Lecture 12: Neural Language Generation

- Natural language generation (NLG)
  - NLG is a sub-field of natural language processing
  - Focused on building systems that automatically produce coherent and useful written or spoken text for human consumption
  - NLG systems are already changing the world we live in

  -Any task involving text production for human consumption requires nautral language generation.
- Basics of natural language generation

  In autoregressive text generation models, at each time step t, our model takes in a sequence of tokens of text as input{y}<t and outputs a new token yt

- **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:**

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

- **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

  - Note: This is just a classification task where each $w \in V$ is a class.
  - The label at each step is the actual word y_t^* in the training sequence
  - This token is often called the "gold" or "ground truth" token
  - This algorithm is often called "teacher forcing"

16

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

$$\mathcal{L} = -\log P(y_1^* | y_0^*)$$

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

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

$f(.)$ is your model

  - Then, we compute a probability distribution P over these scores (usually with a softmax function):

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

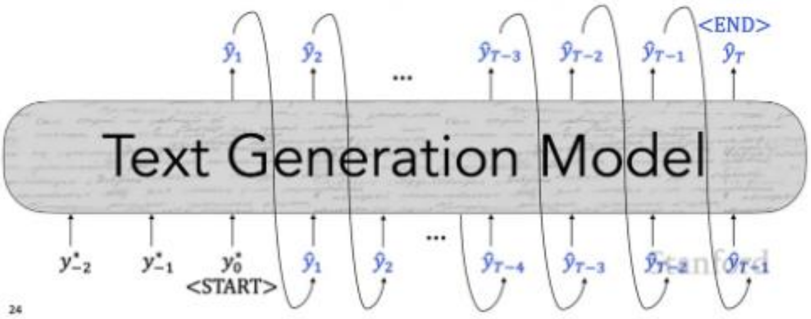
  - Our decoding algorithm defines a function to select a token from this distribution:

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

$g(.)$ is your decoding algorithm

  - 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 $P(y_t|y_{<t})$

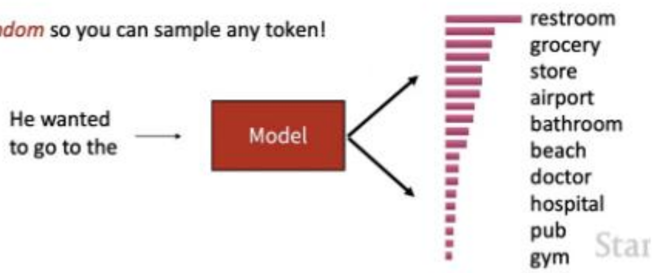
$$\hat{y}_t = \underset{w \in V}{\mathbf{argmax}}\, P(y_t = w|y_{<t})$$

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

- Sampling
  - Sample a token from the distribution of tokens

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

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



- Decoding: Top-k sampling

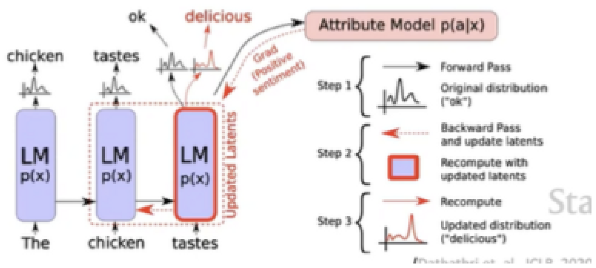
  Problem: Vanilla sampling makes every token in the vocabulary an option

  - Even if most of the probability mass in the distribution is over a limited set of options, the tail of the distribution could be very long
  - Many tokens are probably irrelevant in the current context
  - Why are we giving them individually a tiny chance to be selected
  - Why are we giving them as a group a high chance to be selected

  Solution: Top-k sampling

  - Only sample from the top k tokens in the probability distribution
  - Increase k for more diverse/risky outputs, Decrease k for more generic/safe outputs

- Backpropagation-based distribution re-balancing
  - Can I re-balance my language model's distribution in to encourage other behaviors?
  - Yes. Just define a model that evaluates that behavior.



- Improving Decoding: Re-ranking

  Problem: What if I decode a bad sequence from my model

  Decode a bunch of sequences

  Define a score to approximate quality of sequences and re-rank by this score

- Decoding: Takeaways
  - Decoding is still a challenging problem in natural language generation
  - Human language distribution is noisy and doesn't reflect simple properties
  - Different decoding algorithms can allow us to inject biases that encourage different properties of coherent natural language generation
  - Some of the most impactful advances in NLG of the last few years have come from simple, but effective, modifications to decoding algorithms

- Unlikelihood Training
  - **Given a set of undesired tokens $\mathcal{C}$, lower their likelihood in context**

    $$\mathcal{L}_{UL}^{t} = -\sum_{y_{neg} \in \mathcal{C}} \log(1 - P(y_{neg} \mid \{y^*\}_{<t}))$$

  - **Keep *teacher forcing* objective and combine them for final loss function**

    $$\mathcal{L}_{MLE}^{t} = -\log P(y_t^* \mid \{y^*\}_{<t}) \qquad \mathcal{L}_{ULE}^{t} = \mathcal{L}_{MLE}^{t} + \alpha \mathcal{L}_{UL}^{t}$$

  - **Set $\mathcal{C} = \{y^*\}_{<t}$ and you'll train the model to lower the likelihood of previously-seen tokens!**

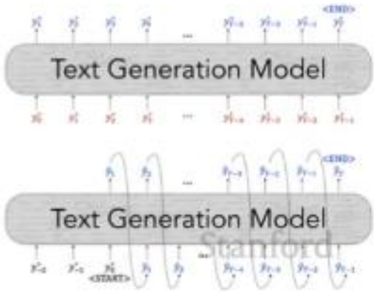
- Exposure Bias

  - Training with teacher forcing leads to *exposure bias* at generation time
    - During training, our model's inputs are gold context tokens from real, human-generated texts

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

    - At generation time, our model's inputs are previously–decoded tokens

      $$\mathcal{L}_{dec} = -\log P(\hat{y}_t | \{\hat{y}\}_{<t})$$



51

  - solution: Sequence re-writing

- REINFORCE: Basics

$$\mathcal{L}_{RL} = -\sum_{t=1}^{T} r(\hat{y}_t) \log P(\hat{y}_t | \{y^*\}; \{\hat{y}_t\}_{<t})$$