CLaC at SemEval-2020 Task 5: Muli-task Stacked Bi-LSTMs

MinGyou Sung, Parsa Bagherzadeh, Sabine Bergler
CLaC Labs
Concordia University
Montreal, Canada
{mi_sung, p_bagher, bergler} @cse.concordia.ca

Abstract

We consider detection of the span of antecedents and consequents in argumentative prose a structural, grammatical task. Our system comprises a set of stacked Bi-LSTMs trained on two complementary linguistic annotations. We explore the effectiveness of grammatical features (POS and clause type) through ablation. The reported experiments suggest that a multi-task learning approach using this external, grammatical knowledge is useful for detecting the extent of antecedents and consequents and performs nearly as well without the use of word embeddings.

1 Introduction

Conditional statements are of interest to investigations into the semantics of natural language text because they inform on factivity of statements as well as establish reasoning and argumentation in text. In formal logic, conditionals obey the principle of explosion (*ex falso quod libed*), and counterfactuals allow any conclusion. In language, however, assuming that facts had been different is frequently used to explore, for instance, causal relations relevant to the ongoing argument. Both cases differ significantly from basic assertions and flagging and classifying these very specific passages of text will enhance other semantic annotations.

The basic structure of a conditional is thus found in many utterances, often as a way to specify presuppositions or assumptions for a given statement. Conditionals consist of two parts, the *antecedent* and the *consequent*, as illustrated in Example 1, where the antecedent (the *if* part) is underlined and the consequent spans the remainder of the sentence. Counterfactuals are conditionals where the antecedent does not hold true.

Example 1 *If there were peace, I wouldn't spend another second here.*

While only few NLP systems attempt to model inference, a significant number is concerned with attributing degrees of factuality to different statements. Before conditional statements can be mined for their contribution to factivity judgments, they have to be detected. SemEval 2020 Subtask 5.2 (Yang et al., 2020) is concerned with identifying the span of antecedent and consequent clauses in text. The data samples consist largely of single sentences, but may involve several sentences. We stipulate that this is a mainly a structural task and experiment with the grammatical notion of clause boundaries. Encoding various clause types on top of POS tags and Glove Word Embeddings (WEs) (Pennington et al., 2014), we find that a clause type layer improves the performance of a baseline of only Glove WEs but barely improves on a two layer architecture encoding WEs and POS only. However, a two layer architecture encoding only POS and clause type shows competitive performance at a drastically reduced parameter space. Our system represented the median in the officially scored systems and demonstrates that simple grammatical notions can be stable contributors to this task.

2 System description

We cast the task of detecting the spans of antecedent (A), consequent (C), and other text (O) as a sequence labeling task. For an input sequence of n tokens $S_i = \langle w_{i_1}, w_{i_2}, \ldots, w_{i_n} \rangle$, we predict a sequence of

Proceedings of the 14th International Workshop on Semantic Evaluation, pages 445–450
Barcelona, Spain (Online), December 12, 2020.

target labels $\hat{Y}_i = \langle \hat{y}_{i_1}, \hat{y}_{i_2}, \ldots, \hat{y}_{i_n} \rangle$, where $\hat{y}_{i_k} \in \{A, C, O\}$ for each token k in sample i. The task data, however, is presented as a sequence of characters and gold annotations use intervals of character offsets: $Ant_i = \langle ch_{ant_{i_1}}, ch_{ant_{i_2}}, \ldots ch_{ant_{i_l}} \rangle$ is the interval of antecedent characters labelled A in S_i, $Con_i = \langle ch_{con_{i_1}}, ch_{con_{i_2}}, \ldots ch_{con_{i_m}} \rangle$ is the character interval for the consequent, labelled C.

Preprocessing We define a strict input mapping f between character labels and token labels, with

$$f(w_k = \langle ch_{k_1}, \ldots ch_{k_j} \rangle) = \begin{cases} A & \text{if } ch_{k_i} \in Ant_i \text{ for } 1 \leq i \leq j \\ C & \text{if } ch_{k_i} \in Con_i \text{ for } 1 \leq i \leq j \\ O & \text{otherwise} \end{cases}$$

The corresponding output mapping trivially maps the label of a token to all its constituent characters.

2.1 Grammatical features

The task describes a strict grammatical pattern. (Kuncoro et al., 2018) observe that LSTMs are not strong on grammatical relations and they show that providing grammatical information can significantly improve LSTM performance. In this spirit, we extract grammatical features using a GATE pipeline with the ANNIE English Tokeniser (Cunningham et al., 2002), OpenNLP POS tagger (Apache Software Foundation, 2014), Stanford Parser (Klein and Manning, 2003) to extract POS tags and constituent tags *S, SBAR, and SINV*.

