Specializing Word Embeddings (for Parsing) by Information Bottleneck

Xiang Lisa Li
Department of Computer Science
Johns Hopkins University
xli150@jhu.edu

Jason Eisner
Department of Computer Science
Johns Hopkins University
jason@cs.jhu.edu

Abstract

Pre-trained word embeddings like ELMo and BERT contain rich syntactic and semantic information, resulting in state-of-the-art performance on various tasks. We propose a very fast variational information bottleneck (VIB) method to nonlinearly compress these embeddings, keeping only the information that helps a discriminative parser. We compress each word embedding to either a discrete tag or a continuous vector. In the *discrete* version, our automatically compressed tags form an alternative tag set: we show experimentally that our tags capture most of the information in traditional POS tag annotations, but our tag sequences can be parsed more accurately at the same level of tag granularity. In the *continuous* version, we show experimentally that moderately compressing the word embeddings by our method yields a more accurate parser in 8 of 9 languages, unlike simple dimensionality reduction.

1 Introduction

Word embedding systems like BERT and ELMo use spelling and context to obtain contextual embeddings of word tokens. These systems are trained on large corpora in a task-independent way. The resulting embeddings have proved to then be useful for both syntactic and semantic tasks, with different layers of ELMo or BERT being somewhat specialized to different kinds of tasks (Peters et al., 2018b; Goldberg, 2019). State-of-the-art performance on many NLP tasks can be obtained by fine-tuning, i.e., back-propagating task loss all the way back into the embedding function (Peters et al., 2018a; Devlin et al., 2018).

In this paper, we explore what task-specific information appears in the embeddings *before* fine-tuning takes place. We focus on the task of dependency parsing, but our method can be easily extended to other syntactic or semantic tasks. Our method compresses the embeddings by extracting just their syntactic properties—specifically, the information needed to reconstruct parse trees (because that is our task). Our nonlinear, stochastic compression function is explicitly trained by variational information bottleneck (VIB) to forget task-irrelevant information. This is reminiscent of canonical correspondence analysis (Anderson, 2003), a method for reducing the dimensionality of an input vector so that it remains predictive of an output vector, although we are predicting an output tree instead. However, VIB goes beyond mere dimensionality reduction to a fixed lower dimensionality, since it also avoids unnecessary *use* of the dimensions that are available in the compressed representation, blurring unneeded capacity via randomness. The effective number of dimensions may therefore vary from token to token. For example, a parser may be content to know about an adjective token only that it is adjectival, whereas to find the dependents of a verb token, it may need to know

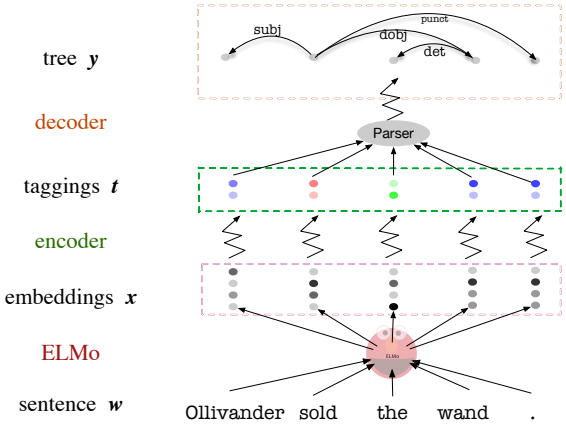


Figure 1: Our instantiation of the information bottleneck, with bottleneck variable T. A jagged arrow indicates a stochastic mapping, i.e. the jagged arrow points from the parameters of a distribution to a sample drawn from that distribution.

the verb's number and transitivity, and to attach a preposition token, it may need to know the identity of the preposition.

We try compressing to both discrete and continuous task-specific representations. Discrete representations yield an interpretable clustering of words. We also extend information bottleneck to allow us to control the contextual specificity of the token embeddings, making them more like type embeddings.

This specialization method is complementary to the previous fine-tuning approach. Fine-tuning introduces *new* information into word embeddings by backpropagating the loss, whereas the VIB method learns to exploit the *existing* information found by the ELMo or BERT language model. VIB also has less capacity and less danger of overfitting, since it fits fewer parameters than fine-tuning (which in the case of BERT has the freedom to adjust the embeddings of all words and word pieces, even those that are rare in the supervised fine-tuning data). VIB is also very fast to train on a single GPU.

We discover that our syntactically specialized embeddings are predictive of the gold POS tags in the setting of few-shot-learning, validating the intuition that a POS tag summarizes a word token's *syntactic* properties. However, our representations are tuned explicitly for discriminative parsing, so they prove to be even more useful for this task than POS tags, even at the same level of granularity. They are also more useful than the uncompressed ELMo representations, when it comes to generalizing to test data. (The first comparison uses discrete tags, and the second uses continuous tags.)

2 Background: Information Bottleneck

The information bottleneck (IB) method originated in information theory and has been adopted by the machine learning community as a training objective (Tishby et al., 2000) and a theoretical framework for analyzing deep neural networks (Tishby and Zaslavsky, 2015b).

Let X represent an "input" random variable such as a sentence, and Y represent a correlated "output" random variable such as a parse. Suppose we know the joint distribution $p(X, Y)$. (In practice, we will use the empirical distribution over a sample of (x, y) pairs.) Our goal is to learn a stochastic map $p_\theta(t \mid x)$ from X to some compressed representation T, which in our setting will be something like a tag sequence. IB seeks to minimize

$$\mathcal{L}_{IB} = -\,\mathrm{I}(Y;T) + \beta \cdot \mathrm{I}(X;T) \qquad (1)$$

where $\mathrm{I}(\cdot;\cdot)$ is the mutual information.[1] A low loss means that T does not retain very much information about X (the second term), while still retaining enough information to predict Y.[2] The balance between the two MI terms is controlled by a Lagrange multiplier β. By increasing β, we increase the pressure to keep $\mathrm{I}(X;T)$ small, which "narrows the bottleneck" by favoring compression over predictive accuracy $\mathrm{I}(Y;T)$. Regarding β as a Lagrange multiplier, we see that the goal of IB is to maximize the predictive power of T subject to some constraint on the amount of information about X that T carries. If the map from X to T were deterministic, then it could lose information only by being non-injective: the traditional example is dimensionality reduction, as in the encoder of an encoder-decoder neural net. But IB works even if T can take values throughout a high-dimensional space, because the randomness in $p_\theta(t \mid x)$ means that T is noisy in a way that wipes out information about X. Using a high-dimensional space is desirable because it permits the amount of effective dimensionality reduction to vary, with T perhaps retaining much more information about some x values than others, as long as the *average* retained information $\mathrm{I}(X;T)$ is small.

3 Formal Model

In this paper, we extend the original IB objective (1) and add terms $\mathrm{I}(T_i; X|\hat{X}_i)$ to control the context-sensitivity of the extracted tags. Here T_i is the tag associated with the ith word, X_i is the ELMo token embedding of the ith word, and $\hat{X}_i$ is the same word's ELMo type embedding (before context is incorporated).

$$\mathcal{L}_{IB} = -\,\mathrm{I}(Y;T) + \beta\,\mathrm{I}(X;T) + \gamma \sum_{i=1}^{n} \mathrm{I}(T_i; X|\hat{X}_i) \qquad (2)$$

In this section, we will explain the motivation for the additional term and how to efficiently estimate variational bounds on all terms (lower bound for $\mathrm{I}(Y;T)$ and upper bound for the rest).[3]

[1] In our IB notation, larger β means more compression. Note that there is another version of IB that puts β as the coefficient in front of $\mathrm{I}(Y;T)$: $\mathcal{L}_{IB} = -\beta \cdot \mathrm{I}(Y;T) + \mathrm{I}(X;T)$ The two versions are equivalent.

[2] Since T is a stochastic function of X with no access to Y, it obviously cannot convey more information about Y than the uncompressed input X does. As a result, Y is independent of T given X, as in the graphical model $T \to X \to Y$.

[3] Traditional Shannon entropy $\mathrm{H}(\cdot)$ is defined on discrete variables. In the case of continuous variables, we interpret H

We instantiate the variational IB (VIB) estimation method (Alemi et al., 2016) on our dependency parsing task, as illustrated in Figure 1. We compress a sentence's word embeddings X_i into continuous vector-valued tags or discrete tags T_i ("encoding") such that the tag sequence T retains maximum ability to predict the dependency parse Y ("decoding"). Our chosen architecture compresses each X_i independently using the same stochastic, information-losing transformation.

