GEOMETRIC DEEP LEARNING

Computational Geometry Project

Team 10
Soulounias Nikolaos (sdi1600156)
Iyamu Perisanidis Simon (sdi1600051)

TABLE OF CONTENTS

01

INTRODUCTION

INTRODUCTION

This presentation was made in the context of the final project of the Computational Geometry course of the Department of Informatics and Telecommunications of NKUA 2020.

The goal of this presentation is to preview what geometric deep learning is and show a real world application as well as efficiency.

The presentation is divided into 6 sections, as shown in the next slide.

National and Kapodistrian University of Athens

PRESENTATION STRUCTURE

SECTIONS 2-3

In these sections we will grasp some basic background knowledge about deep learning and more specifically geometric deep learning.

SECTIONS 4-5

Sections 4-5 are about our implementation of the method. We will preview the dataset we used and some key notes of our code such as the NN architecture.

SECTION 6

Finally, in section 6 the results and concluding remarks are stated.

02

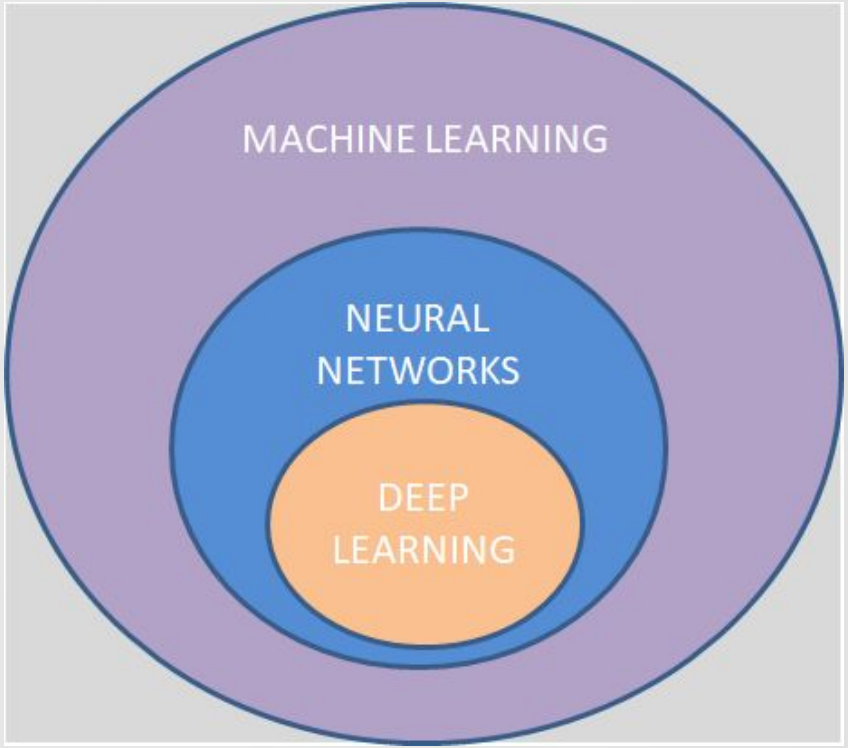
WHAT IS DEEP LEARNING?

Deep Learning Origins

Neural Networks (or Artificial Neural Networks) are a class of Machine Learning models.

Deep Learning (or Deep Structured Learning) is part of a broader family of machine learning methods based on artificial neural networks.

The terms **Neural Network** and **Deep Learning** are often used interchangeably.



Machine Learning

Definition

Arthur Samuel, 1959

"[Machine Learning is] the field of study that gives computers the ability to learn without being explicitly programmed."

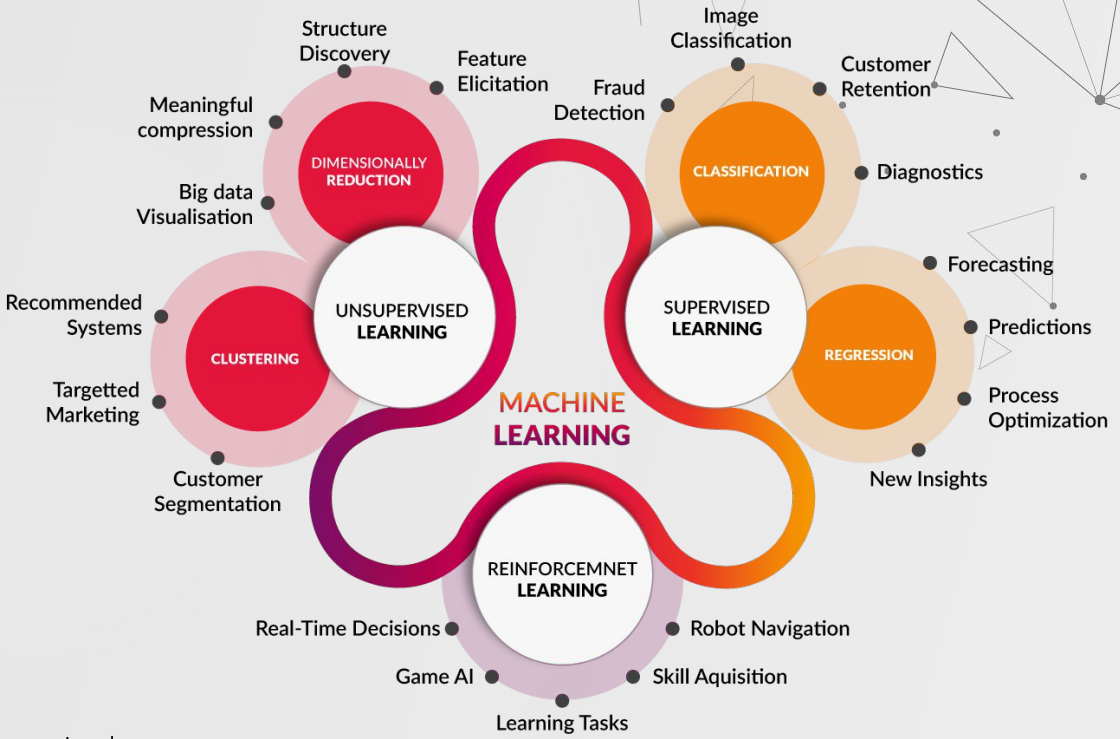
Arthur Samuel was an American pioneer in the field of computer gaming and artificial intelligence. The Samuel Checkers-playing Program was among the world's first successful self-learning programs, and as such a very early demonstration of the fundamental concept of artificial intelligence (AI).

