RNN for language modeling, attention mechanisms and Graph Convolutional Networks (GCN): Summary of results

1. RNN for Language Modeling

(a) Training parameters size:

Training parameter	Description	Size
L	Embedding matrix	$\|V\| \times d$
H	Hidden transformation matrix	$D \times D$
I	Input transformation matrix	$d \times D$
U	Output transformation matrix	$D \times \|V\|$
b_1	Bias for recurrent layer	$1 \times D$
b_2	Bias for projection layer	$1 \times \|V\|$

Where:

- d: Dimension of word embedding.
- $|V|$: Size of the vocabulary.
- D: Number of hidden units.

(b) Backpropagation:

- Compute gradient $\frac{\partial E^{(t)}}{\partial U}$

We define $\alpha^{(t)}$ as:

$$\alpha^{(t)} = h^{(t)}U + b_2 \qquad (1)$$

Therefore we write: $\hat{y}^{(t)} = softmax\left(\alpha^{(t)}\right)$

Using chain rule, we have:

$$\frac{\partial E^{(t)}}{\partial U} = \frac{\partial \alpha^{(t)}}{\partial U}\frac{\partial E^{(t)}}{\partial \alpha^{(t)}} \qquad (2)$$

From (1) we get:

$$\frac{\partial \alpha^{(t)}}{\partial U} = (h^{(t)})^T \qquad (3)$$

Now let's prove that $\frac{\partial E^{(t)}}{\partial \alpha^{(t)}} = \hat{y}^{(t)} - y^{(t)}$

For that, let's compute $\frac{\partial E^{(t)}}{\partial {\alpha_k}^{(t)}}$ for each k, integer in $[1, |V|]$. We have:

$$\frac{\partial E^{(t)}}{\partial {\alpha_k}^{(t)}} = -\frac{\partial \sum_{j=1}^{|V|} y_j^{(t)} \log\left(\hat{y}_j^{(t)}\right)}{\partial {\alpha_k}^{(t)}} = -\frac{\partial y_k^{(t)} \log\left(\hat{y}_k^{(t)}\right)}{\partial {\alpha_k}^{(t)}} - \sum_{\substack{j=1 \\ j\neq k}}^{|V|} \frac{\partial y_j^{(t)} \log\left(\hat{y}_j^{(t)}\right)}{\partial {\alpha_k}^{(t)}}$$

If $j = k$:

$$\frac{\partial y_k^{(t)} \log\left(\hat{y}_k^{(t)}\right)}{\partial {\alpha_k}^{(t)}} = y_k^{(t)} \frac{1}{\hat{y}_k^{(t)}} \frac{\partial \hat{y}_k^{(t)}}{\partial {\alpha_k}^{(t)}}$$

Since:

$$\hat{y}_k^{(t)} = softmax\left({\alpha_k}^{(t)}\right) = \frac{e^{{\alpha_k}^{(t)}}}{\sum_{s=1}^{|V|} e^{{\alpha_s}^{(t)}}}$$

We have:

$$\frac{\partial y_k^{(t)} \log\left(\hat{y}_k^{(t)}\right)}{\partial \alpha_k{}^{(t)}} = \frac{y_k^{(t)}}{\hat{y}_k^{(t)}} \frac{e^{\alpha_k{}^{(t)}} \sum_{s=1}^{|V|} e^{\alpha_s{}^{(t)}} - e^{\alpha_k{}^{(t)}} e^{\alpha_k{}^{(t)}}}{(\sum_{s=1}^{|V|} e^{\alpha_s{}^{(t)}})^2}$$
$$= \frac{y_k^{(t)}}{\hat{y}_k^{(t)}} \left(\frac{e^{\alpha_k{}^{(t)}}}{\sum_{s=1}^{|V|} e^{\alpha_s{}^{(t)}}} - \left(\frac{e^{\alpha_k{}^{(t)}}}{\sum_{s=1}^{|V|} e^{\alpha_s{}^{(t)}}} \right)^2 \right) = \frac{y_k^{(t)}}{\hat{y}_k^{(t)}} \left(\hat{y}_k^{(t)} - \left(\hat{y}_k^{(t)} \right)^2 \right)$$
$$= y_k^{(t)} \left(1 - \hat{y}_k^{(t)} \right)$$

