**ML with graphs Module 3 assignment**

**1. What do you understand by Graph densification in Machine Learning, also enlist its applications?**

**Ans.**

In recent years, there has been considerable interest in graph structures arising in technological, sociological, and scientific settings: computer networks (routers or autonomous systems connected together); networks of users exchanging e-mail or instant messages; citation networks and hyperlink networks; social networks (who-trustswhom, who-talks-to-whom, and so forth); and countless more [40].

The study of such networks has proceeded along two related tracks: the measurement of large network datasets, and the development of random graph models that approximate the observed properties. Many of the properties of interest in these studies are based on two fundamental parameters: the nodes’ degrees (i.e., the number of edges incident to each node), and the distances between pairs of nodes (as measured by shortest-path length).

The node-to-node distances are often studied in terms of the diameter — the maximum distance — and a set of closely related but more robust quantities including the average distance among pairs and the effective diameter (the 90th percentile distance, a smoothed form of which we use for our studies). Almost all large real-world networks evolve over time by the addition and deletion of nodes and edges. Most of the recent models of network evolution capture the growth process in a way that incorporates two pieces of “conventional wisdom:”

(A) Constant average degree assumption: The average node degree in the network remains constant over time. (Or equivalently, the number of edges grows linearly in the number of nodes.)

(B) Slowly growing diameter assumption: The diameter is a slowly growing function of the network size, as in “small world” graphs [6, 11, 38, 50]. For example, the intensively-studied preferential attachment model [5, 40] posits a network in which each new node, when it arrives, attaches to the existing network by a constant number of out-links, according to a “rich-get-richer” rule. Recent work has given tight asymptotic bounds on the diameter of preferential attachment networks [9, 14]; depending on the precise model, these bounds grow logarithmically [30] or even slower than logarithmically in the number of nodes

**2. Describe about Vocabulary-based Summarization in Machine Learning and how it play its key role?**

**Ans.**

The process of producing summaries from the huge sets of information while maintaining the actual context of information is called Text Summarization. The summary should be fluent and concise throughout.

Google uses featured snippets to show the summary of the article or the answer for the user’s query. These snippets are basically extracted from webpages.

Types of Text Summarizer.

**1) Extractive Summarization:** In this process, we focus on the vital information from the input sentence and extract that specific sentence to generate a summary. There is no generation of new sentences for summary, they are exactly the same that is present in the original group of input sentences

**Example :**

**Source text:** DataFlair is an online, immersive, instructor-led, self-paced technology school for students around the world. DataFlair offers lifetime support, quizzes to sharpen student’s knowledge, and various live project participation. DataFlair machine learning projects are best for students to gain practical knowledge for real-world problems.

**Summary:** DataFlair is an online school for students around the world. DataFlair offers lifetime support, quizzes, and live projects. DataFlair machine Learning projects are best to gain knowledge for real-world problems.

**2) Abstract Summarization:** This is the opposite of Extractive summarization where it takes an exact sentence to generate a summary. Abstract Summarization focuses on the vital information of the original group of sentences and generates a new set of sentences for the summary. This new sentence might not be present in the original sentence.

**Example :**

**Source text:** DataFlair is an online, immersive, instructor-led, self-paced technology school for students around the world. DataFlair offers lifetime support, quizzes to sharpen student’s knowledge, and various live project participation. DataFlair machine learning projects are best for students to gain practical knowledge for real-world problems.

**Summary:** DataFlair is an online school where students are offered various quizzes and projects including machine learning to solve real-world problems.

**What is Sequence to Sequence (Seq2Seq) modeling?**

Seq2Seq model is a model that takes a stream of sentences as an input and outputs another stream of sentences. This can be seen in Neural Machine Translation where input sentences is one language and output sentences are translated versions of that language. Encoder and Decoder are the two main techniques used in seq2seq modeling. Let’s see about them.

**Encoder Model:** Encoder Model is used to encode or transform the input sentences and generate feedback after every step. This feedback can be an internal state i.e hidden state or cell state if we are using the LSTM layer. Encoder models capture the vital information from the input sentences while maintaining the context throughout.

