Deep Learning - II

M. Vazirgiannis

LIX, Ecole Polytechnique

December 2019

- **Recurrent NNs + LSTMs**
- Autoencoders
- Graph node embeddings

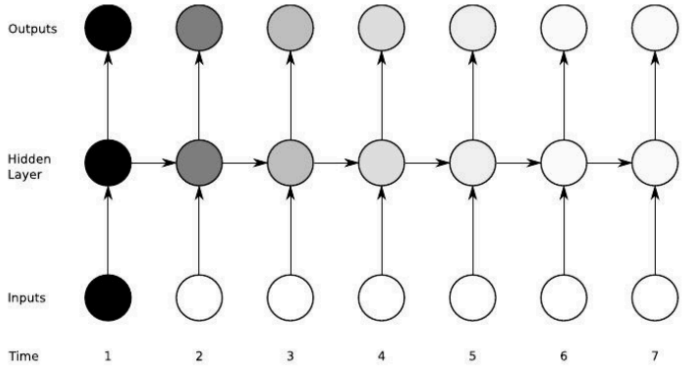
Recurrent Neural Networks

- CNNs are good at image classification
  - Input: an image
  - Output: probability of the class
    - $p(Red\ Panda|\ image) = 0,9$
    - $p(Cat\ |\ image) = 0,1$

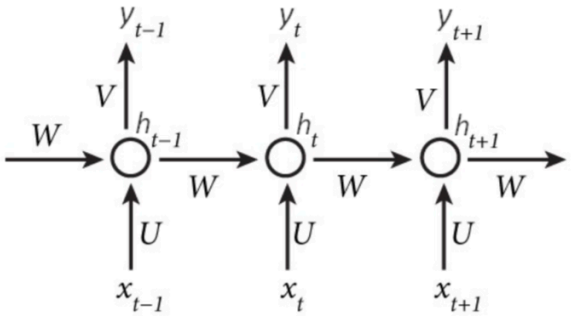
- Sequence learning: study of machine learning algorithms designed for sequential data (time series, language…)
  - Translate: "*machine learning is a challenging topic*" to French: "*L'apprentissage automatique est un sujet difficile*"
  - Predict the next word: "*John visited Paris, the capital of ….* "
  - Input and output strongly correlated within the sequence.

Recurrent Neural Networks

- Update the hidden state in a deterministic nonlinear way.
- RNNs are powerful,
- Distributed hidden state that allows them to store a lot of information about the past efficiently.
- Non-linear dynamics that allows them to update their hidden state in complicated ways.
- No need to infer hidden state, pure deterministic.
- Weight sharing

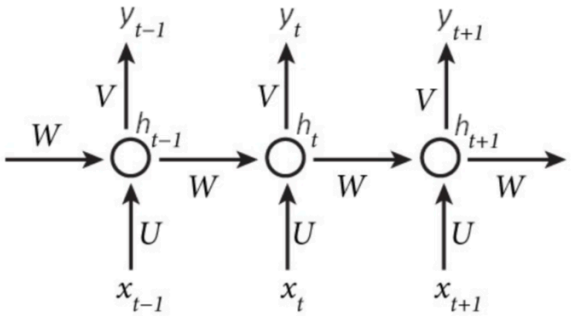


Recurrent Neural Networks



- input: ordered list of input vectors $x_1, \ldots, x_T$ initial hidden state h_0 initialized to all zeros,
- output
  - ordered list of hidden states $h_1, \ldots, h_T$,
  - ordered list of output vectors $y_1, \ldots, y_T$.
  - The output vectors may serve as input for other RNN units, when considering deep architectures .
  - The hidden states correspond to the “short-term” memory of the network.
- The last hidden state represents the encoding (embedding) of the time series

Recurrent Neural Networks



$$h_t = f(Ux_t + Wh_{t-1} + b)$$

- f a nonlinear function
- $x_t \in R^{d_{in}}, U \in R^{Hxd_{in}}, W \in R^{HxH}$,

Parameter matrices

- d_{in} size of the vocabulary
- H: dimension of the hidden layer (H~100)
- $y_t \in R^{d_{out}}$ transforms the current hidden state h_t to depend on the final task:
  - i.e. for classification $y_t = softmax(Vh_t)$
- $V \in R^{d_{out}\chi H}$ parameter matrix shared across all steps (i.e. for word level language model $dout=|V|$

Deep RNNs

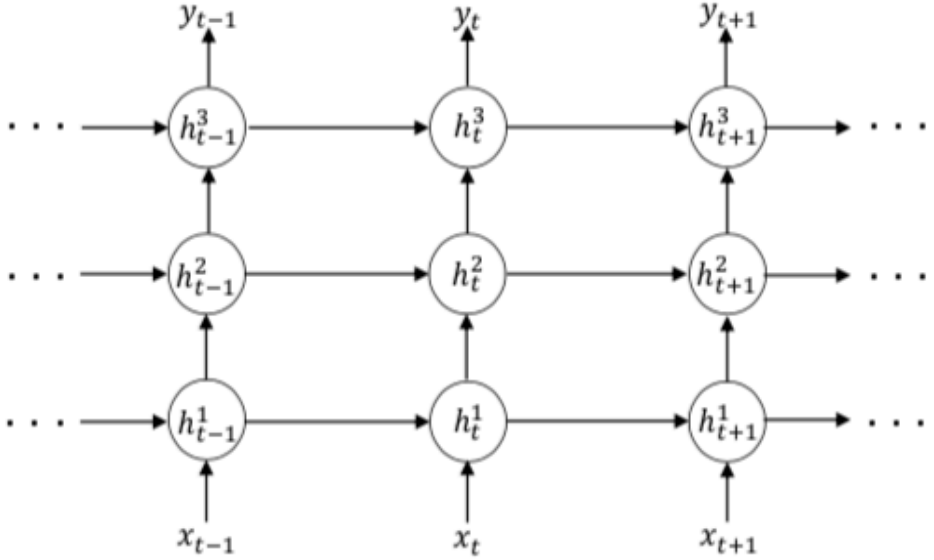
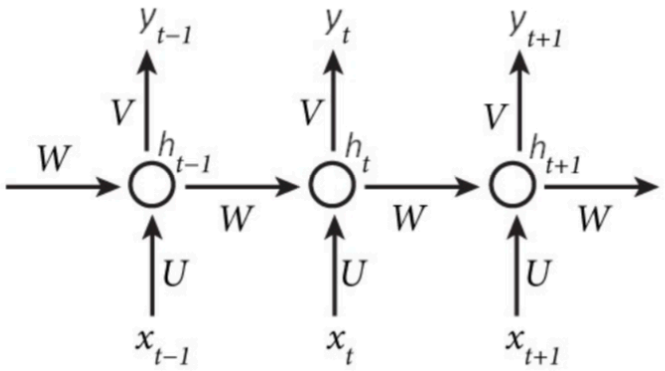


Figure 7: 3 steps of an unrolled deep RNN. Each circle represents a RNN unit. The hidden state of each unit in the inner layers (1 & 2) serves as input to the corresponding unit in the layer above.

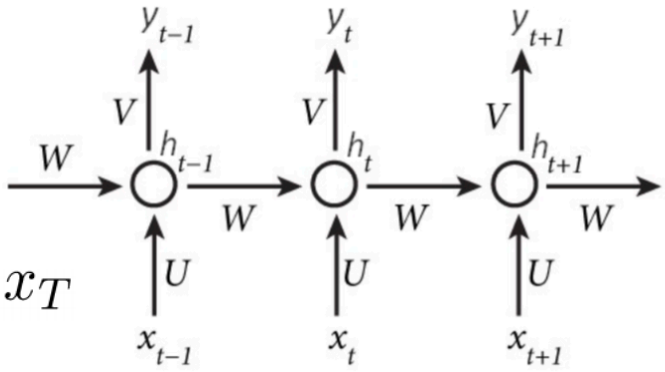
Recurrent Neural Networks



3 steps of an unrolled RNN

- CNNs are naturally efficient with grids[8],
- RNNs were specifically developed to be used with sequences
  - time series, or, in NLP, words (sequences of letters) or sentences (sequences of words).
  - language modeling $P[w_n | w_1, ..., w_{n-1}]$.
  - RNNs trained with such objectives can be used to generate new and quite convincing sentences from scratch
  - RNN can be considered as a chain of simple neural layers that share the same parameters.

