Featureless Domain-Specific Term Extraction with Minimal Labelled Data

Rui Wang
The University of
Western Australia
35 Stirling Highway, Crawley,
W.A. 6009, Australia
rui.wang@research.uwa.edu.au

Wei Liu*
The University of
Western Australia
35 Stirling Highway, Crawley,
W.A. 6009, Australia
wei.liu@uwa.edu.au

Chris McDonald
The University of
Western Australia
35 Stirling Highway, Crawley,
W.A. 6009, Australia
chris.mcdonald@uwa.edu.au

Abstract

Supervised domain-specific term extraction often suffers from two common problems, namely labourious manual feature selection, and the lack of labelled data. In this paper, we introduce a weakly supervised bootstrapping approach using two deep learning classifiers. Each classifier learns the representations of terms separately by taking word embedding vectors as inputs, thus no manually selected feature is required. The two classifiers are firstly trained on a small set of labelled data, then independently make predictions on a subset of the unlabeled data. The most confident predictions are subsequently added to the training set to retrain the classifiers. This co-training process minimises the reliance on labelled data. Evaluations on two datasets demonstrate that the proposed co-training approach achieves a competitive performance with limited training data as compared to standard supervised learning baseline.

1 Introduction

Domain-specific terms are essential for many knowledge management applications, such as clinical text processing, risk management, and equipment maintenance. Domain-specific term extraction aims to automatically identify domain relevant technical terms that can be either unigram words or multi-word phrases. Supervised domain-specific term extraction often relies on the training of a binary classifier to identify whether or not a candidate term is relevant to the domain (da Silva Conrado et al., 2013; Foo and Merkel, 2010; Nazar and Cabré, 2012). In such approaches, term extraction models are built upon manually selected features including the local statistical and linguistic information (e.g. frequency, co-occurrence frequency, or linguistic patterns), and external information form third-party knowledge bases (e.g. WordNet, DBpedia). Designing and evaluating different feature combinations turn the development of term extraction models into a time-consuming and labor-intensive exercise. In addition, these approaches require a large amount of labelled training data to generalise the learning. However, labelled data is often hard or impractical to obtain.

In this paper, our first objective is to minimise the usage of labelled data by training two classifiers in a co-training fashion. Co-training is a weakly supervised learning mechanism introduced by Blum and Mitchell (1998), which tackles the problem of building a classification model from a dataset with limited labelled data among the majority of unlabelled ones. It requires two classifiers, each built upon separate *views* of the data. Each view represents a separate set of manually selected features that must be sufficient to learn a classifier. For example, Blum and Mitchell classify web pages based on words appearing in the content of a web page, and words in hyperlinks pointing to the web page. Co-training starts with training each classifier on a small labelled dataset, then each classifier is used to predict a subset of the unlabelled data. The most confident predictions are subsequently added to the training set to re-train each classifier. This process is iterated a fixed number of times. Co-training algorithms have been applied to many NLP tasks where labelled data are in scarce, including statistical parsing (Sarkar, 2001), word sense disambiguation (Mihalcea, 2004), and coreference resolution (Ng and Cardie, 2003), which demonstrate that it generally improves the performance without requiring additional labelled data.

Our second objective is to eliminate the ef-

*Corresponding author

Rui Wang, Wei Liu and Chris McDonald. 2016. Featureless Domain-Specific Term Extraction with Minimal Labelled Data.
In *Proceedings of Australasian Language Technology Association Workshop*, pages 103−112.

fort of feature engineering by using deep learning models. Applying deep neural networks directly to NLP tasks without feature engineering is also described as NLP from scratch (Collobert et al., 2011). As a result of such training, words are represented as low dimensional and real-valued vectors, encoding both semantic and syntactic features (Mikolov et al., 2013). In our model, word embeddings are pre-trained over the corpora to encode word features that are used as inputs to two deep neural networks to learn different term representations (corresponding to the concept of *views*) over the same dataset. One is a convolutional neural network (CNN) to learn term representations through a single convolutional layer with multiple filters followed by a max pooling layer. Each filter is associated with a region that essentially corresponds to a sub-gram of a term. The underlying intuition is that the meaning of a term can be learnt from its sub-grams by analysing different combinations of words. The other is a Long Short-Term Memory (LSTM) network which learns the representation of a term by recursively composing the embeddings of an input word and the composed value from its precedent, hypothesising that the meaning of a term can be learnt from the sequential combination of each constituent word. Each network connects to a logistic regression layer to perform classifications.

