Student Name: Mohammad Ahmad Khattab Mousa

Student Number: 2002639

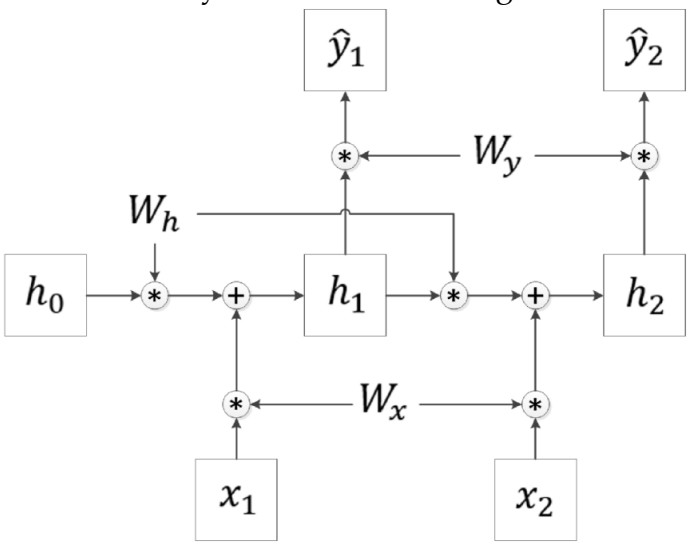
Course: CSE616: Neural Networks and Their Applications

Academic Year: Spring 2022

Affiliation: Ain Shams University

Solution to Assignment 3

1. (8 points) A RNN network is illustrated below (as a computational graph). Assume that the identity function is used to generate the output of all neurons.



Assume that the input at a given time, h0 = 1, x1 = 10, x2 = 10, y1 = 5, and y2 = 5. The initial values of the weights are: Wh = 1, Wx = 0.1, Wy = 2. Answer the following questions:

1. Compute the predicted value $\hat{y}2$.

$h_1 = h_0 * W_h + x_1 * W_x = 1 * 1 + 10 * 0.1 = 2$
$h_2 = h_1 * W_h + x_2 * W_x = 2 * 1 + 10 * 0.1 = 3$
$\hat{y}_2 = h_2 * W_y = 3 * 2 = 6$
$\boldsymbol{\hat{y}_2 = 6}$

2. Compute the total loss, Lt=Σ(yî−yi)2i for the given values of weights and inputs.

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

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

3. Compute the derivative $\partial Lt/\partial h1$.

$$\frac{\partial L_t}{\partial h_1} = \frac{\partial L_t}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial h_1} + \frac{\partial L_t}{\partial \hat{y}_2} * \frac{\partial \hat{y}_2}{\partial h_2} * \frac{\partial h_2}{\partial h_1}$$

