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 :
Using mathematical induction :
*Assuming that the base case holds.
*Assuming that the expression holds for time step $t-1$.
*Proving it for time step $t-1$:

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

$$\boldsymbol{g}_t = \sigma(\boldsymbol{W}(\sigma(\boldsymbol{W}\boldsymbol{g}_{t-2} + \boldsymbol{U}\boldsymbol{x}_{t-1} + \boldsymbol{b})) + \boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{b})$$

*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 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

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

Answer :
First, we need to calculate the gradients of the hidden state. Using Back Propagation Through Time (BPTT) algorithm and by applying chain rule, we get the error equation :

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W} = \sum_{t=1}^{T} \frac{\partial E}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

The term $\frac{\partial h_t}{\partial h_k}$ is the derivative of the hidden state at time t with respect to the hidden state at time k. This term involves products of Jacobians $\frac{\partial h_i}{\partial h_{i-1}}$:

$$\frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{k+1}}{\partial h_k}$$

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

Taking the derivative of h_i with respect to h_{i-1} where diag turns a vector into a diagonal matrix because this recursive partial derivative is a Jacobian matrix, hence :

$$\frac{\partial h_i}{\partial h_{i-1}} = diag(\sigma'(h_{i-1}))W$$

$$\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} diag(\sigma'(h_{i-1}))W$$

If we perform eigen decomposition on the Jacobian matrix $\frac{\partial h_i}{\partial h_{i-1}}$ given by $diag(\sigma'(h_{i-1}))W$ we get the eigenvalues $\lambda_1, \lambda_2 \cdots \lambda_n$ where $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$.
Let's take the norms of the Jacobian :

$$||\frac{\partial h_i}{\partial h_{i-1}}|| = ||diag(\sigma'(h_{i-1}))W||$$

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.

Using the property :$||\boldsymbol{AB}|| \leq ||\boldsymbol{A}||\,||\boldsymbol{B}||$, we get :

$$||\frac{\partial h_i}{\partial h_{i-1}}|| \leq ||W||.||diag(\sigma'(h_{i-1}))||$$

Focus now for the second part of the equation : $||W||.||diag(\sigma'(h_{i-1}))||$:
We know that :$\|D\sigma(\boldsymbol{x})\| \leq \gamma$, for some $\gamma > 0$ and for all $\boldsymbol{x}$, so :

$$||diag(\sigma'(h_{i-1}))|| \leq \gamma$$

Also, we have the property $||\boldsymbol{A}|| = \sqrt{\lambda_1(\boldsymbol{A}^\top \boldsymbol{A})}$, so :

$$||W|| = \sqrt{\lambda_1(W^\top W)}$$

$$||W||^2 = \lambda_1(W^\top W)$$

If the largest eigenvalue λ_1 of the weights is bounded by $\frac{\delta^2}{\gamma^2}$ for some $0 \leq \delta < 1$:

$$\lambda_1(\boldsymbol{W}^\top \boldsymbol{W}) \leq \frac{\delta^2}{\gamma^2}$$

At this point, we have :

$$||\frac{\partial h_i}{\partial h_{i-1}}|| \leq ||W^T||.||diag(\sigma'(h_{i-1}))|| \leq \frac{\delta}{\gamma}$$

The gradient above $\frac{\partial h_t}{\partial h_k}$ is a product of Jacobian matrices that are multiplied $t-k$ times.

$$||\frac{\partial h_t}{\partial h_k}|| = ||\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}|| \leq (\frac{\delta}{\gamma})^{t-k}$$

When $t \to \infty$,then the value of $\frac{\delta}{\gamma}$ will determine the gradient value.
Since $0 \leq \delta < 1$ and $\gamma > 0$, $\frac{\delta}{\gamma} < 1$ Therefore, gradients of the hidden state will vanish over time.

