Deep multi-task learning with low level tasks supervised at lower layers

Anders Søgaard
University of Copenhagen
soegaard@hum.ku.dk

Yoav Goldberg
Bar-Ilan University
yoav.goldberg@gmail.com

Abstract

In all previous work on deep multi-task learning we are aware of, all task supervisions are on the same (outermost) layer. We present a multi-task learning architecture with deep bi-directional RNNs, where different tasks supervision can happen at different layers. We present experiments in syntactic chunking and CCG supertagging, coupled with the additional task of POS-tagging. We show that it is consistently better to have POS supervision at the innermost rather than the outermost layer. We argue that this is because "low-level" tasks are better kept at the lower layers, enabling the higher-level tasks to make use of the shared representation of the lower-level tasks. Finally, we also show how this architecture can be used for domain adaptation.

1 Introduction

We experiment with a multi-task learning (MTL) architecture based on deep bi-directional recurrent neural networks (bi-RNNs) (Schuster and Paliwal, 1997; Irsoy and Cardie, 2014). MTL can be seen as a way of regularizing model induction by sharing representations (hidden layers) with other inductions (Caruana, 1993). We use deep bi-RNNs with task supervision from multiple tasks, sharing one or more bi-RNNs layers among the tasks. Our main contribution is the novel insight that (what has historically been thought of as) low-level tasks are better modeled in the low layers of such an architecture. This is in contrast to previous work on deep MTL (Collobert et al., 2011; Luong et al., 2015) , in which supervision for all tasks happen at the same (outermost) layer. Multiple-tasks supervision at the outermost layer has a strong tradition in neural net models in vision and elsewhere (Caruana, 1993; Zhang and Zhang, 2014; Yim et al., 2015). However, in NLP it is natural to think of some levels of analysis as feeding into others, typically with low-level tasks feeding into high-level ones; e.g., POS tags as features for syntactic chunking (Sang and Buchholz, 2000) or parsing (Nivre et al., 2007). Our architecture can be seen as a seamless way to combine multi-task and cascaded learning. We also show how the proposed architecture can be applied to domain adaptation, in a scenario in which we have high-level task supervision in the source domain, and lower-level task supervision in the target domain.

As a point of comparison, Collobert et al. (2011) improved deep convolutional neural network models of syntactic chunking by also having task supervision from POS tagging at the outermost level. In our work, we use recurrent instead of convolutional networks, but our main contribution is observing that we obtain better performance by having POS task supervision at a lower layer. While Collobert et al. (2011) also experiment with NER and SRL, they only obtain improvements from MTL with POS and syntactic chunking. We show that similar gains can be obtained for CCG supertagging.

Our contributions (i) We present a MTL architecture for sequence tagging with deep bi-RNNs; (ii) We show that having task supervision from all tasks at the outermost level is often suboptimal; (iii) we show that this architecture can be used for domain adaptation.

2 Sequence tagging with deep bi-RNNs

Notation We use $x_{1:n}$ to denote a sequence of n vectors $x_1, \cdots, x_n$. $F_\theta(\cdot)$ is a function parameterized with parameters θ. We write $F_L(\cdot)$ as a shortcut to F_{θ_L} – an instantiation of F with a spe-

Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pages 231–235,
Berlin, Germany, August 7-12, 2016.

cific set of parameters θ_L. We use $\circ$ to denote a vector concatenation operation.

Deep bi-RNNs We use a specific flavor of Recurrent Neural Networks (RNNs) (Elman, 1990) called long short-term memory networks (LSTMs) (Hochreiter and Schmidhuber, 1997). For brevity, we treat RNNs as a black-box abstraction, and LSTMs as an instance of the RNN interface. For further details on RNNs and LSTMs, see (Goldberg, 2015; Cho, 2015). We view RNN as a parameterized function $RNN_\theta(x_{1:n})$ mapping a sequence of n input vectors $x_{1:n}$, $x_i \in \mathbb{R}^{d_{in}}$ to a an output vector $h_n \in \mathbb{R}^{d_{out}}$. The output vector h_n is conditioned on all the input vectors $x_{1:n}$, and can be thought of as a *summary* of $x_{1:n}$. The RNN can be applied to all prefixes $x_{1:i}$, $1 \leq i \leq n$ of $x_{1:n}$, resulting in n output vectors $h_{1:n}$, where $h_{1:i}$ summarizes $x_{1:i}$.

