11. NN Intro
2019년 가을 학기

2019년 08월 18일
https://www.github.com/KU-BIG/KUBIG_2019_Autumn

이 글은 백준걸 교수님의 데이터 마이닝 강의의 강의안과 박희원님의 슬라이드[1]를 많이 참고하였다.

이 강의는 4개의 파트로 구성된다. 딥러닝의 전반적인 부분인 Introduction, 기초인 Perceptron 및 Multi Layer Perceptron, 핵심인 Forward Propagation과 Backward Propagation으로 구성되었다. 자세한 수식이나 원리까지 설명되어 있지 않지만 전반적인 딥러닝의 구성, 작동방법 등이 포함되어 있다.

1. Introduction

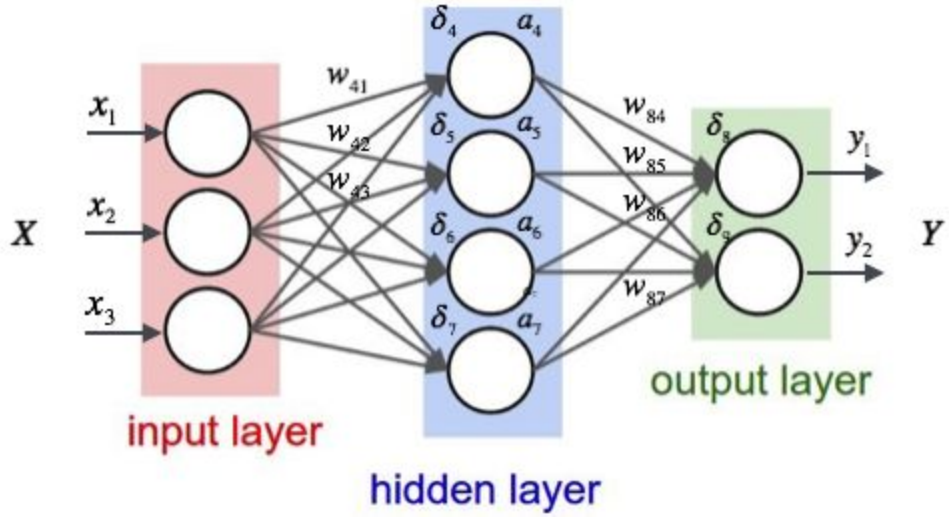
딥러닝 ⊂ 머신러닝. 많은 사람들이 딥러닝을 완전히 다른 분야로 알지만 사실 머신러닝에 속한 심층신경망(Deep Neural Network)을 이용한 기법이다. 심층신경망은 직관적으로 해석해도 깊은 신경망이라는 걸 알 수 있다. Deep Learning은 다른 기법에 비해 Neural Network가 deep하다는 것, 즉 학습이 반복된다는 것을 의미한다. 사실 딥러닝은 그렇게 각광받는 분야가 아니었지만 점차 발전해가며 오늘날에 이르렀다.

1986년 Backward Propagation이라는 모델을 적합하는 획기적인 방법이 고안되었다. 간단히 설명하자면 Backward Propagation은 output layer에서 input layer로 뒤로 가면서 weight, bias 등 factor를 재계산하고 갱신하면서 모델의 성능을 높인다.

딥러닝이 각광받지 않았던 이유는 층이 깊어질수록 연산량이 기하급수적으로 늘어나 최적값을 얻기 힘들었기 때문이다. 많은 연산량을 처리하는 효과적인 알고리즘이 고안되기까지 많은 시간이 흘렀지만 결국 대표적인 문제인 time complexity, vanishing gradient, overfitting의 해결방안이 나오기 시작하며 딥러닝이 주목받기 시작했다.

딥러닝은 input layer(입력층) , hidden layer(은닉층), output layer(출력층)에서 hidden layer가 deep한, 즉 2개의 이상의 layer가 있는 모델들을 말한다.

