# CS 4650: Problem Set 1 (Fall 2025)

Instructor: Kartik Goyal

TAs: Sunho Park, Myungjin Shin, Kunal Daga,

Yunxiang Yan, Jiangyue Zhu, James Dill

Deadline: Friday, November 21 2025, 11:59 PM

## Instructions

- Submissions will be managed via Gradescope. While submitting your solution to Gradescope, please correctly label all the pages which has the solution to the given question. Incorrect/missing labeling will lead to non-evaluation of those questions.
- For legibility purposes, you are encouraged to use a typesetting software like Latex to submit the solutions for this problem set (see Overleaf for a quick and easy setup). We have provided a solution template that you can use. PDF of handwritten solutions is also acceptable. Please write your answers clearly, as **we won't be able to award credit for answers that we deem illegible**.
- Please show your workings for every question unless specifically mentioned.
- You are expected to submit your original work for this problem set. It is your responsibility to abide by academic integrity policies of this course. All incidents of suspected dishonesty, plagiarism, or violations of the Georgia Tech Honor Code will be subject to the institute's Academic Integrity procedures.
- Please refer to Canvas for late days policy.
- For any clarifications or doubts, use the public thread on Piazza so that other students can also benefit from those. Refrain from sharing solution snippets.
- If you have any questions that require you to share solution snippets, please create a private Piazza post. Note that we would not be able to verify generic things like if your answer is correct/wrong, what you should do next, etc. However, we will do our best to help you out and give you a direction if you have a specific query.
- In general, $\mathbb{E}_{x\sim D} f(x)$ refers to the fact that we are computing expected value of function $f$ under the data generating distribution for $x$. This expectation is typically approximated by treating training data instances as samples from the data distribution.

- You will find it useful to look up common notation for nested expectations and joint expectations. For example, in this assignment, $\mathbb{E}_{x\sim D, y\sim D'} f(x,y) = \mathbb{E}_{x\sim D}\mathbb{E}_{y\sim D'} f(x,y)$.

# 1 Softmax and Temperature (6 pts)

Assume, we are working in a 3-way classification setting. Hence, given a score (logits) vector $\mathbf{z} = [z_1, z_2, z_3]$ for the three classes, we perform a softmax operation over this vector to get a probability distribution over the classes:

$$p(\mathbf{y}; \mathbf{z}) = \text{softmax}(\mathbf{z}) = [\frac{\exp(z_1)}{Z}, \frac{\exp(z_2)}{Z}, \frac{\exp(z_3)}{Z}]$$

where $Z = \sum_{i=1}^{3} \exp(z_i) = \exp(z_1) + \exp(z_2) + \exp(z_3)$.
We saw an instantiation of softmax in class that involved a temperature hyperparameter $T$. Specifically, this temperature parameter is used to scale the logits first before passing them through the softmax function. So this function looks like:

$$\text{softmax}(\mathbf{z}; T) = [\frac{\exp(\frac{z_1}{T})}{Z_T}, \frac{\exp(\frac{z_2}{T})}{Z_T}, \frac{\exp(\frac{z_3}{T})}{Z_T}]$$

where $Z_T = \sum_{i=1}^{3} \exp(\frac{z_i}{T})$. If we take the limit of this function such that the temperature tends to 0 i.e. $\lim_{T \to 0} \text{softmax}(\mathbf{z}; T)$, very interesting properties emerge.

(a) Show in detail what this temperature based softmax converges to as $T$ goes to 0.

Consider the following cases:

- $z_1 \neq z_2 \neq z_3$
- $z_1 = z_2 = z_3$
- $z_1 = z_2 \neq z_3$, and $z_1 < z_3$
- $z_1 = z_2 \neq z_3$, and $z_1 > z_3$

*Hint: Expand the formula for the temperature softmax at each position, rearrange the terms such that the numerator is* 1*, and take the limit.*

# 2 Tokenization and Morphology [16 pts, 6 bonus pts]

(a) **Compression gain [5 points]:** Consider training a BPE tokenizer on an English corpus. Below is a list of merges (symbol bigrams) $m_t$ learnt at step $t$ in BPE training along with their corresponding frequencies $f(m_t)$. (The frequency $f(m_t)$ is the number of times the merge $m_t$ appears in the corpus *after applying all merges up to step* $t$.)