Our model is evaluated on two benchmark domain-specific corpora, namely GENIA (biology domain) (Kim et al., 2003), and ACL RD-TEC (computer science domain) (Handschuh and QasemiZadeh, 2014). The evaluation shows that our model outperforms the C-value algorithm (Frantzi et al., 2000) that is often treated as the benchmark in term extraction. We also trained two classifiers using the standard supervised learning approach, and demonstrate that co-training deep neural networks is an effective approach to reduce the usage of labelled data while maintaining a competitive performance.

This paper is organised as follows. In Section 2, we briefly review the related work. Section 3 introduces our proposed model, and in Section 4 we describe our evaluation datasets and discuss the experimental results. Section 5 summarises our study with an outlook to the future work.

2 Related Work

2.1 Supervised Term Extraction

Supervised machine learning approaches for domain-specific term extraction start with candidate identification, usually by employing a phrase chunker based on pre-identified part-of-speech (POS) patterns, then uses manually selected features to train a classifier. The feature may include linguistic, statistical, and semantic features (Foo and Merkel, 2010; Nazar and Cabré, 2012; da Silva Conrado et al., 2013). The work closely related to ours is *Fault-Tolerant Learning* (FTL) (Yang et al., 2010) inspired by Transfer Learning (Ando and Zhang, 2005) and Co-training (Blum and Mitchell, 1998). FTL builds two *support vector machine* (SVM) classifiers using manually selected features, whereas our model uses deep neural networks taking pre-trained word embeddings as inputs, without using any manually selected feature.

2.2 Learning Representations of Words and Semantic Compositionality

Word embeddings are proposed to overcome the *curse of dimensionality* problem by Bengio et al. (2006), who developed a probabilistic neural language model using a feed-forward multi-layer neural network, which demonstrates how word embeddings can be induced. Recently, Mikolov et al. (2013) presented a shallow network architecture that is specifically for learning word embeddings, known as the *word2vec* model. Their work reveals that word embeddings are capable of representing the meaning of words to a certain degree through analogy test.

Semantic composition is to understand the meaning of a multi-word expression by composing the meanings of each constituent word. For example, given the word embeddings of *pet* and *doctor*, the compositional representation of *pet doctor* is expected to have the high cosine similarity or small Euclidian distance to the vector of *veterinarian*. Such characteristics help classifiers to recognise the semantically related expressions or equivalent counterparts of a term, which offers significant advantages for term extraction. Popular deep neural networks for modelling semantic compositionality include *convolutional*, *recurrent*, *recursive*, and *hybrid* networks. Kim (2014) reported a number of experiments of applying a convolutional network with one single

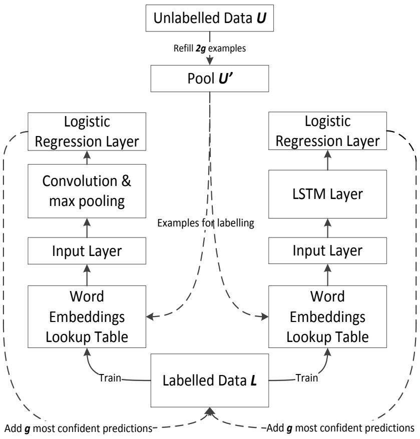
Unlabelled Data U
Refill 2g examples
Pool U'
Logistic Regression Layer
Convolution & max pooling
Input Layer
Word Embeddings Lookup Table
Logistic Regression Layer
LSTM Layer
Input Layer
Word Embeddings Lookup Table
Examples for labelling
Train
Labelled Data L
Train
Add g most confident predictions
Add g most confident predictions

Figure 1: Co-training Network Architecture Overview: Solid lines indicate the training process, dashed lines indicate prediction and labelling processes.

convolutional and 1-Max pooling layer for *Sentiment Analysis* and *Topic Categorisation* tasks producing the state-of-the-art performance on 4 out of 7 datasets, which shows the power of the convolutional architecture. Recurrent neural network is efficient on encoding sequential combinations of data with various lengths, which naturally offers an advantage for capturing semantic compositionality of multi-word terms. Using recurrent neural networks modelling natural languages and their long-term dependencies was attempted by Mikolov et al. (2010). More recently, Sutskever et al. (2014) use LSTM network, and Cho et al. (2014) use *Gated Recurrent Unit* (GRU) network to encode and decode the semantic compositionality of sentences for machine translation. Chung et al. (2015) propose an even deeper architecture named Gated Feedback Recurrent network that stacks multiple recurrent layers for character-level language modelling. Other network architectures for learning semantic compositionality include recursive networks (Socher et al., 2010; Socher et al., 2012), which require using POS tagging texts to produce syntactical tree structures as a prior. Hybrid networks, such as recurrent-convolutional network (Kalchbrenner and Blunsom, 2013; Lai et al., 2015), are designed for capturing document-level semantics.