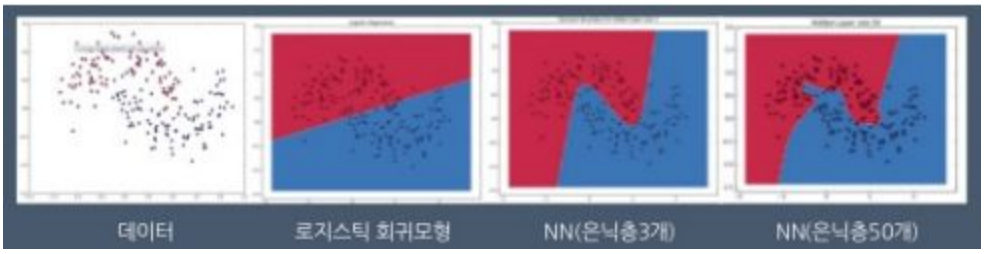
[1] https://www.slideshare.net/HeeWonPark11/ss-80653977



https://miro.medium.com/max/479/1*QVIyc5HnGDWTNX3m-nIm9w.png

위의 그림을 보면 알 수 있듯이 input으로 들어온 데이터가 hidden layer를 거쳐서 output으로 나오게 된다. 이때 hidden layer가 2개 층 이상으로 깊어지면 deep learning이다.

딥러닝의 핵심은 hidden layer의 여러 결합(비선형 활성함수, Activation function)으로 비선형 영역을 표현하는 것이다. 아래의 그래프를 보면 hidden layer가 늘어날수록 Classification이 더 정확한 것을 볼 수 있다.



https://www.slideshare.net/HeeWonPark11/ss-80653977

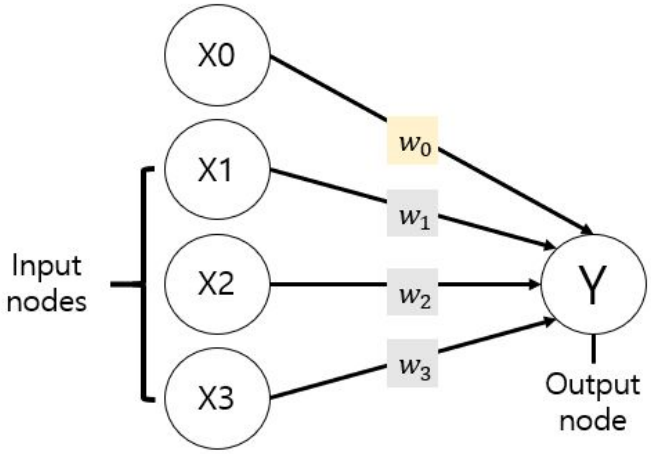
그렇다고 무조건 hidden layer를 깊게 만들면, 레이어를 더하고 더하면 높은 성능의 모델을 만들 수 있을까? 세상은 우리에게 친절하지 않다. 레이어를 계속 더하면 major한 문제점 3가지가 뚜렷하게 드러나게 된다. 앞서 언급한 time complexity(느리다), vanishing gradient(덜하다) 그리고 overfitting(과하다)이 큰 문제가 된다. 12강 Training Neural Network에 문제점 3가지를 해결하는 방안이 설명되었으니 참고하기 바란다.

2. Multi Layer Perceptron

우리는 앞 장에서 딥러닝이 무엇인지 대략적으로 살펴봤지만 아직 가장 중요한 부분을 살피지 않았다. 딥러닝은 deep한 Neural Network라는 것을 배웠지만 정작 Neural Network라는 것을 보지 않은 것이다! 따라서 이 장에서는 deep해지기 전의 Neural Network인 MLP(Multi Layer Perceptron)과 그것의 기본 단위인 Perceptron에 대해 다뤄볼 것이다.

2.1 Perceptron

Perceptron은 신경망을 구성하는 가장 기본적인 단위이다. Input node들의 선형 결합으로 output node를 도출한다. 가장 기본적인 신경망 모델 답게 Perceptron을 구성하고 있는 것은 매우 간단하다. Input node, output node, input node와 output node를 연결할 arc. 딱 세 가지로 표현된다. 이는 밑의 그림을 보면 시각적으로도 간단함을 느낄 수 있다.

