Intelligent Agents
GNNs, ExpressGNN, pLogicNet, MLN Learning

Prof. Dr. Ralf Möller
Universität zu Lübeck
Institut für Informationssysteme
Acknowledgements

• Slides for this presentation are taken from
  – Representation Learning on Networks
    snap.stanford.edu/proj/embeddings-www, WWW 2018
  – Efficient Probabilistic Logic Reasoning with Graph Neural Networks
    Yuyu Zhang, Xinshi Chen, Yuan Yang, Arun Ramamurthy, Bo Li, Yuan Qi
    & Le Song (slides taken from a presentation by Hengda Shi, Gaohong
    Liu and Jian Weng)
  – Probabilistic Logic Neural Network for Reasoning, Meng Qu, Jian Tang
    (slides taken from a presentation by Zijie Huang, Roshni Iyer, Alex
    Wang)

• Slides have been adapted (all faults are mine)
Embedding Nodes

- Encode nodes so that …
  - similarity in the embedding space approximates …
  - similarity in the original network
Embedding Nodes

\[
similarity(u, v) \approx z_v^T z_u
\]

Need to define!

d-dimensional embedding

original network

encode nodes

embedding space

Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018
Recap: Dot Product

\[ \mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \]

\[ \mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos \varphi \]

\[ \cos (\varphi) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} \]

Orthogonale Projektion \( \vec{b}_a \) des Vektors \( \vec{b} \) auf die durch \( \vec{a} \) bestimmte Richtung
Simple ("Shallow") Embedding Approaches

Solve optimization problem
• Select embedding vectors for nodes such that "similar" nodes have similar vectors

Various ways to specify similarity of nodes
• Adjacency-based embedding
• Multi-hop embedding
• Random walk approaches

Vectors with d components (with d being a hyperparameter)

More or less clever approaches, but appropriate similarity features should better be found automatically

From “Shallow” to “Deep”

- Shallow: Define features based on selected features

\[ Z = \text{Dimension/size of embeddings } d \]

Embedding matrix

Embedding vector for a specific node

\[ Z_j \]

Dimension/size of embeddings

One column per node
From “Shallow” to “Deep”

• Limitations of shallow encoding:
  – $O(|V_d|)$ parameters needed
    • No parameter sharing
    • Every node has its own unique embedding vector
  – Inherently “transductive” (not inductive)
    • Impossible to generate embeddings for nodes that were not seen during training
  – Does not incorporate node features
    • Many graphs have nodes with features that we can and should leverage

• Need to find embeddings based on holistic view on graph

Scarselli et al. The Graph Neural Network Model. IEEE Transactions on Neural Networks. 2005,
Setup

• Assume we have a graph $G$:
  – $V$ is the vertex set.
  – $A$ is the adjacency matrix (assume binary).
  – $X \in \mathbb{R}^{m \times |V|}$ is a matrix of nodes and their features.
    • Categorical attributes, text, image data
      – E.g., profile information in a social network.
    • Node degrees, clustering coefficients, etc.
    • Indicator vectors (i.e., one-hot encoding of each node)
Neighborhood Aggregation

- **Key idea**
  - Generate node embeddings based on local neighborhoods
  - Nodes aggregate information from their neighbor
  - Computation graph for every node
Neighborhood Aggregation

- Nodes have embeddings at each layer
- Model can be of arbitrary depth
- “layer-0” embedding of node $u$ is its input features, i.e., $x_u$

**What’s in the boxes?**
Graph Networks (GNNs)

- **Basic approach**: Average neighbor messages and apply a linear transformation with non-linear normalization.
- Define a loss function on the embeddings, $\mathcal{L}(z_u)$.

