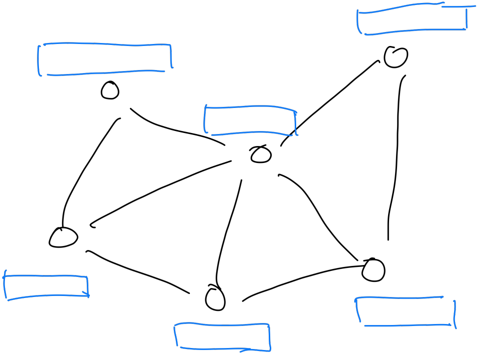
Graph Learning
SD212
7. Graph Neural Networks

Thomas Bonald, Simon Delarue

2023 – 2024

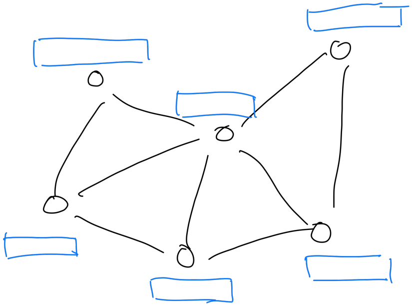
Motivation

Machine learning on enriched graphs, with node **features**



Outline

1. Background on neural networks
2. Graph neural networks
3. Variants



Supervised learning

Objective: Predict the **label** (classification) or the **value** (regression) of a sample by training.

Formally, learn some mapping $f : x \mapsto y$ minimizing:

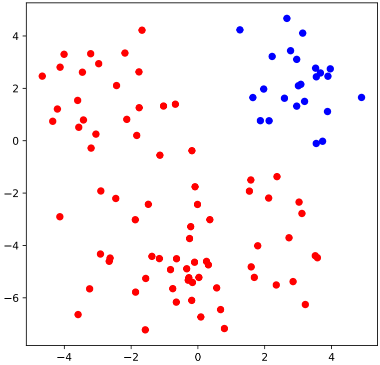
$$\frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(x_i))$$

where

- $x \in \mathbb{R}^d$
- $y \in \{0, 1\}, \{1, \ldots, K\}$ or $\mathbb{R}$
- $(x_1, y_1), \ldots, (x_n, y_n)$ are the training examples
- ℓ is the loss function

Example

$$x \in \mathbb{R}^2,\ y \in \{0, 1\},\ n = 100$$



Binary classification

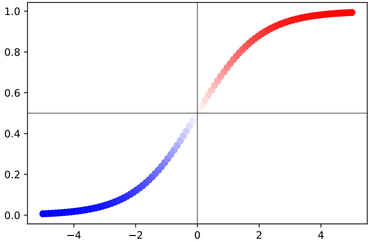
$$x \in \mathbb{R}^d,\ y \in \{0, 1\}$$

Logistic regression

Probability that $y = 1$ for sample x:

$$p = \sigma(w^T x) \in [0, 1]$$

where $w \in \mathbb{R}^d$ is the weight vector (to be learned).

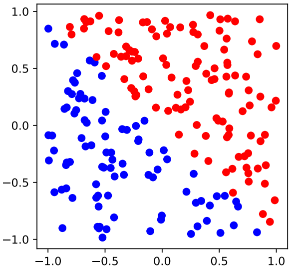


Logistic function $\sigma(u) = \frac{1}{1+e^{-u}}$

Example

$$x \in \mathbb{R}^2,\ y \in \{0, 1\}$$

Training data



Model for $w = (1, 1)$

Bias term

$$x \in \mathbb{R}^d,\ y \in \{0, 1\}$$

Logistic regression

Probability that $y = 1$ for sample x:

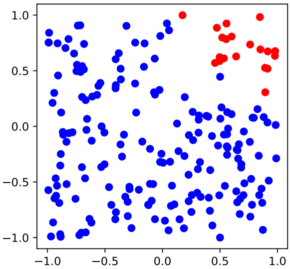
$$p = \sigma(w^T x + b)$$

where $w \in \mathbb{R}^d$ is the **weight** vector and $b \in \mathbb{R}$ the **bias** term (to be learned).