Learning in RNNs



- Assuming input $x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T$
- As a single time: $h_t = \sigma(W h_{t-1} + U x_t)$

$$y_t = softmax(V h_t)$$

Learning in RNNs

$$h_t = \sigma(W h_{t-1} + U x_t)$$
$$y_t = softmax(V h_t)$$

- Main idea: we use the same set of W,U,V weights at all time steps!
- $h_0 \in R^{HxH}$ initialization vector for the hidden layer at time step 0
- $\hat{y} \in R^{|V|}$ is a probability distribution over the vocabulary
- Loss function (@ time t): cross entropy predicting words instead of classes

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}$$

Learning in RNNs – backpropagation in time

- Learning the weights of W:

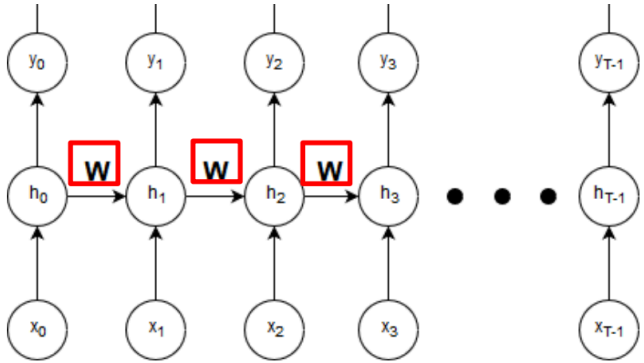
$$\mathbf{W} -> \mathbf{W} - \alpha \frac{\partial \mathbf{y}}{\partial \mathbf{W}}$$

- Computation involves summation over all paths regarding
- i. time

$$\frac{\partial \mathbf{y}}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \frac{\partial \mathbf{y}_j}{\partial \mathbf{W}}$$

- ii. Levels of hidden layers

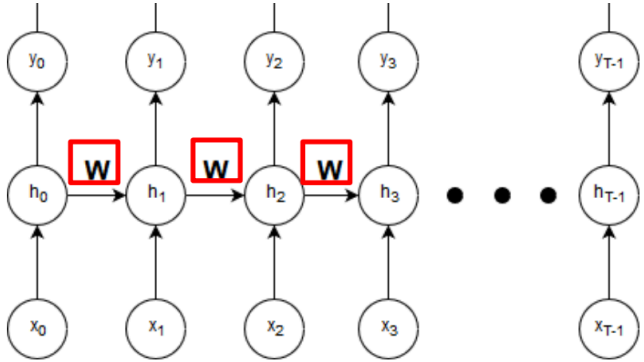
$$\frac{\partial y_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \frac{\partial y_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$



Learning in RNNs – backpropagation in time

- Therefore:

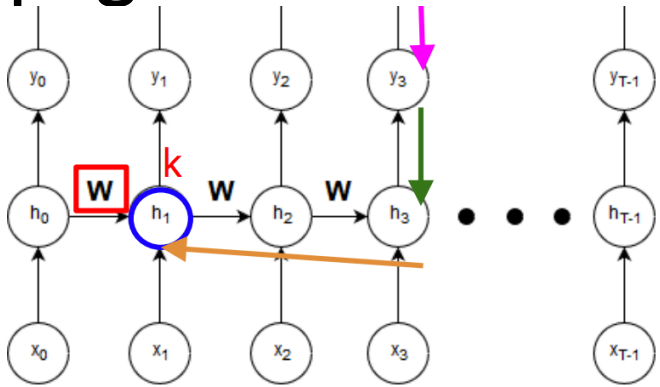
$$\frac{\partial y_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \frac{\partial y_j}{\partial h_j} \frac{\partial h_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$



- Indirect dependency. One final use of the chain rule:

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}}$$

Learning in RNNs – back propagation in time



$$\frac{\partial y_j}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial y_j}{\partial h_j} \left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W}}$$

Learning in RNNs – vanishing/exploding gradients

$$\frac{\partial y_j}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial y_j}{\partial h_j} \left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W}}$$

$$h_m = f(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)$$

$$\frac{\partial h_m}{\partial h_{m-1}} = \mathbf{W}_h^T \mathrm{diag}\left(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)\right)$$

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \mathbf{W}_h^T \mathrm{diag}\left(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)\right)$$

Weight Matrix

Derivative of activation function

- Repeated matrix multiplications leads to vanishing and exploding gradients.

Learning in RNNs – vanishing/exploding gradients

- Initialization of W with 1s
- Using relu as activation function $f(z) = rect(z) = max(z, 0)$
- Using - clip gradients to a maximum value [Mikolov]

Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

LSTM Unit

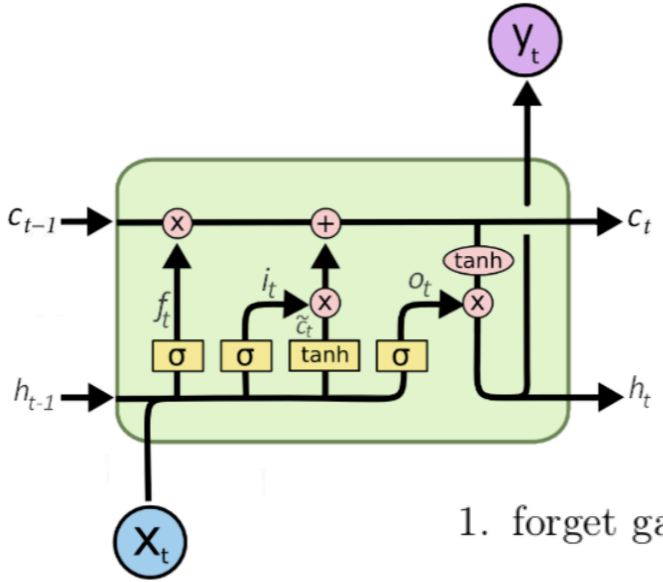
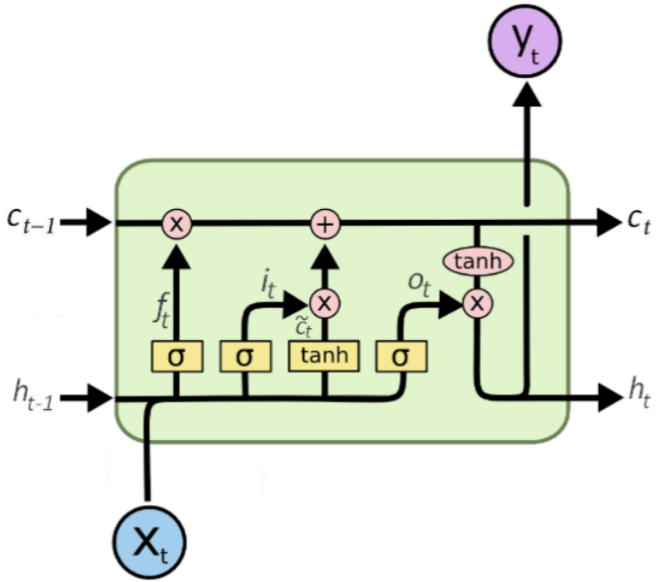


Figure 8: The LSTM unit. Adapted fro

- To tackle RNN problems of i. vanishing gradients and ii. Keep track of information for longer term.
- There is a "memory bus" Ct on which we chose how much to write and read

1. forget gate layer: $f_t = \sigma\left(U_f x_t + W_f h_{t-1} + b_f\right)$
2. input gate layer: $i_t = \sigma\left(U_i x_t + W_i h_{t-1} + b_i\right)$
3. candidate values computation layer: $\tilde{c}_t = \tanh\left(U_c x_t + W_c h_{t-1} + b_c\right)$
4. output gate layer: $o_t = \sigma\left(U_o x_t + W_o h_{t-1} + b_o\right)$

LSTM Unit



1. forget gate layer: $f_t = \sigma\left(U_f x_t + W_f h_{t-1} + b_f\right)$
2. input gate layer: $i_t = \sigma\left(U_i x_t + W_i h_{t-1} + b_i\right)$
3. candidate values computation layer: $\tilde{c}_t = \tanh\left(U_c x_t + W_c h_{t-1} + b_c\right)$
4. output gate layer: $o_t = \sigma\left(U_o x_t + W_o h_{t-1} + b_o\right)$