A *deep RNN* (or k-layer RNN) is composed of k RNN functions $RNN_1, \cdots, RNN_k$ that feed into each other: the output $h^\ell_{1:n}$ of RNN_ℓ becomes the input of $RNN_{\ell+1}$. Stacking RNNs in this way was empirically shown to be effective.

A *bidirectional RNN* (Schuster and Paliwal, 1997; Irsoy and Cardie, 2014) is composed of two RNNs, RNN_F and RNN_R, one reading the sequence in its regular order, and the other reading it in reverse. Concretely, given a sequence $x_{1:n}$ and a desired index i, the function $BIRNN_\theta(x_{1:n}, i)$ is defined as:

$$\begin{aligned} BIRNN_\theta(x_{1:n}, i) &= v_i = h_{F,i} \circ h_{R,i} \\ h_{F,i} &= RNN_F(x_1, x_2, \cdots, x_i) \\ h_{R,i} &= RNN_R(x_n, x_{n-1}, \cdots, x_i) \end{aligned}$$

The vector $v_i = BIRNN(x_{1:n}, i)$ is then a representation of the ith item in $x_{1:n}$, taking into account both the entire history $x_{1:i}$ and the entire future $x_{i:n}$.

Finally, in a *deep bidirectional RNN*, both RNN_F and RNN_R are k-layer RNNs, and $BIRNN^\ell(x_{1:n}, i) = v^\ell_i = h^\ell_{F,i} \circ h^\ell_{R,i}$.

Greedy sequence tagging with deep bi-RNNs In a sequence tagging task, we are given an input $w_1, \cdots, w_n$ and need to predict an output $y_1, \cdots, y_n$, $y_i \in [1, \cdots, |L|]$, where L is a label set of interest; i.e., in a POS tagging task, L is the part-of-speech tagset, and y_i is the pos-tag for word w_i.

If we take the inputs $x_{1:n}$ to correspond to a sequence of sentence words $w_1, \cdots, w_n$, we can think of $v_i = BIRNN(x_{1:n}, i)$ as inducing an infinite window around a focus word w_i. We can then use v_i as an input to a multiclass classification function $f(v_i)$, to assign a tag $\hat{y}_i$ to each input location i. The tagger is greedy: the tagging decisions are independent of each other. However, as shown below and in other recent work using bi-RNNs for sequence tagging, we can still produce competitive tagging accuracies, because of the richness of the representation v_i that takes the entire input sequence into account.

For a k-layer bi-RNN tagger we get:

$$\begin{aligned} tag(w_{1:n}, i) = \hat{y}_i &= f(v^k_i) \\ v^k_i &= BIRNN^k(x_{1:n}, i) \\ x_{1:n} &= E(w_1), E(w_2), \cdots, E(w_n) \end{aligned}$$

where E as an embedding function mapping each word in the vocabulary into a d_{emb}-dimensional vector, and v^k_i is the output of the kth BIRNN layer as defined above.

All the parameters (the embedding vectors for the different vocabulary items, the parameters of the different RNNs and the parameters of the classification function f) are trained jointly in order to minimize the tagging loss over a sentence. The embedding vectors are often initialized using vectors that were pre-trained in a semi-supervised manner.

This sequence tagging architecture was introduced to NLP by Irsoy and Cardie (2014). A similar architecture (with an RNN instead of bi-RNN) was applied to CCG supertagging by Xu et al (2015).

