Due Date: March 22nd 2021, 11pm EST

Instructions

- *For all questions, show your work!*
- *Starred questions are **hard** questions, not **bonus** questions.*
- *Please use a document preparation system such as LaTeX, unless noted otherwise.*
- *Unless noted that questions are related, assume that notation and definitions for each question are self-contained and independent*
- *Submit your answers electronically via Gradescope.*
- *TAs for this assignment are **Krishna Murthy and Tristan Deleu**.*

Question 1 (4-6-4)**.** 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:

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

1.1 Show that applying the activation function in this way is equivalent to the conventional way of applying the activation function: $\boldsymbol{g}_t = \sigma(\boldsymbol{W}\boldsymbol{g}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{b})$ (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 $t-1$.

Answer. Let us suppose that the relation $\boldsymbol{g}_t = \sigma(\boldsymbol{h}_t)$ holds for $t > 0$, which we want to prove by induction. We inspect the base case of $t = 1$:

$$\begin{aligned}\boldsymbol{g}_1 &= \sigma(\boldsymbol{h}_1)\\ &= \sigma\left(\boldsymbol{W}\sigma(\boldsymbol{h}_0) + \boldsymbol{U}\boldsymbol{x}_1 + \boldsymbol{b}\right)\\ &= \sigma\left(\boldsymbol{W}\boldsymbol{g}_0 + \boldsymbol{W}\boldsymbol{x}_1 + \boldsymbol{b}\right)\end{aligned}$$

which satisfies the relation prescribed above. Assuming the relation holds for any $t-1 \geq 0$, we aim to demonstrate that it holds for $(t-1)+1 = t$ as well:

$$\begin{aligned}\boldsymbol{g}_t &= \sigma(\boldsymbol{h}_t)\\ &= \sigma\left(\boldsymbol{W}\sigma(\boldsymbol{h}_{t-1}) + \boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{b}\right)\\ &= \sigma\left(\boldsymbol{W}\boldsymbol{g}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{b}\right).\end{aligned}$$

Once again, we obtain the correct form for $\boldsymbol{g}_t$, and have thus proven by induction that $\boldsymbol{g}_t = \sigma(\boldsymbol{h}_t)$, meaning that the two ways of applying the activation function are equivalent.

*1.2 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 σ, seen as a function $\mathbb{R}^n \to \mathbb{R}^n$, has bounded differential, i.e. $\|D\sigma(\boldsymbol{x})\| \leq \gamma$ (here $\|\cdot\|$ is the L_2 operator norm) for some $\gamma > 0$ and for all $\boldsymbol{x}$. We denote as $\lambda_1(\cdot)$ the largest eigenvalue of a

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.

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, 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

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

Answer. With the upper bound on the largest eigenvalue of the weights and a property of the L_2 norm, we can find a useful relationship for the weight matrix:

$$\begin{aligned}\lambda_1(\boldsymbol{W}^\top \boldsymbol{W}) &\leq \frac{\delta^2}{\gamma^2} \\ \|\boldsymbol{W}\|^2 &\leq \frac{\delta^2}{\gamma^2}\end{aligned}$$

We recall the expression for the preactivation at time t:

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

and we expand the gradients of the hidden state using the chain rule:

$$\frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{h}_0} = \frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{g}_{T-1}}\frac{\partial \boldsymbol{g}_{T-1}}{\partial \boldsymbol{h}_{T-1}}\frac{\partial \boldsymbol{h}_{T-1}}{\partial \boldsymbol{g}_{T-2}} \cdots \frac{\partial \boldsymbol{h}_2}{\partial \boldsymbol{g}_1}\frac{\partial \boldsymbol{g}_1}{\partial \boldsymbol{h}_1}\frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{g}_0}\frac{\partial \boldsymbol{g}_0}{\partial \boldsymbol{h}_0}$$

$$\begin{aligned}\Rightarrow \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{g}_{t-1}} &= \frac{\partial}{\partial \boldsymbol{g}_{t-1}}\left[\boldsymbol{W}\boldsymbol{g}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{b}\right] = \boldsymbol{W} \\ \Rightarrow \frac{\partial \boldsymbol{g}_{t-1}}{\partial \boldsymbol{h}_{t-1}} &= \frac{\partial}{\partial \boldsymbol{h}_{t-1}}\sigma(\boldsymbol{h}_{t-1}) = \sigma'(\boldsymbol{h}_{t-1})\end{aligned}$$

The L_2 norm of the gradients would then be

$$\begin{aligned}\left\|\frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{h}_0}\right\| &= \|\boldsymbol{W}\sigma'(\boldsymbol{h}_{T-1})\ldots \boldsymbol{W}\sigma'(\boldsymbol{h}_0)\| \\ &\leq \|\boldsymbol{W}\|^T \|\sigma'(\boldsymbol{h}_{T-1})\| \ldots \|\sigma'(\boldsymbol{h}_0)\|\end{aligned}$$

The function $\sigma(\boldsymbol{x})$ has a bounded differential for all $\boldsymbol{x}$: $\|\sigma'(\boldsymbol{x})\| \leq \gamma$.

$$\begin{aligned}\left\|\frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{h}_0}\right\| &\leq \|\boldsymbol{W}\|^T \|\sigma'\|^T \\ &\leq \left(\frac{\delta}{\gamma}\right)^T \gamma^T \\ &\leq \delta^T\end{aligned}$$

Since $0 \leq \delta < 1$, the magnitude of these gradients would fall to 0 as $T \to \infty$.

1.3 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 ? (Answer in 1-2 sentences).

Answer. In this situation, the upper bound condition on the derivative of σ is unchanged. This reversed condition on the largest eigenvalue is then necessary, but not sufficient for the gradients to explode, as δ is still $\in [0, 1)$ and the largest eigenvalue of the weights is not necessarily larger than 1, which would lead to ever-increasing gradient magnitudes.

