Neural Network for Heterogeneous Annotations

Hongshen Chen[*†§] **Yue Zhang**[‡] **Qun Liu**[♢†]
†Key Laboratory of Intelligent Information Processing
Institute of Computing Technology, Chinese Academy of Sciences
§University of Chinese Academy of Sciences
‡Singapore University of Technology and Design
♢ ADAPT centre, School of Computing, Dublin City University
chenhongshen@ict.ac.cn, yue_zhang@sutd.edu.sg

Abstract

Multiple treebanks annotated under heterogeneous standards give rise to the research question of best utilizing multiple resources for improving statistical models. Prior research has focused on discrete models, leveraging stacking and multi-view learning to address the problem. In this paper, we empirically investigate heterogeneous annotations using neural network models, building a neural network counterpart to discrete stacking and multi-view learning, respectively, finding that neural models have their unique advantages thanks to the freedom from manual feature engineering. Neural model achieves not only better accuracy improvements, but also an order of magnitude faster speed compared to its discrete baseline, adding little time cost compared to a neural model trained on a single treebank.

1 Introduction

For many languages, multiple treebanks have been annotated according to different guidelines. For example, several linguistic theories have been used for defining English dependency treebanks, including Yamada and Matsumoto (2003), LTH (Johansson and Nugues, 2007) and Stanford dependencies (De Marneffe et al., 2006). For German, there exist TIGER (Brants et al., 2002) and TüBa-D/Z (Telljohann et al., 2006). For Chinese, treebanks have been made available under various segmentation granularities (Sproat and Emerson, 2003; Emerson, 2005; Xue, 2003). These give rise to the research problem of effectively making use of multiple treebanks under heterogeneous annotations for improving output accuracies (Jiang et al., 2015; Johansson, 2013; Li et al., 2015).

The task has been tackled using two typical approaches. The first is based on *stacking* (Wolpert, 1992; Breiman, 1996; Wu et al., 2003). As shown in Figure 1(a), the main idea is to have a model trained using a source treebank, which is then used to guide a target treebank model by offering source-style features. This method has been used for leveraging two different treebanks for word segmentation (Jiang et al., 2009; Sun and Wan, 2012) and dependency parsing (Nivre and McDonald, 2008; Johansson, 2013).

The second approach is based on *multi-view learning* (Johansson, 2013; Li et al., 2015). The idea is to address both annotation styles simultaneously by sharing common feature representations. In particular, Johansson (2013) trained dependency parsers using the domain adaptation method of Daumé III (2007), keeping a copy of shared features and a separate copy of features for each treebank. Li et al. (2015) trained POS taggers by coupling the labelsets from two different treebanks into a single combined labelset. A summary of such multi-view methods is shown in Figure 1(b), which demonstrates their main differences compared to stacking (Figure 1(a)).

Recently, neural network has gained increasing research attention, with highly competitive results being reported for numerous NLP tasks, including word segmentation (Zheng et al., 2013; Pei et al., 2014; Chen et al., 2015), POS-tagging (Ma et al., 2014; Plank et al., 2016), and parsing (Chen and

[*]Work done when the first author was visiting SUTD.

Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 731–741,
Austin, Texas, November 1-5, 2016.

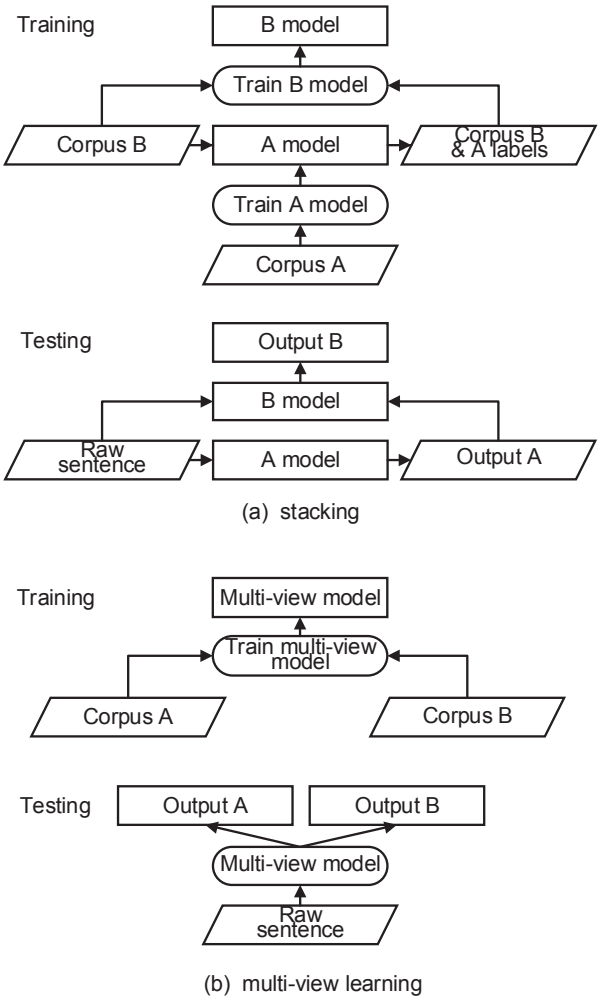

Figure 1: Two main approaches to utilizing heterogeneous annotations.

