Message Passing in Machine Learning

Wee Sun Lee
School of Computing
National University of Singapore
leews@comp.nus.edu.sg

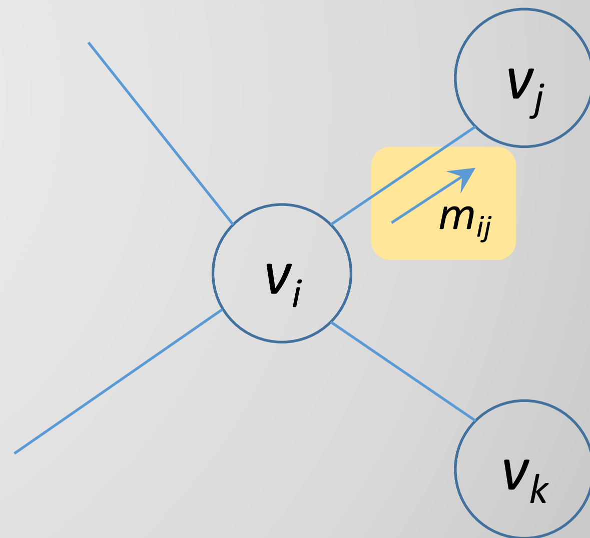
Neural Information Processing Systems December 2021

Message Passing

Machine learning using **distributed algorithm** on graph $G = (V, E)$.

Node $v_i \in V$ do **local computation,** depends only on information at v_i, neighbours, and incident edges $e_{ij} \in E$.

Information sent through e_{ij} to v_j as **message** m_{ij}.



Why Message Passing Algorithms?

Effective in practice!

- Graph neural networks
- Transformer
- Probabilistic graphical model inference
- Value iteration for Markov decision process ....

Potentially easier to parallelize.

Plan for Tutorial

Cover better understood, more **"interpretable"** models and algorithms

- Probabilistic graphical models (PGM), inference algorithms
- Markov decision process (MDP), decision algorithm

What do the components of the models **represent**?

What **objective functions** are the algorithms are optimizing for?

Discuss more flexible, less "interpretable" methods

- Graph neural networks
- Attention networks, transformer

Connect to PGM and MDP, help understand the **inductive biases**.

Outline

- Message Passing
- **Probabilistic Graphical Models**
- Markov Decision Process
- Graph Neural Networks and Attention Networks

Probabilistic Graphical Model

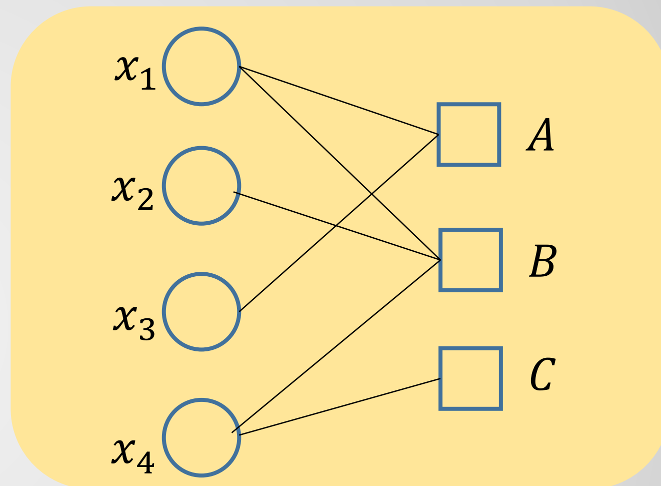
Focus on **Markov random fields**

N random variables $\{X_1, X_2, \dots, X_N\}$

$p(\boldsymbol{x}) = p(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n)$
factors into a product of functions

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_a \psi_a(\boldsymbol{x}_a)$$

- M non-negative **compatibility** or **potential** functions $\psi_A, \psi, \dots, \psi_M$
- x_a, the argument of ψ_a, is a subset of $\{x_1, x_2, \dots, x_N\}$
- Z, the **partition function**, is the normalizing constant



Graphical representation using **factor graph**: bipartite graph, each factor node connected to variable nodes that it depends on

$\frac{1}{Z}\psi_A(x_1, x_3)\psi_B(x_1, x_2, x_4)\psi_C(x_4)$

When compatibility functions always positive, can write

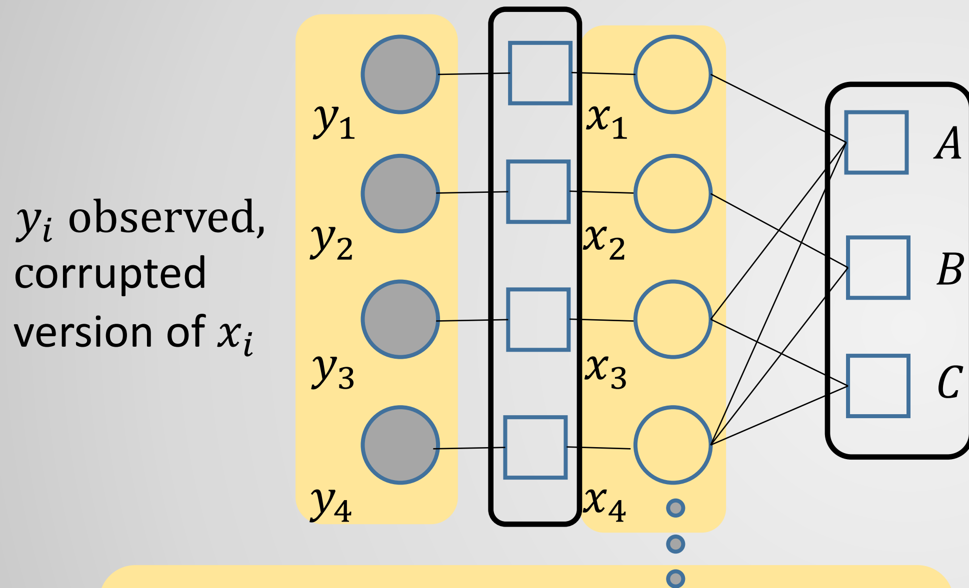
$$p(\boldsymbol{x}) = \frac{1}{Z}\prod_a \psi_a(\boldsymbol{x}_a)$$
$$= \frac{1}{Z}e^{-E(\boldsymbol{x})}$$

where

$$E(\boldsymbol{x}) = -\sum_a^M \ln \psi_a(\boldsymbol{x}_a)$$

is the **energy function**.

Error Correcting Codes



$$\psi_A(x_1, x_3, x_4) = \begin{cases} 1 \text{ if } x_1 \oplus x_2 \oplus x_4 = 1 \\ 0 \; otherwise \end{cases}$$

Codeword

Parity bits

[Fig from http://www.inference.org.uk/mackay/codes/gifs/]

MAP, Marginals, and Partition Function

Maximum a posteriori (MAP):

find a state $\boldsymbol{x}$ that maximizes $p(\boldsymbol{x})$

- Equivalently minimizes the energy $E(\boldsymbol{x})$

Marginal: probabilities for individual variable

$$p_i(x_i) = \sum_{\boldsymbol{x} \backslash x_i} p(\boldsymbol{x})$$

Partition function: Compute the normalizing constant

$$Z = \sum_{\boldsymbol{x}} \prod_{a} \psi_a(\boldsymbol{x}_a)$$

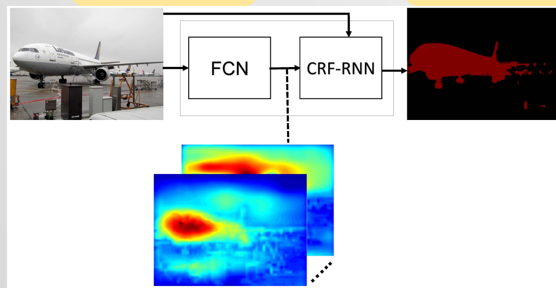
Semantic Segmentation

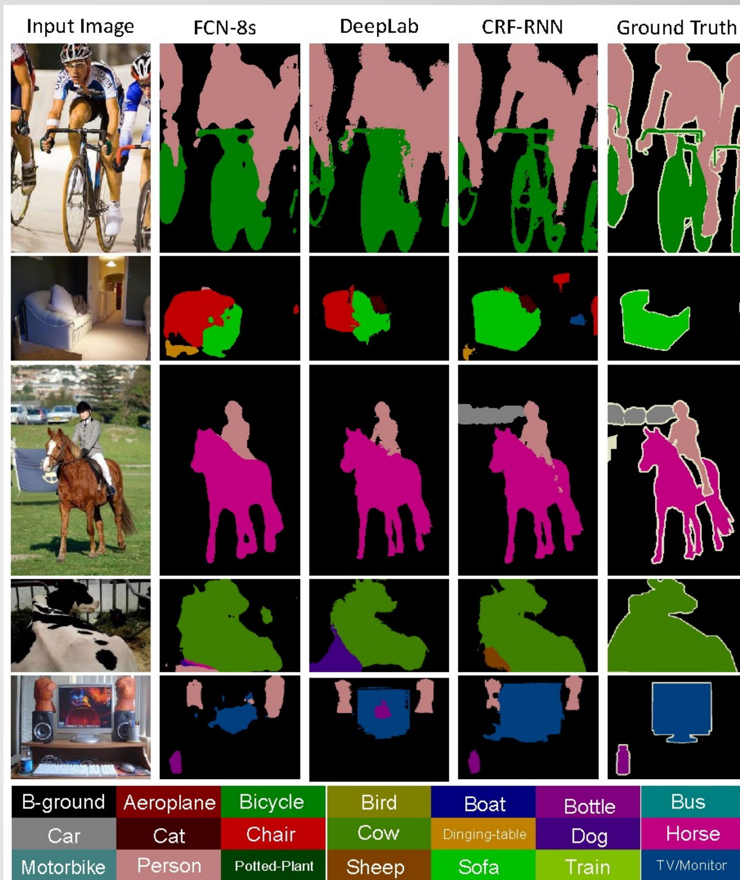
In **conditional random field (CRF)**, the **conditional distribution**

$p(\boldsymbol{x}|\boldsymbol{I}) = \frac{1}{Z(\boldsymbol{I})} e^{-E(\boldsymbol{x}|\boldsymbol{I})}$ is modeled.

- For semantic segmentation $\boldsymbol{I}$ is the image and $\boldsymbol{x}$ is the semantic class label.

  $E(\boldsymbol{x}|\boldsymbol{I}) = \sum_i \phi_u(x_i|\boldsymbol{I}) + \sum_{i<j} \phi_p(x_i, x_j|\boldsymbol{I})$





[Figs from Zheng et. al. 2015]

Belief Propagation

Message passing algorithm

Sum product computes the marginals

$$n_{ia}(x_i) = \prod_{c \in N(i) \setminus a} m_{ci}(x_i)$$

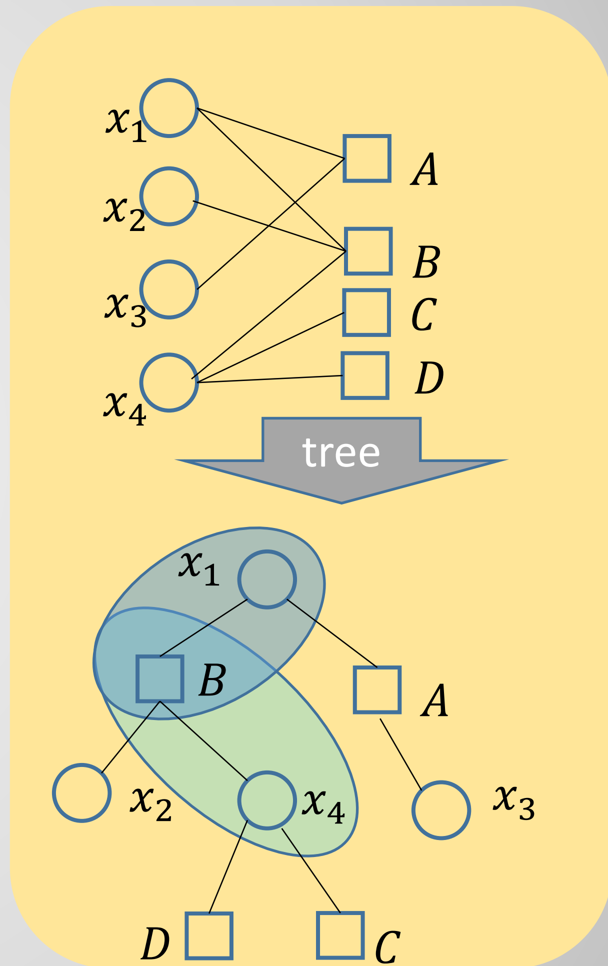
$$m_{ai}(x_i) = \sum_{\boldsymbol{x}_a \setminus x_i} \psi_a(\boldsymbol{x}_a) \prod_{j \in N(a) \setminus i} n_{ij}(x_j)$$

$$b_{i(x_i)} \propto \prod_{a \in N(i)} m_{ai}(x_i)$$

Max product solves the MAP problem: just replace sum with max in message passing

Works exactly on **trees**, dynamic programming

Correctness



Belief Propagation on Trees

Every variable and factor nodes compute messages **in parallel** at each iteration

- After $O(D)$ iterations, where D is the diameter of the tree, all messages and all marginals are correct.

Suffices to pass messages from leaves to the root and back

- More efficient for serial computation

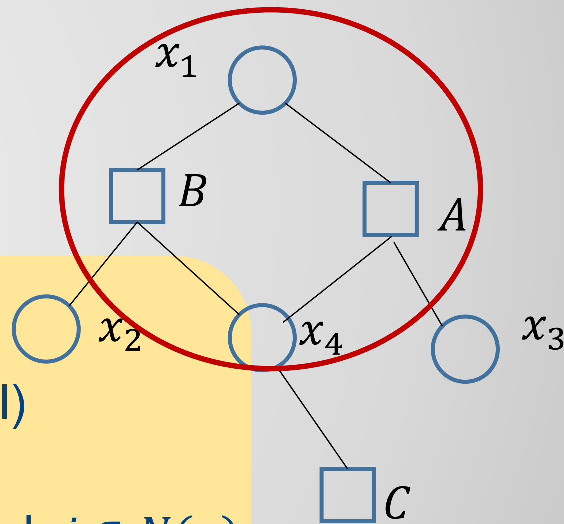
Loopy Belief Propagation

Belief propagation can also be applied to general probabilistic graphical models

Often called **loopy belief propagation**

As a message passing algorithm:

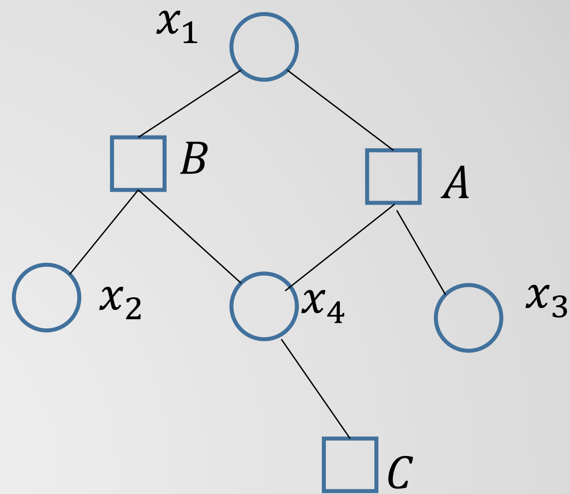
Init all messages n_{ia}, m_{ai} to all-one vectors
repeat T iterations
    for each variable i and factor a compute (in parallel)
        $n_{ia}(x_i) = \prod_{c \in N(i) \setminus a} m_{ci}(x_i)$ for each $a \in N(i)$
        $m_{ai}(x_i) = \sum_{\boldsymbol{x}_a \setminus x_i} \psi_a(\boldsymbol{x}_a) \prod_{j \in N(a) \setminus i} n_{ij}(x_j)$ for each $i \in N(a)$
return $b_{i(x_i)} = \frac{1}{Z_i} \prod_{a \in N(i)} m_{ai}(x_i)$ for each i



Belief propagation may fail when there are cycles

- May not even converge
- Often works well in practice when converges

Variational inference ideas help understand loopy belief propagation



$$n_{ia}(x_i) = \prod_{c \in N(i) \backslash a} m_{ci}(x_i)$$

$$m_{ai}(x_i) = \sum_{\boldsymbol{x}_a \backslash x_i} \psi_a(\boldsymbol{x}_a) \prod_{j \in N(a) \backslash i} n_{ij}(x_j)$$

$$b_{i(x_i)} \propto \prod_{a \in N(i)} m_{ai}(x_i)$$

Variational Principles

View message passing algorithms through lens of variational principles

- A variational principle solves a problem by viewing the solution as an extremum (maximum, minimum, saddle point) of a function or functional
- To understand or "interpret" an algorithm, ask "what objective might the algorithm implicitly be optimizing for?"