3 Proposed Model

The model consists of two classifiers, as shown in Figure 1. The left classifier is a CNN network, and the right one is a LSTM network. Both networks take pre-trained word embedding vectors as inputs to learn the representations of terms independently. The output layer is a logistic regression layer for both networks. Two neural networks are trained using the Co-training algorithm.

The Co-training algorithm requires two separate views of the data, which traditionally are two sets of manually selected features. In our model, however, there is no manually selected features. Thus, two views of the data are carried by our two hypotheses of learning the meaning of terms. The meaning of a term can be learnt by 1) analysing different sub-gram compositions, and 2) sequential combination of each constituent word. The hypotheses are implemented via the CNN and LSTM network. We expect that the rules of composing words can be captured by the networks. The CNN network analyses different regions of a input matrix that is constructed by stacking word embedding vectors, as shown in Figure 2, where the size of regions reflect different n-grams of a term. By scanning the embedding matrix with different region sizes, we expect that the CNN network can learn the meaning of a term by capturing the most representative sub-gram. The LSTM network, on the other hand, learns the compositionality by recursively composing an input embedding vector with the precedent or previously composed value, as shown in Figure3. We expect the LSTM network to capture the meaning of a term through its gates that govern the information flow – how much information (or meaning) of an input word can be added in to the overall meaning, and how much information should be dismissed from the previous composition.

3.1 Term Representation Learning

The objective is to learn a mapping function f that outputs the compositional representation of a term given its word embeddings. Concretely, let V be the vocabulary of a corpus with the size of v. For each word $w \in V$, there is a corresponding d dimensional embedding vector. The collection of all embedding vectors in the vocabulary is a matrix, denoted as C, where $C \in \mathbb{R}^{d \times v}$. C can be thought of as a look-up table, where $C(w_i)$ represents the embedding vectors of word w_i. Given a term $s = (w_1, w_2, ..., w_n)$, the goal is to learn a mapping function $f(C(s))$ that takes the individual vector representation of each word as in-

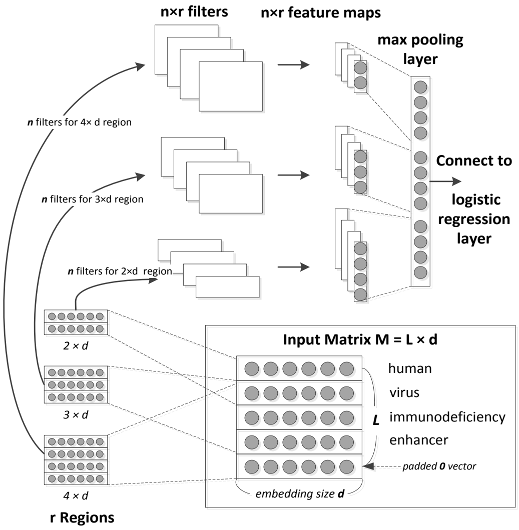

Figure 2: Convolutional Model

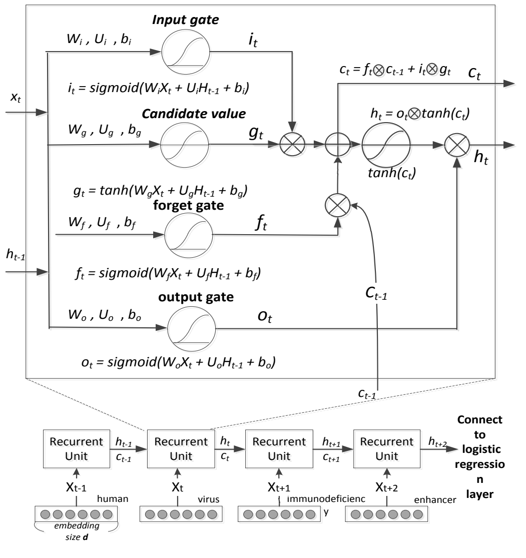

Figure 3: LSTM Model

puts, and output a composed value that represent the compositional meaning of s.

