CSC413H1 - Assignment 4

1.1.2: Given the weights $w_1, \ldots, w_n$, denote $h_0 = x$ and $h_i = \text{ReLU}(w_i h_{i-1})$ for $i \in \{1, \ldots, n\}$ such that $f(x) = h_n$. Then,

$$\frac{\partial h_i}{\partial h_{i-1}} = w_i \cdot \frac{d}{dh_{i-1}} \text{ReLU}(w_i h_{i-1}) = w_i \cdot \mathbb{I}(w_i h_{i-1} > 0)$$
$$\frac{\partial f(x)}{\partial x} = \prod_{i=1}^{n} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=1}^{n} w_i \cdot \mathbb{I}(w_i h_{i-1} > 0) \text{ by the chain rule}$$

for all i. Note that $|\frac{\partial f(x)}{\partial x}| \geq 0$ by definition and $|\frac{\partial f(x)}{\partial x}| \leq \prod_{i=1}^{n} |w_i|$ if all of the indicator functions evaluate to 1. The gradient does not necessarily have to vanish or explode. It vanishes when any $w_i = 0$, when $x = 0$, when $x < 0$ and $w_1 > 0$, when $x < 0$ and any $w_i < 0$ for $i > 1$, or when $x > 0$ and any $w_i < 0$. It explodes when the w_i's are large.

1.2.1: Denote $S(x) = \text{sigmoid}(x)$ and note that $S'(x) \leq \frac{1}{4}$. Since the Jacobian of $S(Wx_t)$ for all t is a diagonal matrix with entries $\leq \frac{1}{4}$, its singular values are therefore also $\leq \frac{1}{4}$. In other words, $\sigma_{\max}(\frac{\partial S(Wx_t)}{\partial Wx_t}) = \frac{1}{4}$. Then, by the chain rule and the hint,

$$\frac{\partial x_n}{\partial x_1} = \prod_{i=1}^{n-1} \frac{\partial x_{t+1}}{\partial x_t} = \prod_{i=1}^{n-1} \frac{\partial S(Wx_t)}{\partial x_t} = \prod_{i=1}^{n-1} \frac{\partial S(Wx_t)}{\partial Wx_t} \frac{\partial Wx_t}{\partial x_t} = \prod_{i=1}^{n-1} \frac{\partial S(Wx_t)}{\partial Wx_t} W$$
$$\sigma_{\max}(\frac{\partial x_n}{\partial x_1}) \leq \prod_{i=1}^{n-1} \sigma_{\max}(\frac{\partial x_{t+1}}{\partial x_t}) \leq \prod_{i=1}^{n-1} \sigma_{\max}(\frac{\partial S(Wx_t)}{\partial Wx_t}) \sigma_{\max}(W) = \prod_{i=1}^{n-1} \frac{1}{4} \cdot \frac{1}{4} = (\frac{1}{16})^{n-1}$$

and furthermore singular values are non-negative by convention, so $0 \leq \sigma_{\max}(\frac{\partial x_n}{\partial x_1}) \leq (\frac{1}{16})^{n-1}$.

1.3.1: Assume that each vector-vector product has $\mathcal{O}(1)$ time. Substituting in the kernel function yields

$$\alpha_i = \frac{\sum_{j=1}^{n} k(Q_i, K_j) V_j}{\sum_{j=1}^{n} k(Q_i, K_j)} = \frac{\sum_{j=1}^{n} \phi(Q_i)^T \phi(K_j) V_j}{\sum_{j=1}^{n} \phi(Q_i)^T \phi(K_j)} = \frac{\phi(Q_i)^T \sum_{j=1}^{n} \phi(K_j) V_j}{\phi(Q_i)^T \sum_{j=1}^{n} \phi(K_j)}$$

and notice that $\sum_{j=1}^{n} \phi(K_j) V_j$ and $\sum_{j=1}^{n} \phi(K_j)$ can be precomputed and stored in $\mathcal{O}(n)$ time. Thus, computing the α_i's for $i \in \{1, \ldots, n\}$ takes $\mathcal{O}(n)$ time in total.

