# [Mid Report] Graph based Neural Architectures

***Team Number: 25 | Team Linear Aggression***

**Timeline and Milestones**

| Timeline | Milestones | Status |
|---|---|---|
| November 1st - 7th | Literature Review | Completed |
| November 7th | Get project proposal approved and submit on Moodle. | Completed |
| November 7th - 12th | Creation of pipeline, preparing dataset and implementing GCN | Completed |
| November 12th - 17th | Implementing GAT, training all models and creating appropriate visualizations | Completed |
| November 17th - 20th | Mid Project Evaluations | Completed |
| November 20th - 21st | Implementing improvements and suggestions based on mid evals | In Progress |
| November 21st - 30th | Implementing multiple different applications of NLP using GAT and GCNs | In Progress |
| December 1st - 4th | Final Presentation | In Progress |
| December 4th | Final Report submission on Moodle | In Progress |

## Introduction

As mentioned in the proposal, our primary goal is to learn about neural architectures used on graph-based data. Text data can be represented as linear graphs. In contrast, images can be described as rectangular graphs. Though ample neural architecture exists that performs well on those graphs, more complicated graph structures require GNNs.

Our project tackles two conceptual problems: first, is there a better way to represent the irregular graph-like structures when it comes to neural architectures, and second, can we create an architecture such that we could "attend" to specific nodes more than the others.

We explore Graph Convolutional Networks (GCNs), a simple graph-based neural architecture with multiple layers. It modifies the node's features based on the adjacency matrix as we go through numerous layers via learnable matrices. Next, we try to answer our second question by building Graph Attention Network (GAT) on top of GCNs to focus on certain edges more, which may or may not help the model. We also visualise the said attention so we can understand the process happening.

## Dataset

Graph Neural Network may or may not work differently than a traditional neural network based on the dataset. Sometimes you may have multiple samples of various graphs representing the main thing, in which case we can easily do the standard 80-20 train-test split without any loss of information. Such a learning mechanism is known as inductive learning.

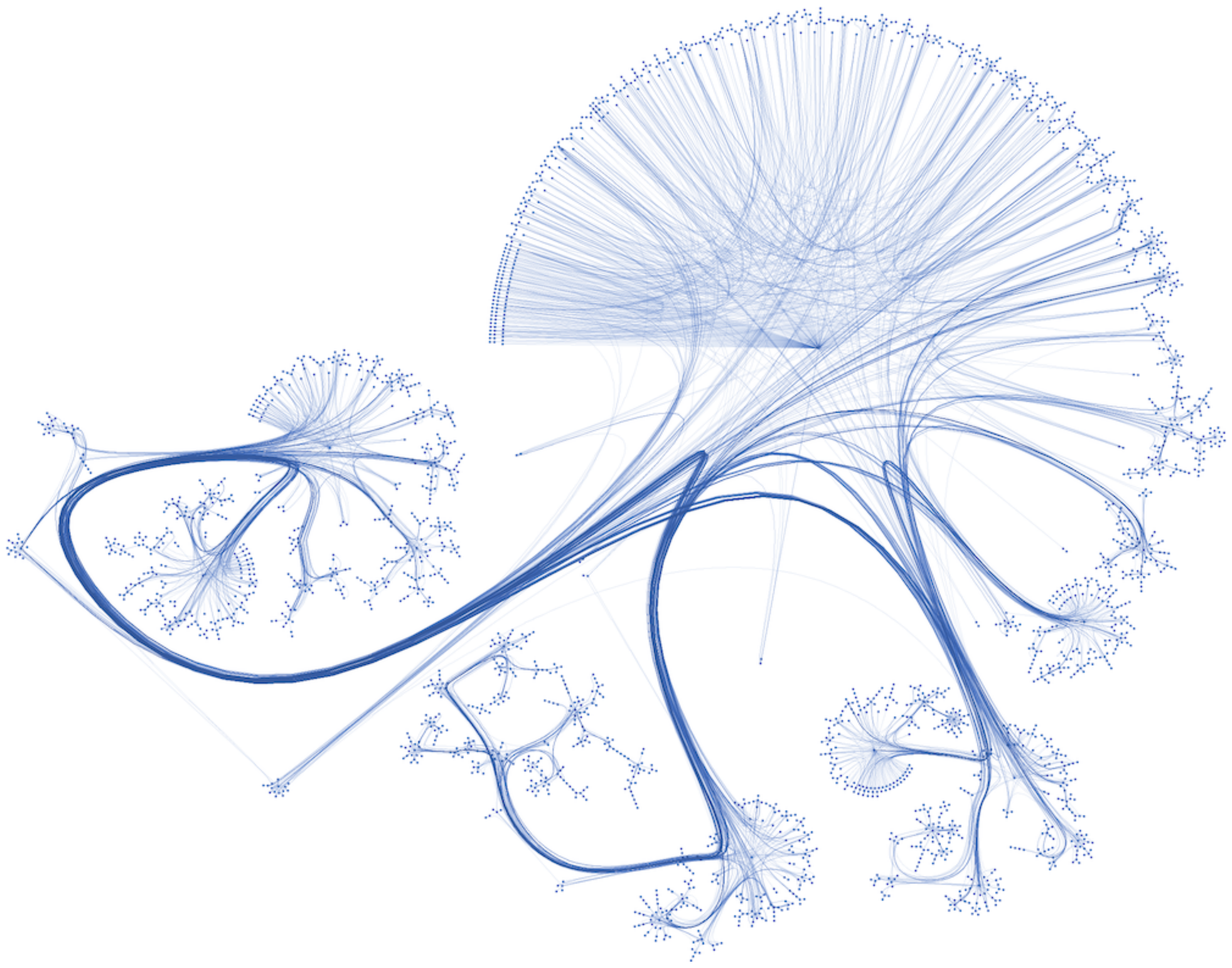
Conversely, in some cases, we have a massive graph as the dataset, which we cannot split without loss of information. In such a case, we send the whole data to training (we show the model all the data) but have masks defined. These masks are used so that the backpropagation happens only to specifically selected notes, while other node features, weights and biases are not updated. Such a learning mechanism is called transductive learning.