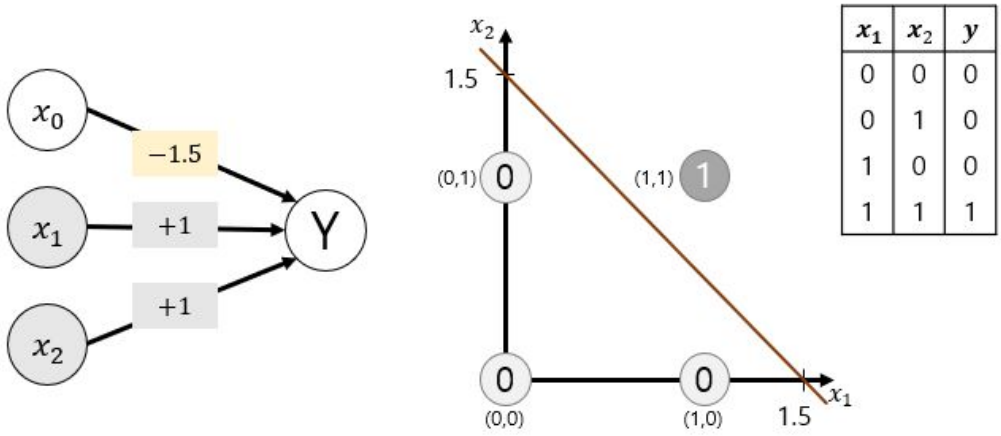


Perceptron Model
$Y = I(\sum_j w_j X_j - w_0)$
$Y = sign(\sum_j w_j X_j - w_0)$
$Y = w_0 + \sum_j w_j X_j$

위의 그림에서 X1, X2, X3은 input data에 해당하는 input node들이다. 이 input node들은 output node인 Y까지 각자의 가중치(weight) w1, w2, w3을 가지고 연결되어있다. 여기서 X0은 input layer에 같이 표시하였는데 input node라 하지 않는 것이 의아할 수도 있다. X0이 가지고 있는 arc에는 w_0은 weight가 아닌 bias이기 때문이다. bias라는 용어는 통계학에서도 존재하지만 여기서는 그 bias가 아닌 간단하게 말하자면 Linear Regression에서의 Intercept에 해당한다.

Perceptron Model은 총 3가지로 표현했다. 위의 2개는 Classification을 수행하는 model이고 밑은 Regression을 수행하는 model이다. 첫 번째 모델을 사용하면 output node는 0과 1만을 가질 것이고, 두 번째 모델을 사용하면 -1과 1만을 가질 것이다. 물론, 마지막 모델을 사용하면 Regression결과와 같이 연속적인 값을 output으로 가질 것이다.

그렇다면 이러한 간단한 신경망으로 어떻게 data들을 분류하는지 간단하게 살펴보자.



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

위의 그림은 Perceptron을 이용하여 주어진 Boolean Data를 학습시킨 결과이다. 주어진 data의 output인 y인 x1 AND x2를 통해 구할 수 있으니 이는 즉 Perceptron으로 AND 연산을 표현한 것으로 봐도 된다. 이 Perceptron model은 $\hat{Y} = I(x_1+x_2-1.5)$로 구성되어있기에 좌표평면에 그어진 선의 밑 부분은 0으로 윗 부분은 1로 잘 분류하고 있다.

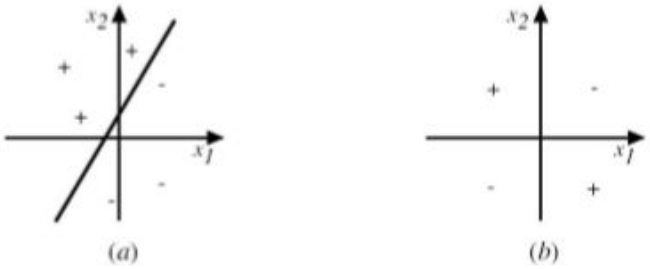
이 Perceptron model을 최적화하기 위해서는 올바른 weight(w_1, w_2)과 bias(w_0)을 구해야한다. 이러한 가중치들을 구하는 방법은 여러가지다. Perceptron model로 구해진 예측값 $\hat{y}$과 y간의 오차를 최소화하는 Gradient Descent 방법을 써도 된다. 여기서 핵심은 y와 $\hat{y}$간의 차이를 최대한 줄이는 방향으로의 가중치를 최적화하는 것이다.

