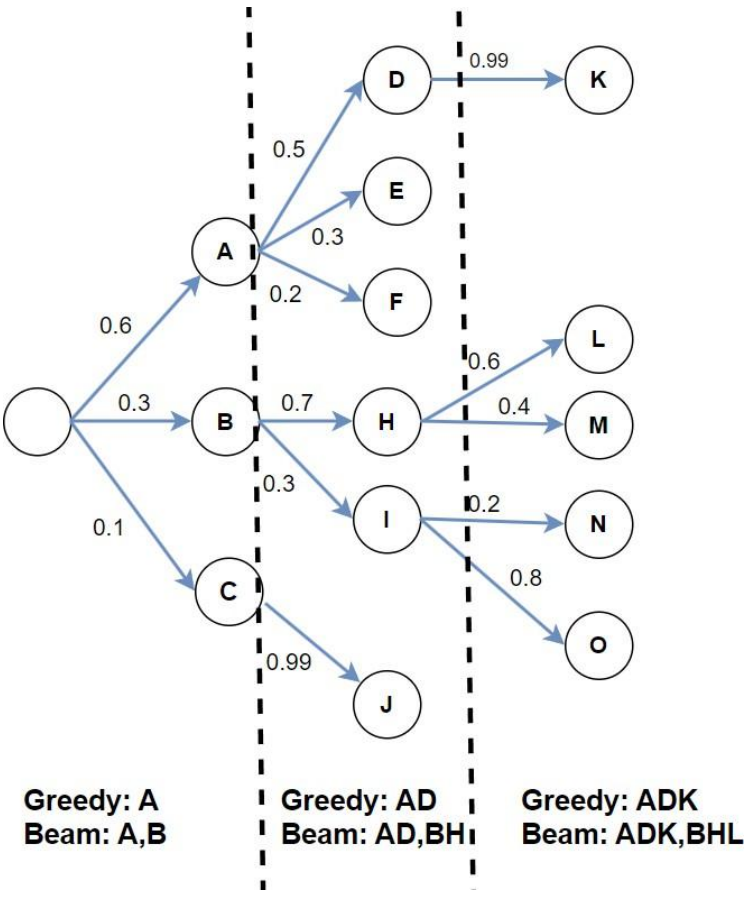
Question 1

1. [4,7,9,15,23]
   [2,8,10,0,0] (If you used the token ids in Figure 1)
   [2,13,5,9,0]

   Or

   [4,7,9,15,23,18]
   [2,8,10,0,0,0] (If you used the corrected token ids announced in Canvas)
   [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. Flaws: (Just choose one of the three)

- Fixed-Length Context: In a Seq2Seq model, the encoder processes the input sequence and generates a fixed-length context vector, known as the encoder's final hidden state. This fixed-length representation can potentially lose important information from the input sequence, especially for long sequences.

- Information Bottleneck: The fixed-length context vector is then used as the initial hidden state of the decoder, which aims to generate the output sequence. This compressed representation can act as an information bottleneck, making it challenging for the decoder to access all the relevant information from the input.

- Lack of Alignment: The Seq2Seq model does not explicitly model the alignment between the input and output sequences. It relies solely on the decoder's hidden state to capture the relevant information from the input. This lack of alignment modeling can lead to suboptimal performance, especially when the input and output sequences have different lengths or complex structures.

Improvement: (Just choose one of the three)

- Variable-Length Context: With attention, the decoder can dynamically attend to different parts of the input sequence at each decoding step. It uses weighted combinations of the encoder's hidden states, known as attention weights, to determine the relevance of each input token to the current decoding step. This allows the model to capture more fine-grained information from the input, even for long sequences.

- Enhanced Information Flow: Attention allows the decoder to access a weighted combination of all the encoder's hidden states, rather than relying solely on a fixed-length context vector. This enables the decoder to overcome the information bottleneck problem by allowing more direct access to the relevant parts of the input sequence. It helps in preserving and utilizing the important information effectively.

- Alignment Modeling: The attention mechanism explicitly models the alignment between the input and output sequences. By assigning different attention weights to different input tokens, the model learns to align the relevant parts of the input with the corresponding parts of the output. This alignment modeling greatly improves the model's ability to generate accurate translations or predictions, especially for complex input-output relationships.

2.

Linear transformation of the input word vector matrix X

Can increase the expressive ability, and the generalization ability of the attention score matrix is higher.

3.

The role of Softmax：

- The value range of the dot product is $(-\infty, +\infty)$ $(-\infty,+\infty)$. The value range after direct multiplication by V is also $(-\infty, +\infty)$ $(-\infty,+\infty)$. Softmax normalizes it Convert it to the (0, 1) (0,1) interval for subsequent multiplication with V;

- At the same time, it also plays the role of scaling the gradient (preventing negative numbers and excessively large results from causing gradient problems).

The role of Scale：

Matrix dot multiplication may cause the values to increase exponentially, making the gradient of softmax very small, so use it for scaling to avoid this problem

4.

The most important part is the Masked Multi-Head Attention (6 points) which enables the Transformer work in the auto-regressive fashion. Specifically, the masked attention can help the model only attend to the previous tokens during the decoding stage. Additionally, the positional encoding (2 points) also plays an important role in modeling sequential features. The PE enables the Transformer to distinguish the tokens in different positions

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.