Softmax Regression

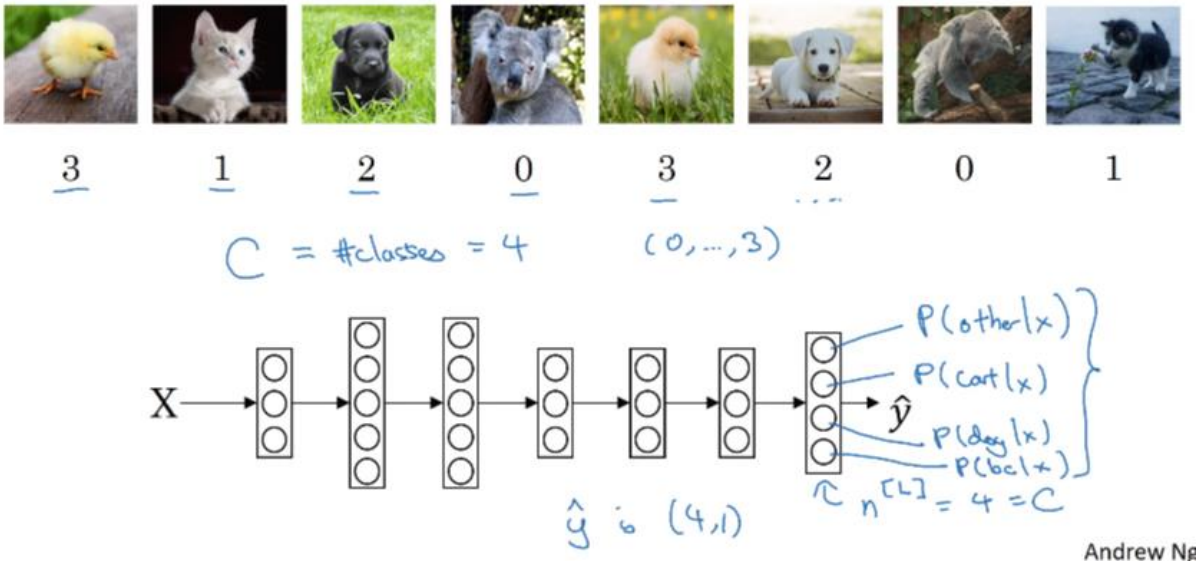
지금까지 우리는 0 과 1 의 두가지 선택이 있는 이진분류 문제에 대한 문제를 봤다**. 여러 개의 선택지가 주어진다면 어떨까? 로지스틱 회귀를 일반화한 소프트맥스 회귀**가 있다. 이것을 클래스가 두 개인 경우에 여러 클래스나 C 중 하나를 인식할 때 예측에 사용할 수 있다.

Recognizing cats, dogs ,and baby chicks other

개, 고양이, 병아리를 인식해보자. 고양이는 클래스 1, 개는 클래스 2, 병아리는 클래스 3 이다. 그리고 여기에 해당하지 않는 동물은 클래스 0 이다. 아래 사진에 이미지와 속해 있는 클래스를 적어두었다. 여기에서 **대문자 C 는 클래스의 숫자를 나타내는데 사용**해 여러분의 입력값을 분류할 것이다. 지금은 C 가 아무 것도 아닌 경우를 포함해 4 의 값을 갖는다. (**C=#classes=4**) 클래스에 숫자를 붙인다면 0 에서 C-1 까지 부여가 된다.

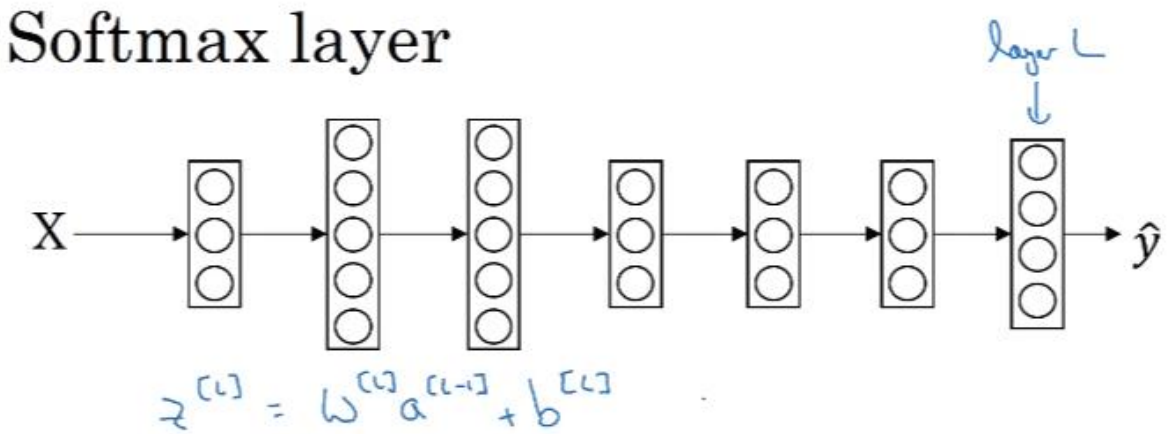
그리고 아래와 같은 신경망을 하나 만들었다고 가정하자. 출력층에는 C 개(이 경우 4 개의 출력 단위가 있는 신경망), 출력층 L 의 단위 개수인 n 은 4, 또는 일반적으로 C 가 될 것이다. 우리는 각 단위와 상위층에서 각 클래스의 확률을 알려줬으면 한다. 첫번째 단위에서 우리가 원하는 출력값은 입력값 X 가 주어졌을 때 기타 클래스, 두번째 단위에서는 고양이의 확률, 세번째 단위에서는 개의 확률, 네번째 단위에서는 병아리의 확률이다(병아리는 BC 로 줄여 필기).

