THE HONG KONG UNIVERSITY OF SCIENCE & TECHNOLOGY

Machine Learning

Homework 3 Solutions

Due Date: See course website

Your answers should be typed, not handwritten. You can submit a Word file or a pdf file. Submissions are to be made via Canvas. Note that penalty applies if your similarity score exceeds 40. To minimize your similarity score, don't copy the questions.

Question 1: Consider the image X and filter F given below. Let X be convolved with F using no padding and a stride of 1 to produce an output Y. Assume the bias is 2 and the activation function is ReLU. What are values of the cells a, b, e and f in the output Y?

$X =$

1	0	-2	3	4	1
2	9	5	6	0	-1
0	-3	1	3	4	4
6	5	2	0	6	8
-5	4	-3	1	3	-2
4	1	2	8	9	7

$F =$

-1	-1	-1
-1	8	-1
-1	-1	-1

$Y =$

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

Solution: TA: Please work out the answers

Question 2: The input of a convolutional layer has shape $27 \times 27 \times 256$ (width, height, depth). The layer uses 384 3×3 filters applied at stride 1 with no zero padding. What is the shape of the output of the layer? How many parameters are there? How many float multiplication operations it will take to compute the net inputs of the all the output units?

Solution: The shape of the output layer is $W_2 \times H_2 \times D_2$ where

$$\begin{aligned} W_2 &= (W_1 - F + 2P)/S + 1 = (27 - 3 + 0)/1 + 1 = 25 \\ H_2 &= (H_1 - F + 2P)/S + 1 = (27 - 3 + 0)/1 + 1 = 25 \\ D_2 &= 384. \end{aligned}$$

The number of parameters is

$$(FFD_1 + 1)K = (3 \times 3 \times 256 + 1)384 = 885,120.$$

The number of float multiplication ops it takes to compute the net input of each output unit is FFD_1. The number of output units is $W_2H_2D_2$. Hence, the total number of ops is:

$$(3 \times 3 \times 256) \times (25 \times 25 \times 384) = 552,960,000$$

Question 3: What is batch normalization? What is layer normalization? What are their pros and cons?

Solution: Batch normalization is a technique to stabilize the learning process of deep neural network. To be specific, each batch of data could possess different distribution which causes "internal covariate shift". Batch normalization normalizes each batch of data in order to avoid the problem by using scaling factor and shifting factor, and in turn, results in stabilized learning process as well as faster convergence. However, there are some cons in this technique. First is the dependency on the size of batch. Since batch normalization is manipulating data of each batch, the data in batch decides the

value of scaling and shifting factor. Therefore, careful choice of the batch size is crucial, and of course, when batch size is 1, batch normalization cannot be applied. In addition, at training phase, the scaling and shifting factor have to be identified and kept, since the values will be used in testing phase as well.

Batch normalization is not suitable for RNN because it destroys sequential dependencies, which are important for NLP. It also requires a lot of memory as statistics for each time step have to be computed and stored. Furthermore, if given a longer sentence in testing than in training data, it is not possible to apply batch normalization considering the absent of statistics.

Layer normalization is related to batch normalization, but different in that it normalizes the inputs across features. Since, layer normalization normalizes inputs across features, any number of batch size could be used. In addition, past experiments show promising results on Recurrent Neural Network models with layer normalization.

Question 4 In an LSMT cell, $\mathbf{h}^{(t)}$ is computed from $\mathbf{h}^{(t-1)}$ and $\mathbf{x}^{(t)}$ using the following formulae:

$$\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f\mathbf{x}^{(t)} + \mathbf{U}_f\mathbf{h}^{(t-1)} + \mathbf{b}_f) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i\mathbf{x}^{(t)} + \mathbf{U}_i\mathbf{h}^{(t-1)} + \mathbf{b}_i) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_q\mathbf{x}^{(t)} + \mathbf{U}_q\mathbf{h}^{(t-1)} + \mathbf{b}_q) \\
\mathbf{c}_t &= \mathbf{f}_t \otimes \mathbf{c}_{t-1} + \mathbf{i}_t \otimes \tanh(\mathbf{U}\mathbf{x}^{(t)} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{b}) \\
\mathbf{h}^{(t)} &= \mathbf{o}_t \otimes \tanh(\mathbf{c}_t)
\end{aligned}$$

(a) Intuitively, what are the functions of the forget gate $\mathbf{f}_t$ and the input gate $\mathbf{i}_t$ do? Answer briefly.

