How to get started with Graph Machine Learning

gordicaleksa.medium.com/how-to-get-started-with-graph-machine-learning-afa53f6f963a

February 7, 2021

This blog is a part of my **"deep learning update"** series and I want to open it up with a question: What have I learned about Graph ML in 2+ months?

Nothing? If that was your first thought, no worries, it's probably true.

(Bad) jokes aside, my "relative knowledge" (the knowledge I "possess" vs the knowledge I'm aware of) is asymptotically approaching **0** whereas my "absolute knowledge" did grow by quite a lot over the last 2 months.

But you never see allthe amazing things you've learned, you see that **0**. You have that weird feeling that **0** is staring at you. At least I have it, sometimes.

It's always like that. You approach a new field and you have this delusional feeling that you'll only need to learn A (e.g. word2vec/node2vec), B (e.g. GPT/GCN), and C (e.g. BERT/GAT) and you'll master the field.

It's finite. You can complete it. It's like a game.

But, as you keep on learning and exploring you discover a whole new world of *ideas* and *terminology*, of *people* and *history*. And that feeling is beautiful and frightening at the same time, and I assume it's the reason many people give up.

Photo by on

Research and learning can get intimidating because it makes you feel so small. The best advice I can give you is — **never give up**. I mean it. I'll delete your GitHub (or Fortnite?) account if you do. I swear.

Nice, after giving you a micro-existential crisis and then bringing you up , let us continue.

If you were following along my journey you knew that I'm going to tackle Graph ML next. In my last blog update on transformers, 2 months ago, I mentioned that my next field of exploration is Graph ML, so here we are.

Now let me tell you why you should learn about this field.

The awesome world of Graph ML

There are so many exciting applications. I mean it. I've never seen a field with this many exciting applications!

All throughout science, you'll find GNNs (Graph Neural Networks) being deployed as we speak. Anything you can model as a graph (and that's pretty much everything!) and you can throw a GNN at it.

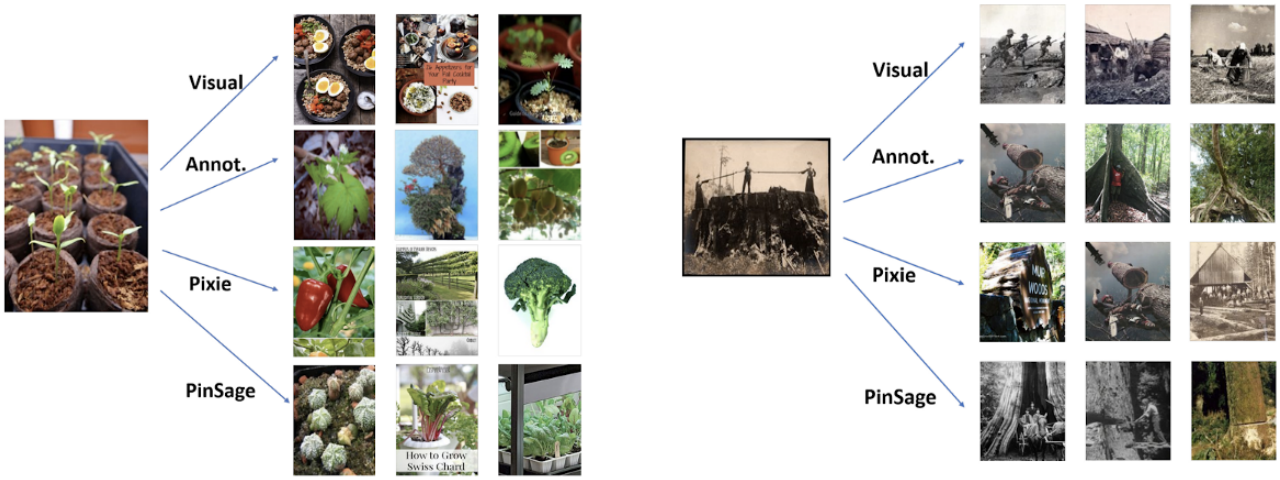
Massive-scale recommender systems, particle physics, computational pharmacology/chemistry/biology, traffic prediction, fake news detection, and the list goes on and on.

You still don't believe me? That's great. I don't want you to take my word for granted, you're a *critical thinker* let me prove it to you!

Uber Eats uses GNNs to recommend you food and restaurants.

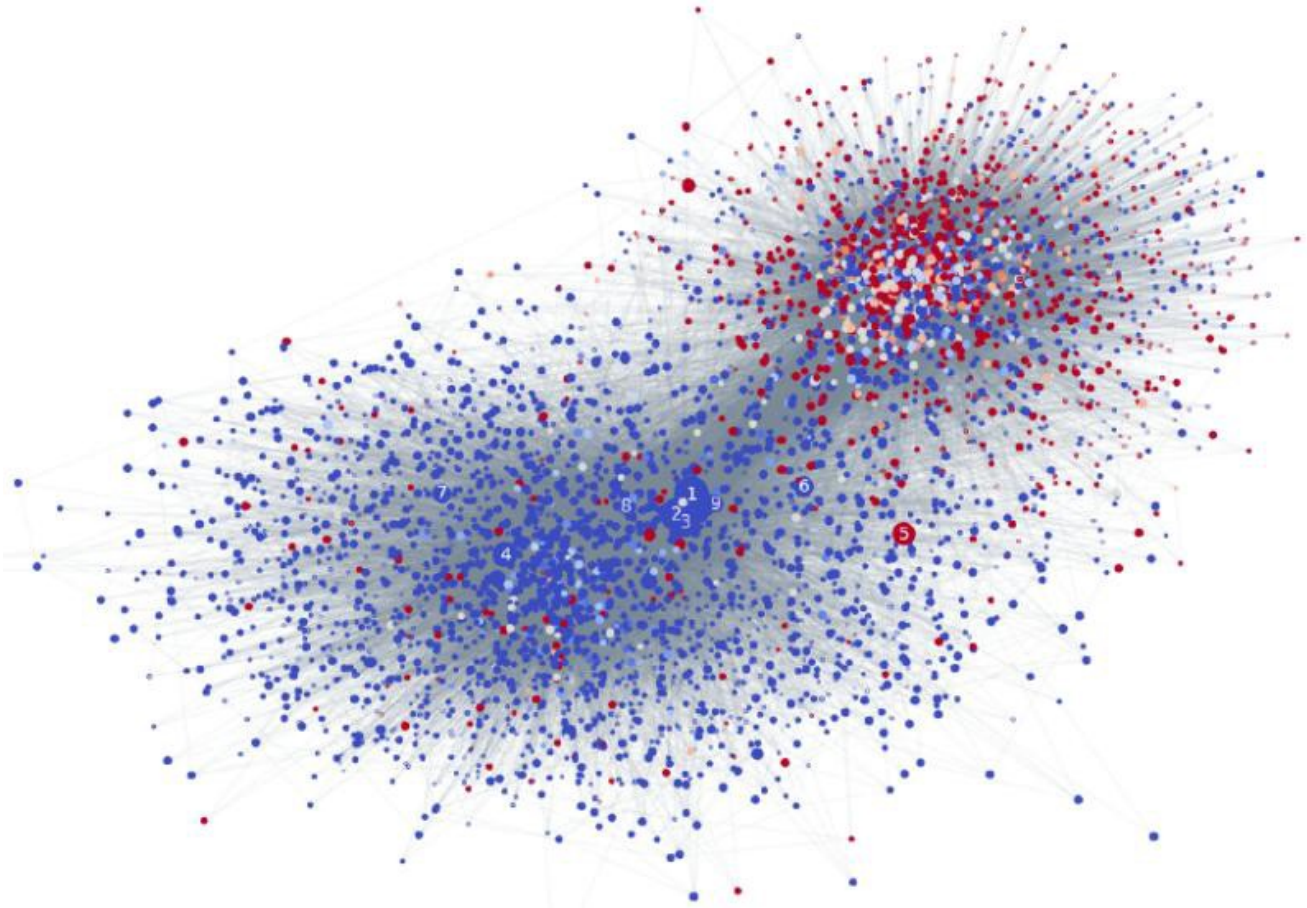
Food discovery with Uber Eats: Using Graph Learning to Power Recommendations

Pinterest deployed PinSage GNN at least 2 years ago to their platform and it's now curating your homefeed and it's suggesting you the most relevant pins.



