

Probabilistic Logic and Deep Learning

Timothy Zhang

Stony Brook University

Research Proficiency Examination

Agenda

- ▶ **Deep Learning**
- ▶ Logical Student-Teacher Network
- ▶ TensorLog
- ▶ Conclusion

Deep Neural Networks

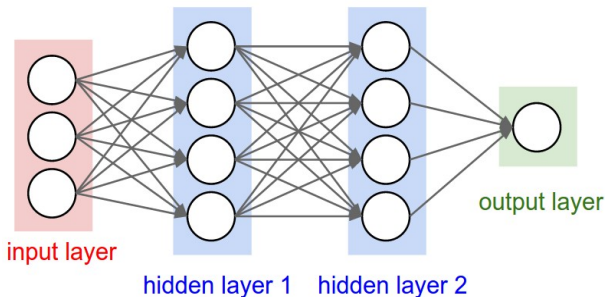


Figure: A two layer neural network $g(\mathbf{x}) = o(h_2(h_1(\mathbf{x})))$.

- ▶ $\mathbf{x} \in \mathbb{R}^3, \mathbf{y} \in \mathbb{R}$
- ▶ $h_1(\mathbf{x}) = \phi_1(W_1\mathbf{x} + b_1)$
- ▶ $h_2(h_1(\mathbf{x})) = \phi_2(W_2h_1(\mathbf{x}) + b_2)$
- ▶ $o(h_2(h_1(\mathbf{x}))) = \psi(W_3\phi_2(\phi_1(\mathbf{x})) + b_3)$

Deep Neural Networks

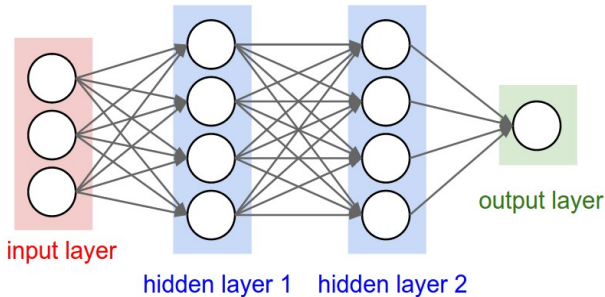


Figure: A two layer neural network $g(\mathbf{x}) = o(h_2(h_1(x)))$.

- ▶ Each ϕ_i is a nonlinear function applied element-wise to the hidden layer output.
- ▶ Ex: tanh, ReLu, logistic sigmoid
- ▶ ψ_j is a function applied to the output layer output.
- ▶ Ex: identity, softmax

Deep Neural Networks

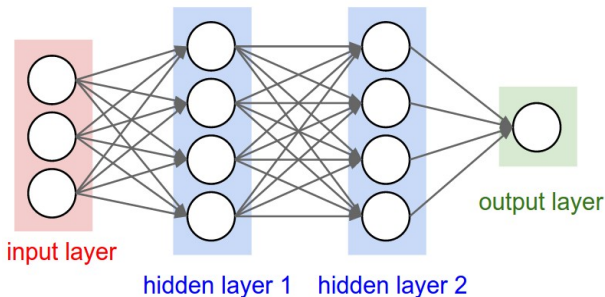


Figure: A two layer neural network $g(\mathbf{x}) = o(h_2(h_1(x)))$.

- ▶ A loss (objective, reward, etc) function is defined with respect to the output of the network and the ground truth label.
- ▶ Ex: MSE is $\sum_i (\mathbf{y}^{(i)} - g(\mathbf{x}^{(i)}))^2$

Deep Neural Networks

Data: Training data $\mathcal{S} = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^N$,

