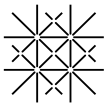


University of Basel

Application of Graph Learning to inverse problems

Master Thesis Preperation

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Data-Analytics
Webpage

Examiner: Prof. Dr. Ivan Dokmanić
Supervisor: Dr. Valentin Debarnot

Cédric Mendelin
cedric.mendelin@stud.unibas.ch
2014-469-274

29.11.2021

Table of Contents

1
Assessment criteria

Written report including:

- Contents of the Master's Thesis project
- Project plan
- summary of relevant related work

2
Foundation

General Questions

- Difference Graph Learning and Graph Representation
- CryoEm, Datasets, Tasks, etc.
- Benchmarking and Dataset

Questions GCN:

- During feature propagation, only node features are considered.
- Spectral Analysis Chapter

Questions Random projection:

- Assumption about uniform(0, 2π) and equally spaced t?
- First-order Chebyshev

Next Steps:

- Familiarize with CryoEm
- Do some coding
- Read paper regarding Manifold Learning

2.1 Introduction to Graph Learning

Graph Representational Learning [1]

2.1.1 Spectral graph theory

Spectral graph theory [9] deals with learning properties and characteristics of graphs, in regard to the graphs eigenvalues and eigenvectors.

[1] https://towardsdatascience.com/introduction-to-graph-representation-learning-a51c963d8d11

2.1.2 Graph deep Learning

Deep Learning with graphs. **TODO: write more**

2.2 Noise

A noisy observation is defined as: $y_n = y + \eta$

2.2.1 Denoising

When we talk from denoising, we want to reconstruct the true observation from a given noisy observation. This reconstruction is done via averaging, which can be performed locally, by the calculus of variations or in the frequency domain.

2.2.1.1 Non local means

Non local means is a state-of-the-art image denoising method [2]. In the name of the method are two important concepts, namely the *mean* and *non local*.
For a given noisy image v, the denoised image is defined as:

$$NL[v](i) = \sum w(i,j)\ v(j) \tag{2.1}$$

where $w(i,j)$ is the weight between pixel i and j and fulfils two conditions:

- $0 \leq w(i,j) \leq 1$
- $\sum_j w(i,j) = 1$

The weight can be seen as a similarity measure of the two pixels. Moreover, these similarities are calculated over square neighbourhoods of the two pixels, where the l2 norm of the neighbourhood is used. Similar pixel neighbourhoods have a large weight and different neighbourhoods have a small weight.
More general, the denoised image pixel i is computed as an weighted average of all pixels in the image, therefore, in a non local way.

2.3 Graph Foundations

A graph is defined as $G = \langle V, E \rangle$, where V is a set of vertices (or nodes) and E is a set of edges (or links). Edges are defined as a set of tuples $\langle i, j \rangle$, where i and j determine the index of the vertices in the graph.

2.3.1 Adjacency Matrix

The adjacency Matrix of G is then defined as follows:

$$A_{ij} = \begin{cases} 1 & \text{if } \langle i,j \rangle \in E \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

2.3.1.1 k-hop neighbourhood

2.3.2 Degree Matrix

The degree Matrix of G is defined as follows:

$$D_{ij} = \begin{cases} deg(v_i) & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

Where $deg(v_i)$ is the degree of the node, formally the number of incoming edges of node v_i.

2.3.2.1 Adjacency normalization

We starting calculating with Matrix A, it is sometimes necessary to normalize. With the degree Matrix D and Adjacency Matrix A, we have all information we need. Mostly, we want to normalize, such that our rows sum to 1.

$$A_{row-norm} = D^{-1}A \tag{2.4}$$

But we can achieve the same for columns, we just need to swap the two matrices:

$$A_{col-norm} = AD^{-1} \tag{2.5}$$

And a final, a probably the most useful normalization, is the symmetric normalization:

$$A_{sym} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \tag{2.6}$$

TODO: Add some nice example

2.3.3 Graph Laplacian

The graph Laplacian is defined as follows:

$$L = D - A \tag{2.7}$$

2.3.3.1 Normalized Graph Laplacian

Symmetric normalized: $L_{sym} = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ Random walk normalized: $L_{rw} = I - D^{-1}A$

