Data C182 Designing, Visualizing & Understanding DNN
Fall 2024 Eric Kim, Naveen Ashish

Homework 2

This homework is due on October 24, at 11:59PM.

1. 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.

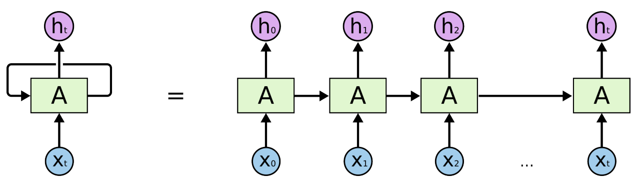


Figure 1: Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

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

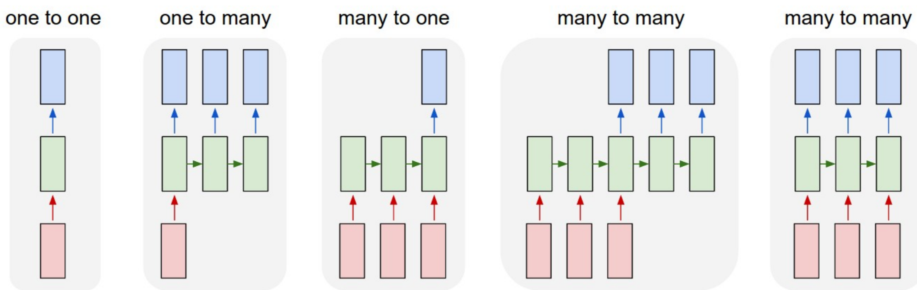
(b) **Implement Section 1.B of the notebook**, in which you'll use this RNN layer in a regression model by adding a final linear layer on top of the RNN outputs.

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

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.

(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.

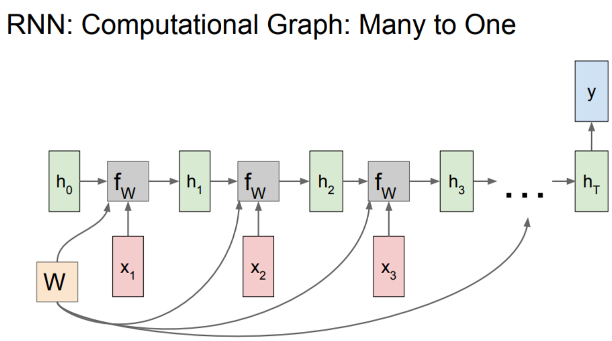


Figure 2: Image source: https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/recurrent_neural_networks

(d) Consider an RNN which outputs a single prediction at timestep T. As shown in Figure 2, each weight matrix W influences the loss by multiple paths. As a result, the gradient is also summed over multiple paths:

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

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?

(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.)

(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.**

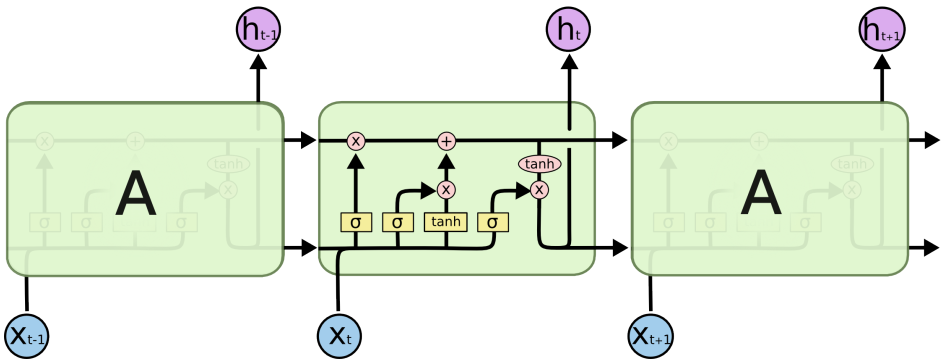


Figure 3: Image source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

The LSTM forward pass is shown below:

$$\begin{aligned}
f_t &= \sigma(x_t U^f + h_{t-1} W^f + b^f) \\
i_t &= \sigma(x_t U^i + h_{t-1} W^i + b^i) \\
o_t &= \sigma(x_t U^o + h_{t-1} W^o + b^o) \\
\tilde{C}_t &= \mathsf{tanh}(x_t U^g + h_{t-1} W^g + b^g) \\
C_t &= f_t \circ C_{t-1} + i_t \circ \tilde{C}_t \\
h_t &= \mathsf{tanh}(C_t) \circ o_t
\end{aligned}$$

where $\circ$ represents the Hadamard Product (elementwise multiplication) and σ is the sigmoid function.

(i) (Optional) When using an LSTM, you should still see vanishing gradients, but the gradients should vanish less quickly. **Interpret why this might happen by considering gradients of the loss with respect to the cell state.** (Hint: consider computing $\frac{\partial \mathcal{L}}{\partial C_{T-1}}$ using the terms $\partial \mathcal{L}, \partial C_T, \partial C_{T-1}, \partial h_T, \partial h_{T-1}$).

(j) (Optional)

Consider a ResNet with simple resblocks defined by $h_{t+1} = \sigma(W_t h_t + b_t) + h_t$. **Draw a connection between the role of a ResNet's skip connections and the LSTM's cell state in facilitating gradient propagation through the network.**

(k) (Optional) We can create multi-layer recurrent networks by stacking layers as shown in Figure 4. The hidden state outputs from one layer become the inputs to the layer above.

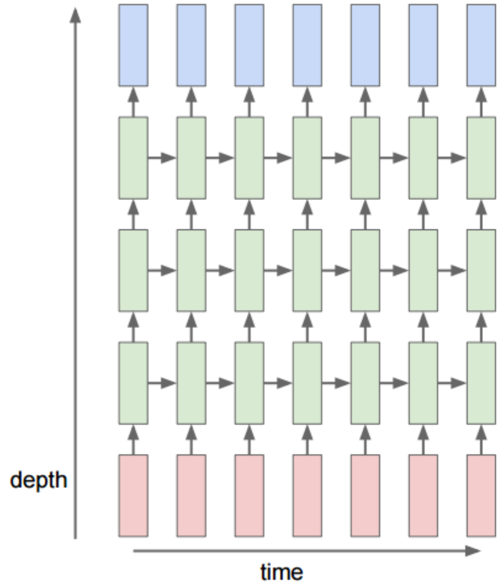


Figure 4: Image source: https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/recurrent_neural_networks

Implement notebook Section 1.K and run the last cell to train your network. You should be able to reach training loss < 0.001 for the 2-layer networks, and <.01 for the 1-layer networks.

2. 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.

(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**?

Contributors:

- Dhruv Shah.
- Olivia Watkins.
- Jerome Quenum.
- Anant Sahai.
- Anrui Gu.
- Matthew Lacayo.
- Past EECS 282 and 227 Staff.
- Linyuan Gong.

