Due Date: November 14th, 2023 at 11:00 pm

Instructions

- *For all questions, show your work!*
- *Use LaTeX and the template we provide when writing your answers. You may reuse most of the notation shorthands, equations and/or tables. See the assignment policy on the course website for more details.*
- *The use of AI tools like Chat-GPT to find answers or parts of answers for any question in this assignment is not allowed. However, you can use these tools to improve the quality of your writing, like fixing grammar or making it more understandable. If you do use these tools, you must clearly explain how you used them and which questions or parts of questions you applied them to. Failing to do so or using these tools to find answers or parts of answers may result in your work being completely rejected, which means you'll receive a score of 0 for the entire theory or practical section.*
- *Submit your answers electronically via Gradescope.*
- *TAs for this assignment are* ***Thomas Jiralerspong, Saba Ahmadi, and Shuo Zhang.***

Question 1 (3)**.** (**Normalization**) This question is about normalization techniques.

Batch normalization, layer normalization and instance normalization all involve calculating the mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma^2}$ with respect to different subsets of the tensor dimensions. Given the following 3D tensor, calculate the corresponding mean and variance tensors for each normalization technique: $\boldsymbol{\mu}_{batch}$, $\boldsymbol{\mu}_{layer}$, $\boldsymbol{\mu}_{instance}$, $\boldsymbol{\sigma}^2_{batch}$, $\boldsymbol{\sigma}^2_{layer}$, and $\boldsymbol{\sigma}^2_{instance}$.

$$\left[\begin{bmatrix}2,4,3\\2,1,2\end{bmatrix},\begin{bmatrix}1,2,3\\4,1,1\end{bmatrix},\begin{bmatrix}2,1,3\\3,2,4\end{bmatrix},\begin{bmatrix}1,4,2\\2,4,3\end{bmatrix}\right]$$

The size of this tensor is 4 x 2 x 3 which corresponds to the batch size, number of channels, and number of features respectively.

Answer 1. Batch Normalization:
As we know we should Normalize over the entire batch and all features so based on the question's assumption ($4 \times 2 \times 3$) we have (only one batch, size of 4 but as mentioned we have 2 channels): the mean, μ_{batch}, is computed by :

$$\mu_{\text{batchchannel1}} = \frac{1}{12}(2+4+3+1+2+3+2+1+3+1+4+2) \tag{1}$$

$$= \frac{57}{24} \approx 2.333 \tag{2}$$

For the given batch and channel 1, the variance ($\sigma^2_{\text{batchchannel1}}$) is:

As we know,

$$\sigma^2_{\text{batchchannel1}} = \frac{1}{12}\sum_{i=1}^{12}(x_i - \mu_{\text{batchchannel1}})^2 \tag{3}$$

$$\sigma^2_{\text{batchchannel1}} = \frac{1}{12}\big[(2-2.333)^2 + (4-2.333)^2 + (3-2.333)^2 + (1-2.333)^2 + (2-2.333)^2 \tag{4}$$

$$+ (3-2.333)^2 + (2-2.333)^2 + (1-2.333)^2 + (3-2.333)^2 + (1-2.333)^2 \tag{5}$$

$$+ (4-2.333)^2 + (2-2.333)^2\big] = 1.055 \tag{6}$$

And now for the second channel:

$$\mu_{\text{batchchannel2}} = \frac{1}{12}(2+1+2+4+1+1+3+2+4+2+4+3) \tag{7}$$

$$= \frac{57}{24} \approx 2.4166 \tag{8}$$

For the given batch and channel 2, the variance ($\sigma^2_{\text{batchchannel2}}$) is:
As we know,

$$\sigma^2_{\text{batchchannel2}} = \frac{1}{12}\sum_{i=1}^{12}(x_i - \mu_{\text{batchchannel2}})^2 \tag{9}$$

$$\sigma^2_{\text{batchchannel2}} = \frac{1}{12}\big[(2-2.416)^2 + (1-2.416)^2 + (2-2.416)^2 + (4-2.416)^2 + (1-2.416)^2 \tag{10}$$

$$+ (1-2.416)^2 + (3-2.416)^2 + (2-2.416)^2 + (4-2.416)^2 + (2-2.416)^2 \tag{11}$$

$$+ (4-2.416)^2 + (3-2.416)^2\big] = 1.243 \tag{12}$$

$$\text{So, } \mu_{\text{batch}} \text{ is } \begin{bmatrix}2.333\\2.416\end{bmatrix}, \text{ and } \sigma^2_{\text{batch}} \text{ is } \begin{bmatrix}1.055\\1.243\end{bmatrix}$$

Layer Normalization:
As we know we should Normalize over the each layer so based on the question's assumption we have (4 layers):

$$\mu_{\text{Layer1}} = \frac{2+4+3+2+1+2}{6} = \frac{14}{6} \approx 2.33$$

$$\sigma^2_{\text{Layer1}} = \frac{1}{6}\left((2-2.33)^2 + (4-2.33)^2 + (3-2.33)^2 + (2-2.33)^2 + (1-2.33)^2 + (2-2.33)^2\right) = 0.8889$$

$$\mu_{\text{Layer2}} = \frac{1+2+3+4+1+1}{6} = \frac{14}{6} = 2$$

$$\sigma^2_{\text{Layer2}} = \frac{1}{6}\left((1-2)^2 + (2-2)^2 + (3-2)^2 + (4-2)^2 + (1-2)^2 + (1-2)^2\right) = 1.333$$