The IB method introduces the new random variable T, the tag sequence that compresses X, by defining the conditional distribution $p_\theta(t \mid x)$. In our setting, p_θ is a stochastic tagger, for which we will adopt a parametric form (§3.1 below). Its parameters θ are chosen to minimize the IB objective (2). By IB's independence assumption,[2] the joint probability can be factored as $p_\theta(x, y, t) = p(x) \cdot p(y \mid x) \cdot p_\theta(t \mid x)$.

3.1 $\mathrm{I}(X;T)$ — the Token Encoder $p_\theta(t \mid x)$

Under this distribution, $\mathrm{I}(X;T) \stackrel{\text{def}}{=} \mathbb{E}_{x,t}\left[\log \frac{p_\theta(t|x)}{p_\theta(t)}\right] = \mathbb{E}_x\left[\mathbb{E}_{t\sim p_\theta(t|x)}\left[\log \frac{p_\theta(t|x)}{p_\theta(t)}\right]\right]$. Making this term small yields a representation T that, on average, retains little information about X. The outer expectation is over the true distribution of sentences x; we use an empirical estimate, averaging over the unparsed sentences in a dependency treebank. To estimate the inner expectation, we could sample, drawing taggings t from $p_\theta(t \mid x)$.

We must also compute the quantities within the inner brackets. The $p_\theta(t \mid x)$ term is defined by our parametric form. The troublesome term is $p_\theta(t) = \mathbb{E}_{x'}\left[p_\theta(t \mid x')\right]$, since even estimating it from a treebank requires an inner loop over treebank sentences x'. To avoid this, variational IB replaces $p_\theta(t)$ with some variational distribution $r_\psi(t)$. This can only increase our objective function, since the difference between the variational and original versions of this term is a KL divergence and hence non-negative:

$$\overbrace{\mathop{\mathbb{E}}_{x}\Big[\mathop{\mathbb{E}}_{t\sim p_\theta(t|x)}\Big[\log \frac{p_\theta(t|x)}{r_\psi(t)}\Big]\Big]}^{\text{upper bound}} - \overbrace{\mathop{\mathbb{E}}_{x}\Big[\mathop{\mathbb{E}}_{t\sim p_\theta(t|x)}\Big[\log \frac{p_\theta(t \mid x)}{p_\theta(t)}\Big]\Big]}^{\mathrm{I}(X;T)}$$
$$= \mathop{\mathbb{E}}_{x}\left[\mathrm{KL}(p_\theta(t) \mid\mid r_\psi(t))\right] \geq 0$$

Thus, the variational version (the first term above) is indeed an upper bound for $\mathrm{I}(X;T)$ (the second term above). We will minimize this upper bound by adjusting not only θ but also ψ, thus making the bound as tight as possible given θ. Also we will no longer need to sample t for the inner expectation of the upper bound, $\mathbb{E}_{t\sim p_\theta(t|x)}\left[\log \frac{p_\theta(t|x)}{r_\psi(t)}\right]$, because this expectation equals $\mathrm{KL}[p_\theta(t \mid x) \mid\mid r_\psi(t)]$, and we will define the parametric p_θ and r_ψ so that this KL divergence can be computed exactly: see §4.

3.2 Two Token Encoder Architectures

We choose to define $p_\theta(t \mid x) = \prod_{i=1}^{n} p_\theta(t_i \mid x_i)$. That is, our stochastic encoder will compress each word x_i individually (although x_i is itself a representation that depends on context): see Figure 1. We make this choice not for computational reasons—our method would remain tractable even without this—but because our goal in this paper is to find the syntactic information in each individual ELMo token embedding (a goal we will further pursue in §3.3 below).

To obtain continuous tags, define $p_\theta(t_i \mid x_i)$ such that $t_i \in \mathbb{R}^d$ is Gaussian-distributed with mean vector and diagonal covariance matrix computed from the ELMo word vector x_i via a feedforward neural network with $2d$ outputs and no transfer function at the output layer. To ensure positive semidefiniteness of the diagonal covariance matrix, we squared the latter d outputs to obtain the diagonal entries.[4]

Alternatively, to obtain discrete tags, define $p_\theta(t_i \mid x_i)$ such that $t_i \in \{1, \ldots, k\}$ follows a softmax distribution, where the k softmax parameters are similarly computed by a feedforward network with k outputs and no transfer function at the output layer.

We similarly define $r_\psi(t) = \prod_{i=1}^{n} r_\psi(t_i)$, where ψ directly specifies the $2d$ or k values corresponding to the output layer above (since there is no input x_i to condition on).

3.3 $\mathrm{I}(T_i; X \mid \hat{X}_i)$ — the Type Encoder $s_\xi(t_i \mid \hat{x}_i)$

While the IB objective (1) asks each tag t_i to be informative about the parse Y, we were concerned that it might not be interpretable as a tag of word i specifically. Given ELMo or any other black-box conversion of a length-n sentence to a sequence of contextual vectors $x_1, \ldots, x_n$, it is possible that x_i

to instead denote differential entropy (which would be $-\infty$ for discrete variables). Scaling a continuous random variable affects its differential entropy—but not its mutual information with another random variable, which is what we use here.

[4] Our restriction to diagonal covariance matrices follows Alemi et al. (2016). In pilot experiments that dropped this restriction, we found learning to be numerically unstable, although that generalization is reasonable in principle.

contains not only information about word i but also information describing word $i+1$, say, or the syntactic constructions in the vicinity of word i. Thus, while $p_\theta(t_i \mid x_i)$ might extract some information from x_i that is very useful for parsing, there is no guarantee that this information came from word i and not its neighbors. Although we *do* want tag t_i to consider context—e.g., to distinguish between noun and verb uses of word i—we want "most" of t_i's information to come from word i itself. Specifically, it should come from ELMo's level-0 embedding of word i, denoted by $\hat{x}_i$—a word *type* embedding that does *not* depend on context.

To penalize T_i for capturing "too much" contextual information, our modified objective (2) adds a penalty term $\gamma \cdot \mathrm{I}(T_i; X|\hat{X}_i)$, which measures the amount of information about T_i given by the sentence X as a whole, beyond what is given by $\hat{X}_i$: $\mathrm{I}(T_i; X \mid \hat{X}_i) \stackrel{\text{def}}{=} \mathbb{E}_x\left[\mathbb{E}_{t_i \sim p_\theta(t_i|x)}\left[\log \frac{p_\theta(t_i|x)}{p_\theta(t_i|\hat{x}_i)}\right]\right]$. Setting $\gamma > 0$ will reduce this contextual information.

In practice, we found that $\mathrm{I}(T_i; X \mid \hat{X}_i)$ was small even when $\gamma = 0$, on the order of 3.5 nats whereas $\mathrm{I}(T_i; X)$ was 50 nats. In other words, the tags extracted by the classical method were already fairly local, so increasing γ above 0 had little qualitative effect. Still, γ might be important when applying our method to ELMo's competitors such as BERT.

We can derive an upper bound on $\mathrm{I}(T_i; X|\hat{X}_i)$ by approximating the conditional distribution $p_\theta(t_i \mid \hat{x}_i)$ with a variational distribution $s_\xi(t_i \mid \hat{x}_i)$, similar to §3.1.

$$\overbrace{\mathop{\mathbb{E}}_{x}\Big[\mathop{\mathbb{E}}_{t_i \sim p_\theta(t_i|x)}\Big[\log \frac{p_\theta(t_i|x)}{s_\xi(t_i|\hat{x}_i)}\Big]\Big]}^{\text{upper bound}} - \overbrace{\mathop{\mathbb{E}}_{x}\Big[\mathop{\mathbb{E}}_{t_i \sim p_\theta(t_i|x)}\Big[\log \frac{p_\theta(t_i|x)}{p_\theta(t_i|\hat{x}_i)}\Big]\Big]}^{\mathrm{I}(T_i;X|\hat{X}_i)}$$
$$= \mathop{\mathbb{E}}_{x}\left[\mathrm{KL}(p_\theta(t_i \mid \hat{x}_i) \mid\mid s_\xi(t_i \mid \hat{x}_i))\right] \geq 0$$

We replace it in (2) with this upper bound, which is equal to $\mathbb{E}_x\left[\sum_{i=1}^n \mathrm{KL}[p_\theta(t_i|x) \mid\mid s_\xi(t_i \mid \hat{x}_i)]\right]$.