Example

$$x \in \mathbb{R}^2,\ y \in \{0,1\}$$

Training data



Model for $w = (1,1)$, $b = -1$

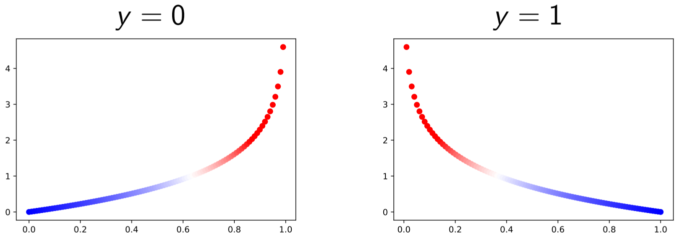
Loss function

$$y \in \{0, 1\},\ p \in [0, 1]$$

Binary cross-entropy

For one sample:

$$-y \log p - (1 - y) \log(1 - p)$$



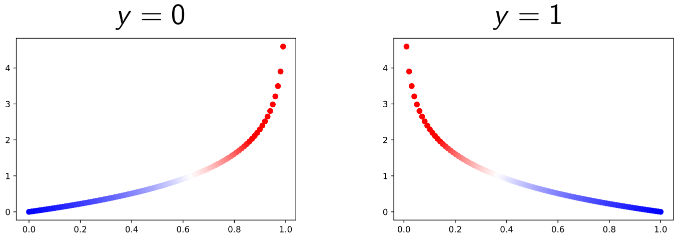
Loss function with respect to $p \in [0, 1]$

Loss function

Binary cross-entropy

For n samples:

$$-\sum_{i=1}^{n}(y_i \log p_i + (1 - y_i)\log(1 - p_i))$$



Loss function with respect to $p \in [0, 1]$

Problem to solve

$$x_1, \ldots, x_n \in \mathbb{R}^d,\ y_1, \ldots, y_n \in \{0, 1\}$$

Objective

Find w and b minimizing:

$$L = -\sum_{i=1}^{n} (y_i \log p_i + (1 - y_i) \log(1 - p_i))$$

with

$$p_1 = \sigma(w^T x_1 + b), \ldots, p_n = \sigma(w^T x_n + b)$$

Regularization

$$x_1, \ldots, x_n \in \mathbb{R}^d,\ y_1, \ldots, y_n \in \{0, 1\}$$

Objective

Find w and b minimizing:

$$L = -\sum_{i=1}^{n} (y_i \log p_i + (1 - y_i) \log(1 - p_i)) + \frac{\lambda}{2}(||w||^2 + b^2)$$

with

$$p_1 = \sigma(w^T x_1 + b), \ldots, p_n = \sigma(w^T x_n + b)$$

where λ is some hyper-parameter.

Gradient descent

Optimization problem:

$$\arg\min_{w,b} L$$

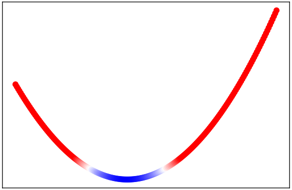
Algorithm

Iterate over:

$$w \leftarrow w - \alpha \frac{\partial L}{\partial w}, \quad b \leftarrow b - \alpha \frac{\partial L}{\partial b}$$

where α is the **learning rate**

Loss L



Gradient expression

For one sample

$$L = -y \log p - (1-y)\log(1-p)$$

$$p = \sigma(w^T x + b)$$

Proposition

$$\frac{\partial L}{\partial w} = (p-y)x$$

$$\frac{\partial L}{\partial b} = p - y$$

Gradient expression

For n samples with regularization:

$$L = -\sum_{i=1}^{n}(y_i \log p_i + (1-y_i)\log(1-p_i)) + \frac{\lambda}{2}(||w||^2 + b^2)$$

