**1. Neural Networks**

Wiki: "Neural networks are computing systems vaguely inspired by the biological neural networks that constitute animal brains."

\* constitute 구성하다, 이루다

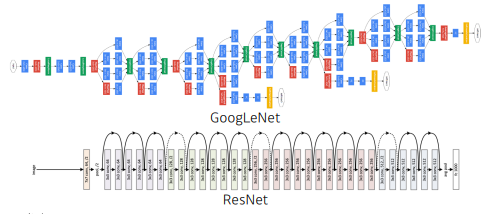


비행기의 발전 과정을 보면 처음에는 날 수 있는 새, 박쥐들을 모방했다. 하지만 이후 날 수 있는 발명품들을 보면 이들을 모방하지 않고 공기 역학 등을 이용했다. 즉, 새처럼 생기진 않았다.

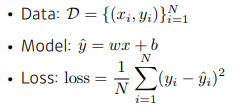
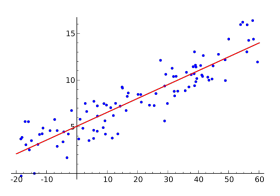
Neural Network도 마찬가지다. 처음 시작은 인간의 뇌를 본뜬다고 생각했지만, 꼭 인간의 뇌를 본뜰 필요는 없다. 사실 연구 트렌드가 인간의 뇌를 모방해서 잘 된다고 말하기에는 너무 달라졌다.

교수님은 아래와 같은 표현을 더 좋아하신다고 한다.

Neural networks are function approximators that stack affine transformations followed by nonlinear transformations



**2. Linear Neural Networks**



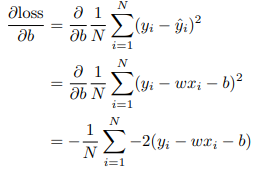
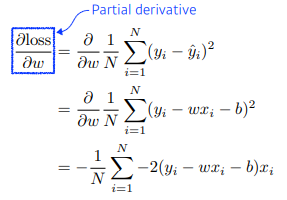
가장 간단한 neural network는 linear function이다. linear regression의 목적은 입력이 1차원이고 출력이 1차원일 때 두 개를 연결하는 모델을 찾는 것이다. 즉, 이 모델은 기울기, 절편 2개의 파라미터를 찾아야 한다.

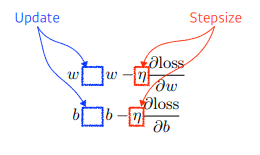
Data는 1차원이여야 하고 N개의 데이터로 이루어진다. 모델은 x를 w라는 스칼라로 곱하고 b라는 bias를 더함으로써 만들어진다. loss는 제곱근 평균을 사용한다. 회귀 문제의 경우 제곱근 평균을 많이 사용한다.

우리가 문제를 선형문제로 만들고, square loss function을 사용하면 w와 b의 최적값을 바로 찾을 수 있다. 하지만 이는 데이터가 어느 정도 작고, loss가 square loss function이여야 한다는 많은 제약조건이 걸린다. 그래서 우리가 사용할 전략이 backpropagation이다.

결국은 loss function이 주어졌을 때 loss function을 줄이는게 목표이다. 따라서 파라미터를 어느 방향으로 움직였을 때 loss function이 줄어드는지를 찾고 그 방향으로 파라미터를 바꾸면 된다. 즉, loss가 줄어드는 것이 목표이므로 loss function을 각각의 파라미터로 미분하는 방향을 역수 방향으로 파라미터를 업데이트하게 되면 loss가 최소화되는 지점에 이르게 된다.

아래 나와있는 수식은 N개의 데이터를 모두 사용했을 때 N개의 데이터에 대한 타깃값과 나의 모델의 출력 사이의 제곱을 minimize하는 loss function에 대한 w의 편미분이다. 즉, loss function을 w로 편미분한 값을 찾아서 w에 편미분한 값을 적절한 값을 곱해서 빼주게 된다. bias도 마찬가지다.



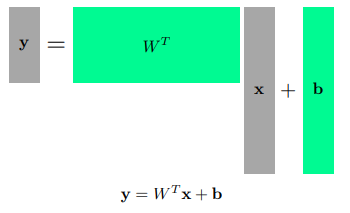


이런 식으로 w, b를 계속 update하는 것을 gradient descent라고 부른다. 왜냐하면 줄이고자 하는 loss function에 대해 w, b라는 파라미터의 편미분을 구하고 이 편미분을 빼주면서 최소값을 찾기 때문이다.

하지만 모델이 이렇게 단순히 선형일 수는 없다. 왜냐하면 neural network의 정의가 단순히 linear한 변화만 있는 것이 아니라 unlinear한 변환도 포함되기 때문이다. 그리고 deep learning이기 때문에 단순히 1층만 있는 것이 아니라 여러 layer가 존재할 것이다. 그래서 마지막에 나온 loss function의 값을 전체 파라미터로 다 미분을 해야 하는 것이 backpropagation이고, backpropagation의 파라미터를 update 시키는 것이 gradient descent이다.

이때 step size가 굉장히 중요하다. step size가 너무 크게 되면 학습이 되지 않는다. 왜냐하면 gradient descent는 굉장히 local한 정보이기 때문이다. 그렇다고 0이 되어도 학습은 안된다. 따라서 적적한 step size를 지정하는 것이 중요하다.