Question 2 (1-12-2-6)**.** 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

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

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

2.1 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.

Answer. We have a vocabulary of N words, out of which we form sentences of length $T+1$. We suppose that words can be repeated in a same sentence, and since every sentence starts with `<BOS>`, we consider sentences of length T. This token is included in N, but it is also supposed that it can reappear anywhere in a possible sentence. We are then looking for the number of possible ordered combinations with repetition, which would be N^T possible sentences.

2.2 In this question only, we will assume that our language model satisfies the *Markov property*

$$\forall\, t > 0,\ p(w_t \mid w_0, \ldots, w_{t-1}) = p(w_t \mid w_{t-1}).$$

Moreover, we will assume that the model is time-homogeneous, meaning that there exists a matrix $\boldsymbol{P} \in \mathbb{R}^{N \times N}$ such that $\forall\, t > 0,\ p(w_t = j \mid w_{t-1} = i) = [\boldsymbol{P}]_{i,j} = P_{ij}$.

2.2.a Let $\boldsymbol{\delta}_t \in \mathbb{R}^N$ be the vector of size N defined for $t > 0$ as

$$\delta_t(j) = \max_{\boldsymbol{w}_{1:t-1}} p(\boldsymbol{w}_{1:t-1}, w_t = j),$$

where $\boldsymbol{w}_{1:0} = \emptyset$. Using the following convention for $\boldsymbol{\delta}_0$

$$\delta_0(j) = \begin{cases} 1 & \text{if } j = \texttt{<BOS>} \\ 0 & \text{otherwise,} \end{cases}$$

give the recurrence relation satisfied by $\boldsymbol{\delta}_t$ (for $t > 0$).

Answer. We first make use of the conditional probability relation:

$$p(A, B) = p(A)p(B|A) \tag{1}$$

in order to write

$$\begin{aligned} p(\boldsymbol{w}_{1:t-1}, w_t = j) &= p(\boldsymbol{w}_{1:t-1})p(w_t = j|\boldsymbol{w}_{1:t-1}) \\ &= p(\boldsymbol{w}_{1:t-1})p(w_t = j|w_{t-1}) \end{aligned}$$

where we employed the Markov property. We expand the first factor:

$$\begin{aligned} p(\boldsymbol{w}_{1:t-1}) &= p(w_1, \ldots, w_{t-2}, w_{t-1}) \\ &= p(\boldsymbol{w}_{1:t-2}, w_{t-1}) \\ &= p(\boldsymbol{w}_{1:t-2})p(w_{t-1}|w_{t-2}). \end{aligned}$$

Returning to $\delta_t(j)$,

$$\begin{aligned} \delta_t(j) &= \max_{\boldsymbol{w}_{1:t-1}} p(\boldsymbol{w}_{1:t-1}, w_t = j) \\ &= \max_{\boldsymbol{w}_{1:t-1}} p(\boldsymbol{w}_{1:t-1})p(w_t = j|w_{t-1}) \\ &= \max_{\boldsymbol{w}_{1:t-1}} p(\boldsymbol{w}_{1:t-2})p(w_{t-1}|w_{t-2})p(w_t = j|w_{t-1}) \end{aligned}$$

We exploit a characteristic of the max operation:

$$\begin{aligned} \max_{\boldsymbol{w}_{1:t-1}}[\ldots] &= \max_{w_1,\ldots,w_{t-2},w_{t-1}}[\ldots] \\ &= \max_{\boldsymbol{w}_{1:t-2},w_{t-1}}[\ldots] \\ &= \max_{w_{t-1}}\left(\max_{\boldsymbol{w}_{1:t-2}}[\ldots]\right) \end{aligned}$$

which implies:

$$\begin{aligned} \delta_t(j) &= \max_{w_{t-1}}\left(\underbrace{\max_{\boldsymbol{w}_{1:t-2}} p(\boldsymbol{w}_{1:t-2})p(w_{t-1}|w_{t-2})}_{\delta_{t-1}(w_{t-1})} p(w_t = j|w_{t-1})\right) \\ &= \max_{w_{t-1}} p(w_t = j|w_{t-1})\delta_{t-1}(w_{t-1}) \\ &= \max_i p(w_t = j|w_{t-1} = i)\delta_{t-1}(i) \\ &= \max_i P_{ij}\delta_{t-1}(i) \end{aligned}$$

which is a recurrence relation satisfied by $\boldsymbol{\delta}_t$ for $t > 0$.

2.2.b Show that $w_T^\star = \arg\max_j \delta_T(j)$.

Answer. We are looking for the last word in the most likely sequence. The most likely sequence up to $T-1$ is the one that maximizes

$$p(\boldsymbol{w}^*_{1:T-1}) = \max_{\boldsymbol{w}_{1:T-1}} p(\boldsymbol{w}_{1:T-1}).$$

The most likely last word w_T^* in this sequence can then be expressed as a conditional:

$$\begin{aligned} w_T^* &= \arg\max_j p(w_T = j|\boldsymbol{w}^*_{1:T-1})p(\boldsymbol{w}^*_{1:T-1}) \\ &= \arg\max_j \max_{\boldsymbol{w}_{1:T-1}} p(w_T = j|\boldsymbol{w}_{1:T-1})p(\boldsymbol{w}_{1:T-1}) \\ &= \arg\max_j \delta_T(j) \end{aligned}$$

2.2.c Let $\boldsymbol{a}_t \in \{1,\dots,N\}^N$ be the vector of size N defined for $t > 0$ as

$$a_t(j) = \arg\max_i P_{ij}\delta_{t-1}(i).$$

Show that $\forall\, 0 < t < T$, $w_t^\star = a_{t+1}(w_{t+1}^\star)$.