Denote the number of characters in the training corpus by $|D|$. Express in terms of $|D|$, the length of the training corpus after you perform the *apply-BPE/segmentation* function after step 7. *Explain* how you arrived at this answer.

For simplicity, assume that the corpus is pretokenized and only consists of lower case English characters $a, \ldots, z$.

1/ Let $m = \max(z_1, z_2, z_3)$

$$\Rightarrow \text{softmax}(z;T)_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} = \frac{\exp((z_i - m)/T)\exp(m/T)}{\sum_j \exp((z_j - m)/T)\exp(m/T)} = \frac{\exp((z_i - m)/T)}{\sum_j \exp((z_j - m)/T)}$$

* Note:

- For any $j$ with $z_j < m$, we have $z_j - m < 0$, so $(z_j - m)/T \to -\infty$ as $T \to 0^+$, hence $\exp((z_j - m)/T) \to 0$.
- For any $j$ with $z_j = m$, $(z_j - m)/T = 0$, so $\exp((z_j - m)/T) = 1$ for all $T$.

Let the set of maximizers be: $M = \{i \mid z_i = m\}$, $|M| = k$

Then in limit: $\begin{cases} i \in M : \text{numerator} \to 1, \text{denominator} \to k \\ i \notin M : \text{numerator} \to 0, \text{denominator} \to k \end{cases}$

So: $\lim_{T \to 0} \text{softmax}(z;T)_i = \begin{cases} \frac{1}{k}, & i \in M, \\ 0, & i \notin M \end{cases}$

Case 1: $z_1 \neq z_2 \neq z_3$ (all distinct)

Exactly one of them is $\max(z_i)$. Then $k = 1$: $\lim_{T \to 0} \text{softmax}(z;T) = \text{one-hot on } \arg\max_i z_i$

Concretely, if $z_1$ is largest: $(1,0,0)$; if $z_2$ is largest: $(0,1,0)$, etc.

Case 2: $z_1 = z_2 = z_3$

All 3 tie for max: $M = \{1,2,3\}$, $k = 3$

$$\lim_{T \to 0} \text{softmax}(z;T) = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$$

Case 3: $z_1 = z_2 \neq z_3$, and $z_1 < z_3$

$z_3$ is unique max, so $M = \{3\}$, $k = 1$

$$\lim_{T \to 0} \text{softmax}(z;T) = (0,0,1)$$

Case 4: $z_1 = z_2 \neq z_3$, $z_1 > z_3$

Now $z_1, z_2$ are max, so $M = \{1,2\}$, $k = 2$

$$\lim_{T \to 0} \text{softmax}(z;T) = \left(\frac{1}{2}, \frac{1}{2}, 0\right)$$

| Step (t) | Merges | Frequency $f(m_t)$ |
|---|---|---|
| 1 | (a,b) | 1000 |
| 2 | (a,c) | 500 |
| 3 | (a,d) | 300 |
| 4 | (ab,c) | 250 |
| 5 | (c,d) | 100 |
| 6 | (abc,d) | 90 |
| 7 | (ab,cd) | 80 |

Table 1: Merges and frequencies

(b) **Evolution of merge frequency [3 pts]:** Suppose we are training a BPE tokenizer on a large corpus consisting of English sentences. Let $M_t$ be the set of all possible merges at step $t$ of BPE training. Let the function $f_t : M_t \to \mathbb{Z}^*$ map each merge to its frequency. (With this notation, the merge learnt at step $t$ is simply $m_t \in \text{argmax}_{m \in M_t} f_t(m)$.) Explain why $f_t$ will attain the same constant value for most $m \in M_t$ after a sufficiently large number of steps. (In fact, we just get $f_t \equiv 1$ except in edge cases.)