1.3.2: Denote the low-rank SVD of P as $\begin{bmatrix} u_1 & \ldots & u_k \end{bmatrix} \text{diag}(\sigma_1, \ldots, \sigma_k) \begin{bmatrix} v_1 & \cdots & v_k \end{bmatrix}^T$ for n-dimensional vectors $u_i, v_i \in \mathbb{R}^n$ and singular values $\sigma_i \in \mathbb{R}$. Then, the self-attention involves computing the following:

- $M_1 := \begin{bmatrix} v_1 & \cdots & v_k \end{bmatrix}^T V$, which is multiplying a $k \times n$ matrix with a $n \times d$ matrix and takes $\mathcal{O}(nkd)$ time;
- $M_2 := \text{diag}(\sigma_1, \ldots, \sigma_k) M_1$, which is multiplying each singular value with the corresponding d-dimensional row and takes $\mathcal{O}(kd)$ time;
- $\begin{bmatrix} u_1 & \ldots & u_k \end{bmatrix}^T M_2$, which is multiplying a $n \times k$ matrix with a $k \times d$ matrix and takes $\mathcal{O}(nkd)$ time.

Thus, the total time complexity is $\mathcal{O}(nkd)$.

1.4.1: Assume that x is indexed from 0 to $n-1$ inclusive so that the matrix of α's is

$$\begin{bmatrix} \alpha_{1,0} & \ldots & \alpha_{1,n-1} \\ \vdots & \ddots & \vdots \\ \alpha_{n,0} & \ldots & \alpha_{n,n-1} \end{bmatrix}$$

and $p = \{p_{-n+2}, \ldots, p_n\}$. Set $p_1 = \ln 2$, $p_{-1} = 0$, $p_k = -\infty$ for all other k's, $W_Q = W_K = 0$, and $W_V = \sum_{k=-n+2}^{n} \exp(p_k) = \exp(p_1) + \exp(p_{-1}) = 3$. This is equivalent to the convolution since for $i \in \{1, \ldots, n-2\}$,

$$\begin{aligned}
\text{softmax}(\alpha(W_Q x, W_K x, p))_i W_V x &= \text{softmax}(\begin{bmatrix}\ldots & \alpha_{i,i-1} & \alpha_{i,i} & \alpha_{i,i+1} & \ldots\end{bmatrix}) W_V x \\
&= \text{softmax}(\begin{bmatrix}\ldots & p_1 & p_0 & p_{-1} & \ldots\end{bmatrix}) W_V x \\
&= \begin{bmatrix}\ldots & \frac{2}{3} & 0 & \frac{1}{3} & \ldots\end{bmatrix} 3x \\
&= 2x_{i-1} + x_{i+1} = \text{Conv1D}(x; w)_i
\end{aligned}$$

and for $i = n-1$, $\text{softmax}(\alpha(W_Q x, W_K x, p))_{n-1} W_V x = \begin{bmatrix}\ldots & \frac{2}{3} & 0\end{bmatrix} W_V x = 2x_{n-2} = \text{Conv1D}(x; w)_{n-1}$ where $x_n = 0$ due to zero-padding. Note that any values of p_1 and p_{-1} such that $p_1 = p_{-1} + \ln 2$ work.

1.4.2: Assume that x is indexed from 1 to n inclusive so that the matrix of α's is

$$\begin{bmatrix} \alpha_{1,1} & \ldots & \alpha_{1,1} \\ \vdots & \ddots & \vdots \\ \alpha_{n,1} & \ldots & \alpha_{n,n} \end{bmatrix}$$

and $p = \{p_{-n+1}, \ldots, p_{n-1}\}$. Set $p_{k'} = 0$ for $k' \in \{-k, \ldots, k\}$ and $-\infty$ otherwise, $W_V = 1$, and $W_Q = W_K = c$ for some large constant c, such as 100. Also, note that $\sqrt{d_k} = 1$. This approximates max pooling since for $i \in \{k+1, \ldots, n-k\}$,

