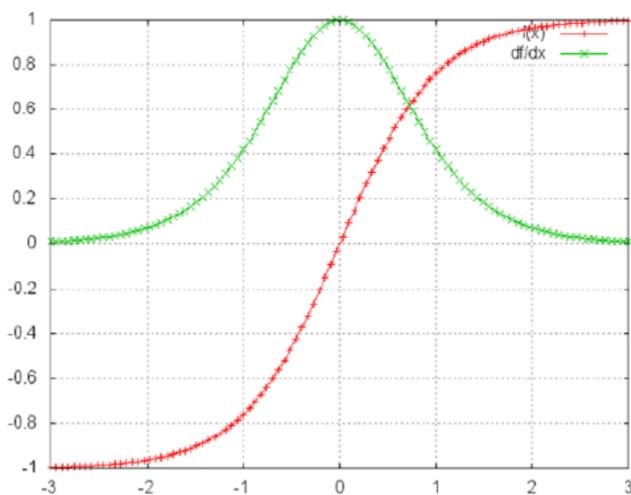




#30 - Basic of Graph Convolution Network

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



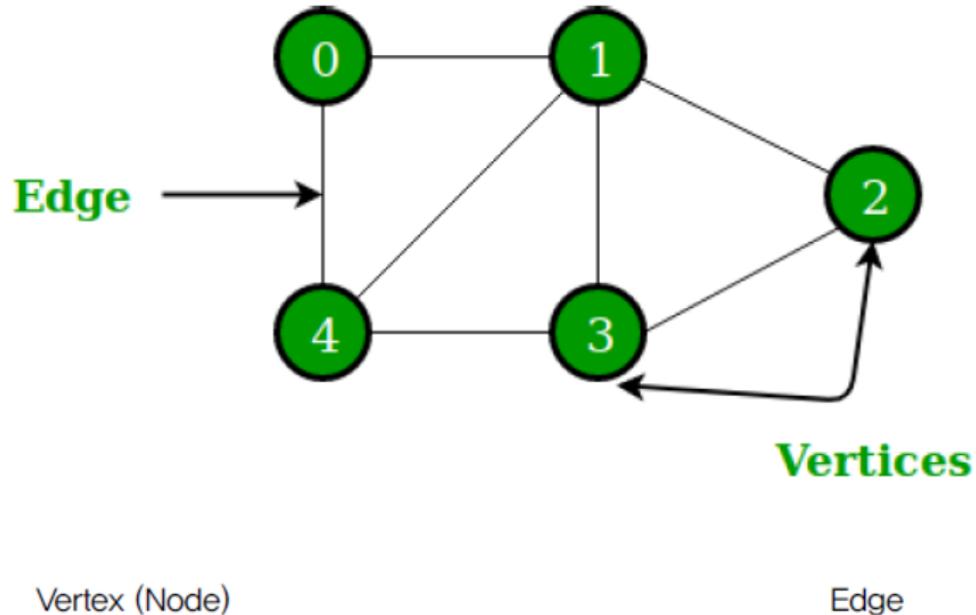
- 빨간색 Tanh 그래프와 초록색 Gradient 그래프

- 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 통해서 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?

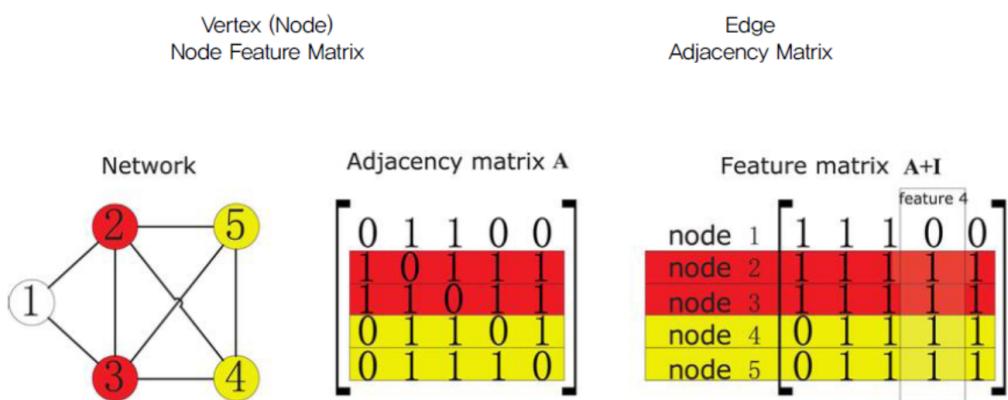
- Convolution : 필터를 이동시키면서 Dot Product하여 새로운 값을 얻는 방식
- Network : Neural Network
- Graph?
 - Data Representation - Image and Sentence
 - 어떤 Data로부터 내재되어 있는 숨겨진 값 (이미지의 Label 또는 Sentence의 의미, 피처를 추출)을 뽑아내는 것을 Neural Network이 그동안 해오던 일