Manning, 2014; Dyer et al., 2015; Weiss et al., 2015; Zhou et al., 2015). On the other hand, the aforementioned methods on heterogeneous annotations are investigated mainly for discrete models. It remains an interesting research question how effective multiple treebanks can be utilized by neural NLP models, and we aim to investigate this empirically.

We follow Li et al. (2015), taking POS-tagging for case study, using the methods of Jiang et al. (2009) and Li et al. (2015) as the discrete stacking and multi-view training baselines, respectively, and building neural network counterparts to their models for empirical comparison. The base tagger is a neural CRF model (Huang et al., 2015; Lample et al., 2016), which gives competitive accuracies to discrete CRF taggers.

Results show that *neural stacking* allows deeper integration of the source model beyond one-best outputs, and further the fine-tuning of the source model during the target model training. In addition, the advantage of *neural multi-view learning* over its discrete counterpart are many-fold. First, it is free from the necessity of manual cross-labelset interactive feature engineering, which is far from trivial for representing annotation correspondence (Li et al., 2015). Second, compared to discrete model, parameter sharing in deep neural network eliminates the issue of exponential growth of search space, and allows separated training of each label type, in the same way as multi-task learning (Collobert et al., 2011).

Our neural multi-view learning model achieves not only better accuracy improvements, but also an order of magnitude faster speed compared to its discrete baseline, adding little time cost compared to a neural model trained on a single treebank. The C++ implementations of our neural network stacking and multi-view learning models are available under GPL, at `https://github.com/chenhongshen/NNHetSeq`.

2 Baseline Neural Network Tagger

We adopt a neural CRF with a Long-Short-Term-Memory (LSTM) (Hochreiter and Schmidhuber, 1997) feature layer for baseline POS tagger. As shown in Figure 2, the model consists of three main neural layers: the **input layer** calculates dense representation of input words using attention model on character embeddings; the **feature layer** employs a bi-directional LSTM model to extract non-local features from input vectors; the **output layer** uses a CRF structure to infer the most likely label for each input word.

2.1 Input Layer

Given a sentence $w^{(1:n)}$, the input layer builds a vector representation $\vec{r}_w^i$ for each word w^i based on both word and character embeddings. In particular, an embedding lookup table is used to convert vocabulary words into their embedding forms directly. To obtain a character based embedding of w^i, we denote the character sequence of w^i with $c^{(1:n)}$, where c^j is the jth character in w^i.

A character lookup table is used to map each char-

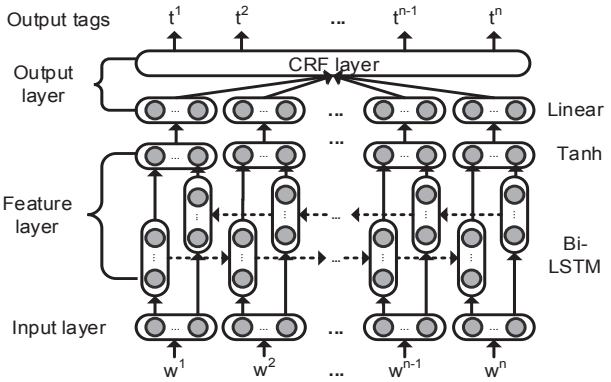

Figure 2: Baseline neural network tagger.

acter c^j into a character embedding $\vec{e}_c^j$. The character embeddings $\vec{e}_c^1, \vec{e}_c^2, ..., \vec{e}_c^m$ are combined using an attention model(Bahdanau et al., 2015): $\vec{w}_c^i = \sum_{j=1}^m a_c^j \odot \vec{e}_c^j$, where a_c^j is the weight for $\vec{e}_c^j$, $\odot$ is the Hadamard product function, and $\sum_{j=1}^m a_c^j = 1$.

Each a_c^j is computed according to both the word embedding vector and 5-character embedding window with the current character $\vec{e}_c^j$ in the middle:

$$
\begin{aligned}
a_c^j =& \frac{t_c^j}{\sum_1^m t_c^j} \\
t_c^j =& \exp(\mathbf{W}^t \vec{h}_c^j + \mathbf{U}^t \vec{e}_w^i + \vec{b}^t) \\
\vec{h}_c^i =& \tanh\big(\mathbf{W}^c(\vec{e}_c^{j-2} \oplus \vec{e}_c^{j-1} \oplus \vec{e}_c^j \oplus \vec{e}_c^{j+1} \\
& \oplus \vec{e}_c^{j+2}) + \vec{b}^c\big)
\end{aligned}
$$

Here $\oplus$ denotes vector concatenation and $\vec{e}_w^i$ is the embedding of current word w^i. $\mathbf{W}^t$, $\mathbf{U}^t$, $\mathbf{W}^c$ and $\vec{b}^t$, $\vec{b}^c$ are model parameters. Finally, $\vec{w}_c^i$ is concatenated with word embedding to form final word representation $\vec{r}_w^i$: $\vec{r}_w^i = \vec{e}_w^i \oplus \vec{w}_c^i$

2.2 Feature Layer

Recently, bi-directional LSTM has been successfully applied in various NLP tasks (Liu et al., 2015; Zhou and Xu, 2015; Klerke et al., 2016; Plank et al., 2016). The feature layer uses a bi-directional LSTM to extract a feature vector $\vec{h}^i$ for each word w^i, respectively. An input vector $\vec{x}^i = (\vec{r}_w^{i-2} \oplus \vec{r}_w^{i-1} \oplus \vec{r}_w^i \oplus \vec{r}_w^{i+1} \oplus \vec{r}_w^{i+2})$ is used to represent each word w^i.

