DS-GA-1011: Natural Language Processing with Representation Learning, Fall 2024

HW2 - Machine Translation

Jiasheng Ni
jn2294

Please write down any collaborators, AI tools (ChatGPT, Copliot, codex, etc.), and external resources you used for this assignment here.
Collaborators:
AI tools:
Resources:

By turning in this assignment, I agree by the honor code of the College of Arts and Science at New York University and declare that all of this is my own work.

Acknowledgement: Problem 1 was developed by Yilun Kuang. Problem 2 is based off of Annotated Transformers from Sasha Rash and developed by Nitish Joshi.

Before you get started, please read the Submission section thoroughly.

Submission

Submission is done on Gradescope.

Written: You can either directly type your solution in the released `.tex` file, or write your solution using pen or stylus. A `.pdf` file must be submitted.

Programming: Questions marked with "coding" at the start of the question require a coding part. Each question contains details of which functions you need to modify. We have also provided some unit tests for you to test your code. You should submit all `.py` files which you need to modify, along with the generated output files as mentioned in some questions.

Compute Budget: For question 2.3, you should expect the total code execution time to be less than 2 hours on a single NVIDIA Quadro RTX 8000 GPU from NYU Greene HPC. Please plan ahead, as requesting GPU resources on the cluster can take several hours or even longer during peak times.

Due Date: This homework is due on October 9, 2024, at noon 12pm Eastern Time.

1 Recurrent Neural Network

In this problem, you will show the problem of vanishing and exploding gradients for Recurrent Neural Network (RNN) analytically. To show this, we will first expand the gradient of the loss function with respect to the parameters using the chain rule. Then, we will bound the norm of each individual partial derivative

with matrix norm inequalities. The last step will be to collect all of the partial derivative terms and show how repeated multiplication of a single weight matrix can lead to vanishing or exploding gradients.

1.1 RNN Derivatives

Let $S = (s_1, \cdots, s_T)$ be a sequence of input word tokens and T be the sequence length. For a particular token $s_t \in \mathcal{V}$ for $1 \leq t \leq T$, we can obtain its corresponding word embedding $x_t \in \mathbb{R}^d$ by applying equation (1), where $\phi_{\text{one-hot}}$ is the one-hot encoding function and W_e is the word embedding matrix.

The RNN forward pass computes the hidden state $h_t \in \mathbb{R}^{d'}$ using equation (2). Here $W_{\text{hh}} \in \mathbb{R}^{d' \times d'}$ is the recurrent weight matrix, $W_{\text{ih}} \in \mathbb{R}^{d' \times d}$ is the input-to-hidden weight matrix, $b_h \in \mathbb{R}^{d'}$ is the hidden states bias vector, and $\sigma : \mathbb{R}^{d'} \to [-1, 1]^{d'}$ is the tanh activation function. $W_{\text{hh}}, W_{\text{ih}}, b_h$ are shared across sequence index t.

The output of RNN $o_t \in \mathbb{R}^k$ at each sequence index t is given by equation (3), where $W_{h_o} \in \mathbb{R}^{k \times d'}$ is the hidden-to-output weight matrix and $b_o \in \mathbb{R}^k$ is the output bias vector. For an input sequence $S = (s_1, \cdots, s_T)$, we have a corresponding sequence of RNN hidden states $H = (h_1, \cdots, h_T)$ and outputs $O = (o_1, \cdots, o_T)$.

$$x_t = W_e \phi_{\text{one-hot}}(s_t) \tag{1}$$

$$h_t = \sigma(W_{\text{hh}} h_{t-1} + W_{\text{ih}} x_t + b_h) \tag{2}$$

$$o_t = W_{h_o} h_t + b_o \tag{3}$$

Let's now use this RNN model for classification. In particular, we consider the last output o_T to be the logits (scores for each class), which we then convert to the class probability vector $p_T \in [0, 1]^k$ by $p_T = g(W_{h_o} h_T + b_o)$ where $g(\cdot)$ is the softmax function and $\|p_T\|_1 = 1$.

1. (1 point, written) Write down the per-example cross-entropy loss $\ell(y, p_T)$ for the classification task. Here $y \in \{0, 1\}^k$ is a one-hot vector of the label and p_T is the class probability vector where $p_T[i] = p(y[i] = 1 \mid S)$ for $i = 1, \ldots, k$. ($[i]$ denotes the i-th entry of the corresponding vector.)

