Attentive Tree-structured Network for Monotonicity Reasoning

Zeming Chen

Department of Computer Science and Software Engineering
Rose-Hulman Institute of Technology
chenz16@rose-hulman.edu

July 15, 2020

Outline

- Problem Overview
- Attentive Tree Network
- Data
- Experimental results and analysis
- Conclusion
- Future Work

Natural Language Inference

Natural Language Inference (**NLI**): Determine whether a given premise **P** semantically entails a given hypothesis **H**

Example

- **P**: An Irishman won the Nobel prize for literature.
- **H**: An Irishman won the Nobel prize.
- **P** entails **H**

Monotonicity Reasoning

- NLI model needs to perform inferences including lexical and logical inferences.
- **Monotonicity Reasoning**: A type of logical inference that is based on word replacement

Example

1. (a) **All** students↓ carry a MacBook↑.
   (b) All students carry a laptop.
   (c) All new students carry a MacBook.
2. (a) **Not All** new students↑ carry a laptop.
   (b) Not All students carry a laptop.

Challenge

1. Many state-of-art neural inference models for NLI did not perform well on **monotonicity reasoning**.
2. Most models have low accuracy on **downward inference**.
3. Most models that do well on **upward inference** perform poorly on **downward inference**.

Motivation

1. **Tree-structured recursive neural networks** can learn logical semantics (Bowman et al. 2014)
2. **Self-attentive network** with multiple views of the sentence guide the model to learn the parts that are important to the task. (Conneau et al. 2018)

Attentive Tree Network

- Takes in four inputs: two embeddings and two dependency parse trees
- Glove 840B pre-trained word vectors, Stanford dependency parser
- Siamese neural network structure, identical tree-LSTMs, shared weights and parameters

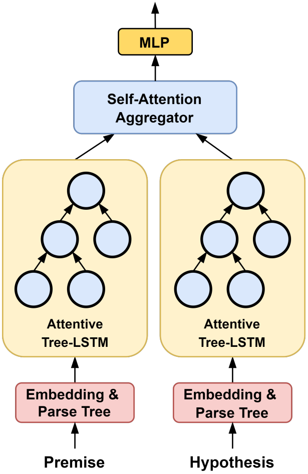

Figure: Overview of the model

Attentive Tree-LSTM

Definition

- **Child-sum Tree-LSTM**: each node is conditioned on its children's hidden states.
- **Attentive Tree-LSTM**: soft-attention over hidden states from the children.

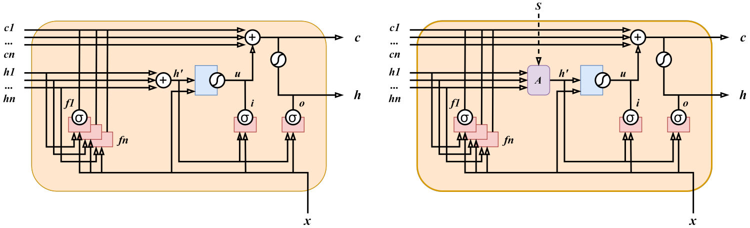

Figure: A comparison between a standard LSTM cell and an attentive LSTM cell

- The information flow in each LSTM cell is controlled by a gating mechanism similar to a sequential LSTM cell:

$$\tilde{h} = \Sigma_{1 \leq k \leq n} h_k,$$
$$i = \sigma(W^{(i)}x + U^{(i)}\tilde{h} + b^{(i)}),$$
$$o = \sigma(W^{(o)}x + U^{(o)}\tilde{h} + b^{(o)}),$$
$$u = tanh(W^{(u)}x + U^{(u)}\tilde{h} + b^{(u)}),$$
$$f_k = \sigma(W^{(f)}x + U^{(f)}h_k + b^{(f)}),$$
$$c = i \odot u + \Sigma_{1<n} f_k \odot c_k,$$
$$h = o \odot tanh(c),$$

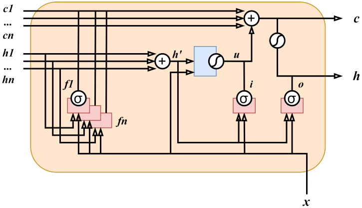

Figure: Child-sum Tree LSTM Cell

- **Soft Attention**: given hidden states $h_1, h_2, ..., h_n$ and an external vector s:

$$m_k = tanh(W^{(m)} h_k + U^{(m)} s),$$

$$\alpha_k = \frac{exp(w^\top m_k)}{\Sigma_{j=1}^{n} exp(w^\top m_j)},$$

$$g = \Sigma_{1 \leq k \leq n} \alpha_k h_k$$

- Apply a transformation to the context vector g:

$$\tilde{h} = tanh(W^{(a)} g + b^{(a)})$$

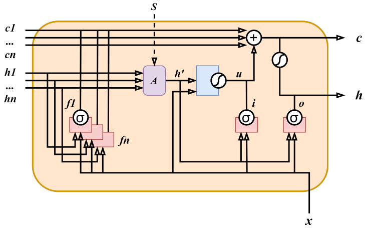

Figure: LSTM cell with attention

Self-attentive Aggregator

- Multi-hop self-attention mechanism
- Three matching methods from **Generic NLI Training Scheme**:
  1. Vector concatenation
  2. Absolute difference
  3. Element-wise product

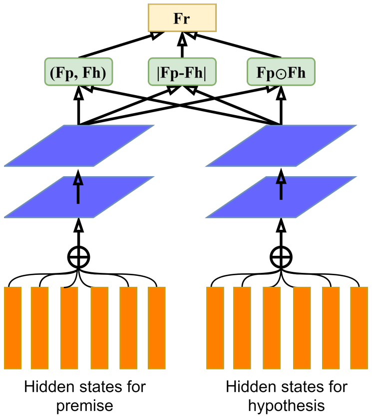

Figure: Self-Attentive Aggregator

Self-attentive Aggregator

- The context of a sentence is formed by multiple components like groups of related words and phrases.

