10-617 Intermediate Deep Learning

HW3P2 Recitation (11/18/2022)

Udaikaran Singh, Anamika Shekhar, Athiya Deviyani

Table of Contents

Variational Autoencoder

Autoencoder

- Autoencoder allows for a non-linear dimensionality reduction.
  - An opposed to PCA, which is linear

$$z = f_\theta(x)$$

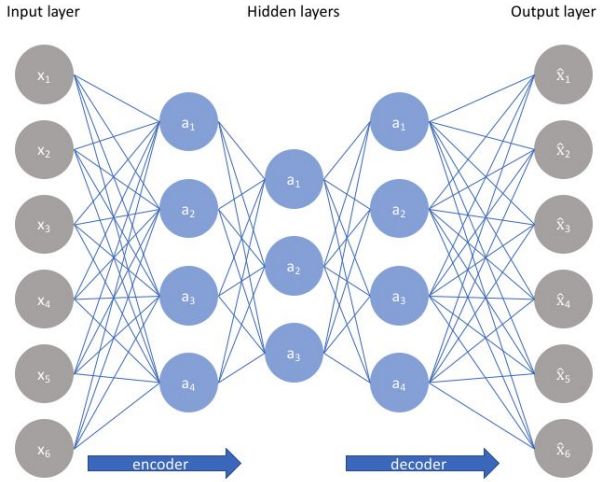
$$\hat{x} = g_\phi(z)$$

$$x \approx \hat{x}$$

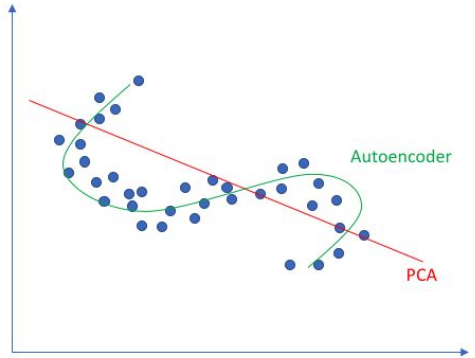
$$x, \hat{x} \in \mathbb{R}^d$$

$$z \in \mathbb{R}^k$$

$$k << d$$

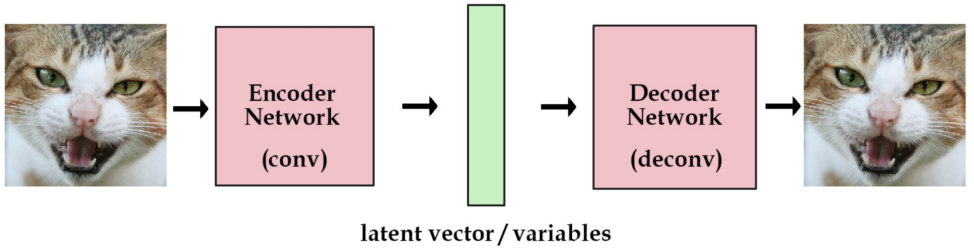


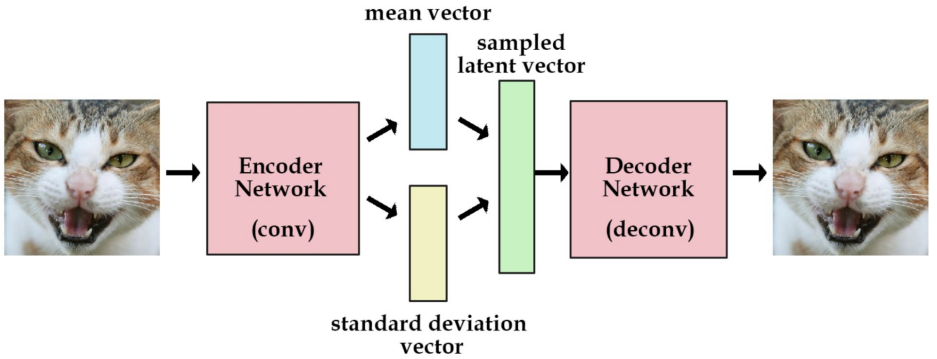
Linear vs nonlinear dimensionality reduction



Why an autoencoder isn't a generative model?

- An autoencoder has no explicit or implicit structure to the latent space
  - A normal autoencoder wants to find the optimal mapping from a vector to another vector
  - A VAE looks to map a vector onto a distribution (i.e find the most optimal distribution - we assume it to be a unit gaussian).
- The learning of an optimal distribution allows for sampling to generate new data.



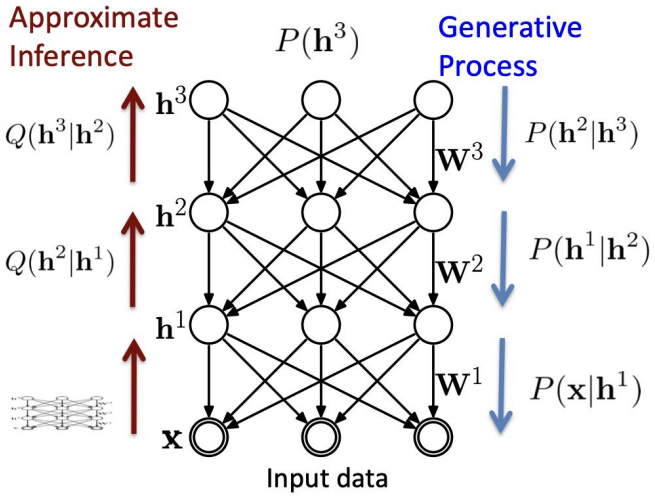


Variational Autoencoder

- VAE is generating distributions **h** in each step (conditioned on the previous layer)
- We assume each layer to be a unit gaussian with diagonal covariance.
- Notice the weights are tied between the encoder/decoder.

$$q(\mathbf{h}|\mathbf{x},\boldsymbol{\theta}) = q(\mathbf{h}^1|\mathbf{x},\boldsymbol{\theta})q(\mathbf{h}^2|\mathbf{h}^1,\boldsymbol{\theta})\cdots q(\mathbf{h}^L|\mathbf{h}^{L-1},\boldsymbol{\theta})$$

Each term may denote a complicated nonlinear relationship



- We assume that $\mathbf{h}^L \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
- The conditionals: $p(\mathbf{h}^\ell|\ \mathbf{h}^{\ell+1})$ $q(\mathbf{h}^\ell|\mathbf{h}^{\ell-1})$ are Gaussians with diagonal covariances

Training a VAE + Reparameterization Trick

- There are 2 terms in the loss:
  - log(p(x)) is the reconstruction loss (i.e. how well we we reconstruct the image)
  - KL divergence is how well we fit the assumed distribution (unit gaussian)

$$\mathcal{L}(\mathbf{x}) = \log p(\mathbf{x}) - \mathrm{D}_{\mathrm{KL}}\left(q(\mathbf{h}|\mathbf{x}))||p(\mathbf{h}|\mathbf{x})\right)$$

- Trading off the data log-likelihood and the KL divergence from the true posterior.

- **Issue**: each layer is generating a sampled hidden layer. We cannot perform gradient descent on a random variable.
- **Solution**: reparameterize the hidden layers to decouple the parameters from randomness.

$$\boldsymbol{\epsilon}^{\ell} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

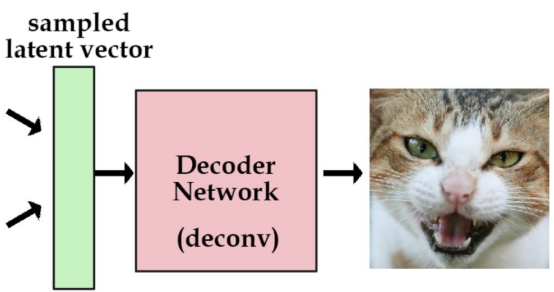
$$\mathbf{h}^{\ell}\left(\boldsymbol{\epsilon}^{\ell}, \mathbf{h}^{\ell-1}, \boldsymbol{\theta}\right) = \boldsymbol{\Sigma}(\mathbf{h}^{\ell-1}, \boldsymbol{\theta})^{1/2}\boldsymbol{\epsilon}^{\ell} + \boldsymbol{\mu}(\mathbf{h}^{\ell-1}, \boldsymbol{\theta})$$

- The recognition distribution $q(\mathbf{h}^{\ell}|\mathbf{h}^{\ell-1}, \boldsymbol{\theta})$ can be expressed in terms of a deterministic mapping:

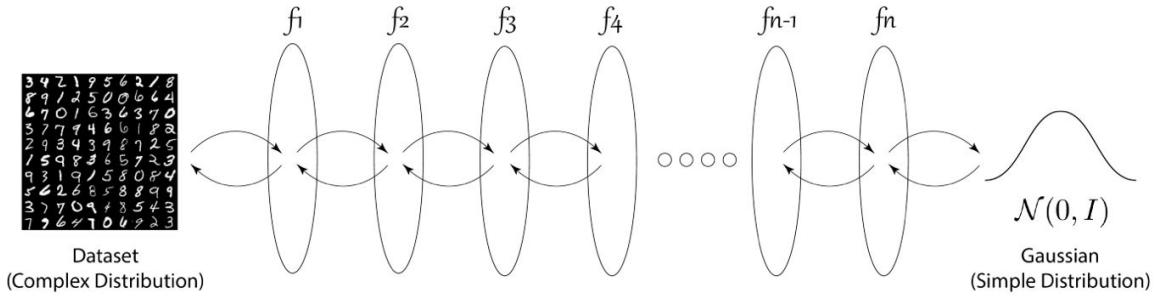
$$\underbrace{\mathbf{h}\left(\boldsymbol{\epsilon}, \mathbf{x}, \boldsymbol{\theta}\right)}_{\text{Deterministic Encoder}}, \quad \text{with} \quad \boldsymbol{\epsilon} = \underbrace{(\boldsymbol{\epsilon}^1, \ldots, \boldsymbol{\epsilon}^L)}_{\text{Distribution of } \boldsymbol{\epsilon} \text{ does not depend on } \boldsymbol{\theta}}$$

Generating Data from VAE

- Generate a random vector of the size of the latent vector from a unit gaussian.
- Feed this vector into the decoder network to generate new samples from your domain.



