Deep Learning 1 - Homework 2

Pedro M.P. Curvo
MSc Artificial Intelligence
University of Amsterdam
pedro.pombeiro.curvo@student.uva.nl

Part 1

Question 1.1

a)

The expression for $f_{1,1}$ is given by:

$$f_{1,1} = g_{1,1}h_{0,0} + g_{1,2}h_{0,-} + g_{2,1}h_{-,0} + g_{2,2}h_{-,-} \tag{1}$$

As we can see, we didn't use the full receptive field of the filter, since it goes beyond the image. To address this problem, we can pad the image using several techniques, such as zero-padding, reflection padding, warp padding, etc.

b)

b. i)

Type	**Dataset**	Round Accuracy (%)										**Mean**	**Std. Dev.**
		0	1	2	3	4	5	6	7	8	9		
Valid	Validation	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0000	0.0000
	Test	0.3	0.1	0.0	0.1	0.0	0.0	0.1	0.3	0.0	0.0	0.0900	0.1136
Replicate	Validation	98.6	98.1	98.8	98.5	97.1	98.8	98.2	98.0	98.5	98.5	98.3100	0.4784
	Test	92.9	92.8	88.8	94.7	95.3	91.7	96.3	94.7	95.8	95.0	93.8000	2.1629
Reflect	Validation	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0000	0.0000
	Test	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0000	0.0000
Circular	Validation	99.9	99.9	99.7	99.9	98.9	99.7	99.7	99.5	99.8	99.4	99.6400	0.2939
	Test	81.5	82.7	77.4	86.9	83.2	82.6	86.5	79.8	81.1	84.4	82.6100	2.7504
sconv	Validation	98.8	99.0	98.8	99.1	99.0	99.3	98.7	98.8	98.9	98.5	98.8900	0.2119
	Test	6.5	6.9	6.5	6.1	6.2	6.5	6.2	6.1	6.4	6.3	6.3700	0.2326
fconv	Validation	88.4	87.6	88.6	89.9	88.1	89.9	85.8	87.1	88.2	87.1	88.0700	1.1984
	Test	88.4	88.6	88.6	89.9	89.1	89.9	88.9	88.4	88.9	87.8	88.8500	0.6249

Table 1: Accuracy results for Net1

b. ii)

Looking at the samples from the dataset, the pattern for class label 0 is to have a green on the right and a red on the left. While the pattern for class label 1 is to have a red on the right and a green on the left.

The samples in the train set and the ones in the test set differ in the position of the green and red boxes, but keeping the same pattern for each class label. For example, in the README.md, we can observe that the samples for class label 0 are concentrated in the top ot the image and the samples

for class label 1 are concentrated in the bottom and in the test set, the samples for class label 0 are concentrated in the bottom and the samples for class label 1 are concentrated in the top. This means that the network will have to learn the pattern for each class label, regardless of the position of the boxes, i.e., the network will have to learn the pattern of the boxes and not the position of the boxes.

c)

c i)

The variables affecting the `conv_type` are the `padding_size` and the `pad_type`. For *valid*, *sconv* and *fconv* we have the following differences:

- *valid*: No padding is added to the image, since the size of the padding is 0
- *sconv*: The padding added is a zero padding, with size 1
- *fconv*: The padding added is a zero padding, with size 2

c ii)

If we look into the network architecture we can see that we first use a kernel size of 3 with a stride of 1 and then a stride of 2 maintaining the kernel size of 3. With this setup, the convolution operation will reduce the spatial dimensions of the feature maps. By using the *valid*, we are adding no padding to the image, which means that the output size of the convolution operation will be the smallest of the tree. This lead to the reduction of the spatial dimensions of the feature maps, leading to less spatial information being passed to the next layer. *sconv* retains slighty more spatial information, since we are adding a padding of 1 to the image, which will lead to a slight increase in the output size of the convolution operation. This will lead to a slight increase in the spatial information passed to the next layer. *fconv* retains the most spatial information, since we are adding a padding of 2 to the image, which will lead to a significant increase in the output size of the convolution operation. This will lead to a significant increase in the spatial information passed to the next layer.

