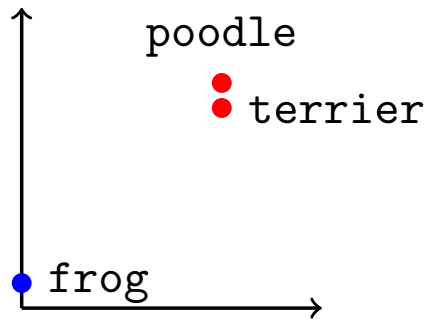
Lectures on Natural Language Processing

6. Language Models

Karl Stratos

Review: Word Embeddings



$$||\texttt{poodle} - \texttt{terrier}|| \ll ||\texttt{poodle} - \texttt{frog}||$$

Text = Sequence of learnable word embeddings $e_{x_1} \dots e_{x_T} \in \mathbb{R}^d$

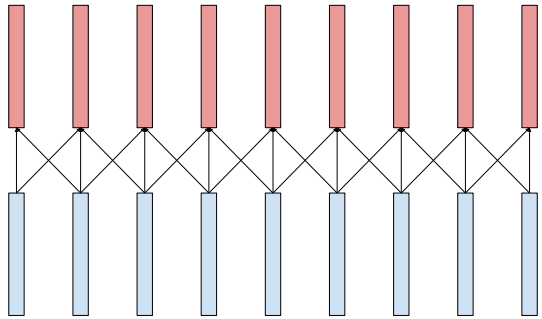
- Can be **contextualized** with additional (e.g., convolutional, recurrent, attention-based) transformations!

Various ways to get a final single-vector representation of text

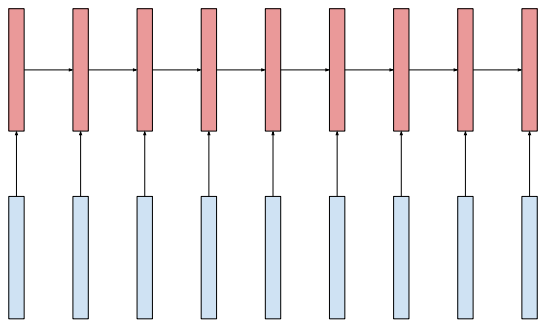
- **Mean pooling.** Average embeddings ("continuous bag-of-words")
- **Max pooling.** Take elementwise maximum (e.g., in CNNs)
- **Single token.** Take the embedding corresponding to a single token, assuming it's already function of all inputs (e.g., "[CLS] token")

Review: Contextualizing Word Embeddings

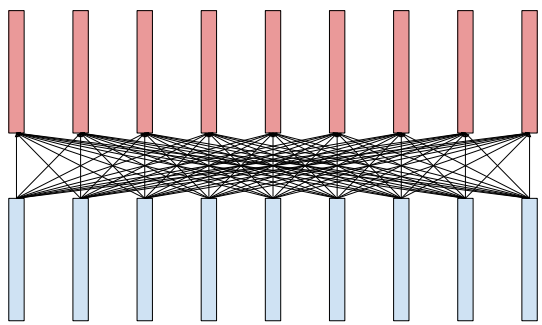
- **Convolutional**: Slide n-gram filters. Can be stacked.



- **Recurrent**: Maintain a recurrent state. Can be stacked and bidirectional.



- **Self-attention**: Compute a convex combination of all inputs. Can be stacked.



Review: Learnable Parameters

- **CNNs**: Filter matrix $U \in \mathbb{R}^{n \times d}$ where n is the width of the filter's n-gram. Can't model word ordering beyond filter sizes. Stacking can enlarge the window size.
- **RNNs**: Parameters used to update state $(h, x) \mapsto h'$ ("cell"). Variants to address gradient problems (e.g., LSTMs). Can model arbitrarily long sequences, but cannot be parallelized and one-sided (or shallowly bidirectional)
- **Transformers**: Parameters of the multi-head attention layer $\mathbf{Attn}_\theta^H$, which "splits" inputs to heads and merge back, plus positionwise feedforward. Deeply bidirectional with stacking. Can be parallelized and made sensitive to word ordering with position encodings, but tricky to train with a large number of parameters.

Any of these encoders can be "plugged in" to optimize a task-specific loss end-to-end

The Classification Framework

Architecture-agnostic abstraction

1. **Model.** Classify $x \in \mathcal{X}$ into $y \in \{1 \dots K\}$

$$\textbf{score}_\theta(x, y) = \underbrace{w_y}_{1\times d} \cdot \underbrace{\textbf{enc}_\theta(x)}_{d\times 1} + \underbrace{b_y}_{1\times 1}$$

$$p_\theta(y|x) \propto \exp(\textbf{score}_\theta(x, y))$$

   - Encoder architecture may be treated as a black box.
   - θ includes linear classifier $W = (w_1 \dots w_K) \in \mathbb{R}^{K\times d}, b \in \mathbb{R}^K$.