$$\mu_{\text{Layer3}} = \frac{2+1+3+3+2+4}{6} = \frac{14}{6} = 2.5$$

$$\sigma^2_{\text{Layer3}} = \frac{1}{6}\left((2-2.5)^2 + (1-2.5)^2 + (3-2.5)^2 + (3-2.5)^2 + (2-2.5)^2 + (4-2.5)^2\right) = 0.9166$$

$$\mu_{\text{Layer4}} = \frac{1+4+2+2+4+3}{6} = \frac{14}{6} = 2.6667$$

$$\sigma^2_{\text{Layer4}} = \frac{1}{6}\left((1-2.66)^2 + (4-2.66)^2 + (2-2.66)^2 + (2-2.66)^2 + (4-2.66)^2 + (3-2.66)^2\right) = 1.22$$

$$\text{So, } \mu_{\text{layer}} \text{ is } \begin{bmatrix} 2.33 \\ 2 \\ 2.5 \\ 2.6667 \end{bmatrix}, \text{ and } \sigma^2_{\text{layer}} \text{ is } \begin{bmatrix} 0.8889 \\ 1.333 \\ 0.9166 \\ 1.22 \end{bmatrix}$$

Instance Normalization:
Given that our data size $(4 \times 2 \times 3)$ comprises 4 instances (representing the batch size) and 2 channels, we should calculate mean and variance separately for each of the 2 channels, taking into account all 4 instances, so we have(8 instances):

$$\mu_{instace1,channel1} = \frac{2+4+3}{3} = 3$$

$$\sigma^2_{instace1,channel1} = \frac{(2-3)^2 + (4-3)^2 + (3-3)^2}{3} = 0.6667$$

$$\mu_{instace1,channel2} = \frac{2+1+2}{3} = 1.6667$$

$$\sigma^2_{instace1,channel2} = \frac{(2-1.6667)^2 + (1-1.6667)^2 + (2-1.6667)^2}{3} = 0.2222$$

$$\mu_{instace2,channel1} = \frac{1+2+3}{3} = 2$$
$$\sigma^2_{instace2,channel1} = \frac{(1-2)^2+(2-2)^2+(3-2)^2}{3} = 0.6667$$

$$\mu_{instace2,channel2} = \frac{4+1+1}{3} = 2$$
$$\sigma^2_{instace2,channel2} = \frac{(4-2)^2+(1-2)^2+(1-2)^2}{3} = 2$$

$$\mu_{instace3,channel1} = \frac{2+1+3}{3} = 2$$
$$\sigma^2_{instace3,channel1} = \frac{(2-2)^2+(1-2)^2+(3-2)^2}{3} = 0.6667$$

$$\mu_{instace3,channel2} = \frac{3+2+4}{3} = 3$$
$$\sigma^2_{instace3,channel2} = \frac{(3-3)^2+(2-3)^2+(4-3)^2}{3} = 0.6667$$

$$\mu_{instace4,channel1} = \frac{1+4+2}{3} = 2.3333$$
$$\sigma^2_{instace4,channel1} = \frac{(1-2.3333)^2+(4-2.3333)^2+(2-2.3333)^2}{3} = 1.5556$$

$$\mu_{instace4,channel2} = \frac{2+4+3}{3} = 3$$
$$\sigma^2_{instace4,channel2} = \frac{(2-3)^2+(4-3)^2+(3-3)^2}{3} = 0.6666$$

So, μ_{instance} is $\begin{bmatrix} 3, 1.6667 \\ 2, 2 \\ 2, 3 \\ 2.333, 3 \end{bmatrix}$, and $\sigma^2_{\text{instance}}$ is $\begin{bmatrix} 0.6667, 0.2222 \\ 0.6667, 2 \\ 0.6667, 0.6667 \\ 1.5556, 0.6667 \end{bmatrix}$

Note: I used Grammerly and ChatGPT to improve the quality of my writing.

Question 2 (4-6-6)**.** (**Regularization**) In this question, you will reconcile the relationship between L2 regularization and weight decay for the Stochastic Gradient Descent (SGD) and Adam optimizers. Imagine you are training a neural network (with learnable weights θ) with a loss function $L(f(\mathbf{x}^{(i)}, \theta), \mathbf{y}^{(i)})$, under two different schemes. The *weight decay* scheme uses a modified SGD update rule: the weights θ decay exponentially by a factor of λ. That is, the weights at iteration $i+1$ are computed as

$$\theta_{i+1} = \theta_i - \eta \frac{\partial L(f(\mathbf{x}^{(i)}, \theta_i), \mathbf{y}^{(i)})}{\partial \theta_i} - \lambda \theta_i$$

where η is the learning rate of the SGD optimizer. The *L2 regularization* scheme instead modifies the loss function (while maintaining the typical SGD or Adam update rules). The modified loss function is

$$L_{\text{reg}}(f(\mathbf{x}^{(i)}, \theta), \mathbf{y}^{(i)}) = L(f(\mathbf{x}^{(i)}, \theta), \mathbf{y}^{(i)}) + \gamma \|\theta\|_2^2$$

2.1 Prove that the *weight decay* scheme that employs the modified SGD update is identical to an *L2 regularization* scheme that employs a standard SGD update rule.

2.2 Consider a "decoupled" weight decay scheme for the Adam algorithm (see lecture slides) with the following two update rules.

