CSE616 Neural Networks and Their Applications
Assignment 3 Submission

Ayman Wagih Mohsen (2000728)

May 27, 2022

1 Question 1

a. Predicted output with identity activation:

$$x_1 = 10, y_1 = 5$$

$$x_2 = 10, y_2 = 5$$

$$h_0 = 1$$

$$W_h = 1, W_x = 0.1, W_y = 2$$

$$h_1 = W_h h_0 + W_x x_1 = 1 * 1 + 0.1 * 10 = 2$$

$$\hat{y}_1 = W_y h_1 = 2 * 2 = 4$$

$$h_2 = W_h h_1 + W_x x_2 = 1 * 2 + 0.1 * 10 = 3$$

$$\hat{y}_2 = W_y h_2 = 2 * 3 = 6$$

b. The total loss:

$$L_t = \sum_i (\hat{y}_i - y_i)^2 = (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 = (5-4)^2 + (5-6)^2 = 2$$

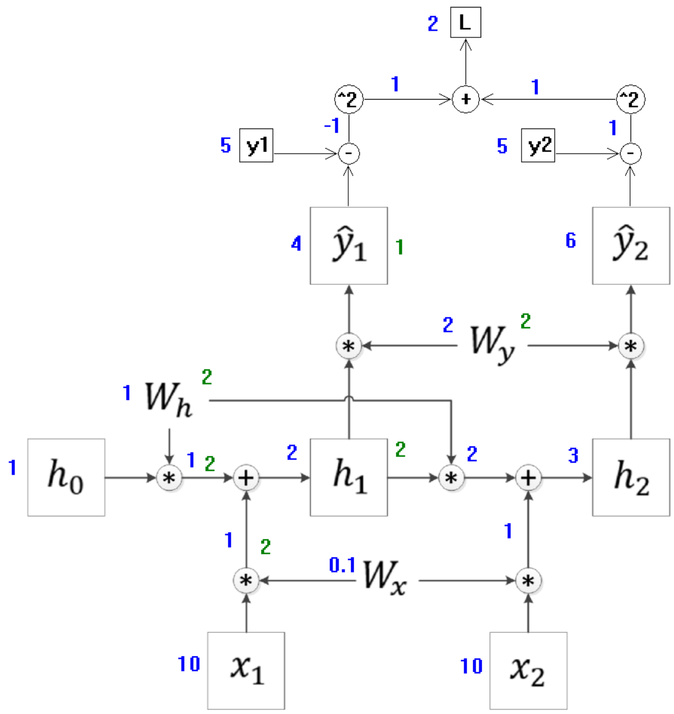
c. Derivative of loss w.r.t. h_1:

$$\begin{aligned}\frac{\partial L_t}{\partial h_1} &= 2(\hat{y}_1 - y_1)W_y + 2(\hat{y}_2 - y_2)W_y \frac{\partial h_2}{\partial h_1} \\ &= 2(\hat{y}_1 - y_1)W_y + 2(\hat{y}_2 - y_2)W_y W_h \\ &= 2(4-5) * 2 + 2(6-5) * 2 * 1 = 0\end{aligned}$$

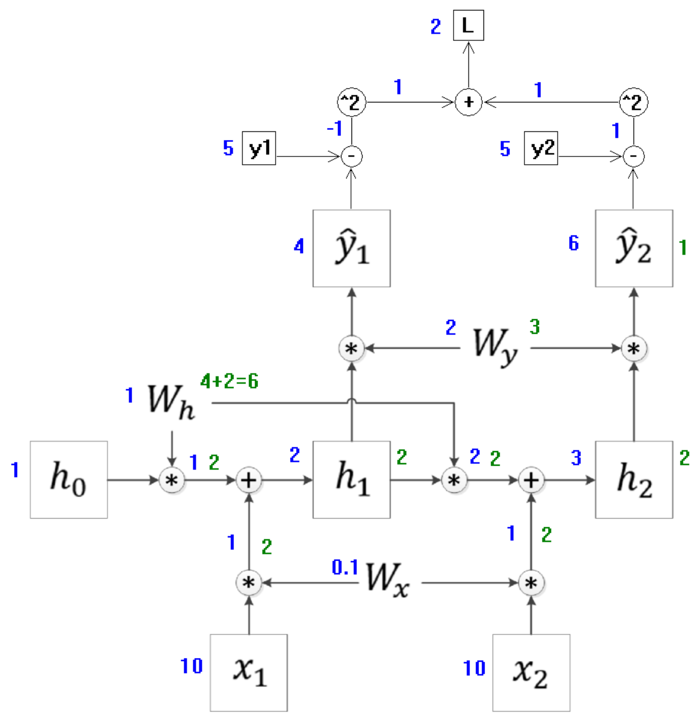
d. Derivative of loss w.r.t. W_h:

$$\begin{aligned}\frac{\partial L_t}{\partial W_h} &= 2(\hat{y}_1 - y_1)\frac{\partial \hat{y}_1}{\partial W_h} + 2(\hat{y}_2 - y_2)\frac{\partial \hat{y}_2}{\partial W_h} \\ &= 2(4-5) * 2 + 2(6-5) * 6 = 8\end{aligned}$$

The derivation of $\partial \hat{y}_1 / \partial W_h$ (blue is value, green is gradient):



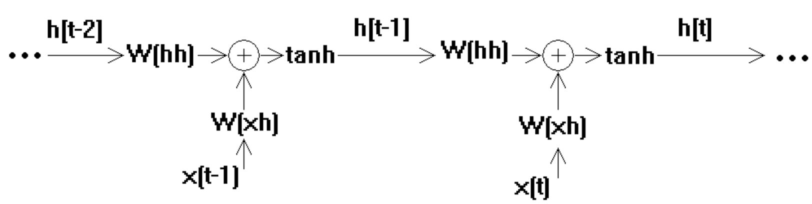
The derivation of $\partial \hat{y}_2 / \partial W_h$ (blue is value, green is gradient):



2 Question 2

With long term dependencies that appear with a long sequence, the gradient passes by many layers, and either explodes or vanishes.

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



The gradient becomes:

$$\frac{\partial J_t}{\partial W_{hh}} = \frac{\partial J_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\left(\sum_{k=0}^{t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W_{hh}}\right)$$

$$= \frac{\partial J_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\left(\left(\frac{\partial h_t}{\partial h_{t-1}}\cdots\frac{\partial h_0}{\partial W_{hh}}\right) + \left(\frac{\partial h_t}{\partial h_{t-1}}\cdots\frac{\partial h_1}{\partial W_{hh}}\right) + \cdots\frac{\partial h_t}{\partial W_{hh}}\right)$$

And a single factor $\partial h_t / \partial h_{t-1}$ is:

$$\frac{\partial h_t}{\partial h_{t-1}} = (1 - h_t^2)(W_{hh}) < 1$$

So multiplying many such factors causes the gradient to vanish.

3 Question 3

Gated Recurrent Units (GRUs) are better than vanilla RNNs with long sequences.

4 Question 4

The advantage of TBTT is fast gradient calculation. The distavantage is memory loss over long time intervals.

5 Question 5

a. That is because an RNN may learn the statistics of the English language and have more accurate decryption. Also an RNN is better suited for handling variable-length sequences.
However using an FC network we can think of Caesar cipher as a 1:1 mapping of independent characters. This way, no hidden state is necessary. And the model will be simpler.

b. The input and output are encoded as 26-dimensional vectors. Where each component represents the probability of the corresponding character.

c. The model should have a many-to-many architecture where each output character corresponds to that character in the input.

d. The training data should be many pairs of ciphertext and plaintext, without spaces. Such as: (khoorzruog, helloworld).

e. The Recurrent Neural Networks are designed to handle variable length sequences. Where each output character is the result of the previous state and current input character.

f. The tokenized characters (as numbers from 0 to 25) are then converted into vectors of 26 dimensions.

g. The model could be as shown in Algorithm 1 below.

h. The deciphered text characters are obtained using argmax of each vector.

Algorithm 1 RNN model with Keras

procedure MODEL_INITIALIZATION
    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras import layers
    $model = keras.Sequential()$
    $model.add(layers.Embedding(input\_dim = 26, output\_dim = 26))$
    $model.add(layers.SimpleRNN(128))$
end procedure

Algorithm 2 RNN model with Pytorch

procedure MODEL_INITIALIZATION
    import torch
    from torch import nn
    model=nn.RNN(sequence_length, hidden_dim, n_layers)
end procedure