$$\begin{aligned}
\text{softmax}(\alpha(W_Q x, W_K x, p))_i W_V x &= \text{softmax}(\begin{bmatrix}\ldots & \alpha_{i,i-k} & \ldots & \alpha_{i,i+k} & \ldots\end{bmatrix}) W_V x \\
&= \text{softmax}(\begin{bmatrix}\ldots & c^2 x_i x_{i-k} + p_k & \ldots & c^2 x_i x_{i+k} + p_{-k} & \ldots\end{bmatrix}) W_V x \\
&= \begin{bmatrix}\ldots & s_i \exp(c^2 x_i x_{i-k}) & \ldots & s_i \exp(c^2 x_i x_{i+k}) & \ldots\end{bmatrix} x \\
&= s_i \exp(c^2 x_i x_{i-k}) x_{i-k} + \ldots + s_i \exp(c^2 x_i x_{i+k}) x_{i+k} \\
&\approx \max(x_{i-k}, \ldots, x_{i+k}) = \text{MaxPool}(x)_i
\end{aligned}$$

where $s_i = \sum_{j=i-k}^{i+k} \exp(x_i x_j)$ and the last line holds since:

- if there is one maximum $x_j \in \{x_{i-k}, \ldots, x_{i+k}\}$, $s_i \exp(c^2 x_i x_j) x_j \approx x_j$;
- if there are multiple maximums $x_j \in \{x_{i-k}, \ldots, x_{i+k}\}$, $\sum_{x_j} s_i \exp(c^2 x_i x_j) x_j \approx x_j$.

2.1.1: Pick $c = 72$ and let $n \in \mathbb{N}$ and $\delta > 0$. If $\delta \geq 1$, by the Cauchy-Schwarz inequality, there are infinitely many $u_i \in \mathbb{R}^n$ such that $||u_i||_2 = 1$ and $|u_i^T u_j| \leq ||u_i||_2^2 ||u_j||_2^2 = 1 \leq \delta$ where $i \neq j$, in which case the claim is satisfied.

If $\delta < 1$, define $\epsilon = \delta/3$ and $d = \lfloor \exp(n\delta^2/c) \rfloor - 1$. Using $X \subset \mathbb{R}^d$ from the hint and the JL lemma, we have that $8 \ln(d+1)/\epsilon^2 = c \ln(\lfloor \exp(\frac{n\delta^2}{c}) \rfloor)/\delta^2 \leq n$, implying there exists a $n \times d$ matrix M satisfying the JL inequality. Then, take this M and notice that for each $(0, e_i)$ pair,

$$1 - \epsilon = (1-\epsilon)||e_i||_2^2 \leq ||Me_i||_2^2 \leq (1+\epsilon)||e_i||_2^2 = 1 + \epsilon \tag{1}$$

and for each (e_i, e_j) pair where $i \neq j$, we have $||e_i - e_j||_2^2 = 2$, so

$$\begin{aligned}
&2(1-\epsilon) = (1-\epsilon)||e_i - e_j||_2^2 \leq ||Me_i - Me_j||_2^2 \leq (1+\epsilon)||e_i - e_j||_2^2 = 2(1+\epsilon) \\
\implies &2(1-\epsilon) \leq ||Me_i||_2^2 + ||Me_j||_2^2 - 2(Me_i)^T(Me_j) \leq 2(1+\epsilon) \\
\implies &\frac{||Me_i||_2^2}{2} + \frac{||Me_j||_2^2}{2} - (1+\epsilon) \leq (Me_i)^T(Me_j) \leq \frac{||Me_i||_2^2}{2} + \frac{||Me_j||_2^2}{2} - (1-\epsilon) \\
\implies &|(Me_i)^T(Me_j)| \leq 2\epsilon \\
\implies &\left| \frac{(Me_i)^T}{||Me_i||_2^2} \frac{(Me_j)}{||Me_j||_2^2} \right| \leq \left| \frac{(Me_i)^T}{\sqrt{1-\epsilon}} \frac{(Me_j)}{\sqrt{1-\epsilon}} \right| \leq \frac{2\epsilon}{1-\epsilon} = \frac{2\delta}{3-\delta} < \frac{2\delta}{2} = \delta
\end{aligned}$$

where the second line follows from the hint, the fourth line follows from (1), and the last line follows from (1), the fourth line, and the fact that $\delta < 1 \implies 3 - \delta > 2$. Finally, define $U = \{Me_i/||Me_i||_2^2\}_{i=1}^d$ such that $|U| = d \in \Omega(\exp(n\delta^2/c))$; $\forall u \in U$, $||u||_2 = 1$; and $\forall u_i, u_j \in U$ where $i \neq j$, $|u_i^T u_j| \leq \delta$ by the result above.