Assume a new training example x_t and the current hidden state h_{t-1},

- forget gate layer f_t determines how much of the previous cell state c_{t-1} should be forgotten (what fraction of the memory should be freed up),
- input gate layer decides how much of the candidate values $\tilde{c_t}$ should be written to the memory: how much of the new information should be learned. Combining the output of the two filters updates the cell state:

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

$$h_t = \tanh(c_t) \circ o_t$$

$$y_t = \text{softmax}(V h_t)$$

References (1/2)

1. Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., & Wierstra, D. (2015). DRAW: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*.
2. Hadsell, Raia, Sumit Chopra, and Yann LeCun. "Dimensionality reduction by learning an invariant mapping." *Computer vision and pattern recognition, 2006 IEEE computer society conference on*. Vol. 2. IEEE, 2006.
3. Luong, Minh-Tang, Hieu Pham, and Christopher D. Manning. "Effective approaches to attention-based neural machine translation." arXiv preprint arXiv:1508.04025 (2015).
4. Mueller, J., & Thyagarajan, A. (2016, February). Siamese Recurrent Architectures for Learning Sentence Similarity. In *AAAI* (pp. 2786-2792).
5. Course slides "Recurrent Neural Networks" Richard Socher, Stanford, 2016
6. Recurrent Neural Network Architectures Abhishek Narwekar, Anusri Pampari, University of Illinois, 2016
7. Neculoiu, Paul, Maarten Versteegh, and Mihai Rotaru. "Learning text similarity with siamese recurrent networks." *Proceedings of the 1st Workshop on Representation Learning for NLP*. 2016.
8. Rush, Alexander M., Sumit Chopra, and Jason Weston. "A neural attention model for abstractive sentence summarization." arXiv preprint arXiv:1509.00685 (2015).
9. Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems. 2014.
10. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., ... & Bengio, Y. (2015, June). Show, attend and tell: Neural image caption generation with visual attention. In International Conference on Machine Learning (pp. 2048-2057).
11. Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016). Hierarchical attention networks for document classification. NAACL 2016 (pp. 1480-1489).

- Recurrent NNs + LSTMs
- **Autoencoders**
- Graph node embeddings

Auto-encoders for unsupervised learning

Autoencoders – the concept

- introduced by Hinton
  - address the problem of "backpropagation without a teacher"
  - Need to formulate an error function – using input data as the teacher [RUM1986]
- more recently, "deep architecture" approach (Hinton et al., 2006; Hinton and Salakhutdinov, 2006; Bengio and LeCun, 2007;Erhan et al., 2010)
  - Restricted Boltzmann Machines (RBMS), stacked, trained bottom up in unsupervised fashion.

Autoencoders

- Assume a set of $n\text{-}dim$ data X_n – we assume transformations $f,\ g$
  - f : *mapping data to a lower dim space* Z_m *(code)*
  - g: *mapping* Z_m *to the an original dim data* $X_n' \sim X_n$
- *Learning: minimize the loss* $L(x(f(g(x))$
  - *|code|< |input|: under-complete AE*
  - *Sparse autoencoders:* $L(x(f(g(x)) + \Omega(Z_m)$

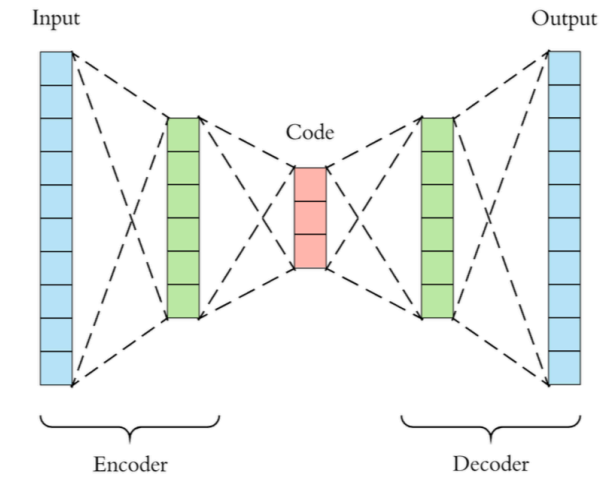
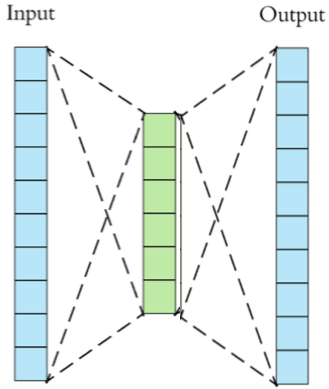


Image from [Dertat 2017]

Applications of AEs

- Learn a low dimensional Z_m representation of the data X
  - learning low dimensional space with less reconstruction error than PCA (Hinton 2006)
  - Features used in classification and improve generalization (Hinton 2007)
- Perform clustering – unsupervised learning
- Information Retrieval
  - Semantic hashing for images and NLP (Hinton 2007, Torralba 2007)
- Learn data features in the absence of a supervised task

Sparse Autoencoders (AE)



- Assume AE: *n input neurons*, m neurons in hidden layer and k training data
- Assume $a_j^{(2)}(x^{(i)})$ activation of *j-th* neuron in hidden layer for *x(i)-th* data point
- average output of *j-th* neuron of the hidden layer after data training:

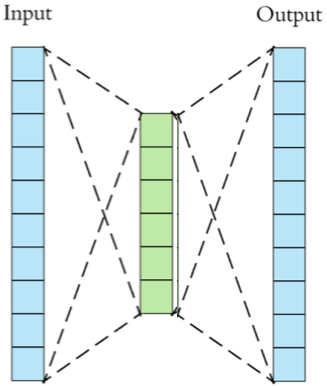
$$\hat{\rho} = \frac{1}{k}\sum_{i}^{k}(a_j^{(2)}(x^{(i)}))$$

- Assume only few hidden layer neurons should fire for each training sample
  - output $\boldsymbol{\rho}$ of a neuron should be small (i.e. 0.02)
  - Penalize $\hat{\boldsymbol{\rho}}$ deviating significantly from $\boldsymbol{\rho}$:

$$\sum_{j=1}^{m}\rho log\frac{\rho}{\hat{\rho}_j} + (1-\rho)log\frac{1-\rho}{1-\hat{\rho}_j} = \sum_{j=1}^{m}KL(\rho||\hat{\rho}_j)$$

  - KL divergence metric of distribution difference – sparsity term

Sparse Autoencoders (AE)



- The overall cost function:

$$J_{sparse}(W,b) = J(W,b) + \beta \sum_{j=1}^{m} KL(\rho||\hat{\rho}_j)$$

- Need to incorporate the KL divergence in the back-propagation:
- The back-prop at the second (hidden) layer: $\delta_i^{(2)} = (\sum_{j=1}^{m} W_{ji}^{(2)} \delta_j^{(3)}) f'(z_i^{(2)})$
- Take into account sparsity:

$$\delta_i^{(2)} = (\sum_{j=1}^{m} W_{ji}^{(2)} \delta_j^{(3)}) + \beta(-\frac{\rho}{\hat{\rho}_i} + \frac{1-\rho}{1-\hat{\rho}_i}) f'(z_i^{(2)})$$

- Images that enable hidden units are image fundamental features (i.e. basic shapes)

Auto-encoders for clustering [Hie te al 2016]

- Deep Embedding Clustering (DEC)
  - Assume data X of n points, cluster them into k. clusters in a lower dimensional space via a nonlinear mapping $f_\theta : X \to Z$, Z, θ: code, trainable parameters
  - learning simultaneously i. the center of the k clusters in the Z space and ιι. θ
  - *DEC phases:*
    - parameter initialization with a deep stacked autoencoder (Vincent 2010)
    - parameter optimization (i.e., clustering), where we iterate between computing an auxiliary target distribution and minimizing the Kullback–Leibler (KL) divergence to it

Autoencoders for clustering [Hie te al 2016]

- soft assignment of embedded point $z_i = f_\theta(x_i)$ and the cluster centroids μ_j

$$q_{ij} = \frac{(1 + \|z_i - \mu_j\|^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{j'}(1 + \|z_i - \mu_{j'}\|^2/\alpha)^{-\frac{\alpha+1}{2}}}$$

- iteratively refine clusters by learning from their high confidence assignments with the help of an auxiliary target distribution

$$p_{ij} = \frac{q_{ij}^2/f_j}{\sum_{j'} q_{ij'}^2/f_{j'}}$$

- Where $f_j = \sum_i q_{ij}$ are the soft cluster frequencies.