PinSage: A new graph convolutional neural network for web-scale recommender systems

Twitter uses GNNs to detect (and hopefully filter out) fake news.



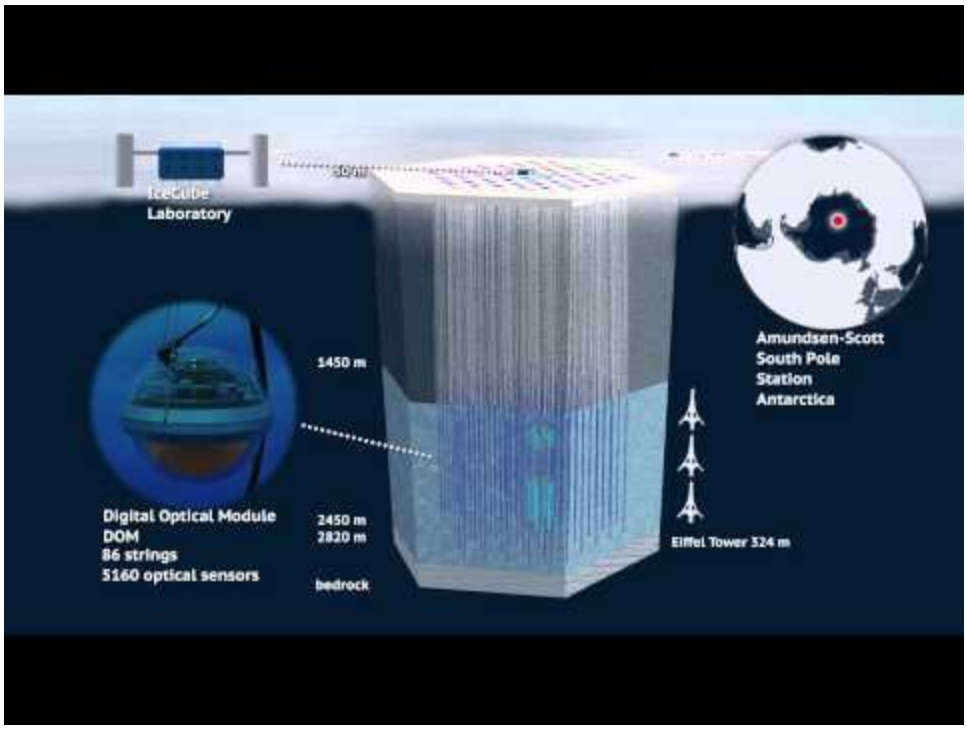
The reliable users are colored in blue and the unreliable ones (prone to spreading fake news) in red.

This is possible thanks to the fact that fake and real news spread differently on social media! We can thus exploit the propagation pattern of fake news over time as a valuable signal.

On the above image, you can notice an interesting pattern that likely-minded people tend to cluster together — nothing new there . When the graph has that property (connected nodes share a similar label) we call it homophilic (from Ancient Greek: *“homos”*-same, *“philie”*-love/friendship).

Let’s continue on into fundamental science!

We’re detecting neutrino particles on the South Pole at the IceCube lab, and we’re classifying them using GNNs!



Neutrino, measuring the unexpected — IceCube

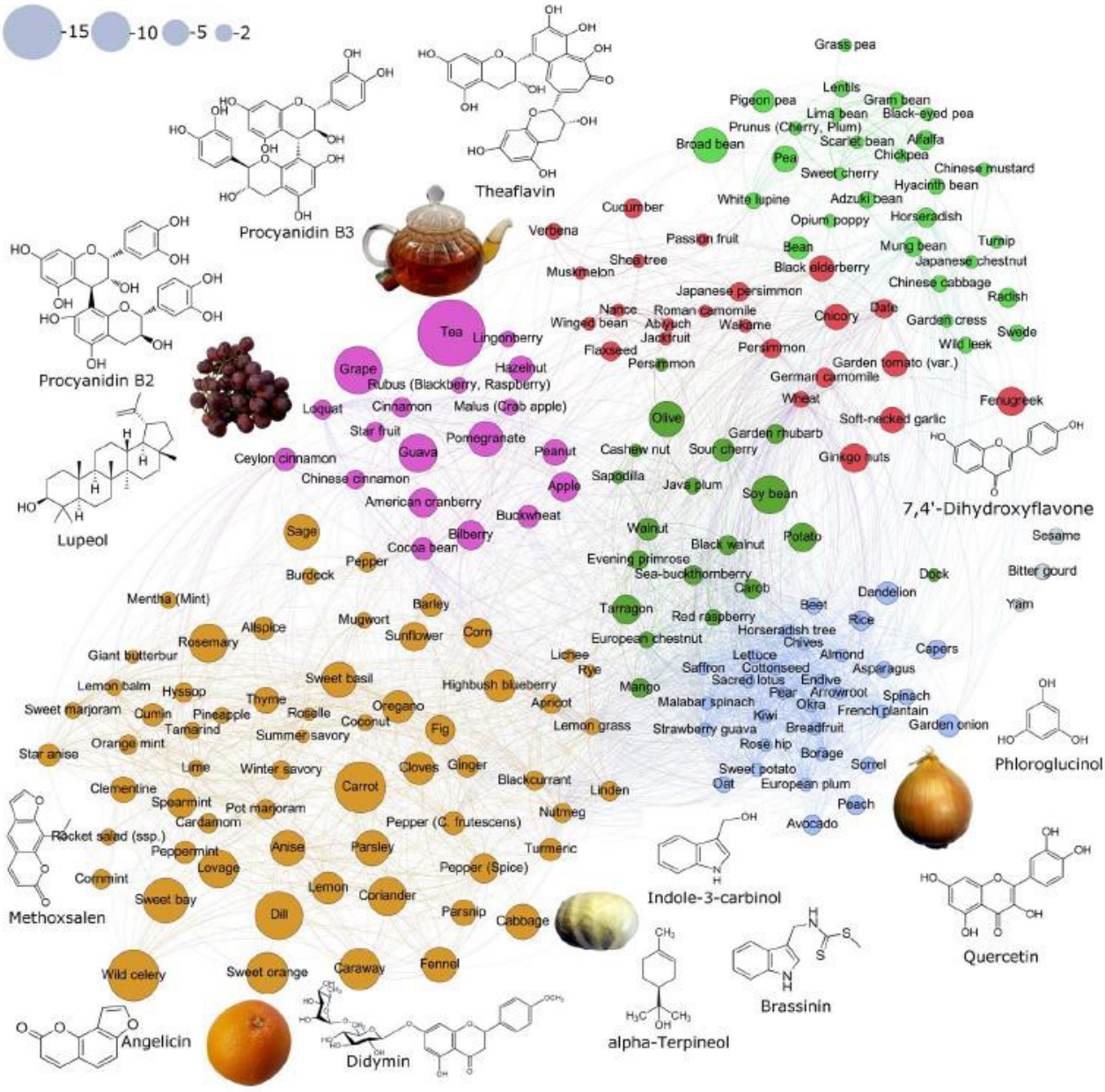
IceCube is a 1 km3 neutrino observatory located at the South Pole and its primary purpose is to look for high-energy (above 100 gigaelectronvolts (GeV)) neutrinos!