- The **Adam-L2-reg** scheme computes the update by employing an L2 regularization scheme (same as the question above).
- The **Adam-weight-decay** scheme computes the update as $\boldsymbol{\Delta}\theta = -\left(\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}}+\delta} + \lambda\theta\right)$.

Now, assume that the neural network weights can be partitioned into two disjoint sets based on their gradient magnitude: $\theta = \{\theta_{\text{small}}, \theta_{\text{large}}\}$, where each weight $\theta_s \in \theta_{\text{small}}$ has a much smaller gradient magnitude than each weight $\theta_l \in \theta_{\text{large}}$. Using this information provided, answer the following questions. In each case, provide a brief explanation as to why your answer holds.

(a) Under the **Adam-L2-reg** scheme, which set of weights among θ_{small} and θ_{large} would you expect to be regularized (i.e., driven closer to zero) more strongly than the other ? Why ?

(b) Would your answer change for the **Adam-weight-decay** scheme ? Why/why not ?

(Note: for the two sub-parts above, we are interested in the rate at which the weights are regularized, *relative* to their initial magnitudes.)

2.3 In the context of all of the discussion above, argue that weight decay is a better scheme to employ as opposed to L2 regularization ; particularly in the context of adaptive gradient based optimizers. (Hint: think about how each of these schemes regularize each parameter, and also about what the overarching objective of regularization is).

Answer 2. Q2.1:

Based on question's assumptions,to demonstrate that the two schemes (weight decay and L2 regularization) are equivalent, it's necessary to establish that their respective update rules lead to identical outcomes (So we should provide derivative of the regularized loss function Lreg). Based on the given assumptions, for the Weight Decay scheme with modified SGD update, we have:

$$\theta_{i+1} = \theta_i - \eta \frac{\partial L(f(x^{(i)}, \theta), y^{(i)})}{\partial \theta_i} - \lambda \theta_i \tag{13}$$

On the other side for the L2 regularization with standard SGD update, the equation is:

$$L_{\text{reg}}(f(x^{(i)}, \theta), y^{(i)}) = L(f(x^{(i)}, \theta), y^{(i)}) + \gamma \|\theta\|^2 \tag{14}$$

So, the update rule for L2 regularization with standard SGD update is:

$$\theta_{i+1} = \theta_i - \eta \frac{\partial L_{\text{reg}}(f(x^{(i)}, \theta), y^{(i)})}{\partial \theta_i} \tag{15}$$

The derivative of the regularized loss function L_{reg} with respect to θ_i is expressed as follows:

$$\begin{aligned} \frac{\partial L_{\text{reg}}(f(x^{(i)}, \theta), y^{(i)})}{\partial \theta_i} &= \frac{\partial L(f(x^{(i)}, \theta), y^{(i)})}{\partial \theta_i} + \frac{\partial \gamma \|\theta\|^2}{\partial \theta_i} \\ &= \frac{\partial L(f(x^{(i)}, \theta), y^{(i)})}{\partial \theta_i} + 2\gamma\theta_i \end{aligned}$$

Substitute the derived gradient into the SGD update equation for L2 regularization:

$$\theta_{i+1} = \theta_i - \eta \left(\frac{\partial L(f(x^{(i)}, \theta), y^{(i)})}{\partial \theta_i} + 2\gamma\theta_i \right) = \tag{16}$$

$$\theta_{i+1} = \theta_i - \eta \frac{\partial L(f(x^{(i)}, \theta), y^{(i)})}{\partial \theta_i} - 2\eta\gamma\theta_i \tag{17}$$

For the two schemes to be equivalent, the following condition must be satisfied:

$$2\eta\gamma = \lambda \tag{18}$$

So we have:

$$\gamma = \frac{\lambda}{2\eta} \tag{19}$$

Therefore, based on this established connection between λ and γ, it can be concluded that the weight decay approach using the modified SGD update is effectively the same as the L2 regularization approach utilizing the standard SGD update rule.

Q2.2.a:
The derivative of the loss function incorporating L_2 regularization is represented by:

$$\nabla_\theta L_{\text{reg}} = \nabla_\theta L(f(x_i), y_i) + 2\gamma\theta$$

For convenience, we denote this combined gradient as $G_{\text{reg}} = G + G_{L2}$, where G is the loss gradient and G_{L2} is the gradient from the L2 penalty relative to the parameters.

The Adam optimizer maintains two momentum vectors (m and v). When incorporating $L2$ regularization into Adam, these vectors are computed as:

$$m = \beta_1 m + (1 - \beta_1) G_{\text{reg}} = \beta_1 m + (1 - \beta_1)(G + G_{L2})$$

$$v = \beta_2 v + (1-\beta_2) G_{\text{reg}}^2 = \beta_2 v + (1-\beta_2)(G + G_{L2})^2$$

Adam includes steps for bias correction which are omitted here for brevity. The modified update equation is thus:

$$\theta_{\text{new}} = \theta - \frac{\alpha}{\sqrt{v} + \eta} \left(\beta_1 m + (1-\beta_1) G + (1-\beta_1) G_{L2}\right)$$

Defining m' and v' as the momentum terms from standard Adam without L_2 regularization, the update mechanism is given by:

$$\theta = \theta_{\text{old}} - \frac{\alpha}{\sqrt{v'} + \eta} \left(m' + (1-\beta_1) G_{L2}\right)$$

This diverges from the standard update formula used by Adam since it incorporates an additional factor in the denominator that corresponds to GG (the magnitudes of the gradients). From this, it can be deduced that an increase in the gradient magnitude (GG) leads to a reduction in the update velocity. Consequently, when applying the L2 norm, the weights with greater values, denoted as θ_{large}, are subjected to a comparatively weaker regularization effect.