- Then the error is $L = \mathrm{KL}(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$

Autoencoders for clustering [Hie te al 2016]

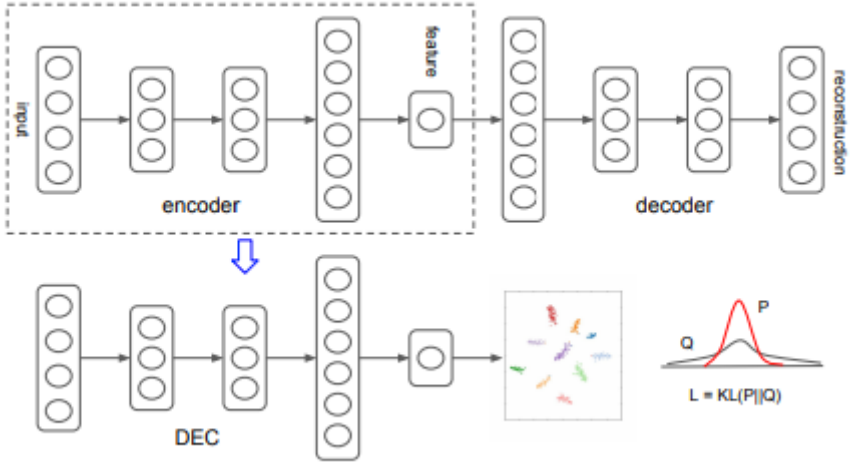
- jointly optimize cluster centers $\{\mu_j\}$ and DNN parameters θ using Stochastic Gradient Descent (SGD):

$$\frac{\partial L}{\partial z_i} = \frac{\alpha+1}{\alpha}\sum_j (1+\frac{\|z_i-\mu_j\|^2}{\alpha})^{-1} \times (p_{ij}-q_{ij})(z_i-\mu_j),$$

$$\frac{\partial L}{\partial \mu_j} = -\frac{\alpha+1}{\alpha}\sum_i (1+\frac{\|z_i-\mu_j\|^2}{\alpha})^{-1} \times (p_{ij}-q_{ij})(z_i-\mu_j).$$

- gradients $\partial L/\partial zi$ are passed down to the DNN
- standard backpropagation compute DNN's parameter gradient $\partial \mathrm{L}/\partial \theta$.
- discovering cluster assignments, stop when less than $tol\%$ of points change cluster assignment between consecutive iterations.

Autoencoders for clustering [Hie te al 2016]



- To initialize the cluster centers:
  - Pass data through the initialized DNN to get embedded data points and then perform standard k-means clustering in the feature space Z to obtain k initial centroids {μj} k j=1.

Autoencoders for clustering [Hie te al 2016]

Table 1. Dataset statistics.

Dataset	# Points	# classes	Dimension	% of largest class
MNIST (LeCun et al., 1998)	70000	10	784	11%
STL-10 (Coates et al., 2011)	13000	10	1428	10%
REUTERS-10K	10000	4	2000	43%
REUTERS (Lewis et al., 2004)	685071	4	2000	43%

Method	MNIST	STL-HOG	REUTERS-10k	REUTERS
k-means	53.49%	28.39%	52.42%	53.29%
LDMGI	84.09%	33.08%	43.84%	N/A
SEC	80.37%	30.75%	60.08%	N/A
DEC w/o backprop	79.82%	34.06%	70.05%	69.62%
DEC (ours)	**84.30%**	**35.90%**	**72.17%**	**75.63%**

Autoencoders for clustering [Hie te al 2016]

(a) MNIST

(b) STL-10

- Top 10 results for each of the 10 clusers for the MNIST and STL data sets

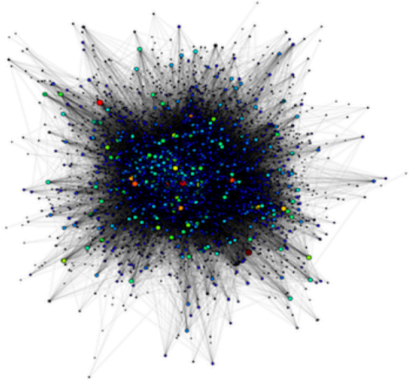
Why is Unsupervised learning important

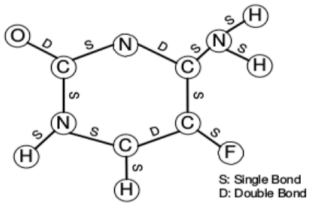
- Machine learning: learning representations (features, latent variables) to capture the statistical dependencies
- Supervised learning – from input variables to output variables, – i.e. categories
  - supervised learning require "teaching" computer concepts that matter to **humans**
  - supervised deep learning discover meaningful intermediate representations in their layers.
- Unsupervised learning
  - capture all possible dependencies between any subset of variables
  - human learning: based on observation and unsupervised interaction (action-perception loop)
- Issues
  - Training sets increasingly difficult to keep the pace with data production
  - Can we trust humans ?

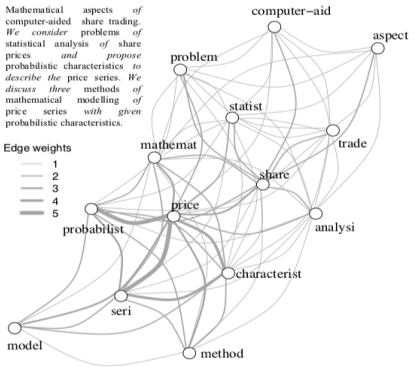
"…hope is that deep unsupervised learning will be able to discover (possibly with a little bit of help from the few labeled examples we can provide) all of the concepts and underlying causes that matter (some being explicitly labeled, some remaining unnamed) to explain what we see around us. So I believe it is essential for approaching AI to make progress in this direction. And we are ;-)" Y. Bengio

- Recurrent NNs + LSTMs
- Autoencoders
- **Graph node embeddings**

Graph structures in abundance



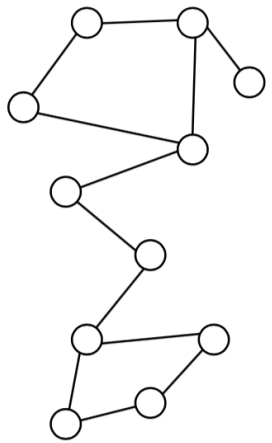




A flexible and powerful data structure

Learning Node embeddings

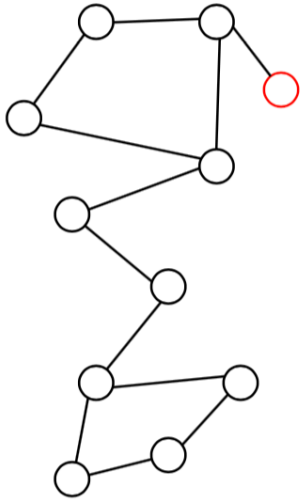
Representation: row of adjacency matrix



$$\rightarrow \quad \begin{pmatrix} 0 & 1 & \dots & 0 \\ 1 & 0 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & \dots & 0 \end{pmatrix}$$

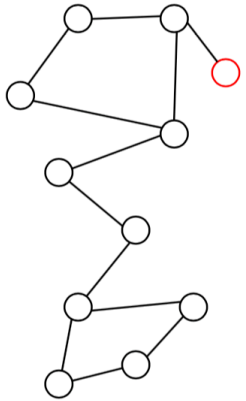
Learning Node embeddings

Representation: row of adjacency matrix



$$\rightarrow \quad \begin{pmatrix} 0 & 1 & \dots & 0 \\ 1 & 0 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & \dots & 0 \end{pmatrix}$$

Representation: row of adjacency matrix



$$\rightarrow \quad \begin{pmatrix} 0 & 1 & \dots & 0 \\ 1 & 0 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & \dots & 0 \end{pmatrix}$$

However, such a representation suffers from:

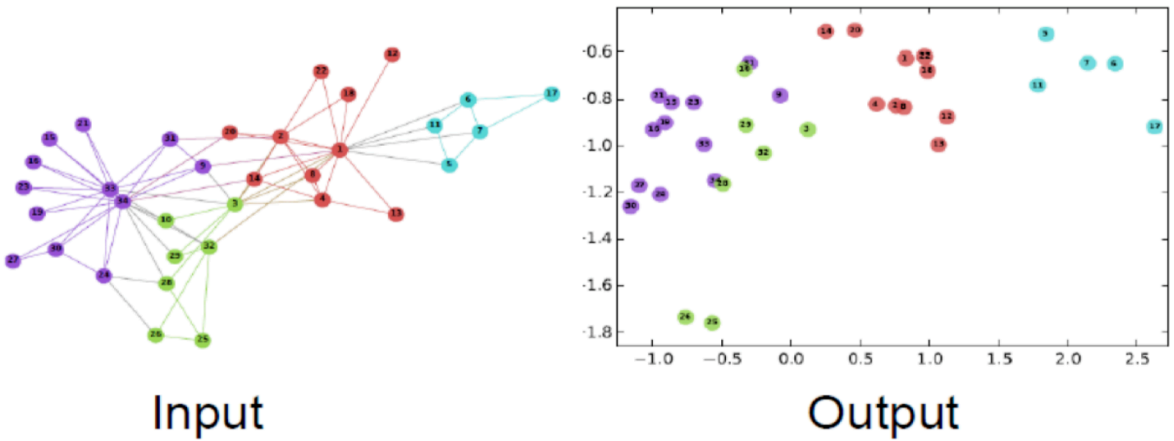
- data sparsity
- high dimensionality

$$\vdots$$

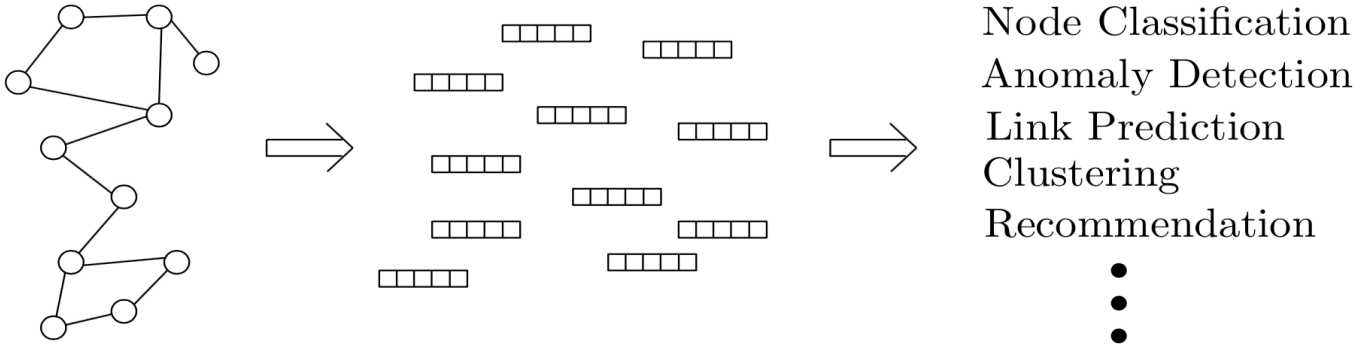
Node Embedding methods

Map vertices of a graph into a low-dimensional space:

- dimensionality $d \ll |V|$
- similar vertices are embedded close to each other in the low-dimensional space



Why learning node representations?



Examples:

- Recommend friends
- Detect malicious users

Early methods

- Focused mainly on matrix-factorization approaches (e. g., Laplacian eigenmaps)
- Laplacian eigenmaps projects two nodes i and j close to each other when the weight of the edge between the two nodes A_{ij} is high
- Embeddings are obtained by the following objective function:

$$y^* = \arg\min \sum_{i \neq j} (y_i - y_j)^2 A_{ij} = \arg\min y^T L y$$

  where L is the graph Laplacian

- The solution is obtained by taking the eigenvectors corresponding to the d smallest eigenvalues of the normalized Laplacian matrix

[1] Belkin and Niyogi. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In NIPS'02

Recent Methods

Most methods belong to the following groups:

1. Random walk based methods: employ random walks to capture structural relationships between nodes
2. Edge modeling methods: directly learn node embeddings using structural information from the graph
3. Matrix factorization methods: generate a matrix that represents the relationships between vertices and use matrix factorization to obtain embeddings
4. Deep learning methods: apply deep learning techniques to learn highly non-linear node representations

Node Embedding Methods

Map vertices of a graph into a low-dimensional space:

- dimensionality $d \ll |V|$
- **similar vertices** are embedded close to each other in the low-dimensional space

When two vertices are **similar** to each other?

$->$ first-order proxmity

$->$ second-order proxmity

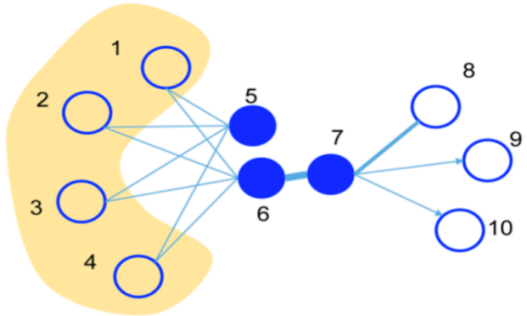
$->$ third-order proxmity

$\vdots$

Node order proximity

First-order proximity: observed links in the network

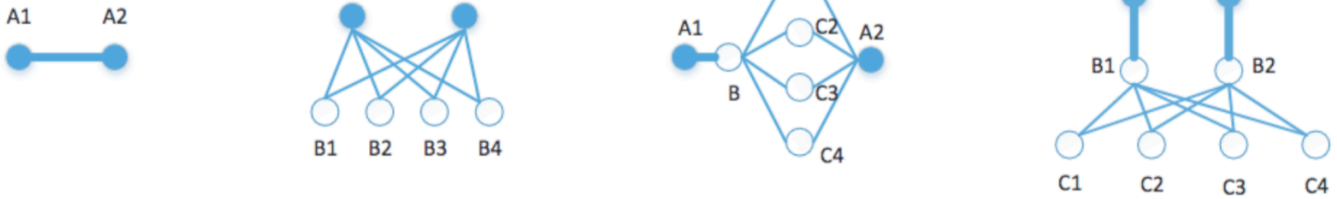
Second-order proximity: shared neighborhood structures



- Vertices 6 and 7 have a high *first-order proximity* since they are connected through a strong tie $\rightarrow$ they should be placed closely in the embedding space
- Vertices 5 and 6 have a high *second-order proximity* since they share similar neighbors $\rightarrow$ they should also be placed closely

Node order proximity

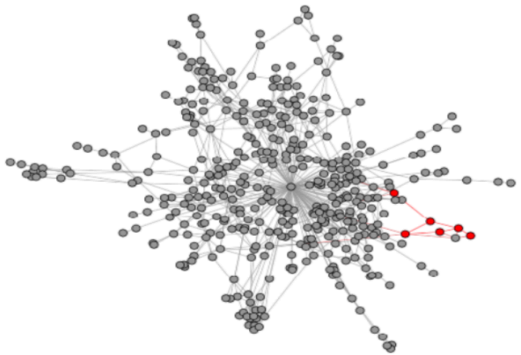
k-order proximities for $k = 1, \ldots, 4$



- Second-order and high-order proximities capture similarity between vertices with similar structural roles
- Higher-order proximities capture more global structure

Deep walk

Inspired by recent advances in language modeling [1]



$\rightarrow$

$v_5 \rightarrow v_8 \rightarrow v_{32} \rightarrow v_{28} \rightarrow v_6 \rightarrow v_{10} \rightarrow v_9$

$v_3 \rightarrow v_5 \rightarrow v_{28} \rightarrow v_8 \rightarrow v_9 \rightarrow v_{10} \rightarrow v_{25}$

$v_{20} \rightarrow v_{10} \rightarrow v_{12} \rightarrow v_6 \rightarrow v_8 \rightarrow v_4 \rightarrow v_5$

$v_{23} \rightarrow v_5 \rightarrow v_{32} \rightarrow v_{10} \rightarrow v_8 \rightarrow v_3 \rightarrow v_1$

$v_4 \rightarrow v_3 \rightarrow v_1 \rightarrow v_5 \rightarrow v_1 \rightarrow v_{12} \rightarrow v_{10}$