- 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



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

- Undirected Graph : Edge의 방향이 없는 Graph
- 중요도 여부
 - Weighted Graph : Edge마다 중요도, Weight (=Distance)가 들어 가 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 $N \times \text{Feature 갯수}(f)$ 표현
 - (i, j) 가 나타내는 바 : i번째 Node의 j번째 Feature가 무엇인가를 나타냄

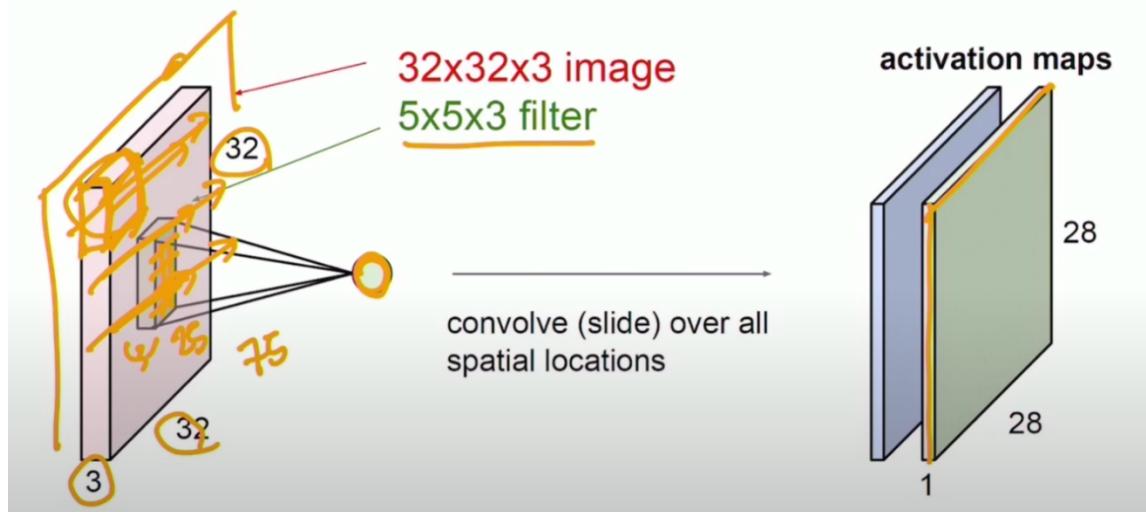
(ex) feature의 각각의 색깔을 나타내는 Column을 가지고 있다고 하고, 1번째 feature는 빨간색 여부를 나타낸다고 가정

$\rightarrow (0,1) = 1$: 0번째 Node의 빨간색 유무는 있다고 표현 (=빨간색이다)

- Adjacency matrix
 - Vertex 간의 Connectivity를 나타내는 Matrix
 - 크기는 $N \times N$ 의 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
 - $\rightarrow (0, 1) = 1$: 0번째 Node는 1번째 노드와 연결이 되어 있다는 뜻