With that being said, we can extrapolate that larger feature maps will keep more information that allows to distinguish between the classes in the dataset, which leads to the better performance of the model in the accuracy metric as observed. Hence, the order $\texttt{acc\_valid} < \texttt{acc\_sconv} < \texttt{acc\_fconv}$ is expected, corresponding to the amount of spatial information retained by the feature maps.

c iii)

The `reflect` padding mirrors the image along its edges, while `fconv` uses zero padding. Reflect padding can reduce abrupt transitions at the edges by reflecting nearby pixel values, but it can introduce patterns at the boundaries that do not exist in the original image. These patterns can alter the feature extraction near the edges and might reduce the ability for the model to generalize.

In this dataset, where each image has a black background with one green box and one red box, zero padding performs better as it adds zero-value borders that match the original black background, allowing the network to focus on the actual box patterns. Reflect padding, applied at all layers, can create artificial features at the edges, such as duplicating and inverting the boxes, which may harm the model's ability to learn the actual patterns, since it can mix the patterns of the boxes.

c iv)

Zero padding can dilute the information of the image by adding zero-value borders that match the original black background. While Replicate, when the boxes are near the edges, can help maintain the boxes' patterns by extending the edge pixels into the padded area, meaning we won't have so much dilutation of the information of the image. This makes the CNN "see" the boxes better while going through the layers, which can lead to better performance.

Zero padding creates a sudden change in pixel values at the image boundaries, resulting in high derivative values across the receptive fields near the edges. This can cause the CNN to confuse these abrupt changes with actual edges, particularly the edges of the green and red boxes in the dataset that we are trying to predict. The network may then interpret the zero padding as an actual feature, leading to worse performance in edge detection, hence worse performance in identifying the boxes.

Replicate padding, on the other hand, extends the edge pixels into the padded area, ensuring a smoother transition, since the pixel will have the same value as the nearest edge pixel. This helps preserve the local context near the edges by maintaining some statistical properties of the image, such as the derivatives. This is beneficial when detecting the edges of the boxes, since replicate padding prevents the network from confusing the padded area with the actual edges. As a result, the replicate padding improves the model's performance in detecting the true edges of the boxes, leading to a better accuracy.

d)

d i)

Type	**Dataset**	Round Accuracy (%)										**Mean**	**Std. Dev.**
		0	1	2	3	4	5	6	7	8	9		
Valid	Validation	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0000	0.0000
	Test	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0000	0.0000
Replicate	Validation	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0000	0.0000
	Test	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0000	0.0000
Reflect	Validation	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0000	0.0000
	Test	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0000	0.0000
Circular	Validation	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0000	0.0000
	Test	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0000	0.0000
sconv	Validation	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0000	0.0000
	Test	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0000	0.0000
fconv	Validation	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0000	0.0000
	Test	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0000	0.0000

Table 2: Accuracy results for Net2

d ii)

As we can observe by the results, the model is able to achieve 100% accuracy on the validation set for all padding types, but it fails to generalize to the test set, achieving 0% accuracy, performing worse in every padding type in `Net2` with lower accuracies than `Net1`.

d iii)

In the `Net2`, we added an additional linear layer and we remove the AdaptiveMaxPool2d layer. The additional linear layer increases exponentially the number of parameters in the model, which can lead to overfitting. If we look into the `utils.py` file, we can see the the test dataset is shifted by 16 pixels relative to the validation and training dataset. This can mean that the model `Net2` is overfitting to the training and validation dataset by memorizing the overall positions of the boxes and, since they are shifted in the test dataset, the model is unable to generalize to the test dataset, leading to the 0% accuracy. This makes sense if we also consider that the model is achieving 100% accuracy in the validation set, while getting 0% accuracy in the test set in all padding types.