It basically consists of a hexagonal grid of light-sensitive sensors that capture the Cherenkov light emerging from the interactions between charged particles with the ice.

You can *nicely model this sensor structure as a graph*! Each light sensor is a vertex and it has associated with it a 6-dimensional feature vector (3 XYZ coordinates + some energy and time of arrival information).

But let's leave the tech details for a bit later and continue the Odyssey.

We can use GNNs to help beat cancer! By taking the protein/gene and drug interaction data and modeling the problem as a graph and throwing some GNN weaponry at it we can find a list of molecules that help beat/prevent cancer.



HyperFoods: Machine intelligent mapping of cancer-beating molecules in foods (the bigger the node the more diverse the set of CBMs)

Once we have the most effective CBMs (cancer-beating molecules) we can create a map of foods that are richest in these CBMs, also called **hyperfoods**.

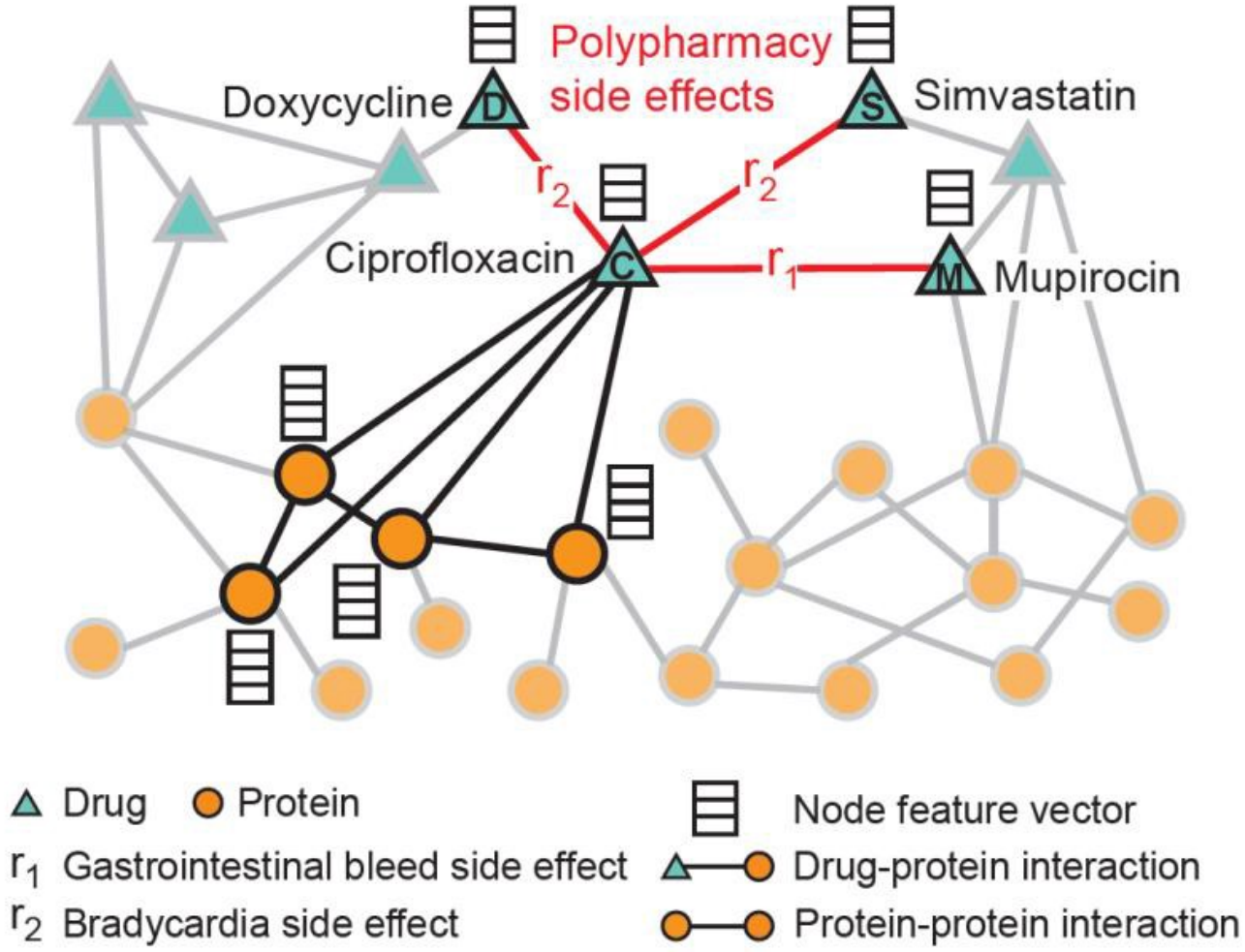
Here is a TL;DR: the hyperfoods are: tea, grape, carrot, coriander, sweet orange, dill, cabbage, and wild celery.

But, don't overdose with these, I beg you!

I know I started drinking green tea regularly after reading this paper. Who says that reading research papers is not healthy!?

On a side note, I take my nutrition very seriously. I'm regularly pushing both my body (I lift weights, etc.) and my mind to its limits so it's very important to think of the recovery and rest as well. High quality sleep, omega-3s, nuts, vitamin D, green tea are some of my tricks.

But if you're not a fan of nutrition (and causality in general) then the following research may excite you even more — predicting (polypharmacy) drug side effects using GNNs!



Modeling polypharmacy side effects with graph convolutional networks

Polypharmacy (using multiple drugs at the same time) side effects is something pretty much everyone will experience and we know that our grandparents are already experiencing it.

We basically can't tell with certainty what will happen to you if you take 2+ random drugs.

Experimentally screening every single combination of drugs for side-effects is impossible due to the combinatorial nature of the problem — so computational methods such as GNNs to the rescue!

Aside from the polypharmacy side effects problem we're facing an ever-growing threat from antibiotic-resistant bacteria that are evolving thanks to our irresponsible use of antibiotics.

Earlier in 2020, a super famous application of GNNs was published in Cell — A deep learning approach to antibiotic discovery.

Note: no free-access pdf, unfortunately (afaik), which makes me sad

Basically, by training a GNN on a dataset of molecules known to inhibit the growth of Escherichia coli, and later doing inference on a dataset of drugs under investigation, they "discovered" halicin-a molecule previously tested for its anti-diabetic properties now repurposed to be used as a potent antibiotic!

Escherichia coli bacteria (Powerful antibiotics discovered using AI)

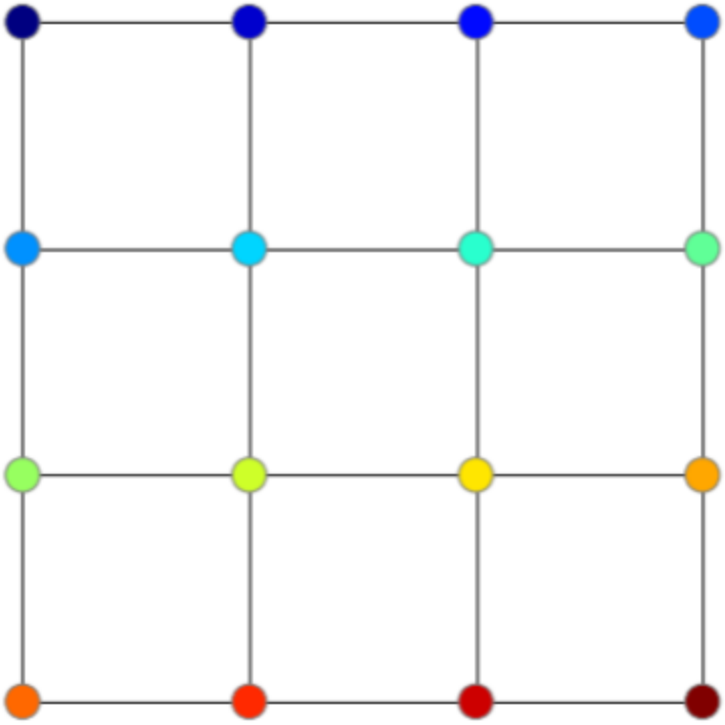
I've left out many exciting apps like predicting quantum properties of molecules, predicting molecule's X chemical property (e.g. solubility), etc.

