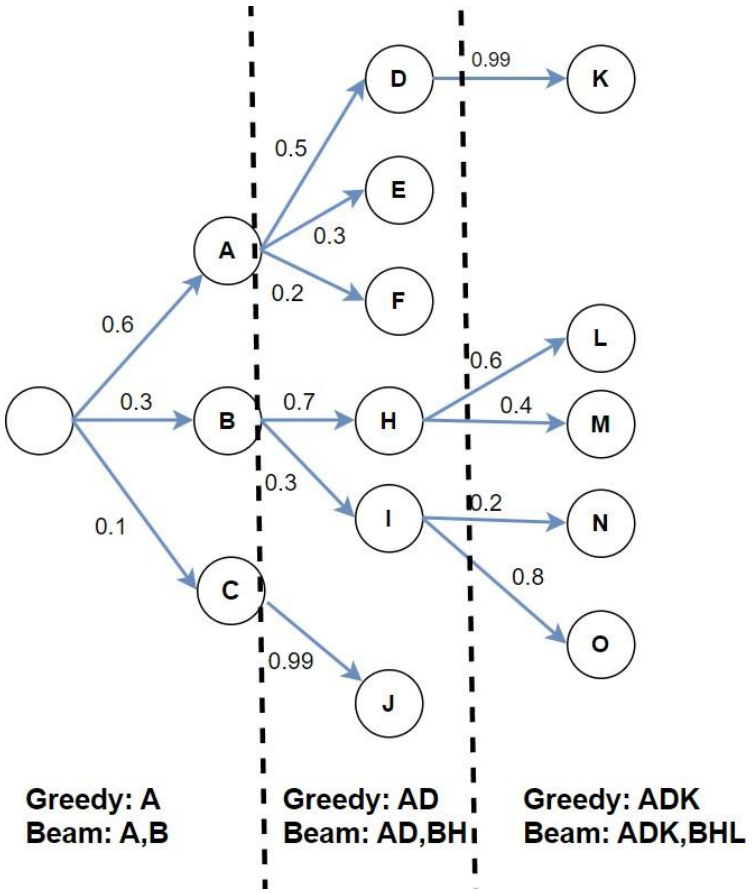
Question 1

1.

[4,7,9,15,23,18]
[2,8,10,0,0,0]
[3,13,2,8,0,0]

2. After determining a batch size, train according to the batch length from large to small.

3.



4.

P1=11/12≈0.917, P2=9/11≈0.818

P3=8/10=0.8, P4=7/9≈0.778

BP=1

BLEU=0.8266

Question 2

1. Limitations:

   - Information Bottleneck: The entire source sequence is compressed into a fixed-length vector, which may lose important semantic information, especially for long sentences.

   - Difficulty in Capturing Long-Term Dependencies: Due to the sequential nature of RNNs and vanishing gradient problems, it's hard to model long-distance relationships effectively.

   - Inefficient Computation: RNNs process tokens sequentially, preventing parallelization during training and inference.

   How attention addresses these limitations:

   - Direct Access to All Encoder States: Attention allows the decoder to access all encoder outputs at each step instead of relying on a single context vector.
   - Better Long-Range Dependency Modeling: By computing weighted sums of all source tokens, attention captures more relevant context across the entire sequence.
   - Parallel Computation: Attention can be implemented using matrix operations, which are parallelizable and GPU-friendly.

   Roles of Q, K, V in attention:

   - Query (Q): Represents the current decoding step.
   - Key (K): Represents positions in the input sequence.
   - Value (V): Encodes the content to be aggregated based on similarity between Q and K.

   Why separate matrices for Q and K?

   - Using different weight matrices for Q and K allows the model to learn asymmetric relationships between query and key representations. Using the same matrix would limit expressiveness and hinder learning.

2. Role of the Scaling Factor $\frac{1}{\sqrt{d_k}}$ and Softmax

- Softmax Role: Converts raw attention scores into a probability distribution, emphasizing more relevant tokens and attenuating less important ones.