Question 1.2

a)

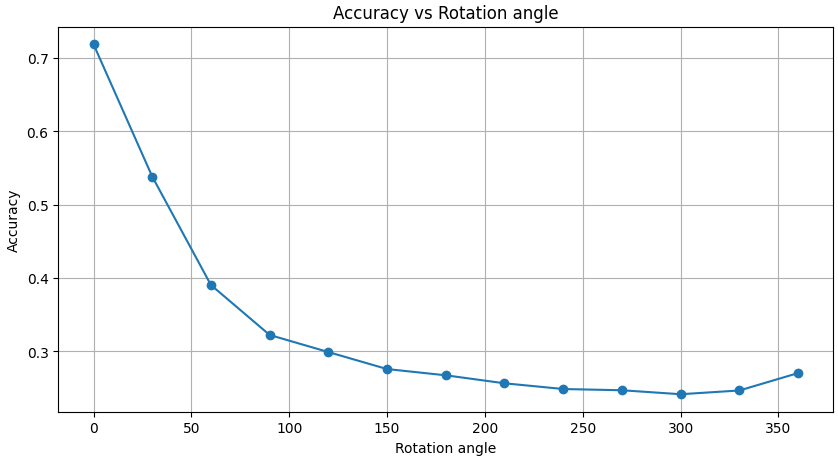


Figure 1: Inference respect to angle of rotation

CNNs are not naturally rotationally invariant because the filters they learn during training are sensitive to the orientation of features in the images. When an image is rotated, the learned filters may no longer align with the features in the new orientation. This misalignment reduces the model's ability to recognize patterns, leading to a drop in accuracy. The convolutional layers detect local patterns such as edges and textures in a fixed direction, and rotating the image changes the spatial relationships between these features, making it harder for the network to match them correctly.

If the model is not trained on rotated images, it cannot learn to recognize objects or patterns from different angles. The performance of the model is highly dependent on the orientation of the test images, and it is likely to perform worse when presented with images that were not seen during training. Even though the model might perform better at certain rotation angles (e.g., 90° or 180°), accuracy typically drops as the angle deviates from those orientations. This demonstrates that CNNs require rotation-invariant training data or additional techniques like data augmentation to improve their ability to recognize rotated images.

b)

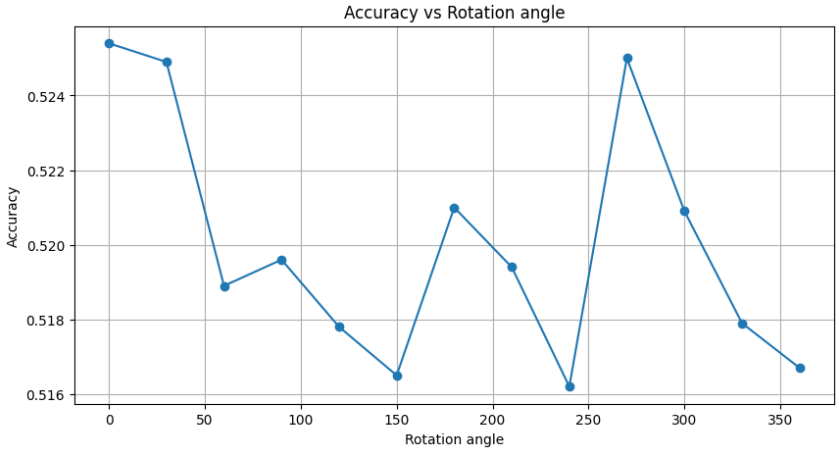


Figure 2: Inference respect to angle of rotation with data augmentation

When the model is trained on an augmented dataset with random rotations, it is exposed to a wide variety of orientations for the objects in the images. This allows the model to learn features that are not dependent on a specific orientation. By incorporating rotated versions of the images during training, the model learns to recognize patterns and objects regardless of their angle. This helps the network develop rotational invariance, as the filters it learns become capable of detecting the same features from different perspectives.

As a result, the model becomes more robust to rotation, and its accuracy stabilizes across different angles during inference. With this training approach, the network does not rely on a fixed alignment of the features, and can instead generalize across a range of orientations. This improved performance across various rotation angles indicates that the network has learned to identify the objects or patterns in the images without being influenced by their orientation, thereby overcoming the limitations of rotational sensitivity found in networks trained only on images with fixed orientations.