We use a LSTM variation with peephole connections (Graves and Schmidhuber, 2005) to extract features based on $\vec{x}^{(1:n)}$. The model computes a hidden vector $\vec{h}^i$ for each input $\vec{x}^i$, passing information from $\vec{h}^1, ..., \vec{h}^{i-1}$ to $\vec{h}^n$ via a sequence of cell states $\vec{c}^1, \vec{c}^2, ..., \vec{c}^n$. Information flow is controlled using an input gate $\vec{g}^i$, a forget gate $\vec{f}^i$, and an output gate $\vec{o}^i$:

$$
\begin{aligned}
\vec{g}^i =& \sigma(\mathbf{W}^{(g)}\vec{x}^i + \mathbf{U}^{(g)}\vec{h}^{i-1} + \mathbf{V}^{(g)}\vec{c}^{i-1} + \vec{b}^{(g)}) \\
\vec{f}^i =& \sigma(\mathbf{W}^{(f)}\vec{x}^i + \mathbf{U}^{(f)}\vec{h}^{i-1} + \mathbf{V}^{(f)}\vec{c}^{i-1} + \vec{b}^{(f)}) \\
\vec{c}^i =& \vec{f}^i \odot \vec{c}^{i-1} + \\
& \vec{g}^i \odot \tanh(\mathbf{W}^{(u)}\vec{x}^i + \mathbf{U}^{(u)}\vec{h}^{i-1} + \vec{b}^{(u)}) \\
\vec{o}^i =& \sigma(\mathbf{W}^{(o)}\vec{x}^i + \mathbf{U}^{(o)}\vec{h}^{i-1} + \mathbf{V}^{(o)}\vec{c}^i + \vec{b}^{(o)}) \\
\vec{h}^i =& \vec{o}^i \odot \tanh(\vec{c}^i),
\end{aligned}
$$

where σ denotes the component-wise sigmoid function. $\mathbf{W}^{(g)}$, $\mathbf{W}^{(f)}$, $\mathbf{W}^{(u)}$, $\mathbf{W}^{(o)}$, $\mathbf{U}^{(g)}$, $\mathbf{U}^{(f)}$, $\mathbf{U}^{(u)}$, $\mathbf{U}^{(o)}$, $\mathbf{V}^{(g)}$, $\mathbf{V}^{(f)}$, $\mathbf{V}^{(o)}$, $\vec{b}^{(g)}$, $\vec{b}^{(f)}$, $\vec{b}^{(u)}$, $\vec{b}^{(o)}$ are model parameters.

Bi-directional extension of the above LSTM structure is applied in both the left-to-right direction and the right-to-left direction, resulting in two hidden vector sequences $h_l^{(1:n)}, h_r^{(1:n)}$, respectively. Each $\vec{h}_l^i$ is combined with its corresponding $\vec{h}_r^i$ for final feature vector $\vec{h}_f^i$:

$$
\vec{h}_f^i = \tanh(\mathbf{W}^l \vec{h}_l^i + \mathbf{W}^r \vec{h}_r^i + \vec{b}),
$$

where $\mathbf{W}^l$, $\mathbf{W}^r$ and $\vec{b}$ are model parameters.

2.3 Output Layer

The output layer employs a conditional random field (CRF) to infer the POS t^i of each word w^i based on the feature layer outputs. The conditional probability of a tag sequence given an input sentence is:

$$
p(\vec{y}|\vec{x}) = \frac{\prod_{i=1}^n \Psi(\vec{x}, \vec{y}^i) \prod_{i=1}^n \Phi(\vec{x}, \vec{y}^i, \vec{y}^{i-1})}{Z(\vec{x})},
$$

where $Z(\vec{x})$ is the partition function:

$$
Z(\vec{x}) = \sum_{\vec{y}} \prod_{i=1}^n \Psi(\vec{x}, \vec{y}^i) \prod_{i=1}^n \Phi(\vec{x}, \vec{y}^i, \vec{y}^{i-1})
$$

In particular, the output clique potential $\Psi(\vec{x}, \vec{y}^i)$ shows the correlation between inputs and output labels: $\Psi(\vec{x}, \vec{y}^i) = \exp(\vec{s}^i)$, with the emission vector $\vec{s}^i$ being defined as:

$$
\vec{s}^i = \vec{\theta}_0 \vec{h}_f^i, \tag{1}
$$

where $\vec{\theta}_0$ is the model parameter.

The edge clique potential shows the correlation between consecutive output labels using a single transition weight $\tau(\vec{y}^i, \vec{y}^{i-1})$.

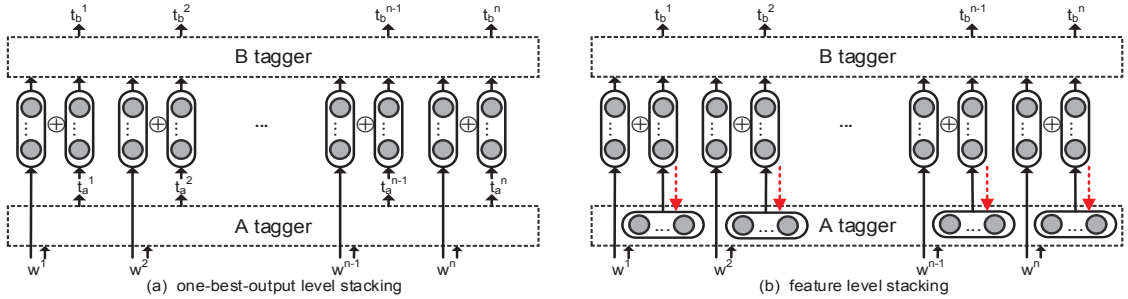

Figure 3: Neural stacking.