Normalizing Flows



Invertible Functions

- Normalizing flows assumes that the function is invertible
- Examples to the side of an invertible and non-invertible function

- In each step, you assume the datapoint is represented by a random variable.
  - Because of this, we use the change-of-variable function to represent the change in distribution

$$z = f_\theta(x)$$

$$x = f'_\theta(z)$$

$$f(x) = 7x + 5$$

$$f'(x) = \frac{x - 5}{7}$$

$$f(x) = x^2$$

$$f'(x) = \pm\sqrt{x}$$

$$p_X(\mathbf{x};\theta) = p_Z(\mathbf{z}) \left| \det \frac{\partial \mathbf{f}_\theta^{-1}(\mathbf{x})}{\partial X} \right|_{X=\mathbf{x}}$$

Function Composition + Objective

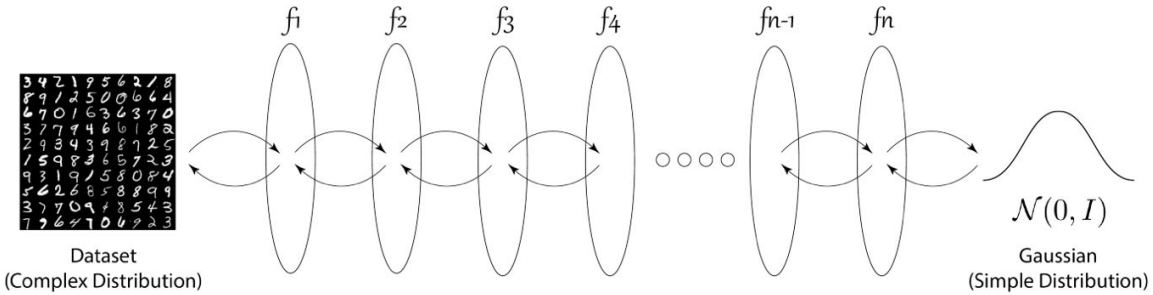
$$\mathbf{x} = \mathbf{f}_\theta^M \circ \cdots \circ \mathbf{f}_\theta^1(\mathbf{z}^0); \qquad p_X(\mathbf{x};\theta) = p_{Z^0}(\mathbf{z}^0) \prod_{m=1}^{M} \left| \det \frac{\partial (\mathbf{f}_\theta^m)^{-1}}{\partial Z^m} \right|_{Z^m = \mathbf{z}^m}$$

Maximum log-likelihood objective

$$\max_\theta \log p_X(\mathcal{D};\theta) = \sum_{\mathbf{x}\in\mathcal{D}} \left(\log p_Z(\mathbf{z}) - \log \left| \det \frac{\partial (\mathbf{f}_\theta)^{-1}}{\partial X} \right|_{X=\mathbf{x}} \right)$$

Sampling from a Normalizing Flow

- The model looks to find a series of composed invertible functions that project from the original data onto a distribution.
- To generate, you sample from the chosen distribution and use the inverse functions to return back a potential sample from the original dataset.
- Flaws: because they are very constrained (by the need for invertible functions), the tend to perform worse than GAN's and VAE's.



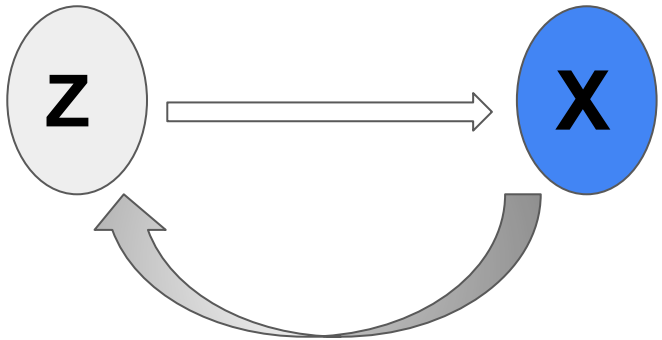
Variational Inference

Why do we need it ??

Posterior Distribution of latent Variable z given observed variable X is given as:

$$p(z|X) = \frac{p(X|z)\,p(z)}{\int p(X|z)\,p(z)\,dz}$$

Denominator is hard to compute !!!



q(z) ~ p(z|X)

Kullback–Leibler divergence

- Metric that quantifies how much one probability distribution differs from another probability distribution

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

$$D_{KL}(P||Q) = \int P(x) \log \frac{P(x)}{Q(x)} dx$$

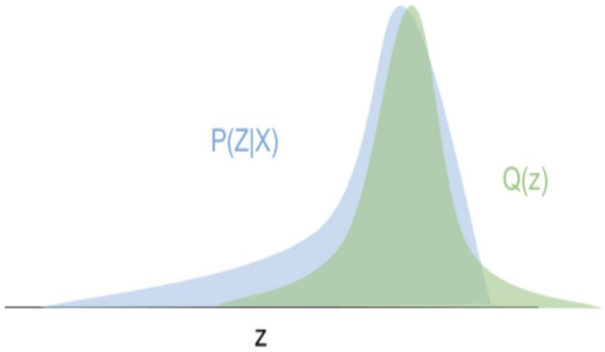
How do we find q(z) ?

We need to minimize the KL divergence of q(z) and p(z|X)

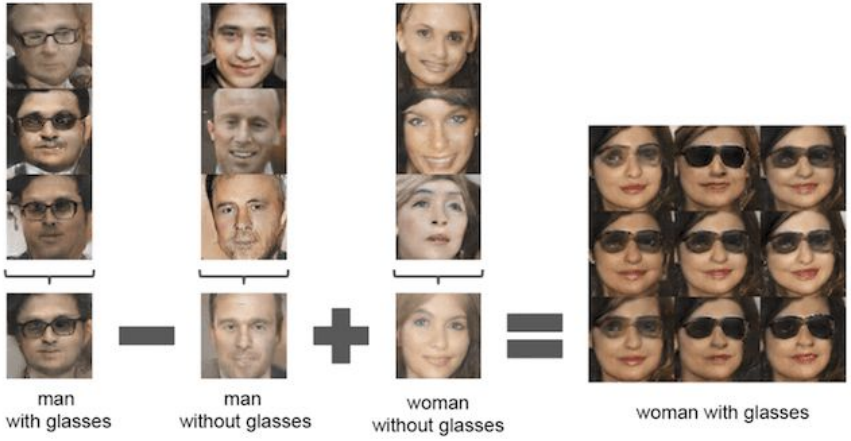
$q^*(z) = \text{argmin}_{q \epsilon Q}\ KL(\ q(\ z)\ \ ||\ p(\ z|X)\ )$

where

$$KL(\ q(\ z)\ \ ||\ p(\ z|X)\ ) = \int q(\ z)\ \log \frac{q(\ z)}{p(\ z|X)} dz$$



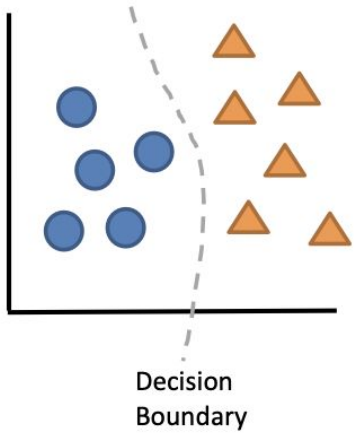
Generative Adversarial Network (GAN)


man
with glasses
man
without glasses
woman
without glasses
woman with glasses

Discriminative vs. Generative

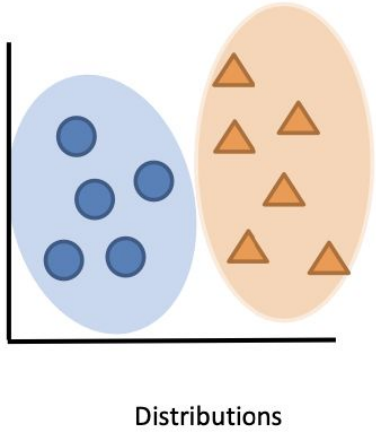
Discriminative models

- Learn conditional distribution P(Y|X)
- Learns decision boundary between classes
- Examples: logistic regression, SVM



Generative models

- Learn the joint distribution P(Y,X)
- Learns actual probability distribution of the data
- Examples: Naive Bayes, Gaussian Mixture Models



Generative Adversarial Network (GAN)

Aim to generate data
similar to the training data,
like VAEs

Generative **Adversarial** Network (GAN)

Made up of two networks ‘competing’ against each other (adversaries)

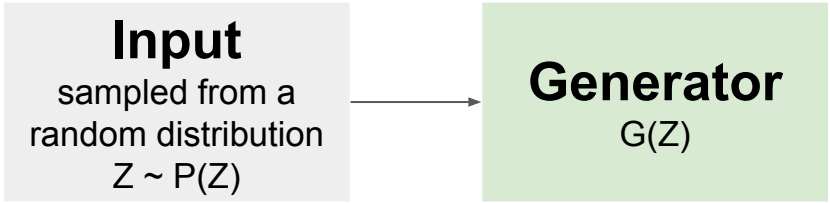
Generative Adversarial **Network** (GAN)

