Strength in Numbers: Averaging and Clustering Effects in Mixture of Experts for Graph-Based Dependency Parsing

Xudong Zhang, Joseph Le Roux, Thierry Charnois
Laboratoire d'Informatique de Paris Nord,
Université Sorbonne Paris Nord – CNRS UMR 7030,
F-93430, Villetaneuse, France
{xudong.zhang,leroux,thierry.charnois}@lipn.fr

Abstract

We review two features of *mixture of experts* (MoE) models which we call *averaging* and *clustering* effects in the context of graph-based dependency parsers learned in a supervised probabilistic framework. *Averaging* corresponds to the ensemble combination of parsers and is responsible for variance reduction which helps stabilizing and improving parsing accuracy. *Clustering* describes the capacity of MoE models to *give more credit* to experts believed to be more accurate given an input. Although promising, this is difficult to achieve, especially without additional data.

We design an experimental set-up to study the impact of these effects.Whereas averaging is always beneficial, clustering requires good initialization and stabilization techniques, but its advantages over mere averaging seem to eventually vanish when enough experts are present.

As a by product, we show how this leads to state-of-the-art results on the PTB and the CoNLL09 Chinese treebank, with low variance across experiments.

1 Introduction

Combinations of elementary parsers are known to improve accuracy. Sometimes called *joint systems*, they often use different representations, *i.e.* lexicalized constituents and dependencies (Rush et al., 2010; Green and Žabokrtský, 2012; Le Roux et al., 2019; Zhou et al., 2020). These approaches have been devised to join the strengths and overcome the weaknesses of elementary systems.

In this work, however, we follow another line of research consisting of mixtures and products of similar experts (Jacobs et al., 1991; Brown and Hinton, 2001), instantiated for parsing in (Petrov et al., 2006; Petrov, 2010) and especially appealing when individual experts have high variance, typically when training involves neural networks. Indeed Petrov (2010) used products of experts trained via Expectation-Maximization (a non-convex function minimization) converging to local minima.

In this work we propose to study the combination of parsers, from a probabilistic point of view, as a mixture model, *i.e.* a learnable convex interpolation of probabilities. This has previously been studied in (Petrov et al., 2006) for PCFGs with the goal of overcoming the locality assumptions, and we want to see if neural graph-based dependency parsers, with non-markovian feature extractors, can also benefit from this framework. It has several advantages: it is conceptually simple and easy to implement, it is not restricted to projective dependency parsing (although we only experiment this case), and while the time and space complexity increases with the number of systems, this is hardly a problem in practice thanks to GPU parallelization.

Simple averaging models, or ensembles, can also be framed as mixture models where mixture coefficients are equal. We are able to quantify the variance reduction, both theoretically and empirically and show that this simple model of graph-based parser combinations perform better on average, and achieve a higher accuracy than single systems.

While the full mixture model is appealing, since it could in principle both decrease variance and find the optimal interpolation weights to better combine parser predictions, the non-convexity of the learning objective is a major issue that, when added to the non-convexity of potential functions, can prevent parameterization to converge to a good solution. By trying to specialize parsers to specific input, the variance is not decreased. More importantly, experiments indicate that useful data, that is data with an effect on parameterization, becomes too scarce to train the clustering device.

Another drawback of finite mixture models is that inference, *i.e.* finding the optimal tree, becomes intractable. We tackle this issue by using an alternative objective similar to Minimal Bayes-

Proceedings of the 17th International Conference on Parsing Technologies (IWPT 2021), pages 106–118
Bangkok, Thailand (online), August 6, 2021. ©2021 Association for Computational Linguistics

Risk (Goel and Byrne, 2000) and PCFG-LA combination (Petrov, 2010) for which decoding is exact.

Our contribution can be summarized as follows:

- We frame dependency parser combinations as finite mixture models (§2) and discuss two properties: averaging and clustering. We derive an efficient decoder (LMBR) merging predictions at the arc level (§3).
- When isolating the averaging effect, we show that resulting systems exhibit an empirical variance reduction which corroborates theoretical predictions, and are more accurate (§4).
- We study the causes of instability in mixture learning, outline why simple regularization is unhelpful and give an EM-inspired learning method preventing detrimental overspecialization (§5). Still, improvement over mere averaging is difficult to achieve.
- These methods obtain state-of-the-art results on two standard datasets, the PTB and the CoNLL09 Chinese dataset (§6), with low variance making it robust to initial conditions.

2 Mixture of Experts

2.1 Notations

We write a sentence as $x = x_0, x_1, \dots, x_n$, with x_0 a dummy root symbol, and otherwise x_i the i^{th} word, and n the number of words. For $h, d \in [n]$ with $[n] = \{0, \dots, n\}$, (h, d) is the directed arc from head x_h to dependent x_d. We note the set of all parse trees (arborescences) for x as $\mathcal{Y}(x)$ and the elements in this set as $y \in \mathcal{Y}(x)$, with $(h, d) \in y$ if (h, d) is an arc in y. $\mathcal{L}$ stands for the set of arc labels. The vector of arc labels in tree y is noted as $l(y) \in \mathcal{L}^n$. We note $l(y)_{hd}$ the label for arc (h, d) in y, or l_{hd} when y is clear from the context.

2.2 Parsers as Experts

Experts can be any probabilistic graph-based dependency parser, provided that we can efficiently compute the energy of a parse tree, the global energy of a sentence (the sum of all parse tree energies, called the partition function) and the marginal probability of an arc in a sentence. In the remaining we focus on projective first- and second-order parsers, where these quantities are computed via tabular methods or backpropagation[1].

[1]Matrix-tree theorem could be used to adapt this work to non-projective first-order models (Smith and Smith, 2007)

Tree structure For a graph-based dependency parser, the tree probability is defined as:

$$p(y|x) = \frac{\exp(s(x, y))}{Z(x) \equiv \sum_{y' \in \mathcal{Y}(x)} \exp(s(x, y'))}$$

with $s(x, y)$ the tree energy giving the correctness of y for x, and $Z(x)$ the partition function.

In first-order models (Eisner, 1996), tree scores are sums of arc scores:

$$s(x, y) = \sum_{(h,d) \in y} s(h, d)$$

Eisner (1997) generalizes scores to the second-order by considering pairs of adjacent siblings:

$$s(x, y) = \sum_{(h,d) \in y} s(h, d) + \sum_{\substack{(h,d_1) \\ (h,d_2) \in y}} s(h, d_1, d_2)$$

with $h < d_1 < d_2$ or $d_2 < d_1 < h$. For projective first- or second-order models, $Z(x)$ and $p(y|x)$ are efficiently calculated (Zhang et al., 2020b). Moreover marginal arc probability $p\big((h, d)|x\big)$ can be efficiently calculated from the partition function by applying backpropagation from $\log Z(x)$ to $s(h, d)$, see (Eisner, 2016; Zmigrod et al., 2020; Zhang et al., 2020a):

$$p((h, d)|x) = \sum_{\substack{y \in \mathcal{Y}(x) \\ (h,d) \in y}} p(y|x) = \frac{\partial \log Z(x)}{\partial s(h, d)}$$

Tree Labelling The labelling model is also a Boltzmann distribution:

$$p(l|(h, d), x) = \frac{\exp(s(l, h, d))}{\sum_{l' \in \mathcal{L}} \exp(s(l', h, d))}$$

where $s(l, h, d)$ is the score for label l on (h, d).

