

CS 533: Natural Language Processing

Language Models, Beam Search, Text Generation

Karl Stratos



Rutgers University

Review: Word Embeddings



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

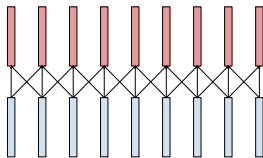
- ▶ Each x_t not contextual
- ▶ Can be **contextualized** with additional transformations!

Once we've settled on (either raw or contextual) word embeddings, can get a **single-vector** representation of text in various ways, e.g.,

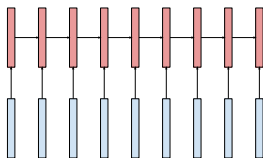
- ▶ **Mean pooling.** Average embeddings (“continuous bag-of-words”)
- ▶ **Max pooling.** Take elementwise maximum (e.g., in CNNs)
- ▶ **Single token.** Take the embedding corresponding to a single token, assuming it's already function of all inputs (e.g., in transformers)

Review: Contextualizing Word Embeddings

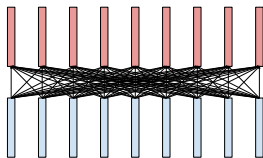
- ▶ **Convolutional:** Slide n -gram filters. Can be stacked.



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



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



Review: Parameters to Learn

- ▶ **CNNs:** Filter matrix $U \times \mathbb{R}^{n \times d}$ 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 of the recurrent cell (feedforward) $\text{RNN}_\theta : \mathbb{R}^{d_h} \times \mathbb{R}^d \rightarrow \mathbb{R}^d$. 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 Attn_θ^H , which “split” inputs to heads and merge back. 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, jointly with any other layers (e.g., final linear classifier)

Architecture-Free Abstraction

In most cases we will work with the following uniform setting:

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

$$p_{\theta}(y|x) = \text{softmax}_y \left(\underbrace{W}_{L \times d} \underbrace{\text{enc}_{\theta}(x)}_{d \times 1} + \underbrace{b}_{L \times 1} \right)$$

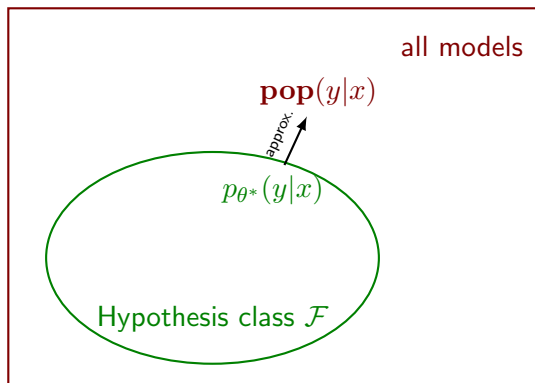
θ includes linear classifier parameters (W, b)

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

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

No need to talk about specific architectures

Review: Error Decomposition



With nonlinear neural encoders, approximation error ≈ 0

Language Models (LMs)

- ▶ **pop**: Distribution over natural language \mathcal{X} in some units (e.g., sentences, paragraphs, articles). Will assume sentences.
 - ▶ For convenience assume finite \mathcal{X} : $|\mathcal{X}| = V^{T_{\max}}$ where T_{\max} is max text length storable in a physical computer
 - ▶ But practically infinite
- ▶ $x \sim \mathbf{pop}$: Single sentence, randomly generated from the true distribution (e.g., human reporter)
- ▶ **Goal. Density estimation**: learn $p_{\theta}(x) \approx \mathbf{pop}(x)$. Use cases
 - ▶ **Ranking**. Apply p_{θ} to sentences to determine which sentences are more likely (that is, under **pop**)
 - ▶ **Generation**. Sample from p_{θ} to generate sentences that may be novel (not part of training data)
- ▶ **Unsupervised problem**: We don't need any explicit annotation to train an LM.
 - ▶ Unlimited available corpora, many from the web (Wikipedia dump: 3 billion tokens, Common Crawl (filtered): 410 billion tokens)

Chain Rule in Probability

- ▶ Direct softmax over \mathcal{X} (space of all sentences) impractical
- ▶ Solution: **Chain rule** in probability (not to be confused with chain rule in differentiation)