$$w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}_i^{(k)})x_{ij}^2, \ \lambda : \text{learning rate (학습 속도)}$$

$$\text{Gradient descent} : w_j^{(k+1)} = w_j^{(k)} + \lambda \frac{\partial}{\partial w_j} ||y_i - \hat{y}_i^{(k)}||_2^2$$

2.2 Multilayer Perceptron (MLP)

우리는 앞서서 신경망의 기초인 Perceptron에 대해서 알아보았다. 이 Perceptron에는 한가지 치명적인 단점이 존재한다. 모델의 형태가 너무나도 단순하여 단순한 Decision Boundary (Class를 나누는 기준선) 이상의 것을 만들어내지 못한다. 하지만 세상은 간편하게 선형의 기준으로 모든 것을 나눌 수 없다. 아래의 그림을 보자.

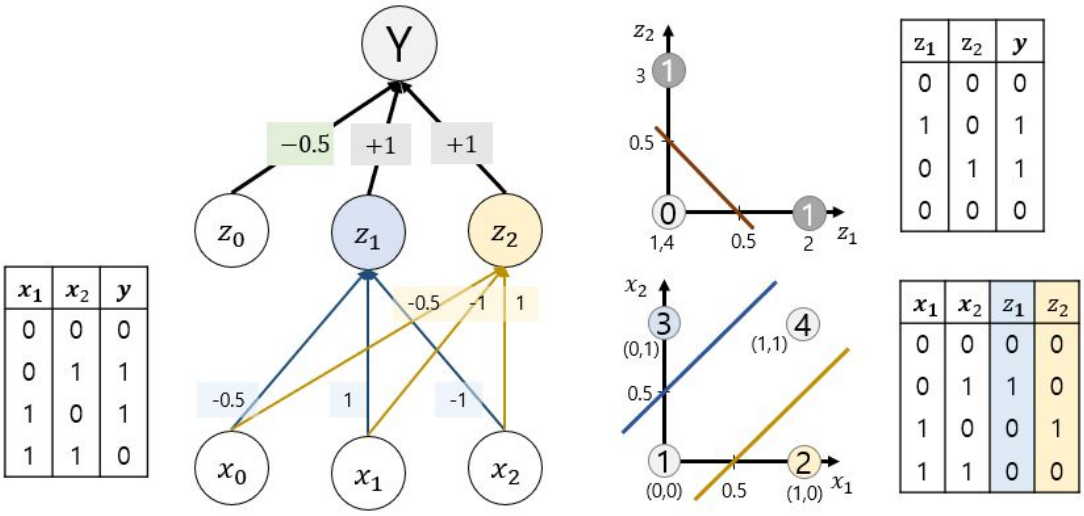


(a)의 경우 고맙게도 직선 하나로 +와 -을 구분할 수 있다. 이러한 경우 Perceptron model을 사용하면 된다. 하지만 (b)에는 그 어떠한 방법을 쓰더라도 직선 하나로는 절대로 올바른 분류를 할 수 없다. 이러한 상황에서 Decision Boundary를 하나만 사용하지 않는다면 이 문제는 해결될 것이다. 그리고 이 개념이 바로 Multilayer Perceptron이다.

Multilayer Perceptron은 Decision Boundary 하나로는 class를 제대로 나눌 수 없으니 여러개의 Decision Boundary를 두자는 아이디어이다. 그리고 별로 어렵게 생기지도 않았다. Input layer와 output layer밖에 없었던 기존의 Perceptron model 중간에 hidden layer을 두는 것이다.

MLP에서 hidden layer의 node의 수가 곧 Perceptron의 수이자, Decision Boundary의 수가 된다. Hidden layer의 수가 많아질수록, hidden layer의 차원이 커질수록 우리는 더욱 복잡한 boundary로 class를 나눌 수 있으며 점점 우리가 원하는 결과에 근접할 것이다. 과적합이라는 장애물을 생각하지 않는다면 말이다.