It's an infinite list.

You can now see how dull and homogeneous the computer vision and NLP fields are!

NLP: *I manipulate text*. CV: *I manipulate images*. Graph ML: *hold my beer!*

Jokes aside, there are some nice connections between CV/NLP and Graph ML. You can treat transformers as Graph Attention Networks operating on fully-connected graphs (but more on that later) and you can treat images/videos as regular graphs (aka grids).



An example of a 4x4 pixel image — we can treat an image as a grid graph. Every node has associated a scalar (grayscale) or a 3D vector ((RGB) with it. That's the **graph signal** in this example.

Through this lens, everything is a graph. Now whether this abstraction makes actual sense is problem specific. Specialized models such as CNNs have some useful biases ingrained into them and you probably wouldn't want to apply a GNN directly to image pixels (although GNNs are getting some serious traction in the CV world in one way or another).

Hopefully I got you hyped up! Let me now show you some resources that I've found useful learning about Graph ML.

Note: I often mention GNNs but they are only a subset of the Graph ML field. Aside from GNNs there are graph embedding methods and many others.

Note 2 (on how you should (probably) read the following chapter): first read it without digressing into the resources which I've linked and only then start using it as a reference manual — because that's what this blog is, it's not your one-shot blog, read it and forget it, noup.

Fantastic GNNs and Where to Find Them

I don't aim to provide you with an extensive overview of the Graph ML literature here, and I don't think that would be useful at all for that matter.

Dumping bunch of resources (which I often see people do in GitHub repos) is not that useful — especially not for beginners.

What I'll try to do is structure some of the resources, which I've found useful, in a way such that 2-month younger Aleksa would be super grateful.

Let's start! I'll try and give you some more generally applicable research tips along the way as well.

Graph embedding methods

If you're trying to dive into a new field it's super *useful* to **organize your learning resources chronologically** (for nerds: the timestamp feature of the research paper is highly valuable — don't ignore it).

Before GNNs became mainstream people used methods analogous to Word2Vec which enjoyed a great deal of popularity in the NLP world.

Most of these were actually directly inspired by the Word2Vec method and to be honest, I'm happy I built my NLP and CV backgrounds before jumping into Graph ML. The field pulled a lot of inspiration from CV and NLP fields as you'll soon see.

Here are some papers I recommend you start your journey with:

- DeepWalk — it's reduced to Word2Vec if you do the following: sample "sentences" from the graph by doing random walks.
- Node2Vec — literally the same idea as DeepWalk with the additional control of how you'll sample from your graph (BFS/DFS). If you give more weight to BFS the tightly coupled nodes will have similar embeddings.
- Planetoid — a semi-supervised setting. Aside from random walks (unsupervised) you can also leverage the labels (supervised). They additionally made sure that their method can be used in an inductive setting (check out my GAT Jupyter Notebook for the short explanation of transductive vs inductive learning, just *ctrl+f* it).

A brief clarification on DeepWalk: if you treat the nodes in the graph as words then if you do a random walk you're basically sampling a random sentence from your graph.

Once you have the sentence you can do the Word2Vec magic and learn your node embeddings such that those nodes that tend to be close in your sentences have similar embeddings.

Graph embedding methods

All right! Now you're ready for some GNN magic!

Graph Neural Networks

Historically there were 2 main approaches to developing GNNs:

- Spectral methods
- Spatial (message passing) methods

Both tried to generalize the mathematical concept of **convolution** (that worked so nicely in the CV world) to graphs.

The spectral methods tried to preserve the strict mathematical notion of convolution and had to resort to the frequency domain (Laplacian eigenbasis) for that.

On the other hand the spatial methods are not convolutions in the strict mathematical sense of the word, but we still informally call them convolutions, as they were very loosely inspired by the spectral methods.

The long story short, spectral methods "died out" because they are:

- **Computationally expensive**
- **Inherently transductive**

Having said that they are definitely still worth researching and you should build up at least some basic foundations on spectral methods, IMHO.

But, before we continue with spectral/spatial GNN papers I strongly suggest you take some time off to do a superficial, high-level overview of what Graph ML is. **Build up that knowledge skeleton first** before you start filling in the blanks. The top to bottom approach, I'm a big fan of it.

Overview of GNNs

Additionally here are some blogs to get you exposed to new terminology and information, let them sink in:

- Tutorial on Graph Neural Networks for Computer Vision and Beyond (a beginner-friendly intro to GNNs)
- A Tale of Two Convolutions: Differing Design Paradigms for Graph Neural Networks
- A high-level overview of some important GNNs (MoNet falls into the realm of geometric deep learning though, but more on that later)

Nice!