Machine Learning Approaches and Examples

Supervised learning: The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs.

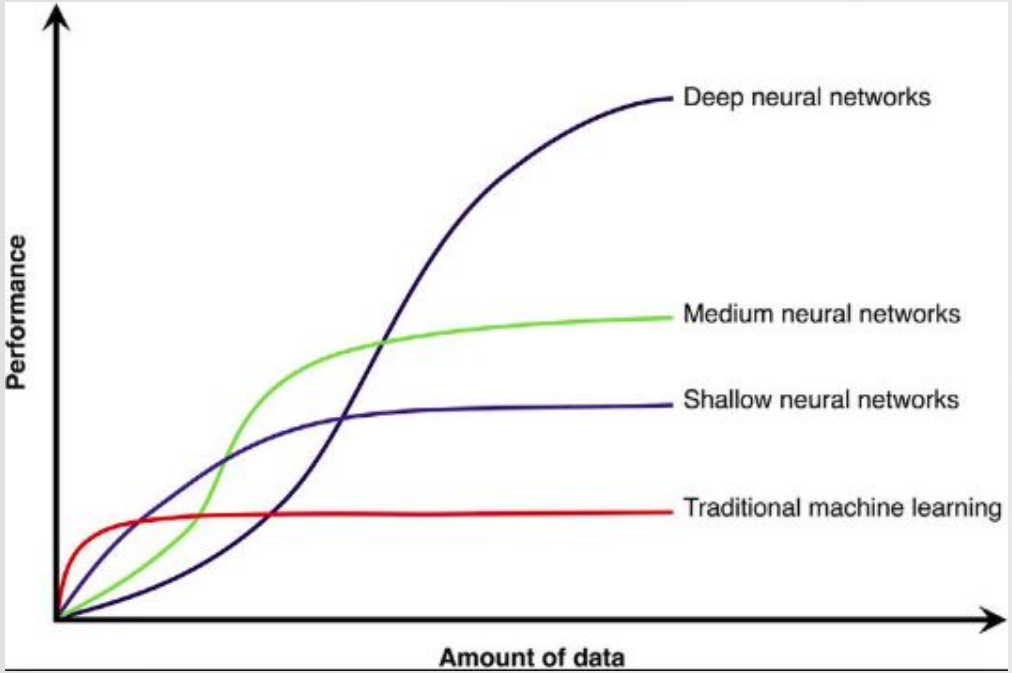
Unsupervised learning: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).

Reinforcement learning: A computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle or playing a game against an opponent) As it navigates its problem space, the program is provided feedback that's analogous to rewards, which it tries to maximise.



Deep Learning can be supervised, unsupervised or semi-supervised.

Why Deep Learning?



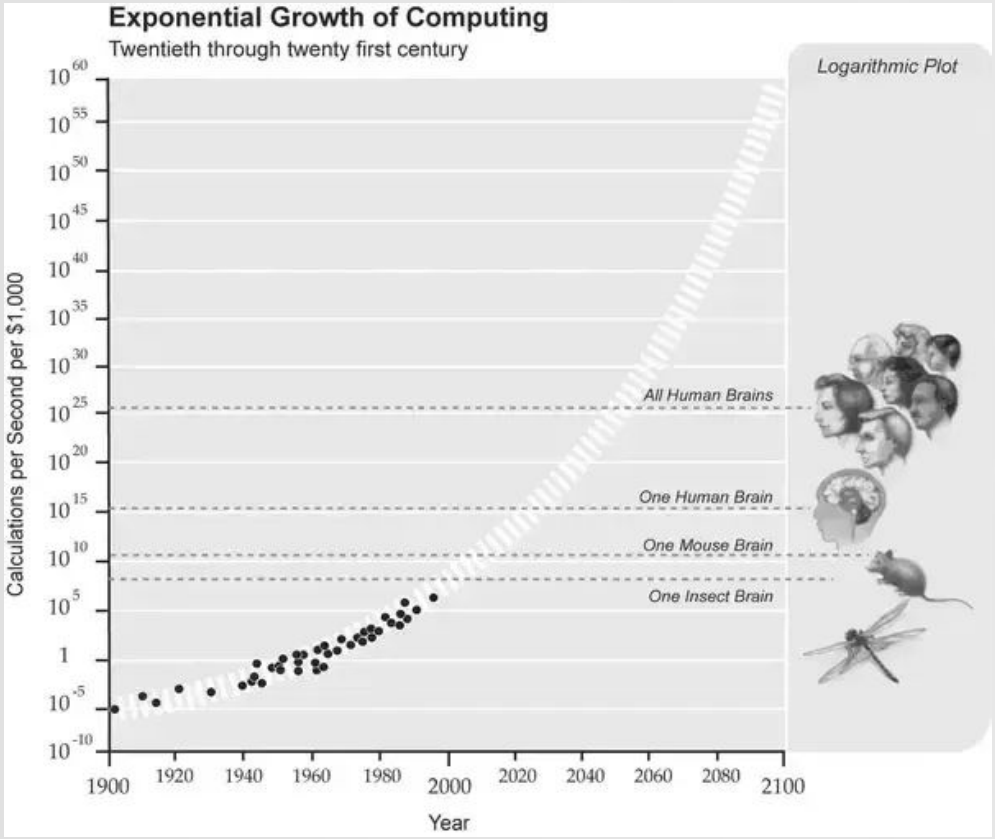
Machine Learning isn't a new field. It existed since the 1950s.

Even some Neural Network theoretical concepts existed for a while back.

Then why is Deep Learning just taking off?

- **Data**
  One of the things that increased the popularity of Deep Learning is the massive amount of data that is now available. The amount of data collected drastically increased over the last 20 years, at the beginning because the Internet became mainstream and then because of the Internet of Things (sensors, mobile phones, etc). This enables Neural Networks to really show their potential since they get better the more data you fed into them.
- **Computational Power**
- **Algorithms**

Why Deep Learning?