MTL in deep bi-RNNs In a multi-task learning (MTL) setting, we have several prediction tasks over the same input space. For example, in sequence tagging, the input may be the words in the sentence, and the different tasks can be POS-tagging, named entity recognition, syntactic chunking, or CCG supertagging. Note that the different tasks do not have to be traditional NLP tasks, but also, say, two POS-annotated corpora with slightly different guidelines. Each task has its own output vocabulary (a task specific tagset), but all of them map the length n input sequence into a length n output sequence.

Intuitively, although NLP tasks such as POS tagging, syntactic chunking and CCG supertagging are different than each other, they also share lot of substructure, e.g., knowing that a word is a

verb can help in determining its CCG supertag and the syntactic chunk it participate in. We would therefore like for these models to share parameters.

The common approach is to share parameters across most of the network. In the k-layers deep bi-RNN tagger described above this is naturally achieved by sharing the bi-RNN part of the network across tasks, but training a specialized classification tagger $f_t(v_i^k)$ for each task t.

This encourages the deep bi-RNN to learn a representation v_i^k that is useful for prediction of the different tasks, allowing them to share parameters.

Supervising different tasks on different layers Previous work in NLP on cascaded learning such as Shen and Sarkar (2005) suggests there is sometimes a natural order among the different tasks: some tasks may benefit more from other tasks, than the other way around. This suggests having task supervision for low-level tasks at the lower bi-RNN layers. This also enables task-specific deep learning of the high-level tasks.

Instead of conditioning all tasks on the outermost bi-RNN layer, we associate an RNN level $\ell(t)$ with each task t, and let the task specific classifier feed from that layer, e.g., $pos\_tag(w_{1:n}, i) = f_{pos}(v_i^{\ell(pos)})$. This enables a hierarchy a task with cascaded predictions, as well as deep task-specific learning for high-level tasks. This means there will be layers shared by all tasks and layers that are specific to some tasks:

$$\begin{aligned} pos\_tag(w_{1:n}, i) &= f_{pos}(v_i^{\ell(pos)}) \\ chunk\_tag(w_{1:n}, i) &= f_{chunk}(v_i^{\ell(chunk)}) \\ ccg\_tag(w_{1:n}, i) &= f_{ccg}(v_i^{\ell(ccg)}) \\ v_i^{\ell} &= BIRNN^{\ell}(x_{1:n}, i) \\ x_{1:n} &= E(w_1), E(w_2), \cdots, E(w_n) \end{aligned}$$

The Multi-task training protocol We assume T different training set, $D_1, \cdots, D_T$, where each D_t contains pairs of input-output sequences $(w_{1:n}, y_{1:n}^t)$, $w_i \in V$, $y_i^t \in L^t$. The input vocabulary V is shared across tasks, but the output vocabularies (tagset) L^t are task dependent.

At each step in the training process we choose a random task t, followed by a random training instance $(w_{1:n}, y_{1:n}^t) \in D_t$. We use the tagger to predict the labels $\hat{y}_i^t$, suffer a loss with respect to the true labels y_i^t and update the model parameters. Notice that a task t is associated with a bi-RNN level $\ell(t)$. The update for a sample from task t affects the parameters of f_t and $BIRNN^1, \cdots, BIRNN^{\ell(t)}$, but not the parameters of $f_{t' \neq t}$ or $BIRNN^{j > \ell(t)}$.

Implementation details Our implementation is based the CNN library[1] for dynamic neural networks. We use CNN's LSTM implementation as our RNN variant. The classifiers $f_t()$ take the form of a linear transformation followed by a softmax $f_t(v) = \arg\max_i softmax(W^{(t)}v + b^t)[i]$, where the weights matrix $W^{(t)}$ and bias vector $b^{(t)}$ are task-specific parameters. We use a cross-entropy loss summed over the entire sentence. The network is trained using back-propagation and SGD with batch-sizes of size 1, with the default learning rate. Development data is used to determine the number of iterations.

