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

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. Using the following Python code:

```python
import torch

tensor = torch.tensor([
    [[2, 4, 3], [2, 1, 2]],
    [[1, 2, 3], [4, 1, 1]],
    [[2, 1, 3], [3, 2, 4]],
    [[1, 4, 2], [2, 4, 3]]
], dtype=torch.float32)
print (tensor.shape)

mean_batch = tensor.mean(dim=[0,2], keepdim=True)
variance_batch = tensor.var(dim=[0,2], keepdim=True, unbiased=False)

mean_layer = tensor.mean(dim=[1, 2], keepdim=True)
variance_layer = tensor.var(dim=[1, 2], keepdim=True, unbiased=False)

mean_instance = tensor.mean(dim=2, keepdim=True)
variance_instance = tensor.var(dim=2, keepdim=True, unbiased=False)
```

$$\boldsymbol{\mu}_{batch} = \begin{bmatrix}2.33\\2.42\end{bmatrix} \qquad \boldsymbol{\sigma}^2_{batch} = \begin{bmatrix}1.06\\1.24\end{bmatrix}$$

$$\boldsymbol{\mu}_{layer} = \left[\left[2.33\right],\left[2\right],\left[2.5\right],\left[2.67\right]\right] \qquad \boldsymbol{\sigma}^2_{layer} = \left[\left[0.89\right],\left[1.33\right],\left[0.92\right],\left[1.22\right]\right]$$

$$\boldsymbol{\mu}_{instance} = \left[\begin{bmatrix}3\\1.67\end{bmatrix},\begin{bmatrix}2\\2\end{bmatrix},\begin{bmatrix}2\\3\end{bmatrix},\begin{bmatrix}2.33\\3\end{bmatrix}\right] \qquad \boldsymbol{\sigma}^2_{instance} = \left[\begin{bmatrix}0.67\\0.22\end{bmatrix},\begin{bmatrix}0.67\\2\end{bmatrix},\begin{bmatrix}0.67\\0.67\end{bmatrix},\begin{bmatrix}1.56\\0.67\end{bmatrix}\right]$$

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. 2.1

The gradient of L_{reg} with respect to θ_i

$$\begin{aligned}\frac{\partial L_{\text{reg}}}{\partial \theta_i} &= \frac{\partial}{\partial \theta_i}\left(L\left(f(x^{(i)},\theta),y^{(i)}\right) + \gamma\|\theta\|_2^2\right) \\ &= \frac{\partial}{\partial \theta_i}\left(L\left(f(x^{(i)},\theta),y^{(i)}\right) + 2\gamma\theta_i\right)\end{aligned}$$

Standard SGD update:

$$\theta_{i+1} = \theta_i - \eta\frac{\partial L_{\text{reg}}}{\partial \theta_i}$$

$$\begin{aligned}\Rightarrow \theta_{i+1} &= \theta_i - \eta\left(\frac{\partial}{\partial \theta_i}L\left(f(x^{(i)},\theta),y^{(i)}\right) + 2\gamma\theta_i\right) \\ &= \theta_i - \eta\frac{\partial}{\partial \theta_i}L\left(f(x^{(i)},\theta),y^{(i)}\right) - 2\eta\gamma\theta_i\end{aligned}$$

If we consider the factor λ to be $2\eta\gamma$, therefore it's equivalent.

2.2

(a) **Adam-L2-reg:**

$$\boldsymbol{\Delta}\theta = -\left(\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta}\right)$$

L2 regularization is applied directly in the loss function. The update rule for Adam remains the same, but the gradients are computed considering the regularization term, and therefore they are adjusted to include a term proportional to the weights themselves.
Due to the smaller gradients of θ_{small} wights, we will see a relatively more significant impact from the regularization term $2\gamma\theta_i$ leading to a stronger regularization effect on them, therefore a more substantial drive to zero.
The larger weights θ_{large} with bigger gradients, will be updated by the gradient of the loss function leading to smaller impact.