3 Stacking

3.1 Discrete Stacking

Stacking integrates corpora A and B by first training a tagger on corpus A, and then using the A tagger to provide additional features to a corpus B model. Figure 1(a) shows the training and testing of discrete stacking models, where the B tagger extracts features from both the raw sentence and A tagger output. This method achieves feature combination at the one-best-output level.

3.2 Neural Stacking

Figure 3(a) and (b) shows the two neural stacking methods of this paper, respectively.

Shallow Integration. Figure 3(a) is a variation of discrete stacking, with the output tags from tagger A being converted to a low-dimensional dense embedding features, and concatenated to the word embedding inputs to tagger B. Formally, for each word w^i, denote the tagger A output as t_a^i, we concatenate the embedding form of t_a^i, denoted as $\vec{e}_a^i$, to the word representation $\vec{r}_w^i$.

$$\vec{r}_w^{i\prime} = \vec{r}_w^i \oplus \vec{e}_a^i = \vec{e}_w^i \oplus \vec{w}_c^i \oplus \vec{e}_a^i \quad (2)$$

Deep Integration. Figure 3(b) offers deeper integration between the A and B models, which is feasible only with neural network features. We call this method feature-level stacking. For feature-level integration, the emission vector $\vec{s}^i$ in Eq.(1) is taken for input to tagger B via a projection:

$$\vec{w}_a^i = \tanh(\mathbf{W}^s \vec{s}^i)$$
$$\vec{r}_w^i = \vec{e}_w^i \oplus \vec{w}_c^i \oplus \vec{w}_a^i,$$

where $\mathbf{W}^s$ is a model parameter.

Fine-tuning. Feature-level stacking further allows tagger A to be fine-tuned during the training of tagger B, with the loss function being back propagated to tagger A via the $\vec{w}_a^i$ layer (shown in the red dotted lines in Figure 3(b)). This is a further benefit of neural stacking compared with discrete stacking.

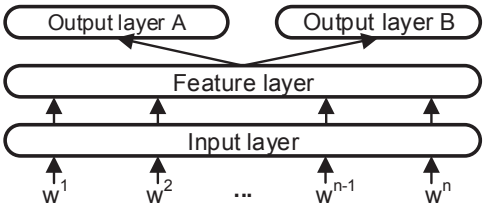

Figure 4: Neural multi-view model.

4 Multi-view Learning

4.1 Discrete Label Coupling

As shown in Figure 1(b), multi-view learning (Li et al., 2015) utilizes corpus A and corpus B simultaneously for training. The coupled tagger directly learns the logistic correspondences between both corpora, therefore can lead a more comprehensive usage of corpus A compared with stacking. In order to better capture such correlation, specifically designed feature templates between two tag sets are essential.

For each training instances, both A and B labels are needed. However, one type of tag is missing. Li et al. (2015) used a mapping function to supplement the missing annotations with the help of the annotated tag. The result is a set of sentence with bundled tags in both annotations, but with ambiguities on one side, due to one-to-many mappings. Li et al. (2015) showed that speed can be significantly improved by manually restricting possible mappings between the labelsets, but a full mapping without restriction yields the highest accuracies.

4.2 Neural Multi-task Learning

Neural multi-task learning is free from manual feature engineering, and avoids manual mapping func-

tions between tag sets by establishing two separate output layers, one for each type of label, with shared low-level parameters. The general structure of a neural multi-view model is shown in Figure 4, which can be regarded as a variation of the parameter sharing model of Caruana (1993) and Collobert et al. (2011). Leveraging heterogeneous annotations for the same task, compared to parameter sharing between different NLP tasks (Collobert et al., 2011), can benefit from tighter integration of information, and hence allows deeper parameter sharing. These are verified empirically in the experiments.

In training and testing, sentences from both corpora go through the same input layer and feature layer. The outputs of each type of tag is then computed separately according to the shared parameters. The conditional probability of a tag sequence given an input sentence and its corpus type is:

$$p(\vec{y}|\vec{x},T) = \frac{\prod_{i=1}^{n} \Psi_T(\vec{x},\vec{y}^{\,i}) \prod_{i=1}^{n} \Phi_T(\vec{x},\vec{y}^{\,i},\vec{y}^{\,i-1})}{Z_T(\vec{x})},$$

where T is the corpus type, $T \in \{A, B\}$. $\Psi_T(\vec{x},\vec{y}^{\,i})$ and $\Phi_T(\vec{x},\vec{y}^{\,i},\vec{y}^{\,i-1})$ are the corresponding output clique potential and edge clique potential, respectively. $Z_T(\vec{x})$ is the partition function:

$$Z_T(\vec{x}) = \sum_{\vec{y}} \prod_{i=1}^{n} \Psi_T(\vec{x},\vec{y}^{\,i}) \prod_{i=1}^{n} \Phi_T(\vec{x},\vec{y}^{\,i},\vec{y}^{\,i-1})$$

This indicates that each time only one output layer is activated according to the corpus type of input sentences.