Q2.2.b:
As previously outlined, the update equation for Adam incorporating weight decay is presented as:

$$\Delta\theta = -\left(\frac{\epsilon \cdot s}{\sqrt{r} + \delta} + \lambda\theta\right),$$

This expression reflects a synergy of adaptive updates with direct weight regularization. The term subtracted from the weights during the update is proportional to the weights' current value, denoted by $\lambda\theta$, and is not influenced by the gradient magnitudes. This is because the adaptive aspect is captured by s and r within the Adam algorithm. Given the context of the question, which implies normalization with respect to the initial size of the weights, the regularization rate applied to the weights is constant, effectively λ times the value of θ itself. This ensures that all weights, regardless of their starting magnitude, undergo regularization at a consistent rate.

Q2.3: As I explained before, L2 regularization introduces an additional term in the loss function, dependent on the square of the weights' magnitudes. In gradient-based optimization, the gradient of this term is proportional to the weight itself, leading each weight to shift towards zero in each update, proportional to its current value. Contrastingly, weight decay regularization moves each weight towards zero by a fixed proportion of its current value, irrespective of the gradient, decaying each weight by a constant factor. One key advantage of weight decay, particularly in adaptive gradient methods like Adam, is the decoupling from gradient magnitude. Adaptive methods adjust learning rates based on historical gradients, reducing learning rates for weights with high gradients, and increasing them for those with lower gradients. Weight decay applies a consistent regularization effect, unlike L2 regularization where the effect is gradient-coupled and can negate adaptive adjustments. Weight decay offers consistent regularization strength, is compatible with momentum-based

optimization, and simplifies hyperparameter tuning. Thus, although weight decay and L2 regularization both serve the purpose of regularizing model weights to avert overfitting, weight decay presents distinct benefits, especially when employed alongside adaptive gradient-based optimizers.

Note: I used Grammerly and ChatGPT to improve the quality of my writing.

Question 3 (1-1-4-6-2)**.** (**Decoding**) Suppose that we have a vocabulary containing N possible words, including a special token `<BOS>` to indicate the beginning of a sentence. Recall that in general, a language model with a full context can be written as

$$p(w_1, w_2, \ldots, w_T \mid w_0) = \prod_{t=1}^{T} p(w_t \mid w_0, \ldots, w_{t-1}).$$

We will use the notation $\boldsymbol{w}_{0:t-1}$ to denote the (partial) sequence $(w_0, \ldots, w_{t-1})$. Once we have a fully trained language model, we would like to generate realistic sequences of words from our language model, starting with our special token `<BOS>`. In particular, we might be interested in generating the most likely sequence $\boldsymbol{w}^{\star}_{1:T}$ under this model, defined as

$$\boldsymbol{w}^{\star}_{1:T} = \arg\max_{\boldsymbol{w}_{1:T}} p(\boldsymbol{w}_{1:T} \mid w_0 = \texttt{<BOS>}). \tag{20}$$

For clarity we will drop the explicit conditioning on w_0, assuming from now on that the sequences always start with the `<BOS>` token.

3.1 How many possible sequences of length $T + 1$ starting with the token `<BOS>` can be generated in total ? Give an exact expression, without the O notation. Note that the length $T + 1$ here includes the `<BOS>` token.

3.2 To determine the most likely sequence $\boldsymbol{w}^{\star}_{1:T}$, one could perform exhaustive enumeration of all possible sequences and select the one with the highest joint probability (as defined in equation 20). Comment on the feasibility of this approach. Is it scalable with vocabulary size ?

3.3 In light of the difficulties associated with exhaustive enumeration, it becomes essential to employ practical search strategies to obtain a reasonable approximation of the most likely sequence.

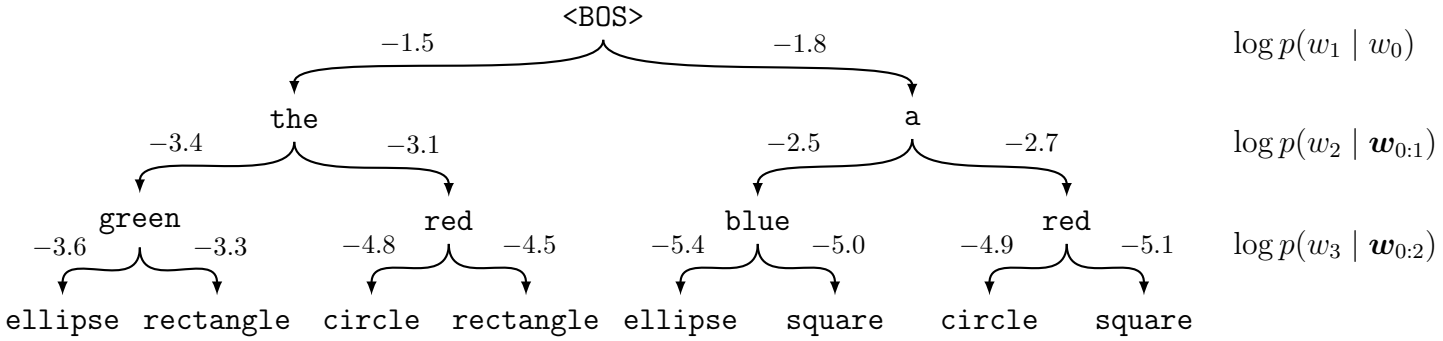
In order to generate B sequences having high likelihood, one can use a heuristic algorithm called *Beam search decoding*, whose pseudo-code is given in Algorithm 1 below