위의 그림에서 (b)의 경우는 사실 XOR 문제이다. 이를 해결하는 MLP에 대해서 알아보는 것을 마지막으로 이 장을 마치고자 한다. x_1 XOR x_2 문제를 해결하는 MLP이며, x_1 XOR x_2는 (x_1 AND ~x_2) OR (~x_1 AND x_2)에 해당한다. 아래의 그림을 통해서 hidden layer의 추가가 class분류에 어떠한 영향을 미치는지 살펴보자.



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

z_1	z_2	y
0	0	0
1	0	1
0	1	1
0	0	0

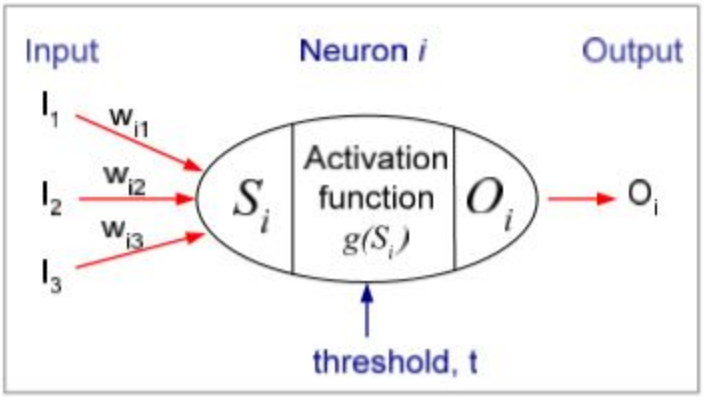
x_1	x_2	z_1	z_2
0	0	0	0
0	1	1	0
1	0	0	1
1	1	0	0

MLP는 input에서 hidden layer로 가면서 input data를 hidden layer의 차원으로 변형시킨다. 그 후 hidden layer에서 output layer로 가면서 변형된 data를 토대로 적절하게 class를 분류한다. 위의 그림을 보면 밑의 그래프에서 input data를 hidden layer의 차원을 가지는 data로 변형시킨다. 파란선보다 위에 있는 3만 z_1에 1의 값을 가지며, 노란 선보다 아래에 있는 2만 z_2에 2의 값을 가진다. 위의 그래프를 보면, 0의 값을 가졌던 1, 4가 제대로 0으로 분류되었으며, 1의 값을 가지던 2, 3이 위의 그래프에서 제대로 1로 분류된다.

우리는 여기서 한 가지 의문점을 가질 수 있다. 아무리 Decision Boundary를 많이 만든다고 해도 그 boundary는 단순한 선형이다. 그러면 좀더 복잡한 형태를 지닌 Decision Boundary는 만들 수 없는 것인가? 만약 그럴 수 없었다면 딥러닝은 크게 성공하지 못했을 것이다.

2.3 Activation Function (활성화 함수)

좀 더 flexible한 boundary를 찾기 위해서는 Activation function을 이용하면 된다. Perceptron과 위에서 다룬 MLP는 선형 결합을 기초로 한다. 따라서 선형의 boundary를 형성할 수 밖에 없는데 여기에 비선형의 활성화 함수를 연결해 줌으로써 boundary를 찌그러뜨릴 수 있다. MLP의 각 Perceptron에서는 앞선 layer의 node들의 선형 결합의 결과를 output으로 가진다. 이때 이 output의 값을 그대로 비선형적인 활성화 함수에 넣어주면 비선형 boundary가 탄생한다. 물론 이를 위해서는 연산량이 늘어나는 것은 감수해야한다.



위의 그림은 Activation function을 적용한 한 Perceptron을 나타낸다. 학습된 weight을 토대로 선형결합한 결과인 S_i를 다시 Activation function에 넣어서 output ($O_i = g(S_i)$)을 뽑아낸다. 이러한 Activation function은 각 층에서 다음 층으로 넘어갈 때 적용한다. 그렇다면 Activation function들은 어떠한 형태를 가지고 있을까?

