Lecture 1: Introduction to Machine Learning with Geometry and Topology

Geometry and Topology in Machine Learning Seminar

June 9th, 2025

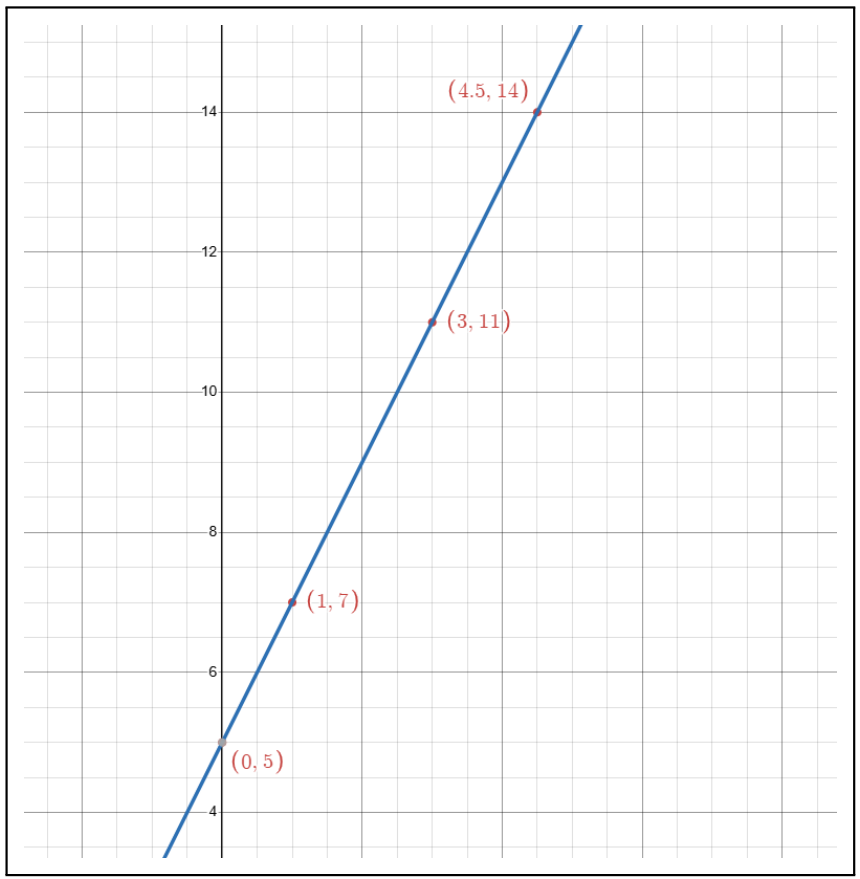
Outline

Programming in the Synthetic World

Q: Program a function whose (partial) inputs and outputs are:

Input	0	1	3	4.5
Output	5	7	11	14

A: Why not consider the function $f(x) = 2x + 5$?



Learning in the Real World is Hard

Many complications arise in the real world, including but not limited to:

1. There may not be an obvious function explicitly programmed by hand that fits the given data.
2. Even if such functions exist, they may not generalize well to additional data.
3. The given data may not have outputs, rather you are trying to learn something out of the data. The goal may not be well-defined.

Rather than trying to construct an explicit program by hand, machine learning (ML) tries to find a suitable model from a family of models.

The General Strategy of Machine Learning

The general strategy of ML tackles these problems in three steps.

1. Function Class: Define a family of functions (ie. models) $\mathcal{F}$ that tries to solve the problem given (ex. all functions $x \mapsto Ax + b$ where $A \in M_{n,n}(\mathbb{R}), b \in \mathbb{R}^n$).
   In practice, a learning problem always reduces to estimating the function f using a parametrized function class $\mathcal{F} = \{f_{\theta \in \Theta}\}$
2. Loss/Score Functions: Define a function $L : \mathcal{F} \to \mathbb{R}_{\geq 0}$[1] that measures error / loss (ex. least square errors). Alternatively, this could be a score function for how well the model does.
3. Optimization: Find the function (or functions) $f \in \mathcal{F}$ that has the least loss or most score in $\mathcal{F}$ (ex. Lagrange multipliers).

[1] The codomain can be made more general.

Example: Supervised Learning

Given: Data consisting of n labeled samples $\mathcal{D} = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n)\}$, where each $\mathbf{x_i} \in \mathbb{R}^d$ is a vector of d features and $y_i \in \mathbb{R}$ is a label.

Goal: Find/Train a model using $\mathcal{D}$ such that it can predict the label of unknown/future inputs, i.e., $\mathbb{P}(Y = y | X = x)$.

In practice, the inputs will be **a feature matrix** of n samples where each one $\mathbf{x}_i$ with d features, and a label vector $y \in \mathcal{Y} \subset \mathbb{R}^n$

$$\mathbf{X} = \begin{bmatrix} —\mathbf{x}_1— \\ —\mathbf{x}_2— \\ \cdots \\ —\mathbf{x}_n— \end{bmatrix} \text{ and } y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

Here we assume $\{(\mathbf{x_i}, y_i)\}_{i=1}^n$ drawn i.i.d. from some unknown distribution $\mathbb{P}(x, y)$.

