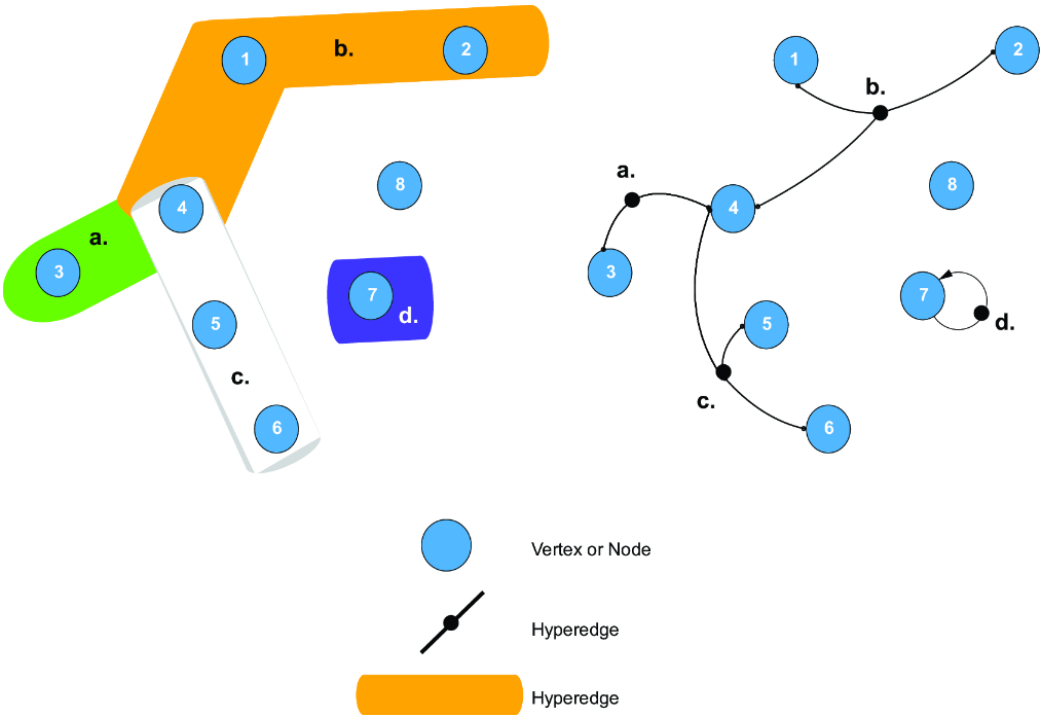
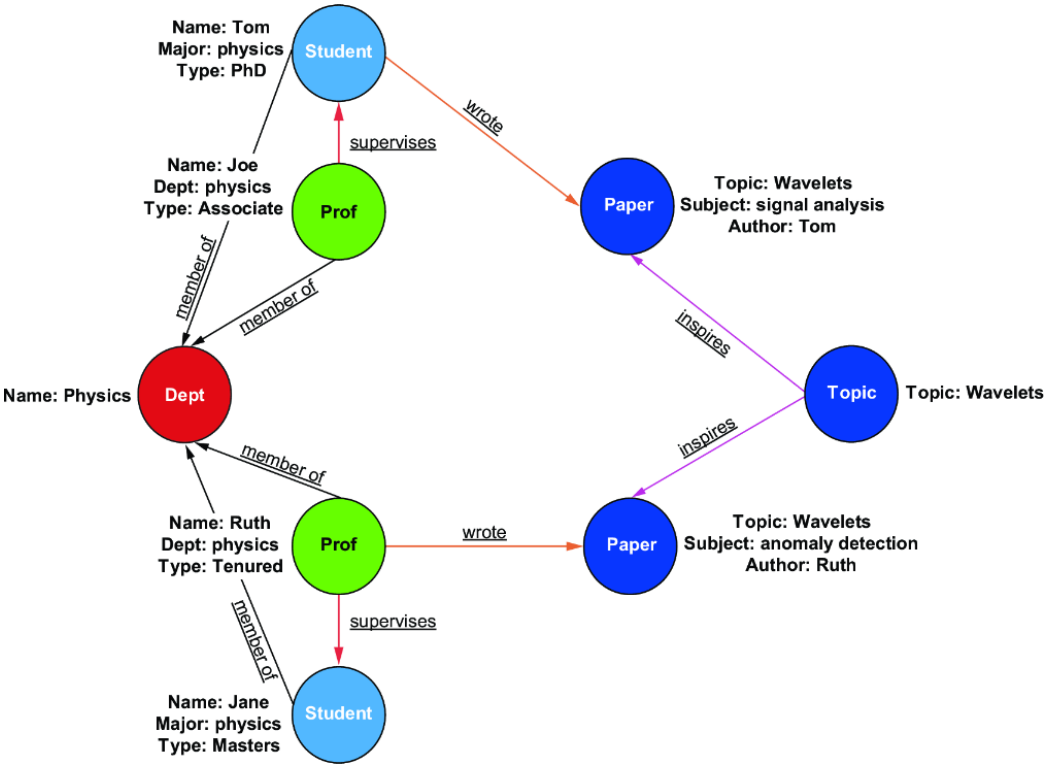
Graph Machine Learning

