Natural Language Processing

Tutorial 8 (Week 11): Sequence Modeling

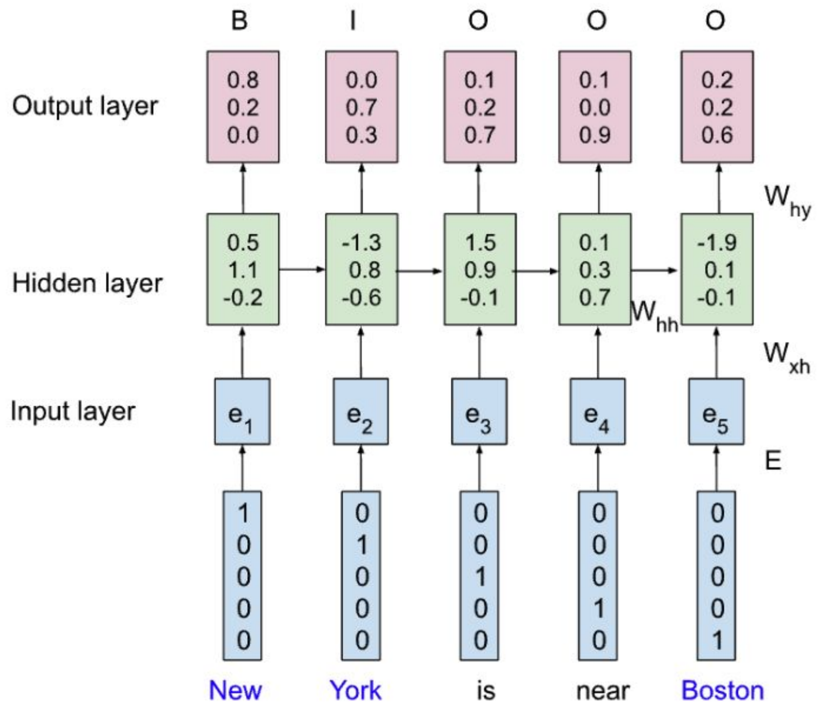
Question 1

RNNs have been used for many problems in NLP. For this exercise, we will consider RNNs for named entity recognition as shown in the Figure below.

1) If W_{xh} and W_{hh} are the weight matrices for input to hidden layer and hidden to hidden layer, write down the composition function of a vanilla unidirectional RNN model.

Question 1 (1)

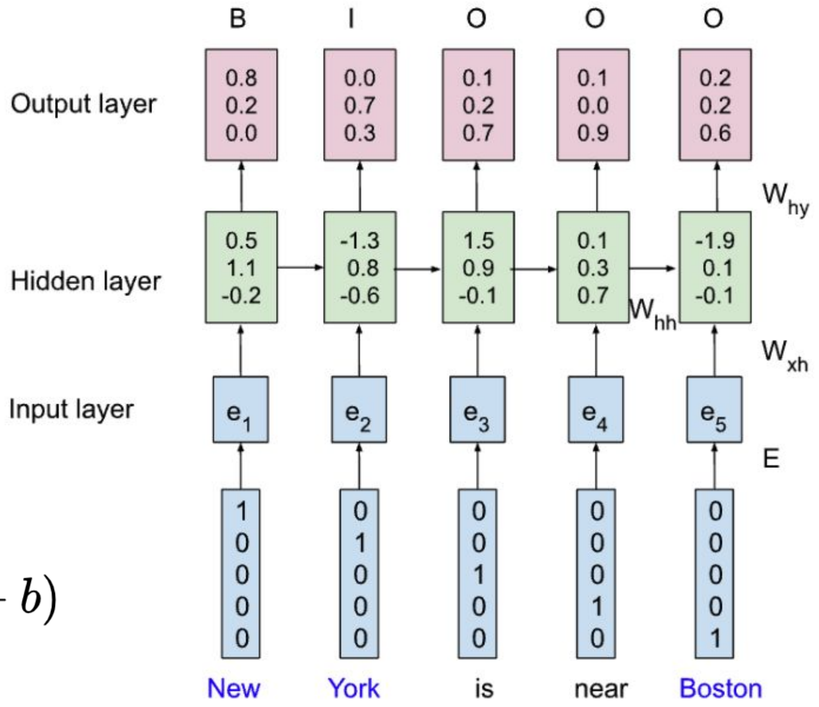
1) If W_{xh} and W_{hh} are the weight matrices for input to hidden layer and hidden to hidden layer, write down the composition function of a vanilla unidirectional RNN model.



Solution 1 (1)

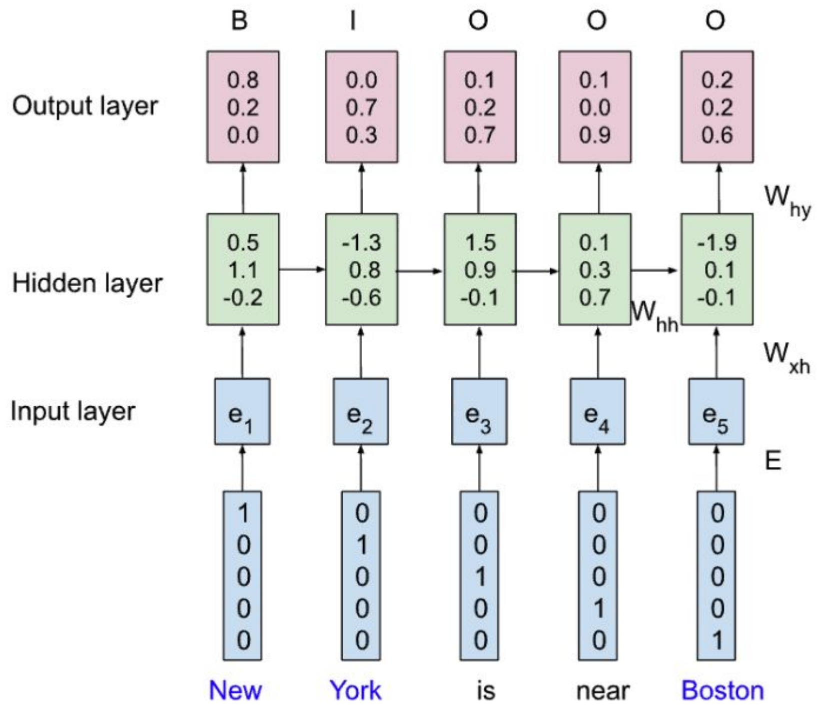
1) If W_{xh} and W_{hh} are the weight matrices for input to hidden layer and hidden to hidden layer, write down the composition function of a vanilla unidirectional RNN model.

$$h_t = \sigma(W_{xh} \cdot e_t + W_{hh} \cdot h_{t-1} + b)$$



Question 1 (2)

2) Write the composition functions for a bidirectional RNN.