Following (Dozat and Manning, 2017; Zhang et al., 2020a), label predictions are independent:

$$p(l(y)|y, x) = \prod_{(h,d) \in y} p(l_{hd}|(h, d), x) \quad (1)$$

Parse Probability Given the structure y and its labelling $l(y)$, the parse probability is:

$$p\big(l(y), y|x\big) = p(y|x) \times p\big(l(y)|y, x\big) \quad (2)$$

Learning Potential functions s can be implemented by feed-forward neural networks or biaffine functions (Dozat and Manning, 2017), and parameterized by maximizing a log-likelihood.

2.3 Mixture and Averaging

For arborescence probabilities a finite mixture model (MoE) is a weighted sum of the probabilities given by all experts:

$$p(y|x) = \sum_{k=1}^{K} \omega_k(x) p_k(y|x) \tag{3}$$

where mixture weights verify $\forall x, \omega_k(x) \geq 0$ and $\sum_{k=1}^{K} \omega_k(x) = 1$ and can be adjusted by a gating network (Jacobs et al., 1991). We can interpret ω as a device whose role is to *cluster* input in K categories and assign each category to an expert.

By forcing $\omega_k(x) = \frac{1}{K}, \forall x$, we have a simpler averaging model, sometimes called *ensemble*:

$$p(y|x) = \frac{1}{K} \sum_{k=1}^{K} p_k(y|x)$$

Note that MoEs combine elementary probabilities, not tree scores: each expert energy is first normalized before the combination.

A similar mixture is applied to labelling, *i.e.*:

$$p(l(y)|y, x) = \sum_{k=1}^{K} \lambda_k(x) p_k(l(y)|y, x)$$

3 Decoding with a Mixture Model

Learning MoEs will be covered in Section 5 and we first turn to the problem of finding an appropriate tree, for instance the most probable parse tree:

$$y^* = \operatorname*{argmax}_{y \in \mathcal{Y}(x)} p(y|x) = \operatorname*{argmax}_{y \in \mathcal{Y}(x)} \sum_{k=1}^{K} \omega_k(x) p_k(y|x)$$

This maximization is difficult, even in the absence of labels, since this isn't a log-linear function of the arc scores anymore: y^* cannot be searched in the log-space among unnormalized arc scores.

3.1 MBR Decoding

In this case, a more attractive alternative is Minimum Bayesian Risk (MBR) decoding (Smith and Smith, 2007), because it decomposes error in a way similar to the metrics used in dependency parsing (UAS/LAS) and is tractable. MBR requires to compute marginal arc probabilities which are the weighted sums of elementary marginals:

$$p\big((h,d)|x\big) = \sum_{k=1}^{K} \omega_k(x) p_k\big((h,d)|x\big)$$

The intuition behind MBR is that instead of maximizing the probability of the parse tree, we try to minimize the risk of choosing wrong arcs, *i.e.* to maximize the arc marginals in the parse tree:

$$y^* = \operatorname*{argmax}_{y \in \mathcal{Y}(x)} \prod_{(h,d) \in y} p\big((h,d)|x\big) = \mathrm{MBR}(x)$$

Once computed marginal log-probabilities, Eisner algorithm (Eisner, 1996), (Eisner, 1997) or Chu-Liu-Edmonds (McDonald et al., 2005) can be applied to solve MBR.

3.2 MBR Decoding with Labels

In many dependency parsing models, decoding of arcs and labels is pipelined, see for instance (Dozat and Manning, 2017; Zhang et al., 2020a; Fossum and Knight, 2009): first arcs are decoded and then, with the decoded arcs, maximization is performed over labels:

$$y^* = \operatorname*{argmax}_{y \in \mathcal{Y}(x)} p(y|x) \text{ then } l^* = \operatorname*{argmax}_{l = l(y^*)} p(l|y^*, x)$$

However, solutions found this way are not the maximizers for $p(l, y|x)$, as defined in Eq. 2. The problem is that the effect of labelling is not considered in arc decoding: a high probability arc can get picked up even with a low label score.

First we remark that each label in l^* is the most probable label l for a pair (h, d), denoted by $L_{hd} = \operatorname{argmax}_{l \in \mathcal{L}} p(l|(h,d), x)$. Decoding becomes:

$$y^* = \operatorname*{argmax}_{y} p(y|x) \prod_{(h,d) \in y} p(L_{hd}|(h,d), x)$$

This way l^* is deterministic wrt to y^* and (y^*, l^*) are maximizers for Eq. 2. We note labelling $L(y)$ where $l(y)_{hd} = L_{hd}, \forall (h,d) \in y$. This can be combined with MBR without changing decoding algorithms, and we call this variant LMBR:

$$y^* = \operatorname*{argmax}_{(y, l = L(y))} \prod_{(h,d) \in y} p\big((h,d)|x\big) p\big(L_{hd}|(h,d), x\big)$$

i.e. we can apply MBR with arc probabilities reparameterized with label probabilities. Experiments show that LMBR exhibits a small but consistent accuracy increase over MBR.

4 Averaging and Variance Reduction

In this section we assume all experts to be equally weighted. We define the variance of the system on $\mathcal{T}$ as the average variance of marginal arc probability:

$$\sigma^2 = \frac{\sum_{(x,y)\in\mathcal{T}} \sum_{(h,d)\in y} \sigma^2\Big(p\big((h,d)|x\big)\Big)}{\sum_{(x,y)\in\mathcal{T}} |y|}$$

with $\sigma^2(p((h,d)|x))$ the variance of the marginal probability.

We show how the variance of the MoE is smaller than the variance of experts. We focus on structure prediction $p(y|x)$, but definitions are applicable to the labelling model as well. This is an already known result for mixture models in general, but the proof is here instantiated for a mixture of graph-based parsers. Moreover, we will recover this result experimentally in Section 6.

Assuming we have a mixture of K elementary systems, we will estimate the marginal probability variance with:

$$\sigma^2\Big(p\big((h,d)|x\big)\Big) = \frac{1}{K}\sum_{k=1}^{K}(\pi(k) - \bar{\pi})^2$$

with $\pi(k)$ the probability $p_k\big((h,d)|x\big)$ given by the k^{th} elementary system and average $\bar{\pi} = \frac{1}{K}\sum_{k=1}^{K}\pi(k)$

Increasing the number of experts in the MoE will decrease variance of the system. To see this, we assume that the marginal probability for a well trained expert, over a fixed sentence and a fixed arc, is a measurable function $f_{(h,d),x} : \mathbb{R} \to \mathbb{R}$ of a random seed $S_k \in \mathbb{R}$, which represents the fact that p_k is the result of a learning process with many sources of randomization[2] (initialization, stochastic batches, dropout...):

$$p_k((h,d)|x) = f_{(h,d),x}(S_k)$$

with $S_k \in \mathbb{R}$ a random seed assigned to k^{th} expert at the beginning of training, assumed to be independent for different experts.

