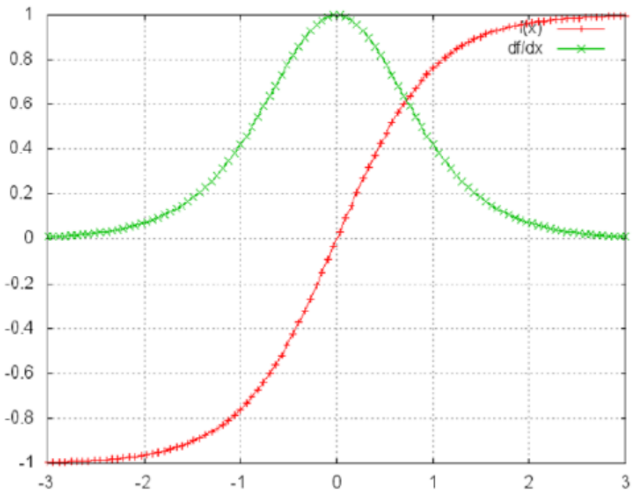
😀

#30 - Basic of Graph Convolution Network

1. Review from last lecture

- RNN
  - CNN의 경우 2D 픽셀 Value에 사용
  - 시간에 따른 Sequential Data를 분석하는 다른 구조가 필요하기 때문에 발생됨
  - 구조
    - Sequential한 Data
    - Cell들이 이어져있는 구조
      - Text의 경우 Sentence의 단어 + 단어 + 단어 (Character + Character + Character)
      - Sequential Data의 경우 깊이가 깊어질 수 있기 때문에 Gradient Vanishing이 발생할 수 있음
  - Gradient Vanishing



    - 빨간색 Tanh 그래프와 초록색 Gradient 그래프

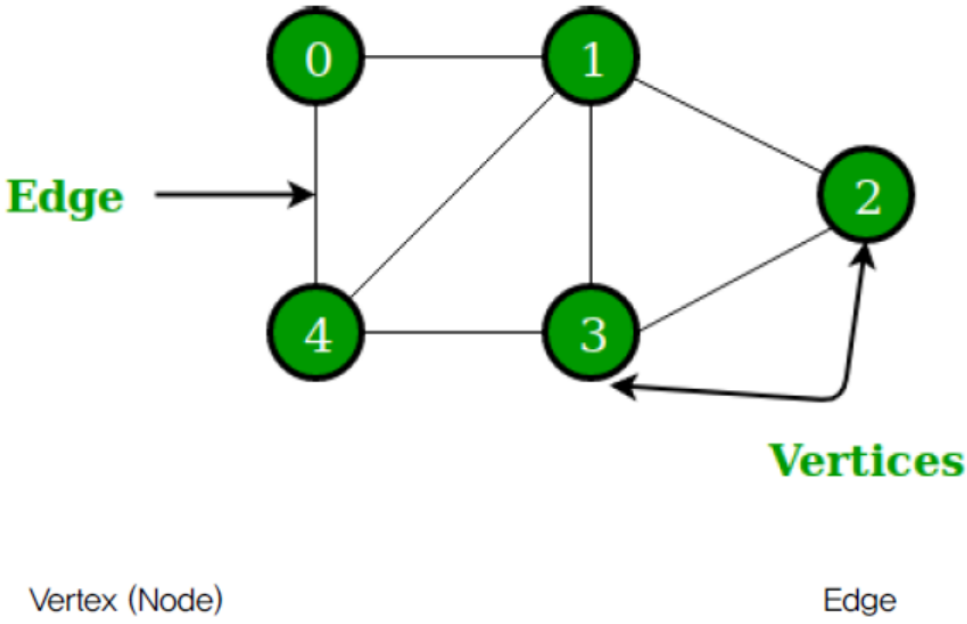
        - 양 끝단으로 갈 수록 Gradient가 0으로 가까워짐
        - 따라서 양 끝단의 뒤에 있는(=얕은 쪽의 Input에 가까운) Data는 저 0에 가까운 Gradient에 해당 Step의 Gradient가 곱해져서 더 작고 0에 가까운 값으로 변하게되는 문제가 발생함