If $j \neq k$:

$$\frac{\partial y_j^{(t)} \log\left(\hat{y}_j^{(t)}\right)}{\partial \alpha_k{}^{(t)}} = \frac{y_j^{(t)}}{\hat{y}_j^{(t)}} \frac{- e^{\alpha_j{}^{(t)}} . e^{\alpha_k{}^{(t)}}}{(\sum_{s=1}^{|V|} e^{\alpha_s{}^{(t)}})^2} = \frac{y_j^{(t)}}{\hat{y}_j^{(t)}} \frac{- e^{\alpha_j{}^{(t)}}}{\sum_{s=1}^{|V|} e^{\alpha_s{}^{(t)}}} \frac{e^{\alpha_k{}^{(t)}}}{\sum_{s=1}^{|V|} e^{\alpha_s{}^{(t)}}} = - \frac{y_j^{(t)}}{\hat{y}_j^{(t)}} \hat{y}_j^{(t)} \hat{y}_k^{(t)}$$
$$= - y_j^{(t)} \hat{y}_k^{(t)}$$

Therefore, for every integer k in $[1, |V|]$ (combining the previous 2 cases of $j = k$ and $j \neq k$):

$$\frac{\partial E^{(t)}}{\partial \alpha_k{}^{(t)}} = - y_k^{(t)} \left(1 - \hat{y}_k^{(t)} \right) + \sum_{\substack{j=1 \\ j \neq k}}^{|V|} y_j^{(t)} \hat{y}_k^{(t)} = - y_k^{(t)} + \left(\sum_{j=1}^{|V|} y_j^{(t)} \right) \hat{y}_k^{(t)} = - y_k^{(t)} + \hat{y}_k^{(t)}$$

Since $y^{(t)}$ is a one-hot vector (of the target word), i.e.: $\sum_{j=1}^{|V|} y_j^{(t)} = 1$.

The previous expression is valid for any integer k in $[1, |V|]$, thus:

$$\frac{\partial E^{(t)}}{\partial \alpha^{(t)}} = \hat{y}^{(t)} - y^{(t)} \qquad (4)$$

Replacing in expression (2), using (3) and (4), we have proven the hint:

$$\frac{\partial E^{(t)}}{\partial U} = \frac{\partial \alpha^{(t)}}{\partial U} \frac{\partial E^{(t)}}{\partial \alpha^{(t)}} = (h^{(t)})^T (\hat{y}^{(t)} - y^{(t)})$$

We can check that $\frac{\partial E^{(t)}}{\partial U}$ is indeed of size $D \times |V|$.

- Compute gradient $\frac{\partial E^{(t)}}{\partial b_2}$

Using chain rule, we write:

$$\frac{\partial E^{(t)}}{\partial b_2} = \frac{\partial E^{(t)}}{\partial \alpha^{(t)}} \frac{\partial \alpha^{(t)}}{\partial b_2}$$

From (4), we have shown that:

$$\frac{\partial E^{(t)}}{\partial \alpha^{(t)}} = \hat{y}^{(t)} - y^{(t)}$$

Also, from (1), we see that:

$$\frac{\partial \alpha^{(t)}}{\partial b_2} = I_{|V| \times |V|}$$

Where $I_{|V| \times |V|}$ is the identity matrix of size $|V| \times |V|$.

Thus:

$$\frac{\partial E^{(t)}}{\partial b_2} = \hat{y}^{(t)} - y^{(t)}$$

We can check that $\frac{\partial E^{(t)}}{\partial b_2}$ is indeed of size $1 \times |V|$.

