HW6

Mengying Lin SID: 3038737132.

1. Backprop through a Simple RNN

Consider the following 1D RNN with no nonlinearities, a 1D hidden state, and 1D inputs u_t at each timestep. (Note: There is only a single parameter w, no bias). This RNN expresses unrolling the following recurrence relation, with hidden state h_t at unrolling step t given by:

$$h_t = w \cdot (u_t + h_{t-1}) \tag{1}$$

The computational graph of unrolling the RNN for three timesteps is shown below:

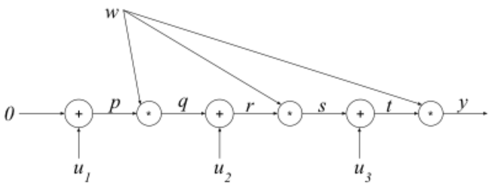


Figure 1: Illustrating the weight-sharing and intermediate results in the RNN.

where w is the learnable weight, u_1, u_2, and u_3 are sequential inputs, and p, q, r, s, and t are intermediate values.

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

$p = u_1$ $\quad q = w \cdot u_1$ $\quad r = u_2 + q = u_2 + w \cdot u_1$

$s = w \cdot r = w \cdot u_2 + w^2 \cdot u_1$

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

$y =$ $w \cdot t = w u_3 + w^2 \cdot u_2 + w^3 \cdot u_1$

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

$$\frac{dy}{dw} = u_3 + 2w u_2 + 3w^2 u_1$$

(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.

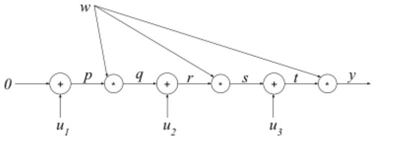
$$\frac{\partial y}{\partial t} = w \qquad \frac{\partial y}{\partial s} = w \qquad \frac{\partial y}{\partial r} = \frac{\partial y}{\partial s} \cdot w \qquad \frac{\partial y}{\partial q} = \frac{\partial y}{\partial r} \cdot 1$$

$\frac{\partial y}{\partial p} =$ $\frac{\partial y}{\partial q} \cdot w$

(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.)

$$y = w(s+u_3)$$

$$\frac{dy}{\partial w} = t + w\frac{\partial(s+u_3)}{\partial w}$$

$$= t + w\frac{\partial s}{\partial w} = t + w\left[r + w\frac{\partial r}{\partial w}\right]$$

$$= t + wr + w^2\frac{\partial q_t}{\partial w} = t + wr + w^2 u_1$$

$$= t + s + wq_t$$

$$= t + s + \frac{\partial y}{\partial t}q_t$$

2. Beam Search

This problem will also be covered in discussion.

When making predictions with an autoregressive sequence model, it can be intractable to decode the true most likely sequence of the model, as doing so would require exhaustively searching the tree of all $O(M^T)$ possible sequences, where M is the size of our vocabulary, and T is the max length of a sequence. We could decode our sequence by greedily decoding the most likely token each timestep, and this can work to some extent, but there are no guarantees that this sequence is the actual most likely sequence of our model.

Instead, we can use beam search to limit our search to only candidate sequences that are the most likely so far. In beam search, we keep track of the k most likely predictions of our model so far. At each timestep, we expand our predictions to all of the possible expansions of these sequences after one token, and then we keep only the top k of the most likely sequences out of these. In the end, we return the most likely sequence out of our final candidate sequences. This is also not guaranteed to be the true most likely sequence, but it is usually better than the result of just greedy decoding.

The beam search procedure can be written as the following pseudocode:

Algorithm 1 Beam Search

```
for each time step $t$ do
    for each hypothesis $y_{1:t-1,i}$ that we are tracking do
        find the top $k$ tokens $y_{t,i,1},...,y_{t,i,k}$
    end for
    sort the resulting $k^2$ length $t$ sequences by their total log-probability
    store the top $k$
    advance each hypothesis to time $t+1$
end for
```

