1. Backprop through a Simple RNN

(a) **Fill in the blanks for the intermediate values during the forward pass, in terms of w and the u_i's:**

$t = s + u_3 = wr + u_3 = w \cdot u_2 + w^2 u_1 + u_3$

$y = t \times w = wu_3 + w^2 u_2 + w^3 u_1$

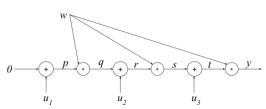
(b) **Using the expression for y from the previous subpart, compute $\frac{dy}{dw}$.**

$\frac{dy}{dw} = u_3 + 2wu_2 + 3w^2 u_1$

(c) **Fill in the blank for the missing partial derivative of y with respect to the nodes on the backward pass.** You may use values for p, q, r, s, t, y computed in the forward pass and downstream derivatives already computed.

$\frac{\partial y}{\partial p} = \frac{\partial y}{\partial q} \cdot w = \frac{\partial y}{\partial r} \cdot w = \frac{\partial y}{\partial s} \cdot w^2 = w^3$

(d) **Calculate the partial derivatives along each of the three outgoing edges from the learnable w in Figure 1, replicated below.** (e.g., the right-most edge has a relevant partial derivative of t in terms of how much the output y is effected by a small change in w as it influences y through this edge. You need to compute the partial derivatives for the other two edges yourself.)
You can write your answers in terms of the p, q, r, s, t and the partial derivatives of y with respect to them.
Use these three terms to find the total derivative $\frac{dy}{dw}$.



(HINT: You can use your answer to part (b) to check your work.)

$\frac{dy}{dw} = \frac{\partial y}{\partial q} \cdot \frac{\partial q}{\partial w} + \frac{\partial y}{\partial s} \cdot \frac{\partial s}{\partial w} + t$

$= \frac{\partial y}{\partial q} \cdot p + \frac{\partial y}{\partial s} \cdot r + t$

2. Beam Search

(a) **At timestep 1, which sequences is beam search storing?**

neural and network

(b) **At timestep 2, which sequences is beam search storing?**

The "neural" and "network" of log probability equaling -0.6

(c) **At timestep 3, which sequences is beam search storing?**

The "neural" and "network" of log probability -0.6

(d) **Does beam search return the overall most-likely sequence in this example? Explain why or why not.**

No. Because it will only store the k most possible sequences. in this case, k=2

(e) **What is the runtime complexity of generating a length-T sequence with beam size k with an RNN?** Answer in terms of T and k and M. (Note: an earlier version of this question said to write it in terms of just T and k. This answer is also acceptable.)

$O(Tk^2 \log K)$

3. Implementing RNNs (and optionally, LSTMs)

(a) **Implement Section 1A in the notebook**, which constructs a vanilla RNN layer. This layer implements the function

```
Expected: 0.1973753273487091, got: 0.1973753273487091, max error: 0.0
Max error all_h: 4.999339580535889e-05
Max error last_h: 2.498924732208252e-05
```

(b) **Implement Section 1.B of the notebook**, in which you'll use this RNN layer in a regression model by adding a final linear layer on top of the RNN outputs.

$$\hat{y}_t = W^f h_t + b^f$$

We'll compute one prediction for each timestep.

Copy the outputs of the "*Tests*" code cell and paste it into your submission of the written assignment.

```
Max error all_h: 4.699826240539551e-05
Max error last_h: 4.3138861656188965e-05
```

(c) RNNs can be used for many kinds of prediction problems, as shown below. In this notebook we will look at many-to-one prediction and aligned many-to-many prediction.

Implement Section 1.C in the notebook, in which you'll look at the synthetic dataset shown and implement a loss function for the two problem variants.

Copy the outputs of the "*Tests*" code cell and paste it into your submission of the written assignment.

```
Max error loss_all: 3.314018249511719e-05
Max error loss_last: 2.384185791015625e-07
```

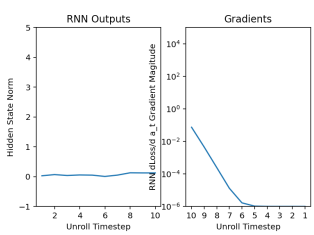
(d) Consider an RNN which outputs a single prediction at timestep T. As shown in Figure 4, each weight matrix W influences the loss by multiple paths. As a result, the gradient is also summed over multiple paths:

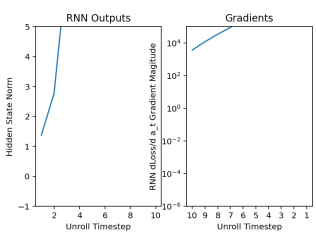
$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial h_T}\frac{\partial h_T}{\partial W} + \frac{\partial \mathcal{L}}{\partial h_{T-1}}\frac{\partial h_{T-1}}{\partial W} + \ldots + \frac{\partial \mathcal{L}}{\partial h_1}\frac{\partial h_1}{\partial W} \tag{2}$$

When you backpropagate a loss through many timesteps, the later terms in this sum often end up with either very small or very large magnitude - called vanishing or exploding gradients respectively. Either problem can make learning with long sequences difficult.

Implement Notebook Section 1.D, which plots the magnitude at each timestep of $\frac{\partial \mathcal{L}}{\partial h_t}$. Play around with this visualization tool and try to generate exploding and vanishing gradients.

Include a screenshot of your visualization in the written assignment submission.





n 1.