Hyperparameters: η learning rate

Result: θ^* which minimizes \mathcal{L}

Initialize DNN parameters θ

while \neg *converged* **do**

 Sample a minibatch $(X, Y) \subset \mathcal{S}$

 Set $g \leftarrow 0$

for $(\mathbf{x}, \mathbf{y}) \in (X, Y)$ **do**

 Compute gradient: $g \leftarrow g + \nabla_{\theta} \mathcal{L}(f_{\theta}(\mathbf{x}), \mathbf{y}; \theta)$

end

 Apply update: $\theta \leftarrow \theta - \eta g$

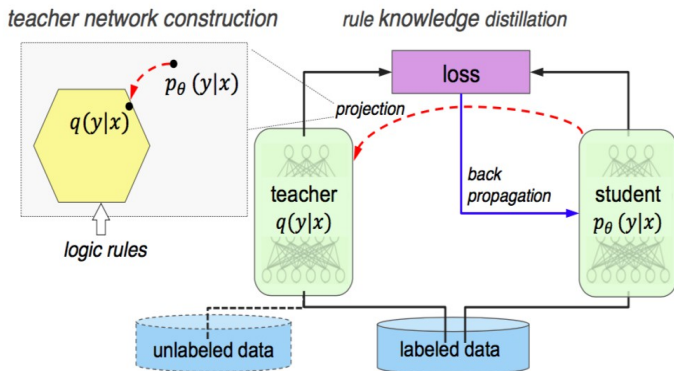
end

Algorithm 1: Stochastic Gradient Descent

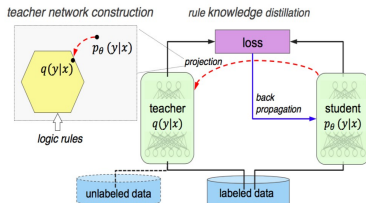
Agenda

- ▶ Deep Learning
- ▶ **Logical Student-Teacher Network**
- ▶ TensorLog
- ▶ Conclusion

Architecture



Loss Function



$$\theta^{(t+1)} = \underset{\theta \in \Theta}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^N (1-\pi) \mathcal{L}(\mathbf{y}_n, \sigma_\theta(\mathbf{x}_n)) + \pi \mathcal{L}(s_n^{(t)}, \sigma_\theta(\mathbf{x}_n)) \quad (1)$$

- ▶ $\mathbf{x}_n, \mathbf{y}_n$: n th sample
- ▶ \mathcal{L} is an arbitrary loss (assume cross-entropy)
- ▶ π is the mixture weight between student and teacher networks
- ▶ $\sigma_\theta(\mathbf{x}_n)$ is the output from the student network (assumed to be a softmax vector)
- ▶ $s_n^{(t)}$: teacher network output at iteration t

Logic Rules

- ▶ $\mathcal{R} = \{(R_l, \lambda_l)\}_{l=1}^L$ where each R_l is a first-order logic formula with an associated confidence $\lambda_l \in [0, \infty]$.
- ▶ These rules are encoded using a continuous logic called soft logic which allows continuous truth values $\in [0, 1]$ with the following semantics:

$$\begin{aligned}\neg A &= 1 - A \\ A \& B &= \max\{A + B - 1, 0\} \\ A \vee B &= \min\{A + B, 1\} \\ A_1 \wedge \dots \wedge A_N &= \sum_i A_i / N\end{aligned}\tag{2}$$

Teacher Network Construction

Loss function

$$\begin{aligned} \min_{q, \xi \geq 0} & \text{KL}(q(Y|X) || p_{\theta}(Y|X)) + C \sum_{l, g_l} \xi_{l, g_l} \\ \text{s.t. } & \lambda_l (1 - \mathbf{E}_q[r_{l, g_l}(X, Y)]) \leq \xi_{l, g_l} \\ & g_l = 1, \dots, G_l, l = 1, \dots, L. \end{aligned} \tag{3}$$

- ▶ Intuitively, we want the teacher network distribution q to be close to the student network p_{θ} , so we use a KL term.
- ▶ Additionally we want the soft logic rules to approximately hold, so we use an expectation term in the constraints and introduce slack variables ξ .