2. **Learning.** Minimize the empirical cross-entropy loss assuming N iid samples $(x_i, y_i) \sim \mathbf{pop}_{XY}$

$$\min_\theta -\frac{1}{N}\sum_{i=1}^{N} \log p_\theta(y_i|x_i)$$

Language Models (LMs)

$$\mathcal{X} = \text{all possible contexts}$$

$$\mathcal{Y} = \text{all possible words} = \mathcal{V}$$

1. **Model.** Given a context $x \in \mathcal{X}$, predict next word $y \in \mathcal{V}$

$$\textbf{score}_\theta(\text{the cat saw the}, \text{dog}) = \underbrace{w_{\text{dog}}}_{1\times d} \cdot \underbrace{\textbf{enc}_\theta(\text{the cat saw the})}_{d\times 1} + \underbrace{b_{\text{dog}}}_{1\times 1}$$

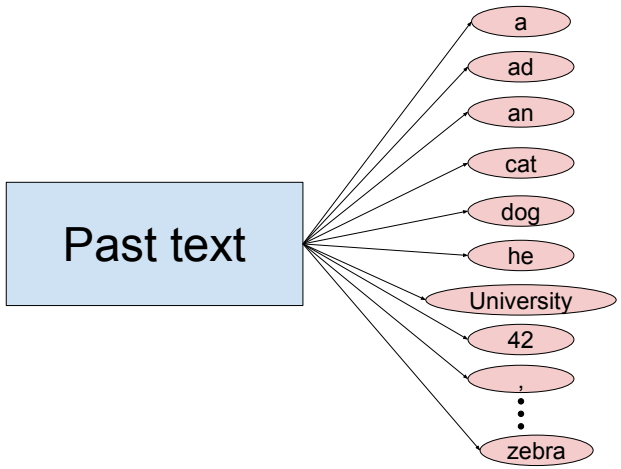
$$p_\theta(\text{dog}|\text{the cat saw the}) \propto \exp(\textbf{score}_\theta(\text{the cat saw the}, \text{dog}))$$

2. **Learning.** Minimize the empirical cross-entropy loss assuming N iid samples $(x_i, y_i) \sim \mathbf{pop}_{XY}$

$$\min_\theta -\frac{1}{N}\sum_{i=1}^{N} \log p_\theta(y_i|x_i)$$

Language Modeling as Classification

- Language modeling is "just" a V-way classification problem ($V = |\mathcal{V}|$).



- **BUT** some important questions
  1. How can we get labeled data for training LMs?
  2. How do we evaluate LMs?
  3. How do we encode all possible contexts (possibly super long)?
  4. What are the implications and applications of language modeling?

Language Modeling Data

- Language modeling doesn't require explicit annotation efforts
- Any piece of text yields a sequence of context-word pairs, e.g., *the dog saw the cat*
  - ([⟨bos⟩], the)
  - ([⟨bos⟩, the], dog)
  - ([⟨bos⟩, the, dog], saw)
  - ([⟨bos⟩, the, dog, saw], the)
  - ([⟨bos⟩, the, dog, saw, the], cat)
- Sometimes referred to as "self-supervised" learning
  - Different from supervised learning since there is no explicit annotation required
  - Different from (classical) unsupervised learning which explicitly optimizes some latent variable
  - More generally, "self-supervised" seems to mean any supervised learning problem crafted from unannotated data
- So training data for LMs is practically infinite!
  - Consider all of the web

Example: GPT-3 Training Data (Brown et al., 2020)

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

(number of stars in the Milky Way: 100–400 billion)

- Different types of text (web, books, Wikipedia), given different priorities (e.g., Wikipedia modeling is emphasized)
- Heavy text preprocessing efforts: automated filtering (train a classifier to identify high quality documents), approximate deduplication
- *Single* batch contains 3.2 million tokens
  - LM makes 3.2 million predictions in every batch during training

Evaluating Language Models by Perplexity

- No downstream performance to monitor
- Model selection based on validation cross-entropy loss
  - If overfitting, validation loss will start increasing
  - If training corpus is enormous, no model selection is necessary (cannot even do a single epoch)
- Popular practice: **exponentiate** LM loss for interpretability

$$\widehat{\mathbf{PPL}}(\theta) = \exp\Bigg(\underbrace{-\frac{1}{N_{\text{preds}}} \sum_{i=1}^{N_{\text{preds}}} \log p_\theta(y_i|x_i)}_{\widehat{J}(\theta) \text{ (empirical cross entropy)}} \Bigg) \quad \text{“perplexity”}$$

(Assuming natural log in $\widehat{J}(\theta)$.) Why?

- $\widehat{J}(\theta)$: # of nats to encode the behavior of $\mathbf{pop}_{Y|X}$ using p_θ
- $\exp(\widehat{J}(\theta))$: “branching factor” or “effective vocab size”

Values of Perplexity

Range of cross-entropy loss

$$\mathop{\mathbf{E}}_{(x,y)\sim \mathbf{pop}_{XY}} [-\log p_\theta(y|x)] \in [\underbrace{H(\mathbf{pop}_{Y|X})}_{\text{conditional entropy}}, \underbrace{\log V}_{\text{log vocab size}}]$$

Range of perplexity

$$\mathbf{PPL}(\theta) \in [\underbrace{\exp(H(\mathbf{pop}_{Y|X}))}_{\text{true branching factor}}, \underbrace{V}_{\text{branching factor of random tokens}}]$$

Range of *empirical* perplexity (i.e., on finite data)

$$\widehat{\mathbf{PPL}}(\theta) \in [\underbrace{1}_{\text{model assigns probability 1 on every instance}}, \underbrace{V}_{\text{model is uniformly random}}]$$

Token-Level Perplexity on Penn Treebank (PTB) Corpus

A preprocessed version of PTB popular for small-scale LM experiments (≈ 1 million training tokens, vocab size 10000)

- Kneser-Ney (classical LM) with cache: 125.7
- Simple feedforward: 140.2
- Simple RNN (BPTT length 50) (Mikolov and Zweig, 2012): 124.7
- LSTM (Bai et al., 2018): 78.93
- Transformer-XL (Dai et al., 2019): 54.55
- GPT-2* (Radford et al., 2019): 35.76
- GPT-3* (zero-shot) (Brown et al., 2020): 20.5

$*$ indicates the model is trained on (a huge amount of) additional text first. "Zero-shot" means the model is not additionally trained on PTB.

