▶ Lecture 21 EN.553.744 Luana Ruiz

→ graph generation

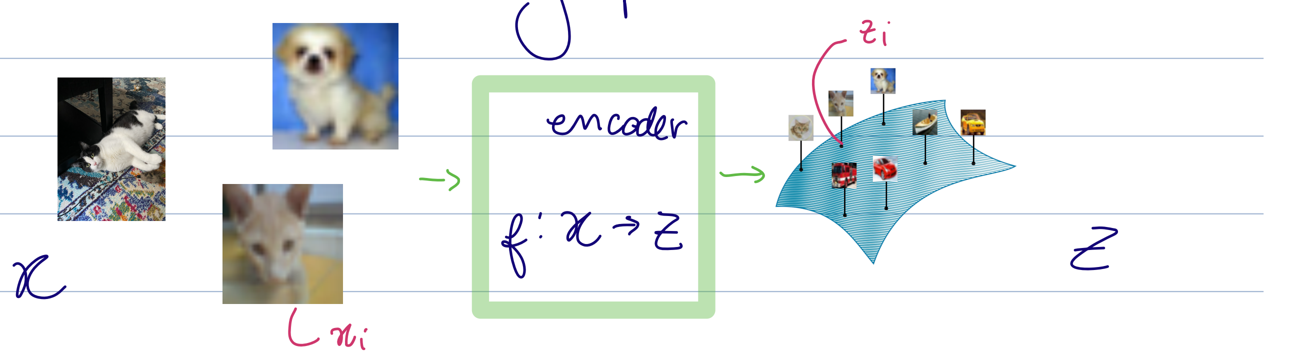
- Applications : - drug discovery
  - materials design
  - social network modeling
  - etc...

"General" generative modeling typically consists of two components:

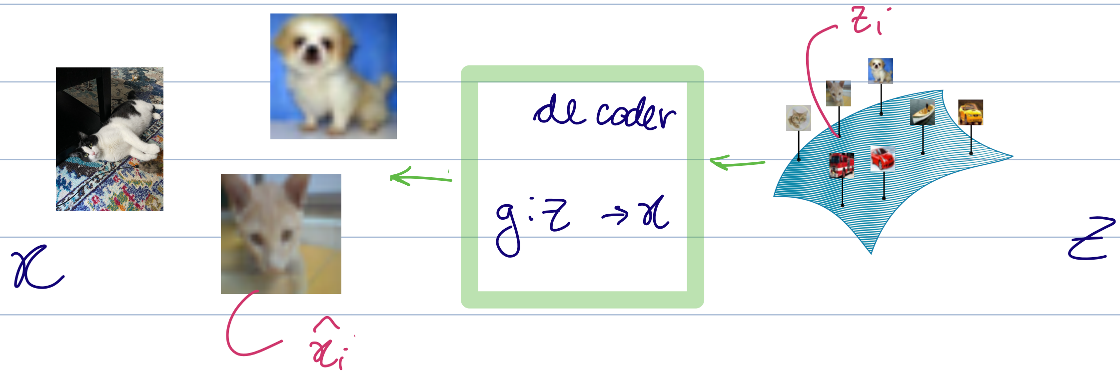
- an encoder
- a decoder

e.g.: AEs, VAEs, GANs, diffusion...

- The encoder's job is to map the data to some (low) dimensional embedding space.



- The decoder's job is to map from embedding space back to ambient space.



- Both are trained simultaneously; the goal is for the decoder outputs to match the encoder inputs:

$$\min_{f,g} \ell(x_i, \hat{x}_i)$$

same loss

- At generation time, we generate samples by

sample $z' \sim \mathcal{Z}$ → decoder $g: \mathcal{Z} \to \mathcal{X}$ → unseen x'!

$\sim p(x' | f, g)$

↳ Ultimately, we want $p(x'|f,g)$ — the modeled distribution — to be close to $p(x)$ — the data distribution.

Idea: maximize likelihood of f,g for samples $x \sim p(x)$

$$\boxed{\max_{f,g} \mathbb{E}_{x \sim p(x)} \log p(x|f,g)}$$

↳ Most commonly $z \sim \mathcal{N}(0,1)$ in embedding space

Back to graphs:

- encoder: GNNs! Map graphs G to tensors in $\mathbb{R}^{n \times d}$ or $\mathbb{R}^d$

↳ Many options:

- convolutional: GNN, GCN, SAGE, etc.
- spectral: Chebnet, spectral GNN
- WL-inspired: GIN & higher order GINs
- attention-based: graphormer & variants
- graphormer, etc...