The formal presentation above does not assume the specific factored model that we adopted in §3.2. When we adopt that model, $p_\theta(t_i \mid x)$ above reduces to $p_\theta(t_i \mid x_i)$—but our method in this section still has an effect, because x_i still reflects the context of the full sentence whereas $\hat{x}_i$ does not.

Type Encoder Architectures Notice that $s_\xi(t_i \mid \hat{x}_i)$ may be regarded as a type encoder, with parameters ξ that are distinct from the parameters θ of our token encoder $p_\theta(t_i \mid x_i)$. Given a choice of neural architecture for $p_\theta(t_i \mid x_i)$ (see §3.2), we always use the same architecture for $s_\xi(t_i \mid \hat{x}_i)$, except that p_θ takes a token vector as input whereas s_ξ takes a context-independent type vector. s_ξ is not used at test time, but only as part of our training objective.

3.4 $\mathrm{I}(Y;T)$ — the Decoder $q_\phi(y \mid t)$

Finally, $\mathrm{I}(Y;T) \stackrel{\text{def}}{=} \mathbb{E}_{y,t\sim p_\theta}\left[\log \frac{p_\theta(y|t)}{p(y)}\right]$. The $p(y)$ can be omitted during optimization as it does not depend on θ. Thus, making $\mathrm{I}(Y;T)$ large tries to obtain a high log-probability $p_\theta(y \mid t)$ for the true parse y when reconstructing it from t alone.

But how do we compute $p_\theta(y \mid t)$? This quantity effectively marginalizes over possible sentences x that *could have* explained t. Recall that p_θ is a joint distribution over x, y, t: see just above §3.1. So $p_\theta(y \mid t) \stackrel{\text{def}}{=} \frac{\sum_x p_\theta(x,y,t)}{\sum_{x,y'} p_\theta(x,y',t)}$. To estimate these sums accurately, we would have to identify the sentences x that are most consistent with the tagging t (that is, $p(x) \cdot p_\theta(t|x)$ is large): these contribute the largest summands, but might not appear in any corpus.

To avoid this, we replace $p_\theta(y \mid t)$ with a variational approximation $q_\phi(y \mid t)$ in our formula for $\mathrm{I}(Y;T)$. Here $q_\phi(\cdot \mid \cdot)$ is a tractable conditional distribution, and may be regarded as a stochastic parser that runs on a compressed tag sequence t instead of a word embedding sequence x. This modified version of $\mathrm{I}(Y;T)$ forms a lower bound on $\mathrm{I}(Y;T)$, for any value of the variational parameters ϕ, since the difference between them is a KL divergence and hence positive:

$$\overbrace{\mathop{\mathbb{E}}_{y,t\sim p_\theta}\left[\log \frac{p_\theta(y|t)}{p(y)}\right]}^{\mathrm{I}(Y;T)} - \overbrace{\mathop{\mathbb{E}}_{y,t\sim p_\theta}\left[\log \frac{q_\phi(y|t)}{p(y)}\right]}^{\text{lower bound}}$$
$$= \mathop{\mathbb{E}}_{t\sim p_\theta}\left[\mathrm{KL}(p_\theta(y \mid t) \mid\mid q_\phi(y \mid t))\right] \geq 0$$

We will maximize this lower bound of $\mathrm{I}(Y;T)$ with respect to both θ and ϕ. For any given θ, the optimal ϕ minimizes the expected KL divergence, meaning that q_ϕ approximates p_θ well.

More precisely, we again drop $p(y)$ as constant and then maximize a *sampling-based estimate* of $\mathbb{E}_{y,t\sim p_\theta}\left[\log q_\phi(y|t)\right]$. To sample y, t from the joint $p_\theta(x, y, t)$ we must first sample x, so we rewrite as $\mathbb{E}_{x,y}\left[\mathbb{E}_{t\sim p_\theta(t|x)}\left[\log q_\phi(y|t)\right]\right]$. The outer expectation $\mathbb{E}_{x,y}$ is estimated as usual over a training treebank. The expectation $\mathbb{E}_{t\sim p_\theta(t|x)}$ recognizes that t is stochastic, and again we estimate it by sampling. In short, when t is a stochastic compression of a treebank sentence x, we would like our variational parser *on average* to assign high log-probability $q_\phi(y \mid t)$ to its treebank parse y.

Decoder Architecture We use the deep biaffine dependency parser (Dozat and Manning, 2016) as our variational distribution $q_\phi(y \mid t)$, which functions as the decoder. This parser uses a Bi-LSTM to extract features from compressed tags or vectors and assign scores to each tree edge, setting $q_\phi(y \mid t)$ proportional to the exp of the total score of all edges in y. During IB training, the code[5] computes only an approximation to $q_\phi(y|t)$ for the gold tree y (although in principle, it could have computed the exact normalizing constant in polytime with Tutte's matrix-tree theorem (Smith and Smith, 2007; Koo et al., 2007; McDonald and Satta, 2007)). When we test the parser, the code does exactly find $\text{argmax}_y\, q_\phi(y \mid t)$ via the directed spanning tree algorithm of Edmonds (1966).

4 Training and Inference

With the approximations in §3, our final minimization objective is this upper bound on (2):

$$\mathop{\mathbb{E}}_{x,y}\Big[\mathop{\mathbb{E}}_{t\sim p_\theta(t|x)}[-\log q_\phi(y|t)] + \beta\,\mathrm{KL}(p_\theta(t|x)||r_\psi(t)) + \gamma\sum_{i=1}^{n}\mathrm{KL}(p_\theta(t_i \mid x)\;||\;s_\xi(t_i \mid \hat{x}_i))\Big] \quad (3)$$

We apply stochastic gradient descent to optimize this objective. To get a stochastic estimate of the objective, we first sample some (x, y) from the treebank. We then have many expectations over $t \sim p_\theta(t \mid x)$, including the KL terms. We could estimate these by sampling t from the token encoder $p_\theta(t \mid x)$ and then evaluating all q_ϕ, p_θ, r_ψ, and s_ξ probabilities. However, in fact we use the sampled t only to estimate the first expectation (by computing the decoder probability $q_\phi(y \mid t)$ of the gold tree y); we can compute the KL terms exactly by exploiting the structure of our distributions. The structure of p_θ and r_ψ means that the first KL term decomposes into $\sum_{i=1}^{n}\mathrm{KL}(p_\theta(t_i|x_i)||r_\psi(t_i))$. All KL terms are now between either two Gaussian distributions over a continuous tagset[6] or two categorical distributions over a small discrete tagset.[7]

To compute the stochastic gradient, we run backpropagation on this computation. We must apply the reparametrization trick to backpropagate through the step that sampled t. This finds the gradient of parameters that derive t from a random variate z, while holding z itself fixed. For continuous t, we use the reparametrization trick for multivariate Gaussians (Rezende et al., 2014). For discrete t, we use the Gumbel-softmax variant (Jang et al., 2016; Maddison et al., 2016).

To evaluate our trained model's ability to parse a sentence x from compressed tags, we obtain a parse as $\text{argmax}_y\, q_\phi(y \mid t)$, where $t \sim p_\theta(\cdot \mid x)$ is a single sample. A better parser would instead estimate $\text{argmax}_y\, \mathbb{E}_t\,[q_\phi(y \mid t)]$ where $\mathbb{E}_t$ averages over many samples t, but this is computationally hard.

[5] We use the implementation from AllenNLP library (Gardner et al., 2017).