3.1: See the attached code.

3.2: See the attached code.

3.3: The output is shown below:

```
Epoch: 0090 loss_train: 0.8286 acc_train: 0.8286 loss_val: 1.1313 acc_val: 0.6223 time: 0.0029s
Epoch: 0091 loss_train: 0.7919 acc_train: 0.8286 loss_val: 1.1238 acc_val: 0.6254 time: 0.0029s
Epoch: 0092 loss_train: 0.7797 acc_train: 0.8429 loss_val: 1.1160 acc_val: 0.6293 time: 0.0047s
Epoch: 0093 loss_train: 0.7539 acc_train: 0.8786 loss_val: 1.1086 acc_val: 0.6336 time: 0.0026s
Epoch: 0094 loss_train: 0.8075 acc_train: 0.8286 loss_val: 1.1020 acc_val: 0.6386 time: 0.0025s
Epoch: 0095 loss_train: 0.7681 acc_train: 0.8500 loss_val: 1.0956 acc_val: 0.6421 time: 0.0026s
Epoch: 0096 loss_train: 0.7314 acc_train: 0.8286 loss_val: 1.0892 acc_val: 0.6452 time: 0.0027s
Epoch: 0097 loss_train: 0.7305 acc_train: 0.8786 loss_val: 1.0826 acc_val: 0.6491 time: 0.0026s
Epoch: 0098 loss_train: 0.7108 acc_train: 0.8786 loss_val: 1.0760 acc_val: 0.6515 time: 0.0027s
Epoch: 0099 loss_train: 0.7099 acc_train: 0.8714 loss_val: 1.0703 acc_val: 0.6488 time: 0.0044s
Epoch: 0100 loss_train: 0.7307 acc_train: 0.8286 loss_val: 1.0643 acc_val: 0.6515 time: 0.0051s
Optimization Finished!
Total time elapsed: 2.1347s
Test set results: loss= 1.0643 accuracy= 0.6515
```

3.4: See the attached code.

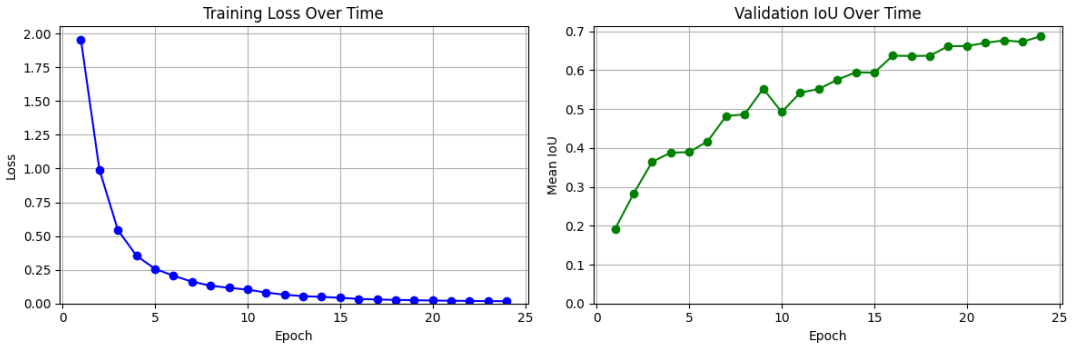
3.5: The output is shown below:

```
Epoch: 0090 loss_train: 0.9998 acc_train: 0.7571 loss_val: 1.0839 acc_val: 0.7921 time: 0.2101s
Epoch: 0091 loss_train: 0.9388 acc_train: 0.7714 loss_val: 1.0775 acc_val: 0.7936 time: 0.2103s
Epoch: 0092 loss_train: 1.0648 acc_train: 0.7214 loss_val: 1.0714 acc_val: 0.7960 time: 0.2104s
Epoch: 0093 loss_train: 0.8915 acc_train: 0.7643 loss_val: 1.0651 acc_val: 0.7979 time: 0.2103s
Epoch: 0094 loss_train: 0.9392 acc_train: 0.7786 loss_val: 1.0588 acc_val: 0.7983 time: 0.2109s
Epoch: 0095 loss_train: 0.9088 acc_train: 0.8429 loss_val: 1.0519 acc_val: 0.8010 time: 0.2106s
Epoch: 0096 loss_train: 0.9053 acc_train: 0.8000 loss_val: 1.0449 acc_val: 0.8026 time: 0.2095s
Epoch: 0097 loss_train: 0.8818 acc_train: 0.7714 loss_val: 1.0385 acc_val: 0.8037 time: 0.2113s
Epoch: 0098 loss_train: 0.8205 acc_train: 0.8786 loss_val: 1.0319 acc_val: 0.8053 time: 0.2104s
Epoch: 0099 loss_train: 0.9013 acc_train: 0.8143 loss_val: 1.0257 acc_val: 0.8057 time: 0.2112s
Epoch: 0100 loss_train: 0.9280 acc_train: 0.7429 loss_val: 1.0200 acc_val: 0.8057 time: 0.2103s
Optimization Finished!
Total time elapsed: 21.3322s
Test set results: loss= 1.0200 accuracy= 0.8057
```