Answer. We begin from the right-hand side:

$$\begin{aligned} w_t^* &= a_{t+1}(w^*_{t+1}) \\ &= \arg\max_i P_{i,j=w^*_{t+1}}\delta_t(i) \\ &= \arg\max_i p(w^*_{t+1}|w_t = i)\delta_t(i) \\ &= \arg\max_i \max_{\boldsymbol{w}_{1:t-1}} p(w^*_{t+1}|w_t = i)p(\boldsymbol{w}_{1:t-1})p(w_t = i|\boldsymbol{w}_{1:t-1}) \end{aligned}$$

We can take apart this expression: the $\max_{\boldsymbol{w}_{1:t-1}}$ indicates the most likely sequence that would be followed by the word $w_t = 1$, and this same word also precedes w^*_{t+1}, which is at position $t+1$ in the most likely sequence. The $\arg\max_i$ then extracts the w_t that maximizes this entire expression, we can then conclude that it is part of the most likely sequence as well. Hence, we have $a_{t+1}(w^*_{t+1}) = w^*_t$. For $t = 0$, this relation is redundant since all sequences have $w_0^* =$ `<BOS>` independent of any w_1^*. For $t = T$, this relation no longer holds as sequences are bounded at length T and no w_{T+1} words exist. This expression is then valid for any $0 < t < T$.

2.2.d Using the previous questions, write the pseudo-code of an algorithm to compute the sequence $\boldsymbol{w}^\star_{1:T}$ using dynamic programming. This is called *Viterbi decoding*.

Algorithm 1: Viterbi decoding algorithm

```
Require: a language model $p(\boldsymbol{w}_{1:T}|w_0)$
Require: matrix $\boldsymbol{P}$ of size $N \times N$
 1: Initialization: $w_0 \leftarrow$ <BOS>
 2: Define $\delta$ of size $T \times N$
 3: Define $a$ of size $T \times N$
 4: Define vector $\boldsymbol{w}^*$ for most likely sequence, of size $T$
 5: Define prior $\delta_0(j) = \mathbb{1}_{j=\texttt{<BOS>}}$
 6: for $t = 1$ to $T$ do
 7:     for $j = 1$ to $N$ do
 8:         Compute from recursion: $\delta_t(j) \leftarrow \max_i P_{ij}\delta_{t-1}(i)$
 9:         Compute from recursion: $a_t(j) = \arg\max_i P_{ij}\delta_{t-1}(i)$
10:     end for
11: end for
12: Determine final state: $w_T^* \leftarrow \arg\max_j \delta_T(j)$
13: for $t = T - 1$ to $1$ do
14:     Obtain $w_t^* \leftarrow a_{t+1}(w_{t+1}^*)$
15: end for
16: return most likely sentence $\boldsymbol{w}_{1:T}^*$
```

2.2.e What is the time complexity of this algorithm ? Its space complexity ? Write the algorithmic complexities using the O notation. Comment on the efficiency of this algorithm compared to naively searching for $\boldsymbol{w}_{1:T}^\star$ by enumerating all possible sequences (based on your answer to question 1).

Answer. The time complexity can be worked out by going through the operations performed by the algorithm. For every t in the first loop, we go through every word j of the vocabulary in the nested loop. For every word j, the $\max_i$ and $\arg\max_i$ operations both require going through the entire vocabulary, contributing a complexity factor of $2N$. This pass through both loops is then of time complexity $O(2TN^2)$. Afterwards, we perform another $\arg\max$ on line 11, contributing another $O(N)$ to the total complexity. Finally, the last loop goes through every T timesteps, meaning $O(T)$ additional time complexity. The overall time complexity of this algorithm is then $O(2TN^2 + N + T) = O(TN^2)$. This is an invaluable efficiency gain compared to the time complexity of a naive search for $\boldsymbol{w}_{1:T}^*$ by enumerating all possible sequences, which would be of $O(N^T)$.

The space complexity is determined by the working storage required by an algorithm. For this Viterbi algorithm, it depends on the space allocated for variables: the space complexity for δ and a totals $O(2TN)$, with an additional $O(T)$ to store the most likely sequence $\boldsymbol{w}$ and $O(N^2)$ for the matrix $\boldsymbol{P}$ representing the language model. The overall space complexity of this algorithm is then $O(2TN + T + N^2) = O(TN + N^2)$.

2.2.f When the size of the vocabulary N is not too large, can you use this algorithm to generate the most likely sequence of a language model defined by a recurrent neural network, such as a GRU or an LSTM ? Why ? Why not ? If not, name a language model you can apply this algorithm to.

Answer. RNNs such as GRU or LSTM define language models with long term dependencies and that heavily rely on the context of each state, causing the time-homogeneous and low n-th order Markov assumptions that are the basis of Viterbi decoding to no longer hold for such models. However, if N is not too large, it would be possible to construct a higher dimensional matrix $\boldsymbol{P}$ with which to encode such dependencies on context or location in the sequence, allowing to use Viterbi decoding to generate a most likely sequence from such language models.

