Prof. Antoine Bosselut

Modern Natural Language Processing – CS-552
09.04.2025 from 11h30 to 13h00
Duration : 90 minutes

1

Midterm 2025 Solutions

SCIPER : **111111**

Do not turn the page before the start of the exam. This document is double-sided, has 16 pages, the last ones possibly blank. Do not unstaple.

- This is a closed book exam. Non-programmable calculators are allowed. No other electronic devices of any kind are allowed.
- Place on your desk: your student ID, writing utensils, one double-sided A4 page cheat sheet if you have one; place all other personal items below your desk.
- You each have a different exam.
- This exam has **multiple-choice** questions of varying difficulty. Each question is worth **one point**.
- Each question has **exactly one** correct answer. For each question, mark the box corresponding to the correct answer. You are not expected to get every question right even for the best grade.
- Only answers in this booklet count. No extra loose answer sheets. You can use the blank pages at the end as scrap paper.
- Use a **black or dark blue ballpen** and clearly erase with **correction fluid** if necessary.
- If a question turns out to be wrong or ambiguous, we may decide to nullify it.

Respectez les consignes suivantes | Observe this guidelines | Beachten Sie bitte die unten stehenden Richtlinien

choisir une réponse | select an answer Antwort auswählen	ne PAS choisir une réponse | NOT select an answer NICHT Antwort auswählen	Corriger une réponse | Correct an answer Antwort korrigieren

ce qu'il ne faut **PAS** faire | what should **NOT** be done | was man **NICHT** tun sollte

Question 1 Rotary Positional Embeddings (RoPE) are a type of positional encoding technique used in Transformer models, where the embedding vectors undergo a rotation operation based on token positions. RoPE encodes positional information by applying a rotation matrix to the query and key vectors in attention heads, effectively injecting relative positional information directly into the self-attention mechanism. Specifically, for position m, the query (q_m) and key (k_n) vectors are transformed as follows:

$$q'_m = f_{\text{rotary}}(q_m, m), \quad k'_n = f_{\text{rotary}}(k_n, n)$$

Which of the following statements correctly identifies a distinguishing property or advantage specific to RoPE positional embeddings?

- ■ Unlike absolute positional embeddings, RoPE explicitly encodes relative positional relationships by applying rotations directly within the attention computation, generalizing better to longer sequence lengths.
- ☐ Unlike sinusoidal positional embeddings, RoPE relies on discrete position indices and thus cannot handle sequences longer than those observed during training.
- ☐ RoPE embeddings exclusively use additive combinations of positional vectors, unlike absolute embeddings, which apply rotations to capture positional dependencies more accurately.
- ☐ RoPE embeddings require a fixed maximum sequence length due to their use of absolute rotation matrices, unlike relative positional embeddings, which scale naturally to longer sequences.

Solution: RoPE encodes relative positional information by rotating query and key vectors based on their positions, allowing the attention mechanism to capture token relationships more effectively and generalize better to longer sequences than absolute embeddings.

Question 2 Which of the following statements regarding Recurrent Neural Networks (RNNs) is **TRUE**?

- ☐ Standard RNNs have a computational complexity that scales quadratically with sequence length, making them impractical for long sequences.
- ☐ Standard RNNs inherently solve the vanishing gradient problem, making them ideal for modeling very long-range dependencies in practice.
- ■ Recurrent Neural Networks can, in theory, model dependencies of unbounded (arbitrary) context length because they recursively apply the same parameters at every timestep.
- ☐ Standard RNNs require positional embeddings to represent sequences effectively.
- ☐ Standard RNNs explicitly encode input tokens independently of each other without recursive state updates.

Solution: RNNs can theoretically model dependencies of any length by recursively applying the same parameters over time, allowing information to flow across the sequence. In practice, however, they struggle with long-range dependencies due to vanishing gradients.

Question 3 We have learned that dense word vectors learned through Word2Vec have many advantages over using sparse one-hot word vectors. Which of the following is **NOT** an advantage dense vectors have over sparse vectors?

- ■ Models using dense word vectors generalize better to unseen words than those using sparse vectors.
- ☐ Dense word vectors are easier to include as features in machine learning systems than sparse vectors.
- ☐ Models using dense word vectors generalize better to rare words than those using sparse vectors.
- ☐ Dense word vectors encode similarity between words while sparse vectors do not.

Solution: Word2Vec does not have representations for unseen words and hence it is not better than sparse representations in this regard.

Question 4 The Global Vectors for Word Representation (GloVe) model is an alternative to Word2Vec that constructs word embeddings by leveraging word co-occurrence statistics rather than predicting context words. The *co-occurrence probability matrix* P captures how often words appear together in a given corpus. It is constructed as follows:

- Let X_{ij} be the **raw co-occurrence count**, representing how many times word j appears in the **context window** of word i across the entire corpus.
- The **co-occurrence probability** P_{ij} is defined as:

$$P_{ij} = \frac{X_{ij}}{\sum_k X_{ik}}$$

  where $\sum_k X_{ik}$ is the total number of times word i appears with **any** word in the corpus.
- This probability represents *how likely word j appears in the context of word i.*

Instead of using a local window-based approach like Skip-gram or CBOW, GloVe directly factorizes the word co-occurrence matrix.