Statistical Learning and ERM

Given known data $\mathcal{D} = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n)\}$ and a hypothesis class $\mathcal{F}$:
We can either think of ML as choosing the parameters that minimize the model's average loss on the training data

- **Empirical Risk Minimization (ERM)**: minimize average loss,

$$\theta^\star \;=\; \arg\min_{\theta \in \mathcal{F}} \; \frac{1}{n} \sum_{i=1}^{n} \ell\big(f_\theta(x_i),\, y_i\big).$$

  In this case, the loss function is

$$\theta \in \mathcal{F} \mapsto \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i),$$

  where $f_\theta(x_i)$ is the evaluation of θ on x_i and $\ell : \mathcal{D} \to \mathbb{R}_{\geq 0}$ is some function.

Probability and MLE

Or as finding the parameters that make the observed training data most probable under your model, i.e., maximize the likelihood of prediction

- **Maximum Likelihood Estimation (MLE)**: maximize data likelihood,

$$\theta^\star \;=\; \arg\max_{\theta\in\mathcal{F}} \prod_{i=1}^{n} p_\theta\big(y_i \mid x_i;\theta\big)$$

  In this case, the score function is

$$\theta \in \mathcal{F} \mapsto \prod_{i=1}^{n} p(y_i|x_i;\theta),$$

  where $\theta \mapsto p(y_i|x_i;\theta)$ is some likelihood function.

Note MLE is equivalent to ERM when $\ell = -n\log p$. They together provide two lens of almost all tasks in ML.

Our Focus in Part A

In **Part A: Groups and Representations**, our primary focus is on designing the family of functions $\mathcal{F}$.

In this lecture, we will introduce two examples of neural networks, which falls in a sub-branch of ML known as deep learning:

1. Convolutional Neural Networks
2. Graph Neural Networks

These two architectures will give us examples for why geometry/topology comes into machine learning.

Outline

What is a Neural Network?

Definition

A (feed forward) neural network is a composition of functions $L_N \circ L_{N-1} \circ ... \circ L_1$ where

1. $L_i : \mathcal{F}_i \to \mathcal{F}_{i+1}$ is called the i-th layer of the neural network.
2. $\mathcal{F}_i$ is called the i-th feature space.

1. Typically the $\mathcal{F}_i$'s are some real vector spaces
2. L_0, the feature space where the domain lives, is sometimes called the input layer. There is no actual function here (or you can think it as an identity map $L_0(x) = x$). L_N is sometimes called the output layer, and $L_1, ..., L_{N-1}$ are sometimes called hidden layers.

Example: Perceptrons (Supervised Algorithm)

Definition

A perceptron is a single layer neural network $L : \mathbb{R}^n \to \mathbb{R}^m$ such that, for each $\mathbf{x} = (x_1, ..., x_n) \in \mathbb{R}^n$

1. Given a new data point $\mathbf{x}_{new}$, the $i-$th component $L_i(\mathbf{x}_{new}) = \sigma\left(w_i^\top \mathbf{x}_{new} + b_i\right)$ computes the activation (or score) for class i.
2. where $w_i \in \mathbb{R}^n$ is some vector (called weight) and $b_i \in \mathbb{R}$ is some real number (called bias).
3. and σ is some non-linear function (called an activation function).

Typically, $\hat{y} = \arg\max_{1 \leq i \leq m} L_i(\mathbf{x}_{new})$ is the predicted label, and we optimize the choice of w_i, b_i's to find the best model.

Perceptrons in Binary Classification

An example task of perceptron can do is binary classification with linear predictor.

In such case, a perceptron is just the indicator of which side of a hyperplane you're on, i.e,

$$x \mapsto \chi(w^\top x + b > 0).$$

Example: Multilayer Perceptrons

Definition

A multilayer perceptron (MLP) is a neural network $L_N \circ ... \circ L_1$ such that $N > 1$ and each L_i is a perceptron.

Note that the weight matrices and bias vectors at each layer need not be the same.

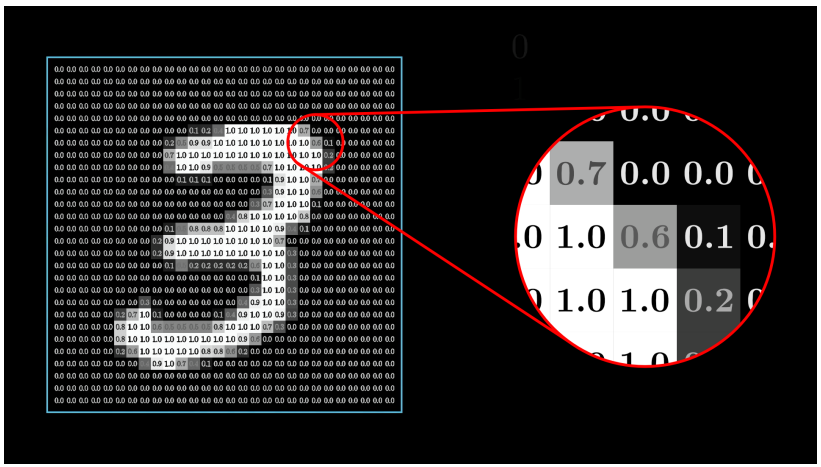
Here are some examples of common activation functions/nonlinearties used:

1. Sign Function $\sigma(x) = \begin{cases} 1, x > 0 \\ 0, x = 0 \\ -1, x < 0 \end{cases}$.
2. Hyperbolic tangent functions
3. Rectified Linear Units (ReLU): $\sigma(x) = \frac{x+|x|}{2}$.