따라서 출력값인 **y^은 출력값**으로 **네 개의 확률값이 주어지므로 (4,1)차원의 벡터**가 될 것이다. 그리고 y^의 각 값들의 합은 1 이 되어야 한다. **이런 신경망을 얻기 위한 가장 표준적인 모델은 출력층에 이런 출력값을 만들 수 있도록 소프트맥스층을 사용하는 것이다.**



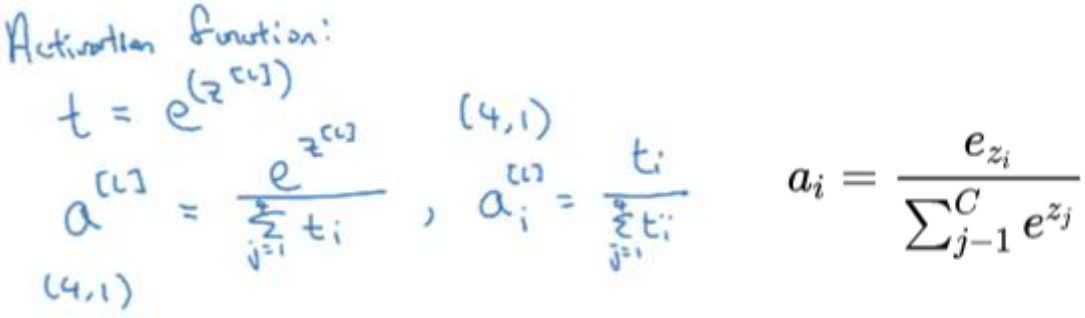
Softmax layer

출력층에 이런 출력값을 만들 수 있도록 소프트맥스 층을 사용해보자. 큰 그림을 그린 뒤에 다시 돌아가서 소프트맥스가 뭐하는 건지 살펴보자.

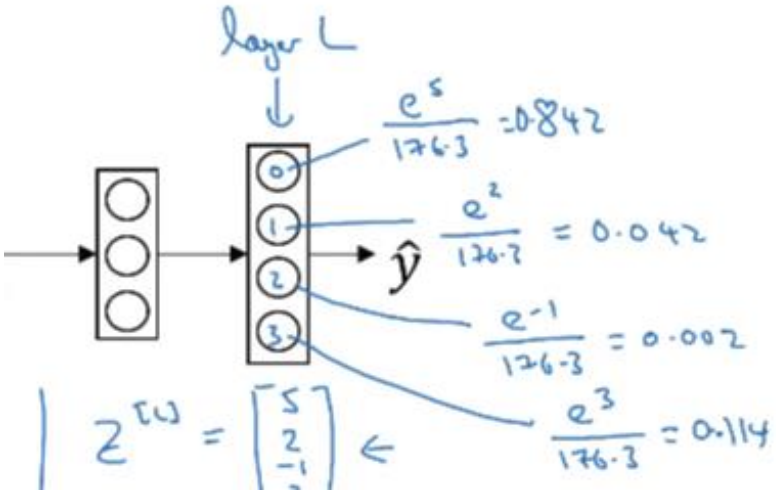


신경망의 **최종층(L)**에서는 평소처럼 층의 선형적인 부분인 **z^[L]을 계산**할 것이다. 최종층인 L 의 z 값이다. z^[L]을 w^[L]과 이전 층의 활성화 함수(a^[L-1])를 곱한 뒤 그 최종층의 편향(b^[L])을 더하여 계산한다.

=> **z^[L] = w^[L]*a^[L-1]+b^[L]**



이제 z 값을 계산하기 위해서 **소프트맥스 활성화 함수**라는 것을 사용해야 한다. 소프트맥스 층의 활성화 함수는 조금 다르다. 우선 각 원소에 대해 계산하는데 **t=e^(z^[L])이라는 임시 변수를 사용**한다. 즉 z^[L]이 (4,1) 차원의 벡터인데 t 는 z^[L]의 각 원소에 e 를 취한 것이다. 즉 결과도 (4,1)이 된다. 그러면 출력값인 a^[L]은 벡터 t 와 같다. 다만 **합이 1 이 되도록 모든 임시 변수값들의 합을 나눠서 정규화**한다. (즉 a^[L]은 z^[L]을 j 가 1 부터 4 까지(원소가 4 개이므로) t_j 를 모두 더한 값으로 나눈다. 그러면 a^[L]은 (4,1)벡터이고 이 벡터의 i 번째 원소(a^[L]_i)는 t_j 를 t_i 값들의 합으로 나눈 것과 같다)



$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$$

$$t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148 \cdot 4 \\ 7 \cdot 4 \\ 0 \cdot 4 \\ 20.1 \end{bmatrix} \cdot \sum_{j=1}^{4} t_j = 176.3$$

$$a^{[L]} = \frac{t}{176.3}$$

예시를 하나 살펴보자. 예를들어 (4,1) 벡터인 z^[L]이 [5, -2, 1, 3]이라고 하자. 우선 원소별로 e 를 취해 t 를 구한다. t 는 e 의 5 승, e 의 2 승, e 의 -1 승, e 의 3 승이 된다. 계산하면 각각 148.4, 7.4, 0.4, 20.1 이 된다.

