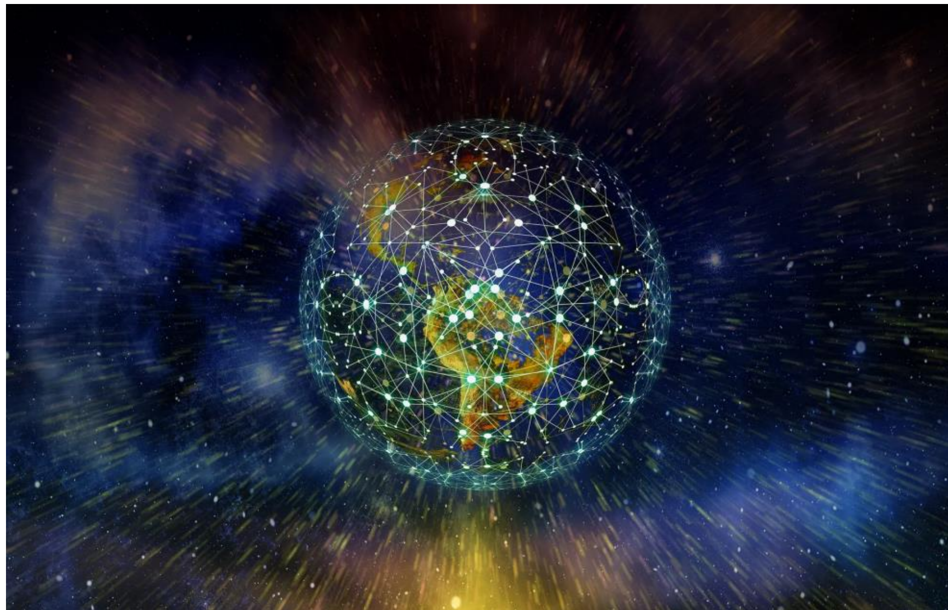
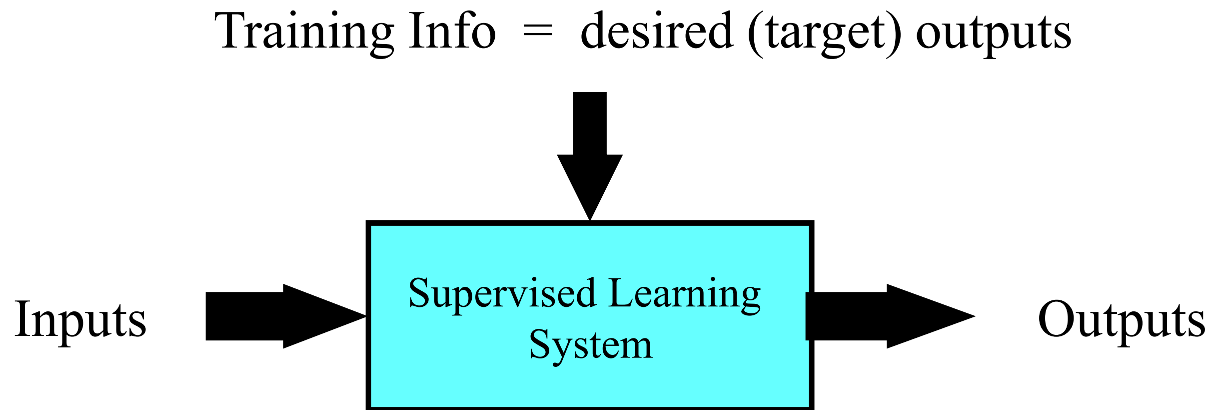
University of Ljubljana, Faculty of Computer and Information Science

Machine learning for combinatorial optimization



Prof Dr Marko Robnik-Šikonja
Analysis of Algorithms and Heuristic Problem Solving
Version 2023

Supervised Learning



Error = (target output – actual output)

Basic notation of predictive modelling

- We have data and the variable of interest
- The statistical variable of interest is called response or target or prediction variable that we wish to predict. We usually refer to the response as Y .
- Other variables are called attributes, features, inputs, or predictors; we name them X_i.
- The input vectors forms a matrix **X**

$$X = \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix}$$

- The model we write as

$$Y = f(X) + \epsilon$$

where ∈ is independent from X, has zero mean and represents measurement errors and other discrepancies.

