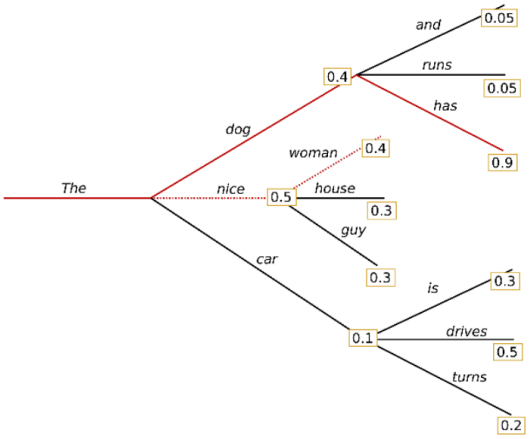
Week2 과제: Seq2Seq 보고서

1️⃣ LSTM 기반 Seq2Seq 모델에서 디코딩할 때 사용하는 Beam Search 동작 방식에 대해서 설명.

Beam Search란?



: 각 스텝에서 **상위 k개의 가장 가능도가 높은 토큰들로 유지**

(↔ Greed Search Decoder: 각 타임스텝에서 가능도가 가장 높은 하나의 토큰 선택)

- 보통 k = 5 ~ 10 선택

$$score(y_1, \dots, y_t) = \log P_{LM}(y_1, \dots, y_t|x) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \dots, y_{i-1}, x)$$

- **과정**
  1. 시작 상태 = <sos> 토큰 하나만 존재
  2. 첫 타임스텝에서 softmax 확률 분포를 가지고 **상위 beam size = k 개의 단어 선택** ⇒ k개의 시퀀스 생성
  3. 확장된 후보에 대해 점수(다음 가능한 모든 단어의 확률) 계산

     `기존 누적 점수 + log(P(새로운 단어 | 이전 시퀀스))`

     ⇒ k개의 시퀀스 × V개의 단어 = k × V 개의 후보 시퀀스 생성
  4. 가능도가 높은 상위 k개의 시퀀스만 남기고, 나머지 후보 제거
  5. 시퀀스가 끝날 때까지 반복
     - 시퀀스의 끝을 정하는 방법

        1. <eos> 토큰 등장
        2. 설정한 최대 길이 도달
        3. threshold 밑으로 가능도가 낮아짐
    6. 최종적으로 누적 로그 확률 기준으로 가장 높은 시퀀스를 출력으로 선택

- 장점
    - 디코딩 시 여러 후보 시퀀스를 고려 → 좋은 결과
- 단점
    - 계산 비용
    - 매우 긴 시퀀스에서는 효율성 감소

2️⃣ Seq2Seq with LSTM 모델은 Attention이 없던 시절 제안된 구조임. 기본 Seq2Seq 모델의 한계와 이후 Attention 메커니즘이 이 한계를 어떻게 보완했는지 설명.

✅ Seq2Seq 모델의 한계 (= RNN 계열의 고질적 한계)

1. 고정된 context vector의 긴 문장 처리 한계
    - Seq2Seq에서 인코더의 출력은 고정 길이의 벡터
        - 가변 길이의 입력 문장을 하나의 고정된 벡터로 압축
        - 긴 문장이 입력되더라도 똑같은 길의 벡터로 밀어 넣기 때문에 → 정보 손실 발생, 성능 저하
2. 장기 의존성(Long-term Dependency) 문제
    - 입력 시퀀스가 길어질수록 초기 단어 정보가 흐려지기 때문에, 앞의 정보가 제대로 전달되지 않음.
3. Parallelization 문제
    - RNN 구조 특성 상, 인코더와 디코더는 순차적으로 동작
    - 입력이 차례대로 들어가기 때문에, 계산을 병렬적으로 처리하지 못함.

✅ Attention 도입 후 (Seq2Seq 모델 + Attention weight)

[Attention 모델 동작 방식]

디코더가 단어를 생성할 때마다

- 디코더의 현재 hidden state
- 인코더의 각 시점별 hidden state

를 비교하여 유사도 계산 → 가중치 벡터 생성

→ 인코더의 hidden state들을 가중합하여 새로운 context vector를 만듦

⇒ Decoder에서 매번 Encoder의 모든 output을 참고하여 그 중 어떤 정보가 중요한지 계산하여 확률값(weight)으로 반영

1. Encoder 개선
    - 기존 seq2seq 문제: 마지막 hidden state 만 디코더에 전달
    - Attention 해결: 모든 타임스텝의 hidden state 정보를 context vector에 반영 **→ 고정 길이 벡터 해결**
2. Decoder 개선
    - 매 타임스텝마다 모든 인코더 hidden state에 접근
    - 각 hidden state의 중요도를 나타내는 가중치 a와 인코더에서 받아온 각 hidden state를 가중합하여 벡터를 얻음 = context vector
    - 각 디코딩 시점에서 필요한 위치의 정보에 가중치를 부여해 참조 **→ 장기 정보도 활용 가능**
3. Parallelization
    - 인코더의 hidden state들을 **한 번에 행렬 연산으로 처리**