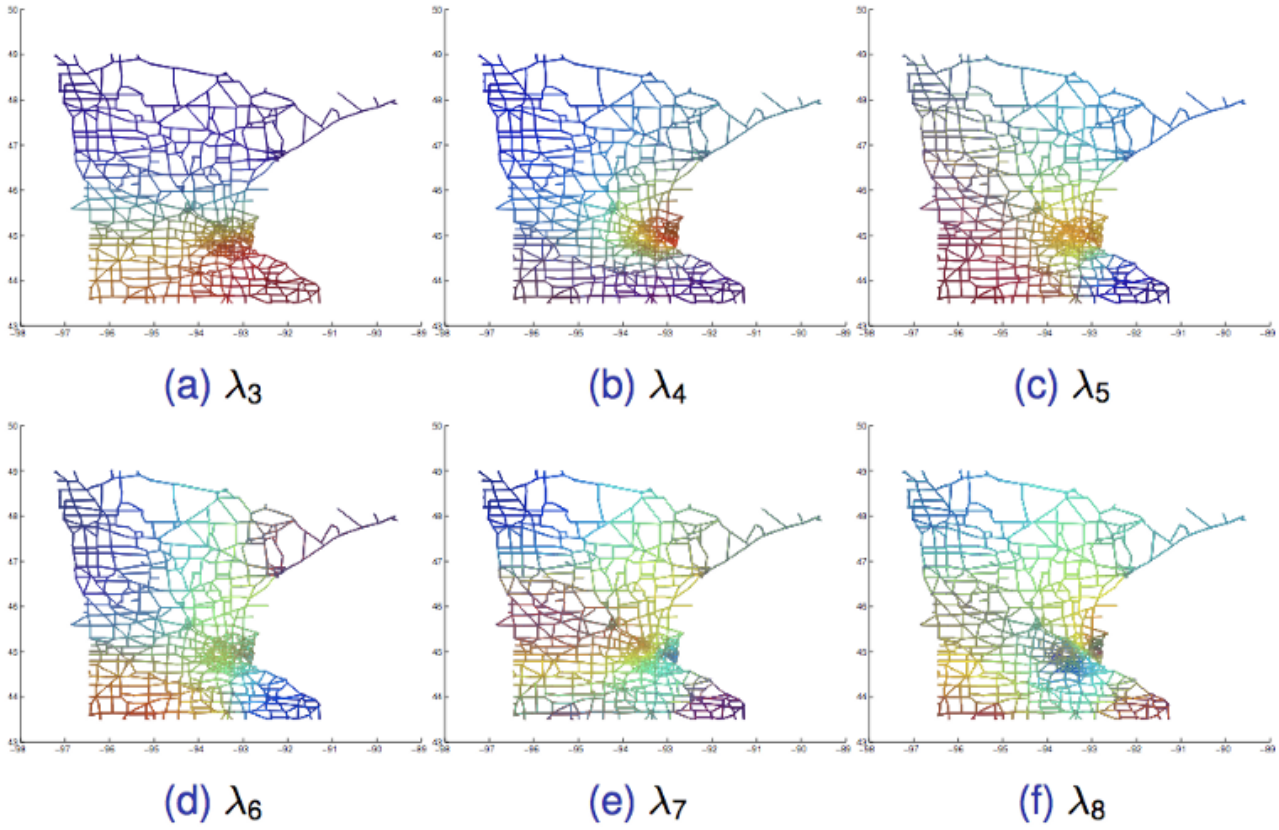
High-level overview of Graph ML

You're now ready to dive into the world of Graph Neural Networks.

The spectral methods

The main "tool" behind the spectral methods is something called the graph Laplacian. Additionally, the **undirected graph assumption** is made so that we can have a **symmetric** graph Laplacian which has some nice mathematical properties — namely the fact that we can find an **orthonormal Laplacian eigenbasis*.**

** it's a generalization of the **Fourier** eigenbasis with which you're familiar if you took any signal processing course, check out 3B1B's video for a gentle intro. In a special case where the underlying graph is a grid (like image) we get sinusoids i.e. Fourier eigenfunctions!!!*



Laplacian eigenvectors visualized on one specific graph (image from on of the above blogs)

Now the main idea is to **project the graph signal** into that eigenbasis, filter the projected graph signal directly in the spectral domain by doing an **element-wise multiplication** with the filter, and **reproject it back** into the spatial domain.

It's ok not to understand all of this right here and right now. You'll have your epiphany moment sooner or later. Here are the main spectral papers:

- Spectral Networks and Locally Connected Networks on Graphs — first introduced the concept of the convolution to graphs.
- ChebNets — One of the main problems with the above method is that it's **not localized** i.e. the updated node features can be affected by any node in the graph.

The ChebNets paper made it so that only **k-hop neighborhoods** can impact the feature representation of any particular node and also made it **computationally efficient** to do so — by using a *Chebyshev polynomia*l of the graph Laplacian there is no need to compute the Laplacian eigenbasis which is an expensive operation (~$O(n^3)$).

Note: in spectral method papers you'll be seeing some "heavy" mathematics which is not a prerequisite for understanding the spatial methods (and they are way more popular so don't beat yourself down if you don't get this the first time).

Aside from the papers here are some other super useful resources that will help you better understand these spectral methods:

- Spectral Graph Convolution Explained and Implemented Step By Step
- The Smallest Eigenvalues of a Graph Laplacian
- A tutorial on Graph Laplacians

And finally, this video made it click for me when it comes to understanding the spectral clustering method which leverages the Laplacian eigenvectors to cluster the nodes (any additional intuition you can collect will be useful).

That's it. I've left out some less important papers to avoid the clutter (and btw. I'm only recommending the resources which I've *read* **and** *found useful*).

Spectral methods!

Now let's continue to the realm of spatial methods.

Spatial/Message passing methods

Here is a high-level, super informal, explanation of message passing methods.

The goal is to update each and every node's representation. You have the feature vectors from your node and from his neighbors at disposal. You then do the following:

1. You somehow transform the feature vectors (maybe a linear projection)
2. You somehow aggregate them (maybe weighing them with attention coefficients, voilà, we get GAT (*you see what I did there*))
3. You update the feature vector (somehow) of the current node by combining its (transformed) feature vector with the aggregated neighborhood representation.

And that's pretty much it, you can fit many different GNNs into this framework.

Let's now explore some of the most famous spatial methods:

> GCN (Graph Convolutional Network) — simplified the ChebNets paper to operate on 1-hop neighborhoods and made GNNs much more efficient and applicable.

I've covered the GCN paper in this video:

My overview of the Graph Convolutional Network (GCN)

> Graph Attention Network (GAT) — made GCN more expressive by allowing it to **learn** the neighborhood aggregation coefficients (GCN uses constants instead of learnable weights).

As you'll later see I've even implemented GAT from scratch and Petar Veličković (the first author) himself recommended this video so I guess it should be good enough!

My overview of the GAT method

Now the problem with GAT and GCN is they were originally **not scalable** which is needed for many real-world, massive-scale, graph datasets like any social network's graph.

The following 2 papers made GNNs applicable to huge graphs:

> GraphSAGE — **SA**mple and aggre**G**at**E**, it introduced the concept of sampling your neighborhood and a couple of optimization tricks so as to constrain the compute/memory budget. It's a direct precursor to PinSage which is deployed at Pinterest as the recommender system!

Here is my overview of GraphSage. Btw. if you were wondering, Graph ML is a pretty young field so you won't find many YT overviews of these papers that's why I'm linking the ones I created. I only wish to link the best content — if that's sometimes my content — well let it be. Anyways! The second paper:

> PinSage — One of the most famous real-world applications of GNNs — recommender system at Pinterest.

And here is my video overview:

My overview of the PinSage method

Those were some of the most important spatial GNNs. Going further here is a paper that generalized all of the above nets into a so-called **Message Passing** framework:

> MPNN (message passing neural net) — aside from having a theoretical significance of unifying the spatial GNNs it also introduced an awesome application of GNNs: predicting quantum properties of molecules e.g. fundamental vibrations, energy needed to break up the molecule in certain states, etc.

Two more seminal papers you should check out are:

- Gated Graph NN — used GRU for neighborhood aggregation and applied the method to program verification via the separation logic. It's not an easy read, but I think it's worth it, it played an important role in kick-starting the spatial method revolution.
- Learning Convolutional Neural Networks for Graphs — gave an idea of how we could impose some order onto the graph neighborhood (via labeling) and apply a convolution that resembles CNNs much closer. I guess it could be considered as a third way to introduce convolution to graphs, but this approach didn't get any serious traction though.

Wrapping up with one more amazing paper which showed that we clearly don't know what we're doing :

> Simplifying Graph Convolutional Networks (SGC)

Jokes aside, what it showed is that **for certain graph families** GNNs are totally unnecessary and a much simpler baseline (that acts as simple **low-pass** filter) can do an even better job!

It's a single layer GNN that first does the aggregation over the k-hop neighborhood (the majority of the benefit arises from this) followed by a simple linear projection and a softmax.

In these early days of spatial methods we were missing answers to some fundamental theoretical questions like why are GNNs so shallow (compared to their extremely deep CNN cousins), what are their limitations, etc.

That brings us to the following section.

Seminal spatial GNN papers

Let's go!