(c) **Ambiguity in tokenization [8 pts]:** Consider the set $\Delta$ consisting of the tokens:

$$\{\texttt{st}, \texttt{ation}, \texttt{tion}, \texttt{stat}, \texttt{ion}, \texttt{union}, \texttt{un}, \texttt{unioniza}, \texttt{i}, \texttt{a}, \texttt{z}, \texttt{at}\} \tag{1}$$

A *segmentation* $T$ of a word is a decomposition in terms of the tokens contained in $\Delta$. For example, an input word `station` has the following segmentations:

$$\begin{aligned} T_1 &= (\texttt{stat}, \texttt{ion}), \\ T_2 &= (\texttt{st}, \texttt{at}, \texttt{ion}), \\ T_3 &= (\texttt{st}, \texttt{a}, \texttt{tion}) \\ T_4 &= (\texttt{st}, \texttt{ation}) \end{aligned} \tag{2}$$

The BPE tokenizer computes a segmentation (Apply-BPE function) by encoding it greedily in a *left-to-right* manner. That is, it assigns a unique segmentation to each word by greedily selecting the longest possible token moving left-to-right. For example, with the above vocabulary, the word `station` would be segmented as $T_2 = (\texttt{stat}, \texttt{ion})$ because `stat` is the longest possible token in the vocabulary starting with `st`. We will investigate if this is the best strategy for tokenization.

Suppose, we had a way to evaluate quality of a segmentation. Assume the tokens in $\Delta$ have the following *unigram probabilities*. (Table 2)

Then, *the probability of a segmentation* $T$ is given by a product of its symbols i.e.

$$p(T) = \prod_{t \in T} p(t) \tag{3}$$

So that the probabilities of the segmentations of `station` are given by:

| Token | Probability |
|---|---|
| ation, z, ion, union | 0.025 |
| unioniza, i | 0.05 |
| st, un, at, tion | 0.05 |
| a, stat | 0.3 |

Table 2: Unigram probabilities. Notice that they sum up to 1.

$$
\begin{aligned}
p(T_1) &= p(\texttt{stat})p(\texttt{ion}) = 0.3 \cdot 0.025 = 0.0075, \\
p(T_2) &= p(\texttt{st})p(\texttt{at})p(\texttt{ion}) = 0.05 \cdot 0.05 \cdot 0.025 = 0.0000625, \\
p(T_3) &= p(\texttt{st})p(\texttt{a})p(\texttt{tion}) = 0.05 \cdot 0.3 \cdot 0.05 = 0.00075 \\
p(T_4) &= p(\texttt{st})p(\texttt{ation}) = 0.05 \cdot 0.025 = 0.00125
\end{aligned}
\tag{4}
$$

Thus, the most probable segmentation of `station` is $T_1 = (\texttt{stat}, \texttt{ion})$ with probability $p(T_1) = 0.0075$. In this particular example, the left-to-right segmentation coincides with the most probably segmentation given by the Unigram LM.

Let's consider how the string $y = \texttt{unionization}$ is tokenized under the vocabulary $\Delta$.

i) What is the left-to-right segmentation of $y$? **[2 pts]**

ii) What are all possible segmentations of $y$ using the vocabulary $\Delta$? **[3 pts]**

iii) Compute the probability of each segmentation according to the unigram probability $p$. **[1 pts]**

iv) Is the left-to-right segmentation the most probable segmentation of $y$? **[2 pts]**

v) **Bonus [6 pts]:** Let's now take a new vocabulary $\Delta'$ given by

$$
\Delta' = \Sigma \cup \Delta \tag{5}
$$

where $\Sigma$ is the entire lower case English alphabet. Now you have added 23 more symbols to the vocabulary. The new unigram probabilities are given in (Table 3).

| Token | Probability |
|---|---|
| ation, z, ion, union | 0.7*0.025 |
| unioniza, i | 0.7*0.05 |
| st, un, at, tion | 0.7*0.05 |
| a, stat | 0.7*0.3 |
| {b, . . . , h, j, . . . , y} | $\frac{0.3}{23}$ |

Table 3: New unigram probabilities after adding all English characters. Notice that they sum up to 1.

vi) How would you compute the number of segmentations of $y = \texttt{unionization}$ in an *efficient* manner? (**4 points**)

2/

a) Compression gain from BPE after step 7:

Each merge replaces two adjacent symbols with a single new symbol. Everytime a merge pair $m_t$ appears, the corpus length shrinks by 1 symbol