Example 2 CLaC system token annotations: **T**=input token, **P**=POS, **C**=clause, **L**=output label

T:	*If*	*there*	*were*	*peace*	*I*	*would*	*n't*	*spend*	*another*	*second*	*here*	.
P:	IN	EX	VBD	NN	PRP	MD	RB	VB	DT	NN	RB	.
C:	SBAR	SBAR	SBAR	SBAR	S	S	S	S	S	S	S	EOS
L:	*A*	*A*	*A*	*A*	*C*	*C*	*C*	*C*	*C*	*C*	*C*	*O*

POS tag Content words do not greatly impact this task, thus the reduction of input tokens to their POS tags should illuminate the structural patterns. The Stanford Parser uses the Penn Treebank tagset with 45 tags (36 main tags and 9 tags for punctuation, parenthesis, *etc.*) The Penn Treebank tag *IN* includes prepositions like *on* and subordinating conjunctions like *that*, masking an important clue for the potential start of a consequent. We thus introduce an additional POS tag SC for subordinating conjunctions. This brings the number of POS tags used to 46.

POS Penn Treebank tagset (45 tags)
POS1 assigns *that* to new POS tag *SC*
POS2 assigns *that* and *then* to *SC*

In ablation studies on a single layer architecture (that is, making the POS sequence the only input stream), POS2 performs better than POS1 and POS (see Table 1).

Clause tag Antecedent and consequent are clauses. To assist detection of the correct clause boundaries, we train a layer for relevant clause boundaries as determined by the Stanford parser.

The Penn Treebank tagset has 5 tags for clause constituents: *S* for simple declarative clauses, SBAR for complement clauses possibly introduced by a subordinating conjunction, SBARQ for direct questions introduced by a wh-word or a wh-phrase, SINV for inverted declarative sentences and SQ for inverted yes/no questions, or *SBARQ* for main clauses of a wh-question, following the wh-phrase (Bies et al., 1995).

In the general case, the antecedent is a subordinate clause, while the consequent is the main clause.[1] Subordinate clauses are labelled as SBAR, we select the lowest SBAR label on the path from a token to the sentence root for SBAR annotations in the input sequence. The same is true for the SINV label. Main clauses, however, are parents to subordinate clauses, not sisters, requiring a different processing.

We experiment with four variants, distinguished by the number and type of clauses included (ϕ specifies the number of tags encoded):

[1] Some data samples do not have both, but consist only of an antecendent or a consequent.

CL1 encodes *S, SBAR*, $\phi = 2$

CL1-1 encodes *S, SBAR, SINV*, $\phi = 3$

CL1-2 encodes *S, SBAR* and additionally recodes *SINV* as *SBAR*, $\phi = 2$

CL2 encodes *S,* S_m*, SBAR*, $\phi = 3$

CL2-1 encodes *S,* S_m*, SBAR, SINV*, $\phi = 4$

CL2-2 encodes *S,* S_m*, SBAR* and additionally recodes *SINV* as *SBAR*, $\phi = 3$

Clause level constituent tags are extracted from the parse tree. Let $path(w_{i_k})$ be the the ordered multiset of constituent labels on the path from token w_{i_k} to the sentence root.

Clause level constituent tag encoding CL1 Let w_{i_k} be a input token.

$$g_{CL1}(w_{i_k}) = \begin{cases} SBAR & \text{if } SBAR \in path(w_{i_k}) \\ S & \text{otherwise} \end{cases}$$

CL1 is modelled on the simplest conditional statements. A refined version distinguishes a wider variety of patterns, namely between S for root clauses, S_m for embedded main clauses, and SBAR and SINV for subordinate clauses.

Clause level constituent tag encoding CL2-1

$$g_{CL2-1}(w_{i_k}) = \begin{cases} SBAR & \text{if } SBAR \in path(w_{i_k}, root) \\ SINV & \text{if } SINV \in path(w_{i_k}, root) \\ S & SBAR, SINV \notin path(w_{i_k}, root) \\ & \text{and there is exactly one } S \in path(w_{i_k}, root) \\ S_m & \text{if } SBAR, SINV \notin path(w_{i_k}, root) \\ & \text{and there is more than one } S \in path(w_{i_k}, root) \end{cases}$$

2.2 Architecture

Word embeddings We initialize the embedding layer of the respective models with Glove[2] pre-trained word vectors, fine-tuned during training. For input sample $S_i = \langle w_{i_k}, ..., w_{i_n} \rangle$ let $X_i = \langle x_{i_1}, x_{i_2}, \ldots, x_{i_n} \rangle$ denote a sequence of word embeddings $x_{i_k} \in \mathbb{R}^d, d = 300$, where x_{i_j} is the embedding for token w_{i_j}.