3.1.1 Convolutional Model

We adopt the CNN network used by Kim (2014), and Zhang and Wallace (2015), which has only one convolutional layer, shown in Figure 2. The inputs $C(s)$ to the network are vertically stacked into a matrix M, where the rows are word embeddings of each $w \in s$. Let d be the length of word embedding vectors, and l be the length of a term, then M has the dimension of $d \times l$ where d and l are fixed. We pad zero vectors to the matrix if the number of tokens of an input term is less than l. The convolutional layer has r predefined regions, and each region has n filters. All regions have the same width d, because each row in the input matrix M represents a word and the goal is to learn the composition of them. However, the regions have various heights h, which can be thought of as different *n-gram* models. For example, when $h = 2$, the region is to represent bigram features. Let W be the weights of a filter, where $W \in \mathbb{R}^{d \times h}$. The filter outputs a feature map $c = [c_1, c_2, ..., c_{l-h+1}]$, and c_i is computed as:

$$c_i = f(W \cdot M[i : i + h - 1] + b) \qquad (1)$$

where $M[i : i + h - 1]$ is a sub-matrix of M from row i to row $i + h - 1$, f is an activation function – we use hyperbolic tangent in this work, and b is a bias unit. A pooling function is then applied to extract values from the feature map. We use the *1*-Max pooling as suggested by Zhang and Wallace (2015) who conduct a sensitivity analysis on one-layer convolutional network showing that *1*-Max pooling consistently outperforms other pooling strategies in sentence classification task. The total number of feature maps in the network is $m = r \times n$, so the output from the max pooling layer $y \in \mathbb{R}^m$ is computed as:

$$y = \max_{i=1}^{m}(c^i) \qquad (2)$$

3.1.2 LSTM Model

We use a LSTM network that is similar to the *vanilla* LSTM (Greff et al., 2015) without peephole connections, shown in Figure 3. The LSTM network features memory cells at each timestamp. A memory cell is to store, read and write information passing through the network at a timestamp t, which consists of four elements, an input gate i, a forget gate f, a candidate g for the current cell state value, and an output gate o. At t, the inputs to the network are the previous cell state value c_{t-1}, the previous hidden state value h_{t-1} , and the input value x_t. The outputs are current cell state c_t and the current hidden state value h_t, which will pass to the next timestamp $t + 1$. At time t, the candidate g for the current cell state value compose the newly input x_t and the previously composed value h_{t-1} to generate a new state value as:

$$g_t = \tanh(W_g \cdot x_t + U_g \cdot h_{t-1} + b_g) \qquad (3)$$

where W_g and U_g are shared weights, and b_i is the bias unit.

The input gate i in the LSTM network decides how much information can pass through from g_t to the actual computation of the memory cell state using a *sigmoid* function $\sigma = \frac{1}{1+e^{-x}}$ that outputs a value between 0 and 1 indicating the percentage, as:

$$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i) \qquad (4)$$

where W_i and U_i are shared weights, and b_i is the bias unit. Likewise, the forget gate f governs how much information to be filtered out from the previous state c_{t-1}:

$$f_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f) \qquad (5)$$

The new cell state value is computed as taking a part of information from the current inputs and the previous cell state value:

$$c_t = i_t \otimes g_t + f_t \otimes c_{t-1} \qquad (6)$$

where $\otimes$ is the element-wise vector multiplication. c_t will be passed to the next timestamp $t+1$, which remains constant from one timestamp to another, representing the *long-short term* memory.

The output gate o can be thought of as the filter that prevents any irrelevant information that may be passed to the next state. The output gate o_t and the hidden state value h_t are computed as:

$$\begin{aligned} o_t &= \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o) \\ h_t &= o_t \otimes \tanh(c_t) \end{aligned} \qquad (7)$$

where h_t is the composed representation of a word sequence from time 0 to t.

3.2 Training Classifier

To build the classifiers, each network is connected to a logistic regression layer for the binary classification task – we are only concerned whether a term is domain relevant or not. The logistic regression layer, however, can be simply replaced by a *softmax* layer for multi-class classification tasks, such as *Ontology Categorisation*.

Overall, the probability that a term s is relevant to the domain is:

$$p(s) = \sigma(W \cdot f(C(s)) + b) \qquad (8)$$

where σ is the sigmoid function, W is the weights for logistic regression layer, b is the bias unit, and f is the mapping function that is implemented by the CNN or the LSTM network.

The parameters of convolutional classifier are θ = (C, W^{conv}, b^{conv}, $W^{convlogist}$, $b^{convlogist}$) where W^{conv} are weights for all m filters, and b^{conv} is the bias vectors. For LSTM classifier, θ = (C, W^{lstm}, b^{lstm}, $W^{lstmlogist}$, $b^{lstmlogist}$) where W^{lstm} = (W_i, W_g, W_f, W_o, U_i, U_g, U_f, U_o), and b^{lstm} = (b_i, b_g, b_f, b_o). Given a training set D, the learning objective for both of the classifiers is to maximise the log probability of correct labels for $s \in D$ by looking for parameters θ:

$$\underset{\theta}{\operatorname{argmax}} \sum_{s \in D} \log p(s_{label} | s; \theta) \qquad (9)$$

θ is updated using *stochastic gradient descent* (SGD) to minimise the negative log likelihood error:

$$\theta := \theta - \varepsilon \frac{\partial \log p(s_{label} | s; \theta)}{\partial \theta} \qquad (10)$$

where ε is the learning rate.