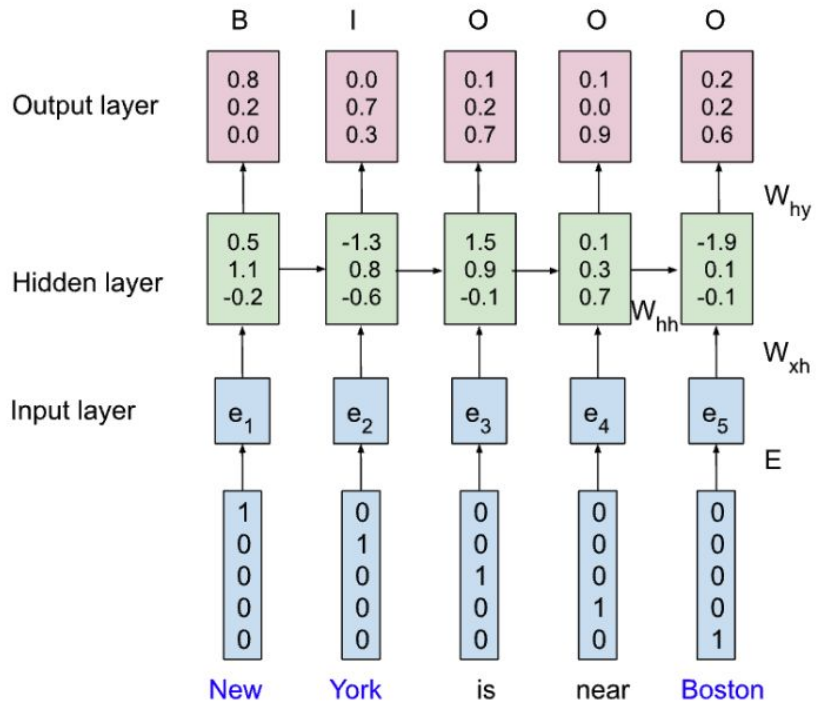
Solution 1 (2)

2) Write the composition functions for a bidirectional RNN.

$$\overleftarrow{h}^{(t)} = \sigma(W_{bx}x^{(t)} + W_{bh}h^{(t+1)} + b_b)$$

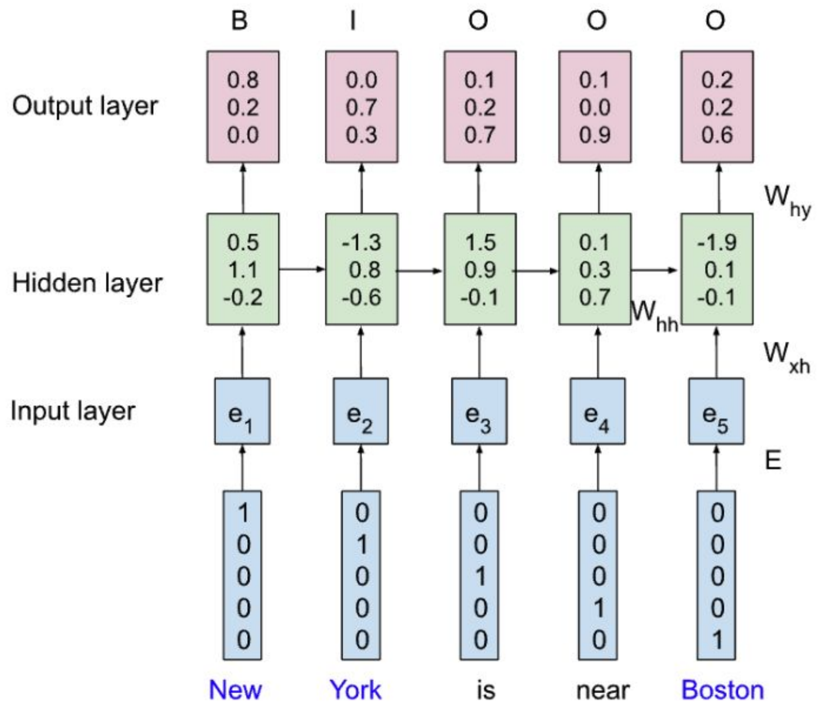
$$\overrightarrow{h}^{(t)} = \sigma(W_{fx}x^{(t)} + W_{fh}h^{(t-1)} + b_f)$$

$$h_t = concat(\overleftarrow{h}^{(t)}, \overrightarrow{h}^{(t)})$$



Question 1 (3)

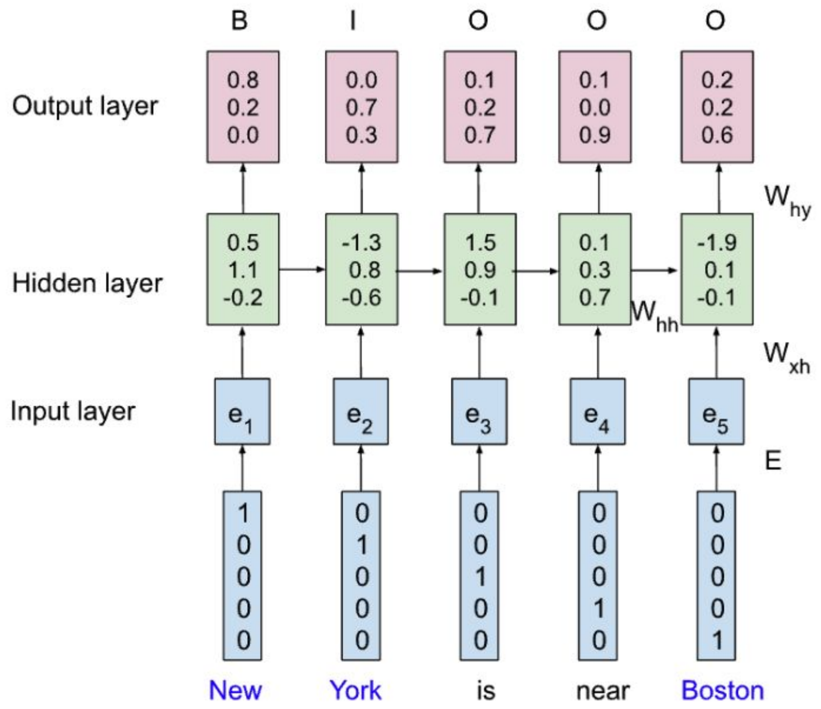
3) Describe how the final label is predicted in the output layer given the hidden representation for each word.



Solution 1 (3)

3) Describe how the final label is predicted in the output layer given the hidden representation for each word.

$$y_t = softmax(W_{hy} \cdot h_t + b_y)$$



Question 1 (4)

4) F1-score is a commonly used metric to evaluate the NER performance. It is computed based on recall and precision: F1=(2*recall*precision) / (recall+precision). Suppose your model predicts “B I O O O” whereas the gold label should be “B I O O B”. What is the F1 score in this case?

Solution 1 (4)

- Identify Predicted and gold labels:
    - Predicted labels: B I O O O
    - Gold labels: B I O O B
- Extract named entities from labels:
    - Predicted: New York (B I)
    - Gold: New York (B I), Boston (B)
- Calculate Precision and Recall:
    - Precision = TP / (TP+FP) = 1 / (1+0) = 1
    - Recall = TP / (TP+FN) = 1 / (1+1) = 0.5
- F1 = 2 * 1 * 0.5 / (1 + 0.5) = 0.67

Question 2

To actually implement RNNs in python, there are several points to take note of.

1) For mini-batch training, the input to RNNs contains a number of sentences with different lengths. In order to perform efficient batch training, how to handle variable-length inputs?