Multi-task stacked Bi-LSTMs Our submitted system used 3 layers of Bi-LSTMs stacked on top of one another, all with input dimensionality $d_{input} = 300$ and hidden dimensionality $d_h = 150$. The first layer of the Bi-LSTM receives the embedding sequence X_i. The output stream of layer 1 feeds into layer 2, the output stream of layer 2 feeds into layer 3.[3]

Inspired by (Søgaard and Goldberg, 2016), the output at each layer is supervised for a different sequence labeling task by making predictions at each time step (see also Example 2).

layer 1 POS supervision, $W_1 \in \mathbb{R}^{300 \times \psi}$, where ψ is the number of POS tags

layer 2 clause supervision, $W_2 \in \mathbb{R}^{300 \times \phi}$, where ϕ is the number of clause tags

layer 3 main task supervision, $W_3 \in \mathbb{R}^{300 \times 3}$

The predicted label $\hat{y}^l_{i_k}$ for time-step k at layer l is determined by a simple linear classifier, parameterized by W_l:

$$\hat{y}^l_{i_k} = Softmax({W_l}^T x^l_{i_k}) \quad k = 1, \ldots, n; \;\; l \in \{1, 2, 3\}$$

where $x^l_{i_k}$ is the representation for token w_{i_k} at layer l. W_l is the classifier weight matrix at layer l.

[2]Glove Common Crawl, 840B tokens, 2.2M vocab, cased, 300d vectors

[3]For single layer and two layer architectures where WEs are not used, the input to the first layer is the POS stream.

Training paradigm At each forward pass, a layer is randomly selected based on a uniform distribution and the loss is calculated for the task corresponding to the selected layer. When performing back-propagation, the parameters of the selected layer as well as the parameters of all lower layers are updated.

Our model is implemented using the PyTorch library (Paszke et al., 2017). The losses at all layers are computed using Cross-Entropy loss and the network is optimized by the Adam optimizer (Kingma and Ba, 2014). The learning rate is $lr = 5 \times 10^{-4}$ and the network is trained for 7 to 10 epochs.

Post-processing There can only be one antecedent and only one consequent per data sample. Thus, if our system output shows disjoint regions for either antecedent or consequent, a post-processing step smooths over the gap.

Example 3 Post processing: **T**=token, **L**=system prediction, **F**= final, post-processed label

T:	*I*	*would*	*not*	*have*	*started*	*with*	*NAFTA*	***and***	*I*	*would*	*not*	*have*	*started*	*with*	*the*	*Europeans*	.
L:	A	A	A	A	A	A	A	**O**	A	A	A	A	A	A	A	A	O
F:	A	A	A	A	A	A	A	**A**	A	A	A	A	A	A	A	A	O

3 Results

We divide the original training set into training (3251 samples) and validation (300 samples) sets.

We present here only a small subset of extensive ablation on our variants. Note that the analysis presented here refers to our experiments on the validation data only. Table entries for test set performance are added here only to contextualize the development results.

Single layer baselines Our first observation is that when used in a single layer architecture, POS1 trails our best three layer systems by less than .05 in F1 measure and less than 10 exact matches. POS2 outperforms Glove WE as a single layer baseline in F1, if not in exact matches, see Table 1.

Method	P_A	R_A	P_C	R_C	P_O	R_O	P	R	F1	Match	Dev Match(%)	Test F1	Test Match(%)
WE	.73	.79	.70	.72	.83	.76	.77	.75	.76	**71**	23.7	.753	20.7
POS	.81	.75	.79	.71	.69	.80	.76	.76	.76	70	23.3	.742	19.5
POS1	.69	.74	.81	.74	.73	.74	.75	.74	.74	65	21.7	.760	22.7
POS2	.78	.78	.83	.70	.72	.82	.78	.77	**.77**	65	21.7	.767	23.2
WE+POS1	.79	.81	.71	.75	.82	.77	.78	.77	.78	85	28.3	.784	30.5
WE+POS2	.83	.80	.72	.78	.80	.77	.78	.78	.78	98	32.7	.786	29.5

Table 1: One and two layer architectures without clause type encodings

Two layer ablations In two layer architectures, POS1 and POS2 perform similarly and both versions show strengths in different combinations. We note that for this structural task, POS features combine effectively with WE, as illustrated in Table 1[4].