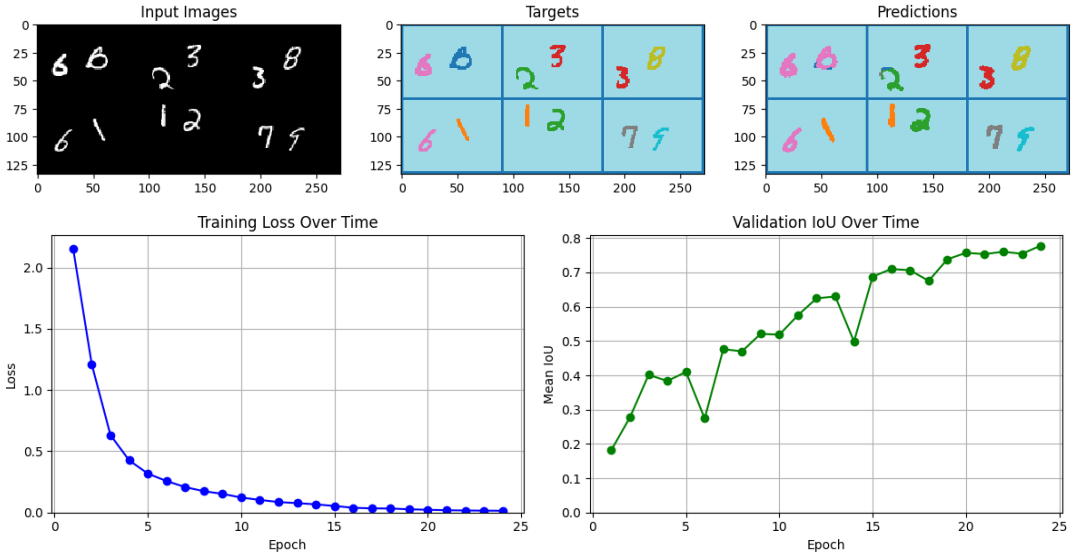
3.6: The GAT performed better than the vanilla GCN with a lower test loss (1.0200 vs. 1.0643) and a significantly higher test accuracy (0.8057 vs. 0.6515). This is likely because the attention layer allows the GAT to learn more features from the nodes, which improves its ability to generalize to different graphs.

4.1: See the attached code.

4.2: The outputs with skip connections disabled are shown below:



The outputs with skip connections enabled are shown below:



The U-Net performed better when skip connections were enabled: it had a higher mean validation IoU (around 0.78 vs. around 0.68) and its predicted segmentations resembled the targets more. This improvement is likely because skip connections help prevent a loss of information and instability in gradients.