Beyond Perplexity

- Majority of words determined by local context
  - Model can assign low perplexity on an unnatural text that is locally coherent but globally gibberish
- Reading comprehension: Test overall context understanding
  - Examples: CNNDM (Hermann et al., 2015), LAMBADA (Paperno et al., 2016)

    Context: "Again, he left that up to you. However, he was adamant in his desire that it remain a private ceremony. He asked me to make sure, for instance, that no information be given to the newspaper regarding his death, not even an obituary.
    Target sentence: I got the sense that he didn't want anyone, aside from the three of us, to know that he'd even _____.
    Target word: died

    Many locally plausible choices (e.g., "married", "divorced", "graduated")
- Nonetheless, truly minimizing perplexity will imply global understanding
  - LAMBADA accuracy 0% with LSTM, 86.4% (!) with GPT-3

The Context Encoder

- Conceptually we need an encoder that can handle contexts of any length

$$\mathbf{enc}_\theta(x_1, x_2, x_3, x_4, x_5, x_6, \ldots, x_{1000000}) \in \mathbb{R}^d$$

- This is infeasible and, in most cases, unnecessary (how often do you need to look at more than past few hundred tokens to know what the next word should be?).
- Thus we typically set a **max input length** T that the encoder is expected handle (e.g., 512, 1024, 2048, more with bigger models)

$$\mathbf{enc}_\theta(x_1, x_2, x_3, x_4, x_5, x_6, \ldots, x_T) \in \mathbb{R}^d$$

- Different LMs only differ in architectural details of the encoder $\mathbf{enc}_\theta : \mathcal{V}^+ \rightarrow \mathbb{R}^d$
  - Feedforward? Recurrent? Transformer?

Batching for Language Modeling

- Raw corpus: [$\langle$bos$\rangle$, the, dog, saw, the, cat, $\langle$eos$\rangle$, $\langle$bos$\rangle$, it, laughed, $\langle$eos$\rangle$]
- Sequences prepared with max length 3

$\langle$bos$\rangle_{\text{the}}$	the$_{\text{dog}}$	dog$_{\text{saw}}$
saw$_{\text{the}}$	the$_{\text{cat}}$	cat$_{\langle\text{eos}\rangle}$
$\langle$eos$\rangle_{\langle\text{bos}\rangle}$	$\langle$bos$\rangle_{\text{it}}$	it$_{\text{laughed}}$
laughed$_{\langle\text{eos}\rangle}$	$\langle$eos$\rangle$	$\langle$pad$\rangle$

- Pick a random batch (batch size specified by max # predictions)

laughed$_{\langle\text{eos}\rangle}$	$\langle$eos$\rangle$	$\langle$pad$\rangle$
$\langle$bos$\rangle_{\text{the}}$	the$_{\text{dog}}$	dog$_{\text{saw}}$

- Loss for that batch

$$-\frac{1}{4}\big(\log p_\theta(\langle\text{eos}\rangle|\text{laughed}) + \log p_\theta(\text{the}|\langle\text{bos}\rangle) + \log p_\theta(\text{dog}|\langle\text{bos}\rangle, \text{the}) + \log p_\theta(\text{saw}|\langle\text{bos}\rangle, \text{the}, \text{dog})\big)$$

Trick: Tying Word Embeddings and Linear Classifier

- Regardless of encoder specifics, we have
  - Word embeddings $E \in \mathbb{R}^{d \times V}$
  - Linear classifier weight matrix $W \in \mathbb{R}^{V \times d}$
- Instead of having separate parameters we can reuse $W = E^\top$

$$p_\theta(y|x) = \text{softmax}_y\Big(\underbrace{E^\top}_{V \times d} \underbrace{\textbf{enc}_\theta(E(x))}_{d \times 1} \Big)$$

  This assumes $\textbf{enc}_\theta$ produces a d-dimensional hidden state.
- Halves the number of parameters for these weights $2Vd \mapsto Vd$
  - Can be viewed as regularization
  - Often easier to train, lower final perplexity

Example: Bengio et al. (2003)

- *A Neural Probabilistic Language Model*: One of the first successful early works on deep learning for NLP
- Simple feedforward network with a residual connection:

$$p_\theta(x_t|x_{t-T}\dots x_{t-1}) = \mathrm{softmax}_{x_t}(W\tanh(Ux+a)+b+Qx)$$

  where $x = (e_{x_{t-T}}\dots e_{x_{t-1}}) \in \mathbb{R}^{Td}$ concatenates word embeddings of cotext $x_{t-T}\dots x_{t-1} \in \mathcal{V}^T$ and parameters W, U, Q, a, b have appropriate shapes
- Improvement over traditional LMs (not neural), test perplexity on the Brown corpus with $V = 16383$:
  - Kneser-Ney back-off (classical LM): 321
  - Class-based back-off (another classical LM): 312
  - Feedforward ($T = 4$, $d = 30$, hidden dimension 100): 252