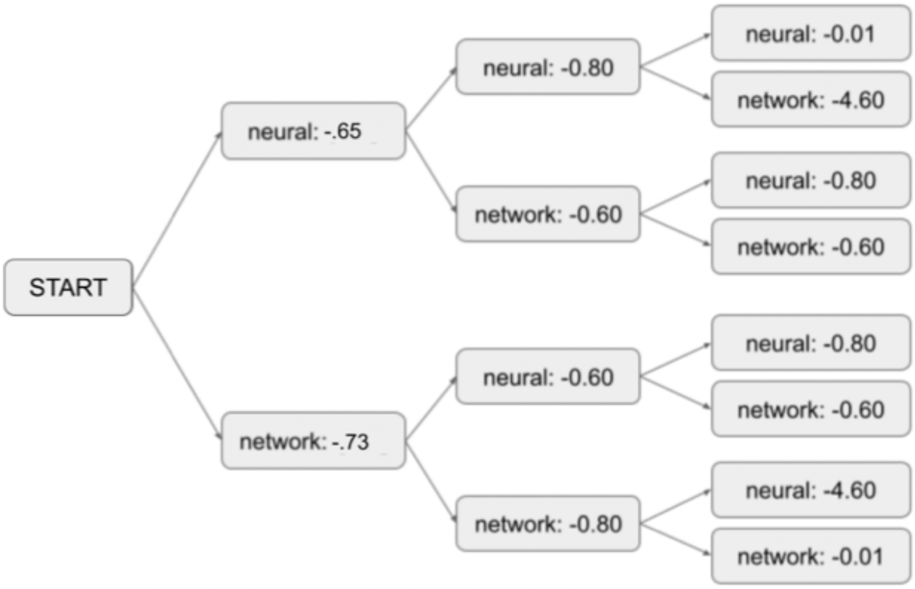
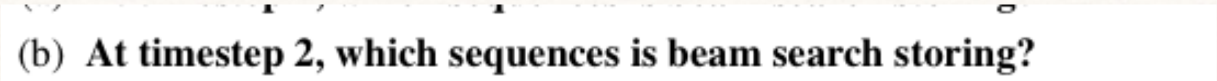


Figure 2: The numbers shown are the decoder's log probability prediction of the current token given previous tokens.

We are running the beam search to decode a sequence of length 3 using a beam search with $k = 2$. Consider predictions of a decoder in Figure 2, where each node in the tree represents the next token **log probability** prediction of one step of the decoder conditioned on previous tokens. The vocabulary consists of two words: "neural" and "network".

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

a). neural, network.



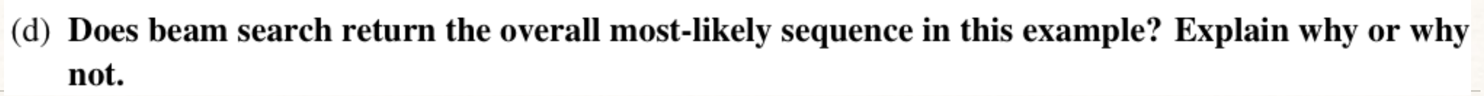
(b) **At timestep 2, which sequences is beam search storing?**

b) neural-network, network-neural

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

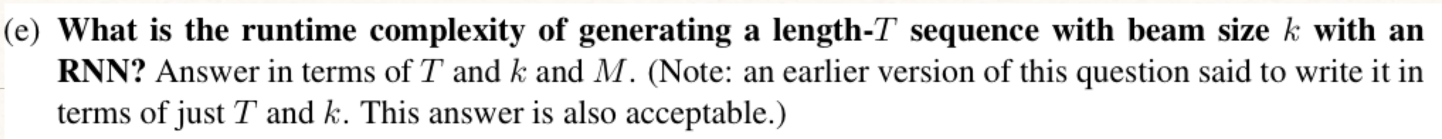
c) neural-network-neural

network-neural-network.



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

No. The sequence "network-network-network" with highest log-probability is not chosen, because this branch has been discarded due to the relatively low probability of "network-network".



(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.)

To figure out the top k sequences out of M candidates needs $O(M\log k)$.

To traverse the k given "beams" needs $O(k)$

Hence, the runtime complexity is $O(kTM\log k)$.

3. Implementing RNNs (and optionally, LSTMs)

This problem involves filling out this notebook.
Note that implementing the LSTM portion of this question is optional and out-of-scope for the exam.