5 Training

A max-margin objective is used to train the full set of model parameters Θ:

$$L(\Theta) = \frac{1}{D} \sum_{d=1}^{D} l(\vec{x_d},\vec{y_d},\Theta) + \frac{\lambda}{2} \|\Theta\|^2,$$

where $\vec{x}_d, \vec{y}_d|_{d=1}^{D}$ are the training examples, λ is a regularization parameter, and $l(\vec{x}_d,\vec{y}_d,\Theta)$ is the max-margin loss function towards one example $(\vec{x}_d, \vec{y_d})$.

The max-margin loss function is defined as:

$$l(\vec{x}_d,\vec{y_d},\Theta) = \max_{y} \left(s(\vec{y}|\vec{x}_d,\Theta) + \delta(\vec{y},\vec{y}_d)\right) - s(\vec{y}_d|\vec{x}_d,\Theta),$$

Algorithm 1 Neural multi-view training

Input: Two training datasets: $D^{(1)} = (x_n^{(1)}, y_n^{(1)})|_{n=1}^{N}$, $D^{(2)} = (x_m^{(2)}, y_m^{(2)})|_{m=2}^{M}$; Parameters: E, A, R

Output: Θ

```
for i = 1 to E do
    Sample A instances from D^(1) and A * R instances from D^(2) to form a new dataset D_i
    Shuffle D_i.
    for each batch D_i^b in D_i do
        Forward: compute the cost
        Backward: compute the loss of each parameter
        Update the parameters
    end for
end for
```

		sentences	tokens
CTB	train	16091	437991
	dev	803	20454
	test	1910	50319
PD	train	100749	5194829
	dev	18875	958026

Table 1: Data statistics.

where $\vec{y}$ is the model output, $s(\vec{y}|\vec{x}) = logP(\vec{y}|\vec{x})$ is the log probability of $\vec{y}$ and $\delta(\vec{y},\vec{y}_d)$ is the Hamming distance between $\vec{y}$ and $\vec{y}_d$.

We adopt online learning, updating parameters using AdaGrad (Duchi et al., 2011). To train the neural stacking model, we first train a base tagger using corpus A. Then, we train the stacked tagger with corpus B, where the parameters of the A tagger has been pretrained from corpus A and the B tagger is randomly initialized.

For neural multi-view model, we follow Li et al. (2015) and take a the corpus-weighting strategy to sample a number of training instances from both corpora for each training iteration, as shown in Algorithm 1. At each epoch, we randomly sample from the two datasets according to a *corpus weights ratio*, namely the ratio between the number of sentences in each dataset used for training, to form a training set for the epoch.

6 Experiments

6.1 Experimental Settings

We adopt the Penn Chinese Treebank version 5.0 (*CTB5*) (Xue et al., 2005) as our main corpus, with the standard data split following previous work (Zhang and Clark, 2008; Li et al., 2015). People's Daily (*PD*) is used as second corpus with a different scheme. We filter out *PD* sentences longer than 200 words. Details of the datasets are listed in Table 1. The standard token-wise POS tagging accuracy is used as the evaluation metric. The systems are implemented with LibN3L (Zhang et al., 2016).

For all the neural models, we set the hidden layer size to 100, the initial learning rate for Adagrad to 0.01 and the regularization parameter λ to 10^{-8}. *word2vec*[1] is used to pretrain word embeddings. The Chinese Giga-word corpus version 5 (Graff and Chen, 2005), segmented by *zpar*[2] (Zhang and Clark, 2011), is used for the training corpus for word embeddings. The size of word embedding is 50.

6.2 Development Experiments

We use the development dataset for two main purposes. First, under each setting, we tune the model parameters, such as the number of training epochs. Second, we study the influence of several important hyper-parameters using the development dataset. For example, for the NN multi-view learning model, the *corpus weights ratio* (section 5) plays an important role for the performance. We determine the parameters of the model by studying the accuracy along with the increasing epochs.

Effect of batch size and dropout. The batch size affects the speed of training convergence and the final accuracies of the neural models, and the dropout rate has been shown to significantly influence the performance (Chen et al., 2015). We investigate the effects of these two hyper-parameters by adopting a corpus weight ratio of $1:1$ (All the *CTB* training data is used, while the same amount of *PD* is sampled randomly), drawing the accuracies of the neural multi-view learning model against the number of training epochs with various combinations of the dropout rate d and batch size b. The results are shown for the multi-view learning model. For the stacking model, we use b=100 for the *PD* sub model.

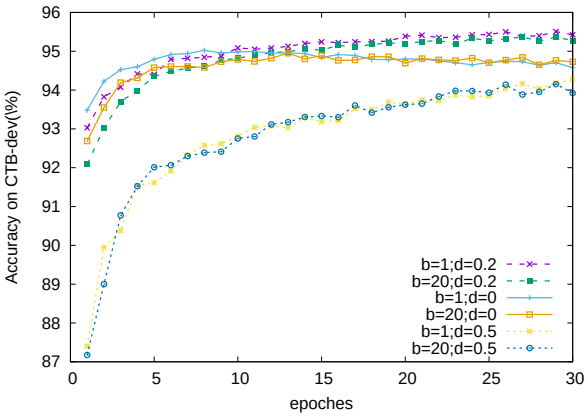

Figure 5: Accuracy on *CTB-dev* with different batch sizes and dropout rates for a multi-view learning model. b represents batch size, d denotes dropout rate.