$\vdots$

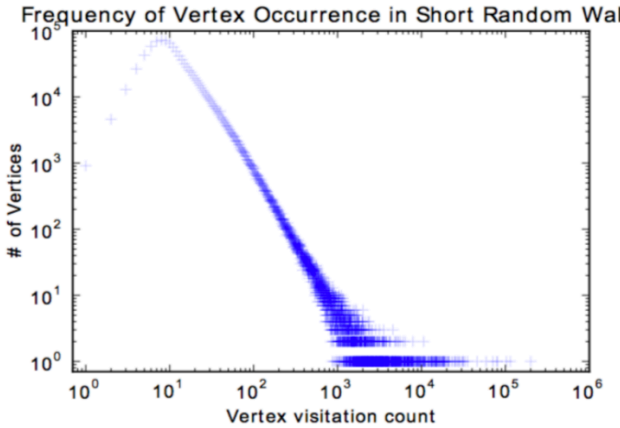
- Simulates a series of short random walks

[1] Mikolov et al. Distributed Representations of Words and Phrases and their Compositionality. In NIPS'13

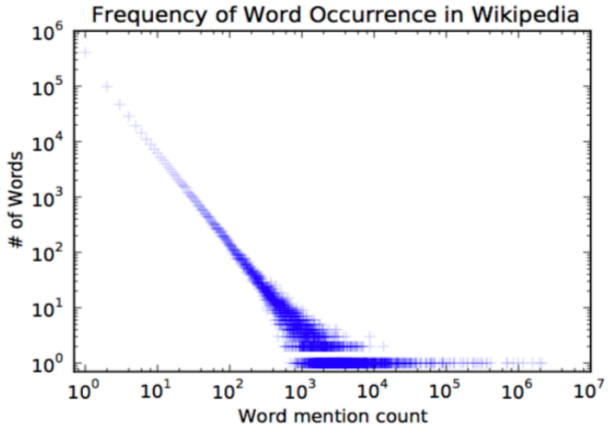
[2] Perozzi et al. DeepWalk: Online Learning of Social Representations. In KDD'14

Deepwalk

Inspired by recent advances in language modeling [1]



(a) YouTube Social Graph



(b) Wikipedia Article Text

- Simulates a series of short random walks
- **Main Idea**: Short random walks = Sentences

[1] Mikolov et al. Distributed Representations of Words and Phrases and their Compositionality. In NIPS'13

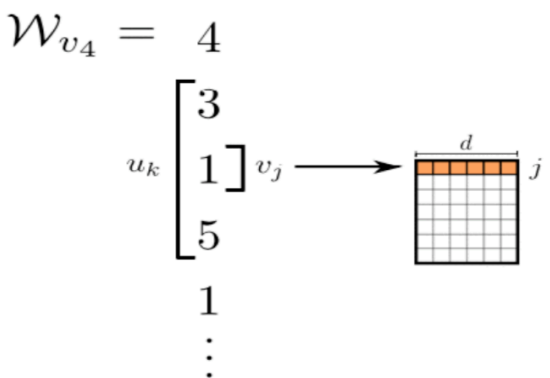
[2] Perozzi et al. DeepWalk: Online Learning of Social Representations. In KDD'14

Skipgram

Skipgram is a recently-proposed language model that:

- uses one word to predict the context
- context is composed of words appearing to both the right and left of the given word
- removes the ordering constraint on the problem (i. e. does not take into account the offset of context words from the given word)

In our setting:



- Slide a window of length $2w + 1$ over the random walk
- Use the representation of central vertex to predict its neighbors

Skipgram

This yields the optimization problem:

$$\underset{f}{\text{minimize}} \quad -\frac{1}{T}\sum_{i=1}^{T} \log P(\{v_{i-w}, \ldots, v_{i+w}\} \setminus v_i | f(v_i))$$

v_i: central vertex
$v_{i-w}, \ldots, v_{i+w}$: neighbors of central vertex
$f(v)$: embedding of vertex v

Skipgram approximates the above conditional probability using the following independence assumption:

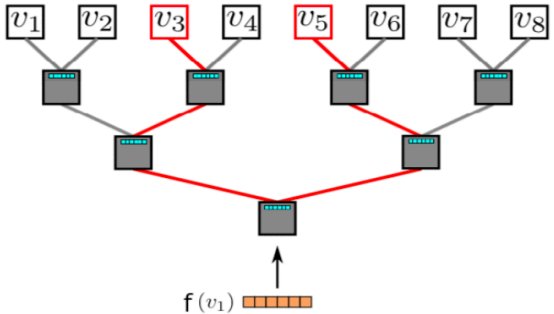
$$\underset{f}{\text{minimize}} \quad -\frac{1}{T}\sum_{i=1}^{T} \sum_{\substack{j=i-w \\ j\neq i}}^{i+w} \log P(v_j | f(v_i))$$

- We can learn such a posterior distribution using several choices of classifiers
- **However**, most of them (e. g., logistic regression) would produce a huge number of labels (i. e. $|V|$ labels)

Hierarchical softmax

Reduces complexity from $\mathcal{O}(|V|)$ to $\mathcal{O}(\log|V|)$ using a binary tree

- Assigns the vertices to the leaves of a binary tree
- New problem: Maximizing the probability of a specific path in the hierarchy



If the path to vertex v_j is identified by a sequence of tree nodes $(b_0, b_1, \ldots, b_{\lceil \log|V| \rceil})$ then

$$P(v_j|f(v_i)) = \prod_{l=1}^{\lceil \log|V| \rceil} P(b_l|f(v_i))$$

where

$$P(b_l|f(v_i)) = 1/(1+e^{-f(v_i)^\top f'(b_l)}) = \sigma(f(v_i)^\top f'(b_l))$$

and $f'(b_l) \in \mathbb{R}^d$ is the representation assigned to tree node b_l's parent

Node2vec

Like DeepWalk, node2vec is also a random walk based method

DeepWalk uses a *rigid* search strategy

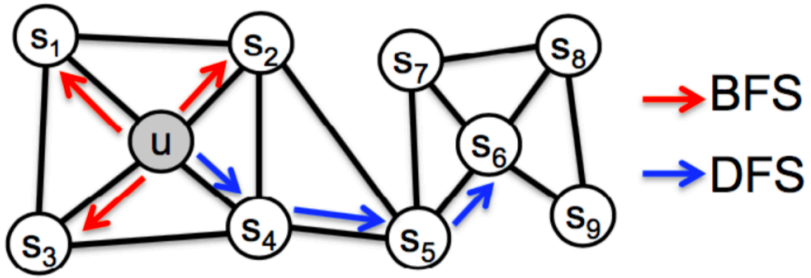
Conversely, node2vec simulates a family of biased random walks which

- explore diverse neighborhoods of a given vertex
- allow it to learn representations that organize vertices based on
  - their network roles
  - the communities they belong to

[1] Grover and Leskovec. node2vec: Scalable Feature Learning for Networks. In KDD'16

Node2vec – extreme sampling strategies

The *breadth-first sampling* (BFS) and *depth-first sampling* (DFS) represent extreme scenarios in terms of the search space

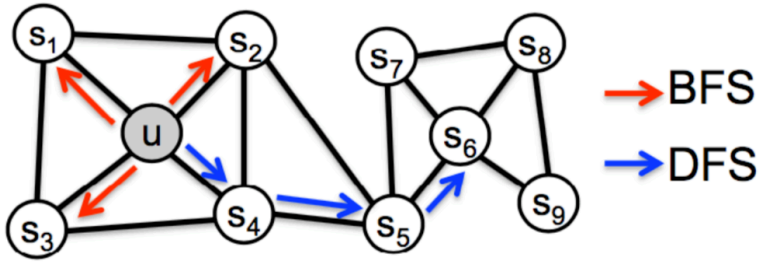


In most applications, we are interested in two kinds of similarities between vertices:

1. homophily: nodes that are highly interconnected and belong to similar communities should be embedded closely together (e. g., s_1 and u)
2. structural equivalence: nodes that have similar structural roles should be embedded closely together (e. g., u and s_6)

Node2vec – extreme sampling strategies

The *breadth-first sampling* (BFS) and *depth-first sampling* (DFS) represent extreme scenarios in terms of the search space



BFS and DFS strategies play a key role in producing representations that reflect these two properties:

- The neighborhoods sampled by BFS lead to embeddings that correspond closely to structural equivalence
- The neighborhoods sampled by DFS reflect a macro-view of the neighborhood which is essential in inferring communities based on homophily

Node2vec – random walks

Given a source node, node2vec simulates a random walk of fixed length l

$$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \ldots \rightarrow v_l$$

The i^{th} node in the walk is generated as follows:

$$P(c_i = x | c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z}, & \text{if } (v, x) \in E \\ 0, & \text{otherwise} \end{cases}$$