2.3 For real-world applications, the size of the vocabulary N can be very large (e.g. $N = 30$k for BERT, $N = 50$k for GPT-2), making even dynamic programming impractical. 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 2 below

```
Algorithm 2: 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 $b \in \{1, \dots, B\}$
```

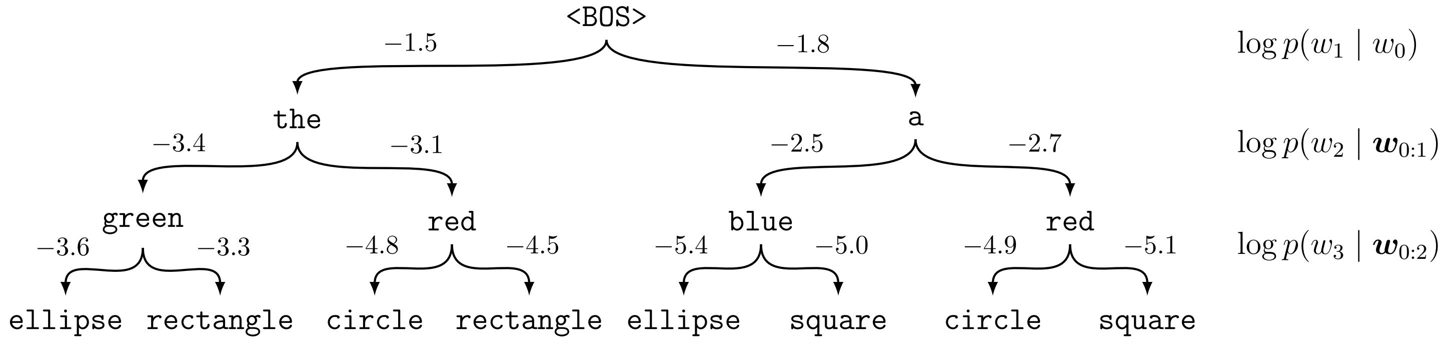
What is the time complexity of Algorithm 2 ? Its space complexity ? Write the algorithmic complexities using the O notation, as a function of T, B, and N. Is this a practical decoding algorithm when the size of the vocabulary is large ?

Answer. We go through the operations in the algorithm to determine its time complexity. For the first set of three loops, the time complexity is $O(TBN)$. For the second nested loop, for every t we go through every b of the beam width B to lookup in the s array for the word j with the b-th largest score. This array is of dimension $B \times N$, so the worst-case sorting algorithm would be of time complexity $O(B^2N^2)$, for a loop complexity of $O(TB^3N^2)$. The overall time complexity of Beam search decoding is then $O(TBN + TB^3N^2) = O(TB^3N^2)$.

The contributions to space complexity for this algorithm are as follows: $O(NT)$ for the language model, $O(2B)$ for $w_0^{(b)}$ and $l_0^{(b)}$, $O(BN)$ for s, $O(BT)$ for $\widetilde{\boldsymbol{w}}_{0:T-1}^{(b)}$, $O(B)$ for $l_t^{(b)}$ and $O(B(T-1))$ for $\boldsymbol{w}_{1:T}^{(b)}$. The overall space complexity is then $O(NT + 2B + BN + BT + B + B(T-1)) = O(NT + BN + BT)$.

For large N, this algorithm should always have a lower space complexity than Viterbi, since it does not involve higher powers of N. However, the comparison of their time complexities depends on the choice of sorting algorithm in Beam search. For the worst-case described above, we compare Viterbi's $O(TN^2)$ to Beam search's $O(TB^3N^2)$, and Beam search is less practical than Viterbi. However, using another sorting algorithm of time complexity $O(BN \log BN)$ for example, the complexity of Beam search would become $O(TB^2N \log BN)$, which can be more practical than Viterbi depending on the beam width B.

2.4 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)



2.4.a If you were given the whole tree, including the log-probabilities of all the missing branches (e.g. $\log p(w_2 = \texttt{a} \mid w_0 = \texttt{<BOS>}, w_1 = \texttt{red})$), could you apply Viterbi decoding from question 2 to this language model in order to find the most likely sequence $\boldsymbol{w}^\star_{1:3}$? Why ? Why not ? Find $\boldsymbol{w}^\star_{1:3}$, together with its corresponding log-likelihood $\log p(\boldsymbol{w}^\star_{1:3}) = \max_{\boldsymbol{w}_{1:3}} \log p(\boldsymbol{w}_{1:3})$.

Answer. Given the complete tree as a language model, it would be possible to obtain a matrix $\boldsymbol{P}$ to compute the recursion relations of the Viterbi algorithm. Viterbi could then be applied to find the most likely sentence $\boldsymbol{w}^*_{1:3}$. This sentence can be found through the expression

$$\boldsymbol{w}^*_{1:3} = \arg\max_{\boldsymbol{w}_{1:3}} \sum_{t=1}^{3} \log p(w_t | w_0, \dots, w_{t-1})$$

The different paths through the tree are laid out in table 1 along with their associated log-likelihoods. As we can see, the most likely sentence is $\boldsymbol{w}^*_{1:3} =$ [`'the'`, `'green'`, `'rectangle'`] with log-likelihood $\log p(\boldsymbol{w}^*_{1:3}) = -8.2$.

2.4.b *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 $\log p(\overline{\boldsymbol{w}}_{1:3})$.

Token 0	Token 1	Token 2	Token 3	Log-likelihood
`<BOS>`	`a`	`red`	`square`	-9.6
			`circle`	-9.4
		`blue`	`square`	-9.3
			`ellipse`	-9.7
	`the`	`red`	`rectangle`	-9.1
			`circle`	-9.4
		`green`	`rectangle`	-8.2
			`ellipse`	-8.5

Table 1 – Possible paths in the reduced tree and their associated log-likelihoods. Most likely sentence with maximal log-likelihood highlighted in red.

Answer. We find every token sequentially:

$$\begin{aligned}
\overline{w}_1 &= \arg\max_{w_1} \log p(w_1|\overline{w_0}) \\
&= \arg\max \{-1.5, -1.8\} \\
&= \texttt{'the'} \\
\overline{w}_2 &= \arg\max_{w_2} \log p(w_2|\overline{\boldsymbol{w}}_{0:1}) \\
&= \arg\max \{-3.4, -3.1\} \\
&= \texttt{'red'} \\
\overline{w}_3 &= \arg\max_{w_3} \log p(w_3|\overline{\boldsymbol{w}}_{0:2}) \\
&= \arg\max \{-4.8, -4.5\} \\
&= \texttt{'rectangle'}
\end{aligned}$$

The sequence found by greedy decoding is $\overline{\boldsymbol{w}}_{1:3} = [$`'the'`, `'red'`, `'rectangle'`$]$, with log-likelihood $\log p(\overline{\boldsymbol{w}}_{1:3}) = -9.1$

2.4.c Apply beam search decoding (question 3) 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.