We have opted to explore transductive learning via the Cora dataset. The Cora dataset represents multiple research publications from various fields as a network of citations. Here, two publications have an edge if one has cited the other (for simplicity, edges are assumed to be non-directional). Each node has a feature vector assigned to it, a binary value based on one hot encoding of words existing in the publication. The words are selected after removing the stop words from all the corpora and ignoring words with less than ten frequencies.

Details about the dataset:

- **2708** Nodes (each node is a research paper)

- **1433** Unique words per document
- **5278** Edges (citation connections)
- **7 Classes:** Neural Networks, Probabilistic Methods, Genetic Algorithms, Theory, Case Based, Reinforcement Learning and Rule Learning.
- Class distribution: { 2: 818, 3: 426, 1: 418,     6: 351, 0: 298, 4: 217, 5: 180}

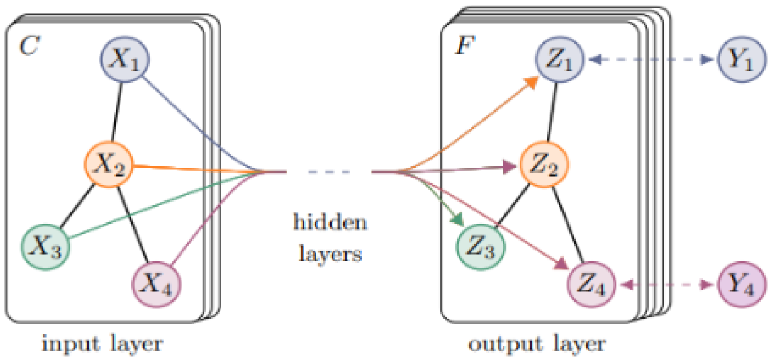


This is a super cool visualisation of the Cora Dataset done using yEd Live and a GraphML file. Here we can see how interconnected the nodes truly are to different nodes.

## Understanding Graphs

A normal graph in this case is assumed to be $G(V, E)$ where, G represents the Graph, V represents the vertices (or nodes), and E represents Edges connecting the vertices. Here, the edges are assumed to be undirected for the sake of simplicity.