Solution 2(1)

- **Padding**: All sentences in the mini-batch are padded with a special token (e.g., <PAD>) to make them the same length as the longest sentence in the batch.

```
Sentence 1: "I love NLP"           -> [I, love, NLP, <PAD>, <PAD>]
Sentence 2: "Chatbots are fun"  -> [Chatbots, are, fun, <PAD>, <PAD>]
```

- **Masking**: While computing the loss or hidden states, masks are used to ignore the padded tokens, ensuring that they do not contribute to the model’s output or loss calculation.

Question 2

To actually implement RNNs in python, there are several points to take note of.

2) In language model training with RNNs, the model uses the correct previous tokens from the dataset to predict the next one. However, during testing, it must rely on its own predictions to generate future tokens. How might this difference between training and testing impact the model's performance? Can you think of a solution to alleviate this issue?

Solution 2(2)

- This phenomenon is called **exposure bias**
- The training phase is called "**teacher forcing**" which helps the model learn to predict the next token accurately based on correct context.
- During testing, the model uses its own predictions as input. If it makes a mistake early on, that incorrect prediction becomes part of the input context, potentially leading to **error propagation**.
- Since the model never sees its own mistakes during training, it struggles to correct or adapt when such mistakes occur during testing. As a result, the model may generate less coherent or inaccurate sequences, especially for long outputs.

Solution 2(2)

Scheduled Sampling

- This technique involves gradually replacing ground-truth tokens with the model's own predictions during training.
- At the beginning of training, the model mostly sees the correct tokens. As training progresses, it increasingly relies on its own predicted tokens.
- This allows the model to adapt to the input conditions it will encounter during inference, reducing exposure bias.

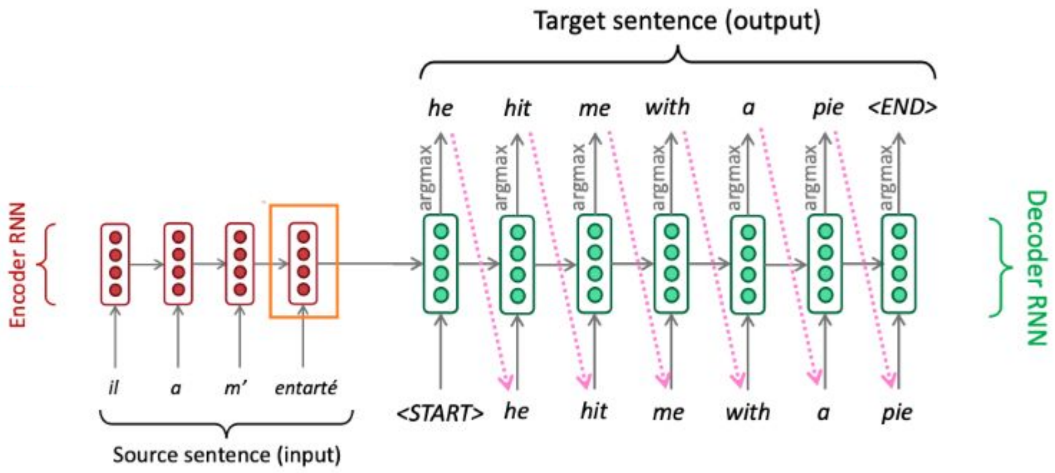
Question 3

Imagine a simplified neural machine translation model that translates sentences from English to French. Consider that the model uses a single-layer RNN for both the encoder and the decoder. For simplicity, assume that each word in both languages has already been represented as a fixed-size vector (e.g., word2vec embedding).

1) Describe the role of the encoder RNN in the neural machine translation process.

Solution 3 (1)

The encoder RNN takes the source sentence, in this case, an English sentence, and processes it word by word. Its role is to compress the information contained in the entire source sentence into a context vector, typically represented by its last hidden state. This context vector aims to encapsulate the semantics and other pertinent details of the input sentence.



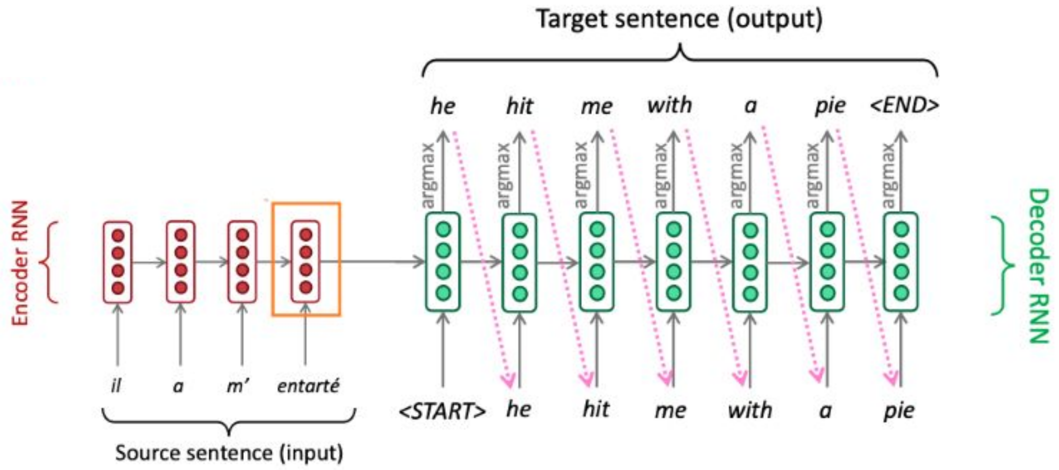
Question 3

Imagine a simplified neural machine translation model that translates sentences from English to French. Consider that the model uses a single-layer RNN for both the encoder and the decoder. For simplicity, assume that each word in both languages has already been represented as a fixed-size vector (e.g., word2vec embedding).

2) What is the purpose of the decoder RNN, and how does it differ in functionality from the encoder?

Solution 3 (2)

The decoder RNN takes the context vector generated by the encoder as its initial hidden state. Starting from a special "start" token, the decoder aims to generate the target sentence word by word in the target language, which is French in this case. Unlike the encoder, which summarizes information, the decoder's role is to expand this information back into a meaningful sentence in a different language.



Question 3

Imagine a simplified neural machine translation model that translates sentences from English to French. Consider that the model uses a single-layer RNN for both the encoder and the decoder. For simplicity, assume that each word in both languages has already been represented as a fixed-size vector (e.g., word2vec embedding).

3) Imagine you have an English sentence: "I am learning". You wish to translate it into French: "Je suis en train d'apprendre". Describe the steps the NMT model would take to perform this translation.

Solution 3 (3)

- Encoder Processing "I am learning":

$$h_x^t = \tanh(W_{xe} \cdot e_t + W_{xh} \cdot h_x^{t-1} + b_x)$$

- Initialization of decoder hidden state:

$$h_y^0 = h_x^T$$

- Decoder generation at each timestamp

$$h_y^t = \tanh(W_{ye} \cdot y_{t-1} + W_{yh} \cdot h_y^{t-1} + b_y)$$

$$\hat{y}_t = softmax(W \cdot h_y^t + b)$$

Question 3

Imagine a simplified neural machine translation model that translates sentences from English to French. Consider that the model uses a single-layer RNN for both the encoder and the decoder. For simplicity, assume that each word in both languages has already been represented as a fixed-size vector (e.g., word2vec embedding).

