In [3]:

```
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:80% !important; }</style>"))

!export PATH=/Library/TeX/texbin:$PATH

import os
print(os.environ['PATH'])
```

```
/home/shane/anaconda3/envs/gnn_env2/bin:/home/shane/anaconda3/condabin:/usr/l
ocal/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/
games:/snap/bin
```

Flow

- information extraction
    - multi-doc
- machine comprehension
    - question answering
        - multihop qa
            - intermediate questions
- RNN's
- transformers
- graph based approaches to mh qa
    - propagation steps
        - compare same weights per layer vs new weights each layer
- text to graph construction
    - NER and corefs
    - sequential mention connections
    - Document Structure
    - AMR
- node encoding
    - contextual embeddings
    - co+self attention summarisation
    - transformer based encoding
    - query aware context encoding
- gnn's
    - history
        - spectral v non spectral
    - gating
    - attention

Abstract

Work on GNN architectures has converged on the highly general message passing pattern. GNN's with attention based message aggregation can now be thought of as generalisations of the highly successful transformer architecture. The SOTA model for the popular Multihop QA dataset Wikihop is a masked variant of the transformer network which is conceptually very similar to an attention based GNN. There is thus reason to beleive that injecting some of the insights gained from GNN based approaches to Multihop QA into the longformer architecure may yeild performance gains. More generally, graph structuring of text may be a powerful tool to overcoming the poor memory scaling of the vanilla transformer architecture.

Introduction

Problem

Information Extraction (IE) refers to the general task of retrieving automatically-extracted structured-information from a collection of unstructured/semi-structured data given a query. When this semi-structured data is Natural Language Text the task is refered to as Machine Reading Comprehension (MRC) which refers to a machines ability to read and interpret both a natural language corpus and query as well as to use some form of reasoning to present an answer. MRC is a longstanding goal in the field of Natural Language Processing (NLP). One can think of the Google search engine as an example of an MRC system.

Question Answering (QA) is an approach to training/testing an MRC system's abilty to interpret Natural Language Text using a collection of (context, query, answer) tuples. Multi-hop QA (MHQA) regards datasets where the answers to given questions can only be figured by integrating distinct, and possibly nonadjacent, pieces of evidence together [cite]. For instance, answering the question: ‘what country is Table Mountain in?’ given the evidence: ‘Table Mountain is in Cape Town’ and ‘Cape Town is a city in South Africa’. A system capable of MHQA demonstrates the ability to perform multi-step reasoning, and must leverage knowledge about the relationships between entities.

MHQA systems are well suited to address the challenging problem of Cross-Document MRC (CD-MRC), which refers to MRC problems where the context may be multiple semi-related/unrelated documents, and the query can only be answered by integrating information which may be scattered across the different documents. To illustrate the value of such a system - consider again the Google search engine, which you provide with a query and which uses all of its cataloged webpages as context. Google uses its patented algorithm to rank webpages in order of relevance to your query. More recently, Google will reach into the most relevant webpage, and pluck out the paragraph it thinks is most relevant to you. A more sophisticated CD-MRC system however would be able to take the most relevant web pages, read all of their contents, interpret the contained semmantics, and incorporate all the read information into a single answer. It is important to note, that since such a system incorporates information from different sources, the presented answer is likely to be novel, not neccesarily having been contained in any of the source documents. Another powerful usecase of CD-MRC is the automatic integration of insights offered by different yet related academic research papers for faster and more powerful research analysis.

Datasets

Wikihop

The QAngaroo [cite] dataset is composed of the Wikihop and Medhop dataset, each of which are constructed heuristically such that context passages are extracted from different yet related documents such as Wikihop pages, and concatenated. The question/answer pairs are chosen such that each question only has 1 possible answer, and this question should be unanswerable given any single piece of information, instead multiple pieces of evidence, possibly found in different passages should be integrated together using reasoning principals to arrive at the answer. The QAngaroo dataset structures QA as multiple choice, where each candidate is an entity found in the context.

Approaches to MRC

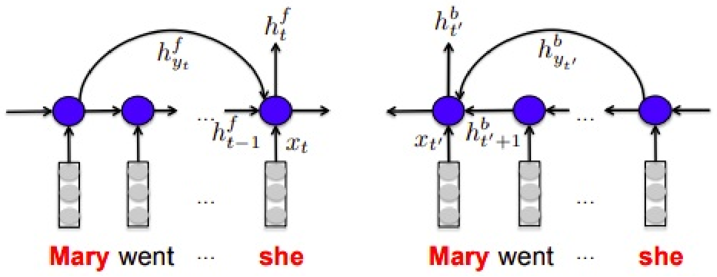
While older MRC systems often relied heavily on heuristic based extraction rules such as text matching, more modern approaches are focused around the use of Deep Neural Networks (DNN's). DNN's have risen to be the defacto solution to almost all high level NLP problems [cite?].

