※ *STEP 0 : 인공 신경망 간단 요약*

인공신경망의 구성요소

- **노드/유닛(Node/Unit)** : 각 층(Layer)를 구성하는 요소
- **층(Layer)**
    - 1. 입력층(Input Layer) : 데이터를 받아들이는 층
    - 2. 은닉층(Hidden Layer) : 데이터를 한 번 이상 처리한 (가중치를 곱하고, 활성함수를 얻은) 노드로 구성된 층
    - 3. 출력층(Output Layer) : 최종 은닉층 또는 입력층에 가중치를 곱하고, 출력함수의 결과를 얻은 노드로 구성된 층
- **가중치(Weight)** : 노드와 노드간의 **연결강도**를 나타냄
- **합(Summation)** : 가중치와 노드의 곱을 합하는 것
- **활성함수(Activation Function)** : 합을 처리해 주는 함수로 은닉층 노드의 결과를 얻을 경우 활성함수로 표현
- **출력함수(Output Function)** : 합을 처리해 주는 함수로 출력층 노드의 결과를 얻을 경우 출력함수로 표현
    - 출력함수는 마지막 은닉층의 노드와 가중치의 곱을 처리해주는 함수로 출력층의 노드값을 계산해 줍니다.
    - 반응 변수 Y의 변수 타입이 범주형(Categorical), 연속형(Continuous)에 따라 크게 Softmax 함수와 Identity 함수로 구성되며, 각각 분류문제, 회귀문제로 연결됩니다.
- **손실함수(Loss Function) = 오차함수** : 가중치의 학습을 위해 출력함수의 결과와 반응(Y)값 간의 오차를 측정하는 함수
    - 분류 : Cross Entropy Loss
    - 회귀 : Mean Square Error Loss(MSE)
- 퍼셉트론과 인공신경망의 차이점
    - 퍼셉트론은 인공신경망에서 활성함수를 step 함수로 적용하되 0 대신 thereshold θ를 이용한 특수한 경우로 생각할 수 있습니다.
    - 다시 말해, **퍼셉트론에서 활성함수의 개념이 추가된 것이 인공신경망**이라고 할 수 있습니다.
- 변수 전처리 : 인공신경망 모형을 적용할 때 변수의 형태에 따라서 간단한 전처리를 해줍니다.
    - **연속형**의 경우 : 가중치 학습시 안정성을 위해 **최소-최대 정규화(Min-max Normalization)** 를 해줍니다.
        - Min-max 정규화를 거치면 처리된 값은 [0,1] 사이의 값을 갖게 됩니다. 여러 연속형 변수가 데이터에 존재할 경우 각 변수의 단위가 다를 수있습니다.
        - 경사하강법으로 가중치를 학습할 때, 단위가 다를 경우 알고리즘을 불안정하게 만드는 경향이 있어 전처리로 정규화를 해줍니다.
    - **범주형**의 경우 : dummy 변수 처리를 해주는데, 인공신경망에서는 One-Hot Encoding 방법을 이용해 범주형 변수의 값을 이진벡터로 맵핑해줍니다.
- 경사하강법(Gradient Descent)
    - 인공신경망의 가중치를 학습하기 위해서 경사하강법을 사용하는데, 경사하강법은 국소 최소값(Local minimum)을 찾기 위한 수치적인 방법으로 기울기의 역방향으로 값을 조금씩 움직여 함수의 출력 값이 수렴할 때까지 반복하는 것입니다.
    - 그 중 학습률(learning rate)은 기울기의 역방향으로 x를 이동할 크기를 조정하는 역할을 합니다.
    - 주의해야할 두 사항은 초기값과 학습률입니다.
- 역전파

    - 역전파 알고리즘은 오차함수로부터 측정된 오차를 출력층에서부터 입력층까지 역전파하여 연쇄적으로 가중치를 학습하는 방법입니다.
    - 역전파 진행 순서
        - 1.주어진 가중치를 이용해 **입력값을 (앞)전파** 시킨다.
        - 2.출력층에 도달했을때 손실함수를 이용하여 **오차를 측정**한다.
        - 3.오차에 대한 정보를 **부모층으로 역전파**한다.
        - 4.모든 가중치의 **gradients를 계산**한다.
        - 5.**경사하강법**을 이용해 가중치를 **update**한다.
        - 6.**수렴할 때까지 1~5를 반복**한다.
- 참고 사이트 : https://yjjo.tistory.com/5

*※ STEP 4 : 역전파 * [책보기]

- Q. **역전파법(backpropagation)** 이 뭔가요?