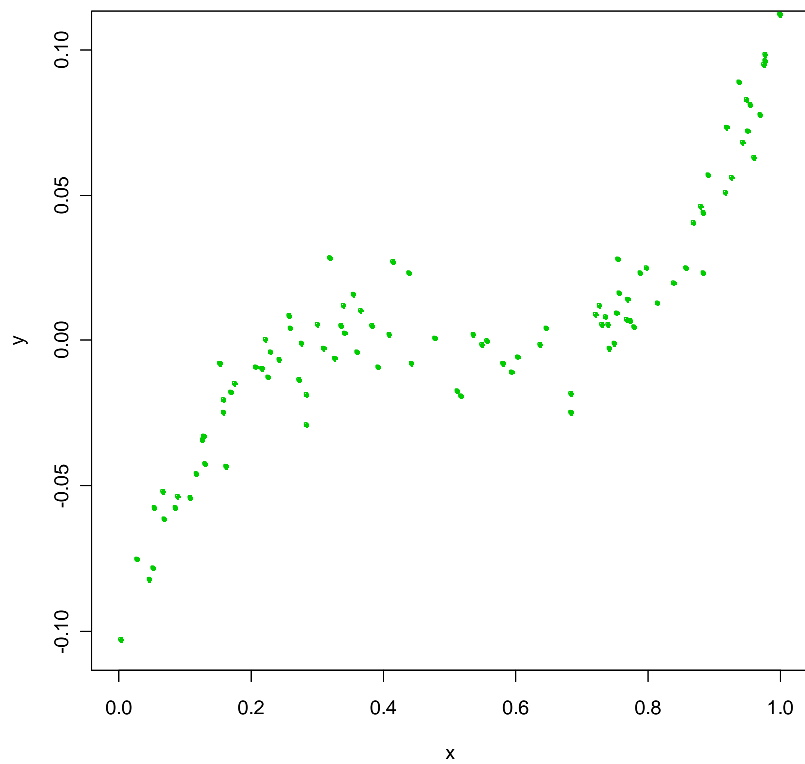
Further notation for instances

- Suppose we observe Y_i and $X_i = (X_{i1}, \ldots, X_{ip})$ for $i = 1, \ldots, n$
- We believe that there is a relationship between Y and at least one of the X's.
- We can model the relationship as

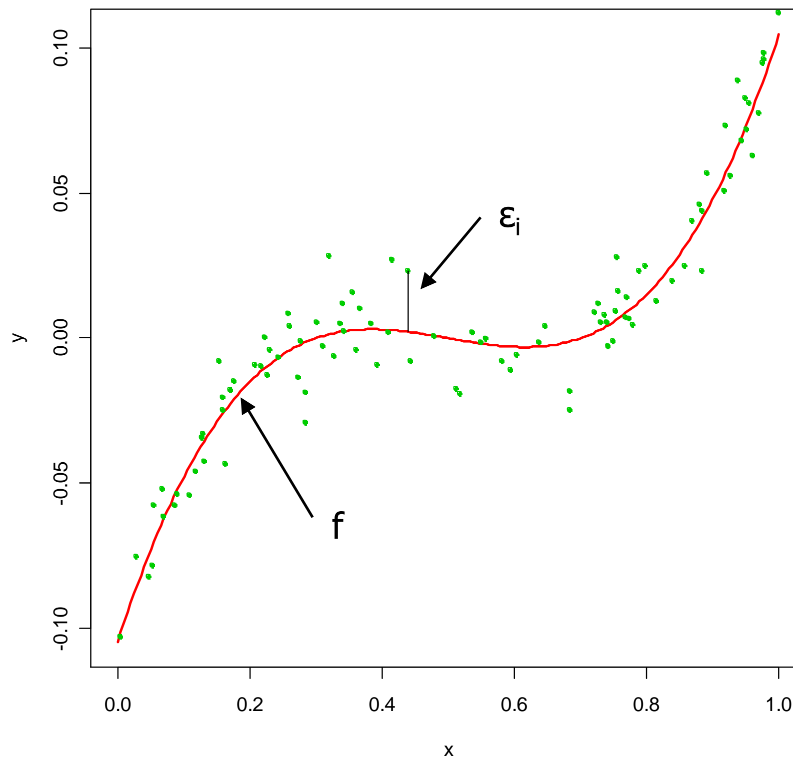
$$Y_i = f(\mathbf{X}_{i.}) + \varepsilon_i$$

- Where f is an unknown function and ε is a random error with mean zero.