We initialize the embedding layer E with pre-trained word embeddings. We use the Senna embeddings[2] in our domain adaptation experiments, but these embeddings may have been induced from data including the test data of our main experiments, so we use the Polyglot embeddings in these experiments.[3] We use the same dimensionality for the hidden layers as in our pre-trained embeddings.

3 Experiments and Results

We experiment with POS-tagging, syntactic chunking and CCG supertagging. See examples of the different tasks below:

WORDS	Vinken	,	61	years	old
POS	NNP	,	CD	NNS	JJ
CHUNKS	B-NP	I-NP	I-NP	I-NP	I-NP
CCG	N	,	N/N	N	(S[adj]\ NP)\ NP

In-domain MTL In these experiments, POS, Chunking and CCG data are from the English Penn Treebank. We use sections 0–18 for training POS and CCG supertagging, 15–18 for training chunking, 19 for development, 20 for evaluating chunking, and 23 for evaluating CCG supertagging. These splits were motivated by the need for comparability with previous results.[4]

[1] http://www.github.com/clab/cnn

[2] http://ronan.collobert.com/senna/

[3] http://polyglot.readthedocs.org

[4] In CCG supertagging, we follow common practice and only evaluate performance with respect to the 425 most frequent labels. For this reason, we also do not calculate any loss from not predicting the other labels during training (but we do suffer a loss for tokens tagged with a different label during evaluation).

	LAYERS		DOMAINS			
	CHUNKS	POS	BROADCAST (6)	BC-NEWS (8)	MAGAZINES (1)	WEBLOGS (6)
BI-LSTM	3	-	88.98	91.84	90.09	90.36
	3	3	88.91	91.84	90.95	90.43
	3	1	**89.48**	**92.03**	**91.53**	**90.78**

Table 1: Domain adaptation results for chunking across four domains (averages over micro-F_1s for individual files). The number in brackets is # files per domain in OntoNotes 4.0. We use the two first files in each folder for POS supervision (for train+dev).

We do MTL training for either (POS+chunking) or (POS+CCG), with POS being the lower-level task. We experiment three architectures: single task training for higher-level tasks (no POS layer), MTL with both tasks feeding off of the outer layer, and MTL where POS feeds off of the inner (1st) layer and the higher-level task on the outer (3rd) layer. OUr main results are below:

	POS	CHUNKS	CCG
BI-LSTM	-	95.28	91.04
	3	95.30	92.94
	1	**95.56**	**93.26**

Our CHUNKS results are competitive with state-of-the-art. Suzuki and Isozaki (2008), for example, reported an F_1-score of 95.15% on the CHUNKS data. Our model also performs considerably better than the MTL model in Collobert et al. (2011) (94.10%). Note that our relative improvements are also bigger than those reported by Collobert et al. (2011). Our CCG super tagging results are also slighly better than a recently reported result in Xu et al. (2015) (93.00%). Our results are significantly better ($p < 0.05$) than our baseline, and POS supervision at the lower layer is consistently better than standard MTL.

Additional tasks? We also experimented with NER (CoNLL 2003), super senses (SemCor), and the Streusle Corpus of texts annotated with MWE brackets and super sense tags. In none of these cases, MTL led to improvements. This suggests that MTL only works when tasks are sufficiently similar, e.g., all of syntactic nature. Collobert et al. (2011) also observed a drop in NER performance and insignificant improvements for SRL. We believe this is an important observation, since previous work on deep MTL often suggests that most tasks benefit from each other.

Domain adaptation We experiment with domain adaptation for syntactic chunking, based on OntoNotes 4.0. We use WSJ newswire as our source domain, and broadcast, broadcasted news, magazines, and weblogs as target domains. We assume main task (syntactic chunking) supervision for the source domain, and lower-level POS supervision for the target domains. The results in Table 1 indicate that the method is effective for domain adaptation when we have POS supervision for the target domain. We believe this result is worth exploring further, as the scenario in which we have target-domain training data for low-level tasks such as POS tagging, but not for the task we are interested in, is common. The method is effective *only when the lower-level POS supervision is applied at the lower layer*, supporting the importance of supervising different tasks at different layers.