4) How would the process change if the encoder is a bi-directional RNN?

Solution 3 (4)

- Encoder Processing "I am learning":

$$\overleftarrow{h}_x^{(t)} = \tanh(W_{be} e^{(t)} + W_{bh} h^{(t+1)} + b_b)$$
$$\overrightarrow{h}_x^{(t)} = \tanh(W_{fe} e^{(t)} + W_{fh} h^{(t-1)} + b_f) \qquad h_x^t = [\overleftarrow{h}_x^{(t)} ; \overrightarrow{h}_x^{(t)}]$$

- Initialization of decoder state:

$$h_y^0 = h_x^T$$

- Decoder generation at each timestamp

$$h_y^t = \tanh(W_{ye} \cdot y_{t-1} + W_{yh} \cdot h_y^{t-1} + b_y)$$

$$\hat{y}_t = softmax(W \cdot h_y^t + b)$$

Question 3

Imagine a simplified neural machine translation model that translates sentences from English to French. Consider that the model uses a single-layer RNN for both the encoder and the decoder. For simplicity, assume that each word in both languages has already been represented as a fixed-size vector (e.g., word2vec embedding).

5) What could be the problem of greedy decoding during translation generation? Can you come up with any possible solution?

Solution 3 (5)

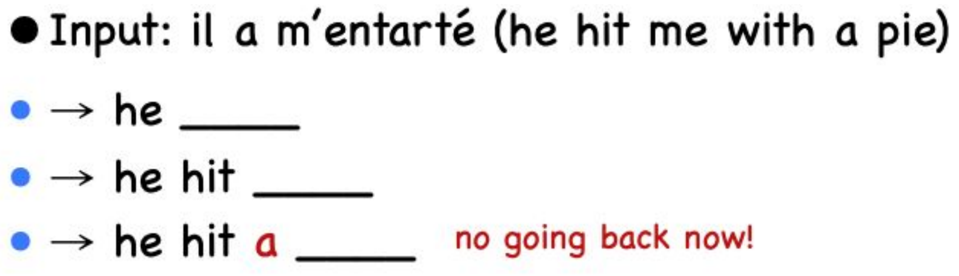
- Greedy decoding makes locally optimal decisions, choosing the best token at each time step. However, these choices might not lead to the globally optimal sequence (i.e., the best overall translation).
- Greedy decoding always selects the most likely token, making the output repetitive and predictable, especially for models trained on frequent patterns.

Greedy Decoding

- Core idea: Always selects the next token with highest probability

$$\hat{y}_t = \underset{w \in V}{\mathbf{argmax}}\, P(y_t = w | y_{<t})$$

- Problem: Greedy decoding cannot correct previous mistakes



- Input: il a m'entarté (he hit me with a pie)
- $\rightarrow$ he _____
- $\rightarrow$ he hit _____
- $\rightarrow$ he hit a _____ no going back now!

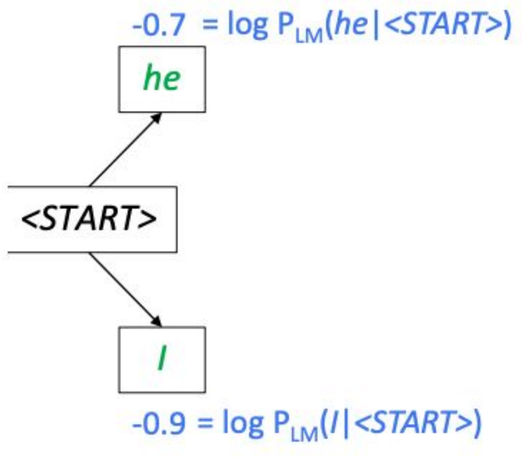
Option 1: Beam search decoding

- Core idea: On each step of decoder, keep track of the k most probable partial translations (which we call hypotheses). k (beam size) usually ranges from 5 to 10.

Decoding Algorithm – Beam Search

Source: stanford 224n

$k = 2$



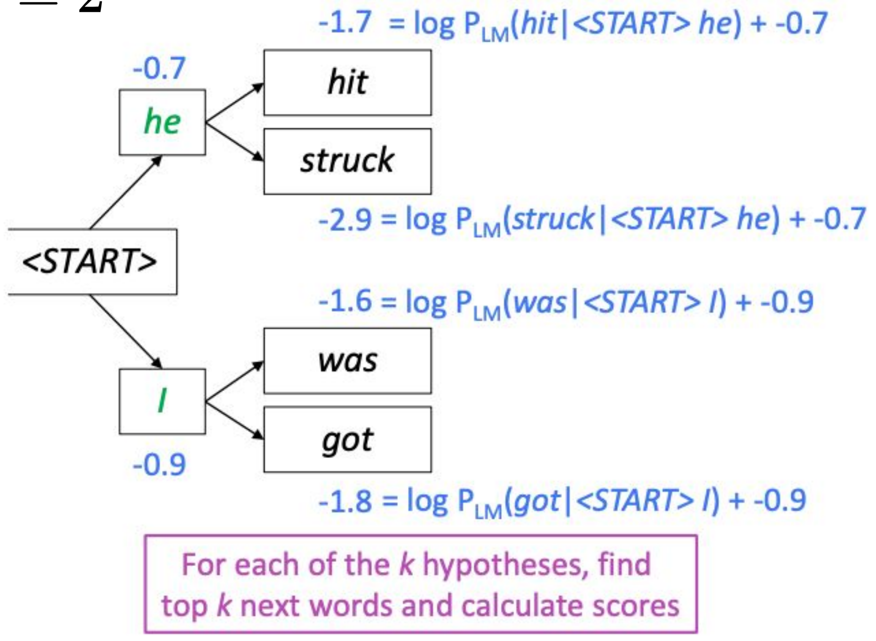
$-0.7 = \log P_{LM}(he|\text{<START>})$
he
<START>
I
$-0.9 = \log P_{LM}(I|\text{<START>})$

Take top *k* words
and compute scores

Decoding Algorithm – Beam Search

Source: stanford 224n

$k = 2$

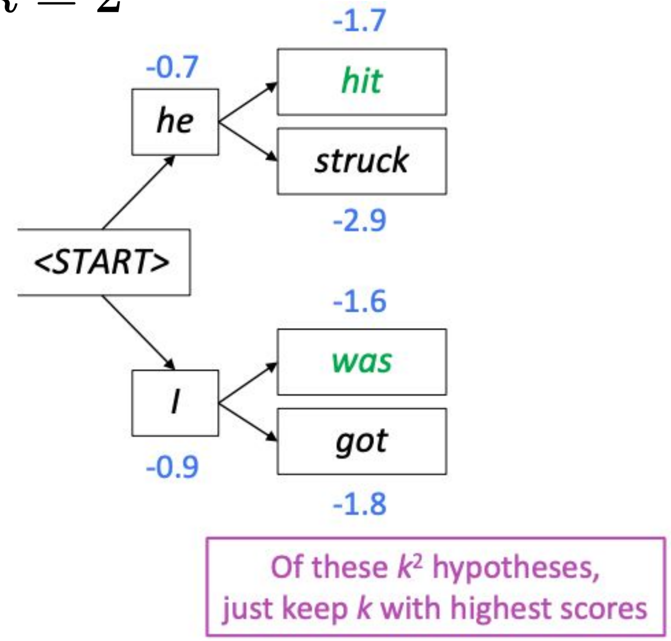


-1.7 = log P_LM(hit|<START> he) + -0.7
-0.7
he
hit
struck
-2.9 = log P_LM(struck|<START> he) + -0.7
<START>
-1.6 = log P_LM(was|<START> I) + -0.9
was
I
got
-0.9
-1.8 = log P_LM(got|<START> I) + -0.9
For each of the k hypotheses, find top k next words and calculate scores