- Compute gradient $\left.\frac{\partial E^{(t)}}{\partial I}\right|_{(t)}$

For this gradient, we compute the derivative for time step *t* only.
We define $\beta^{(t)}$ as:

$$\beta^{(t)} = h^{(t-1)}H + e^{(t)}I + b_1 \qquad (5)$$

Therefore we write: $h^{(t)} = sigmoid\left(\beta^{(t)}\right)$
Using chain rule, we have:

$$\left.\frac{\partial E^{(t)}}{\partial I}\right|_{(t)} = \frac{\partial h^{(t)}}{\partial I}\frac{\partial E^{(t)}}{\partial h^{(t)}} \qquad (6)$$

From (4), we have shown that:

$$\frac{\partial E^{(t)}}{\partial \alpha^{(t)}} = \hat{y}^{(t)} - y^{(t)} \qquad (7)$$

Also, from (1), we see that:

$$\frac{\partial \alpha^{(t)}}{\partial h^{(t)}} = U^T \qquad (8)$$

Thus, using chain rule we write:

$$\frac{\partial E^{(t)}}{\partial h^{(t)}} = \frac{\partial E^{(t)}}{\partial \alpha^{(t)}}\frac{\partial \alpha^{(t)}}{\partial h^{(t)}} = (\hat{y}^{(t)} - y^{(t)})U^T \qquad (9)$$

Let's compute $\frac{\partial h^{(t)}}{\partial I}$. Using chain rule again, we write:

$$\frac{\partial h^{(t)}}{\partial I} = \frac{\partial \beta^{(t)}}{\partial I}\frac{\partial h^{(t)}}{\partial \beta^{(t)}}$$

From (5), we have:

$$\frac{\partial \beta^{(t)}}{\partial I} = \left(e^{(t)}\right)^T$$

Also, we know the derivative of sigmoid as:

$$\frac{\partial sigmoid(x)}{\partial x} = sigmoid(x)\left(1 - sigmoid(x)\right) \qquad (10)$$

Using (10) we compute $\frac{\partial h^{(t)}}{\partial \beta^{(t)}}$, since $h^{(t)} = sigmoid\left(\beta^{(t)}\right)$, and using the right dimensions we have:

$$\frac{\partial h^{(t)}}{\partial \beta^{(t)}} = \left(1_{1\times D} - sigmoid\left(\beta^{(t)}\right)\right)\left(sigmoid\left(\beta^{(t)}\right)\right)^T$$

Where $1_{1\times D}$ is the vector of ones of size $1\times D$.

Therefore:

$$\frac{\partial h^{(t)}}{\partial I} = \left(e^{(t)}\right)^T\left(1_{1\times D} - sigmoid\left(\beta^{(t)}\right)\right)\left(sigmoid\left(\beta^{(t)}\right)\right)^T \qquad (11)$$

Replacing (9) and (11) in (6), we get:

$$\left.\frac{\partial E^{(t)}}{\partial I}\right|_{(t)} = \left(e^{(t)}\right)^T\left(1_{1\times D} - sigmoid\left(\beta^{(t)}\right)\right)\left(sigmoid\left(\beta^{(t)}\right)\right)^T(\hat{y}^{(t)} - y^{(t)})U^T$$

We can check that size of $\left.\frac{\partial E^{(t)}}{\partial I}\right|_{(t)}$ is $d \times D$.

- Compute gradient $\left.\frac{\partial E^{(t)}}{\partial H}\right|_{(t)}$

For this gradient, we compute the derivative for time step *t* only.
Using chain rule, we write:

$$\left.\frac{\partial E^{(t)}}{\partial H}\right|_{(t)} = \frac{\partial E^{(t)}}{\partial h^{(t)}}\frac{\partial h^{(t)}}{\partial H} \qquad (12)$$

Using chain rule again and the right dimensions, we write:

$$\frac{\partial E^{(t)}}{\partial h^{(t)}} = \frac{\partial \alpha^{(t)}}{\partial h^{(t)}}\frac{\partial E^{(t)}}{\partial \alpha^{(t)}} = U(\hat{y}^{(t)} - y^{(t)})^T \qquad (13)$$

And applying chain rule on the other term from (12), we have:

$$\frac{\partial h^{(t)}}{\partial H} = \frac{\partial \beta^{(t)}}{\partial H}\frac{\partial h^{(t)}}{\partial \beta^{(t)}}$$