The GloVe model is trained by optimizing the following function:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{W} \sum_{j=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij})^2$$

where, W is the vocabulary size,
P_{ij} (a scalar) is the probability that word j appears in the context of word i,
$f : \mathbb{R} \to \mathbb{R}$ is a function that gives a weight to each (i, j) pair based on its probability P_{ij}, preventing rare and overly frequent co-occurrences from dominating the optimization process,
u_i is a column vector of shape $(d \times 1)$ representing the word vector for the word i, capturing the representation of the central word in the co-occurrence pair, and
v_j is a column vector of shape $(d \times 1)$ representing the word vector for context word j, capturing the representation of the word appearing in the context of word i.

Consider the following statements about GloVe:
A: The gradient $\frac{\partial J(\theta)}{\partial u_i}$ of the objective function is $\frac{\partial J(\theta)}{\partial u_i} = \sum_{i=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij}) v_j$
B: GloVe constructs word embeddings by factorizing a word co-occurrence matrix, while CBOW and Skip-gram learn embeddings by predicting words in contexts of unbounded lengths.
C: Word2Vec iteratively updates embeddings on local contexts, while GloVe can efficiently leverage global statistics once the co-occurrence matrix has been constructed.
D: GloVe and CBOW both predict a central word from surrounding words, while Skip-gram predicts surrounding words given a central word.

Which of the above statements are **TRUE**:

- [x] C only
- [ ] A only
- [ ] B and C
- [ ] None of the other options is correct
- [ ] All statements are true
- [ ] A, B and D

Solution: Statement A is false because the gradient is $\frac{\partial J(\theta)}{\partial u_i} = \sum_{j=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij}) v_j$. B is false because the contexts used in Word2Vec are not of unbounded lengths. C is correct by the definitions of GloVe and Word2Vec. D is false because GloVe does not predict central word from surrounding words.

Question 5 What is the primary purpose of backpropagation in neural networks?

- ☐ To calculate the forward pass of the network
- ☐ To determine the optimal network architecture
- ☐ To update the model parameters
- ☐ To randomly initialize weights
- ■ To efficiently compute gradients of the loss function with respect to weights

Solution: The primary purpose of backpropagation is to **efficiently compute the gradients** of the loss function with respect to the model's weights. These gradients are essential for updating the weights using optimization algorithms like gradient descent. While updating weights is the end goal, backpropagation itself only computes gradients; it does not perform the update.

Question 6 Which of the following statements correctly describes how ELMo implements bidirectional LSTMs to generate its contextualized embeddings?

- ☐ ELMo uses a single forward-direction LSTM layer followed by a separate backward-direction LSTM layer, concatenating their outputs only after the backward pass has processed the entire sequence.
- ■ ELMo independently runs two separate LSTM networks—one forward (from left to right) and one backward (right-to-left)—over the same input sequence. The final embeddings for each token are computed by concatenating hidden states from both directions at each position.
- ☐ ELMo employs a single LSTM layer that alternates direction every timestep, using hidden states from the previous forward step to inform the subsequent backward step, thus producing bidirectional embeddings sequentially rather than in parallel.
- ☐ ELMo uses transformer-style self-attention layers on top of forward-only LSTM layers, effectively simulating bidirectionality without explicitly implementing backward-direction recurrence.

Solution: ELMo uses two separate LSTM networks: one processes the sequence forward (left to right), and the other backward (right to left). At each position, the hidden states from both directions are concatenated to produce contextualized, bidirectional embeddings.

Question 7 Can decoder-only models (e.g., GPT-family) perform sequence-to-sequence tasks such as machine translation?

- ☐ Decoder-only models are inherently unable to perform any form of machine translation because it does not provide a direct way to encode the source sequence independently of the target sequence.
- ■ Decoder-only models can be used for machine translation by providing the source sequence as a prompt (possibly with special tokens), and then autoregressively generating the translated text.
- ☐ It is impossible to adapt decoder-only models for translation because they lack the cross-attention mechanism that directly aligns source and target sequences during inference.
- ☐ Decoder-only models performs machine translation by constructing an additional encoder module at inference time, allowing it to process source sequences separately from decoding.

Solution: Decoder-only models like GPT can perform machine translation by treating the source sequence as part of the prompt and generating the target text autoregressively. This approach works despite the absence of a separate encoder or cross-attention.

Question 8 Both **top-p (nucleus) sampling** and **top-k sampling** control randomness in language model outputs. Which of the following best describes their difference?

- [x] Top-k sampling always selects the top k most probable tokens, whereas Top-p sampling dynamically adjusts the number of tokens based on cumulative probability.
- [ ] Top-p sampling selects exactly p percent of tokens, while top-k always selects exactly k tokens.
- [ ] Top-k sampling is more dynamic than top-p because it considers the most relevant tokens at every step.
- [ ] Top-p sampling is deterministic, whereas Top-k is probabilistic.

Solution: Top-p adjusts dynamically, while top-k uses a fixed number of tokens. Correct option follows from the definitions of top-p and top-k sampling algorithms.

Question 9 Which of the following accurately describes a key disadvantage of using recurrent neural networks (RNNs) for modeling long sequences?