Variational Inference

In standard variational inference, we approximate the ***Helmholtz free energy***

$$F_H = -\ln Z$$

$$Z = \sum_{\boldsymbol{x}} e^{-E(\boldsymbol{x})}, \quad p(\boldsymbol{x}) = e^{-E(\boldsymbol{x})}/Z$$

by turning it into an optimization problem

Derivation:

$$\ln p(x) = -E(x) - \ln Z$$

$$F_H = -\ln Z = \sum_x q(x)\,E(x) + \sum_x q(x)\ln p(x)$$

$$= \sum_x q(x)\,E(x) + \sum_x q(x)\ln p(x) + \sum_x q(x)\ln q(x) - \sum_x q(x)\ln q(x)$$

$$= \sum_x q(x)\,E(x) + \sum_x q(x)\ln q(x) - \sum_x q(x)\ln q(x)/p(x)$$

For target belief p and arbitrary belief q

$$F_H = F(q) - KL(q||p)$$

where $F(q)$ is the ***variational free energy***

$$F(q) = \sum_x q(\boldsymbol{x})E(\boldsymbol{x}) + \sum_x q(\boldsymbol{x})\ln q(\boldsymbol{x})$$

and $KL(q||p)$ is the ***Kullback Lieber divergence*** between q and p

$$KL(q||p) = \sum_x q(\boldsymbol{x})\ln\frac{q(\boldsymbol{x})}{p(\boldsymbol{x})}$$

$KL(q||p) \geq 0$ and is zero when $q = p$.

From $F_H = F(q) - KL(q||p)$,
$F(q)$ is an upper bound for F_H

- Minimizing $F(q)$ improves approximation, exact when $q = p$

Minimizing $F(q)$ intractable in general

- One approximate method is to use a tractable q
- **Mean field** uses a factorized belief

$$q_{MF}(x) = \prod_{i=1}^{N} q_i(x_i)$$

Terminology

The **variational free energy**

$$F(q) = \sum_x q(\mathbf{x})E(\mathbf{x}) + \sum_x q(\mathbf{x}) \ln q(\mathbf{x})$$
$$= U(q) - H(q)$$

where $U(q)$= $\sum_x q(\mathbf{x})E(\mathbf{x})$

is the ***variational average energy***
and $H(q)$= -$\sum_x q(\mathbf{x}) \ln q(\mathbf{x})$
is the ***variational entropy***.

Mean Field

Mean field often solved by coordinate descent

- Optimize one variable at a time, holding other variables constant

$$q_j(x_j) = \frac{1}{Z_j'} \exp\left(-\sum_{\boldsymbol{x} \backslash \mathrm{x}_j} \prod_{i \neq j}^{N} q_i(x_i)\, E(\boldsymbol{x})\right)$$

Derivation

Coordinate descent converges to local optimum.

- Local optimum is **fixed point** of updates for all variables.
- Parallel updates can also be done but may not always converge

Mean field as message passing

Previously

$$q_j(x_j) = \frac{1}{Z_j'} \exp\left(-\sum_{\boldsymbol{x}\backslash x_j} \prod_{i\neq j}^{N} q_i(x_i)\, E(\boldsymbol{x})\right)$$

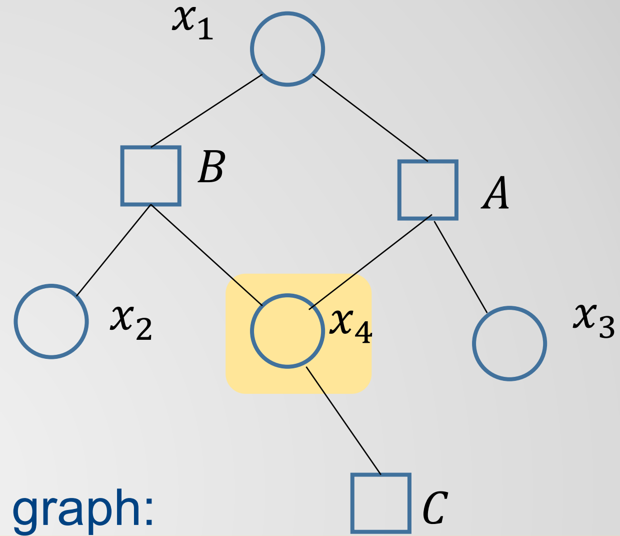
Recall $E(\boldsymbol{x}) = -\sum_a^M \ln \psi_a(\boldsymbol{x}_a)$

$$q_j(x_j) = \frac{1}{Z_j'} \exp\left(\sum_{\boldsymbol{x}\backslash x_j} \prod_{i\neq j}^{N} q_i(x_i) \sum_a^M \ln \psi_a(\boldsymbol{x}_a)\right)$$

$$= \frac{1}{Z_j'} \exp\left(\sum_{a\in N(j)} \sum_{\boldsymbol{x}\backslash x_j} \prod_{i\neq j}^{N} q_i(x_i) \ln \psi_a(\boldsymbol{x}_a) + \sum_{a\notin N(j)} \sum_{\boldsymbol{x}\backslash x_j} \prod_{i\neq j}^{N} q_i(x_i) \ln \psi_a(\boldsymbol{x}_a)\right)$$

Does not depend on x_j, constant

To compute $q_j(x_j)$, only need $\psi_a(\boldsymbol{x}_a)$ for neighbouring factors $a \in N(j)$



As a message passing algorithm on a factor graph:

repeat T iterations

for each variable j compute (serially or in parallel)

$m_{aj}(x_j) = \sum_{\boldsymbol{x}_a \setminus \mathrm{x}_j} \prod_{i \in N(a), i \neq j}^{N} q_i(x_i) \ln \psi_a(\boldsymbol{x}_a)$ for $a \in N(j)$ in parallel

$q_j(x_j) = \frac{1}{Z_j'} \exp\left(\sum_{a \in N(j)} m_{aj}(x_j)\right)$

Loopy Belief Propagation and Bethe Free Energy[1]

For a tree-structured factor graph, the variational free energy

$$F(q) = \sum_{x} q(\boldsymbol{x})E(\boldsymbol{x}) + \sum_{x} q(\boldsymbol{x}) \ln q(\boldsymbol{x})$$

$$= -\sum_{a=1}^{M} \sum_{\boldsymbol{x}_a} q_a(\boldsymbol{x}_a) \ln \psi_a(\boldsymbol{x}_a)$$

Variational average energy

$$+ \sum_{a=1}^{M} \sum_{\boldsymbol{x}_a} q_a(\boldsymbol{x}_a) \ln \frac{q_a(\boldsymbol{x}_a)}{\prod_{i \in N(a)} q_i(x_i)}$$

$$+ \sum_{i=1}^{N} \sum_{x_i} q_i(x_i) \ln q_i(x_i)$$

Variational entropy

Derivation

[1] Yedidia, Jonathan S., William T. Freeman, and Yair Weiss. "Constructing free-energy approximations and generalized belief propagation algorithms." IEEE Transactions on information theory 51.7 (2005): 2282-2312.

For a tree-structured factor graph, the variational free energy

$$F(q) = \sum_{x} q(\boldsymbol{x})E(\boldsymbol{x}) + \sum_{x} q(\boldsymbol{x}) \ln q(\boldsymbol{x})$$

$$= -\sum_{a=1}^{M}\sum_{\boldsymbol{x}_a} q_a(\boldsymbol{x}_a) \ln \psi_a(\boldsymbol{x}_a) + \sum_{a=1}^{M}\sum_{\boldsymbol{x}_a} q_a(\boldsymbol{x}_a) \ln \frac{q_a(\boldsymbol{x}_a)}{\prod_{i\in N(a)} q_i(x_i)} + \sum_{i=1}^{N}\sum_{x_i} q_i(x_i) \ln q_i(x_i)$$

For the Bethe approximation, the following Bethe free energy $F_{Bethe}(q) = U_{Bethe}(q) - H_{Bethe}(q)$ is used even though the graph may not be a tree, where

$$U_{Bethe}(q) = -\sum_{a=1}^{M}\sum_{\boldsymbol{x}_a} q_a(\boldsymbol{x}_a) \ln \psi_a(\boldsymbol{x}_a)$$

$$\mathrm{H}_{Bethe} = -\sum_{a=1}^{M}\sum_{\boldsymbol{x}_a} q_a(\boldsymbol{x}_a) \ln \frac{q_a(\boldsymbol{x}_a)}{\prod_{i\in N(a)} q_i(x_i)} - \sum_{i=1}^{N}\sum_{x_i} q_i(x_i) \ln q_i(x_i)$$

In addition, we impose the constraints

- $\sum_{x_i} q_i(x_i) = \sum_{\boldsymbol{x}_a} q_a(\boldsymbol{x}_a) = 1$
- $q_i(x_i) \geq 0, \quad q_a(x_a) \geq 0$
- $\sum_{\boldsymbol{x}_a \setminus \mathrm{x}_i} q_a(\boldsymbol{x}_a) = q_i(x_i)$

What does loopy belief propagation optimize?

Loopy belief propagation equations give the stationary points of the constrained Bethe free energy.

Derivation

We only specify the factor marginals $q_a(\boldsymbol{x}_a)$ and the variable marginals $q_i(x_i)$.

- There may be no distribution q whose marginals agree with $q_a(\boldsymbol{x}_a)$
- Often called **pseudomarginals** instead of marginal

Furthermore, the Bethe entropy $H_{Bethe}(q)$ is an approximation of the variational entropy when the graph is not a tree

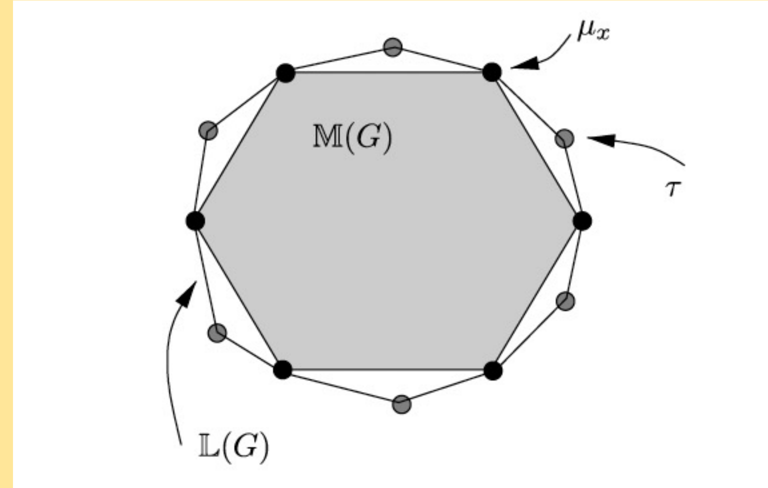


Figure from Wainwright and Jordan 2008. The set of marginals from valid probability distributions $M(G)$ is a strict subset of the set of of pseudomarginals $L(G)$.

Variational Inference Methods

Mean field minimizes the variational free energy

$$F(q) = U(q) - H(q)$$

- Assumes fully factorized q for tractability
- Can be extended to other tractable q: structured mean field
- Minimizes upper bound of Helmholtz free energy $F_H = -\ln Z$
- Converges to local optimum if coordinate descent used, may not converge for parallel update
- Update equations be computed as message passing on graph

Variational Inference Methods

Loopy belief propagation can be viewed as minimizing Bethe free energy, $F_{Bethe}(q) = U_{Bethe}(q) - H_{Bethe}(q)$, an approximation of variational free energy

- May not be an upper bound of F_H
- Resulting q may not be consistent with a probability distribution
- May not converge, but performance often good when converges
- Message passing on a graph, various methods to help convergence, e.g. scheduling messages, damping, etc.
- Extension to generalized belief propagation for other region based free energy, e.g. Kikuchi free energy

Other commonly found variational inference message passing methods include expectation propagation, also max product linear programming relaxations for finding MAP approximations.

Parameter Estimation

Learn parameterized compatibility functions or components of energy function $E(\boldsymbol{x}|\boldsymbol{\theta}) = -\sum_a^M \ln \psi_a(\boldsymbol{x}_a|\theta)$

- Can do maximum likelihood estimation
- If some variables are not observed, can do the EM algorithm
- If inference intractable, variational approximation for estimating the latent variables is one approach: variational EM
  - With mean field approximation, maximize a lower bound of likelihood function
- Can also treat parameters as latent variables: Variational Bayes

For this tutorial, focus on unrolling the message passing algorithm into a deep neural network and doing end-to-end learning (later).