Algorithm 1: Beam search decoding

Input: A language model $p(\boldsymbol{w}_{1:T} \mid w_0)$, the beam width B
Output: B sequences $\boldsymbol{w}_{1:T}^{(b)}$ for $b \in \{1, \dots, B\}$
Initialization: $w_0^{(b)} \leftarrow$ `<BOS>` for all $b \in \{1, \dots, B\}$
Initial log-likelihoods: $l_0^{(b)} \leftarrow 0$ for all $b \in \{1, \dots, B\}$
for $t = 1$ **to** T **do**
  for $b = 1$ **to** B **do**
    for $j = 1$ **to** N **do**
      $s_b(j) \leftarrow l_{t-1}^{(b)} + \log p(w_t = j \mid \boldsymbol{w}_{0:t-1}^{(b)})$
  for $b = 1$ **to** B **do**
    Find (b', j) such that $s_{b'}(j)$ is the b-th largest score
    Save the partial sequence b': $\widetilde{\boldsymbol{w}}_{0:t-1}^{(b)} \leftarrow \boldsymbol{w}_{0:t-1}^{(b')}$
    Add the word j to the sequence b: $w_t^{(b)} \leftarrow j$
    Update the log-likelihood: $l_t^{(b)} \leftarrow s_{b'}(j)$
  Assign the partial sequences: $\boldsymbol{w}_{0:t-1}^{(b)} \leftarrow \widetilde{\boldsymbol{w}}_{0:t-1}^{(b)}$ for all $b \in \{1, \dots, B\}$

What is the time complexity of Algorithm 1 ? Its space complexity ? Write the algorithmic complexities using the O notation, as a function of T, B, and N. Is this a practical decoding algorithm when the size of the vocabulary is large ?

3.4 The different sequences that can be generated with a language model can be represented as a tree, where the nodes correspond to words and edges are labeled with the log-probability $\log p(w_t \mid \boldsymbol{w}_{0:t-1})$, depending on the path $\boldsymbol{w}_{0:t-1}$. In this question, consider the following language model (where the low probability paths have been removed for clarity)



3.4.a *Greedy decoding* is a simple algorithm where the next word $\overline{w}_t$ is selected by maximizing the conditional probability $p(w_t \mid \overline{\boldsymbol{w}}_{0:t-1})$ (with $\overline{w}_0 =$ `<BOS>`)

$$\overline{w}_t = \arg\max_{w_t} \log p(w_t \mid \overline{\boldsymbol{w}}_{0:t-1}).$$

Find $\overline{\boldsymbol{w}}_{1:3}$ using greedy decoding on this language model, and its log-likelihood $\log p(\overline{\boldsymbol{w}}_{1:3})$.

3.4.b Apply beam search decoding with a beam width $B = 2$ to this language model, and find $\boldsymbol{w}_{1:3}^{(1)}$ and $\boldsymbol{w}_{1:3}^{(2)}$, together with their respective log-likelihoods.

3.5 Please highlight the primary limitation that stands out to you for each of the discussed methods (greedy decoding and beam search).

Answer 3. Q3.1:
Based on question's assumptions a vocabulary with N distinct words and sequences initiating with the token <BOS>, the first slot in the sequence is invariably occupied by <BOS>, presenting only a single option for this position. Moving to the second slot, it can be filled with any of the N words from our vocabulary, yielding N possible choices. This pattern of N choices per slot persists consistently for all subsequent positions up to the $T + 1$. Therefore, when calculating the total number of feasible sequences of length $T + 1$ beginning with <BOS>, we multiply N for each slot beyond the initial one. This calculation results in N raised to the power of T, signifying that there are:

$$N^T$$

unique sequences of length $T + 1$ that commence with the token <BOS>.

Q3.2:
As I explianed in the previous question the approach of exhaustive enumeration, used to identify the most probable sequence $w_1^* : T$, involves calculating the joint probability for each of the N^T possible sequences and selecting the one with the highest probability. However, this approach faces significant practical challenges. First of all, computing the joint probability is time-consuming, as it involves calculating probabilities for each word in the sequence relative to its predecessors. On the other side, the computational complexity is enormous (N^T): the number of possible sequences increases exponentially with the sequence length T. Additionally, the memory required to temporarily store probabilities for each sequence is substantial. Owing to these challenges, the exhaustive enumeration approach is not scalable with increasing vocabulary size N, as the number of potential sequences N^T grows exponentially.

