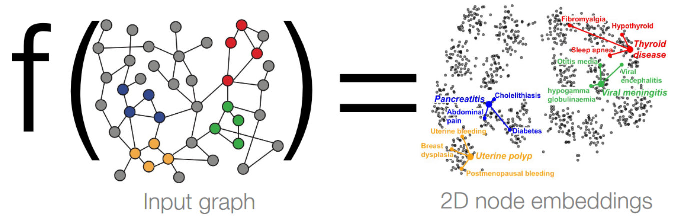
Overview

1. Basics of deep learning
2. Deep learning for graphs
3. Graph Convolutional Networks
4. GNNs subsume CNNs and Transformers

Recap

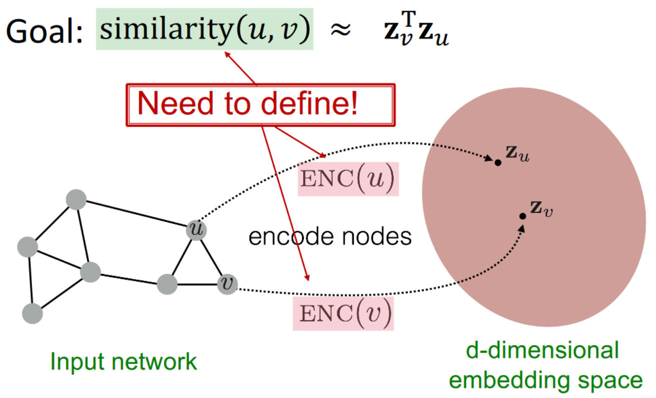
Node embedding

Intuition. Map nodes to d-dimensional embeddings such that similar nodes in the graph are embedded close together.



Question. How to learn mapping function $\boldsymbol{f}$?

Goal.



Two Key Components. Encoder & Similarity function (decoder)

- **Encoder:** Maps each node to a low-dimensional vector

  $\mathrm{ENC}(v) = \mathbf{z}_v$ — d-dimensional embedding; v: node in the input graph

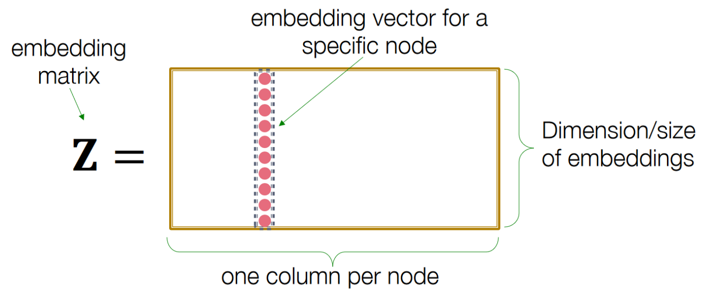
- **Similarity function:** Specifies how the relationships in vector space map to the relationships in the original network

  similarity$(u, v) \approx \mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u$ **Decoder**

  Similarity of u and v in the original network; dot product between node embeddings

"Shallow" Encoding. Encoder is just an embedding-lookup.

- shallow since just memorising embedding of every node



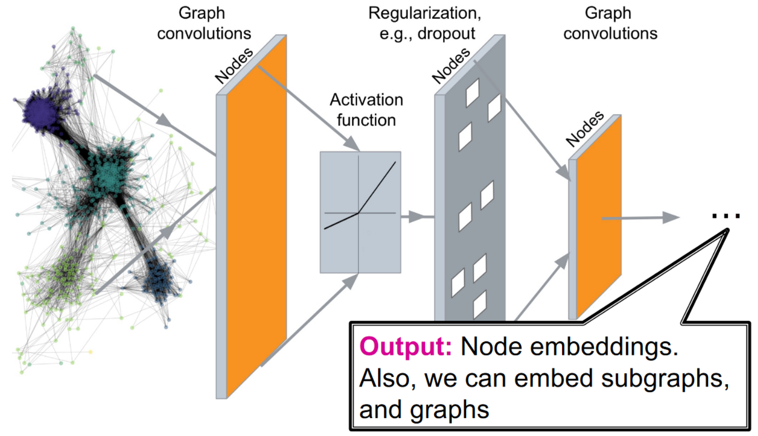
Shallow Encoders.