Using the derivative of $sigmoid$, and the right dimensions, we have:

$$\frac{\partial h^{(t)}}{\partial \beta^{(t)}} = \left(sigmoid\left(\beta^{(t)}\right)\right)^T \left(1_{1\times D} - sigmoid\left(\beta^{(t)}\right)\right) \qquad (14)$$

And from (5):

$$\frac{\partial \beta^{(t)}}{\partial H} = h^{(t-1)}$$

Thus:

$$\frac{\partial h^{(t)}}{\partial H} = h^{(t-1)}\left(sigmoid\left(\beta^{(t)}\right)\right)^T \left(1_{1\times D} - sigmoid\left(\beta^{(t)}\right)\right) \qquad (15)$$

Replacing (13) and (15) in (12), we get:

$$\left.\frac{\partial E^{(t)}}{\partial H}\right|_{(t)} = U(\hat{y}^{(t)} - y^{(t)})^T h^{(t-1)}\left(sigmoid\left(\beta^{(t)}\right)\right)^T \left(1_{1\times D} - sigmoid\left(\beta^{(t)}\right)\right)$$

We can check that the size of $\left.\frac{\partial E^{(t)}}{\partial H}\right|_{(t)}$ is $D \times D$.

- Compute gradient $\left.\frac{\partial E^{(t)}}{\partial b_1}\right|_{(t)}$

For this gradient, we compute the derivative for time step *t* only.

Using chain rule, we write:

$$\left.\frac{\partial E^{(t)}}{\partial b_1}\right|_{(t)} = \frac{\partial E^{(t)}}{\partial \beta^{(t)}}\frac{\partial \beta^{(t)}}{\partial b_1}$$

On one hand:

$$\frac{\partial \beta^{(t)}}{\partial b_1} = I_{D\times D}$$

Where $I_{D\times D}$ is the identity matrix of size $D\times D$.

On the other hand:

$$\frac{\partial E^{(t)}}{\partial \beta^{(t)}} = \frac{\partial E^{(t)}}{\partial h^{(t)}}\frac{\partial h^{(t)}}{\partial \beta^{(t)}}$$

Moreover, we found in (9) and (14) that:

$$\frac{\partial E^{(t)}}{\partial h^{(t)}} = (\hat{y}^{(t)} - y^{(t)})U^T \qquad (9)$$

$$\frac{\partial h^{(t)}}{\partial \beta^{(t)}} = \left(sigmoid\left(\beta^{(t)}\right)\right)^T \left(1_{1\times D} - sigmoid\left(\beta^{(t)}\right)\right) \qquad (14)$$

Thus:

$$\frac{\partial E^{(t)}}{\partial \beta^{(t)}} = (\hat{y}^{(t)} - y^{(t)})U^T \left(sigmoid\left(\beta^{(t)}\right)\right)^T \left(1_{1\times D} - sigmoid\left(\beta^{(t)}\right)\right)$$

Therefore:

$$\left.\frac{\partial E^{(t)}}{\partial b_1}\right|_{(t)} = \frac{\partial E^{(t)}}{\partial \beta^{(t)}}\frac{\partial \beta^{(t)}}{\partial b_1} = (\hat{y}^{(t)} - y^{(t)})U^T \left(sigmoid\left(\beta^{(t)}\right)\right)^T \left(1_{1\times D} - sigmoid\left(\beta^{(t)}\right)\right)$$

We can check that the size of $\left.\frac{\partial E^{(t)}}{\partial b_1}\right|_{(t)}$ is $1 \times D$.

- Compute gradient $\frac{\partial E^{(t)}}{\partial h^{(t-1)}}$

Using chain rule, we write:

$$\frac{\partial E^{(t)}}{\partial h^{(t-1)}} = \frac{\partial E^{(t)}}{\partial h^{(t)}}\frac{\partial h^{(t)}}{\partial h^{(t-1)}} \qquad (15)$$

We have from (9):

$$\frac{\partial E^{(t)}}{\partial h^{(t)}} = (\hat{y}^{(t)} - y^{(t)})U^T \qquad (9)$$

And using chain rule:

$$\frac{\partial h^{(t)}}{\partial h^{(t-1)}} = \frac{\partial h^{(t)}}{\partial \beta^{(t)}}\frac{\partial \beta^{(t)}}{\partial h^{(t-1)}}$$