3. Graph Convolutional Network (GCN)

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

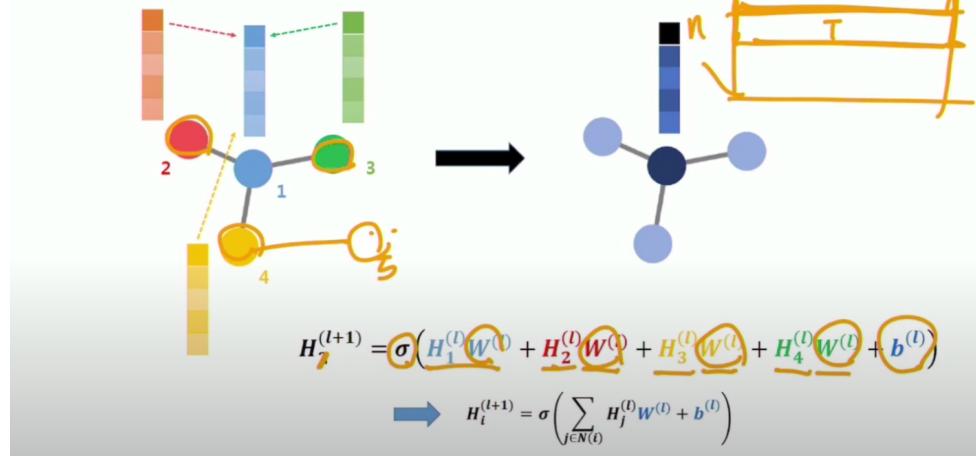


- 크기 32×32 Depth 3짜리 $32 \times 32 \times 3$ Image와 $5 \times 5 \times 3$ 짜리 Filter를 사용
 - Filter를 Image에 덮어씌운 형태
 - 한 층에 Filter의 크기인 $5 \times 5 = 25$ 의 Weight가 있음
 - Depth가 3이므로 총 $25 \times 3 = 75$ 개의 Parameter가 존재하게 됨

- 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 얼마나 이동할 것인가
 - Output Tensor의 Size를 조절하기 위해서 외부에 Zero Padding을 덧댐
 - Zero Padding을 Stride만큼 건너뛰면서 Dot Production하면서 새로운 Tensor를 만들어냄
 - Filter 하나마다, Depth 1짜리의 Activation Map이 도출됨
 - Filter를 여러개를 써으면 Output Tensor의 Depth가 Filter의 갯수에 따라 결정됨
 - Filter를 64개를 쓰면, Output Image는 $28 \times 28 \times 64$
 - 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 CNN에서는 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

- 계산해야 할 값도 적어져 Computational Cost도 낮아짐
- Learn Local Features
 - Image에서 Local Feature(Filter가 적용된 Receptive Field)를 배우게 됨
- Translation Invariance
 - MLP의 경우 Pixel하나만 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 Image 또는 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

How to update hidden states in GCN



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

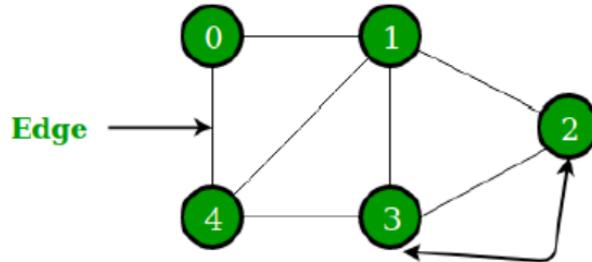
$$\rightarrow H_i^{(l+1)} = \sigma(\sum_{i \in N(i)} H_j^{(l)}W^{(l)} + b^{(l)})$$

- L+1번째의 Layer를 통과하게 되면, H_1 의 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 layer가 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

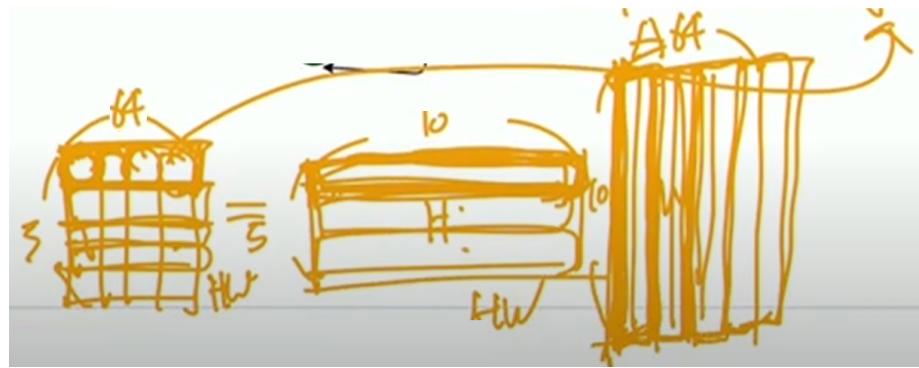
- Graph는 Adjacency matrix를 활용하여 Update를 하게 되는데 Connectivity를 행렬연산으로 처리하면 빠를 수 있음
- 그래서 다음과 같은 연산으로 처리함