Introduction

Machine Learning as Optimization: A Simple Summary

Machine learning is fundamentally an **optimization problem** where we search for the best possible model to perform a specific task.





The Core Process

Goal: Find a mathematical model that achieves optimal performance on a given task

Key Components:

1. **Performance Metric** (Loss Function/Cost Function)
2. Quantifies how well the model is performing
3. Lower loss = better performance
4. **Data-Driven Learning**
5. Algorithm receives data (often large amounts)
6. Uses this data to make iterative improvements
7. **The Learning Cycle** (Training):

```
Make Decision/Prediction
    ↓
Evaluate with Loss Function
```

```
      ↓
   Calculate Error
      ↓
   Update Model Parameters
      ↓
   (Repeat until performance is satisfactory)
```

The Essence: At each iteration, the model makes predictions, measures how wrong it is, and adjusts its internal parameters to do better next time. Through repeated cycles, the model progressively improves its performance.

This iterative process of **learning from mistakes** is what we call **training** - it's how machines get "smarter" at their tasks over time.

🎓 Supervised Learning in Graph ML

<x, y>

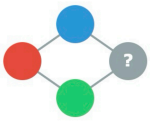
x = Input (graph, node features, network structure)
y = Known output (labels we want to predict)

↓

Goal: Learn from labeled examples, then predict on new data

Node Classification

Discrete Labels

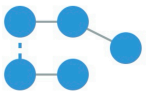


Predict: Node Category

- 📱 Social Network: User interests
- 🧬 Biology: Protein function
- 📄 Citation: Paper topic

Link Prediction

Edge Exists?



Predict: Will edge exist?

- 👥 Social: Friend suggestions
- 🛒 E-commerce: Product recommendations
- 💊 Drug: Protein interactions

Graph Classification

Discrete Labels

Predict: Graph Property

- 🥽 Chemistry: Molecule toxicity
- 📱 Social: Community type
- 🧠 Neuroscience: Brain state

Key Differences from Traditional ML

Traditional ML:

- Independent samples
- Fixed features
- No relationships

Graph ML:

- Connected samples
- Relational features
- Network structure matters

Supervised Learning

<x, y>

Goal: Learn from labeled examples
We know: Inputs AND outputs

All labels known

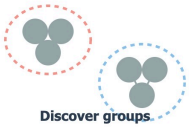
Graph ML Tasks:

- Node classification
- Link prediction
- Graph classification

Unsupervised Learning

<x>

Goal: Find patterns and structure
We know: Only inputs (no labels)



Discover groups

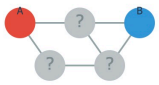
Graph ML Tasks:

- Community detection
- Node clustering
- Graph embedding

Semi-Supervised

<x, y> + <x>

Goal: Use few labels to guide learning
We know: Few labels + many unlabeled



Few labels guide learning

Graph ML Tasks:

- Label propagation
- Few-shot node classification
- Graph SSL methods

Reinforcement Learning

State → Action → Reward/Penalty

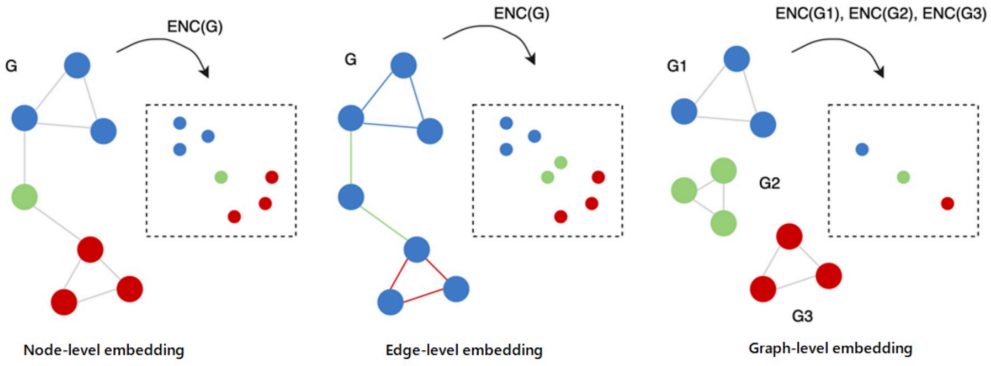
Goal: Learn sequence of decisions
Method: Trial and error with feedback
Objective: Maximize rewards, minimize penalties

Graph ML Tasks:

- Graph navigation/path finding
- Molecule generation
- Network optimization

Learn best path through rewards/penalties

Graph Learning Tasks



Node-level embedding | Edge-level embedding | Graph-level embedding

Node Representation Learning

DeepWalk Architecture

DeepWalk is a two-step approach for learning node representations:

1. **Random Walk Generation**: Generate random walks from the graph (similar to sequences of words)
2. **SkipGram Training**: Use SkipGram to learn node representations

Word2Vec and SkipGram

SkipGram Model: Predicts context words given a target word.

Mathematical Formulation: For a vocabulary of N words: $w_1, w_2, ..., w_N$

The goal is to maximize the probability of context words given the target word:

$$ P(w_{i+j}|w_i) = \frac{exp(v_{w_{i+j}}^T v_{w_i})}{\sum_{w=1}^N exp(v_w^T v_{w_i})} $$

where v_w is the vector representation of word w.

Objective Function:

$$\frac{1}{N}\sum_{n=1}^{N}\sum_{-c\leq j\leq c, j\neq 0}\log p(W_{n+j}|W_n)$$

where c is the context window size.

Node2Vec

Extension of DeepWalk with biased random walks:

- **p parameter**: Controls return to previous node
- **q parameter**: Controls exploration vs exploitation

Random Walk Strategy:

- **BFS-like**: Explores local neighborhood (homophily)
- **DFS-like**: Explores distant nodes (structural roles)

Graph Neural Networks (GNNs)

Message Passing Framework

Core Idea: Update node representations by aggregating information from neighbors.

Mathematical Formulation:

$$h_v^{(l+1)} = \sigma(W^{(l)} \cdot \text{AGGREGATE}^{(l)}(\{h_u^{(l)} : u \in \mathcal{N}(v)\}))$$

where:

- $h_v^{(l)}$ is the representation of node v at layer l
- $\mathcal{N}(v)$ is the neighborhood of node v
- AGGREGATE is an aggregation function
- σ is an activation function

Types of GNNs

1. Graph Convolutional Networks (GCN)

Aggregation Function: $h_v^{(l+1)} = \sigma(W^{(l)} \cdot \frac{1}{\sqrt{|\mathcal{N}(v)|}} \sum_{u \in \mathcal{N}(v)} h_u^{(l)})$

Normalization: Uses symmetric normalization for stability.

2. GraphSAGE

Aggregation Functions:

- Mean
- Max
- LSTM
- Pooling

Inductive Learning: Can generalize to unseen nodes.

3. Graph Attention Networks (GAT)

Attention Mechanism: $\alpha_{ij} = \frac{exp(\text{LeakyReLU}(a^T[Wh_i \| Wh_j]))}{\sum_{k \in \mathcal{N}_i} exp(\text{LeakyReLU}(a^T[Wh_i \| Wh_k]))}$

Weighted Aggregation: $h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} W^{(l)} h_j^{(l)}\right)$

Graph Pooling

Purpose

Reduce graph size while preserving important structural information.