We know from (14) that:

$$\frac{\partial h^{(t)}}{\partial \beta^{(t)}} = \left(sigmoid\left(\beta^{(t)}\right)\right)^T \left(1_{1\times D} - sigmoid\left(\beta^{(t)}\right)\right) \qquad (14)$$

Moreover, from (5) we have:

$$\frac{\partial \beta^{(t)}}{\partial h^{(t-1)}} = H$$

Thus:

$$\frac{\partial h^{(t)}}{\partial h^{(t-1)}} = \left(sigmoid\left(\beta^{(t)}\right)\right)^T \left(1_{1\times D} - sigmoid\left(\beta^{(t)}\right)\right) H \qquad (16)$$

Replacing (9) and (16) in (15), we get:

$$\frac{\partial E^{(t)}}{\partial h^{(t-1)}} = (\hat{y}^{(t)} - y^{(t)})U^T \left(sigmoid\left(\beta^{(t)}\right)\right)^T \left(1 - sigmoid\left(\beta^{(t)}\right)\right) H$$

We can check that the size of $\frac{\partial E^{(t)}}{\partial h^{(t-1)}}$ is $1 \times D$.

(c) Perplexity:

We learnt in class that the perplexity for a given time step t is defined as:

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

On the other hand, the cross entropy for a given time step t is defined as:

$$E^{(t)}(\theta) = CE\left(y_j^{(t)}, \hat{y}_j^{(t)}\right) = -\sum_{j=1}^{|V|} y_j^{(t)} \log\left(\hat{y}_j^{(t)}\right)$$

Considering that $y^{(t)}$ is a one-hot vector, we suppose the only non-zero element of $y^{(t)}$ is $y_k^{(t)}$ and we have that $y_k^{(t)} = 1$. Thus, the perplexity and the cross entropy become:

$$PP^{(t)}\left(y_j^{(t)}, \hat{y}_j^{(t)}\right) = \frac{1}{\hat{y}_k^{(t)}}$$

$$CE\left(y_j^{(t)}, \hat{y}_j^{(t)}\right) = -\log\left(\hat{y}_k^{(t)}\right) = \log\left(\frac{1}{\hat{y}_k^{(t)}}\right)$$

Therefore, we find a relationship between the cross entropy and the perplexity:

$$CE\left(y_j^{(t)}, \hat{y}_j^{(t)}\right) = \log\left(PP^{(t)}\left(y_j^{(t)}, \hat{y}_j^{(t)}\right)\right)$$

Actually, in information theory, the perplexity is defined from the cross entropy:

$$PP^{(t)}\left(y_j^{(t)}, \hat{y}_j^{(t)}\right) = b^{CE\left(y_j^{(t)}, \hat{y}_j^{(t)}\right)} = b^{-\Sigma_{j=1}^{|V|} y_j^{(t)} \log_b\left(\hat{y}_j^{(t)}\right)}$$

Where the base *b* can be any integer, on the condition that this base is the same one used in both the perplexity and in the logarithm of the cross entropy.

(d) Implementation:

- Results:

After implementing RNN in *RNNLM.py*, we ran several models using different hyperparameters. Below is a summary of the main models that we tried and their associated (best) training, (best) validation and test perplexities.

#	Hyperparameters	Best Training perplexity	Best Validation Perplexity	Test Perplexity
1	batch_size = 64 embed_size = 50 hidden_size = 100 num_steps = 10 max_epochs = 50 early_stopping = 2 dropout = 0.9 lr = 0.001	125.56	186.32	167.33
2	**batch_size = 128** embed_size = 50 hidden_size = 100 num_steps = 10 max_epochs = 50 early_stopping = 2 dropout = 0.9 lr = 0.001	121.88	179.12	159.64
3	batch_size = 128 embed_size = 50 **hidden_size = 200** num_steps = 10 max_epochs = 50 early_stopping = 2 dropout = 0.9 lr = 0.001	107.50	171.51	155.67
4	batch_size = 128 embed_size = 50	115.81	178.16	160.75