- $\boldsymbol{O}(|\boldsymbol{V}|)$ **parameters are needed** (each column is the embedding for a node)
  - **No sharing** of **parameters** between nodes
- Inherently **"transductive"**
  - Cannot generate embeddings for nodes that are **not seen** during training
- Do not incorporate **node features**

GNN, Deep Graph Encoders

- ENC(v) = multiple layers of non-linear transformations based on graph structure
  - can be **combined** with node similarity functions

End to end. labels on the right to the graph structure on the left

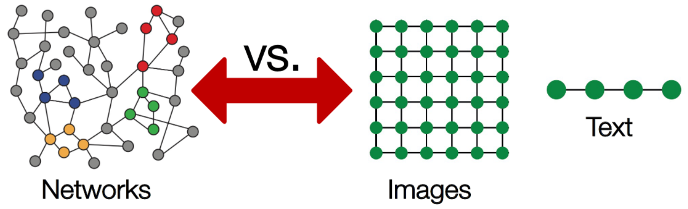


Tasks

- 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
- Network similarity
  - How similar are two (sub)networks

Graph vs Image

- **Arbitrary** size
- complex **topological structure** (i.e., no spatial locality like grids)
- No fixed node ordering or reference point
- Often dynamic and have **multimodal features**



Basics of Deep Learning

Basics of Deep Learning
See slides.

- Iteration: 1 step of SGD on a minibatch (1 step 1 ~ 1 mini-match)
- Epoch: one full pass over the dataset (# iterations is equal to **ratio** of dataset size and batch size)
- SGD is **unbiased** estimator of full gradient

Lec 6.1 – 6.2 Deep learning Basic

Objective Function

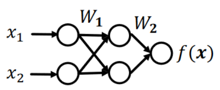
- **Objective:** $\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x}))$
- In deep learning, function f can be very complex
- **Example:**
  - To start simple, consider linear function
    $$f(\boldsymbol{x}) = W \cdot \boldsymbol{x}, \qquad \Theta = \{W\}$$
  - Then, if f returns a scalar, then W is a learnable **vector**
    $$\nabla_W f = \left(\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \frac{\partial f}{\partial w_3} \ldots\right)$$
  - But, if f returns a vector, then W is the **weight matrix**
    $$\nabla_W f = \begin{bmatrix} \frac{\partial f_1}{\partial w_{11}} & \frac{\partial f_2}{\partial w_{12}} \\ \frac{\partial f_1}{\partial w_{21}} & \frac{\partial f_2}{\partial w_{22}} \end{bmatrix}$$
    Jacobian matrix of f

More complex – use chain rule

- **How about a more complex function:**
  $$f(\boldsymbol{x}) = W_2(W_1\boldsymbol{x}), \Theta = \{W_1, W_2\}$$
  In other words:
  $f(\boldsymbol{x}) = W_2(W_1\boldsymbol{x})$
  $h(x) = W_1\boldsymbol{x}$
  $g(z) = W_2 z$
- Recall **chain rule**:
  $$\frac{df}{dx} = \frac{dg}{dh} \cdot \frac{dh}{dx} \text{ or } f'(x) = g'(h(x))h'(x)$$
- **Example:** $\nabla_{\boldsymbol{x}} f = \dfrac{\partial f}{\partial (W_1\boldsymbol{x})} \cdot \dfrac{\partial (W_1\boldsymbol{x})}{\partial \boldsymbol{x}}$
- **Back-propagation**: Use of **chain rule** to propagate gradients of intermediate steps, and finally obtain gradient of $\mathcal{L}$ w.r.t. Θ.