[6] $\mathrm{KL}(\mathcal{N}_0 \,||\, \mathcal{N}_1) = \frac{1}{2}(\mathrm{tr}(\Sigma_1^{-1}\Sigma_0)+(\mu_1-\mu_0)^T\Sigma_1^{-1}(\mu_1-\mu_0)-d+\log(\frac{\det(\Sigma_1)}{\det(\Sigma_0)})$

[7] $\mathrm{KL}(p_\theta(t_i|x_i)||r_\psi(t_i)) = \sum_{t_i=1}^{k} p_\theta(t_i \mid x_i)\log\frac{p_\theta(t_i|x_i)}{r_\psi(t_i)}$

Language	Treebank	#Tokens	H($A \mid \hat{X}$)	H(A)
Arabic	PADT	282k	0.059	2.059
Chinese	GSD	123k	0.162	2.201
English	EWT	254k	0.216	2.494
French	GSD	400k	0.106	2.335
Hindi	HDTB	351k	0.146	2.261
Portuguese	Bosque	319k	0.179	2.305
Russian	GSD	98k	0.049	2.132
Spanish	AnCora	549k	0.108	2.347
Italian	ISDT	298K	0.120	2.304

Table 1: Statistics of the datasets used in this paper. "Treebank" is the treebank identifier in UD, "#Token" is the number of tokens in the treebank, "H(A)" is the entropy of a gold POS tag (in nats), and "H($A \mid \hat{X}$)" is the conditional entropy of a gold POS tag conditioned on a word type (in nats).

5 Experimental Setup

Data Throughout §§6–7, we will examine our compressed tags on a subset of Universal Dependencies (Nivre et al., 2018), or UD, a collection of dependency treebanks across 76 languages using the same POS tags and dependency labels. We experiment on Arabic, Hindi, English, French, Spanish, Portuguese, Russian, Italian, and Chinese (Table 1)—languages with different syntactic properties like word order. We use only the sentences with length ≤ 30. For each sentence, x is obtained by running the standard pre-trained ELMo on the UD token sequence (although UD's tokenization may not perfectly match that of ELMo's training data), and y is the labeled UD dependency parse *without* any part-of-speech (POS) tags. Thus, our tags t are tuned to predict only the dependency relations in UD, and not the gold POS tags a also in UD.

Pretrained Word Embeddings For English, we used the pre-trained English ELMo model from the AllenNLP library (Gardner et al., 2017). For the

other 8 languages, we used the pre-trained models from Che et al. (2018). Recall that ELMo has two layers of bidirectional LSTM (layer 1 and 2) built upon a context-independent character CNN (layer 0). We use either layer 1 or 2 as the input (x_i) to our token encoder p_θ. Layer 0 is the input ($\hat{x}_i$) to our type encoder s_ξ. Each encoder network (§§3.2–3.3) has a single hidden layer with a tanh transfer function, which has $2d$ hidden units (typically 128 or 512) for continuous encodings and 512 hidden units for discrete encodings.

Optimization We optimize with Adam (Kingma and Ba, 2014), a variant of stochastic gradient descent. We alternate between improving the model $p_\theta(t|x)$ on even epochs and the variational distributions $q_\phi(y|t)$, $r_\psi(t)$, $s_\xi(t_i \mid \hat{x}_i)$ on odd epochs.

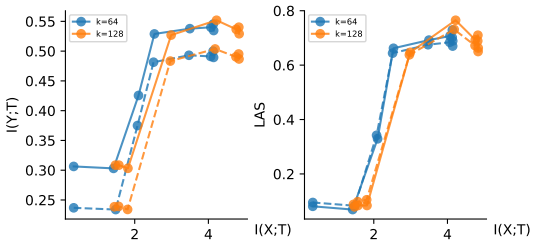
We train for 50 epochs with minibatches of size 20 and L_2 regularization. The learning rate and the regularization coefficients are tuned on dev data for each language separately. For each training sentence, we average over 5 i.i.d. samples of T to reduce the variance of the stochastic gradient. The initial parameters θ, ϕ, ψ, ξ are all drawn from $\mathcal{N}(0, I)$. We experiment with different dimensionalities $d \in \{5, 32, 256, 512\}$ for the continuous tags, and different cardinalities $k \in \{32, 64, 128\}$ for the discrete tag set. We also tried different values $\beta, \gamma \in \{10^{-6}, 10^{-5}, \cdots, 10^{1}\}$ of the compression tradeoff parameter. We use temperature annealing when sampling from the Gumbel-softmax distribution (§4). At training epoch i, we use temperature τ_i, where $\tau_1 = 5$ and $\tau_{i+1} = \max(0.5, e^{-\gamma}\tau_i)$. We set the annealing rate $\gamma = 0.1$. During testing, we use $\tau = 0$, which gives exact softmax sampling.

6 Scientific Evaluation

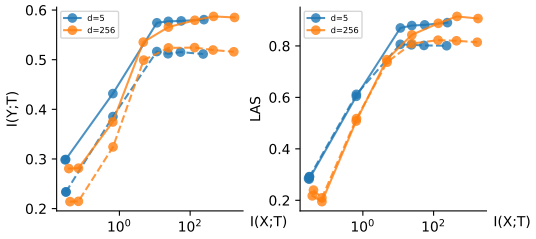
In this section, we study what information about words is retained by our automatically constructed tagging schemes. First, we show the relationship between $I(Y;T)$ and $I(X;T)$ on English as we reduce β to capture more information in our tags.[8]

Second, across 9 languages, we study how our automatic tags correlate with gold part-of-speech tags (and in English, with other syntactic properties), while suppressing information about semantic properties. We also show how decreasing β gradually refines the automatic discrete tag set, giving intuitive fine-grained clusters of English words.

[8] We always set $\gamma = \beta$ to simplify the experimental design.



(a) Discrete Version



(b) Continuous Version

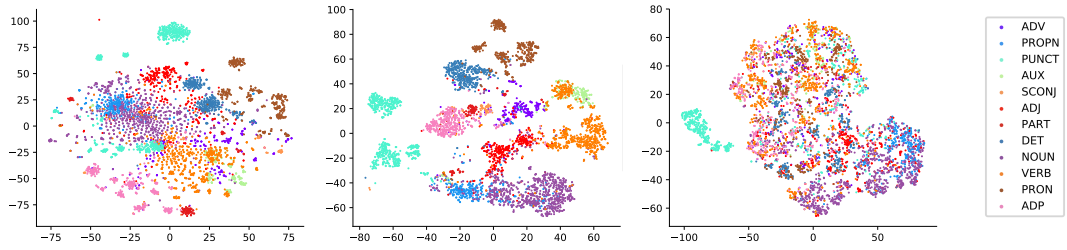
Figure 2: Compression-prediction tradeoff curves of VIB in our dependency parsing setting. The upper figures use discrete tags, while the lower figures use continuous tags. The dashed lines are for test data, and the solid lines for training data. The "dim" in the legends means the dimensionality of the continuous tag vector or the cardinality of the discrete tag set. On the left, we plot predictiveness $I(Y;T)$ versus $I(X;T)$ as we lower β multiplicatively from 10^1 to 10^{-6} on a log-scale. On the right, we alter the y-axis to show the labeled attachment score (LAS) of 1-best dependency parsing. All mutual information and entropy values in this paper are reported in nats per token. Furthermore, the mutual information values that we report are actually our variational upper bounds, as described in §3. The reason that $I(X;T)$ is so large for continuous tags is that it is *differential* mutual information (see footnote 3). Additional tradeoff curves w.r.t. $I(T_i; X \mid \hat{X}_i)$ are in Appendix B.

6.1 Tradeoff Curves

As we lower β to retain more information about X, both $I(X;T)$ and $I(Y;T)$ rise, as shown in Figure 2. There are diminishing returns: after some point, the additional information retained in T does not contribute much to predicting Y. Also noteworthy is that at each level of $I(X,T)$, very low-dimensional tags ($d = 5$) perform on par with high-dimensional ones ($d = 256$). (Note that the high-dimensional stochastic tags will be noisier to keep the same $I(X,T)$.) The low-dimensional tags allow far faster CPU parsing. This indicates that VIB can achieve strong practical task-specific compression.

6.2 Learned Tags vs. Gold POS Tags

We investigate how our automatic tag T_i correlates with the gold POS tag A_i provided by UD.



(a) ELMo, $I(X;T) = H(X) \approx 400.6$ (b) $I(X;T) \approx 24.3$ (c) $I(X;T) \approx 0.069$

Figure 3: t-SNE visualization of VIB model ($d = 256$) on the projected space of the continuous tags. Each marker in the figure represents a word token, colored by its gold POS tag. This series of figures (from left to right) shows a progression from no compression to moderate compression and to too-much compression.

Continuous Version We use t-SNE (van der Maaten and Hinton, 2008) to visualize our compressed continuous tags on held-out test data, coloring each token in Figure 3 according to its gold POS tag. (Similar plots for the discrete tags are in Figure 6 in the appendix.)