Example: GPTs

- A series (GPT-1, GPT-2, ...) of transformer-based LMs
- Transformer encoder, with learnable position embeddings and input-output weight sharing

$$p_\theta(x_t|x_{t-T}\dots x_{t-1}) = \mathrm{softmax}_{x_t}(E^\top \textbf{Transformer}_\theta(E(x_{t-T}\dots x_{t-1})))$$

- Game of scale (many versions within each, showing largest)
  - GPT-1: 117m parameters (12 layers, 12 heads, $d = 768$), batch size 64 of length 512
  - GPT-2: 1.5b parameters (48 layers, $d = 1600$, etc.), batch size 512 of length 1024
  - GPT-3: 175b parameters (96 layers, 96 heads, $d = 12288$)
- GPT-2 and 3 vocabulary: based on byte-pair encoding (BPE) tokenization ($V = 50257$), language-agnostic

Example: Recurrent Language Models

- How can we model arbirarily far past context?

  The man, after pondering for a long while [...$\gg T$ tokens...] *decided that* **[he/she]**

- Solution: Recurrent LM

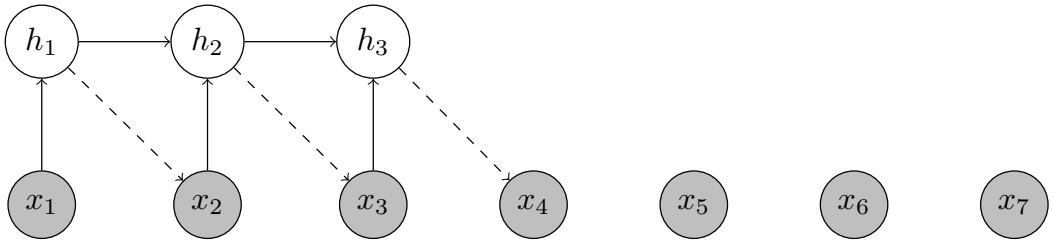
$$p_\theta(x_t|x_{<t}) = \text{softmax}_{x_t}(E^\top \textbf{RNN}_\theta(h_{t-1}, E(x_t)))$$

  h_{t-1} is a function of all history carried by the RNN

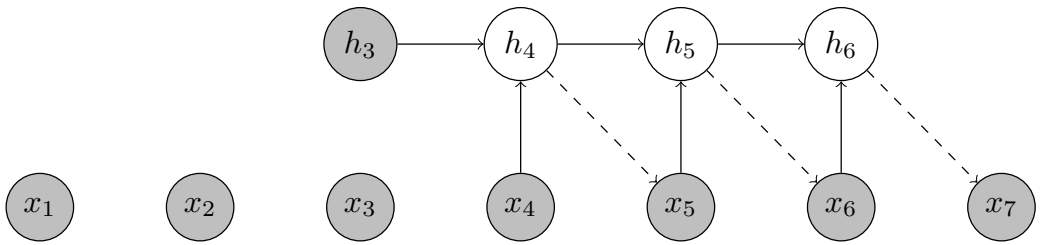
- Recurrent may not necessarily help
  - Very few tokens with long-range dependence
  - But these are the truly hard/interesting instances! (E.g., dependence between book chapters)
- How can we train RNNs with unbounded lengths?
  - Heuristic: Truncate sequence and build computation graphs sequentially, **reusing** previous hidden states without backpropagating
  - This trick is known as "backpropagation through time"

Backpropagation Through Time (BPTT)

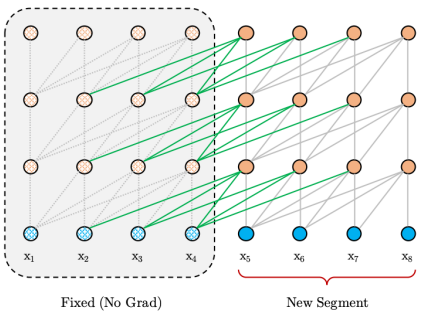
- Initial computation graph (BPTT length 3)



- Next computation graph



- Memory consumption is BPTT length, but can propagate information from arbitrarily far past
- Can be also used to make transformer-based LMs recurrent



Transformer-XL (Dai et al., 2019)

Implications of LM: Distribution Over Language

- If an LM is just a conditional word distribution, why do we call it a "language" model?