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

$$\frac{\boldsymbol{\partial L_t}}{\boldsymbol{\partial h_1}} = \mathbf{0}$$

4. Compute the derivative $\partial L_t / \partial W_h$.

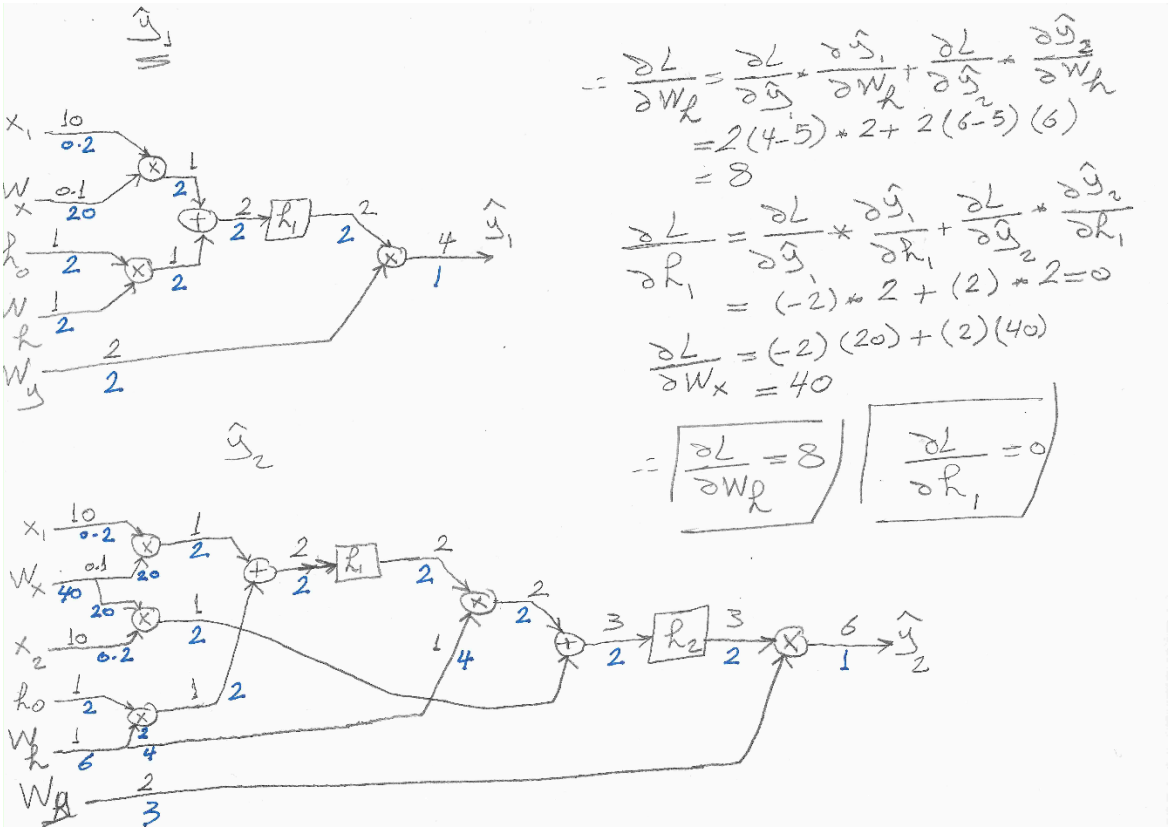
$$\frac{\partial L_t}{\partial W_h} = \frac{\partial L_t}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial h_1} * \frac{\partial h_1}{\partial W_h} + \frac{\partial L_t}{\partial \hat{y}_2} * \frac{\partial \hat{y}_2}{\partial h_2} * \frac{\partial h_2}{\partial W_h} + \boldsymbol{\frac{\partial L_t}{\partial \hat{y}_2} * \frac{\partial \hat{y}_2}{\partial h_2} * \frac{\partial h_2}{\partial h_1} * \frac{\partial h_1}{\partial W_h}}$$

$$\frac{\partial L_t}{\partial W_h} = 2 * (\hat{y}_1 - y_1) * W_y * h_0 + 2 * (\hat{y}_2 - y_2) * W_y * h_1 + 2 * (\hat{y}_2 - y_2) * W_y * W_h * h_0$$

$$\frac{\partial L_t}{\partial W_h} = 2(4-5) * 2 * 1 + 2(6-5) * 2 * 2 + 2(6-5) * 2 * 1 * 1$$

$$\frac{\partial L_t}{\partial W_h} = 8$$

Solution2: Computation using computational graph



2. Why are long term dependencies difficult to learn in a RNN? You may use this equation to explain your answer.

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

This is because of the vanishing gradient problem when making back propagation to old states.
For example, if we need to backpropagate the gradient down to W_{hh}, we will need to calculate the term $\sum_{k=0}^{t} \frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W_{hh}}$ *where* $h' < 1$ (due to first derivative of tanh < 1) and hence for long dependencies, we will be multiplying small numbers leading to vanishing gradient for long dependencies.

3. (2 points) When would the use of Gated Recurrent Units (GRU) be more efficient than vanilla RNNs?
If we have long-term dependencies and suffer from vanishing gradient and hence unable to keep track of these dependencies

4. (2 points) What are the advantage and disadvantage of Truncated Backpropagation Through Time (TBTT)?

According to: https://arxiv.org/abs/1705.08209
Tallec, Corentin, and Yann Ollivier. "Unbiasing truncated backpropagation through time." *arXiv preprint arXiv:1705.08209* (2017).

Advantage: Reduces the computational cost of the back propagation through time (BPTT) algorithm for gradient calculations in recurrent neural networks.