In Figure 3, the first figure shows the original uncompressed level-1 ELMo embeddings of the tokens in test data. In the two-dimensional visualization, the POS tags are vaguely clustered but the boundaries merge together and some tags are diffuse. The second figure is when $\beta = 10^{-3}$ (moderate compression): our compressed embeddings show clear clusters that correspond well to gold POS tags. Note that the gold POS tags were not used in training either ELMo or our method. The third figure is when $\beta = 1$ (too much compression), when POS information is largely lost. An interesting observation is that the purple `NOUN` and blue `PROPN` distributions overlap in the middle distribution, meaning that it was unnecessary to distinguish common nouns from proper nouns for purposes of our parsing task.[9]

Discrete Version We also quantify how well our specialized discrete tags capture the traditional POS categories, by investigating $I(A;T)$. This can be written as $H(A) - H(A \mid T)$. Similarly to §3.4, our probability distribution has the form $p_\theta(x, a, t) = p(x, a) \cdot p_\theta(t \mid x)$, leading us to write $H(A \mid T) \leq \mathbb{E}_{x,a}\left[\mathbb{E}_{t\sim p_\theta(t|x)}\left[-\log q(a \mid t)\right]\right]$ where $q(a \mid t) = \prod_i q(a_i \mid t_i)$ is a variational distribution that we train to minimize this upper bound. This is equivalent to training $q(a \mid t)$ by maximum conditional likelihood. In effect, we are doing transfer learning, fixing our trained IB encoder (p_θ) and now using it to predict A instead of Y, but otherwise following §3.4. We similarly upper-bound $H(A)$ by assuming a model $q'(a) = \prod_i q'(a_i)$ and estimating q' as the empirical distribution over training tags. Having trained q and q' on training data, we estimate $H(A \mid T)$ and $H(A)$ using the same upper-bound formulas on our test data.

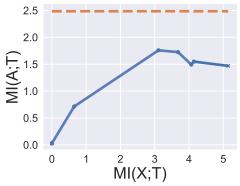
We experiment on all 9 languages, taking T_i at the moderate compression level $\beta = 0.001$, $k = 64$. As Figure 4 shows, averaging over the 9 languages, the reconstruction retains 71% of POS information (and as high as 80% on Spanish and French). We can conclude that the information encoded in the specialized tags correlates with the gold POS tags, but does not perfectly predict the POS.

The graph in Figure 4 shows a "U-shaped" curve, with the best overall error rate at $\beta = 0.01$. That is, moderate compression of ELMo embeddings helps for predicting POS tags. Too much compression squeezes out POS-related information, while too little compression allows the tagger to overfit the training data, harming generalization to test data. We will see the same pattern for parsing in §7.

Syntactic Features As a quick check, we determine that our tags also make syntactic distinctions beyond those that are recognized by the UD POS tag set, such as tense, number, and transitivity. See Appendix D for graphs. For example, even with moderate compression, we achieve 0.87 classification accuracy in distinguishing between transitive and intransitive English verbs, given only tag t_i.

Stem When we compress ELMo embeddings to k discrete tags, the semantic information must be squeezed out because k is small. But what about the continuous case? In order to verify that semantic information is excluded, we train a classifier that predicts the stem of word token i from its mean tag vector $\mathbb{E}[T_i]$. We expect "player" and "buyer" to have similar compressed vectors, because they share syntactic roles, but we should *fail*

[9] Both can serve as arguments of verbs and prepositions. Both can be modified by determiners and adjectives, giving rise to proper NPs like "The Daily Tribune."



embeddings	Arabic	English	Spanish	French	Hindi	Italian	Portuguese	Russian	Chinese
H(A)	2.016	2.486	2.345	2.206	2.247	2.291	2.306	2.131	2.195
ELMo0	67.2%	74.2%	75.7%	79.6%	70.1%	77.9%	76.5%	73.2%	57.3%
ELMo1	67.2%	76.1%	71.7%	78.0%	70.5%	78.1%	72.3%	73.8%	59.8%
ELMo2	63.8%	71.0%	79.7%	78.7%	67.2%	74.5%	75.3%	72.2%	59.4%

Figure 4: Graph at left: I($A;T$) vs. I($X;T$) in English (in units of nats per token). Table at right: how well the discrete specialized tags predict gold POS tags for 9 languages. The H(A) row is the entropy (in nats per token) of the gold POS tags in the test data corpus, which is an upper bound for I($A;T$). The remaining rows report the percentage I($A;T$)/H(A).

to predict that they have different stems "play" and "buy." The classifier is a feedforward neural network with *tanh* activation function, and the last layer is a softmax over the stem vocabulary. In the English treebank, we take the word lemma in UD treebank and use the NLTK library (Bird et al., 2009) to stem each lemma token. Our result (Appendix E in the appendix) suggests that more compression destroys stem information, as hoped. With light compression, the error rate on stem prediction can be below 15%. With moderate compression $\beta = 0.01$, the error rate is 89% for ELMo layer 2 and 66% for ELMo layer 1. Other languages show the same pattern, as shown in Appendix E in the appendix. Thus, moderate and heavy compression indeed squeeze out semantic information.

6.3 Annealing of Discrete Tags

Deterministic annealing (Rose, 1998; Friedman et al., 2001) is a method that gradually decreases β during training of IB. Each token i has a stochastic distribution over the possible tags $\{1, \ldots, k\}$. This can be regarded as a soft clustering where each token is fractionally associated with each of the k clusters. With high β, the optimal solution turns out to assign to all tokens an identical distribution over clusters, for a mutual information of 0. Since all clusters then have the same membership, this is equivalent to having a single cluster. As we gradually reduce β, the cluster eventually splits. Further reduction of β leads to recursive splitting, yielding a hierarchical clustering of tokens (Appendix A).

We apply deterministic annealing to the English dataset, and the resulting hierarchical structure reflects properties of English syntax. At the top of the hierarchy, the model places nouns, adjectives, adverbs, and verbs in different clusters. At lower levels, the anaphors ("yourself," "herself" ...), possessive pronouns ("his," "my," "their" ...), accusative-case pronouns ("them," "me," "him," "myself" ...), and nominative-case pronouns ("I," "they," "we" ...) each form a cluster, as do the *wh*-words ("why," "how," "which," "who," "what," ...).

7 Engineering Evaluation

As we noted in §1, learning how to compress ELMo's tags for a given task is a fast alternative to fine-tuning all the ELMo parameters. We find that indeed, training a compression method to keep only the relevant information does improve our generalization performance on the parsing task.

We compare 6 different token representations according to the test accuracy of a dependency parser trained to use them. The same training data is used to jointly train the parser and the token encoder that produces the parser's input representations.

Continuous tags:

`Iden` is an baseline model that leaves the ELMo embeddings uncompressed, so $d = 1024$.

`PCA` is a baseline that simply uses Principal Components Analysis to reduce the dimensionality to $d = 256$. Again, this is not task-specific.

`MLP` is another deterministic baseline that uses a multi-layer perceptron (as in Dozat and Manning (2016)) to reduce the dimensionality to $d = 256$ in a task-specific and nonlinear way. This is identical to our continuous VIB method except that the variance of the output Gaussians is fixed to 0, so that the d dimensions are fully informative.

`VIBc` uses our stochastic encoder, still with $d = 256$. The average amount of stochastic noise is controlled by β, which is tuned per-language on dev data.

Discrete tags:

`POS` is a baseline that uses the $k \leq 17$ gold POS tags from the UD dataset.

`VIBd` is our stochastic method with $k = 64$ tags. To compare fairly with **`POS`**, we pick a β value for each language such that $\mathrm{H}(T_i \mid X_i) \approx \mathrm{H}(A_i \mid X_i)$.

Runtime. Our VIB approach is quite fast. With minibatching on a single GPU, it is able to train on 10,000 sentences in 100 seconds, per epoch.