Initial "layer 0" embeddings are equal to node features:

$$h^0_v = x_v$$

The $k$th layer embedding of $v$ is given by:

$$h^k_v = \sigma \left( \sum_{u \in N(v)} \frac{h^{k-1}_u}{|N(v)|} W_k \right) + B_k h^{k-1}_v$$

where $\sigma$ is the non-linearity (e.g., ReLU or tanh), and $h^{k-1}_v$ is the previous layer embedding of $v$.
Unsupervised Training

After $K$-layers of neighborhood aggregation, we get output embeddings for each node

Feed these embeddings into any loss function …

and run stochastic gradient descent to train the aggregation parameters
Supervised Training

• E.g., based on node classification \( y_v \in \{0, 1\} \):

\[
\mathcal{L} = \sum_{v \in V} y_v \log(\sigma(z_v^{\theta})) + (1 - y_v) \log(1 - \sigma(z_v^{\theta}))
\]

- Human or bot?
- Classification weights
- Output node embedding
- Node class label

Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018
Overview of Model Design

1) Define a neighborhood aggregation function.

2) Define a loss function on the embeddings, $\mathcal{L}(z_u)$
Overview of Model Design

3) Train on a set of nodes, i.e., a batch of compute graphs
Overview of Model

4) Generate embeddings for nodes as needed

Even for nodes we never trained on!!!!
Inductive Capability

- Same aggregation parameters are shared for all nodes.
- Number of model parameters is sublinear in |V| …
- … and we can generalize to unseen nodes!
Inductive Capability

Inductive node embedding → generalize to entirely unseen graphs

e.g., train on protein interaction graph from model organism A and generate embeddings on newly collected data about organism B
Many application settings constantly encounter previously unseen nodes. e.g., Reddit, YouTube, GoogleScholar, ....

Need to generate new embeddings “on the fly”
Neighborhood Aggregation

- Key distinctions are in how different approaches aggregate messages.

What else can we put in the box?
Graph Convolutional Networks (GCNs)

- Slight variation on the neighborhood aggregation idea:
  \[
  h^k_v = \sigma \left( \sum_{u \in N(v) \cup v} \frac{h^k_{u}}{\sqrt{|N(u)||N(v)|}} W_k X_u \right)
  \]
  - Same matrix for self and neighbor embeddings
  - Per-neighbor normalization

- Empirically, this configuration to give the best results
  - More parameter sharing
  - Down-weights high degree neighbors

Semi-supervised classification with graph convolutional networks
GraphSAGE (SAmple and aggreGatE)

- So far we have aggregated the neighbor messages by taking their (weighted) average. Can we do better?

$$h^k_v = \sigma \left( A_k \cdot \text{AGG} \left( \{ h^k_{u^{-1}}, \forall u \in N(v) \} \right), B_k h^k_{v^{-1}} \right)$$

concatenate self embedding and neighbor embedding

Any differentiable function that maps set of vectors to a single vector.
GraphSAGE Variants

- **Mean:**
  \[ AGG = \sum_{u \in N(v)} \frac{h_{u}^{k-1}}{|N(v)|} \]

- **Pool:**
  - Transform neighbor vectors and apply symmetric vector function
  \[ AGG = \gamma\left(\{Qh_{u}^{k-1}, \forall u \in N(v)\}\right) \]
  \[ AGG = \text{LSTM} \left( [h_{u}^{k-1}, \forall u \in \pi(N(v))] \right) \]

- **LSTM-based RNNs:**
  - Apply LSTM to random permutation of neighbors (LSTMs work on sequences)

- **Transformers?**
Neighborhood Aggregation

- GCNs and GraphSAGE generally only 2-3 layers deep
- What if we want to go deeper?
  - Overfitting from too many parameters.
  - Vanishing/exploding gradients during backpropagation
Gated Graph Networks

- Use techniques from recurrent networks
- Parameter sharing across layers, recurrent state update

Handle >20 layers:
- Allows for complex information about global graph structure to be propagated to all nodes
Neighborhood aggregation with RNN state update

1. Get “message” from neighbors at step k:

\[ m^k_v = W \sum_{u \in N(v)} h^k_{u} \]

Aggregation function does not depend on k

2. Update node “state” using Gated Recurrent Unit (GRU)
New node state depends on the old state and the message from neighbors:

\[ h^k_v = \text{GRU}(h^{k-1}_v, m^k_v) \]

Yujia Li Richard Zemel Marc Brockschmidt Daniel Tarlow,
Gated Graph Sequence Neural Networks
(Sub)graph Embeddings

• So far we have focused on node-level embeddings…

• But what about subgraph embeddings?
(Sub)graph Embeddings

- Use representative as a virtual node

\[ z_S = \sum_{v \in S} z_v \]

- Sum or average node embeddings:

How to embed (sub)graphs with millions or billions of nodes?