Since in practice a pseudo-random generator is used, the value of marginal probability for particular sentence and arc is deterministic when the random seed is fixed. Thus, it is sufficient to use a deterministic function to represent $p_k((h,d),x)$, with random seed S_k as input. Moreover, we just need the function to be measurable.

We can now view $f_{(h,d),x}(S)$ as a random variable and we note its variance as $\sigma^2_{(h,d),x}$. It is in fact the variance of the marginal arc probability given by this expert, for (h,d) given x. For an averaging MoE, the marginal probability becomes:

$$p((h,d)|x) = \frac{1}{K}\sum_{k=1}^{K} f_{(h,d),x}(S_k)$$

with K number of experts in the mixture model.

If random variables $\{S_k\}_{k\in K}$ are independent, $\{f_{(h,d),x}((S_k)\}_{k\in K}$ also are independent (Baldi, 2017). Thus, the variance of the mixture model for particular sentence and arc should be $\frac{1}{K}$ times the variance of experts:

$$\Sigma^2_{(h,d),x} = \frac{\sigma^2_{(h,d),x}}{K} \tag{4}$$

with Σ the variance of the mixture model. In other words, the log-variance of a mixture model decreases linearly with $\log K$, with slope -1, *i.e.*:

$$\log \Sigma^2_{(h,d),x} = \log \sigma^2_{(h,d),x} - \log K$$

Experiments in Section 6 Figure 1 show that the estimated log-variance of the averaging system decreases when the number of experts increases and that this relation is close to linear with a slope approaching -1, comforting our independence assumption.

5 Training with Clustering

When mixture weights are adjustable, MoE models are able to give more credit to experts believed to perform better on specific input. This can be exploited during parameterization. The role of ω is thus to learn how to cluster input into K categories, each category being assigned to an expert.[3]

For input sentence x and corresponding tree y, assuming parameterization is performed by maximizing the log-likelihood of the training set via SGD, the objective of mixture model learning with gating network ω can be written as:

$$L(\phi,\theta) = \log \sum_{k=1}^{K} \omega_k(x;\phi) p_k(y|x;\theta_k) \tag{5}$$

[2] f should also be indexed by the training set, but we omit this for the sake of readability.

[3] We note that averaging MoE models do not require a specific training: experts can be trained separately and the ensemble is gathered at decoding time only.

where ϕ are the parameters of the gating network, and θ_k are the parameters of the k^{th} expert.

Partial derivatives to the gating network are:

$$\frac{\partial L(\phi,\theta)}{\partial \phi} = \sum_{k=1}^{K} \frac{\omega_k(\phi)p_k(\theta_k)}{\sum_{k'=1}^{K}\omega_{k'}(\phi)p_{k'}(\theta_{k'})}\frac{\partial \log \omega_k(\phi)}{\partial \phi} \tag{6}$$

while for expert parameters we have:

$$\frac{\partial L(\phi,\theta)}{\partial \theta_k} = \frac{\omega_k(\phi)p_k(\theta_k)}{\sum_{k'=1}^{K}\omega_{k'}(\phi)p_{k'}(\theta_{k'})}\frac{\partial \log p_k(\theta_k)}{\partial \theta_k}. \tag{7}$$

We found that optimizing directly with equations (6) and (7) causes degeneration, *i.e.* one ω_k approaches 1 while the other $\omega_{k'}$ decrease to almost 0. Indeed, gradient ascent with (6) will increase ω_k for an expert k that gives high weight to training samples while gradient ascent with (7) will generate increased gradient, and in turn increased probabilities, for experts with high value of ω_k. The two processes re-enforce each other and result quickly in an extreme partition between experts.

One may think that the degeneration problem can be alleviated with a smoothing prior or regularization. In practice, we tried entropy as regularization to force towards a uniform distribution on ω_k. We found that a heavy entropy penalization is required to avoid the degeneration problem, which makes ω_k too uniform to be an accurate clustering device.

Avoid Extreme Partition Thus, to alleviate the degeneration problem without forcing a strong smoothing constraint, we propose to modify Eq. (6) into:

$$\frac{\partial L'(\phi,\theta)}{\partial \phi} = \sum_{k=1}^{K} \frac{p_k(\theta_k)}{\sum_{k'=1}^{K}p_{k'}(\theta_{k'})}\frac{\partial \log \omega_k(\phi)}{\partial \phi} \tag{8}$$

i.e. we force the weight update to be proportional to the relative probability. The advantage of Eq. (8) is that gradient are weighted by a more objective quantity $\frac{p_k(\theta_k)}{\sum_{k'=1}^{K}p_{k'}(\theta_{k'})}$. For an example where $p_k(x)$ is close to uniform, we can benefit from the averaging effect, while for an example which shows strong preference for a particular expert, we can also learn the partition coefficients proportional to their correctness.

Stabilize Training Neuron dropout (Srivastava et al., 2014) is a common technique to avoid overfitting which unfortunately proved difficult in this setting. The problem is that $s_k(x,y)$ gives very different results with or without dropout which reflects on $p_k(y|x)$ causing drastic changes from one evaluation to the other. To mitigate this problem, we use probabilities without dropout (noted as $\tilde{p}_k(\theta)$) to calculate the weighted coefficients of gradient.

The final optimization process can be separated into two alternate parts, *(i)* optimization of the gating parameters:

$$\frac{\partial L'(\phi,\theta)}{\partial \phi} = \sum_{k=1}^{K} \frac{\tilde{p}_k(\theta_k)}{\sum_{k'=1}^{K}\tilde{p}_{k'}(\theta_{k'})}\frac{\partial \log \omega_k(\phi)}{\partial \phi}$$

and *(ii)* optimization of experts:

$$\frac{\partial L(\phi,\theta)}{\partial \theta_k} = \sum_{k=1}^{K} \frac{\omega(\phi)\tilde{p}_k(\theta_k)}{\sum_{k'=1}^{K}\omega_{k'}\tilde{p}_{k'}(\theta_{k'})}\frac{\partial \log p_k(\theta_k)}{\partial \theta_k}$$

In practice, this permitted reaching a lower loss value after training.

6 Experiments

Data We run experiments over two datasets for projective dependency parsing: The English Penn Treebank (PTB) data with Stanford Dependencies (Marcus et al., 1993) and CoNLL09 Chinese data (Hajič et al., 2009). We use standard train/dev/test splits and evaluate with UAS/LAS metrics. Customarily, punctuation is ignored on PTB evaluation.

Experts We run tests with first-order (FOP) and second-order parsers (SOP) as mixture model experts, with re-implemented versions of the CRF and CRF2o parsers of Zhang et al. (2020a).[4] For decoding, we use the LMBR decoding presented in Section 3.2, which guarantees a small but consistent improvement over pipeline MBR decoding.