이제 t 에서 a^[L]으로 합이 1 이 되도록 정규화시키기 위해 t 의 4 개의 합을 구하면 176.3 이 된다. 이제 a^[L]은 t 를 176.3 으로 나눈 것이 된다.

즉 예시에서 첫번째 노드의 값은 e^5 을 176.3 으로 나눈 0.842 이다. 만약 이런 z 값을 얻었다면 클래스 0 이 될 확률이 84.2%인 것이다. 다음 노드의 출력값은 e^2 을 176.3 으로 나눈 값인 0.042 이다. 다음은 e^(-1)을 176.3 으로 나눠서 0.02 이다. e^3 을 176.3 으로 나눠서 0.114 값을 얻는다. 이는 11.4%의 확률로 클래스 3(병아리)이 되는 것이다. 이런 식으로 클래스 0.1.2.3 이 될 확률을 구할 수 있다.

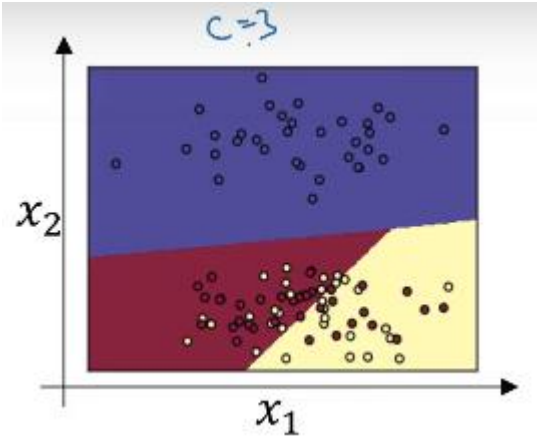
이 신경망의 출력값 y^과도 같은 a^[L]은 (4,1)의 벡터가 되겠고 그 안에는 계산한 숫자들(0.842, 0.042, 0.002, 0.114)이 들어가 있을 것이다. 이 알고리즘은 z^[L]이라는 벡터를 취해서 합이 1 이 되는 4 개의 확률 값을 내 놓는다. z^[L]에서 a^[L]으로 되는 과정을 요약하자면 e 를 취해서 임시 변수 t 를 얻어서 정규화한 이 과정을 소프트맥스 활성화 함수로 요약할 수 있다. 즉 a^[L]은 z^[L] 벡터에 활성함수 g^[L]을 적용한 것이다. 이 **활성화 함수 g 의 특이한 점은 (4,1)벡터를 받아서 (4,1)벡터를 내놓는다는 것**이다. 이전에는 활성함수가 하나의

실수값을 받았다. 예를들어 시그모이드나 Relu 활성화 함수 등은 실수를 받아서 실수를 내놨다. **소프트맥스 활성화 함수의 특이한 점**은 **정규화를 하기위해서 입력값과 출력값이 모두 벡터**라는 것이다.

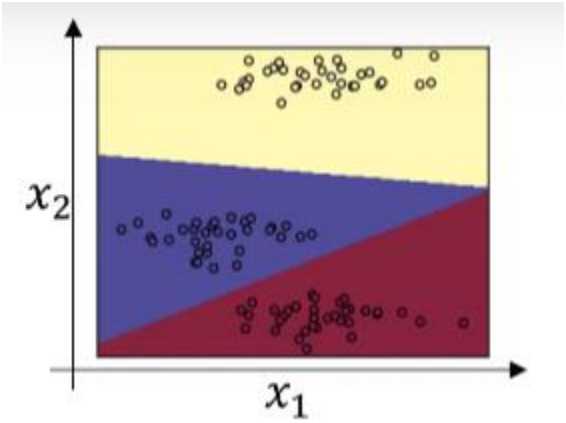
Softmax examples

소프트맥스 분류로 할 수 있는 것을 보이기 위해 x_1 과 x_2 의 입력값이 있다고 가정해보자. 이 값들은 바로 소프트맥스 층에 들어간다. 안에는 서너개의 노드가 있고 출력값은 y^이다. 이런 식으로 은닉층이 없는 신경망을 보인다. 여기에서 하는 일은 z^[1]을 w^[1]과 x 를 곱한 뒤 b^[1]을 더하여 계산하고 출력값 y^이자 a^[1]을 z^[1]에 소프트맥스 활성화함수를 적용시켜 얻는다. 은닉층이 없는 신경망에서 소프트맥스 함수가 무엇을 하는지 감을 잡아보자.