Outline

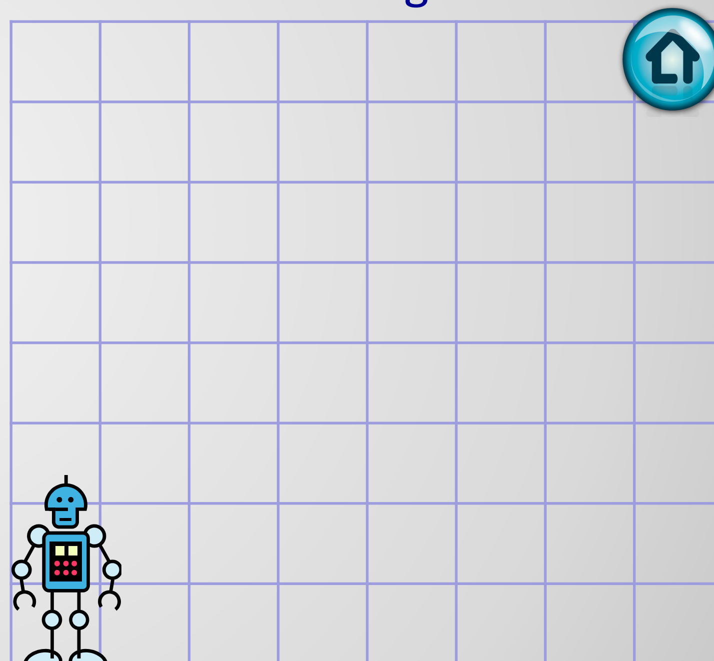
- Message Passing
- Probabilistic Graphical Models
- **Markov Decision Process**
- Graph Neural Networks and Attention Networks

Markov Decision Process

Markov Decision Process (MDP) is defined by $\langle S, A, T, R \rangle$

State S : Current description of the world

- Markov: the past is irrelevant once we know the state
- Navigation example: Position of the robot

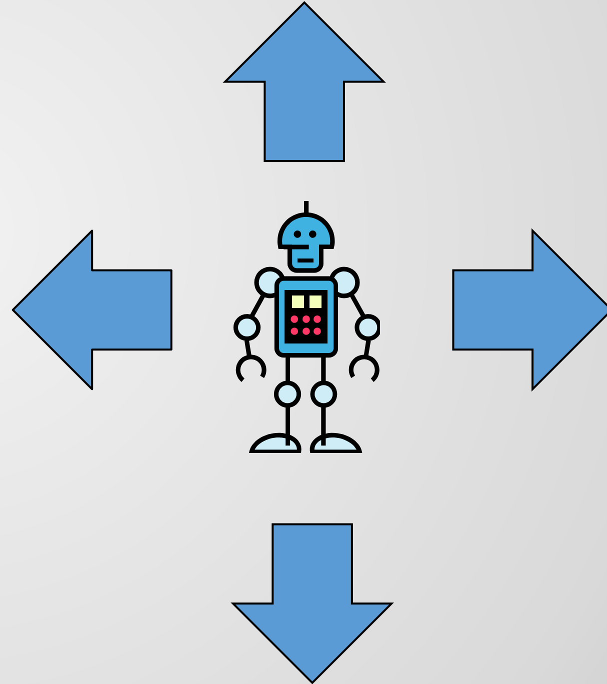


MDP $\langle S, A, T, R \rangle$

Actions A : Set of available actions

- Navigation example:
  - Move North
  - Move South
  - Move East
  - Move West

Robot navigation

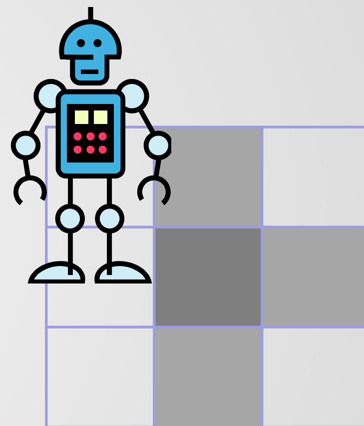


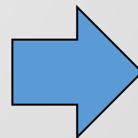
MDP $\langle S, A, T, R \rangle$

Transition function T :

- $T(s, a, s') = P(s'|s, a)$
- Navigation example:
  - Darker shade, higher probability

Robot navigation



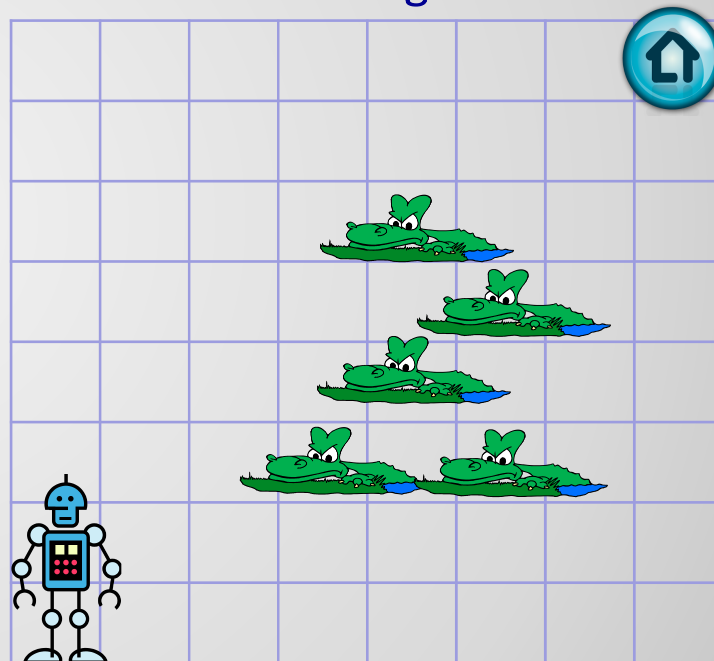


MDP $\langle S, A, T, R \rangle$

Reward function R : Reward received when action a in state s results in transition to state s'

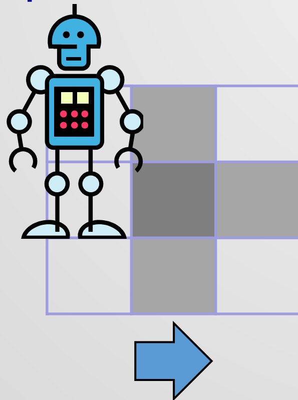
- $R(s, a, s')$
- Navigation example:
  - 100 if s' is Home
  - -100 if s' is in the danger zone
  - -1 otherwise
- Can be a function of a subset of s, a, s' as in navigation example

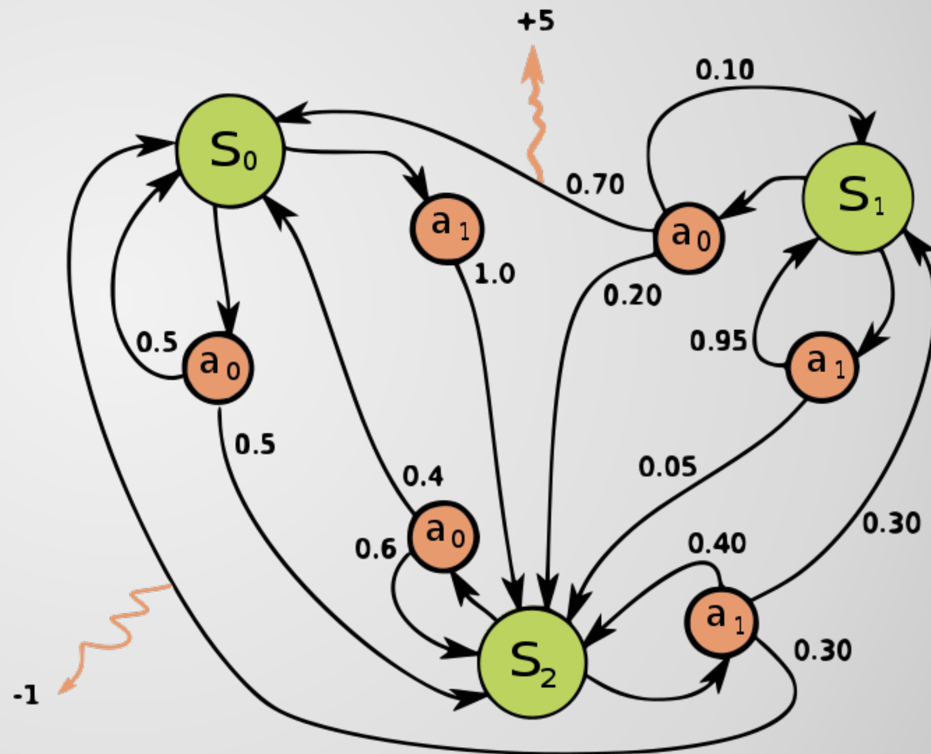
Robot navigation



Example of 3 state, two action Markov Decision Process $\langle S, A, T, R \rangle$

- Transition can be sparse as in navigation example



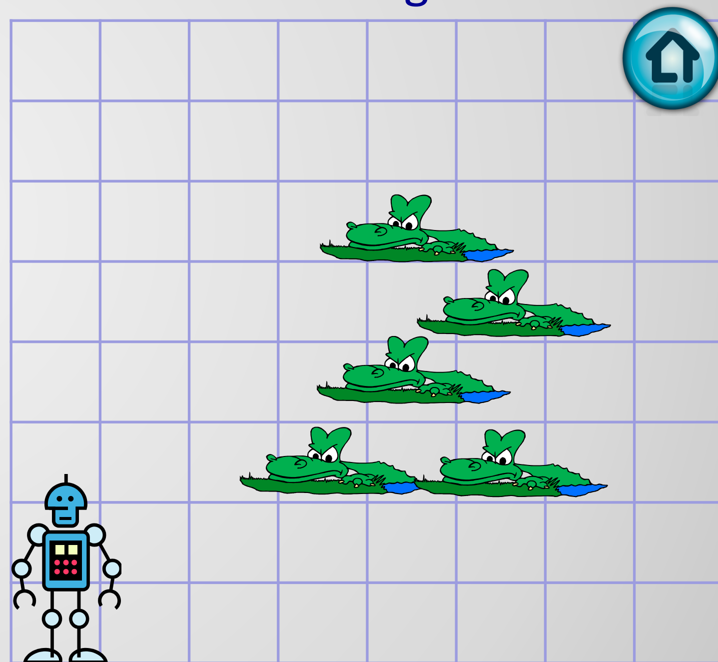


[Fig by waldoalvarez CC BY-SA 4.0]

MDP $\langle S, A, T, R \rangle$

Policy π: Function from state and time step to action

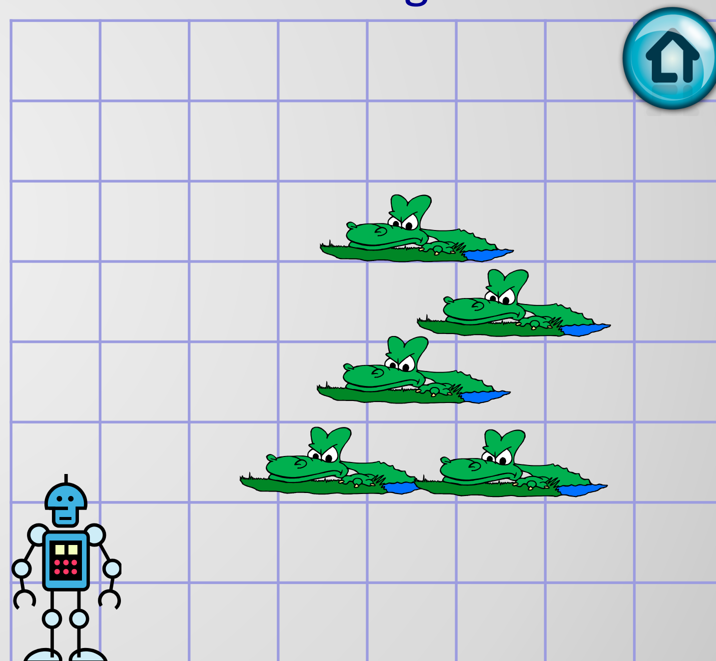
- $a = \pi(s, t)$
- Navigation example:
  - Which direction to move at current location at step t



MDP $\langle S, A, T, R \rangle$

Value function V_π: How good is a policy π when started from state s

- $V_\pi(s_0) = \sum_{t=0}^{T-1} E[R(s_t, \pi(s_t, t), s_{t+1})]$
- T is the horizon
- When horizon is infinite, usually use discounted reward
  - $V_\pi(s) = \sum_{t=0}^{\infty} E[\gamma^t R(s_t, \pi(s_t, t), s_{t+1})]$
  - $\gamma \in (0,1)$ is discount factor

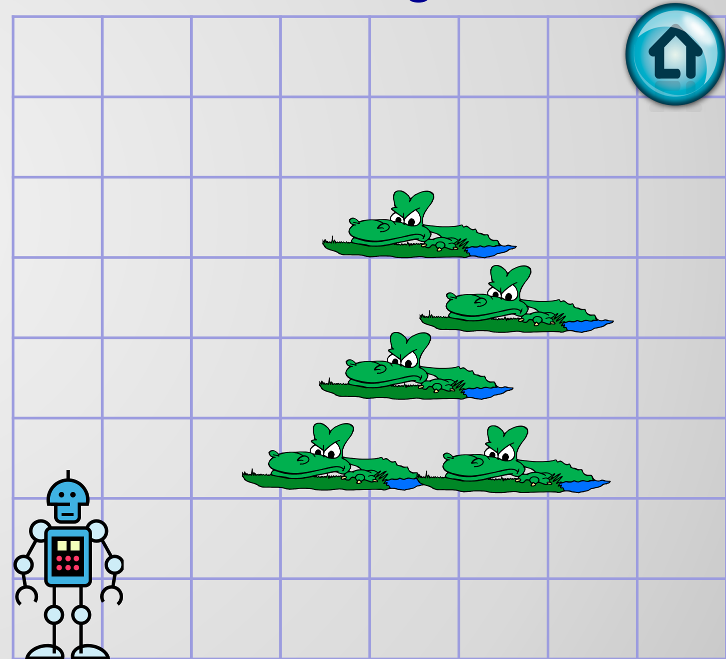


MDP $\langle S, A, T, R \rangle$

Optimal policy π^*: policy that maximizes

$$V_\pi(s_0) = \sum_{t=0}^{T-1} E[R(s_t, \pi(s_t, t), s_{t+1})]$$

- For infinite horizon, discounted reward MDP, optimal policy $\pi^*(s)$ is stationary (independent of t)
- **Optimal value $V(s) = V_{\pi^*}(s)$**: value corresponding to optimal policy



Value Iteration Algorithm

Dynamic programming algorithm for solving MDPs

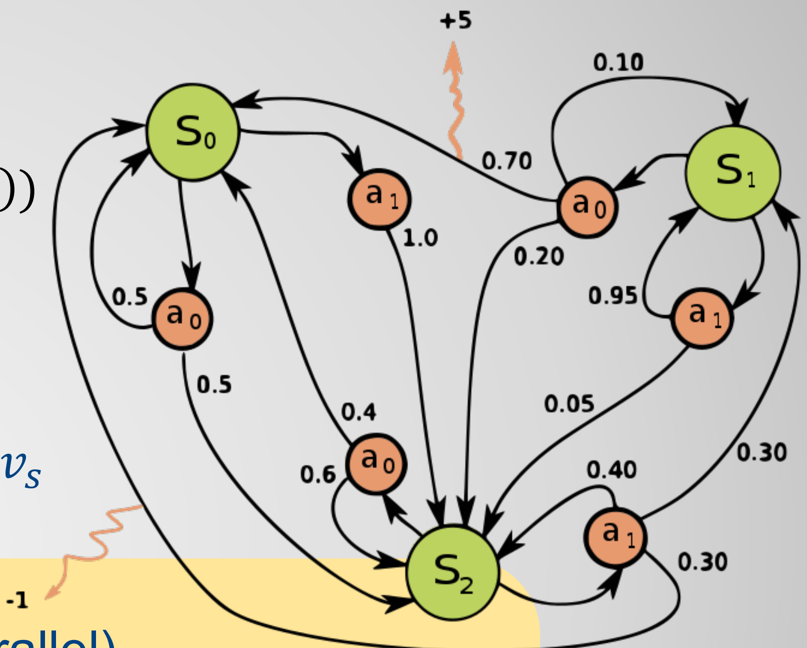
Let $V(s, T)$ denote the optimal value at s when horizon is T, initialized with $V(s, 0) = v_s$.