For each input word, these systems use 3 embeddings: the first is a fixed pretrained vector[5], the second is trainable and looked-up in a table, and the third is computed by a BiLSTM at the character level (CharLSTM). The first two embeddings are summed and concatenated with the char sequence embedding. For FOP and SOP, contextual lexical features are the results of 3-layer BiLSTMs applied to word embedding sequences. The scoring of arcs is then similar to (Dozat and Manning, 2017): lexical features are transformed for head or modifier roles by two feed-forward networks and combined to score arcs via a biaffine transformation.

[4] https://github.com/kidlestar/MOE.git.

[5] For English we used Glove embeddings (Pennington et al., 2014), while for Chinese we extracted pretrained embeddings from the publicly available model of Zhang et al. (2020b).

On PTB, in order to compare with recent parsing results, we set up BFOP and BSOP (B for Bert), variants of the FOP and SOP settings: we follow Fonseca and Martins (2020) and concatenate an additional BERT embedding (Devlin et al., 2019) (the average of the 4 last layers of the *bert-base-uncased* model) to the embedding vector fed to the BiLSTM layers.

Gating (mixture weights ω) is implemented by a K-class softmax over a feed-forward network whose input are the concatenation of initial and final contextual lexical feature vectors returned by the 3-layer BiLSTM. Hyper-parameters are set similarly to Zhang et al. (2020a), with the exception of the learning rate decreased to 10^{-4} and patience (that is the maximum number of epochs without LAS increase on the development set) set to 20.

We train 12 independent models for each expert type, with random seed set to system time.

6.1 Averaging Effect Analysis

The experimental procedure is shown in Experimental Setup 1, with $M_1, \ldots, M_{12}$ denoting the trained experts, K number of experts in the mixture model and r the number of repetitions.

Models: $M_1, \ldots, M_{12}$;
Initialization: K, r;
repeat r **times**
1. Shuffle the order of $M_1, \ldots, M_{12}$;
2. Combine sequentially every K models together, creating $12/K$ mixture averaging models;
3. Compute UAS, LAS of models;
4. Calculate system variance for models;

end
Experimental Setup 1: Averaging Effect

We set K from 1 to 6 with r always set to 5. We show results for PTB and CoNLL09 Chinese on dev data for each type of mixture of experts, and different number of experts in Table 1 and Table 3. For UAS and LAS, each entry is given as:

$$\text{Average}_{\text{min}}^{\text{max}} \pm \text{std}$$

where average is the *average* score for all trials in this setting and *max* (resp. *min*) is the highest (resp. *lowest*) score obtained by an experiment in this setting. We also give standard deviation *std* as a way to see the effects of variance reduction.

Finally the last row gives the average relative error reduction (R.E.R) from single expert mode ($K = 1$) to ensemble mode with $K = 6$.

6.2 Clustering Effect Analysis

We conduct clustering effect analysis over the mixture model with 6 experts. Preliminary experiments showed that, like in most non-convex problems, good initialization is very important. For that reason we use already trained experts as starting points[6] although the mixture could benefit from more diversely trained experts. We leave this for future work. The procedure is described in Experimental Setup 2 and this whole procedure is repeated 5 times to compute average performance.

Models: $M_1, \ldots M_{12}$;
Initialization: $K = 6$;
repeat r **times**
1. Select randomly K models, creating mixture models;
2. Do fine tuning of mixture models with gating network;
3. Calculating UAS, LAS of mixture model after fine tuning;

end
Experimental Setup 2: Clustering Effect

Scores on development set before and after fine tuning are shown in Table 4. Note that because shuffling might give different candidate sets than in the averaging experiments UAS and LAS results are not exactly the same as $K = 6$ results in Table 1, Table 2 and Table 3.

6.3 Discussion

Averaging Tables 1 to 3 show that UAS and LAS generally increase on average with the number of models in the mixture model, and that ensemble performs often on average better than the best single systems in each category (notable exceptions: UAS for FOP and models with BERT on PTB).

Averaging generally decreases the standard deviation, which is evident for (B)FOP. For (B)SOP the decrease trend is less clear. However, we still found that the smallest standard deviation is usually given by high number of experts ($K = 5, 6$).

[6] We tried deterministic annealing with both randomly initialized experts and already trained experts. While it helped in the former case, the latter was more accurate, but still less accurate than systems trained without.

K	FOP		SOP	
	UAS	LAS	UAS	LAS
1	$95.83^{\mathbf{96.04}}_{95.72}$ ±0.08	$94.06^{94.24}_{93.91}$ ±0.08	$95.87^{95.94}_{95.77}$ ±0.06	$94.07^{94.16}_{93.97}$ ±0.05
2	$95.88^{\mathbf{96.04}}_{95.76}$ ±0.06	$94.15^{94.32}_{94.05}$ ±0.07	$95.92^{96.05}_{95.85}$ ±0.05	$94.15^{94.27}_{94.08}$ ±0.04
3	$95.93^{96.03}_{95.84}$ ±0.05	$94.22^{94.32}_{94.11}$ ±0.05	$95.94^{96.04}_{95.85}$ ±0.06	$94.18^{94.27}_{94.08}$ ±0.06
4	$\mathbf{95.95}^{\mathbf{96.04}}_{95.90}$ ±0.04	$\mathbf{94.24}^{\mathbf{94.35}}_{94.16}$ ±0.05	$\mathbf{95.98}^{\mathbf{96.07}}_{95.91}$ ±0.05	$94.22^{\mathbf{94.31}}_{94.15}$ ±0.04
5	$95.93^{96.00}_{95.84}$ ±0.04	$\mathbf{94.24}^{94.33}_{94.14}$ ±0.05	$\mathbf{95.98}^{96.04}_{95.92}$ ±0.04	$\mathbf{94.24}^{94.29}_{\mathbf{94.18}}$ ±0.03
6	$\mathbf{95.95}^{95.98}_{\mathbf{95.91}}$ ±0.02	$\mathbf{94.24}^{94.30}_{\mathbf{94.21}}$ ±0.03	$\mathbf{95.98}^{96.01}_{\mathbf{95.94}}$ ±0.02	$\mathbf{94.24}^{94.28}_{\mathbf{94.18}}$ ±0.03
R.E.R.	2.88%	3.03%	2.66%	2.87%

Table 1: PTB dev results, with First-Order (FOP) and Second-Order (SOP) parsers as experts.

