Karan Vora kv2154
ECE-GY 7123 Introduction To Deep-Learning Assignment 3

Solution 1):

1.a):

The special property of CNN’s is its ability to preserve spatial relationship between pixels by learning features from smaller sections of the input. The filter sizes can be changed to perform operations like edge detection or sharpening which aids in learning features. Having multiple perceptrons and filters allow CNN’s to recognize more features which each passing layer, this makes them the front runners for image classification. Another point to note is, CNN’s take in fixed size inputs to generate fixed size outputs, hence able to classify images faster and more accurately than a RNN.

1.b):

RNN’s are sequence to sequences map, this allows them to maintain class probabilities and update them as more input is fed into the network. RNN’s have the ability to classify images when only a part of the image is given as an input, or when all pixels of the image are not presented. CNN lack this ability as they must be always be given the whole input rather than only a part of it. RNN’s work with sequential data, thus they can recognize actions of objects better than CNN’s. If given a sequence of images of a car moving through the road, a RNN would be able to predict that the car is moving as it is better at processing dependencies and temporal data.

Solution 2):

For $t=1$
$h_1=x_1-h_0$
$y_1=sigmoid(1000\,h_1)$

For $t=2$
$h_2=x_2-h_1$
$y_2=sigmoid(1000\,h_2)$
$y_2=sigmoid(1000\,(x_2-h_1))$
$y_2=sigmoid(1000\,(x_2-x_1+h_0))$

For $t=3$
$h_3=x_3-h_2$
$y_3=sigmoid(1000\,h_3)$

For $t=4$
$h_4=x_4-h_3$
$y_4=sigmoid\,(1000\,h_4)$
$y_4=sigmoid\,(1000(x_4-h_3))$
$y_4=sigmoid\,(1000(x_4-x_3+x_2-x_1+h_0))$

Since the input sequence is of the even length t = 2n, then we can generalize the equation to:-

$y_{2n}=sigmoid((x_{2n}-x_{2n-1}+x_{2n-2}-.....+x_0-x_1+h_0))$

Solution 3):

3.a):

The norm of the vector x_1 is $\sqrt{(2)}*\beta$ and the norms of vector x_2 and x_3 are β.

3.b):

To compute Self-Attention, we need to compute the dot products of each query with all keys, normalize the scores using softmax, and use the resulting weights to compute a weighted sum of the values. Since in this case, the queries, keys, and values are the same as the data points themselves, we can directly compute the dot products between each pair of tokens:

$q_1=x_1, k_1=x_1, v_1=x_1$
$q_2=x_2, k_2=x_2, v_2=x_2$
$q_3=x_3, k_3=x_3, v_3=x_3$

The dot products between each pair of tokens are,

$x_1\cdot x_1=6\beta^2, x_1\cdot x_2=0, x_1\cdot x_3=2\beta^2$
$x_2\cdot x_2=4\beta^2, x_2\cdot x_3=0$
$x_3\cdot x_3=6\beta^2$

After normalizing the scores using softmax, we obtain the following weights:

$w_1=softmax([6,0,2])=[0.8808, 0.0180, 0.1012]$
$w_2=softmax([0,4,0])=[0.0180, 0.9641, 0.0180]$
$w_3=softmax([2,0,6])=[0.1012, 0.0180, 0.8808]$

Finally, we compute the weighted sum of the values:

$y_1=w_1[0]*v_1+w_1[2]*v_2+w_1[2]*v_3=0.8808(d+b)+0.1012(c+b)$
$y_2=w_2[0]*v_1+w_2[1]*v_2+w_3[2]*v_3=0.0180a+0.9641a+0.0180a=a$
$y_3=w_3[0]*v_1+w_3[1]*v_2+w_3[2]*v_3=0.1012(d+b)+0.8808(c+b)$

We can see that y_2 is an exact copy of x_2, while y_1 and y_3 are combination of x_1, x_2 and x_3.

3.c):

Self-attention allows the network to "copy" an input value to the output by assigning a high weight to the value itself as a key and query, and normalizing the scores using softmax. This way, the value is effectively copied to the output vector, with the weight indicating the degree of importance of that value. Moreover, since the attention mechanism is applied to all tokens in parallel, the network can select different parts of the input to be copied to the output depending on the context and the task at hand.