Part 2

Question 2.1

a)

- Long input sequences can be challenging for Transformers due to the quadratic complexity of the self-attention mechanism, that is, in self-attention, each token attends to every other token in the sequence, hence we have a $O(n^2)$ complexity that can be seen by multiplying the Q, K and V matrices. This can lead to both memory and computational issues, which are worse when n is large (larger sentences).
- One way to solve this problem is by using some kind of sparse attention mechanism that focus on a subset of local or key tokens. With this we can reduce the complexity of the self-attention mechanism. For example, the Reformer model uses locality-sensitive hashing to achieve a $O(n \log n)$ complexity.

b)

CNNs have a local receptive field that allows them to capture local patterns in the input data and which is done by the convolution operation between the input and a kernel filter. One the other hand, Transformers have a global receptive field that allows them to capture long-range dependencies in the input data, which is done by the self-attention mechanism, since it allows each token to attend to every other token in the sequence.

c)

The scaling factor $\frac{1}{\sqrt{d_k}}$ is used to stabilize the gradients during training. The dot product of the Q and K matrices is scaled by $\frac{1}{\sqrt{d_k}}$ to prevent the dot products from becoming too large when the dimensionality of the keys d_k is high. If it were the case that some values were too large, the softmax function would produce gradients that are too small, leading to slower learning or vanishing gradients which would harm the gradient descent process.

d)

If we assume we have the same total computation cost then the multiple attentions heads are beneficial because they allow to capture different types of dependencies and patterns simultaneously. Each attention head can focus on different parts of the input sequence, like in CNNs we have several filters. It can also help mitigate some bias that can originate in some heads due to the dominance of some tokens and can help improve generalization because, by learning multiple sets of attention parameters, the model can be more robust to different tasks.

Question 2.2

CODE

Question 2.3

a to b)

CODE

c)

The MLP head is wide and not deep for several factors. In one hand, we want to keep independence of tokens in the MLP, meaning we don't need a deep network to capture the dependencies between the tokens. With a wide network, we are basically representing each token in a higher-dimensional space, which can help to capture more complex patterns independently. If we employed a deep network, we would be capturing dependencies between tokens, which is not necessary in this case, since the self-attention mechanism already does it. In the other hand, with a wide network we can apply a parallel computation, that is, splitting the model across multiple GPUs, which can lead to

more computational efficiency. By keeping the MLP shallow, we also reduce the probability of overfitting and can keep a more stable training process.

Question 2.4

a)

If we do not use a positional embedding, the Transformer model will not be able to distinguish the order of the tokens in the input sequence. This is because the self-attention mechanism treats all tokens as independent, and without positional information, meaning is permutation invariant. For example, without positional embeddings, the model would treat the sentences "The dog bit the man" and "The man bit the dog" as the same, which is not the desired behavior. We need to have a positional embedding for a sequential structure, specially for tasks like syntax, etc... which are characteristics of LLMs.

b)

By using absolute position embeddings, that is, by assigning, for example, 1 to the first token, 2 to the second token, and so on, we are restricting the model to work within the predefined sequence length. This can be problematic if we then want to apply the model to sequences that are longer than the ones seen during training. Besides that, we can have, for example, different types of inputs, where lengths can vary, for example, documents, translation tasks, etc... In these cases, the model would not be able to generalize. Adding to this, we also want the model to have a sense of proximity between tokens, which is not achieved by absolute position embeddings. For example, we might want the model to understand that the tokes "The" and "dog" appear closer than "man" and "dog".

By using relative position embeddings, we can overcome these limitations. Relative position embeddings allow the model to learn the relative distances between tokens, which can be more useful in tasks where the absolute position of tokens is not as important. Adding to this, the model can then extrapolate to different new sequences lengths and can even retain the context in large sequences, for example, in resuming tasks, where we have large documents.