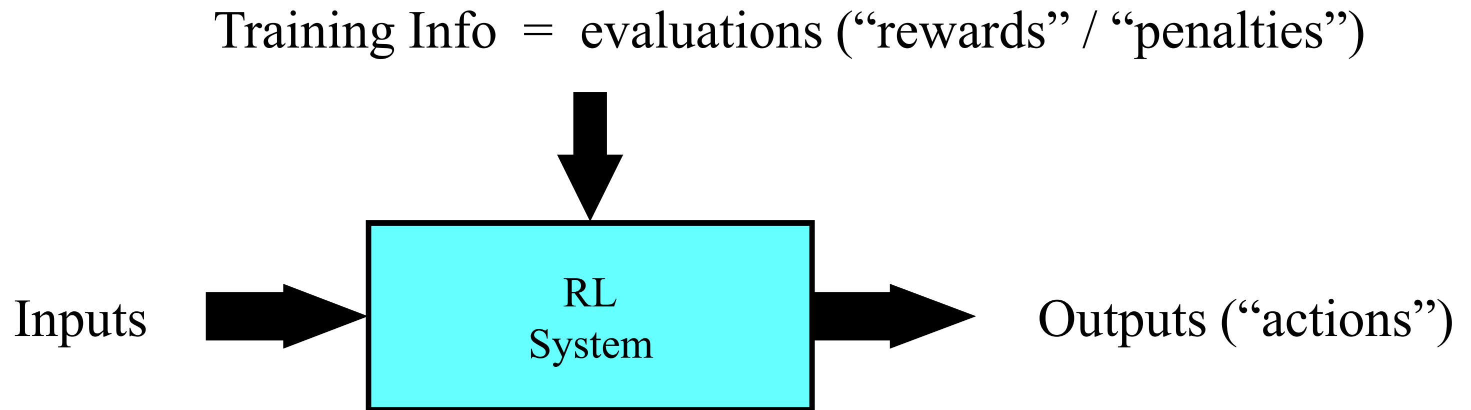
A simple example



A simple example

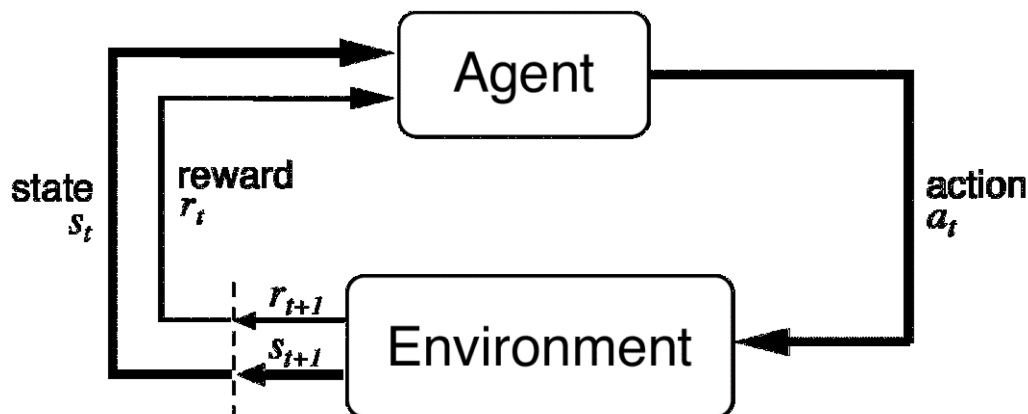


Reinforcement Learning



Objective: get as much reward as possible

The RL Agent-Environment Interface



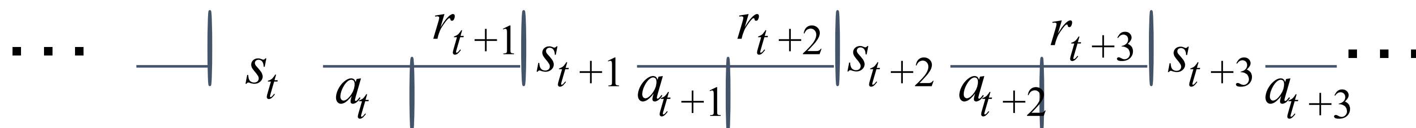
Agent and environment interact at discrete time steps : $t = 0, 1, 2, \ldots$

Agent observes state at step t: $s_t \in S$

produces action at step t: $a_t \in A(s_t)$

gets resulting reward : $r_{t+1} \in \Re$

and resulting next state : s_{t+1}



The Agent Learns a Policy

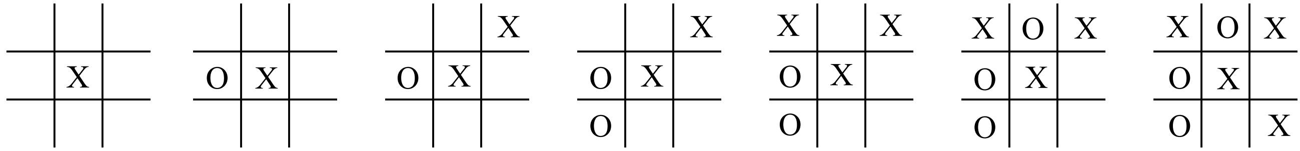
Policy at step t, π_t :