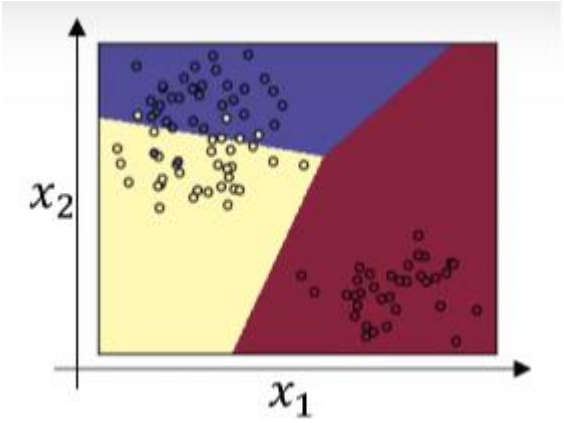
i) 이 예시에서는 입력값 x_1 과 x_2 에 대하여 이 결정 기준을 나타내는 c=3 의 클래스를 가진 소프트맥스 층을 사용했다. 여기서 선형적인 기준에 따라 데이터가 세개의 클래스로 나뉜다. 우리가 했던 것은 이 그림처럼 학습세트를 가져와서 비용 함수와 세 개의 선택지에 따라 분류하는 소프트맥스 함수를 학습시키는 것이다. 색깔은 소프트맥스 분류 함수에 따라 출력값을 나타낸 것이고 입력값은 가장 높은 확률의 출력값에 따라 색을 입혔다. 선형기준을 가지고 있는 로지스틱 회귀의 일반적인 형태이다. 하지만 클래스는 0,1 또는 0,1,2 가 될 수 있다.



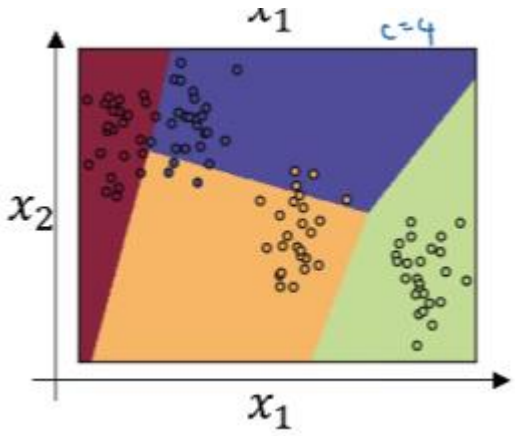
ii) 이는 소프트맥스 분류 함수가 나타낸 또 다른 경우이다. 세개의 클래스에 따라 데이터를 학습시켰다.



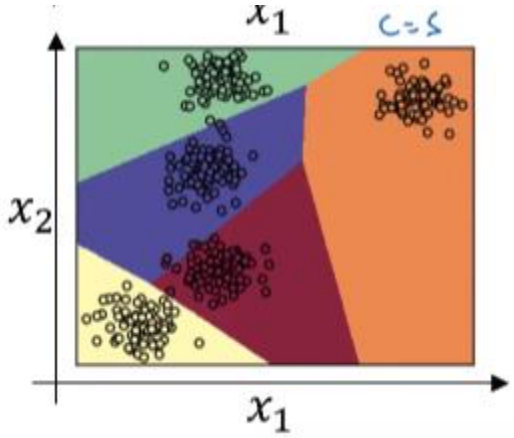
iii) 이는 또 다른 분류다. 여기서 얻을 수 있는 직관은 두 클래스 사이의 경계가 선형이라는 것이다. 그래프를 보면 노란색과 빨간색 사이에도 선형 경계가 그려져 있고 보라색과 빨간색, 빨간색과 노란색 사이에도 선형결정 경계가 그려져 있다. 하지만 다른 선형 함수를 사용해서 공간을 세 개의 클래스로 나눌 수도 있다.



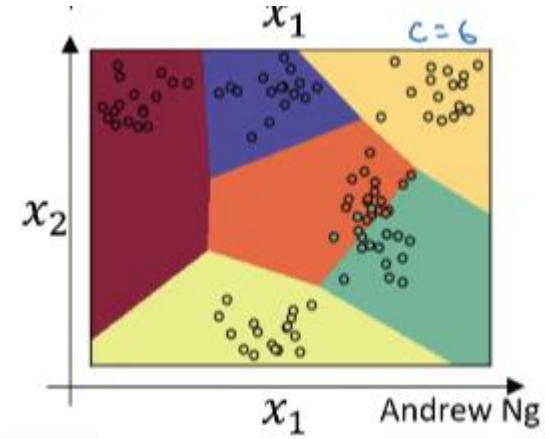
iv) C=4 인 경우, 여기에서 소프트맥스가 선형 경계를 그리고 있다.



v) C=5 인 경우



vi) C=6 인 경우



이렇게 은닉층이 없을 때 소프트맥스 분류 함수가 하는 일을 살펴보았다.
만약 **은닉 유닛이 여러개인, 더 깊은 신경망을 다룬다면 여러 클래스를 분류하기 위해 더 복잡하고 비선형의 경계도 볼 수 있다.**

Softmax 분류기 훈련시키기

Understanding softmax

이전 예시에서 위쪽 층에서 z^[L]을 다음과 같이 계산했다.