Rademacher complexity is the ability of models to fit random noise. We use the procedure in Zhu et al. (2009) to measure Rademacher complexity, i.e., computing the average fit to k random relabelings of the training data. The subtask in our set-up acts like a regularizer, increasing the inductive bias of our model, preventing it from learning random patterns in data. Rademacher complexity measures the decrease in ability to learn such patterns. We use the CHUNKS data in these experiments. A model that does not fit to the random data, will be right in 1/22 cases (with 22 labels). We report the Rademacher complexities relative to this.

LSTM(-3)	LSTM(3-3)	LSTM(1-3)
1.298	1.034	0.990

Our deep single task model increases performance over this baseline by 30%. In contrast, we see that when we predict both POS and the target task at the top layer, Rademacher complexity is lower and close to a random baseline. Interestingly, regularization seems to be even more effective, when the subtask is predicted from a lower layer.

4 Conclusion

MTL and sharing of intermediate representations, allowing supervision signals of different tasks to benefit each other, is an appealing idea. However, in case we suspect the existence of a hierarchy between the different tasks, we show that it is worthwhile to incorporate this knowledge in the MTL architecture's design, by making lower level tasks affect the lower levels of the representation.

Acknowledgments

Anders Søgaard was supported by ERC Starting Grant LOWLANDS No. 313695. Yoav Goldberg was supported by The Israeli Science Foundation grant No. 1555/15 and a Google Research Award.

References

Rich Caruana. 1993. Multitask learning: a knowledge-based source of inductive bias. In *ICML*.

Kyunghyun Cho. 2015. Natural language understanding with distributed representation. *CoRR*, abs/1511.07916.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

Jeffrey L. Elman. 1990. Finding Structure in Time. *Cognitive Science*, 14(2):179–211, March.

Yoav Goldberg. 2015. A primer on neural network models for natural language processing. *CoRR*, abs/1510.00726.

Sepp Hochreiter and Juergen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9:1735–1780.

Ozan Irsoy and Claire Cardie. 2014. Opinion Mining with Deep Recurrent Neural Networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 720–728, Doha, Qatar, October. Association for Computational Linguistics.

M.-T. Luong, Q. V. Le, I. Sutskever, O. Vinyals, and L. Kaiser. 2015. Multi-task Sequence to Sequence Learning. *ArXiv e-prints*, November.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932, Prague, Czech Republic, June. Association for Computational Linguistics.

Erik F. Tjong Kim Sang and Sabine Buchholz. 2000. Introduction to the conll-2000 shared task chunking. In *Fourth Conference on Computational Natural Language Learning and of the Second Learning Language in Logic Workshop*, pages 127–132.

M. Schuster and Kuldip K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, November.

Hong Shen and Anoop Sarkar. 2005. Voting between multiple data representations for text chunking. In *Proceedings of the 18th Meeting of the Canadian Society for Computational Intelligence*.

Jun Suzuki and Hideki Isozaki. 2008. Semi-supervised sequential labeling and segmentation using gigaword scale unlabeled data. In *ACL*.

Wenduan Xu, Michael Auli, and Stephen Clark. 2015. Ccg supertagging with a recurrent neural network. In *ACL*.

Junho Yim, Heechul Jung, ByungIn Yoo amd Changkyu Choi, Dusik Park, and Junmo Kim. 2015. Rotating Your Face Using Multi-task Deep Neural Network. In *CVPR*.

Cha Zhang and Zhengyou Zhang. 2014. Improving Multiview Face Detection with Multi-Task Deep Convolutional Neural Networks. In *WACV*.

Jerry Zhu, Timothy Rogers, and Bryan Gibson. 2009. Human Rademacher complexity. In *NIPS*.