    - 역전파는 앞먹임 신경망 학습에서 가중치와 바이어스에 대한 오차함수의 미분을 계산해야하는데 이러한 미분을 효율적으로 계산하는 방법이에요.
    - Q-1. 역전파법을 왜 사용하죠?
        - **경사 하강법**을 실행하기 위해서는 오차함수 E(w)의 기울기를 계산해야 하는데, 이 미분의 계산이 매우 까다롭기 때문에 역전파법을 사용하는거죠.
        - 각 층의 결합 가중치(w)와 각 유닛의 바이어스(b)에 대한 오차함수의 편미분이 기울기 벡터의 각 성분이고, 자세히는 중간층, 특히 입력이 가까운 깊은 층의 파라미터일수록 미분을 계산하기 까다로워요.

- Q. 오차 역전파를 통해 **오차 기울기(가중치에 대한 오차의 미분)를 계산하는 절차** 를 말해줘요.

    - 1. 각각의 층의 유닛 입력 u과 출력 z을 순서대로 계산한다.
    - 2. 출력층 델타(δ)를 구한다. (통상적으로 δ = z - d : 출력층 L의 유닛 j의 델타 δ는 신경망의 출력(z)과 목표 출력(d)의 차가 된다.)
    - 3. **역전파** : 각 중간층 l(= L-1, L-2, .., 2)에서의 델타 δ를 출력층부터 가까운 순서대로 계산한다.
    - 4. 각 층 l(= 2, ..., L))의 파라미터 w에 대해 미분을 계산한다.
    - **참조** : l-1번째 층의 유닛 i와 l번째 층의 유닛 j를 잇는 결합의 가중치 w_ji에 대한 미분은, 유닛 j에 대한 **델타(δ_j)(L)** 와 유닛 i의 **출력 z_i(L-1)** 의 **곱**에 지나지 않는다.

- Q. 순전파와 역전파 계산의 **공통점과 차이점**은?

    - **공통점** : 순전파와 역전파 계산은 모두 층 단위의 행렬 계산으로 나타낼 수 있으며 식의 형태가 닮았다는 공통점이 있다.
    - **차이점** : 순전파는 **비선형 계산**인데 비해, 역전파는 **선형 계산**이라는 차이점이 있다.
        - 순전파 계산에서는 각 층에 대한 입력은 유닛이 갖는 활성화 함수를 경유하기 때문에, 활성화 함수가 비선형이라면 이 층의 입출력의 관계도 비선형성을 갖는다.
            - ex) 로지스틱 함수를 예로 들면 각 층의 출력은 항상 [0, 1]의 범위로 제약되며, 값이 지나치게 커져서 발산해 버리는 일은 일어나지 않는다.
        - 한편, 역전파 계산은 선형 계산이다. 그 결과, 각 층의 가중치의 값이 크면 델타가 각 층을 거쳐 전달되는 도중에 **급속하게 커지거나(발산)**, 혹은 반대로 기울기가 작으면 **급속하게 작아져 0(소실)** 이 되어 버린다. 어떤 경우든 가중치의 업데잍트가 잘안되며 학습 자체가 어려워진다.