- Why Scaling: Without scaling, dot product scores can grow large when the dimensionality d_k increases, causing the softmax function to saturate. This leads to extremely small gradients and slow learning.

- Normalization Importance: Scaling prevents exploding values and ensures that attention weights are distributed reasonably, stabilizing training and improving convergence.

3. Linear Attention and Its Limitations

What is linear attention?

- Linear attention approximates standard dot-product attention to reduce time and memory complexity from $O(n^2)$ to $O(n)$, where n is the sequence length.

- Instead of computing softmax(QK^T)V, it uses kernel-based feature maps ϕ to linearize the operation:

$$Attention(Q, K, V) \approx \phi(Q)(\phi(K)^T V)$$

Why is it useful for long sequences?

- Greatly reduces computational and memory overhead.

- Makes it feasible to train on very long documents or streams.

Limitations:

- Approximation may hurt accuracy, especially when modeling fine-grained token interactions.

- Less effective in tasks requiring precise token-level alignment (e.g., translation, summarization).

4. Group Attention and Its Application to Multi-modal Tasks

What is group attention?

- Group attention partitions the input sequence into G non-overlapping groups.

- Attention is applied **within each group** separately:

$$GroupAttention(Q,K,V) = Concat(softmax(\frac{Q_g K_g^T}{\sqrt{d_k}})V_g)_{g=1}^{G}$$

Advantages:

- Reduces attention complexity from $O(n^2)$ to $O(n^2/G)$
- Improves scalability and memory efficiency
- Allows parallel computation over groups

Application to multi-modal tasks:

- In vision-language models, visual tokens and text tokens can form separate groups.
- Enables modality-specific attention (intra-modality) and later fusion (inter-modality).
- Useful in models like Flamingo, GIT, and PaLI where images, text, or audio need to be handled in tandem.

5. Decoder-Only Models for NLU and NLG Tasks

Choose a model you prefer.

Question 3

1. NLU is concerned with understanding and interpreting human language input, while NLG focuses on generating human-like language output.

2. Encoder-based models: Good choice for NLU tasks. It can be used to solve NLG tasks, but not first choice.

Decode-based models: Good choice for NLG tasks. Generally, we do not directly use decoder-based models to solve NLU tasks since it just captures unidirectional features.

Encoder-decoder models: Good choice for both NLU and NLG tasks.

3.

Unidirectional LM mask

	Jill	Birt	[MASK]	outdoor	activities	He	was	[MASK]	hiking
Jill	0	1	1	1	1	1	1	1	1
Birt	0	0	1	1	1	1	1	1	1
[MASK]	0	0	0	1	1	1	1	1	1
outdoor	0	0	0	0	1	1	1	1	1
activities	0	0	0	0	0	1	1	1	1
He	0	0	0	0	0	0	1	1	1
was	0	0	0	0	0	0	0	1	1
[MASK]	0	0	0	0	0	0	0	0	1
hiking	0	0	0	0	0	0	0	0	0

Bidirectional LM mask

	Jill	Birt	[MASK]	outdoor	activities	He	was	[MASK]	hiking
Jill	0	0	0	0	0	0	0	0	0
Birt	0	0	0	0	0	0	0	0	0
[MASK]	0	0	0	0	0	0	0	0	0
outdoor	0	0	0	0	0	0	0	0	0
activities	0	0	0	0	0	0	0	0	0
He	0	0	0	0	0	0	0	0	0
was	0	0	0	0	0	0	0	0	0
[MASK]	0	0	0	0	0	0	0	0	0
hiking	0	0	0	0	0	0	0	0	0