Models	Arabic	Hindi	English	French	Spanish	Portuguese	Russian	Chinese	Italian
Iden	0.751	**0.870**	0.824	0.784	0.808	0.813	0.783	0.709	**0.863**
PCA	0.743	**0.866**	0.823	0.749	0.802	0.808	0.777	0.697	0.857
MLP	0.759	**0.871**	0.839	0.816	**0.835**	0.821	0.800	0.734	**0.867**
VIBc	**0.779**	**0.866**	**0.851**	**0.828**	**0.837**	**0.836**	**0.814**	**0.754**	**0.867**
POS	0.652	0.713	0.712	0.718	**0.739**	**0.743**	**0.662**	0.510	0.779
VIBd	**0.672**	**0.736**	**0.742**	**0.723**	**0.725**	0.710	**0.651**	**0.591**	**0.781**

Table 2: Parsing accuracy of 9 languages (LAS). Black rows use continuous tags; gray rows use discrete tags (which does worse). In each column, the best score for each color is boldfaced, along with all results of that color that are not significantly worse (paired permutation test, $p < 0.05$). These results use only ELMo layer 1; results from all layers are shown in Table 3 in the appendix, for both LAS and UAS metrics.

Analysis. Table 2 shows the test accuracies of these parsers, using the standard training/development/test split for each UD language.

In the continuous case, the VIB representation outperforms all three baselines in 8 of 9 languages, and is not significantly worse in the 9th language (Hindi). In short, our VIB joint training generalizes better to test data. This is because the training objective (2) includes terms that focus on the parsing task and also regularize the representations.

In the discrete case, the VIB representation outperforms gold POS tags (at the same level of granularity) in 6 of 9 languages, and of the other 3, it is not significantly worse in 2. This suggests that our learned discrete tag set could be an improved alternative to gold POS tags (cf. Klein and Manning, 2003) when a discrete tag set is needed for speed.

8 Related Work

Much recent NLP literature examines syntactic information encoded by deep models (Linzen et al., 2016) and more specifically, by powerful unsupervised word embeddings. Hewitt and Manning (2019) learn a linear projection from the embedding space to predict the distance between two words in a parse tree. Peters et al. (2018b) and Goldberg (2019) assess the ability of BERT and ELMo directly on syntactic NLP tasks. Tenney et al. (2019) extract information from the contextual embeddings by self-attention pooling within a span of word embeddings.

The IB framework was first used in NLP to cluster distributionally similar words (Pereira et al., 1993). In cognitive science, it has been used to argue that color-naming systems across languages are nearly optimal (Zaslavsky et al., 2018). In machine learning, IB provides an information-theoretic perspective to explain the performance of deep neural networks (Tishby and Zaslavsky, 2015b).

The VIB method makes use of variational upper and lower bounds on mutual information. An alternative lower bound was proposed by Poole et al. (2019), who found it to work better empirically.

9 Conclusion and Future Work

In this paper, we have proposed two ways to syntactically compress ELMo word token embeddings, using variational information bottleneck. We automatically induce stochastic discrete tags that correlate with gold POS tags but are as good or better for parsing. We also induce stochastic continuous token embeddings (each is a Gaussian distribution over $\mathbb{R}^d$) that forget non-syntactic information captured by ELMo. These stochastic vectors yield improved parsing results, in a way that simpler dimensionality reduction methods do not. They also transfer to the problem of predicting gold POS tags, which were not used in training.

One could apply the same training method to compress the ELMo or BERT token sequence x for other tasks. All that is required is a model-specific decoder $q_\phi(y \mid t)$. For example, in the case of sentiment analysis, the approach should preserve only sentiment information, discarding most of the syntax. One possibility that does not require supervised data is to create artificial tasks, such as reproducing the input sentence or predicting missing parts of the input (such as affixes and function words). In this case, the latent representations would be essentially generative, as in the variational autoencoder (Kingma and Welling, 2013).

Acknowledgments

This work was supported by the National Science Foundation under Grant No. 1718846 and by a Provost's Undergraduate Research Award to the first author. The Maryland Advanced Research Computing Center provided computing facilities. We thank the anonymous reviewers and Hongyuan Mei for helpful comments.

References

Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. 2016. Deep Variational Information Bottleneck. *Proceedings of the International Conference on Learning Representations (ICLR)*, abs/1612.00410.

T.W. Anderson. 2003. *An Introduction to Multivariate Statistical Analysis*. Wiley Series in Probability and Statistics. Wiley.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*, 1st edition. O'Reilly Media, Inc.

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.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Timothy Dozat and Christopher D. Manning. 2016. Deep biaffine attention for neural dependency parsing. *CoRR*, abs/1611.01734.

Jack Edmonds. 1966. Optimum Branchings. *Journal of Research of the National Bureau of Standards*.

Nir Friedman, Ori Mosenzon, Noam Slonim, and Naftali Tishby. 2001. Multivariate information bottleneck. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, UAI '01, pages 152–161, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. AllenNLP: A Deep Semantic Natural Language Processing Platform.

Yoav Goldberg. 2019. Assessing BERT's syntactic abilities. *CoRR*, abs/1901.05287.

John Hewitt and Christopher D. Manning. 2019. A structural probe for finding syntax in word representations. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with Gumbel-softmax. *International Conference on Learning Representations*.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *Proceedings of the International Conference on Learning Representations (ICLR)*.

D. Klein and C. D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of ACL*.

Terry Koo, Amir Globerson, Xavier Carreras Pérez, and Michael Collins. 2007. Structured prediction models via the matrix-tree theorem. In *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 141–150.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. 2016. The concrete distribution: A continuous relaxation of discrete random variables. *CoRR*, abs/1611.00712.

Ryan McDonald and Giorgio Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *Proceedings of the 10th International Conference on Parsing Technologies*, pages 121–132. Association for Computational Linguistics.

Joakim Nivre et al. 2018. Universal dependencies 2.3. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Fernando Pereira, Naftali Tishby, and Lillian Lee. 1993. Distributional clustering of English words. In *Proceedings of the 31st Annual Meeting of Association for Computational Linguistics*, pages 183–190. Association for Computational Linguistics.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018a. Deep contextualized word representations. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*.

Matthew E. Peters, Mark Neumann, Luke Zettlemoyer, and Wen-tau Yih. 2018b. Dissecting contextual word embeddings: Architecture and representation. *Empirical Methods in Natural Language Processing (EMNLP)*, abs/1808.08949.

Ben Poole, Sherjil Ozair, Aäron van den Oord, Alexander A. Alemi, and George Tucker. 2019. On variational bounds of mutual information. *CoRR*, abs/1905.06922.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. 2014. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.

Kenneth Rose. 1998. Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. *Proceedings of the IEEE*, 80:2210–2239.

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.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R. Bowman, Dipanjan Das, and Ellie Pavlick. 2019. What do you learn from context? Probing for sentence structure in contextualized word representations.

Naftali Tishby, Fernando C Pereira, and William Bialek. 2000. The information bottleneck method. *arXiv preprint physics/0004057*.

Naftali Tishby and Noga Zaslavsky. 2015a. Deep learning and the information bottleneck principle. *CoRR*, abs/1503.02406.

Naftali Tishby and Noga Zaslavsky. 2015b. Deep learning and the information bottleneck principle. *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5.

L.J.P. van der Maaten and G.E. Hinton. 2008. Visualizing high-dimensional data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

Noga Zaslavsky, Charles Kemp, Terry Regier, and Naftali Tishby. 2018. Efficient human-like semantic representations via the information bottleneck principle. *CoRR*, abs/1808.03353.

Supplementary Material

A Details of Deterministic Annealing

In practice, deterministic annealing (§6.3) is implemented in a way that dynamically increases the number of clusters k (Friedman et al., 2001), leading to a hierarchical clustering. First, we initialize with one cluster, and all the word tokens are mapped to that cluster with probability 1. Second, for each cluster i, duplicate the cluster C_i to form C_{ia}, C_{ib}, and divide the probabilities associated with C_i approximately evenly (with perturbation) between the two clusters, i.e., set $p(c_{ia}|x) = \frac{1}{2}p(c_i|x) + \epsilon_x$ and $p(c_{ib}|x) = \frac{1}{2}p(c_i|x) - \epsilon_x$. Third, update $\beta \leftarrow \beta/\alpha$, and run optimization until convergence. Fourth, for each former cluster i, if C_{ia} and C_{ib} have not differentiated from each other, remerge them by setting $p(c_i|x) = p(c_{ia}|x)+p(c_{ib}|x)$. (Optimization will have pulled them together again for higher β values and pushed them apart for lower β values.) Our heuristic is to re-merge them if for all word tokens x, $|p(c_{ia}|x) - p(c_{ib}|x)| \leq 0.01$. Finally, loop back to the second step, unless the β value has fallen below a given threshold $\beta_{\min}$ or we have reached a desired maximum number of clusters.