Solution: The loss function is defineed to be:

$$\begin{aligned} l(y, p_T) &= -\sum_{i=1}^{k} y[i] log p_T[i] \\ &= -log(p_T[i_{true}]) \end{aligned}$$

where $p_T[i_{true}]$ is the predicted probability for the true class corresponding to $y[i_{true}] = 1$.

2. To perform backpropagation, we need to compute the derivative of the loss with respect to each parameter. Without loss of generality, let's consider the derivative with respect to a single parameter $w = W_{\text{hh}}[i, j]$ where $[i, j]$ denotes the (i, j)-th entry of the matrix. By chain rule, we have

$$\frac{\partial \ell}{\partial w} = \frac{\partial \ell}{\partial o_t} \frac{\partial o_t}{\partial h_t} \frac{\partial h_t}{\partial w} \tag{4}$$

Note that the first two derivatives in the 4 are easy to compute, so let's focus on the last term $\frac{\partial h_t}{\partial w}$. During the lecture, we have shown that

$$\frac{\partial h_t}{\partial w} = \sum_{i=1}^{t} \frac{\partial h_t}{\partial h_i} \frac{\partial h_i^+}{\partial w} \tag{5}$$

Here $\frac{\partial h_i^+}{\partial w}$ denotes the "immediate" gradient where h_{i-1} is taken as a constant.

(a) (1 point, written) Give an expression for $\frac{\partial h_i^+}{\partial w}$.

Solution: Give that we have:

$$z_t = W_{hh} h_{t-1} + W_{ih} x_t + b_h$$
$$h_t = \sigma(z_t)$$

We compute:

$$\frac{\partial h_k^+}{\partial \omega} = \frac{\partial h_k^+}{\partial z_k} \cdot \frac{\partial z_k}{\partial \omega}$$
$$= \begin{cases} h_{k-1,j} \cdot \left[\text{diag}\left(1 - \sigma(z_k)\right)\right]_{:,m}, & m = i \\ 0, & \text{otherwise} \end{cases}$$

where

i. $z_{k,i}$ is the i-th entry of vector z_k $h_{k-1,j}$ is the j-th entry of vector h_{k-1}

ii. ":, m" denotes the m-th column of the diagonal matrix $\text{diag}\left(\mathbf{1} - \sigma(z_k)\right) \in R^{d' \times d'}$ where $\mathbf{1} \in R^{d'}, \sigma(z_k) \in R^{d'}$

iii. $h_{k-1,j} \cdot [\text{diag}(1 - \sigma(z_k)]_{:,m}$ denotes a scalar -vector multiplication. where $h_{k-1,j}$ is applied to every element in the following vector.

(b) (2 points, written) Expand the gradient vector $\frac{\partial h_t}{\partial h_i}$ using the chain rule as a product of partial derivatives of one hidden state with respect to the previous hidden state. You do not need to explicitly do differentiations beyond that.

Solution: Given that we have:

$$z_t = W_{hh} h_{t-1} + W_{ih} x_t + b_h$$
$$h_t = \sigma(z_t)$$

We compute.

$$\frac{\partial h_t}{\partial h_i} = \prod_{k=1}^{t-i} \frac{\partial h_{t-k+1}}{\partial h_{t-k}}$$

3. (2 points, written) Now let's further expand one of the partial derivatives from the previous question. Write down the Jacobian matrix $\frac{\partial h_{i+1}}{\partial h_i}$ by rules of differentiations. You can directly use σ' as the derivative of the activateion function in the expression.

Solution: Give that we have:

$$z_t = W_{hn} h_{t-1} + W_{ih} X_t + b_h$$
$$h_t = \sigma(z_t)$$

We compute.

$$\frac{\partial h_{i+1}}{\partial h_i} = \frac{\partial h_{i+1}}{\partial z_{i+1}} \cdot \frac{\partial z_{i+1}}{\partial h_i}$$
$$= \operatorname{diag}\left(\sigma'(z_{i+1})\right) \cdot W_{hh}$$
$$\text{where } \sigma'(z_{i+1}) = \left(1 - \sigma^2(z_{i+1})\right) \in \mathbb{R}^{d' \times d'}$$

and $\sigma^2(z_{i+1})$ is element-wise square of vector $\sigma(z_{i+1})$.