2.3.3.2 Normalized Graph Laplacian eigen decomposition

$$\begin{aligned} L_{sym} &= U\Lambda U^T \\ U &= [u_0, \ldots, u_{N-1}] \in R^{N\times N} \\ \Lambda &= diag\left([\lambda_0, \ldots, \lambda_{N-1}]\right) \in R^{N\times N} \end{aligned} \tag{2.8}$$

In this scenario, eigenvectors are also known as *graph Fourier modes* and eigenvalues are known as the *spectral frequencies*.
Moreover, with the Graph Fourier Transform, we can calculate these values from a symmetric graph Laplacian.

2.3.4 Graph Properties

2.3.4.1 Directed vs. undirected vs. weighted

2.3.4.2 Dense and sparse Graph

A dense graph is a graph, where the number of edges in close to the maximal number of edges. Contrarily, a sparse graph only consists of a few edges.

2.3.5 Node Properties

2.3.6 Edge Properties

2.3.7 Graph Construction

TODO: KNN

2.4 Graph Denoising

Data acquired by Real-world observations are often noisy, which can lead to poor performance on data analysis tasks. This observed data can already be in the form of a graph, or a graph can be easily constructed. This resulting graph is what we call a noisy graph, as it includes the noise from the observation.
Graph denoising is the task to reconstruct the original graph from a noisy one. Therefore, graph denoising can be seen as a pre-processing step, where noisy data is filtered.
Denoising in general has often to do with averaging and graphs are a well suited data structure for this task[2].

2.4.1 Noisy Graph

For every noisy graph, there exists an original graph $G = \langle V, E \rangle$.
The noisy graph can be defined as follows:

$$\begin{aligned} G_{noisy} &= \langle V, E_{noisy} \rangle, \\ \text{with } E_{noisy} &= E \setminus E^- \cup E^+, \\ E^- &\subseteq E, \\ E^+ \cap E &= \emptyset \end{aligned} \tag{2.9}$$

Basically, the noisy graph consists of the same vertices as the original graph. From the original graphs edges, some are removed (denoted by E^-) and some new edges are added (denoted by E^+).
The adjacency Matrix of G_{noisy} is then defined as follows:

$$\bar{A}_{ij} = \begin{cases} 1 & \text{if } \langle i, j \rangle \in E_{noisy} \\ 0, & \text{otherwise} \end{cases} \tag{2.10}$$

The task of graph denoising, can therefore be written as:

$$\bar{A} \xrightarrow[method]{Graph-denoising} \tilde{A} \approx A \tag{2.11}$$

Where $\bar{A}$ denotes the noisy input graph, $\tilde{A}$ the denoised graph and A the original graph.

2.4.2 Graph link prediction

Link prediction is a task in Graph learning. The idea is to predict the existence of a link (edge) between two nodes. The task can be formulated as a missing value estimation task. A model M_p is learned from a given set of observed edges. The model finally maps links to probabilities:

$$M_p : E' \rightarrow [0, 1] \tag{2.12}$$

Where E' is the set of potential links.
We define U as the set of all possible vertices of G, therefore $E \subseteq U$. Obviously, one could see Graph denoising as a link prediction problem.
The difference is, that in link prediction, we learn a model from a set of observed links $E_{observed} \subseteq E$ and in Graph denoising we learn the model from $E_{observed} \subseteq U$.
On could also say that link prediction problems are a subset of graph denoising problems.

2.5 Graph Laplacian

2.6 Deep Learning on Graphs

2.6.1 Graph Convolutional Network

Graph Convolutional Networks (GCN) [7] can be used for many tasks in the field of Graph Learning, such as node classification or link prediction. Basically, with GCN, a new feature representation is iteratively learned for the node features.
The basic concept is as follows: For a given graph $G = \langle V, E \rangle$, with node features X^{NxD} and adjacency Matrix A where N denotes the number of nodes and D the number of node input attributes,
a novel node representation Z^{NxF} will be learned, where F is the number of output features. Z will be learned within a neural network, and every layer can be written by the following, non-linear function:

$$\begin{aligned} H^{l+1} &= f(H^l, A), \\ \text{with } H^0 &= X, \\ H^L &= Z, \end{aligned} \tag{2.13}$$