Note the importance for σ to be non-linear. Otherwise, an MLP with linear σ's is equivalent to a perceptron with linear σ.

MLPs in Image Processing

Consider the following 28×28 pixel image[2] P:



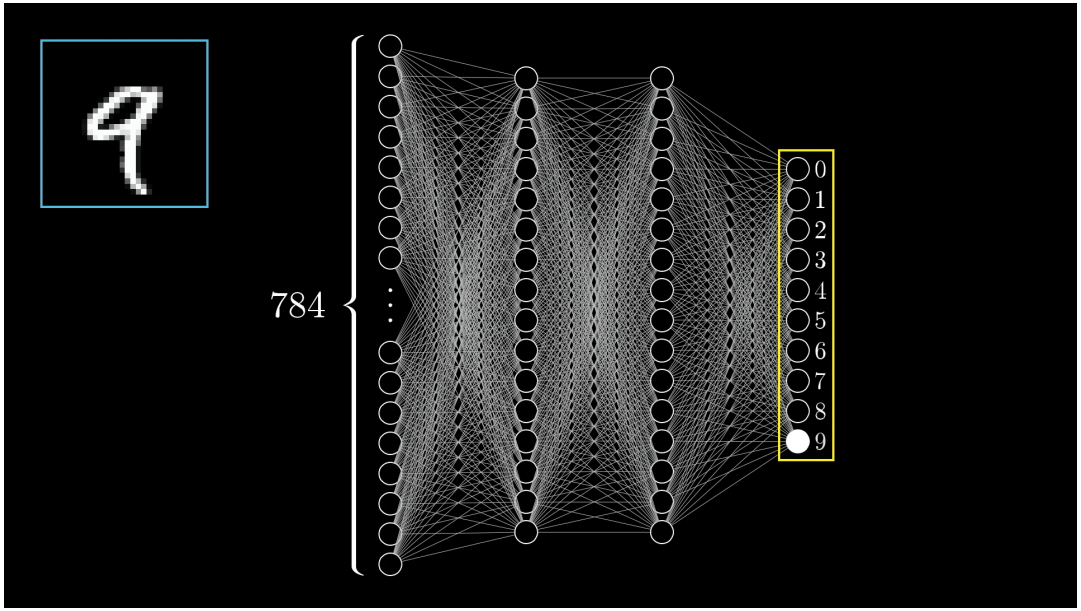
Question: Can you classify which number $N(P)$ that P is?
One idea is to solve this using an MLP:

1. The input layer is $\mathbb{R}^{28\times 28}$, where each pixel has a brightness value between 0.0 (black) and 1.0 (white).
2. The output is $(p(N(P) = 1|P), ..., p(N(P) = 9|P)) \in \mathbb{R}^9$.
3. This is now a maximum likelihood estimation problem.

[2] Picture from the 3Blue1Brown YouTube channel, and MNIST dataset.

Basic Unit

Below is a fully connected neural network.



Picture from 3Blue1Brown.

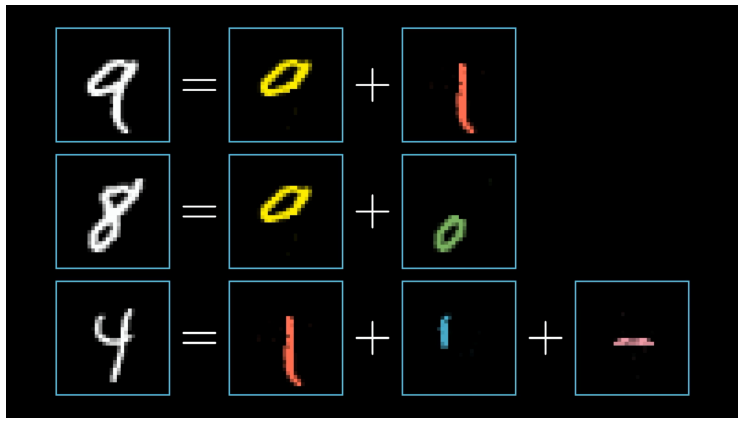
Each neuron has a basic unit $\sigma(w^\top x + b)$.

If you're seeing this illustration for the first time, there are two questions you may have:

1. What do hidden layers represent?
2. How are these layers connected?

Hidden Layers

Heuristically, the hidden neurons are used to capture broken into smaller, recognizable subcomponents (features) as following



Picture from 3Blue1Brown.

But as a hard rule, in a fully-connected net[3] those features are distributed across many neurons and are not tied to specific pixel locations, so they cannot capture the desired spatial patterns.

