Supplemental Material: A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks

Kazuma Hashimoto[*], **Caiming Xiong**[†], **Yoshimasa Tsuruoka, and Richard Socher**
The University of Tokyo
{hassy, tsuruoka}@logos.t.u-tokyo.ac.jp
Salesforce Research
{cxiong, rsocher}@salesforce.com

A Training Details

Pre-training embeddings We used the `word2vec` toolkit to pre-train the word embeddings. We created our training corpus by selecting lowercased English Wikipedia text and obtained 100-dimensional Skip-gram word embeddings trained with the context window size 1, the negative sampling method (15 negative samples), and the sub-sampling method (10^{-5} of the sub-sampling coefficient). We also pre-trained the character n-gram embeddings using the same parameter settings with the case-sensitive Wikipedia text. We trained the character n-gram embeddings for $n = 1, 2, 3, 4$ in the pre-training step.

Embedding initialization We used the pre-trained word embeddings to initialize the word embeddings, and the word vocabulary was built based on the training data of the five tasks. All words in the training data were included in the word vocabulary, and we employed the *word-dropout* method (Kiperwasser and Goldberg, 2016) to train the word embedding for the unknown words. We also built the character n-gram vocabulary for $n = 2, 3, 4$, following Wieting et al. (2016), and the character n-gram embeddings were initialized with the pre-trained embeddings. All of the label embeddings were initialized with uniform random values in $[-\sqrt{6/(dim + C)}, \sqrt{6/(dim + C)}]$, where $dim = 100$ is the dimensionality of the label embeddings and C is the number of labels.

Weight initialization The dimensionality of the hidden layers in the bi-LSTMs was set to 100. We initialized all of the softmax parameters and bias vectors, except for the forget biases in the LSTMs, with zeros, and the weight matrix W_d and the root node vector r for dependency parsing were also initialized with zeros. All of the forget biases were initialized with ones. The other weight matrices were initialized with uniform random values in $[-\sqrt{6/(row + col)}, \sqrt{6/(row + col)}]$, where row and col are the number of rows and columns of the matrices, respectively.

Optimization At each epoch, we trained our model in the order of POS tagging, chunking, dependency parsing, semantic relatedness, and textual entailment. We used mini-batch stochastic gradient decent to train our model. The mini-batch size was set to 25 for POS tagging, chunking, and the SICK tasks, and 15 for dependency parsing. We used a gradient clipping strategy with growing clipping values for the different tasks; concretely, we employed the simple function: $\min(3.0, depth)$, where $depth$ is the number of bi-LSTM layers involved in each task, and 3.0 is the maximum value. The learning rate at the k-th epoch was set to $\frac{\varepsilon}{1.0+\rho(k-1)}$, where ε is the initial learning rate, and ρ is the hyperparameter to decrease the learning rate. We set ε to 1.0 and ρ to 0.3. At each epoch, the same learning rate was shared across all of the tasks.

Regularization We set the regularization coefficient to 10^{-6} for the LSTM weight matrices, 10^{-5} for the weight matrices in the classifiers, and 10^{-3} for the successive regularization term excluding the classifier parameters of the lower-level tasks, respectively. The successive regularization coefficient for the classifier parameters was set to 10^{-2}. We also used *dropout* (Hinton et al., 2012). The dropout rate was set to 0.2 for the vertical connections in the multi-layer bi-LSTMs (Pham et al., 2014), the word representations and the label embeddings of the entailment layer, and the classifier

[*] Work was done while the first author was an intern at Salesforce Research.
[†] Corresponding author.

of the POS tagging, chunking, dependency parsing, and entailment. A different dropout rate of 0.4 was used for the word representations and the label embeddings of the POS, chunking, and dependency layers, and the classifier of the relatedness layer.

B Details of Character N-Gram Embeddings

Here we first describe the pre-training process of the character n-gram embeddings in detail and then show further analysis on the results in Table 12.

B.1 Pre-Training with Skip-Gram Objective

We pre-train the character n-gram embeddings using the objective function of the Skip-gram model with negative sampling (Mikolov et al., 2013). We build the vocabulary of the character n-grams based on the training corpus, the case-sensitive English Wikipedia text. This is because such case-sensitive information is important in handling some types of words like named entities. Assuming that a word w has its corresponding K character n-grams $\{cn_1, cn_2, \ldots, cn_K\}$, where any overlaps and unknown ones are removed. Then the word w is represented with an embedding $v_c(w)$ computed as follows:

$$v_c(w) = \frac{1}{K} \sum_{i=1}^{K} v(cn_i), \tag{1}$$

where $v(cn_i)$ is the parameterized embedding of the character n-gram cn_i, and the computation of $v_c(w)$ is exactly the same as the one used in our JMT model explained in Section 2.1.