$$\rightarrow H_i^{(l+1)} = \sigma(AH_j^{(l)}W^{(l)} + b^{(l)})$$

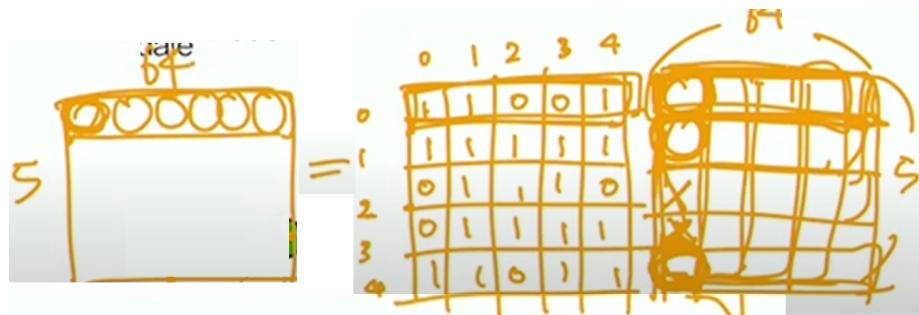
- 위의 수식의 각 구성요소를 예시로 설명해 보자. 다음과 같은 Graph가 있다고 가정



- Adjacency Matrix (A)
 - 각 Node의 갯수만큼 5×5 Matrix가 생성
 - (i, j) 값은 i 번째 노드와 j 번째 노드의 연결값을 나타냄
- Node Feature Matrix (H)
 - 이전 Layer의 Node 정보를 의미
 - Node 갯수 x Feature의 갯수에 따라 5×10 크기로 생성
- Weight (W)
 - Feature Matrix와 동일한 행의 크기 x Filter의 갯수이므로 $10 \times$ Filter 갯수(여기서는 64개라 가정)
 - Filter마다 다른지만, 같은 열의 Filter에서는 동일한 Weight 설정
- Node Feature x Weight (HW)

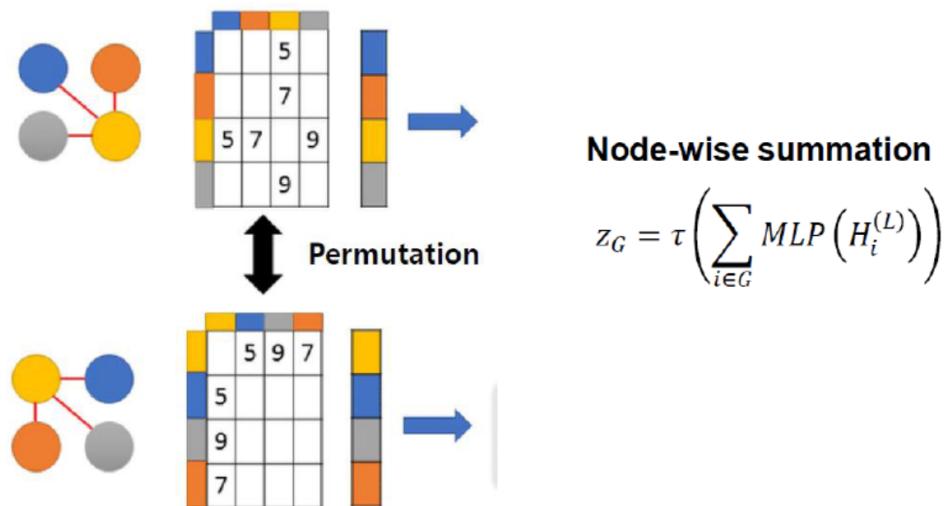


- Node Feature Matrix에 가중치를 적용한 행렬임
- 크기는 Node Feature Matrix의 행의 갯수 x Filter의 갯수, 5×64 의 크기
- (i, j) 의 의미는 i 번째 노드에 j 번째 Filter를 씌운 값
- 하나의 행은 가중치가 적용된 피쳐 정보를 담은 노드정보임
- $\text{Adjacency Matrix} \times \text{HW} (\text{AHW})$