Sequence-to-sequence LM mask

	Jill	Birt	[MASK]	outdoor	activities	He	was	[MASK]	hiking
Jill	0	0	0	0	0	1	1	1	1
Birt	0	0	0	0	0	1	1	1	1
[MASK]	0	0	0	0	0	1	1	1	1
outdoor	0	0	0	0	0	1	1	1	1
activities	0	0	0	0	0	1	1	1	1
He	0	0	0	0	0	0	1	1	1
was	0	0	0	0	0	0	0	1	1
[MASK]	0	0	0	0	0	0	0	0	1
hiking	0	0	0	0	0	0	0	0	0

4.

```
import torch

def mask_for_unidirectional_lm(source_seg: list, target_seg: list):
    max_len = len(source_seg) + len(target_seg)
    return torch.triu(torch.ones(max_len, max_len), diagonal=1)

def mask_for_bidirectional_lm(source_seg: list, target_seg: list):


    max_len = len(source_seg) + len(target_seg)
    return torch.zeros(max_len, max_len)

def mask_for_seq2seq_lm(source_seg: list, target_seg: list):
    src_len, tar_len = len(source_seg), len(target_seg)
    encode = torch.cat((torch.zeros(src_len, src_len), torch.ones(src_len, tar_len)), dim=1)
    decode = torch.cat((torch.zeros(tar_len, src_len),
                        torch.triu(torch.ones(tar_len, tar_len), diagonal=1)),
                        dim=1)
    return torch.cat((encode, decode), dim=0)
```

Question 4

1. (Answer from BERT's paper) We use a simple approach to extend the SQuAD v1.1 BERT model for this task. We treat questions that do not have an answer as having an answer span with start and end at the `[CLS]` token. The probability space for the start and end answer span positions is extended to include the position of the `[CLS]` token. For prediction, we compare the score of the no-answer span: $s_{null} = S \cdot C + E \cdot C$ to the score of the best non-null span $\hat{s_{i,j}} = \max_{j \geq i} S \cdot T_i + E \cdot T_j$. We predict a non-null answer when $\hat{s_{i,j}} \geq s_{null} + \tau$, where the threshold τ is selected on the dev set to maximize F1.

2. (Answer from SpanBERT's paper) Given a sequence of tokens X, we select a subset of tokens $Y \subseteq X$ by iteratively sampling spans of text until the masking budget (e.g., 15% of X) has been spent. At each iteration, we first sample a span length (number of words) from a geometric distribution $l \sim Geo(p)$, which is skewed towards shorter spans. We then randomly (uniformly) select the starting point for the span to be masked. We always sample a sequence of complete words (instead of subword tokens) and the starting point must be the beginning of one word.

   To support span selection models, we would ideally like the representations for the end of the span to summarize as much of the internal span content as possible. We do so by introducing a span boundary objective that involves predicting each token of a masked span using only the representations of the observed tokens at the boundaries.

   Formally, we denote the output of the transformer encoder for each token in the sequence by $\mathbf{x}_1$, ..., $\mathbf{x}_n$. Given a masked span of tokens $(x_s, ..., x_e) \in Y$, where (s, e) indicates its start and end positions, we represent each token x_i in the span using the output encodings of the external boundary tokens $\mathbf{x}_{s-1}$ and $\mathbf{x}_{e+1}$, as well as the position embedding of the target token $\mathbf{p}_{i-s+1}$:

$$\mathbf{y}_i = f(\mathbf{x}_{s-1}, \mathbf{x}_{e+1}, \mathbf{p}_{i-s+1})$$

   where position embeddings $\mathbf{p}_1$, $\mathbf{p}_2$, ... mark relative positions of the masked tokens with respect to the left boundary token x_{s-1}. We implement the representation function $f(\cdot)$ as a 2-layer feed-forward network with GeLU activations and layer normalization. We then use the vector representation y_i to predict the token x_i and compute the cross-entropy loss exactly like the MLM objective.

Reference

[1] Devlin J, Chang M W, Lee K, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[C]//Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). 2019: 4171-4186.

[2] Joshi M, Chen D, Liu Y, et al. Spanbert: Improving pre-training by representing and predicting spans[J]. Transactions of the Association for Computational Linguistics, 2020, 8: 64-77.