Q3.3:
Based on the Algorithm, first of all for the time complexity, the algorithm proceeds with an outer loop that iterates T times. Inside this loop, there are two crucial inner loops. The first of these runs B times, containing a further loop inside it that iterates N times. Inside this innermost loop, operations like calculating log-likelihood are performed, which have a constant time complexity, making the overall time complexity for this segment $O(B \times N)$. Similarly, the second inner loop, also running B times, involves a 'Find' operation (for finding the highest values) to determine the b-th largest score. Assuming a linear search is employed, this operation could take up to $O(B^2 \times N)$. Other operations in this loop, such as saving sequences and updating log-likelihoods, which also have a constant time complexity. Thus, this part of the algorithm also has a time complexity of $O(B^2 \times N)$. Considering that the outer loop runs for T iterations, the cumulative time complexity of the algorithm is:

$$O(T \times B^2 \times N)$$

On the other side, in terms of space complexity, the algorithm stores B sequences, each up to T in length, resulting in a space complexity of $O(B \times T)$. It also keeps a log-likelihood for each sequence, adding $O(B)$ to the space requirements. Additionally, it computes intermediate scores $s_b(j)$ for

$B \times N$ combinations, contributing another $O(B \times N)$ to the space complexity. Therefore, when these components are combined, the overall space complexity of the process is:

$$O(B \times (T + N))$$

As a result, in contrast to the exhaustive search method that exhibits a time complexity of $O(N^T)$, beam search offers markedly greater efficiency. Nonetheless, as the vocabulary size N expands, both the time and space complexities of beam search can escalate, particularly when the beam width B is also substantial. So, practically, beam search is frequently adopted due to its ability to balance computational efficiency with the quality of the search results.

Q3.4.a:
As we know, in the greedy approach, each word is selected based on the highest conditional probability. So, based on question's assumption, starting with w_0 = <BOS>, the next word, w_1, is chosen by evaluating the maximum log probability conditional on w_0. From <BOS>, the words <the> and <a> have log-probabilities of -1.5 and -1.8, respectively. With <the> having the higher log-probability, we set w_1 = <the>. Moving to w_2 given w_1, the options stemming from <the> are <green> and <red>, with log-probabilities of -3.4 and -3.1. Here, <red> has the higher log-probability, leading to w_2 = <red>. This process continues for w_3 given w_2, where from <red> the choices <circle> and <rectangle> have log-probabilities of -4.8 and -4.5. <rectangle> has the highest log-probability, thus w_3 = <ellipse>. Consequently, using greedy decoding in this language model results in the sequence "<BOS>, <the>, <red> and <rectangle>."

Q3.4.b:
For this question, we apply beam search decoding with a beam width of $B = 2$(based on question's assumption), along with their respective log-likelihoods. We start with w_0 = <BOS>, for w_1, our choices are "the" with a log-probability of -1.5 and "a" with -1.8. With our beam width set at 2, both paths are pursued. Proceeding to w_2 following w_1 = "the", we encounter options like "green" with a cumulative log-probability of -4.9 and "red" at -4.6. Similarly, for w_2 succeeding w_1 = "a", the alternatives are "blue" with a log-probability of -4.3 and "red" at -4.5. The foremost paths, considering the beam width, are <BOS> -> "a" -> "blue" with a log-probability of -4.3 and <BOS> -> "a" -> "red" at -4.5. Advancing to w_3 for these sequences, the path from <BOS> -> "a" -> "blue" leads to "square" with a log-probability of -9.3 and from <BOS> -> "a" -> "blue" leads to "ellipse" with a log-probability of -9.7 . From <BOS> -> "a" -> "red", we have choices such as "circle" at -9.4 and "square" at -9.6. The top sequences, as per the beam width (b=2), are "a blue square" with a log-likelihood of -9.3 and "a red circle" at -9.4.

Q3.5:
As we know, greedy decoding and beam search are both methods used in sequence generation. Greedy decoding quickly selects the most probable word at each step (egardless of the words ahead), but this may lead to suboptimal overall sequences due to its lack of foresight in considering future word impacts. Beam search, on the other hand, examines multiple paths simultaneously, but often generates similar sequences, especially with a smaller beam width. But, increasing the beam width for diversity can result in higher computational costs and more memory usage (because storing multiple paths at each step requires more memory, especially with a large beam width.), without necessarily enhancing result quality. Ultimately, both methods aim to balance computational efficiency with sequence quality but are not foolproof in finding the most optimal solutions.

Note: I used Grammerly and ChatGPT to improve the quality of my writing.

Question 4 (3-4-2-2)**.** (**RNN**) This question is about activation functions and vanishing/exploding gradients in recurrent neural networks (RNNs). Let $\sigma : \mathbb{R} \to \mathbb{R}$ be an activation function. When the argument is a vector, we apply σ element-wise. Consider the following recurrent unit:

$$\boldsymbol{h}_t = \sigma(\boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{b})$$

4.1 Show that applying the activation function in the following way: $\boldsymbol{g}_t = \boldsymbol{W}\sigma(\boldsymbol{g}_{t-1}) + \boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{b}$ results in an equivalent recurrence as that defined above (i.e. express $\boldsymbol{g}_t$ in terms of $\boldsymbol{h}_t$). More formally, you need to prove it using mathematical induction. You only need to prove the induction step in this question, assuming your expression holds for time step $t-1$.