where L is the number of layers in the neural network. The model only differ in the choice of $f(\cdot, \cdot)$.
We are ready to define our first GCN. To keep it simple, $f(\cdot, \cdot)$ will be defined as the following:

$$f(H^l, A) = \sigma(AH^lW^l) \tag{2.14}$$

Where $\sigma(\cdot)$ is a non-linear activation function, such as ReLU and W^l is a weight Matrix of the layer l of the neural network. As Kipf and Welling [7] could show during experiments, this choice of $f(\cdot, \cdot)$ is already very powerful and leads to state-of-the-art results.

2.6.1.1 Renormalization trick

With this model, we do have two problems and need to refine it further. First of all, with the multiplication of A, we average over the neighbour nodes but will ignore the node itself. Therefore, self-loops will be added to A. The second problem is, that A is not normalized and if therefore, when multiplying with A, the features of the nodes will change it scale. Therefore, we need to normalize A such that all rows sum to one. This can be done with a simple multiplication with the D.
These two steps are called the Renormalization trick[7]. First of all, we can simple add the self-loops by adding the Identity Matrix to A, $\hat{A} = A + I$ and $\hat{D}$ is the degree Matrix of $\hat{A}$. Now, we can achieve a symmetric normalization by multiplying $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$.
And finally, we can put all things together, and replace A in the original equation:

$$f(H^l, A) = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^l W^l) \tag{2.15}$$

2.6.1.2 Simple Graph Convolutional Network

Simple Graph Convolutional Network (SGC) [13] proposed a simplified version of GCN. They could verify their hypothesis, that GCN is dominated by the local averaging step and the non-linear activation function between layers do not contribute to much to the success of GCN.
This makes the calculation simpler. We denote $S = \hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$ and can use the fact that in every layer of the neural network, the same computation will take place.

$$\begin{aligned} Z &= S \dots S X W^1 W^2 \dots W^L \\ Z &= S^L X W^1 W^2 \dots W^L \\ Z &= S^L X W \end{aligned} \tag{2.16}$$

where W is the matrix of all vector weights.

2.6.1.3 Link go Graph Laplacian:

In the section, we will have a look at the connection between SGC and Graph Laplacian.
We can define $x \in R^n$ as our signals and define the Fourier transform as $\hat{x} = U^T x$ and the inverse as $x = U\hat{x}$. With the transform, we can easily switch between spatial and Fourier(spectral) domain.
Further, we can define the graph convolution operation between signal x and filter g.

$$g \star x = U((U^T g)(U^T x)) = U\hat{G}U^T x, \tag{2.17}$$

where $\hat{G}$ is a diagonal matrix where the elements are the spectral filter coefficients (eigenvalues?)
The graph convolution can be approximated by the k-th order polynomials of Laplacians:

$$\approx \sum_{i=0}^{k} \Delta^i x = U \left(\sum_{i=0}^{k} \Theta_i \Lambda^i \right) U^T x, \tag{2.18}$$

where $\Delta = D - A$ and Θ_i are filter coefficients which correspond to polynomials of the Laplacian eigenvalues, $\hat{G} = \sum_i \Theta_i \Lambda^i$
In the original [7] paper, the approximation is done with $k = 1$

$$g \star x = \Theta(I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}})x, \tag{2.19}$$

, where Kipf and Welling [7] further applies the renormalization trick, ending up replacing $I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ with $\hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}}$.
$I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ is also called first-order Chebyshev filter.
[2]

not read currently: [3]

2.7 Graph Laplacian Tomography From Unknown Random Projections

Coifman et al. [5] introduces a Laplacian-based algorithm, with which reconstruction of a planar object from projects at random unknown directions is possible.
Overall, in computerized tomography (CT), reconstruction of an object with only samples of its projections is a standard problem. In Coifman et al. [5] the problem was extended by the fact, that the projection angle to the object is unknown.
Formally: Given N projection vectors $(P_{\theta_i}(t_1), P_{\theta_i}(t_2), \ldots, P_{\theta_i}(t_N)$ at unknown angles $\{\Theta_i\}_{i=1}^N$ which are drawn from the uniform distribution of $[0, 2\pi]$ and $t_1, t_2, \ldots, t_n$ are fixed n points (all equally spaced due to uniform distribution) find the underlying density function $\rho(x, y)$ of the object.