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

$$h_t = \sigma(W^h h_{t-1} + W^x x_t + b)$$

where W^h, W^x, and b are learned parameter matrices, x is the input sequence, and σ is a nonlinearity such as tanh. The RNN layer "unrolls" across a sequence, passing a hidden state between timesteps and returning an array of hidden states at all timesteps.

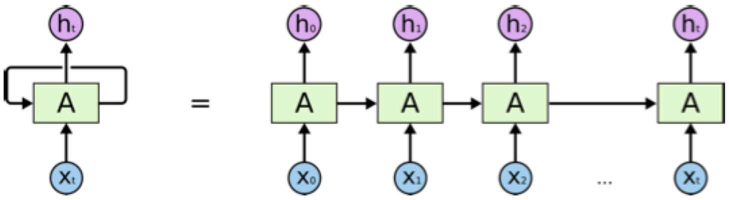
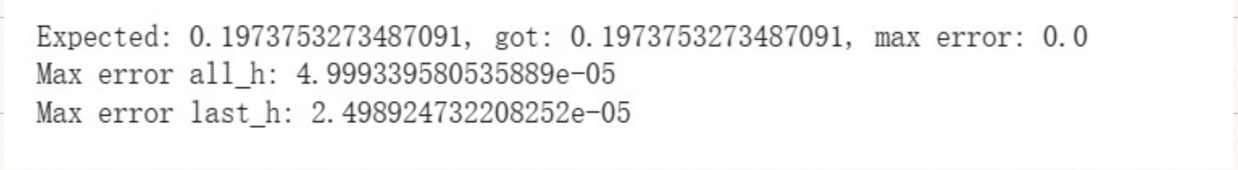


Figure 3: Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

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



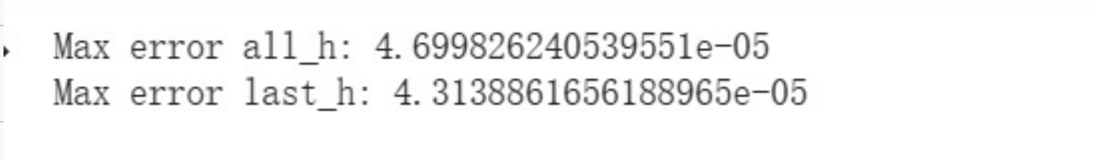
```
Expected: 0.1973753273487091, got: 0.1973753273487091, max error: 0.0
Max error all_h: 4.999339580535889e-05
Max error last_h: 2.498924732208252e-05
```

(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.

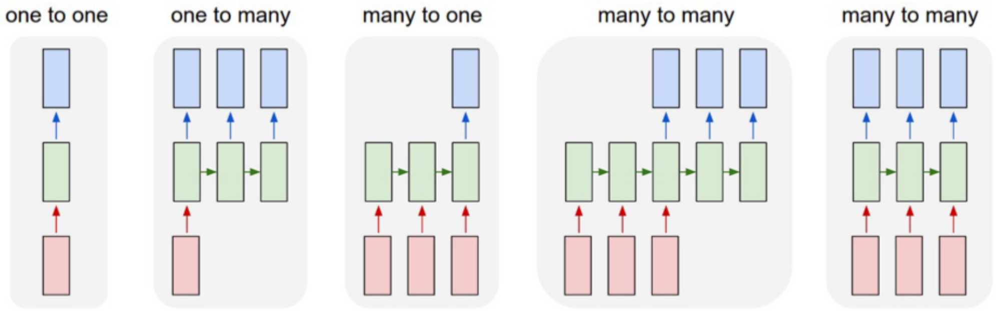
Copy the outputs of the *"Tests"* code cell and paste it into your submission of the written assignment.



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

ment.

(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.



We will use a simple averaging task. The input X consists of a sequence of numbers, and the label y is a running average of all numbers seen so far.