Then

$$V(s,T) = \max_a E[R(s,a,s') + V(s',T-1)]$$

$$= \max_a \sum_{s'} p(s'|a,s)(R(s,a,s') + V(s',T-1))$$

$$V(s,T) = \max_a \sum_{s'} p(s'|a,s)(R(s,a,s') + V(s',T-1))$$



As message passing on a graph

- Node at each state s, initialized to $V(s,0) = v_s$
- Utilize $|A|$ 'heads', one for each action a
- **repeat** T iterations

  for each action a of each state s (in parallel)

  Collect $p(s'|a,s)(R(s,a,s') + V(s'))$ from all s' to a at s if $p(s'|a,s)$ non-zero

  Sum all messages

  for each node s (in parallel)

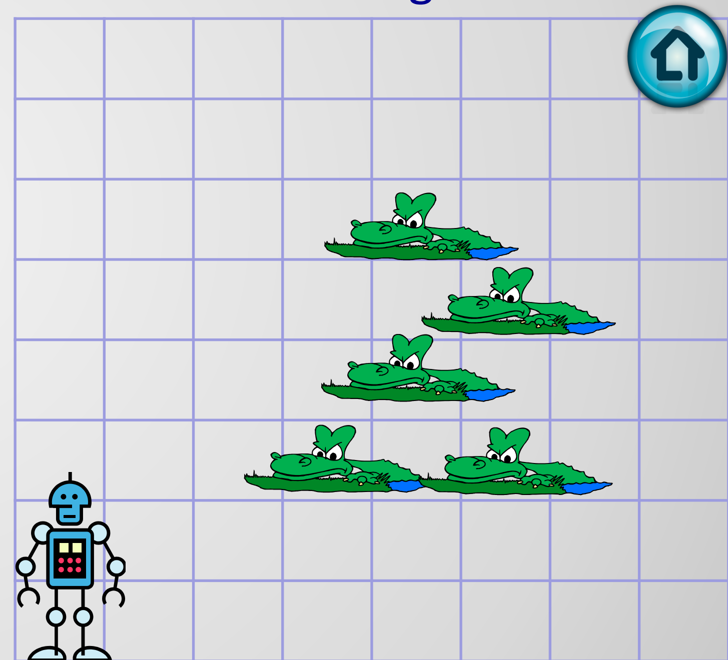
  Collect message from its corresponding actions a

  Take the maximum of the messages

[Fig by waldoalvarez CC BY-SA 4.0]

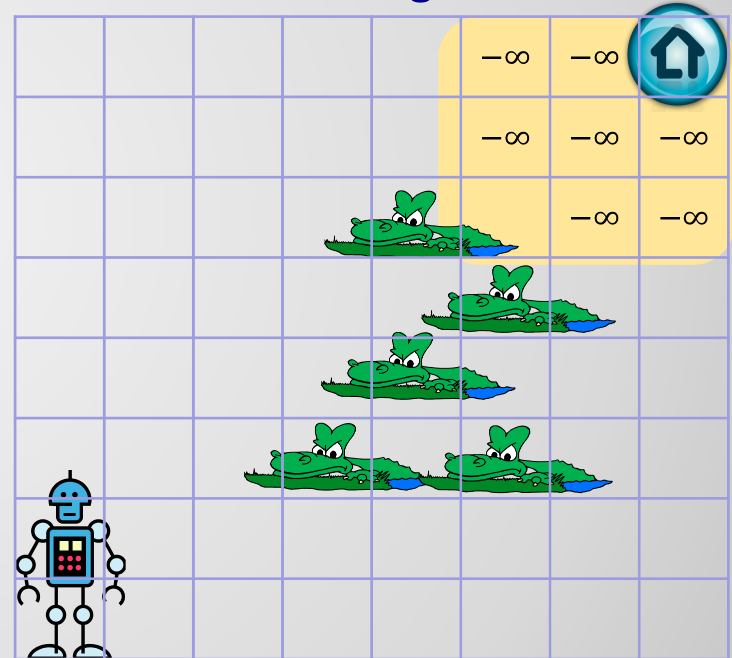
Shortest path example

- Deterministic transition (only one next state with prob 1 from each action)
- Initialize $V(s, 0) = 0$ for goal state, init to $-\infty$ for other states
- Self loop with 0 reward at goal state for all actions
- Reward $-w_{ij}$ for moving from i to j, $-\infty$ if no edge between the two nodes
- Value iteration is **Bellman-Ford** shortest path algorithm



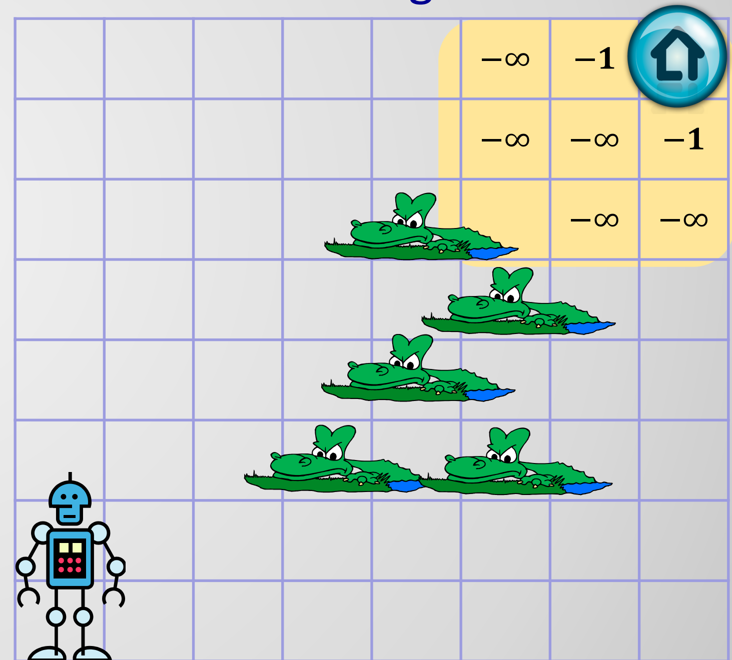
At initialization

After k iterations, values at each node is the value of the (-ve) shortest path from the node to the goal, reachable within k steps.



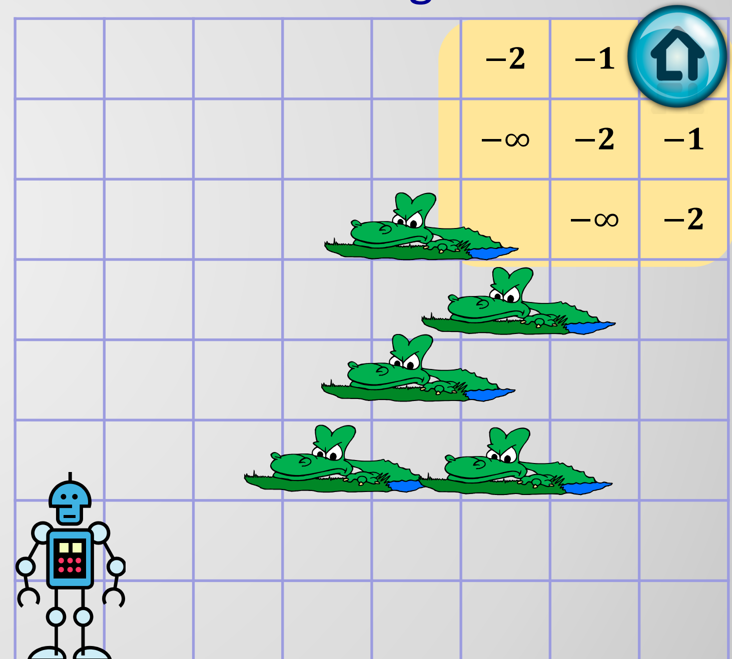
After 1 iteration

After k iterations, values at each node is the value of the (-ve) shortest path from the node to the goal, reachable within k steps.



After 2 iterations

After k iterations, values at each node is the value of the (-ve) shortest path from the node to the goal, reachable within k steps.



For the infinite horizon discounted case, the dynamic programming equation (Bellman equation) is

$$V(s) = \max_a E[R(s,a,s') + \gamma V(s')]$$
$$= \max_a \sum_{s'} p(s'|a,s)(R(s,a,s') + \gamma V(s'))$$

Same value iteration algorithm with message changed to

$$p(s'|a,s)(R(s,a,s') + \gamma V(s'))$$

Converges to the optimal value function

Convergence of Value Iteration

The optimal value function $V(s)$ satisfies Bellman's principle of optimality

$$V(s) = \max_a \sum_{s'} p(s'|a,s)(R(s,a,s') + \gamma V(s'))$$

The Bellman update B in value iteration transforms V_t to V_{t+1} as follows

$$V_{t+1}(s) = \max_a \sum_{s'} p(s'|a,s)(R(s,a,s') + \gamma V_t(s'))$$

We denote this as $V_{t+1} = BV_t$.

The optimal value function is a fixed point of this operator $V = BV$.

The Bellman update is a contraction (for the max norm), i.e.

$$||BV_1 - BV_2|| \leq \gamma ||V_1 - V_2||$$

Derivation

From Bellman's equation, we have $V = BV$ for the optimal value function V.

Applying $V_t = BV_{t-1}$ repeatedly and using contraction property $||BV_1 - BV_2|| \leq \gamma ||V_1 - V_2||$ we have

$$||V_t - V|| = ||BV_{t-1} - BV|| \leq \gamma ||V_{t-1} - V|| \leq \gamma^t ||V_0 - V||$$

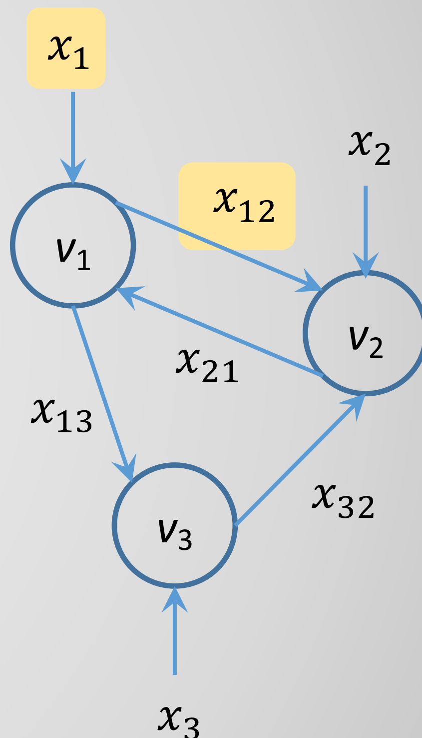
Distance converges exponentially to 0 for any initial value V_0

Outline

- Message Passing
- Probabilistic Graphical Models
- Markov Decision Process
- **Graph Neural Networks and Attention Networks**

Graph Neural Networks

- Many effective graph neural networks (GNN): GCN, GIN, …
- **Message passing neural networks** (MPNN)[1]: general formulation for GNNs as message passing algorithms.
  - Input: $G = (V, E)$, node attributes vectors $x_i, i \in V$, edge attribute vectors $x_{ij}, (i, j) \in E$
  - Output: Label or value for *graph classification* or *regression*, or a label/value for each node in *structured prediction*



[1] Gilmer, Justin, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. "Neural message passing for quantum chemistry." In International conference on machine learning, pp. 1263-1272. PMLR, 2017

MPNN pseudocode

Initialization: $h_i^0 = x_i$ for each $v_i \in V$

for layers $\ell = 1, \dots, d$

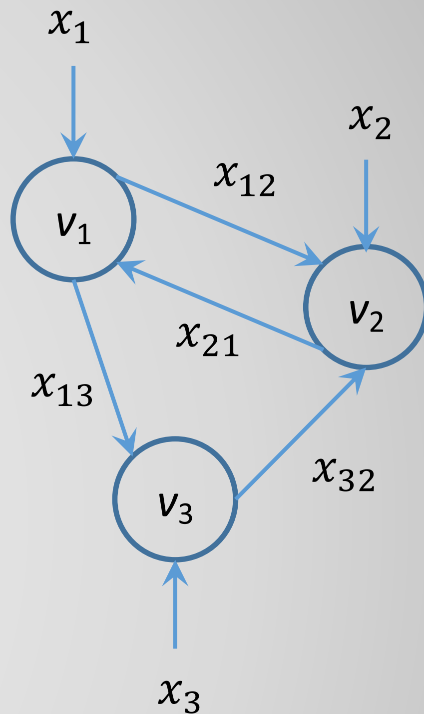
  for every edge $(i, j) \in E$ (in parallel)

    $m_{ij}^l = MSG_\ell(h_i^{\ell-1}, h_j^{\ell-1}, v_i, v_j, x_{ij})$

  for every node $v_i \in V$ (in parallel)

    $h_i^\ell = UP_\ell\left(\{m_{ij}^l : j \in N(i)\}, h_i^{\ell-1}\right)$

return h_i^d for every $v_i \in V$ or $y = READ\left(\{h_i^d : v_i \in V\}\right)$



- MSG_ℓ is arbitrary function, usually a neural net
- UP_ℓ aggregrates the messages from neighbours (usually with a set function) and combine with node embedding
- $READ$ is a set function for graph classification or regression tasks

GNN properties

If depth and width are large enough, message functions MSG_ℓ and update functions UP_ℓ are sufficiently powerful, and nodes can uniquely distinguish each other, then the MPNN is computationally universal[1]

- Equivalent to LOCAL model in distributed algorithms
- Can compute any function computable with respect to the graph and attributes (just send all information, including graph structure, to a single node, then compute there).