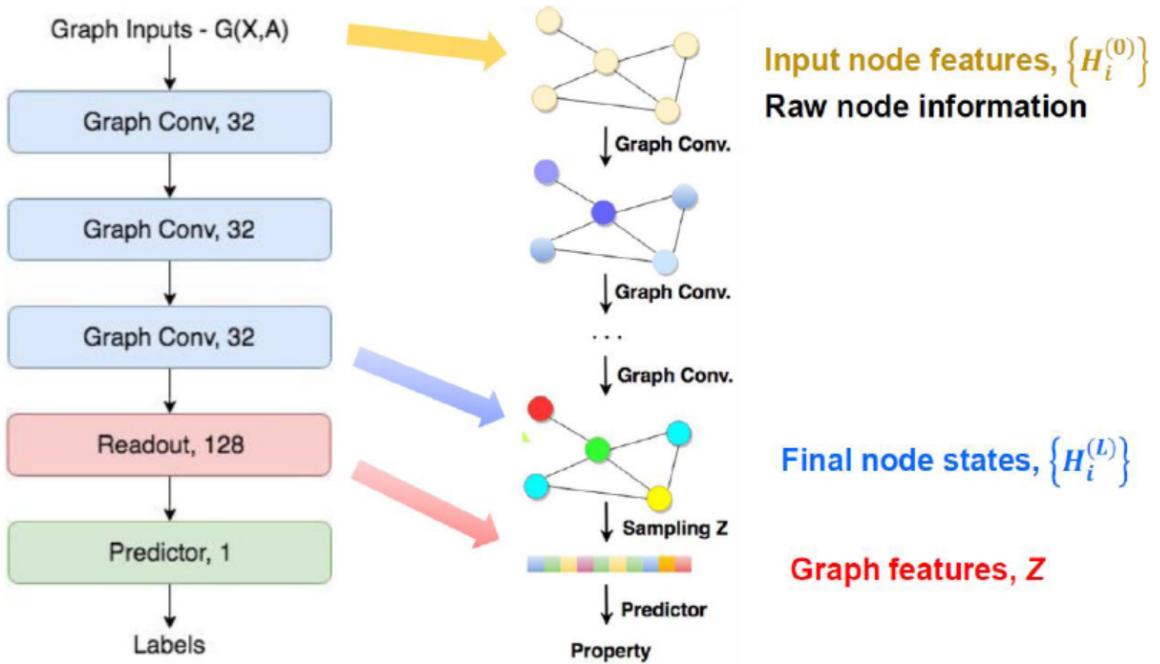


- 노드의 Connectivity 정보에 가중치가 적용된 피쳐 정보를 입한 행렬
- 크기는 노드의 갯수 x Filter의 갯수, 5×64 의 크기
- (i, j) 의 의미는 i 번째 노드와 연결되어 있는 노드들의 j 번째 피쳐들의 정보를 합한 값
- 이는 Convolutional Layer의 연산과 동일함
 - 주변에 있는 Node들의 정보를 가져와 합친 연산이므로
 - 즉, AHW로 최종적으로 산출되는 값들은 (Bias는 제외하고) Activation Function을 취해져서 새로운 Hidden State 값으로 설정되게 됨
- 새로운 Hidden State ($H^{(l+1)}$)

- Layer를 통하여 업데이트된 Node Feature Matrix를 의미
- 각각의 노드의 근처에 있는 노드들의 정보만 받아들여서 Convolutional 연산을 한 효과를 냈음
- Filter안에서 동일한 Weight가 적용되었으므로 Weight Sharing을 구현
- 행렬연산을 하기때문에 For문보다 빠름
- Gradient 계산도 병렬화되어 빠름
- Readout - Permutation Invariance