RNN based approaches

Until recently, RNN based approaches to NLP have been the gold standard. RNN's natually model sequences, and mirror how humans read text ie: start to finish, one word at a time. Vanilla RNN's quickly ran into a problem called the Long-term dependancy problem [cite], this is the problem that arrises when long sequences are fed in, resulting in the early parts having been mostly forgotten by the end of the sequence.

To remedy this, gating functions/forget gates were introduced to RNN's whereby a learned function was able to pick which information was important and should thereby be remembered for longer, and which information was irrelavent to the task at hand - and thus could be forgotten immediately. The two most used variants are the LSTM [cite] and the GRU [cite]. In an ideal situation, this gating method could be used to selectively filter out unimportant information, leaving only important info in the state memory, however it is not always possible to immediately know what info is or isn't important without global context. This is especially true in multihop QA where it may be impossible to recognise an intermediate fact as important without having read another specific fact which may not have been enccountered yet.

[Dhingra et al. (2018)] introducced the Coref-GRU (C-GRU) Layer, which acted as a skip connection over the sequential tokens which are fed into a vanilla RNN. The skip connections were formed by connecting entity mentions and their coreferences as labeled by an external Coreference Resolution (CR) tool. The system learned two separate weight functions, one for propagating information accross a sequential edge, as in vanilla RNN's, and another weight function for propagating information across the C-GRU which only props info from entity mentions to their coreferences. These two RNN layers were used in tandem to improve the performance on multihop QA datasets Wikihop, LAMBADA and the bAbi AI tasks [validate lam and babi types]. Traversing these new skip chains likely reduced the burden on state retention by allowing more direct state communication between mentions of entities



Transformer based approaches

The transformer network was introduced in late 2017 by [Vaswani el al 2017], and quickly was shown to outperform RNN's in virtually all NLP tasks [cite tranformer good]. The paper titled "attention is all you need" demonstrated that dropping RNN and Convolutional features - which were popular at the time - in favor of an attention only model increased performance while also decreasing training time. The transformer works by performing self attention over the entire given input sequence, thus a sequence of length L gets an L*L attention matrix computed, thus the memory required for the attention to operate on a sequence scales with O(L^2). This memory scaling severly limited the length of encodable sequences, with the popular BERT implementation having a max sequence length of 512 tokens, well below what is needed to encode whole documents.

The Longformer [Beltagy et al 2020] is a recent approach to overcoming the memory scaling limitations of the vanilla transformer, it uses a task specific mask generation heuristic to ignore parts of the attention matrix to save on memory. For QA the longformer proposes using a sliding window for local attention accross the long context sequence, as well as a global attention between the query sequence and the context sequence. This means that there is coattentive flow between all context and query tokens, however context tokens only connect to eachother if they are found within a windowlengths distance of eachother in the context sequence. The Longformer has claimed the #1 spot on the wikihop leaderboard as of the writing of this document.

GNN based approaches

Modern GNN's impose no topological constraints on graph data and are highly generalised. They naturally handle cycles and richly connected graphs [cite]. They are also capable of being heterogeneous whereby different edge types use different message passing functions [Ming et al 2019]. Switching out RNN's for GNN's allows for processing of essentially any type of graph structured text.

A recent set of papers have been released which aimed to use the Graph Neual Network (GNN) as well as text-graph structuring of contexts and queries for use in MHQA. A GNN is simply a NN which operates over a graph as input data. Interest in GNN's initially arose - in part - as an attemtpt to generalise the convolution operations [cite early GNN paper] which had given rise to success in the computer vision subfield of DNN's [cite cnns good]. Early GNN's had severe limitations such as having to train separate update filters for nodes of each degree of connectivity [cite], or only being able to operate on graphs of a fixed topology [cite]. More recently GNN's have converged on the message passing pattern, whereby at each step - each node collects a set of messages from its neighbours, aggregates those messages, and then updates its own state. Following this pattern with an aggreagtion function which is invariant to the number of neighbouring nodes allows for a single GNN to operate on variable graphs, with no topological restrictions.

Multihop QA

MHQA requires integration of a handful relevant facts out of a pool of many more irrelevant facts. To do this a system needs to learn to recognise when a piece of information is relevant to the given query. For example given F1 = "Table Mountain is in Cape Town" and F2 = "Cape Town is a city in South Africa", and a query Q = "Which country is Table Mountain in?" A system may recognise that F1 is important as it contains the entity "Table Mountain" which is found in the query. However when considering F2 in isolation, despite being vital to answer the question, it is not immediately clear that this fact is relevant to Q. Only once F1 has been interpretted can the relevance of F2 be recognised. This leads into the idea of intermediate questions which is discussed in [cite QRN]. Given only Q and F1 a reasonable question to addd would be Q1 = "What country is Cape Town in?", now when F2 is encountered, it can be immediately identified as important to the query. Each hop in an n-hop reasoning process updates the intermediate question with new information, leading to a new intermediate question.