Question 2.5

CODE

Question 2.6

CODE

Question 2.7

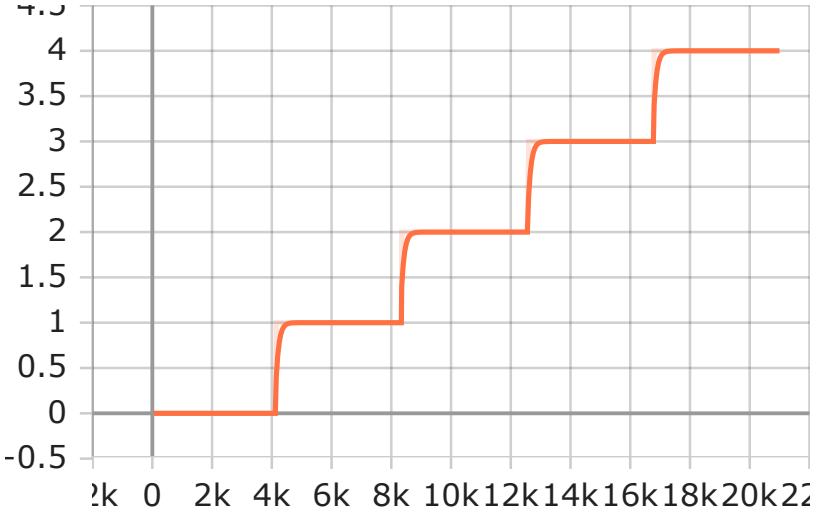
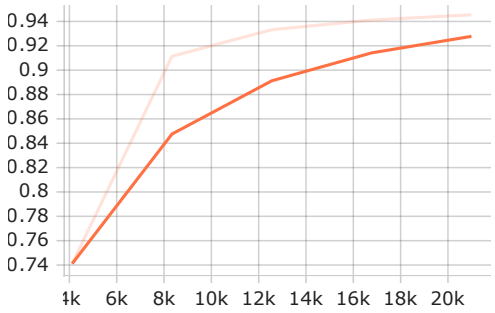
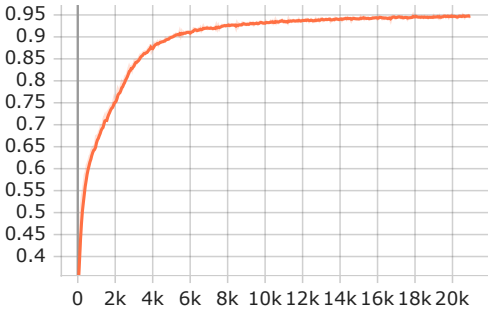


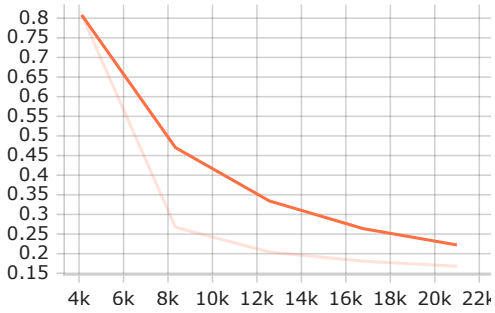
Figure 3: Epochs per Step



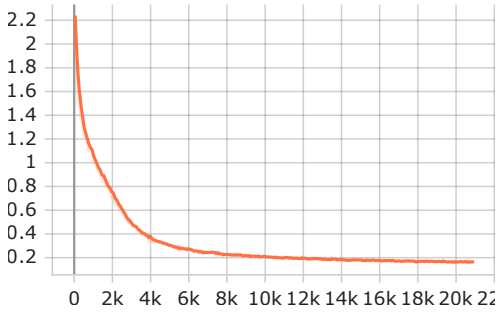
(a) Train Accuracy per Epoch



(b) Train Accuracy per Step



(a) Train Loss per Epoch



(b) Train Loss per Step

Question 2.8

a)

CODE TODO: REVISE

b)

To test the model on **Grimms' Fairy Tales** I used some prompts to test scenarios, characters and objects:

- **Prompt:** *'Hansel said to'*
- Model: Hansel said to Gretel: 'We shall soon find the way,' but they did not find it. They walked the whole night and all the next day too from morning till evening, but they did not get out of the forest
- **Prompt:** *'The wolf said to'*
- Model: The wolf said to the Little Red-Cap, and then he said: 'See, Little Red-Cap, here is a piece of cake and a bottle of wine; take them to your grandmother, she is ill and weak, and they will do her finger.'
- **Prompt:** *'There was once'*
- Model: There was once a man called Frederick: he had a wife whose name was Catherine, and they had not long been married. One day Frederick said. 'Kate! I am going to work in the fields'
- **Prompt:** *'Rapunzel had'*
- Model: Rapunzel had magnificent long hair, fine as spun gold, and when she heard the voice of the enchantress she unfastened her braided tresses