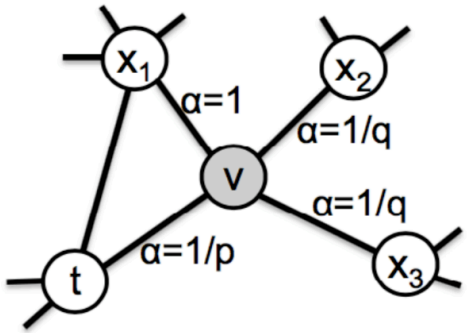
where π_{vx} is the unnormalized transition probability between v and x, and Z is a normalizing factor

To capture both structural equivalence and homophily, node2vec uses a neighborhood sampling strategy which

- is based on a flexible biased random walk procedure
- allows it to smoothly interpolate between BFS and DFS

Node2vec – random walks

The random walk shown below just traversed edge (t, v) and now resides at node v



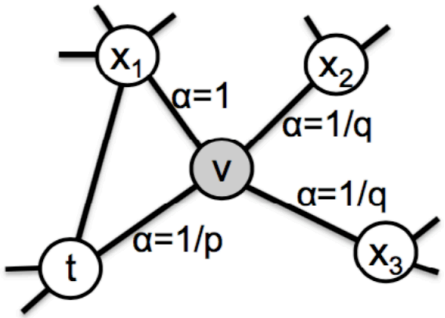
The unnormalized transition probability is $\pi_{vx} = w_{vx}\alpha_{pq}(t, x)$, where:

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

where d_{tx} denotes the shortest path distance between t and x

Node2vec – random walks

The random walk shown below just traversed edge (t, v) and now resides at node v

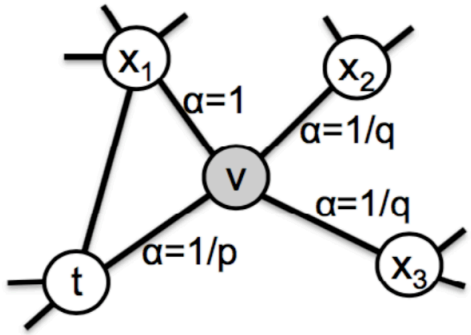


The *return parameter* p controls the likelihood of immediately revisiting a node in the walk

- if p is high, we are less likely to sample an already-visited node in the following two steps
- if p is low, it would keep the walk in the local neighborhood of the starting node

Node2vec – random walks

The random walk shown below just traversed edge (t, v) and now resides at node v



The *in-out parameter* q allows the search to differentiate between "inward" and "outward" nodes.

- if q is high, the random walk is biased towards nodes close to node t
- if q is low, the walk is more inclined to visit nodes which are further away from the node t

Node2vec - Optimization

After defining the neighborhood $\mathcal{N}(v) \subset V$ of each node v, node2vec uses the Skipgram architecture:

$$\underset{f}{\text{minimize}} \quad -\sum_{v \in V} \log \prod_{u \in \mathcal{N}(v)} P(u|f(v))$$

where conditional likelihood is modelled as a softmax unit parametrized by a dot product of their features:

$$P(u|f(v)) = \frac{e^{f'(u)^\top f(v)}}{\sum_{k=1}^{|V|} e^{f'(v_k)^\top f(v)}}$$

and $f'(u) \in \mathbb{R}^d$ is the representation of node u when considered as context

The objective function thus becomes:

$$\underset{f,f'}{\text{minimize}} \quad -\sum_{v \in V} \Big(-\log \sum_{u \in V} e^{f'(u)^\top f(v)} + \sum_{u \in \mathcal{N}(v)} f'(u)^\top f(v) \Big)$$

Since learning the above posterior distribution is very expensive, node2vec approximates it using negative sampling

LINE

LINE employs an objective function that explicitly uses structural information from the graph to learn node representations

Specifically, LINE

- preserves both the first-order and second-order proximities
- trains two models separately
- concatenates the two learned embeddings for each vertex

[1] Tang et al. LINE: Large-scale Information Network Embedding. In WWW'15

LINE with first order proximity

To model the first-order proximity, for each undirected edge (v_i, v_j), define the joint probability between v_i and v_j as follows:

$$P_1(v_i, v_j) = \frac{1}{1 + e^{-f(v_i)^\top f(v_j)}}$$

where $f(v_i) \in \mathbb{R}^d$ is the low-dimensional vector representation of vertex v_i

The empirical probability can be defined as:

$$\hat{P}_1(v_i, v_j) = \frac{w_{ij}}{W}$$

w_{ij}: weight of the edge between v_i, v_j
W: sum of weights of all edges

LINE minimizes the KL-divergence of the two probability distributions:

$$\underset{f}{\text{minimize}} \quad - \sum_{(v_i, v_j) \in E} w_{ij} \log P_1(v_i, v_j)$$

LINE with second order proximity

To model the second-order proximity, for each edge (v_i, v_j), LINE defines the probability of context v_j generated by vertex v_i:

$$P_2(v_j|v_i) = \frac{e^{f'(v_j)^\top f(v_i)}}{\sum_{k=1}^{|V|} e^{f'(v_k)^\top f(v_i)}}$$

$f(v_i)$: representation of v_i when treated as a vertex
$f'(v_i)$: representation of v_i when treated as context

The empirical probability can be defined as:

$$\hat{P}_2(v_j|v_i) = \frac{w_{ij}}{d_i}$$

d_i: out-degree of v_i

LINE minimizes the KL-divergence of the two probability distributions:

$$\underset{f,f'}{\text{minimize}} \quad -\sum_{(v_i,v_j)\in E} w_{ij} \log P_2(v_j|v_i)$$

LINE with second order proximity

Optimizing the objective of the second-order proximity is computationally very *expensive*

Instead, use negative sampling: for each edge, sample multiple negative edges according to some noisy distribution

Every $\log P_2(v_j|v_i)$ term in the objective is replaced with:

$$\log \sigma(f'(v_j)^\top f(v_i)) + \sum_{k=1}^{K} \mathbb{E}_{v_k \sim P_n(v)}[\log \sigma(-f'(v_k)^\top f(v_i))]$$

where $\sigma = 1/(1 + e^{-x})$ is the sigmoid function and K the number of negative edges

Experimental Evaluation

Experimental comparison conducted in [1]

Compared algorithms:

- DeepWalk
- GraRep
- SDNE
- LINE
- Laplacian Eigenmaps (LE)

[1] Wang et al. Structural Deep Network Embedding. In KDD'16

Datasets

Five datasets:

- three social networks
- one citation network
- one language network

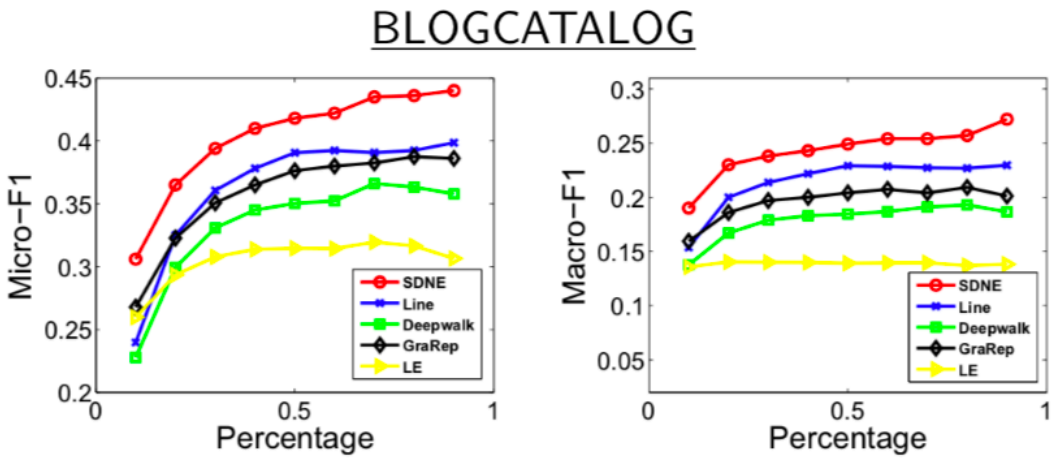
Dataset	#(V)	#(E)
BLOGCATALOG	10312	667966
FLICKR	80513	11799764
YOUTUBE	1138499	5980886
ARXIV GR-QC	5242	28980
20-NEWSGROUP	1720	Full-connected