K	BFOP		BSOP	
	UAS	LAS	UAS	LAS
1	$96.31^{96.46}_{96.23}$ ±0.06	$94.60^{94.77}_{94.53}$ ±0.06	$96.35^{96.42}_{96.23}$ ±0.04	$94.63^{94.68}_{94.55}$ ±0.04
2	$96.37^{96.48}_{96.26}$ ±0.05	$94.69^{94.79}_{94.61}$ ±0.05	$96.38^{96.51}_{96.26}$ ±0.06	$94.71^{94.79}_{95.60}$ ±0.05
3	$96.40^{96.49}_{96.33}$ ±0.04	$94.74^{94.82}_{94.68}$ ±0.04	$96.39^{96.50}_{96.29}$ ±0.06	$94.71^{94.79}_{94.61}$ ±0.04
4	$96.43^{\mathbf{96.53}}_{96.38}$ ±0.04	$94.77^{\mathbf{94.89}}_{94.72}$ ±0.04	$96.38^{96.47}_{96.28}$ ±0.05	$94.72^{94.79}_{94.62}$ ±0.05
5	$\mathbf{96.45}^{96.51}_{96.38}$ ±0.04	$\mathbf{94.79}^{94.85}_{\mathbf{94.74}}$ ±0.04	$\mathbf{96.41}^{\mathbf{96.52}}_{96.29}$ ±0.06	$\mathbf{94.73}^{\mathbf{94.82}}_{94.65}$ ±0.05
6	$96.44^{96.51}_{\mathbf{96.40}}$ ±0.03	$\mathbf{94.79}^{94.85}_{\mathbf{94.74}}$ ±0.03	$96.39^{96.46}_{\mathbf{96.32}}$ ±0.04	$\mathbf{94.73}^{\mathbf{94.82}}_{\mathbf{94.67}}$ ±0.04
R.E.R.	3.52%	3.52%	1.64%	1.86%

Table 2: PTB dev results, with Bert-First-Order (BFOP) and Bert-Second-Order (BSOP) parsers as experts.

K	FOP		SOP	
	UAS	LAS	UAS	LAS
1	$89.20^{89.42}_{89.04}$ ±0.12	$86.28^{86.49}_{86.10}$ ±0.12	$89.40^{89.48}_{89.31}$ ±0.06	$86.45^{86.52}_{86.28}$ ±0.07
2	$89.44^{89.60}_{89.32}$ ±0.08	$86.59^{86.74}_{86.45}$ ±0.08	$89.65^{89.78}_{89.51}$ ±0.07	$86.76^{86.89}_{86.57}$ ±0.07
3	$89.55^{89.66}_{89.39}$ ±0.07	$86.71^{86.82}_{86.54}$ ±0.07	$89.74^{89.86}_{89.62}$ ±0.07	$86.86^{86.98}_{86.72}$ ±0.08
4	$89.62^{89.68}_{89.52}$ ±0.04	$86.80^{86.87}_{86.70}$ ±0.05	$89.83^{89.94}_{89.70}$ ±0.08	$86.96^{87.08}_{86.86}$ ±0.07
5	$\mathbf{89.66}^{89.71}_{89.57}$ ±0.04	$86.83^{86.89}_{86.75}$ ±0.04	$\mathbf{89.87}^{\mathbf{89.98}}_{89.75}$ ±0.07	$\mathbf{87.00}^{\mathbf{87.11}}_{86.87}$ ±0.07
6	$\mathbf{89.66}^{\mathbf{89.77}}_{\mathbf{89.61}}$ ±0.05	$\mathbf{86.85}^{\mathbf{86.93}}_{\mathbf{86.81}}$ ±0.04	$\mathbf{89.87}^{89.93}_{\mathbf{89.79}}$ ±0.05	$\mathbf{87.00}^{87.08}_{\mathbf{86.92}}$ ±0.04
R.E.R.	4.26%	4.15%	4.43%	4.06%

Table 3: CoNLL09 dev results, with First-Order (FOP) and Second-Order (SOP) parsers as experts.

Method	PTB		CoNLL09 Chinese	
	UAS	LAS	UAS	LAS
FOP	$95.94^{95.96}_{95.91}$ ±0.02	$94.23^{94.26}_{94.21}$ ±0.02	$89.67^{89.71}_{89.62}$ ±0.03	$\mathbf{86.86}^{86.91}_{86.81}$ ±0.04
CFOP	$\mathbf{95.98}^{96.00}_{95.94}$ ±0.02	$\mathbf{94.29}^{94.31}_{94.27}$ ±0.02	$\mathbf{89.68}^{89.72}_{89.62}$ ±0.04	$\mathbf{86.86}^{86.90}_{86.80}$ ±0.04
SOP	$95.98^{96.00}_{95.95}$ ±0.02	$94.23^{94.28}_{94.20}$ ±0.03	$89.85^{89.92}_{89.81}$ ±0.04	$86.98^{87.06}_{86.93}$ ±0.06
CSOP	$\mathbf{95.99}^{96.01}_{95.97}$ ±0.01	$\mathbf{94.25}^{94.28}_{94.22}$ ±0.02	$\mathbf{89.89}^{89.95}_{89.82}$ ±0.04	$\mathbf{87.03}^{87.12}_{86.95}$ ±0.06
BFOP	$\mathbf{96.43}^{96.46}_{96.42}$ ±0.02	$\mathbf{94.79}^{94.82}_{94.76}$ ±0.02	-	-
CBFOP	$96.42^{96.46}_{96.39}$ ±0.03	$94.78^{94.81}_{94.72}$ ±0.03	-	-
BSOP	$96.41^{96.46}_{96.37}$ ±0.04	$94.75^{94.82}_{94.67}$ ±0.05	-	-
CBSOP	$\mathbf{96.42}^{96.46}_{96.38}$ ±0.04	$\mathbf{94.76}^{94.82}_{94.70}$ ±0.05	-	-

Table 4: Clustering Effect with $K = 6$ on dev, where CFOP, CSOP, CBSOP represent models after training

We remark that on PTB similar performance on dev was achieved by FOP and SOP, with a slightly better UAS for SOP, which is expected by the capacity of the model to better represent structures. This corroborates findings of (Falenska and Kuhn, 2019). But this contradicts results for CoNLL09 where SOP always gives best results, in line with observations of Fonseca and Martins (2020). For BERT experiments on PTB, BSOP achieves better performance than BFOP with one or two experts. However, when the number of experts increases, BFOP outperforms BSOP.

We complement our discussion with Figure 1[7] which depicts variance reduction by the number of

[7]For CoNLL09, we found similar results. The figure is not shown for space limitation.

experts in log-scale: almost linear of for all models, as predicted by our independence assumption.

We note that UAS and LAS improves little or not at all from $K = 5$ to $K = 6$. This is in accordance with the variance analysis for that the decrease of variance will become smaller when number of experts becomes higher. Indeed, applying Eq. (4), the decrease of variance from $K = 1$ to $K = 2$ is $\frac{1}{2}\sigma^2_{(h,d),x}$, while from $K = 5$ to $K = 6$ it is only $\frac{1}{30}\sigma^2_{(h,d),x}$, 15 times lower. This correponds to the observation the improvements of UAS and LAS tend to decrease with the number of experts until it reaches a plateau.

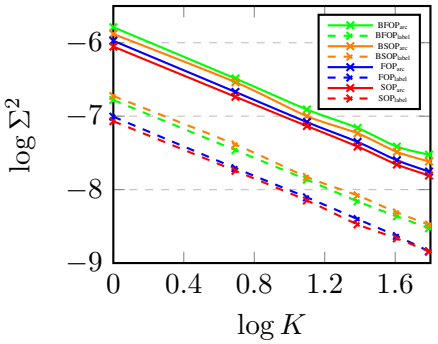

Figure 1: Variance by experts on PTB Dev Data.

