Harnessing Deep Neural Networks with Logic Rules

Reporter: HE WEINAN

School of Data and Computer Science
Sun Yat-sen University

cura4ho@gmail.com

May 8, 2017

- Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. 2016. *Harnessing deep neural networks with logic rules*. In *Proc. of ACL.*
- http://www.cs.cmu.edu/~zhitingh/data/acl16harnessing_slides.pdf

Cited by

- Alashkar et al. (2017). Examples-Rules Guided Deep Neural Network for Makeup Recommendation. In *Thirty-First AAAI Conference on Artificial Intelligence.*
- Hu et al. (2016). Deep neural networks with massive learned knowledge. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP).*

Deep neural networks (DNN) is a powerful mechanism for learning patterns from massive data, achieving great performance on many problem domains such as *image classification* and *machine translation*.

But they still have limitations:

- Relying heavily on massive labeled training data
- The resulting parameters are often uninterpretable
- Hard to encode human knowledge and intention

On contrary, humans learn from

- concrete examples just like DNN do
- general **knowledge**

Logic rules is a flexible way to express structured knowledge. Integrating logic rules into DNNs might help to transfer human intention and domain knowledge the DNN models.

The authors propose an **knowledge distillation** framework to train a neural network using both examples and rules. A "teacher" is iteratively constructed to "teach" the student network.

Firstly, we need to know how traditional *neural networks* works. Consider an example of a classification problem.

Given a *training set* of records of clients who take a loan, we'd like to predict whether an unseen client will default (not paying the loan).

Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Label

Class

not default

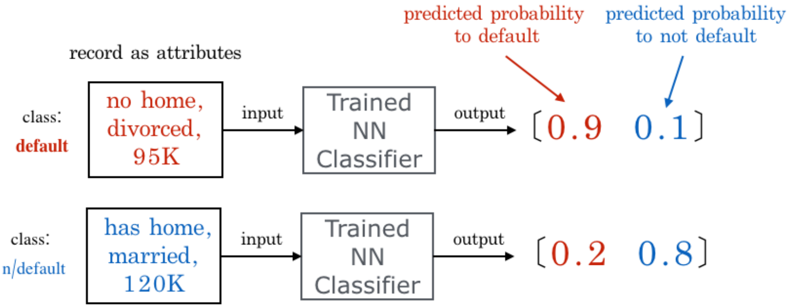
default

Training set for predicting borrowers who will default on loan payments.

If the client *John* has no home, is divorced and makes 50K a year, how likely will he default?

A **neural network** (NN) classifier is a computational model that classifies client records into two **classes**: default or not default. These are called class **labels**

For example, suppose we have already trained such a NN classifier, then for the two training example



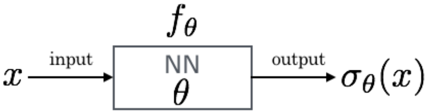
For an input record $\boldsymbol{x}$ with its actual label y, a NN classifier is a function that maps $\boldsymbol{x}$ into a soft prediction vector $\boldsymbol{\sigma_\theta}$:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\sigma_\theta}(\boldsymbol{x})$$

$\boldsymbol{\theta} = (\theta_1, \theta_2, ..., \theta_n)$: the **parameters** of the NN
$\boldsymbol{x} = (a_1, a_2, a_3)$: **input variable**, a_i is the i^{th}attribute of $\boldsymbol{x}$
$\boldsymbol{\sigma_\theta} = (p(y = \text{default}|\boldsymbol{x}),\ p(y = \text{not default}|\boldsymbol{x}))$: **soft prediction** vector
Note that the prediction vector $\boldsymbol{\sigma_\theta}(\boldsymbol{x})$ forms a probability distribution $p_{\boldsymbol{\theta}}(y|\boldsymbol{x})$.

Parameters or Weights

The **parameters** $\boldsymbol{\theta}$, or **weights** of the NN are the result of training on labeled data. $\boldsymbol{\theta}$ determines the performance of the NN.