In this structural task, two layer architectures that do not use WEs are competitive, as shown in Table 2.

The clause encodings do not perform as well in single layer architectures, but in combination with WE, they demonstrate an increase in exact matches, as shown in Table 2. Interestingly, two layer architectures using only POS and clause features rival combinations using WEs but reduce the parameter space to 46×4.

Three layer architectures The three layer architectures shown in Table 3 outperform two layers in F1 as well as in exact matches for most cases, indicating that the grammatical information encoded is not sufficient and WEs stabilize and improve performance.

WE+POS1+CL1-1 is our submitted model (highlighted in Table 3). Other versions have identical F1, but superior exact matches. We see effects of overfitting on our validation set, since the ranking of our methods on the validation set does not always correspond with the ranking on the actual test set. For instance, the overall best performer on the validation set (WE+POS2+CL1-2) does not perform equally well on the test set. Noteworthy is the performance of a two layer architecture with no word embeddings on the test set: POS2+CL2-2 (Table 2). This presents a much reduced feature space for a very strong performance.

[4]Matches are exact matches. *Dev Match(%)* refers to the percentage of exact matches on the development set, *Test Match(%)* refers to the percentage of exact matches on the test set.

Method	P_A	R_A	P_C	R_C	P_O	R_O	P	R	F1	Match	Dev Match(%)	Test F1	Test Match(%)
WE+CL1	.73	.80	.73	.76	.86	.77	.79	.77	.78	100	33.3	.773	28.6
WE+CL1-1	.82	.77	.83	.76	.77	.86	.81	.79	.80	101	33.7	.795	30.8
WE+CL1-2	.80	.82	.77	.78	.81	.77	.80	.79	.79	104	34.7	.784	27.7
WE+CL2	.82	.83	.76	.84	.84	.76	.80	.81	.81	104	34.7	.790	28.9
WE+CL2-1	.80	.83	.75	.76	.80	.77	.78	.79	.78	100	33.3	.773	27.4
WE+CL2-2	.85	.80	.75	.81	.81	.81	.81	.81	.81	108	36.0	.786	28.5
POS+CL1	.82	.78	.79	.74	.73	.80	.78	.77	.78	91	30.3	.762	25.0
POS1+CL1	.83	.79	.80	.77	.75	.81	.80	.79	.79	92	30.6	.780	30.3
POS2+CL1	.85	.79	.81	.77	.74	.83	.80	.79	.80	94	31.3	.782	30.9
POS+CL1-1	.83	.77	.78	.72	.70	.80	.77	.76	.76	86	28.7	.759	24.7
POS1+CL1-1	.85	.79	.81	.78	.74	.81	.80	.79	.80	94	31.3	.780	30.2
POS2+CL1-1	.82	.80	.81	.78	.74	.79	.79	.79	.79	89	29.6	.775	30.0
POS+CL1-2	.81	.80	.79	.73	.73	.79	.78	.77	.77	85	28.3	.761	24.2
POS1+CL1-2	.78	.84	.80	.81	.80	.74	.79	.80	.79	96	32.0	.785	28.7
POS2+CL1-2	.80	.77	.81	.68	.71	.83	.77	.76	.76	83	27.7	789	30.3
POS+CL2	.81	.79	.81	.74	.72	.81	.78	.78	.78	90	30.0	.764	26.3
POS1+CL2	.84	.77	.81	.76	.73	.83	.80	.79	.79	95	31.6	.784	31.9
POS2+CL2	.84	.76	.81	.76	.73	.84	.79	.79	.79	93	31.0	.784	29.2
POS+CL2-1	.81	.75	.81	.71	.68	.80	.77	.76	.76	91	30.3	.763	24.5
POS1+CL2-1	.83	.79	.79	.78	.75	.79	.79	.79	.79	102	34.0	.775	31.3
POS2+CL2-1	.84	.79	.80	.79	.76	.81	.80	.80	.80	99	33.0	.782	29.5
POS+CL2-2	.82	.80	.79	.76	.74	.78	.78	.78	.78	89	29.7	.763	24.5
POS1+CL2-2	.84	.79	.81	.77	.76	.83	.80	.80	.80	81	27.0	.789	30.7
POS2+CL2-2	.80	.79	.78	.75	.75	.78	.78	.78	.78	71	23.7	.784	31.2

Table 2: Two layer architectures with clause type supervision

Official results Our official run represented the median for official runs, taking position 6 for precision (0.81), recall (0.81), and F1 (0.78). The test exact match percentage was 28 and obtained rank 5.