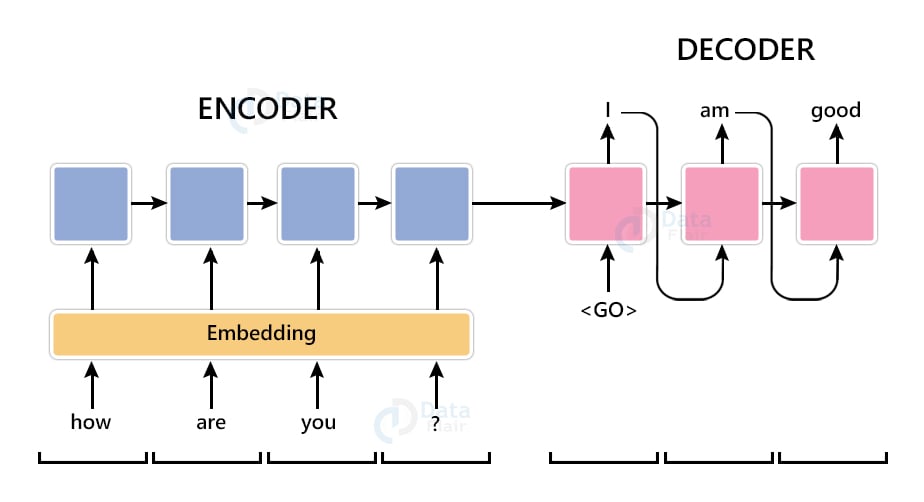
In Neural Machine translation, our input language will be passed into the encoder model where it will capture the contextual information without modifying the meaning of the input sequence. Outputs from the encoder model are then passed into the decoder model to get the output sequences.

**Decoder Model:** The decoder model is used to decode or predict the target sentences word by word. Decoder input data takes the input of target sentences and predicts the next word which is then fed into the next layer for the prediction. ‘<start>’ (start of target sentence) and ‘<end>’ (end of target sentence) are the two words that help the model to know what will be the initial variable to predict the next word and the ending variable to know the ending of the sentence. While training the model, we first provide the word ‘<start>’, the model then predicts the next word that is the decoder target data. This word is then fed as input data for the next timestep to get the next word prediction.

For example, if our sentence is**‘ I Love Python’** so we will add ‘<start>’ at starting and ‘<end>’ at the ending of the sentence therefore our sentence will be**‘ <start> I Love Python <end> ’** now let’s see how it works.

|  |  |  |
| --- | --- | --- |
| **Timestep** | **Input data** | **Target data** |
| 1 | <start> | I |
| 2 | <start> I | Love |
| 3 | <start> I Love | Python |
| 4 | <start> I Love Python | <end> |

As you can see our input data will start from ‘<start>’ and the target will predict the next word with the help of input data at every timestep. Our input data doesn’t contain the last word as our target data at the last timestep is ‘<end>’ which tells us that we have reached the end of our sentence and stop the iteration. The same way our target data will be one-time step ahead as the first word ‘<start>’ is provided by the Input data.

[](https://data-flair.training/blogs/wp-content/uploads/sites/2/2021/08/text-summarization-encoder-decoder-architecture.jpg)

**3. Explain the Knowledge Graph Construction in Machine Learning and its uses?**

**Ans.**

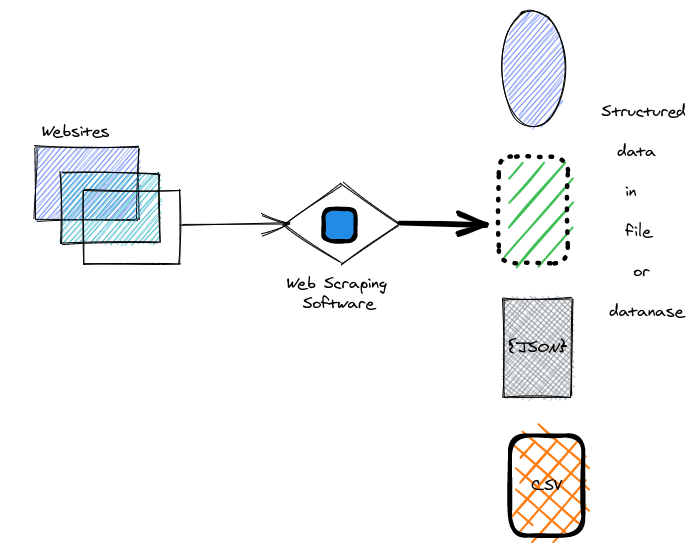
You need to get some information online. For example, a few paragraphs about Usain Bolt. You can copy and paste the information from Wikipedia, it won’t be much work.

But what if you needed to get information about all competitions that Usain Bolt had taken part in, and all related stats about him and his competitors? And then what if you wanted to do that for all sports, not just running?

Machine learning engineers often need to build complex datasets like the example above to train their models. Web scraping is a very useful method to collect the necessary data, but it comes with some challenges.

## **What is web scraping?**

Web scraping (or web harvesting) is data scraping used for data extraction. The **term typically refers to collecting data with a bot or web crawler**. It’s a form of copying in which specific data is gathered and copied from the web, typically into a local database or spreadsheet for later use or analysis.

[Source](https://www.webharvy.com/articles/what-is-web-scraping.html)

You can do web scraping with online services, APIs, or you can write your own code that will do it.