The adjacency matrix is represented as $A \in \mathbb{R}^{N\times N}$, where $N$ is the number of nodes in the graphs. Adjacency matrix is, naturally, symmetrical in nature. To add self loops to the adjacency matrix we perform a small trick $A = A + I$



$\underline{G(V,E)}$

$X_{N\times C}$ : C channels for each N nodes $\xrightarrow{f}$ $Z_{N\times F}$ F output features

$A_{N\times N}$ : Adj. matrix (can be sparse)

non linear

$$H^{l+1} = f(H^l, A)$$

$$H^0 = X, \quad H^L = Z$$

**Issues:**

A will not consider the node features itself, unless there are self loops

A can scale the feature vectors arbitrarily

**Resultions:**

That is the reason we added self loops, $A = A + I$

We the symmetrically normalise the matrix $A = D^{\frac{-1}{2}} A D^{\frac{-1}{2}}$

$$H^{l+1} = \overset{\substack{\text{non} \\ \text{linear} \\ \downarrow}}{f}(H^l, A)$$

$$H^0 = X \;,\; H^L = Z$$

$$H^{l+1} = f(H^l, A) = \sigma\left(\hat{D}^{-1/2} \hat{A} \hat{D}^{-1/2} H^l W\right)$$

$$\hat{A} = A + I \quad (\text{for self loops})$$

$$\hat{D} = \text{diagonal node degree of mat. } \hat{A}$$

Implementation of GCN is available on:
https://colab.research.google.com/drive/1zuJKz4PnoXqt_-hE-8pCx7LaSXVdKP6f#scrollTo=8KivNXrSJIai

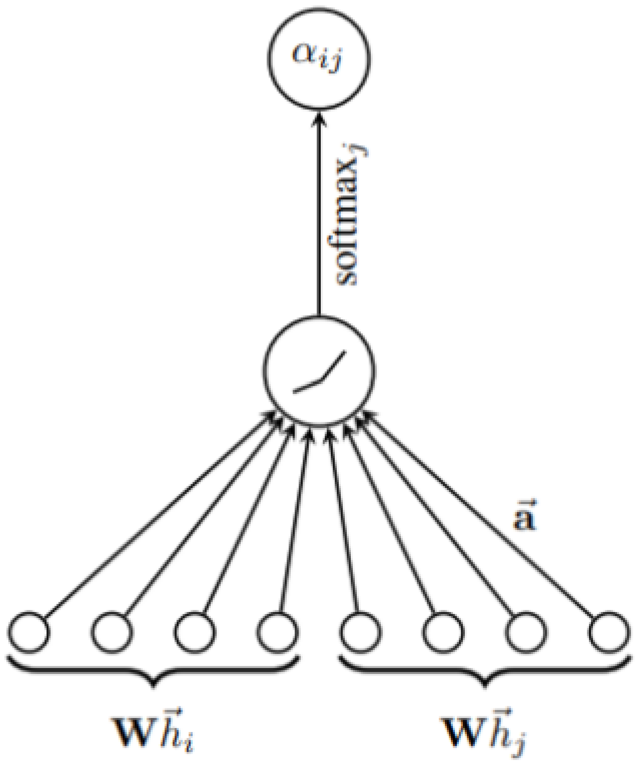
## Attention

Attention is essential to what we are trying to do, by concentrating on certain nodes more we are trying to find out if there are some nodes which are more important than the others. Attention is represented by $a$ , and is a single layer feed-forward neural network. It takes in 2 node features as input and gives attention between the two nodes as the output.