We interpret the consistency in results for precision and recall as a measure of the robustness of the system. The three layer architecture with clause level encoding was not strictly necessary for our performance, as WE+POS1 performed better on the test set in exact matches. However, the less balanced precision (0.864) and recall (0.763) for WE+POS1 on the test set suggests more volatile behaviour. Inversely, this suggests that clause type encoding has a stabilizing effect and Table 3 shows that including clause features has the potential for better performance.

Method	P_A	R_A	P_C	R_C	P_O	R_O	P	R	F1	Match	Dev Match(%)	Test F1	Test Match(%)
WE+POS1+CL1	.87	.83	.84	.78	.73	.79	.80	.81	.80	100	33.3	.786	31.5
WE+POS1+CL1-1	.80	.83	.76	.79	.86	.80	.81	.80	.81	102	34.0	.784	28.2
WE+POS1+CL1-2	.79	.85	.76	.78	.86	.80	.82	.80	.81	107	35.7	.795	29.5
WE+POS1+CL2	.74	.82	.82	.78	.82	.78	.80	.79	.79	101	33.7	.793	30.9
WE+POS1+CL2-1	.79	.84	.70	.75	.86	.78	.80	.78	.79	94	31.3	.795	31.4
WE+POS1+CL2-2	.77	.82	.73	.76	.85	.79	.80	.78	.79	107	35.6	.787	29.6
WE+POS2+CL1	.82	.84	.70	.76	.86	.79	.81	.79	.80	97	32.3	.797	32.0
WE+POS2+CL1-1	.82	.82	.75	.82	.82	.76	.80	.80	80	108	36.0	.792	31.3
WE+POS2+CL1-2	.81	.85	.80	.81	.83	.79	.81	.82	.82	121	40.3	.777	29.9
WE+POS2+CL2	.78	.84	.75	.77	.87	.80	.81	.80	.80	97	32.3	.798	32.6
WE+POS2+CL2-1	.86	.82	.84	.75	.74	.84	.81	.80	.80	100	33.3	.801	32.3
WE+POS2+CL2-2	.86	.84	.86	.74	.75	.86	.82	.81	.81	114	38.0	.788	30.3

Table 3: Three layer architectures including submitted system

4 Conclusion

Our goal was to test the possibility of grammatical information to improve exact matches of antecedent and consequent span detection. Ablation studies show that grammatical features by themselves form a solid baseline in a two layer Bi-LSTM architecture and dramatically reduce the parameter space for the task.

Our experiments demonstrate that multi-task stacked Bi-LSTM models can effectively super-encode the grammatical features POS and clause type, improving performance for both F1 and exact match scores. For this task, the difference in outcome barely justifies the increase in complexity for three layers. However, the combination results in stable systems that operate at the precision-recall break even point and that (slightly) outperform single and two layer models. They form thus a promising basis for semantically more complex tasks.

References

Apache Software Foundation. 2014. openNLP Natural Language Processing Library. http://opennlp.apache.org/.

Ann Bies, Mark Ferguson, Karen Katz, Robert MacIntyre, Victoria Tredinnick, Grace Kim, Mary Ann Marcinkiewicz, and Britta Schasberger. 1995. Bracketing guidelines for Treebank II Style. Penn Treebank Project. Technical report, University of Pennsylvania. https://www.ldc.upenn.edu/sites/www.ldc.upenn.edu/files/penn-etb-2-style-guidelines.pdf.

Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. 2002. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015*, San Diego, CA. arXiv:1412.6980 [cs.LG].

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*.

Adhiguna Kuncoro, Chris Dyer, John Hale, Dani Yogatama, Stephen Clark, and Phil Blunsom. 2018. LSTMs can learn syntax-sensitive dependencies well, but modeling structure makes them better. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL 2018)*. (Long Papers).

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS 2017*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 1532–1543.

Anders Søgaard and Yoav Goldberg. 2016. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 231–235.

Xiaoyu Yang, Stephen Obadinma, Huasha Zhao, Qiong Zhang, Stan Matwin, and Xiaodan Zhu. 2020. SemEval-2020 Task 5: Counterfactual recognition. In *Proceedings of the 14th International Workshop on Semantic Evaluation (SemEval-2020)*, Barcelona, Spain.