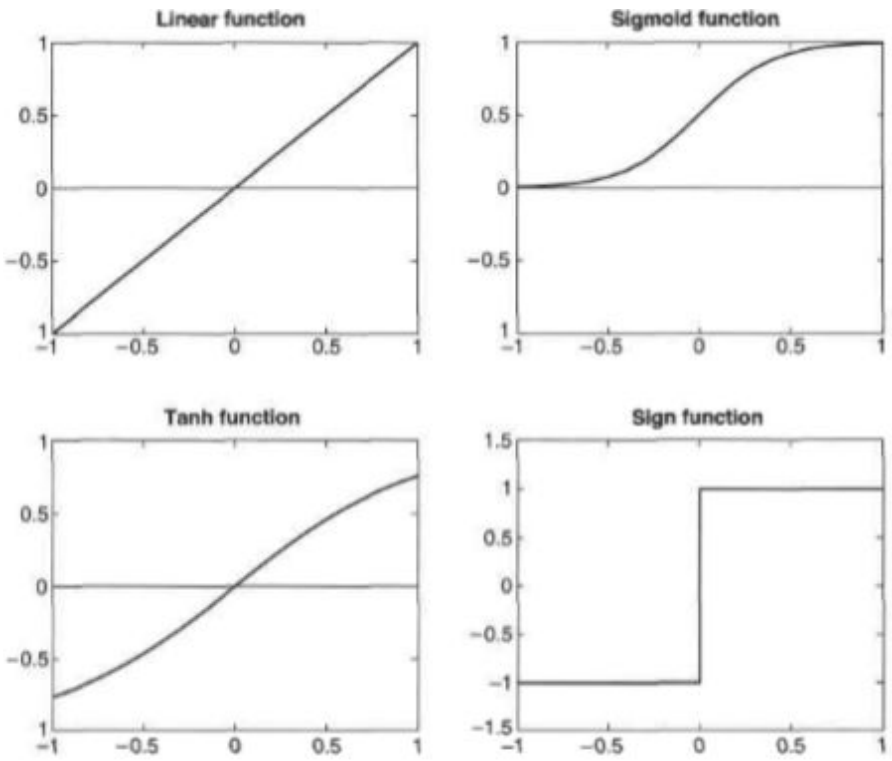


Figure 5.18. Types of activation functions in artificial neural networks. [2]

Activation function은 여러 종류가 있다. Linear function은 아무런 효과가 없다. 나머지는 linear boundary를 찌그러뜨리는 효과를 지닌다. Sigmoid function의 경우 그 output이 0, 1사이에 있으며, tanh function은 -1, 1사이에 있게 된다. Sign function 또한 Activation function으로 활용할 수 있는데 이 함수는 미분 불가 지점이 있어 신경망 학습 시 미분을 활용하는 최적화 알고리즘은 사용할 수 없다는 단점이 있다.

이 외에도 많은 Activation function이 있으며, 좀 더 다양한 Activation function에 대한 설명은 다음 강의에 담겨 있으니 참고하길 바란다.

3. Forward Propagation

앞 장에서는 딥러닝의 기초가 되는 Perceptron과 MLP, Activation function에 대해서 살폈다. 그리고 Introduction을 통해서 hidden layer가 하나 있는 MLP에 hidden layer를 추가하여 2개 이상의 hidden layer을 갖도록 하면 DNN(Deep Neural Net)이 되는 것을 알게 되었다. 하지만 아직까지도 어떻게 하면 이러한 신경망이 학습하는지에 대해서 배우지 않았다.

[2] Tan, et al.(2005), 'Introduction to Data Mining'(First Edition). Pearson Education. p.252.

딥러닝은 학습을 앞으로도 하고 뒤로도 한다. 앞으로 하는 학습은 순전파(Forward Propagation), 뒤로 하는 학습은 역전파(Backward Propagation)라고 한다. 순전파는 미리 계산된 weight와 bias들로 output을 낸다. 역전파는 그렇게 구해진 output과 실제 값과의 차이, 즉 오차를 뒤로 보내면서 weight와 bias들을 학습한다. 이번 장에서는 순전파에 대해서 간단하게 다뤄보겠다.