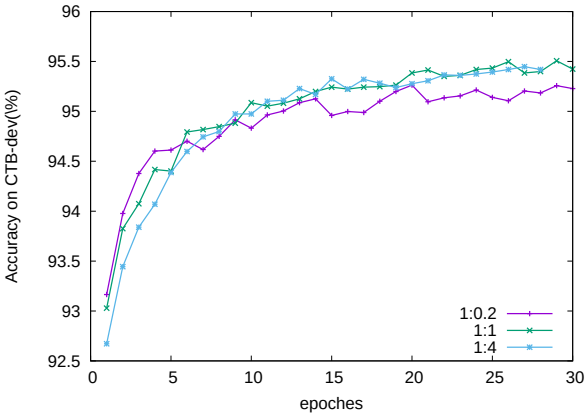

Figure 6: Accuracy on *CTB-dev* with different corpus weights ratio.

The results are shown in Figure 5, where the two dashed lines on the top at epoch 30 represent the dropout rate of 20%, the two solid lines in the middle represent zero dropout rate, and the two dotted lines in the bottom represent a dropout rate 50%. Without using dropout, the performance increases in the beginning, but then decreases as the number of training epochs increases beyond 10. This indicates that the NN models can overfit the training data without dropout. However, when a 50% dropout rate is used, the initial performances are significantly worse, which implies that the 50% dropout rate can be too large and leads to underfitting. As a result, we choose a dropout rate of 20% for the remaining experiments, which strikes the balance between over-

[1]https://code.google.com/p/word2vec
[2]https://github.com/SUTDNLP/ZPar

System	Accuracy
CRF Baseline (Li et al., 2015)	94.10
CRF Stacking (Li et al., 2015)	94.81
CRF Multi-view (Li et al., 2015)	95.00
NN Baseline	94.24
NN Stacking	94.74
NN Feature Stacking	95.01
NN Feature Stacking & Fine-tuning	95.32
NN Multi-view	95.40
Integrated NN Multi-view & Stacking	**95.53**

Table 2: Accuracies on *CTB-test*.

fitting and underfitting.

Figure 5 also shows that the batch size has a relative small influence on the accuracies, which varies according to the dropout rate. We simply choose a batch size of 1 for the remaining experiments according to the performance at the dropout rate 20%.

Effect of corpus weights ratio. Figure 6 shows the effects of different corpus weights ratios. In particular, a corpus weights ratio of 1:0.2 yields relative low accuracies. This is likely because it makes use of the least amount of *PD* data. The ratios of 1:1 and 1:4 give comparable performances. We choose the former for our final tests because it is a much faster choice.

6.3 Final Results

Table 2 shows the final results on the *CTB* test data. We lists the results of stacking method of Jiang et al. (2009) re-implemented by Li et al. (2015), and CRF multi-view method reported by Li et al. (2015). We adopt pair-wise significance test (Collins et al., 2005) when comparing the results between two different models.

Stacking. For baseline tagging using only *CTB*, NN model achieves a result of 94.24, slightly higher than CRF baseline (94.10). NN stacking model integrating *PD* data achieves comparable performance (94.74) compared with CRF stacking model (94.81). Compared with NN baseline, NN stacking model boosts the performance from 94.24 to 94.74, which is significant at the confidence level $p < 10^{-5}$. This demonstrates that neural network model can utilize one-best prediction of the *PD* model for the *CTB* task as effectively as the discrete stacking method of Jiang et al. (2009).

One advantage of NN stacking as compared with discrete stacking method is that it can directly leverage features of *PD* model for *CTB* tagging. Comparison between feature-level stacking and one-best-output level stacking of the NN stacking model shows that the former gives significantly higher results, namely 95.01 vs 94.74 at the confidence level $p < 10^{-3}$.

One further advantage of NN stacking is that it allows the *PD* model to be fine-tuned as an integral sub-model during *CTB* training. This is not possible for the discrete stacking model, because the output of the *PD* model are used as atomic feature in the stacking *CTB* model rather than a gradient admissive neural layer. By fine-tuning the *PD* sub-model, the performance is further improved from 95.01 to 95.32 at the confidence level $p < 10^{-3}$. The final NN stacking model improves over the NN baseline model from 94.24 to 95.32. The improvement is significantly higher compared to that by using discrete stacking which improves over the discrete baseline from 94.01 to 94.74. The final accuracy of the NN stacking model is higher than that of the CRF stacking model, namely 94.81 vs 95.32 at the confidence level $p < 10^{-3}$. This shows that neural stacking is a preferred choice for stacking.

Multi-view training. With respect of the multi-view training method, the NN model improves over the NN baseline from 94.24 to 95.40, by a margin of +1.16, which is higher than that of 0.90 brought by discrete method of Li et al. (2015) over its baseline, from 94.10 to 95.00. NN multi-view training method gives relatively higher improvements compared with NN stacking method. This is consistent with the observation of Li et al. (2015), who showed that discrete label coupling training gives slightly better improvement compared with discrete stacking. The final accuracies of NN multi-view training is also higher than that of its CRF counterpart, namely 95.00 vs 95.40 at the confidence level $p < 10^{-3}$. The difference between the final NN multi-view training result of 95.40 and the final NN stacking results is not significant.[3]

Integration. The flexibility of the NN models further allows both stacking (on the input) and multi-viewing (on the output) to be integrated. When

[3]Note, however, NN stacking method with one-best *PD* output gives significantly lower accuracies (94.74). It is the fine-tuning strategy that allows stacking to give comparable results compared to multi-view training for the NN models.