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 :
As mentioned in the previous question, when $t \rightarrow \infty$,then the value of $\frac{\delta}{\gamma}$ will determine the gradient value. Here, the largest eigenvalue of the weights is larger than $\frac{\delta^2}{\gamma^2}$ So, $\frac{\delta}{\gamma} > 1$ Therefore, gradients of the hidden state will explode over time, and it is a sufficient condition.

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

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

We will use the notation $\boldsymbol{w}_{0:t-1}$ to denote the (partial) sequence $(w_0, \dots, w_{t-1})$. 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 :
with a vocabulary size N, and a sequence of $T+1$ words, there are N^{T+1} different possible sequences.

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

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

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 :

$\delta_t(j)$ stores the probability of the best path ending in $w = j$ at time step t. This probability is calculated by maximising over the best ways of transmitting into $w = j$ for each $w = i$.

Therefore, to achieve $\delta_t(j)$, we need :
— the probability of being in state $w = i$ at time $t-1$, $\rightarrow \delta_{t-1}(i)$
— the transition probability from $w = i$ to $w = j$, $\rightarrow p(w_t = j \mid w_{t-1} = i) = [\boldsymbol{P}]_{i,j} = P_{ij}$
So,the recurrence relation satisfied by $\boldsymbol{\delta}_t$ (for $t > 0$) :

$$\boldsymbol{\delta}_t(j) = \boldsymbol{\delta}_{t-1}(i).P_{ij}$$

Proof for $t+1$:

$$\delta_{t+1}(j) = \max_{\boldsymbol{w}_{1:t}} p(\boldsymbol{w}_{1:t}, w_{t+1} = j)$$

$$\delta_{t+1}(j) = \max_{\boldsymbol{w}_{1:t}} p(w_{t+1} = j|\boldsymbol{w}_{1:t})p(\boldsymbol{w}_{1:t}, w_t)$$

Using *Markov property*

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

$$\delta_{t+1}(j) = p(w_{t+1} = j|\boldsymbol{w}_t = i) \max_{\boldsymbol{w}_{1:t-1}} p(\boldsymbol{w}_{1:t}, w_t = i)$$

$$\delta_{t+1}(j) = P_{ij}\delta_t(i)$$

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

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

2.2.c Let $\boldsymbol{a}_t \in \{1, \ldots, N\}^N$ 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 :

$$a_{t+1}(w_{t+1}^\star) = \arg\max_i P_{ij}\delta_t(i)(\arg\max_j \delta_{t+1}(j))$$

We have $\delta_{t+1}(j) = \boldsymbol{\delta}_t(i).P_{ij}$:

$$a_{t+1}(w_{t+1}^\star) = \arg\max_i P_{ij}\delta_t(i)(\arg\max_j P_{ij}\boldsymbol{\delta}_t(i))$$

$$a_{t+1}(w_{t+1}^\star) = \arg\max_j \max_i P_{ij}\delta_t(i)(P_{ij}\boldsymbol{\delta}_t(i))$$

$$a_{t+1}(w_{t+1}^\star) = \arg\max_j \boldsymbol{\delta}_t(j))$$

$$a_{t+1}(w_{t+1}^\star) = w_t^\star$$

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.*

Answer :

Viterbi Algoritm :

```
Initialization step
. Initialize the matrix P
. Initialize $\delta_0(j)$
Recurrence phase
for j=1 to N :
$\boldsymbol{\delta}_t(j) = \boldsymbol{\delta}_{t-1}(i).P_{ij}$
end for
$w^{\star}_T = \arg\max_j \delta_T(j)$.
back-tracing phase
for j=N-1 to 1 :
$w^{\star}_t = a_{t+1}(w^{\star}_{t+1})$
end for
return $w$
```

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 $\boldsymbol{w}^{\star}_{1:T}$ by enumerating all possible sequences (based on your answer to question 1).
Answer :
Space complexity : $O(TN)$, because we are storing a T*N sized matrix
Time complexity : $O(T^2N)$, because we need to do T work for each cell and T*T.N $= T^2N$
Compared to naively searching :

Viterbi algorithm uses much less memory than naive approach

In fact, the naive solution considers all the N^T possible paths, and returns the maximal one. The Viterbi algorithm solves the problem in $O(T^2N)$ time by exploiting a dynamic programming strategy

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 :
Viterbi decoding is inefficient when with large vocabulary or when a stronger language model, as LSTM,GRU. is used. A language model that can apply this algorithm to is speech signals.