(b) **Adam-weight-decay:**

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

The weight decay approach applies a more uniform reduction across all weights, making it less sensitive to the gradient magnitudes.
For smaller weights θ_{large}, since their updates from the gradient-based component of the rule is relatively small because of their smaller gradient, the weight decay term $\lambda\theta$ becomes a more dominant factor in their update. As a result, smaller weights are likely to be driven towards zero more effectively.
For the larger weights θ_{large}, since their updates are primarily driven by their large gradients, they will be subject to the same rate of decay due from the term. However, they are less likely to be driven towards zero as aggressively as smaller weights because their updates are primarily influenced by the gradient-based component of the Adam update rule. Meaning means they will still maintain higher values despite the decay.

2.3

Weight decay offers a uniform approach to regularization, applying the same rate of decay to all weights, regardless of their gradient magnitudes. This ensures proportional regularization relative to each weight's current value, aligning with the goal of preventing overfitting by penalizing large weights. In contrast, L2 regularization impact can vary in adaptive optimizers like Adam, which adjust learning rates based on gradient magnitudes. This can result in less effective regularization for weights with smaller gradients, leading to an uneven control over weight growth.

Weight decay preserves the relative importance of weights as determined during training. While larger weights are decayed, they are not penalized disproportionately compared to less significant weights. However with L2 regularization in adaptive optimizers, larger weights can be penalized more heavily due to their size and larger learning rates, whhich could lead to exagerated penalization of weights important for the model's predictive capability.

Since regularization aims to prevent overfitting by penalizing large weights, weight decay aligns better with this objective by applying a consistent dampening effect across all weights, ensuring that the model does not rely too heavily on any particular weight or set of weights. With adaptive optimizers, where learning rate are not constant across parameters, weight decay ensures a more uniform and predictable regularization effect while maintaining model capacity.

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

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

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 1). 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.

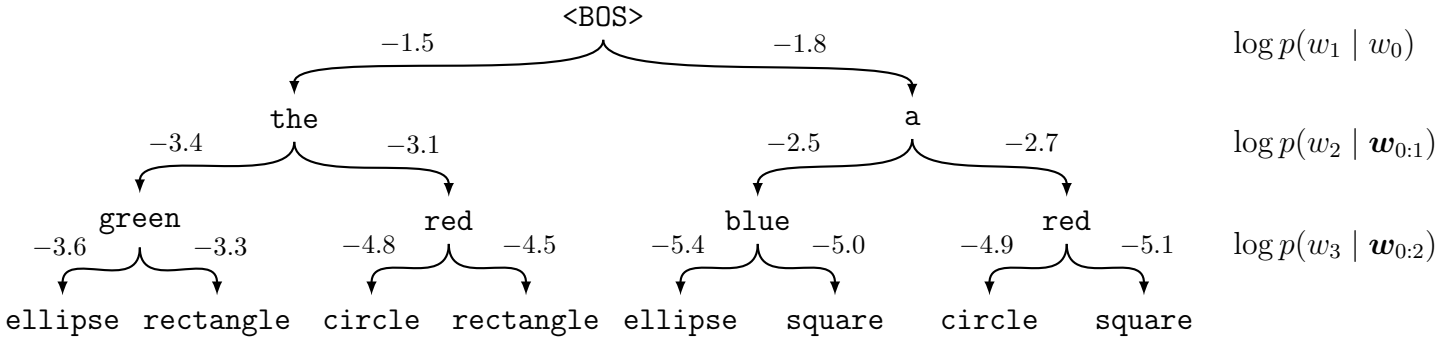
In order to generate B sequences having high likelihood, one can use a heuristic algorithm called *Beam search decoding*, whose pseudo-code is given in Algorithm 1 below

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. 1. Since $T+1$ includes $< BOS >$ we only use T. The vocabulary contains N possible words so: N^T possibilities.

2. Since the possibilities have an exponential growth depending on the vocabulary size, it would be very impractical and long to determine. We would have complexity $O(N^T)$ which is horrible in terms of computational time and ressources. Scaling to a bigger vocabulary would just make it worse because as we see, we have an exponential growth.