How to do the analog of CNN “pooling” on networks?
Summary so far

- **Key idea:** Generate node embeddings based on local neighborhoods.
  - **GraphSAGE**
    - Generalized neighborhood aggregation
  - **Gated Graph Networks**
    - Neighborhood aggregation + recursion (same mappings for a layer) + GRUs
  - **Graph Convolutional Networks**
    - Average neighborhood information and stack computational networks
Recent Advances in Graph Networks

- **Attention-based** neighborhood aggregation *(Weightings for neighbors)*
  - Graph Attention Networks *(Velickovic et al., 2018)*
  - GeniePath *(Liu et al., 2018)*

- Generalizations based on **spectral convolutions** *(eigen-decomposition of graph Laplacian $L$)*
  - Geometric Deep Learning *(Bronstein et al., 2017)*
  - Mixture Model CNNs *(Monti et al., 2017)*

- **Speed improvements via** subsampling
  - FastGCNs *(Chen et al., 2018)*
  - Stochastic GCNs *(Chen et al., 2017)*

$L = D - A$ (degree matrix – adjacency matrix)
Graph Networks, Embeddings, and KGs

• Graph networks allow for the computation of embeddings for nodes in a KG
• With embeddings, existence of links between nodes can be estimated (KG completion)
  – See also, e.g., node2vec
• If nodes originate from words …
• … we have another way to embed nodes
  – See also, e.g., word2vec
  – KG completion based on word embeddings

node2vec: Scalable Feature Learning for Networks. A. Grover, J. Leskovec. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2016
Approaches for Completing KGs

- Graph Network approach
- Embedding approach
- What about probabilistic graphical models?
  - Markov logic networks for link estimation?
Advantages of MLNs

- Incorporate domain knowledge with first-order logic
  - Composition
    - $\forall X, Y, Z: R_1(X, Y) \land R_2(Y, Z) \rightarrow R_3(X, Z)$
  - Inverse relations
    - $\forall X, Y: R_1(X, Y) \rightarrow R_2(Y, X)$
  - Symmetry
    - $\forall X, Y: R_1(X, Y) \rightarrow R_1(Y, X)$
  - Subrelation
    - $\forall X, Y: R_1(X, Y) \rightarrow R_2(X, Y)$
MLNs and Knowledge Graphs

Ground KG of MLN

Knowledge graph (e.g., from text) as evidence
Markov Logic Networks

Pros:

• Logic formulas incorporate prior knowledge
• Allows MLN to generalize in tasks with small amount of labeled data

Cons:

• Inference in MLN is computationally intensive
• Logic formulas can only cover a small part of the possible combinations of knowledge graph relations in real-world texts
Graph Networks / Word-based Embeddings

Pros:

- **Efficiency** – Directly work on KG
- **Compactness** – GNNs with shared parameters can be memory efficient
- **Expressiveness** – GNN can capture structure knowledge encoded in the KG, and so-called *tunable embeddings* can encode *entity-specific information* (see below)

Cons:

- GNNs do not explicitly incorporate prior knowledge into models and may require many labeled examples for a target task
Combining MLNs and KG Embeddings

- KG Completion
  - With MLN model $M_i$
  - With embedding approaches
    - GNN-based (ExpressGNN)
      - Efficient Probabilistic Logic Reasoning with Graph Neural Networks
        Yuyu Zhang, Xinshi Chen, Yuan Yang, Arun Ramamurthy, Bo Li, Yuan Qi & Le Song.
    - Embedding based (pLogicNet)
      - Probabilistic Logic Neural Network for Reasoning, Meng Qu, Jian Tang.
- Learn new MLN model $M_{i+1}$ based on completed KG
Variational EM for MLN Learning