1.2 Bounding Gradient Norm

To determine if the gradient will vanish or explode, we need a notion of magnitude. For the Jacobian matrix, we can use the induced matrix norm (or operator norm). For this question, we use the spectral norm $\|A\|_2 = \sqrt{\lambda_{\max}(A^\top A)} = s_{\max}(A)$ for a matrix $A \in \mathbb{R}^{m \times n}$. Here $\lambda_{\max}(A^\top A)$ denotes the maximum eigenvalue of the matrix $A^\top A$ and $s_{\max}(A)$ denotes the maximum singular value of the matrix A. You can learn more about matrix norms at this Wikipedia entry.
Now, to determine if the gradient $\frac{\partial \ell}{\partial w}$ will vanish or explode, we can focus on $\|\frac{\partial h_t}{\partial h_i}\|$. Note that if $\|\frac{\partial h_t}{\partial h_i}\|$ vanishes or explodes, $\|\frac{\partial \ell}{\partial w}\|$ also vanishes or explodes based on (4) and (5).

1. (2 points, written) Given the mathematical form of the Jacobian matrix $\frac{\partial h_{i+1}}{\partial h_i}$ we derived earlier, we can now bound the norm of the Jacobian with the following matrix norm inequality

$$\|AB\|_2 \leq \|A\|_2 \cdot \|B\|_2 \tag{6}$$

for matrices A, B with matched shapes. Write down a bound for $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2$.

Solution: Given the previous results:

$$\begin{aligned}
\frac{\partial h_{i+1}}{\partial h_i} &= \frac{\partial h_{i+1}}{\partial z_{i+1}} \cdot \frac{\partial z_{i+1}}{\partial h_i} \\
&= \operatorname{diag}\left(\sigma'\left(z_{i+1}\right)\right) \cdot W_{hh} \\
\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 &= \left\| \frac{\partial h_i}{\partial z_i} \cdot \frac{\partial z_i}{\partial h_{i-1}} \right\|_2 \\
&\leq \left\|\operatorname{diag}\left(\sigma'\left(z_i\right)\right)\right\|_2 \left\|W_{hh}\right\|_2 \\
&= \mathrm{s}_{\max}\left(\operatorname{diag}\left(\sigma'\left(z_i\right)\right)\right) \cdot s_{\max}\left(W_{hh}\right)
\end{aligned}$$

Where $\operatorname{diag}\left(\sigma'\left(z_i\right)\right)$, Whin $\in R^{d' \times d'}$:

2. (4 points, written) Now we have all the pieces we need. Derive a bound on the gradient norm $\|\frac{\partial h_t}{\partial h_i}\|$. Explain how the magnitude of the maximum singular value of W_{hh} can lead to either vanishing or exploding gradient problems. [**HINT:** You can use the fact that for the tanh activation function $\sigma(\cdot)$, the derivative $\sigma'(\cdot)$ is always less than or equal to 1.]

Solution: Given the previous results, we have:

$$\begin{aligned}\frac{\partial h_t}{\partial h_i} &= \prod_{k=1}^{t-i} \frac{\partial h_{t-k+1}}{\partial h_{t-k}} \\ &= \prod_{k=1}^{t-i} \text{diag}\left(\sigma'\left(z_{t-k}\right)\right) \cdot W_{hh}\end{aligned}$$

Thus

$$\begin{aligned}\left\|\frac{\partial ht}{\partial h_i}\right\|_2 &\leq \prod_{k=1}^{t-i} \left\|\text{diag}\left(\sigma'\left(z_{t-k}\right)\right)\right\|_2 \prod_{k=1}^{t-i} \|W_{hh}\|_2 \\ &= \prod_{k=1}^{t-i} \text{s}_{\max}\left(\text{diag}\left(\sigma'\left(z_{t-k}\right)\right)\right) \prod_{k=1}^{t-i} \text{s}_{\max}\left(W_{hh}\right)\end{aligned}$$

$\because \text{diag}\left(\sigma'\left(z_{t-k}\right)\right)$ only has diagonal terms which are always less than or equal to 1

$$\therefore s_{\max}\left(\text{diag}\left(\sigma'\left(z_{t-k}\right)\right) \leq 1\right.$$

Thus

$$\left\|\frac{\partial h_t}{\partial h_i}\right\| \leq \prod_{k=1}^{t-i} s_{\max}\left(W_{hh}\right)$$

$\therefore$ If $s_{max}\left(W_{hh}\right) < 1$, the gradient will vanish. If $\text{s}_{\max}\left(W_{\text{hh}}\right) > 1$, the gradient will explode.