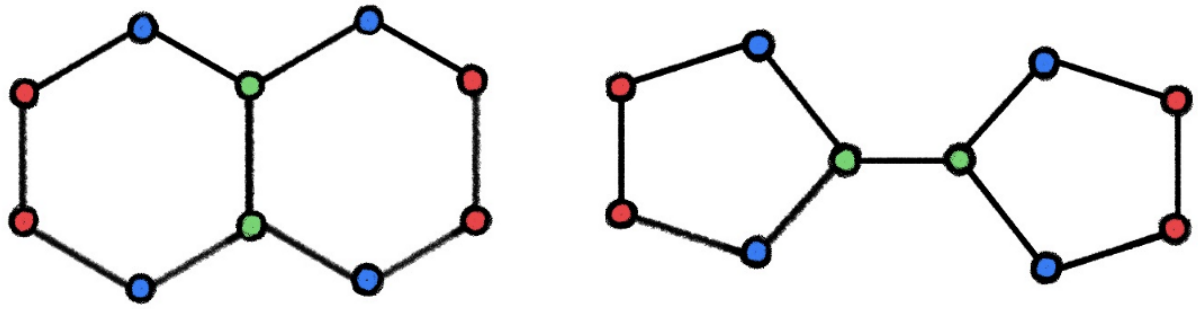
GNN *expressivity*

Early spatial methods, such as the famous GCN paper, started making some links between GNNs and old graph theory methods such as the **Weisfeiler Lehman algorithm.**

Weisfeiler Lehman (1-WL) also known as the *color refinement* algorithm is a simple graph isomorphism test: given 2 graphs verify they are the same even though their node ids may be permuted. WL can tell you with **certainty** whether your 2 graphs are non-isomorphic and it can tell you, with **high probability**, whether they are isomorphic.

Check out this awesome blog for a great explanation of how WL works.

Now there are certain graph families which can "confuse" the WL test. Here is one example of where the WL algorithm outputs the same color histograms (and thus it "thinks" that these 2 graphs are isomorphic — but they're not):



Two non-isomorphic graphs on which the WL graph isomorphism test fails (M. Bronstein's blog)

Ok, now you may think this is all nice Aleksa, you're brushing up our fundamental graph theory knowledge, but what the heck? Well.

This seminal paper showed that MPNNs are, at best, **as powerful as 1-WL**:

> GIN — Graph Isomorphism Network

It introduced certain requirements needed for the MPNN to be as expressive as 1-WL like using **sum** as the aggregation function (hint it's injective) instead of the popular **max** and **mean** aggregators.

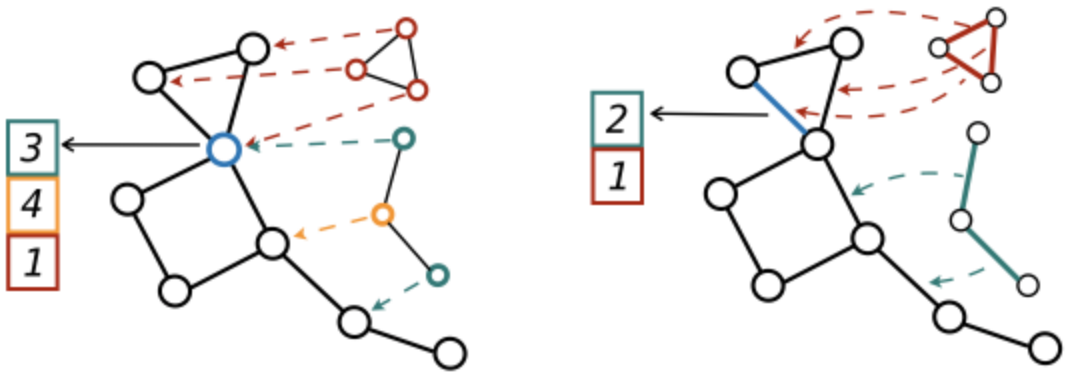
Michael Bronstein has a nice series of 3 blogs on this topic so do check it out!

After GIN, many methods tried to unleash GNNs and give them even more expressive power, and so various methods were advised which were as powerful as **k-WL** algorithms.

Also other families of methods were advised which broke up with the WL hierarchy altogether like this one:

> Improving GNN expressivity via subgraph isomorphism counting

The main idea is that since some previous research showed that GNNs lack the subgraph structure counting ability we can add those explicitly as additional node/edge features.



counting triangles and paths and appending those numbers to node features

Note on expressivity: if a model is more expressive that means that it would produce different embeddings for 2 graphs that are non-isomorphic with higher probability than some less expressive model (as we saw some graphs confuse 1-WL test (and thus MPNNs) and so in those cases we'd get the same embeddings even though the 2 graphs are non-isomorphic!)

As you can see you could be doing just this subfield of Graph ML research as it's fairly complex and there are still many open questions!

GNN expressivity

Let's take a step back from the hard-core theory and investigate some exciting app papers, some of which I've mentioned in the previous chapter.

Papers on exciting GNN applications

Photo by Terry Vlisidis on Unsplash

I've already mentioned some of these:

- GNNs for IceCube signal classification
- Hyperfoods paper (anti cancer molecules and foods) — it's a bit harder to read than the other papers but you could skim it to get a rough idea of what they did.

- Modeling polypharmacy side effects with GNNs — models the problem as a multi-modal graph (proteins/drugs), with the goal of multi-relational link prediction (given a pair of drugs there can be up to 964 different edges in between each corresponding to a different side effect!)
- Fake News Detection on Social Media using Geometric Deep Learning
- Convolutional Networks on Graphs for Learning Molecular Fingerprints — beautiful paper it replaced the old heuristics with a learnable GNN and gave awesome results on predicting different properties of molecules.
- Modeling Relational Data with GCNs — applying GNNs to knowledge base entity and link prediction.
- Graph Structure of Neural Networks — uses network/graph analysis knowledge to do a smarter NAS (neural architecture search). Neural networks themself are graphs, right? So let's apply some Graph ML knowledge! Note this is less of a GNN paper but still an interesting read.

Aside from all of these and the stuff I mentioned in the previous chapter I also suggest these 2 blogs for more awesome applications:

- Geometric ML becomes real in fundamental sciences (Michael Bronstein)
- Top Applications of Graph Neural Networks 2021 (Sergey Ivanov)

As well as this paper for a holistic overview of different methods used in the Graph ML field:

> Representation Learning on Graphs: Methods and Applications

Exciting applications of GNNs

We're getting closer to the end. Go drink a green tea and get back.

Handling dynamic graphs

So far all of the papers I linked deal with the so called **static graphs** i.e. graphs that **don't evolve over time**. But the truth is many graphs of interest do change over time. Take Twitter/Facebook/<your fav social net if any> graph as an example.

Here are 2 nice papers dealing with continuous time step dynamic graphs:

- Temporal Graph Networks for Deep Learning on Dynamic Graphs — this paper came straight from Twitter with M.Bronstein being one of the coauthors.
- Inductive Representation Learning on Temporal Graphs (TGAT) — fairly imilar to TGN above, it can actually be considered a special case.

And again here is my overview of TGN paper (Rossi endorses it):

Temporal Graph Networks explained

This was just to tickle you imagination. You can imagine dedicating a couple of years of research to only this subtopic of Graph ML. There are still many open questions and engineering problems left.

The 2 “main” tricks are:

- Find a way to represent time and include it as an important feature
- Define a temporal neighborhood over which to aggregate

Dynamic graphs

Finally let me leave you with some totally optional papers.

Note: pretty much every subsection, including this one on dynamic graphs, is orthogonal to other subsections. Just saying.

Go unsupervised young ML human

If you’re passionate about unsupervised learning you’ll find these 2 interesting:

Aside from the unsupervised learning you may wish to place your foot into the Geometric-DLandia (Geometric DL mostly deals with manifolds although there are many connections with the Graph ML field):

Geodesic CNNs on Riemannian manifold (GCNN) — this one broke my mind , working on manifolds involves so many new details which I haven’t seen anywhere else across all the different deep learning fields I’ve explored so far.