Implications of LM: Distribution Over Language

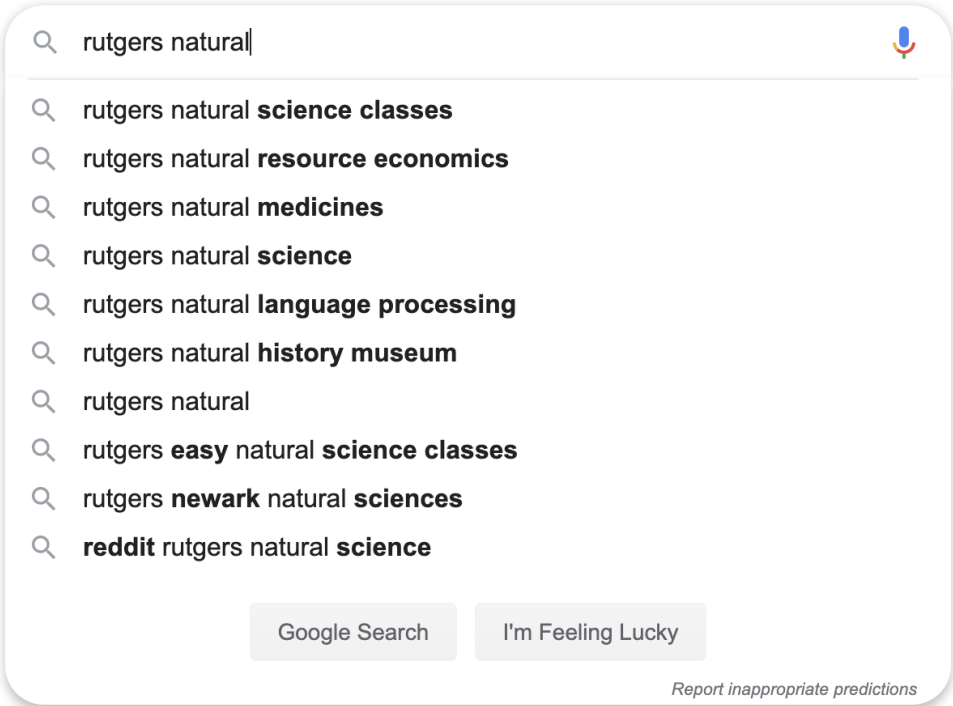
- If an LM is just a conditional word distribution, why do we call it a "language" model?
- By the **chain rule** in probability (not to be confused with chain rule in differentiation)

$$\begin{aligned}\mathbf{pop}(x_1 \dots x_T) = \mathbf{pop}(x_1|\langle\text{bos}\rangle) \\ \times \mathbf{pop}(x_2|\langle\text{bos}\rangle, x_1) \\ \times \mathbf{pop}(x_3|\langle\text{bos}\rangle, x_1, x_2) \times \cdots \\ \times \mathbf{pop}(x_T|\langle\text{bos}\rangle, x_1, x_2, \dots, x_{T-1}) \\ \times \mathbf{pop}(\langle\text{eos}\rangle|\langle\text{bos}\rangle, x_1, x_2, \dots, x_{T-1}, x_T)\end{aligned}$$

- Conditional word distribution implies distribution over language!

$$-\log p_\theta(x_1 \dots x_T) = -\sum_{t=1}^{T} \log p_\theta(x_t|x_{<t})$$

Applications of LM: Everywhere!



- **Ranking.** Apply p_θ to sentences and sort by probability
- **Generation.** Sample from p_θ to generate novel sentences

Generating Text from a Language Model

- How can we sample a random sentence from the (trained) model? Stepwise sampling by the chain rule:

$$
\begin{aligned}
(x_1 \dots x_T) \sim p_\theta \quad \Leftrightarrow \quad & x_1 \sim p_\theta(\cdot | \langle \text{bos} \rangle) \\
& x_2 \sim p_\theta(\cdot | \langle \text{bos} \rangle, x_1) \dots \\
& x_T \sim p_\theta(\cdot | \langle \text{bos} \rangle, x_1 \dots x_{T-1}) \\
& \langle \text{eos} \rangle \sim p_\theta(\cdot | \langle \text{bos} \rangle, x_1 \dots x_{T-1}, x_T)
\end{aligned}
$$

- Can calibrate randomness by introducing a "temperature" parameter $\tau > 0$ in softmax

$$
p_{\theta,\tau}(x | x_{<t}) = \frac{\exp\left(\frac{\textbf{score}(x_{<t}, x)}{\tau}\right)}{\sum_{x' \in \mathcal{V}} \exp\left(\frac{\textbf{score}(x_{<t}, x')}{\tau}\right)}
$$

Point-mass as $\tau \to 0$ (greedy decoding), uniform as $\tau \to \infty$

Other Generation Methods

- **Top-k sampling** (Fan et al., 2018). At each step, sample only from k most probable words $\mathcal{C} \subset \mathcal{V}$ (e.g., $k = 10, 20$)

$$p_{\theta,\mathcal{C}}(x|x_{<t}, \mathcal{C}) = \begin{cases} \frac{\exp(\textbf{score}(x_{<t}, x))}{\sum_{x' \in \mathcal{C}} \exp(\textbf{score}(x_{<t}, x'))} & \text{if } x \in \mathcal{C} \\ 0 & \text{otherwise} \end{cases}$$

  - Idea: Eliminate weird words by truncating vocabulary
  - Need to select right k. If too small and distribution is flat, miss out creativity. If too large, reintroduce weird words.