Answer. We go through the first two timesteps of the algorithm as an example, initializing $w_0^{(1)} = w_0^{(2)} =$ `<BOS>` and $l_0^{(1)} = l_0^{(2)} = 0$:

$$
\begin{aligned}
t =& 1 : \\
& b = 1, 2 : \\
& \quad s = \begin{bmatrix} s_1(\texttt{'the'}) & s_1(\texttt{'a'}) \\ s_2(\texttt{'the'}) & s_2(\texttt{'a'}) \end{bmatrix} = \begin{bmatrix} -1.5 & -1.8 \\ -1.5 & -1.8 \end{bmatrix} \\
& b = 1 : \\
& \quad s_{1'}(\texttt{'the'}) = -1.5 \\
& \quad \widetilde{w}_0^{(1)} = [\texttt{<BOS>}] \\
& \quad w_1^{(1)} = \texttt{'the'} \\
& \quad l_1^{(1)} = -1.5 \\
& b = 2 : \\
& \quad s_{2'}(\texttt{'a'}) = -1.8 \\
& \quad \widetilde{w}_0^{(2)} = [\texttt{<BOS>}] \\
& \quad w_1^{(2)} = \texttt{'a'} \\
& \quad l_1^{(2)} = -1.8
\end{aligned}
$$

$$
\begin{aligned}
t =& 2 : \\
& b = 1, 2 : \\
& \quad s = \begin{bmatrix} s_1(\texttt{'green'}) & s_1(\texttt{'red'}) \\ s_2(\texttt{'blue'}) & s_2(\texttt{'red'}) \end{bmatrix} = \begin{bmatrix} -4.9 & -4.6 \\ -4.3 & -4.5 \end{bmatrix} \\
& b = 1 : \\
& \quad s_{2'}(\texttt{'blue'}) = -4.3 \\
& \quad \widetilde{w}_{0:1}^{(1)} = w_{0:1}^{(2)} = [\texttt{<BOS>},\ \texttt{'a'}] \\
& \quad w_2^{(1)} = \texttt{'blue'} \\
& \quad l_1^{(1)} = -4.3 \\
& b = 2 : \\
& \quad s_{2'}(\texttt{'red'}) = -4.5 \\
& \quad \widetilde{w}_{0:1}^{(2)} = w_{0:1}^{(2)} = [\texttt{<BOS>},\ \texttt{'a'}] \\
& \quad w_1^{(2)} = \texttt{'red'} \\
& \quad l_1^{(2)} = -4.5
\end{aligned}
$$

Carrying on with this procedure for $t = 3$, we would finally obtain the sequences $\boldsymbol{w}_{1:3}^{(1)} =$ ['a', 'blue', 'square'] and $\boldsymbol{w}_{1:3}^{(2)} =$ ['a', 'red', 'circle'], with associated likelihoods $\log p(\boldsymbol{w}_{1:3}^{(1)}) = -9.3$ and $\log p(\boldsymbol{w}_{1:3}^{(2)}) = -9.4$.

2.4.d Compare the behaviour of these 3 decoding algorithms on this language model (in particular greedy decoding vs. maximum likelihood, and beam search decoding vs. the other two). How can you mitigate the limitations of beam search ?

Answer. Greedy decoding behaves in a rather narrow minded way, as it will go for the token with the largest immediate reward at every step, and cannot change paths once it has begun going down a certain sequence. Maximum likelihood, on the other hand, will consider the total likelihood of every possible sequence in the tree before settling on the path with the largest. Like greedy decoding, it returns a single estimate. With beam search however, the path is not fixed once a direction has been chosen as the B hypotheses can entirely change through time as more information becomes available. It is in some way a compromise between greedy decoding and maximum likelihood, by going down preferential paths but keeping its options open. Unlike the two others, it returns multiple sequence hypotheses. Issues with beam search can arise for a beam width that is too large, as some low-confidence hypotheses can be returned. This can be mitigated by reducing beam width.

Question 3 (2-3-4-4)**. Weight decay as L2 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 $L(f(\mathbf{x}^{(j)}, \theta), \mathbf{y}^{(j)})$, 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

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

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

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

3.1 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.

Answer. A standard SGD parameter update for a network with L2 regularization is

$$\begin{aligned}
\theta_{i+1} &= \theta_i - \eta \left[\frac{1}{m} \frac{\partial}{\partial \theta_i} \sum_{j=1}^{m} L_{reg}(f(\mathbf{x}^{(j)}, \theta_i), \mathbf{y}^{(j)}) \right] \\
&= \theta_i - \frac{\eta}{m} \frac{\partial}{\partial \theta_i} \left[\sum_{j=1}^{m} L(f(\mathbf{x}^{(j)}, \theta_i), \mathbf{y}^{(j)}) + \gamma \|\theta\|_2^2 \right] \\
&= \theta_i - \frac{\eta}{m} \left[\gamma m \frac{\partial}{\partial \theta_i} \left(\theta_1^2 + \theta_2^2 + \ldots + \theta_i^2 + \ldots \right) + \frac{\partial}{\partial \theta_i} \sum_{j=1}^{m} L(f(\mathbf{x}^{(j)}, \theta_i), \mathbf{y}^{(j)}) \right] \\
&= \theta_i - \frac{\eta}{m} \frac{\partial}{\partial \theta_i} \sum_{j=1}^{m} L(f(\mathbf{x}^{(j)}, \theta_i), \mathbf{y}^{(j)}) - 2\eta\gamma\theta_i
\end{aligned}$$

We find that an L2 regularization with SGD update is equivalent to weight decay with SGD if $\lambda = -2\eta\gamma$.