2.7.1 Radon transform

The radon transform $P_\Theta(t)$ is the line integral of ρ along parallel lines L at angle Θ and distance t from the orign.

$$\begin{aligned} P_\Theta(t) &= \int_L \rho(x, y) ds \\ &= \int_{-\infty}^{\infty} \rho(x, y)\ \delta(x \cos\Theta + y \sin\Theta - t) dx\ dy \end{aligned} \tag{2.20}$$

An algorithm for estimating angles from given projections have been introduced by [1]. The introduced algorithm consists of three steps:

1. Angle estimation
2. Angle Ordering
3. Joint maximum likelihood refinement of angles and shifts

Step 2 was implemented by some nearest neighbour algorithm. In the work of [5], they introduced a new way of ordering the angles, using Graph Laplacian.

[2] https://towardsdatascience.com/spectral-graph-convolution-explained-and-implemented-step-by-step-2e495b57f801
[3] https://towardsdatascience.com/tutorial-on-graph-neural-networks-for-computer-vision-and-beyond-part-2-be6d71d70f49

2.7.2 Laplace-Beltrami operator

[5] could show, that the graph Laplacian approximates the Laplace-Beltrami operator, if data points are uniformly distributed over the manifold.
Further, they showed that in the case of non-uniformly distributed data points, the Laplacian approximated the backward Fokker-Planck operator (which is a generalization of the Laplace-Beltrami operator). With that, at least the ordering of the angles can be estimated. Finally, with a small normalization of the Laplacian, the Laplace-Beltrami operator can also be approximated in the non-uniform distributed case.

2.7.3 Algorithm

For a given set of projections vector $x_i = (P_{\Theta_i}(t_1), \ldots, P_{\Theta_i}(t_n))$ for $i = 1, 2, \ldots, mN$
The algorithm proposed in [5] consists of five steps:

1. Double the number of projections to 2mN (due to the fact that projections are symmetric)
2. Construct the co-called density invariant Graph Laplacian $\tilde{L}$
3. Compute $\theta_1(i)$ and $\theta_2(i)$ the first two nontrivial eigenvector of $\tilde{L}$
4. Sort x_i according to $\phi_i = \tan^{-1}(\theta_1(i) \,/\, \theta_2(i))$
5. Reconstruct image using the sorted projections and estimated angles.

Where $\tilde{L}$ can be constructed by the following way:

$$W_{ij} = k\left(\frac{\|x_i - x_j\|^2}{2\epsilon}\right), \quad i, j = 1, \ldots, N \tag{2.21}$$

where $||\cdot||$ is the euclidean-norm, k a semi-positive kernel and $\epsilon > 0$ the bandwidth of the kernel. As mentioned in [5], the kernel $k(x) = \exp(-x)$ is a popular choice.
With the newly computed weight Matrix W and the degree Matrix D corresponding to W, we can finally define $\tilde{L}$.

$$\begin{aligned} \tilde{W} &= D^{-1} W D^{-1} \\ \tilde{D} &= \text{Degree matrix corresponding to } \tilde{W} \\ \tilde{L} &= \tilde{D}^{-1}\tilde{W} - I \end{aligned} \tag{2.22}$$

2.8 Folded spectrum Method

Calculation of eigenvalues and eigenvectors of a given Hamiltonian matrix H is a fundamental mathematical problem. Often, we are interested in just the smallest values, which can be efficiently computed. But if we are interested in selected values, this can be hard. H is needed to be diagonalized (bring matrix H into diagonal form) which is computationally expensive and for big matrices impossible.

Currently, the best way to solve such problems is the Folded spectrum (FS) [12] method, which iteratively solves the problem. During calculation, the eigenvalue spectrum will be folded around a reference value ϵ.

$$v^{t+1} = v^t - \alpha(H - \epsilon I)^2 v^t, \tag{2.23}$$

with $0 < \alpha < 1$. When $t \to \infty$, then v^∞ should be the eigenvector with respect to the reference value ϵ.

