우나 미리 정해진 횟수만큼 학습이 반복된 후에 완료할 수 있도록 하였다.



[그림 11-23] 학습에 따른 출력 뉴런의 가중치 변화 및 생성된 특징지도

[그림 11-23]에슨 학습횟수에 따른 특징지도의 변화를 나타내었다. 앞서 설명한 바와 같이, 학습 시작 전에는 [그림 11-23a]와 같이 무질서하게 벡터들이 나열되어 있다. 100개의 데이터에 대해 학습이 수행되고 난 후 얻어진 가중치 벡터를 이용하여 그려진 [그림 11-23b]에서는 왼쪽 윗부분과 오른쪽 아랫부분과 같이 일부 영역에서 각 뉴런들이 대표하는 특징이 뚜렷해지고, 또 이웃한 뉴런들도 비슷한 특징을 대표하도록 학습되어짐을 확인할 수 있다. 1000개의 데이터에 대해 모두 학습이 수행된 후 얻어진 [그림 11-23c]에서는 각 뉴런들이 대표하는 특징이 뚜렷해지고, 이웃 뉴런들 간의 관계도 명확해져서 전체적으로 2차원 격자로 배치된 출력층이 하나의 의미 있는 특징지도를 형성하게 됨을 볼 수 있다. 1000개의 학습 데이터에 대한 학습을 5번 반복하고 난 후 얻어진 [그림 11-23d]는 (c)와 거의 유사하나

그 특징이 좀 더 뚜렷해 졌음을 볼 수 있다.
[프로그램 11-1]과 [프로그램 11-2]에 주어진 MLP와 SOM은 2차원 입력 데이터 뿐 아니라 일반적인 경우에도 적용할 수 있도록 만들어졌다. 단지 입출력 크기를 결정하는 변수값만 데이터에 맞게 설정함으로써 다양한 데이터에 대한 분류 및 군집화 문제에 적용해 볼 수 있을 것이다.

연습문제

1. 다음 순서에 따라 XOR 함수를 학습하는 다층 퍼셉트론을 구현하시오.
   (1) 네 개의 학습 데이터를 다음과 같이 만드시오.
      데이터 1: 입력 (0,0) 출력 (0)
      데이터 2: 입력 (1,0) 출력 (1)
      데이터 3: 입력 (0,1) 출력 (1)
      데이터 4: 입력 (1,1) 출력 (0)
   (2) 입력 노드 2개 - 은닉 노드 5개 - 출력 노드 1개로 구성된 다층 퍼셉트론을 만들고, 위에서 생성한 4개의 데이터를 이용하여 학습을 수행하시오.
   (3) 학습이 진행되는 과정에서, 4개의 데이터가 한 번씩 학습에 사용되고 난 뒤, 신경망의 출력 오차를 계산하여 그 오차가 줄어드는 과정을 그래프로 나타내시오.
   (4) 학습이 완료된 후, 다음 데이터에 대한 출력값을 계산해 보시오.
      데이터 1: 입력 (0.1, 0.1)
      데이터 2: 입력 (0.9, 0.9)
      데이터 3: 입력 (0.1, 0.9)
      데이터 4: 입력 (0.9, 0.1)

2. 같은 데이터에 대해, 다층 퍼셉트론의 구조를 다음과 같이 변경하여 학습해 보고, 학습 오차의 변화 과정을 비교해 보시오.
   (1) 입력 노드 2개 - 은닉 노드 3개 - 출력 노드 1개
   (2) 입력 노드 2개 - 은닉 노드 1개 - 출력 노드 1개
   (3) 입력 노드 2개 - 은닉 노드 10개 - 출력 노드 1개

3. 2차원 데이터를 이용하여 SOM을 학습하되, 다음과 같은 처리 순서를 따르시오.
   (1) 0에서 1사이의 좌표값을 가지는 2차원 데이터를 200개 생성하고, 2차원 평면상에 표시하시오.
   (2) 출력층의 구조는 10행 10열의 2차원 격자 구조를 가지도록 설정하시오.
   (3) 가중치의 초기값은 0.5를 평균으로 하여 표준편차가 0.1이 되는 정규분포를 따르도록 랜덤하게 설정하시오.
   (4) (2)에서 결정된 가중치 값을 가지는 노드들을 2차원 평면상에 표시하고, 이웃하는 노드들끼리 선으로 연결하여 나타내시오.
   (5) 학습률을 0.2에서 시작하여 서서히 줄이면서 학습을 수행하고, 이웃 반경도 적절히 조정하여 학습을 수행하면서 변화된 가중치를 이용하여 (3)과 같은 그림으로 표현하고

그 변화를 살펴보시오.

4. [식 11-11]에서 활성화함수가 시그모이드 함수인 경우와 하이퍼탄젠트 함수의 경우에 각각 그 도함수를 계산하여 z에 관한 식으로 나타내어 보시오.

5. 다층퍼셉트론의 학습에서 과다적합이 발생하는 것을 방지하기 위한 방법의 하나로 페널티 (penalty) 항 (혹은 정규화 (regularization) 항)을 추가하는 방법이 있다. 대표적인 페널티항으로는 가중치벡터의 크기가 있는데, [식 11-10]에서 정의된 오차함수에 가중치벡터의 크기를 더하여 다음과 같이 새로운 오차함수가 정의될 수 있다.

$$E_R(x, \boldsymbol{\theta}) = (t_k - f_k(x, \boldsymbol{\theta}))^2 + \boldsymbol{\theta}^T\boldsymbol{\theta}$$

이와 같이 새롭게 정의된 오차함수에 대한 기울기강하 학습식을 유도해 보시오.

6. SOM의 학습에서 대표적인 출력층의 형태는 2차원 격자 형태를 가진다. 그러나 이에 대한 변형으로 1차원 배열이나 6각형 형태의 이웃을 가지는 배열도 생각해 볼 수 있다. 이 밖에도 다양한 형태로 출력층에 변형을 줄 수 있다. 한 가지 형태를 생각하여 그에 적합하게 매트랩 코드를 구현해 보시오.

참고자료

이 장에서는 대표적인 신경망 모델인 다층퍼셉트론과 자기조직화 신경망에 대하여 알아보았다. 이밖에도 매우 다양한 신경망 모델들이 존재하는데, 이와 관련하여서는 [Haykin 99]와 [Bishop 96]을 참고하기 바란다. 또한 신경망 초기 모델에 대해서는 [Wasserman 89]에 알기 쉽게 소개되어 있다.

[Haykin 99] S. Haykin. Neural Networks: a Comprehensive Foundation, Prentice Hall, 1999.
[Bishop 96] C. M. Bishop. Neural Networks for Pattern Recognition. Oxford University Press, 1995.
[Wasserman 89] P. D. Wasserman, Neural computing theory and practice, Van Nostrand Reinhold, 1989.