We will consider two tasks with this dataset:

- Task 1: predict the running average at all timesteps
- Task 2: predict the average at the last timestep only

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: 0.6933331489562988
```

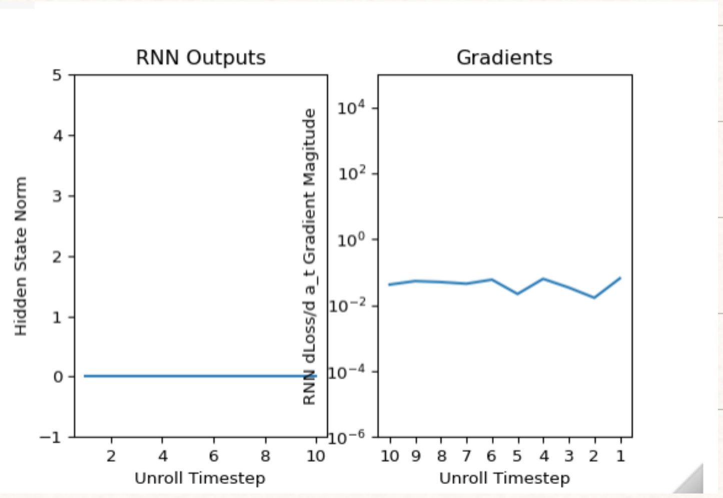
(d) Consider an RNN which outputs a single prediction at timestep T. As shown in Figure 4, each weight matrix W influences the loss by multiple paths. As a result, the gradient is also summed over multiple paths:

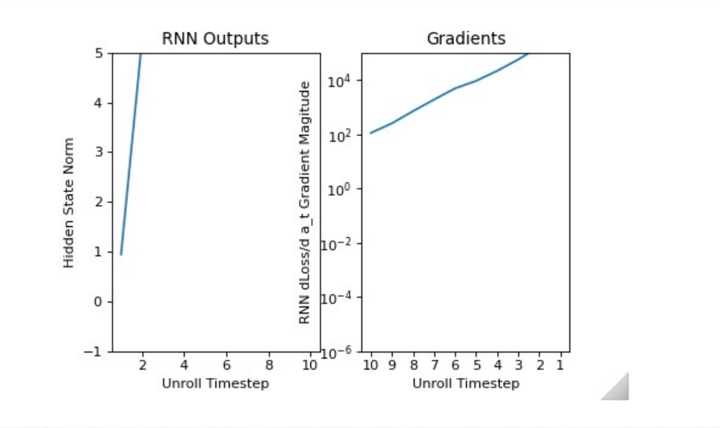
$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial h_T}\frac{\partial h_T}{\partial W} + \frac{\partial \mathcal{L}}{\partial h_{T-1}}\frac{\partial h_{T-1}}{\partial W} + \ldots + \frac{\partial \mathcal{L}}{\partial h_1}\frac{\partial h_1}{\partial W} \tag{2}$$

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.





(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?

when $\sigma(x)=x$,

$$\frac{\partial L}{\partial h_t} = \sum_{l=t}^{T} \frac{\partial L}{\partial \hat{y}_l} \cdot \frac{\partial \hat{y}_l}{\partial h_t}$$

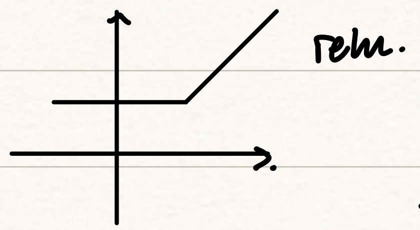
$$= 2(\hat{y}-y) W^{T} [W^{h} + \cdots W^{h(T-t+1)}]$$