3. **Time:** The time complexity will depend on how scores are stored when we do the finding step. We assume it's naive then it would be $O(BN)$, and because it is in a loop over B, the time complexity for the 2nd nested loop inside the T loop is $O(B^2N)$. As a result the overall time complexity of the program would be $O(T(BN + B^2N))$.
   Space: The most significant storage complexity is storing the B sequences of length T. Therefore we have $O(BT)$.
   It significantly reduces the time all while giving a pretty decent quality of sequences. So compared to our previous approach with exponential time complexity, yes it's more practical. It will get slower when the vocabulary size gets larger but not as much as our exponential alternative.
   The size is not as big of a problem :)

4. (a) $< BOS > \rightarrow the(-1.5) \rightarrow red(-3.1) \rightarrow rectangle(-4.5)$
   $\overline{\boldsymbol{w}}_{1:3} =$ "*the red rectangle*"
   $\log p(\overline{\boldsymbol{w}}_{1:3}) = -1.5 - 3.1 - 4.5 = -9.1$

   (b) **Depth 1 ($\boldsymbol{w}_1$):**
   we have [("the", **-1.5**), ("a", **-1.5**)]
   (1) "the"
   (2) "a"
   Depth 2 ($\boldsymbol{w}_2$):
   [("*the green*", -4.9), ("*the red*", -4.6), ("*a blue*", **-4.3**), ("*a red*", **-4.5**)]
   (1) "a blue"
   (2) "a red"
   Depth 3 ($\boldsymbol{w}_3$):
   [("*a blue ellipse*", -9.7), ("*a blue square*", **-9.3**), ("*a red circle*", **-9.4**), ("*a red square*", -9.6)]
   (1) "a blue square"
   (2) "a red circle"

5. It seems like the greedy algorithm is short sighted and will mainly find the local optimum. It makes decisions based solely on its immediate probabilities at hand, without considering the overall context or sequence, leading outcomes that aren't so good.
   Beam search is limited by the size of the beam, so it's a hyperparameter to balance, but it has a greater chance to find the global optimum although it is not guaranteed. As a result, the larger the beam we have, the higher probability it has of approaching the global optimum, but at the cost of increased computational resources.

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 ?

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.

Answer 4. 1.

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

Hypothesis:

$$g_{t-1} = Wh_{t-2} + Ux_{t-1} + b$$

Substitute hypothesis in g_t:

$$\begin{aligned} g_t &= W\sigma(Wh_{t-2} + Ux_{t-1} + b) + Ux_t + b \\ &\Rightarrow \sigma(Wh_{t-2} + Ux_{t-1} + b) = h_{t-1} \\ g_t &= Wh_{t-1} + Ux_t + b \end{aligned}$$

Therefore:

$$h_t = \sigma(g_t)$$

We have shown that for time step t, g_t can be expressed in terms of h_t using the recurrence relation, and assuming that the hypothesis is true for time step $t-1$, it follows by mathematical induction that the relation $h_t = \sigma(g_t)$ holds for all steps.

2. We have:

$$\begin{aligned} h_t &= \sigma(g_t), \\ g_t &= W\sigma(g_{t-1}) + U_{x_t} + b \\ &\text{and} \\ h_{t-1} &= \sigma(g_{t-1}) \end{aligned}$$

Over time:

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

We can extract for one step:

$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial h_t}{\partial g_t} \frac{\partial g_t}{\partial h_{t-1}} \qquad \text{(chain rule)}$$

$$\begin{aligned} \frac{\partial h_t}{\partial g_t} &= \frac{\partial}{\partial g_t}\sigma(g_t) = \sigma'(g_t) \\ \frac{\partial g_t}{\partial h_{t-1}} &= \frac{\partial}{\partial \sigma(g_{t-1})} W\sigma(g_{t-1}) + U_{x_t} + b = W \end{aligned}$$

$$\frac{\partial h_t}{\partial h_{t-1}} = W\sigma'(g_t)$$

Given:

$$||AB|| \leq ||A|| \cdot ||B||$$

We have:

$$\left|\left|\frac{\partial h_t}{\partial h_{t-1}}\right|\right| \leq ||W|| \cdot ||\sigma'(g_t)||$$

Given:

-

$$|\sigma'(g_t)| \leq \gamma$$

-

$$||W|| = \sqrt{\lambda_1(W^\top W)} \quad \text{and} \quad \lambda_1(W^\top W) \leq \frac{\delta^2}{\gamma^2}$$

So:

$$||W|| \leq \frac{\delta}{\gamma}$$

We end up for one step t:

$$\left|\left|\frac{\partial h_t}{\partial h_{t-1}}\right|\right| \leq \frac{\delta}{\gamma} \cdot \gamma = \delta$$

Thus over T steps:

$$\left|\left|\frac{\partial h_T}{\partial h_0}\right|\right| \leq \delta^T$$

Since $\delta < 1$ and $T \to \infty$, we have:

$$\delta^T \to 0$$

Conforming it converge towards 0 and vanish.