There are two key elements to web scraping:

* **Crawler**: The crawler is an algorithm that browses the web to search for particular data by exploring links across the internet.
* **Scraper**: The scraper extracts data from websites. The **design of scrapers can vary**a lot. It depends on the **complexity and scope of the project**. Ultimately it has to quickly and accurately extract the data.

A good example of a ready-made library is the Wikipedia scraper library. It does a lot of the heavy lifting for you. You provide URLs with the required data, it loads all the HTML from those sites. The scraper takes the data you need from this HTML code and outputs the data in your chosen format. This can be an excel spreadsheet or CSV, or a format like [JSON](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON).

## **Knowledge graph**

The amount of content available on the web is incredible already, and it’s expanding at an increasingly fast rate. Billions of websites are linked with the World Wide Web, and [**search engines**](https://neptune.ai/blog/building-search-engine-with-pre-trained-transformers-guide) can go through those links and serve useful information with great precision and speed. This is in part thanks to knowledge graphs.

Different organizations have different knowledge graphs. For example, the [**Google Knowledge Graph**](https://en.wikipedia.org/wiki/Google_Knowledge_Graph)is a knowledge base **used by Google and its services to enhance search engine results with information gathered from a variety of sources**. Similar techniques are used in Facebook, or Amazon products for a better user experience, and to store and retrieve useful information.

There’s no formal definition of a knowledge graph (KG). Broadly speaking, a KG is a  kind of semantic network with added constraints. Its scope, structure and characteristics, and even its uses aren’t fully realized in the process of development.

Bringing knowledge graphs and machine learning (ML) together can systematically improve the accuracy of systems and extend the range of machine learning capabilities. **Thanks to knowledge graphs, results inferred from machine learning models will have better explainability and trustworthiness**.

Bringing knowledge graphs and ML together creates some interesting opportunities. In cases where we might have insufficient data, KGs can be used to augment training data. One of the major challenges in ML models is explaining predictions made by ML systems. Knowledge graphs can help overcome this issue by mapping explanations to proper nodes in the graph and summarizing the decision-making process.

**4. How Large-scale Graphs help the user of applications based on Machine Learning?**

**Ans.**

Graph-structured data is present everywhere. The problems that GNNs resolve can be classified into these categories:

Node Classification: the task here is to determine the labeling of samples (represented as nodes) by looking at the labels of their neighbors. Usually, problems of this type are trained in a semi-supervised way, with only a part of the graph being labeled.

Graph Classification: the task here is to classify the whole graph into different categories. It’s like image classification, but the target changes into the graph domain. The applications of graph classification are numerous and range from determining whether a protein is an enzyme or not in bioinformatics, to categorizing documents in NLP, or social network analysis.

Graph visualization: is an area of mathematics and computer science, at the intersection of geometric graph theory and information visualization. It is concerned with the visual representation of graphs that reveals structures and anomalies that may be present in the data and helps the user to understand the graphs.

Link prediction: here, the algorithm has to understand the relationship between entities in graphs, and it also tries to predict whether there’s a connection between two entities. It’s essential in social networks to infer social interactions or to suggest possible friends to the users. It has also been used in recommender system problems and in predicting criminal associations.

Graph clustering: refers to the clustering of data in the form of graphs. There are two distinct forms of clustering performed on graph data. Vertex clustering seeks to cluster the nodes of the graph into groups of densely connected regions based on either edge weights or edge distances. The second form of graph clustering treats the graphs as the objects to be clustered and clusters these objects based on similarity.

**5. What do you mean by TimeCrunch algorithm in Machine Learning, discuss with details.**

**Ans.**

Given a large phonecall network over time, how can we describe it to a practitioner with just a few phrases? Other than the traditional assumptions about real-world graphs involving degree skewness, what can we say about the connectivity? For example, is the dynamic graph characterized by many large cliques which appear at fixed intervals of time, or perhaps by several large stars with dominant hubs that persist throughout? Our work aims to answer these questions, and specifically, we focus on constructing concise summaries of large, real-world dynamic graphs in order to better understand their underlying behavior.

The related work falls into three main categories: static graph mining, temporal graph mining, and graph compression and summarization. Table 1 gives a visual comparison of TIMECRUNCH with existing methods. Static Graph Mining. Most works find specific, tightly-knit structures, such as (near-) cliques and bipartite cores: eigendecomposition [23], cross-associations [6], modularity-based optimization methods [19, 5]. Dhillon et al. [9] propose information theoretic co-clustering based on mutual information optimization. However, these approaches have limited vocabularies and are unable to find other types of interesting structures such as stars or chains. [13, 17] propose cut-based partitioning, whereas [3] suggests spectral partitioning using multiple eigenvectors – these schemes seek hard clustering of all nodes as opposed to identifying communities, and are not usually parameter-free. Subdue [7] and other fast frequentsubgraph mining algorithms [11] operate on labeled graphs. Our work involves unlabeled graphs and lossless compression. Temporal Graph Mining. [2] aims at change detection in streaming graphs using projected clustering. This approach focuses on anomaly detection rather than finding recurrent temporal patterns. GraphScope [25] uses graph search for hard-partitioning of temporal graphs to find dense temporal cliques and bipartite core.