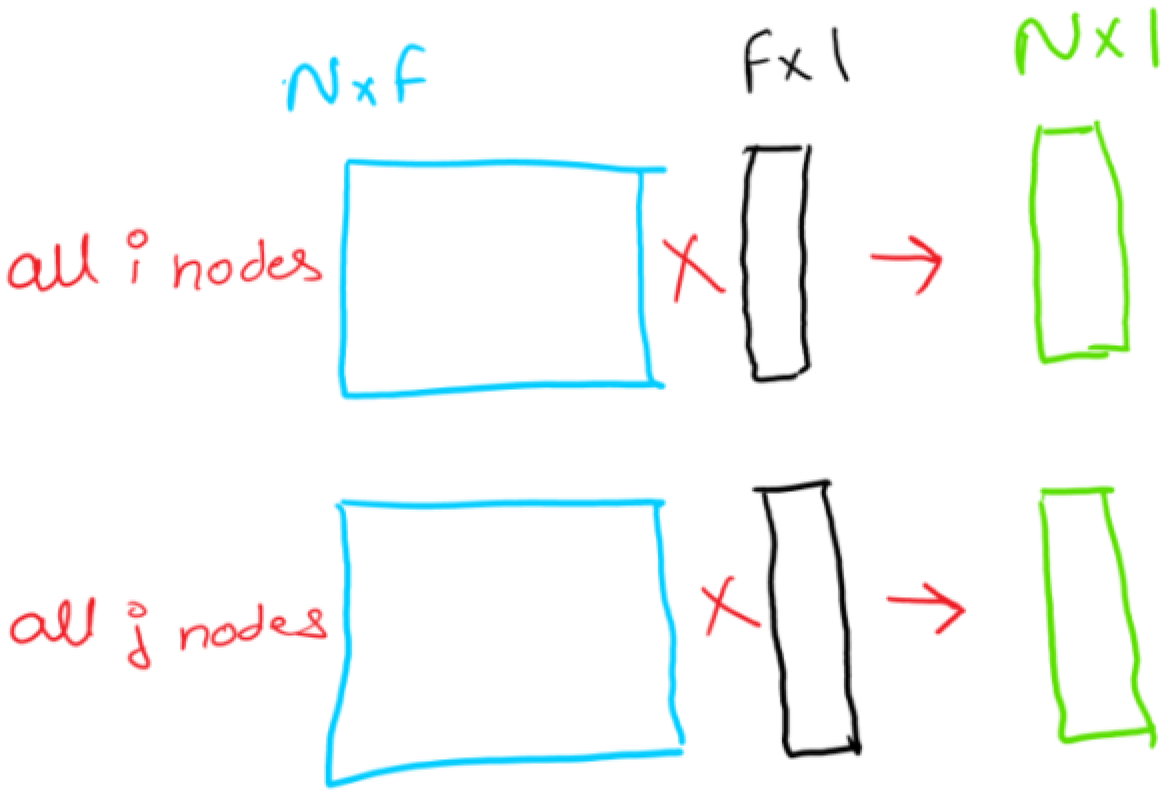
$$\underset{\nearrow}{e_{ij}} = a(\underset{\uparrow}{W\vec{h}_1}, \underset{\uparrow}{W\vec{h}_2})$$

attention given to edge connecting node $i$ to node $j$

node feature vector



One could also assume $a$ to be a learnable weight matrix which needs to be multiplied with the features. Once we get the softmax, we apply LeakyReLU (with default parameters) as the activation function.



This is how we calculate the attention, we multiple all the nodes at once with each other and hence calculate attention for all nodes together. We then sum up the last two vectors as $a + a.T$ and get a $N \times N$ matrix as our attention matrix, which we then multiply with our adjacency matrix $A$ to get a symmetric matrix, since attention should only exist between nodes which are connected.

Implementation of GAT: https://colab.research.google.com/drive/14ThhHtUjVfOwhxbNUl923FC_GZtQ7BZ5?usp=sharing#scrollTo=lM4sgiscrflR