A reference that I already mentioned in the first mail: standard approach that we need to compare with. Maybe their setting (2D tomography with unknown angle) is a good setting to start with.

2.8.1 Manifolds

A manifold is a topological space, where locally Euclidean distances make sense. More formally, a n-dimensional manifold is a topological space where each point has e neighbourhood, that is homeomorphic to an subset of a n.dimensional Euclidean space.
Some example for a 1-D manifold is a line or a circle. 2-D manifolds can already become pretty complex and are basically any surfaces like planes, sphere but also the torus, Klein bottle or others.

2.8.1.1 Manifold assumption

The manifold assumption is a popular assumption for high-dimensional datasets. Even if a given dataset is in high-dimension and consists of many features, one can assume, that these data points are samples from a low-dimensional manifold, which embeds the high-dimensional space.
Therefore, if one can approximate the underlying Manifold, one solved a dimensionality reduction as we can embed the data points in the low-dimensional manifold space.
There is a complete area of research devotes to this manifold assumption, and basically we then talk from Manifold Learning.
TODO: write more [3]

2.8.2 Power Iterations

Power iteration (also called power method) is a iteratively method, which approximates the biggest eigenvalue of a diagonalizable matrix A.
TODO

3
TODO

3.1 Foundations todo

3.1.1 cryoEm

- Graph construction with k-NN
- Curvature
- Convolution
- Fourier transform and Fourier frequencies
- SO(3), S
- Manifold assumption
- Power Iterations
- Non linear dimensionality reduction

Nice to have:

- Hilbert Space
- Signal Processing
- LIE Group
- Local PCA
- Curvature
- Graphlets
- Link prediction
- Geodesic distance
- Katz Index

- Eigenvector centrality
- NN forward passing
- NN dropout after layer

3.2 Convolution

$$(f \star g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{3.1}$$

3.2.1 Eigenvector centrality

3.2.2 NN forward passing

$$H^{i+1} = \sigma(W^i H^i + b^i) \tag{3.2}$$

[4]

[4] https://towardsdatascience.com/understanding-graph-convolutional-networks-for-node-classification-a2bfdb7aba7b

4

Paper notes

4.1 Diffusion Maps:

Coifman and Lafon [4] [4]

Dimensionality reduction: In essence, the goal is to change the representation of data sets, originally in a form involving a large number of variables, into a low-dimensional description using only a small number of free parameters.

meaningful structures in data sets: Analogous to the problem of dimensionality reduction is that of finding meaningful structures in data sets. The idea here is slightly different and the goal is to extract relevant features out of the data in order to gain insight and understanding of the phenomenon that generated the data.

Markov Chain:

Random walk:

PageRank: Stationary distribution of random walk

Kernel eigenmap methods: - local linear embedding - Laplacian eigenmaps, - hessian eigenmaps - local tangent space alignment

The remarkable idea emerging from these papers is that eigenvectors of Markov matrices can be thought of as coordinates on the data set. Therefore, the data, originally modelled as a graph, can be represented (embedded) as a cloud of points in a Euclidean space. two major advantages over classical dimensionality reduction (PCA, MDS): The first aspect is essential as most of the time, in their original form, the data points do not lie on linear manifolds. The second point is the expression of the fact that in many applications, distances of points that are far apart are meaningless, and therefore need not be preserved.

Unnormalized Graph Laplacian: $L = D - W$

Normalized Graph Laplacian construction: $L_{sym} = D^{-1/2}LD^{-1/2} = I - D^{-1/2}WD^{-1/2}$

$L_{rw} = D^{-1}L = I - D^{-1}W$

Markov chain has a stationary distribution. If graph is connected, stationary is unique. If X is finite, is ergodic.

Diffusion distance: Diffusion map ψ embeds the data into the Euclidean space so that in this space, the Euclidean distance is equal to the diffusion distance.

