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 defintions 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 $t-1$.

Assuming $g_0 = \sigma(h_0)$

$$\boldsymbol{g}_1 = \sigma(\boldsymbol{W}\boldsymbol{g}_0 + \boldsymbol{U}\boldsymbol{x}_1 + \boldsymbol{b})$$
$$\boldsymbol{h}_1 = \boldsymbol{W}\sigma(\boldsymbol{h}_0) + \boldsymbol{U}\boldsymbol{x}_1 + \boldsymbol{b}$$
$$\boldsymbol{h}_1 - \boldsymbol{W}\sigma(\boldsymbol{h}_0) = \boldsymbol{U}\boldsymbol{x}_1 + \boldsymbol{b}$$
$$\boldsymbol{g}_1 = \sigma(\boldsymbol{W}\boldsymbol{g}_0 + (\boldsymbol{h}_1 - \boldsymbol{W}\sigma(\boldsymbol{h}_0)))$$
$$\boldsymbol{g}_1 = \sigma(\boldsymbol{W}\cdot 0 + (\boldsymbol{h}_1 - \boldsymbol{W}\cdot 0))$$

Using induction case, we can assume that $g_{t-1} = \sigma(h_{t-1})$

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

*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.

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

Use the following properties of the L_2 operator norm

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

Multivariate Chain rule:

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

$$\frac{\partial h_t}{\partial h_{t-1}} = W \cdot \frac{\partial \sigma(h_{t-1})}{\partial h_{t-1}}$$

$$\left\|\frac{\partial h_t}{\partial h_{t-1}}\right\| = \left\|W \cdot \frac{\partial \sigma(h_{t-1})}{\partial h_{t-1}}\right\|$$

Use the following properties of the L_2 operator norm

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

$$\left\|\frac{\partial h_t}{\partial h_{t-1}}\right\| \leq \|W\| \cdot \left\|\frac{\partial \sigma(h_{t-1})}{\partial h_{t-1}}\right\|$$

$$\left\|\frac{\partial h_t}{\partial h_{t-1}}\right\| \leq \sqrt{\lambda(W^\top W)} \cdot \left\|\frac{\partial \sigma(h_{t-1})}{\partial h_{t-1}}\right\|$$

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

Substituting it back to the first equation, $\delta \in$ [0,1)

$$\frac{\partial h_T}{\partial h_0} \leq \Pi_{i=1}^T \left\|\frac{\partial h_i}{\partial h_{i-1}}\right\| \leq \delta^T$$

So as $T \to \infty$, δ^T tends to Zero

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).

In that case the, vanishing gradient would explode.

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

This is a necessary condition but not sufficient condition for the gradient to explode, because the product of the norms can be greater than norm of product, so we cant say that 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.
there are T words and N possibility available hence it N^T

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 $t > 0$).

$$\delta_t(j) = \max_j p(w_{1:t-1}, w_t = j)$$

$$\delta_t(j) = \max_j p(w_t = j | w_1, w_2, \cdots w_{t-1}) \cdot p(w_1, w_2, \cdots w_{t-1})$$

Using the markov property

$$\delta_t(j) = \max_j p(w_t = j | w_{t-1}) \cdot p(w_{t-1} | w_{t-2}) \cdot p(w_1 | w_0)$$

We know that $p(w_0)$ is 1 because of the <BOS>. the above equation can be written as following.

$$\delta_t(j) = \max_j P_{ij} \cdot \delta_{t-1}(i) \quad i = w_{t-1} \& t > 0$$

2.2.b Show that $w_T^\star = \arg\max_j \delta_T(j)$.
$w_T^\star = \arg\max_j P_{ij}\delta_{t-1}(i)$

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

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

Viterbi algorithm is nothing but finding the best possible way to find a way to start from <BOS> to <EOS>. As we have seen that the there are N^T combination possible, computing the sum of log probabilities is very expensive, in order to do that efficiently we use this we use a dynamic programming approach to solve it.