Notation

Depth: number of layers

Width: largest embedding dimension

[1] Loukas, Andreas. "What graph neural networks cannot learn: depth vs width." ICLR 2020

When using graph neural networks, we are often interested in permutation invariance and equivariance

Given an adjacency matrix $\boldsymbol{A}$, a permutation matrix $\boldsymbol{P}$, and attribute matrix $\boldsymbol{X}$ containing the attributes x_i on the i-th row

- Permutation invariance: $f\left(\boldsymbol{PAP}^T, \boldsymbol{PX}\right) = f(\boldsymbol{A}, \boldsymbol{X})$
- Permutation equivariance: $f\left(\boldsymbol{PAP}^T, \boldsymbol{PX}\right) = \boldsymbol{P}f(\boldsymbol{A}, \boldsymbol{X})$

Example:

$$\boldsymbol{P} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$\boldsymbol{P}$ permutes vertices $(1,2,3) \rightarrow (3,1,2)$

$\boldsymbol{PA}$ swap the rows

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 7 & 8 & 9 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$(\boldsymbol{PA})\boldsymbol{P}^T$ swap the columns after that

$$\begin{bmatrix} 7 & 8 & 9 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 9 & 7 & 8 \\ 3 & 1 & 2 \\ 6 & 4 & 5 \end{bmatrix}$$

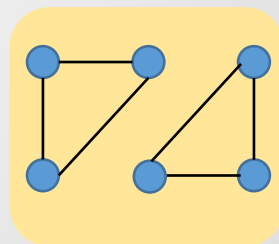
If MSG_ℓ does not depend on node ids v_i, v_j, MPNN is permutation invariant and equivariant for any permutation matrix $\boldsymbol{P}$,

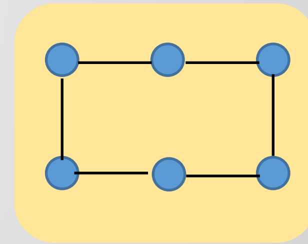
$$MPNN\left(\boldsymbol{PAP^T}, \boldsymbol{PX}\right) = \boldsymbol{P}MPNN(\boldsymbol{A}, \boldsymbol{X})$$

- However, lose approximation power if messages do not depend on node ids – cannot distinguish some graph structures
- Discrimination power at most as powerful as the 1-dimensional Weisfeiler-Lehman (WL) graph isomorphism test, which cannot distinguish certain graphs
  - Graph isomorphism network (GIN)[1] as powerful as 1-WL

MPNN pseudocode

Initialization: $h_i^0 = x_i$ for each $v_i \in V$
for layers $\ell = 1, \dots, d$
    for every edge $(i, j) \in E$ (in parallel)
        $m_{ij}^l = MSG_\ell(h_i^{\ell-1}, h_j^{\ell-1}, \sout{v_i, v_j}, x_{ij})$
    for every node $v_i \in V$ (in parallel)
        $h_\ell^\ell = UP_i(\{m_{ij}^l : j \in N(i)\}, h_i^{\ell-1})$
return h_i^d for every $v_i \in V$ or $y = READ(\{h_i^d : v_i \in V\})$



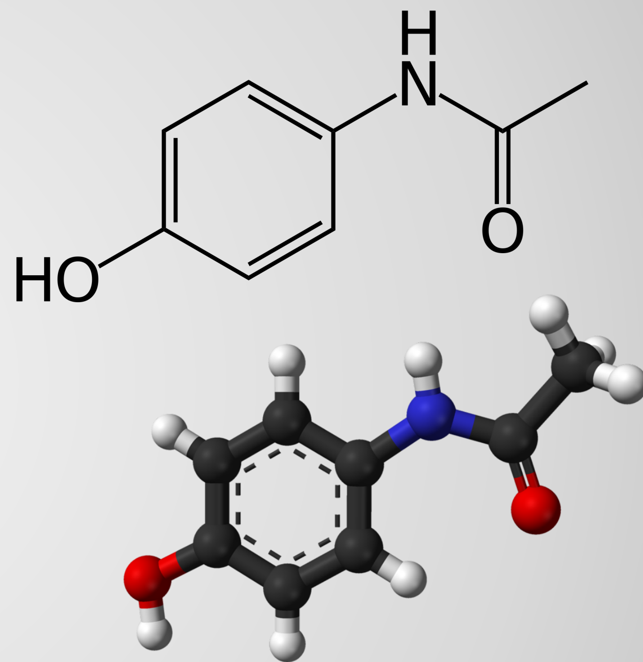


[1] Xu, Keyulu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. "How powerful are graph neural networks?" ICLR 2019

Drug Discovery

Molecules naturally represented as graphs

GNNs commonly used for predicting properties of molecules, e.g. whether it inhibits certain bacteria, etc.



[Figs on paracetamol by Benjah-bmm27 and Ben Mills are in the public domain]

Algorithmic Alignment

Sample complexity for learning a GNN is smaller for tasks that the GNN is algorithmically aligned with[1].

Graph Neural Network

for layers $\ell = 1, \dots, d$

for $v \in V$ (in parallel)

$$h_v^\ell = UP(\{MLP\left(h_u^{\ell-1}, h_v^{\ell-1}, w(u,v)\right)\}, h_v^{\ell-1})$$

$\mathrm{d}[\ell][v]$ easy function to approximate by MLP as function of $d[\ell-1][u], w$. Same MLP shared by all nodes, good inductive bias, so low sample complexity

Bellman Ford Shortest Path

for $\ell = 1, \dots, d$

for $v \in V$ (in parallel)

$$\mathrm{d}[\ell][v] = \min_u d[\ell-1][u] + w(u,v)$$

In contrast, learning entire for loop with a single function requires higher sample complexity

[1] Xu, Keyulu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. "What can neural networks reason about?." ICLR 2020

Alignment of GNN and VI

Value iteration algorithm for MDPs can be represented in a network form – value iteration network (VIN)[1].

- GNN well aligned with value iteration

Value Iteration

for $\ell = 1, \ldots, d$

for $v \in V$ (in parallel)

$$\mathcal{V}(v, \ell) = \max_{a} \sum_{u} p(u|a, v)(R(v, a, u) + \mathcal{V}(u, \ell - 1))$$

Graph Neural Network

for layers $\ell = 1, \ldots, d$

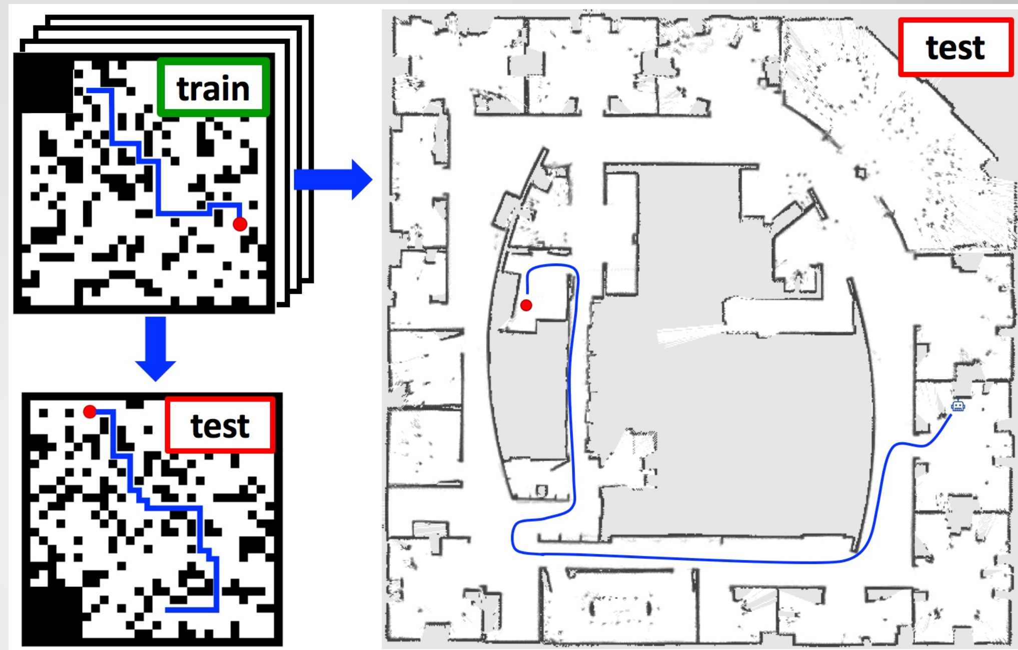
for $v \in V$ (in parallel)

$$h_v^{\ell} = UP(\{MLP(h_u^{\ell-1}, h_v^{\ell-1}, \{p(u|a, v), R(v, a, u)\})\}, h_v^{\ell-1})$$

[1] Tamar, Aviv, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. "Value iteration networks." NeurIPS 2016

Example: robot navigation using a map, using VIN[1]

- The transition $p(u|a, v)$ and reward $R(v, a, u)$ function may also need to be learned, instead of being provided.
- Message passing structure suggests
  - represent transition and reward separately, learned as function of image
  - use as input to same function at all states



Value Iteration

for $\ell = 1, \ldots, d$

for $v \in V$ (in parallel)

$$\mathcal{V}(v, \ell) = \max_a \sum_u p(u|a, v)(R(v, a, u) + \mathcal{V}(u, \ell - 1))$$

[1] Karkus, Peter, David Hsu, and Wee Sun Lee. "QMDP-net: Deep learning for planning under partial observability." NeurIPS 2017

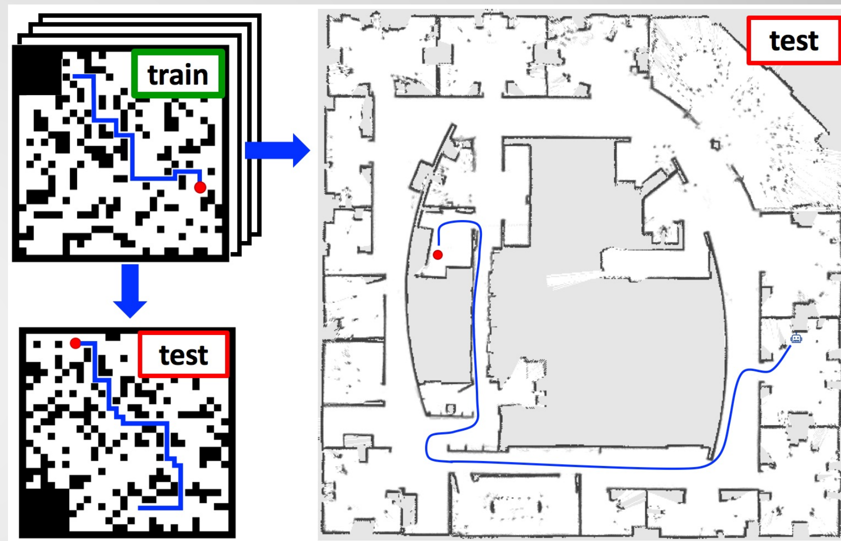
GNN well aligned with value iteration: may work well here

VIN has stronger inductive bias: encode value iteration equations directly

- Action heads: for each action, take weighted sum from neighbours
- Then max over actions

GNN potentially more flexible: also aligned other similar algorithms, particularly dynamic programming algorithms

- May work better if MDP assumption is not accurate
- Optimization may be easier for some types of architectures[1]



[1] Lee, Lisa, Emilio Parisotto, Devendra Singh Chaplot, Eric Xing, and Ruslan Salakhutdinov. "Gated path planning networks." ICML 2018.

Alignment of GNN and Graphical Model Algorithms

Mean field

repeat T iterations

for each variable j compute (serially or in parallel)

$m_{aj}(x_j) = \sum_{\boldsymbol{x}_a \setminus x_j} \prod_{i \in N(a), i \neq j}^{N} q_i(x_i) \ln \psi_a(\boldsymbol{x}_a)$ for $a \in N(j)$ in parallel

$q_j(x_j) = \frac{1}{Z_j'} \exp\left(\sum_{a \in N(j)} m_{aj}(x_j)\right)$

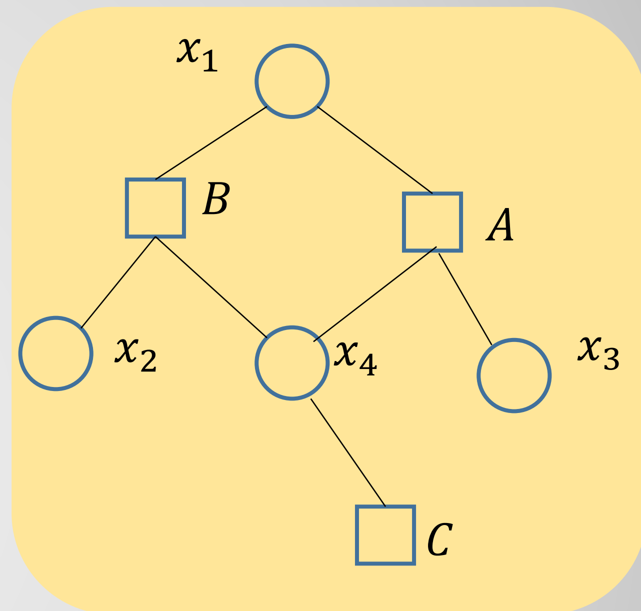
When all potential functions are pairwise $\psi_a(\boldsymbol{x}_a) = \psi_{i,j}(x_i, x_j)$, then $m_{aj}(x_j)$ is a function of only $q_i(x_i)$.

- Can send message directly from variable to variable without combining the messages at factor nodes
- Can interpret node embedding as feature representation of belief and learn a mapping from one belief to another as a message function[1]
  - GNN algorithmically well aligned with mean field for pairwise potential functions
  - Similarly well aligned with loopy belief propagation for pairwise potential

[1] Dai, Hanjun, Bo Dai, and Le Song. "Discriminative embeddings of latent variable models for structured data." ICML 2016.

What about with higher order potentials $\psi_a(\boldsymbol{x}_a)$ where $\boldsymbol{x}_a$ consists of n_a variables?

- In tabular representation, size of $\psi_a(\boldsymbol{x}_a)$ grows exponentially with n_a, so even loopy belief propagation is not efficient
- But if $\psi_a(\boldsymbol{x}_a)$ is low-ranked tensor, then loopy belief propagation can be efficiently implemented[1,2]
  - Represent $\psi_a(\boldsymbol{x}_a)$ as a tensor decomposition (CP decomposition), where k_a is the rank $\psi_a(\boldsymbol{x}_a) = \sum_{i=1}^{k_a} w_{a,1}^{i}\,(x_{a,1})\, w_{a,2}^{i}\,(x_{a,2}) \ldots w_{a,n_a}^{i}\,(x_{a,n_a})$
- Factor graph neural network (FGNN), which passes messages on a factor graph is well aligned with this



[1] Dupty, Mohammed Haroon, and Wee Sun Lee. "Neuralizing Efficient Higher-order Belief Propagation." arXiv preprint arXiv:2010.09283 (2020)
[2] Zhang, Zhen, Fan Wu, and Wee Sun Lee. "Factor graph neural network." NeurIPS 2020.

$$m_{ai}(x_i) = \sum_{\boldsymbol{x}_a \backslash \mathrm{x}_i} \psi_a(\boldsymbol{x}_a) \prod_{j \in N(a) \backslash \mathrm{i}} n_{ij}(x_j)$$

$$= \sum_{\boldsymbol{x}_a \backslash \mathrm{x}_i} \sum_{\ell=1}^{k_a} \prod_{j \in N(a)} w_{a,j}^{\ell}(x_j) \prod_{j \in N(a) \backslash \mathrm{i}} n_{ij}(x_j)$$

$$= \sum_{\ell=1}^{k_a} w_{a,i}^{\ell}(x_i) \sum_{\boldsymbol{x}_a \backslash \mathrm{x_}i} \prod_{j \in N(a) \backslash \mathrm{i}} w_{a,j}^{\ell}(x_j) n_{ij}(x_j)$$

$$= \sum_{\ell=1}^{k_a} w_{a,i}^{\ell}(x_i) \prod_{j \in N(a) \backslash \mathrm{i}} \sum_{\mathrm{x}_j} w_{a,j}^{\ell}(x_j) n_{ij}(x_j)$$

Implement message functions

$$n_{ia}(x_i) = \prod_{c \in N(i) \backslash \mathrm{a}} m_{ci}(x_i)$$

$$m_{ai}(x_i) = \sum_{\boldsymbol{x}_a \backslash \mathrm{x}_i} \psi_a(\boldsymbol{x}_a) \prod_{j \in N(a) \backslash \mathrm{i}} n_{ij}(x_j)$$

using

$$\psi_a(\boldsymbol{x}_a) = \sum_{\ell=1}^{k_a} \prod_{j \in N(a)} w_{a,j}^{\ell}(x_j)$$

$$n_{ia}(x_i) = \prod_{c \in N(i)\backslash a} m_{ci}(x_i)$$

$$m_{ai}(x_i) = \sum_{\ell=1}^{k_a} w_{a,i}^{\ell}(x_i) \prod_{j \in N(a)\backslash i} \sum_{x_j} w_{a,j}^{\ell}(x_j) n_{ij}(x_j)$$

In matrix notation

$$m_{ai} = \boldsymbol{W}_{ai}{}^{T} \left(\odot_{j \in N(a)\backslash i} \boldsymbol{W}_{aj} n_{ja} \right)$$

$$n_{ia} = \odot_{c \in N(i)\backslash a}\, m_{ci}$$

To make factor and variable messages symmetric

$$m'_{ai} = \odot_{j \in N(a)\backslash i} \boldsymbol{W}_{aj} n_{ja}$$

$$n_{ia} = \odot_{c \in N(i)\backslash a} \boldsymbol{W}_{ci}^{T} m'_{ci}$$

Matrix vector multiplication followed by product aggregation.