Consider a new set of relevant facts: {F1 = "Table Mountain is in Cape Town", F2 = "Cape Town is in the same country as Joburg", F3 = "Joburg is in South Africa"} and the same query Q as above, it should be noted that to derive the correct intermediate questions to allow for the integraion of these facts, they must be read in a particular order. Incorporating, or recognising either F2 or F3 without knowing F1 is vastly more difficult that incorporating F1 first. Similarly, incorporating F3 without having interpretted F1 and F2 is difficult

In general for an $n$-hop question, where $n+1$ Facts must be incorporated there may be instances where certain facts can be permuted without obscuring the reasoning process, however there may be as few as 1 ordering in which the facts can be injested for satisfactory reasoning. In QA problems where facts $\{F_i | i \epsilon \{1..n+1\}\}$ can only be injested in one order ie F1,F2,..,$F_{n+1}$ we will refer to these facts as a fact sequence.

Each fact $F_i$ in a reasoning chain contains a set of related entities $E_F(i)$. A common form for traversal of a reasoning chain typically follows starting in fact F1, and moving to another fact which contains any of the same entities [cite qa reasoning chains]. So if F1 contains entities ${a,b,c}$ and F2 contains ${c, d, e}$ then a reasoning chain may flow from F1 to F2 since they share entity $c$.

Transformer Model

The Transformer Model was developed by [Vaswani el al 2017] - under the employ of Google - to be a powerful new approach to sequence-to-sequence (Seq2Seq) models in NLP. The model primarily utilises Dot-product attention and uses the (query, key, value) pattern. The pattern is as follows: Given a sequence of queries as well as a sequence of (Key,Value) pairs the algorithm will attend each query to each key, and use that (query,key) pair to scale the key's value by an importance score. While iterating theough the sequences - all 3 input tensors are passed through there own linear layer, ie there is a Q,K and V linear layer. Then the output from the Q and K linear layers are dot-producted with eachother to output an importance score. For each Q value, the importance scores for all K values are normalised using a Softmax activation, this ensures that for any given Q value, there is a fixed amount of importance that can be assigned to the evaluated K values. Finally these normalised importance scores are multiplied into the V values and summed to obtain the attention layers output. Effectively, for each Q value, a weighted sum of V values is outputted, where the weights are determined as a function of both Q and K.

Note that since the (Q,K) pairs are dot producted with eachother, the sequence elements must match dimensionally. The paper mentions another attention mechanism which performs similarly albeit with a higher memory burden called additive attention. In this method, the importance scores are calculated as the output of a linear layer where the inputs are the Q,K values concattenated together, this method conceptually allows for performing attention over sequences with elements of differing dimensionality/ differing feature count.

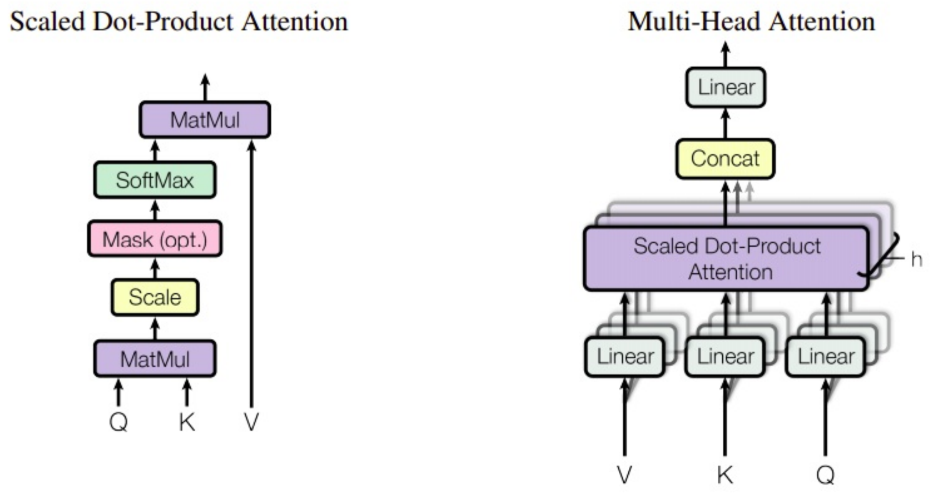


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

The model also makes use of the encoder decoder pattern which is common in Seq2Seq models. In this pattern, the model takes in an input sequence X, and encodes it into an intermediate sequence Z, the decoder then takes in sequence Z and outputs sequence $Y^{'}$ which is trained to match the target sequence Y. The model is trained using a self supervision training regiment whereby it aims to predict the next token in the output sequence Y given the full input sequence X and a partial target sequence. In particular the input and target sequences are sentences of equivilant meaning in different natural languages such as English and German. To combine the output of the encoder Z and the partial target sequence $\{Y_j | j<i\}$, a coattentive layer is used, where the queries are elements from the self-attended partial target sequence, and the key,value pairs are the elements from the output of the encoder block.