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

$\langle \text{bos} \rangle$ and $\langle \text{eos} \rangle$ special symbols in vocab \mathcal{V} (“beginning/end of sentence”)

- ▶ Thus only need a **conditional word distribution** over $x_t \in \mathcal{V}$
 - ▶ Conditioning on history/context $x_{<t} \in \mathcal{V}^{|x_{<t}|}$

Language Models Are Everywhere



Q rutgers natural 

- Q rutgers natural **science classes**
- Q rutgers natural **resource economics**
- Q rutgers natural **medicines**
- Q rutgers natural **science**
- Q rutgers natural **language processing**
- Q rutgers natural **history museum**
- Q rutgers natural
- Q rutgers **easy** natural **science classes**
- Q rutgers **newark** natural **sciences**
- Q **reddit** rutgers natural **science**

[Report inappropriate predictions](#)

LM Training Loss

- ▶ Sample sentences denoted as $x^{(1)} \dots x^{(M)} \sim \mathbf{pop}$
- ▶ $x^{(i)} = (x_1^{(i)}, \dots, x_{T_i}^{(i)})$, $x_0^{(i)} = \langle \text{bos} \rangle$, $x_{T_i+1}^{(i)} = \langle \text{eos} \rangle$
 - ▶ Let $N = M + \sum_{i=1}^M T_i$, total number of token predictions
- ▶ Empirical cross-entropy loss (token-level):

$$\hat{J}(\theta) = -\frac{1}{N} \sum_{i=1}^M \sum_{t=1}^{T_i+1} \log p_{\theta}(x_t^{(i)} | x_{<t}^{(i)})$$

where $x_{<t}^{(i)} = (x_0^{(i)}, \dots, x_{t-1}^{(i)}) \in \mathcal{V}^t$.

- ▶ **Data-efficient:** $T + 1$ “labeled examples” in each sentence

(inputs, targets) = $((x_0, x_{0:1}, \dots, x_{0:T}), (x_1, x_2, \dots, x_{T+1}))$

- ▶ In practice sentences batched with special padding symbol $\langle \text{pad} \rangle$ to have equal length T .
 - ▶ Mask $\langle \text{pad} \rangle$ losses: $(l_1, l_2, l_3, l_4, l_5) \mapsto (l_1, l_2, l_3, 0, 0)$

Perplexity

- ▶ Language modeling is unsupervised (i.e., 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

$$\widehat{\text{Perplexity}}(\theta) = \exp\left(\widehat{J}(\theta)\right)$$

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

- ▶ $\widehat{J}(\theta)$: Number of **bits** to encode the behavior of **pop**($x_t|x_{<t}$) using $p_\theta(x_t|x_{<t})$
- ▶ $\exp(\widehat{J}(\theta))$: “branching factor” or “effective vocab size”

Values of Perplexity

- ▶ True (token-level) cross-entropy loss

$$J(\theta) = \mathbf{E}_{x \sim \mathbf{pop}, t \sim \text{Unif}\{1, \dots, |x|+1\}} [-\log p_{\theta}(x_t | x_{<t})]$$

- ▶ Largest when p_{θ} uniform: $J(\theta) \leq \log V$
- ▶ Smallest when $p_{\theta} = \mathbf{pop}$: $J(\theta) \geq H(\mathbf{pop}(x_t | x_{<t}))$
- ▶ Thus range of true perplexity $\exp(J(\theta))$ is

$$\mathbf{Perplexity}(\theta) \in \left[\underbrace{\exp(H(\mathbf{pop}(x_t | x_{<t})))}_{\text{True branching factor under } \mathbf{pop}}, \underbrace{V}_{\text{branching factor of random tokens}} \right]$$

- ▶ Range of **empirical** perplexity on finite training data

$$\widehat{\mathbf{Perplexity}}(\theta) \in \left[\underbrace{1}_{\text{Model assigns probability 1 to every training instance}}, V \right]$$