Matrix notation

- m_{ai} vector of length n_i
- n_{ia} vector of length n_i
- $w_{a,i}^{\ell}$ vector of length n_i
- $\boldsymbol{W}_{ai}$ matrix of k_a rows where each row is $(w_{a,i}^{\ell})^T$
- $\odot$ element-wise multiplication

Factor graph provides an easy way to specify dependencies, even higher order ones.

Loopy belief propagation can be approximated with the following message passing equations

$$m_{ai} = \bigodot_{j \in N(a) \setminus \mathrm{i}} \boldsymbol{W}_{aj} n_{ja}$$

$$n_{ia} = \bigodot_{c \in N(i) \setminus \mathrm{a}} \boldsymbol{W}_{ci}^{T} m_{ci}$$

Optimizes Bethe free energy if it converges.

- Uses low rank approximation for potential functions.
- Increasing number of rows of $\boldsymbol{W}_{ai}$ increases rank of tensor decomposition approximation.

Neuralizing Loopy Belief Propagation

Alignment shows that a neural network with small number of parameters can approximate an algorithm, smaller sample complexity in learning: **analysis**.

Can also use the ideas in **design.**

Neuralizing the algorithm

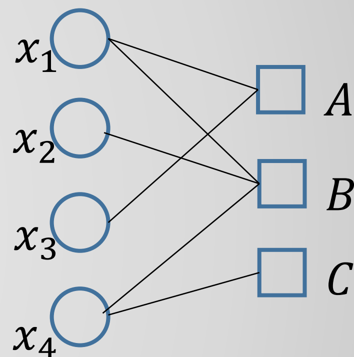
- Start with the network representing the algorithm to capture the inductive bias.
- Modify the algorithm, e.g. add computational elements to enhance approximation capability, while mostly maintaining the structure to keep the inductive bias.

- Start with an algorithm in network form, e.g.

$$m_{ai} = \bigodot_{j \in N(a) \setminus i} \boldsymbol{W}_{aj} n_{ja}$$
$$n_{ia} = \bigodot_{c \in N(i) \setminus a} \boldsymbol{W}_{ci}^T m_{ci}$$

- Add network elements to potentially make the network more powerful – enlarge the class of algorithms that can be learned, e.g.

$$m_{ai} = MLP(\bigodot_{j \in N(a) \setminus i} \boldsymbol{W}_{aj} n_{ja})$$
$$n_{ia} = MLP(\bigodot_{c \in N(i) \setminus a} \boldsymbol{W}_{ci}^T m_{ci})$$



- It is usually simpler to keep messages only on nodes instead of on edges, simplify while keeping message passing structure. Can also change aggregator, e.g. to sum, max, etc. Works well in practice

$$m_a = MLP(AGG_{j \in N(a)} \boldsymbol{W}_{aj} n_j)$$
$$n_i = MLP(AGG_{c \in N(i)} \boldsymbol{W}_{ci}^T m_c)$$

Message passing neural net on factor graph

Attention Network

Distributed Representation of Graphs and Matrices

A graph can be represented using an adjacency matrix

A matrix A, in turn can be factorized $A = UV^T$

In factorized form, $u_i^T v_j = A_{ij}$

- Entry (i, j) of matrix A is the inner product of row i of matrix U with row j of matrix V
- Node i of graph has an embeddings u_i, v_i such that the value of edge $(i.j)$ can be computed as $u_i^T v_j$
- Distributed representation of graph – information distributed to the nodes

$$A = UV^T \qquad A_{ij} = u_i^T v_j = \sum_k u_{ik} v_{jk}$$

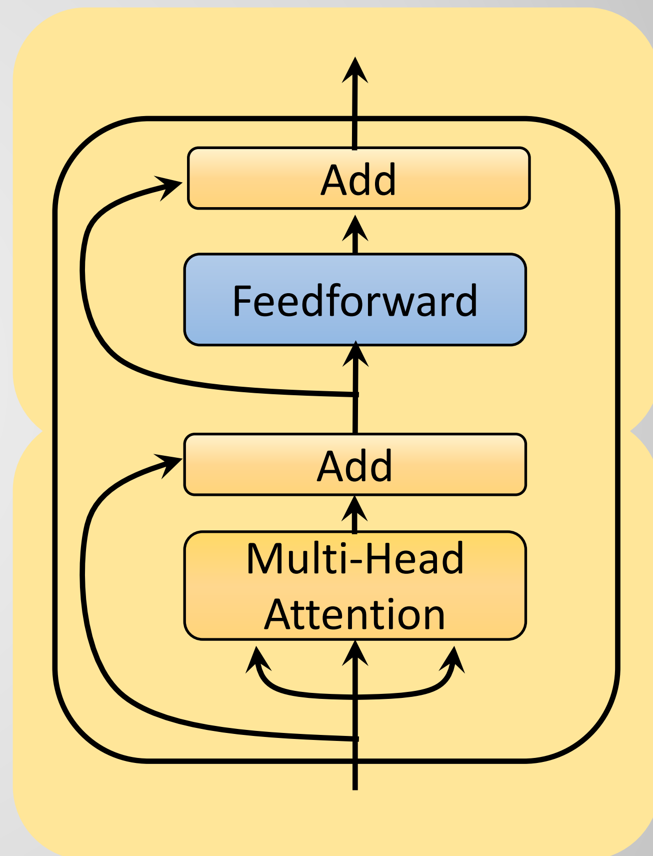
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \end{bmatrix} \begin{bmatrix} v_{11} & v_{21} & v_{31} \\ v_{12} & v_{22} & v_{32} \end{bmatrix}$$

With distributed representation, using a subset of embeddings u_i, v_i gives representation of subgraph!

Attention Network

Using matrix factorization, we can show that the transformer-type attention network is well aligned with value iteration for MDP

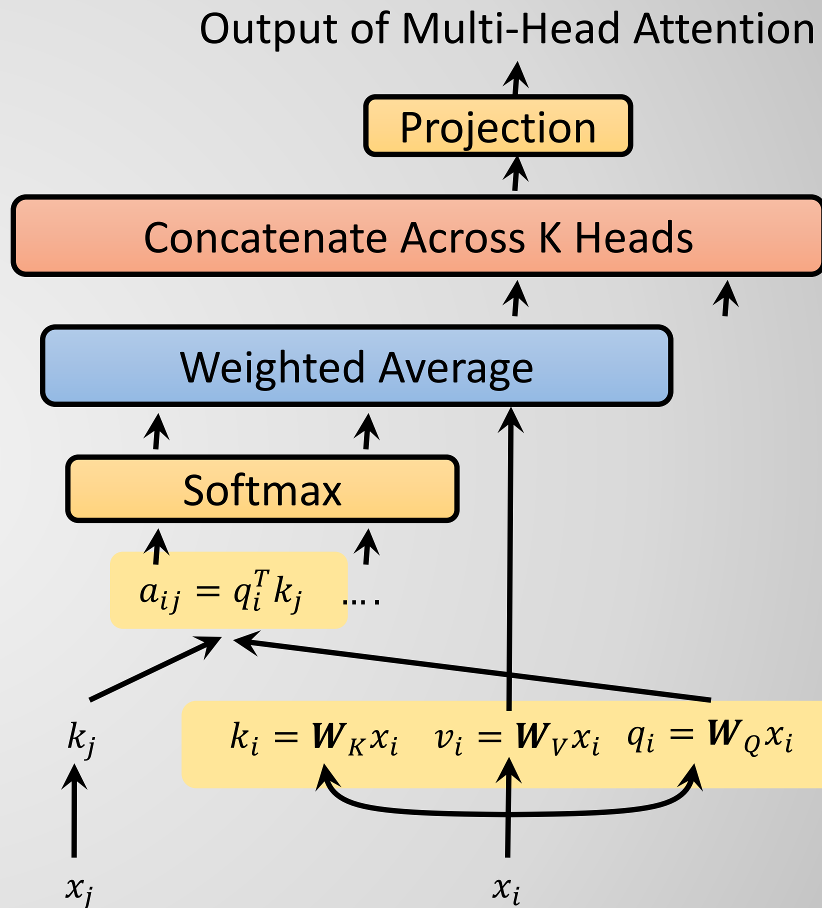
In the attention network, we have a set of nodes, each of which has an embedding as input, and the same operations (implemented with a network) are applied at each node in a layer.



Multi-head attention

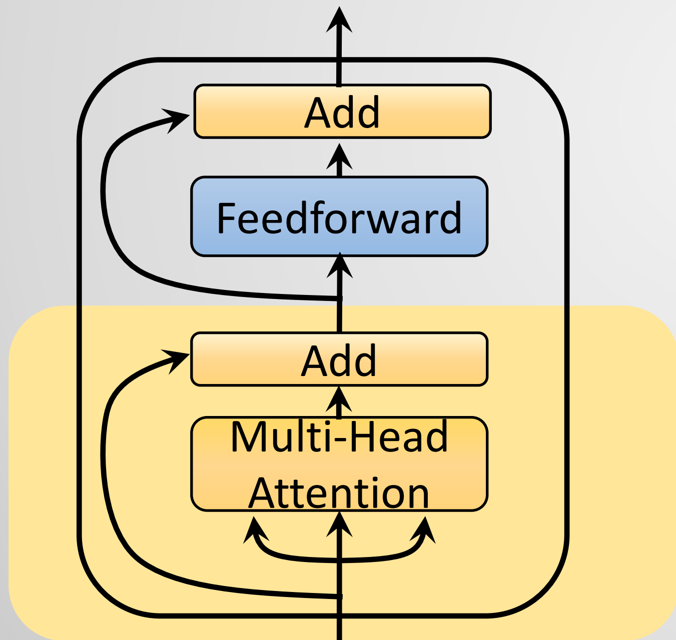
Let embedding at node i be x_i

- Each node has K attention heads
- Each attention head has weight matrices: query weights $\boldsymbol{W}_Q$ used to compute query $q_i = \boldsymbol{W}_Q x_i$, key weights $\boldsymbol{W}_K$ used to compute key $k_i = \boldsymbol{W}_K x_i$ and value weights $\boldsymbol{W}_V$ used to compute $v_i = \boldsymbol{W}_V x_i$.
- Compute $a_{ij} = q_i^T k_j$ with all other nodes j. Compute probability vector p_{ij} for all j using the softmax function $p_{ij} = e^{a_{ij}} / \sum_\ell e^{a_{i\ell}}$. The output of the head is a weighted average of all the values $\sum_j p_{ij} v_j$.



Output of multi-head attention

- Concatenate outputs of K attention heads
- Project back to vector of the same length
- Passed to a feedforward network through a residual operation (added to x_i)



Output of Multi-Head Attention

Projection

Concatenate Across K Heads

Weighted Average

Softmax

$a_{ij} = q_i^T k_j$ ….

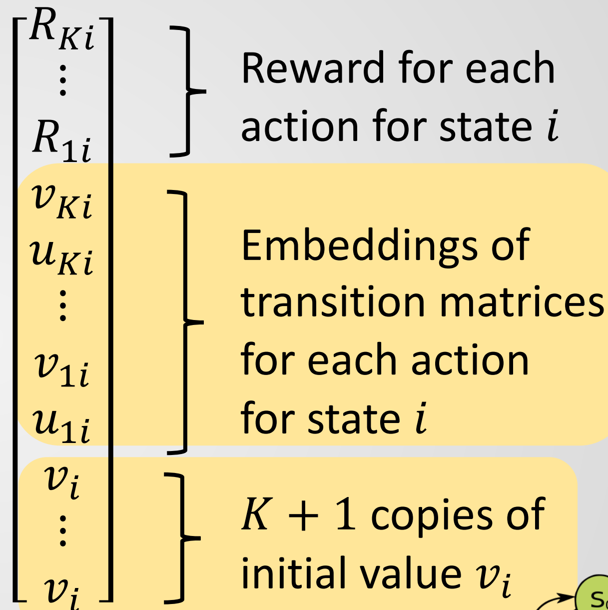
k_j $k_i = \boldsymbol{W}_K x_i$ $v_i = \boldsymbol{W}_V x_i$ $q_i = \boldsymbol{W}_Q x_i$

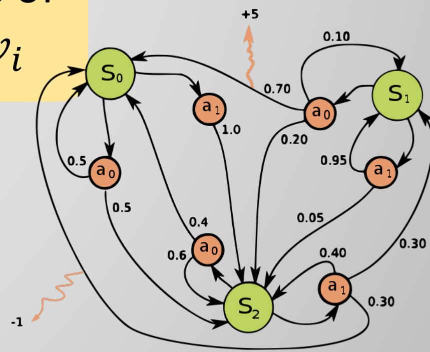
x_j x_i

Alignment of Attention Network with Value Iteration

Constructing input x_i

- Use matrix factorization to get a **distributed representation** of the log of each transition matrix $L_a = U_a V_a^T$ for each action a where $L_a[s, s'] = \log P(s'|s, a)$.
- Construct input for node i, x_i, by stacking up $K + 1$ copies of initial value v_i followed by embeddings of transition matrices



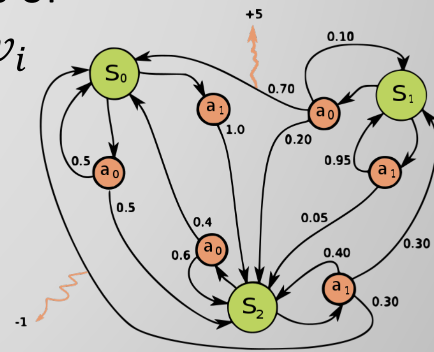


Constructing input x_i

- Compute expected reward $R_{as} = E[R(s,a,s')]$ for each action of the K actions.
- Place expected reward R_{ai} for each action a into a single vector input and concatenate to the earlier input vector.

$$\begin{bmatrix} R_{Ki} \\ \vdots \\ R_{1i} \\ v_{Ki} \\ u_{Ki} \\ \vdots \\ v_{1i} \\ u_{1i} \\ v_i \\ \vdots \\ v_i \end{bmatrix}$$

- $R_{Ki} \ldots R_{1i}$: Reward for each action for state i
- $v_{Ki}, u_{Ki} \ldots v_{1i}, u_{1i}$: Embeddings of transition matrices for each action for state i
- $v_i \ldots v_i$: $K+1$ copies of initial value v_i



At each layer of value iteration:

for each action a of each state s (in parallel)

Collect $p(s'|a,s)(R(s,a,s') + V(s'))$ from all s' to a at s if $p(s'|a,s)$ non-zero

Sum all messages

for each node s (in parallel)