In short we start from state-'t' and see the best possible ways to come to state-'t' from state-'t-1'(markov property). Here we already would have a log-probability associated with state-'t-1' and to come to state-'t' there is a log probability. Considering this combination which has maximum log-probability , we choose that path.
$delta = \delta$ and $Trans_p = P$ in reference to previous question.

```
viterbi(delta, Trans_p):

    # Set internal variables
    s <- 0.0
    Y <- []
    Trellis <- empty TxN matrix
    Backpointers <- empty (T-1)xN matrix

    # Set first row of Trellis
    Trellis[0, :] <- delta[0,:]

    # Construct rest of Trellis table and keep Backpointers table
    for each word i in {1, ..., T-1}:
     for each label j in {0, ..., N-1}:
     Trellis[i,j] <- delta[i,j] + max_k(Trans_p[k,j] + Trellis[i,k])
     Backpointers[i-1,j] <- argmax_k(Trans_p[k,j] + Trellis[i,k])

    # Do backpropagation for remaining N-1 words
    for word i in {N-1, ..., 0}:
     b_next <- Backpointers[i, b_next]
     Y[i] <- b_next

    s = Trellis[-1,k] # Max log probability
    return (s,Y)  # Y is the sequence
```

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).

Two for loop on T and N, and then argmax or finding the Maximum log-probability at state results in the following time complexity.

Time Complexity - $O(TN * N)$

Space Complexity - $O(T * N)$

From Question 1 The the naive approach has N^T combination possible, but the viterbi algorithm reduces it to $O(TN * N)$.

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.

No, because Viterbi follows the markov property. So for the current step it see the previous step and makes a decision. Hence we cannot apply this algorithm to recurrent neural network will has long term dependencies.