**6. Briefly discuss the key features and applications of SlashBurn algorithm in the context of Machine Learning.**

**Ans.**

we describe SLASHBURN, our proposed ordering method for compressing graphs. Given a graph G, the SLASHBURN method defines a permutation π : V → [n] of a graph so that nonzero elements in the adjacency matrix of G are grouped together. Algorithm 1 shows the high-level idea of SLASHBURN. The lines 1 and 2 remove top k highest centrality scoring nodes and incident edges, thereby decomposing nodes in G into the following three groups:

• k-hubset: top k highest centrality scoring nodes in G.

Algorithm 1: SLASHBURN

Input: Edge set E of a graph G = (V, E), a constant k (default = 1).

Output: Array Γ containing the ordering V → [n].

1: Remove k-hubset from G to make the new graph G 0 . Add the removed k-hubset to the front of Γ.

2: Find connected components in G 0 . Add nodes in non-giant connected components to the back of Γ, in the decreasing order of sizes of connected components they belong to.

3: Set G to be the giant connected component (GCC) of G 0 . Go to step 1 and continue, until the number of nodes in the GCC is smaller than k.

• GCC: nodes belonging to the giant connected component of G0 . Colored blue in Fig. 5. • Spokes to the k-hubset: nodes belonging to the nongiant connected component of G0 . Colored green in Fig. 5. Fig. 5 shows a graph before and after 1 iteration of SLASHBURN. After removing the ‘hub’ node at the center, the graph is decomposed into the GCC and the remaining ‘spokes’ which we define to be the non-giant connected components connected to the hubs. The hub node gets the lowest id (1), the nodes in the spokes get the highest ids (9∼16) in the decreasing order of sizes of connected components they belong to, and the GCC takes the remaining ids (2∼8). The same process applies to the nodes in GCC, recursively. If there exist more than one GCCs having the same size, we choose one of them randomly, which is a very rare case in real graphs.

Fig. 4(a) shows the AS-Oregon graph after the lines 1 and 2 of Algorithm 1 are executed for the first time with k = 256. In the figure, we see that a k-hubset comes first with GCC and spokes following after them. The difference between (spokes1) and (spokes2) is that the nodes in (spokes2) are connected only to some of the nodes in k-hubset, thereby making large empty spaces in the adjacency matrix. Notice also that nodes in (spokes1) make a thin diagonal line, corresponding to the edges among themselves. A remarkable result is that the remaining GCC takes only 45% of the nodes in the original graph, after removing 256 (=1.8 %) high degree nodes. Fig. 4(b) and (c) shows the adjacency matrix after doing the same operation on the remaining GCC, recursively. Observe that nonzero elements in the final adjacency matrix are concentrated on the left, top, and diagonal areas of the adjacency matrix, creating an arrow-like shape. Observe also that the final matrix has huge empty spaces which could be utilized for better compression, since the empty spaces need not be stored. An advantage of our SLASHBURN method is that it works on any power-law graphs without requiring any domain-specific knowledge or a well defined natural ordering on the graph for better permutation.

Finally, we note that setting k to 1 often gives the best compression by making the wing width ratio w(G) minimum or close to minimum. However, setting k to 1 requires (a) Before SLASHBURN (b) After SLASHBURN Fig. 5: [Best viewed in color.] A graph before and after 1 iteration of SLASHBURN. Removing a hub node creates many smaller ‘spokes’, and the GCC. The hub node gets the lowest id (1), the nodes in the spokes get the highest ids (9∼16) in the decreasing order of sizes of connected components they belong to, and the GCC takes the remaining ids (2∼8). The next iteration starts on the GCC. many iterations and longer running time. We found that setting k to 0.5% of the number of nodes gives good compression results with small number of iterations on most real world graphs.

2.5 Improvements of SLASHBURN The basic SLASHBURN algorithm results in good compression which we will show in Section 4, but there is still a room for improvement in both compression quality and running time. In this section, we discuss possible improvements for SLASHBURN in efficiency and effectiveness. To this end, we consider SLASHBURN to alternate two main operations as described in Algorithm 1: 1) selecting k-hubset, and 2) putting the remaining connected components (CCs) in appropriate places in ordering. The goal is to refine the two operations with the goal of obtaining compression quality and running time comparable to the basic SLASHBURN with k = 1 and k 1, respectively