Decoding Algorithm – Beam Search

Source: stanford 224n

$k = 2$

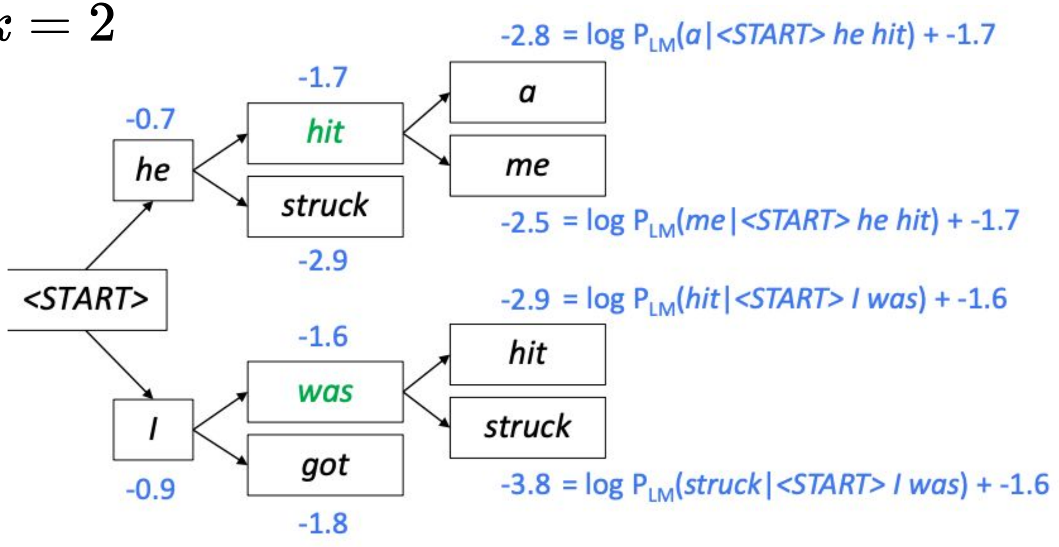


Of these k^2 hypotheses, just keep k with highest scores

Decoding Algorithm – Beam Search

Source: stanford 224n

$k = 2$

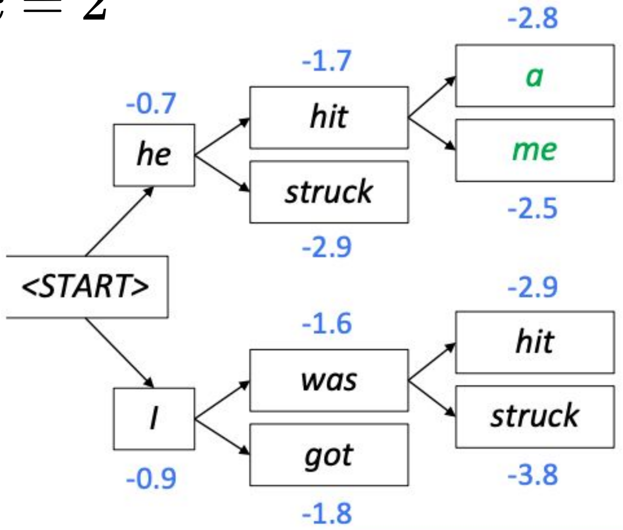

-2.8 = log P_{LM}(a|<START> he hit) + -1.7
-1.7
-0.7
hit
he
struck
a
me
-2.5 = log P_{LM}(me|<START> he hit) + -1.7
-2.9
<START>
-2.9 = log P_{LM}(hit|<START> I was) + -1.6
-1.6
was
I
got
hit
struck
-0.9
-3.8 = log P_{LM}(struck|<START> I was) + -1.6
-1.8