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 $\boldsymbol{w}^{(b)}_{1:T}$ for $b \in \{1, \dots, B\}$
Initialization: $w^{(b)}_0 \leftarrow$ <BOS> for all $b \in \{1, \dots, B\}$
Initial log-likelihoods: $l^{(b)}_0 \leftarrow 0$ 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
            $s_b(j) \leftarrow l^{(b)}_{t-1} + \log p(w_t = j \mid \boldsymbol{w}^{(b)}_{0:t-1})$
    for $b = 1$ to $B$ do
        Find $(b', j)$ such that $s_{b'}(j)$ is the $b$-th largest score
        Save the partial sequence $b'$: $\widetilde{\boldsymbol{w}}^{(b)}_{0:t-1} \leftarrow \boldsymbol{w}^{(b')}_{0:t-1}$
        Add the word $j$ to the sequence $b$: $w^{(b)}_t \leftarrow j$
        Update the log-likelihood: $l^{(b)}_t \leftarrow s_{b'}(j)$
    Assign the partial sequences: $\boldsymbol{w}^{(b)}_{0:t-1} \leftarrow \widetilde{\boldsymbol{w}}^{(b)}_{0:t-1}$ for all $b \in \{1, \dots, B\}$
```

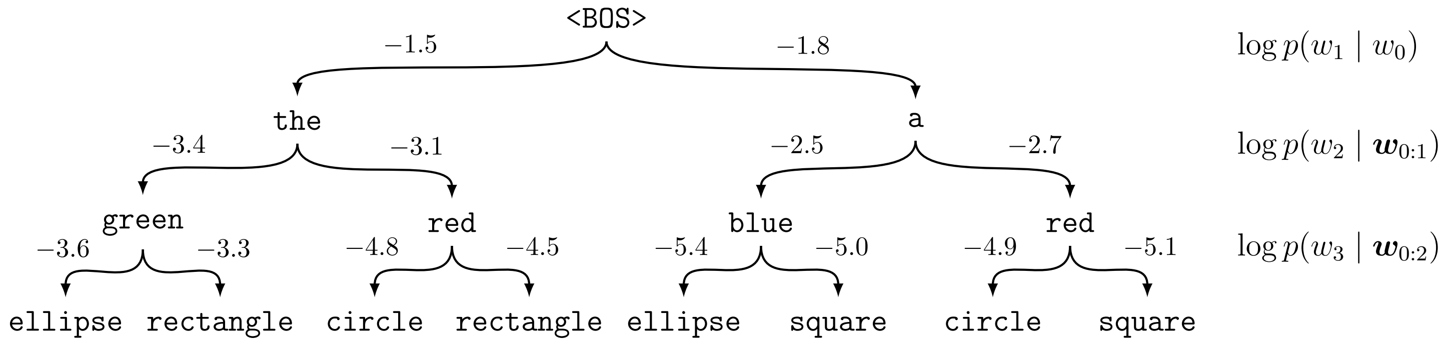
What is the time complexity of Algorithm 1 ? Its space complexity ? Write the algorithmic complexities using the O notation, as a function of T, B, and N. Is this a practical decoding algorithm when the size of the vocabulary is large ?

The algorithm depends on the sorting procedure use to find the top 'B' possibilities for the next step.

Case1:

The time complexity of this algorithm is O(T(BN + BN*log(BN))) if we use a sorting algorithm. The 1st BN is corresponding to B selection and N next possible combination. The 2nd BN*log(BN) is corresponding finding the B largest score depends on the sorting algorithm.
Case2:
we can use quick sort routine to find the top B Which might give us a complexity O(T(BN+BN)). The space complexity of this algorithm is $O(BN)$ - for storing the log probability and $O(BT)$ for storing the word sequence. Total $O(BN + BT)$. This algorithm is a practical approach for Larger vocabulary size. This algorithm works good for larger Vocabulary compared to the Viterbi mainly because of the Algorithm complexity of Beam is better than Viterbi. Viterbi has squared complexity on Vocabulary size whereas Beam is linear in Vocabulary size.

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

The green rectangle. Log probability is -1.5 + -3.4 + -3.3 = -8.2.
In order to use the Viterbi we need the $P(w_t|w_{t-1})$, whereas here they have given $P(w_t|w_{0:t-1})$.
Hence we cannot compute the Viterbi here .

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 $\overline{w}_0 = \texttt{<BOS>}$)

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

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

The red rectangle. The Log probability is -1.5+ -3.1+ -4.5 = -9.1, taking the max log-probability at each step.

2.4.c Apply beam search decoding (question 3) with a beam width $B = 2$ to this language model, and find $\boldsymbol{w}^{(1)}_{1:3}$ and $\boldsymbol{w}^{(2)}_{1:3}$, together with their respective log-likelihoods.
Here B = 2, so at each step 2 most probable words are chosen.
T = 1: the, a

T = 2: the green(-4.9), the red(-4.6), **a blue(-4.3), a red (-4.5)**
T = 3: a blue ellipse(-9.7), **a blue square(-9.3),** a red circle(-9.4), a red square(-9.6)
Solution is - **a blue square(-9.3)**

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 ?
Naive approach(Maximum likelihood) has a high complexity(N^T) hence it is not the best way compared to greedy in terms of the time complexity. The results show that Maximum likelihood(-8.2) approach had the better log-probability score compared to greedy (-9.1).

beam search has a log-probability score of (-9.3) which is bad compared to greedy approach. Overall the best scores were achieved with Maximum likelihood (-8.2). Beam search of 2 has not yield better results, hence we increasing the B value would be a good option.

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.

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

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

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

Hence, the modified SGD update is identical to an *L2 regularization* scheme that employs a standard SGD update rule with $\lambda = -2\eta\gamma$.

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.
In the compute gradient step of Adam is a follows

$$\frac{1}{m} \cdot \nabla_\theta \sum_i L(f(\mathbf{x}^{(j)}, \theta), \mathbf{y}^{(j)})$$

With regularization it becomes

$$\frac{1}{m} \cdot \nabla_\theta \left(\sum_i L(f(\mathbf{x}^{(j)}, \theta), \mathbf{y}^{(j)}) \right) + \gamma \|\theta\|_2^2$$

Hence the new gradient will be