To accomodate this training regiment the decoder takes in a masked version of the tagret sequence Y such that when predicting token $Y_i^{'}$, it is given $\{Y_j | j<i\}$ tokens only.

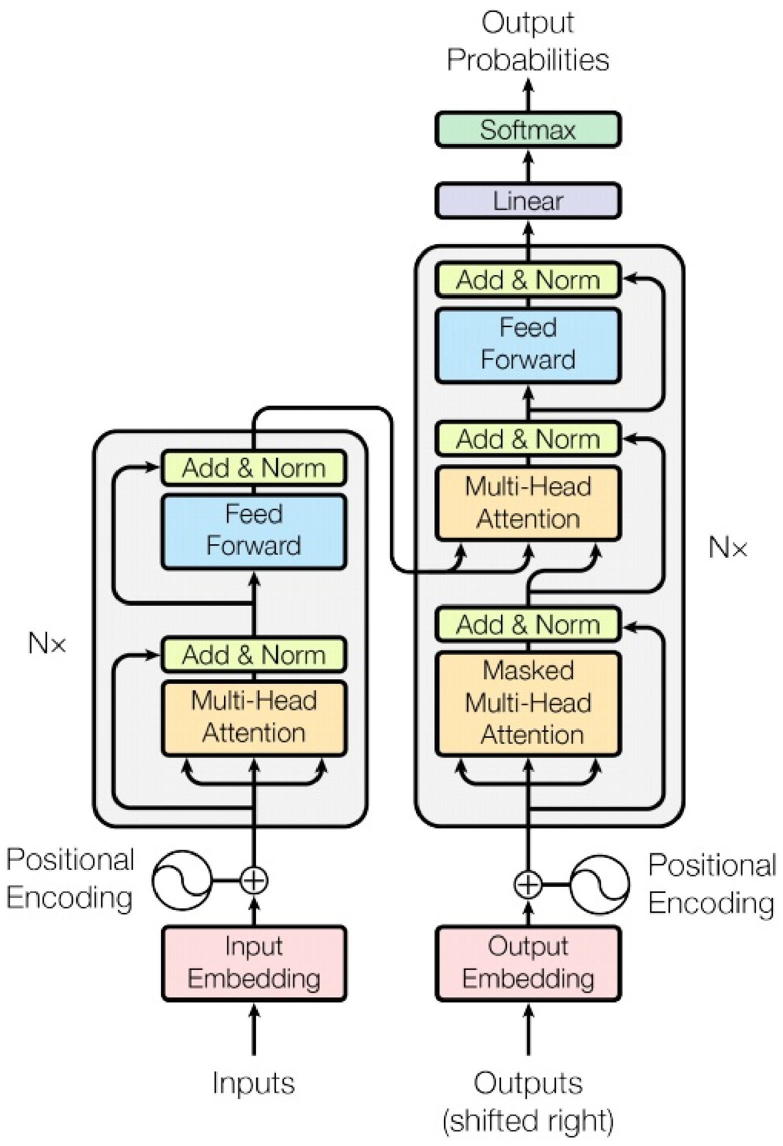


Figure 1: The Transformer - model architecture.

It should be noted that the model as described above is incapable of leveraging information about the ordering of the elements in the given sequences. Since the ordering of words in Natural Language is particulaly important, the model has an addition feature called positional encoding, which injects positional information into the elements of the input sequences so that the model may learn to use this information.

The transformer also utilises multiheaded attention which is essentially a way of scoring the importance of token pairs along multiple independant dimensions. For an n-headed attention layer, n attention scores are calculated for each Q,K pair, and n attended sequences are outputted and concattenated along the feature dimension. The paper found that swapping out a single high dimensional attention head for 8 lower dimensional attention heads increased performance while keeping computation demands constant.

