HW 4

Garrison Neel
CSCE 636

November 19, 2020

I Answers to the non-Programming Part

1 RNN

(a) Provide the size of each training parameter

L	$\|V\| \times d$
H	$D \times D$
I	$d \times D$
U	$D \times \|V\|$
b_1	$1 \times D$
b_2	$1 \times \|V\|$

(b) Compute gradients of Cross Entropy

(c) Show the relationship between cross-entropy and perplexity

Because $y^{(t)}$ is one-hot encoded, only one term in the sum will have any value. Suppose $y_z^{(t)}$ is the nonzero element in $y^{(t)}$. This means the z^{th} term of the sum is the only nonzero term. Perplexity can be rewritten as

$$PP(y^{(t)}, \hat{y}^{(t)}) = \frac{1}{\sum_{j=1}^{|V|} y_j^{(t)} \hat{y}_j^{(t)}} = \frac{1}{\hat{y}_z^{(t)}}$$

The same simplification can be done to the sum in cross entropy:

$$CE(y^{(t)}, \hat{y}^{(t)}) = -\sum_{j=1}^{|V|} y_j^{(t)} \log(\hat{y}_j^{(t)}) = -\log(\hat{y}_z^{(t)}) = \log(\frac{1}{\hat{y}_z^{(t)}})$$

From this, the relationship between perplexity and cross entropy can be seen to be

$$CE(y^{(t)}, \hat{y}^{(t)}) = \log(PP(y^{(t)}, \hat{y}^{(t)}))$$

2 Attention

(a) Compute and compare the number of parameters between the single-head and multi-head attention

Single head:

$$A = \text{softmax}(QW^Q(KW^K)^T)VW^V$$

W^Q, W^K, and W^V each are $d \times d$ matrices, meaning single-head attention has $3d^2$ trainable parameters.

Multi-head:

$$A = \text{concat}(\text{softmax}(QW_i^Q(KW_i^K)^T)VW_i^V, i = 1, ..., h)$$

Multi-head attention uses smaller $d \times \frac{d}{h}$ matrices. Each head has $3\frac{d^2}{h}$ parameters, and since there are h of them, there are $3d^2$ parameters in total. Same as single-head.

(b) Compute and compare the amount of computation between the single-head and multi-head attention

Note: Matrix multiplication of an $m \times n$ matrix with an $n \times d$ matrix has time complexity $\mathcal{O}(mnd)$

Single-head:

Algorithm 1: Single-head attention

```
$\tilde{Q} = QW^Q$ // $n \times d \cdot d \times d \to \mathcal{O}(nd^2)$
$\tilde{K} = KW^K$ // $n \times d \cdot d \times d \to \mathcal{O}(nd^2)$
$\tilde{V} = VW^V$ // $n \times d \cdot d \times d \to \mathcal{O}(nd^2)$
$S = \tilde{Q}\tilde{K}^T$ // $n \times d \cdot d \times n \to \mathcal{O}(n^2 d)$
for $i \leftarrow 1$ to $n$ do // softmax                    // $\mathcal{O}(n^2)$
    $\Sigma =$ sum row $i$ of $S$ // $\mathcal{O}(n)$
    for $j \leftarrow 1$ to $n$ do                            // $\mathcal{O}(n)$
        $A_{i,j} = \frac{exp(S_{i,j})}{\Sigma}$
    end
end
Attention $= A\tilde{V}$ // $n \times n \cdot n \times d \to \mathcal{O}(n^2 d)$
```

The time complexity of matrix multiplication dominates the single-head attention algorithm, making the time complexity $\mathcal{O}(n^2d)$. Were d to be greater than n for some reason, the time complexity could become $\mathcal{O}(nd^2)$, but I would assume typically $d << n$.