Deep neural network architectures

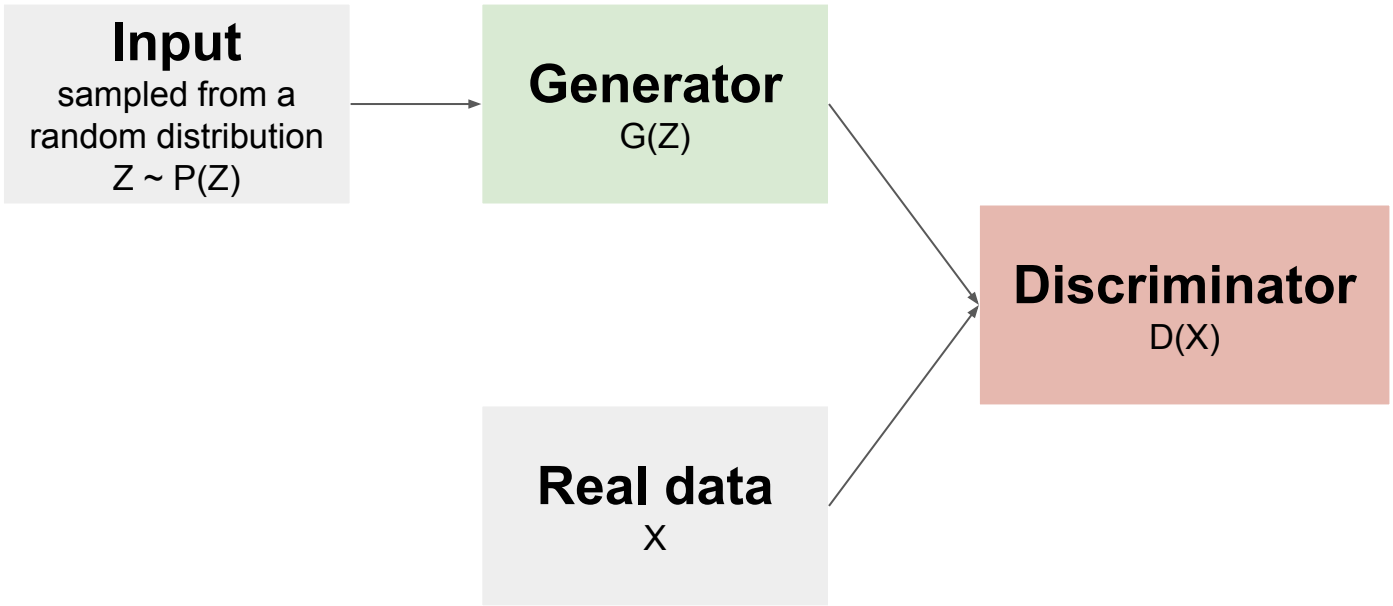
GAN Diagram

Input
sampled from a random distribution
Z ~ P(Z)

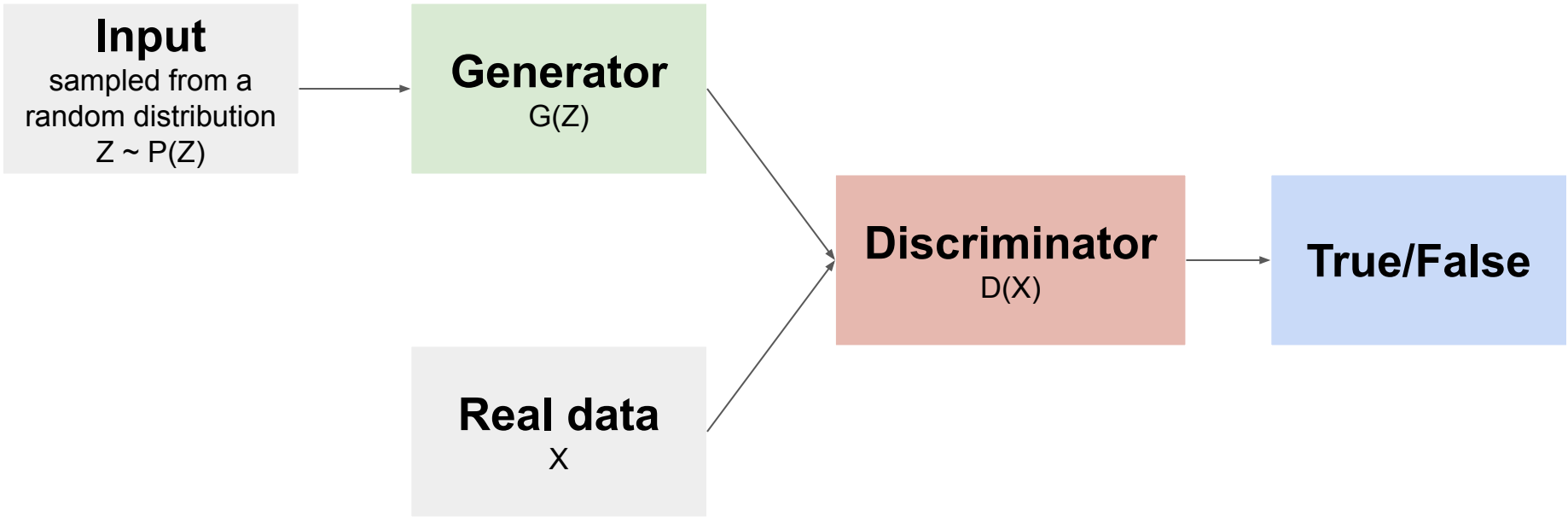
GAN Diagram



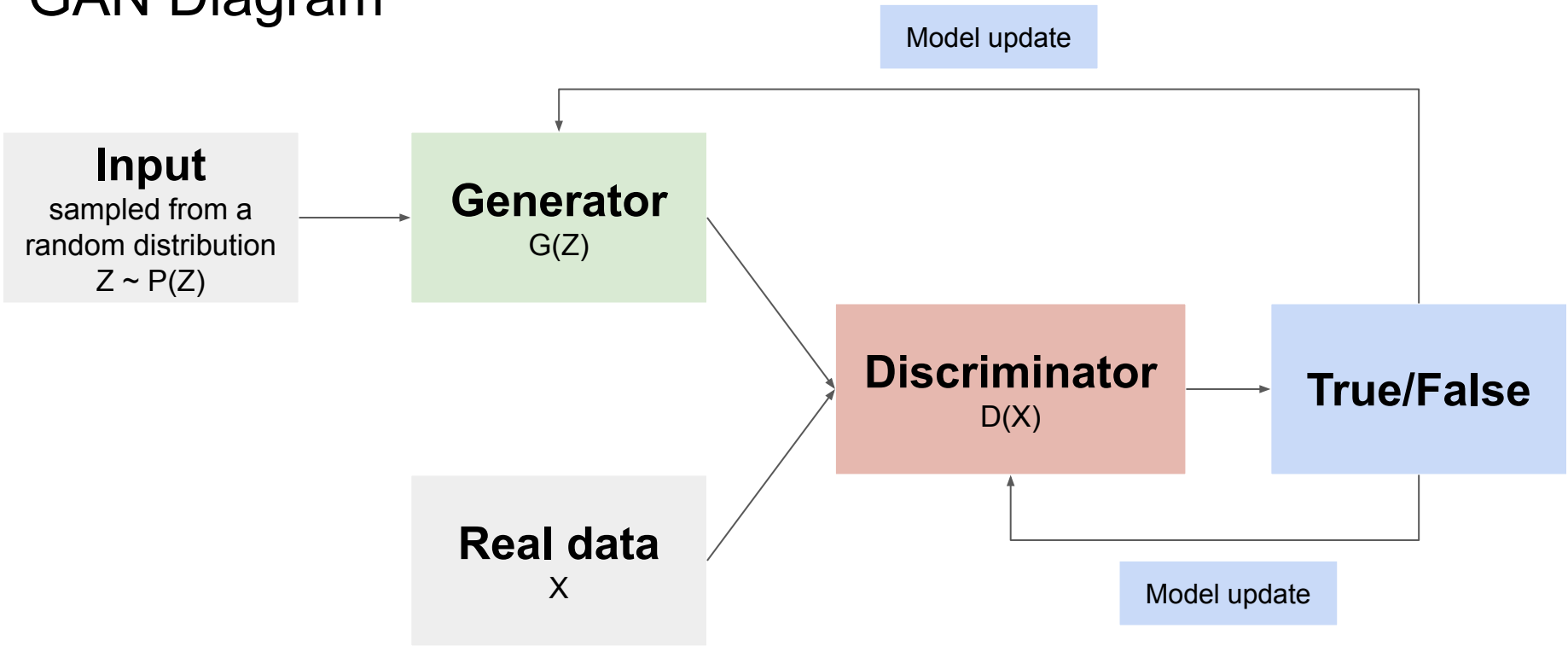
GAN Diagram


Input
sampled from a
random distribution
Z ~ P(Z)
Generator
G(Z)
Discriminator
D(X)
Real data
X

GAN Diagram


Input
sampled from a
random distribution
Z ~ P(Z)
Generator
G(Z)
Discriminator
D(X)
True/False
Real data
X

GAN Diagram


Model update
Input
sampled from a
random distribution
Z ~ P(Z)
Generator
G(Z)
Discriminator
D(X)
True/False
Real data
X
Model update

Each seeks to maximize its own success and minimize the success of the other: related to **minimax theory**

Generator
G(Z)

Goal: get better at fooling the discriminator, i.e. making fake data look real

Discriminator
D(X)

Goal: get better at distinguishing real data from fake data

Generator
G(Z)

Let $G(z, \theta^{(G)})$ be a differentiable function mapping from the latent space to the data space

Discriminator
D(X)

Let $D(x, \theta^{(D)})$ be a differentiable function mapping from the data space to a scalar between 0 and 1

Discriminator loss

Goal: get better at distinguishing real data from fake data, i.e. recognizing real images better and recognizing **generated images** better

Formalized goal:

$$\max_D V(D,G) = \mathbb{E}_{x\sim pdata(x)}[\log D(x)] + \mathbb{E}_{z\sim pz(z)}[\log(1 - D(\mathbf{G}(\mathbf{z})))]$$

Generator loss

Goal: get better at fooling the discriminator, i.e. making fake data look real

Formalized goal:

$$\min_G V(D,G) = \mathbb{E}_{z \sim pz(z)} [\log(1 - D(G(z)))]$$

0 <= D(X) <= 1
Where 0 is fake and 1 is real

GAN loss

Discriminator loss

$$\max_D V(D,G) = \mathbb{E}_{x\sim pdata(x)}[\log D(x)] + \mathbb{E}_{z\sim pz(z)}[\log(1 - D(G(z)))]$$