Teacher Network Construction

Dual Lagrangian

Rewrite (3) in standard form.

$$\begin{aligned} \min_{q, \xi} \quad & \text{KL}(q(Y|X) || p_{\theta}(Y|X)) + C \sum_{l, g_l} \xi_{l, g_l} \\ \text{s.t.} \quad & \lambda_l (1 - \mathbf{E}_q[r_{l, g_l}(X, Y)]) - \xi_{l, g_l} \leq 0 \\ & -\xi_{l, g_l} \leq 0 \\ & \sum_Y q(Y|X) - 1 = 0 \\ & g_l = 1, \dots, G_l, l = 1, \dots, L \end{aligned}$$

Then add Lagrange multipliers and simplify (algebra)

$$\begin{aligned} \max_{\eta \geq 0, \mu \geq 0, \alpha \geq 0} \min_{q, \xi} \quad & \text{KL}(q(Y|X) || p_{\theta}(Y|X)) + C \sum_{l, g_l} \xi_{l, g_l} \\ & + \sum_{l, g_l} \eta_{l, g_l} (\mathbf{E}_q[\lambda_l (1 - r_{l, g_l}(X, Y))] - \xi_{l, g_l}) - \sum_{l, g_l} \mu_{l, g_l} \xi_{l, g_l} + \alpha \left(\sum_Y q(Y|X) - 1 \right) \end{aligned}$$

Teacher Network Construction

Teacher Construction Rule

By solving the dual Lagrangian we obtain the teacher network construction rule:

$$q^*(Y|X) \propto p_\theta(Y|X) \exp \left\{ - \sum_{l, g_l} C \lambda_l (1 - r_{l, g_l}(X, Y)) \right\}. \quad (4)$$

- ▶ $q^*(Y|X)$ can be computed efficiently for a given example \mathbf{x}
- ▶ We simply feed \mathbf{x} as input to the student network and use the output vector $\sigma_\theta(\mathbf{x})$ as $p_\theta(Y|X)$ in the above expression.
- ▶ Thus an inference step in the teacher network will have complexity of the order $O(lg_l + \mathcal{I})$ where \mathcal{I} is the complexity of inference in the student network.

Learning Algorithm

Data: Training data $\mathcal{S} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$,
Rule set $\mathcal{R} = \{(R_l, \lambda_l)\}_{l=1}^L$,
Hyperparameters: π imitation parameter,
 C regularization strength

Result: Trained networks p_θ and q

Initialize DNN parameters θ

while \neg *converged* **do**

 Sample a minibatch $(X, Y) \subset \mathcal{S}$

 Construct teacher network q using (4)

 Update θ using (1)

end

Algorithm 2: Teacher and Student Network Training

Sentence Level Sentiment Analysis

Student Network Architecture

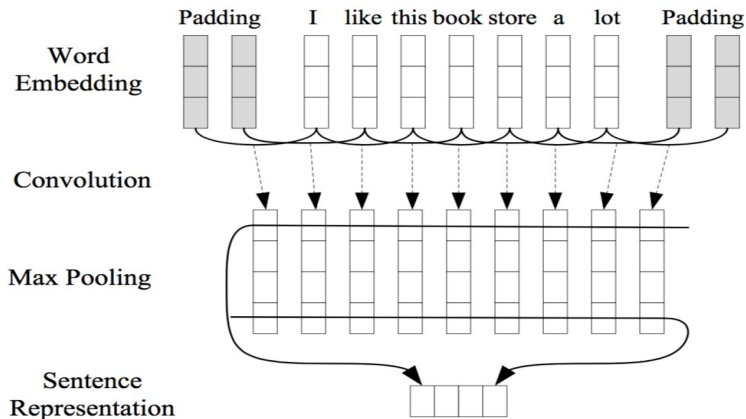


Figure: CNN architecture using Word2Vec vectors.

Sentence Level Sentiment Analysis

Logic Rule

$$\text{has-A-but-B-structure}(S) \Rightarrow \\ (\mathbf{1}(y = +) \Rightarrow \sigma_{\theta}(B)_{+} \wedge \sigma_{\theta}(B)_{+} \Rightarrow \mathbf{1}(y = +)),$$

- ▶ Intuitively, if a sentence has a "but" in it we should take the sentiment of the subsentence after the "but".
- ▶ "At first I thought the movie was great but it turned out to be derivative and boring."