Machine Learning isn't a new field. It existed since the 1950s.

Even some Neural Network theoretical concepts existed for a while back.

Then why is Deep Learning just taking off?

- **Data**
- **Computational Power**
  Another very important reason is the computational power that is available nowadays, which enables us to process more data. According to Ray Kurzweil, a leading figure in Artificial Intelligence, computational power is multiplied by a constant factor for each unit of time (e.g., doubling every year) rather than just being added to incrementally. This means that computational power is increasing exponentially.
- **Algorithms**

Why Deep Learning?

The 2018 ACM A.M. Turing Award goes to deep learning pioneers Yann LeCun, Yoshua Bengio, and Geoffrey Hinton for their conceptual work that has demonstrated the practical advantages of deep neural networks.

Their early work has laid the foundation for the incredible breakthroughs in fields such as robotics, computer vision, and natural language processing.

Machine Learning isn't a new field. It existed since the 1950s.

Even some Neural Network theoretical concepts existed for a while back.

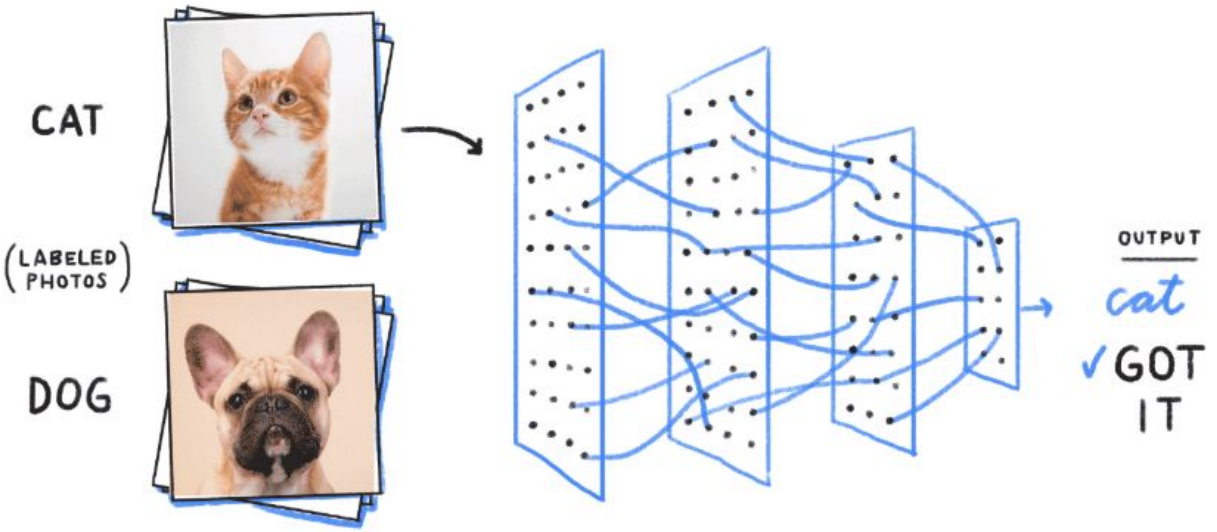
Then why is Deep Learning just taking off?

- **Data**
- **Computational Power**
- **Algorithms**
  The third factor that increased the popularity of Deep Learning is the advances that have been made in the algorithms itself. These recent breakthroughs in the development of algorithms are mostly due to making them run much faster than before, which makes it possible to use more and more data.

Deep Learning Achievements

Deep Neural Networks perform remarkably well on:

- **Images and Videos**
- **Text**
- **Audio**
- **and other euclidean data**



03

WHAT IS GEOMETRIC DEEP LEARNING

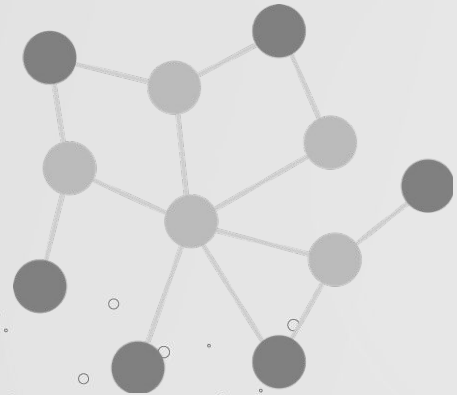
Deep learning on non-euclidean data

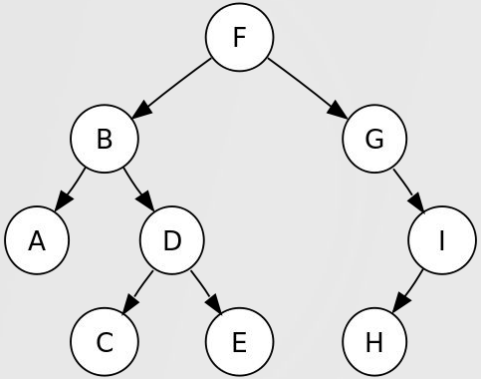
GEOMETRIC DEEP LEARNING

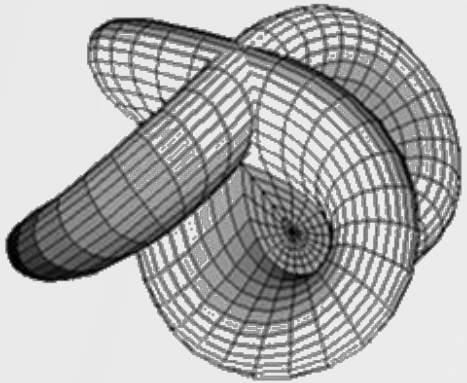
Geometric deep learning is a new field of machine learning that can learn from complex data like graphs and multi-dimensional points.

NON-EUCLIDEAN DATA

As we saw, Neural Networks are good at handling euclidean data, such as images, text and audio. Apart from euclidean of course, there are data like graphs, trees and 3D objects that belong to the non-euclidean space.