각 layer에서는 이전 layer의 node들을 주어져 있는 weight와 bias로 선형 결합하여 S_i를 구한다. 이후 S_i를 Activation function에 적용하여 output $g(S_i)$를 구한다. 그 뒤, 이 output들을 input으로 하여 다시 다음 layer로 넘긴다. 이 때 한 가지 조심하자. Activation function을 적용하지 않으면 Decision Boundary가 선형으로 결정된다. 또한 Activation function은 hidden layer로 넘어갈 때 적용되는 것으로 output layer로 넘어갈 때는 다른 함수를 사용한다.

Output layer에서는 분류를 위해 softmax라는 함수를 사용한다. 쉬운 예를 들어보자. 우리는 어떤 사진이 고양이인지를 인식하도록 잘 학습시키고 싶다. 그래서 딥러닝으로 학습시키고 이제 데이터를 신경망에 흘려 고양이인지 아닌지 결정해야 하는데 이때 softmax를 사용하여 결과를 출력한다. 더 자세히 말하자면 softmax 함수는 output layer에 전파된 input들을 벡터로 받아 각 성분의 크기 비율(출력 벡터 성분합이 1)을 출력하는 함수이다.

신경망 모델에서의 계산 과정을 정리하면 다음과 같다.

1) 입력층에 데이터가 입력되면 각 뉴런들은 arc에 부여된 가중치를 고려하여 다음 층으로 데이터를 전달한다.
2) 데이터를 전달받은 뉴런들은 다음 층으로 데이터를 전달하기 위해 활성화 함수(Activation function)를 통해 전달할 값을 계산한다.
3) 다시 다음 arc에 부여된 가중치를 고려하여 데이터를 전달한다.
4) 이러한 과정이 출력층까지 이루어지면 출력층 또한 활성화 함수를 통해 출력할 값을 계산하여 최종적인 값을 출력한다.

4. Backward Propagation

앞 장에서 간단히 살펴본 Forward Propagation을 통해서 결과가 나온 후, input과 output간의 차이로 오차가 관측된다. 이때 오차의 기울기를 output layer에서 input layer로 역으로 전달하는 것이 Backpropagation이다. 각 layer에 전달된 기울기를 통해서 각 layer별로 weight, bias를 업데이트한다. 우리의 목적은 최적화된 weight 와 bias를 찾는 것이다. 따라서 Cost function의 기울기를 구해서 Cost를 최소화 할 수 있는 방향으로 갱신한다.

신경망은 구체적으로 어떻게 학습되는가? 신경망이 학습하는 것을 순서대로 나열해보면, input layer로부터 각 layer를 지나치며 weight들을 곱하고, Activation function을 통해 선형 결합을 비선형 결합으로 변환한다. 이후 Output layer에서 오차 값을 계산하고 weight의 업그레이드를 위해 순차적으로 거꾸로(back) 돌아가야(propagation)한다. 밑은 자세한 수식과 알고리즘이다.