- [x] RNNs compress all historical context into a single fixed-size state vector, causing information loss.
- [ ] RNNs cannot learn nonlinear representations because their hidden states must remain linear combinations of past inputs.
- [ ] RNNs always have quadratic computational complexity with respect to sequence length, making them impractical for long sequences.
- [ ] RNNs require extremely large datasets for training because each timestep uses different parameter sets.
- [ ] RNNs inherently require positional embeddings, significantly increasing model complexity.

Solution: RNNs compress all historical context into a single fixed-size state vector, which can lead to information loss, especially over long sequences. This compression limits the network's ability to retain detailed or distant past information, making it harder to capture long-range dependencies effectively.

Question 10 Peephole connections in Long Short-Term Memory (LSTM) networks modify the standard gating mechanism by allowing gates to directly utilize information from the cell state c_{t-1}. Specifically, peephole connections introduce additional parameters—called peephole weights—that explicitly connect the gates to the previous cell state.

Given the above description, which of the following equations correctly represents the calculation of the input gate i_t in an LSTM cell with peephole connections?

- x_t: input vector at timestep t
- h_{t-1}: hidden state vector at timestep $t-1$
- c_{t-1}: cell state vector at timestep $t-1$
- W_{ix}, W_{ih}, W_{ic}: learned weight matrices/vectors
- b_i: bias vector
- σ: sigmoid activation
- $\odot$: element-wise multiplication

- [ ] $i_t = \tanh(W_{ix}x_t + W_{ih}h_{t-1} + W_{ic}c_{t-1} + b_i)$
- [ ] $i_t = \sigma(W_{ix}x_t + W_{ih}h_t + W_{ic}c_t + b_i)$
- [x] $i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1} + W_{ic}c_{t-1} + b_i)$
- [ ] $i_t = \sigma(W_{ix}x_t + W_{ic} \odot h_{t-1} + b_i)$
- [ ] $i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1} + b_i)$

Solution: The correct equation for the input gate in a peephole LSTM is $i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1} + W_{ic}c_{t-1} + b_i)$ because, unlike standard LSTMs, peephole connections allow gates to directly access the previous cell state c_{t-1} through additional learnable weights W_{ic}. This enhances the gating mechanism by enabling more precise control over information flow based on the cell's prior memory, while still using the sigmoid activation to determine how much new information should enter the cell.

Question 11 Consider the following statements in the context of evaluating the quality of text generation:
A: A perplexity of k can be interpreted as the model being confused among k tokens in the vocabulary on average.
B: A perplexity of k can be interpreted as the model being confused among e^k tokens in the vocabulary on average.
C: BLEU is a better metric than perplexity for evaluating text generation quality because it compares generated text to references.
D: BLEU is a good metric for evaluating creative text generation quality because it has a very high positive correlation with human evaluation.
E: Perplexity is a better indicator of generated text quality than BLEU.
F: SPIDEr evaluates text quality more holistically compared to BLEU as it focuses on both semantic and syntactic similarity.

Which of the above statements are **TRUE**:

- ☐ B and D
- ☐ None of the other options is correct
- ☐ E and F
- ☐ A, E and F
- ☒ A, C and F
- ☐ A only
- ☐ A, D and F
- ☐ B, C and F
- ☐ F only
- ☐ A and C
- ☐ B and C

Solution: Statement A is true by the definition of perplexity. Hence B is false. C is true since perplexity doesn't use any references to compare with. Since BLEU is better than perplexity, E is false. D is false since BLEU is not a good metric for evaluating creative text generation quality. F is true by the definition of SPIDEr metric.

Question 12 In Transformer models, the attention mechanism explicitly defines three distinct vector types:

- **Query**: Represents the element for which context is being retrieved (usually from the decoder).
- **Key**: Encodes representations used to calculate relevance to each query.
- **Value**: Encodes the actual content to be combined and passed forward according to attention weights.

Consider the original attention mechanism used in encoder-decoder LSTM models (e.g., Bahdanau attention). Although not explicitly labeled as Query, Key, and Value, these concepts have analogous interpretations. Which of the following correctly describes the analogy between Transformers' Query-Key-Value framework and the attention mechanism in encoder-decoder LSTM models?

■
- **Query:** Decoder hidden state at the current decoding step (s_t).
- **Keys:** Encoder hidden states (h_i).
- **Values:** Encoder hidden states (h_i).

☐
- **Query:** Encoder hidden states (h_i).
- **Keys:** Decoder hidden state at the previous decoding step (s_{t-1}).
- **Values:** Decoder hidden states at previous steps ($s_{1:t-1}$).

☐
- **Query:** Input embeddings (x_i) of encoder tokens.
- **Keys:** Encoder hidden states (h_i).
- **Values:** Decoder hidden states (s_{t-1}).

☐
- **Query:** Attention weights ($\alpha_{t,i}$).
- **Keys:** Decoder hidden state at previous decoding steps (s_{t-1}).
- **Values:** Current input embedding (x_t).

Solution: The correct analogy is: Query = decoder hidden state at the current step (s_t), Keys = encoder hidden states (h_i), and Values = encoder hidden states (h_i). In encoder-decoder LSTM models with attention (like Bahdanau attention), the decoder uses its current hidden state as a query to attend over all encoder hidden states, which serve both as keys (for computing attention weights) and values (the content to be aggregated). This mirrors the Transformer's attention mechanism conceptually, even though the operations are implemented differently.

Question 13 You have access to:

(a) A **very large** unlabeled text corpus (hundreds of gigabytes) covering general topics.