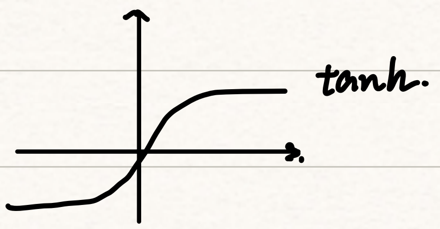
If there exists singular value of W^h greater than 1, the gradients will explode. Otherwise, they are more likely to vanish.

(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.)

Relu results more exploding gradients, while tanh's are more vanishing.

This can be intuitively explained through examing the plot of both functions:

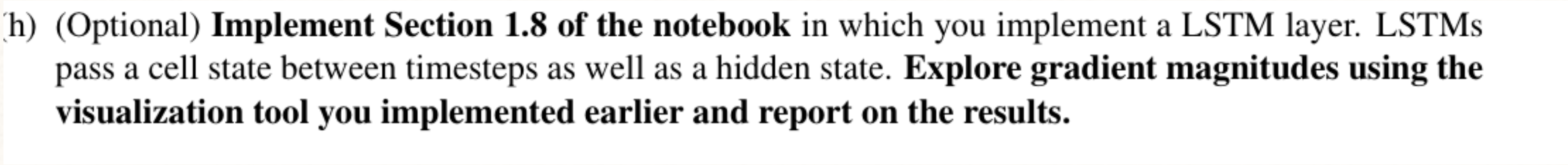




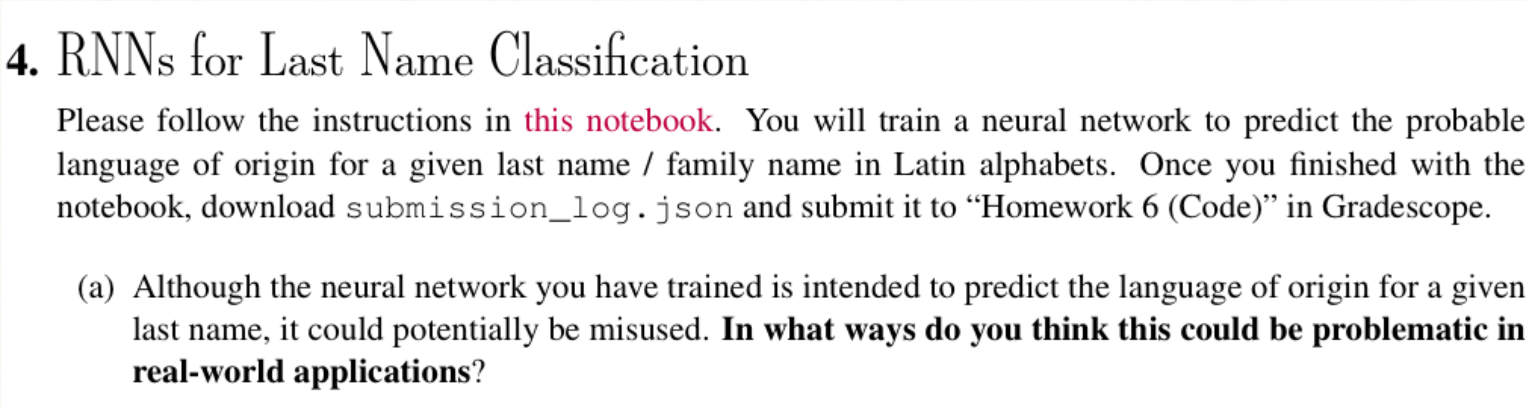
Relu is non-saturating while tanh is not.

(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.)

?



(h) (Optional) **Implement Section 1.8 of the notebook** in which you implement a LSTM layer. LSTMs pass a cell state between timesteps as well as a hidden state. **Explore gradient magnitudes using the visualization tool you implemented earlier and report on the results.**

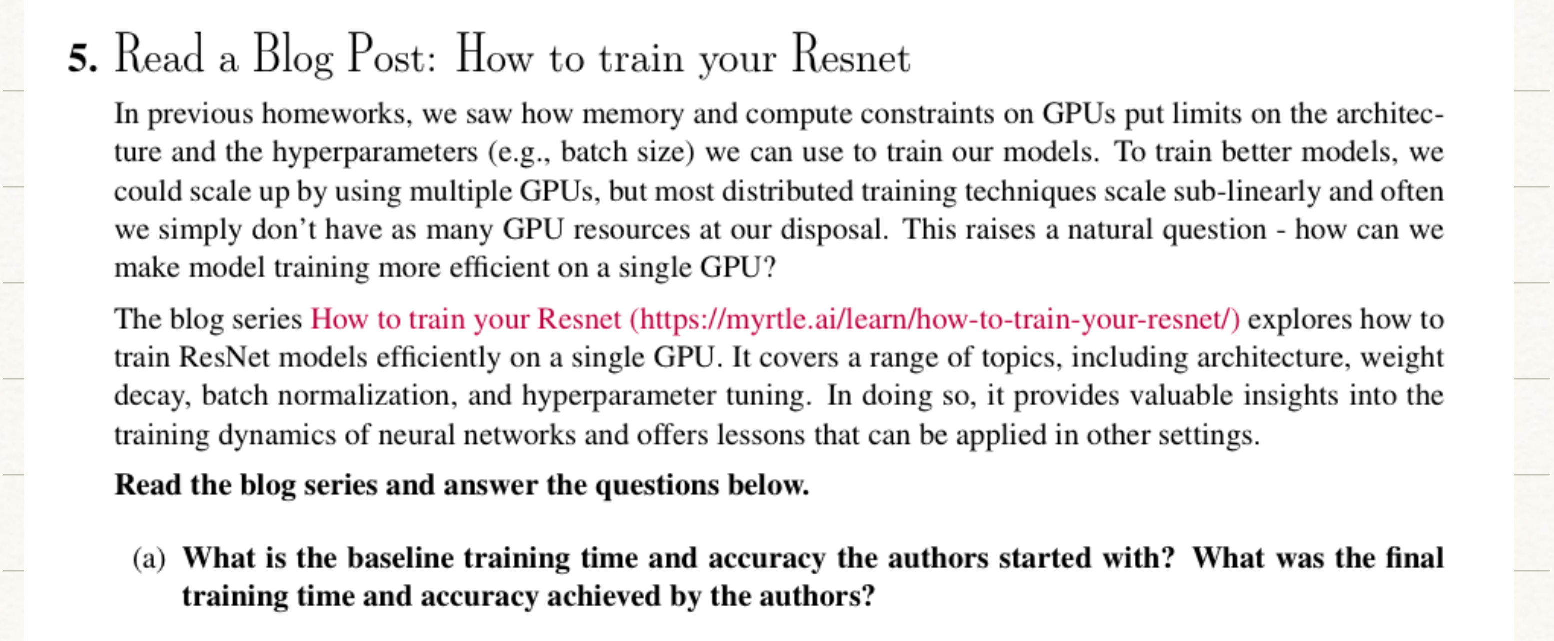


4. RNNs for Last Name Classification

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**?

Some institutions might predict their users'/candidates' ethnicity and deprive them of good service/job opportunities according to the predictions.