- challenge is the decoding step:

Fixing a size limit N, graphs can have as many as N nodes and N^2 edges!

impractical even for moderate n

↳ graph VAEs, for instance, have only been applied successfully to small graph generation (e.g. molecules)

Possible solution: move away from end to end simultaneous prediction (of $N + N^2$ binary variables) and toward sequential / autoregressive prediction:

$$p(x \mid \theta) = \prod_{t=1}^{T} p(x_t \mid x_1, \dots, x_{t-1}; \theta)$$

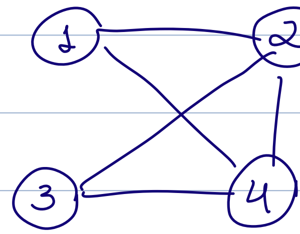
E.g.: x is a sentence; x_{t-1} is the current word, x_t the next

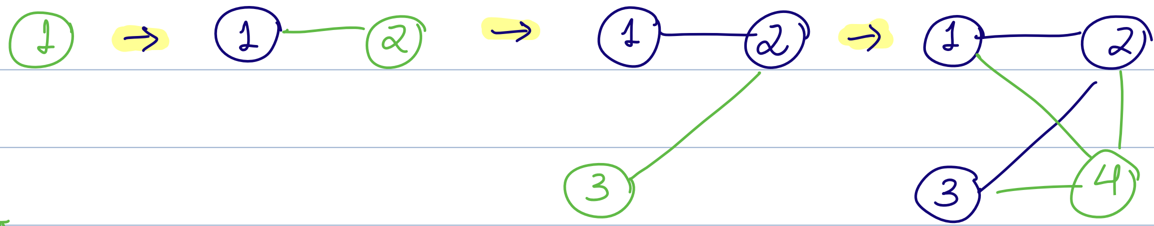
Here: X is a graph; x_t is the next action (add node or add edge)

→ GraphRNNs (You, Ying, Ren, Hamilton, Leskovec, 2018)

Given a fixed labeling $\pi = \{1, 2, \dots, n\}$, a graph can be mapped one-to-one to a sequence of node and edge additions.

E.g.:

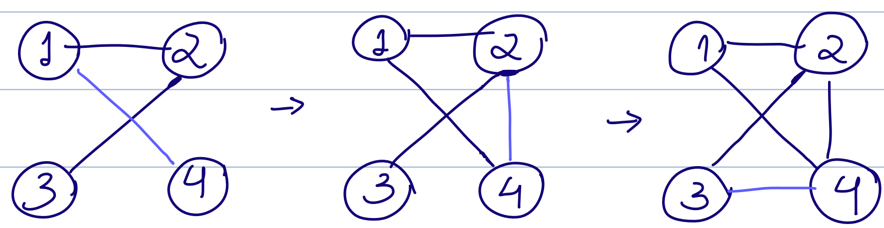




$S^\pi = \{ S_1^\pi \; , \; S_2^\pi \; , \; S_3^\pi \; , \; S_4^\pi \}$

$\downarrow$ add node 1 $\quad$ $\downarrow$ add node 2 $\quad$ $\downarrow$ add node 3 $\quad$ $\downarrow$ add node 4

$S_4^\pi = \Big\{$



$\Big\}$

$S_{4,1}^\pi \qquad S_{4,2}^\pi \qquad S_{4,3}^\pi$

The sequence S^π has two levels: node additions; edge additions per node addition

E.g. 2: Adjacency matrix perspective:

$$
\begin{array}{cccc}
1 & 2 & 3 & 4
\end{array}
$$

$$
\begin{bmatrix}
0 & 1 & 0 & 1 \\
 & 0 & 1 & 1 \\
 & & 0 & 1 \\
 & & & 0
\end{bmatrix}
\rightarrow \text{symmetrize}
$$

$\longrightarrow$ node additions

process 1

$$
\begin{array}{cccc}
1 & 2 & 3 & 4
\end{array}
$$

$$
\begin{bmatrix}
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 \\
1 & 1 & 1 & 0
\end{bmatrix}
$$

$\downarrow$ edge additions

process 2

We'll use recurrent neural nets (RNNs) to model each of these processes

▷ RNNs:

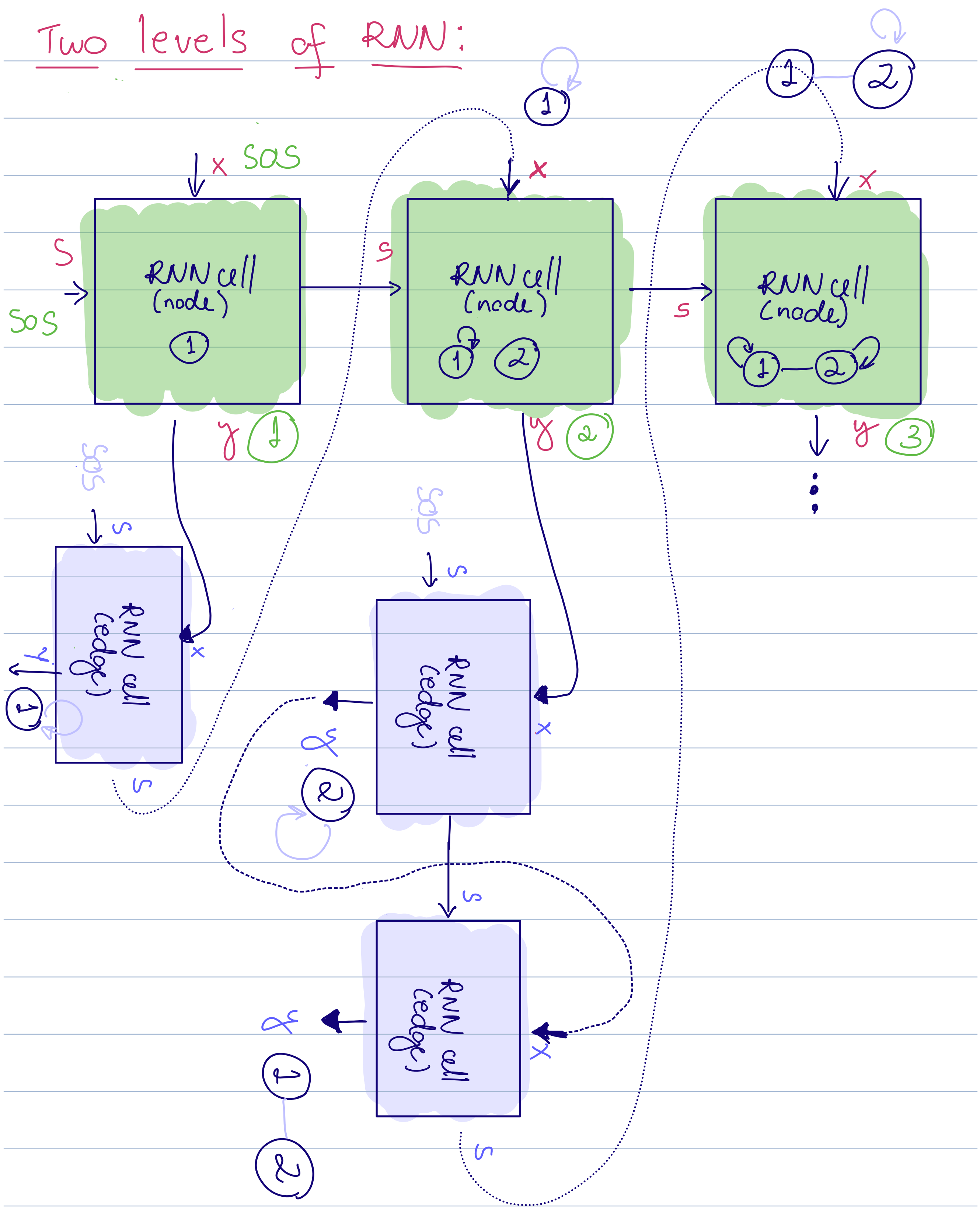
An RNN cell does the following operations:

"hidden state" encodes "memory" about process

$$h_t = \sigma(A x_t + B h_{t-1})$$

$$y_t = \rho(C \cdot h_t)$$

Two levels of RNN:



SOS: start of seq.   EOS: end of seq.

- Sequence ends when edge-level RNN outputs EOS token

↳ Model works, but it is deterministic

We need $y = p(x_{t+1} | x_1, \dots, x_t; \theta)$

Explicitly, instead of reusing y_{t-1} (output of edge RNN at $t-1$) as x_t (input of edge RNN at t), we will do:

$$x_t \sim y_{t-1}$$

↳ seen as the probability of sampling the edge

- We understand inference (sampling), but how to train? Teacher forcing.