B Additional Tradeoff Curves

Figure 5 supplements the tradeoff curves in Figure 2 by plotting the relationship between $\mathrm{I}(T_i; X \mid \hat{X}_i)$ vs. $\mathrm{I}(Y;T)$, and $\mathrm{I}(T_i; X \mid \hat{X}_i)$ vs. LAS. Moving leftward on the graphs, each T_i contains less *contextual* information about word i (because γ in equation (2) is larger) as well as less information overall about word i (because we always set $\gamma = \beta$, so β is larger as well). The graphs show that the tag sequence T then becomes less informative about the parse Y.

C Additional t-SNE plots

Recall that Figure 3 (in §6.2) was a row of t-SNE visualizations of the continuous *token* embeddings $p_\theta(t_i \mid x_i)$ under no compression, moderate compression, and too much compression. Figure 6 gives another row visualizing the continuous *type* embeddings $s_\xi(t_i \mid \hat{x}_i)$ in the same way. In both cases, the "moderate compression" condition shows $\beta = 0.01$.

Figure 6 also shows rows for the *discrete* type and token embeddings. In both cases, the "moderate compression" condition shows $\beta = 0.001$.

In the continuous case, each point given to t-SNE is the mean of a Gaussian-distributed stochastic embedding, so it is in $\mathbb{R}^d$. In the discrete case, each point given to t-SNE is a vector of k tag probabilities, so it is in $\mathbb{R}^k$ and more specifically in the $(k-1)$-dimensional simplex. The t-SNE visualizer plots these points in 2 dimensions.

The message of all these graphs is that the tokens or types with the same gold part of speech (shown as having the same color) are most nicely grouped together in the moderate compression condition.

D Syntactic Feature Classification

Figure 7 shows results for the **Syntactic Features** paragraph in §6.2, by showing the prediction accuracy of subcategorization frame, tense, and number from t_i as a function of the level of compression. We used an SVM classifier with a radial basis function kernel.

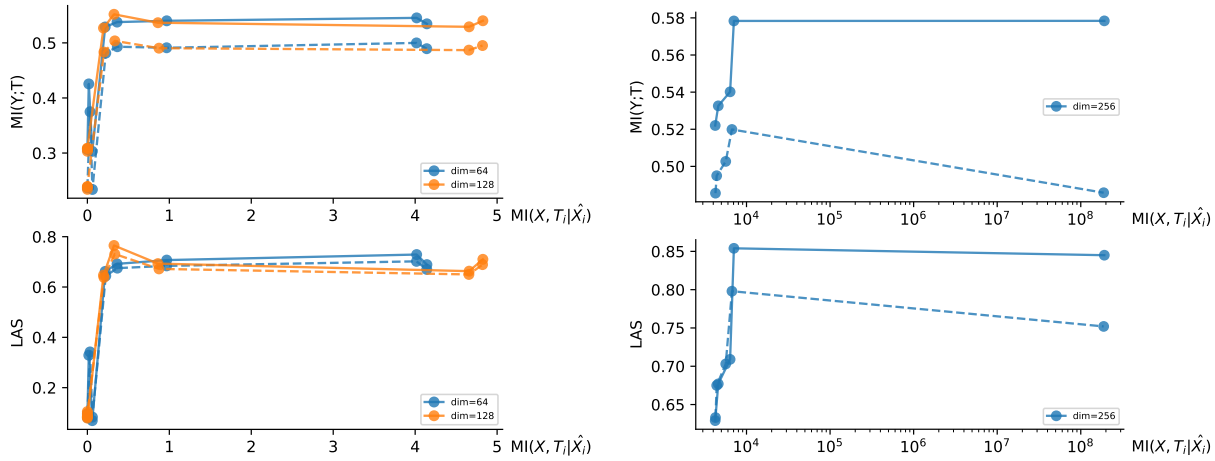
All results are on the English UD data with the usual training/test split. To train and test the classifiers, we used the gold UD annotations to identify the nouns and verbs and their correct syntactic features.

E Plot & Table for Stem Prediction

Figures 8–9 supplement the **Stem** paragraph in §6.2. Figure 8 plots the error rate of reconstructing English stems as a function of the level of compression. Figure 9 shows the reconstruction error rate for the other 8 languages.

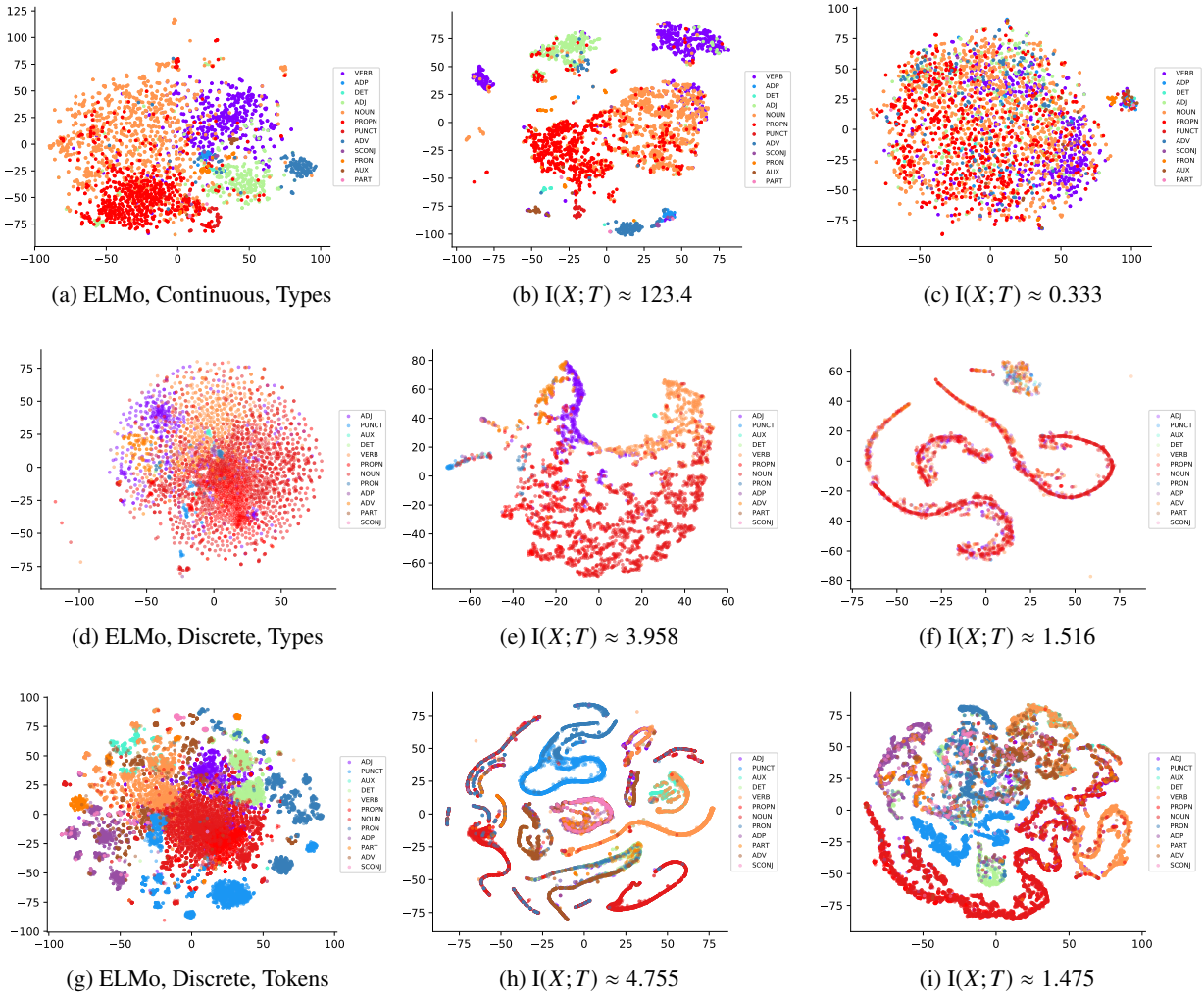
F Additional Table of Parsing Performance

Table 3 is an extended version of Table 2 in §7. It includes parsing performance (measured by LAS and UAS) using ELMo layer 0, 1, and 2.