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

In previous homeworks, we saw how memory and compute constraints on GPUs put limits on the architecture and the hyperparameters (e.g., batch size) we can use to train our models. To train better models, we could scale up by using multiple GPUs, but most distributed training techniques scale sub-linearly and often we simply don't have as many GPU resources at our disposal. This raises a natural question - how can we make model training more efficient on a single GPU?

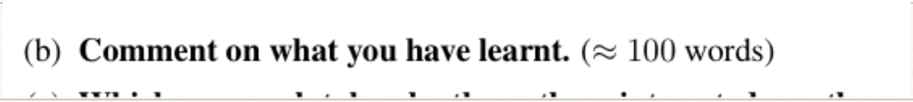
The blog series How to train your Resnet (https://myrtle.ai/learn/how-to-train-your-resnet/) explores how to train ResNet models efficiently on a single GPU. It covers a range of topics, including architecture, weight decay, batch normalization, and hyperparameter tuning. In doing so, it provides valuable insights into the training dynamics of neural networks and offers lessons that can be applied in other settings.

Read the blog series and answer the questions below.

(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?**

① 297s .94%

② 26s, 94.1%.

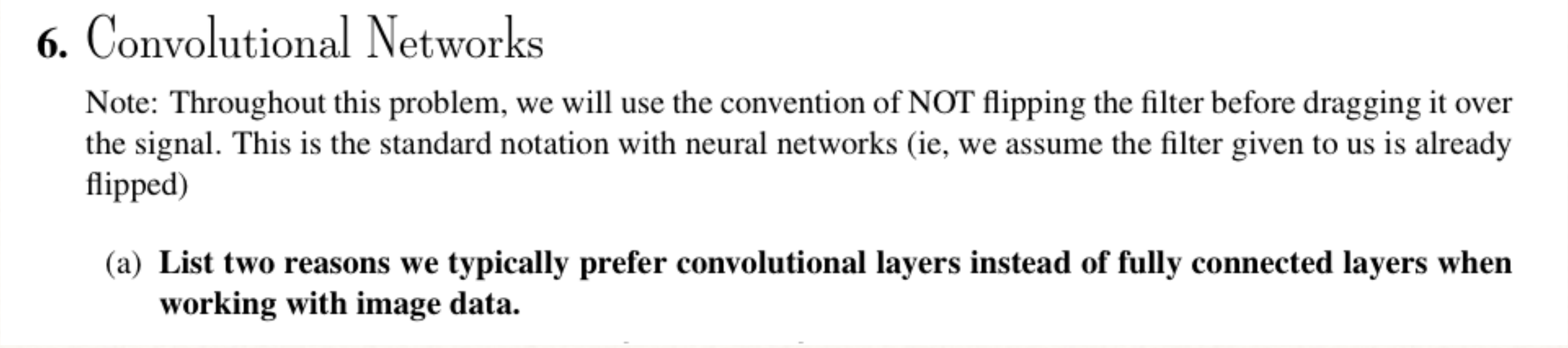


(b) **Comment on what you have learnt.** (≈ 100 words)

Choosing moderate batch size, adding regularization and BN, refining the architecture of model (e.g. add residual blocks), are all good practices to shrink training time.

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

I'm impressed by the way they tune their hyperparameters. Instead of doing coordinate decent in (λ, ρ, α) space, they perform the descent in ($\frac{\lambda\alpha}{1-\rho}$, ρ, α) space, so that the most flat direction is aligned with the axes.



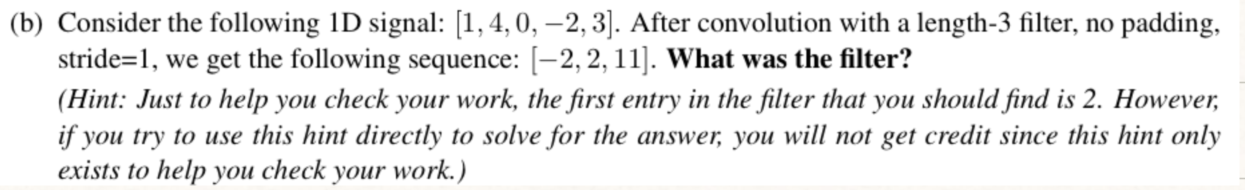
6. Convolutional Networks

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.**

a). Convolutional layers can greatly reduce the amount of parameters through weight-sharing, while fully connected layers have large amount of parameters and take connections between every pair of neurons into consideration, which demands rather large computing power.

In addition, conv layers pay more attention to local spatial information which are usually helpful in image-related tasks.

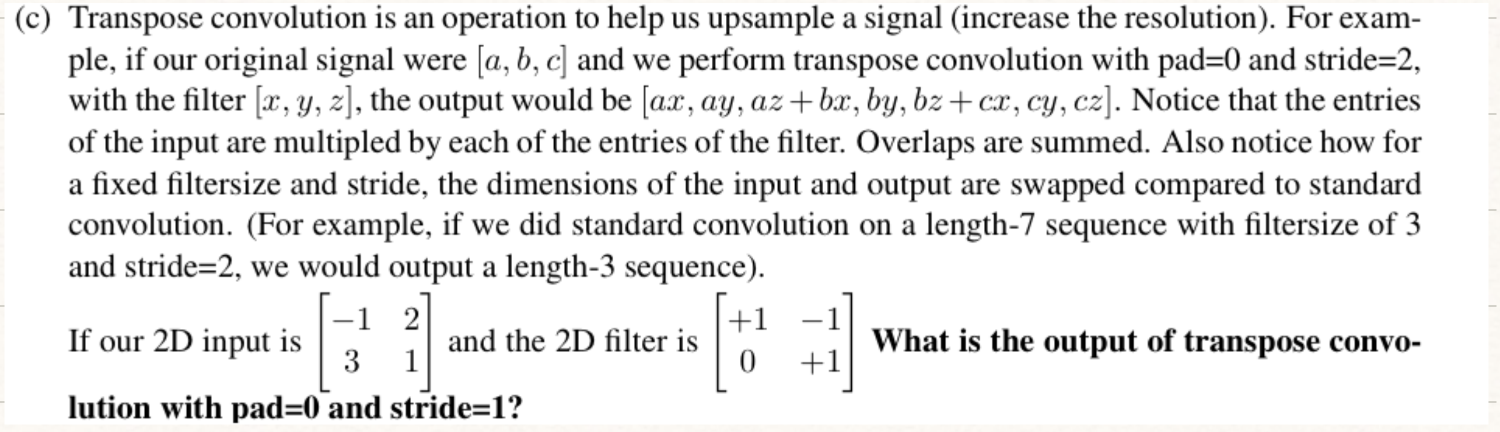