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \quad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$$

$$a^{[L]} = g^{[L]}\left(z^{[L]}\right) = \begin{bmatrix} e^5/\left(e^5+e^2+e^{-1}+e^3\right) \\ e^2/\left(e^5+e^2+e^{-1}+e^3\right) \\ e^{-1}/\left(e^5+e^2+e^{-1}+e^3\right) \\ e^3/\left(e^5+e^2+e^{-1}+e^3\right) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

여기서 C=4 개의 클래스를 다루니 z^[L]은 (4,1)벡터이다. 그리고 원소 단위로 e 르를 취해 임시 변수 t 를 얻었다. 활성화 함수인 g^[L]은 소프트맥스 활성화함수였다. 이 함수는 t 를 합이 1 이 되도록 정규화시켰고 이는 a^[L]과 같다. 여기서 z^[L]의 가장 큰 원소가 5 였고 가장 큰 확률도 첫번째 확률이다.

소프트맥스라는 이름은 하드맥스와 반대이다. 하드맥스는 z 벡터를 받아와서 이런 벡터로 대응시킨다. 하드맥스는 z 의 원소를 살펴보고 가장 큰 값이 있는 곳에 1 을 나머지에는 0 을 갖는 벡터로 대응시킨다. 가장 큰 원소만 1 을 가지므로 단호하다.

$$\text{“half max”} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

반면에 소프트맥스는 부드러운 느낌으로 z 를 이른 확률들로 대응시킨다. 왜 소프트맥스라 부를까 생각해보면 이런 직관을 얻을 수 있다. 하드맥스와는 반대되는 것이다. 꼭 알아야 하는 점은 **소프트맥스 회귀나 활성화 함수가 두 클래스만 다루는 로지스틱 회귀를 일반화**했다는 것이다.

만약 **소프트맥스에서 C=2 라면 결국 로지스틱 회귀와 같아진다.**

증명의 흐름을 보자면 C=2 에서 소프트맥스를 적용했을 때 출력층 a^[L]은 C=2 에서 출력값 두 개를 모아둔 것이다. 0.842 와 0.158 이라고 하면 이 두 숫자의 합은 항상 1 이 된다. 합이 항상 1 이므로 둘 다 계산하면 번거로우므로 하나만 계산해도 된다. 그러면 그 숫자를 계산하는 방식이 로지스틱 회귀가 하나의 출력값을 계산하는 방식과 같다. 알아둬야 할 것은 **소프트맥스 회귀가 클래스가 둘 이상인 경우 로지스틱 회귀를 일반화한 것**이라는 것이다.

$$\text{If } C = 2,$$
$$\text{softmax redusec to logistic regression}$$
$$a^{[L]} = \begin{bmatrix} 0.842 \\ 0.158 \end{bmatrix}$$

Loss function

이제 소프트맥스 출력층을 이용해 신경망을 학습하는 법을 살펴보자. 우선 신경망을 학습하기위해 사용했던 손실함수를 정의해보자.

예를 들면, 한 샘플이 목표로 하는 출력 값이 관측을 기반으로 0,1,0,0 이라고 하자. 지난 강의를 생각해보면 이 벡터는 클래스가 1 이므로 고양이를 의미한다. 그리고 신경망의 출력 값 y^은 다음과 같으며 이는 곧 a^[L]과 같다. (y^은 합이

1 인 확률로 구성된 벡터) 이 샘플에 대해서는 고양이일 확률이 20%에 불과하므로 신경망이 잘 작동하지는 않을 것이다.

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad a^{[L]} = \hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 04 \end{bmatrix}$$

여기에서 **신경망을 학습시키기 위한 손실함수**는 무엇일까? 소프트맥스 분류에서 주로 사용하는 손실함수는 j=1 부터 4 까지(일반적으로는 1 부터 C 까지) y_j*log(y^_j)합의 음수값이다.

위 단락의 식에서 y_1=y_3=y_4=0 이고 y_2 만 유일하게 1 의 값을 갖는다. 이 합을 구할 때 y_j 가 0 이면 고려해주지 않아도 되므로 유일하게 남는 항은 -y_2*log(y^-2)이다. 왜냐하면 이 항들을 모두 합할 때 j 가 2 인 경우를 제외하고는 모두 0 이기 때문이다. 여기에다 y_2 는 1 이니 결국 -log (y^_2)가 된다.

$$L(\hat{y}, y) = -\sum_{j-1}^{4} y_j \log y_j \qquad -y_2 \log \hat{y}_2 = -\log \hat{y}_2$$

이러한 학습 알고리즘이 **경사하강법을 이용해서 이 손실 함수의 값을 작게 만들려고 할 것**이다. **결국 -log (y^_2)의 값을 작게 만드는 것이다. 이는 또한 y^_2 의 값을 가능한 한 크게 만들어야 한다는 의미**이다. 확률이 1 보다 커질 수는 없다. 여기서 말이 되는 것이 입력 값 x 가 고양이의 사진이었으니 그에 대응하는 출력 값인 확률(0.2)을 최대한 키워야 한다.