Sentence Level Sentiment Analysis

Results

	Model	SST2	MR	CR
1	CNN (Kim, 2014)	87.2	81.3±0.1	84.3±0.2
2	CNN-Rule- <i>p</i>	88.8	81.6±0.1	85.0±0.3
3	CNN-Rule- <i>q</i>	89.3	81.7±0.1	85.3±0.3
4	MGNC-CNN (Zhang et al., 2016)	88.4	–	–
5	MVCNN (Yin and Schutze, 2015)	89.4	–	–
6	CNN-multichannel (Kim, 2014)	88.1	81.1	85.0
7	Paragraph-Vec (Le and Mikolov, 2014)	87.8	–	–
8	CRF-PR (Yang and Cardie, 2014)	–	–	82.7
9	RNTN (Socher et al., 2013)	85.4	–	–
10	G-Dropout (Wang and Manning, 2013)	–	79.0	82.1

Agenda

- ▶ Deep Learning
- ▶ Logical Student-Teacher Network
- ▶ **TensorLog**
- ▶ Conclusion

Probabilistic Logic Programming (in PRISM)

```
values(gene,[a,b,o]).
```

```
bloodtype(P) :-  
    genotype(X,Y),  
    ( X=Y -> P=X  
    ; X=o -> P=Y  
    ; Y=o -> P=X  
    ; P=ab  
    ).
```

```
genotype(X,Y) :- msw(gene,X),msw(gene,Y).
```

```
?- learn([count(bloodtype(a),40),count(bloodtype(b),20),  
count(bloodtype(o),30),count(bloodtype(ab),10)]).
```

Logical Inference and Neural Networks

- ▶ Logical specifications allow one to model aspects of the world and perform **logical inference**.
- ▶ In Prolog this is done using SLD-resolution, which is a search procedure over proofs for a query.
- ▶ Discrete search is not a differentiable function, and thus cannot directly be integrated in gradient-based learning systems.

TensorLog

- ▶ TensorLog proposes a differentiable (probabilistic) logical inference procedure.
- ▶ This algorithm can be seen as an instance of the Belief Propagation algorithm for probabilistic graphical models.
- ▶ Thus TensorLog offers a method of incorporating traditional logical inference as a component of a DNN.
- ▶ Alternatively, TensorLog offers a method of incorporating DNNs for traditional logical inference.

Example TensorLog Model

```
% Theory T
uncle(X, Y) :- child(X, Z), brother(Z, Y).
uncle(X, Y) :- aunt(X, W), husband(W, Y).
status(X, tired) :- child(W, X), infant(W).

% Knowledge Base DB
0.99 :: child(liam, eve).      0.7 :: infant(liam).
0.99 :: child(dave, eve).     0.1 :: infant(dave).
0.75 :: child(liam, bob).     0.9 :: aunt(joe, eve).
0.9 :: husband(eve, bob).     0.9 :: brother(eve, chip).
```

Figure: Example \mathcal{DB} and \mathcal{T} .

Logical Preliminaries

- ▶ A **database** $\mathcal{DB} = \{f_1, \dots, f_N\}$ where each f_i is a ground **fact**.
- ▶ A fact is of the form $p(a, b)$ or $q(c)$ where p and q are predicate symbols and $a, b, c \in \mathcal{C}$ are constant symbols from domain \mathcal{C} .
- ▶ A theory \mathcal{T} , is a set of function-free Horn clauses.
- ▶ The least model for $(\mathcal{DB}, \mathcal{T})$ is written $Model(\mathcal{DB}, \mathcal{T})$.
- ▶ A fact f is true iff $f \in Model(\mathcal{DB}, \mathcal{T})$

Probabilistic Logical Preliminaries

- ▶ Θ is a parameter vector over the facts $f \in \mathcal{DB}$ s.t. $\theta_f \in [0, 1]$.
- ▶ The semantics of this parameter vary in different probabilistic deductive database models
- ▶ Θ defines a distribution $Pr(f|\mathcal{T}, \mathcal{DB}, \Theta)$ over facts in $Model(\mathcal{T}, \mathcal{DB})$.