For each of the *k* hypotheses, find top *k* next words and calculate scores

Decoding Algorithm – Beam Search

Source: stanford 224n

$k = 2$

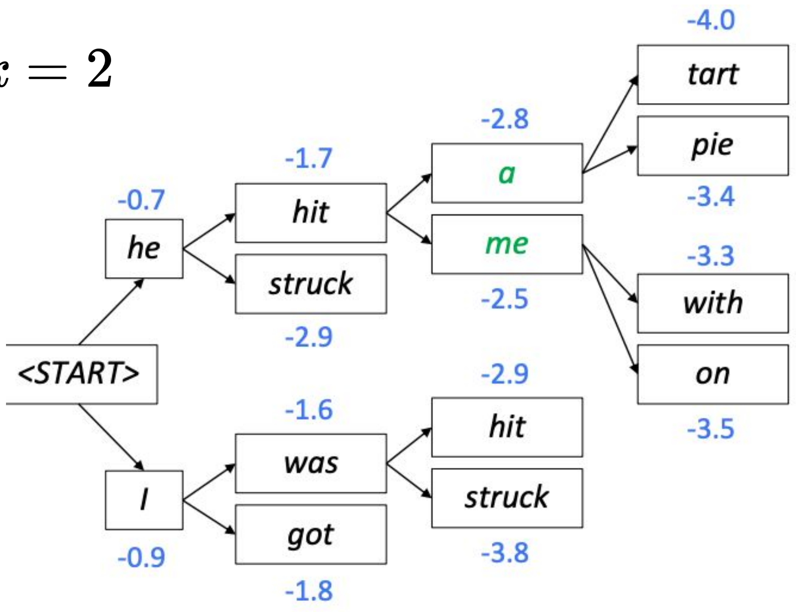


Of these k^2 hypotheses, just keep k with highest scores

Decoding Algorithm – Beam Search

Source: stanford 224n

$k = 2$

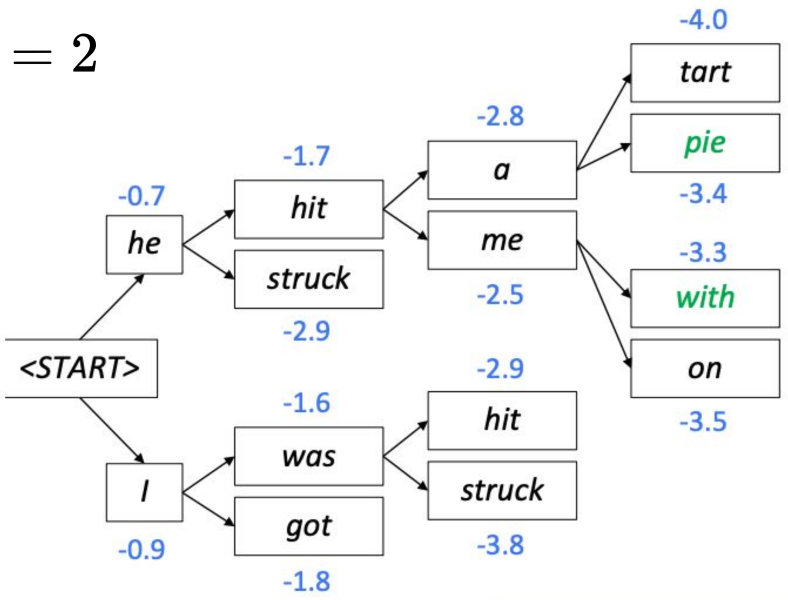


For each of the *k* hypotheses, find top *k* next words and calculate scores

Decoding Algorithm – Beam Search

Source: stanford 224n

$k = 2$

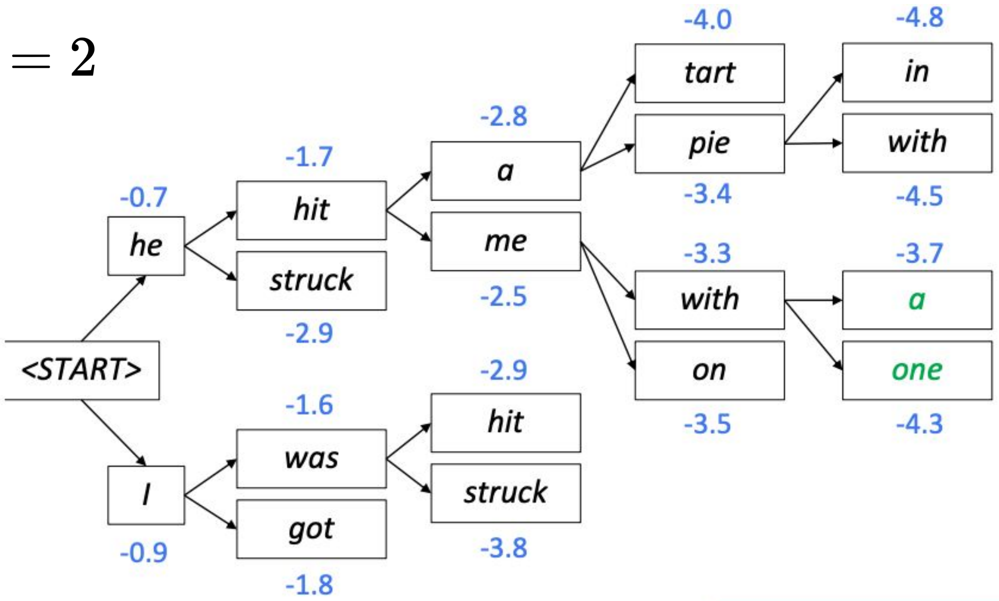


Of these k^2 hypotheses, just keep k with highest scores

Decoding Algorithm – Beam Search

Source: stanford 224n

$k = 2$

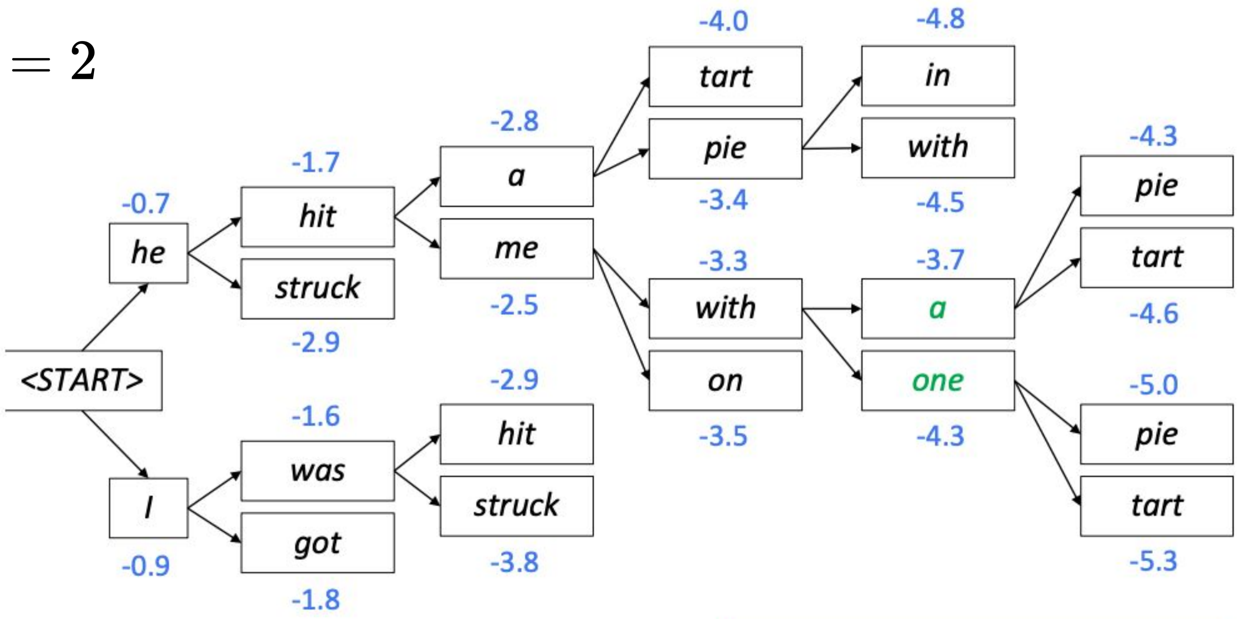


Of these k^2 hypotheses, just keep k with highest scores

Decoding Algorithm – Beam Search

Source: stanford 224n

$k = 2$

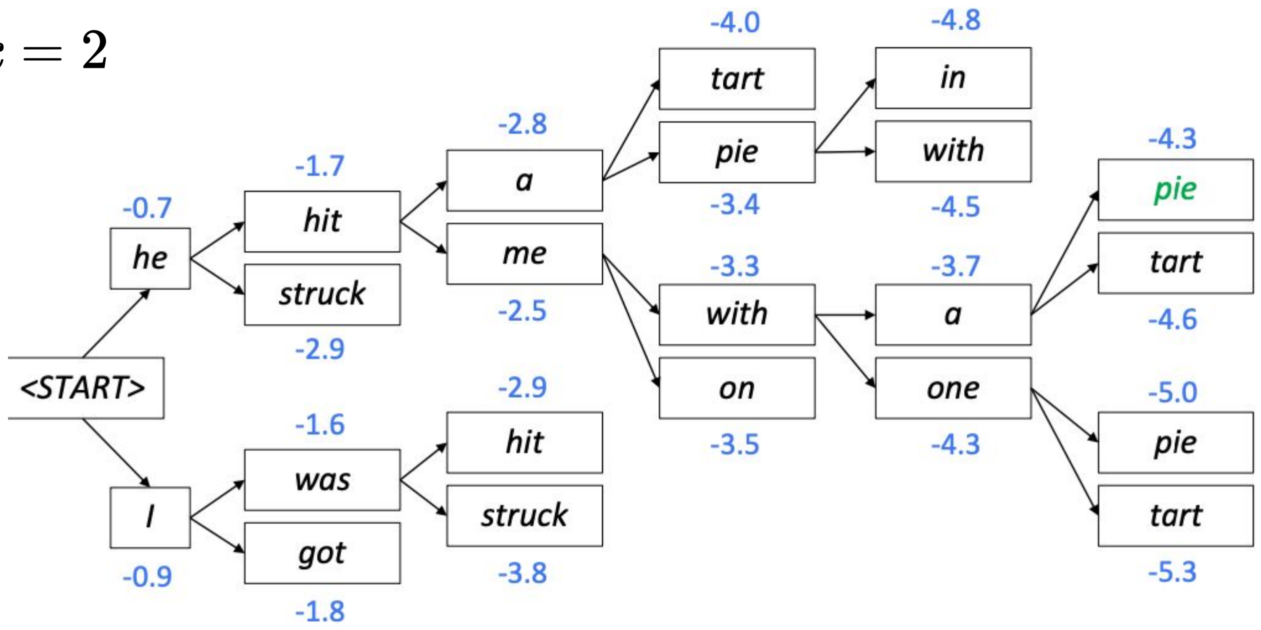


For each of the *k* hypotheses, find top *k* next words and calculate scores