Definition

Multi-hop Self-attention Mechanism: get multiple attentions that each focus on a different part of the sentence.

$$A = Softmax(W_{s2} tanh(W_{s1} H^{\top}))$$
$$M = AH$$

Self-attentive Aggregator

- Set of multiplicative interactions combining the two representations. Inspired by **Factored Gated Autoencoder** (Memisevic, 2013)
- A batched dot product between a matrix and the a weight tensor

$$F_p = tanh(\mathbf{bmm}(M_p, W_f)),$$
$$F_h = tanh(\mathbf{bmm}(M_h, W_f))$$

- Apply the matching methods:

$$F_r = [F_p; F_h; |F_p - F_h|; F_p \odot F_h],$$

Classifier

- **Multi-layer Perceptron (MLP)**: 3 layer feed-forward network with a 2-way softmax output
- **Objective Function**: Binary Cross-Entropy Loss

$$-\sum_{c} \mathbb{1}(X, c) log(p(c|X)), \tag{1}$$

Data

Training Data

- HELP
- Multi-Genre Natural Language Inference Corpus (MNLI)
- HELP+SubMNLI
- HELP+SubMNLI+Simple-Monotonicity-Training-Fragments
- HELP+SubMNLI+Hard-Monotonicity-Training-Fragments
- HELP+SubMNLI+Simple/Hard-Monotonicity-Training-Fragments

Test Data

- Monotonicity Entailment Dataset (MED)
- Semantic Fragments (monotonicity part)

Evaluation

Model	Train Data	Upward	Downward	None	All
BiMPM	SNLI	53.5	57.6	27.4	54.6
ESIM	SNLI	71.1	45.2	41.8	53.8
DeComp	SNLI	66.1	42.1	**64.4**	51.4
KIM	SNLI	78.8	30.3	53.1	48.0
BERT	MNLI	**82.7**	22.8	52.7	44.7
BERT	HELP+MNLI	76.0	70.3	59.9	71.6
AttentiveTreeNet (ours)	MNLI	54.7	60.4	37.8	58.6
AttentiveTreeNet (ours)	HELP	55.7	72.6	57.9	66.0
AttentiveTreeNet (ours)	HELP+SubMNLI	81.4	**74.5**	53.8	**75.7**

Table: Accuracy of our model and other state-of-art NLI models evaluated on MED.

- Our model had higher overall accuracy and downward inference accuracy
- Our model's upward inference accuracy comes close to BERT's performance

Abalation Test

Test	Model	Training Data	Upward	Downward	None	All
-	Full Model w/ vector-concat	HELP	55.7	72.6	57.9	66.0
1	–Self-Attentive Aggregator	HELP	65.1	67.1	53.7	65.7
2	–Tree-LSTM	HELP	36.6	65.5	94.8	49.5
3	Full Model w/ mean-dist	HELP	59.3	71.2	46.2	65.9
-	Full Model w/ vector-concat	HELP+SubMNLI	**81.4**	**74.5**	53.8	**75.7**
1	–Self-Attentive Aggregator	HELP+SubMNLI	70.5	66.9	85.6	69.1
2	–Tree-LSTM	HELP+SubMNLI	54.7	60.4	37.8	58.6
3	Full Model w/ mean-dist	HELP+SubMNLI	68.9	73.7	**91.0**	73.0

Table: Accuracy of ablation test trained on HELP and HELP+SubMNLI.

Three ablation tests:

1. Remove self-attentive aggregator (–Self-Attentive Aggregator)
2. Replace tree-LSTM with regular LSTM (–Tree-LSTM)
3. Use mean distance as a matching method (Full Model w/ mean-dist).

Addtional Tests

Training Data	SF	HF	MED
Pre-Trained Models			
HELP	57.0	56.8	66.0
HELP+SubMNLI	46.0	63.0	75.7
Re-trained Models w/ SF-training fragments			
HELP+frag	98.1	80.6	64.5
HELP+SubMNLI+frag	97.8	74.8	81.5
Re-trained Models w/ HF-training fragments			
HELP+frag	74.3	95.6	68.9
HELP+SubMNLI+frag	73.9	93.2	73.3
Re-trained Models w/ SF&HF-training fragments			
HELP+frag	96.9	94.6	64.5
HELP+SubMNLI+frag	96.4	98.3	75.4

Table: This table shows the testing accuracy on Simple/Hard Monotonicity.

- Pre-trained models did not perform well on simple/hard monotonicity fragments.
- Models re-trained with additional training data can master both simple and hard monotonicity reasoning, while retaining accuracy on original benchmark.

Conclusion

1. Incorporating **syntactic parse tree** can improve the model's performance on monotonicity reasoning.
2. **Self-structured attention mechanism** in the aggregation process provides a more precise guidance regarding learning monotonicity reasoning.

Future Work

1. Replace the LSTM cell with newer and better language models such as the **Transformer** (Vaswani ET AL, 2017).
2. Experiment with different attention mechanism such as the **Gaussian prior self-attention mechanism** (Guo el al. 2019).

Thank You!