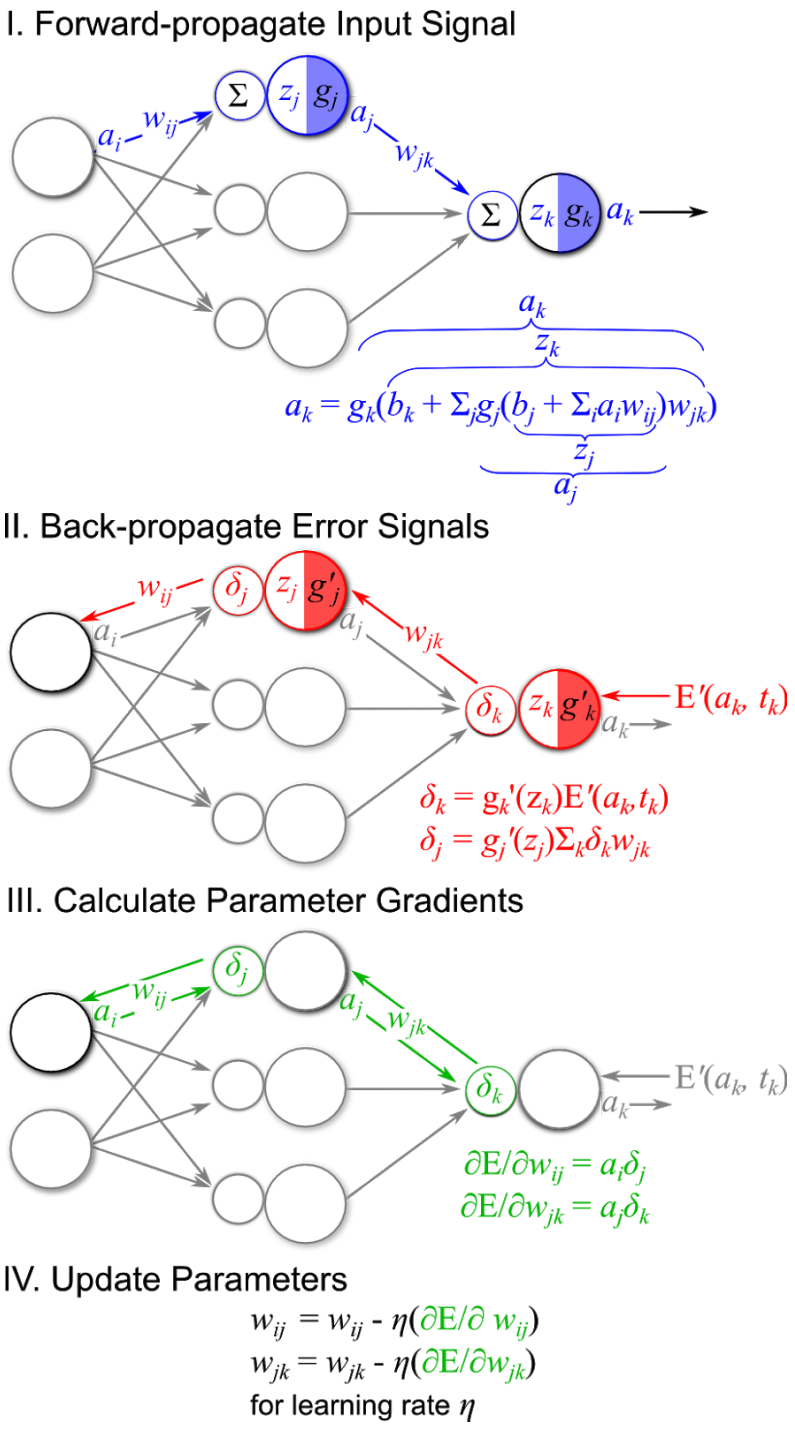
1. 전방향 연산(Forwardpropagate)을 수행해서 Hidden Layer 1-L_2-, Hidden Layer 2-L_3-, 계속해서 최종적으로 Output Layer-L_n-까지 노드들의 activation을 계산한다.
2. Output Layer-n_l-의 각각의 노드(output unit) i에 대해서 아래와 같이 에러값($\delta_i^{(n_l)}$)을 계산한다.
$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} ||y - h_{W,b}(x)||^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$
3. Output Layer를 제외한 각각의 노드(output unit) i에 대해서 아래와 같이 에러값($\delta_i^{(n_l)}$)을 계산한다.
$$\delta_i^{(n_l)} = (\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)}) f'(z_i^{(n_l)})$$
4. Neural Networks의 파라미터들(parameters)-W, b-에 대한 미분값(derivative)를 아래와 같이 계산한다.
$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)}$$
$$\frac{\partial}{\partial b_{ij}^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)}$$

http://solarisailab.com/archives/2112

Cost function을 최소화 시키기 위해 error를 output에서 구해서 input을 변환시킬 수 없으니 그 사이 hidden layer들에서 weight를 chain rule로 미분 계산하여 최소화시킨 결과를 갱신한다. 이를 통해 weight의 최적화를 수행하여 network의 성능을 개선한다. 자세한 수식과 설명은 상당히 수학적이고 많은 이론이 들어가기에 참고를 위해 링크를 첨부한다.

https://kakalabblog.wordpress.com/2017/04/04/understanding-of-backpropagation/

총정리 :



https://theclevermachine.files.wordpress.com/2014/09/fprop_bprop5.png