$$\frac{1}{m} \cdot \nabla_\theta \left(\sum_i L(f(\mathbf{x}^{(j)}, \theta), \mathbf{y}^{(j)}) \right) + \gamma \nabla_\theta \|\theta\|_2^2$$

$$\frac{1}{m} \cdot \nabla_\theta \left(\sum_i L(f(\mathbf{x}^{(j)}, \theta), \mathbf{y}^{(j)}) \right) + 2\gamma\theta$$

the change is addition of $2\gamma\theta$ to the initial gradient.

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 ?
θ_{Small} once will have larger push in this case, because the $\hat{r}$ term has product of the gradient h. where h is the very small, compared to θ_{large}, θ_{Small} will have higher push as per the formula $\boldsymbol{\Delta}\theta = -\left(\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}}+\delta} +\right)$. hence in this case is θ_{Small}.

(b) Would your answer change for the **Adam-weight-decay** scheme ? Why/why not ?
As per the formula for Adam-weight-decay $\boldsymbol{\Delta}\theta = -\left(\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}}+\delta} + \lambda\theta\right)$. This is θ directly proportional to the update $\boldsymbol{\Delta}\theta$. In the question if they meant the partition was based on the gradient of θ then both of θ_{large} & θ_{small} would not have any partiality in the update. because it depends on the value of θ and here we information only about the gradient of the partition.

(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).
Weight decay is better compared to L2 regularization because, In the case of Adam-l2-reg θ_{Small} will have larger push than θ_{large}. This is not going to help the model generalize well. Whereas in

the case of Adam-weight-decay it doesnot have any partiality θ_{large} & θ_{small}. Also in the case of Adam-weight-decay the $\lambda\theta$ is directly added to the update, this would have better effect than the Adam-L2-reg where regularization gradient goes under moving average term for s and r. the complete effect of the regularization loss is not carried out in this case.

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.

BatchNorm:
We compute the Mean each the channel,
channel 1 = [1,3,2,3,3,2,4,2,2,3,3,2] $\mu_1 = 30/12 = 2.50\,;\, \sigma_1^2 = 0.58$
channel 2 = [1,2,3,2,4,4,1,2,4,3,3,2] $\mu_2 = 31/12 = 2.58\,;\, \sigma_2^2 = 1.07$.

$$\mu_{batch} = [2.50, 2.58]$$
$$\sigma^2_{batch} = [0.58, 1.07]$$

Layer Norm: Normalization is done for each batch

$$sample1 : \mu = 12/6 = 2.00; \sigma^2 = 0.66$$
$$sample2 : \mu = 18/6 = 3.00; \sigma^2 = 0.66$$
$$sample3 : \mu = 15/6 = 2.50; \sigma^2 = 1.25$$
$$sample4 : \mu = 16/6 = 2.66; \sigma^2 = 0.22$$
$$\mu_{layer} = [2.00; 3.00; 2.50; 2.66]$$
$$\sigma^2_{layer} = [0.66; 0.66; 1.25; 0.22]$$

Instance Norm :

$$sample : 1 : channel : 1 : \mu = 6/3 = 2.00; \sigma^2 = 0.66$$
$$sample : 1 : channel : 2 : \mu = 6/3 = 2.00; \sigma^2 = 0.66$$
$$sample : 2 : channel : 1 : \mu = 8/3 = 2.66; \sigma^2 = 0.22$$
$$sample : 2 : channel : 2 : \mu = 10/3 = 3.33; \sigma^2 = 0.88$$
$$sample : 3 : channel : 1 : \mu = 8/3 = 2.66; \sigma^2 = 0.88$$
$$sample : 3 : channel : 2 : \mu = 7/3 = 2.33; \sigma^2 = 1.55$$
$$sample : 4 : channel : 1 : \mu = 8/3 = 2.66; \sigma^2 = 0.22$$

$$sample : 4 : channel : 2 : \mu = 8/3 = 2.66; \sigma^2 = 0.22$$

$$\mu_{Inst} = [[2.00, 2.00]; [2.66, 3.33]; [2.66, 2.33]; [2.66, 2.66]]$$

$$\sigma^2_{Inst} = [[0.66, 0.66]; [0.22, 0.88]; [0.88, 1.55]; [0.22, 0.22]]$$

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

$$E(u^T \cdot x) = u^T \cdot x = u^T \cdot E(x) = 0$$

$$E(u^T \cdot x) = u^T \cdot x = u^T \cdot E(x) = 0$$

Similarly, $var(aX) = a^2 \cdot Var(x)$ using this we get $\sigma_y = ||u|| \cdot \sqrt{Var(I)}$. Subtitute the above equation into $\hat{y}$.

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

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

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

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] $\boldsymbol{W}^\perp \boldsymbol{u} = \boldsymbol{0}$.

$$\nabla_{\boldsymbol{u}} L = \frac{dw}{du} \nabla_{\boldsymbol{w}} L$$

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

Differentiate by quotient rule, and $\nabla_{\boldsymbol{u}} ||u|| = u^T / ||u||$ (gradient of the norm)

$$\frac{dw}{du} = \gamma \cdot \frac{||u|| - \frac{u \cdot u^\top}{||u||}}{||u^2||}$$

$$\nabla_{\boldsymbol{u}} L = \gamma \cdot \frac{||u|| - \frac{u \cdot u^\top}{||u||}}{||u^2||} \cdot \nabla_{\boldsymbol{w}} L$$

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.

$$\nabla_{\boldsymbol{u}} L = \frac{\gamma}{||u||} \cdot (I - \frac{u \cdot u^\top}{||u^2||}) \cdot \nabla_{\boldsymbol{w}} L$$

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