Clustering We found that a modest improvement on UAS and LAS (0.01%-0.06% absolute) can be achieved by clustering (except for FOP on CoNLL09 Chinese). The average performance benefits generally from clustering while a tiny decrease (0.01%) is observed for BFOP on PTB.

Since FOP, SOP, BFOP and BSOP are all strong learners for PTB and CoNLL09 Chinese, *i.e.* UAS and LAS approaches 99% for both PTB and CoNLL09 on training data for all models, we can assume that an expert belonging to one of these models can learn efficiently most of the training data, as opposed to just a portion of it. Thus, only a a few of training instances can significantly be better covered by clustering. Moreover, as averaging has already achieved a considerable improvement (around 0.2%-0.6% absolute), a biased ω_k obtained from clustering may harm the gain from averaging.

6.4 Results on Test

Tables 5 and 6 show test results on PTB and CoNLL09, comparisons with recent models. We show test results of SOP and CSOP with 6 experts for PTB and CoNLL09. Additionally for PTB, we show BFOP, CBFOP, BSOP and CBSOP with 6 experts to make comparison with recent parsers, often more sophisticated than our approach, with BERT. We give the results with the same typographical system as Zhang et al. (2020a) Please note that, while average results keep the same semantics, *max* and *min* give test results of the LAS highest- and lowest- (resp.) scoring systems *on the development set*. We note that results of Zhang et al. (2020a) would correspond our model with $K = 1$.

For averaging models, we apply significance t-tests (Dror et al., 2018) with level $\alpha = 0.05$ to FOP, BFOP, SOP, BSOP with $K = 6$ against $K = 1$. For PTB and CoNLL09, p-value is always smaller than 0.005. We note that for parsers without BERT, averaging can achieve a considerable improvement with SOP and gives new SOTA. We also point out that, if FOP and SOP could find equivalently good models on dev, SOP models seem to better generalize. For parsers with BERT, with a simple averaging of BSOP, we achieve comparable performances (or even better in case of LAS) when comparing to more involved methods such as (Li et al., 2020; Mohammadshahi and Henderson, 2021). It remains to be seen whether they can also benefit from MoEs.

Regarding clustering, even if we obtained an average improvement on dev, test data hardly benefits from it. Still, we note a small improvement of UAS on SOP CoNLL09. Finally we stress that best performing settings on PTB test, namely BSOP and CBSOP, were not better performing than BFOP and CBFOP on development data on average (although max systems were similar): second-order models seem to slightly better handle unseen data.

6.5 Parallel Training and Decoding

Training averaging ensembles can be paralleled with sufficient GPUs, since each expert is trained independently. For fine tuning with clustering, most of the training could in principle be paralleled as well, although for the sake of simplicity we didn't implement such a training procedure: the training time of clustering model increases linearly with number of experts. As we only need a few epochs for fine tuning, the overall training time is comparable to training a single expert.

For decoding, calculations are performed in parallel as well. First marginal probabilities for arcs and labels are computed for every expert in parallel. Then they are combined either as a simple average or as a weighted sum. Finally, we apply the decoding algorithm (LMBR) *once* over the combined

Method	PTB		CoNLL09 Chinese	
	UAS	LAS	UAS	LAS
(Dozat and Manning, 2017)	95.74	94.08	88.90	85.38
(Li et al., 2019)	95.93	94.19	88.77	85.58
(Ji et al., 2019)	95.97	94.31	-	-
(Zhang et al., 2020a)	96.14	94.49	89.63	86.52
FOP,$K = 6$	$96.20^{96.19}_{96.20}$ ±0.02	$94.64^{94.63}_{94.64}$ ±0.02	$89.91^{89.84}_{89.99}$ ±0.06	$87.00^{86.94}_{87.09}$ ±0.07
CFOP,$K = 6$	$96.20^{96.18}_{96.18}$ ±0.02	$94.65^{94.62}_{94.63}$ ±0.02	$89.94^{89.92}_{89.93}$ ±0.04	$87.03^{87.02}_{87.00}$ ±0.04
SOP,$K = 6$	$\mathbf{96.29}^{96.30}_{96.29}$ ±0.02	$\mathbf{94.71}^{94.72}_{94.73}$ ±0.02	$90.06^{90.14}_{89.97}$ ±0.07	$\mathbf{87.12}^{87.19}_{87.00}$ ±0.07
CSOP,$K = 6$	$96.27^{96.27}_{96.32}$ ±0.03	$94.69^{94.70}_{94.72}$ ±0.03	$\mathbf{90.07}^{90.00}_{89.99}$ ±0.08	$\mathbf{87.12}^{87.24}_{87.02}$ ±0.09

Table 5: Comparison on test sets without BERT.

Method	PTB	
	UAS	LAS
(Li et al., 2020)	96.44	94.63
(Mohammadshahi and Henderson, 2021)	**96.66**	95.01
BFOP,$K = 6$	$96.58^{96.60}_{96.57}$ ±0.02	$95.06^{95.07}_{95.02}$ ±0.02
CBFOP,$K = 6$	$96.58^{96.59}_{96.54}$ ±0.02	$95.06^{95.07}_{95.02}$ ±0.02
BSOP,$K = 6$	$96.64^{\mathbf{96.66}}_{96.58}$ ±0.02	$\mathbf{95.09}^{95.11}_{95.12}$ ±0.03
CBSOP,$K = 6$	$96.62^{\mathbf{96.66}}_{96.64}$ ±0.03	$95.07^{\mathbf{95.12}}_{95.07}$ ±0.03

Table 6: Comparison of BERT models on PTB test set.

probability. The overhead is thus quite limited, for instance with $K = 6$ the overall decoding time is only around 10% higher than with a single expert.

7 Related Work

Ensembling parsers showed good results in shared tasks (Che et al., 2018)[8] and were framed as a combination of experts in (Petrov, 2010). In this work we show how this is related to mixtures and distinguish averaging and clustering effects.

The use of mixture model for syntactic parsing was introduced in (Petrov et al., 2006) for PCFG models, where it provided an access to non-local features unreachable to mere PCFGs. However, now that powerful non-Markovian feature extractors (*i.e.* BiLSTMs or Transformers) are widely used, the expected gain is more difficult to characterize, but we hypothesize that it is related to the softmax bottleneck (Yang et al., 2018) implied by using different exponential models in all predictions, even when richly parameterized.

We modelled parser combinations with finite mixture models, but more sophisticated parsing models (Kim et al., 2019) use infinite mixture models. In this case it might be more difficult to discriminate between averaging and clustering. Our mixture is essentially a latent variable model where the latent variables range over experts. Although inspired from EM with neural networks, similarly to (Nishida and Nakayama, 2020), other methods based on ELBo and sampling could also be utilized (Corro and Titov, 2019; Zhu et al., 2020).

8 Conclusion

We framed dependency parser combination as a finite mixture model, showed that this model presents two distinct properties –an averaging effect and a clustering effect– and devised an efficient decoding method. Moreover, we studied the impact of the averaging effect, namely variance reduction during training, and consequently better accuracy. We investigated the reasons of instability when learning mixture models, and proposed an EM-inspired method to avoid over-specialization. When used as fine-tuning, this method may improve accuracy over averaging. As a by-product, this method gives state-of-the-art results when combined with first-order and second-order projective parsers on two standard datasets.