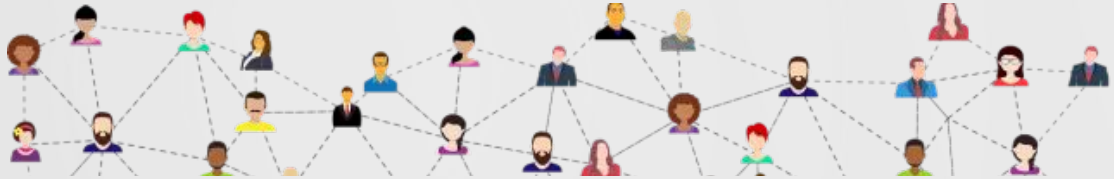




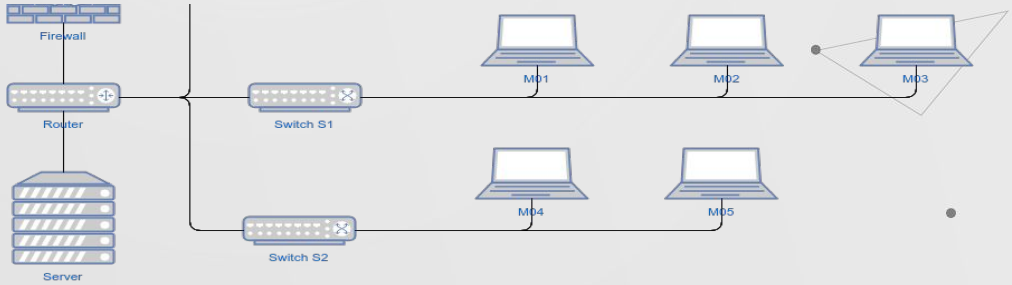


NON-EUCLIDEAN DATA

These types of data come in handy in many real world applications. For example social networks can be modeled as graphs, where each user is a node and each interaction between users is an edge.

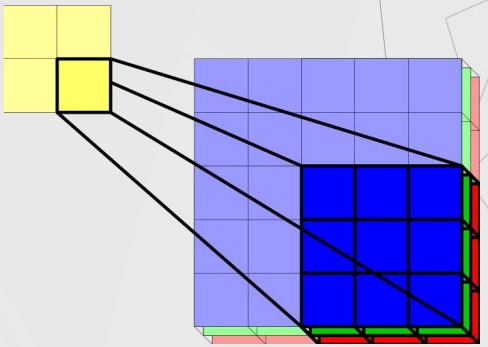


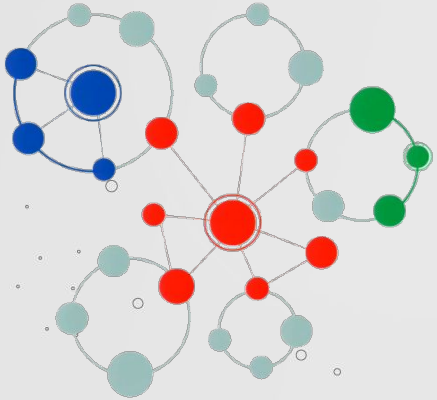
Another example is that of a computer network. It can be considered as a graph, where the nodes are computers and the edges are connections with other computers.



CONVOLUTION

The problem with this type of data is that traditional deep neural network are not able to parse it correctly. The reason is that most of these networks are based on convolutions. Convolutions work well on euclidean data, such as images, because they exploit the intensity and the position of each pixel in the 2D space.





This however cannot be applied to non-euclidean data. For example, different vertices in a graph can contain very different neighbourhoods, that can vary on the number of neighbours or the connectivity. This makes it impossible to apply convolution as we would do on euclidean domains.

HOW

There are two main approaches in order to work with non-euclidean data.

Spectral approach

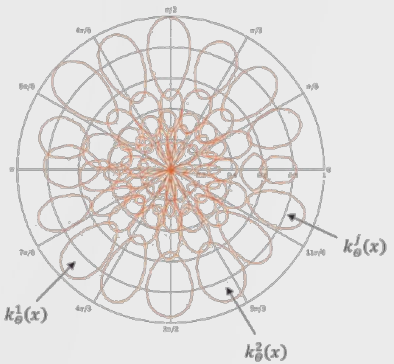
The idea is to generalize the Fourier transform theorem for graph and manifold data in order to perform convolution on the spectral domain. The generalization of the Fourier transform uses the already defined eigenfunctions of graph laplacian as bases for the Fourier transform.

$\Phi diag(\vartheta)\Phi^T f$

$\Psi diag(\vartheta)\Psi^T f$

Spatial approach

The main idea is to apply a template on a neighborhood representation which is obtained by mapping the neighbors on a fixed structure. This is the same idea of applying a convolution over images, the difference is that on images the neighbour structure is constant for all vertices.



04

OUR EXPERIMENT

Dataset and classification goal

Product Classification

Humans inevitably develop a sense of the relationships between objects. Some pairs of objects might be seen as being alternatives to each other (such as two pairs of jeans), while others may be seen as being complementary (such as a pair of jeans and a matching shirt).

As a result some products are frequently bought together more than others.

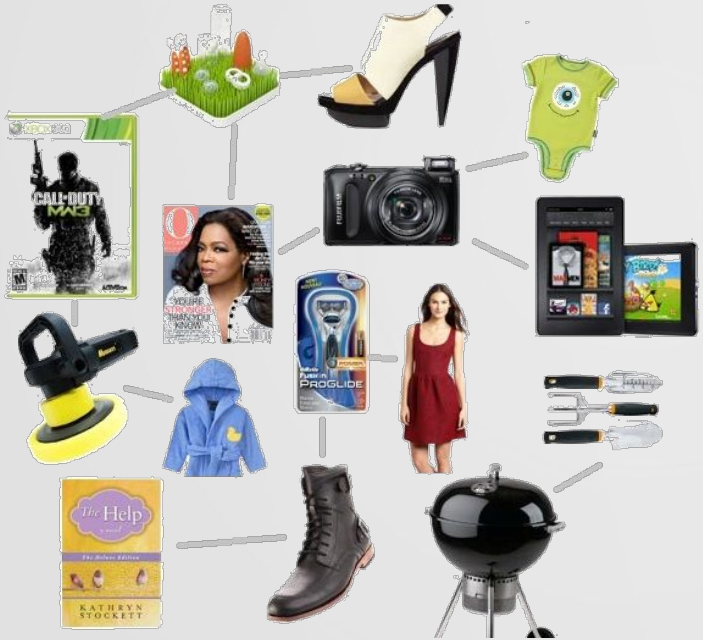
Product Classification

The goal of our experiment is, given a number of products, their online reviews and how frequently two of them are bought together, to classify them in categories.