3.2 This question refers to the Adam algorithm as described in the lecture slide (also identical to Algorithm 8.7 of the deep learning book). It turns out that a one-line change to this algorithms gives us Adam with an L2 regularization scheme. Identify the line of the algorithm that needs to change, and provide this one-line modification.

Answer. We transcribe the Adam algorithm from the deep learning book in algorithm 3. To obtain Adam with L2 regularization, we must simply modify the loss to have a regularization term. Line 5 of algorithm 3 should then be $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i \left(L(f(\mathbf{x}^{(i)}, \boldsymbol{\theta}), \mathbf{y}^{(i)}) \right) + 2\gamma\boldsymbol{\theta}$, where the regularizer term has been differentiated

Algorithm 3: The Adam algorithm

```
Require: Step size $\epsilon$ (default: 0.001)
Require: Exponential decay for moment estimates, $\rho_1$ and $\rho_2$ in $[0, 1)$. (default: 0.9 and 0.99)
Require: Small constant $\delta$ used for numerical stabilization. (default: $10^{-8}$)
Require: Initial parameters $\boldsymbol{\theta}$
 1: Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$
 2: Initialize time step $t = 0$
 3: while stopping criterion no met do
 4:     Sample a minibatch of $m$ examples from the training set $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ with
        corresponding targets $\mathbf{y}^{(i)}$.
 5:     Compute gradients: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\mathbf{x}^{(i)}, \boldsymbol{\theta}), \mathbf{y}^{(i)})$
 6:     $t \leftarrow t + 1$
 7:     Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
 8:     Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
 9:     Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$
10:     Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
11:     Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (element-wise)
12:     Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
13: end while
```

3.3 Consider a "decoupled" weight decay scheme for the original Adam algorithm (see lecture slides, or equivalently, Algorithm 8.7 of the deep learning book) 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 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 ?

Answer. Referring ourselves to the Adam algorithm below with its L2 modification, we note that the presence of $\sqrt{\hat{\boldsymbol{r}}}$ in the denominator of line 11 implies a normalization of the sum of the gradients of the loss and the regularizer by their summed magnitudes (from line 8). Weights θ_l with larger gradient magnitudes will then have a larger denominator in line 11 relative to other weights, and then be regularized less strongly relative to their initial magnitudes than weights θ_s with smaller gradient magnitudes.

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

Answer. For Adam-weight-decay, the gradient of the regularizer does not get scaled along with that of the loss. All weights are regularized by a same amount λ, proportional to the weight at a previous time step, which themselves depend on the magnitude of the weight gradients in the past. The answer would then be different, since weights with larger gradients magnitudes would be strongly regularized relative to their counterparts in this case.

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

3.4 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. The intent of regularization is to penalize models with high complexity so that they have a better capacity for generalization. For a given architecture, a complex model would be one with large weights of large gradient magnitudes, characteristics that could facilitate overfitting to the training data. We would then wish to penalize such models, so to reduce their weight and gradient magnitudes. L2 regularization may be equivalent to weight decay in the context of an SGD gradient update scheme, but we have determined that L2 reg. has the adverse effect of regularizing **smaller** weights strongly compared to larger weights for Adam, an adaptive gradient based optimizer. This hinders the generalization capacity, whereas Adam-weight-decay penalizes the large weights as desired. Adaptive gradient based optimizers generally involve normalization by the gradient magnitudes, which is where this problem arises. It can then be said that weight decay is a better scheme to employ than L2 regularization in general.

Question 4 (4-6-6)**.** This question is about normalization techniques.

4.1 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 $\boldsymbol{\sigma}^2_{instance}$.

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

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. In the following, we denote the batch, feature and channel dimension sizes B, C and F respectively.

- **Batch normalization**: for BN, the mean tensor $\boldsymbol{\mu}_{BN}$ collapses a 3D tensor by averaging over the batch and feature dimensions. Here, the result is then a vector of length 2 of components:

$$\begin{aligned}
\mu_{BN,1} &= \frac{1}{BF}\sum_{b=1}^{B}\sum_{f=1}^{F} x_{b1f} \\
&= \frac{1}{12}\left[(1+3+2)+(3+3+2)+(4+2+2)+(3+3+2)\right] \\
&= 2.5 \\
\mu_{BN,2} &= \frac{1}{BF}\sum_{b=1}^{B}\sum_{f=1}^{F} x_{b2f} \\
&= \frac{1}{12}\left[(1+2+3)+(2+4+4)+(1+2+4)+(3+3+2)\right] \\
&= 2.583 \\
\Rightarrow \boldsymbol{\mu}_{BN} &= \begin{bmatrix}2.5 & 2.583\end{bmatrix}
\end{aligned}$$

The variance $\boldsymbol{\sigma}^2_{BN}$ is computed for both channels in a similar fashion:

$$\begin{aligned}
\sigma^2_{BN,1} &= \frac{1}{BF}\sum_{b=1}^{B}\sum_{f=1}^{F}(x_{b1f}-\mu_{BN,1})^2 \\
&= 0.583 \\
\sigma^2_{BN,2} &= \frac{1}{BF}\sum_{b=1}^{B}\sum_{f=1}^{F}(x_{b2f}-\mu_{BN,2})^2 \\
&= 1.076 \\
\Rightarrow \boldsymbol{\sigma}^2_{BN} &= \begin{bmatrix}0.583 & 1.076\end{bmatrix}
\end{aligned}$$