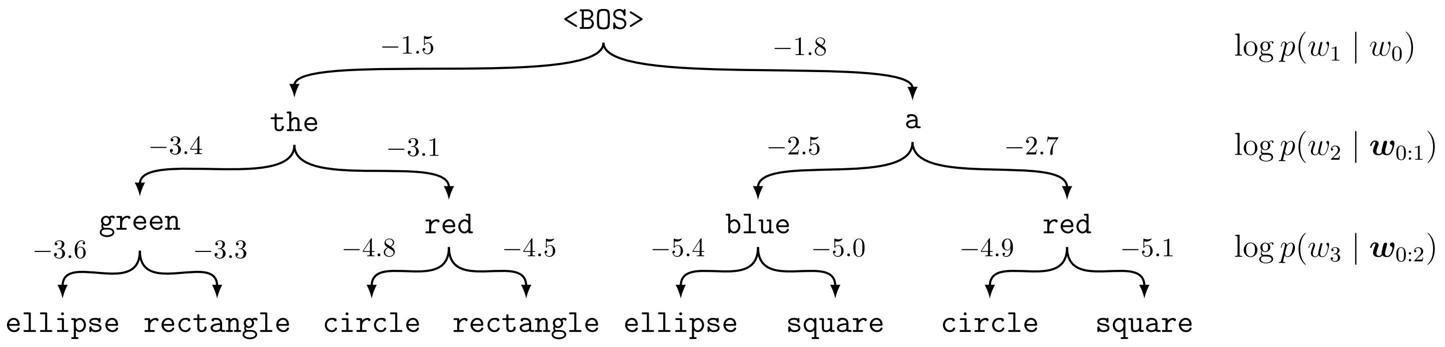
2.3 For real-world applications, the size of the vocabulary N can be very large (e.g. $N = 30$k for BERT, $N = 50$k for GPT-2), making even dynamic programming impractical. In order to generate B sequences having high likelihood, one can use a heuristic algorithm called *Beam search decoding*, whose pseudo-code is given in Algorithm 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?
Answer :
Time complexity : In the worst case, the Beam Search performs all the way to the deepest level in the search. At each subsequent time step, based on the B candidate output sequences at the previous time step, we continue to select B candidate output sequences with the highest conditional probabilities from B.N possible choices. So, we have $O(B.N.T)$ the worst case time complexity.
Space complexity : Beam algorithm only stores B nodes at each level in the search. So, worst-case space complexity is equal to $O(B.N.T)$.
Large vocabulary : Beam search does not need a lot of memory and it can maintain tractability in large vocabulary, but it is possible that the algorithm miss the goal node because it is not complete.

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 :
Viterbi decoding has underflow/overflow issues which is referred to the fact that there is a limit on how small (or large) of a number can be represented. The algorithm described above, we will regularly encounter very small numbers, because we are multiplying together a large number of probabilities. However, the solution is to work with logs probabilities (for logs of products of probabilities, use sum of logs) as mentioned here, so yes we can apply the viterbi algorithm. Also, we nned to know the transition probabilities for the hidden part of your model.
Find $\boldsymbol{w}^\star_{1:3}$, together with its corresponding log-likelihood $\log p(\boldsymbol{w}^\star_{1:3}) = \max_{\boldsymbol{w}_{1:3}} \log p(\boldsymbol{w}_{1:3})$:

$$\log p(\boldsymbol{w}^\star_{1:3}) = \max_{\boldsymbol{w}_{1:3}} \log p(\boldsymbol{w}_{1:3})$$