Example: Simple 2-layer linear network

- $f(\boldsymbol{x}) = g(h(x)) = W_2(W_1\boldsymbol{x})$



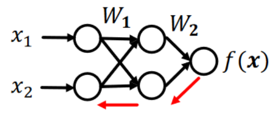
- $\mathcal{L} = \sum_{(x,y)\in\mathcal{B}} \left\| (\boldsymbol{y}, -f(\boldsymbol{x})) \right\|_2$
  - The loss $\mathcal{L}$ sums the L2 loss in a minibatch $\mathcal{B}$.
- **Hidden layer:**
  - Intermediate representation of input $\boldsymbol{x}$
  - Here we use $h(x) = W_1\boldsymbol{x}$ to denote the hidden layer
  - $f(\boldsymbol{x}) = W_2 h(x)$

Backprop

- The w.r.t. is bacdward i.e. the opposite direction to the forward pass

Remember:
$f(\boldsymbol{x}) = W_2(W_1\boldsymbol{x})$
$h(x) = W_1\boldsymbol{x}$
$g(z) = W_2 z$

- **Forward propagation:**
  Compute loss starting from input
  - $\boldsymbol{x} \xrightarrow{\text{Multiply } W_1} h \xrightarrow{\text{Multiply } W_2} g \xrightarrow{\text{Loss}} \mathcal{L}$

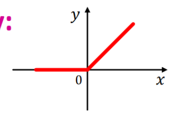


- **Back-propagation to compute gradient of**
  $$\Theta = \{W_1, W_2\}$$
  Start from loss, compute the gradient
  $$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial f} \cdot \frac{\partial f}{\partial W_2}, \qquad \frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial f} \cdot \frac{\partial f}{\partial W_2} \cdot \frac{\partial W_2}{\partial W_1}$$
  Compute backwards Compute backwards

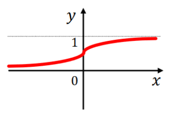
Non-linearity

- Improve the expressiveness of the model

  - Note that in $f(\boldsymbol{x}) = W_2(W_1\boldsymbol{x})$, $W_2 W_1$ is another matrix (vector, if we do binary classification)
    - Hence $f(\boldsymbol{x})$ is still linear w.r.t. $\boldsymbol{x}$ no matter how many weight matrices we compose
  - **We introduce non-linearity:**
    - **Rectified linear unit (ReLU)**
      $ReLU(x) = \max(x, 0)$

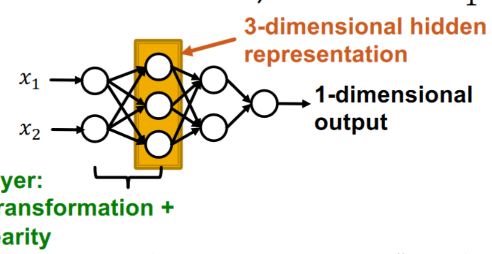


    - **Sigmoid**
      $$\sigma(x) = \frac{1}{1 + e^{-x}}$$



MLP – combine linear transformation & non-linearity

- **Each layer of MLP combines linear transformation and non-linearity:**
  $$\boldsymbol{x}^{(l+1)} = \sigma(W_l \boldsymbol{x}^{(l)} + b^l)$$
  - where W_l is weight matrix that transforms hidden representation at layer l to layer $l+1$
  - b^l is bias at layer l, and is added to the linear transformation of $\boldsymbol{x}$
  - σ is non-linearity function (e.g., sigmod)
- Suppose $\boldsymbol{x}$ is 2-dimensional, with entries x_1 and x_2



Summary

- **Objective function:**
  $$\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x}))$$
- f can be a simple linear layer, an MLP, or other neural networks (e.g., a GNN later)
- Sample a minibatch of input $\boldsymbol{x}$
- **Forward propagation:** Compute $\mathcal{L}$ given $\boldsymbol{x}$
- **Back-propagation:** Obtain gradient $\nabla_W \mathcal{L}$ using a chain rule.
- Use **stochastic gradient descent (SGD)** to optimize for Θ over many iterations.