Generator loss

$$\min_G V(D,G) = \mathbb{E}_{z\sim pz(z)}[\log(1 - D(G(z)))]$$

GAN loss

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim pdata(x)}[\log D(x)] + \mathbb{E}_{z\sim pz(z)}[\log(1 - D(G(z)))]$$

Training

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Training: freeze one when training the other

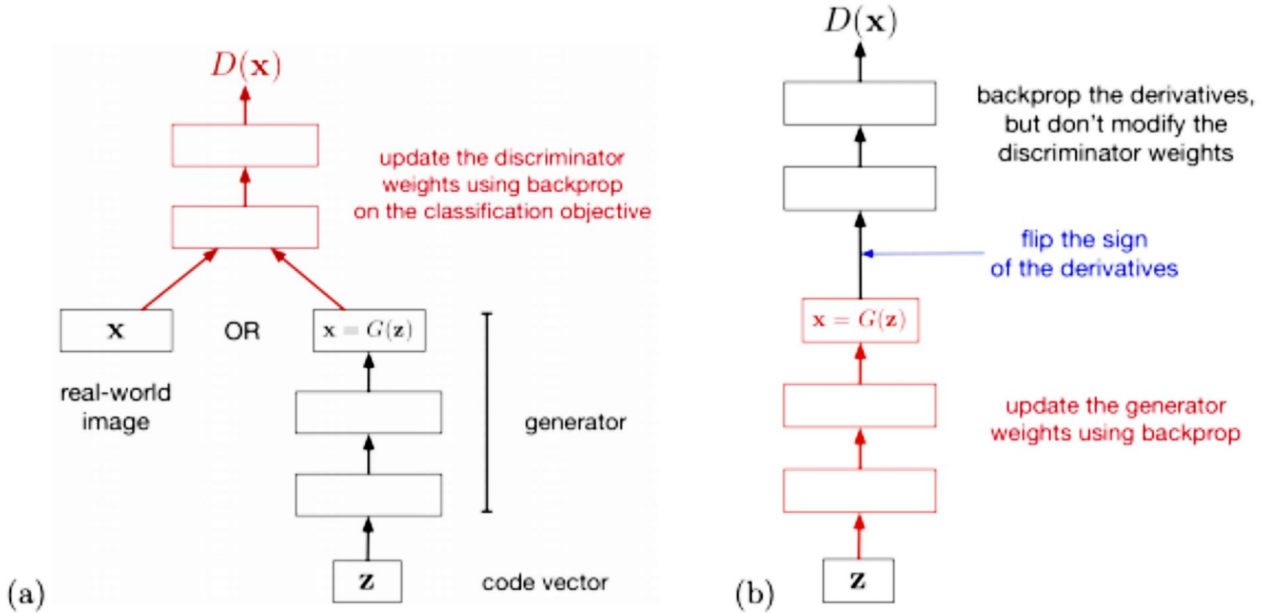


Figure 3: **(a)** Updating the discriminator. **(b)** Updating the generator.

Training: need to train them jointly but not at the same time

- Training the discriminator is relatively easy: binary classification problem of real and fake images
  - Easy because we have ground truth labels!
- While training the generator we don’t have such a ground truth
  - The only “evaluation metric” the generator has is its ability to fool the discriminator
  - However, if we train both the discriminator and generator together, there is the risk for the discriminator to start predicting always “true”, so that the generator obtains very good evaluation metrics
  - Those metrics will add no value to the strength of the generator: it would be like the discriminator is giving in to the generator. **Our final goal is that of having a strong generator, not a weak discriminator.**
  - This is the reason why, while training the generator, **we must freeze the discriminator’s weights**. By doing so, we do not allow the discriminator to weaken itself to accommodate the generator.

Rewrite the loss

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]$$

$$\begin{aligned} V(D,G) &= \int_x p_{\text{data}}(x)\log D(x)dx + \int_z p_z(z)\log(1-D(G(z)))dz \\ &= \int_x p_{\text{data}}(x)\log D(x)dx + \int_x p_G(x)\log(1-D(x))dx \\ &= \int_x p_{\text{data}}(x)\log D(x) + p_G(x)\log(1-D(x))dx \end{aligned}$$

Optimal discriminator

$$V(D,G) = \int_x p_{\text{data}}(x)\log D(x) + p_G(x)\log(1-D(x))dx$$

$$\frac{d}{dD(x)}\left(p_{\text{data}}(x)\log D(x) + p_G(x)\log(1-D(x)\right) = 0$$

$$\Leftrightarrow p_{\text{data}}(x)\frac{1}{D(x)} - p_G(x)\frac{1}{1-D(x)} = 0$$

$$\Leftrightarrow p_{\text{data}}(x)\frac{1}{D(x)} = p_G(x)\frac{1}{1-D(x)}$$

$$\Leftrightarrow p_{\text{data}}(x)(1-D(x)) = p_G(x)D(x)$$

$$\Leftrightarrow D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)}$$

Slides from 10-703 Deep Reinforcement Learning by Katerina Fragkiadaki

Optimal discriminator

$$V(D,G) = \int_x p_{\text{data}}(x)\log D(x) + p_G(x)\log(1-D(x))dx$$

$$\frac{d}{dD(x)}\left(p_{\text{data}}(x)\log D(x) + p_G(x)\log(1-D(x)\right) = 0$$

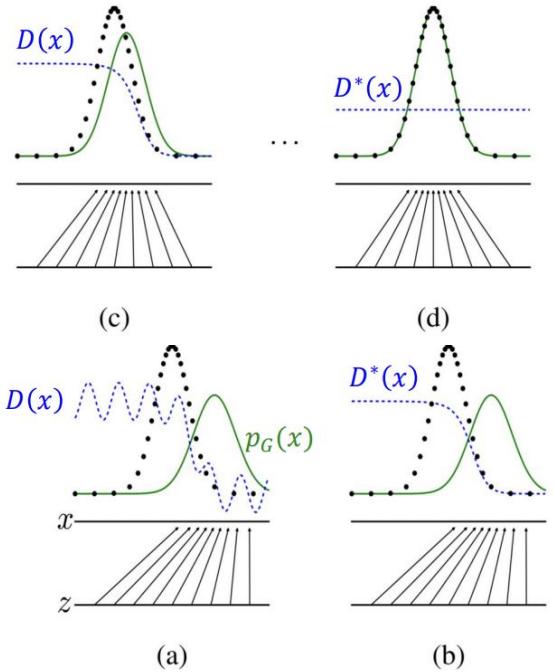
$$\Leftrightarrow p_{\text{data}}(x)\frac{1}{D(x)} - p_G(x)\frac{1}{1-D(x)} = 0$$

$$\Leftrightarrow p_{\text{data}}(x)\frac{1}{D(x)} = p_G(x)\frac{1}{1-D(x)}$$

$$\Leftrightarrow p_{\text{data}}(x)(1-D(x)) = p_G(x)D(x)$$

$$\Leftrightarrow D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)}$$

The best strategy for the discriminator is to learn the ratio of the probabilities of x under the data distribution and the generator distribution **p(data|x)**



Optimal generator (**HOMEWORK HINT!**)

$$
\begin{aligned}
C(G) &= \max_D V(G, D) \\
&= \mathbb{E}_{x \sim p_{data}(x)}[\log D_G^*(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D_G^*(G(z))] \\
&= \mathbb{E}_{x \sim p_{data}(x)}[\log D_G^*(x)] + \mathbb{E}_{x \sim p_G(x)}[\log(1 - D_G^*(x)] \\
&= \mathbb{E}_{x \sim p_{data}(x)}[\log \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)}] + \mathbb{E}_{x \sim p_G(x)}[\log(1 - \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)})]
\end{aligned}
$$

… continue it yourself :)

KL Divergence* between $p_{data}(x)$ and $p_{data}(x)+p_G(x)$!

$D_{KL}(p_{data}(x) \,||\, p_{data}(x)+p_G(x))$

*$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$

Problems and challenges

- **Mode collapse:** the generator produces limited varieties of samples (generated images converge to the same image)
- **Diminished gradient:** the discriminator gets too successful that the gradients vanish and the generator learns nothing
- **Non-convergence:** the model parameters oscillate, destabilize and never converge
- Unbalance between the generator and discriminator causes overfitting
- Highly sensitive to hyperparameters

Graphical Convolutional Networks (GCN)

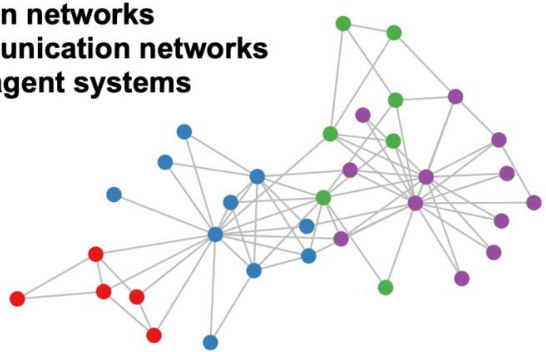
Paper: Semi-supervised Classification with Graph Convolutional Networks (Kipf et al., 2017)

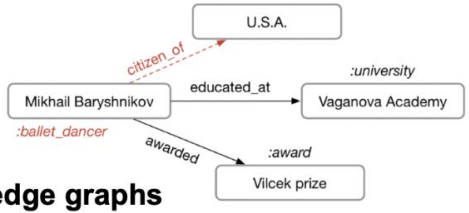
Slides derived from a really cool blog post https://ai.plainenglish.io/graph-convolutional-networks-gcn-baf337d5cb6b

Graphs, graphs, everywhere…

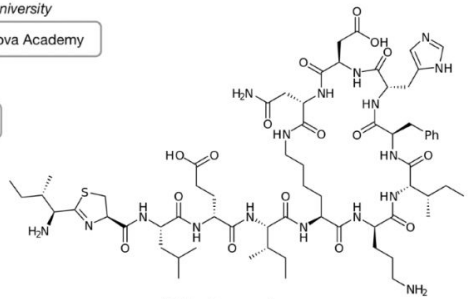
A lot of real-world data does not "live" on grids