$$\log p(\boldsymbol{w}^\star_{1:3}) = \max((-1.8 - 2.7 - 5.1), (-1.8 - 2.7 - 4.9), (-1.8 - 2.5 - 5),$$

$$(-1.8-2.5-5.4), (-1.5, -3.1-4.5), (-1.5-3.1-4.8), (-1.5-3.4-3.3), (-1.5-3.4-3.6))$$

$$\log p(\boldsymbol{w}^\star_{1:3}) = \max(-9.6, -9.4, -9.3, -9.7, -9.1, -9.4, -8.2, -8.5)$$

$$\log p(\boldsymbol{w}^\star_{1:3}) = -8.2$$

$\boldsymbol{w}^\star_{1:3}$ is the , green,rectangle
and $\log p(\boldsymbol{w}^\star_{1:3}) = -8.2$

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 $\log p(\overline{\boldsymbol{w}}_{1:3})$.
Answer :
Step1 : $-1.5 > -1.8 \rightarrow$ the $(-.15)$
Step2 : $-3.1 > -3.4 \rightarrow$ red (-3.1)
Step3 : $-4.5 > -4.8 \rightarrow$ rectangle (-4.5)

$\boldsymbol{w}^{\star}_{1:3}$ is the , red,rectangle
and $\log p(\boldsymbol{w}^{\star}_{1:3}) = -9.1$

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 :
Step1 : take both of them because B=2 $\rightarrow$ a and the
Step2 : we take 2 maximum log p (because B=2) :
a $\rightarrow$ blue
a $\rightarrow$red

Step3 :
a $\rightarrow$ blue $\rightarrow$ square
a $\rightarrow$red $\rightarrow$ circle

$\boldsymbol{w}_{1:3}^{(1)}$ a,blue,square
and $\log p(\boldsymbol{w}_{1:3}^{(1)}) = -9.3$ $\boldsymbol{w}_{1:3}^{(2)}$ a,red,circle
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 search vs maximum likelihood : maximum likelihood is more accurate but has a large computational cost comparing to greedy search.
Beam search vs. Greedy search : Beam search is an improved version of greedy search, with beam as hyperparameter. Beam search considers multiple best options based on beamwidth using conditional probability, which is better than the sub-optimal.
Beam search vs. maximum likelihood : maximum likelihood is more accurate and Beam search uses less computational cost comparing to greedy search. Beam search performs between accuracy versus computational cost via its flexible choice of the beam size.

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 $L(f(\mathbf{x}^{(i)}, \theta), \mathbf{y}^{(i)})$, 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

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

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

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

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 :
The standard gradient descent method for weight parameter is given by :

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

The partial derivative(Gradient) of the L2 regularization :

$$\frac{\partial L_{\text{reg}}(f(\mathbf{x}^{(i)}, \theta), \mathbf{y}^{(i)})}{\partial \theta_i} = \frac{\partial L(f(\mathbf{x}^{(i)}, \theta), \mathbf{y}^{(i)})}{\partial \theta_i} + 2\gamma \theta_{\mathbf{i}}$$

We substitute the result in the gradient descent learning rule :

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

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

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

To make the weight decay equation identical to L2 regularization equation, we reparametrize the L2 regularization equation by replacing $\lambda = -2\frac{\eta}{m}\gamma$

3.2 This question refers to the Adam algorithm as described in the lecture slide (also identical to Algorithm 8.5 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 :
Referring to Adam Algorithm 8.5 of the deep learning book, the line that needs to change is :

Compute gradient : $g \leftarrow \frac{1}{m}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)}, \theta), \mathbf{y}^{(i)})$

The modification :

Compute gradient : $g \leftarrow \frac{1}{m}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)}, \theta), \mathbf{y}^{(i)}) + \gamma\|\theta\|_2^2$

3.3 Consider a "decoupled" weight decay scheme for the original Adam algorithm (see lecture slides, or equivalently, Algorithm 8.5 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 :

Answer :
With L2 regularization, the regularization term is added to the cost function which is then derived to calculate the gradients. Both types of gradients are normalized by their magnitudes, and therefore weights θ_{large} with large typical gradient magnitude are regularized by a smaller relative amount than other weights. However θ_{small} , with small gradient magnitude, would be regularized more strongly than θ_{large}.