- 내 맘대로 퀴즈 1. 그걸 왜 하죠?
    - DNN에서 Pre-training을 왜 하죠?
        - 다층 feed-forward 신경망은 gradient vanishing 문제로 인해 일반적으로는 학습이 잘 되지 않기 때문에 여러 방법 중 하나로 pre-training을 합니다.
    - 오토인코더는 왜 하죠?
        - 오토인코더는 다양한 pre-training 중에 기본적인 방법 중 하나인데요. 가중치의 좋은 초기값을 얻는 목적으로 이용하고 목표 출력없이 입력만으로 구성된 트레이닝 데이터로 비지도 학습을 수행해 데이터의 특징을 나타내기 위해 오토인코더를 합니다.
        - 오토인코더는 머신러닝에서 PCA와 비슷한 역할을 한다고 생각하시면 이해하기 쉬울 것입니다. 이제 오토인코더가 자주 쓰이는 이유에 대해 알려드리도록 하겠습니다.
            - 1. 데이터 압축 : 데이터 압축이란 메모리 측면에서의 장점입니다. AutoEncoder를 이용하여 이미지나 음성 파일을 중요 Feature만 가지고 압축할 경우, 용량도 작고 품질도 더 좋다고 합니다.
            - 2. Curse of dimensionality : 일명 차원의 저주라고 불리는 문제를 예방할 수 있습니다. 예를 들어서, 1,2, ..., 10의 값을 가질수 있는는 X1에 대하여 8개의 관측치가 있다고 생각해봅시다. 이 경우 데이터의 밀도는 0.8로 어느정도 높은 값을 가집니다. 하지만 X1, X2, X3로 3차원으로 넓히게 되면 데이터가 가질 수 있는 분포가 10X10X10=1000개가 되서 밀도는 0.008로 매우 작아지게 됩니다. 이처럼 데이터의 차원이 증가할수록 모델추정에 필요한 샘플 데이터의 개수가 기하급수적으로 증가하게 되는 것을 차원의 저주라고 합니다. AutoEncoder는 Feature의 수를 줄여줌으로써, 데이터의 차원이 감소하여 차원의 저주를 피하게 할 수 있게됩니다.
            - 3. Discovering most important features : AutoEncoder는 unsupervised learning으로 자동으로 중요한 Feature를 찾아줍니다. 예를 들어서, 숫자 인식하는 문제에서 숫자의 두께나, 회전각도, 크기와 같은 중요한 Feature를 자동으로 찾아주게 됩니다. 특히, A1과 A2중에 B와 가까운 것이 무엇이냐는 질문에, Raw data에서 본다면 A1이 가깝지만, 실제로 중요한 Feature에 대해서는 A2가 가까운 것으로 나타납니다. 이처럼 고차원 공간에서의 거리는 아무런 의미가 없을 수 있습니다. 따라서 중요한 Feature인 Manifold를 잘 찾아내는 것이 중요합니다. AutoEncoder가 이 때문에 각광을 받고 있는것입니다.
    - sparse(희소) 오토인코더는 왜 하죠?
        - 바로 overfitting 문제를 줄이는 효과가 있기 때문입니다. 실제로 AutoEncoder의 문제점중 하나로 Feature를 압축하다보면, 다른 데이터가 들어와도 training set과 비슷하게 만들어버리는 overfitting 문제가 있습니다. 이를 방지하기 위해 Sparse AutoEncoder를 통해 sparse(0이 많은)한 노드들을 만들고, 그 중에서 0과 가까운 값들은 전부 0으로 보내버리고 0이 아닌 값들만 사용하여 네트워크를 학습시킵니다. 이로써 기존의 AutoEncoder의 overfitting 문제도 줄이고, 차원 축소 효과도 얻을 수 있다고합니다.
    - stacked(적층) 오토인코더는 뭐고 왜 하죠?
        - Stacked AutoEncoder는 말그대로 AutoEncoder를 층으로 쌓아서 사용하는 방법입니다. Stacked AutoEncoder는 주로 네트워크를 Pretraining할 때, initial weight를 초기화하는 방법으로 사용됩니다.
        - stacked 오토인코더를 사용하는 이유는 feed-forward 신경망의 gradient vanishing 문제를 해결할 수 있고 좋은 초기값을 줘서 학습을 좀 더 잘할 수 있게 만들기 위함입니다.
    - denosing 오토인코더는 뭐고 왜 하죠?

    - Denoising AutoEncoder는 입력층에서 Hidden layer로 갈 때, Noise를 추가한 것입니다. 이 때 Noise는 기존의 입력 데이터를 변화 시키는데, 사람의 인식하에서 같은 데이터라고 느낄 정도의 수준만 허용합니다.
    - 따라서 데이터는 실제로 달라지지만, 인식하는데에는 지장을 주지 않게 됩니다. 이러한 노이즈를 추가하였을 때 실제로 성능에 있어서 더 좋아진다고 합니다.

- 내 맘대로 딥러닝 퀴즈 2
    - 1. GD와 SGD, MGD(Mini-batch GD)의 차이점이 무엇인가?
        - GD : 전체 데이터를 가지고 반복적으로 루프(loop)를 돌려서 그라디언트(gradient)를 계산하고 파라미터(parameter/weight)를 업데이트하는 것
        - MGD : 학습데이터(training data)의 배치(batches)만 이용해서 그라디언트(gradient)를 구하는 것이다.
        - SGD : MGD 방법의 극단적인 형태는 미니배치(mini-batch)가 데이터 달랑 한개로 이루어졌을 때이다. 이게 SGD이다. 1 epoch동안 학습데이터 개수만큼의 업데이트가 수행됩니다.
        -
    - 2. epoch와 iteration, batch의 차이점은?
        - 1 epoch : 즉, 전체 데이터 셋에 대해 한 번 학습을 완료한 상태
        - 1 iteration : 1회 학습 = epoch를 나누어서 실행하는 횟수
        - batch : 데이서 셋을 batch size 크기로 쪼개서 학습
        -

```
ex1) 총 데이터가 100개, batch size가 10개이면,
    1 iteration = 10개의 데이터에 대해서 학습
    1 epoch = 100/batch size = 10 iteration
ex2) 전체 2000개 데이터 셋, epoch = 20, batch size = 500이면,
    1 epoch는 각 데이터의 size가 500인 batch가 들어간 네 번의 iteration으로 나누어진다.
    전체 데이터셋에 대해서 20번의 학습이 이루어졌으며, iteration 기준으로 보면 총 80번의 학습이 이루어진 것이다.
```

