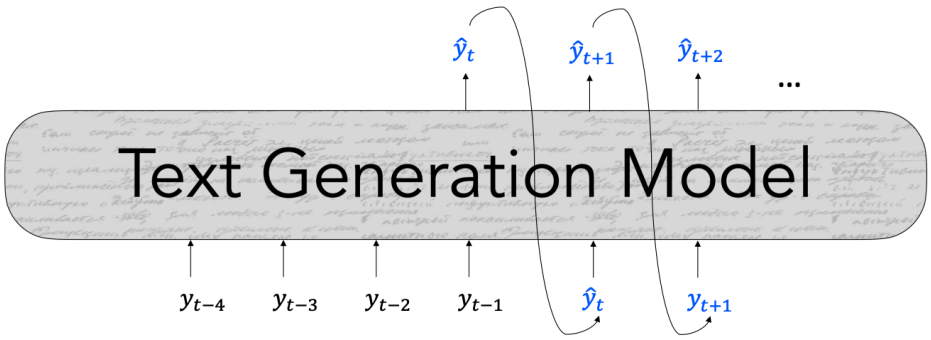
NLP_WEEK12

Lecture 12 - Natural Language Generation

What is natural language generation?
Focused on building systems that automatically produce coherent and useful written or spoken text for human consumption
Any task involving text production for human consumption requires natural language generation

Basics of natural language generation



- At each time step t, our model computes a vector of scores for each token in our vocabulary, $S \in \mathbb{R}^V$:

$$S = f(\{y_{<t}\}, \theta)$$

$f(.)$ is your model

- Then, we compute a probability distribution P over $w \in V$ using these scores:

$$P(y_t = w|\{y_{<t}\}) = \frac{\exp(S_w)}{\sum_{w' \in V} \exp(S_{w'})}$$

- At inference time, our decoding algorithm defines a function to select a token from this distribution:

$$\hat{y}_t = g(P(y_t|\{y_{<t}\}))$$

$g(.)$ is your decoding algorithm

- We train the model to minimize the negative loglikelihood of predicting the next token in the sequence:

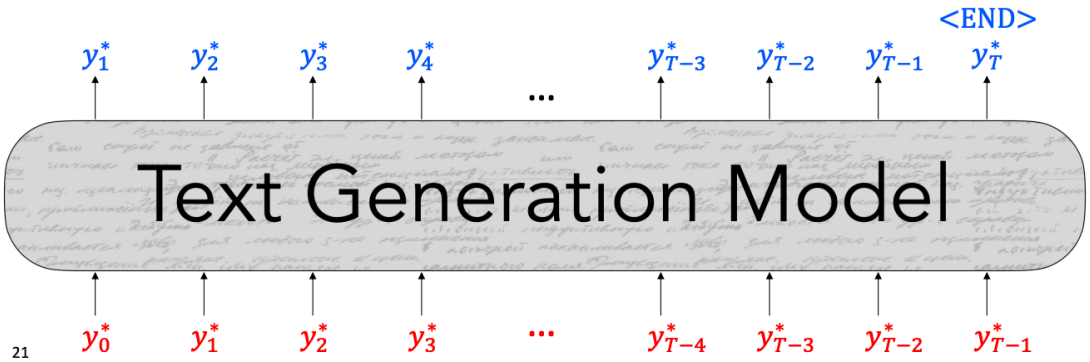
$$\mathcal{L}_t = -\log P(y_t^*|\{y_{<t}^*\})$$

Sum $\mathcal{L}_t$ for the entire sequence

Maximum Likelihood Training (i.e., teacher forcing)

- Trained to generate the next word y_t^* given a set of preceding words $\{y^*\}_{<t}$

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(y_t^* | \{y^*\}_{<t})$$

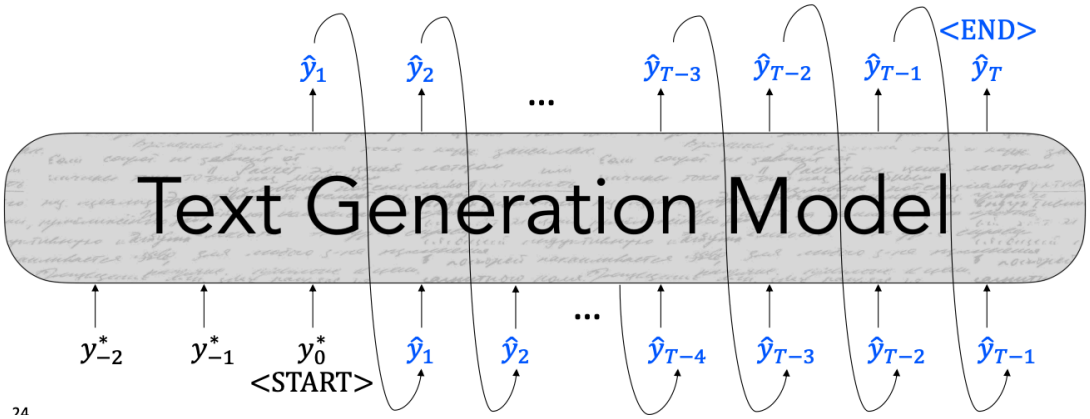


21

Decoding: what is it all about?