Social networks
Citation networks
Communication networks
Multi-agent systems

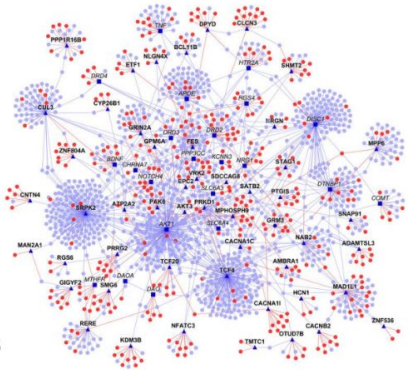




Knowledge graphs



Molecules



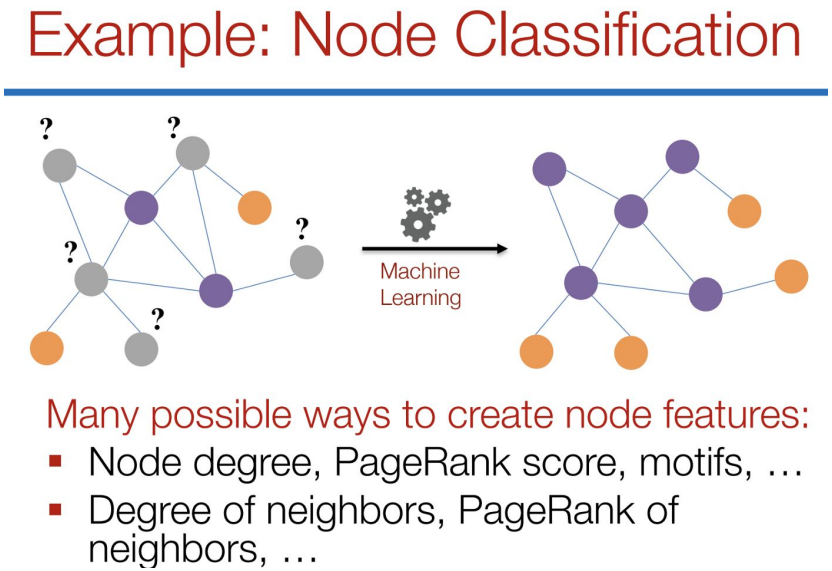
Protein interaction networks

Road maps

Homophily/birds of a feather assumption: nodes which are connected to each other are similar/related/informative of each other

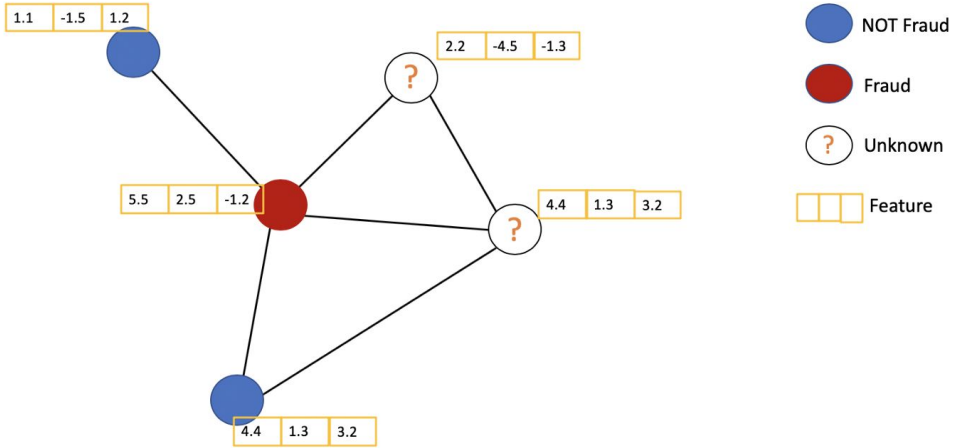
Tasks on graphical data

- **Node classification:** Predict a type of a given node
- **Link prediction:** Predict whether two nodes are linked
- **Community detection:** Identify densely linked clusters of nodes, also known as subgraph detection
- **Network similarity:** How similar are two (sub)networks



Graph Convolutional Networks (GCNs)

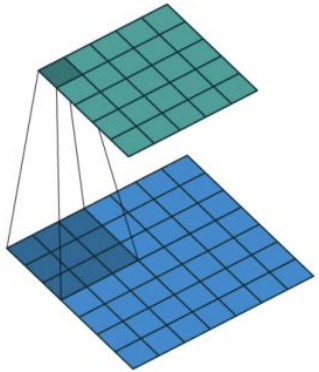
- GCN is a type of convolutional neural network that can work directly on graphs and take advantage of their structural information
- It solves the problem of classifying nodes (such as documents) in a graph (such as a citation network), where labels are only available for a small subset of nodes (semi-supervised learning)



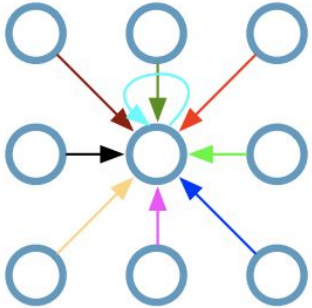
Graph **Convolutional** Networks (GCNs)

- Image as a graph: each pixel has 8 neighbors
- The node attributes are scalar values for grayscale image and 3-dimensional for RGB images
- The edge weights are binary (0 or 1), either present or absent

Single CNN layer with 3x3 filter:



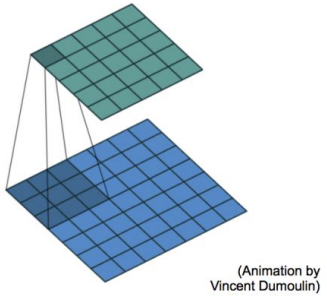
Image



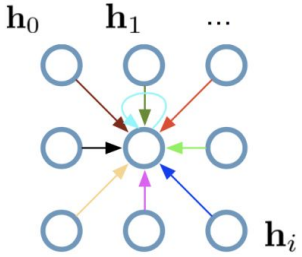
Graph

GCN updates

Single CNN layer with 3x3 filter:



(Animation by Vincent Dumoulin)



Update for a single pixel:

- Transform messages individually $\mathbf{W}_i\mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i\mathbf{h}_i$

$\mathbf{h}_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

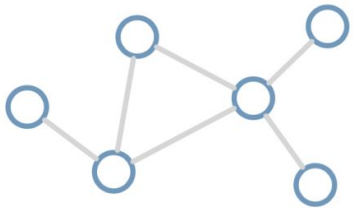
Full update:

$$\mathbf{h}_4^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l)} + \mathbf{W}_1^{(l)}\mathbf{h}_1^{(l)} + \cdots + \mathbf{W}_8^{(l)}\mathbf{h}_8^{(l)}\right)$$

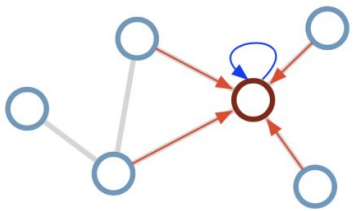
Slides from Thomas Kipf, University of Amsterdam

GCN updates

Consider this undirected graph:



Calculate update for node in red:



Desirable properties:

- Weight sharing over all locations
- Invariance to permutations
- Linear complexity O(E)
- Applicable both in transductive and inductive settings

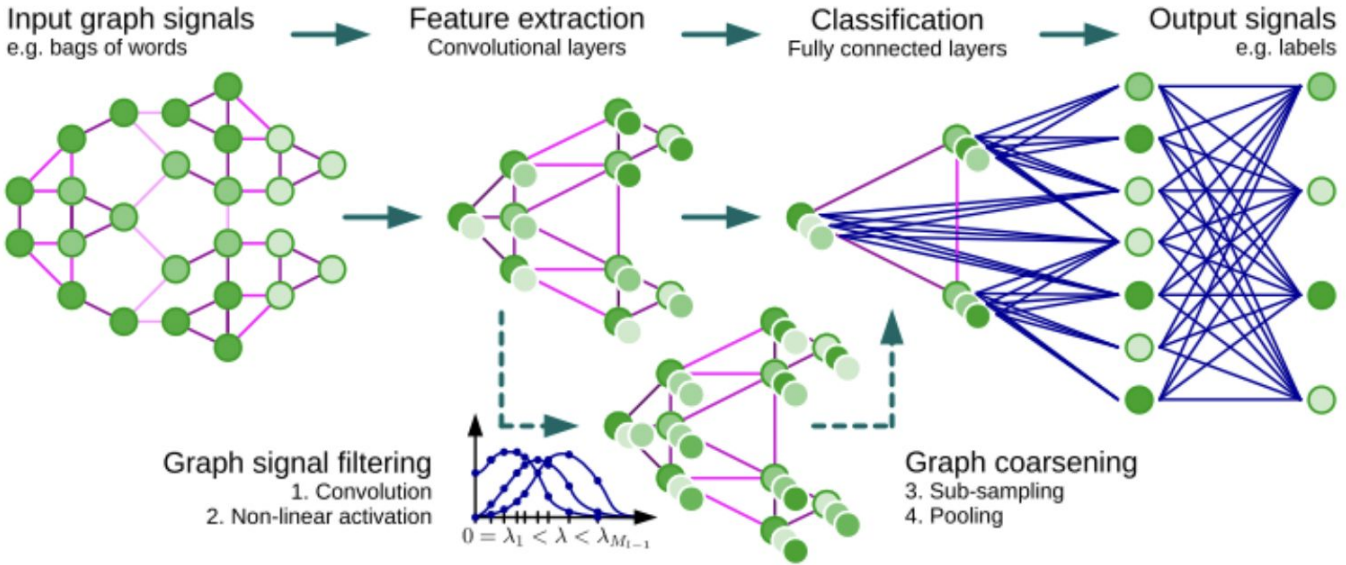
Update rule: $$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

Scalability: subsample messages [Hamilton et al., NIPS 2017]

$\mathcal{N}_i$: neighbor indices

c_{ij} : norm. constant (fixed/trainable)