The remaining part of the pre-training process is the same as the original Skip-gram model. For each word-context pair $(w, \overline{w})$ in the training corpus, N negative context words are sampled, and the objective function is defined as follows:

$$\sum_{(w,\overline{w})} \Bigg(-\log \sigma(v_c(w) \cdot \tilde{v}(\overline{w})) - \sum_{i=1}^{N} \log \sigma(-v_c(w) \cdot \tilde{v}(\overline{w}_i)) \Bigg), \tag{2}$$

where $\sigma(\cdot)$ is the logistic sigmoid function, $\tilde{v}(\overline{w})$ is the weight vector for the context word $\overline{w}$, and $\overline{w}_i$ is a negative sample. It should be noted that the weight vectors for the context words are parameterized for the words without any character information.

B.2 Effectiveness on Unknown Words

One expectation from the use of the character n-gram embeddings is to better handle unknown words. We verified this assumption in the single task setting for POS tagging, based on the results reported in Table 12. Table 13 shows that the joint use of the word and character n-gram embeddings improves the score by about 19% in terms of the accuracy for unknown words.

We also show the results of the single task setting for dependency parsing in Table 14. Again, we can see that using the character-level information is effective, and in particular, the improvement of the LAS score is large. These results suggest that it is better to use not only the word embeddings, but also the character n-gram embeddings by default. Recently, the joint use of word and character information has proven to be effective in language modeling (Miyamoto and Cho, 2016), but just using the simple character n-gram embeddings is fast and also effective.

C Analysis on Dependency Parsing

Our dependency parser is based on the idea of predicting a head (or parent) for each word, and thus the parsing results do not always lead to correct trees. To inspect this aspect, we checked the parsing results on the development set (1,700 sentences), using the "JMT$_{\mathrm{ABC}}$" setting.

In the dependency annotations used in this work, each sentence has only one root node, and we have found 11 sentences with multiple root nodes and 11 sentences with no root nodes in our parsing results. We show two examples below:

(a) Underneath the headline " Diversification , " it **counsels** , " Based on the events of the past week , all investors **need** to know their portfolios are balanced to help protect them against the market 's volatility . "

(b) Mr. Eskandarian , who resigned his Della Femina post in September , becomes chairman and chief executive of Arnold .

In the example (a), the two boldfaced words "counsels" and "need" are predicted as child nodes

Single (POS)	Overall Acc.	Acc. for unknown words
W&C	97.52	90.68 (3,502/3,862)
Only W	96.26	71.44 (2,759/3,862)

Table 13: POS tagging scores on the development set with and without the character n-gram embeddings, focusing on accuracy for unknown words. The overall accuracy scores are taken from Table 12. There are 3,862 unknown words in the sentences of the development set.

	Overall scores		Scores for unknown words	
Single (Dependency)	UAS	LAS	UAS	LAS
W&C	93.38	91.37	92.21 (900/976)	87.81 (857/976)
Only W	92.90	90.44	91.39 (892/976)	81.05 (791/976)

Table 14: Dependency parsing scores on the development set with and without the character n-gram embeddings, focusing on UAS and LAS for unknown words. The overall scores are taken from Table 12. There are 976 unknown words in the sentences of the development set.

of the root node, and the underlined word "counsels" is the correct one based on the gold annotations. This example sentence (a) consists of multiple internal sentences, and our parser misunderstood that both of the two verbs are the heads of the sentence.

In the example (b), none of the words is connected to the root node, and the correct child node of the root is the underlined word "chairman". Without the internal phrase "who resigned... in September", the example sentence (b) is very simple, but we have found that such a simplified sentence is still not parsed correctly. In many cases, verbs are linked to the root nodes, but sometimes other types of words like nouns can be the candidates. In our model, the single parameterized vector r is used to represent the root node for each sentence. Therefore, the results of the examples (a) and (b) suggest that it would be needed to capture various types of root nodes, and using sentence-dependent root representations would lead to better results in future work.

D Analysis on Semantic Tasks

We inspected the development set results on the semantic tasks using the "JMT_{all}" setting. In our model, the highest-level task is the textual entailment task. We show an example premise-hypothesis pair which is misclassified in our results:

> Premise: "A surfer is riding a *big* wave across dark green water", and
>
> Hypothesis: "The surfer is riding a *small* wave".