where

$$p_1 = \sigma(w^T x_1 + b), \ldots, p_n = \sigma(w^T x_n + b)$$

Proposition

$$\frac{\partial L}{\partial w} = \lambda w + \sum_{i=1}^{n}(p_i - y_i)x_i$$

$$\frac{\partial L}{\partial b} = \lambda b + \sum_{i=1}^{n}(p_i - y_i)$$

Multi-class extension

$$x \in \mathbb{R}^d,\ y \in \{1, \ldots, K\}$$

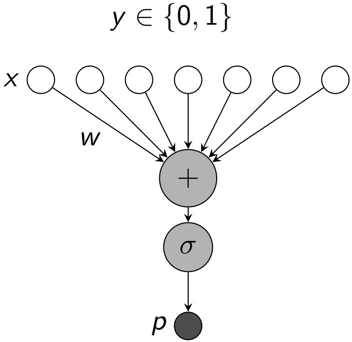
Softmax regression

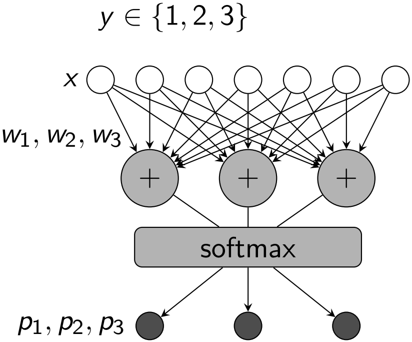
For $k = 1, \ldots, K$, probability that $y = k$ for sample x:

$$p(k) = \frac{e^{w_k^T x + b_k}}{e^{w_1^T x + b_1} + \ldots + e^{w_K^T x + b_K}}$$

where $w_1, \ldots, w_K$ are the **weight** vectors and $b_1, \ldots, b_K$ the **bias** terms (to be learned).

Logistic vs. softmax regression





Loss function

$$y \in \{1, \ldots, K\},\ p \in [0,1]^K$$

Cross entropy

For one sample:

$$-\sum_{k=1}^{K} 1_{\{y=k\}} \log p(k)$$

with

$$p(k) \propto e^{w_k^T x + b_k}$$

Problem to solve

$$x_1, \ldots, x_n \in \mathbb{R}^d,\ y_1, \ldots, y_n \in \{1, \ldots, K\}$$

Objective

Find $w_1, \ldots, w_K$ and $b_1, \ldots, b_K$ minimizing:

$$L = -\sum_{i=1}^{n} \sum_{k=1}^{K} 1_{\{y=k\}} \log p_i(k) + \frac{\lambda}{2} \sum_{k=1}^{K} (||w_k||^2 + b_k^2)$$

with

$$p_i(k) \propto e^{w_k^T x_i + b_k}$$

where λ is some hyper-parameter.

Gradient expression

For one sample:

$$L = -\sum_{k=1}^{K} 1_{\{y=k\}} \log p(k)$$

$$p(k) \propto e^{w_k^T x + b_k}$$

Proposition

$$\frac{\partial L}{\partial w_k} = (p(k) - 1_{\{y=k\}})x$$

$$\frac{\partial L}{\partial b_k} = p(k) - 1_{\{y=k\}}$$

Gradient expression

For n samples with regularization:

$$L = -\sum_{i=1}^{n}\sum_{k=1}^{K} 1_{\{y_i=k\}} \log p_i(k) + \frac{\lambda}{2}\sum_{k=1}^{K}(||w_k||^2 + b_k^2)$$

Proposition

$$\frac{\partial L}{\partial w_k} = \lambda w_k + \sum_{i=1}^{n}(p_i(k) - 1_{\{y_i=k\}})x_i$$

$$\frac{\partial L}{\partial b_k} = \lambda b_k + \sum_{i=1}^{n}(p_i(k) - 1_{\{y_i=k\}})$$

Neural network

A neural network is a **composition** of functions of the form:

$$x \mapsto \sigma(Wx + b)$$

with

- W weight matrix (to be learned)
- b bias vector (to be learned)
- σ activation function (typically non-linear)

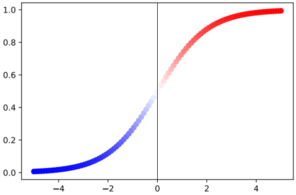
Each such function is a **layer** of the network.

The **output** of the neural network is a **probability distribution** (for classification) or a **value** (for regression).