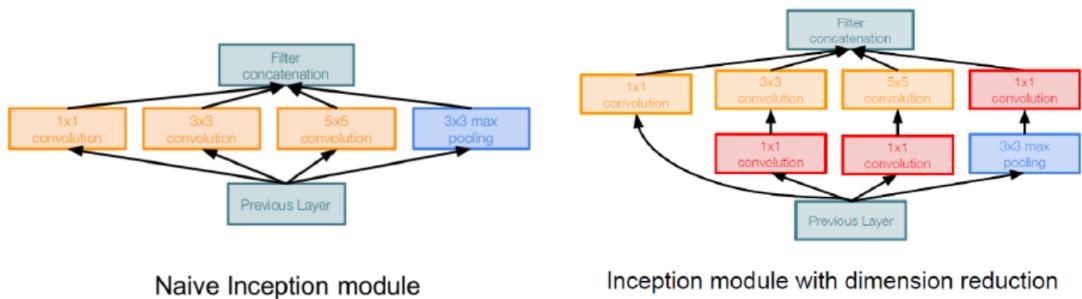


- Permutation
 - Graph를 표현할 때 활용되는 Node Feature Matrix에는 Node의 순서가 있음
- Permutation Variance
 - Node의 배치가 바뀔 경우 Node의 특성, Edge 정보는 동일한 데, Node Feature Matrix가 바뀌어 버려 결과에 영향을 주는 것
- Readout Layer
 - Node의 배치가 바뀌더라도 Node Feature Matrix 연산 결과가 바뀌지 않도록(Permutation Invariance) 만들어주는 Layer
 - 다양한 방식 중 가장 간단한 방식은 마지막 Layer에 MLP를 통째로 써우고 Activation Function을 써우는 방식
- Overall Structure of GCN



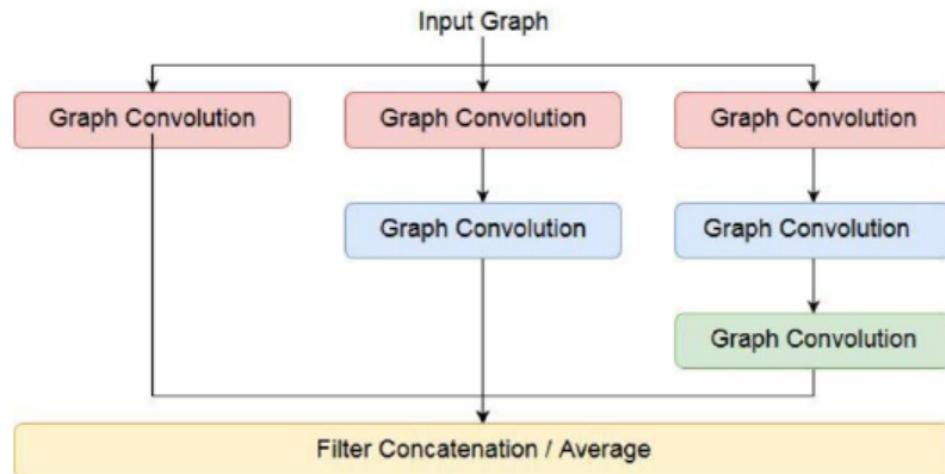
- ① Adjacency A와 Node Feature X로 나타내어지는 Input X, A에 Graph Convolution Layer를 적용 + Activation Function
- ② 각 Node와 연결된 인근 Node들의 피처 정보를 담은 고차원 정보의 결과값 도출
- ③ Readout Layer를 거쳐서 Permutation Invariance한 하나의 Vector 형태로 재생성
- ④ CNN에서의 Fully Connected와 같은 Predictor Layer를 거침
- ⑤ Property 결과를 얻음

- Advanced Techniques of GCN
 - CNN에서 사용했던(GoogLeNet) Advanced Skill을 GCN에 적용해보자
 - Inception Module



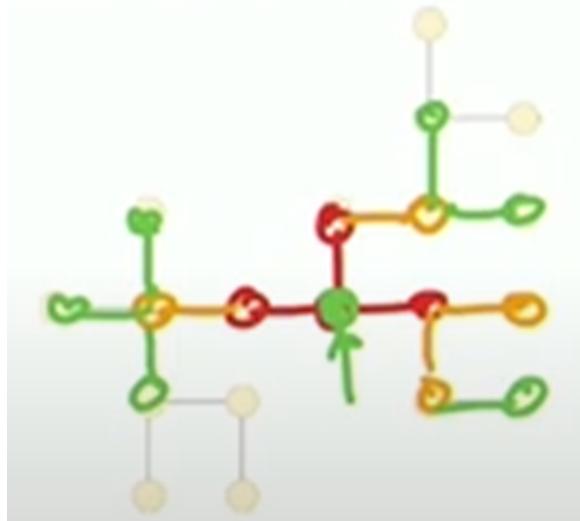
- 목적
 - Model Computation Efficiency를 높이고, Error Rate를 낮추기 위함
- 기능
 - Layer에 병렬적으로 여러 Size의 Filter를 써운 후 Concatenate시켜서 다음 Layer를 만드는 모듈
- 특징
 - Inception Module을 거칠 수록 Depth가 무한하게 커지게 되고 Computation Efficiency도 떨어지는 단점이 있음
 - 중간에 1×1 Convolution을 통해서 Bottle Neck 효과를 주어 Depth와 Computation Efficiency 문제를 해결
- Graph 적용 방식