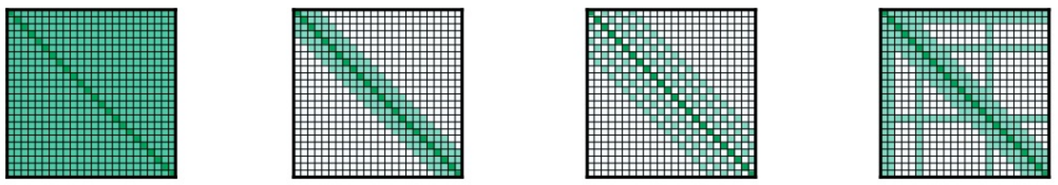
Longformer

The recently proposed Longformer model [Beltagy et al 2020] addresses the memory scaling issues which are present in vanilla transformer models by using task-specific heuristic rules to mask the attention matrix used such that the total number of (Q,K) pairs scales linearly with the length of the input sequence, not quadtrically as in the vanilla transformer. In the context of an attention matrix used for self attention - partial attention is when an element only attends over a subset of the other elements of the sequence. Global/full attention is where an element attends over every other element in the sequence.

An early workaround to the memory scaling issue was breaking the sequence up into chunks, fully self attending each chunk, and then applying some sort of recombination technique such as concattenation or a more complex combination model. The longformer uses a more elegant form of chunking called sliding window attention. This is where a pair of tokens are allowed to attend over eachother only if their distance in the text is less than some threshold. This allows for full attentive flow in local areas, such as sentences, or given a sufficiently large window - even inside of full passages. [Beltagy et al 2020] found it useful to use small window sizes in early layers, and increase the window size in subsequent layers. The authors claim this allows the early layers to capture low level local representations at a low computational cost, while allowing later layers to capture high level relational representations.

Using only a sliding window does not allow for global attentive flow however. This means that information in 2 passages which are sufficiently far away cannot communicate directly inside a single attention layer. To address this the longformer adds in a task specific global attention heuristic. For text classification, a single output token is given global attention, meaning that it is able to directly communicate with every other token. For QA the query sequence is appended to the context sequence, and all of the query tokens are given global attention. Since the query sequence is updated each attentive step as a function of itself and all context tokens, the longformer architecture as propsed by [Beltagy et al 2020] implicitly supports the concept of intermediate queries.

Note that stacking multiple attentive layers allows indirect communication between elements of a sequence which are not directly connected in the attention matrix. In the case where at least one element of a sequence has global attention, all other elements are able to communicate through this element indirectly. In step 1, a non-global element e_1 may communicate with the globally accessible element g, and in step 2, g can communicate state from e_1 to another non-global element e_2 which is not directly connected to e_1. While this allows for full communication among sequence elements, it is unlikely this communication is as powerful as the communication which can occur between direclty connected elements.



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Figure 2: Comparing the full self-attention pattern and the configuration of attention patterns in our Longformer.

Thinking of the masked adjacency matrix used in the Longformer architecture as a graph adjacency matrix may be useful in that it may allow for the transfering of insights gained from text graph structuring approaches to QA - into the transformer domain.

GNN Architectures

[history of spectral v non - build up to message passing pattern] The original goal of early GNN architectures was to automatically generate graph encodings which were aware of graph topology, and possibly node features [cite early GNNs]. Early approaches to this involved using random walk statistics and adjacency-matrix factorization-based learning objectives, as well as spectral clustering [cite]

The message pasing pattern defines how node states should be updated in a way which is dependant on their features, as well as topology. First nodes collect and transform the states from each neighbouring node (Message), then they Aggregate these messages, finally they Update their states as a function of their current state, and the aggregated messages from neighbours. Thus the pattern is simply 3 phases Message, Aggregate, Update.
$$ \Large x_i^k = \gamma^k (x_i^{k-1} , \biguplus_{j \epsilon N(i)} \phi^k (x_i^{k-1}, x_j^{k-1}, e_{i,j})) $$

where x_i^k denotes the state of node i at layer k, $N(i)$ is the set of neighbour indices for node i. In the case of featureless edges - e is a graph adjacency matrix, otherwise e is the edge feature matrix.

ϕ^k is the message function at layer k, which transforms a neighbour nodes state before being sent along an edge, this transformation can be a function of the neighbour-nodes current state x_j^{k-1}, as well as the receiving-nodes current state x_i^{k-1}, and optionally the feature matrix of the travered edge $e_{i,j}$. A simple message function could be a single Linear layer and activation.

$\biguplus$ is the aggregation function, which receives a transformed node state from each of node x_i^k's neighbours $N(i)$ - it then combines that variable number of messages into a single message, this could simply be the sum function, or more complexely an attention based weighted sum.