- **Layer normalization** : for LN, the tensor is collapsed over the feature and channel dimensions, resulting in a vector of length 4:

$$\begin{aligned}
\mu_{LN,1} &= \frac{1}{CF}\sum_{c=1}^{C}\sum_{f=1}^{F} x_{1cf} \\
&= 2 \\
\mu_{LN,2} &= \frac{1}{CF}\sum_{c=1}^{C}\sum_{f=1}^{F} x_{2cf} \\
&= 3 \\
\mu_{LN,3} &= \frac{1}{CF}\sum_{c=1}^{C}\sum_{f=1}^{F} x_{3cf} \\
&= 2.5 \\
\mu_{LN,4} &= \frac{1}{CF}\sum_{c=1}^{C}\sum_{f=1}^{F} x_{4cf} \\
&= 2.67 \\
\Rightarrow \boldsymbol{\mu}_{LN} &= \begin{bmatrix} 2 & 3 & 2.5 & 2.67 \end{bmatrix}
\end{aligned}$$

The variance $\boldsymbol{\sigma}^2_{LN}$ is computed for the four batch members in a similar fashion:

$$\begin{aligned}
\sigma^2_{LN,1} &= \frac{1}{CF}\sum_{c=1}^{B}\sum_{f=1}^{F} (x_{1cf} - \mu_{LN,1})^2 \\
&= 0.67 \\
\sigma^2_{LN,2} &= \frac{1}{CF}\sum_{c=1}^{B}\sum_{f=1}^{F} (x_{2cf} - \mu_{LN,2})^2 \\
&= 0.67 \\
\sigma^2_{LN,3} &= \frac{1}{CF}\sum_{c=1}^{B}\sum_{f=1}^{F} (x_{3cf} - \mu_{LN,3})^2 \\
&= 1.25 \\
\sigma^2_{LN,4} &= \frac{1}{CF}\sum_{c=1}^{B}\sum_{f=1}^{F} (x_{4cf} - \mu_{LN,4})^2 \\
&= 0.22 \\
\Rightarrow \boldsymbol{\sigma}^2_{LN} &= \begin{bmatrix} 0.67 & 0.67 & 1.25 & 0.22 \end{bmatrix}
\end{aligned}$$

- **Instance normalization** : for IN, the tensor is collapsed over the feature dimension, resulting in a matrix of shape 4×2:

$$\mu_{IN,bc} = \frac{1}{F}\sum_{f=1}^{F} x_{bcf}$$
$$\Rightarrow \boldsymbol{\mu}_{IN} = \begin{bmatrix} 2 & 2 \\ 2.67 & 3.33 \\ 2.67 & 2.33 \\ 2.67 & 2.67 \end{bmatrix}$$

The variance $\boldsymbol{\sigma}^2_{IN}$ is computed for the four feature slices in a similar fashion:

$$\sigma^2_{IN,bc} = \frac{1}{F}\sum_{f=1}^{F}(x_{bcf} - \mu_{IN,bc})^2$$
$$\Rightarrow \boldsymbol{\sigma}^2_{IN} = \begin{bmatrix} 0.67 & 0.67 \\ 0.22 & 0.89 \\ 0.89 & 1.56 \\ 0.22 & 0.22 \end{bmatrix}$$

4.2 For the next two subquestions, we consider the following parameterization of a weight vector $\boldsymbol{w}$:

$$\boldsymbol{w} := \gamma\frac{\boldsymbol{u}}{||\boldsymbol{u}||}$$

where γ is scalar parameter controlling the magnitude and $\boldsymbol{u}$ is a vector controlling the direction of $\boldsymbol{w}$.

Consider one layer of a neural network, and omit the bias parameter. To carry out batch normalization, one normally standardizes the preactivation and performs elementwise scale and shift $\hat{y} = \gamma \cdot \frac{y-\mu_y}{\sigma_y} + \beta$ where $y = \boldsymbol{u}^\top\boldsymbol{x}$. Assume the data $\boldsymbol{x}$ (a random vector) is whitened ($\mathrm{Var}(\boldsymbol{x}) = \boldsymbol{I}$) and centered at 0 ($\mathbb{E}[\boldsymbol{x}] = \boldsymbol{0}$). Show that $\hat{y} = \boldsymbol{w}^\top\boldsymbol{x} + \beta$.

Answer. We find the expectation of the preactivation $\mu_y = \mathbb{E}[y]$ by exploiting the linearity of the expected value operator:

$$\begin{aligned}\mathbb{E}[y] &= \mathbb{E}\left[\frac{||\boldsymbol{u}||}{\gamma}\boldsymbol{w}^\top\boldsymbol{x}\right] \\ &= \frac{||\boldsymbol{u}||}{\gamma}\boldsymbol{w}^\top\mathbb{E}[\boldsymbol{x}] \\ &= \boldsymbol{0}\end{aligned}$$

These manipulations are possible since neither $\boldsymbol{u}$ nor $\boldsymbol{w}$ are vectors of random variables. We now find the variance $\sigma_y^2 = \mathrm{Var}[y]$ of the preactivation, using the following identity for the variance of the dot product of a vector of scalars and another of random variables : $\mathrm{Var}[\boldsymbol{w}^\top\boldsymbol{x}] = \boldsymbol{w}^\top\Sigma\boldsymbol{w}$, where Σ is the covariance matrix of $\boldsymbol{x}$.