Other Generation Methods

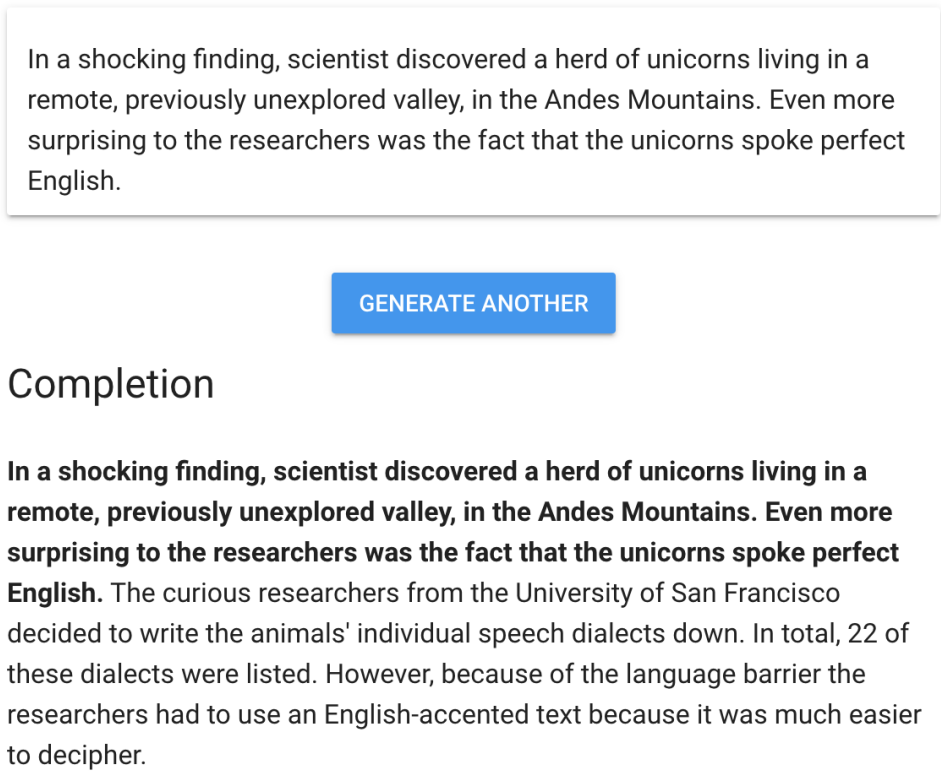
- **Top-k sampling** (Fan et al., 2018). At each step, sample only from k most probable words $\mathcal{C} \subset \mathcal{V}$ (e.g., $k = 10, 20$)

$$p_{\theta,\mathcal{C}}(x|x_{<t}, \mathcal{C}) = \begin{cases} \frac{\exp(\textbf{score}(x_{<t},x))}{\sum_{x' \in \mathcal{C}} \exp(\textbf{score}(x_{<t},x'))} & \text{if } x \in \mathcal{C} \\ 0 & \text{otherwise} \end{cases}$$

  - Idea: Eliminate weird words by truncating vocabulary
  - Need to select right k. If too small and distribution is flat, miss out creativity. If too large, reintroduce weird words.
- **Top-p sampling (aka. "nucleus sampling")** (Holtzman et al., 2020). Same thing but set $\mathcal{C} \subset \mathcal{V}$ to be top words whose probabilities sum to $> p$ (e.g., $p = 0.9$).
  - Idea: Alleviate the need to tune k in top-k sampling by making $|\mathcal{C}|$ dynamic (only care about probability mass coverage)
  - Still need to select p

Contextual Text Generation

- Start with a prompt $x_1 \dots x_J \in \mathcal{V}$, sample from $p_\theta(\cdot|x_1 \dots x_J)$ to complete the story.
- Online demo with a transformer-based LM: https://talktotransformer.com/



Evaluating Text Generation

- Automatic evaluation: Perplexity on a held-out corpus
  - Easy to obtain, ensures that model is diverse (must assign high probs to unseen text)
  - Quality score can be degenerate: perplexity infinite if model assigns prob ≈ 0 to *any* of test tokens
- Human evaluation: Overall quality of text on a scale of 1 to 5
  - Uses the human quality bar
  - Not scalable, diversity score can be degenerate: model might regurgitate training sentences (which will score high)
- Other automatic metrics based on n-gram overlaps with reference text
  - Examples: BLEU, ROUGE
  - Better quality estimation than perplexity, but can correlate poorly with human judgment
- Hybrid approaches
  - Example: HUSE (Hashimoto et al., 2019)
  - Classification error of predicting if a machine generated text or human, use human scores as features

Human-Like: Not Most Probable

- Naive sampling output:

  On Monday, the president of the United States of America and the president of the United States of America and the president of the United States of America and the president of the United States of America and . . .

- Why is human text not the most probable text? Possible hypotheses
  - Models are optimized for likelihood/perplexity: sufficient to only rely on short history
  - Grice's Maxims of Communication (Grice, 1975): Humans optimize against stating the obvious
- In practice, have to mess around with choices of top-k, top-p, softmax temperature to get desired generation qualities.

The Search Problem

- A different question: what's the *most* likely sentence under the (trained) model?

$$x^\star = (x_1^\star \dots x_{T^\star}^\star),\ T^\star = \underset{\substack{x_1 \dots x_T \in \mathcal{V}^T \\ T \in \{1 \dots T_{\max}\}}}{\arg\max} \sum_{t=1}^{T} \log p_\theta(x_t | x_{<t})$$

The Search Problem

- A different question: what's the *most* likely sentence under the (trained) model?

$$x^\star = (x_1^\star \dots x_{T^\star}^\star),\ T^\star = \underset{\substack{x_1 \dots x_T \in \mathcal{V}^T \\ T \in \{1 \dots T_{\max}\}}}{\arg\max} \sum_{t=1}^{T} \log p_\theta(x_t | x_{<t})$$

- **Greedy decoding.** Predict argmax at each time step while holding previous predictions fixed:

$$\hat{x}_1 = \underset{x_1 \in \mathcal{V}}{\arg\max}\ \log p_\theta(x_1 | x_0)$$

$$\hat{x}_2 = \underset{x_2 \in \mathcal{V}}{\arg\max}\ \log p_\theta(\hat{x}_1 | x_0) + \log p_\theta(x_2 | \hat{x}_1)$$

  and so on until $\hat{x}_T = \langle \text{eos} \rangle$. Return $\hat{x} = (\hat{x}_1 \dots \hat{x}_T)$.