Inception module in GCN

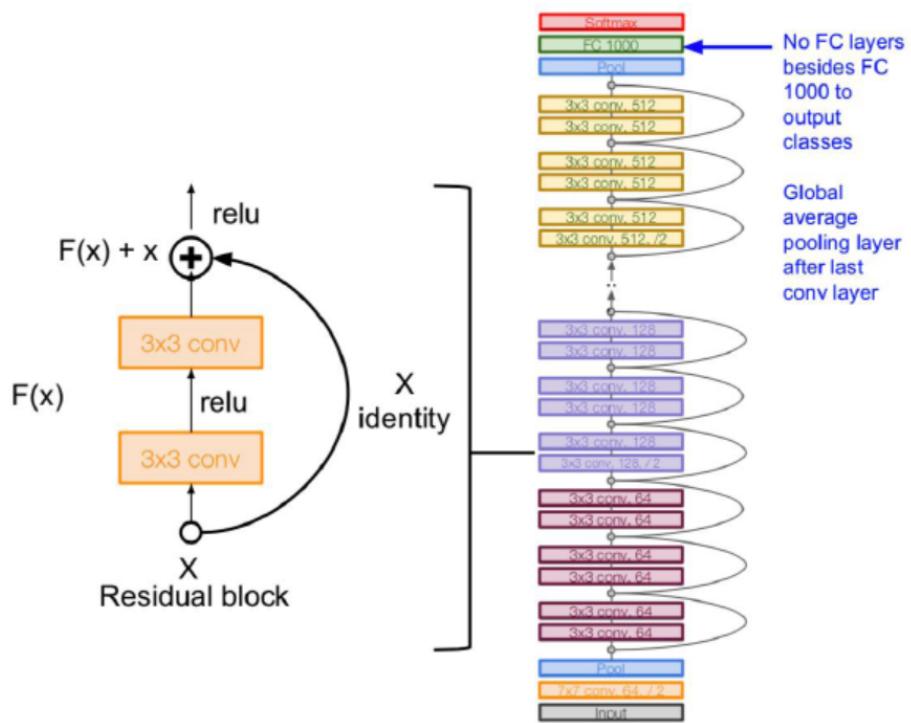


- Make network wider
- Avoid vanishing gradient
- 핵심은 Filter Size가 다른 것이므로 이를 어떻게 구현하느냐를 중점적으로 보아야함
- Filter Size는 다음 Neuron의 Receptive Field의 크기를 나타냄
 - Receptive Field는 중심 노드로 부터의 거리를 의미

- Filter Size가 클 수록 다음 Neuron에서 더 큰 값을 받아서 볼 수 있음
- 즉, 얼마나 멀리있는 Node를 볼 수 있느냐가 Receptive Field를 결정
- Adjacency Matrix를 한번 더 곱해서 Receptive Field Size를 조절할 수 있음
 - Graph Convolution 연산을 한번 더 적용한 것
 - 연산 1 : A노드와 거리가 1인 노드들을(빨간색) 볼 수 있음 → 연산2 : 연산1의 빨간 노드와 거리가 1인 노드들(노란색)까지 볼 수 있음