a mapping from states to action probabilities

$\pi_t(s, a) =$ probability that $a_t = a$ when $s_t = s$

- Reinforcement learning methods specify how the agent changes its policy as a result of experience.
- Roughly, the agent's goal is to get as much reward as it can over the long run.

An Example: Tic-Tac-Toe



} x’s move

} o’s move

} x’s move

} o’s move

} x’s move

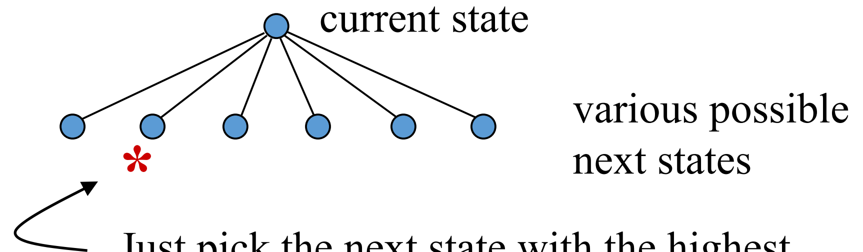
Assume an imperfect opponent:
—he/she sometimes makes mistakes

An RL Approach to Tic-Tac-Toe

1. Make a table with one entry per state:

State	$V(s)$ – estimated probability of winning	
	.5	?
	.5	?
⋮	⋮	
	1	win
⋮	⋮	
	0	loss
⋮	⋮	
	0	draw