	hidden_size = 100 num_steps = 10 max_epochs = 50 **early_stopping = 5** dropout = 0.9 lr = 0.001			
5	batch_size = 128 embed_size = 50 hidden_size = 100 num_steps = 10 max_epochs = 50 early_stopping = 2 **dropout = 0.2** lr = 0.001	340.10	299.55	261.87
6	batch_size = 128 embed_size = 50 hidden_size = 100 num_steps = 10 max_epochs = 50 early_stopping = 2 **dropout = 0.6** lr = 0.001	176.91	204.05	179.25
7	batch_size = 128 embed_size = 50 hidden_size = 100 num_steps = 10 max_epochs = 50 early_stopping = 2 dropout = 0.9 **lr = 0.01**	105.40	172.02	158.97
8	batch_size = 64 embed_size = 1000 hidden_size = 100 num_steps = 10 max_epochs = 50 early_stopping = 5 dropout = 0.9 lr = 0.001	91.54	183.35	154.83
9	**batch_size = 128** embed_size = 1000 hidden_size = 100 num_steps = 10 max_epochs = 50 **early_stopping = 2** dropout = 0.9 **lr = 0.01**	136.97	208.73	193.57
10	**batch_size = 128** embed_size = 1000 hidden_size = 100	99.09	166.33	150.61

	num_steps = 10 max_epochs = 50 **early_stopping = 2** dropout = 0.9 lr = 0.001			
11	batch_size = 128 embed_size = 1000 **hidden_size = 200** num_steps = 10 max_epochs = 50 early_stopping = 2 dropout = 0.9 lr = 0.001	82.02	154.97	142.12
12	**batch_size = 256** embed_size = 1000 hidden_size = 200 num_steps = 10 max_epochs = 50 early_stopping = 2 dropout = 0.9 lr = 0.001	80.24	151.32	138.38

Few notes:

- Model 9 converges fast: after only 4 epochs, the model yields a test perplexity of 193.57. This is due to the larger learning rate that fastens the optimization of the loss. However, the result is still not the best among other models we tried.
- Model 10 converges after 15 epochs.
- **The best model is Model 12. The latter converges after 17 epochs, providing a test perplexity of 138.38.**

Conclusion:

- Best model: **Model 12**.
- Best model's test perplexity: **138.38**
- Best model's hyperparameters:
  - batch_size = 256
  - embed_size = 1000
  - hidden_size = 200
  - num_steps = 10
  - max_epochs = 50
  - early_stopping = 2
  - dropout = 0.9
  - lr = 0.001
- This model converges after 17 epochs.

- Observations:

We report below few interesting outputs that we obtained using the best model above (**Model 12**).

Input sequence	Output sequence
In palo alto	palo alto auto industries and morgan stanley controlled by mr. azoff in his <unk> to <unk> the <unk> <unk> named manitoba <eos>
Hello	is for theatrical issues in new york stock exchange composite trading yesterday <unk> shares in efforts to acquire valley armstrong <unk> subsidiary to assess the utility of increasing plant 's contributions <eos>
The latest news	latest news that in the corner it was hired to make the number of the leadership inspection of public affairs he and the toy time <eos>
Unfortunately	about hurricane hugo and atlantic richfield corp. up a morning block including <unk> & electric ltd. 's <unk> vans <unk> to several suits <eos>
Tourism in Italy	in italy 's affiliated business tower last spring <eos>
Reporters	reports sent each of a five spot still have money supply the company 's fifth avenue <eos>
Happiness	N million shares outstanding <eos>

From the last output sequence: **to an AI agent, does happiness rhyme with money?** ☺

2. Attention Mechanism

(a) Number of parameters:

- *Single-head attention:* The number of parameters corresponds to the number of components of the learnable weight matrices W^Q, W^K and W^V, all of which are of size $d \times d$. Therefore, the number of parameters in a single-head attention is $3d^2$.

- *Multi-head attention:* The number of parameters corresponds to the number of components of the learnable weight matrices W_i^Q, W_i^K and W_i^V for each head i, all of which are of size $d \times \frac{d}{h}$. Therefore, the number of parameters for one head i is $\frac{3d^2}{h}$, and for h heads, the number of parameters becomes $3d^2$.