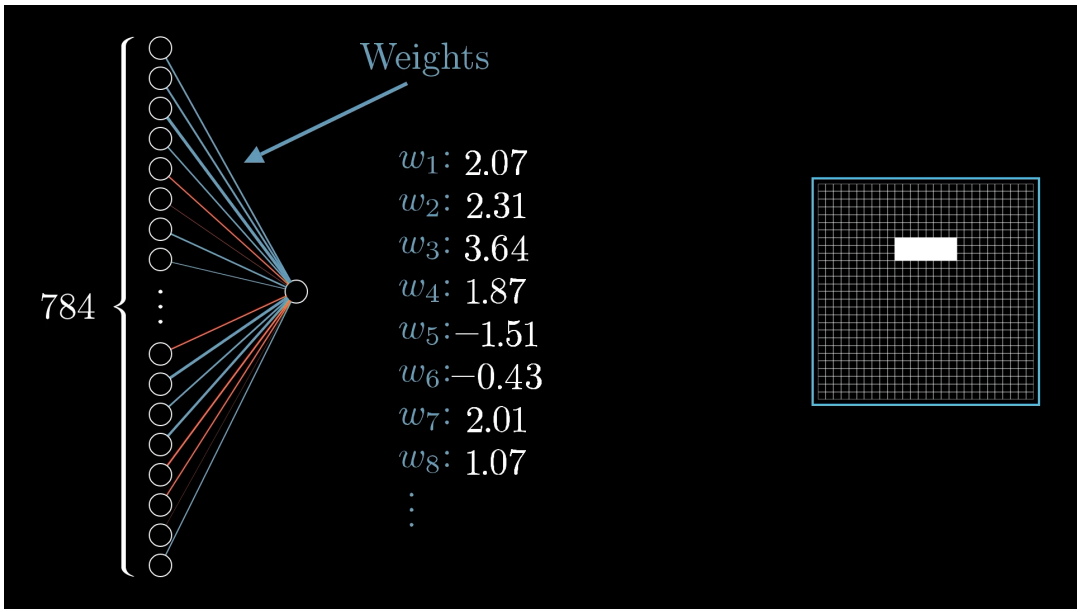
CNNs, as we will cover later, work more like this pattern.

[3] ex. MLPs

Weighted–Sum Mechanism

We want to construct such features using all these neurons.

The key, still heuristically, is just a simple word **learnable weighted sum**. Each neuron, or we say each pixel in the first layer, contribute a bit to the output feature of the next layer.



Picture from 3Blue1Brown.

Universal Approximation

The choice of perceptrons may seem arbitrary, but they turn out to approximate functions quite effectively! This is known as universal approximation.

We state this informally as follows. Let $K \subset \mathbb{R}^n$ be a nice compact subset[4] and $f : K \to \mathbb{R}$ be continuous

1. The function f can be arbitrary approximated by a perceptron (but possibly with high width).
2. Even when f is not continuous, in many reasonable cases[5], there is a way to approximate f using ReLU-based MLPs with minimal width.

[4] Think of a shape

[5] For analysts, Bochner–Lebesgue p-integrable functions

Drawbacks of MLPs

Image classification with MLPs have several limitations:

1. If the image has more pixels, the computation costs could go up to be really high.
2. There is no spatial structure on the image since it is flattened as a vector (nuisances in the image reorder the inputs arbitrarily). In practice, MLPs do not perform well under translations of images.

In practice, however, many problems involving pictures from data have a natural notion of translational invariance or translational equivariance[6] encoded in them:

1. What number is the digit in the picture? (Invariance)
2. Where is the number in the picture? (Equivariance)

[6] The general definition of invariance and equivariance will be given next lecture.

Outline

Invariance and Equivariance in the Architecture

Observation: Rather than trying to find a model that is relatively well-behaved with translations, why don't we just restrict our family $\mathcal{F}$ to neural networks that are translational invariant or equivariant?

Slogan: "Translation equivariant linear maps are convolutions."

Convolution

Recall for two functions $f, g : \mathbb{R} \to \mathbb{R}$, their convolution is defined as

$$f * g(t) = \int_{\mathbb{R}} f(\tau) g(t - \tau) d\tau.$$

In the discrete world, for two signal $x, w : \mathbb{Z} \to \mathbb{R}$, their convolution is

$$(x * w)_k = \sum_{\tau=-\infty}^{\infty} x_\tau w_{k-\tau}, k \in \mathbb{Z}.$$

For two matrices[7] $x, w : \mathbb{Z}^2 \to \mathbb{R}$, their convolution is:

$$(x * w)_{i,j} = \sum_{m \in \mathbb{Z}} \sum_{n \in \mathbb{Z}} x(m, n) w(i - m, j - n).$$

[7] You can set x, w to be 0 outside of its grid.

Convolutions are Translation Equvariant!

Lemma:

Convolutions are translation equvariant.

Proof: Suppose $(a, b) \in \mathbb{Z}^2$, then we observe that

$$\begin{aligned}(x * w)_{i-a,j-b} &= \sum_{m\in\mathbb{Z}}\sum_{n\in\mathbb{Z}} x(m,n)w((i-a)-m,(j-b)-n)) \\ &= \sum_{m'=m+a\in\mathbb{Z}}\sum_{n'=n+b\in\mathbb{Z}} x(m'-a,n'-b)w(i-m',j-n') \\ &= (x_{\bullet-a,\bullet-b} * w)_{i,j}.\end{aligned}$$

Convolutional Neural Network

While there is no set definition for convolutional neural network (CNNs), a general paradigm is that it is a neural network that involves a convolutional layer.