For example the input set could contain books, electronics, toys e.t.c.

The goal of our model would be to find out which product belongs to which category.

Product Classification



A good way to represent our data is a graph, where the nodes represent the products and the edge weights represent how frequently two products are bought together.

For this experiment, we will use the “Amazon Computers” dataset which was presented in the paper by Shchur et al [2].

It consists of a graph of 13381 nodes(products) and 245778 edges between them. Each node has 767 features which derive from the online reviews of the products.

Bag-of-words

The features of each node should be the reviews of the product. However, even though our models are excellent at handling numbers, they are not familiar with our natural language. Therefore the reviews must be represented as numbers.

The representation that is used is called Bag-of-Words.

Bag-of-words

The main idea is to count the occurences of the most popular words, so that we can feed those to the network instead.

The features of the dataset are represented in a two dimensional matrix, where columns are the features and rows are the nodes(products). Each node has 767 features. That means that the 767 most common words of the reviews were chosen (excluding the stop words). Therefore a cell in the position [i,j] in the matrix is the count of occurences of the i-th word of the bag in the reviews of the j-th product.

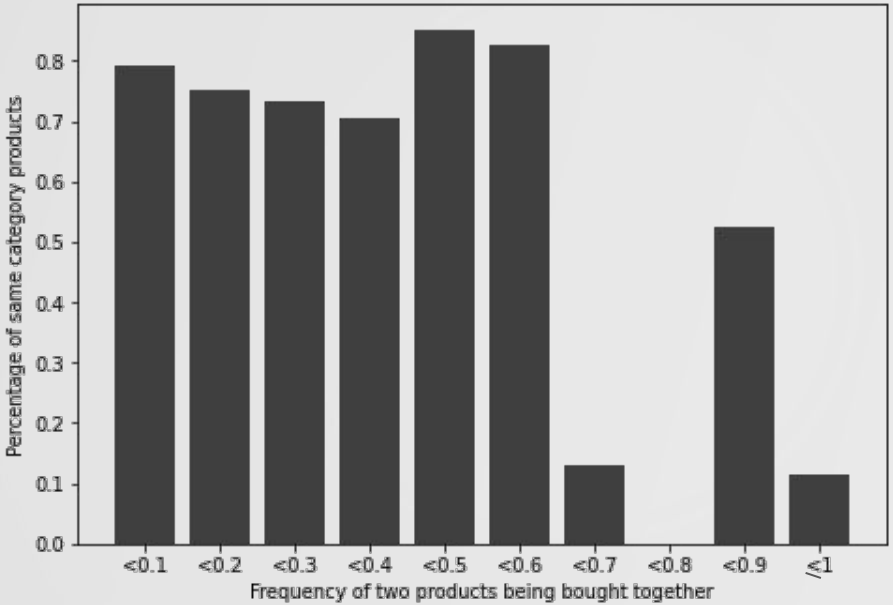
Relations between our data

Given our dataset, a question that might come to mind is whether the frequency of two products being bought together is related with their category.

Using python, we will plot a Histogram that will answer our concern of whether two items that are frequently bought together are also more likely to belong in the same class.

Relations between our data

The x axis is for the frequency of two products being bought together (divided into 10 bars) and the y axis is the percentage of items that belong in the same category.



As we see, the items that are frequently bought together, are less likely to belong in the same category.

Therefore we can not just apply a simple method such as linear programming for our task. We need a more powerful method that also takes advantage of the nodes features.

05

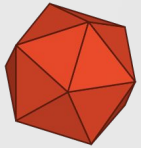
IMPLEMENTATION

Libraries used and NN architecture

IMPLEMENTATION NOTES

In machine learning, Python is the way to go, since its quick and simple. We will use Jupyter notebooks to demonstrate our work.

For our implementation we used PyTorch Geometric, which is a geometric deep learning extension library for PyTorch.



PyTorch geometric

PyTorch Geometric

Pytorch Geometric is an open-source library that consists of various methods for deep learning on graphs from a variety of published papers.

Additionally, there is a mini-batch loader, a large number of common benchmark datasets, and helpful transforms for graphs, 3D meshes and point clouds.

```
pip install torch-geometric
```

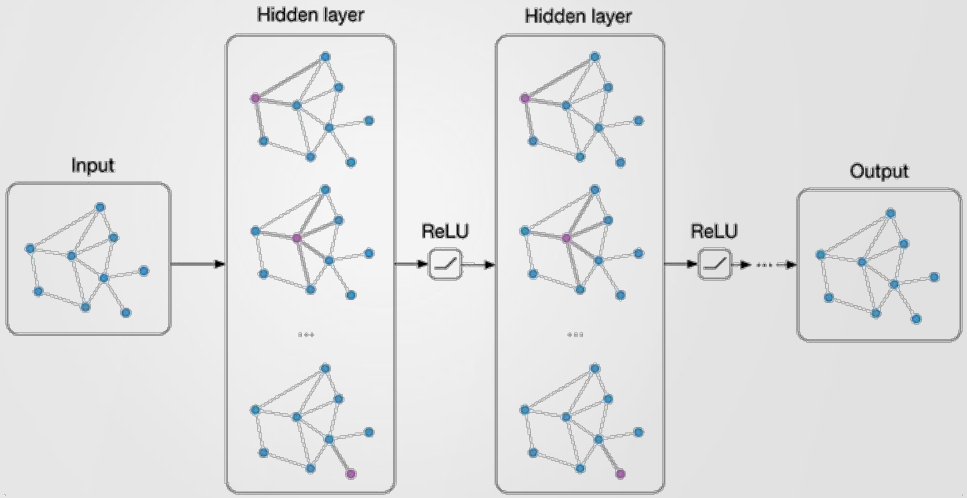
GRAPH NEURAL NETWORK

```
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = SplineConv(dataset.num_features, 16, dim=1, kernel_size=2)
        self.conv2 = SplineConv(16, dataset.num_classes, dim=1, kernel_size=2)

    def forward(self):
```

Our NN architecture

Our network consists of two convolutional layers. PyTorch geometric has many options when it comes to choosing a convolutional layer. (ex. GCNConv, ChebConv, SplineConv, ...). These layers transform the input graph data, in a way that convolution can be applied. In this experiment we used SplineConv since we found out that it works well on our data.