$$\begin{aligned}\mathrm{Var}[y] &= \mathrm{Var}\left[\frac{\|\boldsymbol{u}\|}{\gamma}\boldsymbol{w}^\top\boldsymbol{x}\right]\\ &= \frac{\|\boldsymbol{u}\|^2}{\gamma^2}\mathrm{Var}[\boldsymbol{w}^\top\boldsymbol{x}]\\ &= \frac{\|\boldsymbol{u}\|^2}{\gamma^2}\boldsymbol{w}^\top\Sigma\boldsymbol{w}\\ &= \frac{\|\boldsymbol{u}\|^2}{\gamma^2}\boldsymbol{w}^\top\boldsymbol{I}\boldsymbol{w}\\ &= \frac{\|\boldsymbol{u}\|^2}{\gamma^2}\left(\frac{\gamma^2}{\|\boldsymbol{u}\|^2}\|\boldsymbol{u}\|^2\right)\\ &= \|\boldsymbol{u}\|^2\end{aligned}$$

We can now develop the expression for $\hat{y}$:

$$\begin{aligned}\hat{y} &= \gamma\cdot\frac{y-\mu_y}{\sigma_y}+\beta\\ &= \gamma\left(\frac{\|\boldsymbol{u}\|}{\gamma}\boldsymbol{w}^\top\boldsymbol{x}-\boldsymbol{0}\right)\left(\frac{1}{\|\boldsymbol{u}\|}\right)+\beta\\ &= \boldsymbol{w}^\top\boldsymbol{x}+\beta\end{aligned}$$

4.3 Show that the gradient of a loss function $L(\boldsymbol{u},\gamma,\beta)$ with respect to $\boldsymbol{u}$ can be written in the form $\nabla_{\boldsymbol{u}}L = s\boldsymbol{W}^\perp\nabla_{\boldsymbol{w}}L$ for some s, where $\boldsymbol{W}^\perp = \left(\boldsymbol{I}-\frac{\boldsymbol{u}\boldsymbol{u}^\top}{\|\boldsymbol{u}\|^2}\right)$. Note that [2] $\boldsymbol{W}^\perp\boldsymbol{u}=\boldsymbol{0}$.

Answer. We write the jacobian of a function $f(\boldsymbol{x})\in\mathbb{R}^m$ at a point $\boldsymbol{x}\in\mathbb{R}^n$ as $\nabla_{\boldsymbol{x}}f(\boldsymbol{x})\in\mathbb{R}^{m\times n}$. We may also write it as $\frac{\partial f(\boldsymbol{x})}{\partial\boldsymbol{x}}$. For a scalar function, we would have $\nabla_{\boldsymbol{u}}L\in\mathbb{R}^{1\times n}$. The loss function L is a function of $\boldsymbol{w}$ as well, so we make use of the chain derivative:

$$\begin{aligned}\nabla_{\boldsymbol{u}}L = \frac{\partial L}{\partial\boldsymbol{u}} &= \frac{\partial L}{\partial\boldsymbol{w}}\frac{\partial\boldsymbol{w}}{\partial\boldsymbol{u}}\\ &= \frac{\partial L}{\partial\boldsymbol{w}}\frac{\partial}{\partial\boldsymbol{u}}\left(\gamma\frac{\boldsymbol{u}}{\|\boldsymbol{u}\|}\right)\\ &= \frac{\partial L}{\partial\boldsymbol{w}}\left[\frac{\gamma}{\|\boldsymbol{u}\|}\frac{\partial\boldsymbol{u}}{\partial\boldsymbol{u}}+\gamma\boldsymbol{u}\frac{\partial}{\partial\boldsymbol{u}}\left(\frac{1}{\|\boldsymbol{u}\|}\right)\right]\\ &= \frac{\partial L}{\partial\boldsymbol{w}}\left[\frac{\gamma}{\|\boldsymbol{u}\|}\boldsymbol{I}+\gamma\boldsymbol{u}\frac{\partial}{\partial\boldsymbol{u}}\left(\frac{1}{\|\boldsymbol{u}\|}\right)\right]\end{aligned}$$

2. As a side note: $\boldsymbol{W}^\perp$ is an orthogonal complement that projects the gradient away from the direction of $\boldsymbol{w}$, which is usually (empirically) close to a dominant eigenvector of the covariance of the gradient. This helps to condition the landscape of the objective that we want to optimize.

We take a closer look at a single element of the derivative in the second term:

$$\begin{aligned}
\frac{\partial}{\partial u_i}\left(\frac{1}{\|\boldsymbol{u}\|}\right) &= \frac{\partial}{\partial u_i}\left(\frac{1}{\sqrt{u_1^2+u_2^2+\ldots+u_i^2+\ldots}}\right) \\
&= -\frac{1}{2}\frac{\partial}{\partial u_i}\left(u_1^2+u_2^2+\ldots+u_i^2+\ldots\right)\frac{1}{\|\boldsymbol{u}\|^3} \\
&= -\frac{u_i}{\|\boldsymbol{u}\|^3}
\end{aligned}$$

Returning to the derivation above, and recalling the appropriate dimension for the gradient of a scalar,

$$\begin{aligned}
\nabla_{\boldsymbol{u}} L &= \frac{\partial L}{\partial \boldsymbol{w}}\left(\frac{\gamma}{\|\boldsymbol{u}\|}\boldsymbol{I} - \frac{\gamma}{\|\boldsymbol{u}\|}\frac{\boldsymbol{u}\boldsymbol{u}^\top}{\|\boldsymbol{u}\|^2}\right) \\
&= \frac{\gamma}{\|\boldsymbol{u}\|}\nabla_{\boldsymbol{w}} L\left(\boldsymbol{I} - \frac{\boldsymbol{u}\boldsymbol{u}^\top}{\|\boldsymbol{u}\|^2}\right) \\
&= s\nabla_{\boldsymbol{w}} L \boldsymbol{W}^\perp
\end{aligned}$$

with $s = \frac{\gamma}{\|\boldsymbol{u}\|}$. Note that in the chosen convention for dimensions, the product is in reverse order than that of the question.