- LSTM
    - Cell State와 Hidden State를 받아서 Sigmoid나 Tanh를 취하여 나온 결과값을 Coefficient로 사용
    - 결과값으로 나온 Coefficient를 얼마나 중요하게 생각할 것인가, 중요한 Coefficient를 붙여서 다음 Cell로 넘길 것인가 (=Gate)
    - 여러 Gate를 통과해서 다양한 정보를 적절한 비율로 섞어서 다음 Cell로 전해주는 것
    - Hidden State에 Activation Function을 거치지 않고 단순한 곱셈과 덧셈 연산만 수행하여 Gradient Vanishing 문제 해결
    - 문제점 : Gate의 연산량이 많아져서 시간이 오래걸림 (=Computation Cost가 높다)
- GRU
    - Gate수가 더 작아서 연산이 빠름
    - 다음 Cell로 넘겨주는 값이 Hidden State 하나만 넘겨줌 (LSTM은 Hidden State와 Cell State)
    - LSTM과 GRU의 성능은 비슷하지만, GRU는 Computation Cost가 낮아서 연산이 빠르므로 효율적임

2. What is Graph?

- Convolution : 필터를 이동시키면서 Dot Product하여 새로운 값을 얻는 방식
- Network : Neural Network
- Graph?
    - Data Representation - Image and Sentence
        - 어떤 Data로부터 내재되어 있는 숨겨진 값 (이미지의 Label 또는 Sentence의 의미, 피처를 추출)을 뽑아내는 것을 Neural Network이 그동안 해오던 일

- 그럼 Data를 어떻게 표현해 왔는가?
    - Image : Grid 형태로 Pixel이 배열되어 있고, 흑백 or 컬러에 따라 채널 (1 or 3)으로 표현
    - Sequential Data (Sentence) : Sequential을 구성하는 하나의 단어 또는 글자가 배열이 되어 있는 형태
    - 따라서 Neural Network를 구성시, Data의 구조에 따라 적절한 Neural Network 방식을 선택해왔음 (CNN → Image, RNN → Sentence)
    - 하지만 Data의 다른 형태들도 존재함
        - 그래프 형태
            - Social Graph : 나와 친구간의 관계정보를 담고 있는 Data
            - 3D Mesh : 게임과 영상에 사용되는 Data
            - Molecular Graph : 화학 분야에서 사용하는 분자 그래프
- 그래프의 정의
    - 전산학적인 정의 : Vertices의 Set과 Edge의 Set으로 이루어져 있는 형태