The predicted label is `entailment`, but the gold label is `contradiction`. This example is very easy by focusing on the difference between the two words "big" and "small". However, our model fails to correctly classify this example because there are few opportunities to learn the difference. Our model relies on the pre-trained word embeddings based on word co-occurrence statistics (Mikolov et al., 2013), and it is widely known that such co-occurrence-based embeddings can rarely discriminate between antonyms and synonyms (Ono et al., 2015). Moreover, the other four tasks in our JMT model do not explicitly provide the opportunities to learn such semantic aspects. Even in the training data of the textual entailment task, we can find only one example to learn the difference between the two words, which is not enough to obtain generalization capacities. Therefore, it is worth investigating the explicit use of external dictionaries or the use of pre-trained word embeddings learned with such dictionaries (Ono et al., 2015), to see whether our JMT model is further improved not only for the semantic tasks, but also for the low-level tasks.

E How Do Shared Embeddings Change

In our JMT model, the word and character n-gram embedding matrices are shared across all of the five different tasks. To better qualitatively explain the importance of the shortcut connections shown in Table 7, we inspected how the shared embeddings change when fed into the different bi-

LSTM layers. More concretely, we checked closest neighbors in terms of the cosine similarity for the word representations before and after fed into the forward LSTM layers. In particular, we used the corresponding part of W_u in Eq. (1) to perform linear transformation of the input embeddings, because u_t directly affects the hidden states of the LSTMs. Thus, this is a context-independent analysis.

Table 15 shows the examples of the word "standing". The row of "Embedding" shows the cases of using the shared embeddings, and the others show the results of using the linear-transformed embeddings. In the column of "Only word", the results of using only the word embeddings are shown. The closest neighbors in the case of "Embedding" capture the semantic similarity, but after fed into the POS layer, the semantic similarity is almost washed out. This is not surprising because it is sufficient to cluster the words of the same POS tags: here, `NN`, `VBG`, etc. In the chunking layer, the similarity in terms of verbs is captured, and this is because it is sufficient to identify the coarse chunking tags: here, `VP`. In the dependency layer, the closest neighbors are adverbs, gerunds of verbs, and nouns, and all of them can be child nodes of verbs in dependency trees. However, this information is not sufficient in further classifying the dependency labels. Then we can see that in the column of "Word and char", jointly using the character n-gram embeddings adds the morphological information, and as shown in Table 12, the LAS score is substantially improved.

In the case of semantic tasks, the projected embeddings capture not only syntactic, but also semantic similarities. These results show that different tasks need different aspects of the word similarities, and our JMT model efficiently transforms the shared embeddings for the different tasks by the simple linear transformation. Therefore, without the shortcut connections, the information about the word representations are fed into the semantic tasks after transformed in the lower layers where the semantic similarities are not always important. Indeed, the results of the semantic tasks are very poor without the shortcut connections.

	Word and char	Only word
Embedding	leaning	stood
	kneeling	stands
	saluting	sit
	clinging	pillar
	railing	cross-legged
POS	warning	ladder
	waxing	rc6280
	dunking	bethle
	proving	warning
	tipping	f-a-18
Chunking	applauding	fight
	disdaining	favor
	pickin	pick
	readjusting	rejoin
	reclaiming	answer
Dependency	guaranteeing	patiently
	resting	hugging
	grounding	anxiously
	hanging	resting
	hugging	disappointment
Relatedness	stood	stood
	stands	unchallenged
	unchallenged	stands
	notwithstanding	beside
	judging	exists
Entailment	nudging	beside
	skirting	stands
	straddling	pillar
	contesting	swung
	footing	ovation

Table 15: Closest neighbors of the word "standing" in the embedding space and the projected space in each forward LSTM.

References

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Easy-First Dependency Parsing with Hierarchical Tree LSTMs. *Transactions of the Association for Computational Linguistics*, 4:445–461.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119.

Yasumasa Miyamoto and Kyunghyun Cho. 2016. Gated Word-Character Recurrent Language Model. In *Proceedings of the 2016 Conference on Empiri-*

cal Methods in Natural Language Processing, pages 1992–1997.

Masataka Ono, Makoto Miwa, and Yutaka Sasaki. 2015. Word Embedding-based Antonym Detection using Thesauri and Distributional Information. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 984–989.

Vu Pham, Theodore Bluche, Christopher Kermorvant, and Jerome Louradour. 2014. Dropout improves Recurrent Neural Networks for Handwriting Recognition. *CoRR*, abs/1312.4569.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2016. CHARAGRAM: Embedding Words and Sentences via Character n-grams. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1504–1515.