3. (1 point, written) Propose one way to get around the vanishing and exploding gradient problem.

Solution: From the lecture we know that, we could:

(a) Reduce the number of repeated multiplication by truncating after k steps so that (h_{t-k} has no influence on h_t).

(b) Limit the norm or value of the gradient in each step by clipping the gradient to mitigate the gradient explosion.

2 Machine Translation

The goal of this homework is to build a machine translation system using sequence-to-sequence transformer models `https://arxiv.org/abs/1706.03762`. More specifically, you will build a system which translates German to English using the Multi30k dataset (`https://arxiv.org/abs/1605.00459`) You are provided with a code skeleton, which clearly marks out where you need to fill in code for each sub-question.

First go through the file `README.md` to set up the environment required for the class.

2.1 Attention

Transformers use scaled dot-product attention — given a set of queries Q (each of dimension d_k), a set of keys K (also each dimension d_k), and a set of values V (each of dimension d_v), the output is a weighted sum of the values. More specifically,

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{7}$$

Note that each of Q, K, V is a matrix of vectors packed together.

1. (2 points, written) The above function is called 'scaled' attention due to the scaling factor $\frac{1}{\sqrt{d_k}}$. The original transformers paper mentions that this is needed because dot products between keys and queries get large with larger d_k.

   For a query q and key k both of dimension d_k and each component being an independent random variable with mean 0 and variance 1, compute the mean and variance (with steps) of the dot product $q.k$ to demonstrate the point.

Solution: Given the assumption that $\mathbb{E}[q_i] = \mathbb{E}[k_i] = 0, \mathbb{V}[q_i] = \mathbb{V}[k_i] = 1$, we know that the mean is computed as:

$$\begin{aligned}
\mathbb{E}[\vec{q}^\top \vec{k}] &= \sum_{i=1}^{d_k} \mathbb{E}[q_i \cdot k_i] \\
&= \sum_{i=1}^{d_k} \mathbb{E}[q_i]\mathbb{E}[k_i] && \text{(Indepdence Assumption)} \\
&= 0
\end{aligned}$$

The variance is computed as:

$$
\begin{aligned}
\mathbb{V}[\vec{q}^\top \vec{k}] &= \sum_{i=1}^{d_k} \mathbb{V}[q_i \cdot k_i] + 2 \sum_{i \neq j, i<j} Cov(q_i k_i, q_j k_j) \\
&= \sum_{i=1}^{d_k} \mathbb{V}[q_i \cdot k_i] && \text{(Indepdence Assumption)} \\
&= \sum_{i=1}^{d_k} \mathbb{E}[(q_i \cdot k_i)^2] - (\mathbb{E}[q_i k_i])^2 \\
&= \sum_{i=1}^{d_k} \mathbb{E}[q_i^2] \cdot \mathbb{E}[k_i^2] && \text{(Indepdence Assumption)} \\
&= \sum_{i=1}^{d_k} 1 \\
&= d_k
\end{aligned}
$$

2. (2 points, coding) Implement the above scaled dot-product attention in the `attention()` function present in `layers.py`. You can test the implementation after the next part.

Solution: See the corresponding coding implementations.

3. (2 point, coding) In this part, you will modify the `attention()` function by making use of the parameters `mask` and `dropout` which were input to the function. The `mask` indicates positions where the attention values should be zero (e.g. when we have padded a sentence of length 5 to length 10, we do not want to attend to the extra tokens). `dropout` should be applied to the attention weights for regularization.

   To test the implementation against some unit tests, run `python3 test.py --attention.`

Solution: See the corresponding coding implementations.

4. (3 points, coding) Instead of a single attention function, transformers use multi-headed attention function. For original keys, queries and values (each of dimension say d_{model}), we use h different projection matrices to obtain queries, keys and values of dimensions d_k, d_k and d_v respectively. Implement the function `MultiHeadedAttention()` in `layers.py`. To test the implementation against some unit tests, run `python3 test.py --multiheaded_attention.`

Solution: See the corresponding coding implementations.