(b) Would your answer change for the **Adam-weight-decay** scheme ? Why/why not ?
Answer :
With weight decay, only the gradients of the loss function are adapted. The moving averages of the gradient and its square keep track of the gradients of the loss function and the regularization term.
The weight decay is normalized by $\sqrt{\hat{\mathbf{r}}}$, for that reason, θ_{small}, with small gradient magnitude, the update step will be large, in contrast, θ_{large} would be regularized more strongly. Also weight decay regularizes all weights with the same rate, regularizing weights with large magnitude more than L2 regularization does.

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 :

Adaptive gradient based optimizers run on a loss function L plus L2 regularization, weights that tend to have large gradients in L do not get regularized as much as they would with decoupled weight decay, since the gradient of the regularizer gets scaled along with the gradient of L.In L2 regularization the gradient of the loss function and the gradient of the regularizer are normalized by their summed magnitudes, and therefore the weights with large typical gradient magnitude are regularized by a smaller relative amount than other weights. However, decoupled weight decay regularizes all weights with the same rate , effectively regularizing weights with large magnitude more than standard L2 regularization does. Therefore,weight decay is a better scheme to employ as opposed to L2 regularization in the context of adaptive gradient.

Note : my answer is based on paper :'DECOUPLED WEIGHT DECAY REGULARIZATION', Ilya Loshchilov Frank Hutter

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 **Batch Normalization**, mean and variance are calculated for each individual channel across all samples and both spatial dimensions :

$$\boldsymbol{\mu}_{batch} = \big[\big[2.7500, 2.7500, 2.0000\big], \big[1.7500, 2.7500, 3.2500\big]\big]$$

$$\boldsymbol{\sigma}^2_{batch} = \big[\big[1.1875, 0.1875, 0.0000\big], \big[0.6875, 0.6875, 0.6875\big]\big]$$

— In **Instance Normalization**, mean and variance are calculated for each individual channel for each individual sample across both spatial dimensions :

$$\boldsymbol{\mu}_{instance} = \big[\big[1, 2.5, 2.5\big], \big[2.5, 3.5, 3\big], \big[2.5, 2, 3\big], \big[3, 3, 2\big]\big]$$

$$\boldsymbol{\sigma}^2_{instance} = \big[\big[0, 0.25, 0.25\big], \big[0.25, 0.25, 1\big], \big[2.25, 0, 1\big], \big[0, 0, 0\big]\big]$$

— In **Layer Normalization**, mean and variance are calculated for each individual sample across all channels and both spatial dimensions :

$$\boldsymbol{\mu}_{layer} = \left[\begin{bmatrix}2\\2\end{bmatrix}, \begin{bmatrix}2.6667\\3.3333\end{bmatrix}, \begin{bmatrix}2.6667\\2.3333\end{bmatrix}, \begin{bmatrix}2.6667\\2.6667\end{bmatrix}\right]$$

$$\boldsymbol{\sigma}^2_{layer} = \left[\begin{bmatrix}0.6667\\0.6667\end{bmatrix}, \begin{bmatrix}0.2222\\0.8889\end{bmatrix}, \begin{bmatrix}0.8889\\1.5556\end{bmatrix}, \begin{bmatrix}0.2222\\0.2222\end{bmatrix}\right]$$

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 :

$$\hat{y} = \gamma \cdot \frac{y - \mu_y}{\sigma_y} + \beta$$

We Know that $y = \boldsymbol{u}^\top \boldsymbol{x}$:
Compute μ_y :

$$\mu_y = \mu_{\boldsymbol{u}^\top \boldsymbol{x}} = \boldsymbol{u}^\top .0 = 0$$

because $\mathbb{E}[\boldsymbol{x}] = \boldsymbol{0}$.
Compute σ_y :

$$\sigma_y = \sigma_{\boldsymbol{u}^\top \boldsymbol{x}} = (\boldsymbol{u}^\top)^2 .\boldsymbol{I} = (\boldsymbol{u}^\top)^2$$

because $\mathrm{Var}(\boldsymbol{x}) = \boldsymbol{I}$.