Finally γ^k denotes the update function, which takes in the receiving nodes current state x_i^{k-1} and the aggregated neighbour node messages from $\biguplus$ and decides the nodes next state x_i^k

GraphSAGE (SAmple and aggreGatE)

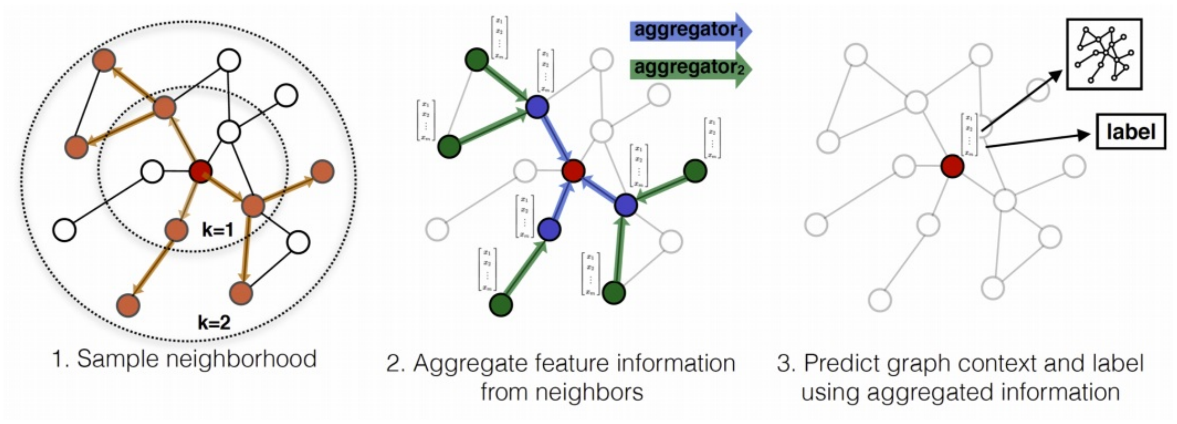

k=1
k=2
1. Sample neighborhood
aggregator1
aggregator2
2. Aggregate feature information from neighbors
label
3. Predict graph context and label using aggregated information

Figure 1: Visual illustration of the GraphSAGE sample and aggregate approach.

Gating

Gating is a technique which operates in the Update phase of message passing. The general form of the update function γ is $ \Large x_i^k = \gamma^k (x_i^{k-1} , agg)$ where agg is the aggregated messages from neighbours ie the output of the Aggregate layer. The trivial update function is $\gamma^k (x_i^{k-1} , agg) = agg$. This is the Update function used by GAT, which is described in the next section. A gating mechanism, in the context of GNNs, regulates how much of the agg state propagates to the nodes next state x_i^k [De Cao et al 2018]. Thus gating in essence is any Update function which transforms and returns the agg state as a function of the nodes current state. While this is very general, there are a few common patterns such as using a GRU-like update gate [GGNN etc], or an LSTM-like gating [Song et al 2018]. [De Cao et al 2018] used an even simpler gating method where a gate vector is produced as $\Large a_i = F(x_i^{k-1}, agg)$ with F as a learnable function. The Update function then goes $\Large \gamma^k (x_i^{k-1} , agg) = a_i \bigodot agg + (1 - a_i) \bigodot x_i^{k-1}$ where $\bigodot$ is element-wise multiplication.

or $\gamma^k (x_i^{k-1} , agg) = F(x_i^{k-1} || agg)$ where $||$ is concattenation and F is a trainable linear layer. ?? is this gating

Graph Attention

[Velickovic et al 2018] introduced the Graph Attention Network (GAT) as an attempt to apply the highly successful attention mechanisms to the GNN pattern. Specifically the GAT proposed Uses a linear layer to predict an importance score for each message being passed during a step. The messages are then aggregated by using the importance scores to perform a weighted sum over each message for a particular node. This type of attention which uses a linear layer to compute importance scores is refered to by [Vaswani el al 2017] in the paper "Attention is all you need" as additive attention. [Vaswani el al 2017] use Dot-product attention instead, they state that these two approaches are similar in performance, but that dot-product attention is faster and more memory efficient than additive attention. This warrants trying to swap out additive attention for dot-product attention in the GAT. The attention method principally operates in the Aggregate phase of the message passing pattern.