3.3 Pre-training Word Embedding

We use the SkipGram model (Mikolov et al., 2013) to learn word embeddings. Given a word w, the SkipGram model predicts the context (surrounding) words $S(w)$ within a pre-defined window size. Using the *softmax* function, the probability of a context word $s \in S$ is:

$$p(s|w) = \frac{e^{{v'_w}^\top \cdot v_s}}{\sum_{t=1}^{V} e^{{v'_t}^\top \cdot v_s}} \qquad (11)$$

where V is the vocabulary, v'_w is the output vector representations for w, v_s is the input vector representations for contexts s, respectively. The learning objective is to maximise the conditional probability distribution over vocabulary V in a training corpus D by looking for parameters θ:

$$\underset{\theta}{\operatorname{argmax}} \sum_{w \in D} \sum_{s \in S(w)} \log p(s|w; \theta) \qquad (12)$$

3.4 Co-training Algorithm

Given the unlabelled data U, a pool U' of size p, and a small set of labelled data L, firstly each classifier $c \in C$ are trained over L. After training, the classifiers make predictions on U', then choose the most confident g predictions from each classifier and add them to L. The size of U' now becomes $p - 2g$, and $L := L + 2g$. U' then is refilled by randomly selecting $2g$ examples from U. This process iterates k times. Algorithm 1 documents the details.

Algorithm 1 Co-training

```
Input: $L, U, C, p, k, g$
  create $U'$ by randomly choosing $p$ example from $U$
  while $iteration < k$ do
    for $c \in C$ do
      use L to train $C$
    end for
    for $c \in C$ do
      use $c$ to posit label in $U'$
      add most confident $g$ example to $L$
    end for
    refill $U'$ by randomly choosing $2 \times g$ example from $U$
  end while
```

4 Experiments

4.1 Datasets

We evaluate our model on two datasets. The first dataset is the GENIA corpus[1]. The current Version 3.02 is a collection of 1,999 article abstracts in the field of *molecular biology* including 451,562 tokens and 30,893 ground truth terms. The second dataset is the ACL RD-TEC corpus[2], which consists of 10,922 articles published between 1965 to 2006 in the domain of *computer science*. The ACL RD-TEC corpus classifies terms into three categories, invalid terms, general terms, and computational terms. We only treat computational terms as ground truth in our evaluation. The dataset has 36,729,513 tokens and 13,832 ground truth terms.

4.2 Preprocessing

We firstly clean the datasets by extracting text content and ground truth terms, removing plurals of nouns, and converting all tokens into lowercases. The ACL RD-TEC corpus provides a pre-identified candidate list, so we only need to identify candidate terms from the GENIA corpus. We build two candidate identifiers. The first identifier uses noun phrase chunking with pre-defined POS patterns, we call it *POS identifier*. We use a common POS pattern `<JJ>*<NN.*>+`, that is, a number of adjectives followed by a number of nouns. The second identifier uses n-gram based chunking, so called *N-gram identifier*, which decomposes a sequence of words into all possible n-grams. However, there would be too many candidates if we simply decompose every sentence into all possible n-grams. Thus, we use stop-words as delimiters to decompose any expression between two stop-words into all possible n-grams as candidates. For example, *novel antitumor antibiotic* produces 6 candidates, *novel antitumor antibiotic*, *novel antitumor*, *antitumor antibiotic*, *novel*, *antitumor*, and *antibiotic*.

4.3 Experiment Settings

The co-training requires a few parameters. We set the small set of labelled data $L = 200$ and the size for the pool U' 500 for all evaluations. The number of iterations k is 800 for POS identified candidates, 500 for N-grams identified candidates in the GENIA dataset, 500 for the ACL RD-TED dataset. The growth size is 20 for POS, 50 for N-grams, and 20 for ACL RD-TED. The evaluation data is randomly selected from candidate sets. For each e in the evaluation set E, $e \notin L$. Table 1 show the class distributions and statistics.

All word embeddings are pre-trained with 300 dimensions on each corpus. The maximum length of terms is 13 on GENIA and 5 on ACL RD-TED, therefore the CNN classifier has 5 different region size , {2, 3, 4, 5, 6} for GENIA and 3 region size {2, 3, 4} for ACL RD-TED. Each region has 100 filters. There are no specific hyper-parameters required for training the LSTM model. The learning rate for SGD is 0.01.