Question 2.9

a)

CODE TODO: REVISE

b)

The benefits of Flash-Attention come from the nature of Load between Memory and Compute. So basically, we have HBM (High Bandwidth Memory) that is used to store, write and read the matrices Q, K and V and intermediate operations. HBM has a large memory, but is slow to compute things. Meanwhile, we have SRAM (Static Random Access Memory) that is used to compute the intermediate results of the attention mechanism, since SRAM is fast to compute things. In a normal attention mechanism, the HBM loads the matrices Q and K into the SRAM, which then computes the product QK^T. This product is then written back to the HBM, and the HBM loads the QK^T into the SRAM to compute the softmax. Which then is written back to the HBM and loaded again into the SRAM with the V matrix to compute the final attention output, which is then written back to the HBM. This process of loads and writes between the HBM and SRAM is slow and inefficient. Flash Attention solves this problem by using a Flash Memory, that is, it fuses all those operations into a single operation, a so called fused-kernel, which is then loaded into the SRAM with the Q, K and V matrices. Then everything is computed in the SRAM at once and written back to the HBM. This reduces the number of loads and writes between the HBM and SRAM, which makes the process faster and more efficient. This in conjuction with the parallel computation of the multiple attention heads, allows the model to be more efficient and faster. Indeed, with flash-attention we can have a $O(n)$ complexity in memory, which is faster than the $O(n^2)$ complexity of the self-attention mechanism.

Part 3

Question 3.1

a)

For the first graph we have:

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

For the second graph we have:

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

b)

By squaring the first adjacency matrix we have:

$[[3,1,1,1,0],[1,2,1,1,1],[1,1,3,0,1],[1,1,0,1,0],[0,1,1,0,1]]$

When we square the matrix $A \to A^2$, we are counting the number of paths of length 2 between each pair of nodes, that is, A^2_{ij} is the number of paths of length 2 between nodes i and j. For example the entry $A^2_{12} = 1$ means that there is 1 path of length 2 between nodes 1 and 2, which is the path $1 \to 3 \to 2$.

More generally, the entry A^n_{uv} is the sum of the number of paths of length n between nodes u and v in the graph.

Note: In the diagonals we have the number of paths of length 2 that start and end in the same node, for example, if node 1 and 5 are connected this represents also a path of length 2, because we can go from node 1 to node 5 and then back to node 1. This is only true in the case of undirected graphs.

c)

The Laplacian matrix is defined as $L = D - A$, where D is the degree matrix and A is the adjacency matrix. The degree matrix is a diagonal matrix where the diagonal entries are the sum of the weights of the edges connected to each node.

For the second graph we have that D is:

$[[1,0,0,0,0,0],[0,3,0,0,0,0],[0,0,2,0,0,0],[0,0,0,3,0,0],[0,0,0,0,3,0],[0,0,0,0,0,2]]$

And the Laplacian matrix is:

$[[1,-1,0,0,0,0],[-1,3,-1,0,-1,0],[0,-1,2,0,-1,0],[0,0,-1,3,-1,-1],[0,-1,0,-1,3,-1],[0,0,0,-1,-1,2]]$

By applying the Laplacian matrix to an input matrix X, we have that $F = L \cdot X$. If we consider the way matrix multiplication works, we see that F is the result of the span of the row space of X by the columns of L. The columns of L have larger value if D is larger, that is, if the node is more connected to other nodes, which means that if a node has high degree, it will be more influenced by its neighbors because the corresponding row in L will have larger values. We can conclude, in this case, that the nodes that change the most are nodes 2, 4 and 5, since they have higher degree (3).