Geometric deep learning: going beyond Euclidean data — again, really math-heavy and includes a complex survey on manifolds, differential geometry, graphs, different methods and different applications.

MoNets — Geometric DL on graphs/manifolds using mixture models. It generalizes some of the previous methods like the above-mentioned GCNN.

Again, note that there is some heavy mathematics, new jargons and terminology involved with Geometric DL — promise that you won’t get overwhelmed!

You don’t need to understand these in order to understand Graph ML.

Misc

Let’s wrap up this brain dump of mine.

Graph visualization toolkit, datasets, where to next?

Here I'll list some of the other things you should care about. It's not only about papers. Tooling is super important. Data is arguably even more important than the algorithms. Here we go!

This blog contains a nice overview of different graph drawing packages:

Large Graph Visualization Tools and Approaches

What to do, if you need to visualize a large network graph but all tools you try can only draw a hairball or eat all…

towardsdatascience.com

I ended up using **igraph** as it seemed like the most powerful graph drawing tool which I could use directly from Python.

Here is a nice beginner friendly intro to igraph and here is the official tutorial (it's pretty good I've used it)

And again, graph drawing is a subfield for itself! You can dedicate some of your research years to researching only this topic. Similarly to how tokenization in transformers is a subfield for itself even though we usually use these "tools" as a black box.

While we're talking about visualizations I think I should mention the graph datasets!

First of, you'll be seeing **Cora**, **Citeseer**, **PPI** and **Pubmed** everywhere those are the classical benchmarks (like MNIST in CV). The problem with these is, as people later found out, that they are *treacherous* — we can't exactly discriminate between more expressive methods and the less expressive ones.

So people started advising new graph benchmarks, and we should all be cognizant of how important this is. If you're coming from the CV background, as I do, you know what **ImageNet** did for the CV world.

It even became a metaphor! People say that NLP got its **"ImageNet moment"** with the arrival of big data, transfer learning and deep contextualized representations. So data is a big thing! (pun probably intended)

So what's happening with ImageNet in Graph ML?

Open Graph Benchmark initiative looks very promising, the time will tell!

I'd like to wrap up this mind-breaking blog, which had a **bold** attempt to cover the Graph ML literature, with 2 additional resources you should take into consideration.

If I were you I'd read every single blog from Michael (as I did):

Michael Bronstein - Medium

J. M. Stokes et al., A deep learning approach to antibiotic discovery (2020) Cell 180(4):688-702. What? A graph neural…

medium.com

And you should follow Sergey's newsletter. Sebastian Ruder showed that this format works wonders for the NLP world, so let's see whether Sergey makes the same thing for the Graph ML world!

Graph Machine Learning

The one and only newsletter about the latest research, current trends, and upcoming events in Machine Learning…

graphml.substack.com

We're done with the theory! Put your relaxed hat on and lemme walk you through my GAT project.

Graph Attention Network (GAT) Project

As always, I'm a strong believer that reading through theory is just not enough. You also need to test your understanding, because, how else would you know you're not just overfitting the training dataset?

And what better way to do it than to engineer it from scratch?

> If you can't build it, you don't understand it — somebody, somewhere, somewhen

With that mindset, I decided to implement the Graph Attention Network. It just felt like a natural step since they are a generalization of transformers and I've previously implemented Vaswani's transformer.

I mean: **attention** + **GNNs** = , right?

Additionally, I am lucky enough that I get to chat with *Petar Veličković* (the first author of the GAT paper) on a regular basis — so that made it easier for me to figure out why certain design decisions were made.

So after ~3 weeks of my free time (including weekends and aside from my daily job at Microsoft, I'm crazy I know), I pulled it out:

gordicaleksa/pytorch-GAT

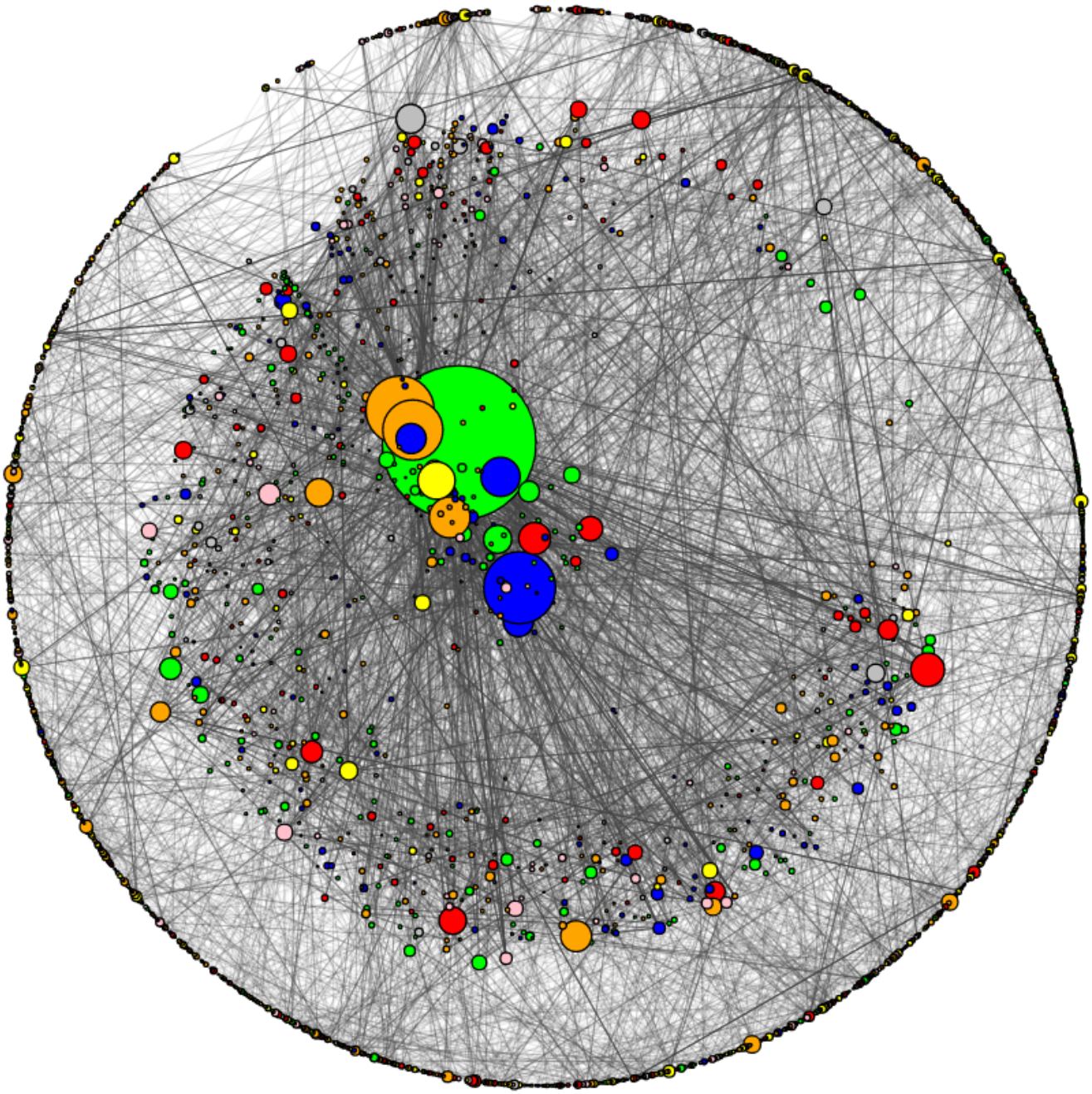
My implementation of the original GAT paper (Veličković et al.). I've additionally included the playground.py file for…

github.com

I've really learned a lot.