3. If $\lambda_1(\boldsymbol{W}^\top \boldsymbol{W}) \geq \frac{\delta^2}{\gamma^2}$, it could lead to the gradients of the hidden state exploding, especially if this value is significantly larger than the threshold since the weight matrix will have a strong influence on the magnitude of the gradients during backpropagation through time. This condition is a sufficient but not necessary condition for gradient explosion, they may not explode if the sequence length T is not very large as we don't have many steps who can amplify the gradients due to the weight matrix's contribution.

4. We can use gradient clipping which takes into account the direction of the gradient. It involves setting a threshold value, and if the L2 norm of the gradient exceeds this threshold, the gradient is scaled down uniformly across all dimensions to keep its direction but limit its magnitude.

   We can also use weight regularization which does not take into account the direction of the gradient. Like L1 or L2 regularization. it adds a penalty term to the loss function that is proportional to the size of the weights. It encourages the network to learn smaller weights, which can indirectly help in preventing the gradients from exploding.

   Gradient clipping would be preferred for addressing exploding gradients since it directly controls the magnitude of the gradient during backprpagation, ensuring that the updates remain within a certain range. Avoids the risk of making disproportionately large updates to the weights. Regularization seems more of a prevention that does not handle the case where a gradient has already exploded. It does not intervene in the training process as dynamically as gradient clipping does.

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. Summary:
The paper introduces an attention-based model for automatic image caption generation, using CNNs for feature extraction and LSTM networks for captioning, guided by an attention mechanism. It employs two types of attention: 'soft' (deterministic, focusing on different parts of the image for each word) and 'hard' (stochastic, randomly selecting image regions). The model performs decently in datasets like Flickr8k, Flickr30k, and MS COCO, particularly in BLEU and METEOR metrics, with hard attention often slightly outperforming soft attention.

Strength
The paper's novel attention mechanism in image captioning allows the model to focus dynamically on different image regions, imitating human behaviour. By focusing on specific image parts, the model generates more detailed and contextually relevant captions, a notable improvement in image captioning capabilities. While the paper primarily addresses challenges in image captioning through attention mechanisms, it also lays the groundwork for new problems in the field, particularly regarding the implementation, optimization, and extension of attention-based models.
The paper does not provide any code, but it does explain the new algorithm/method for image captioning. The experiments are credible, using well-recognized datasets and established metrics, and the comparative analysis highlights the model's contributions, influencing applications beyond image captioning.

Weakness
The model could be tested on more diverse datasets for varied scenes. There are issues with abstract concepts or ambiguous contexts, as seen in the giraffe example misinterpreted as a large white bird. Improvements could be made in computational efficiency, especially concerning the hard attention mechanism which requires sampling methods for training. The use of VGG for feature extraction, while standard at the time, is overshadowed by more effective CNNs like ResNet or EfficientNet, and the paper does not fully justify this choice.
Furthermore, the paper lacks a clear explanation of when and why to prefer one attention mechanism over another. Details on hyperparameter tuning and neural network model configurations, crucial for reproducibility, are not sufficiently detailed. The paper might also benefit from more discussion on the model's practical applications and potential, showcasing its significance and why it merits attention.

Reflection
This work relates to concepts discussed in class, such as converting speech to text using RNNs, and combines various subjects like feature extraction, LSTM, and biases in image recognition. Surprisingly, the paper doesn't delve into image bias intricacies. For future research, extending the model to video captioning, exploring different forms of attention, and optimizing the model's computational efficiency are promising directions.
Reading this paper gave me the idea of combining unclear audio and its associated visual elements to enhance understanding through visual context for caption generating or audio cleaning. We could also apply this model to assist the visually impaired by describing images (converting text to speech of course). Indireclty, a thought came to my mind while reading the paper: the concept of encoding natural language into numerical representations, where each word is mapped to an integer and its variations (like conjugations for example) are represented as floats within a specific range. This is a potentially more resource-efficient method than traditional token storage, offering a more compact and potentially faster way to process language data. However, it would be complicated with for example different language complexities, semantic relationships or even being a scalable system for

range assignment. So it probably wouldn't be effective like classical vectorized word representations.