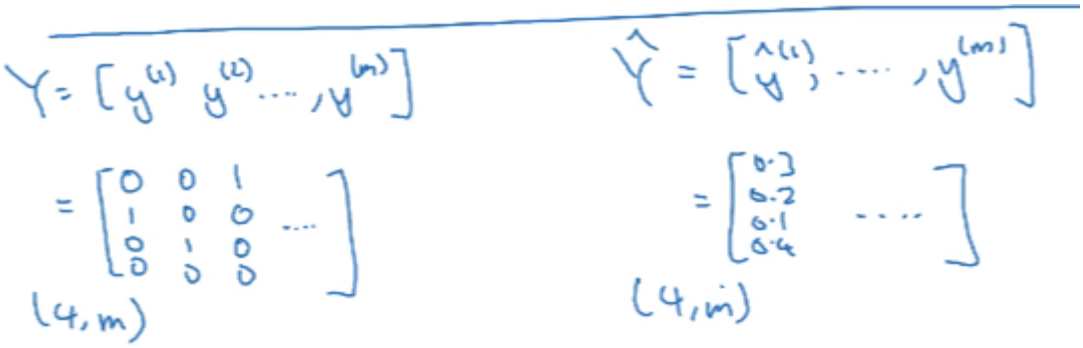
즉 일반적으로 손실함수는 훈련세트에서 관측에 따른 클래스가 무엇이든 간에 그 클래스에 대응하는 확률을 가능한 한 크게 만드는 것이다.

하나의 훈련 샘플에서 손실함수를 보았는데 **전체 훈련 세트에 대해 비용함수 J 는 무엇일까?** 편향 등의 매개변수를 설정할 때 비용함수는전체 훈련 세트에서 학습 알고리즘의 예측에 대한 손실함수를 합하는 것이다(훈련샘플들에 대해서). 그리고 이 비용함수를 최소로 하기 위해 경사하강법을 써야 한다.

$$J\left(\omega^0, b^{(0)}, \ldots\right) = \frac{1}{m}\sum_{i=1}^{n} l\left(\hat{y}^{(i)}, y^{(i)}\right)$$

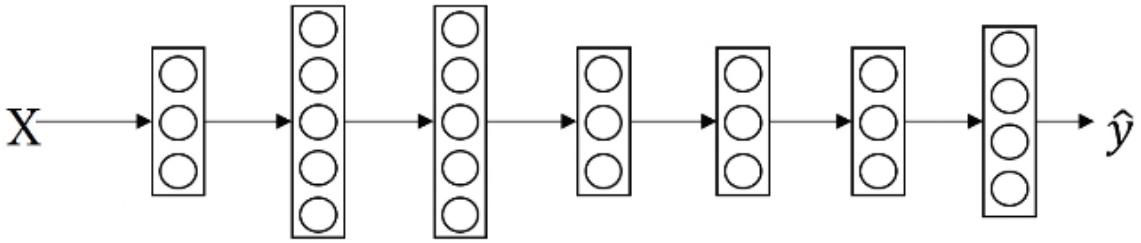
마지막으로 구현에 관해 살펴보자면 C=4 이고 (4,1) 벡터인 상황에서 y^도 (4,1) 벡터이다. Y 라는 행렬은 y^[1], y^[2]부터 y^[m]이 된다. 위에서 했던 샘플이 첫번째 훈련샘플이라고 한다면 첫번째 열은 0 1 0 0 이 된다. 이런식으로 반복하면 Y 는 결국 (4,m) 차원의 행렬이 된다.

비슷하게 y^은 y^(1)부터 y^(m)을 수평하게 쌓은 것이다. 즉 이는 첫번째 훈련샘플에 대한 출력 값이므로 y^(1)이 된다. 그러면 Y 는 0.3 0.2 0.1 0.4 를 갖게 된다. 그러면 Y 도 (4,m)차원이 된다.