If a merge $m_t$ has frequency $f(m_t)$ after merges up to step $t$ => apply that merge reduces length by exactly $f(m_t)$

Total reduction in length after step 7:

$$\Delta L = \sum_{t=1}^{7} f(m_t) = 1000 + 500 + 300 + 250 + 100 + 90 + 80 = 2320$$

If original corpus length in characters is $|D|$, then after step 7 length is => $|D| - 2320$

b) At each step $t$:

- Choose the most frequent pair $m_t$.
- Merge all its occurrences -> the corpus becomes more and more "chunked" into bigger tokens.
- High frequency bigrams get eaten first; remaining bigrams become rarer.

Eventually, in a finite corpus:

- Most substrings appear once or not at all.
- So for almost every possible merge $m \in M$, the frequency is $f_t(m) = 1$

=> For sufficiently large $t$, distribution of frequencies collapses: almost all candidate merges have frequency 1, $f_t(m) \approx 1$ for most $m$.

c) i) Left to right greedy segmentation

Start from left and pick the longest matching token:

- Prefixes starting at position 0: "u", "un", "uni...", "union", "unioni...", "unioniza..."
- Token match prefix:
  - "un"
  - "union"
  - "unioniza"
- Longest is "unioniza". Remaining string: "tion" (in vocab)

=> So left to right segmentation: $T_{greedy}$ = (unioniza, tion)

ii) List all ways to cover "unionization" with given tokens:

1/ (union, i, z, ation)

2/ (union, i, z, a, tion)

3/ (union, i, z, at, ion)

4/ (un, ion, i, z, ation)

5/ (un, ion, i, z, a, tion)

6/ (un, ion, i, z, at, ion)

7/ (unioniza, tion)

iii) Probability of each segmentation

$p(T) = \prod_{t \in T} p(t)$

1/ $T_1 = (union, i, z, ation)$

$p(T_1) = 0.025 \cdot 0.05 \cdot 0.025 \cdot 0.025 = 7.8125 \times 10^{-7}$

2/ $T_2 = (union, i, z, a, tion)$

$p(T_2) = 0.025 \cdot 0.05 \cdot 0.025 \cdot 0.3 \cdot 0.05 = 4.6875 \times 10^{-7}$

3/ $T_3 = (union, i, z, at, ion)$

$p(T_3) = 0.025 \cdot 0.05 \cdot 0.025 \cdot 0.05 \cdot 0.025 = 3.90625 \times 10^{-8}$

4/ $T_4 = (un, ion, i, z, ation)$

$p(T_4) = 0.05 \cdot 0.025 \cdot 0.05 \cdot 0.025 \cdot 0.025 = 3.90625 \times 10^{-8}$

5/ $T_5 = (un, ion, i, z, a, tion)$

$p(T_5) = 0.05 \cdot 0.025 \cdot 0.05 \cdot 0.025 \cdot 0.3 \cdot 0.05 = 2.34375 \times 10^{-8}$

6/ $T_6 = (un, ion, i, z, at, ion)$

$p(T_6) = 0.05 \cdot 0.025 \cdot 0.05 \cdot 0.025 \cdot 0.05 \cdot 0.025 = 1.953125 \times 10^{-9}$

7/ $T_7 = (unioniza, tion)$

$p(T_7) = 0.05 \cdot 0.05 = 0.0025$

iv) Left to right segmentation is $T_{greedy} = (unioniza, tion) = T_7$ => its probability much larger than others

=> Left to right greedy segmentation is also the most probable under unigram LM.

vi) # of segmentations of $y$ efficiently:

Use dynamic programming over positions of the string

Let $n = |y|$, $count[i]$ = # of segmentations of suffix $y[i:]$

Recurrence:

- Base: $count[n] = 1$ (empty suffix).
- For $i = n-1, \dots, 0$

$$count[i] = \sum_{\substack{t \in \Sigma' \\ y[i:].startswith(t)}} count[i + |t|]$$

Answer: $count[0]$

Complexity: $O(n \cdot L_{max})$ where $L_{max}$ is max token length, assuming efficient "startswith" lookup

vii) Total probability of all segmentations of $y$:

Let: $P[i]$ = total probability of all segmentations of $y[i:]$

Then: - Base: $P[n] = 1$
\ For $i = n-1, \dots, 0$

$$\Rightarrow P[i] = \sum_{\substack{t \in \Sigma' \\ y[i:].startswith(t)}} p(t) \cdot P[i + |t|]$$

Then $P[0]$ is the probability mass of all segmentations of $y$ (forward algo on 1D chain WFSA).

vii) How would you compute the probability of all the possible segmentations of $y =$ `unionization` in an *efficient* manner? (**2 points**)

# 3 Generation from Autoregressive Models (18 pts)

Autoregressive models generate text iteratively one word at a time. At each time step, it follows the below outlined steps regardless of architecture:

1. Computes a score (logit) for each word in the vocabulary at the current timestep given all the previously generated words at earlier steps. Alternatively, stop generating if a special token [EOS] was generated at the previous step.
2. Obtains a distribution over the possible words at the current timestep.
3. Selects a word on the basis of this distribution with some defined strategy.
4. Appends the selected word to the previously generated words and repeats.

Suppose you have an already generated part of a sequence defined as: *I eat*

Now, you are generating the next word in the sequence.
The scores computed in step 1 of the 7 possible words from the vocabulary are:

$$[\text{I, eat, paper, broccoli, clean, ., [EOS]}] = [-1000, -5000, 500, 2000, 500, 450, 400]$$

You could tweak steps 2 and 3 to yield a wide variety of generation strategies. Answer the following:

(a) **2 pts** If you performed softmax over these scores and decided to perform ancestral sampling, what would be the ratio of probability of selecting *paper* over probability of selecting *broccoli*? Write one disadvantage of this strategy.

(b) **2 pts** If you decided to perform greedy decoding, what would the probability of you selecting *paper* as the next word? What word would you select? Write one disadvantage of this strategy.

(c) **2 pts** If you were doing temperature-based softmax with the temperature $T = 10^{20}$, what would be the probability of you selecting *paper* as the next word? Write one disadvantage of this strategy.

(d) **2 pts** If you were doing temperature-based softmax with the temperature $T = 10^{-100}$, what would be the probability of you selecting *paper* as the next word? Write one disadvantage of this strategy.

(e) **1 pt** Let's say you had a choice to pick one of these three temperature values: $[0.5, 1.0, 1.5]$. You wish to make the probability of selecting *paper* as small as possible. Which temperature value would you pick?

(f) **2 pts** Let's say you absolutely didn't wish to end the generation at this step by producing [EOS]. How would you adjust the temperature to achieve this goal. Write one disadvantage of this strategy.

(g) **2 pts** Let's say you absolutely didn't wish to end the generation at this step by producing [EOS]. But now, you wish to perform *top-k* sampling. What values of $k$ will satisfy your goal? Describe whether this strategy suffers from the same disadvantage as the one you described in the previous question.

(h) **2 pts** Let's say you absolutely didn't wish to end the generation at this step by producing [EOS]. But now, you wish to perform *nucleus* sampling. What values of $p$ will satisfy your goal? Describe how this strategy could be superior to the top-k sampling approach in general.

(i) **3 pts** Let's say you wish the words *paper, broccoli*, and, *clean* to be equally likely to be selected as the next word but the rest of the items in the vocabulary to absolutely not be selected. Using some combination of the strategies in the questions above, design a generation strategy that will achieve this purpose. Describe one disadvantage of adopting this strategy in general.

# 4 Un-normalized Generation from Autoregressive Models (20 pts)

In the previous sections, you noticed that softmax is an incredibly important function for generating from autoregressive models. However, softmax is a bit expensive. Think about exponentiation, sum, and division operations required for a large vocabulary. In this section, we will explore generation by avoiding softmax. In the following questions, unless explicitly specified, assume we are interested in distribution one would get via plain softmax ($T = 1.0$, no truncation of vocabulary as in top-k or nucleus sampling.)

(a) **2 pts** Can you perform greedy decoding without computing the softmax function? If yes, describe how. If no, describe why.