Methods

1. Top-K Pooling

Select top-k nodes based on learnable scores.

2. DiffPool

Learn hierarchical clustering of nodes.

3. SAGPool

Use self-attention for node selection.

Training GNNs

Loss Functions

Node Classification:

$$\mathcal{L} = -\sum_{v \in \mathcal{V}_L} y_v \log(\hat{y}_v)$$

Link Prediction: $\mathcal{L} = -\sum_{(u,v) \in \mathcal{E}} \left[y_{uv} \log(\hat{y}_{uv}) + (1 - y_{uv}) \log(1 - \hat{y}_{uv}) \right]$

Graph Classification:

$$\mathcal{L} = -\sum_{G \in \mathcal{G}} y_G \log(\hat{y}_G)$$

Regularization Techniques

1. **Dropout**: Randomly zero node features during training
2. **Weight Decay**: L2 regularization on model parameters
3. **Early Stopping**: Monitor validation performance
4. **Graph Augmentation**: Add/remove edges, mask features

Applications

1. Social Networks

- **Node Classification**: User interest prediction, fake account detection
- **Link Prediction**: Friend suggestions, influence prediction
- **Community Detection**: Identifying cohesive groups

2. Biological Networks

- **Protein Function Prediction**: Predicting protein roles
- **Drug Discovery**: Molecular property prediction
- **Disease Prediction**: Identifying disease-related genes

3. Computer Vision

- **Scene Understanding**: Modeling object relationships
- **Point Cloud Analysis**: 3D shape classification
- **Image Segmentation**: Pixel-level predictions

4. Natural Language Processing

- **Document Classification**: Hierarchical text classification
- **Knowledge Graphs**: Entity and relation extraction
- **Semantic Parsing**: Converting text to structured representations

Challenges and Limitations

1. Scalability

- **Computational Complexity**: $O(|V|^2)$ for dense graphs
- **Memory Requirements**: Large graphs may not fit in memory
- **Training Time**: Slow convergence for large networks

2. Over-smoothing

- **Problem**: Node representations become indistinguishable after many layers
- **Solutions**: Residual connections, normalization, attention mechanisms

3. Heterogeneous Graphs

- **Multiple Node Types**: Different types of entities
- **Multiple Edge Types**: Different types of relationships
- **Solutions**: Heterogeneous GNNs, meta-paths

4. Dynamic Graphs

- **Temporal Evolution**: Graphs change over time
- **Solutions**: Temporal GNNs, recurrent architectures

Evaluation Metrics

Node Classification

- **Accuracy**: Overall correct predictions
- **F1-Score**: Balance between precision and recall
- **AUC-ROC**: Area under the receiver operating characteristic curve

Link Prediction

- **AUC**: Area under the curve
- **Precision@k**: Precision at top-k predictions
- **MRR**: Mean Reciprocal Rank

Graph Classification

- **Accuracy**: Overall correct predictions
- **Cross-validation**: K-fold cross-validation
- **Statistical Significance**: Paired t-tests, Wilcoxon tests

Future Directions

1. Theoretical Understanding

- **Expressiveness**: What functions can GNNs represent?
- **Convergence**: When and how fast do GNNs converge?
- **Generalization**: How well do GNNs generalize?

2. Scalability

- **Sampling Methods**: Efficient neighbor sampling
- **Distributed Training**: Multi-GPU, multi-machine training
- **Approximation**: Fast approximate algorithms

3. Interpretability

- **Attention Visualization**: Understanding what GNNs attend to
- **Feature Attribution**: Which features contribute to predictions?
- **Graph Explanations**: Explaining graph-level predictions

References

- Hamilton, W. L. (2020). Graph representation learning. Synthesis Lectures on Artificial Intelligence and Machine Learning, 14(3), 1-159.
- Stamile, Claudio, et al. Graph Machine Learning : Take Graph Data to the Next Level by Applying Machine Learning Techniques and Algorithms, Packt Publishing,
- Keita Broadwater and Namid Stillman. Graph Neural Networks in Action. Manning Publications, 2025. ISBN: 9781617299056.