2.2 Positional Encoding

Since the underlying blocks in a transformer (namely attention and feed forward layers) do not encode any information about the order of the input tokens, transformers use 'positional encodings' which are added to the input embeddings. If d_{model} is the dimension of the input embeddings, pos is the position, and i is the dimension, then the encoding is defined as:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \tag{8}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}) \tag{9}$$

1. (2 points, written) Since the objective of the positional encoding is to add information about the position, can we simply use $PE_{pos} = sin(pos)$ as the positional encoding for pos position? Why or why not?

Solution: There are at least two reasons why we cannot simply use $PE_{pos} = sin(pos)$:

- This will cause **non-uniqueness** of the positional embedding across the position direction, which will limit the model's understanding of sequence length and position. For example, if our training set consists of 100 tokens, then the value at the 101-th position would repeat the same value as the first position. Due to the periodic nature of the sinusodal functions, the positional encoding for the 1st token and 101-th token would be arbitrarily close to each other. And this is one of the reasons why we want to introduce multi-dimensional positional encoding as described in the research paper.
- In many literature, there is a critical aspect of positional encoding: the ability to predict the encoding for a position slightly different from one we've explicitly seen and this should be a straightforward operation like linear transformation(e.g. $PE(pos + \Delta pos) = PE(pos)T(pos)$, where T is a linear transformation). Using single $sin(pos)$ will fail to reach this since $sin(x+\delta x) \approx sin(x) + \delta x * cos(x)$, which is not a linear function of single $sin(x)$. This will cause the model **fail to easily learn to attend by relative positions.**

2. (2 points, coding) Implement the above positional encoding in the function `PositionalEncoding()` in the file `utils.py`. To test the implementation against some unit tests, run `python3 test.py --positional_encoding`.

Solution: See the corresponding coding implementations.

2.3 Training

1. (2 points, written) The above questions should complete the missing parts in the training code and we can now train a machine translation system!

   Use the command `python3 main.py` to train your model. For the purpose of this homework, you are not required to tune any hyperparameters. You should submit the generated `out_greedy.txt` file containing outputs. You must obtain a BLEU score of atleast 35 for full points (By default we are using BLEU-4 for this and all subsequent questions).

Solution: The BLEU score is 38.448482417549926. See the out_greedy.txt file.

2.4 Decoding & Evaluation

In the previous question, the code uses the default `greedy_decode()` to decode the output. In practice, people use algorithms such as beam search decoding, which have been shown to give better quality outputs. (Note: In the following questions, use a model trained with the default i.e. given hyperparameters)

1. (2 points, written) In the file `utils.py` you will notice a function `subsequent_mask()`. What does that function do and why is it required in our model?

Solution: In the context of a transformer model, the **self-attention** mechanism allows each token to attend to every other token in the sequence. This is fine with encoder part. However, when generating text in decoder where future information should not be available, this mask ensures that the model only attends to the current and past tokens, not future ones. This is crucial for tasks like language modeling, where the next token is predicted based on previous tokens.

2. (5 points, coding) Implement the `beam_search()` function in `utils.py`. We have provided the main skeleton for this function and you are only required to fill in the important parts (more details in the code). You can run the code using the arguments `--beam_search` and `--beam_size`. You should submit the generated file `out_beam.txt` when `beam_size = 2`.

   To test the implementation against some unit tests, run `python3 test.py --beam_search`.

Solution: See the corresponding file out_beam.txt.

3. (3 points, written) For the model trained in question 1.3, plot the BLEU score as a function of beam size. You should plot the output from beam size 1 to 5. Is the trend as expected? Explain your answer.

