Deep Learning HW2

Yipu Li

November 2024

1 1. Properties of CNN

(a) $f(1,1) = g(1,1)h(0,0) + g(1,2)h(0,1) + g(2,1)h(1,0) + g(2,2)h(1,1)$

(b) i. The boarder pixels are missing and we can use padding to solve the problem.

	validation score	test score
valid	100	0.09
replicate	98.32	93.58
reflect	100	0
circular	99.59	83.07
sconv	98.96	6.51
fconv	88.09	89.06

ii. pictures with label 0 has red box left of the green one, and label 1 has green box left of the red one. For train set, the boxes in label 0 has boxes on the upper half of the picture, while label 1 has boxes on the lower half. The converse holds for the test set.

iii.

(c)i. There are pad type, padding size that affects the convolution type. They differ in padding size.

ii. Their pad size follow the order valid ¡ sconv ¡ fconv, and the filter is of the size 3×3. To maintain the output image size, we need 2 padding, hence fconv performs the best. sconv pads for 1, and hence it out performs valid which does not pad.

iii. The feature that we are trying to learn is the right-or-left of the red box and green box. In intermediate layers, reflect padding may distort the relative right-or-left positional information and lead to worse outcome. For instance, padding may reflect feature standing for red to the right of green when the initial image has the red box left of green one. Then the learning would be distorted by the reflect padding.

iv. In intermediate layers, replicate padding may enhance the relative right-or-left positional information and lead to better outcome. For instance, padding may copy feature standing for red to the right of green when the initial image has the red box right of green one. Then the learning would be enhanced by the replicate padding.

(d)i.

	validation score	test score
valid	100	0.00
replicate	100	0
reflect	100	0
circular	100	0
sconv	100	0
fconv	100	0

ii. It decreases.

iii. Because a max pooling layer in net 1 is changed to a linear layer in net 2. The linear layer fail to accurately ratain the most important information.

2 Attention, Transformers, and LLMs

Question 2.1

(a) The complexity of transformer scales quadratically with the number of inputs, hence long sequence is highly computationally costly for transformers. Moreover, the transformer requires more memeory of th GPU as the input scales, making it more demanding for GPUs.

We can change the multi-headed attention in the transformer, to make it work on part of the tokens (such as neighbouring tokens), instead of attending to all tokens.

(b) The receptive field of CNN is local, and it grows as you iterate CNN layers. The receptive field of CNN is global, it receives information from the whole data, however far away they are.

CNN is bad at capturing long range dependency as it suffers from vanishing gradient for tokens that are far away. Transformers are relatively good at it as the attention part in it allows model assign different weight to all relevant inputs based on relevancy, thus the model is able to focus on inputs that are far away into considerarion.

(c) It is used to scale down the Query-Key Similarity to maintain relatively equal variance throughout the model. Without it, the variance of Query-Key Similarity is d_k times greater than expected, leading the softmax step to over-represent one value and under-represent the others.

(d) Multiple attention heads splits the Queries, Keys and Values into different sub-queries, sub-keys and sub-values via a linear transformation. They experience their own optimization and thus the Multiple attention heads are better able to capture different features and aspects of the input.

Question 2.3

(c) The MLP is focused on operating independently on each token, rather than operating on the connection of different tokens. A wide but not deep MLP does exactly that.

Question 2.4

(a) The multi head attention module takes input tokens symmetrically, hence the positional input in a textual message is lost if we don't add positional embedding. For instance, the model would not be able to make a distinction between 'I am working' and 'am I working'.

(b) Absolute positional embedding overlooks the relative positional relation between words, for them, the relation between the first word and the second word is the similar to the relation between the first word and the last word. On the other hand, positional embedding are better at capturing the relative positional information between words.

Question 2.9

(b) It speeds up the computation of attention and saves memory space. It stems from consideration of something called Input-Output-aware, which I don't understand precisely. But essentially it has something to do with the memory structure of GPU, hence it is safe to say that flash attention is inspired by attempts to exploiting the memory structure of GPU in order to maximize efficiency.

Question 3.1

(a) $[[0,1,1,0,1],[1,0,1,0,0],[1,1,0,1,0],[0,0,1,0,0],[1,0,0,0,0]]$

$[[0,1,0,0,0,0],[1,0,1,0,1,0],[0,1,0,1,0,0],[0,0,1,0,1,1],[0,1,0,1,0,1],[0,0,0,1,1,0]]$

(b) $[[3,1,1,1,0],[1,2,1,1,1],[1,1,3,0,1],[1,1,0,1,0],[0,1,1,0,1]]$

the i,j index of A^2 represents the number of common adjacents of two nodes i and j, or the number of two step walks between i,j

the i,j index of A^n represents the number of n step walks between i,j.

(c) $[[1,-1,0,0,0,0],[-1,3,-1,0,-1,0],[0,-1,2,-1,0,0],$
$[0,0,-1,3,-1,-1],[0,-1,0,-1,3,-1],[0,0,0,-1,-1,2]]$

Generally speaking, each node changes relatively equally. That's because each vector of the Laplacian matrix sums up to zero, the impact of number of adjacent nodes on the update is evened out by the diagonal, where the number of adjacent nodes is stored.

But for specific input, the node with a relatively great abolute valued input updates the most.

3.2

(a) Without the term, the information from the node itself is not taken in the update. And with the information is taken into consideration.

(b) 6 can be somehow seen as a special case of 7. We can take $message(h_v^l, h_u^l, e_{uv})$ as $\frac{h_u^l}{|N(v)|}$ and $update(h_v^l, m_v^{l+1}) = \sigma(W^l \sum_{u \in N(v)} message(h_v^l, h_u^l, e_{uv}) + B^l h_v^l)$. Admittedly, here the message function is not linear, but this perspective makes 6 a special case of 7.

3.3

(a)

$$H^{l+1} = \sigma(D^{-1} H^l A {W^l}^T + H^l {B^l}^T)$$

(b) No, applying the median aggregation function, we have to order the feature vectors from adjacent nodes and pick the median per dimension. It is impossible to write it in matrix form as we are sorting the vectors per dimension.

3.4

(a) Instead of aggregating the feature from adjacent nodes with equal weight, the features from adjacent nodes are weighted with attention in GAN.

(b) The GAN may decrease its attention to the node, thus the feature of the node contributes less to the eventual output. While for GNN the weight it assigns to each node is fixed, thus it is more likely to be affected by these nodes. Hence the overall quality of GAN might be better than GNN if there are such nodes.

(c) We can view a transformer as a complete graph with weighted 'soft' adjacent matrix. We can view tokens of words as nodes and every tokens are connected. Then we compute the weight matrix, it represents the 'soft' or 'weighted' adjacency matrix between edges.

(d) 1. For tasks involving graph structures, the graph structure is built in GAT and thus it better respects the graphical structure of the problem. 2. The GAT is more scalable as in real world applications, each nodes are often not linked to that many adjacent, and thus the parametres would not scale that quickly as the nodes we are considering scales. This is in contrast with transformers where parametres scale quadratically with input size.