NEEDS REVISION

Question 3.2

a)

The bias term defined as $B^{(l)}h_v^{(l)}$ allows to introduce a self-loop to each node in the graph, that is, it allows a node v to retain some of its previous information independent of the information from its neighbors. This is important, since otherwise, the updated embedding $h_v^{(l+1)}$ would be completely dependent on the information from the neighbors as seen in the term $\sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|}$, meaning it would dilute its unique characteristics.

b)

Correlating with equation 6, we can see that that equation can be interpreted as simple instance of message passing, that is:

- Message: The term $\frac{h_u^{(l)}}{|N(v)|}$ can be seen as the message passed from node u to node v. Hence, the $W^{(l)} \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|}$ term can be seen as the aggregation of the messages from all neighbors of node v where $W^{(l)}$ parametrizes how these messages are combined.

- Update: The term $B^{(l)} h_v^{(l)}$ and the activation function σ can be seen as the update function that incorporates the message from the neighbors and the node's own information to update the node's embedding.

Question 3.3

a)

Defining $D \in \mathbb{R}^{N \times N}$ as the degree matrix of the graph, we have that $D_{ii} = |N_v|$. We also define $A \in \mathbb{R}^{N \times N}$ as the adjacency matrix of the graph with $A_{ij} = 1$ if there is an edge between nodes i and j, and 0 otherwise. And define $H^{(l)} \in \mathbb{R}^{N \times D}$ as the matrix of node embeddings at layer l.

From equation 6, if we stack them for all nodes, we have that the update rule for all nodes can be written as:

$$\begin{bmatrix} h_v^{(l+1)} \\ \vdots \\ h_N^{(l+1)} \end{bmatrix} = \begin{bmatrix} \sigma\left(W^{(l)} \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B^{(l)} h_v^{(l)}\right) \\ \vdots \\ \sigma\left(W^{(l)} \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B^{(l)} h_N^{(l)}\right) \end{bmatrix}$$

Now, instead of creating a higher dimensional tensor with a repeated weight matrix $W^{(l)}$ and a repeated bias matrix $B^{(l)}$, we can rewrite with a single weight matrix $W^{(l)}$ and a single bias matrix $B^{(l)}$ by transposing it in the other side, the same way we do it in batch dimensions in neural networks. We can then rewrite the equation as:

$$H^{(l+1)} = \sigma\left(\left(\begin{bmatrix} \sum_{u \in N(1)} \frac{h_u^{(l)}}{|N(1)|} \\ \vdots \\ \sum_{u \in N(N)} \frac{h_u^{(l)}}{|N(N)|} \end{bmatrix}\right) W^{(l)^T} + \begin{bmatrix} h_1^{(l)} \\ \vdots \\ h_N^{(l)} \end{bmatrix} B^{(l)^T}\right)$$

Now, we can separate the terms $\frac{1}{|N(v)|}$ by using a matrix multiplication and knowing that, since D is a diagonal matrix, then its inverse is also a diagonal matrix with the inverse of the diagonal entries. We can then rewrite the equation as:

$$
\begin{aligned}
H^{(l+1)} &= \sigma\left(\left(\begin{bmatrix} \frac{1}{|N(1)|} & 0 & \cdots & 0 \\ 0 & \frac{1}{|N(2)|} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{|N(N)|} \end{bmatrix} \begin{bmatrix} \sum_{u \in N(1)} h_u^{(l)} \\ \vdots \\ \sum_{u \in N(N)} h_u^{(l)} \end{bmatrix}\right) W^{(l)^T} + \begin{bmatrix} h_1^{(l)} \\ \vdots \\ h_N^{(l)} \end{bmatrix} B^{(l)^T}\right) \\
&= \sigma\left(\left(D^{-1} \begin{bmatrix} \sum_{u \in N(1)} h_u^{(l)} \\ \vdots \\ \sum_{u \in N(N)} h_u^{(l)} \end{bmatrix}\right) W^{(l)^T} + H^{(l)} B^{(l)^T}\right)
\end{aligned}
$$

Now, looking at the terms $\sum_{u \in N(v)} h_u^{(l)}$, we can see that we are summing the embeddings of the neighbors of node v. Which is the same as summing over all nodes with a delta that is equal to one if there is an edge between nodes v and u. This delta can be represented by the adjacency matrix A, meaning that that sum can be rewritten as $\sum_v A_{uv} h_u^{(l)}$.