4.2 Let $||\boldsymbol{A}||$ denote the L_2 operator norm [1] of matrix $\boldsymbol{A}$ ($||\boldsymbol{A}|| := \max_{\boldsymbol{x}:||\boldsymbol{x}||=1} ||\boldsymbol{A}\boldsymbol{x}||$). Assume $\sigma(x)$ has bounded derivative, i.e. $|\sigma'(x)| \leq \gamma$ for some $\gamma > 0$ and for all x. We denote as $\lambda_1(\cdot)$ the largest eigenvalue of a symmetric matrix. Show that if the largest eigenvalue of the weights is bounded by $\frac{\delta^2}{\gamma^2}$ for some $0 \leq \delta < 1$, gradients of the hidden state will vanish over time (here, use g_t as the definition of the hidden state), i.e.

$$\lambda_1(\boldsymbol{W}^\top \boldsymbol{W}) \leq \frac{\delta^2}{\gamma^2} \quad \Longrightarrow \quad \left|\left|\frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{h}_0}\right|\right| \to 0 \text{ as } T \to \infty$$

Use the following properties of the L_2 operator norm

$$||\boldsymbol{A}\boldsymbol{B}|| \leq ||\boldsymbol{A}||\,||\boldsymbol{B}|| \qquad \text{and} \qquad ||\boldsymbol{A}|| = \sqrt{\lambda_1(\boldsymbol{A}^\top \boldsymbol{A})}$$

4.3 What do you think will happen to the gradients of the hidden state if the condition in the previous question is reversed, i.e. if the largest eigenvalue of the weights is larger than $\frac{\delta^2}{\gamma^2}$? Is this condition *necessary* and/or *sufficient* for the gradient to explode ? If this condition is not sufficient, in what scenario is it not exploding given the condition is met ?(Answer in 1-2 sentences)

4.4 Assume we have the reverse problem of exploding gradient, what measures can we take to address it ? Propose 2 strategies where one takes into account the direction of the gradient and the other does not. Which is preferred according to you and why ?

Answer 4. Q4.1:
As mentioned, we should prove that $g_t = W\sigma(g_{t-1}) + Ux_t + b$ is equivalent to the recurrence $h_t = \sigma(Wh_{t-1} + Ux_t + b)$. So Assuming the expression is valid for time step $t-1$ (based on question's assumptions), we have our inductive hypothesis:

$$g_{t-1} = \sigma(Wh_{t-2} + Ux_{t-1} + b)$$

1. The L_2 operator norm of a matrix $\boldsymbol{A}$ is is an *induced norm* corresponding to the L_2 norm of vectors. You can try to prove the given properties as an exercise.

on the other side our objective is to prove that:

$$g_t = \sigma(Wh_{t-1} + Ux_t + b)$$

Starting with our given recurrence for g_t:

$$g_t = W\sigma(g_{t-1}) + Ux_t + b$$

We substitute g_{t-1} from our inductive hypothesis:

$$g_t = W\sigma(\sigma(Wh_{t-2} + Ux_{t-1} + b)) + Ux_t + b$$

We know that:

$$h_t = \sigma(Wh_{t-1} + Ux_t + b)$$

To establish equivalence, we need to show that the inner expression of σ for g_t matches that for h_t. From our hypothesis, we have:

$$h_{t-1} = \sigma(Wh_{t-2} + Ux_{t-1} + b)$$

Using this, our expression for g_t simplifies to:

$$g_t = Wh_{t-1} + Ux_t + b$$

Which aligns with the desired recurrence in terms of h_t.

Q4.2:
Assuming that $\lambda_1(W^TW) \leq \frac{\delta^2}{\gamma^2}$, where λ_1 denotes the largest eigenvalue of W^TW, and utilizing the properties of the L2 norm, for any vector v, the norm $\|Wv\|$ can be bounded by the square root of the largest eigenvalue of W^TW times the norm of v. This is due to the relationship between the operator norm and eigenvalues:

$$\|Wv\| = \sqrt{v^TW^TWv} \leq \sqrt{\lambda_1(W^TW)}\|v\|$$

Considering the bound $\lambda_1(W^TW) \leq \frac{\delta^2}{\gamma^2}$, we deduce:

$$\|Wv\| \leq \delta\gamma\|v\|$$

In the context of backpropagation through time for the gradient of the hidden state with respect to its initial state, we have:

$$\frac{\partial h_T}{\partial h_0} = \frac{\partial h_T}{\partial h_{T-1}} \times \frac{\partial h_{T-1}}{\partial h_{T-2}} \cdots \frac{\partial h_1}{\partial h_0}$$

Each term in this product is a Jacobian matrix of partial derivatives, represented as $W^T \cdot \text{diag}(\sigma'(Wh_{t-1} + Ux_t + b))$. Given that $\sigma'(x)$ is bounded by γ, the norm of each Jacobian is less than or equal to $\delta\gamma$.

The norm of the entire product can be bounded by:

$$\|\frac{\partial h_T}{\partial h_0}\| \leq (\delta\gamma)^T$$

With $0 \leq \delta < 1$, raising $\delta\gamma$ to the power of T results in the norm approaching zero as T goes to infinity. Thus, this proves that the norm of the gradient of the hidden state with respect to the initial state vanishes as T approaches infinity, under the condition that the largest eigenvalue of W^TW is suitably bounded.

Q4.3:
When the largest eigenvalue of the weights exceeds $\frac{\delta^2}{\gamma^2}$, there is a heightened risk of the hidden state gradients exploding. It's necessary but it's not sufficient as other factors like deep network architectures, certain activation functions (because output of activation function might be lesser), so can prevent the occurrence of exploding.

Q4.4:
For resolving the Exploding problem, we have two options:
Gradient Clipping: This approach sets an upper limit for the size of the gradient. If the gradient exceeds this predetermined threshold, its size is reduced to this maximum allowable limit. The process ensures that the weight adjustments don't become excessively large, but it doesn't adjust the gradient's direction. Weight Regularization methods like L1 or L2 regularization add a penalty to the loss function based on the size of the weight coefficients. The penalty is a function of the weights themselves and doesn't consider the direction of the gradient. L2 regularization, for example, adds a penalty term that is the square of the magnitude of the weights, encouraging the weights to be small and thus indirectly controlling the size of the gradient updates. I would prefer gradient clipping when specifically dealing with exploding gradients. Gradient clipping preserves the direction of the gradient, which means that the optimizer can still make progress in the correct direction,

even if the gradient's magnitude is reduced. This can be important for maintaining the integrity of the learning trajectory (so our process is more trackable).