- Our decoding algorithm defines a function to select a token from this distribution

$$\hat{y}_t = g(P(y_t | \{y^*\}, \{\hat{y}\}_{<t}))$$



24

Greedy methods

- **Recall**: Lecture 7 on Neural Machine Translation…
- Argmax Decoding
  - Selects the highest probability token in $P(y_t | y_{<t})$

$$\hat{y}_t = \underset{w \in V}{\text{argmax}}\, P(y_t = w | y_{<t})$$

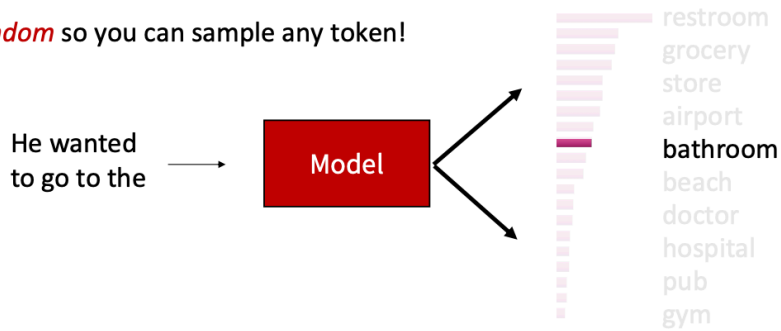
- Beam Search
  - Discussed in Lecture 7 on Machine Translation
  - Also a greedy algorithm, but with wider search over candidates

Time to get random : Sampling!

- Sample a token from the distribution of tokens

$$\hat{y}_t \sim P(y_t = w \mid \{y\}_{<t})$$

- It's *random* so you can sample any token!



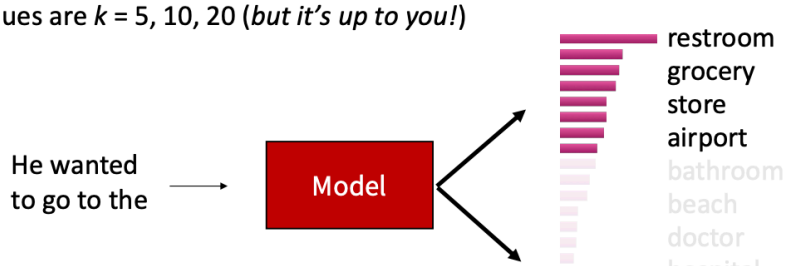
Decoding: Top-k sampling

• Problem: Vanilla sampling makes every token in the vocabulary an option

• Solution: Top-k sampling

Decoding: Top-k sampling

- Solution: Top-*k* sampling
  - Only sample from the top *k* tokens in the probability distribution
  - Common values are *k* = 5, 10, 20 (*but it's up to you!*)



  - Increase *k* for more **diverse**/**risky** outputs
  - Decrease *k* for more **generic**/**safe** outputs

Decoding: Top-p (nucleus) sampling

• Problem: The probability distributions we sample from are dynamic

• Solution: Top-p sampling

- Solution: Top-p sampling
  - Sample from all tokens in the top p cumulative probability mass (i.e., where mass is concentrated)
  - Varies k depending on the uniformity of P_t

$P_t^1(y_t = w \mid \{y\}_{<t})$ $P_t^2(y_t = w \mid \{y\}_{<t})$ $P_t^3(y_t = w \mid \{y\}_{<t})$

Scaling randomness: Softmax temperature

- Recall: On timestep t, the model computes a prob distribution P_t by applying the softmax function to a vector of scores $s \in \mathbb{R}^{|V|}$

$$P_t(y_t = w) = \frac{\exp(S_w)}{\sum_{w' \in V} \exp(S_{w'})}$$

- You can apply a *temperature hyperparameter* τ to the softmax to rebalance P_t:

$$P_t(y_t = w) = \frac{\exp(S_w/\tau)}{\sum_{w' \in V} \exp(S_{w'}/\tau)}$$

- Raise the temperature $\tau > 1$: P_t becomes more uniform
  - **More** diverse output (probability is spread around vocab)
- Lower the temperature $\tau < 1$: P_t becomes more spiky
  - **Less** diverse output (probability is concentrated on top words)

Note: softmax temperature is not a decoding algorithm!

It's a technique you can apply at test time, in conjunction with a decoding algorithm (such as beam search or sampling)