Laplace–Beltrami operator on manifolds

What are diffusion maps

4.1.1 Vector Diffusion Maps (VDM)

[8] VDMis a mathematical and algorithmic generalization of diffusion maps and other nonlinear dimensionality reduction methods, such as LLE, ISOMAP, and Laplacian eigenmaps. While existing methods are either directly or indirectly related to the heat kernel for functions over the data, VDM is based on the heat kernel for vector fields.
Main concept: Edge consists of weight and linear orthogonal transformation. If linear orthogonal transformation is big, nodes are more like to be equal. If small, there are different Diffusion is calculated on vectors fields, where tangents are mapped to the manifold. A way to globally connect Local PCAs.
SNR: signal-to-noise-ratio
LLE: ISOMAP: Laplacian eigenmaps:

4.1.2 Riemannian Manifold Assumption:

One of the main objectives in the analysis of a high-dimensional large data set is to learn its geometric and topological structure. Even though the data itself is parametrized as a point cloud in a high-dimensional ambient space R^p, the correlation between parameters often suggests the popular "manifold assumption" that the data points are distributed on (or near) a single low-dimensional Riemannian manifold Md embedded in Rp, where d is the dimension of the manifold and $d << p$.

4.1.3 Multi-Frequency Vector Diffusion Maps (MFVDM)

[6] **For a direct link between manifold embedding and tomography, very close to what Ivan explained this morning. If we have a graph denoising method, we will need to compare with this approach (or the original vector diffusion maps).**
Basically same as VDM, but with multiple frequencies per edge.
Diffusion maps (DM) only consider scalar weights over the edges and the vector diffusion maps (VDM) only take into account consistencies of the transformations along connected edges using only one representation of $SO(2)$, i.e. $e^{ia_{i,j}}$. In this paper, we generalize VDM and use not only one irreducible representation, i.e. $k = 1$, but also higher order k up to k_{max}.

4.2 Learning to Drop

Graph denoising

4.3 WalkPooling

Image denoising

4.4 Point Clouds

4.4.1 Dynamic graph Cnn for learning on point clouds

One of the few reference related to graph neural network and learning of graph structure.

4.4.2 CryoEm and related

2. Estimation of Orientation and Camera Parameters from Cryo-Electron Microscopy Images with Variational Autoencoders and Generative Adversarial:
learning framework where the manifold embedding is estimated.
3. Computational Methods for Single-Particle Cryo-EM: review around cryo-EM.
This reference doesn't talk about manifold embedding, but it is a nice one if you want to know more about the acquisition system and standard approaches to solve the cryo-EM problem.
3.bis) Single-Particle Cryo-Electron Microscopy: another review similar to the previous one. The section "Mathematical frameworks for cryo-EM data analysis" and especially the subsection MRA (multireference alignement) introduce a toy model that is related to cryo-EM and where the symmetries are of importance.
4. Bispectrum Inversion with Application to Multireference Alignment: for a paper that introduce several algorithms to solve MRA.

5 Introduction

This is the introduction to the thesis template. The goal is to give students a starting point on how to format and style their Bachelor or Master thesis[5].
Please make sure to always use the most current version of this template, by downloading it always from the original git repository:

http://www.github.com/ivangiangreco/unibas-latex

We will use throughout this tutorial some references to Turing's imitation game [11] and the Turing machine [10]. You may be interested in reading these papers.

The package comes with an option regarding the bibliography style. You can include the package with

```
\usepackage[citeauthor]{basilea}
```

to be able to cite authors directly with

```
\citet{turing:1950}
```

If the option is enabled, then the following reference should print Turing [2]: Turing [11]

[5] This document also shows how to use the template.

6 Body of the Thesis

This is the body of the thesis.

6.1 Structure

6.1.1 Sub-Section

6.1.1.1 Sub-Sub-Section

Paragraph

Even Sub-Paragraph This is the body text. Make sure that when you reference anything you use labels and references. When you refer to anything, you normally capitalise the type of object you reference to, e.g. Section 6.1 instead of section 6.1. You may also just use the `cref` command and it will generate the label, e.g., for Section 6.1, we did not specify the word "Section".
Hint: Try to structure your labels as it is done with `sec:my-label` and `fig:machine`, etc.

6.2 Equations

A Turing Machine is a 7-Tuple:

$$M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle \tag{6.1}$$

A Turing Machine is a 7-Tuple even if defined in the text, as in $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$.