(b) Consider the following 1D signal: $[1, 4, 0, -2, 3]$. After convolution with a length-3 filter, no padding, stride=1, we get the following sequence: $[-2, 2, 11]$. **What was the filter?**

(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.)

Let filter $F = [f_1, f_2, f_3]$.

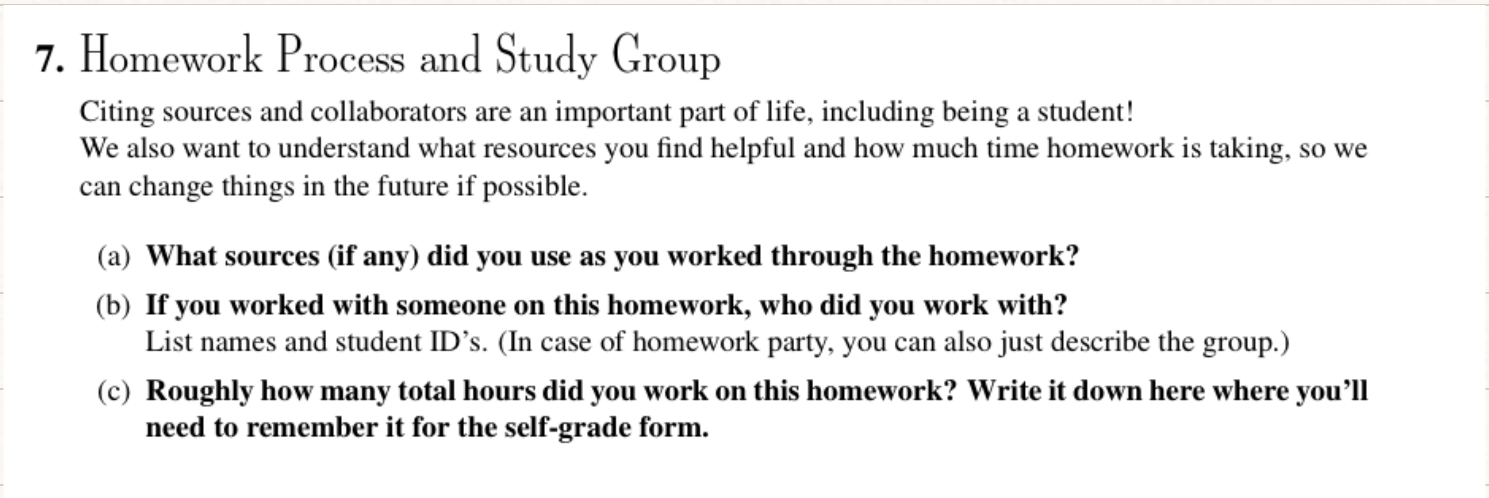
$$\begin{cases} f_1 + 4f_2 = -2 \\ 4f_1 - 2f_3 = 2 \\ -2f_2 + 3f_3 = 11 \end{cases} \Rightarrow \begin{cases} f_1 = 2. \\ f_2 = -1 \\ f_3 = 3 \end{cases}$$



(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?**

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



7. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student!
We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

(a) **What sources (if any) did you use as you worked through the homework?**

(b) **If you worked with someone on this homework, who did you work with?**
List names and student ID's. (In case of homework party, you can also just describe the group.)

(c) **Roughly how many total hours did you work on this homework? Write it down here where you'll need to remember it for the self-grade form.**

a) None.

b) Name : Xiang Fei

SID : 3038733024.

c) 15h