TensorLog Logical Restrictions

- ▶ TensorLog is function free.
- ▶ Predicates in TensorLog have arity at most 2.
- ▶ Queries are of the form $q(a, X) \leftarrow b_1(a, c), \dots, b_k(z, b)$. or $q(X, b) \leftarrow b_1(a, c), \dots, b_k(z, b)$.
- ▶ Thus we are interested in all relationships between some constant a and all other constants b s.t. $q(a, X)[X/b]$ holds in the theory.
- ▶ TensorLog only models restricted Datalog programs of this form and does so using matrix representations of predicates and constants.

TensorLog Matrix Representation

- ▶ It is assumed that there is some arbitrary order on $c \in C$ corresponding to indices $[1, \dots, |C|]$ which holds in all matrices and vectors.
- ▶ Constants $c \in C$ are represented using a one-hot row-vector $\mathbf{u} \in \mathbf{R}^{|C|}$.
- ▶ Binary predicates are represented as a sparse matrix \mathbf{M}_r where $\mathbf{M}_r[i, j] = \theta_{r(i, j)}$ if $r(i, j) \in \mathcal{DB}$, and 0 otherwise.
- ▶ A unary predicate q is represented analogously as a row-vector \mathbf{v}_q .

TensorLog Matrix Representation Example

% Knowledge Base DB

```
0.99 :: child(liam, eve).      0.7 :: infant(liam).
0.99 :: child(dave, eve).      0.1 :: infant(dave).
0.75 :: child(liam, bob).       0.9 :: aunt(joe, eve).
0.9  :: husband(eve, bob).      0.9 :: brother(eve, chip).
```

When the order is $[liam, eve, dave, bob, joe, chip]$,

$$\mathbf{M}_{child} = \begin{bmatrix} \theta_{child(liam,liam)} & \theta_{child(liam,eve)} & \cdots & \theta_{child(liam,chip)} \\ \theta_{child(eve,liam)} & \theta_{child(eve,eve)} & \cdots & \theta_{child(eve,chip)} \\ \cdots & \cdots & \cdots & \cdots \\ \theta_{child(chip,liam)} & \theta_{child(chip,eve)} & \cdots & \theta_{child(chip,chip)} \end{bmatrix}$$

$$\mathbf{v}_{infant} = [0.7 \quad 0 \quad 0.1 \quad 0 \quad 0 \quad 0]$$

We will use matrix multiplication to implement logical inference.

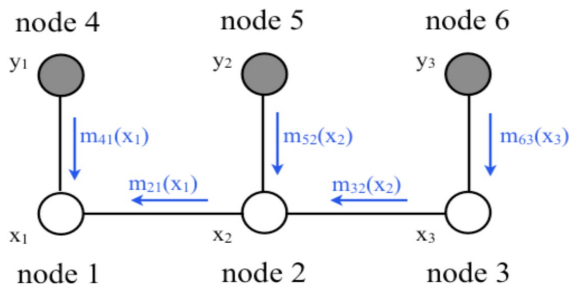
Probabilistic Graphical Models I

- ▶ A probabilistic graphical model (PGM) represents the joint probability distribution of a set of random variables $Pr(X)$.
- ▶ Naively the joint factorizes as $Pr(X = x_1, x_2, \dots, x_n) = Pr(x_1) \times Pr(x_2|x_1) \times \dots \times Pr(x_n|x_{n-1}, x_{n-2}, \dots, x_2, x_1)$
- ▶ Generally, we want factorizations which minimize the number of parameters. We achieve this by introducing conditional independence relations.
- ▶ Ex: Bayesian networks (directed graphical models) encode the factorization as

$$p(X) = \prod_i p(x_i | \pi(x_i))$$

where $\pi(x) = \{x' : (x', x) \in E\}$.