Collect message from its corresponding actions a

Take the maximum of the messages

Need to extract out u_{ai}, v_{aj} to compute

$p(j|a,i) = \exp(u_{ai}^T v_{aj})$

- Setting $W_Q = [0, I_K, 0]$ where I_k is a k by k identity matrix at the appropriate columns will extract $u_{ai} = W_Q x_i$ as query.
- Similarly can construct W_K to extract v_{aj} as key, allowing softmax of inner product $u_{ai}^T v_{aj}$ to correctly compute transition probabilities.
- Set W_V to extract out the value component from x_i

With this construction, output of head a is

$o_a = \sum_j p(j|a,i)\, v_j$

$$\begin{bmatrix} 0 \cdots 0 & & 0 \cdots 0 \\ \vdots \ddots \vdots & I_k & \vdots \ddots \vdots \\ 0 \cdots 0 & & 0 \cdots 0 \end{bmatrix} \begin{bmatrix} R_{Ki} \\ \vdots \\ R_{1i} \\ v_{Ki} \\ u_{Ki} \\ \vdots \\ v_{1i} \\ u_{1i} \\ v_i \\ \vdots \\ v_i \end{bmatrix} = u_{ai}$$

At each layer of value iteration:

for each action a of each state s (in parallel)

Collect $p(s'|a, s)(R(s, a, s') + V(s'))$ from all s' to a at s if $p(s'|a, s)$ non-zero

Sum all messages

for each node s (in parallel)

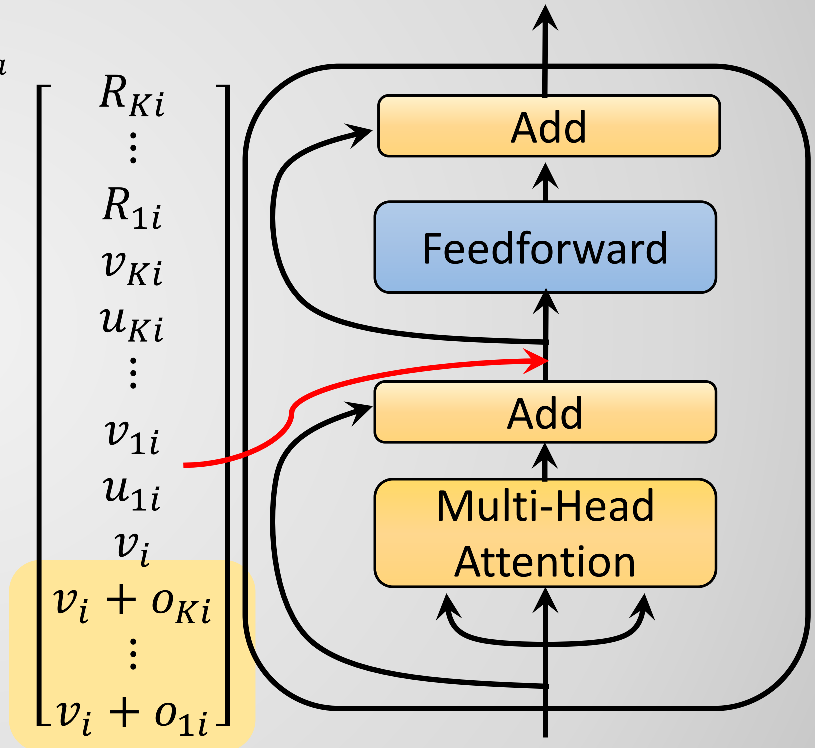
Collect message from its corresponding actions a

Take the maximum of the messages

Output of head a is

$$o_{ai} = \sum_j p(j|a, i)\, v_j$$

Concatenate as $[o_{1i} \ldots o_{Ki}]^T$ and add to first K components of x_i to form input to feedforward network,



At each layer of value iteration:

for each action a of each state s (in parallel)

Collect $p(s'|a, s)(R(s, a, s') + V(s'))$ from all s' to a at s if $p(s'|a, s)$ non-zero

Sum all messages

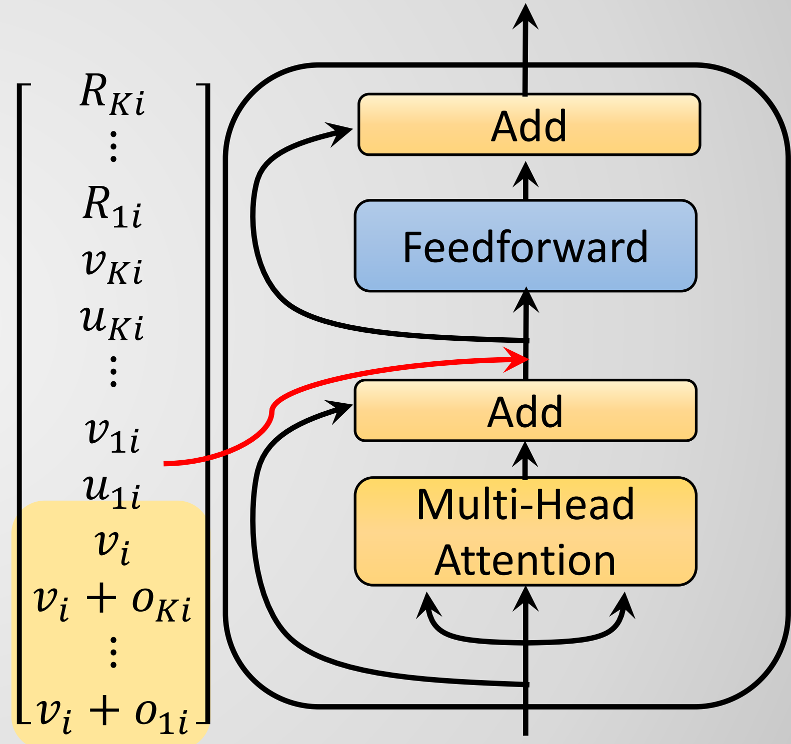
for each node s (in parallel)

Collect message from its corresponding actions a

Take the maximum of the messages

Feedforward network:

- Subtract v_i from $v_i + o_{ai}$ to get o_{ai}
- Compute $v_i' = \max_a\{R_{ai}+o_{ai}\}$
- Construct output so that adding back the input of the feedforward network gets v_i' in the first $K + 1$ positions and the original MDP parameters in the remaining positions:
  output the value $v_i' - (v_i + o_{ai})$ as first K elements and $v_i' - v_i$ as the $K + 1^{\text{st}}$ element,



Learning

Backpropagation and Recurrent Backpropagation

All methods discussed are message passing methods on a graph

- Messages are constructed using operations such as addition, multiplication, max, exponentiation, etc.
- Can be represented using network of computational elements
- Usually same network at each graph node

Each iteration of message passing forms a layer

Putting together layers form a deep network

- Different layers can have different parameters, additional flexibility

With appropriate loss functions, can learn using gradient descent if all network elements are differentiable: backpropagation

If different layers all have the same parameters, we have a recurrent neural network.

For some recurrent networks, the inputs the same at each iteration and we are aiming for solution at convergence

- Loopy belief propagation
- Value iteration for discounted MDP
- Some graph neural networks[1]
- Some tranformer architectures[2]

Recurrent backpropagation and other optimization methods based on implicit functions can be used[3]

- Constant memory usage

[1] Scarselli, Franco, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. "The graph neural network model." IEEE Transactions on Neural Networks, 2008.

[2] Bai, Shaojie, J. Zico Kolter, and Vladlen Koltun. "Deep equilibrium models." NeurIPS 2019

[3] Zico Kolter, David Duvenaud, and Matt Johnson, "Deep Implicit Layers - Neural ODEs, Deep Equilibirum Models, and Beyond." NeurIPS 2020 Tutorial http://implicit-layers-tutorial.org/

Summary

Message Passing in Machine Learning

We viewed some “classic” message passing algorithms in machine learning through variational principles

- Loopy belief propagation
- Mean field
- Value iteration

What optimization problems are they solving?

We relate graph neural networks and attention networks to the “classic” message passing algorithms

- Algorithmic alignment (analysis): Can they simulate those algorithms using a small network?
- Neuralizing algorithms (design): Can we enhance those algorithms into more powerful neural versions

References[1]

- David Mackay's Error Correcting Code demo http://www.inference.org.uk/mackay/codes/gifs/
- Zheng, Shuai, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. "Conditional random fields as recurrent neural networks." In Proceedings of the IEEE international conference on computer vision, pp. 1529-1537. 2015.
- Yedidia, Jonathan S., William T. Freeman, and Yair Weiss. "Constructing free-energy approximations and generalized belief propagation algorithms." *IEEE Transactions on information theory* 51.7 (2005): 2282-2312.
- Wainwright, Martin J., and Michael Irwin Jordan. Graphical models, exponential families, and variational inference. Now Publishers Inc, 2008.

[1] This list contains only references referred to within the slides and not the many other works related to the material in the tutorial.

- Markov Decision Process (figure) by waldoalvarez - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=59364518
- Gilmer, Justin, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. "Neural message passing for quantum chemistry." In International conference on machine learning, pp. 1263-1272. PMLR, 2017.
- Loukas, Andreas. "What graph neural networks cannot learn: depth vs width." ICLR 2020
- Xu, Keyulu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. "How powerful are graph neural networks?" ICLR 2019

- Xu, Keyulu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. "What can neural networks reason about?." ICLR 2020
- Skeletal formula of paracetamol by Benjah-bmm27 is in the public domain, https://en.wikipedia.org/wiki/Paracetamol#/media/File:Paracetamol-skeletal.svg
- Ball and stick model of paracetamol by Ben Mills is in the public domain, https://en.wikipedia.org/wiki/Paracetamol#/media/File:Paracetamol-from-xtal-3D-balls.png
- Tamar, Aviv, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. "Value iteration networks." NeurIPS 2016.
- Lee, Lisa, Emilio Parisotto, Devendra Singh Chaplot, Eric Xing, and Ruslan Salakhutdinov. "Gated path planning networks." ICML 2018.

- Karkus, Peter, David Hsu, and Wee Sun Lee. "QMDP-net: Deep learning for planning under partial observability." NeurIPS 2017.
- Dai, Hanjun, Bo Dai, and Le Song. "Discriminative embeddings of latent variable models for structured data." ICML 2016.
- Dupty, Mohammed Haroon, and Wee Sun Lee. "Neuralizing Efficient Higher-order Belief Propagation." *arXiv preprint arXiv:2010.09283* (2020).
- Zhang, Zhen, Fan Wu, and Wee Sun Lee. "Factor graph neural network." NeurIPS 2020.

- Scarselli, Franco, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. "The graph neural network model." IEEE Transactions on Neural Networks 20, no. 1 (2008): 61-80.
- Bai, Shaojie, J. Zico Kolter, and Vladlen Koltun. "Deep equilibrium models." NeurIPS 2019.
- Zico Kolter, David Duvenaud, and Matt Johnson, “Deep Implicit Layers - Neural ODEs, Deep Equilibirum Models, and Beyond.” NeurIPS 2020 Tutorial http://implicit-layers-tutorial.org/

Appendix

Correctness of Belief Propagation on Trees

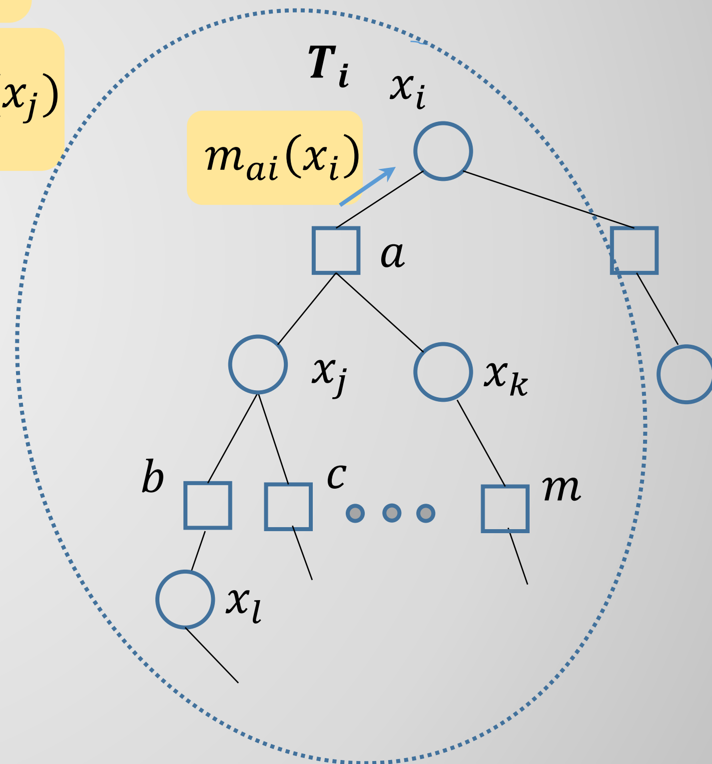
$$n_{ia}(x_i) = \prod_{c \in N(i) \backslash a} m_{ci}(x_i)$$

$$m_{ai}(x_i) = \sum_{\boldsymbol{x}_a \backslash x_i} \psi_a(\boldsymbol{x}_a) \prod_{j \in N(a) \backslash i} n_{ji}(x_j)$$

$$b_{i(x_i)} \propto \prod_{a \in N(i)} m_{ai}(x_i)$$

Assume T_i is the subtree along edge (a, i), rooted at x_i and message $m_{ai}(x_i)$ is sent from a to i.

After k **iterations** of message passing, the message correctly marginalizes away other variables in the subtree
$m_{ai}(x_i) = \sum_{x' \in T_i \backslash x_i} \prod_{a \in T_i} \psi_a(\boldsymbol{x}'_a)$
where k is the height of the subtree.

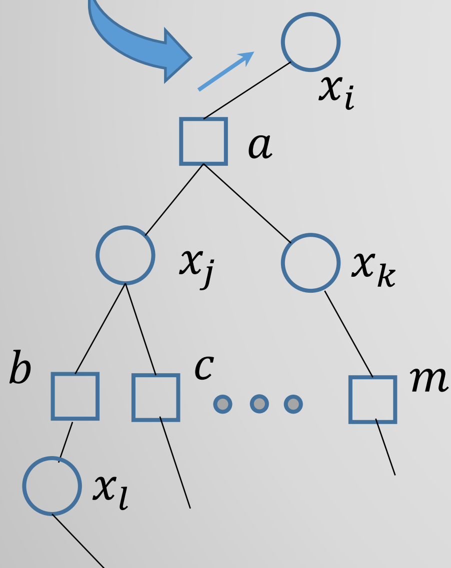


Correctness

Init all messages to all-one vectors.