Activation functions

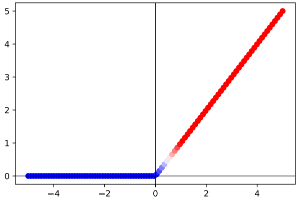
Logistic function

$$u \mapsto \frac{1}{1 + e^{-u}}$$

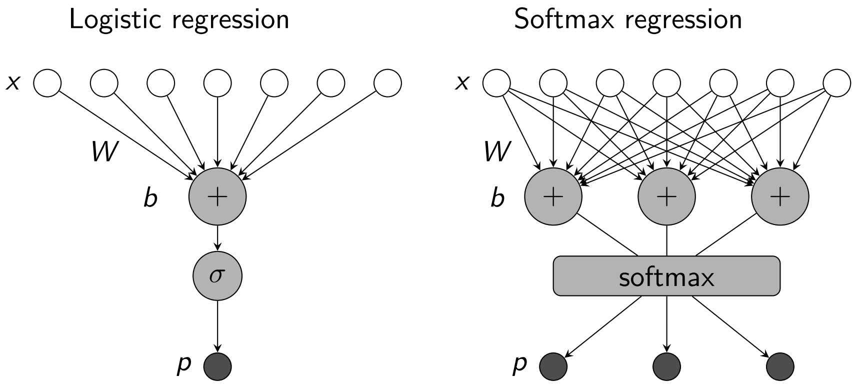


ReLU function

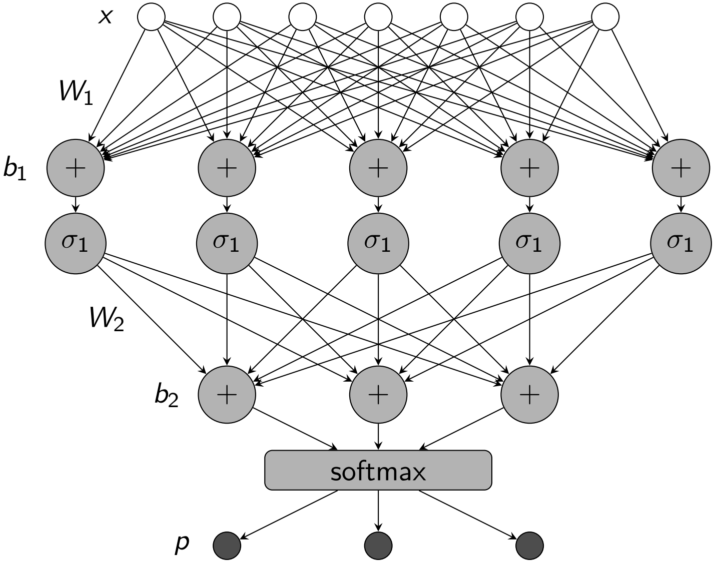
$$u \mapsto \max(u, 0)$$



Single-layer networks



A neural network with 2 layers



Problem to solve

Consider a neural network with N layers

Objective

Find **weight matrices** $W_1, \ldots, W_N$ and **bias vectors** $b_1, \ldots, b_N$ minimizing:

$$L = -\sum_{i=1}^{n} \sum_{k=1}^{K} 1_{\{y=k\}} \log p_i(k) + \frac{\lambda}{2} \sum_{l=1}^{N} (||W_l||^2 + ||b_l||^2)$$

with

$$p_i = f_N \circ \ldots \circ f_1(x_i)$$

and

$$f_l(x) = \sigma_l(W_l x + b_l)$$

Parameters to learn

$$x \in \mathbb{R}^d,\ y \in \{1, \ldots, K\}$$

$$\boxed{x \mapsto Wx + b}$$

Layer	Weights W	Biases b
1	$d_1 \times d$	d_1
2	$d_2 \times d_1$	d_2
$\vdots$	$\vdots$	$\vdots$
N	$K \times d_{n-1}$	K

Gradient: Single-layer neural network

$$x \mapsto u = Wx + b \mapsto p = \underbrace{\text{softmax}}_{\sigma}(u) \mapsto L = -\sum_k y_k \log p(k)$$

Backpropagation

Given the loss L for some sample x, compute:

1. $\dfrac{\partial L}{\partial p}$
2. $\dfrac{\partial L}{\partial u} = \dfrac{\partial L}{\partial p}\dfrac{\partial p}{\partial u}$
3. $\dfrac{\partial L}{\partial W} = \dfrac{\partial L}{\partial u}\dfrac{\partial u}{\partial W}$

   $\dfrac{\partial L}{\partial b} = \dfrac{\partial L}{\partial u}\dfrac{\partial u}{\partial b}$

Gradient: A 2-layer neural network

$$x \mapsto u_1 = W_1 x + b_1 \mapsto v_1 = \sigma_1(u_1)$$

$$v_1 \mapsto u_2 = W_2 v_1 + b_2 \mapsto p = \underbrace{\mathrm{softmax}}_{\sigma_2}(u_2) \mapsto L = -\sum_k y_k \log p(k)$$

Backpropagation

Given the loss L for some sample x, compute:

- Layer 2:

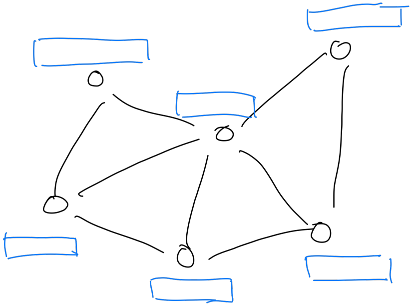
$$\frac{\partial L}{\partial p} \;\rightarrow\; \frac{\partial L}{\partial u_2} = \frac{\partial L}{\partial p}\frac{\partial p}{\partial u_2} \;\rightarrow\; \frac{\partial L}{\partial W_2}, \frac{\partial L}{\partial b_2}$$

- Layer 1:

$$\frac{\partial L}{\partial v_1} = \frac{\partial L}{\partial u_2}\frac{\partial u_2}{\partial v_1} \rightarrow \frac{\partial L}{\partial u_1} = \frac{\partial L}{\partial v_1}\frac{\partial v_1}{\partial u_1} \rightarrow \frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial b_1}$$

Outline

1. Background on neural networks
2. **Graph neural networks**
3. Variants

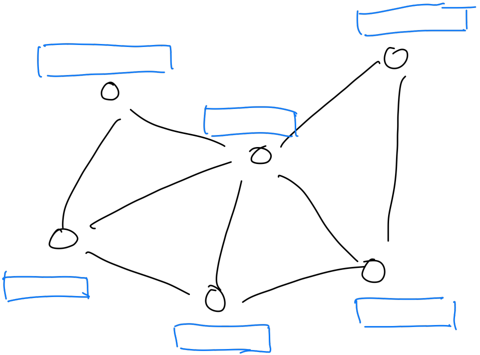


Graph neural networks

Principle

Learn **node embeddings** using both:

- the **node features** $\rightarrow$ neural net
- the **graph** $\rightarrow$ message passing (cf. diffusion)



Graph neural network

A graph neural network is a **composition** of a **diffusion step**:

$$X \mapsto U = PX$$

where $P = D^{-1}A$ is the **transition matrix** of the random walk, and X the matrix of features (dimension $n \times d$) followed by a **linear transformation**:

$$U \mapsto U' = UW^T + 1b^T$$

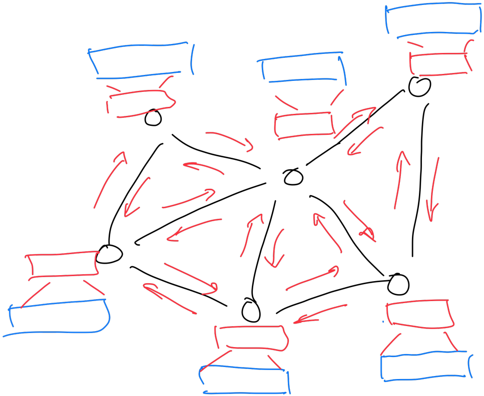
followed by the **activation function**:

$$U' \mapsto V = \sigma(U')$$

Each such function is a **layer** of the network.