- *Comparison:* Both *single-head* and *multi-head attention* have the same number of parameters in this case, since we used a reduced dimension $\frac{d}{h}$ in the *multi-head attention*. This bolsters the claim of the "Attention is all you need" paper [2]: "Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality."

(b) Computational complexity:

- *Single-head attention:*

$$Attention(QW^Q, KW^K, VW^V) = softmax(QW^Q(KW^K)^T)VW^V$$

According to the above, we perform matrix multiplication of a $n \times n$ matrix with a $n \times d$ matrix. The associated complexity for this operation is $O(n^2 d)$[1].
The $softmax$ operation computes the $softmax$ on $n \times n$ components. Thus, the complexity associated with this operation is $O(n^2)$.
The total complexity is then $O(n^2 d + n^2) = O(n^2(d+1))$.
Note: If the depth $d \gg 1$, then the complexity becomes $O(n^2 d)$.

- *Multi-head attention:* For head i, $i = 1, \dots, h$:

$$head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right) = softmax\left(QW_i^Q\left(KW_i^K\right)^T\right)VW_i^V$$

According to the above, for one head i we perform matrix multiplication of a $n \times n$ matrix with a $n \times \frac{d}{h}$ matrix. The associated complexity for this operation is $O(n^2 \frac{d}{h})$ [1]. For h heads, the complexity becomes $O(n^2 d)$.
To combine all h heads, we perform matrix multiplication of a $n \times \frac{d}{h}$ matrix with a $\frac{d}{h} \times d$ matrix. The associated complexity for this operation is $O\left(n\frac{d^2}{h}\right)$ [1]. Since h is a constant, this complexity is same as $O(nd^2)$.
For one head i, the $softmax$ operation computes the $softmax$ on $n \times n$ components. Thus, the complexity associated with this operation is $O(n^2)$. For h heads, the complexity becomes $O(n^2 h)$.
The total complexity is then $O(n^2 d + nd^2 + n^2 h) = O(n^2(d+h) + nd^2)$.
Note: If the depth $d \gg h$, then the complexity becomes $O(n^2 d + nd^2)$.

- *Comparison:* the *multi-head attention*'s complexity has an extra term $O(nd^2)$ that is due to the concatenation of the h heads results. This extra term would not be too expensive compared to the *single-head attention* if d is not too large.

3. Graph Convolutional Networks (GCN)

(c) Possible solution for the first limitation:

We can consider that there is an edge between each node of the graph and itself (self-loop). This way the adjacency matrix will contain the center node on which the convolution is performed, and this center node's feature vectors will also be considered when aggregating neighbors' information in GCN:

$$X^{l+1} = \sigma\left(AX^l W^l\right)$$

In practice, we can add self-loops to each node of the graph by summing the identity matrix with the adjacency matrix A, before running the propagation rule.

(d) Possible solution for the second limitation:

The adjacency matrix A contains neighboring nodes of each node. The sum of its i^{th} row represents the degree of the i^{th} node of the graph. Therefore, normalizing A comes back to reducing each node's degree to 1. In other words, we need to divide each row in the adjacency matrix by the degree of the node corresponding to that row. By computing the diagonal degree matrix D of the graph, and multiplying its inverse D^{-1} with the adjacency matrix A, we get to divide each row i of A by the degree of node i, which allows the rows of A to sum to 1. In practice, we can use a symmetric normalization, by performing the following modification of A, as in [1]: $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$

[1] Computed using a naïve algorithm for matrix computation. There are better implementations of matrix multiplication that can achieve better complexities than a polynomial of order 3.

Note: There might also be other ways to normalize A, such as applying $softmax$ to each row.

References

[1] Kipf, T. N. and Welling M. Semi-Supervised Classification with Graph Convolutional Networks (2017). *5th International Conference on Learning Representations (ICLR-17).*

[2] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., Polosukhin, I. Attention Is All You Need. *31st Conference on Neural Information Processing Systems (NIPS 2017)*. Long Beach, CA, USA. Online version: https://arxiv.org/pdf/1706.03762.pdf