Decoding Algorithm – Beam Search

Source: stanford 224n

$k = 2$

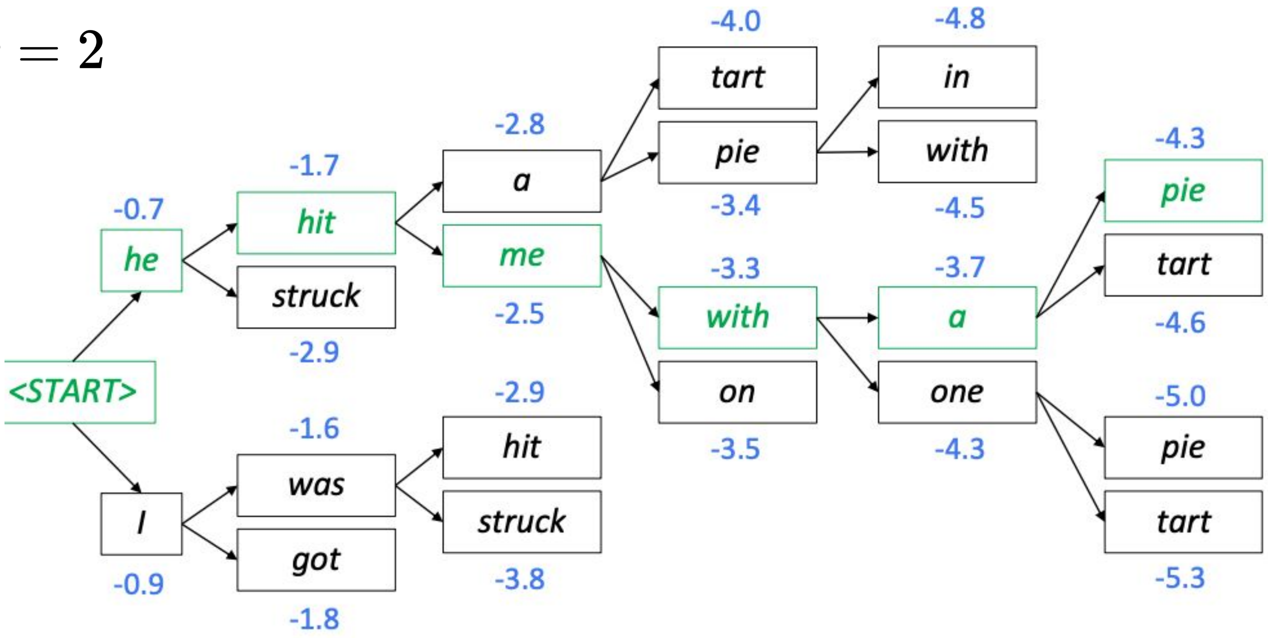


This is the top-scoring hypothesis!

Decoding Algorithm – Beam Search

Source: stanford 224n

$k = 2$



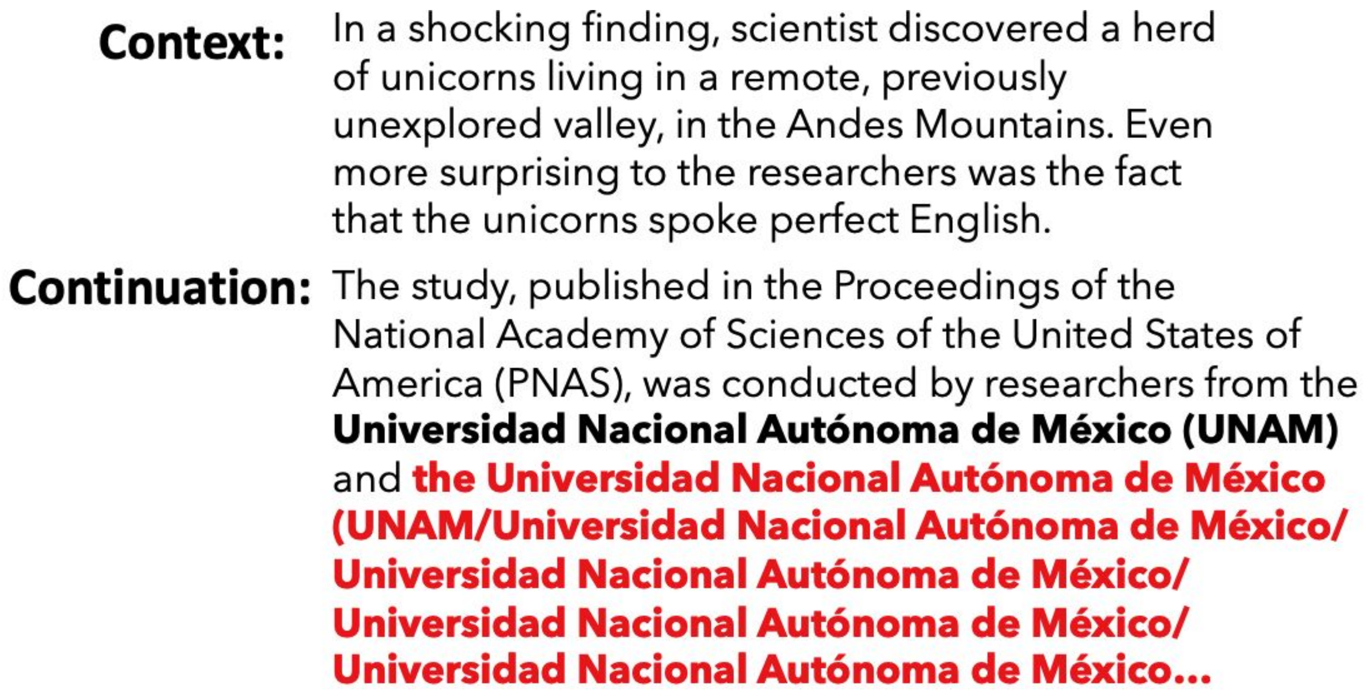
Backtrack to obtain the full hypothesis

Decoding Algorithm

Source: stanford 224n

- In greedy decoding, usually we decode until the model produces a <END> token
  - For example: *<START> he hit me with a pie <END>*
- In beam search decoding, different hypotheses may produce <END> tokens on different timesteps
  - When a hypothesis produces <END>, that hypothesis is complete.
  - Place it aside and continue exploring other hypotheses via beam search.
- Usually we continue beam search until:
  - We reach timestep *T* (where *T* is some pre-defined cutoff), or
  - We have at least *n* completed hypotheses (where *n* is pre-defined cutoff)

What's Wrong with Most Likely String?