6.3 Tables

Some tables can also be used as shown in Table 6.1[6]. Remember that tables might be positioned elsewhere in the document. You can force positioning by putting a `ht!` in the definition.

[6] Table captions are normally above the table.

Table 6.1: Frequency of Paper Citations. By the way: Make sure to put the label always after the caption, otherwise LaTeX might reference wrongly!

Title	*f*	Comments
The chemical basis of morphogenesis	7327	
On computable numbers, with an application to the ...	6347	Turing Machine
Computing machinery and intelligence	6130	

6.4 Figures

Figures are nice to show concepts visually. For organising well your thesis, put all figures in the Figures folder. Figure 6.1 shows how to insert an image into your document. Figure 6.2 references a figure with multiple sub-figures, whereas the sub-figures are referenced by Fig. 6.2(a), etc.

Missing: Description figure.

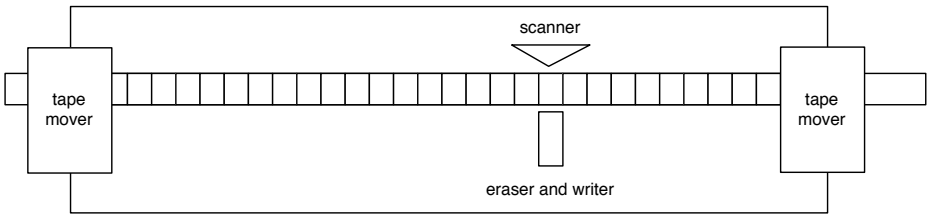
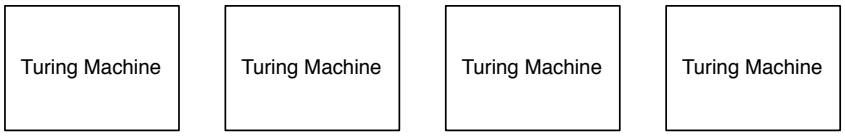


Figure 6.1: A Turing machine.



(a) Turing Machine 1 (b) Turing Machine 2 (c) Turing Machine 3 (d) Turing Machine 4

Figure 6.2: Plots of four Turing machines

6.5 Packages

These packages might be helpful for writing your thesis:

`caption` to adjust the look of your captions

`glossaries` for creating glossaries (also list of symbols)

`makeidx` for indexes and the back of your document

`algorithm, algorithmicx, algpseudocode` for adding algorithms to your document

7 Conclusion

This is a short conclusion on the thesis template documentation. If you have any comments or suggestions for improving the template, if you find any bugs or problems, please contact me.

Good luck with your thesis!

Bibliography

[1] Samit Basu and Yoram Bresler. Feasibility of tomography with unknown view angles. *IEEE Transactions on Image Processing*, 9(6):1107–1122, 2000.

[2] Antoni Buades, Bartomeu Coll, and J-M Morel. A non-local algorithm for image denoising. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 60–65. IEEE, 2005.

[3] Lawrence Cayton. Algorithms for manifold learning. *Univ. of California at San Diego Tech. Rep*, 12(1-17):1, 2005.

[4] Ronald R Coifman and Stéphane Lafon. Diffusion maps. *Applied and computational harmonic analysis*, 21(1):5–30, 2006.

[5] Ronald R Coifman, Yoel Shkolnisky, Fred J Sigworth, and Amit Singer. Graph laplacian tomography from unknown random projections. *IEEE Transactions on Image Processing*, 17(10):1891–1899, 2008.

[6] Yifeng Fan and Zhizhen Zhao. Multi-frequency vector diffusion maps. In *International Conference on Machine Learning*, pages 1843–1852. PMLR, 2019.

[7] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[8] Amit Singer and H-T Wu. Vector diffusion maps and the connection laplacian. *Communications on pure and applied mathematics*, 65(8):1067–1144, 2012.

[9] Daniel Spielman. Spectral graph theory. *Combinatorial scientific computing*, 18, 2012.

[10] Alan M Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 42(2):230–265, 1936.

[11] Alan M Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[12] Lin-Wang Wang and Alex Zunger. Solving schrödinger's equation around a desired energy: Application to silicon quantum dots. *The Journal of Chemical Physics*, 100(3): 2394–2397, 1994.

[13] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.