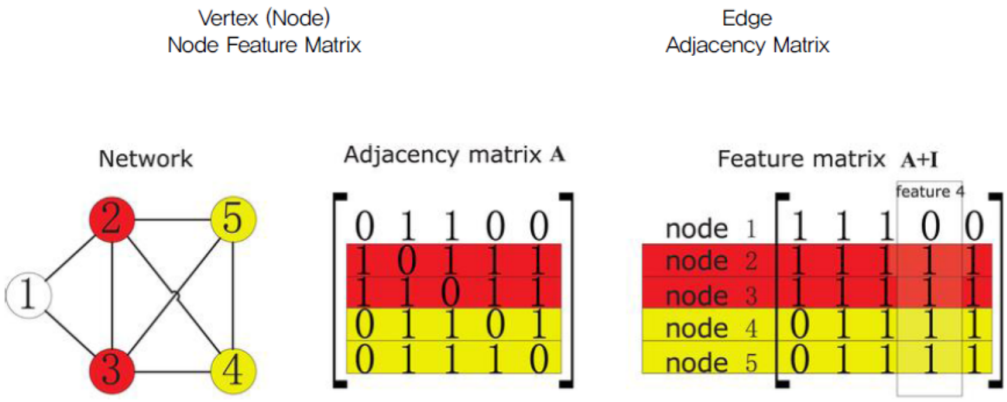
Vertex (Node) Edge

- 종류
    - 방향성 여부
        - Directed Graph : Edge의 방향성이 있는 Graph

    - Undirected Graph : Edge의 방향이 없는 Graph
- 중요도 여부
    - Weighted Graph : Edge마다 중요도, Weight (=Distance)가 들어가 있는 Graph
        - Vertices를 Node라고도 부름
    - Unweighted Graph : Edge 마다 중요도가 없는 Graph

  ex) Social Graph의 예시
    - Edge : 관계
    - Vertices : User, 사람들의 정보

  ex) Molecular Graph 화학
    - Edge : Atom들의 bond → 1차, 2차 결합, 공유 결합, 이온 결합
    - Vertices : 원자의 기호, Chemical Balance 등 정보
- 구조
    - Edge와 Vertex를 어떻게 Discrete 정보의 형태로 표현하는가?



    - Node Feature Matrix
        - 각각의 Vertex (Node)에 담긴 정보들을 나타낸 Matrix
        - 크기는 N x Feature 갯수로(f) 표현
        - (i, j)가 나타내는 바 : i번째 Node의 j번째 Feature가 무엇인가를 나타냄

      (ex) feature의 각각의 색깔을 나타내는 Column을 가지고 있다고 하고, 1번째 feature는 빨간색 여부를 나타낸다고 가정

→ (0,1) = 1 : 0번째 Node의 빨간색 유무는 있다로 표현 (=빨간색이다)

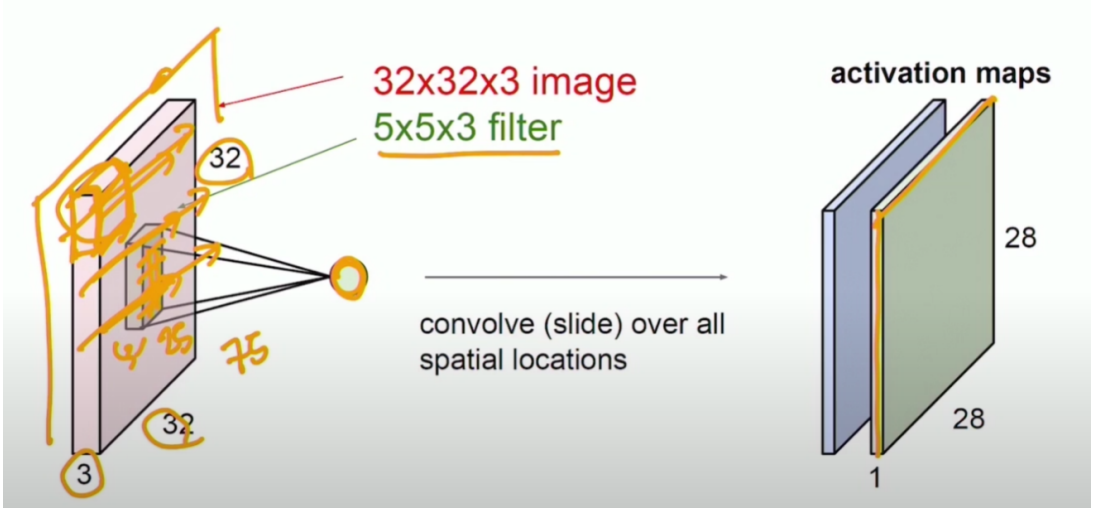
- Adjacency matrix
  - Vertex간의 Connectivity를 나타내는 Matrix
  - 크기는 N x N의 Node 갯수로 표현
  - Matrix (i, j)가 나타내는 바 : i번째 Node와 j번째 Node가 서로 연결이 되어 있는가 없는 가를 나타냄