(b) A **small, labeled** dataset (a few thousand examples) focused on a **specific domain** (e.g., medical records).

You want to build an NLP model for domain-specific classification (e.g., classifying patient notes). Which of the following strategies best reflects effective use of transfer learning?

☐ Train an LSTM or Transformer from scratch on the small labeled dataset alone.

■ Pretrain a large model on the massive general corpus, then finetune on the smaller domain-specific labeled dataset to adapt the model to your classification task.

☐ Perform no finetuning, but rely exclusively on hyperparameter search over the small dataset to compensate for domain mismatch.

☐ Pretrain a large model on the massive general corpus, then immediately deploy it to your task without any additional training.

Solution: The best approach is to pretrain a large model on the general corpus to learn broad language patterns, then finetune it on the smaller, domain-specific labeled dataset. This leverages the strengths of both datasets—general knowledge from pretraining and task-specific adaptation from finetuning.

Question 14 Word2Vec represents a family of embedding algorithms that are commonly used in a variety of contexts. Suppose in a recommender system for online shopping, we have information about co-purchase records for items $x_1, x_2, \ldots, x_n$ (for example, item x_i is commonly bought together with item x_j). You are tasked to use ideas similar to Word2Vec to recommend similar items to users who have shown interest in any one of the items. Which one of the following would **NOT** be a part of your strategy?

- [ ] I will make item recommendations by finding items with high cosine similarity to the average of item embeddings in the purchase basket.
- [ ] I will build item embeddings by minimizing the negative log probability of the missing item based on co-purchase records.
- [x] I will construct item embeddings such that each item is represented as the average of the embeddings of all items it was co-purchased with.
- [ ] I will treat items that are co-purchased with item x_i to be in the 'context' of item x_i.

Solution: If we were using ideas similar to Word2Vec, we would learn embeddings through an optimization process rather than representing each item as the average of embeddings of all items it is purchased with. All other options indicate valid steps comprising the recommendation strategy inspired by Word2Vec.

Question 15 In a Transformer encoder-decoder model, each decoder block utilizes masked self-attention followed by encoder-decoder cross-attention. Which of the following correctly explains the role of masked self-attention in the decoder?

- [ ] It allows the model to consider future tokens to improve prediction accuracy.
- [ ] It prevents tokens from attending to themselves to avoid trivial solutions.
- [ ] It allows the decoder layer to attend exclusively to encoder representations, discarding decoder self-context.
- [x] It ensures the decoder does not attend to subsequent tokens during training to maintain autoregressive property.

Solution: Masked self-attention ensures the decoder does not attend to subsequent tokens during training to maintain the autoregressive property. In sequence generation tasks, the decoder must generate one token at a time, relying only on previously generated tokens. The mask prevents the model from "seeing the future" by blocking attention to future positions, ensuring that predictions are made based solely on past and current context, just as they would be during inference.

Question 16 Which of the following sets consists **solely of bidirectional** Transformer models?

- [x] BERT, DistilBERT, RoBERTa
- [ ] BART, T5, GPT
- [ ] GPT, BART, BERT
- [ ] GPT, BERT, T5

Solution: BERT, DistilBERT, and RoBERTa are all bidirectional Transformer models—they process input by attending to both left and right context.

Question 17 Why are positional embeddings required in Transformer architectures?

- [ ] To improve the numerical stability of self-attention computations.
- [x] Because Transformers lack explicit sequential order information.
- [ ] To stabilize gradient updates during training.
- [ ] To prevent attention mechanisms from attending uniformly to all tokens.

Solution: Positional embeddings are required because Transformers lack explicit sequential order information. Unlike RNNs, Transformers process all tokens in parallel and do not have a built-in notion of token order. Positional embeddings inject information about the position of each token in the sequence, allowing the model to capture the order-dependent structure of language.

Question 18 Consider the following statements regarding fine-tuning and prompt tuning in large language models:
A: Fine-tuning modifies the model's parameters, while prompt tuning keeps them fixed but optimizes a soft prompt.
B: LoRA is an efficient prompt tuning method where we freeze the model's parameters and only train newly initialized FFN layers.
C: Fine-tuning requires more task-specific labeled data whereas prompt tuning requires fewer examples to modify the input prompt representation.
D: Prompt tuning always performs better than fine-tuning because it does not require any labeled data.
E: Prompt tuning performs on par with fine-tuning at all model scales.
F: Prompt tuning is more parameter-efficient than fine-tuning because it only optimizes a small number of prompt embeddings instead of updating all model parameters.
G: Soft prompts are inherently more interpretable.

Which of the above statements are **TRUE**:

- ☐ A, F and G
- ☐ A only
- ☒ A, C and F
- ☐ A and F
- ☐ A and G
- ☐ All statements are true
- ☐ All except D, E and G
- ☐ A, B, C, F and G

Solution: Statements A, C and F are true because: fine-tuning updates the model's parameters, requiring more data and computational resources. Prompt tuning optimizes only a small set of prompt embeddings, making it more parameter-efficient and effective for low-data scenarios. B, D, E and G are false because: LoRA is for fine-tuning and not prompt tuning; prompt tuning does require labeled data; it only performs on par with fine-tuning at large model scales; soft prompts are non-interpretable.