Example GCN architecture



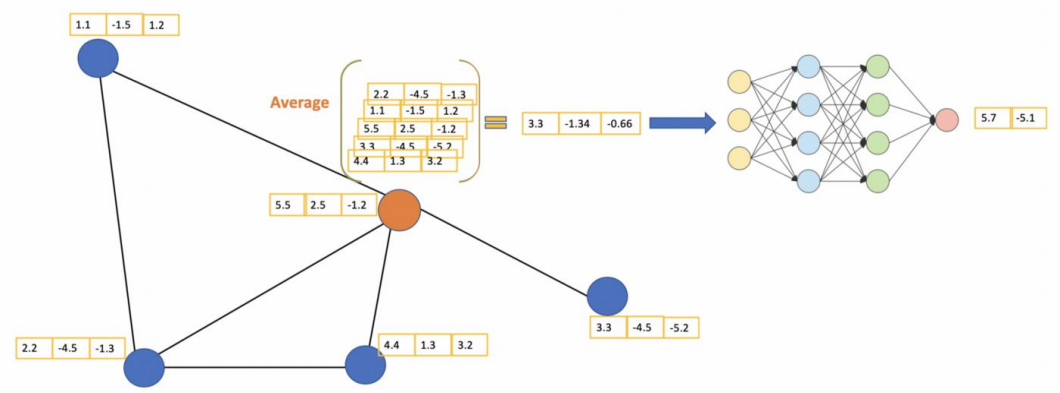
An example architecture of a GCN being used for classification

Diagram from : Defferrard et al. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering (2016)

GCN: the general idea

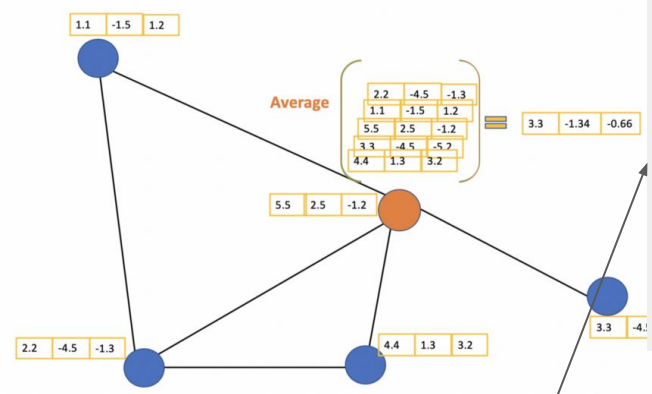
- For each node, we get the feature information from all its neighbors and its own features
  - Assume we use the average() function
  - We do the same for all nodes
- Feed these average values into a neural network

GCN: citation network example



- Each node represents a research paper, while the edges are the citations
  - We might pre-process by converting the raw papers into vectors (e.g. tf-idf, Doc2Vec)
- Let's consider the **orange node**: we get all the feature values of its neighbors, including itself, then take the average
- The result will be passed through a neural network to return a resulting vector

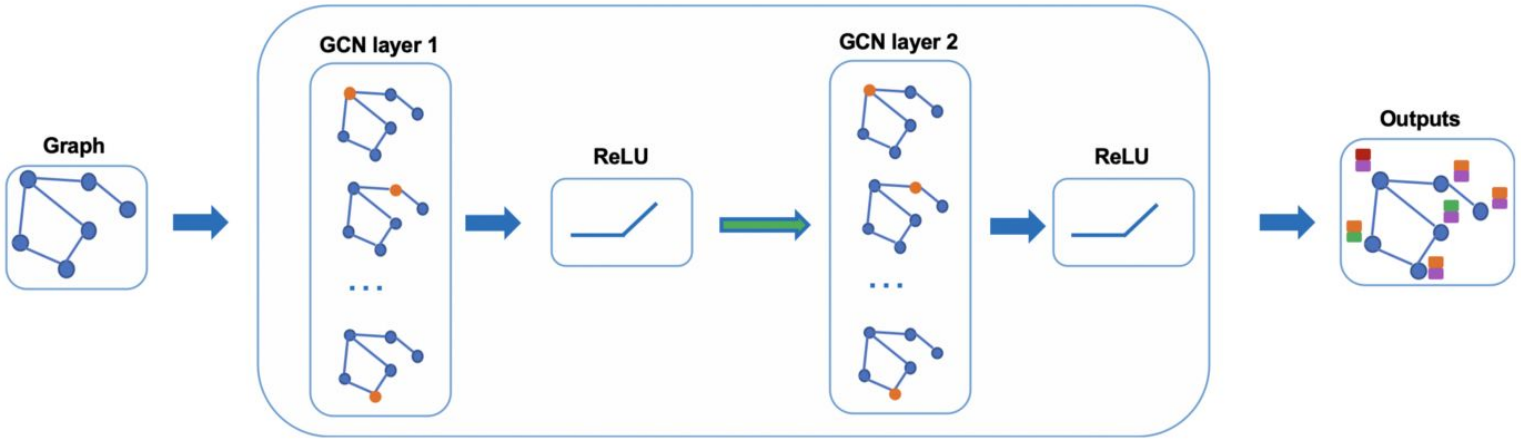
GCN: citation network example



In practice, we can use more sophisticated aggregate functions rather than the average function. We can also stack more layers on top of each other to get a deeper GCN. The output of a layer will be treated as the input for the next layer.

- Each node represents a research paper, while the edges are the citations
  - We might pre-process by converting the raw papers into vectors (e.g. tf-idf, Doc2Vec)
- Let's consider the **orange node**: we get all the feature values of its neighbors, including itself, then take the **average**
- The result will be passed through a neural network to return a resulting vector

Example of 2-layer GCN



the MATH behind
Graphical Convolutional Networks (GCN)

Paper: Semi-supervised Classification with Graph Convolutional Networks (Kipf et al., 2017)

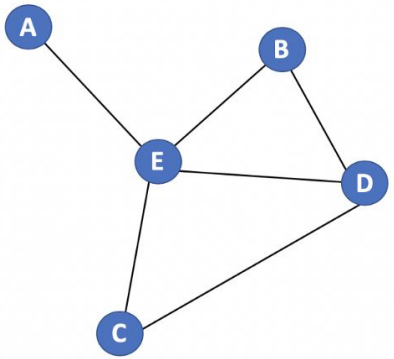
Slides derived from a really cool blog post https://ai.plainenglish.io/graph-convolutional-networks-gcn-baf337d5cb6b

Notation

Given an undirected graph $G = (V, E)$ with N nodes $v_i \in V$, edges $(v_i, v_j) \in E$, an adjacency matrix $A \in R^{N \times N}$ (binary or weighted), degree matrix $D_{ii} = \sum_j A_{ij}$ and feature vector matrix $X \in R^{N \times C}$ (N is #nodes, C is the #dimensions of a feature vector).

Consider a graph G below.

How can we get all the feature values from neighbors for each node? The solution lies in the multiplication of A and X.



Graph G

	A	B	C	D	E
A	0	0	0	0	1
B	0	0	0	1	1
C	0	0	0	1	1
D	0	1	1	0	1
E	1	1	1	1	0

Adjacency matrix A

	A	B	C	D	E
A	1	0	0	0	0
B	0	2	0	0	0
C	0	0	2	0	0
D	0	0	0	3	0
E	0	0	0	0	4

Degree matrix D

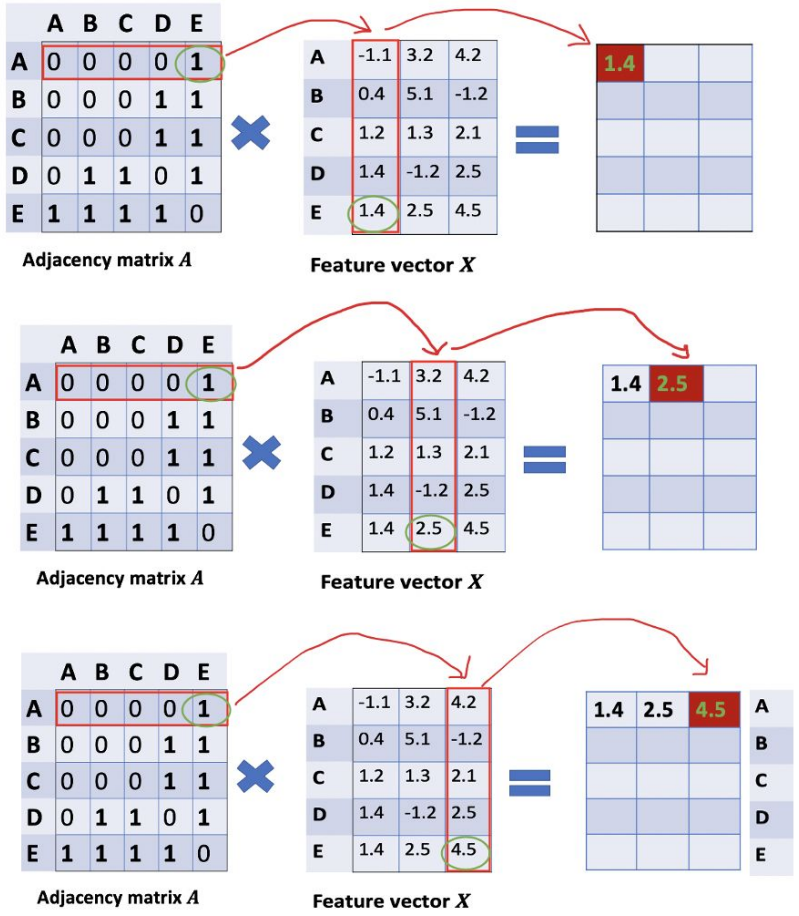
A	-1.1	3.2	4.2
B	0.4	5.1	-1.2
C	1.2	1.3	2.1
D	1.4	-1.2	2.5
E	1.4	2.5	4.5

Feature vector X

Take a look at the first row of the adjacency matrix, we see that node A has a connection to E.

The first row of the resulting matrix is the feature vector of E, which A connects to (Figure below).

Similarly, the second row of the resulting matrix is the sum of feature vectors of D and E. By doing this, we can get the sum of all neighbors' vectors.