System	Time Cost(s)
CRF Baseline	176.925
CRF Multi-view (Li et al., 2015)	3992.27
NN Baseline	416.338
NN Multi-view	418.484

Table 3: Time for testing *CTB* training data.

NN multi-view training is combined with a fine-tuned NN feature stacking model, the performance further increases from 95.40 to 95.53. This is the best results we are aware of on this dataset. The improvement is significant at the confidence level $p < 10^{-2}$ compared with fine-tuned NN stacking model (95.32). This indicates that NN multi-view training and stacking model each provide unique advantages for heterogeneous annotations.

6.4 Speed Test

We compare the efficiencies of neural and discrete multi-view training by running our models and the model of Li et al. (2015)[4] with default configurations on the *CTB5* training data. The CRF baseline is adapted from Li et al. (2015). All the systems are implemented in C++ running on an Intel E5-1620 CPU. The results are shown in Table 3.

The NN baseline model is slower than the CRF baseline model. This is due to the higher computation cost of a deep neural network on a CPU. Compared with the CRF baseline, the CRF multi-view model is significantly slower because of its large feature set and the multi-label search space. However, the NN multi-view model achieves almost the same time cost with the NN baseline, and is much more efficient than the CRF counterpart. This shows the efficiency advantage of the NN multi-view model by parameter sharing and output splitting.

7 Related Work

Early research on heterogeneous annotations focuses on annotation conversion. For example, Gao et al. (2004) proposed a transformation-based method to convert the annotation style of a word segmentation corpus to that of another. Manually designed transformation templates are used, which makes it difficult to generalize the method to other tasks and treebanks.

Jiang et al. (2009) described a stacking-based model for heterogeneous annotations, using a pipeline to integrate the knowledge from one corpus to another. Sun and Wan (2012) proposed a structure-based stacking model, which makes use of structured features such as sub-words for model combination. These feature integration is stronger compared to those of Jiang et al. (2009). Johansson (2013) introduced path-based feature templates in using one parser to guide another. In contrast to the above discrete methods, our neural stacking method offers further feature integration by directly connecting the feature layer of the source tagger with the input layer of the target tagger. It also allows the fine-tuning of the source tagger. As one of the reviewers mentioned, two extensions of CRFs, dynamic CRFs (Sutton et al., 2004) and hidden-state CRFs (Quattoni et al., 2004), can also perform similar deep integration and fine-tuning.

For multi-view training, Johansson (2013) used a shared feature representation along with separate individual feature representation for each treebank. Qiu et al. (2013) proposed a multi-task learning model to jointly predict two labelsets given an input sentences. The joint model uses the union of baseline features for each labelset, without considering additional features to capture the interaction between the two labelsets. Li et al. (2015) improves upon this method by using a tighter integration between the two labelsets, treating the Cartesian product of the base labels as a single combined labelset, and exploiting joint features from two labelsets. Though capturing label interaction, their method suffers speed penalty from the sharply increased search space. In contrast to their methods, our neural approach enables parameter sharing in the hidden layers, thereby modeling label interaction without directly combining the two output labelsets. This leads to a lean model with almost the same time efficiency as a single-label baseline.

Recently, Zhang and Weiss (2016) proposed a stack-propagation model for learning a stacked pipeline of POS tagging and dependency parsing. Their method is similar to our neural stacking in fine-tuning the stacked module which yields features for the target model. While their multi-task learning is on heterogenous tasks, our multi-task learning is

[4]http://hlt.suda.edu.cn/zhli/resources/zhenghua-acl2015-resources.zip

defined on heterogenous treebanks.

8 Conclusion

We investigated two methods for utilizing heterogeneous annotations for neural network models, showing that they have respective advantages compared to their discrete counterparts. In particular, neural stacking allows tighter feature integration compared to discrete stacking, and neural multi-view training is free from the feature and efficiency constraints of discrete one. On a standard *CTB* test, the neural method gives the best integration effect, with a multi-view training model enjoying the same speed as its single treebank baseline.

Acknowledgments

The corresponding author is Yue Zhang. We thank Zhenghua Li and Meishan Zhang for providing data and the anonymous reviewers for their constructive comments, which helped to improve the paper. This work is supported by Singapore Ministry of Education Tier 2 Grant T2MOE201301 and Natural Science Foundation of China (61379086).

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR*.

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. The tiger treebank. In *Proceedings of the workshop on treebanks and linguistic theories*, volume 168.

Leo Breiman. 1996. Stacked regressions. *Machine learning*, 24(1):49–64.

Richard A Caruana. 1993. Multitask learning: A knowledge-based source of inductive bias1. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 41–48. Citeseer.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750.

Xinchi Chen, Xipeng Qiu, Chenxi Zhu, Pengfei Liu, and Xuanjing Huang. 2015. Long short-term memory neural networks for chinese word segmentation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Michael Collins, Philipp Koehn, and Ivona Kučerová. 2005. Clause restructuring for statistical machine translation. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 531–540. 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.

Hal Daumé III. 2007. Frustratingly easy domain adaptation. pages 256–263.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependeny parsing with stack long short-term memory. In *Proceedings of the 53^{rd} Annual Meeting of the Association of Computational Linguistics and the 7^{th} International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing (ACL-IJCNLP 2015)*. ACL.