This work can be further expanded in future research: the increase of parameters can be seen as overparameterization, and many parameters must be redundant. A potentially fruitful avenue of research could be the investigation of the subnetwork hypothesis, *i.e.* whether distillation could give a smaller network with similar performance.

Acknowledgments

This work is partially supported by a public grant overseen by the French National Research Agency (ANR) as part of the program Investissements d'Avenir (ANR-10-LABX-0083). It contributes to the IdEx Université de Paris (ANR-18-IDEX-0001). This work is partially supported by a public grant overseen by the French ANR (ANR-16-CE33-0021).

[8] Ensembling is widely used in Machine Translation shared tasks, such as WMT.

References

Paolo Baldi. 2017. Stochastic calculus. In *Stochastic Calculus*, pages 1–14. Springer.

Andrew D. Brown and Geoffrey E. Hinton. 2001. Products of hidden markov models. In *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics, AISTATS 2001, Key West, Florida, USA, January 4-7, 2001*. Society for Artificial Intelligence and Statistics.

Wanxiang Che, Yijia Liu, Yuxuan Wang, Bo Zheng, and Ting Liu. 2018. Towards better UD parsing: Deep contextualized word embeddings, ensemble, and treebank concatenation. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 55–64, Brussels, Belgium. Association for Computational Linguistics.

Caio Corro and Ivan Titov. 2019. Differentiable perturb-and-parse: Semi-supervised parsing with a structured variational autoencoder. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Rotem Dror, Gili Baumer, Segev Shlomov, and Roi Reichart. 2018. The hitchhiker's guide to testing statistical significance in natural language processing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1383–1392, Melbourne, Australia. Association for Computational Linguistics.

Jason Eisner. 1996. Efficient normal-form parsing for Combinatory Categorial Grammar. In *34th Annual Meeting of the Association for Computational Linguistics*, pages 79–86, Santa Cruz, California, USA. Association for Computational Linguistics.

Jason Eisner. 1997. Bilexical grammars and a cubic-time probabilistic parser. In *Proceedings of the Fifth International Workshop on Parsing Technologies*, pages 54–65, Boston/Cambridge, Massachusetts, USA. Association for Computational Linguistics.

Jason Eisner. 2016. Inside-outside and forward-backward algorithms are just backprop (tutorial paper). In *Proceedings of the Workshop on Structured Prediction for NLP*, pages 1–17, Austin, TX. Association for Computational Linguistics.

Agnieszka Falenska and Jonas Kuhn. 2019. The (non-)utility of structural features in BiLSTM-based dependency parsers. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 117–128, Florence, Italy. Association for Computational Linguistics.

Erick Fonseca and André F. T. Martins. 2020. Revisiting higher-order dependency parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8795–8800, Online. Association for Computational Linguistics.

Victoria Fossum and Kevin Knight. 2009. Combining constituent parsers. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 253–256, Boulder, Colorado. Association for Computational Linguistics.

Vaibhava Goel and William J. Byrne. 2000. Minimum bayes-risk automatic speech recognition. *Comput. Speech Lang.*, 14(2):115–135.

Nathan Green and Zdeněk Žabokrtský. 2012. Hybrid combination of constituency and dependency trees into an ensemble dependency parser. In *Proceedings of the Workshop on Innovative Hybrid Approaches to the Processing of Textual Data*, pages 19–26, Avignon, France. Association for Computational Linguistics.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pages 1–18, Boulder, Colorado. Association for Computational Linguistics.

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

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87.

Tao Ji, Yuanbin Wu, and Man Lan. 2019. Graph-based dependency parsing with graph neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2475–2485, Florence, Italy. Association for Computational Linguistics.

Yoon Kim, Chris Dyer, and Alexander Rush. 2019. Compound probabilistic context-free grammars for grammar induction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2369–2385, Florence, Italy. Association for Computational Linguistics.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, San Diego, California. Association for Computational Linguistics.

Joseph Le Roux, Antoine Rozenknop, and Mathieu Lacroix. 2019. Representation learning and dynamic programming for arc-hybrid parsing. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 238–248, Hong Kong, China. Association for Computational Linguistics.

Ying Li, Zhenghua Li, Min Zhang, Rui Wang, Sheng Li, and Luo Si. 2019. Self-attentive biaffine dependency parsing. In *IJCAI*, pages 5067–5073.

Zuchao Li, Hai Zhao, and Kevin Parnow. 2020. Global greedy dependency parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8319–8326.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

Alireza Mohammadshahi and James Henderson. 2021. Recursive Non-Autoregressive Graph-to-Graph Transformer for Dependency Parsing with Iterative Refinement. *Transactions of the Association for Computational Linguistics*, 9:120–138.

Noriki Nishida and Hideki Nakayama. 2020. Unsupervised discourse constituency parsing using Viterbi EM. *Transactions of the Association for Computational Linguistics*, 8:215–230.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Slav Petrov. 2010. Products of random latent variable grammars. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 19–27, Los Angeles, California. Association for Computational Linguistics.

Slav Petrov, Leon Barrett, and Dan Klein. 2006. Non-local modeling with a mixture of PCFGs. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 14–20, New York City. Association for Computational Linguistics.

Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. 2018. On the convergence of adam and beyond. In *International Conference on Learning Representations*.

Alexander M. Rush, David Sontag, Michael Collins, and Tommi Jaakkola. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Cambridge, MA. Association for Computational Linguistics.

David A. Smith and Noah A. Smith. 2007. Probabilistic models of nonprojective dependency trees. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 132–140, Prague, Czech Republic. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. 2018. Breaking the softmax bottleneck: A high-rank RNN language model. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Yu Zhang, Zhenghua Li, and Min Zhang. 2020a. Efficient second-order TreeCRF for neural dependency parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3295–3305, Online. Association for Computational Linguistics.

Yu Zhang, Houquan Zhou, and Zhenghua Li. 2020b. Fast and accurate neural CRF constituency parsing. In *Proceedings of IJCAI*, pages 4046–4053.

Junru Zhou, Zuchao Li, and Hai Zhao. 2020. Parsing all: Syntax and semantics, dependencies and spans. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4438–4449, Online. Association for Computational Linguistics.

Hao Zhu, Yonatan Bisk, and Graham Neubig. 2020. The return of lexical dependencies: Neural lexicalized pcfgs.

Ran Zmigrod, Tim Vieira, and Ryan Cotterell. 2020. Efficient computation of expectations under spanning tree distributions.