- MLN used to model the joint probabilistic distribution of all observed and latent variables, $O$ and $H$, respectively

$$P_w(O, H) := \frac{1}{Z(w)} \exp\left(\sum_{f \in F} w_f \sum_{a_f \in A_f} \phi(a_f)\right)$$

- Training an MLN ($w, F$) means to determine the weights $w_f$ of the formulas $f \in F$

- An MLN can be trained by maximizing the log-likelihood of all observed facts $\log P_w(O)$, i.e., $\hat{w}_{ML} = \arg\max_w \log P_w(O)$ and $w = \hat{w}_{ML}$

- Due to intractability caused by hidden variables, instead of optimizing the log-likelihood, we optimize the evidence lower bound (ELBO) s.t.

$$\log P_w(O) \geq L_{ELBO}(Q_\theta, P_w)$$

- The goal is to find a distribution $Q_\theta$ that approximates $P_w$ “from below”
E step: Inference

- Infer the posterior distribution of the latent variables, where $P_w$ is fixed and $Q_\theta$ is optimized to minimize the KL divergence between $Q_\theta(H|O)$ and $P_w(H|O)$
- Estimate the true posterior with a mean-field approximation
- In mean-field variational approximation, each unobserved ground formula $r(a_r) \in H$ is independently inferred as:
  $$Q_\theta(H|O) := \prod_{r(a_r) \in H} Q_\theta(r(a_r))$$
- Each $Q_\theta(r(a_r))$ follows a Bernoulli distribution
- Parameterize $Q_\theta$ with GNN

$a_r$ is a sequence of parameters that fits the arity of $r$

$r(a_r) \in H$ is a slight abuse of notation
E step: Inference

- With mean-field approximation, $L_{ELBO}(Q_\theta, P_w)$ can be reorganized as:

$$L_{ELBO}(Q_\theta, P_w) = \mathbb{E}_{Q_\theta(H|O)}[\log P_w(O, H) - \log Q_\theta(H|O)]$$

$$= \mathbb{E}_{Q_\theta(H|O)} \left[ \log \left( \frac{1}{Z(w)} \exp \left( \sum_{f \in F} w_f \sum_{a_f \in A_f} \phi_f(a_f) \right) \right) \right] - \mathbb{E}_{Q_\theta(H|O)} \left[ \log \prod_{r(a_r) \in H} Q_\theta(r(a_r)) \right]$$

$$= \mathbb{E}_{Q_\theta(H|O)} \left[ \sum_{f \in F} w_f \sum_{a_f \in A_f} \phi_f(a_f) - \log(Z(w)) \right] - \mathbb{E}_{Q_\theta(H|O)} \left[ \sum_{r(a_r) \in H} \log Q_\theta(r(a_r)) \right]$$

$$= \sum_{f \in F} w_f \sum_{a_f \in A_f} \mathbb{E}_{Q_\theta(H|O)}[\phi_f(a_f)] - \log(Z(w)) - \sum_{r(a_r) \in H} \mathbb{E}_{Q_\theta(H|O)}[\log Q_\theta(r(a_r))]$$

$$\mathbb{E}_{P(x)}[f(x)] = \sum_{i=1}^{I} p_i f(a_i).$$
E step: Inference

- The term $\sum_{f \in F} w_f \sum_{a_f \in A_f} \mathbb{E}_{Q_\theta(H|O)}[\phi_f(a_f)]$ sums over all possible logic formulae and all possible assignments to each formula.

- The term $\sum_{r(a_r) \in H} \mathbb{E}_{Q_\theta(H|O)}[\log Q_\theta(r(a_r))]$ sums over all possible latent variables.

- Therefore, both terms used in the objective function make the computational problem intractable.
E step: Inference with Sampling

• How can we deal with this problem?

• Do not iterate over all possible values but use sampling
  • In each optimization iteration, a batch of ground formulae will be sampled
  • For each formula in sampled batch, do the computations w.r.t. observations of involved latent variables
  • $Z(w)$ needs to be adapted in order to compensate for sampling
E step: Add-on

- If the task has sufficient labeled data, a supervised learning objective will be added to enhance parameter estimation

\[ L_{\text{label}}(Q_\theta) = \sum_{r(a_r) \in O} \log Q_\theta(r(a_r)) \]

- The label loss function is complementary to ELBO on predicates that are not well covered by logic rules but have enough observed facts.