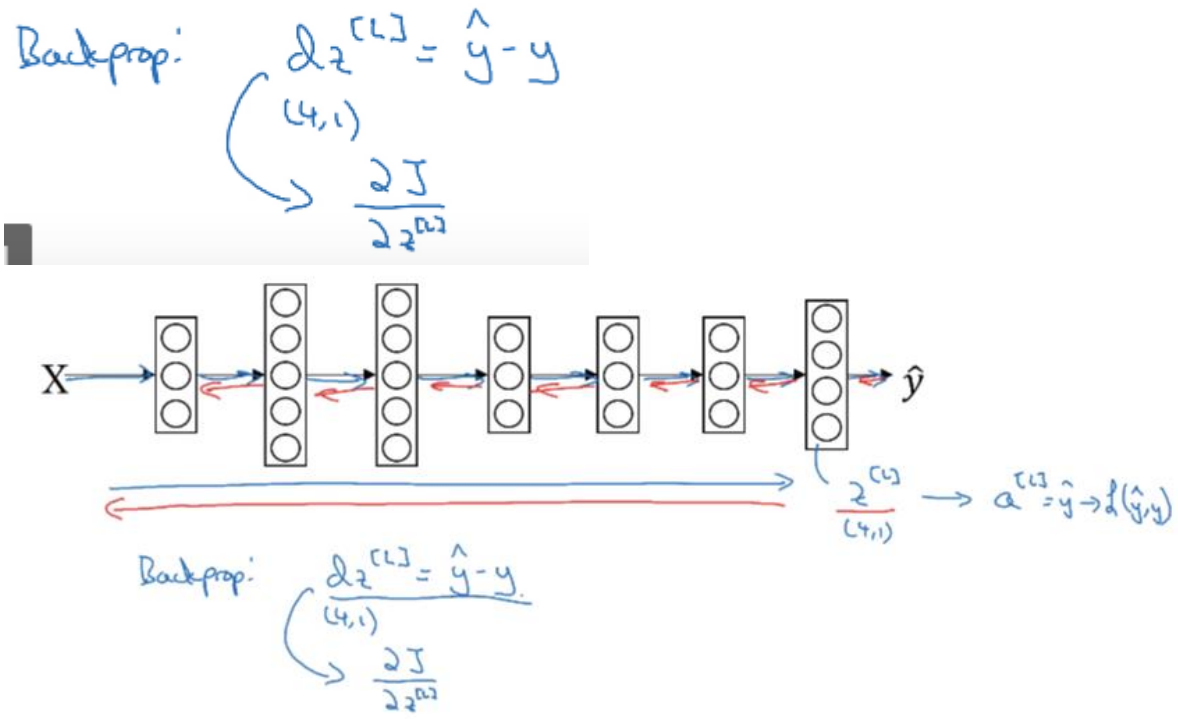
Gradient descent with softmax

끝으로 **소프트맥스 출력층이 있는 경우 경사하강법**을 어떻게 구현할지 보자.



마지막 출력층이 z^[L]을 계산한다. (C,1) 차원이 되고(우리 예시에서는 (4,1)차원임) 여기에 소프트맥스 활성화 함수를 취해서 a^[L] 또는 y^을 얻는다. 그 값을 이용해서 손실함수를 계산할 수 있다. 이전에 신경망의 정방향 전파를 배웠었다. **정방향전파는 손실함수를 구하는데 사용**했었다.

역방향전파나 경사하강법은 어떠했는지 생각해보면 **역방향 전파에서** 초기화를 위한 핵심단계 핵심이 되는 식은 마지막 층에서 z^[L]의 미분이 (4,1) 벡터인 y 를 뺀 것과 같다. 클래스가 4 개 일 때 모두 (4,1) 벡터가 되는 것이다(일반적으로는 (C,1) 벡터임). 일반적인 정의를 빌리자면 여기서 dz^[L]은 비용함수 z^[L]에 대해 편미분한 것이다. 미적분으로 유도 가능하지만 이 **공식을 이용**해도 된다. 이런 식으로 시작해서 신경망 전체에 대한 미분을 역방향전파로 구하는 것이다.



지역 최적값 문제

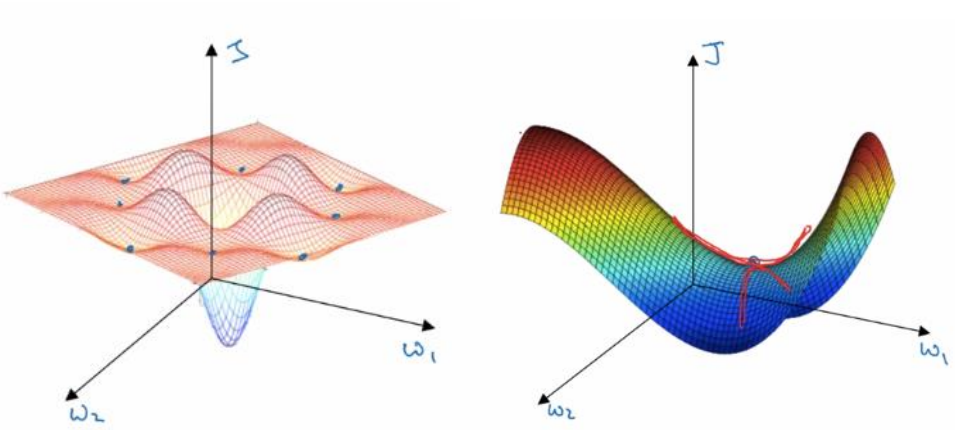
Local optima in neural networks

사람들이 지역 최적값에 대해 고민할 때 아래 그림을 생각한다. 만약 우리가 w_1 과 w_2 라는 매개변수를 최적화한다고 할 때 이 면적의 높이가 비용함수다.

아래 그림에서는 지역 최적값이 많아보인다. 경사하강법 등의 알고리즘이 전역 최적값에 도달하기 전에 지역 최적값에 갇혀버리기 쉽다.

이렇게 2 차원에서 그림을 그린다면 서로 다른 지역 최적값이 많은 그림을 쉽게 접할 수 있다. 이렇게 낮은 차원의 그림을 통해서 직관을 얻곤 한다. 하지만 직관이 항상 옳지만은 않다. **경사가 0 인 점은 대부분 지역 최적값이 아니라 비용함수가 0 인 경우 대개 안장점이다. 경사가 0 일 점 중에 하나인 것**이다. **사실 고차원 함수에서 경사가 0 이면 각 방향에서 볼록함수나 오목함수가 되기 마련**이다.

예를 들어 20,000 차원의 공간에서 지역 최적값이 되기 위해서는 20,000 개의 방향이 모두 U(아래로 볼록 함수)와 같은 모양으로 생겨야 한다. 그러나 그런 일이 일어날 확률을 매우 낮다. (확률 적으로 2^(-20,000)정도) **대신 어떤 방향에서는 위로 볼록한 형태가, 어떤 방향에서는 아래로 볼록한 형태가 주로 발생**할 것이다. 따라서 **고차원 공간에서는 오른쪽 그래프처럼 지역 최적값보다 안장점이 되기 쉽다**. 여기서 왜 안장점이라고 불리냐하면 이 모양이 말에 얹는 안장과 비슷하기 때문이다.