    - 3. 왜 딥러닝이 ann보다 좋은가? (활성화 함수 측면에서 설명하라)
        - 비선형 활성화함수를 사용하면서 히든레이어가 많아질 수록 굴곡과 곡선이 더 많아지면서 더욱 복잡하게 만들 수 있습니다. 그래서 더 복잡한 문제를 잘 해결할 수 있습니다.
    - 4. cnn 20(채널)*36*36을 3X3의 필터 10개를 적용했을 때(스트라이드 1, 패딩 0) 결과의 dimension은 무엇이고 파라미터 수는 몇개인가?
        - dimension : 36-3+1=34 -> 34X34 : (36 + 2*0 - 3)/1 +1 = 36-3+1 = 34
        - 필요 파라미터 개수 : 20*3*3*10 = 1800 : 입력채널수 X 필터폭 X 필터높이 X 출력 채널 수
        - 출력 데이터 크기
            - OutputHeight(OH) = (H(입력 데이터 높이) + 2P(패딩 사이즈) - FH(필터 높이))/S(strid 크기) + 1
            - OutputWeight(OW) = (w(입력 데이터 폭) + 2P(패딩 사이즈) - FW(필터 폭))/S(strid 크기) + 1

- CNN과 FC(Fully Connected) NN의 파라미터 개수 비교
    - FC(Fully Connected) NN은 입력노드x출력노드 : ex) (600,300) -> 파라미터 수 = 180,000
- CNN은 Fully Connected Neural Network과 비교하여 다음과 같은 특징을 갖는다.
    - CNN은 학습 파라미터 수가 매우 작음
    - 학습 파라미터가 작고, 학습이 쉽고 네트워크 처리 속도가 빠름
- CNN(Convolutional Neural Network)은 이미지의 공간 정보를 유지하면서 인접 이미지와의 특징을 효과적으로 인식하고 강조하는 방식으로 이미지의 특징을 추출하는 부분과 이미지를 분류하는 부분으로 구성됩니다. 특징 추출 영역은 Filter를 사용하여 공유 파라미터 수를 최소화하면서 이미지의 특징을 찾는 Convolution 레이어와 특징을 강화하고 모으는 Pooling 레이어로 구성됩니다.
- CNN은 Filter의 크기, Stride, Padding과 Pooling 크기로 출력 데이터 크기를 조절하고, **필터의 개수로 출력 데이터의 채널을 결정합니다.**
- CNN는 같은 레이어 크기의 Fully Connected Neural Network와 비교해 볼 때, 학습 파라미터양은 20% 규모입니다. 은닉층이 깊어질 수록 학습 파라미터의 차이는 더 벌어집니다. CNN은 Fully Connected Neural Network와 비교하여 더 작은 학습 파라미터로 더 높은 인식률을 제공합니다.
- 참고 : http://taewan.kim/post/cnn/
- 5. cost function(loss function)과 activation function
    - cost function은 가중치의 학습을 위해 출력함수의 결과와 반응(Y)값 간의 오차를 측정하는 함수 : MSE, 크로스 엔트로피
    - 활성화 함수는 **Data를 비선형으로 바꾸기 위해서**사용함 : RELU, SIGMOID, tanh(x), softmax
        - tanh - sigmoid 함수를 재활용하기 위한 함수. sigmoid의 범위를 -1에서 1로 넓혔다.
        - ReLU - max(0, x)처럼 음수에 대해서만 0으로 처리하는 함수
        - Leaky ReLU - ReLU 함수의 변형으로 음수에 대해 1/10로 값을 줄여서 사용하는 함수
        - ELU - ReLU를 0이 아닌 다른 값을 기준으로 사용하는 함수
        - maxout - 두 개의 W와 b 중에서 큰 값이 나온 것을 사용하는 함수
    - 번외
        - tanh는 시그모이드보다 학습 수렴 속도가 빠르지만, 1차 미분했을 때, -5보다 크고 5보다 작으면 gradient가 0으로 작아지는 단점이 있다.
        - 시그모이드는 학습이 느리고 입력값이 너무 크거나 작으면 gradient 값이 지나치게 작아지는 단점이 있다.
- 6. 시그모이드보다 RELU를 사용하는 이유?
    - Sparsity와 Vanishing Gradient의 장점 때문에 사용한다.
    - Sparsity (a가 0보다 작을 때 / a≤0)
        - 활성화 값(a)이 0보다 작은 뉴런들이 많을수록 더욱 더 sparse한 모습을 띄게 된다. 하지만 기본적으로 (혹은 전통적으로) neural network에서 사용해온