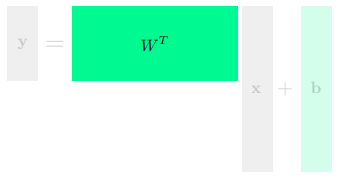
Of course, we can handle multi dimensional input and output



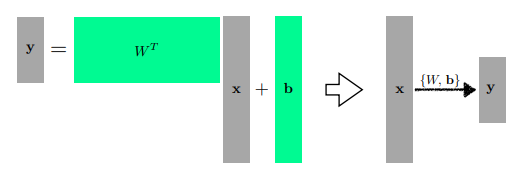
앞에서 말한 것처럼 세상은 선형으로만 이루어져 있지 않다. 혹은 단순히 1차원 입력에서 1차원 출력을 받고 싶지 않을 수도 있다. 예를 들어, 100차원 입력에서 20차원의 출력을 받고 싶을 수 있다. 그럴 때는 행렬을 사용하면 된다. n차원에서 m차원으로의 변환을 affine transform이라고 한다.

앞에서는 단순히 y = Wx + b였다면 위의 수식은 W, b가 행렬과 벡터고 y는 W\_T\*x + b로 표현된다.

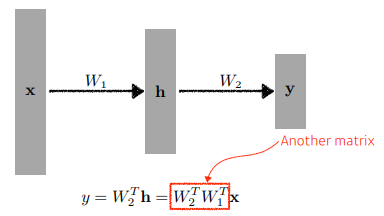
One way of interpreting a matrix is to regard it as a mapping between two vector spaces



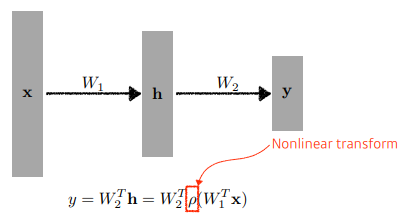
이런 행렬의 곱은 두 개의 vector space상의 변환이라고 해석하는 것이 좋다. 선형대수 수업을 듣게 되면 선형성을 가지는 어떠한 변환이 있을 때 그 변환은 항상 행렬로 표현될 수 있는 것을 배운다. 마찬가지로 행렬을 찾겠다는 것은 두 개의 vector space 혹은 서로 다른 차원 사이에 선형변환을 찾겠다는 것이다. 즉, 아래와 같이 표현될 수 있다.



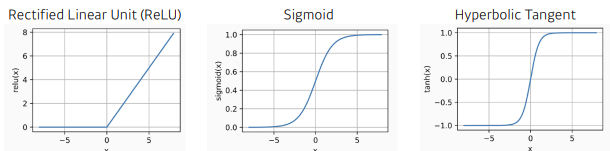
W(eight)와 b(ias)로 x라는 입력을 y라는 출력으로 보낸다는 의미다.



딥러닝이라는 것은 결국 뉴럴 네트워크를 shallow 하지 않게 여러 개 쌓겠다는 의미이다. 그러면 네트워크를 여러 개 쌓으려면 어떻게 해야 할까? 앞에 네트워크가 W1\_T\*x(bias는 생략)라고 하면 이렇게 나오는 벡터를 hidden vector, hidden layer라고 하고, 해당 벡터를 W2\_T에 다시 곱할 수도 있다. 하지만 위 과정을 결국 행렬 곱의 과정과 동일하며 1 layer 구조와 다를 것이 없다.

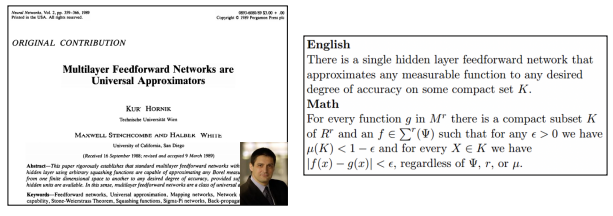


그래서 우리가 필요한 것이 Nonlinear transform이다. 결국, x라는 입력에서 y라는 출력으로 가는 이 mapping이 표현할 수 있는 표현력을 최대한 극대화하기 위해서는 단순히 선형결합을 n번 반복하는 것이 아니라 한 번 선형 결합을 하면 activation function을 이용해 nonlinear transform을 거치고 그렇게 얻어지는 feature 벡터를 다시 선형변환하고, nonlinear transform을 거치는 과정을 반복한다.



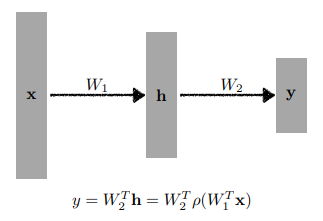
위 함수들이 activation function들이다. 어떤 함수가 좋은지는 문제마다 다르다.

Caution: It only guarantees the existence of such networks

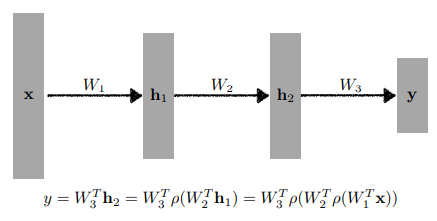


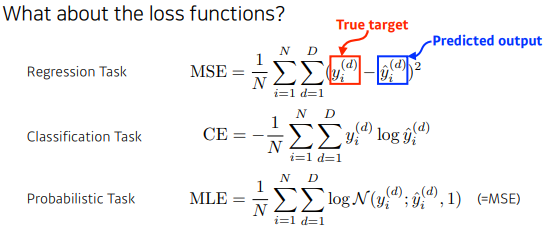
**3. MLP(Multi-Layer Perceptron)**

This class of architectures are often called multi-layer perceptrons



Of course, it can go deeper





앞에서 우리가 간단히 회귀를 살펴볼 때의 목적은 MSE를 줄이는 것이 목적이다. 이때 MSE의 값을 줄이는 것이 항상 목적을 달성하게 하지는 않는다. 따라서 loss function이 어떤 성질을 가지고 있고 왜 우리가 원하는 결과를 얻어낼 수 있는지에 대한 생각을 꼭 해야 한다.