Markov Assumption

- ▶ Inconvenient to work with arbitrarily long context $x_{<t}$ (unless we're using recurrent models)
- ▶ **Markov assumption:** Threshold context size to K , e.g., $K = 2$

$$\begin{aligned} \mathbf{pop}(x_1 \dots x_T) &\approx \mathbf{pop}(x_1 | \langle \mathbf{bos} \rangle_0, \langle \mathbf{bos} \rangle) \\ &\quad \times \mathbf{pop}(x_2 | \langle \mathbf{bos} \rangle, x_1) \\ &\quad \times \mathbf{pop}(x_3 | x_1, x_2) \times \dots \\ &\quad \times \mathbf{pop}(x_T | x_{T-2}, x_{T-1}) \\ &\quad \times \mathbf{pop}(\langle \mathbf{eos} \rangle | x_{T-1}, x_T) \end{aligned}$$

- ▶ Idea: Choice of current word depends on recent history only
- ▶ Now LM just a classifier from \mathbb{R}^{Kd} (K word embeddings) to \mathcal{V}

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 $W = E^\top$

$$p_\theta(x_t | x_{<t}) = \text{softmax}_{x_t} \left(\underbrace{E^\top}_{V \times d} \underbrace{\text{enc}_\theta(E(x_{<t}))}_{d \times 1} \right)$$

This assumes enc_θ produces a d -dimensional hidden state.

- ▶ Halves the number of parameters for these weights
 $2Vd \mapsto Vd$
 - ▶ Can be viewed as regularization
 - ▶ Empirically almost always helps (easier to train, lower final perplexity)

Example: Bengio et al. (2003)

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

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

where $x \in \mathbb{R}^{Kd}$ concatenates context word embeddings and parameters W, U, Q, a, b are appropriately shaped

- ▶ 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 ($K = 4, d = 30$, hidden dimension 100): 252

Example: Generative Pre-Training (GPT) LMs

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

$$p_{\theta}(x_t|x_{t-K} \dots x_{t-1}) = \text{softmax}_{x_t}(E^{\top} \mathbf{Transformer}_{\theta}(E(x_{t-K} \dots x_{t-1})))$$

- ▶ 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

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

- ▶ 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* mini-batch contains 3.2m tokens
 - ▶ LM makes 3.2 million predictions in every batch during training

Recurrent LMs

- ▶ Markov assumption can be limiting.

*The man, after pondering for a long while [... > K tokens...]
decided that [he/she]*

- ▶ Solution: Recurrent LM

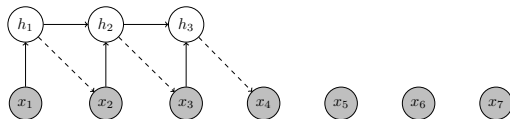
$$p_{\theta}(x_t|x_{<t}) = \text{softmax}_{x_t}(E^{\top} \mathbf{RNN}_{\theta}(h_{t-1}, E(x_t)))$$

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

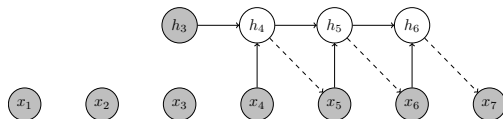
- ▶ Making LM recurrent may not help if context window is already large (e.g., GPT-3 uses $K = 2048$ tokens)
 - ▶ Few tokens with extreme-length dependence
 - ▶ But these are the truly hard/interesting instances! (E.g., dependence between book chapters)
- ▶ How can we train RNNs with unbounded lengths?
 - ▶ 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)

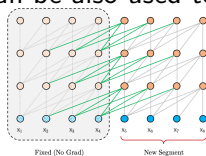
- ▶ Initial computation graph (BPTT length 3)



- ▶ 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)

Token-Level Perplexity on Penn Treebank (PTB) Corpus

A preprocessed version of PTB popular for small-scale LM experiments (\approx 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
 - ▶ Only diminishing returns since most tokens are resolvable without global understanding
 - ▶ LAMBADA accuracy 0% with LSTM, 86.4 (!) with GPT-3

The Search Problem

- ▶ What's the most likely sentence under the (trained) model?

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

The Search Problem

- ▶ What's the most likely sentence under the (trained) model?

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

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

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

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

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 $\hat{x} = x^*$?