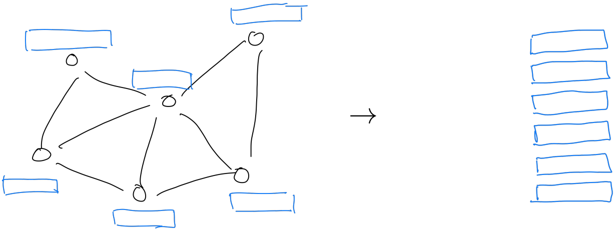
The **output** of the graph neural network is a **probability distribution** (for classification) or a **value** (for regression).

Message passing



Vectorial representation

Let X be the matrix of features (dimension $n \times d$)
Row i gives the features of node i



Graph neural network

A graph neural network is a **composition** of functions of the form:

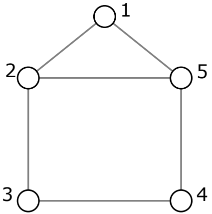
$$X \mapsto U = PX \mapsto U' = UW^T + 1b^T \mapsto V = \sigma(U')$$

Exercise

The feature matrix X is:

$$\begin{pmatrix} 0 & -1 \\ 2 & 1 \\ -1 & 0 \\ 0 & 0 \\ -1 & 1 \end{pmatrix}$$

A neuron of the first layer has weights $w = (-1, 1)$ and bias $b = 0$. What is the output of this neuron with a ReLu activation function?



Problem to solve

Consider a graph neural network with N layers
Let S be the **training set**.

Objective

Find **weight matrices** $W_1, \dots, W_N$ and **bias vectors** $b_1, \dots, b_N$ minimizing:

$$L = -\sum_{i \in S} \sum_{k=1}^{K} 1_{\{y=k\}} \log p_i(k) + \frac{\lambda}{2} \sum_{l=1}^{N} (||W_l||^2 + ||b_l||^2)$$

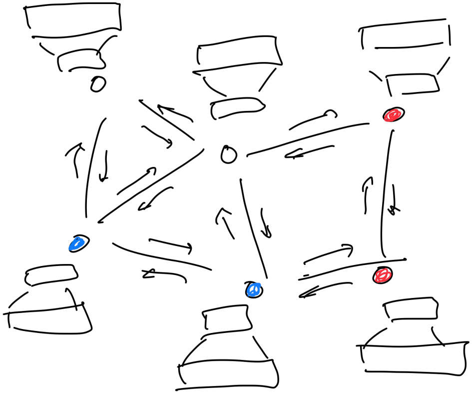
with

$$p_i = f_N \circ \ldots \circ f_1(X)_i$$

and

$$f_l(X) = \sigma_l(PXW_l^T + 1b_l^T)$$

Training set



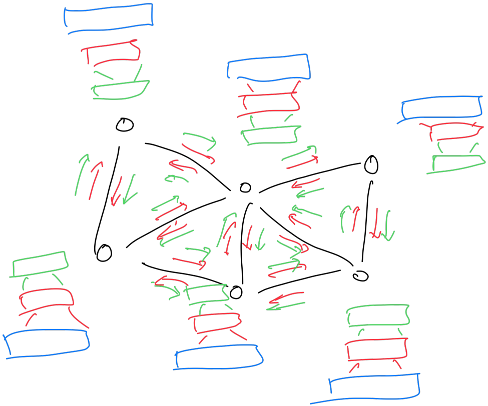
Parameters to learn

$$x \in \mathbb{R}^d,\ y \in \{1, \ldots, K\}$$

$$\boxed{x \mapsto Wx + b}$$

Layer	Weights W	Biases b
1	$d_1 \times d$	d_1
2	$d_2 \times d_1$	d_2
$\vdots$	$\vdots$	$\vdots$
N	$K \times d_{n-1}$	K

Message passing in a GNN with 2 layers



Gradient: Single-layer graph neural network

$$X \mapsto U = XW^T + 1b^T \mapsto U' = PU \mapsto p = \sigma(U') \mapsto L$$

Backpropagation

Given the loss L for each sample of the training set S, compute:

1. $$\frac{\partial L}{\partial p}$$
2. $$\frac{\partial L}{\partial U} = \frac{\partial L}{\partial p}\frac{\partial p}{\partial U}$$
3. $$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial U}\frac{\partial U}{\partial W}$$
   $$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial U}\frac{\partial U}{\partial b}$$