(b) Why do we use the sigmoid function for $\mathbf{f}_t$ and $\mathbf{i}_t$, but tanh for the memory cell $\mathbf{c}_t$ and the output $\mathbf{h}^{(t)}$? Answer briefly.

Solution: (a) The forget gate $\mathbf{f}_t$ determines which components of the previous state $\mathbf{c}_{t-1}$ and how much of them to remember/forget. The input gate $\mathbf{i}_t$ determines which components of the input from $\mathbf{h}^{(t-1)}$ and $\mathbf{x}^{(t)}$ and how much of them should go into the current state.

(b) The sigmoid function is used for $\mathbf{f}_t$ and $\mathbf{i}_t$ so that their values are likely to be close to 0 or 1, and hence mimicking the close and open of gates. The tanh function is used for $\mathbf{c}_t$ and $\mathbf{h}^{(t)}$ so that strong gradient signals can be backpropagated from $\mathbf{h}^{(t)}$ to $\mathbf{c}_t$, and then to $\mathbf{h}^{(t-1)}$.

Question 5 Consider a self-attention layer with the following input token embeddings:

x_1: 0.0, 1.0
x_2: 1.0, 0.0
x_3: 0.5, 0.5
x_4: 0.8, 0.2
x_5: 0.2, 0.8

Let the key matrix W^K and the value matrix W^V both be the identity matrix.

(a) Suppose the query matrix is $W_1^Q = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. What are the attention weights when calculating the output token embeddings $z_1, \ldots, z_5$ of the self-attention layer? What are the output token embeddings.

(b) Suppose the query matrix is $W_2^Q = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. What are the attention weights when calculating the output token embeddings $z_1, \ldots, z_5$ of the self-attention layer? What are the output token embeddings.

(c) In Part (a), what would be the attention weights if the input taken embeddings were:

x_1: 0.8, 0.2
x_2: 0.2, 0.8
x_3: 0.0, 1.0
x_4: 1.0, 0.0
x_5: 0.5, 0.5

Note how the attention matrix influences the output (a vs b), and how the attention weights changes with respect to input (a vs c).

Solution: **(a)** Since W_Q, W_K, W_V are all identity matrices, we have

$$Q = K = V = \begin{bmatrix} 0.0 & 1.0 \\ 1.0 & 0.0 \\ 0.5 & 0.5 \\ 0.8 & 0.2 \\ 0.2 & 0.8 \end{bmatrix}. \tag{1}$$

Then,

$$\frac{QK^\top}{\sqrt{d_k}} = \begin{bmatrix} 0.7071 & 0. & 0.3536 & 0.1414 & 0.5657 \\ 0. & 0.7071 & 0.3536 & 0.5657 & 0.1414 \\ 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \\ 0.1414 & 0.5657 & 0.3536 & 0.4808 & 0.2263 \\ 0.5657 & 0.1414 & 0.3536 & 0.2263 & 0.4808 \end{bmatrix} \tag{2}$$

and thus the attention weights are

$$\text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) = \begin{bmatrix} 0.2754 & 0.1358 & 0.1934 & 0.1564 & 0.2391 \\ 0.1358 & 0.2754 & 0.1934 & 0.2391 & 0.1564 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.1598 & 0.2443 & 0.1976 & 0.2244 & 0.174 \\ 0.2443 & 0.1598 & 0.1976 & 0.174 & 0.2244 \end{bmatrix}. \tag{3}$$

Finally, the output token embeddings are

$$\text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V = \begin{bmatrix} 0.4054 & 0.5946 \\ 0.5946 & 0.4054 \\ 0.5 & 0.5 \\ 0.5574 & 0.4426 \\ 0.4426 & 0.5574 \end{bmatrix}. \tag{4}$$

Note that $\mathbf{z_1 = 0.28x_1 + 0.16x_2 + 0.19x_3 + 0.15x_4 + 0.24x_5}$, x_1 and x_5 receive the most attentions.