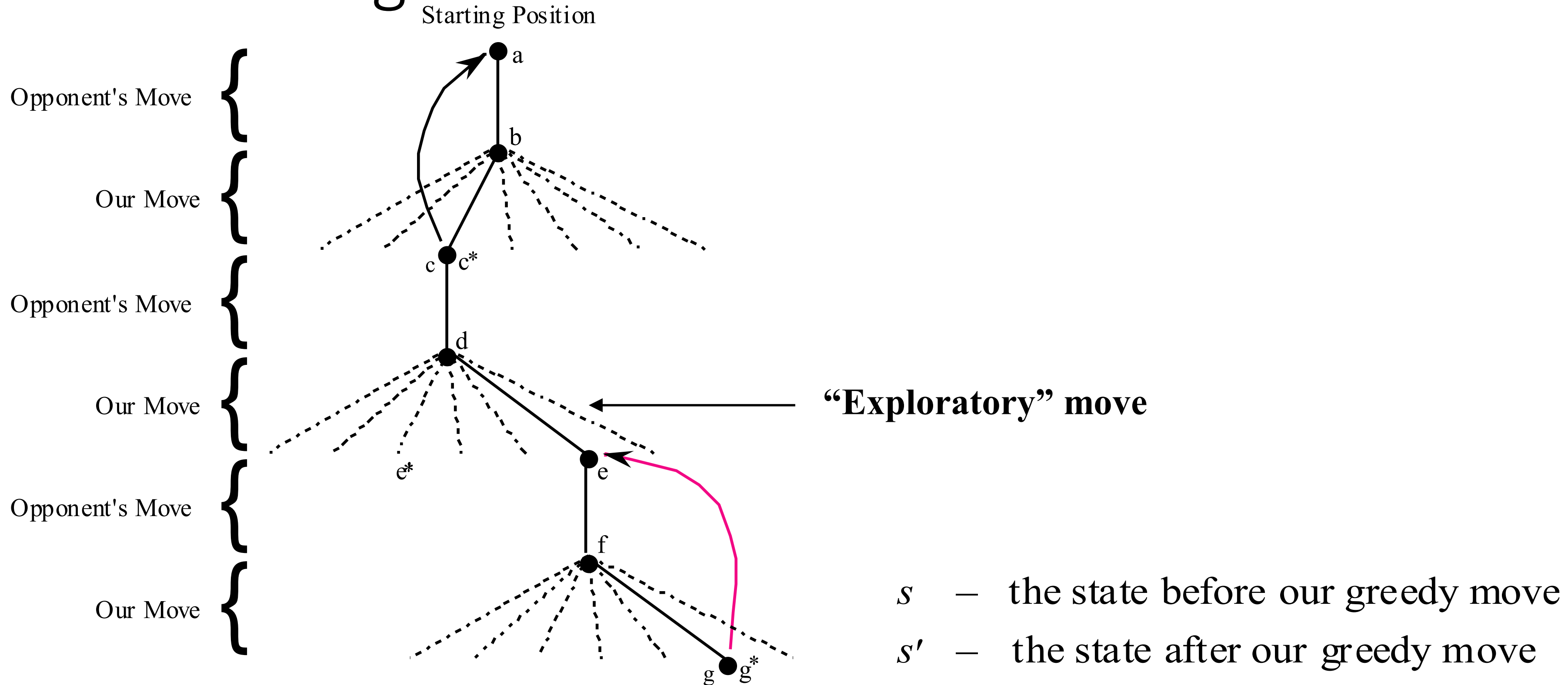
2. Now play lots of games.
To pick our moves,
look ahead one step:



Just pick the next state with the highest estimated prob. of winning — the largest $V(s)$; a ***greedy*** move.

But 10% of the time pick a move at random; an ***exploratory move***.

RL Learning Rule for Tic-Tac-Toe



We increment each $V(s)$ toward $V(s')$ – a ***backup***:

$$V(s) \leftarrow V(s) + \alpha\left[V(s') - V(s)\right]$$

a small positive fraction, e.g., $\alpha = .1$

the ***step-size parameter***

Standard approach of ML for CO

- Collect data from different runs
- Predict the quality of next move or predict the quality of the final solution

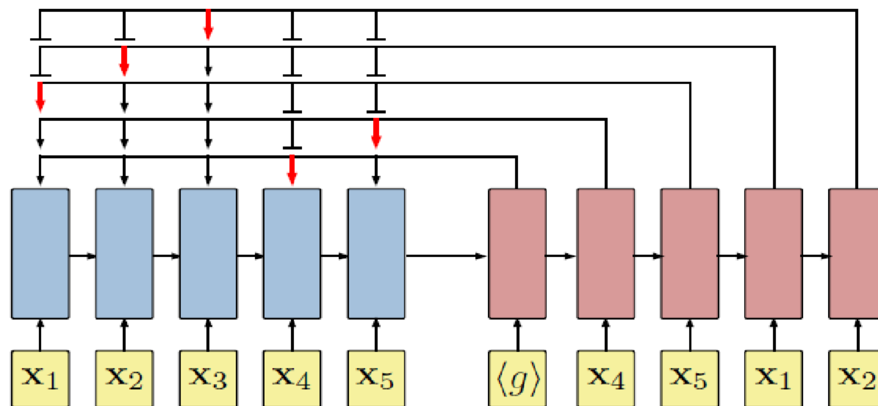
TSP with ML

- We focus on a 2D Euclidean TSP. And let the input be the sequence of cities (points) $s = \{x_i\}_{i=1}^{n}$, where each $x_i \in \mathbb{R}^2$.
- The target is to find a permutation π of these points, terms as a tour, that visits each city and has minimum length.
- Define the length of a tour π as:

$$L(\pi|s) = \left\|x_{\pi(n)} - x_{\pi(1)}\right\|_2 + \sum_{i=1}^{n-1} \left\|x_{\pi(i+1)} - x_{\pi(i)}\right\|_2$$

TSP with the recurrent network

Pointer network



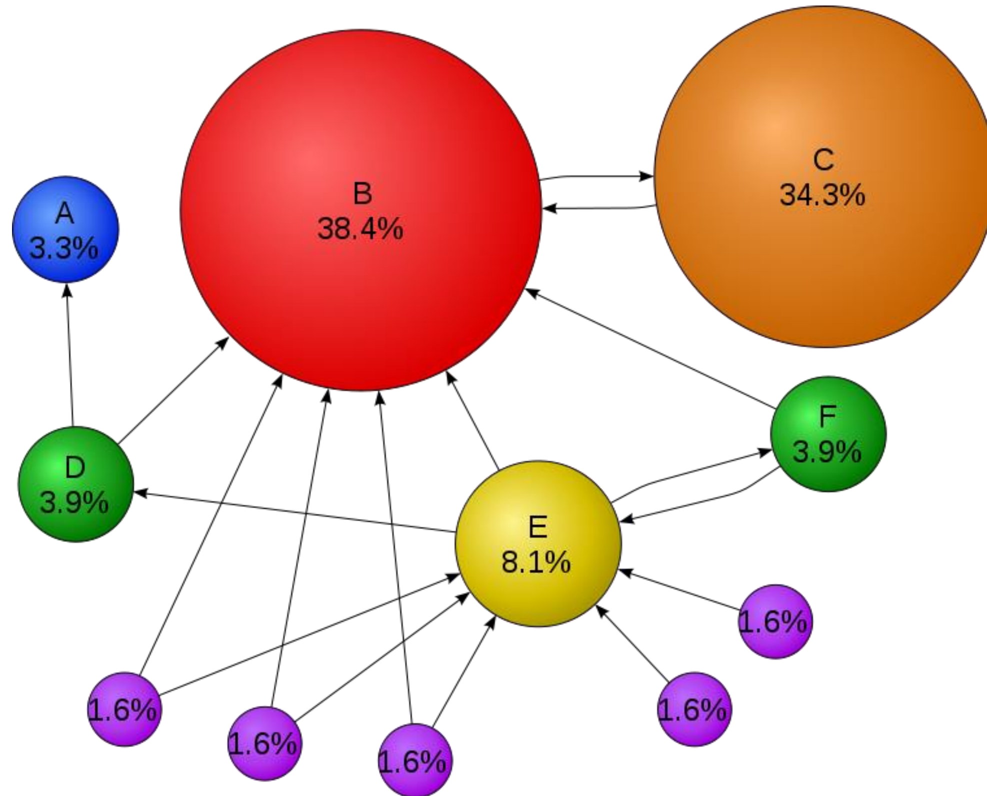
Encoder: reads the input sequence s, one city at a time, and transforms it into a sequence of latent memory states $\{enc_i\}_{i=1}^{n}$, and each $enc_i \in \mathbb{R}^d$

Decoder: uses a pointing mechanism to produce a distribution over the next city to visit in the tour.