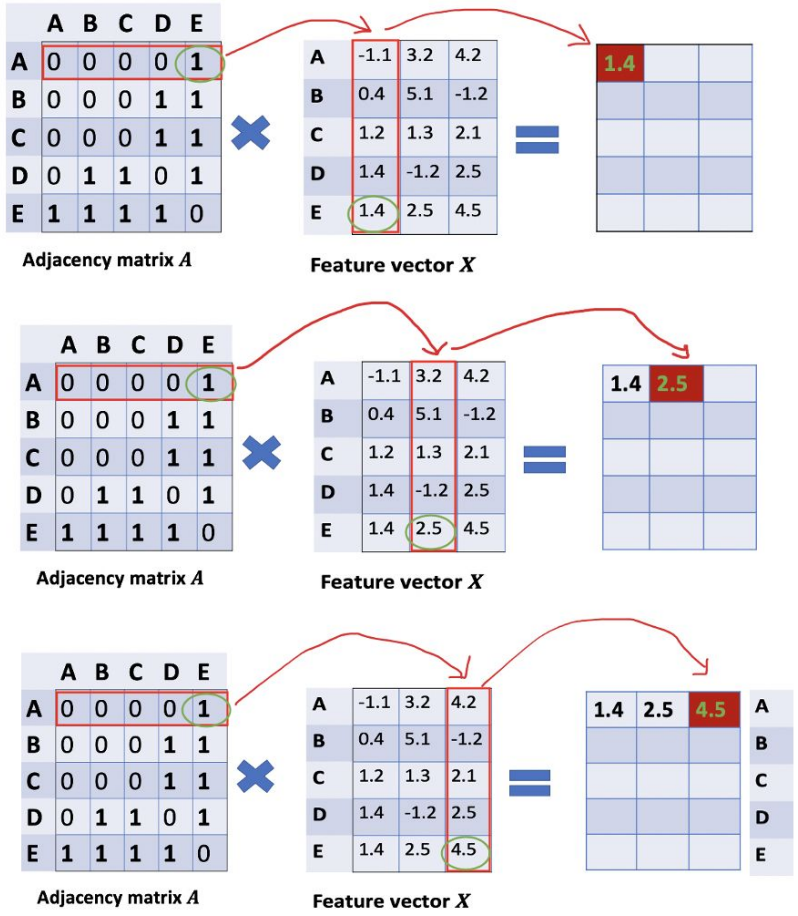


Some things missing…

1. We need to include the feature of the node itself! The first row of the result matrix should contain features of node A as well.
2. Instead of the sum() function, we need to take the average, or even better, the weighted average of neighbors' feature vectors.

Why don't we use the sum() function?

The reason is that when using the sum() function, high-degree nodes are likely to have huge v vectors, while low-degree nodes tend to get small aggregate vectors, which may later cause exploding or vanishing gradients (e.g., when using sigmoid). Besides, Neural networks seem to be sensitive to the scale of input data. Thus, we need to normalize these vectors to get rid of the potential issues.



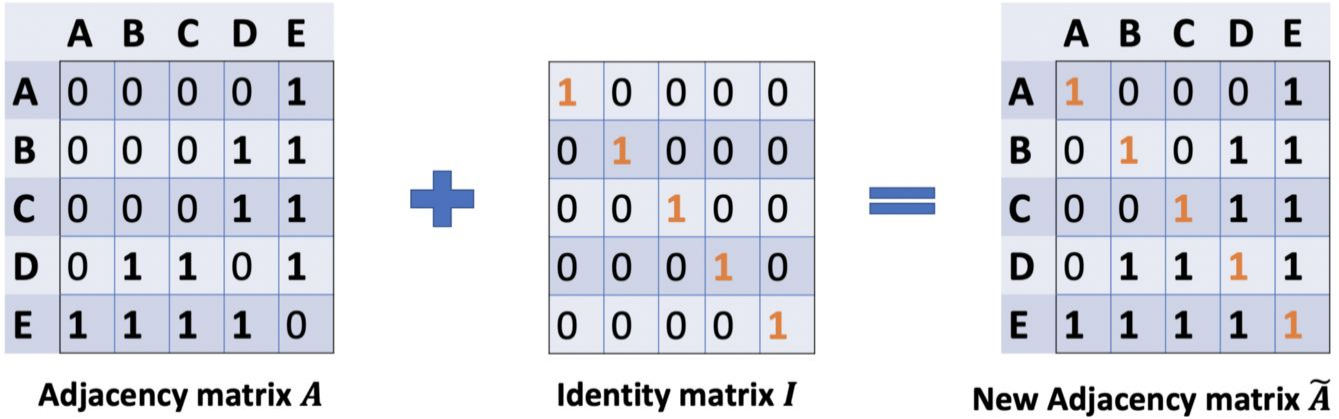
Problem 1: including feature of node itself

We can fix it by adding an identity matrix I to A to get a new adjacency matrix Ã.

$$\tilde{A} = A + \lambda I_N$$

Pick lambda = 1 (the feature of the node itself is just important as its neighbors), we have Ã = A + I. Note that we can treat lambda as a trainable parameter, but for now, just assign the lambda to 1 (in the original paper, lambda is just simply assigned to 1 as well).

Problem 1: including feature of node itself



Problem 2: how do we pass the information from neighbors to a specific node?

- Let D̃ be the degree matrix of our new adjacency matrix Ã
- How do we use D̃ to get the average of all the neighbor's features?

	A	B	C	D	E
A	1	0	0	0	1
B	0	1	0	1	1
C	0	0	1	1	1
D	0	1	1	1	1
E	1	1	1	1	1

New adjacency matrix $\tilde{A}$

2	0	0	0	0
0	3	0	0	0
0	0	3	0	0
0	0	0	4	0
0	0	0	0	5

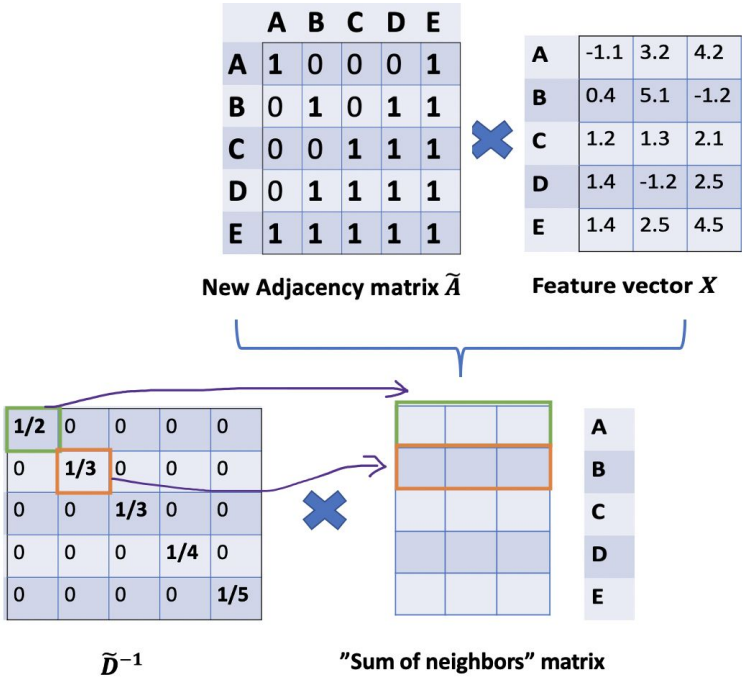
New degree matrix $\tilde{D}$

1/2	0	0	0	0
0	1/3	0	0	0
0	0	1/3	0	0
0	0	0	1/4	0
0	0	0	0	1/5

$\tilde{D}^{-1}$

Problem 2: how do we pass the information from neighbors to a specific node? (aggregation)

- Average of all neighbors' feature vectors = multiply D̃ inverse with the feature vector X

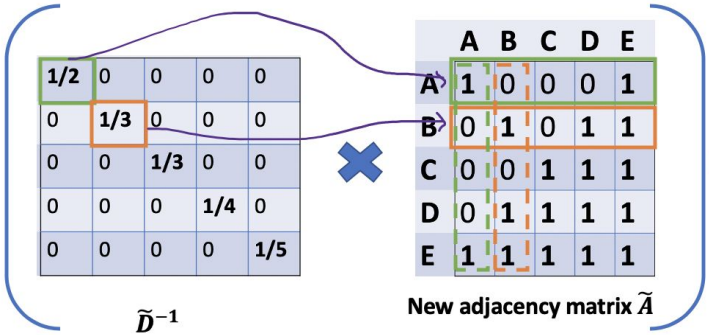


Problem 2: how do we pass the information from neighbors to a specific node? (aggregation)

What about the *weighted* average?

Let's take a deeper look at the average() approach that we've just mentioned. From the *Associative property of matrix multiplication*, for any three matrices A, B, and C, (A*B*)*C*=*A*(*BC*). Rather $\tilde{D}^{-1}(\tilde{A}X)$, we consider $(\tilde{D}^{-1}\tilde{A})X$, so $\tilde{D}^{-1}$ **can also be seen as the scale factor of** $\tilde{A}$. From this perspective, each row i of $\tilde{A}$ will be scaled by $\tilde{D}_{ii}$ (Figure below). Note that $\tilde{A}$ is a symmetric matrix, it means row i is the same value as column i. If we scale each row i by $\tilde{D}_{ii}$, intuitively, we have a feeling that we should do the same for its corresponding column too.

Mathematically, we're scaling $\tilde{A}_{ij}$ only by $\tilde{D}_{ii}$. We're ignoring the j index. So, what would happen when we scale $\tilde{A}_{ij}$ by both $\tilde{D}_{ii}$ and $\tilde{D}_{jj}$?