(b) First, we have

$$Q = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \\ 0.5 & 0.5 \\ 0.2 & 0.8 \\ 0.8 & 0.2 \end{bmatrix}, \qquad K = V = \begin{bmatrix} 0.0 & 1.0 \\ 1.0 & 0.0 \\ 0.5 & 0.5 \\ 0.8 & 0.2 \\ 0.2 & 0.8 \end{bmatrix}. \tag{5}$$

Then,

$$\frac{QK^\top}{\sqrt{d_k}} = \begin{bmatrix} 0. & 0.7071 & 0.3536 & 0.5657 & 0.1414 \\ 0.7071 & 0. & 0.3536 & 0.1414 & 0.5657 \\ 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \\ 0.5657 & 0.1414 & 0.3536 & 0.2263 & 0.4808 \\ 0.1414 & 0.5657 & 0.3536 & 0.4808 & 0.2263 \end{bmatrix} \tag{6}$$

and thus the attention weights are

$$\text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) = \begin{bmatrix} 0.1358 & 0.2754 & 0.1934 & 0.2391 & 0.1564 \\ 0.2754 & 0.1358 & 0.1934 & 0.1564 & 0.2391 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2443 & 0.1598 & 0.1976 & 0.174 & 0.2244 \\ 0.1598 & 0.2443 & 0.1976 & 0.2244 & 0.174 \end{bmatrix}. \tag{7}$$

Finally, the output token embeddings are

$$\text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V = \begin{bmatrix} 0.5946 & 0.4054 \\ 0.4054 & 0.5946 \\ 0.5 & 0.5 \\ 0.4426 & 0.5574 \\ 0.5574 & 0.4426 \end{bmatrix}. \tag{8}$$

Note that $\mathbf{z_1 = 0.16x_1 + 0.28x_2 + 0.19x_3 + 0.24x_4 + 0.16x_5}$, x_2 and x_4 receive the most attentions. (c) Since W_Q, W_K, W_V are all identity matrices, we have

$$Q = K = V = \begin{bmatrix} 0.8 & 0.2 \\ 0.2 & 0.8 \\ 0.0 & 1.0 \\ 1.0 & 0.0 \\ 0.5 & 0.5 \end{bmatrix}. \tag{9}$$

Then,

$$\frac{QK^\top}{\sqrt{d_k}} = \begin{bmatrix} 0.4808 & 0.2263 & 0.1414 & 0.5657 & 0.3536 \\ 0.2263 & 0.4808 & 0.5657 & 0.1414 & 0.3536 \\ 0.1414 & 0.5657 & 0.7071 & 0. & 0.3536 \\ 0.5657 & 0.1414 & 0. & 0.7071 & 0.3536 \\ 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \end{bmatrix} \tag{10}$$

and thus the attention weights are

$$\text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) = \begin{bmatrix} 0.2244 & 0.174 & 0.1598 & 0.2443 & 0.1976 \\ 0.174 & 0.2244 & 0.2443 & 0.1598 & 0.1976 \\ 0.1564 & 0.2391 & 0.2754 & 0.1358 & 0.1934 \\ 0.2391 & 0.1564 & 0.1358 & 0.2754 & 0.1934 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \end{bmatrix}. \tag{11}$$

Note that $\mathbf{z_1 = 0.22x_1 + 0.17x_2 + 0.16x_3 + 0.24x_4 + 0.20x_5}$. The attentions different from (a) even the query, key and value matrices are the same. The demonstrate the concept of dynamic weights.

Question 6 BERT input representation is the sum of token embedding, segment embedding, and positional embedding. Briefly explain why positional embedding is introduced in the architecture.

Solution: In BERT, unlike AutoRegressive approach, input tokens are fed into the architecture at once. Without positional embedding, the input representation of a token holds no information about its position in a sentence, in other words, each token has only one input representation (assuming same segment embedding). By adding positional embedding to input representation, each token can have different representations according to its position, and holds positional information.

[No more questions for WA3]