View a stack of feature maps (ex. images) as a map $f : \mathbb{Z}^2 \to \mathbb{R}^{K_\ell}$.

Definition

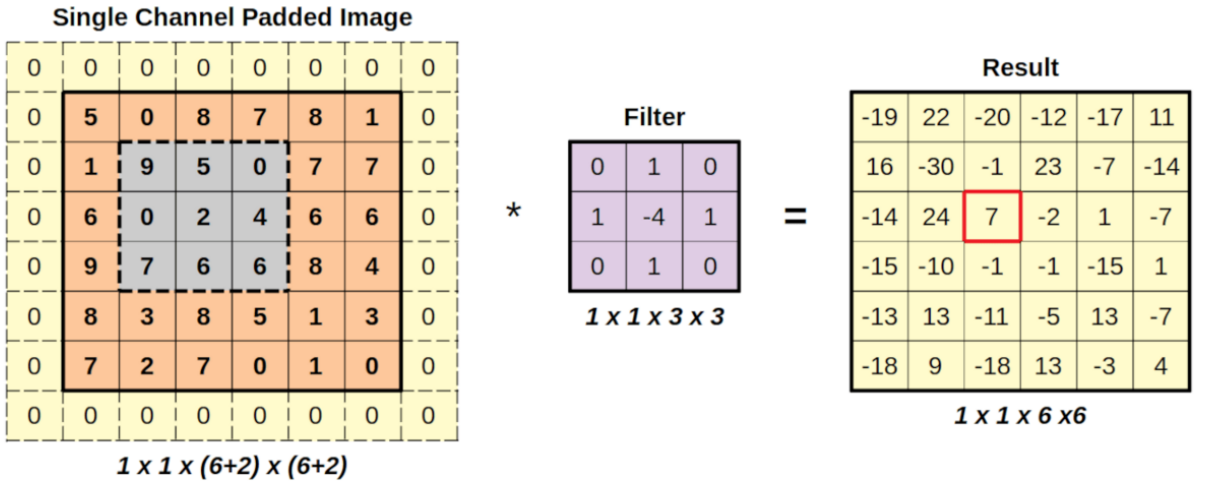
A convolutional layer is a map that takes a stack of feature maps $f : \mathbb{Z}^2 \to (\mathbb{R})^{K_\ell}$:

$$f \mapsto \{(x,y) \to \{ \sum_{(m,n)\in\mathbb{Z}^2} \sum_{i=1}^{K_\ell} f_i(m,n)\psi_i^k(x-m, y-n)\}_{k=1,\dots,K_{\ell+1}}\}$$

where $\psi_i^k : \mathbb{Z}^2 \to \mathbb{R}$ are a collection of matrices known as filters.

Here the filters are not fixed and is learned / optimized by the network.

Example of a Channel in a Convolutional Layer



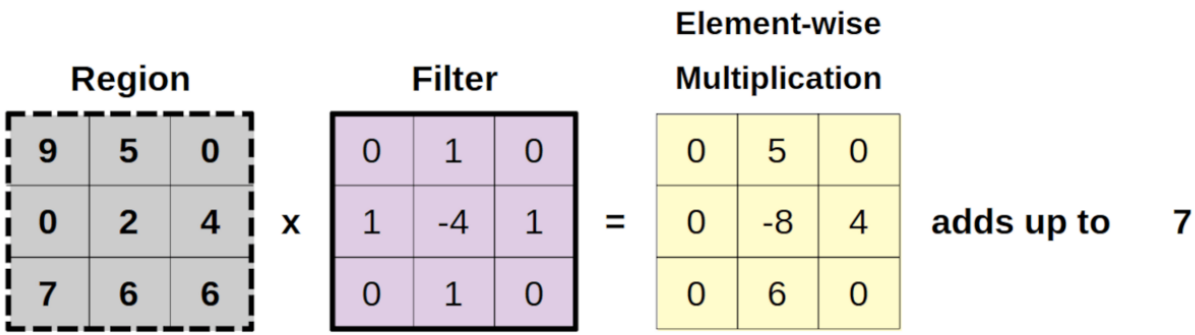


Figure: Visualization by Daniel Godoy.

Convolution Neural Networks in Practice

Although we will not get into details:

1. In practice, there is a stride length that controls "how far the filters are moving".
2. There is another layer called pooling that ensures some translational invariance in the model. The pooling is down typically runs through $k \times k$ sub-images of the current image and takes the max or average of pixel values.
3. A typical CNN could go something in the order of:

Convolution, Pooling, Convolution, Pooling, ...,

Pooling, Fully Connected Layer, Readout.

Generalizations

More generally, one can view the convolutional layer of CNN abstract as an integral transform:

$$I_\kappa : L^2(\mathbb{R}^2, \mathbb{R}^{c_{in}}) \to L^2(\mathbb{R}^2, \mathbb{R}^{c_{out}})$$

where

- $L^2(\mathbb{R}^2, \mathbb{R}^n)$ is the collection of functions $f : \mathbb{R}^2 \to \mathbb{R}^n$ such that
$$\int_{\mathbb{R}^n} ||f||^2 dx < \infty.$$
- $\kappa : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^{c_{out} \times c_{in}}$ is called the kernel functions and
$$I_\kappa[f](x) := \int_{\mathbb{R}^2} \kappa(x, y) f(y) dy$$
Here $\kappa(x, y)f(y)$ is thought of as a matrix multiplication.