Gradient: A 2-layer graph neural network

$$X \mapsto U_1 = XW_1^T + 1b_1^T \mapsto U_1' = PU_1 \mapsto V_1 = \sigma_1(U_1')$$
$$V_1 \mapsto U_2 = V_1 W_2^T + 1b_2^T \mapsto U_2' = PU_2 \mapsto p = \sigma_2(U_2') \mapsto L$$

Backpropagation

Given the loss L for each sample of the training set S, compute:

- Layer 2:

$$\frac{\partial L}{\partial p} \rightarrow \frac{\partial L}{\partial U_2} = \frac{\partial L}{\partial p}\frac{\partial p}{\partial U_2} \rightarrow \frac{\partial L}{\partial W_2}, \frac{\partial L}{\partial b_2}$$

- Layer 1:

$$\frac{\partial L}{\partial V_1} = \frac{\partial L}{\partial U_2}\frac{\partial U_2}{\partial V_1} \rightarrow \frac{\partial L}{\partial U_1} = \frac{\partial L}{\partial V_1}\frac{\partial V_1}{\partial U_1} \rightarrow \frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial b_1}$$

Observations

GNN as...

- A **neural network** $\rightarrow$ use an **empty** graph
- An **embedding** technique $\rightarrow$ use the **last (hidden) layer**
- A **diffusion** process $\rightarrow$ use **one-hot encoding** of labels + **identity** mapping (no training, $W = I, b = 0$)

Outline

1. Background on neural networks
2. Graph neural networks
3. **Variants**

Message passing

$$X \mapsto U = PX \mapsto U' = UW^T + 1b^T \mapsto V = \sigma(U')$$

Some variants

Replace the transition matrix $P = D^{-1}A$ by:

- $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ (symmetric normalization)
- $I + P$ (add self-embedding)
- (I, P) (concatenate self-embedding) $\rightarrow$ GraphSAGE

Sampling

$$X \mapsto U = PX \mapsto U' = UW^T + 1b^T \mapsto V = \sigma(U')$$

Variant

Replace the transition matrix P by $\tilde{P} = \tilde{D}^{-1}\tilde{A}$ where:

- $\tilde{A}$ is the adjacency matrix of a sampled graph (e.g., at most k neighbors per node)
- The sampling can depend on the layer $\rightarrow$ GraphSAGE

Normalization

$$X \mapsto U = PX \mapsto U' = UW^T + 1b^T \mapsto V = \sigma(U')$$

Variant

Normalize V so that each embedding lies on the **unit sphere**:

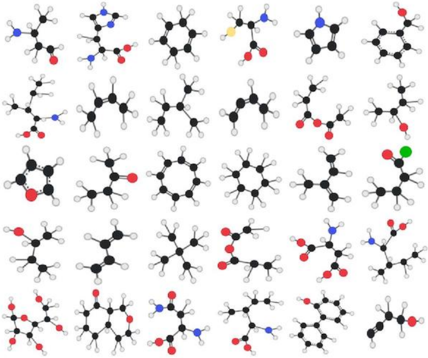
$$V \mapsto V' = \frac{V}{||V||} \quad \to \text{GraphSAGE}$$

Pooling

Idea: From **node** embedding to **graph** embedding

$$X \mapsto U = PX \mapsto U' = UW^T + 1b^T \mapsto V = \sigma(U') \mapsto \frac{1^T V}{n}$$

Each sample = one graph!



Source: Molecules (Javascript library)

Link Prediction

Idea: From **node** embedding to **link prediction**

$$X \mapsto U = PX \mapsto U' = UW^T + 1b^T \mapsto V = \sigma(U') \mapsto S = \sigma(VV^T)$$

Similarity

S is a similarity matrix of size $(n \times n)$.
$S_{i,j}$ is the probability that a link exists between nodes i and j.

In practice, we show both **positive** and **negative** training examples to our model.

Summary

Graph neural networks

- Machine learning for **enriched** graphs
- Neural network boosted by **message passing**
- Many **variants** (diffusion, sampling, normalization, pooling)