Let’s check out the code

Opening the Jupyter Notebook

BACK END

Now we will present the math operations that hapoend in the background in the implementation of SplineConv.

SplineCNN: Introduction

- SplineConv method is implemented in the **PyTorch Geometric(PyG)** library.

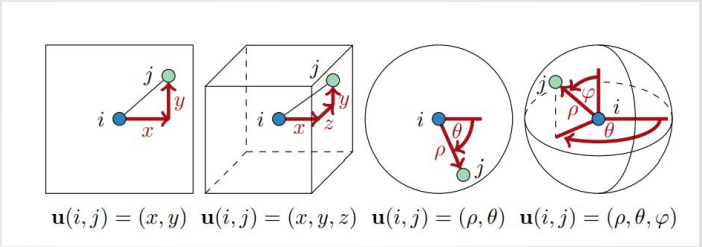
github repo: rusty1s/pytorch_geometric: Geometric Deep Learning Extension Library for PyTorch

- The SplineConv method implementation is based on the paper::
SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels by **Matthias Fey , Jan Eric Lenssen , Frank Weichert, Heinrich Muller**

SplineCNN: Introduction

- SplineCNN follows a **spatial approach**: Convolution is performed in local Euclidean neighborhoods.

- Local positional relations between points are represented, for example, as polar, spherical or Cartesian coordinates. The local positional relations are stored as **pseudo-coordinates** for each edge (i,j) of the graph.



SplineCNN: Input Graph

The input of the spliceCNN convolution operator is a **directed** graph ***G=(V,E,U)***.

- **V={1,2,..,N}** is the **set of the N nodes** of the graph G.
- **E ⊆ VxV** is the **set of edges** of the graph G.
- **$U \in [0, 1]^{NxNxd}$** is a **NxNxd matrix** containing **d-dimensional pseudo-coordinates $u(i,j) \in [0, 1]^d$** for each directed edge **$(i, j) \in E$**.

  U can be interpreted as an **adjacency matrix** with d-dimensional, **normalized** entries $u(i, j) \in [0, 1]^d$ if $(i, j) \in E$ and 0 otherwise.

  U is usually **sparse** with $|E| << N^2$ entries.

SplineCNN: Input Node Features

- Each node **i ∈ V={1,2,..,N}** has M_{in} features.
  Each feature of a node i is represented in $\mathbb{R}$.

- Consider a **feature extraction function** $f : V \to \mathbb{R}^{M_{in}}$.

  f(i) denotes a **vector of M_{in} input features of the node i ∈ V.**

- For each $1 \leq j \leq M_{in}$ we reference the set **$\{f_j(i) \mid i \in V\}$** as **j-input feature map**.

B-spline basis functions

- Let U be a set of $n + 1$ non-decreasing numbers, $u_0 \leq u_1 \leq u_2 \leq \ldots \leq u_n$.
  The u_i's are called ***knots***, the set U ***knot vector***, and the half-open interval $[u_i, u_{i+1})$ the ***i-th knot span***.
  Note that since some u_i's may be equal, **some knot spans may not exist**.

- If a knot $\boldsymbol{u_i}$ **appears** $\boldsymbol{k}$ **times** (*i.e.*, $u_i = u_{i+1} = \ldots = u_{i+k-1}$), where $k > 1$, u_i is a ***multiple knot* of *multiplicity* $\boldsymbol{k}$**, written as $\boldsymbol{u_i(k)}$. Otherwise, if u_i **appears only once**, it is a ***simple knot***.

- If the knots are **equidistant** (*i.e.*, $u_{i+1} - u_i$ is a constant for $0 \leq i \leq n - 1$), the knot vector or the knot sequence is said ***uniform***; otherwise, it is ***non-uniform***.

- The knots can be considered as **division points** that **subdivide the interval $[u_0, u_n]$ into knot spans**.
  All B-spline basis functions are supposed to have their domain on $[u_0, u_n]$.

B-spline basis functions

So, we have already defined ***U*** as the **knot vector with n+1 knots $(u_0, u_1, u_2, \dots, u_n)$.**

To define B-spline basis functions, we need one more parameter, the **degree** of these basis functions, ***m***.

The ***j*-th B-spline basis function of degree *m*** is written as $N_{j,m}(u)$.

Cox-de Boor recursion formula:

$$N_{j,0}(u) = \begin{cases} 1, & u_j \le u < u_{j+1} \\ 0, & otherwise \end{cases}$$

$$N_{j,m}(u) = \frac{u - u_j}{u_{j+m} - u_j} N_{j,m-1}(u) + \frac{u_{j+m+1} - u}{u_{j+m+1} - u_{j+1}} N_{j+1,m-1}(u)$$

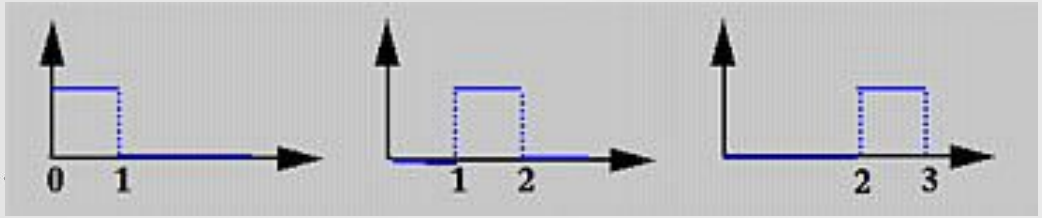
B-spline basis functions

For **degree m = 0**, these basis functions are all ***step functions***.
That is, basis function $N_{j,0}(u)$ is 1 if u is in the i-th knot span $[u_i, u_{i+1})$.

$$N_{j,0}(u) = \begin{cases} 1, & u_j \le u < u_{j+1} \\ 0, & otherwise \end{cases}$$

For example, if we have **four knots $u_0 = 0$, $u_1 = 1$, $u_2 = 2$ and $u_3 = 3$**, then:

1. knot spans are [0,1), [1,2), [2,3) and
2. the basis functions of degree 0 are $N_{0,0}(u)$ = 1 on [0,1) and 0 elsewhere,
   $N_{1,0}(u)$ = 1 on [1,2) and 0 elsewhere and
   $N_{2,0}(u)$ = 1 on [2,3) and 0 elsewhere.