Example (Classifying A Record)

For the record $\boldsymbol{x} = (\text{no home, divorced, 95K})$ which actually defaults,

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\sigma_\theta}(\boldsymbol{x}) = (0.9,\ 0.1)$$

The NN predicts that the probability of $\boldsymbol{x}$ $\mathrm{default}$ is 0.9. If the NN is perfect, the prediction should be $(1,\ 0)$.
We use a *label vector* $(1,\ 0)$ to represent $\mathrm{default}$, $(0,\ 1)$ for $\mathrm{not\ default}$.

To measure the performance: compare the prediction $(0.9,\ 0.1)$ with the actual label $y = (1, 0)$.

Loss Function

The **loss function** $\ell(y, \boldsymbol{\sigma_\theta}(\boldsymbol{x}))$ measures the difference between the prediction vector and the label vector for every training example.

To improve the performance: change parameters $\boldsymbol{\theta}$ to minimize the loss function. (*Gradient Descent*)

Objective

Iteratively change parameters $\boldsymbol{\theta}$ to $\mathrm{minimize}$ the loss function for all training examples.

Goal

Find $\boldsymbol{\theta}$ that minimize the loss ℓ on N training examples.

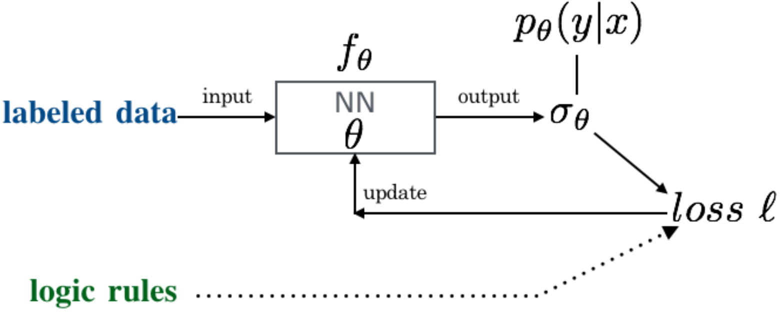
$$\boldsymbol{\theta} = \arg\min_{\theta \in \Theta} \frac{1}{N} \sum_{n=1}^{N} \ell(\boldsymbol{y}_n, \boldsymbol{\sigma}_\theta(\boldsymbol{x}_n))$$

Input: labeled training data $\mathcal{D} = \{(\boldsymbol{x}_n,\ y_n)\}_{n=1}^{N}$
Output: parameters (weights) $\boldsymbol{\theta}$ and prediction $p_{\boldsymbol{\theta}}$
Initialize the params $\boldsymbol{\theta}^{(0)}$, at iteration t, **repeat**:

1. Sample a subset $(\boldsymbol{X}, \boldsymbol{Y}) \subset \mathcal{D}$
2. Compute the loss on $(\boldsymbol{X}, \boldsymbol{Y})$
3. Update the params $\boldsymbol{\theta}^{(t)}$ from the previous params $\boldsymbol{\theta}^{(t-1)}$

until $\boldsymbol{\theta}^{(t)}$ converges, then let $\boldsymbol{\theta} = \boldsymbol{\theta}^{(t)}$.

Note: the traditional NNs ONLY train $\boldsymbol{\theta}$ using labeled training data.



Idea

1. Encode the knowledge rules using **Soft Logic**
2. Change the loss function to incorporate logic rules
3. When updating, θ is influenced by the rules

- In *first order logic*, propositions are evaluated to truth value $True$ or $False$.
- In *soft logic*, propositions and groundings (expressions with all variables being instantiated) evaluate to a continuous truth value from the interval $[0,\ 1]$.

The Boolean operators are extended as:

$$\begin{aligned} A\&B &= max\{A+B-1,0\} \\ A \vee B &= min\{A+B,1\} \\ \neg A &= 1-A \end{aligned} \tag{1}$$

Denote

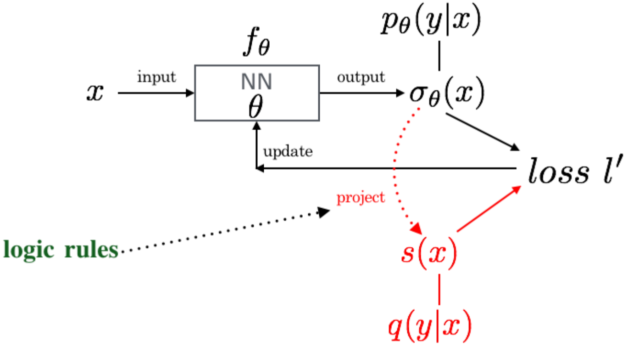
- R_l as the l^{th} rule over the input - target label space,
- λ_l as confidence level of rule R_l, with $\lambda_l = \infty$ indicating all groundings must be true.