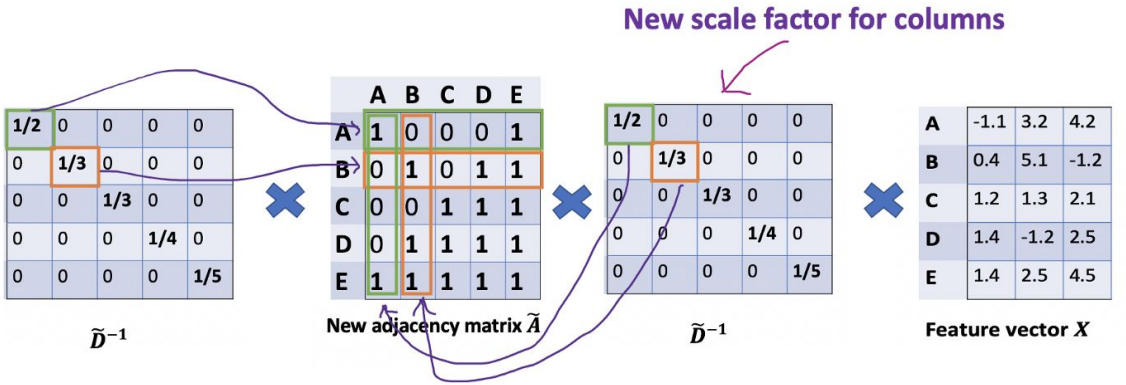
We are only scaling by rows here, and we are ignoring the corresponding columns denoted by the dash boxes

A	-1.1	3.2	4.2
B	0.4	5.1	-1.2
C	1.2	1.3	2.1
D	1.4	-1.2	2.5
E	1.4	2.5	4.5

Feature vector X

Problem 2: how do we pass the information from neighbors to a specific node? (aggregation)

We try new scaling strategy: instead of using $\tilde{D}^{-1}\tilde{A}X$, we use $\tilde{D}^{-1}\tilde{A}\tilde{D}^{-1}X$.

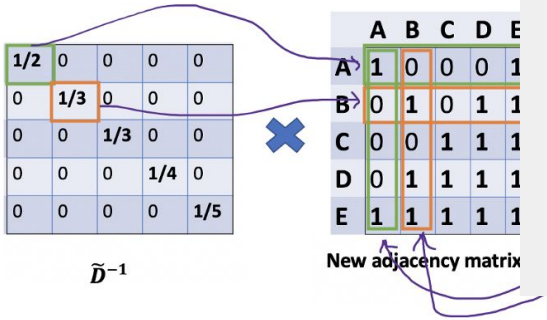


The new scaler gives us the “weighted” average. What we are doing here is to put more weights on the nodes that have low-degree and reduce the impact of high-degree nodes.

The idea of this weighted average is that we assume low-degree nodes would have bigger impacts on their neighbors, whereas high-degree nodes generate lower impacts as they scatter their influence at too many neighbors.

Problem 2: how do we pass the information from neighbors to a specific node? (aggregation)

We try new scaling strategy: instead of using $\tilde{D}^{-1}\tilde{A}X$, we use $\tilde{D}^{-1}\tilde{A}\tilde{D}^{-1}X$.

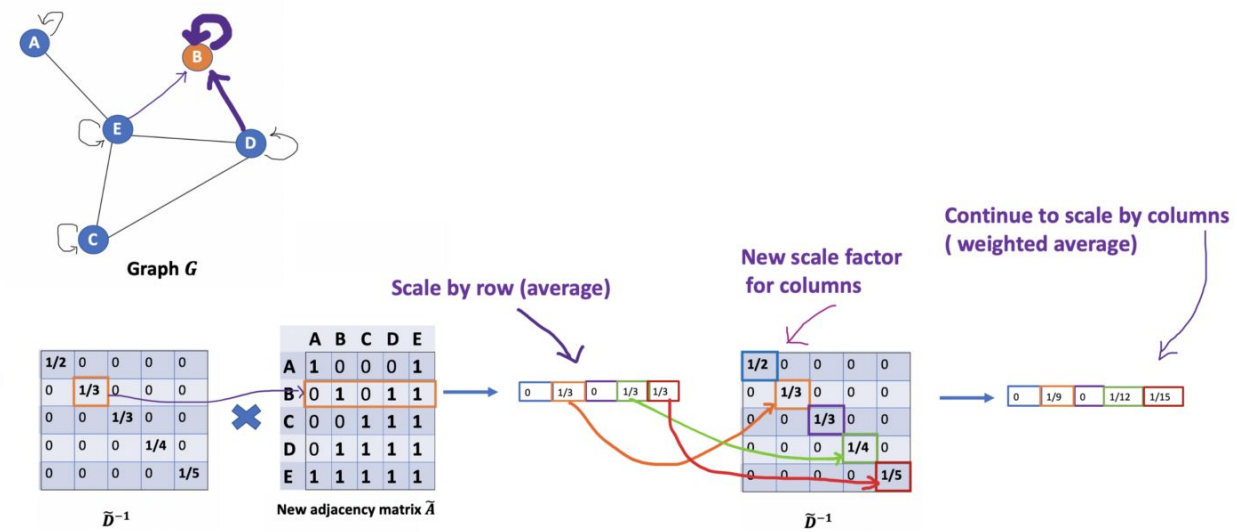


In the paper, they used $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-½}$. Why? We are normalizing twice with $\tilde{D}_{ii}$ and $\tilde{D}_{jj}$, so it makes sense to re-balance with $(\tilde{D}_{ii}\tilde{D}_{jj})^{-1/2}$

The new scaler gives us the “weighted” average. What we are doing here is to put more weights on the nodes that have low-degree and reduce the impact of high-degree nodes.

The idea of this weighted average is that we assume low-degree nodes would have bigger impacts on their neighbors, whereas high-degree nodes generate lower impacts as they scatter their influence at too many neighbors.

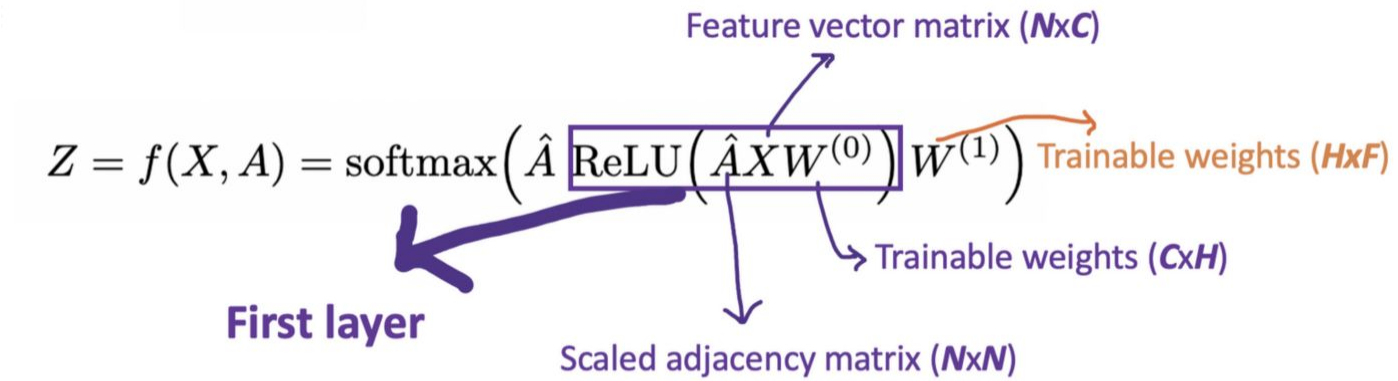
Problem 2: how do we pass the information from neighbors to a specific node? (aggregation)



When aggregating feature at node B, we assign the biggest weight for node B itself (degree of 3) and the lowest weight for node E (degree of 5)

Putting things together

Let's call $\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ just for a clear view. With 2-layer GCN, we have the form of our forward model as below.



Recall that ***N*** is #nodes, ***C*** is #dimensions of feature vectors. We also have ***H*** is #nodes in the hidden layer, and ***F*** is the dimensions of resulting vectors.

For example, we have a multi-classification problem with 10 classes, F will be set to 10. After having the 10-dimension vectors at layer 2, we pass these vectors through a softmax function for the prediction.

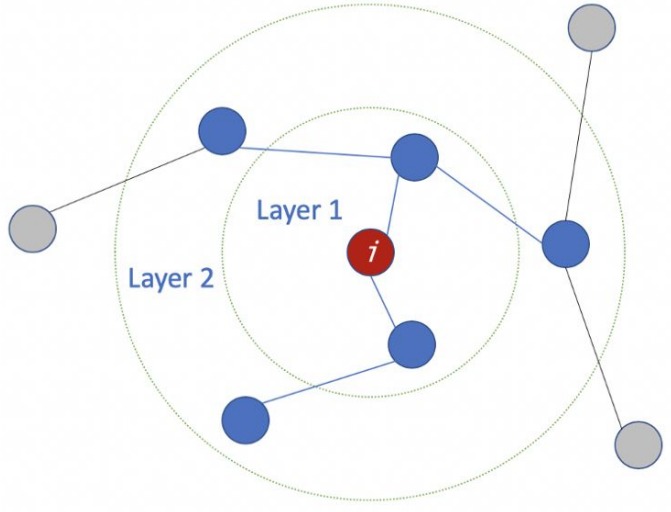
The loss function in node classification

The Loss function is simply calculated by the cross-entropy error over all labeled examples, where Y_L is the set of node indices that have labels.

$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}$$

GCN: number of layers

- The number of layers is the **farthest distance that node features can travel**
  - Example: in a 1 layer GCN, each node can only get the information from its neighbors. The gathering information process takes place independently, at the same time for all the nodes.
- When stacking another layer on top of the first one, we repeat the gathering info process, but this time, the neighbors already have information about their own neighbors (from the previous step).
  - It makes the number of layers as the maximum number of hops that each node can travel
  - Depending on how far we think a node should get information from the networks, we can config a proper number of layers
  - Normally we don't want to go too far: with 6–7 hops, we almost get the entire graph, which makes the aggregation less meaningful



Example: Gathering info process with 2 layers of target node i

So… how many layers?

In the paper, the authors also conducted some experiments with shallow and deep GCNs. From the figure below, we see that the best results are obtained with a 2- or 3-layer model. Besides, with a deep GCN (more than 7 layers), it tends to get bad performances (dashed blue line). One solution is to use the residual connections between hidden layers (purple line).