The model was trained in an *online* fashion. We trained our model on a NVIDIA GeForce 980 TI GPU. The training time linearly increases at each iteration, since the model incrementally adds training examples into the training set. At the beginning, it only took less than a second for one iteration. After 100 iterations, the training set was increased by 1,820 examples that took a few seconds to train. Thus the training time is not critical – even the standard supervised training only took a few hours to converge.

4.4 Evaluation Methodology

We use $precision = \frac{TP}{TP+FP}$, $recall = \frac{TP}{TP+FN}$, $F = 2 \times \frac{precision \times recall}{precision + recall}$ and $accuracy = \frac{TP+TN}{TP+FP+FN+TN}$ for our evaluation. We illustrate the set relationships of *true postive* (TP) , *true negative* (TN), *false postive* (FP), and *false negative* (FN) in Figure 4.

[1] publicly available at http://www.geniaproject.org/

[2] publicly available at https://github.com/languagerecipes/the-acl-rd-tec

Table 1: Evaluation Dataset Statistics

	Test Examples	Positive	Negative
GENIA POS	5,000	2558 (51.0%)	2442 (49.0%)
GENIA N-gram	15,000	1,926 (12.8%)	13,074 (87.2%)
ACL RD-TEC	15,000	2,416 (16.1%)	12,584 (83.9%)

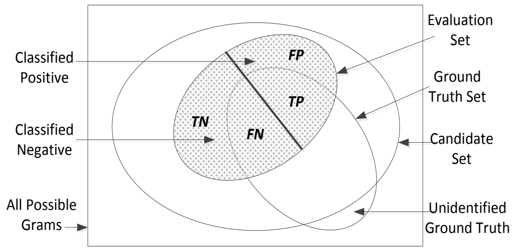

Figure 4: Relationships in TP, TN, FP, and FN for Term Extraction.

4.5 Result and Discussion

We use C-value (Frantzi et al., 2000) as our baseline algorithm. C-value is an unsupervised ranking algorithm for term extraction, where each candidate term is assigned a score indicating the degree of domain relevance. We list the performance of C-value by extracting top scored terms. Since we treat the task as a binary classification task, we also list random guessing scores for each dataset, where recall and accuracy scores are always 50% and precisions correspond to the distribution of positive class of each evaluation set. As a comparison to the Co-training model, we also trained each classifier individually using the standard supervised machine learning approach. The training is conducted by dividing the candidate set into 40% for training, 20% for validation, and 40% for evaluation. For Co-training model, we found that the CNN classifier outperforms the LSTM classifier in all the evaluation, so we only present the performance of the CNN classifier, as shown in Table 2.

The supervised approach unsurprisingly produces the best results on all evaluation sets. However, it uses much more labelled data than the Co-training model, while delivering only 2 percent better performance (F-score) on the GENIA corpus, and 6 percent on the ACL RD-TEC corpus. In comparison to standard supervised machine learning approach, Co-training is more "cost-effective" since it only requires 200 labelled data as seed terms.

On the GENIA corpus, all algorithms produce much better F-score for the POS evaluation set. This is because of different class distributions – on the POS evaluation set, the proportion of positive (ground truth) terms is 50.5% whereas only 12.8% positive terms in the N-gram evaluation set. Therefore, we consider that the results from POS and N-gram evaluation sets are not directly comparable. However, the actual improvements on F-score over random guessing on both evaluation sets are quite similar, suggesting that evaluating performance of a classifier should not only consider the F-score, but should also analyse the actual improvement over random guessing.

It is also interesting to note that the GENIA N-gram evaluation set has 12.8% positive examples, which has similar unbalanced class distribution as ACL RD-TED, 16.1% positives. However, all algorithms produce much better performance on the ACL RD-TEC corpus. We found that in the ACL RD-TEC corpus, the negative terms contain a large number of invalid characters (e.g.˜), mistakes made by the tokeniser (e.g. `evenunseeneventsare`), and none content-bearing words (e.g. `many`). The classifiers can easily spot these noisy data. Another reason might be that the ACL RD-TEC corpus is bigger than GENIA, which not only allows C-value to produce better performance, but also enables the *word2vec* algorithm to deliver more precise word embedding vectors which are required inputs to our deep learning model.