Algorithm 2: Multi-head attention

```
for $i \leftarrow 1$ to $h$ do                                  // $\mathcal{O}(n^2 d)$
    $\tilde{Q} = QW_i^Q$ // $n \times d \cdot d \times \frac{d}{h} \to \mathcal{O}(nd\frac{d}{h})$
    $\tilde{K} = KW_i^K$ // $n \times d \cdot d \times \frac{d}{h} \to \mathcal{O}(nd\frac{d}{h})$
    $\tilde{V} = VW_i^V$ // $n \times d \cdot d \times \frac{d}{h} \to \mathcal{O}(nd\frac{d}{h})$
    $S = \tilde{Q}\tilde{K}^T$ // $n \times \frac{d}{h} \cdot \frac{d}{h} \times n \to \mathcal{O}(n^2\frac{d}{h})$
    for $j \leftarrow 1$ to $n$ do // softmax                // $\mathcal{O}(n^2)$
        $\Sigma =$ sum row $j$ of $S$ // $\mathcal{O}(n)$
        for $k \leftarrow 1$ to $n$ do                        // $\mathcal{O}(n)$
            $A_{j,k} = \frac{\exp(S_{j,k})}{\Sigma}$ // $\mathcal{O}(1)$
        end
    end
    head$_i = A\tilde{V}$ // $\mathcal{O}(n^2\frac{d}{h})$
end
Attention = concat(heads) // $\mathcal{O}(h)$
```

Multi-head attention has time complexity $\mathcal{O}(n^2d)$ as well. The individual head calculations have complexity $\mathcal{O}(n^2\frac{d}{h})$, but this process is repeated h times resulting in the final time complexity. Multi-head attention does however take longer than single head realistically. Although other operations are reduced dimensionality compared to single-head and therefore faster, softmax is the same complexity since it is done over an $n \times n$ matrix each iteration. This term doesn't show up in the final time complexity since $d > h$.

3 Graph Convolutional Networks

(a) Fix self adjacency

The $n \times n$ adjacency matrix A doesn't include each node itself. To make this the case, simply add identity of the same dimension, I_n, to A.

$$\tilde{A} = A + I_n$$

(b) Fix feature vector scaling

The rows of $\tilde{A}$ are not normalized, so multiplication with A changes the scale of the feature vectors. To rectify this, we can divide each element in $\tilde{A}$ by the degree of the nodes it connects. If $\tilde{D}$ is the diagonal node degree matrix of the graph, a scaled adjacency matrix can be created:

$$\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$$

This form ensures that the rows in A are normalized, so that they do not scale the feature vectors.

II Programming part

After playing with the hyperparameters, a batch size of 128 and learning rate of 0.001 were chosen. These were selected as arbitrary good starting points and not altered much since they had success in other models, and early stopping makes training largely insensitive to them. An embedding dimension of 256 was chosen and a hidden dimension of 512 was used. These parameters resulted in a perplexity score of around 200, which was a large improvement over the default parameters which resulted in a perplexity score of just over 300. The parameter num_steps was experimented with but didn't seem to affect the perplexity by much. An extremely high value produced incoherent babbling while a small number did the same. As a result, the default value of 10 was used. The final hyperparameters used were:

batch size	128
learning rate	0.001
embedding size	256
hidden dimension	512
dropout	0.1
num_steps	10

The model trained with these parameters achieved a perplexity of 164. For the sake of trying, a model was trained with an embedding of size 2048 and a hidden dimension of size 4096. This model achieved a moderately better score of 140 but took over 10x longer to train.

The following are some sentences produced by the model:

"in palo alto calif. valley that has been notified caused by golf to study gm 's <eos>"

“the two plants which narrowed had been financed <eos>”

“the president bush is a decent character with <unk>without any gain from the streets <eos>”

“this one time next wednesday <unk>to three times that brooks brothers says kansas <unk>a vigorous provision start in a <unk><unk><unk>in front ways and <unk>fax in <unk>president <unk>congress i schwab <eos>”

“target sales expected only sell as employers as men increased to be offered and <unk>operations <eos>”

Honestly none of these sentences make sense but a lot of the grammar is OK. I certainly couldn’t do as good of a job with only an hour or so of time to learn a new language.

Looking at the dataset, it seems like a lot of the sentences are headlines and financial news from the early 90’s so it makes sense that the model has trouble writing about anything else.