- Runtime linear in V: $O(VT_{\max})$. But is $\hat{x} = x^\star$?

General Intractability of Exact Search

- Example: $\mathcal{V} = \{a, b\}$, assume $T = 2$ always so $\mathcal{X} = \{aa, ab, ba, bb\}$ (omitting $\langle \text{bos} \rangle$ and $\langle \text{eos} \rangle$)

$$p_\theta(a) = 0.6 \qquad p_\theta(a|a) = 0.5 \qquad p_\theta(b|a) = 0.5$$
$$p_\theta(b) = 0.4 \qquad p_\theta(a|b) = 0.9 \qquad p_\theta(b|b) = 0.1$$

Greedy decoding: aa or ab (prob 0.3), $x^\star = ba$ (prob 0.36)

General Intractability of Exact Search

- Example: $\mathcal{V} = \{a, b\}$, assume $T = 2$ always so $\mathcal{X} = \{aa, ab, ba, bb\}$ (omitting $\langle \text{bos} \rangle$ and $\langle \text{eos} \rangle$)

$$p_\theta(a) = 0.6 \qquad p_\theta(a|a) = 0.5 \qquad p_\theta(b|a) = 0.5$$
$$p_\theta(b) = 0.4 \qquad p_\theta(a|b) = 0.9 \qquad p_\theta(b|b) = 0.1$$

  Greedy decoding: aa or ab (prob 0.3), $x^\star = ba$ (prob 0.36)
- We never know how the future unfolds.
  - Even if a choice looks suboptimal now, it may *eventually* lead to better sequences.
- In general, we cannot avoid an exhaustive search over all $O(V^{T_{\max}})$ possible sequences to find $x^\star$.
- This is largely because x_t can arbitrarily condition on the entire history $x_{>t}$.
  - If distribution is Markov $p_\theta(x_t|x_{>t}) = p_\theta(x_t|x_{t-M} \dots x_{t-1})$, there exists a dynamic programming approach to find $x^\star$ in $O(V^M T_{\max})$ (will cover later in the course).

Approximate Search

- While exact search is intractable, we can approximate it effectively by **beam search**.
- Has a controllable hyperparameter β (beam size)
  - $\beta = 1$: Greedy decoding
  - $\beta = V^{T_{\max}}$: Exact search
- Idea: Maintain only the top β candidate sequences (called "hypotheses") at each time step.
- $t = 1$: Hypotheses $(h^{(1)} \dots h^{(\beta)})$ are simply words with highest $\log p_\theta(x_1 | \langle \text{bos} \rangle)$ values
- $t = 2$: Construct a stepwise hypothesis space of size βV

$$\mathcal{H}_2 = \left\{ h^{(b)} x_2 | b \in \{1 \dots \beta\}, x_2 \in \mathcal{V} \right\}$$

Update hypotheses to be the top β elements of $\mathcal{H}_2$, each has score $\log p_\theta(h^{(1)} | \langle \text{bos} \rangle) + \log p_\theta(x_2 | \langle \text{bos} \rangle, h^{(1)})$

General Beam Search

- Assumption: Additive sequence score (e.g., log prob)