- Therefore, the E step objective function that combines knowledge in the MLN and supervision from labeled data would be (\( \lambda \) is a hyperparameter):

\[ L_\theta = L_{\text{ELBO}}(Q_\theta, P_w) + \lambda L_{\text{label}}(Q_\theta) \]
M step: Learning

- In the M step, the weights of logic formulae in MLN will be learned with the variational posterior $Q_\theta(H|O)$ being fixed.

- The partition function $Z(w)$ is not a constant anymore.

- Due to exponential number of terms in $Z(w)$, pseudo log-likelihood needs to be adopted as an alternative objective for optimization.

- Recall the pseudo log-likelihood
  \[ \log P_w(O) \approx \sum_i \log P_w(o_i|o_{N(i)}) \]

- For the neighborhood $N(i)$ we use the Markov blanket $MB$. 
Pseudo-Likelihood

\[ P^*_w(O, H) := \mathbb{E}_{Q_\theta(H|O)} \left[ \sum_{r(a_r) \in H} \log P_w(r(a_r)|MB_r(a_r)) \right] \]

\[ \nabla_{w_i} \mathbb{E}_{Q_\theta(H|O)} \left[ \sum_{r(a_r) \in H} \log P_w(r(a_r)|MB_r(a_r)) \right] \]

\[ \approx y_{r(a_r)} - P_w(r(a_r)|MB_r(a_r)) \]

- where \( y_{r(a_r)} = 0 \) or 1 if \( r(a_r) \) is an observed fact or \( y_{r(a_r)} = Q_\theta(r(a_r)) \) otherwise

- \( MB_r(a_r) \) is the Markov blanket of the ground predicate \( r(a_r) \), i.e., the set of ground predicates that appear in some grounding of a formula with \( r(a_r) \)
Sampling Scheme for M step

- **Computationally intractable** to use all possible ground predicates to compute the gradients

- The solution is to only consider all ground formulae with **at most one latent predicate** and pick up the ground predicate if its truth value determines the formula’s truth value

- Using this sampling scheme, only a small subset of ground predicates is kept and each of which can directly determine the truth value of a ground formula in MLN

- Intuitively, this small subset contains all representative ground predicates, and makes **good estimation of the gradients** with much cheaper computational cost
Representing graph structures

• Foundation of MLN learning is a knowledge graph $\mathcal{G}_K$ for (deterministic) observations
• For MLN learning with hidden variables, additional graph structures need to be generated (albeit with sampling)
• Can graph structures be represented in a clever way such that we gain efficiency?
• Can we use, e.g., GNN operations to derive the variational distribution $Q$?
ExpressGNN

- ExpressGNN has three parts:
  - Vanilla Graph Network
  - Tunable Embeddings
  - Define posterior using embeddings
Vanilla GNN

- Vanilla GNN is built on knowledge graph $G_K$, which is much smaller than the ground graph of MLN (for simplicity it is assumed that each predicate has arity 2).

- GNN parameters $\theta_1$ and $\theta_2$ are independent of number of entities.

- There are $O(d^2)$ parameters given $d$-dimensional embeddings, $\mu_c \in \mathbb{R}^d$

---

**Algorithm 1: GNN()**

- Initialize entity node: $\mu_c^{(0)} = \mu_0, \ \forall c \in \mathcal{C}$
- for $t = 0$ to $T - 1$
  - Compute message $\forall r(c, c') \in \mathcal{O}$
    - $m_{c' \rightarrow c}^{(t)} = \text{MLP}_1(\mu_{c'}^{(t)}, r; \theta_1)$
  - Aggregate message $\forall c \in \mathcal{C}$
    - $m_{c}^{(t+1)} = \text{AGG}(\{m_{c' \rightarrow c}^{(t)}\}_{c':r(c,c') \in \mathcal{O}})$
  - Update embedding $\forall c \in \mathcal{C}$
    - $\mu_c^{(t+1)} = \text{MLP}_2(\mu_c^{(t)}, m_c^{(t+1)}; \theta_2)$
- return embeddings $\{\mu_c^{(T)}\}$
Tunable Embeddings