Example

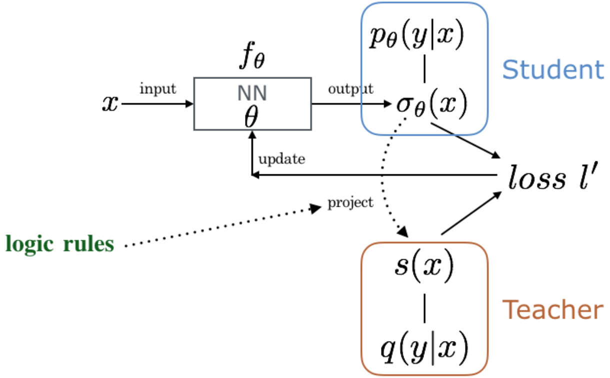
Rule: "People who earn less than 100K and are not married will default"

$$R \equiv (less_{100}(\boldsymbol{x}) \; \& \; \neg married(\boldsymbol{x})) \rightarrow default(\boldsymbol{x})$$

Now consider using the extracted rules to influence the loss function.



- NN maps the input $\boldsymbol{x}$ into prediction vector $\boldsymbol{\sigma_\theta}(\boldsymbol{x})$ that forms probability $p_{\boldsymbol{\theta}}(y|\boldsymbol{x})$.
- Project $\boldsymbol{\sigma_\theta}(\boldsymbol{x})$ into $s(\boldsymbol{x})$, a rule-regularized prediction that forms probability $q(y|\boldsymbol{x})$.
- Then $q(y|\boldsymbol{x})$ contains information of the rules.



Example (Rule-regularized Projection)

For a client $\boldsymbol{x}$ that $less_{100}(\boldsymbol{x})$ & $\neg married(\boldsymbol{x})$,
the NN outputs a prediction $\boldsymbol{\sigma_\theta}(\boldsymbol{x}) = [0.7, 0.3]$,
the projected prediction might be $s(\boldsymbol{x}) = [0.8, 0.2]$.
The new loss function l' will also consider the difference between $\boldsymbol{\sigma_\theta}(\boldsymbol{x})$ and $s(\boldsymbol{x})$.

Knowledge Distillation

- Prediction $p_{\boldsymbol{\theta}}(y|\boldsymbol{x})$ based only on examples (*student*)
- Prediction $q(y|\boldsymbol{x})$ also considers rules (*teacher*)
- **Distillation**: training $\boldsymbol{\theta}$ to imitate the outputs that consider the rules. (train the *student* $p_{\boldsymbol{\theta}}$ to imitate the *teacher* q)

at iteration t:

true hard label

soft prediction of student p_θ

$$\boldsymbol{\theta}^{(t+1)} = \arg\min_{\theta \in \Theta} \frac{1}{N} \sum_{n=1}^{N} (1-\pi)\ell(\boldsymbol{y}_n, \boldsymbol{\sigma}_\theta(\boldsymbol{x}_n)) + \pi\ell(\boldsymbol{s}_n^{(t)}, \boldsymbol{\sigma}_\theta(\boldsymbol{x}_n)),$$

balancing parameter

soft prediction of the teacher network q

Construct Teacher: Projection

With the set of rules $\mathcal{R} = \{(R_l, \lambda_l)\}_{l=1}^{L}$, the goal is to construct the teacher q at each iteration from $p_{\boldsymbol{\theta}}$ such that:

1. q fits the rules
2. q stays close to p_θ

Essentially we can compute such a q by solving the optimization problem:

$$\begin{aligned} \min_{q,\boldsymbol{\xi}\geq 0} \; & \boxed{\mathrm{KL}(q \| p_\theta(\boldsymbol{Y}|\boldsymbol{X}))} + C\sum\nolimits_l \xi_l \\ \text{s.t.} \; & \boxed{\lambda_l(1-\mathbb{E}_q[r_l(\boldsymbol{X},\boldsymbol{Y})]) \leq \xi_l} \\ & l = 1,\ldots,L \end{aligned}$$

close to p_θ

rule constraints

The closed-form solution:

$$q^*(\boldsymbol{Y}|\boldsymbol{X}) \propto p_\theta(\boldsymbol{Y}|\boldsymbol{X}) \exp\left\{-\sum_l C\lambda_l(1-r_l(\boldsymbol{X},\boldsymbol{Y}))\right\}$$

Goal

Given $\mathcal{R}$, find $\boldsymbol{\theta}$ that minimize the loss ℓ' on N training examples.
$\boldsymbol{\theta} = \arg\min_{\theta\in\Theta} \frac{1}{N}\sum_{n=1}^{N}(1-\pi)\ell(\boldsymbol{y}_n, \boldsymbol{\sigma}_\theta(\boldsymbol{x}_n)) + \pi\ell(\boldsymbol{s}_n(\boldsymbol{x}_n), \boldsymbol{\sigma}_\theta(\boldsymbol{x}_n))$

Input: Training data $\mathcal{D} = \{(\boldsymbol{x}_n,\ y_n)\}_{n=1}^{N}$,
The rule set $\mathcal{R} = \{(R_l, \lambda_l)\}_{l=1}^{L}$,
π:imitation parameter, C: regularization strength

Output: student network $p_{\boldsymbol{\theta}}$ and teacher network q

Initialize the parameters $\boldsymbol{\theta}$, then **repeat**:

1. Sample a subset $(\boldsymbol{X}, \boldsymbol{Y}) \subset \mathcal{D}$
2. Compute student $p_{\boldsymbol{\theta}}$
3. Construct teacher network q
4. Transfer knowledge into $p_{\boldsymbol{\theta}}$ by updating the params $\boldsymbol{\theta}$

until $\boldsymbol{\theta}$ converges

- Sentiment classification: sentence -> positive / negative
- Base neural network: CNN with accuracy about 87% (SST2)

For example, the sentence "This soup is good, but I don't like it" has negative sentiment although the first part seems to be positive. Notice "but" changes the sentiment, we thus consider a simple rule:

If a sentence S has a structure "A-but-B", then

$$sentiment(S) = sentiment(B)$$

Using the above rule and the knowledge distillation method, the authors observe a boost on accuracy.

Accuracy (%) of Sentiment Classification with all labeled data

	Model	SST2	MR	CR
1	CNN (Kim, 2014)	87.2	81.3±0.1	84.3±0.2
2	CNN-Rule-p	88.8	81.6±0.1	85.0±0.3
3	CNN-Rule-q	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

Row 2 and 3 are networks enhanced by the knowledge distillation, they outperform the base CNN and achieve better or comparable result with others.

Accuracy (%) of Sentiment Classification on SST2 dataset with varying sizes size of labeled data and semi-supervised learning.

	Data size	5%	10%	30%	100%
1	CNN	79.9	81.6	83.6	87.2
2	-Rule-p	81.5	83.2	84.5	88.8
3	-Rule-q	82.5	83.9	85.6	**89.3**
4	-semi-PR	81.5	83.1	84.6	–
5	-semi-Rule-p	81.7	83.3	84.7	–
6	-semi-Rule-q	**82.7**	**84.2**	**85.7**	–

Row 1 to 3 use only labeled data, while row 4 to 6 use the remaining data as unlabeled examples to train the NN. Row 2, 3, 5, 6 are results of the authors.

Contributions:

- Combine DNNs with logic rules to integrate knowledge
- Iteratively atransfer knowledge
- General applicability: CNNs/RNNs
- Better performance using only one rule

Drawbacks:

- We have to feed rules to the NN
- For many tasks other than NLP (such as CV), clear rules are difficult to come up with
- Can more rules lead to better performance?

Possible future work:

- Derive rules from data?
- In another domain of problems? (QA / Image QA / Vision)