(b) **2 pts** Can you perform sampling without computing the softmax. If yes, describe how. If no, describe why. *Hint: Use the same justification as for greedy decoding above. Refer to the class material to find a method that turns token sampling operations into greedy-decoding like operations.*

(c) **2 pts** Can you perform temperature-based sampling without computing the softmax. If yes, describe how. If no, describe why.

(d) **2 pts** Can you perform top-k sampling without computing the softmax. If yes, describe how. If no, describe why.

(e) **2 pts** Can you perform nucleus sampling without computing the softmax. If yes, describe how. If no, describe why.

(f) **2 pts** Write down pseudocode for performing beam search with log-softmax scores at each step.

(g) **2 pts** Write down pseudocode for a similar algorithm that performs beam search without ever computing softmax (or log-softmax) scores (only uses logits) at each step.

(h) **6 pts** Would the two pseudocodes you wrote always yield the same output. Why/ Why not? Explain in detail with convincing examples/counter-examples and appropriate theoretical justification.

3/ a) Ratio $p(\text{paper}) / p(\text{broccoli})$ under softmax

Softmax: $p(w) = \frac{\exp(z_w)}{\sum_v \exp(z_v)}$

$\Rightarrow \frac{p(\text{paper})}{p(\text{broccoli})} = \frac{\exp(500)}{\exp(2000)} = \exp(-1500) \approx 0$

Con: Ancestral sampling draw exactly from distribution. When diversity is peaked:
↳ almost always sample top token => low diversity
↳ Still have a tiny but non zero chance of sampling very low probability tokens => occasionally produces weird outputs.

b) Greedy Decoding:

Greedy picks argmax logit => highest logit is $z_{\text{broccoli}} = 2000$ => choose broccoli

Under Greedy, $p_{\text{greedy}}(\text{paper}) = 0$

Con: Greedy decoding is deterministic and tend to produce generic, repetitive text, ignoring plausible, slightly lower probability options => lack diversity

c) Temperature $T = 10^{20}$

Temperature softmax: $p_T(w) \propto \exp(z_w/T)$

For huge $T = 10^{20}$, all $z_w/T \approx 0$, so $\exp(z_w/T) \approx 1$ for every token. The distribution becomes nearly uniform over the 7 words: $p_T(\text{paper}) \approx \frac{1}{7}$

Con: Model's learned preferences are almost erased, high quality and low quality words become likely probable leading to incoherent, noisy text.

d) Temperature $T = 10^{-100}$ (extremely small)

Temperature softmax: $p_T(w) \propto \exp(z_w/T)$

The largest logit (broccoli, 2000) dominates enormously. For "paper" & "broccoli": $\frac{p_T(\text{paper})}{p_T(\text{broccoli})} = \exp((500-2000)/T) = \exp(-1500/T) \approx 0$

$\Rightarrow p_T(\text{paper}) \approx 0$

Con: Distribution becomes almost a hard argmax: single top token is always picked => diversity vanished => repetitive, deterministic outputs.

e) Choosing $T$ from $\{0.5, 1.0, 1.5\}$ to minimize $p_T(\text{paper})$

Lower temperature => distribution more peaked around highest logit
We know that broccoli has higher logit than paper, decreasing $T$ shifts probability mass from paper to broccoli

Hence, choose smallest $T$ to make $p_T(\text{paper})$ as small as possible => $T = 0.5$

f) Avoiding [EOS] with temperature

Since [EOS] has a lower logit (400) than some other tokens, decreasing $T$ concentrates probability on the highest logit tokens and decrease the chance of picking [EOS]

=> adjust temperature downwards to reduce $p_T([\text{EOS}])$

Con: Very low temperature makes distribution extremely peaked => diversity vanished => repetitive, deterministic outputs.

g) Avoiding [EOS] with top-k sampling

Sort token by logit in descending order:

1. "broccoli" (2000)
2. "paper" (500)
3. "clean" (500)
4. "." (450)
5. "[EOS]" (400)
6. "I" (-1000)
7. "eat" (-5000)

Top-k sampling keeps only the top k tokens and renormalizes

To exclude [EOS] from candidates, it must not be in top-k: for $k \geq 5$, [EOS] is not included.