I never played with graphs before (aside from implementing some classical CS graph algorithms in C++ back in the days) so this made me really excited.

Understanding graph data and visualizing it gave me a lot of satisfaction. I've broadened my toolkit with new packages such as **igraph** and **NetworkX,** and I created beautiful visualizations such as this one:



Cora citation network visualized

I've used the **Kamada Kawai** layout to draw this one**.** Obviously**,** I got some background in network analysis and graph drawing.

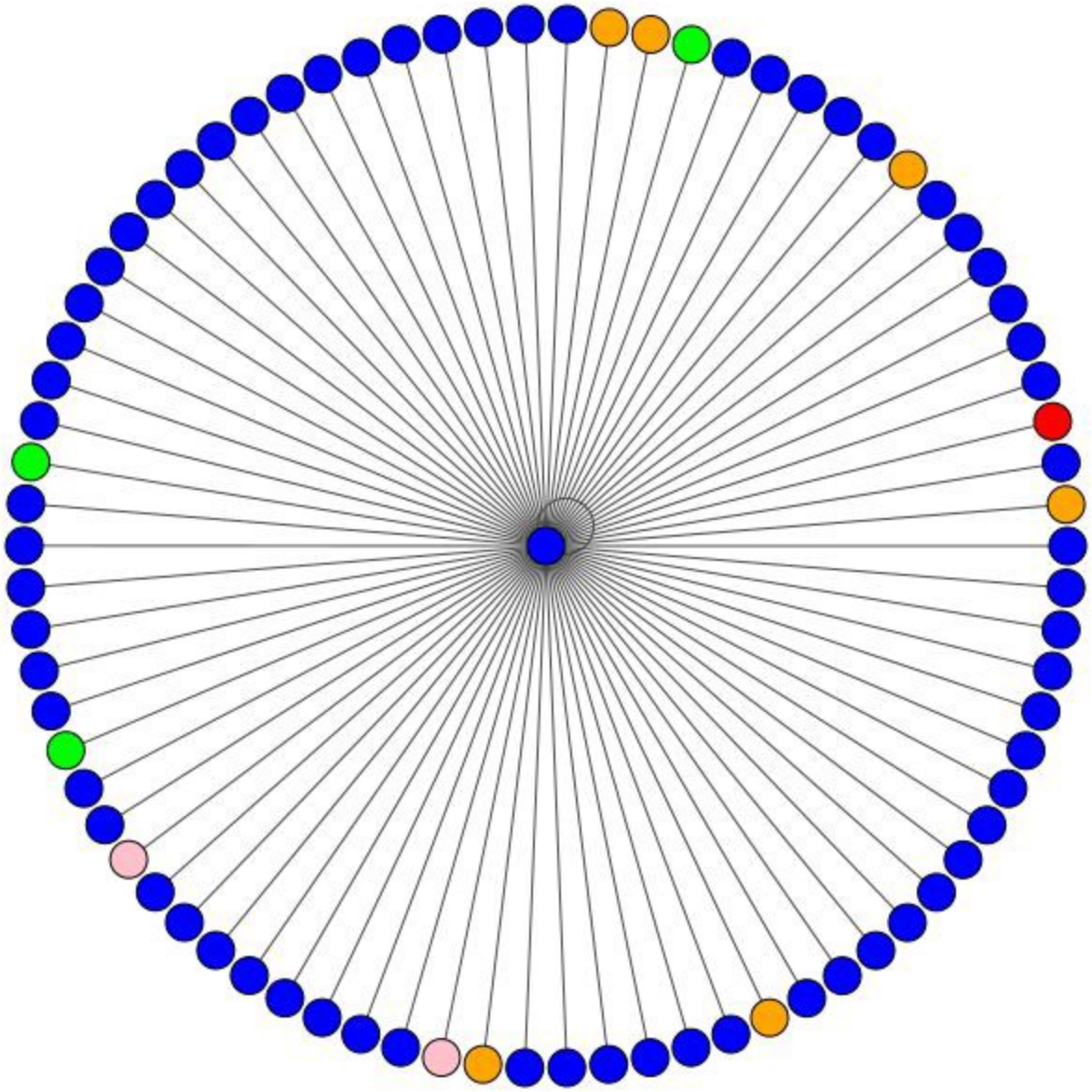
And of course, I've learned how to implement GAT in many different ways (I finally ended up with 3 implementations!). The main tradeoff was between the ease of understanding and efficiency — as it often is.

After I've trained the model I decided to add as many visualizations as possible so that I can better understand what's happening. The 3 main visualizations I did were:

- Embedding visualization via **t-SNE**

- Neighborhood **attention visualization** via edge thickness
- Attention pattern visualization via **entropy histograms**

Once you train the GAT on the Cora dataset here is how a single neighborhood looks like (I've cherry-picked a high-degree node here):



Neighborhood attention visualized (colors = labels, edge thickness ~ attention coefficients)

The first thing you notice is that the edges are of the same thickness — meaning GAT learns constant attention on Cora.

For more details, you can check out the README and the accompanying Jupyter Notebook!

If you need additional help understanding the project I created a project walkthrough video:

GAT project walkthrough (readme, jupyter notebook, Cora, and implementation #3)

I've put an emphasis on explaining the hardest-to-understand implementation (implementation #3 as I've dubbed it) so hopefully, in addition to Jupyter Notebook, this will help you get a deep understanding of how GAT works.

I also created this project update video, while I was working on GAT, which you may find interesting if you're interested in my journey:

GAT project update + my other deep learning projects

Aside from my GAT implementation, I want to mention that you have at your disposal an awesome framework called PyTorch Geometric which you should use if you don't necessarily want to dig into the nitty-gritty details, but you just want to quickly test some GNN out-of-the-box.

PyTorch Geometric is to Graph ML field what HuggingFace is to NLP.

My next steps

Phew. If you made it till the very end, **congrats!**

I hope this blog will inspire you to start exploring Graph ML on your own! Or deep learning/machine learning in general for that matter.

One thing you notice when you start exploring Graph ML is how many related fields (but with a completely new set of terminology, equations, and theory) there are.

Examples would be:

- Geometric deep learning (learning on manifolds) — which is closely related to Graph ML since both are concerned with learning on non-Euclidean domains (graphs/manifolds).
- Equivariance deep learning (exploiting symmetries to make your models statistically efficient i.e. use less data to achieve the same perf) — related to CNN and geometric deep learning fields (I've mostly seen it in this context: group CNNs, spherical CNNs, gauge CNNs (they work on manifolds)) but is also much more generally applicable (equivariant transformers, etc.).
- KG field (knowledge graphs) — which lies somewhere in between graph ML and NLP.

I did some "digressions" into these fields and I tell you — it can get overwhelming. It's not that it's hard it's just that there is a lot of new information and it takes time.

Especially interesting are GDL and equivariance DL where you'll start seeing physics theory perturbing its tentacles and injecting ideas from quantum/particle/classical physics into deep learning.

You find yourself asking questions like how did the general theory of relativity end up here, but there you have it.

When it comes to my next steps nothing has changed since my last blog post update. In a couple of days, I'll start exploring the (deep) RL field!

I'm super excited as some of the most famous AI-breakthroughs happened exactly in RL like AlphaGo/AlphaZero/MuZero/AlphaStar line of research coming from DeepMind.

Be sure that I'll extensively cover all of these on my YT channel and, again, I'll probably implement one of these from scratch, although I'm still not aware of how complex it is to implement something similar — so I'll stay flexible here.