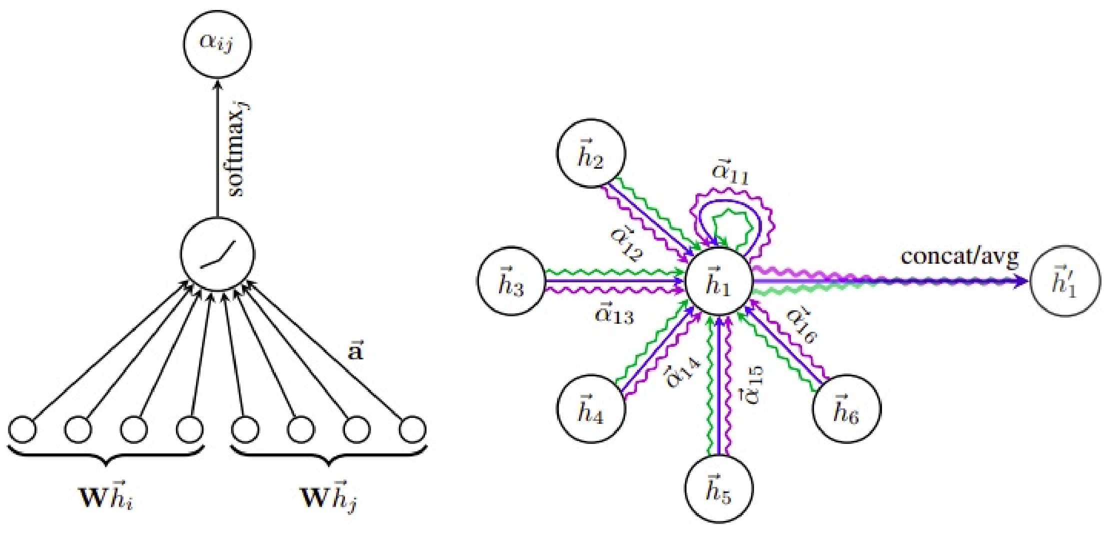


Figure 1: **Left:** The attention mechanism $a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$ employed by our model, parametrized by a weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2F'}$, applying a LeakyReLU activation. **Right:** An illustration of multi-head attention (with $K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}'_1$.

Thinking of the longformer as a GAT, a difference stands out which is that the GAT decouples graph topology and the DNN, whereas longformer couples topology/connectivity and layers, meaning each layer of a longformer may operate on a different adjacency matrix, while GNN based approaches typically operate on a fixed graph throughout the forward pass. While there are methods for simplifying topology during a forward pass by removing nodes/edges or aggregating nodes [cite edgePool], there does not seem to be any work on adaptively adding in new connections, as is the case in the longformer, where the sliding window used increases in size as the data passes through each successive layer. This increasing window size is analogous to adding in new graph connections during the forward pass.

Since gating and attention operate on different layers of the GNN - Update and Aggregate - the two methods could be used together.

Graph based Multihop QA

[compare EGCN,HDE to GSPR architecture with reference to tranform(sum) vs sum(transform)]

Text to Graph Contruction

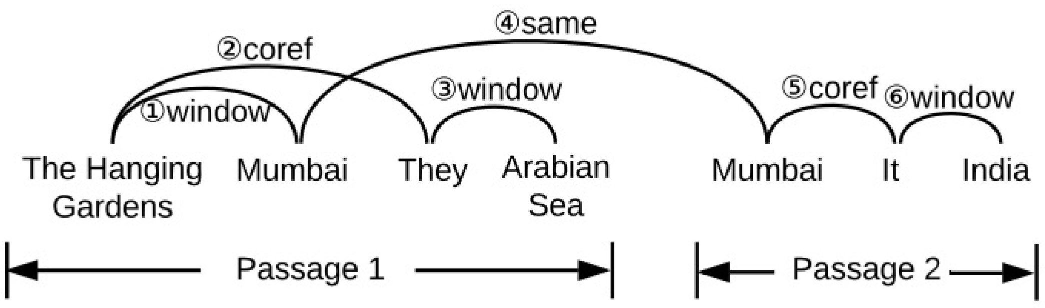
There are many potential ways a body of text can be converted into graph structure, we will be focusing on 3 recent papers which all evaluate on Wikihop. The method used by EGCN [De Cao et al 2018], GSPR [Song et al 2018], HDE [Ming et al 2019] principally used the entity-relation pattern whereby named entities are extracted from the text using an external Named Entity Recognition (NER) tool. Coreferences for each named entity are also extracted using an external tool. Each mention of the extracted entities as well as their coreferences are encoded via a contextual word embedder such as BERT [cite] or ELMO [cite], and each representation vector is placed in its own node. In all 3 works, these entity nodes are connected via the COREF and SAME edge, whereby SAME connects two identical mentions of the same entity, possibly accross documents. The COREF edge connects a mention to its coreference.

All 3 papers use query aware context embeddings. The process for generating these embeddings involves convolving all of the contextual word embeddings with a query representation such that the final entity embeddings represent information about their entities, the context of their entities, and the relation between the entity and the starting query representation. Forcing the query into the node embeddings may dilute its meaning, and without a centralised location for query information - learning support for intermediate queries in this way may be difficult.

The divergence in their graph structuring will be adressed below:

Graph structured passage representation (GSPR)

GSPR extracts entity and coref nodes for every named entity in the given text. In addition to the SAME and COREF edges, GSPR adds in the WINDOW edge, which connects entity nodes which are sequential in the text. This connection may connect two mentions of the same entity, and may also connect mentions of different entities.