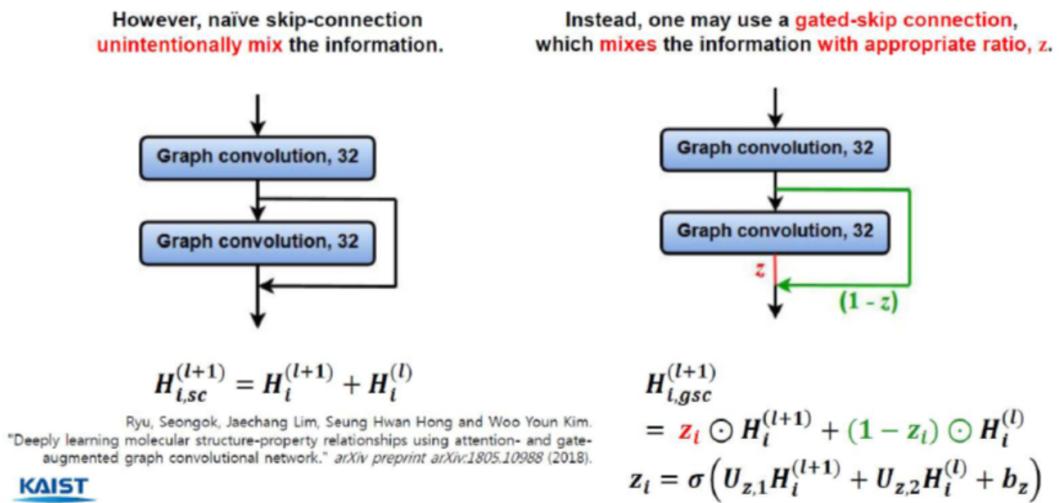


- Skip Connection
 - CNN (ResNet)에서 사용한 Advanced Skill 종류 중 하나임



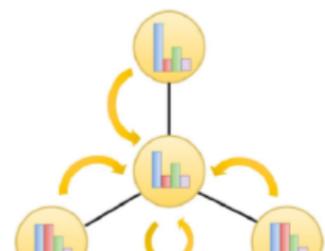
- 정의
 - Residual Connection을 사용하여 Layer가 Deep해질 수록 발생하는 문제를 해결하는 방식
 - Layer가 Deep할 수록 발생하는 문제 : Parameter가 증가하여 Overfitting 발생, Training Error가 높아짐
- 개념
 - Neural Network의 목적은 결국 Input값을 통해 Feature를 도출하고 도출된 Feature로부터 학습을 하는 것이 목적이나, Layer가 깊으면 이를 쉽게 학습하기가 어려움
 - 따라서 어느 정도 Layer가 지났을 때 Output과 그때까지의 결과값을 전달하여 줌으로써 학습을 더욱 빠르게 만들어 줌
 - 해당 Layer가 지난 후 분석한 결과값은 추후 전달하여 학습능력을 개선시킴
 - 즉, 깊은 Layer의 모델에서 학습해야 할 범위를 조금 앞당겨서 수행함으로써 학습 속도를 높이는 개념
- Graph Convolution Network에 적용하는 방식
 - 기존과 똑같이 중간 Layer에서 나온 결과값을 Output쪽에 더해주기만 하면 됨

- 단, Filter Size가 다르기 때문에 Node Feature Matrix의 크기를 맞춰줘야 하는 과정을 거쳐야 함
- Gated Skip Connection



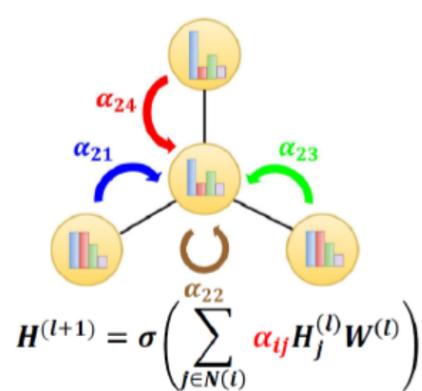
- Skip-Connection의 경우, 본의 아니게 mix된 정보를 그대로 받아드렸음
- Gated Skip Connection은 적절한 비율에 따라 선별된 정보를 받아드리는 개념
 - gate를 통과할 때 가중치를 각 Layer에 적용하여 구현
- Attention
- CNN에서는 사용하지 않던 방식임

Vanilla GCN updates information of neighbor atoms **with same importance**.



$$H^{(l+1)} = \sigma \left(\sum_{j \in N(l)} H_j^{(l)} W^{(l)} \right)$$

Attention mechanism enables it to update nodes **with different importance**.



- 기존방식은 노드의 정보를 업데이트 할 때 인근 노드들의 정보를 동일한 비율로 더하여 업데이트하였음
- 인근 노드의 정보를 적절한 비율로 적용하여 더하는 개념 → 비율을 학습
- 비율을 곱하는 것도 Elementalwise Multiply 방식으로 적용
- 비율 α 는 두 노드의 상관관계, 중요도에 따라 결정됨 $\rightarrow \alpha_{ij} = f(H_i W, H_j W)$
 - 여기서 적절한 함수 f는 \tanh 같은 함수임