    (ex) 연결이 되어 있으면(=Edge가 있으면) 1 없으면 0

    → (0, 1) = 1 : 0번째 Node는 1번째 노드와 연결이 되어 있다는 뜻

3. Graph Convolutional Network (GCN)

- 정의
  - Graph형태의 Data에 대해서 Convolutional 연산을 통해 Feature를 추출하는 Neural Network
  - Convolutional 연산



- 크기 32 x 32 Depth 3짜리 32 x 32 x 3 Image와 5 x 5 x 3짜리 Filter를 사용
  - Filter를 Image에 덮어씌운 형태
    - 한 층에 Filter의 크기인 5 x 5 = 25의 Weight가 있음
    - Depth가 3이므로 총 25 x 3 = 75개의 Parameter가 존재하게 됨

        - Filter의 Depth
            - Filter Size, Kernel Size라고도 부름
            - Image의 Depth와 동일하게 설정하며, Input Tensor에 의하여 자동으로 설정할 수도 있고 Manual 설정도 가능함
        - Filter를 Image에 덮여있는 영역에 Dot Product하여 하나의 값을 뽑아냄
        - Stride : Filter Sliding을 얼만큼 이동할 것인가
        - Output Tensor의 Size를 조절하기 위해서 외부에 Zero Padding을 덧댐
        - Zero Padding을 Stride만큼 건너 뛰면서 Dot Production하면서 새로운 Tensor를 만들어냄
        - Filter 하나마다, Depth 1짜리의 Activation Map이 도출됨
        - Filter를 여러개를 씌우면 Output Tensor의 Depth가 Filter의 갯수에 따라 결정됨
            - Filter를 64개를 쓰면, Output Image는 28 x 28 x 64
        - Receptive Field : Output 값이 다음 Neuron에 입력되면, 해당 Neuron이 볼 수 있는 범위
            - 각 Layer마다 깊어질 수록 Receptive Field는(RF라 지칭) 쌓이게 됨
            - 결과적으로 얕은 쪽 Layer에서 깊은 쪽 Layer로 점점 Effective RF가 넓어짐
- Weight Sharing
    - MLP에서는 어떤 Layer에 있는 하나의 Node가 이전 Layer에 있는 모든 Node와 연결이 되어 있음 → 깊어질 수록 Parameter 수가 증가하여 Overfitting이 일어날 수 있음
    - Image의 경우도 Image가 살짝 이동되어도 제대로 적용이 안됨
    - 하지만 CNN 에서는 Filter를 이동시켜 연산을 하기 때문에 작은 Weight를 전체 Image에 공유가 된다
    - RNN에서도 똑같이 적용됨
    - 장점
        - Reduce the number of parameters

        - Parameter의 수가 적어진다
        - 모델 Capacity도 낮아져서 Overfitting이 낮아짐
        - 계산해야할 값도 적어져 Computational Cost도 낮아짐
    - Learn Local Features
        - Image에서 Local Feature(Filter가 적용된 Receptive Field)를 배우게 됨
    - Translation Invariance
        - MLP의 경우 Fixel하나만 이동해도 그 값이 크게 변했지만, CNN의 경우 Filter로 적용되는 Weight가 동일하므로, 그 값이 크게 변하지 않는다
- What should we update?
    - CNN을 Graph에 적용한다면 구조를 어떻게 바꾸어야 할지는 고민해보자
        - CNN Layer를 하나 거치게 되면, Layer는 Activation Map Value를 Update 함
        - Updated Activation Map에 있는 값들이 Activation Map의 상태를 결정함
        - Convolutional Layer가 하는 역할
            - Activation Map Value가 담고 있는 Image또는 Activation Map의 상태를 Update해주는 역할
        - Graph의 상태를 결정하는 것
            - Image는 Activation Map에 있는 Value → Graph는 각 node에 담긴 Value
- How to update hidden states in GCN
    - 어떤 방식으로 업데이트해야 타당할까?
        - Convolution은 작은 Weight를 이동시키면서 연산을 하는 것