Fun Remark for Analysis Enjoyers

This is just a fun remark for analysis enjoyers, it is totally okay if the following theorem goes over your head.

Theorem [Dunford-Pettis, see Theorem 1 of [Duits, 2005]]:

Let $\mathcal{K} : L^2(X) \to L^\infty(X)$ be linear, bounded operator, then there exists a kernel $k \in L^1(X, X)$ such that

$$(Kf)(y) = \int_X k(y, x) f(x) d\mu_X(x)$$

with $f \in L_2(X)$ and $d\mu_X$ is a Radon measure on X.

Spherical CNNs

To give a glimpse at some of the interactions between CNNs and geometry.

In [Cohen et al., 2018] considered spherical images (on S^2) and, corresponding, spherical CNNs that can process spherical images.

- The primary idea is to replace the translation group with the rotational group $\mathrm{SO}(3)$.
- Inner product of spherical signals should be:

$$\langle \psi, f \rangle := \int_{S^2} \sum_{k=1}^{K} \psi_k(x) f_k(x) d\mu_{S^2}$$

where $d\mu_{S^2}$ is the surface measure on S^2.

Outline

From CNNs to GNNs

1. Typically CNNs process images, which can be viewed as a matrix with pixel values at each coordinate.
2. On another hand, we can reinterpret an $n \times m$ matrix as an $n \times m$ grid graph[8] with features (ie. pixel values) on each vertex.
3. This leads to a more general questions - can we design neural networks to handle graphs?
4. This is the study of graph neural networks (GNNs).

[8] Or grid graph with diagonals

Motivation for Graph Neural Networks

At a high level, a graph neural networks (GNN) is a model that takes in data represented as graphs and perform tasks on graph data.

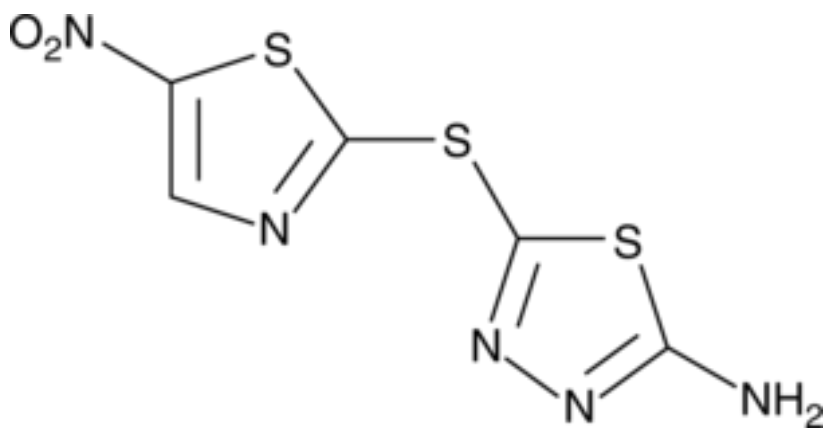
There are many examples of data in the world that can be represented as graphs.

- Molecules can be represented as multi-graphs. The multiplicity of edges between two nodes is the bond type.
- Recommendation data can be represented as bipartite graphs between the users and the items.
- Financial transactions can be represented as directed graphs where the users are nodes and transactions are edges.

Graph Neural Networks in Industry

GNNs have found a number of significant industrial applications. On molecular data,

- GNNs can be trained to predict the potency of a molecule. This led to the (re)discovery of Halicin in [Stokes et al., 2020].



- GNNs showed that Halicin has strong antibiotic properties.
- In subsequent laboratory tests, the drug successfully treated *Acinetobacter baumannii*, which is a bacterium resistant to all previously known anti-biotics.

Graph Neural Networks in Industry

On recommendation data,

- Alibaba [Zhu et al., 2019] developed a GNN model (called AliGraph), which is currently used to support product recommendation and personalized search at Alibaba's E-Commerce platform
- Uber Eats uses the GraphSAGE model [Hamilton et al., 2017] to provide personalized restaurant recommendations to users.

On financial transactions,

- Amazon researchers have built a real-time fraud detection system with graph neural networks and the Amazon Web Services (AWS) framework.

GNNs Inputs

Let us look at GNNs. Typically the input of a GNN is the following.

Definition (Graph)

A graph (with features) is the data $G = (V, E, c, X)$ where V is a finite vertex set, $E \subseteq V \times V$, X is a space of features, and $c : V \to X$ is a function that assigns each vertex a value in X.

Let $v \in G$, its neighboring vertices are denoted $N_G(v) = \{u \in V : (u, v) \in E\}$.

Similar to how CNNs look at adjacent pixels in its model, we want to look at features on adjacent vertices in designing GNNs.