Thomas Emerson. 2005. The second international chinese word segmentation bakeoff. In *Proceedings of the fourth SIGHAN workshop on Chinese language Processing*, volume 133.

Jianfeng Gao, Andi Wu, Mu Li, Chang-Ning Huang, Hongqiao Li, Xinsong Xia, and Haowei Qin. 2004. Adaptive chinese word segmentation. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 462. Association for Computational Linguistics.

David Graff and Ke Chen. 2005. Chinese gigaword. *LDC Catalog No.: LDC2003T09, ISBN*, 1:58563–58230.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Wenbin Jiang, Liang Huang, and Qun Liu. 2009. Automatic adaptation of annotation standards: Chinese word segmentation and pos tagging: a case study. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International*

Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1, pages 522–530. Association for Computational Linguistics.

Wenbin Jiang, Yajuan , Liang Huang, and Qun Liu. 2015. Automatic adaptation of annotations. *Computational Linguistics*, 41(1):1–29.

Richard Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for English. In *Proceedings of NODALIDA 2007*, pages 105–112, Tartu, Estonia, May 25-26.

Richard Johansson. 2013. Training parsers on incompatible treebanks. In *HLT-NAACL*, pages 127–137.

Sigrid Klerke, Yoav Goldberg, and Anders Søgaard. 2016. Improving sentence compression by learning to predict gaze. *arXiv preprint arXiv:1604.03357*.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.

Zhenghua Li, Jiayuan Chao, Min Zhang, and Wenliang Chen. 2015. Coupled sequence labeling on heterogeneous annotations: pos tagging as a case study. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, volume 1, pages 1783–1792.

Pengfei Liu, Shafiq Joty, and Helen Meng. 2015. Fine-grained opinion mining with recurrent neural networks and word embeddings. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2015)*.

Ji Ma, Yue Zhang, and Jingbo Zhu. 2014. Tagging the web: Building a robust web tagger with neural network. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 144–154, Baltimore, Maryland, June. Association for Computational Linguistics.

Joakim Nivre and Ryan T McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *ACL*, pages 950–958.

Wenzhe Pei, Tao Ge, and Baobao Chang. 2014. Max-margin tensor neural network for chinese word segmentation. In *ACL*, pages 293–303.

Barbara Plank, Anders Søgaard, and Yoav Goldberg. 2016. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. *arXiv preprint arXiv:1604.05529*.

Xipeng Qiu, Jiayi Zhao, and Xuanjing Huang. 2013. Joint chinese word segmentation and pos tagging on heterogeneous annotated corpora with multiple task learning. In *EMNLP*, pages 658–668.

Ariadna Quattoni, Michael Collins, and Trevor Darrell. 2004. Conditional random fields for object recognition. *Advances in Neural Information Processing Systems*, pages 1097–1104.

Richard Sproat and Thomas Emerson. 2003. The first international chinese word segmentation bakeoff. In *Proceedings of the second SIGHAN workshop on Chinese language processing-Volume 17*, pages 133–143. Association for Computational Linguistics.

Weiwei Sun and Xiaojun Wan. 2012. Reducing approximation and estimation errors for chinese lexical processing with heterogeneous annotations. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 232–241. Association for Computational Linguistics.

Charles Sutton, Andrew Mccallum, and Khashayar Rohanimanesh. 2004. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. In *ICML*, pages 693–723.

Heike Telljohann, Erhard W Hinrichs, Sandra Kübler, Heike Zinsmeister, and Kathrin Beck. 2006. Stylebook for the tübingen treebank of written german (tüba-d/z). In *Seminar fur Sprachwissenschaft, Universitat Tubingen, Tubingen, Germany*.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 323–333, Beijing, China, July. Association for Computational Linguistics.

David H Wolpert. 1992. Stacked generalization. *Neural networks*, 5(2):241–259.

Dekai Wu, Grace Ngai, and Marine Carpuat. 2003. A stacked, voted, stacked model for named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 200–203. Association for Computational Linguistics.

Naiwen Xue, Fei Xia, Fu-Dong Chiou, and Marta Palmer. 2005. The penn chinese treebank: Phrase structure annotation of a large corpus. *Natural language engineering*, 11(02):207–238.

Nianwen Xue. 2003. Chinese word segmentation as character tagging. *Computational Linguistics and Chinese Language Processing*, 8(1):29–48.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, volume 3, pages 195–206.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: investigating and combining graph-based

and transition-based dependency parsing using beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 562–571. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2011. Syntactic processing using the generalized perceptron and beam search. *Computational linguistics*, 37(1):105–151.

Yuan Zhang and David Weiss. 2016. Stack-propagation: Improved representation learning for syntax. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1557–1566. Association for Computational Linguistics.

Meishan Zhang, Jie Yang, Zhiyang Teng, and Yue Zhang. 2016. Libn3l:a lightweight package for neural nlp. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Sara Goggi, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Helene Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France, may. European Language Resources Association (ELRA).

Xiaoqing Zheng, Hanyang Chen, and Tianyu Xu. 2013. Deep learning for chinese word segmentation and pos tagging. In *EMNLP*, pages 647–657.

Jie Zhou and Wei Xu. 2015. End-to-end learning of semantic role labeling using recurrent neural networks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Hao Zhou, Yue Zhang, Shujian Huang, and Jiajun Chen. 2015. A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1213–1222, Beijing, China, July. Association for Computational Linguistics.