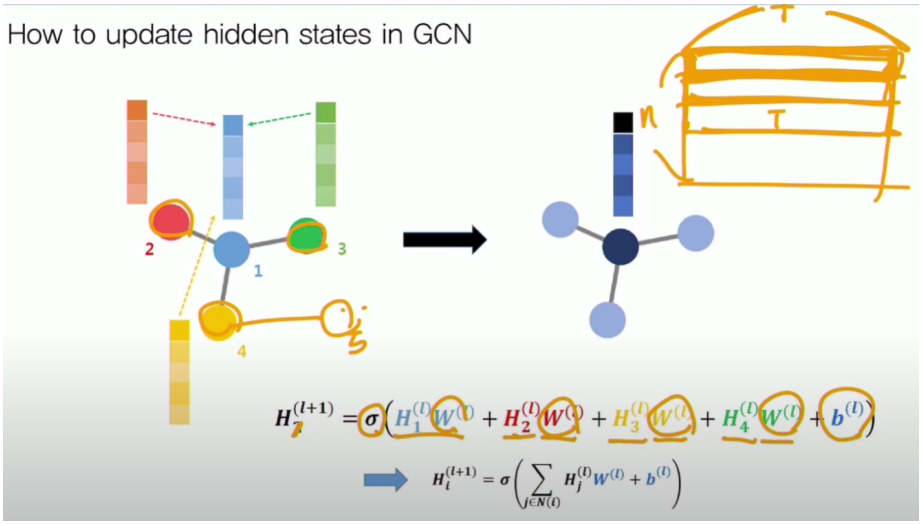
          ① Weight Sharing을 한다

          ② Local Feature만 배운다

          ③ 해당 Neuron이 Receptive Field를 갖게 됨
        - Graph에는 어떻게 적용할까?
            - Receptive Field의 적용

- Node의 Feature를 업데이트할 때 CNN과 같이 주변에 있는 Data들의 정보만 받아서 업데이트

ex) 다음 식과 같이 업데이트 하게됨



$$H_1^{l+1} = \sigma(H_1^{(l)}W^{(l)} + H_2^{(l)}W^{(l)} + H_3^{(l)}W^{(l)} + H_4^{(l)}W^{(l)} + b^{(l)})$$

$$\rightarrow H_i^{(l+1)} = \sigma(\Sigma_{i \in N(i)} H_j^{(l)}W^{(l)} + b^{(l)})$$

- L+1번째의 Layer를 통과하게 되면, H_1의 정보는 자기 자신의 Weight를 더하고 연결되어 있는 2번과 3번과 4번의 Hidden State에 Weight를 곱하고 Bias를 더한 후 Activation Function을 거친 값으로 업데이트
- 1번 Node와 연결되어 있는 2, 3, 4번의 정보만 Weight를 업데이트하여 다음 노드로 전달하였으므로 Local Feature만 배우게 됨
- 1, 2, 3, 4에 적용된 Weight가 동일하기 때문에 Weight Sharing을 함
- $W^{(l)}$: L번째 Layer의 Weight
- $H_1^{(l)}$: Hidden State의 첫번째 low가 L번째 Layer를 통과하고 나왔을 때의 값
  - CNN에서는 Activation Tensor를 hidden state
  - Graph에서는 각 Feature Matrix를 hidden state

- 그럼 모든 Node를 다 살펴보고 연결된 Node들의 업데이트를 일일이 수행해야 할까?
    - 반복적인 작업이 많아지게 되므로 비효율적임
    - Graph는 Adjacency matrix를 활용하여 Update를 하게 되는데 Connectivity를 행렬연산으로 처리하면 빠를 수 있음

4. Implementing GCN with Pytorch