Although the accuracy measure is commonly used in classification tasks, it does not reflect the true performance of a model when classes are not evenly distributed in an evaluation set. For example, on the N-gram evaluation set, the positive examples are about 12.8% whereas the *negative* examples are about 87.2%. At the beginning of the training, both models tend to classify most of the examples as negative thus the accuracy score is close to 87%. While the training progress, the accuracy starts dropping. However, it is still difficult

Table 2: Evaluation Results

	Labelled Data	Precision	Recall	F-score	Accuracy
GENIA POS					
Random Guessing	–	51%	50%	50.5%	50%
C-value (Top 1000)	–	62.4%	24.4%	35.1%	–
C-value (Top 2500)	–	53.7%	52.5%	53.1%	–
Supervised	16,400	**64.7%**	**78.0%**	**70.7%**	67.1%
Co-training	200	64.1%	76.0%	69.5%	65.5%
GENIA N-gram					
Random Guessing	–	12.8%	50%	20.4%	50%
C-value (Top 2500)	–	12.9%	16.7%	14.6%	–
C-value (Top 7500)	–	11.4%	44.3%	18.1%	–
Supervised	91,924	**35.0%**	**59.1%**	**44.0%**	81.4%
Co-training	200	34.3%	56.6%	42.7%	75.5%
ACL RD-TEC					
Random Guessing	–	16.1%	50%	24.4%	50%
C-value (Top 2500)	–	14%	14.6%	14.3%	–
C-value (Top 7500)	–	21.8%	68.2%	33.3%	–
Supervised	33,538	**70.8%**	**67.7%**	**69.2%**	85.2%
Co-training	200	66%	60.5%	63.1%	79.7%

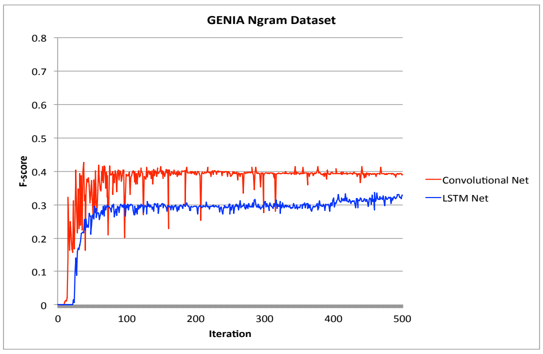

Figure 5: Performance on Ngram evaluation set

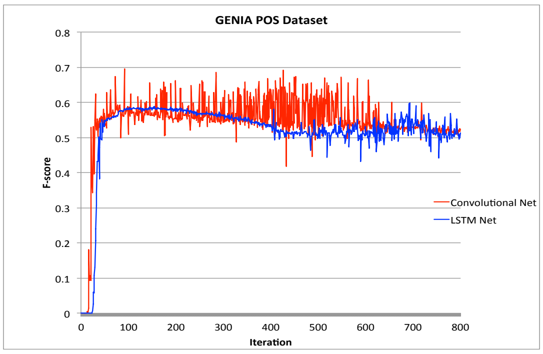

Figure 6: Performance on POS evaluation set

to understand how exactly the model performs according to the accuracy score. On the other hand, because the classes are evenly distributed on POS evaluation set, we can clearly identify how the accuracy measure corresponds to F-scores.

The CNN classifier outperforms the LSTM on all evaluation sets. It also requires much fewer iterations to reach the best F-score. We plot F-score for both classifiers over a few hundreds iterations on the GENIA corpus, shown in Figure 5 and 6. Both classifiers reach their best performance within 100 iterations. For example, the CNN classifier on POS evaluation set, produced a good F-score around 62% at just about 30 iterations, then reached its best F-score 69.5% after 91 iterations. However, the size of the training set is still quite small – by 91 iterations, the training set only grows by 1,820 examples. This phenomenon leads us to consider two more questions 1) what is the exact performance boosted by Co-training? 2) How different numbers of training examples affect the performance of a deep learning model, and do deep learning models still need large amount of labelled training examples to produce the best performance? In the rest of the paper, we will answer the first question, and leave the second question for our future work.

To investigate how Co-training boosts the performance of classifiers, we trained our model using only 200 seed terms over 800 iterations, results are shown in Figure 7. The best F-score is from the convolutional model, about 53%, just slightly

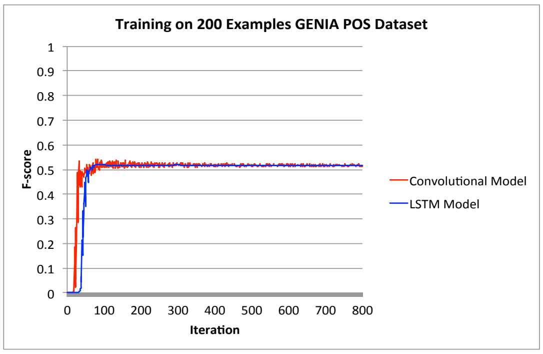

Figure 7: Convolutional and LSTM Classifier Training on 200 Examples on POS Evaluation set

higher than random guessing. On the other, by applying Co-training we obtain the best F-score of 69.5% which is a 16.5% improvement. In fact, the improvement achieved by just adding a small number of training examples to the training set was also report by (Clark et al., 2003). Consequently, it is clear that our co-training model is an effective approach to boost the performance of deep learning models without requiring much training data.