Text Generation

The following tree of tokens in Figure 1 is relevant for the next few questions. The boxes represent tokens. Each arrow $a \xrightarrow{q} b$ represents the choice of next token being b given the token prefix ending at a. The probability of b given prefix ending at a is q. Only the highest probability tokens for each prefix are shown.

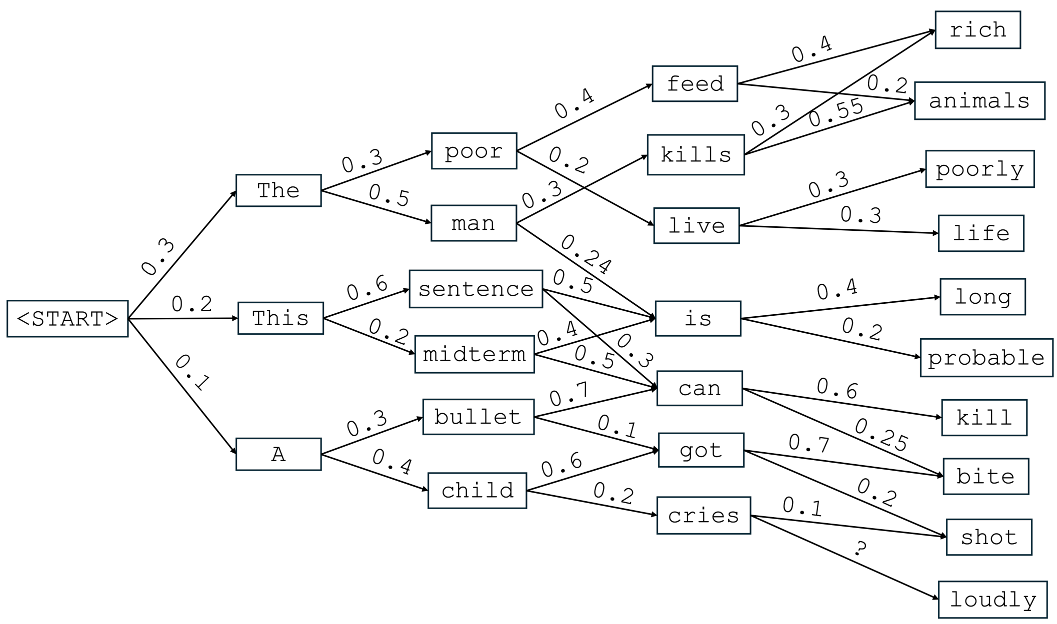

Figure 1. Tree of tokens and probabilities.

Question 19 Consider the following sequences:
A: `The man kills rich`
B: `This midterm can kill`
C: `This sentence is probable`
D: `A child cries loudly`
E: `This midterm is long`
F: `A bullet got bite`
G: `A child got bite`
H: `A bullet got shot`
I: `This sentence can bite`
J: `A child got shot`

Which of the above sequences will never be generated if top-p (nucleus) sampling is used for decoding with $p = 0.52$?

- [ ] All except A, F, I and J
- [ ] A, B, D, E, H, I and J
- [ ] All except C, G and I
- [ ] C and G
- [ ] D only
- [ ] None of the other options is correct
- [ ] B, D, E and H
- [x] All except C and G
- [ ] A, B, D, E, F, G, H, I and J
- [ ] A, F, I and J

Solution: In top-p (nucleus) sampling, at each step, we sample from the smallest set of tokens whose cumulative probability $\geq p$. Any token outside this set cannot be selected, so any sequence that includes such a token at any step can never be generated. At the first step, all three tokens `The`, `This` and `A` are considered given the cumulative probability of 0.6. Assuming `The` is sampled at the first step, both `poor` and `man` are candidates since we need both for cumulative probability to equal or exceed 0.52. Assuming `This` is sampled at the first step, `midterm` (and any branches emanating from it) will never be considered since `sentence` alone suffices the cumulative probability criteria. Following this line of reasoning, we can prune the tree and conclude that only the sequences C and G can be generated.

Question 20 What are the probabilities assigned to the following three sequences by top-k sampling with $k = 2$:

```
This midterm can kill
A bullet got bite
The poor feed animals
```

- [x] $(\frac{2}{51}, 0, \frac{1}{20})$
- [ ] $(\frac{2}{51}, 0, 0)$
- [ ] $(0, 0, \frac{1}{20})$
- [ ] $(\frac{5}{153}, 0, \frac{1}{24})$
- [ ] $(0.012, 0, 0.0072)$
- [ ] $(\frac{5}{153}, \frac{1}{144}, \frac{1}{24})$

Solution: At each step in top-k sampling, we select top k tokens and then normalize their probabilities. Doing so, we get that the probability assigned by top-2 sampling to `This midterm can kill` is $2/5 * 2/8 * 5/9 * 60/85 = 2/51$. `A bullet got bite` cannot be produced as it gets cut off in the very first step, so it is assigned a probability of 0. `The poor feed animals` gets assigned the probability $3/5 * 3/8 * 4/6 * 2/6 = 1/20$.

Question 21 If the probability assigned to the sequence `"A child cries loudly"` by top-p (nucleus) sampling with $p = 0.62$ is $\frac{1}{56}$, then what is the value of probability represented by `"?"` in the tree?