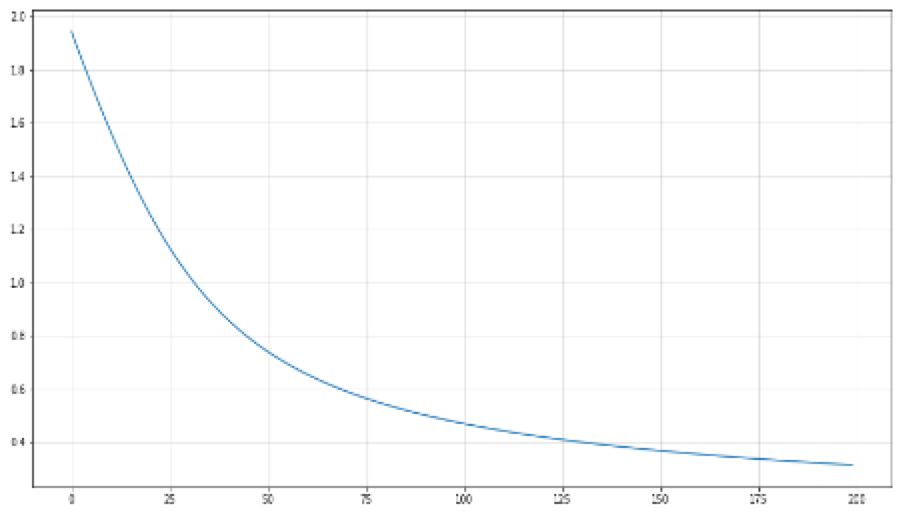
## Results

**Hyperparameters:**

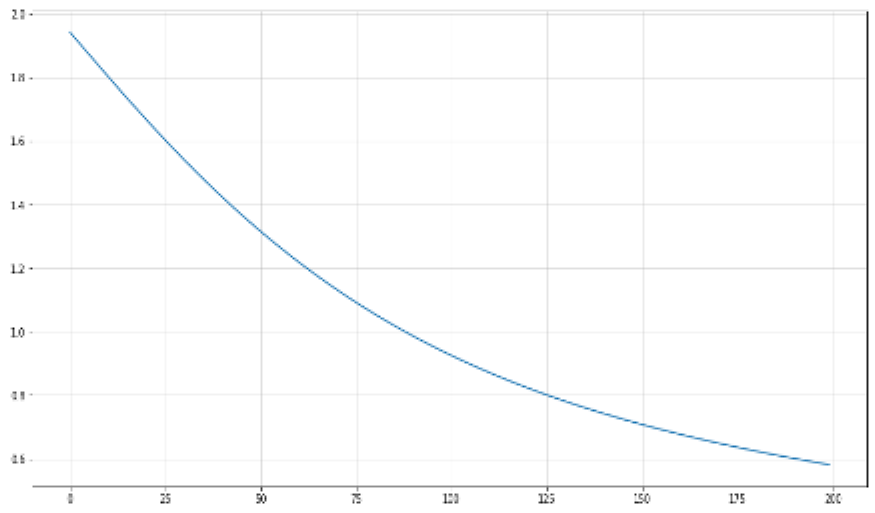
We tried 2 things with both the datasets:

- Hyperparameters as mentioned in the paper:
    - Training Nodes: 140 (20 from each label)
    - Test Nodes: 1000 (randomly selected)
    - Epochs: 200
    - Layers: 2
    - Hidden Features: 32
- Hyperparameters which gave best results:
    - Training Nodes: 1647 (atmax 250 from each label)
    - Test Nodes: 1061 (remaining)
    - Hidden Features: 1-24

**Loss:**



GCN



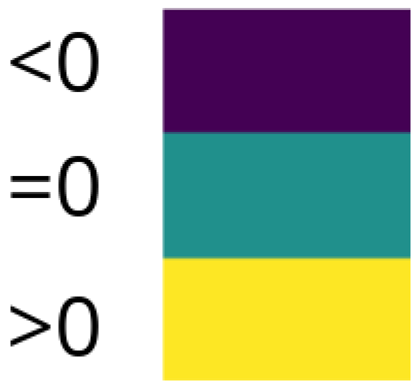
GAT

**Interpretations:**

As we can see, the loss for GCN is lower (and decreases faster) compared to that of GAT for the same number of training data and epochs. There are a couple of plausible explanations for this phenomenon:

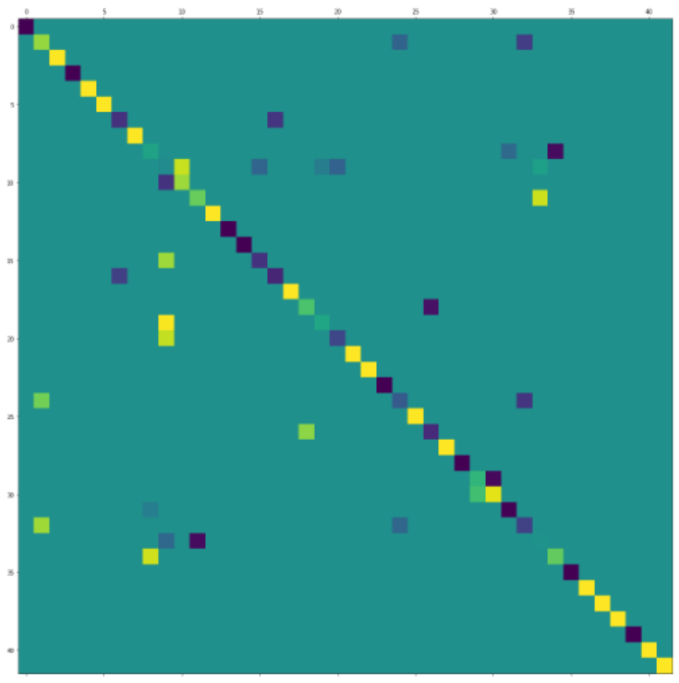
- We observed with our experiments that as the number of layers of GAT increases the loss decreases as well (we could do an experiment on how the loss changes as we increase layers and compare between GCN and GAT)
- Another explanation is that since initially a matrix is randomized, probably GAT would perform better as we increase the number of epochs
- We observe that test accuracy for **GCN: 76.5%**, and **GAT: 82.38%**, hence it could also be that GCN overfits hence gives lesser loss.