$\Rightarrow k \in \{1, 2, 3, 4\}$

Not really same con as in f: < Top-k still allows randomness among remaining tokens, preserving some diversity
However, it hard masks all tokens outside top-k. => distort distribution in a different way

b) Avoiding [EOS] with nucleus sampling:

To avoid [EOS], nucleus must be formed entirely from token before [EOS] in the sorted list. Let these be $w_1, \dots w_r$ where $w_{r+1} =$ [EOS]. Then: $p_{thresh} \leq \sum_{i=1}^{r} p(w_i)$

In this case: $p_{thresh} \leq p(\text{broccoli}) + p(\text{paper}) + p(\text{clean}) + p(.)$

Why better than top-k: < top-p adapts nucleus size to the shape of distribution
gives more natural diversity than a fixed k

### 3(i) Making Only {paper, broccoli, clean} Equally Likely

We want the next-word distribution to satisfy

$$p(\text{paper}) = p(\text{broccoli}) = p(\text{clean}) = \tfrac{1}{3}, \qquad p(w) = 0 \text{ for all other tokens.}$$

A working strategy:

1. **Mask all other tokens**. Set their logits to $-\infty$, or equivalently apply top-$k$ with $k = 3$ and keep only {paper, broccoli, clean}.
2. **Apply a very large temperature** $T \to \infty$. Then for the remaining tokens:
$$\exp(z_w/T) \to 1, \qquad p_T(w) \to \tfrac{1}{3}.$$

Thus,
$$p(\text{paper}) = p(\text{broccoli}) = p(\text{clean}) = \tfrac{1}{3}.$$

A disadvantage is that this strategy strongly overrides the model's learned distribution, reducing coherence.

## 4 Un-normalized Generation from Autoregressive Models

### 4(a)

Greedy decoding selects

$$\arg\max_i p(i) = \arg\max_i \exp(z_i) = \arg\max_i z_i.$$

Thus greedy decoding can be done using logits only.

### 4(b)

Sampling can be done without softmax using the Gumbel–Max trick:

$$g_i = -\log(-\log U_i), \qquad U_i \sim \text{Uniform}(0, 1),$$

$$s_i = z_i + g_i, \qquad \hat{i} = \arg\max_i s_i.$$

This samples exactly from softmax($z$).

### 4(c)

Temperature sampling uses

$$p_T(i) \propto \exp(z_i/T).$$

Apply Gumbel–Max to $z_i/T$:

$$s_i = z_i/T + g_i.$$

Then $\arg\max_i s_i$ samples from $\mathrm{softmax}(z/T)$.

### 4(d)

Top-$k$ sampling:

1. Identify the $k$ largest logits.
2. Restrict to that set $S_k$.
3. Apply Gumbel–Max on $\{z_i : i \in S_k\}$.

### 4(e)

Nucleus sampling requires cumulative normalized probabilities:

$$\sum_{i \in S} \frac{\exp(z_i)}{\sum_j \exp(z_j)} \geq p.$$

Thus it requires computing (log-)softmax or an equivalent normalization. Therefore exact nucleus sampling cannot avoid softmax.

### 4(f)

Beam search with log-softmax scores:

$$\mathrm{score}_t(y_{1:t}) = \sum_{s=1}^{t} \log p(y_s \mid y_{<s}, x).$$

Extend every sequence with every token, accumulate log-probabilities, and keep the top $B$ sequences at each step.

## 4(g)

Beam search with logits only:

$$\log p(y_s) = z_s(y_s) - \log Z_s.$$

Since $\log Z_s$ is constant across all candidates at step $s$, we may instead use:

$$\text{score}'_t(y_{1:t}) = \sum_{s=1}^{t} z_s(y_s).$$

## 4(h)

For equal-length sequences:

$$\sum_{s=1}^{t} \log p(y_s) = \sum_{s=1}^{t} z_s(y_s) - \sum_{s=1}^{t} \log Z_s,$$

and the constant term cancels when comparing sequences. Thus both ranking methods are identical at each beam step.

The logits-only and log-prob beam searches always agree (standard setup).