Replacing all the quantities in the first equation, we get :

$$\hat{y} = \gamma \cdot \frac{-\mu_y}{\sigma_y} + \beta$$

$$\hat{y} = \gamma \cdot \frac{\boldsymbol{u}^\top \boldsymbol{x} - 0}{||\boldsymbol{u}^\top||} + \beta$$

$$\hat{y} = \gamma \cdot \frac{\boldsymbol{u}^\top}{||\boldsymbol{u}^\top||} \boldsymbol{x} + \beta$$

$$\hat{y} = \boldsymbol{w}^\top \boldsymbol{x} + \beta$$

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 :

Applying chain rule :

$$\frac{\partial L}{\partial \boldsymbol{u}} = \frac{\partial L}{\partial \boldsymbol{W}}\frac{\partial \boldsymbol{W}}{\partial \boldsymbol{u}}$$

We have :

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

$$\frac{\partial \boldsymbol{W}}{\partial \boldsymbol{u}} = \gamma\frac{\partial}{\partial \boldsymbol{u}}\frac{\boldsymbol{u}}{||\boldsymbol{u}||}$$

$||\boldsymbol{u}||^2 = \boldsymbol{u}^\top\boldsymbol{u} \rightarrow 2||\boldsymbol{u}||||\boldsymbol{u}||' = 2\boldsymbol{u}^\top\boldsymbol{u}'$

$$||\boldsymbol{u}||' = \frac{\boldsymbol{u}^\top}{||\boldsymbol{u}||}\boldsymbol{u}'$$

$$\frac{\partial \boldsymbol{W}}{\partial \boldsymbol{u}} = \gamma\frac{\boldsymbol{u}'||\boldsymbol{u}|| - \boldsymbol{u}||\boldsymbol{u}||'}{||\boldsymbol{u}||^2}$$

$$\frac{\partial \boldsymbol{W}}{\partial \boldsymbol{u}} = \gamma\frac{\boldsymbol{u}'||\boldsymbol{u}|| - \boldsymbol{u}||\boldsymbol{u}||'}{||\boldsymbol{u}||^2}$$

$$\frac{\partial \boldsymbol{W}}{\partial \boldsymbol{u}} = \gamma\frac{\boldsymbol{u}'||\boldsymbol{u}|| - \boldsymbol{u}\frac{\boldsymbol{u}^\top}{||\boldsymbol{u}||}\boldsymbol{u}'}{||\boldsymbol{u}||^2}$$

$$\frac{\partial \boldsymbol{W}}{\partial \boldsymbol{u}} = \gamma\frac{\boldsymbol{I}}{||\boldsymbol{u}||}\left(\boldsymbol{I} - \frac{\boldsymbol{u}\boldsymbol{u}^\top}{||\boldsymbol{u}||^2}\right)$$

Therefore,

$$\frac{\partial L}{\partial \boldsymbol{u}} = \frac{\partial L}{\partial \boldsymbol{W}}\frac{\partial \boldsymbol{W}}{\partial \boldsymbol{u}}$$

$$\frac{\partial L}{\partial \boldsymbol{u}} = \frac{\partial L}{\partial \boldsymbol{W}}\gamma\frac{\boldsymbol{I}}{||\boldsymbol{u}||}\left(\boldsymbol{I} - \frac{\boldsymbol{u}\boldsymbol{u}^\top}{||\boldsymbol{u}||^2}\right)$$

$$\nabla_{\boldsymbol{u}} L = s\boldsymbol{W}^{\perp}\nabla_{\boldsymbol{w}} L$$

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.