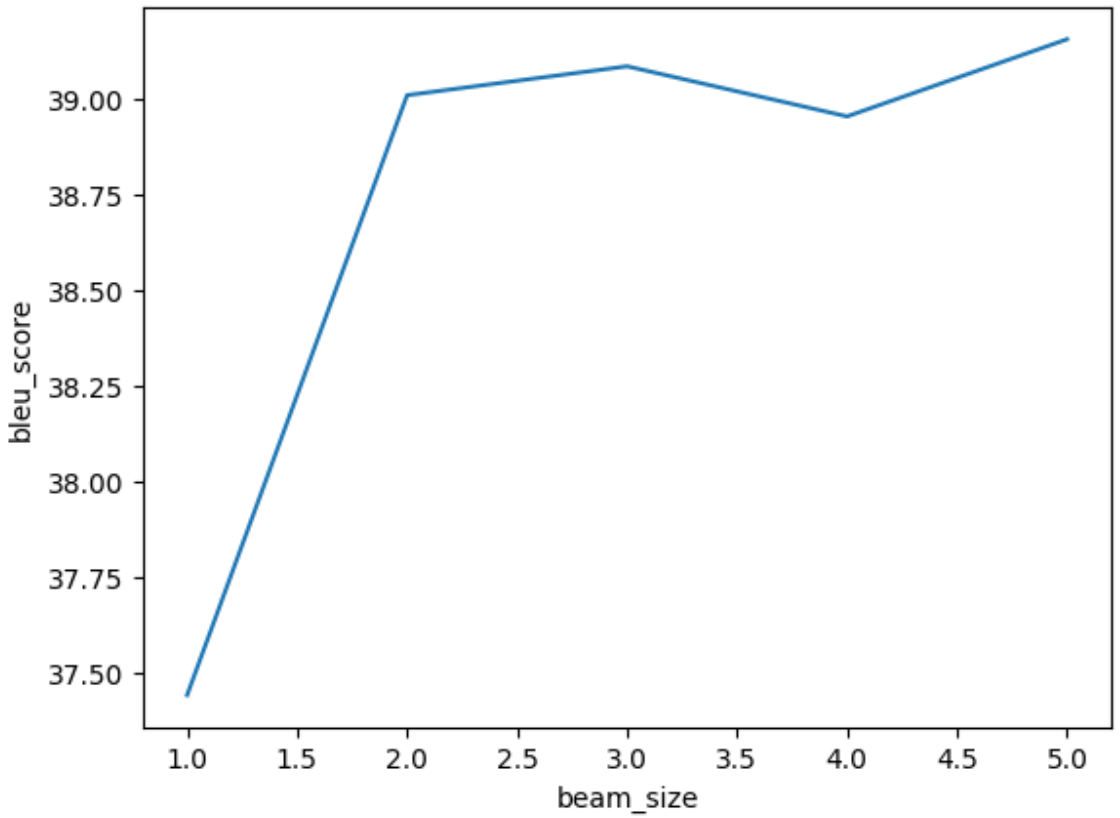

Figure 1: BLEU Score Plot

Explanations: This is expected for the following observations:

(a) When beam_size is 1, it is effectively greedy decoding, which might not always yield the best overall sequence.

(b) When beamsize is larger(from 1 to 2), it allows the model to explore more possible sequences and should improve the quality of the generated output up to a point.

(c) However, when the size grows from 3 to 4, we notice a drop, this could be because the model starts to explore the lower probability portion, which could decrease the BLEU score.

4. (2 points, written) You might notice that some of the sentences contain the '⟨unk⟩' token which denotes a word not in the vocabulary. For systems such as Google Translate, you might not want such tokens in the outputs seen by the user. Describe a potential way to avoid (or reduce) the occurrence of these tokens in the output.

Solution: A potential way would be replace the current tokenizer(Spacy tokenizer, which is only word-based) with a subword tokenizer like sentencePiece tokenizer from Huggingfacec (which is a subword-based tokenizer). The reason is that most of words can be represented by the combination of those subwords, which doesn't require exact matching/representation. Alternatively, we could just increase the vocabulary size to mitigate the lack of tokenization representation.

5. (2 points, written) In this homework, you have been using BLEU score as your evaluation metric. Consider an example where the reference translation is "I just went to the mall to buy a table.", and consider two possible model generations: "I just went to the mall to buy a knife." and "I just went to the mall to buy a desk.". Which one will BLEU score higher? Suggest a potential fix if it does not score the intended one higher.

Solution: Both candidates are likely to get similar BLEU scores since it will treat "desk" and "knife" equally in that BLEU only considers spatial information of words(n-gram) instead of semantic meaning. One possible fix would be to use BERTScore metric mentioned in the slide to consider the semantic similarities between "desk" and "table" so that this candidate can be scored higher.