Finally, we have that the update rule can be written as:

$$
\begin{aligned}
H^{(l+1)} &= \sigma\left(\left(D^{-1} \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1N} \\ A_{21} & A_{22} & \cdots & A_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N1} & A_{N2} & \cdots & A_{NN} \end{bmatrix} \begin{bmatrix} h_1^{(l)} \\ \vdots \\ h_N^{(l)} \end{bmatrix}\right) W^{(l)^T} + \begin{bmatrix} h_1^{(l)} \\ \vdots \\ h_N^{(l)} \end{bmatrix} B^{(l)^T}\right) \\
&= \sigma\left(D^{-1} A H^{(l)} W^{(l)^T} + H^{(l)} B^{(l)^T}\right)
\end{aligned}
$$

Hence, the final update rule for the graph convolutional network is:

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

Where σ is applied row-wise.

Checking the dimensions, we have that $D^{-1} A H^{(l)} W^{(l)^T}$ is a product of a $N \times N$ matrix with a $N \times N$ with a $N \times D$ with a $D \times (D+1)$, which results in a $N \times (D+1)$ matrix. The term $H^{(l)} B^{(l)^T}$ is a product of a $N \times D$ with a $D \times (D+1)$, which results in a $N \times (D+1)$ matrix. Hence, the final output is a $N \times (D+1)$ matrix, which is the same as the output matrix $H^{(l+1)}$.

CHECK DIMENSIONS, SPECIALLY ON W AND B

b)

Median aggregation is a type of aggregation that is used to combine the information from the neighbors of a node in a graph by taking the median of the embeddings of the neighbors. The problem is that Median aggregation is not a linear operation, which means that it cannot be represented as a matrix multiplication. Hence, the matrix form is not directly applicable to the median aggregation.

Question 3.4

a)

In GCNS, the weights that are used for aggregation are shared across all nodes in the graph, that is, they are uniform or fixed for all nodes. This means that all neighbors of a node are aggregated in the

same way equally. In GATs, the the aggregation uses weights given by attention coefficients that are learned during training, which means that the weights are different for each node and each neighbor. This allows the model to learn assign different importance to different neighbors.

b)

As explained in the previous question, the weights in GCNs are shared across all nodes meaning all neighbors contribute equally. This means the model is less robust to noise since it cannot suppress the influence of some noisy neighbors. In GATs, the model can learn to assign different importance to different neighbors, which allows it to suppress the influence of noisy neighbors and focus on the most relevant ones. This makes GATs more robust to noise. With this being said, is expected that GCNs will have a lower learned embedding quality than GATs.

c)

A standard transformer applied to a sentence can be interpreted as a GNN if we consider the following structure:

- **Nodes:** Each token in the sentence is a node in the graph. The embeddings of the tokens are the node features.
- **Edges:** The edges in the graph are the connections between the tokens in the sentence, that is, the edges are defined by the attention mechanism. The edge weights are the attention coefficients.
- **Graph Structure:** In a standard transformer, we are attending to all tokens in the sentence, which means that the graph is fully connected, that is, each node is connected to every other node.

With this, we can make the following analogies:

- The message passage in GNNs is equivalent to the attention mechanism in transformers, since the attention mechanism allows each token to attend to every other token in the sequence.
- The attention scores tell us the strength of the relationship between the tokens, which is similar to the edge weights in GNNs.

d)

Two advantages of GATs over transformers are:

- **Computational Efficiency**: Standard transformers compute the attention for all pairs of tokens in the sequence, meaning it has a global attention mechanism. GATs compute attention only for the neighbors of each node, which makes them more computationally efficient and more scalable to larger graphs.
- **Interpretability and Graph Structure**: GATs are designed from the ground-up to work on graphs and capture the graph structure, where transformers require explicit encoding of the graph structure, which is not intuitive for most problems. Besides that, GATs allow us to have a better interpretability of the model, since we can see the relationships between the nodes, while in transformers we are not aware of this.