B-spline basis functions

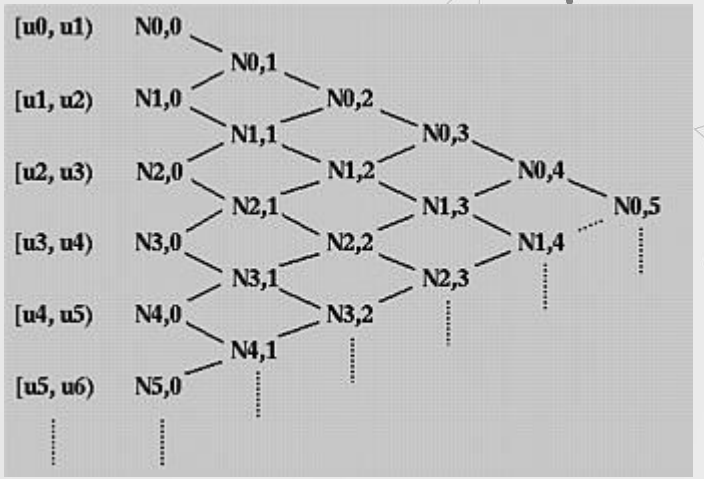
$$N_{j,m}(u) = \frac{u - u_j}{u_{j+m} - u_j} N_{j,m-1}(u) + \frac{u_{j+m+1} - u}{u_{j+m+1} - u_{j+1}} N_{j+1,m-1}(u)$$

For **degree m > 0**, to compute $N_{j,m}(u)$, we must first compute:

- $N_{j,m-1}(u)$
- $N_{j+1,,m-1}(u)$

$N_{j,m}(u)$ is a non-zero function in the interval $[u_j, u_{j+m+1})$.

The $N_{j,m}(u)$ function will have different branches for each of the intervals $[u_j, u_{j+1})$, $[u_{j+1}, u_{j+2})$, …, $[u_{j+m}, u_{j+m+1})$.

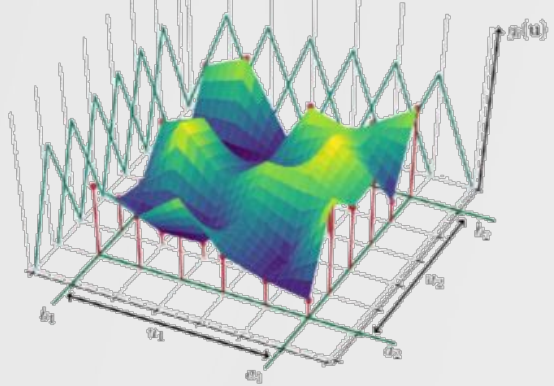


SplineCNN: B-spline basis functions

- Let $\left(\left(N^{m}_{1,i} \right)_{i \, \varepsilon \, [1,k1]} , \left(N^{m}_{2,i} \right)_{i \, \varepsilon \, [1,k2]} , \dots , \left(N^{m}_{d,i} \right)_{i \, \varepsilon \, [1,kd]} \right)$ denote **d B-spline bases of degree m**, based on **uniform, i.e. equidistant, knot vectors**, with **k = (k1, . . . , kd) defining our d-dimensional kernel size**.

- So, $\left(N^{m}_{t,i} \right)_{i \, \varepsilon \, [1,kt]}$ is the **t-th set of B-spline basis functions**, which contains **kt basis functions** of **degree m**.

- The **degree m** of the the B-spline basis functions and **the number of basis function per dimension (k1, . . . , kd)** are hyperparameters.



(a) Linear B-spline basis functions

SplineCNN: Trainable Parameters

- Independent from the type of coordinates stored in U, a **trainable, continuous kernel** function, which **maps** each **u(i, j) to a scalar** that is used as a weight for feature aggregation.

- We introduce a trainable parameter $\mathbf{w_{p,l}} \in \mathbf{W}$ for each element **p from the Cartesian product** $\boldsymbol{P} = \left(\left(\mathbf{N}^{\mathbf{m}}_{1,i} \right)_{i \, \varepsilon \, [1,k1]} \mathbf{x} \left(\mathbf{N}^{\mathbf{m}}_{2,i} \right)_{i \, \varepsilon \, [1,k2]} \mathbf{x} \ \ldots \ \mathbf{x} \left(\mathbf{N}^{\mathbf{m}}_{d,i} \right)_{i \, \varepsilon \, [1,kd]} \right)$ of the B-spline bases and each of the $\mathbf{M_{in}}$ **input feature maps**, indexed by **l**.

  This results in $\mathbf{K = M_{in} \cdot \Pi_{i=1}^{d} k_i}$ trainable parameters.

SplineCNN: Trainable Parameters

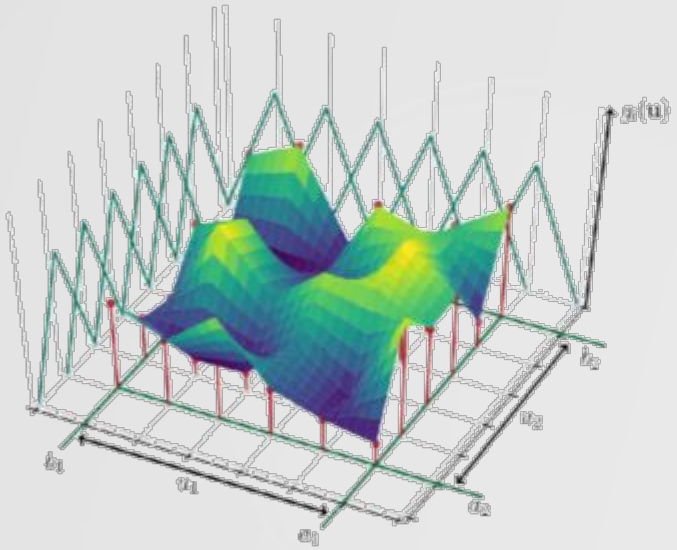
We define our continuous convolution kernel $g_l : [a_1, b_1] \times \cdots \times [a_d, b_d] \to \mathbb{R}$ as functions with