5 Conclusion

In this paper, we have shown a deep learning model using Co-training – a weakly supervised bootstrapping paradigm, for automatic domain-specific term extraction. Experiments show that our model is a "cost-effective" way to boost the performance of deep learning models with very few training examples. The study also leads to further questions such as how the number of training examples affects the performance of a deep learning model, and whether deep learning models still need as many labelled training examples as required in other machine learning algorithms to reach their best performance. We will keep working on these question in the near future.

6 Acknowledgment

This research was funded by the Australian Postgraduate Award (APA), Safety Top-Up Scholarships from The University of Western Australia, and partially funded by Australian Research Council grants DP150102405 and LP110100050.

References

Rie Kubota Ando and Tong Zhang. 2005. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6(Nov):1817–1853.

Yoshua Bengio, Holger Schwenk, Jean-Sébastien Senécal, Fréderic Morin, and Jean-Luc Gauvain. 2006. Neural probabilistic language models. In *Innovations in Machine Learning*, pages 137–186. Springer.

Avrim Blum and Tom Mitchell. 1998. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100. ACM.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Junyoung Chung, Caglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. 2015. Gated feedback recurrent neural networks. *CoRR, abs/1502.02367*.

Stephen Clark, James R Curran, and Miles Osborne. 2003. Bootstrapping pos taggers using unlabelled data. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 49–55. Association for Computational Linguistics.

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.

Merley da Silva Conrado, Thiago Alexandre Salgueiro Pardo, and Solange Oliveira Rezende. 2013. A machine learning approach to automatic term extraction using a rich feature set. In *HLT-NAACL*, pages 16–23.

Jody Foo and Magnus Merkel. 2010. Using machine learning to perform automatic term recognition. In *LREC 2010 Workshop on Methods for automatic acquisition of Language Resources and their evaluation methods, 23 May 2010, Valletta, Malta*, pages 49–54.

Katerina Frantzi, Sophia Ananiadou, and Hideki Mima. 2000. Automatic recognition of multi-word terms:. the c-value/nc-value method. *International Journal on Digital Libraries*, 3(2):115–130.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. 2015. Lstm: A search space odyssey. *arXiv preprint arXiv:1503.04069*.

Siegfried Handschuh and Behrang QasemiZadeh. 2014. The acl rd-tec: A dataset for benchmarking terminology extraction and classification in computational linguistics. In *COLING 2014: 4th International Workshop on Computational Terminology*.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent convolutional neural networks for discourse compositionality. *arXiv preprint arXiv:1306.3584*.

J-D Kim, Tomoko Ohta, Yuka Tateisi, and Junichi Tsujii. 2003. Genia corpusa semantically annotated corpus for bio-textmining. *Bioinformatics*, 19(suppl 1):i180–i182.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent convolutional neural networks for text classification. In *AAAI*, pages 2267–2273.

Rada Mihalcea. 2004. Co-training and self-training for word sense disambiguation. In *CoNLL*, pages 33–40.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH*, volume 2, page 3.

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*, pages 3111–3119.

Rogelio Nazar and Maria Teresa Cabré. 2012. Supervised learning algorithms applied to terminology extraction. In *Proceedings of the 10th Terminology and Knowledge Engineering Conference*, pages 209–217.

Vincent Ng and Claire Cardie. 2003. Weakly supervised natural language learning without redundant views. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 94–101. Association for Computational Linguistics.

Anoop Sarkar. 2001. Applying co-training methods to statistical parsing. In *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, pages 1–8. Association for Computational Linguistics.

Richard Socher, Christopher D Manning, and Andrew Y Ng. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, pages 1–9.

Richard Socher, Brody Huval, Christopher D Manning, and Andrew Y Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1201–1211. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Yuhang Yang, Hao Yu, Yao Meng, Yingliang Lu, and Yingju Xia. 2010. Fault-tolerant learning for term extraction. In *PACLIC*, pages 321–330.

Ye Zhang and Byron Wallace. 2015. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*.