- [ ] $e/9$
- [x] 0.3
- [ ] 0.033
- [ ] $\pi/10$
- [ ] 0.2232
- [ ] $\pi/10e$
- [ ] 0.012

Solution: At each step in nucleus sampling, we select the tokens based on top-p cumulative probability and then normalize them. Doing so, we get that the probability assigned by top-0.62 (nucleus) sampling to `"A"` is $0.1/(0.1 + 0.2 + 0.3) = 1/6$. Proceeding similarly, we get that the probability assigned to the indicated sequence is $1/6 \ * \ 4/7 \ * \ 2/8 * ?/(0.1 + ?)$. We can equate this to the provided probability and solve for `"?"` which gives 0.3 as the answer.

Code Comprehension: Sampling

You are given the following top-p (nucleus) sampling function:

```python
def top_p_sampling(probs, p=0.9):
    sorted_indices = np.argsort(probs)[::-1]
    sorted_probs = probs[sorted_indices]
    cumulative_probs = np.cumsum(sorted_probs)
    cutoff_idx = np.searchsorted(cumulative_probs, p)
    top_p_indices = sorted_indices[:cutoff_idx]
```

```
7        top_p_probs = sorted_probs[:cutoff_idx]
8        top_p_probs /= np.sum(top_p_probs)
9        return np.random.choice(top_p_indices, p=top_p_probs)
```

Here are short descriptions of the NumPy functions involved:

- `np.argsort(probs)`: Returns the indices that would sort the array in ascending order. Example: `np.argsort([0.2, 0.8, 0.5])` returns `[0, 2, 1]`.
- `np.cumsum(sorted_probs)`: Computes the cumulative sum of an array. Example: `np.cumsum([0.8, 0.15, 0.05])` returns `[0.8, 0.95, 1.0]`.
- `np.searchsorted(cumulative_probs, p)`: Finds the smallest index where cumulative probability reaches or exceeds p. Example: `np.searchsorted([0.6, 0.85, 1.0], 0.75)` returns `1`.
- `np.random.choice(top_p_indices, p=top_p_probs)`: Samples an element from `top_p_indices` with probabilities given by `top_p_probs`.
  Example: `np.random.choice([2, 0], p=[0.8, 0.2])` randomly returns `2` or `0` with `2` being more likely.

Question 22 We want to modify the above function to implement **top-k sampling**, where the model always considers the k most probable tokens. Assume that line `1` is changed to `def top_k_sampling (probs, k=5, p=0.9):` A group of students came up with the following two modifications in this regard:

A: Insert `p = get_cum_prob_for_top_k(cumulative_probs, k)` between lines `4` and `5`. Assume that this new method correctly computes the cumulative probability corresponding to top-k tokens.
B: Replace line `5` by `cutoff_idx = k`.

However, the students could not reach a consensus on which of these will yield a top-k sampler. What do you think?

- ■ A and B
- ☐ A only
- ☐ Neither A nor B
- ■ B only

Solution: The code contains an off-by-one index error in line 5, where 1 must be added to the index to fix it. The method in modification **A** dynamically finds the cumulative probability that is needed to do top-k sampling. But with the bug in place, **A** will implement a top-$(k-1)$ sampler instead. Modification **B** explicitly selects top k tokens. So, **"B only"** is technically the correct answer.
However, given that the bug is not explicitly stated, it is fair to assume that the code is a correct top-p sampler. So, for grading, we considered **"A and B"** and **"B only"** both as correct answers.

N-gram Language Modeling

Given a training set containing the following sequences:
`"the cat sat on the mat"`
`"the dog chased after the cat"`
`"a cat and a dog ran"`
`"she sat on the sofa"`
`"they bought a new mat"`
`"the sofa was comfortable"`

The training set includes 18 unique tokens and 32 tokens in total. We consider the words split by whitespace as the tokens in the vocabulary. We don't consider adding any special tokens (`<UNK>` or `<stop>`) without specifically claiming. Please answer the following questions in the context of N-gram language modeling:

Question 23 Out-of-vocabulary (OOV) words are words that *have not appeared* in the training set, or have appeared with *very low frequency* in the training set such that we do not include them in the vocabulary. We here consider all tokens appearing **less than or equal to 1 time** in the training data set as an OOV token.

Assume you implement a unigram model with special tokens `<stop>` (end of sequence), and `<UNK>` for out-of-vocabulary words. `<stop>` is now appended to all of the original sequences in the training set. What is the probability of the sequence `"the rabbit ran <stop>"`? Do not apply smoothing in this question.

- ☐ $\frac{256}{38^4}$.
- ■ $\frac{3600}{38^4}$.
- ☐ None of the other options is correct.
- ☐ $\frac{3600}{32^4}$.

Solution: $\frac{3600}{38^4}$ is correct.
After adding special tokens into the training set, we have $P(\texttt{<UNK>}) = \frac{10}{38}$, $P(\texttt{<stop>}) = \frac{6}{38}$. $P(\text{the}) \times P(\texttt{<UNK>}) \times P(\texttt{<UNK>}) \times P(\texttt{<stop>}) = \frac{6}{38} \times \frac{10}{38} \times \frac{10}{38} \times \frac{6}{38} = \frac{3600}{38^4}$.

Question 24 Which of the following sequences will have the **highest** perplexity according to the **unigram** model with add-one Laplace smoothing and special tokens (`<UNK>` and `<stop>`)?
For this question, consider only the words that have not appeared in the training set as OOV.