At training time, we know all $y_1, \dots, y_T$

$x_0 = \text{SOS} \rightarrow$ model $\rightarrow \hat{y}_1 = p(x_1 | x_0; \theta) = y_1$?

$x_1 = y_1 \longrightarrow$ model $\longrightarrow \hat{y}_2 = p(x_2 | x_0, x_1; \theta) = y_2$?

$x_2 = y_2 \longrightarrow$ model $\longrightarrow \hat{y}_3 = p(x_3 | x_0 \dots x_2; \theta) = y_3$?

$\vdots$

comparison (and optimization) done by computing the binary cross entropy (BCE) loss

$$\ell(\hat{y}, y) = -\left(y \log(\hat{y}) + (1-y)\log(1-\hat{y})\right)$$

→ Transformers & graph transformers

Consider a sequence $x_1, x_2, \dots, x_T$

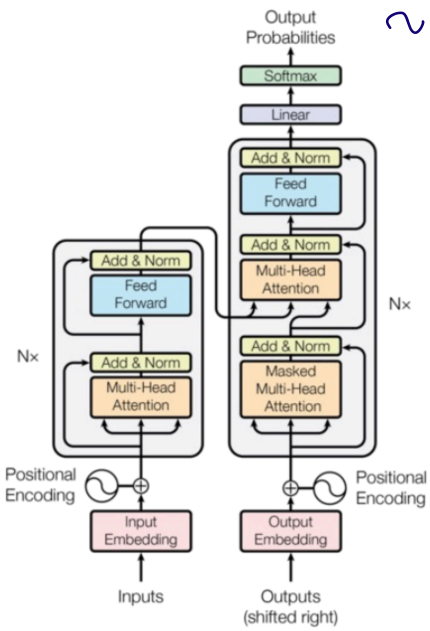
For each token x_i, the attention layer of a transformer layer does:

$q_i = MLP_q(x_i)$; $k_i = MLP_k(x_i)$; $v_i = MLP_v(x_i)$

and outputs $y_i = \sum_{j \in [T]} \langle q_i, k_j \rangle v_j$

($\langle \cdot, \cdot \rangle$ involves a normalization by softmax)

Obs.: The attention layer is a 1-hop convolution (or message-passing layer) over a complete graph with learned edge weights $\langle q_i, k_j \rangle$.



~ output

(at inference time)

E.g. (train):

In: "Translate EN to PT: The cat is black."

Out: "O gato é preto."

Loss computation:

1. Translate EN to PT: The cat is black. → $\hat{y}_1 = O?$

2. Translate EN to PT: The cat is black. O → $\hat{y}_2 = gato?$
y_1

3. Translate EN to PT: The cat is black. O gato → $\hat{y}_3 = é?$
y_1 y_2

E.g. (test):

In: "Translate PT to EN: O gato é preto."
Out: ?

1. Translate PT to EN: O gato é preto → $\hat{y}_1$ = The

2. Translate PT to EN: O gato é preto. The → $\hat{y}_2$ = cat
$\hat{y}_1$

. . .

Obs.: Note that in the transformer, order does not matter, while in language, it typically does. Sometimes, the tokens $x_1, x_2, \ldots, x_T$ are "augmented" with positional encodings, e.g. $[x_1|1], [x_2|2], \ldots, [x_T|T]$.

➔ Adapting transformers to graphs

- Node features $x_1, x_2, \ldots, x_n$ become tokens
- Adjacency information can be incorporated in two ways:

↳ Attention mask: we might only allow $\langle k_i, q_j \rangle \neq 0$ if $(i,j) \in \mathcal{E}$.

this is the graph attention (GAT)

Or, we might compute

$$\begin{cases} w_{ij} = [\langle k_i, q_j \rangle \mid A_{ij}] \\ v_i = \sum_{j \in [n]} w_{ij} v_j \end{cases}$$

Positional encodings: to the node features $x_1, x_2, \ldots, x_n$, we might want to concatenate P.Es $[x_1 | p_1], [x_2 | p_2], \ldots [x_n | p_n]$

Typical choices are:

- degree, $p_i = d_i = [A\mathbb{1}]_i$
- eigenvectors, $p_i = [[v_1]_i [v_2]_i \ldots [v_k]_i]$
- k-length random walk, PageRank, etc...

THANK YOU! ☺

I hope your summer is as restful as Dandan's whole existence!