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 $\langle \text{eos} \rangle$)

$$p_{\theta}(a) = 0.6 \quad p_{\theta}(a|a) = 0.5 \quad p_{\theta}(b|a) = 0.5$$

$$p_{\theta}(b) = 0.4 \quad p_{\theta}(a|b) = 0.9 \quad p_{\theta}(b|b) = 0.1$$

Greedy decoding: aa or ab (prob 0.3), $x^* = ba$ (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 $\langle \text{eos} \rangle$)

$$\begin{array}{lll} p_{\theta}(a) = 0.6 & p_{\theta}(a|a) = 0.5 & p_{\theta}(b|a) = 0.5 \\ p_{\theta}(b) = 0.4 & p_{\theta}(a|b) = 0.9 & p_{\theta}(b|b) = 0.1 \end{array}$$

Greedy decoding: aa or ab (prob 0.3), $x^* = ba$ (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^* .
- ▶ This is largely because x_t can arbitrarily condition on the entire history $x_{>t}$.
 - ▶ If distribution is Markov $p_{\theta}(x_t|x_{>t}) = p_{\theta}(x_t|x_{t-K} \dots x_{t-1})$, there exists a dynamic programming approach to find x^* in $O(V^K T_{\max})$ (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**

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

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

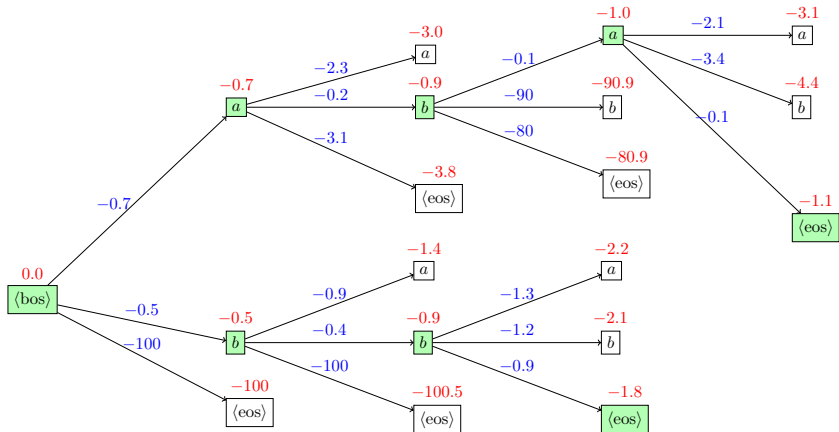
$$\mathcal{H}_t = \left\{ h^{(b)}x \mid b \in \{1 \dots \beta\}, x \in \mathcal{V} \right\}$$

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^* .

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
 - ▶ $O(VT_{\max})$ 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 that result in a repeated n -gram

Generating Text from a Language Model

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

$$\begin{aligned}(x_1 \dots x_T) \sim p_\theta &\Leftrightarrow & \mathbf{x}_1 &\sim p_\theta(\cdot | \langle \text{bos} \rangle) \\ & & \mathbf{x}_2 &\sim p_\theta(\cdot | \langle \text{bos} \rangle, \mathbf{x}_1) \dots \\ & & \mathbf{x}_T &\sim p_\theta(\cdot | \langle \text{bos} \rangle, \mathbf{x}_1 \dots \mathbf{x}_{T-1}) \\ & & \langle \text{eos} \rangle &\sim p_\theta(\cdot | \langle \text{bos} \rangle, \mathbf{x}_1 \dots \mathbf{x}_{T-1}, \mathbf{x}_T)\end{aligned}$$

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

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

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

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., $k = 10, 20$)

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

- ▶ 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., $k = 10, 20$)

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

- ▶ 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** (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

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., $k = 10, 20$)

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

- ▶ 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** (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
- ▶ Any of these sampling schemes can be used in conjunction with beam search: Replace β -argmax with β samples (without replacement)

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/>

In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

GENERATE ANOTHER

Completion

In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English. The curious researchers from the University of San Francisco decided to write the animals' individual speech dialects down. In total, 22 of these dialects were listed. However, because of the language barrier the researchers had to use an English-accented text because it was much easier to decipher.

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 beam search/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 the choice of beam search (with penalties/blocking), top- k , top- p , softmax temperature to get desired generation qualities.

Conditional Language Modeling

- ▶ Idea: make LM condition on something.

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

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

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

Again approximate by beam search