- ☐ `"the cat chased after the rabbit <stop>"`
- ☐ `"the cat chased after the dog <stop>"`
- ☐ `"the rabbit ran <stop>"`
- ■ `"the rabbit ran very very fast <stop>"`

Solution: `"the rabbit ran very very fast <stop>"` is correct.
The sequence with the lowest **average** probability would have the highest perplexity.

Question 25 To avoid the OOV problem, we apply the add-one Laplace smoothing, where the probability of each bigram can be computed as:

$$P(\mathrm{A}|\mathrm{B}) = \frac{\mathrm{count}(\mathrm{B}, \mathrm{A}) + \alpha}{\mathrm{count}(\mathrm{B}) + \alpha \cdot |V|}$$

After applying add-one Laplace smoothing ($\alpha = 1$) to the bigram model, what are the probabilities of $P(\text{sat}|\text{on})$ and $P(\text{on}|\text{sat})$?

- ☐ None of the other options is correct.
- ■ $P(\text{sat}|\text{on}) = \frac{1}{20}$, $P(\text{on}|\text{sat}) = \frac{3}{20}$.
- ☐ $P(\text{sat}|\text{on}) = \frac{1}{2}$, $P(\text{on}|\text{sat}) = \frac{1}{2}$.
- ☐ $P(\text{sat}|\text{on}) = \frac{3}{20}$, $P(\text{on}|\text{sat}) = \frac{1}{20}$.
- ☐ $P(\text{sat}|\text{on}) = 0$, $P(\text{on}|\text{sat}) = 1$.

Solution: $\frac{1}{20}$, $\frac{3}{20}$ is correct.
$P(\text{sat}|\text{on}) = \frac{\text{count(on,sat)}+\alpha}{\text{count(on)}+\alpha\cdot|V|} = \frac{0+1}{2+1\times 18} = \frac{1}{20}$.
$P(\text{on}|\text{sat}) = \frac{\text{count(sat,on)}+\alpha}{\text{count(sat)}+\alpha\cdot|V|} = \frac{2+1}{2+1\times 18} = \frac{3}{20}$.

Question 26 What is the probability of the sequence "the dog chased after the new cat" using a bigram (2-gram) language model, without smoothing?

- [x] 0.
- [ ] $\frac{6}{32\times 26^6}$.
- [ ] $\frac{6}{32\times 26^6} \times \frac{32}{6} \times \frac{32}{2} \times \frac{32}{1} \times \frac{32}{1} \times \frac{32}{6} \times \frac{32}{1} \times \frac{32}{3}$.
- [ ] None of the other options is correct.

Solution: 0 is correct.
`(new, cat)` has not appeared in the training dataset, so the probability is zero without smoothing.

Classification and Dataset Bias

Consider a training set for sentiment classification containing the following sequences:

```
"I liked this love movie because it was exciting" [LABEL=+]
"action movies are always terrible" [LABEL=-]
"this comedy made me laugh" [LABEL=+]
"I disliked the drama because it was too slow" [LABEL=-]
```

We are now conducting sentiment classification by **Naive Bayes** algorithm.

Naive Bayes finds the probability of a label given the probability of the sequences occurred in the training set. Mathematically, it can be defined as:

$$P(c|X) = \frac{P(X|c)P(c)}{P(X)}, \qquad P(X) = \sum_{c\in\{+,-\}} P(X|c)P(c)$$

where c is one of the labels and X is the corresponding sequence.
The training dataset contains 28 tokens in total, including 23 unique tokens. We add one extra token "<UNK>" into vocabulary to handle the unseen out-of-vocabulary words. We always apply add-one Laplace smoothing (smoothing factor $\alpha = 1$) to handle zero probabilities. The conditional probability of a single word w given label c can be computed as follows:

$$P(w|c) = \frac{\text{count}(w, c) + \alpha}{\sum_{w_i\in V} \text{count}(w_i, c) + \alpha \cdot |V|}$$

Question 27 Which of the following can reduce the bias against action movies in our classifier?

(a) Remove all instances of "action" from the training data

(b) Add more negative examples of non-action movies

(c) Add positive examples containing "action movie"

- [ ] (b) and (c).
- [ ] Only (c).
- [x] (a) and (c).
- [ ] None of the other options is correct.
- [ ] Only (a).
- [ ] (a) and (b).

Solution: (a) and (c) are correct.

- (a): removing all instances will lead to $P(+|\texttt{action}) = P(-|\texttt{action})$;

- (b): adding other non-action words would not affect the probabilities of $P(+|\texttt{action})$ and $P(-|\texttt{action})$;
- (c): adding positive examples containing "action movie" would increase the $P(+|\texttt{action})$ thus reduce the bias towards negative.

Question 28 Consider all the words occurred in positive and negative classes from our training data, which word shows the strongest bias toward negative sentiment?