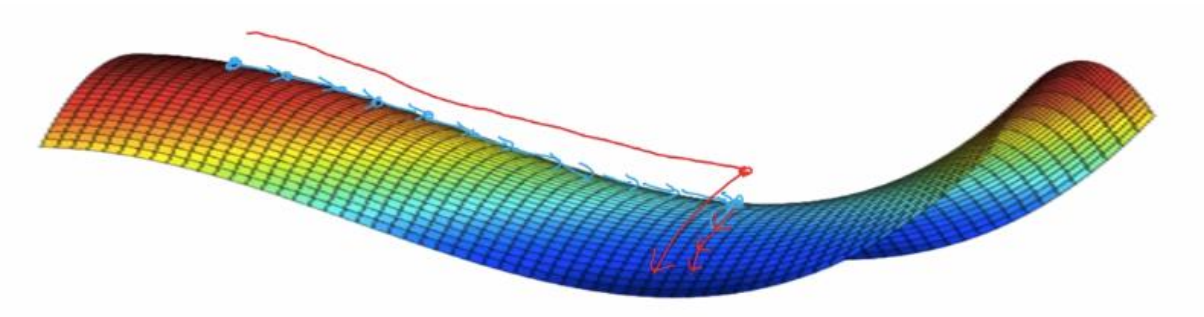


딥러닝의 역사에서 배울 수 있는 것은 왼쪽 그림처럼 **낮은 차원의 공간에서 얻었던 직관이 학습 알고리즘이 높은 차원에서 돌아갈 때 적용되지 않을 수도 있다는 것**이다. 왜냐하면 20,000 개의 매개변수가 있을 때 J 는 20,000 차원의 벡터에 대한 함수이고 지역 최적값보다 안장점을 훨씬 많이 볼 수 있을 것이다.

Problem of plateaus

그럼 지역 최적값이 문제는 아니고 무엇이 진짜 문제일까? **안정지대가 학습을 아주 지연**시킬 수 있다. **안정 지대는 미분값이 아주 오랫동안 0 에 가깝게 유지되는 지역**이다. 만약 그림의 왼쪽 표시에 위치한다고 하면 경사하강법에 따라 면을 따라서 아래로 움직인다. **여기서 경사가 거의 0 에 가까울테니 면이 거의 평평할 것**이다. 여기서 아주 오랜 시간이 걸리게된다.

왼쪽이나 오른쪽에 무작위로 작은 변화가 주어지면 빨간색 화살표처럼 알고리즘이 안정지대를 벗어날 수 있다. 하지만 하늘색의 마지막 지점에 도달하기 전에 아주 긴 시간동안 경사를 탄 후에야 안정지대를 벗어날 수 있다.



알아두어야 할 것은

- 충분히 큰 신경망을 학습시킨다면 지역최적값에 갇힐 일이 잘 없다. 여러 매개변수와 비용함수 J 가 상대적으로 고차원에서 정의된다면 말이다.

- 안정지대는 학습속도가 매우 느려지기 때문에 문제이다. 여기서는 모멘텀이나 RMSprop, Adam 등 이런 알고리즘의 도움을 받을 수 있다. 이런 경우에는 Adam 과 같은 최적화 알고리즘이 안정지대 내에서 움직이거나 벗어나는 속도를 올릴 수 있다.

<<정리>>

- 고차원 비용함수에서 경사가 0 인경우는 대부분 지역 최적값이 아니라 대개 안장점이다.
- 안장점으로 향하는 구간인 안정지대는 미분값이 아주 오랫동안 0 에 가깝게 유지되는 지역을 말한다.
- 대개 충분히 큰 Network 학습시 지역 최적값에 갇이는 일은 거의 없다.
- 안정지대의 문제점은 경사가 거의 0 에 가깝기 때문에 학습속도가 느려진다. 또 다른 쪽으로 방향변환이 없다면 안정지대에서 벗어나기 어려우며 이는 Adam, RMSprop 등 알고리즘이 해결해준다.

신경망이 일반적으로 고차원에서 최적화 문제를 해결할 때 사실 어떤 사람도 이 공간이 어떻게 생겼는지 잘 모를 것이다. 이 공간에 대한 이해도 계속 발전하고 있다.

Tensorflow

Motivating problem

최소화하고 싶은 비용함수 J 가 있다고 하자. J(w) = w^2-10w+25 라는 비용함수가 주어졌다. J(w)=(w-5)^2 와 같다. 따라서 이 식을 최소로 하는 w=5 이다. 제곱식을 모른다고 하고 처음 식만 안다고 할 경우 tensorflow 에서 이 식을 어떻게 최소화하는지 살펴보자.

프로그램이 모든 매개변수와 관련이 있어서 다소 복잡한 비용함수 J(w,b)를 학습시키는 신경망과 구조가 비슷하므로 비슷한 방법으로 텐서플로우를 이용해서 자동으로 이 비용함수를 최소화하는 w 와 b 의 값을 찾을 수 있다.