Message-Passing GNNs

The Message-Passing GNN is typically given as follows,

Message Passing GNNs

1. Initialize $h_v^{(0)} = c(v)$ for all $v \in V$.
2. At layer $t+1$, we define the GNN embedding recursively as:
   - $m_v^{(t+1)} = \sum_{\omega \in N_G(v)} \mathrm{AGG}^{(t)}(h_v^{(t)}, h_w^{(t)})$.
   - $h_v^{(t+1)} = \mathrm{Update}^{(t)}(h_v^t, m_v^{t+1})$.

   Here $\mathrm{AGG}^{(t)}$ and $\mathrm{Update}^{(t)}$ are arbitrary non-linear mappings.
3. After reaching the final layer T, the output is given out using some readout function $\hat{y} = R(\{h_v^{(T)} \mid v \in V\})$.

Typically the readout function will be used to perform node, edge, or graph classifications.

Example: Graph Convolutional Networks (GCNs)

By choosing

$$\mathrm{AGG}^{(t)}(h_v, h_w) = \frac{1}{\sqrt{\deg(v)\,\deg(w)}}\, W^{(t)}\, h_w$$

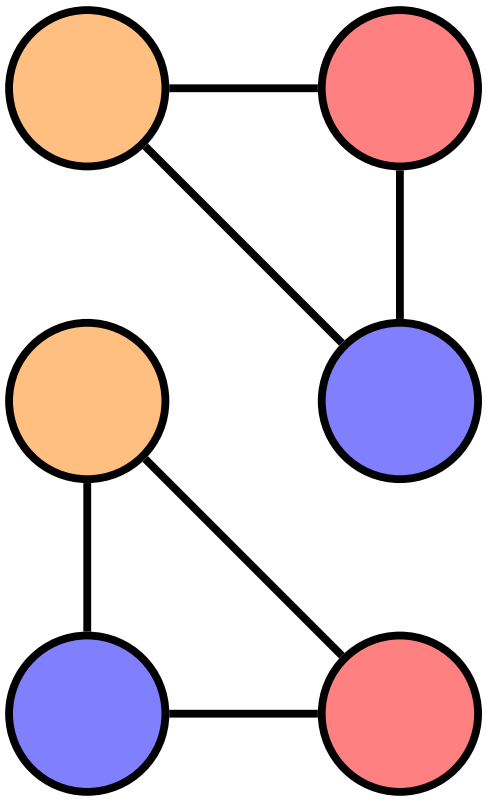
$$\mathrm{Update}^{(t)}(h_v, m_v) = \sigma\big(m_v\big).$$

the general message-passing GNN reduces to special case known as the Graph Convolutional Network (GCN).

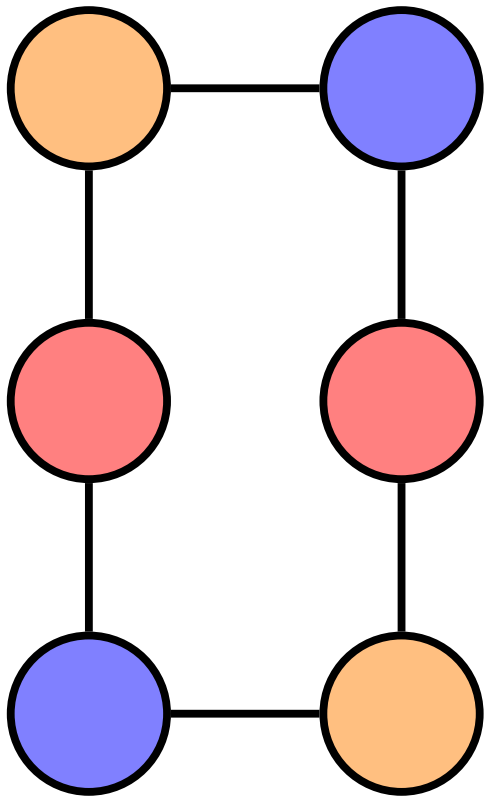
Note that the update function on v does not care about what is happening on the current node.

Why Do We Need Topology in GNNs?

Let us take a look at the following 2 graphs.