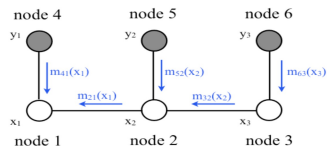
Probabilistic Graphical Models II



Markov chain with three observed variables y_1, y_2, y_3 and three hidden variables x_1, x_2, x_3 . The factorization of this example is:

$$p(X, Y) = \frac{1}{Z} \psi_{12}(x_1, x_2) \psi_{23}(x_2, x_3) \phi_1(x_1, y_1) \phi_2(x_2, y_2) \phi_3(x_3, y_3).$$

Probabilistic Graphical Models III

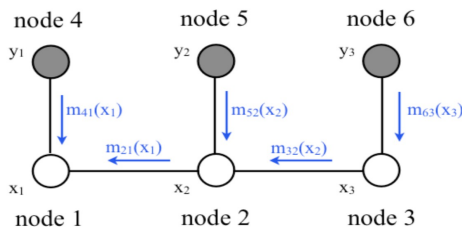


- ▶ In practice we will be interested in using a PGM to perform inference over some subset of the random variables; ie. computing the probability of the subset $p(X' \subseteq X)$.
- ▶ Assume that we wish to compute the marginal probability of x_1 given Y :

$$p(x_1|Y) = \frac{1}{p(Y)} \sum_{x_2} \sum_{x_3} p(X, Y),$$

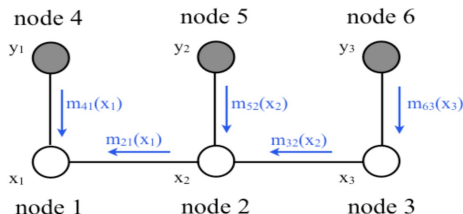
where the equation is due to the fact that $p(X|Y) = \frac{p(X,Y)}{p(Y)}$.

Belief Propagation I



$$\begin{aligned}
 p(x_1|Y) &= \frac{1}{p(Y)} \sum_{x_2} \sum_{x_3} p(X, Y) \\
 &= \frac{1}{p(Y)} \sum_{x_2} \sum_{x_3} \psi_{12}(x_1, x_2) \psi_{23}(x_2, x_3) \phi_1(x_1, y_1) \phi_2(x_2, y_2) \phi_3(x_3, y_3). \\
 &= \frac{1}{p(Y)} \phi_1(x_1, y_1) \sum_{x_2=x_2} \psi_{12}(x_1, x_2) \phi_2(x_2, y_2) \sum_{x_3=x_3} \psi_{23}(x_2, x_3) \phi_3(x_3, y_3) \\
 &= \frac{1}{p(Y)} m_{41}(x_1) \sum_{x_2=x_2} \psi_{12}(x_1, x_2) m_{52}(x_2) \sum_{x_3=x_3} \psi_{23}(x_2, x_3) m_{63}(x_3)
 \end{aligned}$$

Belief Propagation II



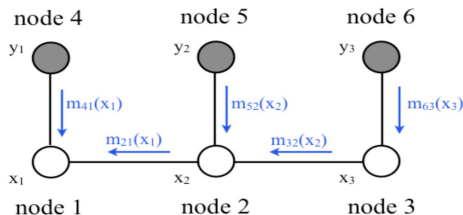
$$\frac{1}{p(Y)} m_{41}(x_1) \sum_{X_2=x_2} \psi_{12}(X_1, x_2) m_{52}(x_2) \sum_{X_3=x_3} \psi_{23}(x_2, x_3) m_{63}(x_3)$$

Let's take a closer look at these messages.

- ▶ $m_{41}(x_1) = \phi_1(x_1, y_1)$
- ▶ $m_{52}(x_2) = \phi_2(x_2, y_2)$
- ▶ $m_{63}(x_3) = \phi_3(x_3, y_3)$

All of these messages are scalars which are known.