$$s(x_1 \dots x_t) = s(x_1) + s(x_2|x_1) + \dots + s(x_t|x_1 \dots x_{t-1})$$

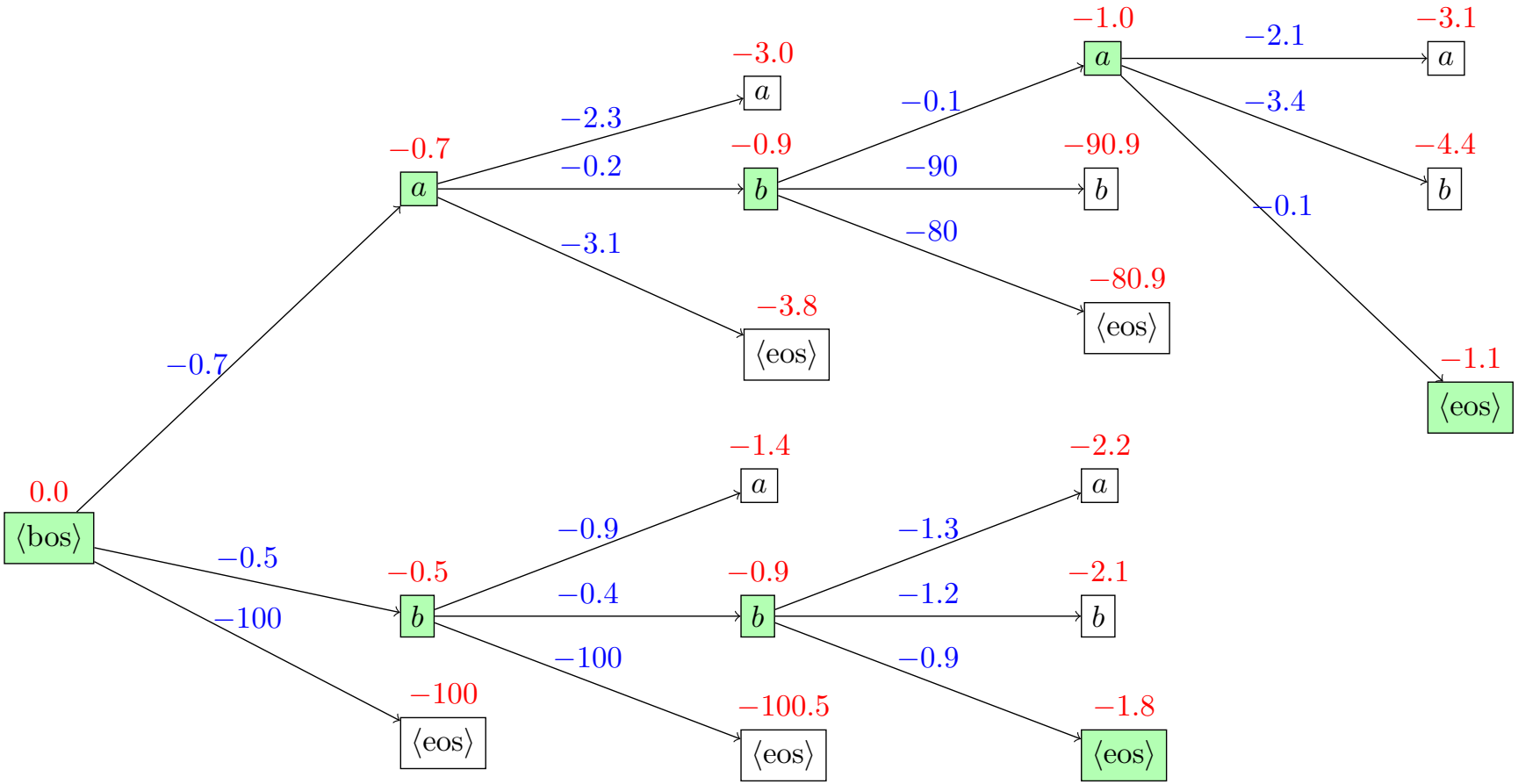
- At every step $t > 1$, we assume
  - β best hypotheses so far: $h^{(1)} \dots h^{(\beta)}$ with scores $s^{(1)} \dots s^{(\beta)}$
- Stepwise hypothesis space of size βV

$$\mathcal{H}_t = \left\{ h^{(b)} x | b \in \{1 \dots \beta\}, x \in \mathcal{V} \right\}$$

  Each $h(b, x) \in \mathcal{H}_t$ is scored $s^{(b,x)} = s^{(b)} + s(x|h^{(b)})$.
- Update hypotheses/scores to be β elements of $\mathcal{H}_t$ with highest $s^{(b,x)}$.
  - Must align $h(b, x)$ to $h^{(b)}$ carefully so that we can continue with correct previous hidden states

Beam Search: A Complete Example

Beam size 2, vocab $\{a, b, \langle\text{bos}\rangle, \langle\text{eos}\rangle\}$ (omitting $\langle\text{bos}\rangle$ branches when expanding assuming impossible), edge value $\log p_\theta(x_t|x_{<t})$, node value $\log p_\theta(x_1 \dots x_t)$. At each time step, green boxes indicate top-2 hypotheses to continue.



Completed hypothesis-score pairs: $(aba, -1.1), (bb, -1.8)$. Return aba as an approximation to $x^\star$.

Other Beam Search Details

- Specify max number of steps $T_{\max}$
  - If no hypothesis is completed within $T_{\max}$ steps, return the top (unfinished) element.
- Runtime: $O(\text{top}_\beta(\beta V) T_{\max})$ where $\text{top}_\beta(\beta V)$ is the time needed to get β highest-scoring items from $\beta \times V$ items
  - $O(V T_{\max})$ if $\beta = 1$ (greedy), a great improvement over brute-force $O(V^{T_{\max}})$
- **Length penalty.** Instead of using $\log p_\theta(x_1 \dots x_t)$, consider $\frac{1}{t} \log p_\theta(x_1 \dots x_t)$ during beam search
  - Alleviates the problem of rewarding sequences for being short.
- More generally, can introduce various penalties/constraints during beam search
  - **Coverage penalty**: Encourage hypotheses to cover certain token types
  - n-**gram blocking**: Prune branches with repeated n-grams
- Replace β-argmax with β samples (without replacement) to get a "sampling version" of beam search

Conditional Language Modeling

- Idea: make LM condition on something.

$$p_\theta(y_1 \dots y_T | x) = \prod_{t=1}^{T+1} p_\theta(y_t | x, y_{<t})$$

- Applications: Machine translation, summarization, dialogue, image captioning, video captioning, ...
- By the chain rule, only need to model conditional word distribution, but now conditioning on x as well as history $y_{<t}$.
- Same training: Per-token cross-entropy loss
- **Conditional decoding**: Given x_{test}, find

$$y^\star, T^\star = \underset{\substack{y_1 \dots y_T \in \mathcal{V}^T \\ T \in \{1 \dots T_{\max}\}}}{\arg\max} \sum_{t=1}^{T} \log p_\theta(y_t | x_{\text{test}}, y_{<t})$$

Again approximate by beam search