- [ ] `"movie"`
- [ ] `"horrible"`
- [ ] `"I"`
- [ ] `"was"`
- [x] `"action"`

Solution: We compute $P(-|\text{movie})$ using Naive Bayes and unigram language modeling:

$$\begin{aligned}
P(-|X) &= P(X|-)P(-)/P(X) \\
&= P(\text{movie}|-)P(-)/P(\text{movie}) \\
&= \frac{0+1}{14+24} \times \frac{1}{2} \times \frac{1}{P(\text{movie})} \\
&= \frac{1}{14+24} \times \frac{1}{2} \times \frac{1}{P(\text{movie})}
\end{aligned}$$

we can then derive $P(X)$ as:

$$\begin{aligned}
P(X) &= P(X|+) + P(X|-) \\
&= \frac{1+1}{14+24} \times \frac{1}{2} + \frac{0+1}{14+24} \times \frac{1}{2} \\
&= \frac{3}{14+24} \times \frac{1}{2}
\end{aligned}$$

Therefore, $P(-|\text{movie}) = \frac{1}{3}$.

Following the same computation, we compute the probability of each word to be negative:

$$\begin{aligned}
&P(-|\text{movie}) = \frac{1}{3} \\
&P(-|\text{action}) = \frac{2}{3} \\
&P(-|\text{I}) = \frac{1}{2} \\
&P(-|\text{was}) = \frac{1}{2} \\
&P(-|\text{horrible}) = \frac{1}{2}
\end{aligned}$$

Thus the correct answer is `"action"`.

Question 29 The sequence `"this movie was fantastic but it was an action movie"` has conflicting sentiment signals. Using Naive Bayes with our training data, what is the most likely classification?

- [ ] Negative with 63% confidence
- [ ] Positive with 51% confidence
- [ ] Negative with 67% confidence
- [x] Positive with 86% confidence
- [ ] Negative with 73% confidence

Solution:

$$\begin{aligned}
P(+|X) &= P(X|+)P(+)/P(X) \\
&= P(X|+)/(P(X|+) + P(X|-)) \\
&= (\texttt{count}(\text{this}) + 1|\texttt{+}) \times (\texttt{count}(\text{movie}) + 1|\texttt{+}) \ldots (\texttt{count}(\text{movie}) + 1|\texttt{+}) \\
&/((\texttt{count}(\text{this}) + 1|\texttt{+}) \times (\texttt{count}(\text{movie}) + 1|\texttt{+}) \ldots (\texttt{count}(\text{movie}) + 1|\texttt{+}) \\
&+ (\texttt{count}(\text{this}) + 1|\texttt{-}) \times (\texttt{count}(\text{movie}) + 1|\texttt{-}) \ldots (\texttt{count}(\text{movie}) + 1|\texttt{-})) \\
&= \frac{3 \times 2 \times 2 \times 2 \times 2 \times 2}{(3 \times 2 \times 2 \times 2 \times 2 \times 2 + 2 \times 2 \times 2 \times 2)} \\
&= 0.86
\end{aligned}$$

Question 30 Which of the following sequences would have the lowest probability of being classified as positive?

- ☐ `"this drama was fantastic"`
- ☐ `"I enjoyed this action movie"`
- ☐ `"I like this documentary"`
- ■ `"the movie was cool but slow"`

Solution: We compute $P(+|\text{"I enjoyed this action movie"})$ using Naive Bayes and unigram language modeling:

$$\begin{aligned}
P(+|X) &= P(X|+)P(+)/P(X) \\
&= P(\text{"I"}|+)P(\text{"enjoyed"}|+, \text{"I"}) \ldots P(\text{"movie"}|+, \text{"I"}, \text{"enjoyed"}, \text{"this"}, \text{"action"})P(+)/P(X) \\
&= P(\text{"I"}|+)P(\text{"enjoyed"}|+)P(\text{"this"}|+)P(\text{"action"}|+)P(\text{"movie"}|+)P(+)/P(X) \\
&= \frac{1+1}{14+24} \times \frac{0+1}{14+24} \times \frac{2+1}{14+24} \times \frac{0+1}{14+24} \times \frac{1+1}{14+24} \times \frac{1}{2} \times \frac{1}{P(X)} \\
&= \frac{12}{14+24} \times \frac{1}{2} \times \frac{1}{P(X)}
\end{aligned}$$

we can then derive $P(X)$ as:

$$\begin{aligned}
P(X) &= P(X|+) + P(X|-) \\
&= \frac{1+1}{14+24} \times \frac{0+1}{14+24} \times \frac{2+1}{14+24} \times \frac{0+1}{14+24} \times \frac{1+1}{14+24} \times \frac{1}{2} \\
&+ \frac{1+1}{14+24} \times \frac{0+1}{14+24} \times \frac{0+1}{14+24} \times \frac{1+1}{14+24} \times \frac{0+1}{14+24} \times \frac{1}{2} \\
&= \frac{16}{14+24} \times \frac{1}{2}
\end{aligned}$$

Then we finally get:

$$\begin{aligned}
P(+|X) &= (\frac{12}{14+24} \times \frac{1}{2})/(\frac{16}{14+24} \times \frac{1}{2}) \\
&= 0.75
\end{aligned}$$

Following the same computation, the probabilities of each sequence to be classified as positive are:

$$\begin{aligned}
&P(+|\text{I like this documentary}) = 0.75 \\
&P(+|\text{I enjoyed this action movie}) = 0.75 \\
&P(+|\text{this drama was fantastic}) = 0.60 \\
&P(+|\text{the movie was cool but slow}) = 0.33
\end{aligned}$$

Thus the correct answer is `"the movie was cool but slow"`.