Context: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Continuation: The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the **Universidad Nacional Autónoma de México (UNAM)** and **the Universidad Nacional Autónoma de México (UNAM/Universidad Nacional Autónoma de México/ Universidad Nacional Autónoma de México/ Universidad Nacional Autónoma de México/ Universidad Nacional Autónoma de México...**

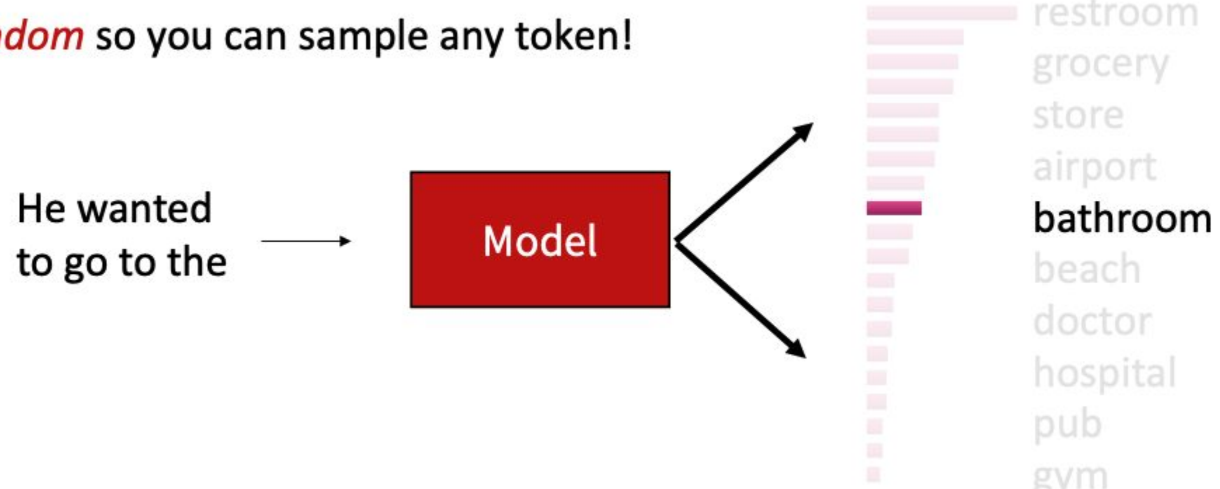
[Holtzman et al., 2020]

Option 2: Sampling

- Sample a token from the distribution of tokens

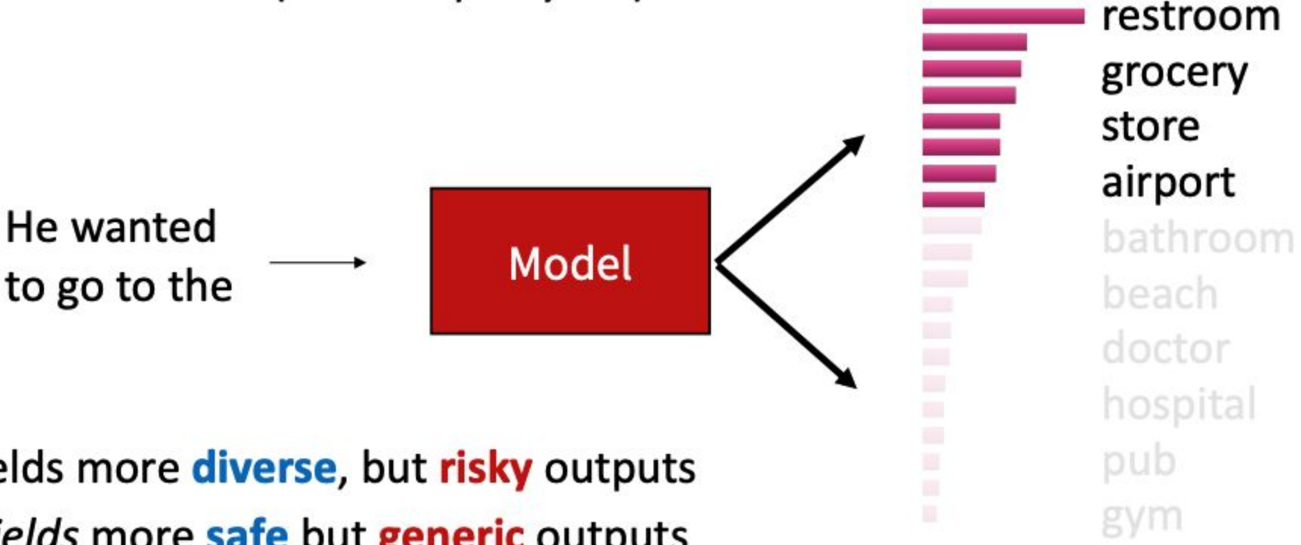
$$\hat{y}_t \sim P(y_t = w \mid \{y\}_{<t})$$

- It’s *random* so you can sample any token!



Option 2: Top-k Sampling

- Only sample from the top *k* tokens in the probability distribution
- Common values are *k* = 50 (*but it's up to you!*)



- Increase *k* yields more **diverse**, but **risky** outputs
- Decrease *k yields* more **safe** but **generic** outputs

Option 3: Temperature scaling

- Recall: On timestep *t*, the model computes a prob distribution P_t by applying the softmax function to a vector of scores $s \in \mathbb{R}^{|V|}$

$$P_t(y_t = w) = \frac{\exp(S_w)}{\sum_{w' \in V} \exp(S_{w'})}$$

- You can apply a *temperature hyperparameter* τ to the softmax to rebalance P_t:

$$P_t(y_t = w) = \frac{\exp(S_w/\tau)}{\sum_{w' \in V} \exp(S_{w'}/\tau)}$$

- Raise the temperature $\tau > 1$: P_t becomes more uniform
  - More diverse output (probability is spread around vocab)
- Lower the temperature $\tau < 1$: P_t becomes more spiky
  - Less diverse output (probability is concentrated on top words)

Temperature is a hyperparameter for decoding:
It can be tuned for both beam search and sampling.

Question 3

Imagine a simplified neural machine translation model that translates sentences from English to French. Consider that the model uses a single-layer RNN for both the encoder and the decoder. For simplicity, assume that each word in both languages has already been represented as a fixed-size vector (e.g., word2vec embedding).

6) What are some potential drawbacks or limitations of using RNNs for NMT?

Solution 2 (6)

- **Sequential Processing**: They process sequences in a step-by-step manner. This makes parallelization across sequence elements difficult, which can lead to slower training times, especially for longer sequences.
- **Vanishing and Exploding Gradients**: This can hinder the model's ability to capture long-range dependencies in the data.
- **Fixed-Length Bottleneck**: The encoder compresses the entire input sequence into a fixed-size context vector (usually the hidden state of the last time step). This single vector is expected to capture all the semantics of the source sentence, which becomes a bottleneck, especially for long sentences.

Coding

https://drive.google.com/file/d/1aDFQtSCYBpgRwkj9eq8Rao9eq5_68C31/view?usp=sharing