(e) **If the network has no nonlinearities, under what conditions would you expect the exploding or vanishing gradients with for long sequences? Why?** (Hint: it might be helpful to write out the formula for $\frac{\partial \mathcal{L}}{\partial h_t}$ and analyze how this changes with different t). **Do you see this pattern empirically using the visualization tool in Section 1.D in the notebook** with last_step_only=True?

Because, the W will continue to multiply for some certain h_t, when W is smaller than then the h_t term will become zero.

Yes

(f) Compare the magnitude of hidden states and gradients when using ReLU and tanh nonlinearities in Section 1.D in the notebook. **Which activation results in more vanishing and exploding gradients? Why?** (This does not have to be a rigorous mathematical explanation.)

① ReLU will lead more to exploding gradients because when the value is greater than zero, then the value will be preserved by ReLU, and negative value will become zero, which makes that in some cases, the value can be very large and cause gradient exploding

② tanh will lead more to vanishing gradients because it output between 0 and 1. When the input is very large, its gradient will be close to zero, causing gradient vanishing.

(g) **What happens if you set last_target_only = False in Section 1.D in the notebook? Explain why this change affects vanishing gradients. Does it help the network's ability to learn dependencies across long sequences?** (The explanation can be intuitive, not mathematically rigorous.)

It can reduce the vanishing gradients, it helps the network to learn dependencies across long sequences

Reason: It can provide a more strong signal to reduce vanishing gradients, which means that by set the last_target_only to false, we can increase the magnitude of the gradients

4. RNNs for Last Name Classification

Please follow the instructions in this notebook. You will train a neural network to predict the probable language of origin for a given last name / family name in Latin alphabets. Once you finished with the notebook, download `submission_log.json` and submit it to "Homework 6 (Code)" in Gradescope.

(a) Although the neural network you have trained is intended to predict the language of origin for a given last name, it could potentially be misused. **In what ways do you think this could be problematic in real-world applications**?

It can know where a person comes from through his/her name, which is a leakage of personal information, causing

5. Read a Blog Post: How to train your Resnet

(a) **What is the baseline training time and accuracy the authors started with? What was the final training time and accuracy achieved by the authors?**

Baseline: training time: 341s accuracy = 94%

Final: training time: 26s accuracy: 94%

(b) **Comment on what you have learnt.** (≈ 100 words)

To improve the training time of a model, we have the following ideas. Firstly, we can remove the repetition architecture of the network. Secondly, we can delete the duplicate preprocessing operation, for example, we can keep the data loader alive all the time. And there are also other effective methods, such as preprocessing the GPU, moving max-pool layers, label smoothing, using CELU activation, using ghost batch norm, frozen batch norm scales, input patch whitening, and exponential moving averages and test-time augmentations.

(c) **Which approach taken by the authors interested you the most? Why?** (≈ 100 words)

The approach that impresses me most is label smoothing, which will make the boundary of the true value and the false value more ambiguous, resulting in the improvement of the generation ability of the model, and preventing the model from overfitting. And the improvement of generation ability will lead the model to converge faster since it doesn't need to use more parameters to simulate the target function. That method impresses me the most.

6. Convolutional Networks

Note: Throughout this problem, we will use the convention of NOT flipping the filter before dragging it over the signal. This is the standard notation with neural networks (ie, we assume the filter given to us is already flipped)

(a) **List two reasons we typically prefer convolutional layers instead of fully connected layers when working with image data.**

① It can extract the local features of image (translation invariance) a part of

② It can use weight sharing to reduce the cost of heavy computation and less prone to overfitting

(b) Consider the following 1D signal: $[1, 4, 0, -2, 3]$. After convolution with a length-3 filter, no padding, stride=1, we get the following sequence: $[-2, 2, 11]$. **What was the filter?**

(Hint: Just to help you check your work, the first entry in the filter that you should find is 2. However, if you try to use this hint directly to solve for the answer, you will not get credit since this hint only exists to help you check your work.)

Assume that the filter is $[x\ y\ z]$, then we have the following equations:

$$\begin{cases} x+4y=-2 \\ 4x-2z=2 \\ -2y+3z=11 \end{cases} \Rightarrow \begin{cases} x=2 \\ y=-1 \\ z=3 \end{cases}$$

the filter is $[2\ -1\ 3]$

(c) Transpose convolution is an operation to help us upsample a signal (increase the resolution). For example, if our original signal were $[a, b, c]$ and we perform transpose convolution with pad=0 and stride=2, with the filter $[x, y, z]$, the output would be $[ax, ay, az + bx, by, bz + cx, cy, cz]$. Notice that the entries of the input are multipled by each of the entries of the filter. Overlaps are summed. Also notice how for a fixed filtersize and stride, the dimensions of the input and output are swapped compared to standard convolution. (For example, if we did standard convolution on a length-7 sequence with filtersize of 3 and stride=2, we would output a length-3 sequence).

If our 2D input is $\begin{bmatrix} -1 & 2 \\ 3 & 1 \end{bmatrix}$ and the 2D filter is $\begin{bmatrix} +1 & -1 \\ 0 & +1 \end{bmatrix}$ **What is the output of transpose convolution with pad=0 and stride=1?**

$$= \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 2 & -2 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 3 & -3 & 0 \\ 0 & 3 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -1 & 3 & -2 \\ 3 & -3 & 1 \\ 0 & 3 & 1 \end{bmatrix}$$