$$g_l(\vec{u}) = \sum_{\vec{p} \in P} w_{\vec{p},l} B_{\vec{p}}(\vec{u})$$

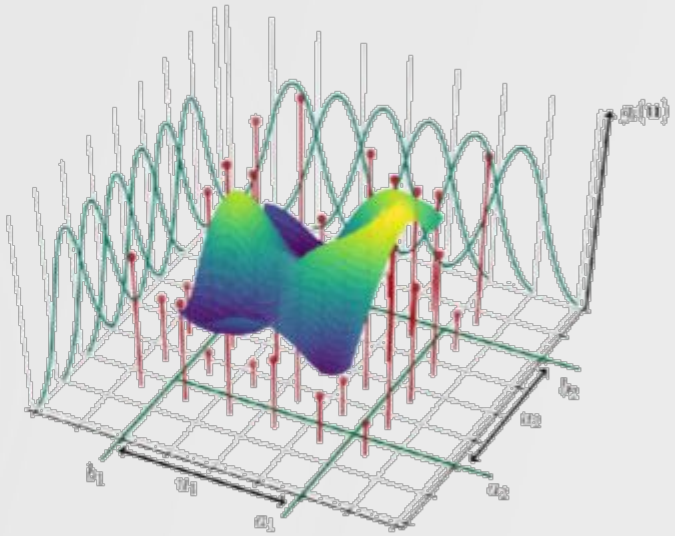
with $B_{\vec{p}}(\vec{u})$ being the product of the basis functions in $\vec{p}$:

$$B_{\vec{p}}(\vec{u}) = \prod_{i=1}^{d} N_{i,p_i}^{m}(u_i)$$

SplineCNN: Trainable Parameters



(a) Linear B-spline basis functions



(b) Quadratic B-spline basis functions

Figure 3: Examples of our continuous convolution kernel for B-spline basis degrees (a) $m = 1$ and (b) $m = 2$ for kernel dimensionality $d = 2$. The heights of the red dots are the trainable parameters for a single input feature map. They are multiplied by the elements of the B-spline tensor product basis before influencing the kernel value.

SplineCNN: Convolution Operator

Given:

1. the kernel functions $\vec{g} = (g_1, g_2, \ldots, g_{M_{in}})$ and
2. input node features $\vec{f} = (f_1, f_2, \ldots, f_{Min})$,

the spline convolution operator for a node is defined as:

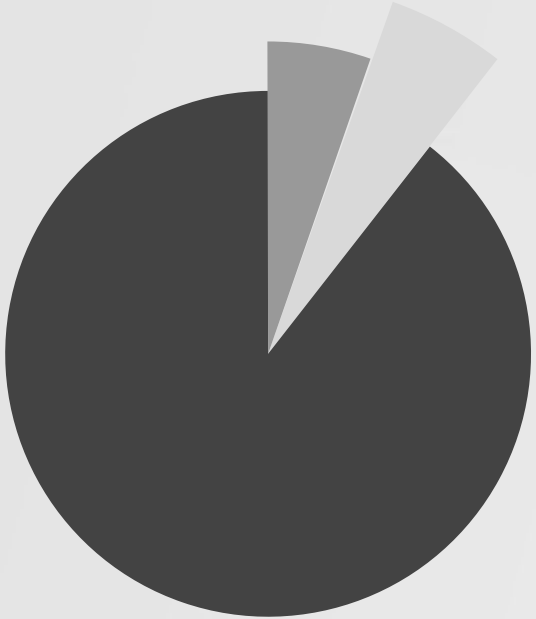
$$(\vec{f} * \vec{g})(i) = \frac{1}{|N(i)|} \sum_{l=1}^{M_{in}} \sum_{j \in N(i)} f_l(i)\ g_l(\vec{u}(i,j))$$

06

RESULTS

Accuracy and concluding notes

SPLITTING THE DATASET



Training samples: 12381

Test samples: 500

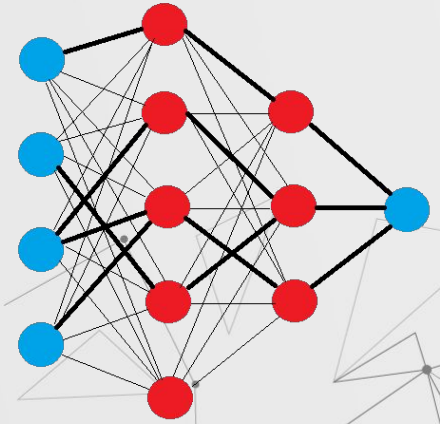
Validation samples: 500

Accuracy

On 200 epochs of training and the use of Adam optimizer

CONCLUSION

The difficulty of applying convolution in non-euclidean data, does not limit us from using deep learning on them. In our work, we used a GNN architecture to perform empirical evaluation on graph node classification. We found out that libraries like PyTorch Geometric can greatly simplify such sophisticated methods, and give the opportunity to focus on the hyperparameter selection instead of the implementation of complex operations. We hope our demonstration was insightful and will inspire future work.



THANKS

Any questions?

REFERENCES

- Shchur, Oleksandr, et al. "Pitfalls of graph neural network evaluation." arXiv preprint arXiv:1811.05868 (2018).
- Fey, Matthias, and Jan Eric Lenssen. "Fast graph representation learning with PyTorch Geometric." arXiv preprint arXiv:1903.02428 (2019).
- Fey, Matthias, et al. "SplineCNN: Fast geometric deep learning with continuous B-spline kernels." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018.
- Bronstein, Michael M., et al. "Geometric deep learning: going beyond euclidean data." IEEE Signal Processing Magazine 34.4 (2017): 18-42
- Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J. and Bronstein, M.M., 2017. Geometric deep learning on graphs and manifolds using mixture model cnns. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 5115–5124)

github.com/SimonIyamu/Geometric-Deep-Learning