Representing graphs with embeddings

- Embeddings are more practical than the adjacency matrix since they pack node properties in a vector with a smaller dimension.
- **Embeddings shall describe the properties of the graphs**. They need to represent the graph topology, node connections, and node neighborhood. The performance of prediction depends on the quality of embeddings.
- **The embedding computation shall be fastprocess.**
  Graphs are usually large. Imagine the social networks with millions of people. A good embedding approach needs to be efficient on large graphs.
- **The size of embedding has to be appropriate**
  Longer embeddings preserve more information while they induce higher time and space complexity than sorter embeddings. Typical embedding size is between 128 and 256.
- A simple node embedding example is Personal PageRank
- A more advanced ML-based technique is DeepWalk
- Many algorithms to represent nodes and edges

PageRank for ranking documents



PageRank formalization

- *p* = web page
- *O(p)* = pages pointed to by p
- $I(p) = \{i_1, i_2, \ldots, i_n\}$ pages pointing to *p*
- *d* = damping factor between 0 and 1 (default 0.85 or 0.9)

- Page quality π(p) depends on quality of pages pointing to it

$$\pi(p) = (1-d) + d\,\frac{\pi(i_1)}{|O(i_1)|} + \ldots + d\,\frac{\pi(i_n)}{|O(i_n)|}$$

PageRank computation

- Iterative computation,
- matrix form
- random surfer, intentional surfer
- Personal PageRank produces node embeddings

Deep Walk

- Based on word2vec text embedding

Word2vec predicts neighbouring words

frequent words often provide little information

frequent words often provide little information

frequent words often provide little information

frequent words often provide little information

word2vec (skip-gram) training data

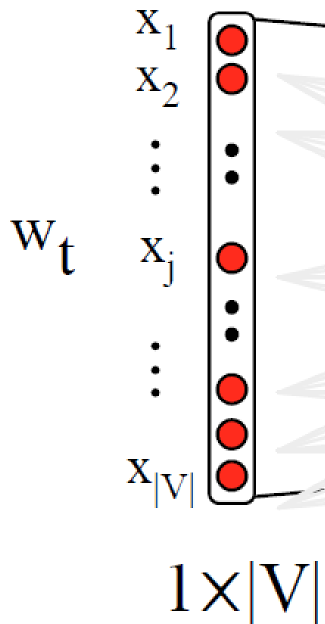
- Training sentence:
- ... lemon, a tablespoon of **apricot** jam a pinch ...
- c1 c2 target c3 c4

- Asssume context words are those in +/- 2 word window
- Get negative training examples randomly
- train a neural network to predict probability of a co-occurring word

Neural network based word2vec embedding

Input layer

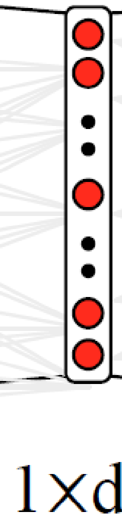
1-hot input vector



$W_{|V| \times d}$

Projection layer

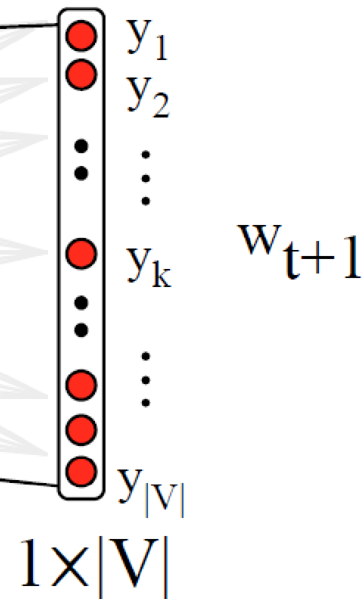
embedding for w_t



$C_{d \times |V|}$

Output layer

probabilities of context words



DeepWalk

- DeepWalk uses random walks to produce embeddings. The random walk starts in a selected node then we move to the random neighbor from a current node for a defined number of steps.
- The method consists of three steps:
  - *Sampling:* A graph is sampled with random walks. Few random walks from each node are performed. Usually it is sufficient to perform from 32 to 64 random walks from each node, and each walk has a length of about 40 steps.
  - *Training skip-gram:* Random walks are comparable to sentences in word2vec approach. The skip-gram network accepts a node from the random walk as a one-hot vector as an input and maximizes the probability for predicting neighbor nodes. It is typically trained to predict around 20 neighbor nodes — 10 nodes left and 10 nodes right.
  - *Computing embeddings:* Embedding is the output of a hidden layer of the network. The DeepWalk computes embedding for each node in the graph.