Note: I used Grammerly and ChatGPT to improve the quality of my writing.

Question 5 (5-5-5-5). (**Paper Review: Show, Attend and Tell**) In this question, you are going to write a **one page review** of the Show, Attend and Tell paper. Your review should have the following four sections: Summary, Strengths, Weaknesses, and Reflections. For each of these sections, below we provide a set of questions you should ask about the paper as you read it. Then, discuss your thoughts about these questions in your review.

(5.1) **Summary:**

(a) What is this paper about ?
(b) What is the main contribution ?
(c) Describe the main approach and results. Just facts, no opinions yet.

(5.2) **Strengths:**

(a) Is there a new theoretical insight ?
(b) Or a significant empirical advance ? Did they solve a standing open problem ?
(c) Or a good formulation for a new problem ?
(d) Any good practical outcome (code, algorithm, etc) ?
(e) Are the experiments well executed ?
(f) Useful for the community in general ?

(5.3) **Weaknesses:**

(a) What can be done better ?
(b) Any missing baselines ? Missing datasets ?
(c) Any odd design choices in the algorithm not explained well ? Quality of writing ?
(d) Is there sufficient novelty in what they propose ? Minor variation of previous work ?
(e) Why should anyone care ? Is the problem interesting and significant ?

(5.4) **Reflections:**

(a) How does this relate to other concepts you have seen in the class ?
(b) What are the next research directions in this line of work ?
(c) What (directly or indirectly related) new ideas did this paper give you ? What would you be curious to try ?

This question is subjective and so we will accept a variety of answers. You are expected to analyze the paper and offer your own perspective and ideas, beyond what the paper itself discusses.

Answer 5. The paper "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention" presents a sophisticated model that merges the intricacies of neural networks with the ingenuity of visual attention mechanisms to generate captions for images. This model is a pivotal stride in the realm of computational vision and language processing, signaling a move towards more nuanced and context-aware systems. At its core, the model introduces an innovative approach by employing a dual attention mechanism, classified as "hard" and "soft." This distinction is not merely academic ; it reflects a strategic choice to mimic the nuances of human attention. The "hard"

attention component is discrete and selective, akin to the way a spotlight focuses on a specific area, while the "soft" attention is continuous, resembling a gradient that softly transitions across different regions. This bifurcation allows the model to adaptively concentrate on various parts of an image, thereby generating a more accurate and descriptive caption that aligns with the salient features of the visual input. The strength of this model is exemplified by its performance on various benchmark datasets such as Flickr8k, Flickr30k, and MS COCO, where it sets new records, surpassing existing methods. The authors substantiate their claims with rigorous quantitative evaluations, showcasing improvements not just in standard metrics like "BLEU" and "METEOR" but also through human-centric qualitative assessments. Furthermore, the model's attention mechanism provides an auxiliary benefit—it acts as a window into the model's reasoning process, offering visualizations that map the focus of the network during each step of the captioning process. This transparency is invaluable, as it aids in troubleshooting and refining the model, and also serves as an educational tool to better understand how attention-driven models process visual information. The "Show, Attend and Tell" paper is particularly valuable in the contemporary AI landscape because it focuses on multimodal neural networks, which are crucial for integrating visual and linguistic data, a cornerstone in the development of intelligent systems. As multimodal approaches become increasingly central to AI research, this paper's contributions to the field are ever more relevant, cementing its place as a hub for cutting-edge multimodal learning strategies. However, while the model excels in a controlled experimental setting, the paper does not thoroughly address its applicability to more diverse and complex real-world scenarios. Nowadays, the CLIP model,Contrastive Language–Image Pre-training, is designed to understand images in the context of natural language. It's trained on a variety of images and text pairs and learns to predict which images and texts are paired together. The CLIP model uses a contrastive learning approach, where it learns to match the corresponding text and image while distinguishing from non-matching ones. Unlike the focused attention mechanism in "Show, Attend and Tell," CLIP aims to understand broader contexts and associations between images and text. The absence of discussion on the model's performance with non-standard, ambiguous, or low-quality images leaves a gap in our understanding of its robustness and versatility. Also, this paper uses a convolutional neural network (VGG) as its encoder to extract features from images but it might be better to evaluate other SOTA architecture like, ResNet and Transformer. The distinction between soft and hard attention raises questions about the computational efficiency and the differentiability of the model during training. Hard attention, which is non-differentiable and requires more complex training techniques such as reinforcement learning, may introduce additional challenges and instabilities. On the other hand, soft attention, although differentiable and easier to train, may not always focus as precisely as hard attention, potentially leading to less sharpness in the model's visual focus and the resulting captions. The ideas in the paper are connected to the larger field of machine learning, especially the use of attention mechanisms in translating languages. The paper suggests new possibilities for research, like using the model to describe videos like "CLIP for CLIP" paper or to help computers discuss images with people. Overall, "Show, Attend and Tell" is a significant step forward in the field of image captioning. It presents a model that not only works well but also gives us insights into how machines can understand and describe visual information.

Note: I used Grammerly and ChatGPT to improve the quality of my writing.