(a) Discrete Version – ELMo layer-1 (b) Continuous Version – ELMo layer-1

Figure 5: . Tradeoff curves for $I(T_i; X \mid \hat{X}_i)$ vs. $I(Y; T)$ and $I(T_i; X \mid \hat{X}_i)$ vs. LAS, complementary to Figure 2.



(a) ELMo, Continuous, Types (b) $I(X;T) \approx 123.4$ (c) $I(X;T) \approx 0.333$

(d) ELMo, Discrete, Types (e) $I(X;T) \approx 3.958$ (f) $I(X;T) \approx 1.516$

(g) ELMo, Discrete, Tokens (h) $I(X;T) \approx 4.755$ (i) $I(X;T) \approx 1.475$

Figure 6: t-SNE visualization of our continuous tags ($d = 256$) and our distributions over discrete tags ($k = 128$), supplementing Figure 3. Each marker in the figure represents a word token, colored by its gold POS tag. For each row, the series of figures (from left to right) shows a transition from no compression to moderate compression and to too-much compression. The first row (a-c) shows the continuous type embeddings; the second row (d-f) shows the discrete type embeddings; the third row (g-i) shows the discrete token embeddings.

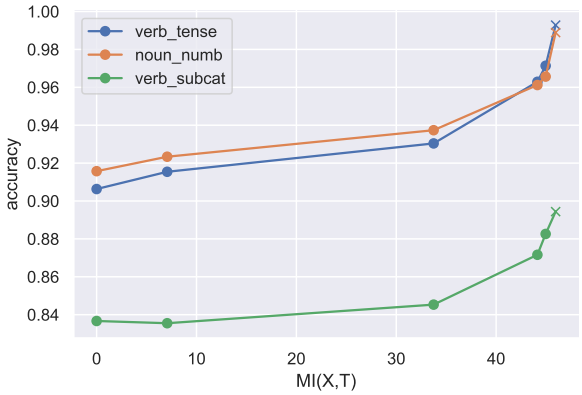


Figure 7: The accuracy of predicting the subcategorization frame of verbs (transitive/intransitive), number of nouns (plural/singular), and tense of verbs (past/present/future), as we change the level of compression of ELMo layer-1 (see the **Subcategorization frame** paragraph in §6.2). As we move from right to left and squeeze irrelevant information out of the tags, they retain these three syntactic distinctions quite well.

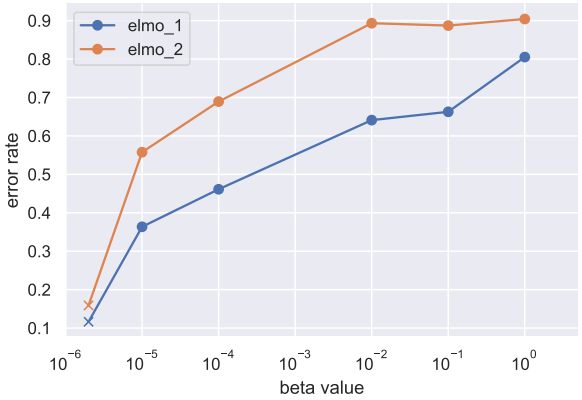


Figure 8: The error rate of reconstructing the stem of a word from the specialized continuous tags. The legend indicates whether we are compressing the ELMo layer-1 or ELMo layer-2 (see the **Stem** paragraph of §6.2).

Compression	layer	Arabic	Spanish	French	Hindi	Italian	Portuguese	Russian	Chinese
Slight	1	26.7%	24.0%	25.5%	20.5%	29.9%	32.5%	38.8%	26.7%
Moderate	1	89.5%	79.8%	66.7%	94.6%	94.7%	93.7%	94.0%	89.6%
Slight	2	34.9%	34.9%	34.9%	34.9%	34.9%	34.9%	34.9%	34.9%
Moderate	2	94.3%	94.3%	94.3%	94.3%	94.3%	94.3%	94.3%	94.3%

Figure 9: Error rate in reconstructing the stem of a word from the compressed version of the ELMo layer-1 and layer-2 embedding). Slight compression refers to $\beta = 0.0001$, and moderate compression refers to $\beta = 0.01$.

UAS										
Models	Layer	Arabic	Hindi	English	French	Spanish	Portuguese	Russian	Chinese	Italian
Iden	0	0.817	0.914	0.793	0.836	0.851	0.844	0.859	0.775	0.904
Iden	1	0.821	**0.915**	0.868	0.833	0.852	0.842	0.860	0.771	0.903
Iden	2	0.820	0.914	0.843	0.833	0.856	0.841	0.859	0.773	0.901
PCA	0	0.814	0.912	0.787	0.814	0.847	0.857	0.831	0.773	0.897
PCA	1	0.815	0.912	0.865	0.807	0.846	0.855	0.828	0.759	0.899
PCA	2	0.814	0.915	0.832	0.808	0.846	0.858	0.829	0.766	0.902
MLP	0	0.830	0.918	0.742	0.856	0.829	0.869	0.852	0.797	0.910
MLP	1	0.831	0.923	0.823	0.870	0.832	0.867	0.852	0.800	0.908
MLP	2	0.833	0.918	0.787	0.859	0.813	0.871	0.849	0.790	**0.914**
VIBc	0	0.852	**0.915**	0.866	**0.879**	**0.881**	0.871	0.862	0.800	0.831
VIBc	1	**0.860**	**0.913**	**0.871**	**0.877**	**0.880**	**0.877**	**0.865**	**0.814**	**0.913**
VIBc	2	0.851	0.894	**0.880**	0.876	0.879	**0.877**	0.843	0.768	0.878
POS	-	0.722	0.819	0.762	0.800	0.802	**0.808**	0.739	0.570	**0.843**
VIBd	0	**0.783**	0.823	0.784	**0.821**	**0.821**	0.793	**0.777**	0.671	**0.855**
VIBd	1	**0.784**	**0.862**	**0.825**	**0.822**	**0.822**	**0.805**	**0.776**	**0.691**	**0.857**
VIBd	2	0.754	**0.861**	0.816	**0.822**	0.812	0.790	0.768	0.672	0.849

LAS										
Models	layer	Arabic	Hindi	English	French	Spanish	Portuguese	Russian	Chinese	Italian
Iden	0	0.747	**0.867**	0.745	0.789	0.806	0.812	0.788	0.713	**0.864**
Iden	1	0.751	**0.870**	0.824	0.784	0.808	0.813	0.783	0.709	**0.863**
Iden	2	0.743	0.867	0.798	0.782	0.811	0.813	0.787	0.713	0.861
PCA	0	0.746	0.864	0.742	0.758	0.804	0.811	0.781	0.706	0.856
PCA	1	0.743	**0.866**	0.823	0.749	0.802	0.808	0.777	0.697	0.857
PCA	2	0.744	**0.870**	0.787	0.750	0.801	0.811	0.780	0.700	**0.865**
MLP	0	0.754	**0.869**	0.801	0.814	0.772	0.817	0.798	0.739	**0.871**
MLP	1	0.759	**0.871**	0.839	0.816	**0.835**	0.821	0.800	0.734	**0.867**
MLP	2	0.760	**0.871**	0.834	0.814	0.755	0.822	0.797	0.726	0.869
VIBc	0	**0.778**	0.865	0.822	0.822	**0.839**	0.827	0.807	0.739	0.862
VIBc	1	**0.779**	**0.866**	**0.851**	**0.828**	**0.837**	**0.836**	**0.814**	**0.754**	**0.867**
VIBc	2	0.777	0.838	0.840	**0.826**	0.840	0.829	0.786	0.710	0.818
POS	-	0.652	0.713	0.712	0.718	**0.739**	**0.743**	**0.662**	0.510	0.779
VIBd	0	**0.671**	0.702	0.721	**0.723**	**0.724**	0.710	0.648	0.544	**0.780**
VIBd	1	**0.672**	**0.736**	**0.742**	**0.723**	**0.725**	0.710	**0.651**	**0.591**	**0.781**
VIBd	2	0.643	**0.735**	**0.741**	0.721	0.719	0.698	0.646	0.566	0.763

Table 3: Parsing accuracy of 9 languages (LAS and UAS); Table 2 is a subset of this table. Black rows use continuous tags; gray rows use discrete tags (which does worse). The "layer" column indicates the ELMo layer we use. In each column, the best score for each color is boldfaced, along with all results of that color that are not significantly worse (paired permutation test, $p < 0.05$).