A Marginal Probability of Arc for Mixture Model

With Eq. (3). The marginal probability of mixture model can be written as:

$$p((h,d)|x) = \sum_{\substack{y \in \mathcal{Y}(x) \\ (h,d) \in y}} \sum_{k=1}^{K} \omega_k p_k(y|x)$$

By changing the order of sum, we can have:

$$p((h,d)|x) = \sum_{k=1}^{K} \omega_k \sum_{\substack{y \in \mathcal{Y}(x) \\ (h,d) \in y}} p_k(y|x)$$

The inner part is exactly $p_k((h,d)|x)$. Thus, we have:

$$p((h,d)|x) = \sum_{k=1}^{K} \omega_k p_k((h,d)|x)$$

B Quick Gradient Analysis of Gating Network

We start from Eq. (6).

For mixture model with well trained experts, most of the data are equivalent for all experts, which means $p_k(y|x)$ have similar value for all experts. To see quickly why gradient approaches 0 in this case, we assume further that $p_k(y|x)$ has the same value for equivalent data. Thus, Eq. (6) becomes:

$$\frac{\partial L(\phi,\theta)}{\partial \phi} = \sum_{k=1}^{K} \omega_k(\phi) \frac{\partial \log \omega_k(\phi)}{\partial \phi}$$

With a little more deduction, we have:

$$\begin{aligned}
\frac{\partial L(\phi,\theta)}{\partial \phi} &= \sum_{k=1}^{K} \omega_k(\phi) \frac{1}{\omega_k(\phi)} \frac{\partial \omega_k(\phi)}{\partial \phi} \\
&= \sum_{k=1}^{K} \frac{\partial \omega_k(\phi)}{\partial \phi} \\
&= \frac{\partial \sum_{k=1}^{K} \omega_k(\phi)}{\partial \phi} \\
&= \frac{\partial 1}{\partial \phi} \\
&= 0
\end{aligned}$$

As the function is continuous w.r.t. p_k, for data which provides similar value of probability on all experts, the gradient will approaches zero. Thus, for training with Eq. (6), only a small part of data, which shows strong preference of particular experts, is used to train the gating network.

For training with Eq. (8), all the data is useful for training the gating network. In fact, the gradient of Eq. (8) becomes zero when:

$$\omega_k(\phi) = \frac{p_k(\theta_k)}{\sum_{k'=1}^{K} p_{k'}(\theta_{k'})}$$

Thus for data which are equivalent for all experts, a uniform weight will be learnt while for data with strong preference of particular experts, a biased weight proportional to the probability correctness on each expert can also be learnt.

C Gating Network Structure, hyper-parameters of training

The gating network structure is similar to the structure of parse model.

Embedding Word embedding for word x_i is an concatenation of two parts: normal word embedding and CharLSTM embedding:

$$e_i = \text{emb}(x_i) \bigoplus \text{CharLSTM}(x_i)$$

when there is pre-trained embedding, the first item is the sum of word embedding calculated by neural network, and the exterior pretrained embedding:

$$\text{emb}(x_i) = \text{WordEMB}(x_i) + \text{PreEMB}(x_i)$$

We suppose that PreEMB has the same size as WordEMB

BiLSTM The embedding vectors are then passed to 3 layers of BiLSTM, with the output at position i is noted as h_i.

Coefficient Extractor The coefficient extractor part is constructed of one layer of LSTM (Hochreiter and Schmidhuber, 1997) and one layer of MLP. The last hidden state of LSTM is passed to MLP, which compress the vector size to the number of experts in the mixture model. Two groups of coefficient extractor are used to calculate separately the weight of combination for arc and label. We note the output of MLP as $\mathbf{C} \in \mathbb{R}^K$, with:

$$\mathbf{C}_{\text{arc}} = \text{MLP}_{\text{arc}}(\text{LSTM}_{\text{arc}}(h_0, ..., h_n))$$

$$\mathbf{C}_{\text{label}} = \text{MLP}_{\text{label}}(\text{LSTM}_{\text{label}}(h_0, ..., h_n))$$

The output of MLP is passed to Softmax to calculate the weight for each expert:

$$[\omega_1, ..., \omega_K] = \text{Softmax}(\mathbf{C}_{\text{arc}})$$

$$[\omega_1^l, ..., \omega_K^l] = \text{Softmax}(\mathbf{C}_{\text{label}})$$

Model hyper-parameters of fine tuning is shown in Table 7. We use also Adam (Reddi et al., 2018) for training, with learning rate set to $2e^{-4}$ (10 times smaller than learning rate used for training experts). The patience is set to 20 instead of the original value 100. For fine tuning, we found that best score is usually achieved in less than 20 epochs and does not increase later.

Param	Value	Param	Value
WordEMB size	100	Embedding dropout	0.33
CharLSTM size	50	CharLSTM dropout	0.00
BiLSTM size	400	BiLSTM dropout	0.33
LSTM_{arc} size	400	LSTM_{arc} dropout	0.00
$\text{LSTM}_{\text{label}}$ size	400	$\text{LSTM}_{\text{label}}$ dropout	0.00
MLP_{arc} size	K	MLP_{arc} dropout	0.00
$\text{MLP}_{\text{label}}$ size	K	$\text{MLP}_{\text{label}}$ dropout	0.00
Learning Rate	$\mathbf{2e^{-4}}$	β_1, β_2	0.90
Annealing	$0.75^{\frac{t}{5000}}$	Patience	**20**

Table 7: Hyper-parameters of Fine Tuning

D Implementation Differences

We implement Zhang et al. (2020a) CRF model and CRF2o model with two tiny technical differences.

The first one is that the CharLSTM (Lample et al., 2016) part in Zhang et al. (2020a) treats the beginning of the sentence `<bos>` (the special token to represent the beginning of the sentence) as five separate characters: `<,b,o,s,>`.

Our implementation treats the beginning of sentence as one special character for CharLSTM.

Another difference is that Zhang et al. (2020a) treats the lengths of every sentence as $n + 2$ by considering two special tokens `<bos>` and `<eos>` (although in practice, only `<bos>` was added to every sentence). In our implementation, we keep the length of sentence as the number of words n. This is because the log probability of arc and label only considers the words in the sentence without special tokens. Thus our batch size should be a little bit higher than Zhang et al. (2020a).

One final difference is that for MBR decoding, (Zhang et al., 2020a) maximizes the sum of marginal arc probability. While in our implementation of MBR, we maximize the product of marginal arc probability.

E Variance Reduction on CoNLL09

We note that the label variance for FOP and SOP are quite similar that they overlap together for CoNLL09 Chinese.

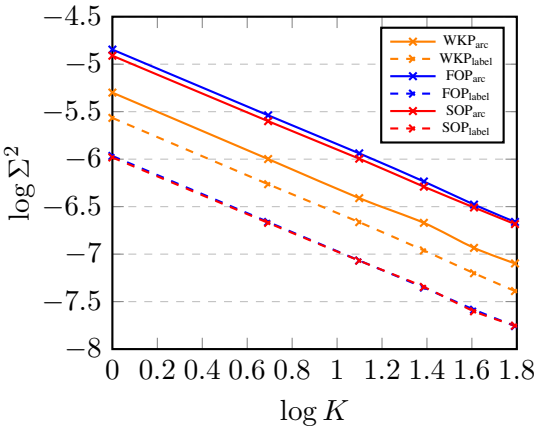

Figure 2: Variance of System to CoNLL09 Chinese