Three real-world applications

- node classification
- link prediction
- visualization

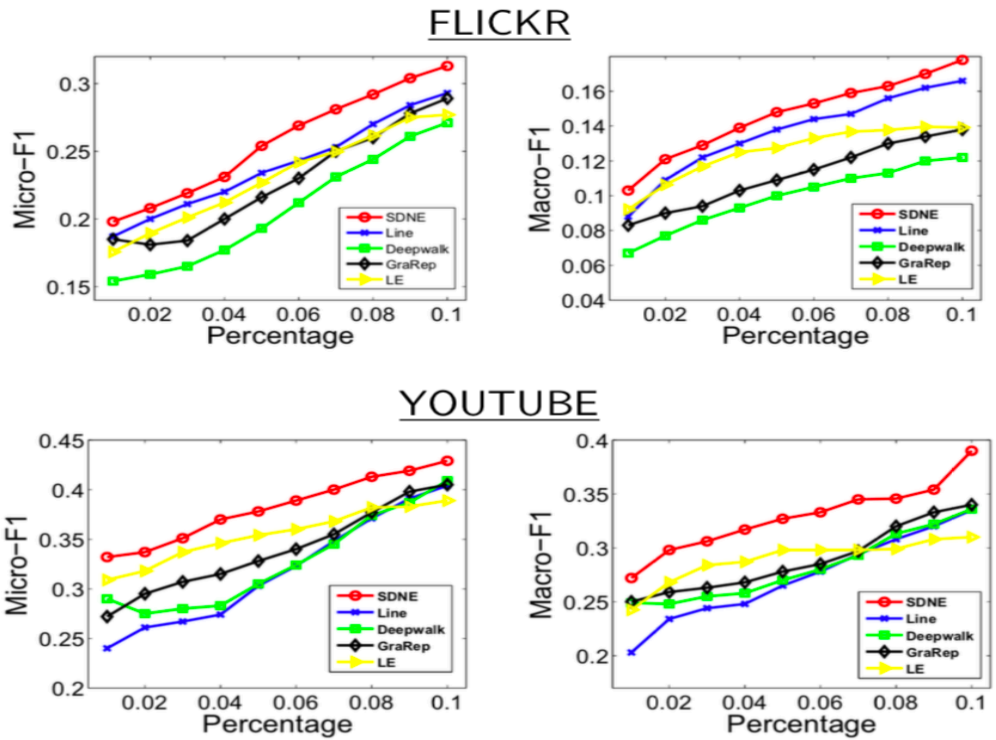
Node Classification

Vertex representations generated from node embedding methods and given as input to a logistic regression classifier to predict a set of labels for each vertex



For BLOGCATALOG, the training/test ratio is increased from 10% to 90%

Node Classification

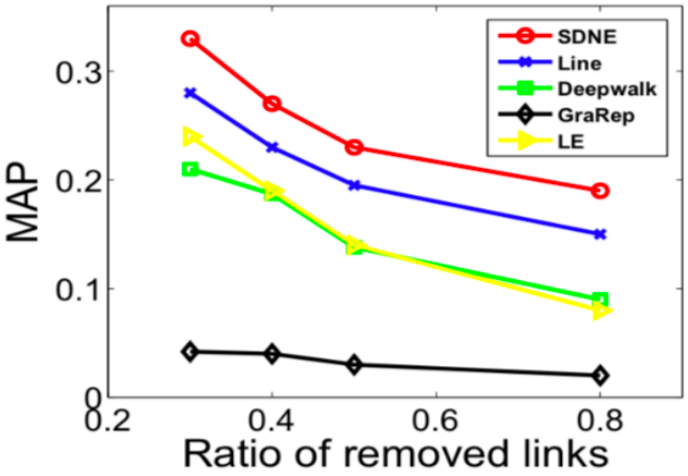


For FLICKR and YOUTUBE, the training/test ratio is increased from 1% to 10%

Link Prediction

Followed procedure:

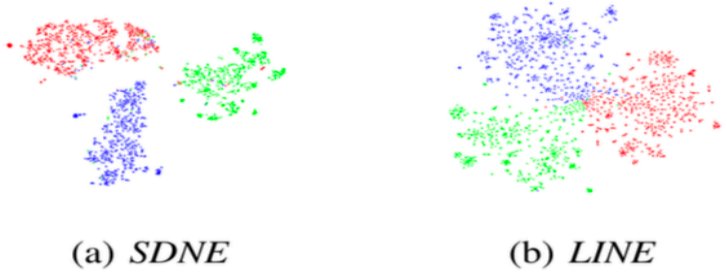
- Remove a portion of ARXIV GR-QC's edges
- Use the emerging network to learn node embeddings
- Predict missing links



Visualization

Visualization of 20-NEWSGROUP

- Each point indicates one document
- Color of a point indicates the category of the document



(a) *SDNE* (b) *LINE*

(c) *DeepWalk* (d) *GraRep*

References - Autoencoders

- [Baldi, 2012] Baldi, P. (2012). Autoencoders, unsupervised learning, and deep architectures. In Proceedings of ICML workshop on unsupervised and transfer learning, pages 37–49.
- [Rum1986] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In Parallel Distributed Processing. Vol 1: Foundations. MIT Press, Cambridge, MA, 1986.
- [Bengio and LeCun, 2007] Bengio, Y. and LeCun, Y. (2007). Scaling learning algorithms towards ai. Large-scale kernel machines, 34(5) :1–41.
- [Dertat, 2017] Dertat, A. (2017). Applied deep learning - part 3 : Autoencoders, medium post, towards data science.
- [Hinton and Salakhutdinov, 2006] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. science, 313(5786) :504–507.
- [Hou et al., 2017] Hou, X., Shen, L., Sun, K., and Qiu, G. (2017). Deep feature consistent variational autoencoder. In Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on, pages 1133–1141. IEEE.
- [Hie te al 2016] Unsupervised Deep Embedding for Clustering Analysis, J. Xie, R. Girschick, A. Farhadi, ICML 2016

References - Autoencoders

- [Vincent et al., 2008] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on Machine learning, pages 1096–1103. ACM.
- [Vincent et al., 2010] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders : Learning useful representations in a deep network with a local denoising criterion. Journal of machine learning research, 11(Dec) :3371–3408.
- Restricted Boltzman Machines, https://www.cs.toronto.edu/~rsalakhu/papers/rbmcf.pdf

References - Graph classification

[1] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009). The graph neural network model. IEEE Transactions on Neural Networks, 20(1), 61-80.
[2] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural Message Passing for Quantum Chemistry. In International Conference on Machine Learning (pp. 1263-1272).
[3] Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., & Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. In Advances in neural information processing systems (pp. 2224-2232).
[4] Li, Y., Tarlow, D., Brockschmidt, M., & Zemel, R. (2015). Gated graph sequence neural networks. arXiv preprint arXiv:1511.05493.
[5] Zhang, M., Cui, Z., Neumann, M., & Chen, Y. (2018). An end-to-end deep learning architecture for graph classification. In Proceed
[6] Atwood, J., & Towsley, D. (2016). Diffusion-convolutional neural networks. In Advances in Neural Information Processing Systems (pp. 1993-2001).
[7] Lei, T., Jin, W., Barzilay, R., & Jaakkola, T. (2017). Deriving neural architectures from sequence and graph kernels. arXiv preprint arXiv:1705.09037.
[8] Bruna, J., Zaremba, W., Szlam, A., & LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203.
[9] Simonovsky, M., & Komodakis, N. (2017). Dynamic edge-conditioned filters in convolutional neural networks on graphs. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 29-38). IEEE.
[10] Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W.,& Leskovec, J. (2018). Hierarchical graph representation learning with differentiable pooling. In Advances in Neural Information Processing Systems (pp. 4805-4815).
[11] Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., & Grohe, M. (2018). Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks. arXiv preprint arXiv:1810.02244.
[12] Niepert, M., Ahmed, M., & Kutzkov, K. (2016). Learning convolutional neural networks for graphs. In International conference on machine learning (pp. 2014-2023).
[13] Nikolentzos, G., Meladianos, P., Tixier, A. J. P., Skianis, K., & Vazirgiannis, M. (2018). Kernel graph convolutional neural networks. In International Conference on Artificial Neural Networks (pp. 22-32). Springer, Cham.
[14] Tixier, A. J. P., Nikolentzos, G., Meladianos, P., & Vazirgiannis, M. (2017). Classifying graphs as images with convolutional neural networks. arXiv preprint arXiv:1708.02218.