sigmoid 함수는 항상 0이 아닌 어떠한 값(eg. 0.2, 0.01, 0.003 ...)을 만들어내는 경향이 있어 dense한 모습을 나타내게 된다.
    - 뉴런의 활성화값이 0인 경우, 어차피 다음 레이어로 연결되는 weight를 곱하더라도 결과값은 0을 나타내게 되서 계산할 필요가 없기에 sparse한 형태가 dense한 형태보다 더 연산량을 월등히 줄여준다. (하지만 0으로 한번 할당되면 다시 활성화 되지 않으므로 해당 뉴런을 dead neuron / dying Relu 이라고 표현하기도 한다.)
  - Vanishing Gradient (a가 0보다 클 때 / a>0)
    - sigmoid의 gradient는 x의 절댓값이 증가하는 만큼 작아지게 되는 것에 비해, ReLU의 역함수는 1이므로 ReLU의 경우에는 gradient로 상수를 갖게된다. 일정한 gradient값은 빠르게 학습하는 것을 도와준다.
  - x 가 양수이기만 하면 그래디언트가 1로 일정하므로 그래디언트가 죽는 현상을 피할 수 있고, 미분하기도 편리해 계산복잡성이 낮습니다. 하지만 x 가 음수이면 그래디언트가 무조건 0이 된다는 단점이 있습니다. (leaky relu가 고안된 이유)
- 7. Normalization, Standardization, Regularization
  - 공통점은 Overfitting을 방지하기 위해 사용한다.
  - Normalization과 Standardization 모두 데이터를 좀 더 compact 하게 만들기 위하여 사용한다.
    - scale의 범위가 너무 크면 노이즈 데이터가 생성되거나 overfitting이 될 가능성이 높아지기 때문에 한다.
    - 또한, 활성화 함수(activation function)를 거치는 의미가 사라집니다. 값이 너무 커지게 되므로 활성화 함수를 거친다고 하여도 한쪽으로 값이 쏠릴 가능성이 높기 때문이죠.
  - Normalization : 다양한 방법이 존재.
  - Standardization : 표준화 확률변수를 구하는 방법이며 Z-Score 구하기라고 할 수도 있겠다.
  - Regularization : 말 그대로 '제약'을 거는 작업이다. 제약을 건다는 것은 모델을 좀 더 complex 혹은 flexible 하게 만든다는 것입니다. Overfitting은 모델이 train data에 너무 딱 맞게 학습이 돼서 발생하는 문제인데 여기서 특정 penalty 값만 더해주거나 빼주어서 모델의 복잡도를 조정해주어도 train data에 딱 맞게 돼버리는 상황을 어느 정도 예방할 수 있습니다.
    - **가중치 감쇠(weight decay), 가중치 상한, 드롭아웃(drop-out)** 등이 있음.
- 8. 파라미터(parameter)와 하이퍼 파라미터(hyperparameter)의 차이
  - 파라미터(parameter) : 파라미터란 모델 내부에서 확인이 가능한 변수이다. 즉, 데이터를 통해서 산출이 가능한 가능한 값이다.
    - 예를 들어 인공신경망에서의 가중치, 선형 회귀나 로지스틱 회귀분석에서의 결정계수.
  - 하이퍼 파라미터(hyperparameter) : 하이퍼 파라미터는 모델에서 외적인 요소라고 할 수 있다. 즉, 데이터 분석을 통해 얻어지는 값이 아니고 주로 사용자에 의해 정해진다. (grid search, random search)
- 9. rnn과 lstm
  - rnn : RNN은 히든 노드가 방향을 가진 엣지로 연결돼 순환구조를 이루는(directed cycle) 인공신경망의 한 종류입니다. RNN은 관련 정보와 그 정보를 사용하는 지점 사이 거리가 멀 경우 역전파시 그래디언트가 점차 줄어 학습능력이 크게 저하되는 것(vanishing gradient problem)으로 알려져 있습니다.

- lstm : 이 문제를 극복하기 위해서 고안된 것이 바로 LSTM입니다. LSTM은 RNN의 히든 state에 cell-state를 추가한 구조입니다.