- For each entity in the KG, we then augment its GNN embedding with a tunable embedding $\mathbf{w}_c \in \mathbb{R}^k$ as $\hat{\mu} = [\mu_c, \mathbf{w}_c]$
- The tunable embeddings increase the expressiveness of the model
- Otherwise, the same embeddings could be produced for nodes that should be distinguished [Zhang et al. 20]
- As there are $M$ entities, the number of parameters in tunable embeddings is $O(kM)$
Finally, define the variational posterior with augmented embeddings of $c_1$ and $c_2$

- Define the posterior $Q_\theta(r(c_1, c_2)) = \sigma \left( MLP_3(\hat{\mu}_{c_1}, \hat{\mu}_{c_2}, r; \theta_3) \right)$
  where $\sigma(\cdot)$ is the sigmoid function

- The number of parameters in $\theta_3$ is $O(d + k)$
In summary, ExpressGNN can be viewed as two-level encoding of entities:

- First two MLPs assign similar embeddings to similar entities in the KG
- Expressive tunable embeddings provide additional model capacity to encode entity information beyond graph structures

By tuning $d$ and $k$, ExpressGNN can trade-off between model compactness and expressiveness

When the number of entities $M$ is large, ExpressGNN can reduce $k$ to save parameters

Summary so far: ExpressGNN
pLogicNet: Embeddings again

- Each entity $e \in E$ and relation $r \in R$ in a KG is associated with an embedding $x_e$ and $x_r$
- The joint distribution of all triplets can be defined as:

$$p(v_O, v_H) = \prod_{(h,r,t) \in O \cup H} \text{Ber}(v_{(h,r,t)} | f(x_h, x_r, x_t))$$

- Where $f(\cdot, \cdot, \cdot)$ is the scoring function on the entity and relation embeddings that computes the probability of the triplet $(h, r, t)$ being true
- Example: the $f$ used in TransE (a KG embedding model) is formulated as:

$$\sigma(\gamma - ||x_h + x_r - x_t||)$$
Combining MLNs with Embedding Approaches

\[ p_w(\mathbf{v}_{(h,r,t)}|\mathbf{v}_O) \propto \left\{ q_\theta(\mathbf{v}_{(h,r,t)}) + \lambda p_w(\mathbf{v}_{(h,r,t)}|\hat{\mathbf{v}}_{\text{MB}}(h,r,t)) \right\} \]

KGE model \hspace{2cm} Markov Logic Network

MLNs w/ Node Attributes

- Model the joint distribution of the object labels given object features, i.e., \( p(\mathbf{y}_V|\mathbf{x}_V) \)

\[
p(\mathbf{y}_V|\mathbf{x}_V) = \frac{1}{Z(\mathbf{x}_V)} \prod_{(i,j) \in E} \psi_{i,j}(y_i, y_j, \mathbf{x}_V)
\]

\[
\psi_{i,j}(y_i, y_j, \mathbf{x}_V) = \exp\left( \sum_{k=1}^{K} \lambda_k f_k(y_i, y_j, \mathbf{x}_i, \mathbf{x}_j) + \mu_k g_k(y_i, \mathbf{x}_i) \right)
\]
MLN Learning with pLogicNet

- pLogicNet formulates the **joint distribution of all graph triplets** with a **Markov logic network**, which is trained with the **variational EM algorithm**.

  In **E-step**, a KGE model infers missing (hidden) triplets. **Knowledge preserved by logic rules can be effectively distilled into the learned embeddings**.

  In **M-step**, the weights of the logic rules are updated based on both the observed triplets and those inferred by KGE model. Therefore, **KGE model provides extra supervision for weight learning**.

57
Summary

- MLNs help to augment KGs
  - Constraints on graphs describe completion rules
- KGs can be used to support MLN learning
  - Embedding-based graph completion to infer missing (hidden) information, which can be used for learning
  - GNN embeddings of KG nodes for variational EM

- We can in principle combine the advantages of both worlds, logic and embedding approaches!