The graph construction process is independant of the query or answer candidates, and thus is suited for non-mulitple-choice QA

Entity-Graph Convolutional Network (EGCN)

EGCN extracts nodes for entities found in the query or candidates list only, reducing the number of nodes drastically. This simplification adds to the information loss during the graph construction process. EGCN (66.4) outperforms GSPR (65.4) on Wikihop, possibly due to the more distilled graph representation, but more likely due to a superior message function ϕ.

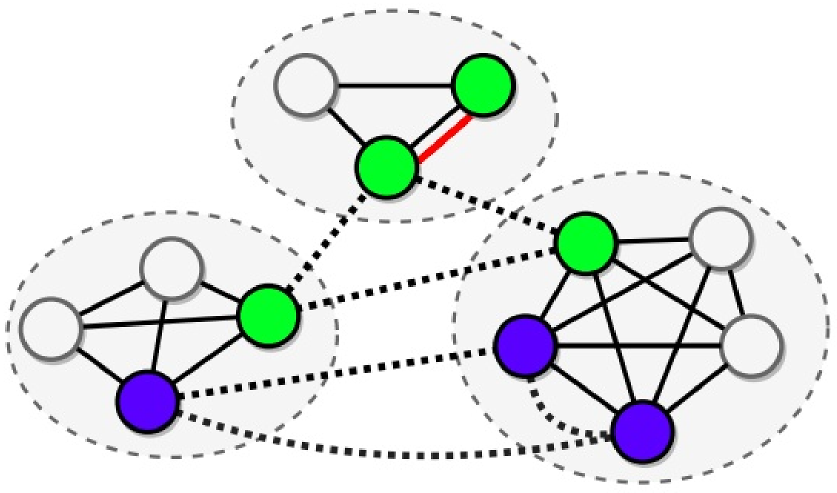


Figure 2: Supporting documents (dashed ellipses) organized as a graph where nodes are mentions of either candidate entities or query entities. Nodes with the same color indicates they refer to the same entity (exact match, coreference or both). Nodes are connected by three simple relations: one indicating co-occurrence in the same document (solid edges), another connecting mentions that exactly match (dashed edges), and a third one indicating a coreference (bold-red line).

EGCN's minimal set of nodes may be a barrier to Multi-hop reasoning capabilities. Consider a fact sequence $S = \{F_i | i \epsilon \{1..n+1\}\}$ where F1 and F_{n+1} have entities found in the question. In this case, the entities found in the intermediate facts, which are required to hop through the fact sequence - are likely not to be found in the question, thus EGCN would ommit them, however this may break the reasoning chain, making answering the question much more difficult.

Heterogeneous Document-Entity graph (HDE)

HDE is the highest performing of these GNN Wikihop models scoring 70.9% on Wikihop, beating EGCN significantly. HDE also uses the most complex graph construction process of the three, seemingly combining the graph elements from both GSPR and EGCN. HDE incorporates entity and coref nodes for all context entities as in GSPR, it also adds in candidate entity nodes similar to EGCN. HDE introduces the document node, which in the case of Wikihop, links all entities mentions to their containing passage. In this way HDE's graph represents the structure of the dataset more explicitly, with all entities able to communicate with the other entities found in the same passage with at most 2 graph hops. Comparing HDE to EGCN, which use the same GNN architecture may indicate that larger and more complex graph structuring is benneficial to MHQA.

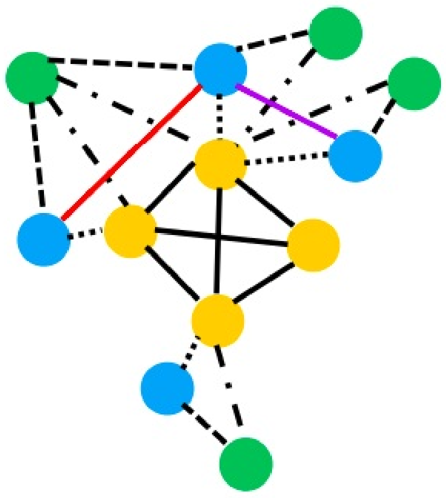


Figure 3: A toy example of HDE graph. The dash dot lines connecting documents (green nodes) and candidates (yellow nodes) correspond to type 1 edge. The normal dash lines connecting documents and entities (blue nodes) correspond to type 2 edge. The square dot lines connecting entities and candidates correspond to type 3 edge. The red solid line connecting two entities correspond to type 4 edge. The purple solid line correspond to type 5 edge. The black solid lines connecting two candidates correspond to type 6 edge. For good visualization, we ignore the type 7 edge in this figure.

Conclusion