Then:

$$m_{ai}(x_i) = \sum_{x_j}\sum_{x_k}\sum_{x_l}\sum_{x_{\dots}} \psi_a(x_i, x_j, x_k)\psi_{b(\dots)} \dots$$

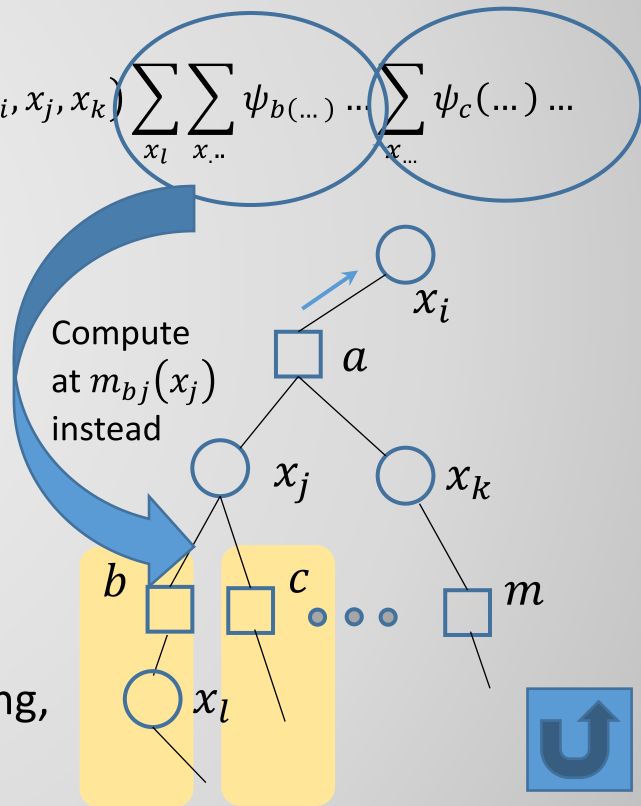


$$m_{ai}(x_i) = \sum_{x_j}\sum_{x_k} \psi_a(x_i, x_j, x_k) \sum_{x_l}\sum_{x_{\dots}} \psi_{b(\dots)} \dots \sum_{x_{\dots}} \psi_c(\dots) \dots$$

Assume marginalization correctly done for subtrees of height < k

- Push summation inside product, can group at subtrees depth 2 below a because of tree structure
- By inductive hypothesis, marginalization correct at depth 2 below

So after k iterations of message passing, marginalization at height k correct



Mean Field Update Derivation

In mean field approximation, we find q that minimizes the variational free energy

$F(q) = \sum_x q(\boldsymbol{x}) E(\boldsymbol{x}) + \sum_x q(\boldsymbol{x}) \ln q(\boldsymbol{x})$

when q is restricted to a factored form

$q_{MF}(x) = \prod_{i=1}^{N} q_i(x_i)$.

Consider the dependence on a single variable $q_j(x_j)$ with all other variables fixed

$$F(q) = \sum_{\boldsymbol{x}} \prod_{i=1}^{N} q_i(x_i)\, E(\boldsymbol{x}) + \sum_{\boldsymbol{x}} \prod_{i=1}^{N} q_i(x_i) \ln \prod_{i=1}^{N} q_i(x_i)$$

$$= \sum_{x_j} q_j(x_j) \sum_{\boldsymbol{x} \backslash x_j} \prod_{i \neq j}^{N} q_i(x_i)\, E(\boldsymbol{x}) + \sum_{x_j} q_j(x_j) \ln q_j(x_j) + const$$

$$= \sum_{x_j} q_j(x_j) \ln \frac{1}{p_j'(x_j)} + const + \sum_{x_j} q_j(x_j) \ln q_j(x_j) + const$$

where $p_j'(x_j) = \frac{1}{Z_j'} \exp\left(-\sum_{\boldsymbol{x} \backslash x_j} \prod_{i \neq j}^{N} q_i(x_i)\, E(\boldsymbol{x}) \right)$

$$= KL(q_j || p_j') + const$$

which is minimized at $q_j(x_j) = p_j'(x_j)$.

Variational Free Energy for Trees

We can write a tree distribution as

$$\frac{\prod_a q_a(\boldsymbol{x}_a)\prod_i q_i(x_i)}{\prod_a \prod_{i\in N(a)} q_i(x_i)}$$

To see this, we can write a tree distribution as

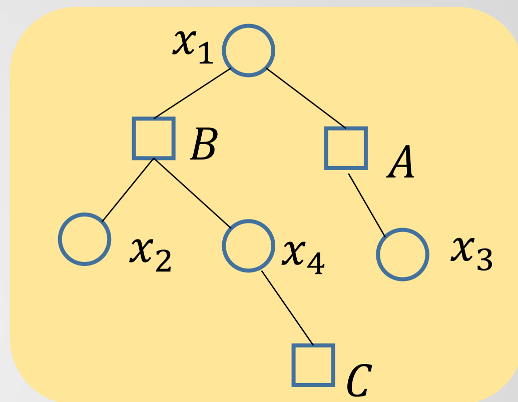
$$q_r(x_r)\prod_a q_a(\boldsymbol{x}_a|pa(\boldsymbol{x}_a))$$

where x_r is the root, and $pa(\boldsymbol{x}_a)$ is the parent variable for factor a in the tree

We write

$$q_a(\boldsymbol{x}_a|pa(\boldsymbol{x}_a)) = \frac{q_a(\boldsymbol{x}_a)}{q_{pa(\boldsymbol{x}_a)}}$$

Each variable appears as a parent $d-1$ times, where d is the degree of the variable node, except the root which appears d times.



As $q_r(x_r)$ also appears on the numerator, we can write the distribution as

$$\frac{\prod_a q_a(\boldsymbol{x}_a)}{\prod_i q_i(x_i)^{d-1}}$$

Multiplying the numerator and denominator by $\prod_i q_i(x_i)$ and observing that $\prod_a \prod_{i\in N(a)} q_i(x_i) = \prod_i q_i(x_i)^d$ gives the required expression

Substituting

$$q(\boldsymbol{x}) = \frac{\prod_a q_a(\boldsymbol{x}_a) \prod_i q_i(x_i)}{\prod_a \prod_{i \in N(a)} q_i(x_i)}$$

into

$$F(q) = \sum_x q(\boldsymbol{x}) E(\boldsymbol{x}) + \sum_x q(\boldsymbol{x}) \ln q(\boldsymbol{x})$$

we get

$$F(q) = -\sum_{a=1}^{M} \sum_{\boldsymbol{x}_a} q_a(\boldsymbol{x}_a) \ln \psi_a(\boldsymbol{x}_a) + \sum_{a=1}^{M} \sum_{\boldsymbol{x}_a} q_a(\boldsymbol{x}_a) \ln \frac{q_a(\boldsymbol{x}_a)}{\prod_{i \in N(a)} q_i(x_i)} + \sum_{i=1}^{N} \sum_{x_i} q_i(x_i) \ln q_i(x_i)$$

Belief Propagation as Stationary Point of Bethe Free Energy

Consider optimizing Bethe free energy subject to constraints described. We form the Lagrangian

$$L = F_{Bethe} + \sum_a \gamma_a [1 - \sum_{\boldsymbol{x}_a} q_a(\boldsymbol{x}_a)] + \sum_i \gamma_i [1 - \sum_{x_i} q_i(x_i)]$$

$$+ \sum_i \sum_{a \in N(i)} \sum_{x_i} \lambda_{ai}(x_i) [q_i(x_i) - \sum_{\boldsymbol{x}_a \setminus x_i} q_a(\boldsymbol{x}_a)]$$

Differentiating with respect to $q_i(x_i)$ and setting to 0

$$0 = -d_i + 1 + \ln q_i(x_i) - \gamma_i + \sum_{a \in N(i)} \lambda_{ai}(x_i)$$

$$q_i(x_i) \propto \exp\left(-\sum_{a \in N(i)} \lambda_{ai}(x_i)\right) = \prod_{a \in N(i)} m_{ai}(x_i)$$

where d_i is the degree of node and $m_{ai}(x_i) = \exp(-\lambda_{ai}(x_i))$

$$F_{Bethe}(q) = U_{Bethe}(q) - H_{Bethe}(q)$$

$$U_{Bethe}(q) = -\sum_{a=1}^{M} \sum_{x_a} q_a(\boldsymbol{x}_a) \ln \psi_a(\boldsymbol{x}_a)$$

$$\mathrm{H}_{Bethe} = -\sum_{a=1}^{M} \sum_{\boldsymbol{x}_a} q_a(\boldsymbol{x}_a) \ln \frac{q_a(\boldsymbol{x}_a)}{\prod_{i \in N(a)} q_i(x_i)} - \sum_{i=1}^{N} \sum_{x_i} q_i(x_i) \ln q_i(x_i)$$

Consider optimizing Bethe free energy subject to constraints described

$$L = F_{Bethe} + \sum_a \gamma_a[1 - \sum_{\boldsymbol{x}_a} q_a(\boldsymbol{x}_a)] + \sum_i \gamma_i[1 - \sum_{x_i} q_i(x_i)]$$

$$+ \sum_i \sum_{a \in N(i)} \sum_{x_i} \lambda_{ai}(x_i)[q_i(x_i) - \sum_{\boldsymbol{x}_a \setminus x_i} q_a(\boldsymbol{x}_a)]$$

Differentiating with respect to $q_a(\boldsymbol{x}_a)$ and setting to 0

$$0 = -\ln \psi_a(\boldsymbol{x}_a) + 1 + \ln q_a(\boldsymbol{x}_a)$$

$$- \ln \prod_{i \in N(a)} q_i(x_i) - \gamma_a - \sum_{i \in N(a)} \lambda_{ai}(x_i)$$

$$= -\ln \psi_a(\boldsymbol{x}_a) + \ln q_a(\boldsymbol{x}_a) + const$$

$$+ \sum_{i \in N(a)} \sum_{a \in N(i)} \lambda_{ai}(x_i) - \sum_{i \in N(a)} \lambda_{ai}(x_i)$$

$$q_a(\boldsymbol{x}_a) \propto \psi_a(\boldsymbol{x}_a) \exp\left(-\sum_{i \in N(a)} \sum_{b \in N(i) \setminus a} \lambda_{bi}(x_i)\right)$$

$$= \psi_a(x_a) \prod_{i \in N(a)} \prod_{b \in N(i) \setminus a} m_{bi}(x_i)$$

where $m_{bi}(x_i) = \exp(-\lambda_{bi}(x_i))$

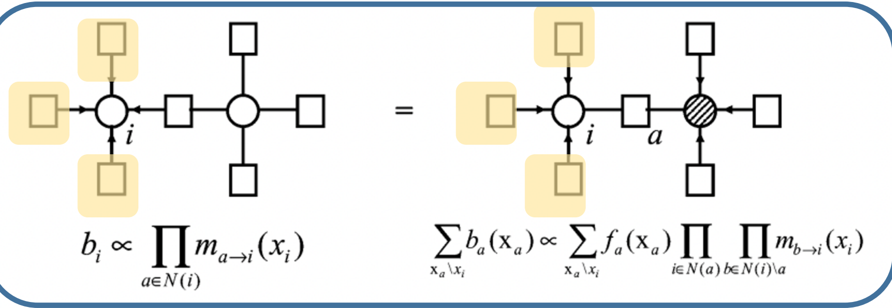
$$F_{Bethe}(q) = U_{Bethe}(q) - H_{Bethe}(q)$$

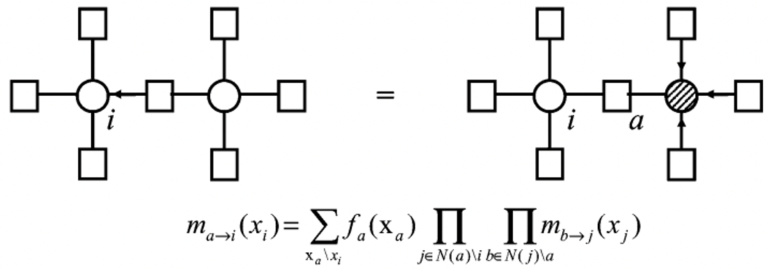
$$U_{Bethe}(q) = -\sum_{a=1}^{M} \sum_{\boldsymbol{x}_a} q_a(\boldsymbol{x}_a) \ln \psi_a(\boldsymbol{x}_a)$$

$$\mathrm{H}_{Bethe} = -\sum_{a=1}^{M} \sum_{\boldsymbol{x}_a} q_a(\boldsymbol{x}_a) \ln \frac{q_a(\boldsymbol{x}_a)}{\prod_{i \in N(a)} q_i(x_i)} - \sum_{i=1}^{N} \sum_{x_i} q_i(x_i) \ln q_i(x_i)$$

Previously:

$$q_i(x_i) \propto \exp\left(-\sum_{a \in N(i)} \lambda_{ai}(x_i)\right)$$





Previously:

$$q_i(x_i) = \prod_{b \in N(i)} m_{bi}(x_i)$$

$$q_a(\boldsymbol{x_a}) = \psi_a(x_a) \prod_{j \in N(a)} \prod_{b \in N(j) \setminus a} m_{bj}(x_j)$$

[Fig from Yedidia et. al. 2005]

For $\sum_{x_a \setminus i} q_a(x_a)$ to be consistent with $q_i(x_i) = \prod_{b \in N(i)} m_{bi}(x_i)$, we get the belief propagation equation

$$m_{ai}(x_i) = \sum_{x_a \setminus i} \psi_a(x_a) \prod_{j \in N(a) \setminus i} \prod_{b \in N(j) \setminus a} m_{bj}(x_j)$$

Stationary points of the Bethe free energy are fixed points of loopy belief propagation updates

Bellman Update Contraction

We show that Bellman update is a contraction

$$||BV_1 - BV_2|| \leq \gamma ||V_1 - V_2||$$

First we show that for any two functions $f(a)$ and $g(a)$

$$|\max_a f(a) - \max_a g(a)| \leq \max_a |f(a) - g(a)|$$

To see this, assume consider the case where $\max_a f(a) \geq \max_a g(a)$. Then

$$
\begin{aligned}
|\max_a f(a) - \max_a g(a)| &= \max_a f(a) - \max_a g(a) \\
&= f(a^*) - \max_a g(a) \\
&\leq f(a^*) - g(a^*) \\
&\leq \max_a |f(a) - g(a)|
\end{aligned}
$$

where $a^* = argmax_a\, f(a)$. The case where $\max_a g(a) \geq \max_a f(a)$ is similar.

Now, for any state s

$$|BV_1(s) - BV_2(s)|$$

$$= \left| max_a \left\{ \sum_{s'} P(s'|s,a)\big(R(s,a,s') + \gamma V_1(s')\big) \right\} - max_a \left\{ \sum_{s'} P(s'|s,a)\big(R(s,a,s') + \gamma V_2(s')\big) \right\} \right|$$

$$\leq max_a \left| \sum_{s'} P(s'|s,a)\big(R(s,a,s') + \gamma V_1(s')\big) - \sum_{s'} P(s'|s,a)\big(R(s,a,s') + \gamma V_2(s')\big) \right|$$

$$= \gamma max_a \left| \sum_{s'} P(s'|s,a)(V_1(s') - V_2(s')) \right|$$

$$= \gamma \left| \sum_{s'} P(s'|s,a^*)(V_1(s') - V_2(s')) \right|$$

where we have use $|\max_a f(a) - \max_a g(a)| \leq \max_a |f(a) - g(a)|$.

Finally, we show contraction

$$
\begin{aligned}
\|BV_1 - BV_2\| &= \max_s |BV_1(s) - BV_2(s)| \\
&\leq \gamma \max_s \left| \sum_{s'} P(s'|s, a^*)\big(V_1(s') - V_2(s')\big) \right| \\
&\leq \gamma \max_s |V_1(s) - V_2(s)| \\
&= \gamma \|V_1 - V_2\|
\end{aligned}
$$