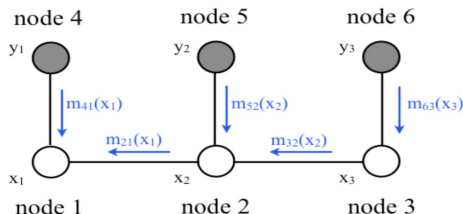
Belief Propagation III



$$\begin{aligned} & \frac{1}{p(Y)} m_{41}(x_1) \sum_{X_2=x_2} \psi_{12}(X_1, x_2) m_{52}(x_2) \sum_{X_3=x_3} \psi_{23}(x_2, x_3) m_{63}(x_3) \\ &= \frac{1}{p(Y)} m_{41}(x_1) \sum_{X_2=x_2} \psi_{12}(X_1, x_2) m_{52}(x_2) m_{32}(x_2) \\ &= \frac{1}{p(Y)} m_{41}(x_1) m_{21}(x_1). \end{aligned}$$

Finally, if we were to compute say $p(x_2|Y)$ we would be able to reuse many of these messages.

Belief Propagation IV

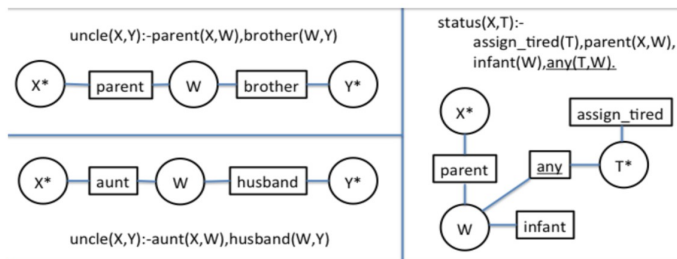


- ▶ BP for undirected graphical models is defined as follows:
 1. Convert the PGM to an equivalent PGM with only pairwise potentials.
 2. Compute $m_{ji}(x_i)$ as:

$$m_{ji}(x_i) = \sum_{x_j} \psi_{ij}(x_i, x_j) \prod_{k \in N(j) \setminus i} m_{kj}(x_j).$$

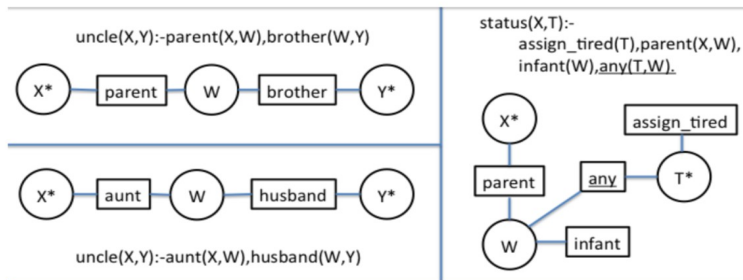
- ▶ In English: The message from some x_j to x_i is computed as the sum over products of all neighbors of n_j (except n_i) and then weighted by the "compatibility" between n_j and n_i .

TensorLog Factor Graphs I



- ▶ TensorLog converts rules $r \in \mathcal{T}$ into factor graphs G_r .
- ▶ A factor graph is like an undirected graphical model which explicitly represents factors as nodes.
- ▶ Predicate symbols become factors which have edges connected to logical variables which are the arguments of the predicate.
- ▶ The matrix of a factor node for predicate p is exactly \mathbf{M}_p .

TensorLog Factor Graphs II



- ▶ Special any/2 and assign/1 predicates are introduced.
- ▶ any/2 holds for any two constants in the language with probability 1. It is used to connect disjoint factor graphs.
- ▶ assign/1 is a syntactic restriction which is “algorithmically convenient”.

TensorLog Inference I

G_r defines a distribution over possible groundings of the variables in r . Let X_1, \dots, X_m be the variables in r . Then:

$$Pr_{G_r}(X_1 = c_1, \dots, X_m = c_m) = \frac{1}{Z} \prod_{(c_i, c_j) \in E_r} \phi_r(c_i, c_j) = \prod_{(c_i, c_j) \in E_r} \theta_{r(c_i, c_j)}.$$

If $r(c_i, c_j) \notin \mathcal{DB}$ any substitution $r(X_i, X_j)\{X_i/c_i, X_j/c_j\}$ will have 0 probability.

