Faculty of Engineering

Ain Shams University

2021-2022

Neural Networks and Their Applications

CSE 616

PG2015

Assignment #: 3

Title: ***RNN Architecture Usage and Training***

Made by:

2100966 حمدي أسامه محمد السيد فرج

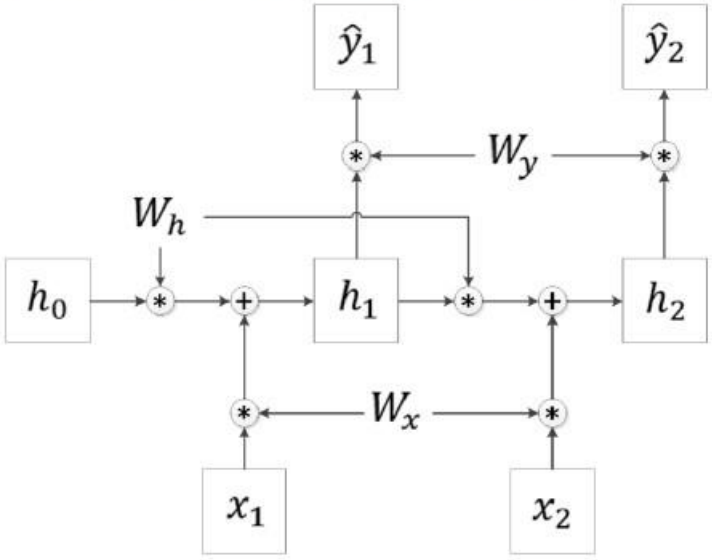
Mechatronics PG Student

Neural Networks – CSE 616

Assignment 3 Solution

Made by: Hamdy Osama (2100966)

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, $h_0 = 1$, $x_1 = 10$, $x_2 = 10$, $y_1 = 5$, and $y_2 = 5$. The initial values of the weights are: $W_h = 1$, $W_x = 0.1$, $W_y = 2$. Answer the following questions:

1. Compute the predicted value $\hat{y}_2$.
2. Compute the total loss, $L_t = \sum_i (\hat{y}_i - y_i)^2$ for the given values of weights and inputs.
3. Compute the derivative $\frac{\partial L_t}{\partial h_1}$.
4. Compute the derivative $\frac{\partial L_t}{\partial W_h}$.

Solution:

1- Solving the outputs of each block based on the given input in the question:

$$h_1 = W_h h_0 + W_x x_1 = (1 \times 1) + (0.1 \times 10) = 2$$
$$h_2 = W_h h_1 + W_x x_2 = (1 \times 2) + (0.1 \times 10) = 3$$
$$\hat{y}_1 = W_y h_1 = 2 \times 2 = 4$$
$$\hat{y}_2 = W_y h_2 = 2 \times 3 = 6$$

2- Solving for the loss of each prediction based on the expected true ground label:

$$L_t = \sum_{i=1}^{2} (\hat{y}_i - y_i)^2 = (\hat{y_1} - y_1)^2 + (\hat{y_2} - y_2)^2$$
$$L_t = (4 - 5)^2 + (6 - 5)^2 = 2$$

3- Computing the derivative $\frac{\partial L_t}{\partial h_1}$ using the chain rule: Note that h2 is dependent on h1, so we can't neglect the $\hat{y}_2$ during calculating the derivative needed in this question.

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

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

$$\frac{\partial L_t}{\partial h_1} = -4 + 4 = 0$$

4- Computing the derivative $\frac{\partial L_t}{\partial W_h}$ using the chain rule:

$$By\ using\ \frac{\partial L_t}{\partial W_h} = \sum_{k=0}^{t} \frac{\partial L_t}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial h_t} \times \frac{\partial h_t}{\partial h_k} \times \frac{\partial h_k}{\partial W_h}$$

$$\frac{\partial L_t}{\partial W_h} = \frac{\partial L_1}{\partial W_h} + \frac{\partial L_2}{\partial W_h} = \sum_{k=0}^{1} \frac{\partial L_1}{\partial \hat{y}_1} \times \frac{\partial \hat{y}_1}{\partial h_1} \times \frac{\partial h_1}{\partial h_k} \times \frac{\partial h_k}{\partial W_h} + \sum_{k=0}^{2} \frac{\partial L_2}{\partial \hat{y}_2} \times \frac{\partial \hat{y}_2}{\partial h_2} \times \frac{\partial h_2}{\partial h_k} \times \frac{\partial h_k}{\partial W_h}$$

$$\frac{\partial L_t}{\partial W_h} = \frac{\partial L_1}{\partial \hat{y}_1} \times \frac{\partial \hat{y}_1}{\partial h_1} \times \left(\frac{\partial h_1}{\partial W_h} + \frac{\partial h_1}{\partial h_0} \times \frac{\partial h_0}{\partial W_h}\right) + \frac{\partial L_2}{\partial \hat{y}_2} \times \frac{\partial \hat{y}_2}{\partial h_2} \times \left(\frac{\partial h_2}{\partial W_h} + \frac{\partial h_2}{\partial h_1} \times \frac{\partial h_1}{\partial W_h} + \frac{\partial h_2}{\partial h_0} \times \frac{\partial h_0}{\partial W_h}\right)$$

$$\frac{\partial L_t}{\partial W_h} = 2W_y(\hat{y}_1 - y_1) \times (h_0 + W_h \times 0) + 2W_y(\hat{y}_2 - y_2) \times \left(h_1 + W_h h_0 + W_h^2 \times 0\right)$$

$$\frac{\partial L_t}{\partial W_h} = 2 \times 2 \times (4-5) \times (1+0) + 2 \times 2 \times (6-5) \times (2 + 1 \times 1 + 0)$$

$$\frac{\partial L_t}{\partial W_h} = -4 + 12 = 8$$

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

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

Solution:

Long term dependencies are difficult to train because of the vanishing gradient phenomenon. In this phenomenon, the gradient of weights as time sequence gets deeper gets closer to zero because of the multiplication of multiple numbers less than one. This causes the training of the network to saturate as dependencies get longer than 10-time samples. We can use the given equation to explain our answer mathematically:

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

$$\frac{\delta h_t}{\delta h_{t-1}} = [1 - tanh^2(W_{hh}h_{t-1}, W_{xh}x_t)] \times W_{hh}$$

$$If\ W_{hh}\ is\ smaller\ than\ one, the\ usage\ of\ more\ iterations\ of\ the\ derivative\ to\ compute\ gradient\ at\ deeper\ time\ samples\ will\ cause\ the\ gradient\ to\ reach\ zero\ as$$

$$[1 - tanh^2(W_{hh}h_{t-1}, W_{xh}x_t)] \leq 1$$

3. (2 points) When would the use of Gated Recurrent Units (GRU) be more efficient than vanilla RNNs?

Solution:

In the GRU architecture, each cell computes a reset gate based on current input and hidden state. Next, it computes the hidden state based on current input and hidden state. If reset gate unit is ~0, then this ignores previous memory and only stores the new input information. Finally, it computes an update gate again based on current input and hidden state and combine current and previous timestep output at the final memory. GRU uses gates to control information flow, so the hidden state is not required to undergo transformation for each time step, which reduces the vanishing gradient effect and enables longer dependencies than conventional RNN.

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

Solution:

Advantage: Faster parameter updates because instead of propagating to the beginning of the sequence you only propagate backwards k steps.

Disadvantage: Not able to capture longer dependencies than the truncated length as this is the main criteria upon which the algorithm is built, with a maximum backpropagation sequence, so long dependencies will be lost.

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?

   b. Describe the nature of the input and output data of the proposed model.

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

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

   e. What is a good way to handle variable length texts?

   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?

   g. What should be the architecture of the simple RNN that can be used? You might use Keras API to describe the architecture.

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

Solution:

a) An RNN will perform better than a conventional feedforward fully connected neural network because RNN can learn the sequence of characters composing a word. This is done because of the ability of RNNs to deal with temporal dependencies of coming input sequence and can determine which sequence of characters is more likely to come after the current input.

b) The input data for this model would be normal text, like sentences and words that can be read by humans easily. The output would consist of the same input sentence length, but

all letters are shifted by 3. For example, the word “data” as input to the model will come out as “gdwd”. Input shape can be a sentence of 100 characters, including spaces and punctuation marks. The output would be a sentence with 100 characters that are shifted. Input shape = (1*100*number of training examples in batch).

c) This model will be “one to one architecture” because the output sentence will have the same length as the input sentence; in fact, each word in the input sentence corresponds to a group of letters in the output stream of the same number of letters as the input word. This can be a drawback for this ciphering technique as it can be deciphered by trial and error as we try shifting characters (only 26 iterations are needed to check if sensible words are made from the deciphering of the encoded message).

d) Training data would have a normal sentence and its corresponding Caesar Cipher that is the expected output. Example -> “I Like Apples” corresponds to “L Olnh Dssohv”

e) A good way to handle sentences of variable length in characters would be to introduce padding to the end of the sentence. For example, all sentences shorter than 100 characters would have “0” as the padding value till it reaches 100 characters. Try making the padding value different from the character IDs you have that transform characters to numbers so that you don’t bias the data to favor the sequence of padding values.

f,h) After tokenization of input (the process of splitting words into characters and replacing each character with its corresponding ID number), padding will occur to let all training samples of the same length. After that, training of the model will begin and the output of the model will be used in backpropagation during training. A num_to_char function (converts character ID into character) should be used to process the output of the model to get the deciphered text.

g) A simple model with one RNN hidden layer containing 64 nodes is illustrated here. Weights of the hidden layer of the RNN are applied to the whole sequence of input characters because of the “TimeDistributed” function.

```
def simple_rnn(input_shape, learning_rate):
    input_sequence = Input(input_shape)
    rnn = SimpleRNN(64, activation='relu', return_sequences=True)(input_sequence)
    z = TimeDistributed(Dense(plaintext_vocab_size))(rnn)
    out = Activation('softmax')(z)
    model = Model(inputs=input_sequence, outputs=out)
    model.compile(loss=sparse_categorical_crossentropy,
                  optimizer=Adam(learning_rate),
                  metrics=['accuracy'])
    return model
```