**Understanding Attention:**

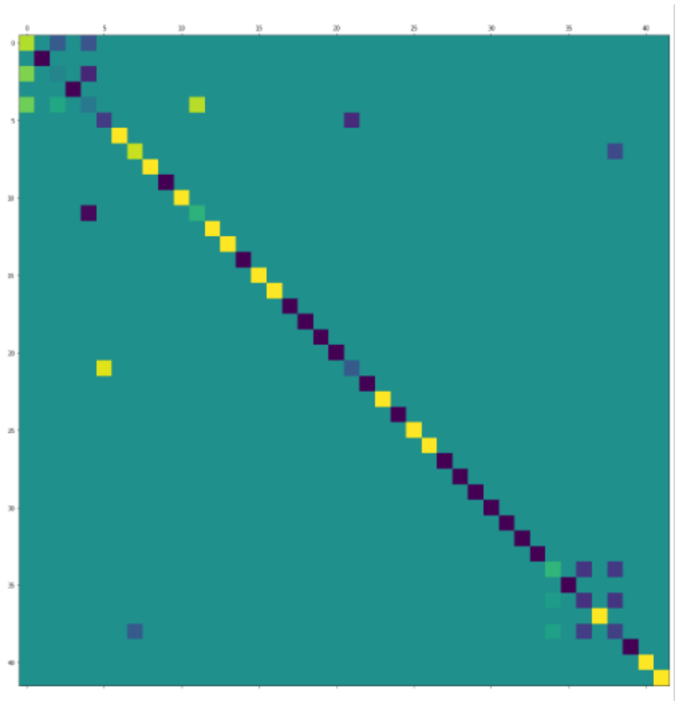


We compare the **difference** between the **first layer** of the model and the **last layer** (first & second layer in our case). We do so to see how much the attention matrix changes in a single iteration, later, we plan to do this for more number of layers. Since a 2708x2708 matrix visualisation is not easy for the human eye to understand or make sense of, we take the first 42 nodes for each class, and hence have 7 42x42 matrices. We expect to see high changes on self loops.

Class 0



Class 2



Images of other classes are available in the presentation as well as in the github link (which is private as of now)

In the execution part of attention, we have initially kept the attention matrix $a$ randomized, hence, even in the losses we see that GAT has higher losses initially which then goes lower. Since we are plotting change in the matrices, this does not play a huge role here. Highest change (whether negative or positive) occur in self edges, which means that the nodes most effecting a node’s class is most probably itself

As seen in the images of attention plotted for same labels, all attention matrices are symmetric in nature. This can be explained since they are being element-wise multiplied by adjacency matrix, hence we expect to see a symmetrical matrix. Now, one would notice that though these images are indeed symmetrical matrices, but the colours are different. One simple hypothesis explaining this observation is that since we are just observing first-degree connections, some nodes might be more important than others.

## Future Work

While we have presented the implementation of Graph Attention Networks, we aim to utilise the same for certain NLP tasks. We believe the evaluation of the model can be seen explicitly while applying the model to complete these tasks. We already know that various GNNs are used for tasks such as Text Classification, Relation Extraction etc. We plan to explore how GATs can be used to perform similar tasks. Furthermore, considering publicly available textual datasets (such as Wikipedia, news articles etc.), if these can be described by graphs, we can perform inference on the same to compare the aforementioned NLP application across different models. The performance metric can be compared to existing baseline models of other GNNs on one specified task.