Disadvantage: truncation favors short-term dependencies and hence the gradient estimation of truncated BPTT is biased. Biased gradient estimation could lead to divergence as unbiasedness is the key property of to converge to a local minimum in a stochastic gradient decent procedure.

5. (8 points) You are required to define a simple RNN to decrypt a Caesar Cipher. A Caesar Cipher is a cipher that encodes sentences by replacing the letters by other letters shifted by a fixed size. For example, a Caesar Cipher with a left shift value of 3 will result in the following:
Input: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher: DEFGHIJKLMNOPQRSTUVWXYZABC
Notice that there is a 1-to-1 mapping for every character, where every input letter maps to the letter below it. Because of this property you can use a character-level RNN for this cipher, although word-level RNNs may be more common in practice.
Answer the following questions:
a. A Caesar Cipher can be solved as a multiclass classification problem using a fully-connected feedforward neural network since each letter X maps to its cipher value Y. However, an RNN will perform much better. Why?

As RNN could learn the dependency between letters within the same word. FC would ignore these dependencies and assume full weigh matrix (inputs are independent on each other) leading to over-parametrizing the model and overfitting.

b. Describe the nature of the input and output data of the proposed model.
Input: Sequence of the preprocessed cipher characters.
Output: Sequence of characters of the deciphered text.

c. Will the model by a character-level one-to-one, one-to-many, or many-to-many (matching) architecture? Justify your answer.

Many-to-Many architecture as we will receive a sequence of cipher characters as input and will output a sequence of deciphered characters as output.

d. How should the training data look like? Give an example of a sample input and the corresponding output.

Input: QHXUDO QHWZRUNV DUH DZHV RPH
Output: NEURAL NETWORKS ARE AWESOME

e. What is a good way to handle variable length texts?
Use padding to unify the size of the input sequence.

f. In order for the model to function properly, the input text has to go through several steps. For example, the first step is to tokenize is the text, i.e., to convert it into a series of characters. What should be the other required steps in order to train the model?

1. Tokenization: Breakdown the cipher text into a series of characters
2. Normalization: Convert small letters to capital letters
3. Embedding: Convert these characters into equivalent vector representation
4. Padding: Add trailing zeros to unify the length of the input sequence

g. What should be the architecture of the simple RNN that can be used? You might use Keras API to describe the architecture.
The RNN should have at least two layers:
1. SimplyRNN keras layer: which is a Fully-connected RNN where the output is to be fed back to input. Tanh activation function is used by default.
The layer is set to return the full output sequence as we have a sequence-to-sequence problem

https://www.tensorflow.org/api_docs/python/tf/keras/layers/SimpleRNN

2. Time Distributed Keras layer: It's a fully connected layer with softmax activation for classification (where the number of units should be equal to the number of classes). The layer is a wrapper around the fully connected layer where the same layer (with the same weights) is applied to each output out of the output sequence.
https://www.tensorflow.org/api_docs/python/tf/keras/layers/TimeDistributed

```
### model hyper parameters
sequence_size=64  #size of the input & output sequences
output_size=16      #size of the output vector (y)
numOfClasses =28  #26 letters + space + zero padding
lr = 0.001              #learning rate

###  layers definitions
rnnLayer = tf.keras.layers.SimpleRNN( units=output_size, return_sequences=True)
temporalClassificationHead = tf.keras.layers.TimeDistributed(layers.Dense( units= numOfClasses, activation='softmax'))

### build the model
input = tf.keras.Input(shape=(sequence_size,embedding_size))
x = rnnLayer(input)
output = temporalClassificationHead(x)
model = tf.keras.Model(input, output)

### compile & fit the model
optimizer = tf.optimizers.Adam(learning_rate=lr)
model.compile(optimizer=optimizer, loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
history = model.fit( train_ds, validation_data=val_ds,epochs=100)
```

h. Are there any processing operations that should be applied to the output in order to generate the deciphered text?

1. Apply a dense layer as a classification head with softmax activation. For example if we consider only capital letters and spaces, we will need 28 units to count for the 28 classes (26 letters + space character + zero padding classes)
2. Select the unit which has the largest softmax output (highest class probability)
3. Map the unit class back to the corresponding character
4. Concatenate all output characters as one sentence
5. Remove all trailing zero padding