TensorLog Inference II

Assume that we are interested in the query $\text{uncle}(c, Y)$ for some $c \in C$. We will perform BP over G_{uncle} conditioned on $X = c$. This corresponds to the following linear transformation:

$$(\mathbf{u}_c \mathbf{M}_{parent}) \mathbf{M}_{brother}.$$

Recall that \mathbf{u}_c is a one-hot vector which encodes the position of the constant $c \in C$. Then $\mathbf{v}_W = \mathbf{u}_c \mathbf{M}_{parent}$ is a vector where $\mathbf{v}_W[c'] = \theta_{parent(c, c')}$. Finally $\mathbf{v}_Y = \mathbf{v}_W \mathbf{M}_{brother}$ computes the marginal probability for Y which is exactly what BP does in the message passing steps.

TensorLog Inference III

- ▶ BP in G_r corresponds to simple linear transformations of the probability vector representations over \mathcal{KB} .
- ▶ In the case where some predicate is defined in multiple rules we simply return the sum of the response vectors for the individual definitions.
- ▶ For example, $g_{io}^{\text{uncle}}(\mathbf{u}_c) = g_{io}^{\text{uncle}_1}(\mathbf{u}_c) + g_{io}^{\text{uncle}_2}(\mathbf{u}_c)$ where uncle_i is the i th definition for the predicate.
- ▶ If a predicate r is defined by some $r' \in \mathcal{T}$ (as opposed to some $r' \in \mathcal{DB}$) we replace $\mathbf{v}_{F,X} \leftarrow \mathbf{v}_i \mathbf{M}_r$ with $\mathbf{v}_{F,X} \leftarrow g_{io}^{r'}(\mathbf{v}_i)$ in `compileMessage($F \rightarrow X$)`.

TensorLog BP Algorithm

Function *compileMessage*($F \rightarrow X$):

Data: Factor F representing $r(X)$ or $r(X_i, X_o)$,
Random variable X representing the logical variable

Result: Message from F to X : $\mathbf{v}_{F,X}$

if $F = r(X)$ **then**

$\mathbf{v}_{F,X} \leftarrow \mathbf{v}_r$

else if X is the output variable X_o of F **then**

$\mathbf{v}_i \leftarrow \text{compileMessage}(X_i \rightarrow F)$

$\mathbf{v}_{F,X} \leftarrow \mathbf{v}_i \mathbf{M}_r$

else if X is the input variable X_i of F **then**

$\mathbf{v}_o \leftarrow \text{compileMessage}(X_o \rightarrow F)$

$\mathbf{v}_{F,X} \leftarrow \mathbf{v}_o \mathbf{M}_r^\top$

end

return $\mathbf{v}_{F,X}$

TensorLog BP Algorithm

Function *compileMessage*($X \rightarrow F$):

Data: Random variable X representing the logical variable,
Factor F representing $r(X)$ or $r(X_i, X_o)$

Result: Message from X to F : $\mathbf{v}_{X,F}$

if X is the given variable to the query **then**

$\mathbf{v}_{X,F} \leftarrow \mathbf{u}_c$ // return the one-hot vector

// if the only factor which has X as an argument is F :

else if $\eta(X) = \{F\}$ **then**

$\mathbf{v}_{X,F} \leftarrow \mathbf{1}$ // return a vector of all 1's

else

 // loop over all k of X 's neighbor literals F_i except F

foreach $F_i \in \eta(X) \setminus F$ **do**

$\mathbf{v}_i \leftarrow \text{compileMessage}(F_i \rightarrow X)$

end

$\mathbf{v}_{X,F} \leftarrow \mathbf{v}_1 \circ \dots \circ \mathbf{v}_k$ // \circ is the element-wise product

end

return $\mathbf{v}_{X,F}$

Agenda

- ▶ Deep Learning
- ▶ Logical Student-Teacher Network
- ▶ TensorLog
- ▶ **Conclusion**

Discussion

- ▶ The Student-Teacher network incorporates logical rules as a regularization term over the objective function.
- ▶ This general framework can be applied in any context where some common sense knowledge may be useful.
- ▶ TensorLog gives a differentiable probabilistic logical inference procedure which may be used as a component in a DNN.
- ▶ The parameters of TensorLog models lose the interpretation as probabilities but may be trained via SGD (as opposed to EM).