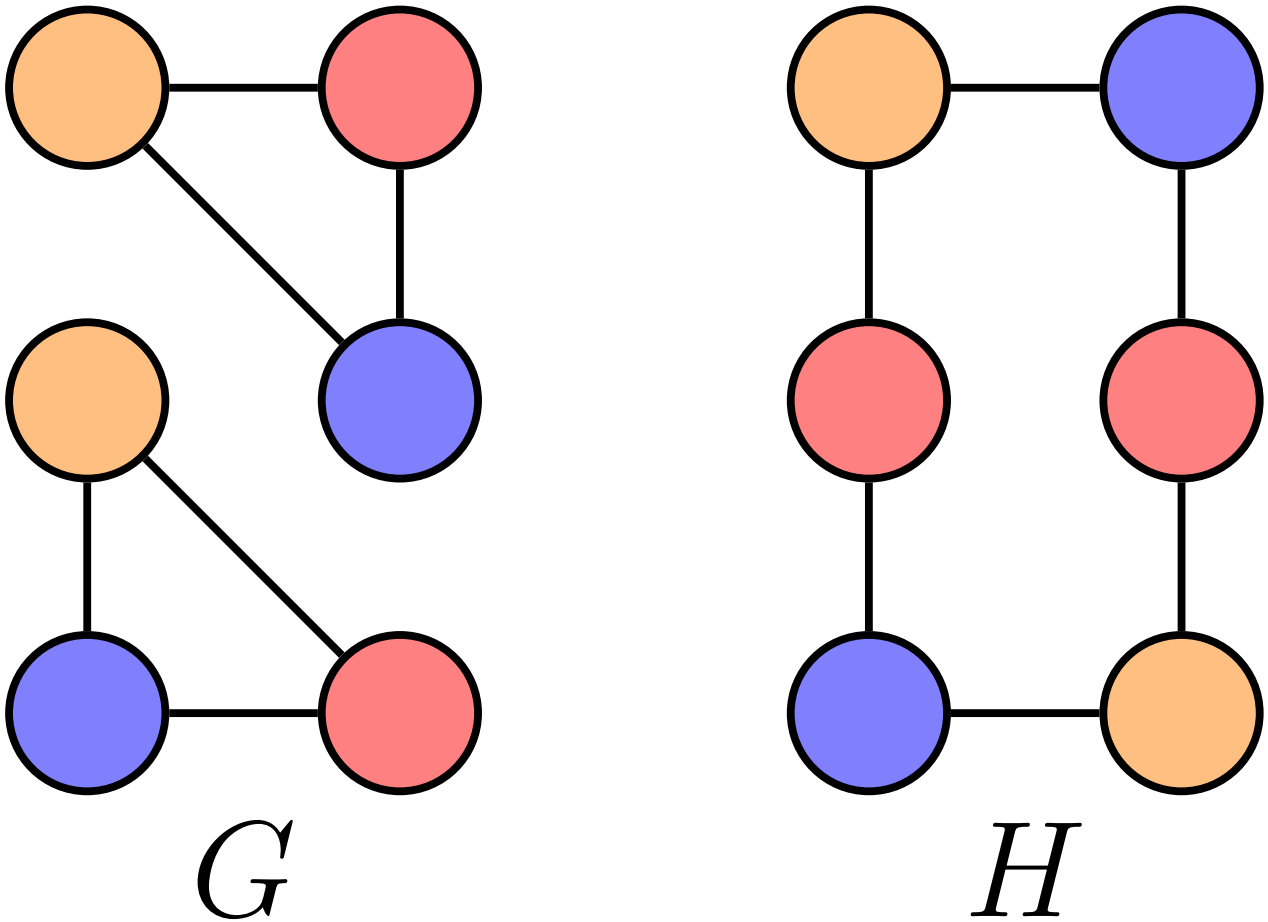
G



H

Claim: Our GNN model, at the present, produces the same output on G and H.

Why Do We Need Topology in GNNs?



Recall the aggregate function is given by:

$$\mathrm{AGG}^{(t)}(\{h_w^{(t)} \mid w \in N_G(v)\}).$$

There is no difference in the input to the aggregate function!

Topology in GNNs

Thus, we see that message passing GNNs cannot count the number of connected components or the number of (independent) cycles on a graph.

Although this will not be the focus of this seminar, if the reader knows some algebraic topology, these two data corresponds exactly to:

1. The zeroth homology $H_0(G)$.
2. The first homology $H_1(G)$.

We can add homology as an additional parameter in the model to increase expressivity!

Why Do We Need Geometry in GNNs?

Molecules can be viewed as a graph, and molecules have a natural embedding in $\mathbb{R}^3$:

1. This means that one feature each vertex v of a molecular graph G can have is a coordinate $x(v) \in \mathbb{R}^3$.
2. We want GNNs to be equivariant/invariant on these molecules with respect to translations, rotations, etc. in $\mathbb{R}^3$.

More generally, the feature space of a graph could be $\mathbb{R}^n$ or any arbitrary vector space V with the action of some group G.[9]

More on this next lecture!

[9] Also known as a representation.

References I


Cohen, T. S., Geiger, M., Köhler, J., and Welling, M. (2018).
Spherical CNNs.
In *International Conference on Learning Representations*.


Duits, R. (2005).
Perceptual organization in image analysis : a mathematical approach based on scale, orientation and curvature.
Phd thesis 1 (research tu/e / graduation tu/e), Biomedical Engineering.


Hamilton, W. L., Ying, R., and Leskovec, J. (2017).
Inductive representation learning on large graphs.
In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1025–1035, Red Hook, NY, USA. Curran Associates Inc.

References II

Stokes, J. M., Yang, K., Swanson, K., Jin, W., Cubillos-Ruiz, A., Donghia, N. M., MacNair, C. R., French, S., Carfrae, L. A., Bloom-Ackermann, Z., Tran, V. M., Chiappino-Pepe, A., Badran, A. H., Andrews, I. W., Chory, E. J., Church, G. M., Brown, E. D., Jaakkola, T. S., Barzilay, R., and Collins, J. J. (2020).
A deep learning approach to antibiotic discovery.
Cell, 180(4):688–702.e13.

Zhu, R., Zhao, K., Yang, H., Lin, W., Zhou, C., Ai, B., Li, Y., and Zhou, J. (2019).
Aligraph: a comprehensive graph neural network platform.
Proc. VLDB Endow., 12(12):2094–2105.